MMagic

MMagic Authors

2023 年 08 月 24 日

MMagic 社区

1 贡献代码 3

2 生态项目 23

3 概述 25

4 安装教程 29

5 快速运行 35

6 教程 1 了解 MMagic 的配置文件 37

7 教程 2：准备数据集 63

8 教程 3：使用预训练模型推理 65

9 教程 4：在 MMagic 环境下训练与测试 75

10 教程 5：使用评价指标 81

11 教程 6：可视化 89

12 教程 7：实用工具（待更新） 97

13 教程 8：模型部署指南 103

14 评估器 109

15 Data Structure 115

16 数据预处理器（待更新） 119

17 数据流 121

18 如何设计自己的模型 125

19 如何自定义数据集 143

20 如何设计自己的数据变换 157

21 如何设计自己的损失函数 165

22 常见问题解答 175

23 概览 177

24 去模糊 179

25 图像修复 183

26 视频超分辨率 191

27 扩散模型 203

28 图像到图像的翻译 207

29 图像恢复 211

30 jpeg 压缩伪影移除 213

31 条件生成对抗网络 231

32 视频插帧 235

33 图文生成 241

34 图像上色 245

35 图像去噪，图像去模糊，图像去雨 247

36 图像去噪 255

37 图像超分辨率 273

38 图像抠图 303

39 概览 309

40 准备 DF2K_OST 数据集 311

41 准备 RealSRSet 数据集 315

42 准备 VideoLQ 数据集 317

43 准备 GLEAN 数据集 319

44 准备 CelebA-HQ 数据集 323

45 准备 GoPro 数据集 325

46 准备 Vimeo90K 数据集 327

47 为 Pix2pix 准备配对数据集 331

48 准备 SIDD 数据集 333

49 准备 DPDD 数据集 335

50 准备 UDM10 数据集 337

51 为 CycleGAN 准备未配对数据集 339

52 准备 HIDE 数据集 341

53 准备 Paris Street View 数据集 343

54 准备 Vid4 数据集 345

55 准备 NTIRE21 decompression 数据集 347

56 准备 Deraining 数据集 349

57 准备 REDS 数据集 351

58 准备 Denoising 数据集 355

59 准备 Composition-1k 数据集 357

60 准备 RealBlur 数据集 361

61 准备 SPMCS 数据集 363

62 准备 Places365 数据集 365

63 准备 LIVE1 数据集 367

64 准备 Vimeo90K-triplet 数据集 369

65 准备 Classic5 数据集 371

66 准备 DIV2K 数据集 373

67 变更日志 377

68 mmagic.apis.inferencers 389

69 mmagic.structures 403

70 mmagic.datasets 407

71 mmagic.datasets.transforms 431

72 mmagic.evaluation 479

73 mmagic.visualization 509

74 mmagic.engine.hooks 519

75 mmagic.engine.optimizers 529

76 mmagic.engine.runner 535

77 mmagic.engine.schedulers 541

78 mmagic.models.archs 545

79 mmagic.models.base_models 567

80 mmagic.models.losses 595

81 mmagic.models.data_preprocessors 623

82 mmagic.models.editors 633

83 mmagic.utils 781

84 概览 791

85 运行设置的迁移 793

86 模型的迁移 795

87 评估与测试设置的迁移 797

88 调度器的迁移 801

89 Data Settings 的迁移 803

90 分布式训练的迁移 809

91 优化器的迁移 811

92 可视化的迁移 815

93 混合精度训练的迁移 817

94 NPU (华为🅵腾) 821

95 English 823

96 简体中文 825

97 Indices and tables 827

Python 模块索引 829

索引 831

您可以在页面左下角切换中英文文档。

备注： 目前英文版有更多的内容，欢迎加入我们一起提升中文文档！您可以通过 issue，discussion 或者我们的社区群来联系我们！

CHAPTER 1

贡献代码

欢迎加入 MMagic 社区，我们致力于打造新一代人工智能内容生成（AIGC）工具箱，我们欢迎任何类型的贡献，包括但不限于

修复错误

修复代码实现错误的步骤如下：

1. 如果提交的代码改动较大，建议先提交 issue，并正确描述 issue 的现象、原因和复现方式，讨论后确认修复方案。
2. 修复错误并补充相应的单元测试，提交拉取请求。

新增功能或组件

1. 如果新功能或模块涉及较大的代码改动，建议先提交 issue，确认功能的必要性。
2. 实现新增功能并添单元测试，提交拉取请求。

文档补充

修复文档可以直接提交拉取请求

添加文档或将文档翻译成其他语言步骤如下

1. 提交 issue，确认添加文档的必要性。
2. 添加文档，提交拉取请求。

1.1 拉取请求工作流

如果你对拉取请求不了解，没关系，接下来的内容将会从零开始，一步一步地指引你如何创建一个拉取请求。如果你想深入了解拉取请求的开发模式，可以参考 github 官方文档

1.1.1 1. 复刻仓库

当你第一次提交拉取请求时，先复刻 OpenMMLab 原代码库，点击 GitHub 页面右上角的 **Fork** 按钮，复刻后的代码库将会出现在你的 GitHub 个人主页下。

将代码克隆到本地

```
git clone git@github.com:{username}/mmagic.git
```

添加原代码库为上游代码库

```
git remote add upstream git@github.com:open-mmlab/mmagic
```

检查 remote 是否添加成功，在终端输入 `git remote -v`

```
origin        git@github.com:{username}/mmagic.git (fetch)
origin        git@github.com:{username}/mmagic.git (push)
upstream        git@github.com:open-mmlab/mmagic (fetch)
upstream        git@github.com:open-mmlab/mmagic (push)
```

备注： 这里对 origin 和 upstream 进行一个简单的介绍，当我们使用 git clone 来克隆代码时，会默认创建一个 origin 的 remote，它指向我们克隆的代码库地址，而 upstream 则是我们自己添加的，用来指向原始代码库地址。当然如果你不喜欢他叫 upstream，也可以自己修改，比如叫 open-mmlab。我们通常向 origin 提交代码（即 fork 下来的远程仓库），然后向 upstream 提交一个 pull request。如果提交的代码和最新的代码发生冲突，再从 upstream 拉取最新的代码，和本地分支解决冲突，再提交到 origin。

1.1.2 2. 配置 pre-commit

在本地开发环境中，我们使用 pre-commit 来检查代码风格，以确保代码风格的统一。在提交代码，需要先安装 pre-commit（需要在 mmagic 目录下执行）：

```
pip install -U pre-commit
pre-commit install
```

检查 pre-commit 是否配置成功，并安装 `.pre-commit-config.yaml` 中的钩子：

```
pre-commit run --all-files
```

备注： 如果你是中国用户，由于网络原因，可能会出现安装失败的情况，这时可以使用国内源

pre-commit install -c .pre-commit-config-zh-cn.yaml

pre-commit run –all-files -c .pre-commit-config-zh-cn.yaml

如果安装过程被中断，可以重复执行 `pre-commit run ...` 继续安装。

如果提交的代码不符合代码风格规范，pre-commit 会发出警告，并自动修复部分错误。

如果我们想临时绕开 pre-commit 的检查提交一次代码，可以在 `git commit` 时加上 `--no-verify`（需要保证最后推送至远程仓库的代码能够通过 pre-commit 检查）。

```
git commit -m "xxx" --no-verify
```

1.1.3 3. 创建开发分支

安装完 pre-commit 之后，我们需要基于 main 创建开发分支，建议的分支命名规则为 `username/pr_name`。

```
git checkout -b yhc/refactor_contributing_doc
```

在后续的开发中，如果本地仓库的 main 分支落后于 upstream 的 main 分支，我们需要先拉取 upstream 的代码进行同步，再执行上面的命令

```
git pull upstream main
```

1.1.4 4. 提交代码并在本地通过单元测试

- mmagic 引入了 mypy 来做静态类型检查，以增加代码的鲁棒性。因此我们在提交代码时，需要补充 Type Hints。具体规则可以参考教程。
- 提交的代码同样需要通过单元测试

```
# 通过全量单元测试
pytest tests

# 我们需要保证提交的代码能够通过修改模块的单元测试，以 runner 为例
pytest tests/test_runner/test_runner.py
```

  如果你由于缺少依赖无法运行修改模块的单元测试，可以参考指引-单元测试

- 如果修改/添加了文档，参考指引确认文档渲染正常。

1.1.5 5. 推送代码到远程

代码通过单元测试和 pre-commit 检查后，将代码推送到远程仓库，如果是第一次推送，可以在 `git push` 后加上 `-u` 参数以关联远程分支

```
git push -u origin {branch_name}
```

这样下次就可以直接使用 `git push` 命令推送代码了，而无需指定分支和远程仓库。

1.1.6 6. 提交拉取请求（PR）

(1) 在 GitHub 的 Pull request 界面创建拉取请求

(2) 根据指引修改 PR 描述，以便于其他开发者更好地理解你的修改

描述规范详见拉取请求规范

注意事项

(a) PR 描述应该包含修改理由、修改内容以及修改后带来的影响，并关联相关 Issue（具体方式见文档）

(b) 如果是第一次为 OpenMMLab 做贡献，需要签署 CLA

(c) 检查提交的 PR 是否通过 CI（集成测试）

mmagic 会在不同的平台（Linux、Window、Mac），基于不同版本的 Python、PyTorch、CUDA 对提交的代码进行单元测试，以保证代码的正确性，如果有任何一个没有通过，我们可点击上图中的 `Details` 来查看具体的测试信息，以便于我们修改代码。

(3) 如果 PR 通过了 CI，那么就可以等待其他开发者的 review，并根据 reviewer 的意见，修改代码，并重复 4-5 步骤，直到 reviewer 同意合入 PR。

所有 reviewer 同意合入 PR 后，我们会尽快将 PR 合并到主分支。

1.1.7 7. 解决冲突

随着时间的推移，我们的代码库会不断更新，这时候，如果你的 PR 与主分支存在冲突，你需要解决冲突，解决冲突的方式有两种：

```
git fetch --all --prune
git rebase upstream/main
```

或者

```
git fetch --all --prune
git merge upstream/main
```

如果你非常善于处理冲突，那么可以使用 rebase 的方式来解决冲突，因为这能够保证你的 commit log 的整洁。如果你不太熟悉 `rebase` 的使用，那么可以使用 `merge` 的方式来解决冲突。

1.2 指引

1.2.1 单元测试

在提交修复代码错误或新增特性的拉取请求时，我们应该尽可能的让单元测试覆盖所有提交的代码，计算单元测试覆盖率的方法如下

```
python -m coverage run -m pytest /path/to/test_file
python -m coverage html
# check file in htmlcov/index.html
```

1.2.2 文档渲染

在提交修复代码错误或新增特性的拉取请求时，可能会需要修改/新增模块的 docstring。我们需要确认渲染后的文档样式是正确的。本地生成渲染后的文档的方法如下

```
pip install -r requirements/docs.txt
cd docs/zh_cn/
# or docs/en
make html
# check file in ./docs/zh_cn/_build/html/index.html
```

1.3 代码风格

1.3.1 Python

PEP8 作为 OpenMMLab 算法库首选的代码规范，我们使用以下工具检查和格式化代码

- flake8: Python 官方发布的代码规范检查工具，是多个检查工具的封装
- isort: 自动调整模块导入顺序的工具
- yapf: Google 发布的代码规范检查工具
- codespell: 检查单词拼写是否有误
- mdformat: 检查 markdown 文件的工具
- docformatter: 格式化 docstring 的工具

yapf 和 isort 的配置可以在 setup.cfg 找到

通过配置 pre-commit hook ，我们可以在提交代码时自动检查和格式化 `flake8`、`yapf`、`isort`、`trailing whitespaces`、`markdown files`，修复 `end-of-files`、`double-quoted-strings`、`python-encoding-pragma`、`mixed-line-ending`，调整 `requirments.txt` 的包顺序。pre-commit 钩子的配置可以在 .pre-commit-config 找到。

pre-commit 具体的安装使用方式见拉取请求。

更具体的规范请参考*OpenMMLab 代码规范*。

1.3.2 C++ and CUDA

C++ 和 CUDA 的代码规范遵从 Google C++ Style Guide

1.4 拉取请求规范

1. 使用 pre-commit hook，尽量减少代码风格相关问题
2. 一个拉取请求对应一个短期分支
3. 粒度要细，一个拉取请求只做一件事情，避免超大的拉取请求
   - Bad：实现 Faster R-CNN
   - Acceptable：给 Faster R-CNN 添加一个 box head
   - Good：给 box head 增加一个参数来支持自定义的 conv 层数
4. 每次 Commit 时需要提供清晰且有意义 commit 信息
5. 提供清晰且有意义的拉取请求描述
   - 标题写明白任务名称，一般格式:[Prefix] Short description of the pull request (Suffix)
   - prefix: 新增功能 [Feature], 修 bug [Fix], 文档相关 [Docs], 开发中 [WIP] (暂时不会被 review)
   - 描述里介绍拉取请求的主要修改内容，结果，以及对其他部分的影响, 参考拉取请求模板
   - 关联相关的议题 (issue) 和其他拉取请求
6. 如果引入了其他三方库，或借鉴了三方库的代码，请确认他们的许可证和 mmagic 兼容，并在借鉴的代码上补充 `This code is inspired from http://`

1.5 代码规范

1.5.1 代码规范标准

PEP 8 ——Python 官方代码规范

Python 官方的代码风格指南，包含了以下几个方面的内容：

- 代码布局，介绍了 Python 中空行、断行以及导入相关的代码风格规范。比如一个常见的问题：当我的代码较长，无法在一行写下时，何处可以断行？
- 表达式，介绍了 Python 中表达式空格相关的一些风格规范。
- 尾随逗号相关的规范。当列表较长，无法一行写下而写成如下逐行列表时，推荐在末项后加逗号，从而便于追加选项、版本控制等。

```
# Correct:
FILES = ['setup.cfg', 'tox.ini']
# Correct:
FILES = [
    'setup.cfg',
    'tox.ini',
]
# Wrong:
FILES = ['setup.cfg', 'tox.ini',]
# Wrong:
FILES = [
    'setup.cfg',
    'tox.ini'
]
```

- 命名相关规范、注释相关规范、类型注解相关规范，我们将在后续章节中做详细介绍。

  “A style guide is about consistency. Consistency with this style guide is important. Consistency within a project is more important. Consistency within one module or function is the most important.” PEP 8 –Style Guide for Python Code

---

备注： PEP 8 的代码规范并不是绝对的，项目内的一致性要优先于 PEP 8 的规范。OpenMMLab 各个项目都在 setup.cfg 设定了一些代码规范的设置，请遵照这些设置。一个例子是在 PEP 8 中有如下一个例子：

```
# Correct:
hypot2 = x*x + y*y
# Wrong:
hypot2 = x * x + y * y
```

这一规范是为了指示不同优先级，但 OpenMMLab 的设置中通常没有启用 yapf 的 ARITHMETIC_PRECEDENCE_INDICATION 选项，因而格式规范工具不会按照推荐样式格式化，以设置为准。

Google 开源项目风格指南

Google 使用的编程风格指南，包括了 Python 相关的章节。相较于 PEP 8，该指南提供了更为详尽的代码指南。该指南包括了语言规范和风格规范两个部分。

其中，语言规范对 Python 中很多语言特性进行了优缺点的分析，并给出了使用指导意见，如异常、Lambda 表达式、列表推导式、metaclass 等。

风格规范的内容与 PEP 8 较为接近，大部分约定建立在 PEP 8 的基础上，也有一些更为详细的约定，如函数长度、TODO 注释、文件与 socket 对象的访问等。

推荐将该指南作为参考进行开发，但不必严格遵照，一来该指南存在一些 Python 2 兼容需求，例如指南中要求所有无基类的类应当显式地继承 Object, 而在仅使用 Python 3 的环境中，这一要求是不必要的，依本项目中的惯例即可。二来 OpenMMLab 的项目作为框架级的开源软件，不必对一些高级技巧过于避讳，尤其是 MMCV。但尝试使用这些技巧前应当认真考虑是否真的有必要，并寻求其他开发人员的广泛评估。

另外需要注意的一处规范是关于包的导入，在该指南中，要求导入本地包时必须使用路径全称，且导入的每一个模块都应当单独成行，通常这是不必要的，而且也不符合目前项目的开发惯例，此处进行如下约定：

```
# Correct
from mmagic.cnn.bricks import (Conv2d, build_norm_layer, DropPath, MaxPool2d,
                               Linear)
from ..utils import ext_loader

# Wrong
from mmagic.cnn.bricks import Conv2d, build_norm_layer, DropPath, MaxPool2d, \
                              Linear  # 使用括号进行连接，而不是反斜杠
from ...utils import is_str  # 最多向上回溯一层，过多的回溯容易导致结构混乱
```

OpenMMLab 项目使用 pre-commit 工具自动格式化代码，详情见贡献代码。

1.5.2 命名规范

命名规范的重要性

优秀的命名是良好代码可读的基础。基础的命名规范对各类变量的命名做了要求，使读者可以方便地根据代码名了解变量是一个类 / 局部变量 / 全局变量等。而优秀的命名则需要代码作者对于变量的功能有清晰的认识，以及良好的表达能力，从而使读者根据名称就能了解其含义，甚至帮助了解该段代码的功能。

基础命名规范

注意：

- 尽量避免变量名与保留字冲突，特殊情况下如不可避免，可使用一个后置下划线，如 class_
- 尽量不要使用过于简单的命名，除了约定俗成的循环变量 i，文件变量 f，错误变量 e 等。
- 不会被用到的变量可以命名为 _，逻辑检查器会将其忽略。

命名技巧

良好的变量命名需要保证三点：

1. 含义准确，没有歧义
2. 长短适中
3. 前后统一

```
# Wrong
class Masks(metaclass=ABCMeta):  # 命名无法表现基类；Instance or Semantic？
    pass

# Correct
class BaseInstanceMasks(metaclass=ABCMeta):
    pass

# Wrong，不同地方含义相同的变量尽量用统一的命名
def __init__(self, inplanes, planes):
    pass

def __init__(self, in_channels, out_channels):
    pass
```

常见的函数命名方法：

- 动宾命名法：crop_img, init_weights
- 动宾倒置命名法：imread, bbox_flip

注意函数命名与参数的顺序，保证主语在前，符合语言习惯：

- check_keys_exist(key, container)
- check_keys_contain(container, key)

注意避免非常规或统一约定的缩写，如 nb -> num_blocks，in_nc -> in_channels

1.5.3 docstring 规范

为什么要写 docstring

docstring 是对一个类、一个函数功能与 API 接口的详细描述，有两个功能，一是帮助其他开发者了解代码功能，方便 debug 和复用代码；二是在 Readthedocs 文档中自动生成相关的 API reference 文档，帮助不了解源代码的社区用户使用相关功能。

如何写 docstring

与注释不同，一份规范的 docstring 有着严格的格式要求，以便于 Python 解释器以及 sphinx 进行文档解析，详细的 docstring 约定参见 PEP 257。此处以例子的形式介绍各种文档的标准格式，参考格式为 Google 风格。

1. 模块文档

   代码风格规范推荐为每一个模块（即 Python 文件）编写一个 docstring，但目前 OpenMMLab 项目大部分没有此类 docstring，因此不做硬性要求。

```
"""A one line summary of the module or program, terminated by a period.

Leave one blank line. The rest of this docstring should contain an
overall description of the module or program. Optionally, it may also
contain a brief description of exported classes and functions and/or usage
examples.

Typical usage example:

foo = ClassFoo()
bar = foo.FunctionBar()
"""
```

2. 类文档

   类文档是我们最常需要编写的，此处，按照 OpenMMLab 的惯例，我们使用了与 Google 风格不同的写法。如下例所示，文档中没有使用 Attributes 描述类属性，而是使用 Args 描述 **init** 函数的参数。

   在 Args 中，遵照 `parameter (type): Description.` 的格式，描述每一个参数类型和功能。其中，多种类型可使用 `(float or str)` 的写法，可以为 None 的参数可以写为 `(int, optional)`。

```
class BaseRunner(metaclass=ABCMeta):
    """The base class of Runner, a training helper for PyTorch.

    All subclasses should implement the following APIs:

    - ``run()``
    - ``train()``
```

(下页继续)

(续上页)

```
    - ``val()``
    - ``save_checkpoint()``

    Args:
        model (:obj:`torch.nn.Module`): The model to be run.
        batch_processor (callable, optional): A callable method that process
            a data batch. The interface of this method should be
            ``batch_processor(model, data, train_mode) -> dict``.
            Defaults to None.
        optimizer (dict or :obj:`torch.optim.Optimizer`, optional): It can be
            either an optimizer (in most cases) or a dict of optimizers
            (in models that requires more than one optimizer, e.g., GAN).
            Defaults to None.
        work_dir (str, optional): The working directory to save checkpoints
            and logs. Defaults to None.
        logger (:obj:`logging.Logger`): Logger used during training.
             Defaults to None. (The default value is just for backward
             compatibility)
        meta (dict, optional): A dict records some import information such as
            environment info and seed, which will be logged in logger hook.
            Defaults to None.
        max_epochs (int, optional): Total training epochs. Defaults to None.
        max_iters (int, optional): Total training iterations. Defaults to None.
    """

    def __init__(self,
                 model,
                 batch_processor=None,
                 optimizer=None,
                 work_dir=None,
                 logger=None,
                 meta=None,
                 max_iters=None,
                 max_epochs=None):
        ...
```

另外，在一些算法实现的主体类中，建议加入原论文的链接；如果参考了其他开源代码的实现，则应加入 modified from，而如果是直接复制了其他代码库的实现，则应加入 copied from ，并注意源码的 License。如有必要，也可以通过.. math:: 来加入数学公式

```
# 参考实现
# This func is modified from `detectron2
# <https://github.com/facebookresearch/detectron2/blob/
↪ffff8acc35ea88ad1cb1806ab0f00b4c1c5dbfd9/detectron2/structures/masks.py#L387>`_.
```

(下页继续)

(续上页)

```
# 复制代码
# This code was copied from the `ubelt
# library<https://github.com/Erotemic/ubelt>`_.

# 引用论文 & 添加公式
class LabelSmoothLoss(nn.Module):
    r"""Initializer for the label smoothed cross entropy loss.

    Refers to `Rethinking the Inception Architecture for Computer Vision
    <https://arxiv.org/abs/1512.00567>`_.

    This decreases gap between output scores and encourages generalization.
    Labels provided to forward can be one-hot like vectors (NxC) or class
    indices (Nx1).
    And this accepts linear combination of one-hot like labels from mixup or
    cutmix except multi-label task.

    Args:
        label_smooth_val (float): The degree of label smoothing.
        num_classes (int, optional): Number of classes. Defaults to None.
        mode (str): Refers to notes, Options are "original", "classy_vision",
            "multi_label". Defaults to "classy_vision".
        reduction (str): The method used to reduce the loss.
            Options are "none", "mean" and "sum". Defaults to 'mean'.
        loss_weight (float):  Weight of the loss. Defaults to 1.0.

    Note:
        if the ``mode`` is "original", this will use the same label smooth
        method as the original paper as:

        .. math::
            (1-\epsilon)\delta_{k, y} + \frac{\epsilon}{K}

        where :math:`\epsilon` is the ``label_smooth_val``, :math:`K` is
        the ``num_classes`` and :math:`\delta_{k,y}` is Dirac delta,
        which equals 1 for k=y and 0 otherwise.

        if the ``mode`` is "classy_vision", this will use the same label
        smooth method as the `facebookresearch/ClassyVision
        <https://github.com/facebookresearch/ClassyVision/blob/main/classy_vision/
↪losses/label_smoothing_loss.py>`_ repo as:
```

(下页继续)

(续上页)

```
        .. math::
            \frac{\delta_{k, y} + \epsilon/K}{1+\epsilon}

        if the ``mode`` is "multi_label", this will accept labels from
        multi-label task and smoothing them as:

        .. math::
            (1-2\epsilon)\delta_{k, y} + \epsilon
```

备注： 注意 “here“、‘here‘、” here” 三种引号功能是不同。

在 reStructured 语法中，“here“ 表示一段代码；‘here‘ 表示斜体；” here” 无特殊含义，一般可用来表示字符串。其中 ‘here‘ 的用法与 Markdown 中不同，需要多加留意。另外还有:obj:‘type‘ 这种更规范的表示类的写法，但鉴于长度，不做特别要求，一般仅用于表示非常用类型。

---

3. 方法（函数）文档

   函数文档与类文档的结构基本一致，但需要加入返回值文档。对于较为复杂的函数和类，可以使用 Examples 字段加入示例；如果需要对参数加入一些较长的备注，可以加入 Note 字段进行说明。

   对于使用较为复杂的类或函数，比起看大段大段的说明文字和参数文档，添加合适的示例更能帮助用户迅速了解其用法。需要注意的是，这些示例最好是能够直接在 Python 交互式环境中运行的，并给出一些相对应的结果。如果存在多个示例，可以使用注释简单说明每段示例，也能起到分隔作用。

```python
def import_modules_from_strings(imports, allow_failed_imports=False):
    """Import modules from the given list of strings.

    Args:
        imports (list | str | None): The given module names to be imported.
        allow_failed_imports (bool): If True, the failed imports will return
            None. Otherwise, an ImportError is raise. Defaults to False.

    Returns:
        List[module] | module | None: The imported modules.
        All these three lines in docstring will be compiled into the same
        line in readthedocs.

    Examples:
        >>> osp, sys = import_modules_from_strings(
        ...     ['os.path', 'sys'])
        >>> import os.path as osp_
        >>> import sys as sys_
        >>> assert osp == osp_
```

(下页继续)

(续上页)

```
        >>> assert sys == sys_
    """
    ...
```

如果函数接口在某个版本发生了变化，需要在 docstring 中加入相关的说明，必要时添加 Note 或者 Warning 进行说明，例如：

```
class CheckpointHook(Hook):
    """Save checkpoints periodically.

    Args:
        out_dir (str, optional): The root directory to save checkpoints. If
            not specified, ``runner.work_dir`` will be used by default. If
            specified, the ``out_dir`` will be the concatenation of
            ``out_dir`` and the last level directory of ``runner.work_dir``.
            Defaults to None. `Changed in version 1.3.15.`
        file_client_args (dict, optional): Arguments to instantiate a
            FileClient. See :class:`mmagic.fileio.FileClient` for details.
            Defaults to None. `New in version 1.3.15.`

    Warning:
        Before v1.3.15, the ``out_dir`` argument indicates the path where the
        checkpoint is stored. However, in v1.3.15 and later, ``out_dir``
        indicates the root directory and the final path to save checkpoint is
        the concatenation of out_dir and the last level directory of
        ``runner.work_dir``. Suppose the value of ``out_dir`` is
        "/path/of/A" and the value of ``runner.work_dir`` is "/path/of/B",
        then the final path will be "/path/of/A/B".
```

如果参数或返回值里带有需要展开描述字段的 dict，则应该采用如下格式：

```
def func(x):
    r"""
    Args:
        x (None): A dict with 2 keys, ``padded_targets``, and ``targets``.

            - ``targets`` (list[Tensor]): A list of tensors.
              Each tensor has the shape of :math:`(T_i)`. Each
              element is the index of a character.
            - ``padded_targets`` (Tensor): A tensor of shape :math:`(N)`.
              Each item is the length of a word.

    Returns:
        dict: A dict with 2 keys, ``padded_targets``, and ``targets``.
```

(下页继续)

(续上页)

```
        - ``targets`` (list[Tensor]): A list of tensors.
          Each tensor has the shape of :math:`(T_i)`. Each
          element is the index of a character.
        - ``padded_targets`` (Tensor): A tensor of shape :math:`(N)`.
          Each item is the length of a word.
    """
    return x
```

重要： 为了生成 readthedocs 文档，文档的编写需要按照 ReStructrued 文档格式，否则会产生文档渲染错误，在提交 PR 前，最好生成并预览一下文档效果。语法规范参考：

- reStructuredText Primer - Sphinx documentation
- Example Google Style Python Docstrings – napoleon 0.7 documentation

1.5.4 注释规范

为什么要写注释

对于一个开源项目，团队合作以及社区之间的合作是必不可少的，因而尤其要重视合理的注释。不写注释的代码，很有可能过几个月自己也难以理解，造成额外的阅读和修改成本。

如何写注释

最需要写注释的是代码中那些技巧性的部分。如果你在下次代码审查的时候必须解释一下，那么你应该现在就给它写注释。对于复杂的操作，应该在其操作开始前写上若干行注释。对于不是一目了然的代码，应在其行尾添加注释。——Google 开源项目风格指南

```
# We use a weighted dictionary search to find out where i is in
# the array. We extrapolate position based on the largest num
# in the array and the array size and then do binary search to
# get the exact number.
if i & (i-1) == 0:  # True if i is 0 or a power of 2.
```

为了提高可读性, 注释应该至少离开代码 2 个空格. 另一方面, 绝不要描述代码. 假设阅读代码的人比你更懂 Python, 他只是不知道你的代码要做什么. ——Google 开源项目风格指南

```
# Wrong:
# Now go through the b array and make sure whenever i occurs
# the next element is i+1
```

(下页继续)

(续上页)

```
# Wrong:
if i & (i-1) == 0:  # True if i bitwise and i-1 is 0.
```

在注释中，可以使用 Markdown 语法，因为开发人员通常熟悉 Markdown 语法，这样可以便于交流理解，如可使用单反引号表示代码和变量（注意不要和 docstring 中的 ReStructured 语法混淆）

```
# `_reversed_padding_repeated_twice` is the padding to be passed to
# `F.pad` if needed (e.g., for non-zero padding types that are
# implemented as two ops: padding + conv). `F.pad` accepts paddings in
# reverse order than the dimension.
self._reversed_padding_repeated_twice = _reverse_repeat_tuple(self.padding, 2)
```

注释示例

1. 出自 mmcv/utils/registry.py，对于较为复杂的逻辑结构，通过注释，明确了优先级关系。

```
# self.build_func will be set with the following priority:
# 1. build_func
# 2. parent.build_func
# 3. build_from_cfg
if build_func is None:
    if parent is not None:
        self.build_func = parent.build_func
    else:
        self.build_func = build_from_cfg
else:
    self.build_func = build_func
```

2. 出自 mmcv/runner/checkpoint.py，对于 bug 修复中的一些特殊处理，可以附带相关的 issue 链接，帮助其他人了解 bug 背景。

```
def _save_ckpt(checkpoint, file):
    # The 1.6 release of PyTorch switched torch.save to use a new
    # zipfile-based file format. It will cause RuntimeError when a
    # checkpoint was saved in high version (PyTorch version>=1.6.0) but
    # loaded in low version (PyTorch version<1.6.0). More details at
    # https://github.com/open-mmlab/mmpose/issues/904
    if digit_version(TORCH_VERSION) >= digit_version('1.6.0'):
        torch.save(checkpoint, file, _use_new_zipfile_serialization=False)
    else:
        torch.save(checkpoint, file)
```

1.5.5 类型注解

为什么要写类型注解

类型注解是对函数中变量的类型做限定或提示，为代码的安全性提供保障、增强代码的可读性、避免出现类型相关的错误。Python 没有对类型做强制限制，类型注解只起到一个提示作用，通常你的 IDE 会解析这些类型注解，然后在你调用相关代码时对类型做提示。另外也有类型注解检查工具，这些工具会根据类型注解，对代码中可能出现的问题进行检查，减少 bug 的出现。需要注意的是，通常我们不需要注释模块中的所有函数：

1. 公共的 API 需要注释
2. 在代码的安全性，清晰性和灵活性上进行权衡是否注释
3. 对于容易出现类型相关的错误的代码进行注释
4. 难以理解的代码请进行注释
5. 若代码中的类型已经稳定，可以进行注释. 对于一份成熟的代码，多数情况下，即使注释了所有的函数，也不会丧失太多的灵活性.

如何写类型注解

1. 函数 / 方法类型注解，通常不对 self 和 cls 注释。

```
from typing import Optional, List, Tuple

# 全部位于一行
def my_method(self, first_var: int) -> int:
    pass

# 另起一行
def my_method(
        self, first_var: int,
        second_var: float) -> Tuple[MyLongType1, MyLongType1, MyLongType1]:
    pass

# 单独成行（具体的应用场合与行宽有关，建议结合 yapf 自动化格式使用）
def my_method(
    self, first_var: int, second_var: float
) -> Tuple[MyLongType1, MyLongType1, MyLongType1]:
    pass

# 引用尚未被定义的类型
class MyClass:
    def __init__(self,
```

(下页继续)

(续上页)

```
                stack: List["MyClass"]) -> None:
        pass
```

注：类型注解中的类型可以是 Python 内置类型，也可以是自定义类，还可以使用 Python 提供的 wrapper 类对类型注解进行装饰，一些常见的注解如下：

```
# 数值类型
from numbers import Number

# 可选类型，指参数可以为 None
from typing import Optional
def foo(var: Optional[int] = None):
    pass

# 联合类型，指同时接受多种类型
from typing import Union
def foo(var: Union[float, str]):
    pass

from typing import Sequence  # 序列类型
from typing import Iterable  # 可迭代类型
from typing import Any  # 任意类型
from typing import Callable  # 可调用类型

from typing import List, Dict  # 列表和字典的泛型类型
from typing import Tuple  # 元组的特殊格式
# 虽然在 Python 3.9 中，list, tuple 和 dict 本身已支持泛型，但为了支持之前的版本
# 我们在进行类型注解时还是需要使用 List, Tuple, Dict 类型
# 另外，在对参数类型进行注解时，尽量使用 Sequence & Iterable & Mapping
# List, Tuple, Dict 主要用于返回值类型注解
# 参见 https://docs.python.org/3/library/typing.html#typing.List
```

2. 变量类型注解，一般用于难以直接推断其类型时

```
# Recommend: 带类型注解的赋值
a: Foo = SomeUndecoratedFunction()
a: List[int]: [1, 2, 3]          # List 只支持单一类型泛型，可使用 Union
b: Tuple[int, int] = (1, 2)      # 长度固定为 2
c: Tuple[int, ...] = (1, 2, 3)   # 变长
d: Dict[str, int] = {'a': 1, 'b': 2}

# Not Recommend: 行尾类型注释
# 虽然这种方式被写在了 Google 开源指南中，但这是一种为了支持 Python 2.7 版本
# 而补充的注释方式，鉴于我们只支持 Python 3，为了风格统一，不推荐使用这种方式。
```

(下页继续)

(续上页)

```
a = SomeUndecoratedFunction()  # type: Foo
a = [1, 2, 3]  # type: List[int]
b = (1, 2, 3)  # type: Tuple[int, ...]
c = (1, "2", 3.5)  # type: Tuple[int, Text, float]
```

3. 泛型

上文中我们知道，typing 中提供了 list 和 dict 的泛型类型，那么我们自己是否可以定义类似的泛型呢?

```
from typing import TypeVar, Generic

KT = TypeVar('KT')
VT = TypeVar('VT')

class Mapping(Generic[KT, VT]):
    def __init__(self, data: Dict[KT, VT]):
        self._data = data

    def __getitem__(self, key: KT) -> VT:
        return self._data[key]
```

使用上述方法，我们定义了一个拥有泛型能力的映射类，实际用法如下:

```
mapping = Mapping[str, float]({'a': 0.5})
value: float = example['a']
```

另外，我们也可以利用 TypeVar 在函数签名中指定联动的多个类型:

```
from typing import TypeVar, List

T = TypeVar('T')  # Can be anything
A = TypeVar('A', str, bytes)  # Must be str or bytes


def repeat(x: T, n: int) -> List[T]:
    """Return a list containing n references to x."""
    return [x]*n


def longest(x: A, y: A) -> A:
    """Return the longest of two strings."""
    return x if len(x) >= len(y) else y
```

更多关于类型注解的写法请参考 typing。

类型注解检查工具

mypy 是一个 Python 静态类型检查工具。根据你的类型注解，mypy 会检查传参、赋值等操作是否符合类型注解，从而避免可能出现的 bug。

例如如下的一个 Python 脚本文件 test.py:

```
def foo(var: int) -> float:
    return float(var)

a: str = foo('2.0')
b: int = foo('3.0')  # type: ignore
```

运行 mypy test.py 可以得到如下检查结果，分别指出了第 4 行在函数调用和返回值赋值两处类型错误。而第 5 行同样存在两个类型错误，由于使用了 type: ignore 而被忽略了，只有部分特殊情况可能需要此类忽略。

```
test.py:4: error: Incompatible types in assignment (expression has type "float",␣
↪variable has type "int")
test.py:4: error: Argument 1 to "foo" has incompatible type "str"; expected "int"
Found 2 errors in 1 file (checked 1 source file)
```

CHAPTER 2

生态项目

欢迎来到 MMagic 社区!

MMagic 社区由来自学术界和工业界的广大研究人员、机器学习和应用工程师编写的教程、库和项目组成。

该社区的目标是支持、加速和帮助您使用 MMagic 探索 AIGC，例如图像、视频、3D 内容生成、编辑和处理。

这里有一些基于 MMagic 的项目。它们是如何将 MMagic 用作库的示例，以使您的项目更易于维护。请在MMagic Ecosystem中找到更多项目。

2.1 在 OpenMMLab 社区上展示您的项目

您可以提交您的项目，以便它可以显示在OpenMMLab的主页上。

2.2 添加示例项目到 MMagic

这是一个关于如何将项目添加到 MMagic 的示例项目。

您可以从 example project 复制并创建自己的项目。

我们还提供了下面列出的一些文档供您参考:

- 贡献指南

  新贡献者指南，了解如何将您的项目添加到 MMagic。

- 新模型指南

  添加新模型的文档。

- 讨论

  欢迎开始讨论！

2.3 库和工具箱的项目

- PowerVQE：基于 PyTorch 和 MMagic 的压缩视频质量增强开放框架
- VR-Baseline：视频修复工具箱
- Derain-Toolbox：单图像去雨工具箱和基准

2.4 研究论文项目

- Towards Interpretable Video Super-Resolution via Alternating Optimization, ECCV 2022[github]
- SepLUT:Separable Image-adaptive Lookup Tables for Real-time Image Enhancement, ECCV 2022[github]
- TTVSR: Learning Trajectory-Aware Transformer for Video Super-Resolution, CVPR 2022[github]
- Arbitrary-Scale Image Synthesis, CVPR 2022[github]
- Investigating Tradeoffs in Real-World Video Super-Resolution(RealBasicVSR), CVPR 2022[github]
- BasicVSR++: Improving Video Super-Resolution with Enhanced Propagation and Alignment, CVPR 2022[github]
- Multi-Scale Memory-Based Video Deblurring, CVPR 2022[github]
- AdaInt:Learning Adaptive Intervals for 3D Lookup Tables on Real-time Image Enhancement, CVPR 2022[github]
- A New Dataset and Transformer for Stereoscopic Video Super-Resolution, CVPRW 2022[github]
- Liquid warping GAN with attention: A unified framework for human image synthesis, TPAMI 2021[github]
- BasicVSR:The Search for Essential Components in Video Super-Resolution and Beyond, CVPR 2021[github]
- GLEAN:Generative Latent Bank for Large-Factor Image Super-Resolution, CVPR 2021[github]
- DAN:Unfolding the Alternating Optimization for Blind Super Resolution, NeurIPS 2020[github]

CHAPTER 3

概述

欢迎来到 MMagic！在本节中，您将了解

- *MMagic 是什么？*
- *为什么要使用 MMagic？*
- 新手入门
- 基础教程
- 进阶教程

3.1 MMagic 是什么？

MMagic (**M**ultimodal **A**dvanced, **G**enerative, and **I**ntelligent **C**reation) 是一个供专业人工智能研究人员和机器学习工程师去处理、编辑和生成图像与视频的开源 AIGC 工具箱。

MMagic 允许研究人员和工程师使用最先进的预训练模型，并且可以轻松训练和开发新的定制模型。

MMagic 支持各种基础生成模型，包括：

- 无条件生成对抗网络 (GANs)
- 条件生成对抗网络 (GANs)
- 内部学习
- 扩散模型

- 还有许多其他生成模型即将推出!

MMagic 支持各种应用程序，包括:

- 图文生成
- 图像翻译
- 3D 生成
- 图像超分辨率
- 视频超分辨率
- 视频插帧
- 图像补全
- 图像抠图
- 图像修复
- 图像上色
- 图像生成
- 还有许多其他应用程序即将推出!

3.2 为什么要使用 MMagic ?

- **SOTA 算法**

  MMagic 提供了处理、编辑、生成图像和视频的 SOTA 算法。

- **强有力且流行的应用**

  MMagic 支持了流行的图像修复、图文生成、3D 生成、图像修补、抠图、超分辨率和生成等任务的应用。特别是 MMagic 支持了 Stable Diffusion 的微调和许多激动人心的 diffusion 应用，例如 ControlNet 动画生成。MMagic 也支持了 GANs 的插值，投影，编辑和其他流行的应用。请立即开始你的 AIGC 探索之旅!

- **高效的框架**

  通过 OpenMMLab 2.0 框架的 MMEngine 和 MMCV，MMagic 将编辑框架分解为不同的组件，并且可以通过组合不同的模块轻松地构建自定义的编辑器模型。我们可以像搭建“乐高”一样定义训练流程，提供丰富的组件和策略。在 MMagic 中，你可以使用不同的 APIs 完全控制训练流程。得益于 MMSeparate-DistributedDataParallel, 动态模型结构的分布式训练可以轻松实现。

3.3 新手入门

安装说明见安装。

3.4 基础教程

对于初学者，我们建议从基础教程 学习 MMagic 的基本用法。

3.5 进阶教程

对于熟悉 MMagic 的用户，可能想了解 MMagic 的进阶实用，以及如何扩展算法库，如何使用多个算法库框架等高级用法，请参考进阶教程。

3.6 开发指南

想要使用 MMagic 进行深度开发的用户，可以参考开发指南。

CHAPTER 4

安装教程

在本节中，你将了解到：

- 安装教程
  - 安装
    - 前提条件
    - 最佳实践
    - 自定义安装
      - CUDA 版本
      - 不使用 MIM 安装 MMCV
      - 在 Docker 中使用 MMagic
      - 问题解决
    - 使用多个 *MMagic* 版本开发

4.1 安装

我们建议用户按照我们的最佳实践来安装 MMagic。然而，整个过程是高度可定制的。更多信息请参阅自定义安装部分。

4.1.1 前提条件

在本节中，我们将演示如何使用 PyTorch 准备环境。

MMagic 可以在 Linux, Windows, 和 macOS 上运行。它要求：

- Python >= 3.7
- PyTorch >= 1.8
- MMCV >= 2.0.0

如果您对 PyTorch 有经验并且已经安装了它，直接跳过这一部分，跳到下一节。否则, 您可以按照以下步骤来准备环境。

Step 0. 从官方网站下载和安装 Miniconda.

Step 1. 创建一个conda 虚拟环境并激活它

```
conda create --name mmagic python=3.8 -y
conda activate mmagic
```

Step 2. 按照官方说明安装 PyTorch，例如

- 在 GPU 平台上：

  ```
  conda install pytorch torchvision cudatoolkit=11.3 -c pytorch
  ```

- 在 CPU 平台上：

  ```
  conda install pytorch=1.10 torchvision cpuonly -c pytorch
  ```

4.1.2 最佳实践

Step 0. 使用MIM安装MMCV。

```
pip install -U openmim
mim install 'mmcv>=2.0.0'
```

Step 1. 安装MMEngine。

```
mim install 'mmengine'
```

或者

```
pip install mmengine
```

或者

```
pip install git+https://github.com/open-mmlab/mmengine.git
```

Step 2. 安装 MMagic。

```
mim install 'mmagic'
```

或者

```
pip install mmagic
```

或者从源代码安装MMagic。

```
git clone https://github.com/open-mmlab/mmagic.git
cd mmagic
pip3 install -e . -v
```

Step 5. 检查 MMagic 是否安装成功。

```
cd ~
python -c "import mmagic; print(mmagic.__version__)"
# 示例输出: 1.0.0
```

显示正确的版本号，则表示安装成功。

备注: 你可能想知道附加在 `pip install` 后面的`-e .`是什么意思。下面是说明:

- `-e` 表示可编辑模式. 当 `import mmagic` 时，将导入克隆目录下的模块。如果 `pip install` 没有附加`-e`, pip 会将克隆的代码复制到类似 `lib/python/site-package` 的地方。因此，除非再次执行 `pip install` 命令，否则在克隆目录下修改后的代码不会生效。因此，`pip install` 命令附带`-e` 对于开发人员来说特别方便。如果修改了一些代码，下次导入新的代码时不需要重新安装。
- `.` 表示此目录中的代码。

你也可以使用 `pip install -e .[all]` 命令，这将安装更多的依赖项，特别是对于预提交 hooks 和单元测试。

4.1.3 自定义安装

CUDA 版本

安装 PyTorch 时, 您需要指定 CUDA 的版本。如果您不清楚该选择哪一个，请遵循我们的建议:

- 对于基于 Ampere 的 NVIDIA GPUs，如 GeForce 30 系列和 NVIDIA A100，必须使用 CUDA 11。
- 对于较老的 NVIDIA GPUs，是向后兼容的，但 CUDA 10.2 提供了更好的兼容性，更轻量。

请确保 GPU 驱动程序满足最低版本要求。更多信息请参见此表。

注意如果遵循我们的最佳实践，安装 CUDA runtime 库就足够了，因为不会在本地编译 CUDA 代码。但是，如果您希望从源代码编译 MMCV 或开发其他 CUDA 算子，则需要从 NVIDIA 的开发者网站安装完整的 CUDA 工具包，其版本应与 PyTorch 的 CUDA 版本匹配。即，在 `conda install` 命令中指定的 cudatoolkit 版本。

不使用 MIM 安装 MMCV

MMCV 包含 c++ 和 CUDA 扩展，因此以一种复杂的方式依赖于 PyTorch。MIM 自动解决了这种依赖关系，并使安装更容易。然而，这并不是必须的。

要使用 pip 而不是 MIM 安装 MMCV，请遵循MMCV 安装指南。这需要根据 PyTorch 版本及其 CUDA 版本手动指定 find-url。

例如，以下命令 install mmcv-full 是针对 PyTorch 1.10.x 和 CUDA 11.3 构建的。

```
pip install 'mmcv>=2.0.0' -f https://download.openmmlab.com/mmcv/dist/cu113/torch1.10/
↪index.html
```

在 Docker 中使用 MMagic

我们提供一个Dockerfile来构建一个镜像。请确保您的docker 版本>=19.03。

```
# 使用PyTorch 1.8, CUDA 11.1构建一个镜像
# 如果您喜欢其他版本，只需修改Dockerfile
docker build -t mmagic docker/
```

使用如下命令运行

```
docker run --gpus all --shm-size=8g -it -v {DATA_DIR}:/mmagic/data mmagic
```

问题解决

如果在安装过程中遇到问题，请先查看*FAQ*页面。如果找不到解决方案，可以在 GitHub 上open an issue。

4.1.4 使用多个 MMagic 版本开发

训练和测试脚本已经修改了 `PYTHONPATH`，以确保脚本使用当前目录中的 `MMagic`。

要使用环境中安装的默认 MMagic，而不是您正在使用的 MMagic，可以删除这些脚本中的以下行

```
PYTHONPATH="$(dirname $0)/..":$PYTHONPATH
```

CHAPTER 5

快速运行

成功安装 MMagic 后，现在您可以玩转 MMagic 了！如果您要从文本生成图像，只需要 MMagic 的几行代码！

```
from mmagic.apis import MMagicInferencer
sd_inferencer = MMagicInferencer(model_name='stable_diffusion')
text_prompts = 'A panda is having dinner at KFC'
result_out_dir = 'output/sd_res.png'
sd_inferencer.infer(text=text_prompts, result_out_dir=result_out_dir)
```

或者您可以运行以下命令。

```
python demo/mmagic_inference_demo.py \
    --model-name stable_diffusion \
    --text "A panda is having dinner at KFC" \
    --result-out-dir ./output/sd_res.png
```

您将在文件夹 `output/`中看到一个新图像 `sd_res.png`，其中包含生成的样本。

更重要的是，如果您想让这些照片更清晰，MMagic 的超分辨率只需要几行代码！

```
from mmagic.apis import MMagicInferencer
config = 'configs/esrgan/esrgan_x4c64b23g32_1xb16-400k_div2k.py'
checkpoint = 'https://download.openmmlab.com/mmediting/restorers/esrgan/esrgan_
↪x4c64b23g32_1x16_400k_div2k_20200508-f8ccaf3b.pth'
img_path = 'tests/data/image/lq/baboon_x4.png'
editor = MMagicInferencer('esrgan', model_config=config, model_ckpt=checkpoint)
output = editor.infer(img=img_path,result_out_dir='output.png')
```

现在，您可以在 `output.png` 中查看您想要的图片。

CHAPTER 6

教程 1 了解 MMagic 的配置文件

我们在我们的配置系统中采用了模块化和继承设计，方便进行各种实验。如果您希望查看配置文件，您可以运行 `python tools/misc/print_config.py /PATH/TO/CONFIG` 来查看完整的配置。

您可以根据以下教程了解我们配置系统的使用方法。

- 教程 1：了解 MMagic 中的配置
  - 通过脚本参数修改配置
  - 配置文件结构
  - 配置文件命名风格
  - EDSR 的示例
    - 模型配置
    - 数据配置
      - 数据流程
      - 数据加载器
    - 评估配置
    - 训练和测试配置
    - 优化配置
    - 钩子配置
    - 运行时配置

  - StyleGAN2 的示例
    * 模型配置
    * 数据集和评估器配置
    * 训练和测试配置
    * 优化配置
    * 钩子配置
    * 运行时配置
  - 其他示例
    * 修复任务的配置示例
    * 抠图任务的配置示例
    * 恢复任务的配置示例

6.1 通过脚本参数修改配置

使用 `tools/train.py` 或 `tools/test.py` 来运行时，您可以通过指定 `--cfg-options` 来临时修改配置。

- 更新字典链中的配置键

  可以按照原始配置中字典键的顺序指定配置选项。例如，`--cfg-options test_cfg.use_ema=False` 将默认的采样模型更改为原始生成器，`--cfg-options train_dataloader.batch_size=8` 将训练数据加载器的批大小更改为 8。

- 更新配置列表中的键

  您的配置中有些配置字典是作为列表组成的。例如，训练流程 `train_dataloader.dataset.pipeline` 通常是一个列表，例如 `[dict(type='LoadImageFromFile'), ...]`。如果您想要在流程中将 `'LoadImageFromFile'` 更改为 `'LoadImageFromWebcam'`，可以指定 `--cfg-options train_dataloader.dataset.pipeline.0.type=LoadImageFromWebcam`。训练流程 `train_pipeline` 通常也是一个列表，例如 `[dict(type='LoadImageFromFile'), ...]`。如果您想要将 `'LoadImageFromFile'` 更改为 `'LoadMask'`，可以指定 `--cfg-options train_pipeline.0.type=LoadMask`。

- 更新列表/元组的值

  如果要更新的值是列表或元组，您可以设置 `--cfg-options key="[a,b]"` 或 `--cfg-options key=a,b`。它还允许嵌套的列表/元组值，例如 `--cfg-options key="[(a,b),(c,d)]"`。请注意，为了支持列表/元组数据类型，引号 " 是必需的，并且在指定的值内引号之间不允许有空格。

6.2 配置文件结构

在 config/_base_ 目录下有三种基本组件类型：数据集（datasets）、模型（models）和默认运行时（default_runtime）。许多方法都可以通过使用其中的每种组件之一进行简单构建，例如 AOT-GAN、EDVR、GLEAN、StyleGAN2、CycleGAN、SinGAN 等。由 _base_ 组件组成的配置被称为原始配置。

对于同一文件夹下的所有配置文件，建议只有一个原始配置。所有其他配置文件都应该继承自原始配置。同时，最大的继承层级为 3。

为了便于理解，我们建议贡献者从现有方法中继承。例如，如果基于 BasicVSR 进行了某些修改，用户可以通过在配置文件中指定 _base_ = ../basicvsr/basicvsr_reds4.py 来首先继承基本的 BasicVSR 结构，然后修改配置文件中的必要字段。如果基于 StyleGAN2 进行了某些修改，用户可以通过在配置文件中指定 _base_ = ../styleganv2/stylegan2_c2_ffhq_256_b4x8_800k.py 来首先继承基本的 StyleGAN2 结构，然后修改配置文件中的必要字段。

如果您正在构建一种完全不与任何现有方法共享结构的全新方法，您可以在 configs 目录下创建一个名为 xxx 的文件夹。

详细的文档请参考MMEngine。

6.3 配置文件命名风格

配置文件按照下面的风格命名。我们建议社区贡献者使用同样的风格。

```
{model}_[module setting]_{training schedule}_{dataset}
```

{xxx} 是必填字段，[yyy] 是可选字段。

- {model}：模型类型，如 stylegan、dcgan、basicvsr、dim 等。原始论文中提到的设置也包含在此字段中（例如 Stylegan2-config-f、edvrm 的 edvrm_8xb4-600k_reds）。
- [module setting]：某些模块的具体设置，包括 Encoder、Decoder、Generator、Discriminator、Normalization、loss、Activation 等。例如 c64n7 的 basicvsr-pp_c64n7_8xb1-600k_reds4，dcgan 的学习率 Glr4e-4_Dlr1e-4，stylegan3 的 gamma32.8，sagan 中的 woReLUInplace。在这个部分，来自不同子模块（例如 generator 和 discriminator）的信息用 _ 连接起来。
- {training_scheduler}：训练的特定设置，包括批量大小、训练计划等。例如，学习率（例如 lr1e-3），使用的 GPU 数量和批量大小（例如 8xb32），总迭代次数（例如 160kiter）或在 discriminator 中显示的图像数量（例如 12Mimgs）。
- {dataset}：数据集名称和数据大小信息，例如 deepfillv1_4xb4_celeba-256x256 的 celeba-256x256，basicvsr_2xb4_reds4 的 reds4，ffhq，lsun-car，celeba-hq。

6.4 EDSR 的示例

为了帮助用户对完整的配置文件有一个基本的了解，我们对我们实现的EDSR 模型的配置文件 进行简要说明，如下所示。关于每个模块的更详细用法和相应的替代方案，请参考 API 文档和MMEngine 中的教程。

6.4.1 模型配置

在 MMagic 的配置文件中，我们使用 `model` 字段来设置模型。

```
model = dict(
    type='BaseEditModel',  # 模型的名称
    generator=dict(  # 生成器的配置
        type='EDSRNet',  # 生成器的类型
        in_channels=3,  # 输入的通道数
        out_channels=3,  # 输出的通道数
        mid_channels=64,  # 中间特征的通道数
        num_blocks=16,  # 主干网络中的块数
        upscale_factor=scale,  # 上采样因子
        res_scale=1,  # 用于缩放残差块中的残差
        rgb_mean=(0.4488, 0.4371, 0.4040),  # RGB图像的均值
        rgb_std=(1.0, 1.0, 1.0)),  # RGB图像的标准差
    pixel_loss=dict(type='L1Loss', loss_weight=1.0, reduction='mean'),  # 配置像素损失
    train_cfg=dict(),  # 训练模型的配置
    test_cfg=dict(),  # 测试模型的配置
    data_preprocessor=dict(  # 数据预处理器的配置
        type='DataPreprocessor', mean=[0., 0., 0.], std=[255., 255., 255.])
)
```

6.4.2 数据配置

训练、验证和测试运行器 (runner) 都需要使用数据加载器 (Dataloader)。为了构建数据加载器，需要设置数据集 (Dataset) 和数据处理流程 (data pipeline)。由于这部分的复杂性，我们使用中间变量来简化数据加载器配置的编写。

数据流程

```
train_pipeline = [  # 训练数据处理流程
    dict(type='LoadImageFromFile',  # 从文件中加载图像
         key='img',  # 在结果中查找对应路径的键名
         color_type='color',  # 图像的颜色类型
         channel_order='rgb',  # 图像的通道顺序
```

(下页继续)

(续上页)

```
        imdecode_backend='cv2'),  # 解码后端
    dict(type='LoadImageFromFile',  # 从文件中加载图像
        key='gt',  # 在结果中查找对应路径的键名
        color_type='color',  # 图像的颜色类型
        channel_order='rgb',  # 图像的通道顺序
        imdecode_backend='cv2'),  # 解码后端
    dict(type='SetValues', dictionary=dict(scale=scale)),  # 将值设置给目标键名
    dict(type='PairedRandomCrop', gt_patch_size=96),  # 随机裁剪成配对图像
    dict(type='Flip',  # 翻转图像
        keys=['lq', 'gt'],  # 需要翻转的图像键名
        flip_ratio=0.5,  # 翻转比例
        direction='horizontal'),  # 翻转方向
    dict(type='Flip',  # 翻转图像
        keys=['lq', 'gt'],  # 需要翻转的图像键名
        flip_ratio=0.5,  # 翻转比例
        direction='vertical'),  # 翻转方向
    dict(type='RandomTransposeHW',  # 随机交换图像的宽高
        keys=['lq', 'gt'],  # 需要交换的图像键名
        transpose_ratio=0.5  # 交换比例
        ),
    dict(type='PackInputs')  # 从当前处理流程中收集数据的配置
]

test_pipeline = [  # 测试数据处理流程
    dict(type='LoadImageFromFile',  # 从文件中加载图像
        key='img',  # 在结果中查找对应路径的键名
        color_type='color',  # 图像的颜色类型
        channel_order='rgb',  # 图像的通道顺序
        imdecode_backend='cv2'),  # 解码后端
    dict(type='LoadImageFromFile',  # 从文件中加载图像
        key='gt',  # 在结果中查找对应路径的键名
        color_type='color',  # 图像的颜色类型
        channel_order='rgb',  # 图像的通道顺序
        imdecode_backend='cv2'),  # 解码后端
    dict(type='PackInputs')  # 从当前处理流程中收集数据的配置
]
```

数据加载器

```
dataset_type = 'BasicImageDataset'  # 数据集的类型
data_root = 'data'  # 数据的根路径

train_dataloader = dict(
    num_workers=4,  # 每个 GPU 预取数据的工作进程数
    persistent_workers=False,  # 是否保持工作进程中的数据集实例处于活动状态
    sampler=dict(type='InfiniteSampler', shuffle=True),  # 数据采样器的类型
    dataset=dict(  # 训练数据集配置
        type=dataset_type,  # 数据集的类型
        ann_file='meta_info_DIV2K800sub_GT.txt',  # 注释文件的路径
        metainfo=dict(dataset_type='div2k', task_name='sisr'),
        data_root=data_root + '/DIV2K',  # 数据的根路径
        data_prefix=dict(  # 图像路径的前缀
            img='DIV2K_train_LR_bicubic/X2_sub', gt='DIV2K_train_HR_sub'),
        filename_tmpl=dict(img='{}', gt='{}'),  # 文件名模板
        pipeline=train_pipeline)
)

val_dataloader = dict(
    num_workers=4,  # 每个 GPU 预取数据的工作进程数
    persistent_workers=False,  # 是否保持工作进程中的数据集实例处于活动状态
    drop_last=False,  # 是否丢弃最后一个不完整的批次
    sampler=dict(type='DefaultSampler', shuffle=False),  # 数据采样器的类型
    dataset=dict(  # 验证数据集配置
        type=dataset_type,  # 数据集的类型
        metainfo=dict(dataset_type='set5', task_name='sisr'),
        data_root=data_root + '/Set5',  # 数据的根路径
        data_prefix=dict(img='LRbicx2', gt='GTmod12'),  # 图像路径的前缀
        pipeline=test_pipeline)
)

test_dataloader = val_dataloader
```

6.4.3 评估配置

评估器用于计算在验证集和测试集上训练模型的指标。评估器的配置包括一个或多个指标配置：

```
val_evaluator = [
    dict(type='MAE'),  # 要评估的指标名称
    dict(type='PSNR', crop_border=scale),  # 要评估的指标名称
    dict(type='SSIM', crop_border=scale),  # 要评估的指标名称
```

(下页继续)

(续上页)

```
]

test_evaluator = val_evaluator  # 测试评估器的配置与验证评估器相同
```

6.4.4 训练和测试配置

MMEngine 的运行器使用 Loop 来控制训练、验证和测试过程。用户可以使用这些字段设置最大训练迭代次数和验证间隔。

```
train_cfg = dict(
    type='IterBasedTrainLoop',  # 训练循环类型的名称
    max_iters=300000,  # 总迭代次数
    val_interval=5000,  # 验证间隔迭代次数
)
val_cfg = dict(type='ValLoop')  # 验证循环类型的名称
test_cfg = dict(type='TestLoop')  # 测试循环类型的名称
```

6.4.5 优化配置

`optim_wrapper` 是用于配置优化相关设置的字段。优化器包装器不仅提供优化器的功能，还支持梯度裁剪、混合精度训练等功能。在optimizer wrapper 教程 中可以了解更多信息。

```
optim_wrapper = dict(
    type='OptimWrapper',
    optimizer=dict(type='Adam', lr=0.00001),
)  # 用于构建优化器的配置，支持所有与PyTorch中参数相同的优化器。
```

`param_scheduler` 是一个配置优化超参数（如学习率和动量）调整方法的字段。用户可以结合多个调度器来创建所需的参数调整策略。在parameter scheduler 教程中可以了解更多信息。

```
param_scheduler = dict(  # 学习策略的配置
    type='MultiStepLR', by_epoch=False, milestones=[200000], gamma=0.5)
```

6.4.6 钩子配置

用户可以将钩子（hooks）附加到训练、验证和测试循环中，在运行过程中插入一些操作。有两个不同的钩子字段，一个是 default_hooks，另一个是 custom_hooks。

default_hooks 是一个包含钩子配置的字典。default_hooks 是运行时必需的钩子，它们具有默认的优先级，不应修改。如果未设置，默认值将被使用。要禁用默认钩子，用户可以将其配置设置为 None。

```
default_hooks = dict(  # 用于构建默认钩子的配置
    checkpoint=dict(  # 配置检查点钩子
        type='CheckpointHook',
        interval=5000,  # 保存间隔为5000次迭代
        save_optimizer=True,
        by_epoch=False,  # 以迭代次数计数
        out_dir=save_dir,
    ),
    timer=dict(type='IterTimerHook'),
    logger=dict(type='LoggerHook', interval=100),  # 配置注册日志钩子
    param_scheduler=dict(type='ParamSchedulerHook'),
    sampler_seed=dict(type='DistSamplerSeedHook'),
)
```

custom_hooks 是一个钩子配置的列表。用户可以开发自己的钩子并将其插入到该字段中。

```
custom_hooks = [dict(type='BasicVisualizationHook', interval=1)]  # 可视化钩子的配置
```

6.4.7 运行时配置

```
default_scope = 'mmagic'  # 用于设置注册表位置
env_cfg = dict(  # 设置分布式训练的参数，端口也可以设置
    cudnn_benchmark=False,
    mp_cfg=dict(mp_start_method='fork', opencv_num_threads=4),
    dist_cfg=dict(backend='nccl'),
)
log_level = 'INFO'  # 日志记录的级别
log_processor = dict(type='LogProcessor', window_size=100, by_epoch=False)  #␣
↪用于构建日志处理器
load_from = None  # 从给定路径加载模型作为预训练模型。这不会恢复训练。
resume = False  # 从给定路径恢复检查点，训练将从检查点保存的时期继续。
```

6.5 StyleGAN2 的示例

以Stylegan2 在 1024x1024 分辨率上的配置文件 为例，我们根据不同的功能模块介绍配置中的各个字段。

6.5.1 模型配置

除了包括生成器、鉴别器等神经网络组件之外，还需要 `data_preprocessor`、`loss_config` 等字段，其中一些还包含 `ema_config`。

`data_preprocessor` 负责处理数据加载器输出的一个批次数据。

`loss_config` 负责设置损失项的权重。

`ema_config` 负责为生成器执行指数移动平均（EMA）操作。

```
model = dict(
    type='StyleGAN2',  # 模型的名称
    data_preprocessor=dict(type='DataPreprocessor'),  #␣
↪数据预处理器的配置，通常包括图像归一化和填充
    generator=dict(  # 生成器的配置
        type='StyleGANv2Generator',  # 生成器的名称
        out_size=1024,  # 生成器的输出分辨率
        style_channels=512),  # 生成器的风格通道数
    discriminator=dict(  # 鉴别器的配置
        type='StyleGAN2Discriminator',  # 鉴别器的名称
        in_size=1024),  # 鉴别器的输入分辨率
    ema_config=dict(  # EMA的配置
        type='ExponentialMovingAverage',  # 平均模型的具体类型
        interval=1,  # EMA操作的间隔
        momentum=0.9977843871238888),  # EMA操作的动量
    loss_config=dict(  # 损失项的配置
        r1_loss_weight=80.0,  # r1梯度惩罚的权重
        r1_interval=16,  # r1梯度惩罚的间隔
        norm_mode='HWC',  # r1梯度惩罚的归一化模式
        g_reg_interval=4,  # 生成器的正则化间隔
        g_reg_weight=8.0,  # 生成器的正则化权重
        pl_batch_shrink=2))  # 路径长度正则化中缩减批次大小的因子
```

6.5.2 数据集和评估器配置

训练、验证和测试runner 需要使用数据加载器Dataloaders。需要设置数据集和数据处理流程来构建数据加载器。由于这部分的复杂性，我们使用中间变量来简化数据加载器配置的编写。

```
dataset_type = 'BasicImageDataset'  # 数据集类型，将用于定义数据集
data_root = './data/ffhq/'  # 数据的根目录

train_pipeline = [  # 训练数据处理流程
    dict(type='LoadImageFromFile', key='img'),  # 第一个处理流程，从文件路径加载图像
    dict(type='Flip', keys=['img'], direction='horizontal'),  #␣
↪图像翻转的数据增强处理流程
    dict(type='PackInputs', keys=['img'])  #␣
↪最后一个处理流程，格式化注释数据（如果有）并决定哪些键应该打包到data_samples中
]
val_pipeline = [
    dict(type='LoadImageFromFile', key='img'),  # 第一个处理流程，从文件路径加载图像
    dict(type='PackInputs', keys=['img'])  #␣
↪最后一个处理流程，格式化注释数据（如果有）并决定哪些键应该打包到data_samples中
]
train_dataloader = dict(  # 训练数据加载器的配置
    batch_size=4,  # 单个GPU的批次大小
    num_workers=8,  # 每个单个GPU的数据预取工作线程数
    persistent_workers=True,  #␣
↪如果为True，则数据加载器将在一个epoch结束后不会关闭工作进程，这可以加速训练速度。
    sampler=dict(  # 训练数据采样器的配置
        type='InfiniteSampler',
        # 用于迭代训练的InfiniteSampler。参考 https://github.com/open-mmlab/mmengine/
↪blob/fe0eb0a5bbc8bf816d5649bfdd34908c258eb245/mmengine/dataset/sampler.py#L107
        shuffle=True),  # 是否随机打乱训练数据
    dataset=dict(  # 训练数据集的配置
        type=dataset_type,
        data_root=data_root,
        pipeline=train_pipeline))
val_dataloader = dict(  # 验证数据加载器的配置
    batch_size=4,  # 单个GPU的批次大小
    num_workers=8,  # 每个单个GPU的数据预取工作线程数
    dataset=dict(  # 验证数据集的配置
        type=dataset_type,
        data_root=data_root,
        pipeline=val_pipeline),
    sampler=dict(  # 验证数据采样器的配置
        type='DefaultSampler',
        # 支持分布式和非分布式训练的DefaultSampler。参考 https://github.com/open-
↪mmlab/mmengine/blob/fe0eb0a5bbc8bf816d5649bfdd34908c258eb245/mmengine/dataset/
↪sampler.py#L14
```

(下页继续)

(续上页)

```
        shuffle=False),  # 是否随机打乱验证数据
    persistent_workers=True)
test_dataloader = val_dataloader  # 测试数据加载器的配置与验证数据加载器相同
```

评估器用于计算在验证和测试数据集上训练模型的指标。评估器的配置由一个或多个指标配置组成：

```
val_evaluator = dict(  # 验证评估器的配置
    type='Evaluator',  # 评估类型
    metrics=[  # 指标的配置
        dict(
            type='FrechetInceptionDistance',
            prefix='FID-Full-50k',
            fake_nums=50000,
            inception_style='StyleGAN',
            sample_model='ema'),
        dict(type='PrecisionAndRecall', fake_nums=50000, prefix='PR-50K'),
        dict(type='PerceptualPathLength', fake_nums=50000, prefix='ppl-w')
    ])
test_evaluator = val_evaluator  # 测试评估器的配置与验证评估器相同
```

6.5.3 训练和测试配置

MMEngine 的 runner 使用 Loop 来控制训练、验证和测试过程。用户可以使用以下字段设置最大训练迭代次数和验证间隔：

```
train_cfg = dict(  # 训练配置
    by_epoch=False,  # 设置`by_epoch`为False以使用基于迭代的训练
    val_begin=1,  # 开始验证的迭代次数
    val_interval=10000,  # 验证间隔
    max_iters=800002)  # 最大训练迭代次数
val_cfg = dict(type='MultiValLoop')  # 验证循环类型
test_cfg = dict(type='MultiTestLoop')  # 测试循环类型
```

6.5.4 优化配置

optim_wrapper 是配置优化相关设置的字段。优化器包装器不仅提供优化器的功能，还支持梯度裁剪、混合精度训练等功能。在optimizer wrapper tutorial 中可以找到更多信息。

```
optim_wrapper = dict(
    constructor='MultiOptimWrapperConstructor',
    generator=dict(
```

(下页继续)

(续上页)

```
        optimizer=dict(type='Adam', lr=0.0016, betas=(0, 0.9919919678228657))),
    discriminator=dict(
        optimizer=dict(
            type='Adam',
            lr=0.0018823529411764706,
            betas=(0, 0.9905854573074332))))
```

param_scheduler 是一个配置优化超参数（如学习率和动量）调整方法的字段。用户可以组合多个调度器来创建所需的参数调整策略。在parameter scheduler tutorial中可以找到更多信息。由于 StyleGAN2 不使用参数调度器，我们以CycleGAN 的配置作为示例：

```
# CycleGAN配置中的参数调度器
param_scheduler = dict(
    type='LinearLrInterval',  # 调度器的类型
    interval=400,  # 更新学习率的间隔
    by_epoch=False,  # 调度器按迭代调用
    start_factor=0.0002,  # 在第一次迭代中乘以参数值的数值
    end_factor=0,  # 在线性变化过程结束时乘以参数值的数值
    begin=40000,  # 调度器的起始迭代次数
    end=80000)  # 调度器的结束迭代次数
```

6.5.5 钩子配置

用户可以在训练、验证和测试循环中附加钩子，以在运行过程中插入一些操作。这里有两个不同的钩子字段，一个是 default_hooks ，另一个是 custom_hooks。

default_hooks 是一个钩子配置的字典。default_hooks 是在运行时必须的钩子。它们具有默认的优先级，不应该被修改。如果没有设置，运行器将使用默认值。要禁用一个默认钩子，用户可以将其配置设置为 None。

```
default_hooks = dict(
    timer=dict(type='IterTimerHook'),
    logger=dict(type='LoggerHook', interval=100, log_metric_by_epoch=False),
    checkpoint=dict(
        type='CheckpointHook',
        interval=10000,
        by_epoch=False,
        less_keys=['FID-Full-50k/fid'],
        greater_keys=['IS-50k/is'],
        save_optimizer=True,
        save_best='FID-Full-50k/fid'))
```

custom_hooks 是一个钩子配置的列表。用户可以开发自己的钩子并将它们插入到这个字段中。

```
custom_hooks = [
    dict(
        type='VisualizationHook',
        interval=5000,
        fixed_input=True,
        vis_kwargs_list=dict(type='GAN', name='fake_img'))
]
```

6.5.6 运行时配置

```
default_scope = 'mmagic'  # 默认的注册表作用域，用于查找模块。参考 https://mmengine.
↪readthedocs.io/en/latest/advanced_tutorials/registry.html

# 环境配置
env_cfg = dict(
    cudnn_benchmark=True,  # 是否启用cudnn基准测试
    mp_cfg=dict(mp_start_method='fork', opencv_num_threads=0),  # 设置多进程参数
    dist_cfg=dict(backend='nccl')  # 设置分布式参数
)

log_level = 'INFO'  # 日志级别
log_processor = dict(
    type='LogProcessor',  # 日志处理器，用于处理运行时日志
    by_epoch=False)  # 按迭代打印日志
load_from = None  # 从给定路径加载模型检查点作为预训练模型
resume = False  # 是否从`load_from`定义的检查点恢复训练。如果`load_
↪from`为`None`，将恢复`work_dir`中的最新检查点
```

6.6 其他示例

6.6.1 修复任务的配置示例

为了帮助用户对修复系统的完整配置和模块有一个基本的了解，我们对全局和局部修复的配置进行简要注释，如下所示。有关更详细的用法和每个模块的替代选项，请参考 API 文档。

```
model = dict(
    type='GLInpaintor',  # 修复模型的名称
    data_preprocessor=dict(
        type='DataPreprocessor',  # 数据预处理器的名称
        mean=[127.5],  # 数据归一化时使用的均值
```

(下页继续)

(续上页)

```
        std=[127.5],  # 数据归一化时使用的标准差
    ),
    encdec=dict(
        type='GLEncoderDecoder',  # 编码器-解码器的名称
        encoder=dict(type='GLEncoder', norm_cfg=dict(type='SyncBN')),  # 编码器的配置
        decoder=dict(type='GLDecoder', norm_cfg=dict(type='SyncBN')),  # 解码器的配置
        dilation_neck=dict(
            type='GLDilationNeck', norm_cfg=dict(type='SyncBN'))),  # 膨胀模块的配置
    disc=dict(
        type='GLDiscs',  # 判别器的名称
        global_disc_cfg=dict(
            in_channels=3,  # 判别器的输入通道数
            max_channels=512,  # 判别器中间通道的最大数量
            fc_in_channels=512 * 4 * 4,  # 最后一个全连接层的输入通道数
            fc_out_channels=1024,  # 最后一个全连接层的输出通道数
            num_convs=6,  # 判别器中使用的卷积层数量
            norm_cfg=dict(type='SyncBN')  # 归一化层的配置
        ),
        local_disc_cfg=dict(
            in_channels=3,  # 判别器的输入通道数
            max_channels=512,  # 判别器中间通道的最大数量
            fc_in_channels=512 * 4 * 4,  # 最后一个全连接层的输入通道数
            fc_out_channels=1024,  # 最后一个全连接层的输出通道数
            num_convs=5,  # 判别器中使用的卷积层数量
            norm_cfg=dict(type='SyncBN')  # 归一化层的配置
        ),
    ),
    loss_gan=dict(
        type='GANLoss',  # GAN损失的名称
        gan_type='vanilla',  # GAN损失的类型
        loss_weight=0.001  # GAN损失的权重
    ),
    loss_l1_hole=dict(
        type='L1Loss',  # L1损失的类型
        loss_weight=1.0  # L1损失的权重
    )
)

train_cfg = dict(
    type='IterBasedTrainLoop',  # 训练循环的类型
    max_iters=500002,  # 总迭代次数
    val_interval=50000  # 验证间隔的迭代次数
)
```

(下页继续)

(续上页)

```
val_cfg = dict(type='ValLoop')  # 验证循环的类型
test_cfg = dict(type='TestLoop')  # 测试循环的类型

val_evaluator = [
    dict(type='MAE', mask_key='mask', scaling=100),  # 用于评估的指标名称
    dict(type='PSNR'),  # 用于评估的指标名称
    dict(type='SSIM'),  # 用于评估的指标名称
]
test_evaluator = val_evaluator

input_shape = (256, 256)  # 输入图像的形状

train_pipeline = [
    dict(type='LoadImageFromFile', key='gt'),  # 加载图像的配置
    dict(
        type='LoadMask',  # 加载掩膜的类型
        mask_mode='bbox',  # 掩膜的类型
        mask_config=dict(
            max_bbox_shape=(128, 128),  # 边界框的形状
            max_bbox_delta=40,  # 边界框高度和宽度的变化范围
            min_margin=20,  # 边界框与图像边界的最小间距
            img_shape=input_shape)),  # 输入图像的形状
    dict(
        type='Crop',  # 裁剪的类型
        keys=['gt'],  # 需要裁剪的图像的键
        crop_size=(384, 384),  # 裁剪后的大小
        random_crop=True,  # 是否随机裁剪
    ),
    dict(
        type='Resize',  # 调整大小的类型
        keys=['gt'],  # 需要调整大小的图像的键
        scale=input_shape,  # 调整大小的比例
        keep_ratio=False,  # 是否保持比例
    ),
    dict(
        type='Normalize',  # 归一化的类型
        keys=['gt_img'],  # 需要归一化的图像的键
        mean=[127.5] * 3,  # 归一化时使用的均值
        std=[127.5] * 3,  # 归一化时使用的标准差
        to_rgb=False),  # 是否将图像通道转换为RGB
    dict(type='GetMaskedImage'),  # 获取掩膜图像的配置
    dict(type='PackInputs'),  # 收集当前流水线中的数据的配置
]
```

(下页继续)

(续上页)

```
test_pipeline = train_pipeline  # 构建测试/验证流水线

dataset_type = 'BasicImageDataset'  # 数据集的类型
data_root = 'data/places'  # 数据的根路径

train_dataloader = dict(
    batch_size=12,  # 单个GPU的批处理大小
    num_workers=4,  # 每个单个GPU预取数据的工作线程数
    persistent_workers=False,  # 是否保持工作线程的数据集实例
    sampler=dict(type='InfiniteSampler', shuffle=False),  # 数据采样器的类型
    dataset=dict(  # 训练数据集的配置
        type=dataset_type,  # 数据集的类型
        data_root=data_root,  # 数据的根路径
        data_prefix=dict(gt='data_large'),  # 图像路径的前缀
        ann_file='meta/places365_train_challenge.txt',  # 注释文件的路径
        test_mode=False,
        pipeline=train_pipeline,
    ))

val_dataloader = dict(
    batch_size=1,  # 单个GPU的批处理大小
    num_workers=4,  # 每个单个GPU预取数据的工作线程数
    persistent_workers=False,  # 是否保持工作线程的数据集实例
    drop_last=False,  # 是否丢弃最后一个不完整的批次
    sampler=dict(type='DefaultSampler', shuffle=False),  # 数据采样器的类型
    dataset=dict(  # 验证数据集的配置
        type=dataset_type,  # 数据集的类型
        data_root=data_root,  # 数据的根路径
        data_prefix=dict(gt='val_large'),  # 图像路径的前缀
        ann_file='meta/places365_val.txt',  # 注释文件的路径
        test_mode=True,
        pipeline=test_pipeline,
    ))

test_dataloader = val_dataloader

model_wrapper_cfg = dict(type='MMSeparateDistributedDataParallel')  # 模型包装器的名称

optim_wrapper = dict(  #␣
↪用于构建优化器的配置，支持PyTorch中的所有优化器，其参数与PyTorch中的优化器的参数相同
    constructor='MultiOptimWrapperConstructor',
    generator=dict(
```

(下页继续)

(续上页)

```
        type='OptimWrapper', optimizer=dict(type='Adam', lr=0.0004)),
    disc=dict(type='OptimWrapper', optimizer=dict(type='Adam', lr=0.0004)))

default_scope = 'mmagic'  # 用于设置注册表位置
save_dir = './work_dirs'  # 保存模型检查点和日志的目录
exp_name = 'gl_places'  # 实验名称

default_hooks = dict(  # 用于构建默认挂钩
    timer=dict(type='IterTimerHook'),
    logger=dict(type='LoggerHook', interval=100),  # 注册记录器挂钩的配置
    param_scheduler=dict(type='ParamSchedulerHook'),
    checkpoint=dict(  # 设置检查点挂钩的配置
        type='CheckpointHook',
        interval=50000,
        by_epoch=False,
        out_dir=save_dir),
    sampler_seed=dict(type='DistSamplerSeedHook'),
)

env_cfg = dict(  # 用于设置分布式训练的参数，也可以设置端口
    cudnn_benchmark=False,
    mp_cfg=dict(mp_start_method='fork', opencv_num_threads=0),
    dist_cfg=dict(backend='nccl'),
)

vis_backends = [dict(type='LocalVisBackend')]  # 可视化后端的名称
visualizer = dict(  # 用于构建可视化器的配置
    type='ConcatImageVisualizer',
    vis_backends=vis_backends,
    fn_key='gt_path',
    img_keys=['gt_img', 'input', 'pred_img'],
    bgr2rgb=True)
custom_hooks = [dict(type='BasicVisualizationHook', interval=1)]  # 用于构建自定义挂钩

log_level = 'INFO'  # 记录级别
log_processor = dict(type='LogProcessor', by_epoch=False)  # 用于构建日志处理器

load_from = None  # 从给定路径加载预训练模型
resume = False  # 从给定路径恢复检查点

find_unused_parameters = False  # 是否在DDP中设置未使用的参数
```

6.6.2 抠图任务的配置示例

为了帮助用户对完整的配置有一个基本的了解，我们对我们实现的原始 DIM（Deep Image Matting）模型的配置进行了简要的注释，如下所示。有关每个模块的更详细用法和相应的替代方案，请参阅 API 文档。

```
# 模型设置
model = dict(
    type='DIM',  # 模型的名称（我们称之为mattor）。
    data_preprocessor=dict(  # 数据预处理器的配置
        type='DataPreprocessor',
        mean=[123.675, 116.28, 103.53],
        std=[58.395, 57.12, 57.375],
        bgr_to_rgb=True,
        proc_inputs='normalize',
        proc_trimap='rescale_to_zero_one',
        proc_gt='rescale_to_zero_one',
    ),
    backbone=dict(  # 骨干网络的配置。
        type='SimpleEncoderDecoder',  # 骨干网络的类型。
        encoder=dict(  # 编码器的配置。
            type='VGG16'),  # 编码器的类型。
        decoder=dict(  # 解码器的配置。
            type='PlainDecoder')),  # 解码器的类型。
    pretrained='./weights/vgg_state_dict.pth',  # 要加载的编码器的预训练权重。
    loss_alpha=dict(  # alpha损失的配置。
        type='CharbonnierLoss',  # 预测的alpha融合图像的损失类型。
        loss_weight=0.5),  # alpha损失的权重。
    loss_comp=dict(  # 合成损失的配置。
        type='CharbonnierCompLoss',  # 合成损失的类型。
        loss_weight=0.5),  # 合成损失的权重。
    train_cfg=dict(  # DIM模型的训练配置。
        train_backbone=True,  # 在DIM阶段1中，训练骨干网络。
        train_refiner=False),  # 在DIM阶段1中，不训练refiner。
    test_cfg=dict(  # DIM模型的测试配置。
        refine=False,  # 是否使用refiner输出作为输出，在阶段1中我们不使用它。
        resize_method='pad',
        resize_mode='reflect',
        size_divisor=32,
    ),
)

# 数据设置
dataset_type = 'AdobeComp1kDataset'  # 数据集类型，用于定义数据集。
data_root = 'data/adobe_composition-1k'  # 数据的根路径。
```

(下页继续)

(续上页)

```
train_pipeline = [  # 训练数据处理流程。
    dict(
        type='LoadImageFromFile',  # 从文件加载alpha融合图像。
        key='alpha',  # 注释文件中alpha融合图像的键。该流程将从路径`alpha_
↪path`读取alpha融合图像。
        color_type='grayscale'),  # 加载为灰度图像，具有形状（高度，宽度）。
    dict(
        type='LoadImageFromFile',  # 从文件加载图像。
        key='fg'),  # 要加载的图像的键。该流程将从路径`fg_path`读取前景图像。
    dict(
        type='LoadImageFromFile',  # 从文件加载图像。
        key='bg'),  # 要加载的图像的键。该流程将从路径`bg_path`读取背景图像。
    dict(
        type='LoadImageFromFile',  # 从文件加载图像。
        key='merged'),  # 要加载的图像的键。该流程将从路径`merged_path`读取合并图像。
    dict(
        type='CropAroundUnknown',  # 在未知区域（半透明区域）周围裁剪图像。
        keys=['alpha', 'merged', 'fg', 'bg'],  # 要裁剪的图像。
        crop_sizes=[320, 480, 640]),  # 候选裁剪大小。
    dict(
        type='Flip',  # 翻转图像的增强流程。
        keys=['alpha', 'merged', 'fg', 'bg']),  # 要翻转的图像。
    dict(
        type='Resize',  # 调整图像大小的增强流程。
        keys=['alpha', 'merged', 'fg', 'bg'],  # 要调整大小的图像。
        scale=(320, 320),  # 目标大小。
        keep_ratio=False),  # 是否保持高度和宽度之间的比例。
    dict(
        type='GenerateTrimap',  # 从alpha融合图像生成trimap。
        kernel_size=(1, 30)),  # 腐蚀/膨胀内核的大小范围。
    dict(type='PackInputs'),  # 从当前流程中收集数据的配置
]
test_pipeline = [
    dict(
        type='LoadImageFromFile',  # 加载alpha融合图像。
        key='alpha',  # 注释文件中alpha融合图像的键。该流程将从路径`alpha_
↪path`读取alpha融合图像。
        color_type='grayscale',
        save_original_img=True),
    dict(
        type='LoadImageFromFile',  # 从文件加载图像。
        key='trimap',  # 要加载的图像的键。该流程将从路径`trimap_path`读取trimap。
```

(下页继续)

(续上页)

```
        color_type='grayscale',  # 加载为灰度图像，具有形状（高度，宽度）。
        save_original_img=True),  # 保存trimap的副本用于计算指标。它将以键`ori_
↪trimap`保存。
    dict(
        type='LoadImageFromFile',  # 从文件加载图像。
        key='merged'),  # 要加载的图像的键。该流程将从路径`merged_path`读取合并图像。
    dict(type='PackInputs'),  # 从当前流程中收集数据的配置
]

train_dataloader = dict(
    batch_size=1,  # 单个GPU的批处理大小
    num_workers=4,  # 每个单个GPU预提取数据的工作线程数
    persistent_workers=False,  # 是否保持工作线程Dataset实例处于活动状态
    sampler=dict(type='InfiniteSampler', shuffle=True),  # 数据采样器的类型
    dataset=dict(  # 训练数据集的配置
        type=dataset_type,  # 数据集的类型
        data_root=data_root,  # 数据的根路径
        ann_file='training_list.json',  # 注释文件的路径
        test_mode=False,
        pipeline=train_pipeline,
    ))

val_dataloader = dict(
    batch_size=1,  # 单个GPU的批处理大小
    num_workers=4,  # 每个单个GPU预提取数据的工作线程数
    persistent_workers=False,  # 是否保持工作线程Dataset实例处于活动状态
    drop_last=False,  # 是否丢弃最后一个不完整的批次
    sampler=dict(type='DefaultSampler', shuffle=False),  # 数据采样器的类型
    dataset=dict(  # 验证数据集的配置
        type=dataset_type,  # 数据集的类型
        data_root=data_root,  # 数据的根路径
        ann_file='test_list.json',  # 注释文件的路径
        test_mode=True,
        pipeline=test_pipeline,
    ))

test_dataloader = val_dataloader

val_evaluator = [
    dict(type='SAD'),  # 要评估的指标名称
    dict(type='MattingMSE'),  # 要评估的指标名称
    dict(type='GradientError'),  # 要评估的指标名称
    dict(type='ConnectivityError'),  # 要评估的指标名称
```

(下页继续)

(续上页)

```
]
test_evaluator = val_evaluator

train_cfg = dict(
    type='IterBasedTrainLoop',  # 训练循环类型的名称
    max_iters=1_000_000,  # 总迭代次数
    val_interval=40000,  # 验证间隔迭代次数
)
val_cfg = dict(type='ValLoop')  # 验证循环类型的名称
test_cfg = dict(type='TestLoop')  # 测试循环类型的名称

# 优化器
optim_wrapper = dict(
    dict(
        type='OptimWrapper',
        optimizer=dict(type='Adam', lr=0.00001),
    )
)  # 用于构建优化器的配置，支持PyTorch中所有优化器，其参数也与PyTorch中的参数相同。

default_scope = 'mmagic'  # 用于设置注册表位置
save_dir = './work_dirs'  # 保存当前实验的模型检查点和日志的目录。

default_hooks = dict(  # 用于构建默认钩子
    timer=dict(type='IterTimerHook'),
    logger=dict(type='LoggerHook', interval=100),  # 注册日志记录器钩子的配置
    param_scheduler=dict(type='ParamSchedulerHook'),
    checkpoint=dict(  # 配置检查点钩子
        type='CheckpointHook',
        interval=40000,  # 保存间隔为40000次迭代。
        by_epoch=False,  # 按迭代计数。
        out_dir=save_dir),
    sampler_seed=dict(type='DistSamplerSeedHook'),
)

env_cfg = dict(  # 设置分布式训练的参数，也可以设置端口
    cudnn_benchmark=False,
    mp_cfg=dict(mp_start_method='fork', opencv_num_threads=4),
    dist_cfg=dict(backend='nccl'),
)

log_level = 'INFO'  # 日志级别
log_processor = dict(type='LogProcessor', by_epoch=False)  #␣
↪用于构建日志处理器的配置。
```

(下页继续)

(续上页)

```
# 导入模块并设置MMDetection的配置
load_from = None  # 从给定路径加载模型作为预训练模型，这不会恢复训练。
resume = False  # 从给定路径恢复检查点，训练将从检查点保存的时期恢复。
```

6.6.3 恢复任务的配置示例

为了帮助用户对完整配置有一个基本的理解，我们对我们实现的 EDSR 模型的配置进行了简要的注释，如下所示。有关更详细的用法和每个模块的相应替代方案，请参阅 API 文档。

```
exp_name = 'edsr_x2c64b16_1x16_300k_div2k'  # 实验名称
work_dir = f'./work_dirs/{experiment_name}'
save_dir = './work_dirs/'

load_from = None  # 基于预训练的x2模型

scale = 2  # 上采样的比例
# 模型设置
model = dict(
    type='BaseEditModel',  # 模型名称
    generator=dict(  # 生成器的配置
        type='EDSRNet',  # 生成器的类型
        in_channels=3,  # 输入的通道数
        out_channels=3,  # 输出的通道数
        mid_channels=64,  # 中间特征的通道数
        num_blocks=16,  # 主干网络中的块数
        upscale_factor=scale,  # 上采样因子
        res_scale=1,  # 用于缩放残差块中的残差
        rgb_mean=(0.4488, 0.4371, 0.4040),  # 图像的RGB均值
        rgb_std=(1.0, 1.0, 1.0)),  # 图像的RGB标准差
    pixel_loss=dict(type='L1Loss', loss_weight=1.0, reduction='mean')  #␣
↪像素损失的配置
train_cfg = dict(),  # 训练模型的配置
test_cfg = dict(),  # 测试模型的配置
data_preprocessor = dict(  # 数据预处理器的配置
    type='DataPreprocessor', mean=[0., 0., 0.], std=[255., 255., 255.]))

train_pipeline = [  # 训练数据处理的流程
    dict(type='LoadImageFromFile',  # 从文件加载图像
         key='img',  # 结果中寻找对应路径的关键字
         color_type='color',  # 图像的颜色类型
         channel_order='rgb',  # 图像的通道顺序
```

(下页继续)

(续上页)

```
        imdecode_backend='cv2'),  # 解码后端
    dict(type='LoadImageFromFile',  # 从文件加载图像
        key='gt',  # 结果中寻找对应路径的关键字
        color_type='color',  # 图像的颜色类型
        channel_order='rgb',  # 图像的通道顺序
        imdecode_backend='cv2'),  # 解码后端
    dict(type='SetValues', dictionary=dict(scale=scale)),  # 设置目标关键字的值
    dict(type='PairedRandomCrop', gt_patch_size=96),  # 随机裁剪配对图像
    dict(type='Flip',  # 翻转图像
        keys=['lq', 'gt'],  # 需要翻转的图像
        flip_ratio=0.5,  # 翻转的比例
        direction='horizontal'),  # 翻转的方向
    dict(type='Flip',  # 翻转图像
        keys=['lq', 'gt'],  # 需要翻转的图像
        flip_ratio=0.5,  # 翻转的比例
        direction='vertical'),  # 翻转的方向
    dict(type='RandomTransposeHW',  # 随机转置图像的高度和宽度
        keys=['lq', 'gt'],  # 需要转置的图像
        transpose_ratio=0.5  # 转置的比例
        ),
    dict(type='PackInputs')  # 收集当前流程中的数据的配置
]
test_pipeline = [  # 测试流程
    dict(type='LoadImageFromFile',  # 从文件加载图像
        key='img',  # 结果中寻找对应路径的关键字
        color_type='color',  # 图像的颜色类型
        channel_order='rgb',  # 图像的通道顺序
        imdecode_backend='cv2'),  # 解码后端
    dict(type='LoadImageFromFile',  # 从文件加载图像
        key='gt',  # 结果中寻找对应路径的关键字
        color_type='color',  # 图像的颜色类型
        channel_order='rgb',  # 图像的通道顺序
        imdecode_backend='cv2'),  # 解码后端
    dict(type='ToTensor', keys=['img', 'gt']),  # 将图像转换为张量
    dict(type='PackInputs')  # 收集当前流程中的数据的配置
]

# 数据集设置
dataset_type = 'BasicImageDataset'  # 数据集的类型
data_root = 'data'  # 数据的根路径

train_dataloader = dict(
    num_workers=4,  # 每个GPU预提取数据的工作进程数
```

(下页继续)

(续上页)

```
    persistent_workers=False,  # 是否保持工作进程中的数据集实例处于活动状态
    sampler=dict(type='InfiniteSampler', shuffle=True),  # 数据采样器的类型
    dataset=dict(  # 训练数据集的配置
        type=dataset_type,  # 数据集的类型
        ann_file='meta_info_DIV2K800sub_GT.txt',  # 注释文件的路径
        metainfo=dict(dataset_type='div2k', task_name='sisr'),
        data_root=data_root + '/DIV2K',  # 数据的根路径
        data_prefix=dict(  # 图像路径的前缀
            img='DIV2K_train_LR_bicubic/X2_sub', gt='DIV2K_train_HR_sub'),
        filename_tmpl=dict(img='{}', gt='{}'),  # 文件名模板
        pipeline=train_pipeline))
val_dataloader = dict(
    num_workers=4,  # 每个GPU预提取数据的工作进程数
    persistent_workers=False,  # 是否保持工作进程中的数据集实例处于活动状态
    drop_last=False,  # 是否丢弃最后一个不完整的批次
    sampler=dict(type='DefaultSampler', shuffle=False),  # 数据采样器的类型
    dataset=dict(  # 验证数据集的配置
        type=dataset_type,  # 数据集的类型
        metainfo=dict(dataset_type='set5', task_name='sisr'),
        data_root=data_root + '/Set5',  # 数据的根路径
        data_prefix=dict(img='LRbicx2', gt='GTmod12'),  # 图像路径的前缀
        pipeline=test_pipeline))
test_dataloader = val_dataloader

val_evaluator = [
    dict(type='MAE'),  # 用于评估的指标的名称
    dict(type='PSNR', crop_border=scale),  # 用于评估的指标的名称
    dict(type='SSIM', crop_border=scale),  # 用于评估的指标的名称
]
test_evaluator = val_evaluator

train_cfg = dict(
    type='IterBasedTrainLoop', max_iters=300000, val_interval=5000)  #␣
↪训练循环类型的配置
val_cfg = dict(type='ValLoop')  # 验证循环类型的名称
test_cfg = dict(type='TestLoop')  # 测试循环类型的名称

# 优化器
optim_wrapper = dict(
    dict(
        type='OptimWrapper',
        optimizer=dict(type='Adam', lr=0.00001),
    )
```

(下页继续)

(续上页)

```
)  # 用于构建优化器的配置，支持PyTorch中所有优化器，参数与PyTorch中的相同。

param_scheduler = dict(  # 学习策略的配置
    type='MultiStepLR', by_epoch=False, milestones=[200000], gamma=0.5)

default_hooks = dict(  # 用于构建默认钩子
    checkpoint=dict(  # 配置保存检查点的钩子
        type='CheckpointHook',
        interval=5000,  # 保存间隔为5000次迭代
        save_optimizer=True,
        by_epoch=False,  # 以迭代计数
        out_dir=save_dir,
    ),
    timer=dict(type='IterTimerHook'),
    logger=dict(type='LoggerHook', interval=100),  # 注册记录器钩子的配置
    param_scheduler=dict(type='ParamSchedulerHook'),
    sampler_seed=dict(type='DistSamplerSeedHook'),
)

default_scope = 'mmagic'  # 用于设置注册表位置
save_dir = './work_dirs'  # 保存当前实验的模型检查点和日志的目录。

env_cfg = dict(  # 设置分布式训练的参数，端口也可以设置
    cudnn_benchmark=False,
    mp_cfg=dict(mp_start_method='fork', opencv_num_threads=4),
    dist_cfg=dict(backend='nccl'),
)

log_level = 'INFO'  # 记录的级别
log_processor = dict(type='LogProcessor', window_size=100, by_epoch=False)  #␣
↪用于构建日志处理器

load_from = None  # 从给定路径加载模型作为预训练模型，这不会恢复训练。
resume = False  # 从给定路径恢复检查点，训练将从检查点保存的时期恢复。
```

CHAPTER 7

教程 2：准备数据集

在本节中，我们将详细介绍如何准备数据并在本仓库的不同任务中采用适当的数据集。

我们支持不同任务的多个数据集。在 MMagic 中，有两种方法可以将数据集用于训练和测试模型：

1. 直接使用下载的数据集
2. 在使用下载的数据集之前对其进行预处理

本文的结构如下：

- [教程 2：准备数据集](# 教程 2：准备数据集)
  - 下载数据集
  - 准备数据集
  - MMagic 中的数据集概述

7.1 下载数据集

首先，建议您从官方的页面下载数据集。大多数数据集在下载后都是可用的，因此您只需确保文件夹结构正确，无需进一步准备。例如，您可以通过从主页下载，来简单地准备 Vimeo90K-triplet 数据集.

7.2 准备数据集

一些数据集需要在训练或测试之前进行预处理。我们在 tools/dataset_converters中支持许多用来准备数据集的脚本。您可以遵循每个数据集的教程来运行脚本。例如，我们建议将 DIV2K 图像裁剪为子图像。我们提供了一个脚本来准备裁剪的 DIV2K 数据集。可以运行以下命令：

```
python tools/dataset_converters/div2k/preprocess_div2k_dataset.py --data-root ./data/
↪DIV2K
```

7.3 MMagic 中的数据集概述

我们支持详细的教程，并根据不同的任务进行拆分。

请查看我们的数据集概览，了解不同任务的数据准备。

如果您对 MMagic 中数据集的更多细节感兴趣，请查看进阶教程。

CHAPTER 8

教程 3：使用预训练模型推理

MMagic 提供了高级 API，让您可以轻松地在自己的图像或视频上使用最先进的模型进行操作。在新的 API 中，仅需两行代码即可进行推理。

```
from mmagic.apis import MMagicInferencer

# 创建MMagicInferencer实例
editor = MMagicInferencer('pix2pix')
# 推理图片.需要输入图片路径与输出图片路径
results = editor.infer(img='../resources/input/translation/gt_mask_0.png', result_out_
↪dir='../resources/output/translation/tutorial_translation_pix2pix_res.jpg')
```

MMagic 支持各种基础生成模型，包括无条件生成对抗网络（GANs）、条件 GANs、扩散模型等。MMagic 同样支持多种应用，包括：文生图、图生图的转换、3D 感知生成、图像超分、视频超分、视频帧插值、图像修补、图像抠图、图像恢复、图像上色、图像生成等。在本节中，我们将详细说明如何使用我们预训练的模型进行操作。

- 教程 3: 使用预训练模型推理
  - 准备一些图片或者视频用于推理
  - 生成模型
    * 无条件生成对抗网络 (GANs)
    * 条件生成对抗网络 (GANs)
    * 扩散模型

  - 应用
    * 文生图
    * 图生图的转换
    * 3D 感知生成
    * 图像超分
    * 视频超分
    * 视频帧插值
    * 图像修补
    * 图像抠图
    * 图像恢复
    * 图像上色
- 以前的版本

8.1 准备一些图片或者视频用于推理

请参考我们的教程获取详细信息。

8.2 生成模型

8.2.1 无条件生成对抗网络（GANs）

MMagic 提供了用于使用无条件 GANs 进行图像采样的高级 API。无条件 GAN 模型不需要输入，并输出一张图像。我们以’styleganv1’为例。

```
import mmcv
import matplotlib.pyplot as plt
from mmagic.apis import MMagicInferencer

# 创建MMagicInferencer实例，并进行推理
result_out_dir = './resources/output/unconditional/tutorial_unconditional_styleganv1_
↪res.png'
editor = MMagicInferencer('styleganv1')
results = editor.infer(result_out_dir=result_out_dir)
```

确实，我们已经为用户提供了一个更友好的演示脚本。您可以使用以下命令使用demo/mmagic_inference_demo.py：

```
python demo/mmagic_inference_demo.py \
        --model-name styleganv1 \
        --result-out-dir demo_unconditional_styleganv1_res.jpg
```

8.2.2 条件生成对抗网络 (GANs)

MMagic 提供了使用条件 GAN 进行图像采样的高级 API。条件 GAN 模型接受一个标签作为输入，并输出一张图像。我们以’biggan’为例。

```
import mmcv
import matplotlib.pyplot as plt
from mmagic.apis import MMagicInferencer

# 创建MMagicInferencer实例，并进行推理
result_out_dir = './resources/output/conditional/tutorial_conditinal_biggan_res.jpg'
editor = MMagicInferencer('biggan', model_setting=1)
results = editor.infer(label=1, result_out_dir=result_out_dir)
```

我们已经为用户提供了一个更友好的演示脚本。您可以使用以下命令使用demo/mmagic_inference_demo.py:

```
python demo/mmagic_inference_demo.py \
        --model-name biggan \
        --model-setting 1 \
        --label 1 \
        --result-out-dir demo_conditional_biggan_res.jpg
```

8.2.3 扩散模型

MMagic 提供了使用扩散模型进行图像采样的高级 API。

```
import mmcv
import matplotlib.pyplot as plt
from mmagic.apis import MMagicInferencer

# 创建MMagicInferencer实例，并进行推理
editor = MMagicInferencer(model_name='stable_diffusion')
text_prompts = 'A panda is having dinner at KFC'
result_out_dir = './resources/output/text2image/tutorial_text2image_sd_res.png'
editor.infer(text=text_prompts, result_out_dir=result_out_dir)
```

通过以下命令使用demo/mmagic_inference_demo.py

```
python demo/mmagic_inference_demo.py \
        --model-name stable_diffusion \
        --text "A panda is having dinner at KFC" \
        --result-out-dir demo_text2image_stable_diffusion_res.png
```

8.3 应用

8.3.1 文生图

文生图模型将文本作为输入，输出一张图片。我们以’controlnet-canny’为例。

```
import cv2
import numpy as np
import mmcv
from mmengine import Config
from PIL import Image

from mmagic.registry import MODELS
from mmagic.utils import register_all_modules

register_all_modules()

cfg = Config.fromfile('configs/controlnet/controlnet-canny.py')
controlnet = MODELS.build(cfg.model).cuda()

control_url = 'https://user-images.githubusercontent.com/28132635/230288866-99603172-
↪04cb-47b3-8adb-d1aa532d1d2c.jpg'
control_img = mmcv.imread(control_url)
control = cv2.Canny(control_img, 100, 200)
control = control[:, :, None]
control = np.concatenate([control] * 3, axis=2)
control = Image.fromarray(control)

prompt = 'Room with blue walls and a yellow ceiling.'

output_dict = controlnet.infer(prompt, control=control)
samples = output_dict['samples']
```

通过以下命令使用demo/mmagic_inference_demo.py

```
python demo/mmagic_inference_demo.py \
        --model-name controlnet \
```

(下页继续)

(续上页)

```
        --model-setting 1 \
        --text "Room with blue walls and a yellow ceiling." \
        --control 'https://user-images.githubusercontent.com/28132635/230297033-
↪4f5c32df-365c-4cf4-8e4f-1b76a4cbb0b7.png' \
        --result-out-dir demo_text2image_controlnet_canny_res.png
```

8.3.2 图生图的转换

MMagic 提供了使用图像翻译模型进行图像翻译的高级 API。下面是构建 Pix2Pix 并获取翻译图像的示例。

```
import mmcv
import matplotlib.pyplot as plt
from mmagic.apis import MMagicInferencer

# Create a MMagicInferencer instance and infer
editor = MMagicInferencer('pix2pix')
results = editor.infer(img=img_path, result_out_dir=result_out_dir)
```

通过以下命令使用demo/mmagic_inference_demo.py

```
python demo/mmagic_inference_demo.py \
        --model-name pix2pix \
        --img ${IMAGE_PATH} \
        --result-out-dir ${SAVE_PATH}
```

8.3.3 3D 感知生成

```
import mmcv
import matplotlib.pyplot as plt
from mmagic.apis import MMagicInferencer

# Create a MMagicInferencer instance and infer
result_out_dir = './resources/output/eg3d-output'
editor = MMagicInferencer('eg3d')
results = editor.infer(result_out_dir=result_out_dir)
```

通过以下命令使用demo/mmagic_inference_demo.py

```
python demo/mmagic_inference_demo.py \
    --model-name eg3d \
    --result-out-dir ./resources/output/eg3d-output
```

8.3.4 图像超分

图像超分辨率模型接受一张图像作为输入，并输出一张高分辨率图像。我们以‘esrgan’为例。

```
import mmcv
import matplotlib.pyplot as plt
from mmagic.apis import MMagicInferencer

# Create a MMagicInferencer instance and infer
img = './resources/input/restoration/0901x2.png'
result_out_dir = './resources/output/restoration/tutorial_restoration_esrgan_res.png'
editor = MMagicInferencer('esrgan')
results = editor.infer(img=img, result_out_dir=result_out_dir)
```

通过以下命令使用demo/mmagic_inference_demo.py

```
python demo/mmagic_inference_demo.py \
        --model-name esrgan \
        --img ${IMAGE_PATH} \
        --result-out-dir ${SAVE_PATH}
```

8.3.5 视频超分

```
import os
from mmagic.apis import MMagicInferencer
from mmengine import mkdir_or_exist

# Create a MMagicInferencer instance and infer
video = './resources/input/video_interpolation/b-3LLDhc4EU_000000_000010.mp4'
result_out_dir = './resources/output/video_super_resolution/tutorial_video_super_
↪resolution_basicvsr_res.mp4'
mkdir_or_exist(os.path.dirname(result_out_dir))
editor = MMagicInferencer('basicvsr')
results = editor.infer(video=video, result_out_dir=result_out_dir)
```

通过以下命令使用demo/mmagic_inference_demo.py

```
python demo/mmagic_inference_demo.py \
        --model-name basicvsr \
        --video ./resources/input/video_restoration/QUuC4vJs_000084_000094_400x320.
↪mp4 \
        --result-out-dir ./resources/output/video_restoration/demo_video_restoration_
↪basicvsr_res.mp4
```

8.3.6 视频帧插值

视频插值模型接受一个视频作为输入，并输出一个插值后的视频。我们以‘flavr’为例。

```
import os
from mmagic.apis import MMagicInferencer
from mmengine import mkdir_or_exist

# Create a MMagicInferencer instance and infer
video = './resources/input/video_interpolation/b-3LLDhc4EU_000000_000010.mp4'
result_out_dir = './resources/output/video_interpolation/tutorial_video_interpolation_
↪flavr_res.mp4'
mkdir_or_exist(os.path.dirname(result_out_dir))
editor = MMagicInferencer('flavr')
results = editor.infer(video=video, result_out_dir=result_out_dir)
```

8.3.7 图像修补

修复模型接受一对屏蔽图像和屏蔽蒙版作为输入，并输出一个修复后的图像。我们以‘global_local’为例。

```
import mmcv
import matplotlib.pyplot as plt
from mmagic.apis import MMagicInferencer

img = './resources/input/matting/GT05.jpg'
trimap = './resources/input/matting/GT05_trimap.jpg'

# 创建MMagicInferencer实例，并进行推理
result_out_dir = './resources/output/matting/tutorial_matting_gca_res.png'
editor = MMagicInferencer('gca')
results = editor.infer(img=img, trimap=trimap, result_out_dir=result_out_dir)
```

8.3.8 图像抠图

抠图模型接受一对图像和修剪映射作为输入，并输出一个 alpha 图像。我们以‘gca’为例。

```
import mmcv
import matplotlib.pyplot as plt
from mmagic.apis import MMagicInferencer

img = './resources/input/matting/GT05.jpg'
trimap = './resources/input/matting/GT05_trimap.jpg'

```

(下页继续)

(续上页)

```
# 创建MMagicInferencer实例，并进行推理
result_out_dir = './resources/output/matting/tutorial_matting_gca_res.png'
editor = MMagicInferencer('gca')
results = editor.infer(img=img, trimap=trimap, result_out_dir=result_out_dir)
```

通过以下命令使用demo/mmagic_inference_demo.py

```
python demo/mmagic_inference_demo.py \
        --model-name gca  \
        --img ./resources/input/matting/GT05.jpg \
        --trimap ./resources/input/matting/GT05_trimap.jpg \
        --result-out-dir ./resources/output/matting/demo_matting_gca_res.png
```

8.3.9 图像恢复

```
import mmcv
import matplotlib.pyplot as plt
from mmagic.apis import MMagicInferencer

# Create a MMagicInferencer instance and infer
img = './resources/input/restoration/0901x2.png'
result_out_dir = './resources/output/restoration/tutorial_restoration_nafnet_res.png'
editor = MMagicInferencer('nafnet')
results = editor.infer(img=img, result_out_dir=result_out_dir)
```

通过以下命令使用demo/mmagic_inference_demo.py

```
python demo/mmagic_inference_demo.py \
        --model-name nafnet \
        --img ./resources/input/restoration/0901x2.png \
        --result-out-dir ./resources/output/restoration/demo_restoration_nafnet_res.
↪png
```

8.3.10 图像上色

```
import mmcv
import matplotlib.pyplot as plt
from mmagic.apis import MMagicInferencer

# Create a MMagicInferencer instance and infer
img = 'https://github-production-user-asset-6210df.s3.amazonaws.com/49083766/
↪245713512-de973677-2be8-4915-911f-fab90bb17c40.jpg'
```

(下页继续)

(续上页)

```
result_out_dir = './resources/output/colorization/tutorial_colorization_res.png'
editor = MMagicInferencer('inst_colorization')
results = editor.infer(img=img, result_out_dir=result_out_dir)
```

通过以下命令使用demo/mmagic_inference_demo.py

```
python demo/mmagic_inference_demo.py \
        --model-name inst_colorization \
        --img https://github-production-user-asset-6210df.s3.amazonaws.com/49083766/
↪245713512-de973677-2be8-4915-911f-fab90bb17c40.jpg \
        --result-out-dir demo_colorization_res.png
```

8.4 以前的版本

如果您想使用已弃用的演示，请使用MMagic v1.0.0rc7并参考旧教程。

CHAPTER 9

教程 4：在 MMagic 环境下训练与测试

在该部分中，您将学到如何在 MMagic 环境下完成训练与测试

我们提供如下教程：

- 预先准备
- 在 MMagic 中测试模型
  - 在单个 GPU 上测试
  - 在多个 GPU 上测试
  - 在 Slurm 上测试
  - 使用特定指标进行测试
- 在 MMagic 中训练模型
  - 在单个 GPU 上训练
  - 在多个 GPU 上训练
  - 在多个节点上训练
  - 在 Slurm 上训练
  - 使用特定的评估指标进行训练

9.1 预先准备

用户需要首先准备数据集 从而能够在 MMagic 环境中训练和测试。

9.2 在 MMagic 中测试模型

9.2.1 在单个 GPU 上测试

您可以通过如下命令使用单个 GPU 来测试预训练模型。

```
python tools/test.py ${CONFIG_FILE} ${CHECKPOINT_FILE}
```

例如：

```
python tools/test.py configs/example_config.py work_dirs/example_exp/example_model_
↪20200202.pth
```

9.2.2 在多个 GPU 上测试

MMagic 支持使用多个 GPU 测试，能够极大地节约模型测试时间。可以通过如下命令使用多个 GPU 来测试预训练模型。

```
./tools/dist_test.sh ${CONFIG_FILE} ${CHECKPOINT_FILE} ${GPU_NUM}
```

例如：

```
./tools/dist_test.sh configs/example_config.py work_dirs/example_exp/example_model_
↪20200202.pth
```

9.2.3 在 Slurm 上测试

如果您在由 slurm 管理的集群上运行 MMagic，可以使用脚本 `slurm_test.sh`。(此脚本还支持单机测试。)

```
[GPUS=${GPUS}] ./tools/slurm_test.sh ${PARTITION} ${JOB_NAME} ${CONFIG_FILE} $
↪{CHECKPOINT_FILE}
```

下面是一个使用 8 个 GPU 在“dev”分区上测试一个示例模型的例子，作业名称为“test”。

```
GPUS=8 ./tools/slurm_test.sh dev test configs/example_config.py work_dirs/example_exp/
↪example_model_20200202.pth
```

您可以检查 slurm_test.sh 以获取完整的参数和环境变量。

9.2.4 使用特定指标进行测试

MMagic 提供各种评**估值指标**，例如：MS-SSIM、SWD、IS、FID、Precision&Recall、PPL、Equivarience、TransFID、TransIS 等。我们在tools/test.py中为所有模型提供了统一的评估脚本。如果用户想用一些指标来评估他们的模型，你可以像这样将 `metrics` 添加到你的配置文件中:

```
# 在文件 configs/styleganv2/stylegan2_c2_ffhq_256_b4x8_800k.py 的末尾
metrics = [
    dict(
        type='FrechetInceptionDistance',
        prefix='FID-Full-50k',
        fake_nums=50000,
        inception_style='StyleGAN',
        sample_model='ema'),
    dict(type='PrecisionAndRecall', fake_nums=50000, prefix='PR-50K'),
    dict(type='PerceptualPathLength', fake_nums=50000, prefix='ppl-w')
]
```

如上所述，`metrics` 由多个指标字典组成。每个指标包含 `type` 来表示其类别。`fake_nums` 表示模型生成的图像数量。有些指标会输出一个结果字典，您也可以设置 `prefix` 来指定结果的前缀。如果将 FID 的前缀设置为 `FID-Full-50k`，则输出的示例可能是

```
FID-Full-50k/fid: 3.6561  FID-Full-50k/mean: 0.4263  FID-Full-50k/cov: 3.2298
```

然后用户可以使用下面的命令测试模型:

```
bash tools/dist_test.sh ${CONFIG_FILE} ${CKPT_FILE}
```

如果您在 slurm 环境中，请使用如下命令切换到 tools/slurm_test.sh：

```
sh slurm_test.sh ${PLATFORM} ${JOBNAME} ${CONFIG_FILE} ${CKPT_FILE}
```

9.3 在 MMagic 中训练模型

MMagic 支持多种训练方式:

1. 在单个 GPU 上训练
2. 在单个 GPU 上训练
3. 在多个节点上训练
4. 在 Slurm 上训练

Specifically, all outputs (log files and checkpoints) will be saved to the working directory, which is specified by `work_dir` in the config file.

9.3.1 在单个 GPU 上训练

```
CUDA_VISIBLE=0 python tools/train.py configs/example_config.py --work-dir work_dirs/
↪example
```

9.3.2 在多个节点上训练

要在多台机器上启动分布式训练，这些机器可以通过 IP 访问，运行以下命令:

在第一台机器上:

```
NNODES=2 NODE_RANK=0 PORT=$MASTER_PORT MASTER_ADDR=$MASTER_ADDR tools/dist_train.sh
↪$CONFIG $GPUS
```

在第二台机器上:

```
NNODES=2 NODE_RANK=1 PORT=$MASTER_PORT MASTER_ADDR=$MASTER_ADDR tools/dist_train.sh
↪$CONFIG $GPUS
```

为了提高网络通信速度，建议使用高速网络硬件，如 Infiniband。请参考 PyTorch docs 以获取更多信息。

9.3.3 在多个 GPU 上训练

```
./tools/dist_train.sh ${CONFIG_FILE} ${GPU_NUM} [optional arguments]
```

9.3.4 在 Slurm 上训练

如果您在由 slurm 管理的集群上运行 MMagic，可以使用脚本 `slurm_train.sh`。(此脚本还支持单机测试。)

```
[GPUS=${GPUS}] ./tools/slurm_train.sh ${PARTITION} ${JOB_NAME} ${CONFIG_FILE} ${WORK_
↪DIR}
```

下面是一个使用 8 个 gpu 在 dev 分区上训练 inpainting 模型的示例。

```
GPUS=8 ./tools/slurm_train.sh dev configs/inpainting/gl_places.py /nfs/xxxx/gl_places_
↪256
```

你可以在 slurm_train.sh 上查阅完整参数和环境变量。

9.3.5 可选参数

- `--amp`：此参数用于固定精度训练。
- `--resume`：此参数用于在训练中止时自动恢复。

9.4 使用特定的评估指标进行训练

受益于 mmengine 的 Runner，我们可以在训练过程中对模型进行简单的评估，如下所示。

```
# 定义指标
metrics = [
    dict(
        type='FrechetInceptionDistance',
        prefix='FID-Full-50k',
        fake_nums=50000,
        inception_style='StyleGAN')
]

# 定义dataloader
val_dataloader = dict(
    batch_size=128,
    num_workers=8,
    dataset=dict(
        type='BasicImageDataset',
        data_root='data/celeba-cropped/',
        pipeline=[
            dict(type='LoadImageFromFile', key='img'),
            dict(type='Resize', scale=(64, 64)),
            dict(type='PackInputs')
        ]),
    sampler=dict(type='DefaultSampler', shuffle=False),
    persistent_workers=True)

# 定义 val interval
train_cfg = dict(by_epoch=False, val_begin=1, val_interval=10000)

# 定义 val loop 和 evaluator
val_cfg = dict(type='MultiValLoop')
val_evaluator = dict(type='Evaluator', metrics=metrics)
```

可以设置 val_begin 和 val_interval 来调整何时开始验证和验证间隔。

有关指标的详细信息，请参考*metrics' guide*.

CHAPTER 10

教程 5：使用评价指标

MMagic 支持 **17 个指标**以评估模型质量。

有关用法，请参阅*MMagic 中的训练与测试*。

在这里，我们将逐个介绍不同指标的详细信息。

本文的结构如下:

1. *MAE*
2. *MSE*
3. *PSNR*
4. *SNR*
5. *SSIM*
6. *NIQE*
7. *SAD*
8. *MattingMSE*
9. *GradientError*
10. *ConnectivityError*
11. FID and TransFID
12. IS and TransIS
13. *Precision and Recall*

14. *PPL*
15. *SWD*
16. *MS-SSIM*
17. *Equivarience*

10.1 MAE

MAE 是图像的平均绝对误差。要使用 MAE 进行评估，请在配置文件中添加以下配置:

```
val_evaluator = [
    dict(type='MAE'),
]
```

10.2 MSE

MSE 是图像的均方误差。要使用 MSE 进行评估，请在配置文件中添加以下配置:

```
val_evaluator = [
    dict(type='MSE'),
]
```

10.3 PSNR

PSNR 是峰值信噪比。我们的实现方法来自 https://en.wikipedia.org/wiki/Peak_signal-to-noise_ratio。要使用 PSNR 进行评估，请在配置文件中添加以下配置:

```
val_evaluator = [
    dict(type='PSNR'),
]
```

10.4 SNR

SNR 是信噪比。我们的实现方法来自 https://en.wikipedia.org/wiki/Signal-to-noise_ratio。要使用 SNR 进行评估，请在配置文件中添加以下配置:

```
val_evaluator = [
    dict(type='SNR'),
]
```

10.5 SSIM

SSIM 是图像的结构相似度，在图像质量评估: 从错误可见性到结构相似度中提出。我们实现的结果与 https://ece.uwaterloo.ca/~z70wang/research/ssim/官方发布的 MATLAB 代码相同。要使用 SSIM 进行评估，请在配置文件中添加以下配置:

```
val_evaluator = [
    dict(type='SSIM'),
]
```

10.6 NIQE

NIQE 是自然图像质量评估指标，在制作’完全盲’图像质量分析仪中提出。我们的实现可以产生几乎与官方 MATLAB 代码相同的结果:http://live.ece.utexas.edu/research/quality/niqe_release.zip。要使用 NIQE 进行评估，请在配置文件中添加以下配置:

```
val_evaluator = [
    dict(type='NIQE'),
]
```

10.7 SAD

SAD 是图像抠图的绝对误差和。该指标计算每个像素的绝对差和所有像素的总和。要使用 SAD 进行评估，请在配置文件中添加以下配置:

```
val_evaluator = [
    dict(type='SAD'),
]
```

10.8 MattingMSE

MattingMSE 是图像抠图的均方误差。要使用 MattingMSE 进行评估，请在配置文件中添加以下配置:

```
val_evaluator = [
    dict(type='MattingMSE'),
]
```

10.9 GradientError

GradientError 是用于评估 alpha matte 预测的梯度误差。要使用 GradientError 进行评估，请在配置文件中添加以下配置:

```
val_evaluator = [
    dict(type='GradientError'),
]
```

10.10 ConnectivityError

ConnectivityError 是用于评估 alpha matte 预测的连通性误差。要使用 ConnectivityError 进行评估，请在配置文件中添加以下配置:

```
val_evaluator = [
    dict(type='ConnectivityError'),
]
```

10.11 FID 和 TransFID

Fréchet 初始距离是两个图像数据集之间相似度的度量。它被证明与人类对视觉质量的判断有很好的相关性，最常用于评估生成对抗网络样本的质量。FID 是通过计算两个高斯函数之间的 Fréchet 距离来计算的，这些高斯函数适合于 Inception 网络的特征表示。

在 `MMagic` 中，我们提供了两个版本的 FID 计算。一个是常用的 PyTorch 版本，另一个用于 StyleGAN。同时，我们在 StyleGAN2-FFHQ1024 模型中比较了这两种实现之间的差异 (详细信息可以在这里找到 [https://github.com/open-mmlab/mmagic/blob/main/configs/styleganv2/README.md])。幸运的是，最终结果只是略有不同。因此，我们建议用户采用更方便的 PyTorch 版本。

关于 PyTorch 版本和 Tero 版本: 常用的 PyTorch 版本采用修改后的 InceptionV3 网络提取真假图像特征。然而，Tero 的 FID 需要 Tensorflow InceptionV3 的脚本模块。注意，应用此脚本模块需要’ PyTorch >= 1.6.0 ‘。

关于提取真实的初始数据: 为了方便用户，在测试时自动提取真实的特征并保存在本地，存储的特征在下次测试时自动读取。具体来说，我们将根据用于计算实际特性的参数计算一个哈希值，并使用哈希值来标记特性文件，在测试时，如果’inception_pkl ‘没有设置，我们将在’MMAGIC_CACHE_DIR ‘(~/.cache/openmmlab/mmagic/) 中寻找该特性。如果未找到缓存的初始 pkl，则将执行提取。

要使用 FID 指标，请在配置文件中添加以下配置:

```
metrics = [
    dict(
        type='FrechetInceptionDistance',
        prefix='FID-Full-50k',
        fake_nums=50000,
        inception_style='StyleGAN',
        sample_model='ema')
]
```

如果您在一台新机器上工作，那么您可以复制’MMAGIC_CACHE_DIR’中的’pkl’文件，将它们复制到新机器并设置’inception_pkl’字段。

```
metrics = [
    dict(
        type='FrechetInceptionDistance',
        prefix='FID-Full-50k',
        fake_nums=50000,
        inception_style='StyleGAN',
        inception_pkl=
        'work_dirs/inception_pkl/inception_state-capture_mean_cov-full-
↪33ad4546f8c9152e4b3bdb1b0c08dbaf.pkl',  # copied from old machine
        sample_model='ema')
]
```

‘TransFID’与’FID’的用法相同，但 TransFID 是为’Pix2Pix’和’CycleGAN’等翻译模型设计的，适用于我们的评估器。更多信息您可以参考*evaluation*。

10.12 IS 和 TransIS

Inception 评分是评估生成图像质量的客观指标，在改进的训练 GANs 技术中提出。它使用一个 InceptionV3 模型来预测生成的图像的类别，并假设: 1) 如果图像质量高，它将被归类到特定的类别。2) 如果图像具有较高的多样性，则图像的类别范围将很广。因此，条件概率和边际概率的 kl -散度可以指示生成图像的质量和多样性。您可以在’metrics.py’中看到完整的实现，它指向 https://github.com/sbarratt/inception-score-pytorch/blob/master/inception_score.py。如果您想使用’IS’指标评估模型，请在配置文件中添加以下配置:

```
# at the end of the configs/biggan/biggan_2xb25-500kiters_cifar10-32x32.py
metrics = [
    xxx,
    dict(
        type='IS',
        prefix='IS-50k',
        fake_nums=50000,
        inception_style='StyleGAN',
        sample_model='ema')
]
```

需要注意的是，Inception V3 的选择和图像大小的调整方法会显著影响最终的 IS 评分。因此，我们强烈建议用户可以下载Tero’s script model of Inception V3(加载此脚本模型需要 torch >= 1.6)，并使用’Bicubic’插值与’Pillow’后端。

对应于 config，您可以设置’resize_method’和’use_pillow_resize’用于图像大小的调整。您也可以将’inception_style’设置为’StyleGAN’用于推荐的 tero 的初始模型，或’PyTorch’用于 torchvision 的实现。对于没有互联网的环境，您可以下载初始的权重，并将’inception_path’设置为您的初始模型。

我们还调查了数据加载管线和预训练的 Inception V3 版本对 IS 结果的影响。所有 IS 都在同一组图像上进行评估，这些图像是从 ImageNet 数据集中随机选择的。

‘TransIS’与’IS’的用法相同，但 TransIS 是为’Pix2Pix’和’CycleGAN’这样的翻译模型设计的，这是为我们的评估器改编的。更多信息可参考*evaluation*。

10.13 Precision and Recall

我们的’Precision and Recall’实现遵循 StyleGAN2 中使用的版本。在该度量中，采用 VGG 网络对图像进行特征提取。不幸的是，我们还没有发现 PyTorch VGG 实现与 StyleGAN2 中使用的 Tero 版本产生类似的结果。(关于差异，请参阅这个文件。) 因此，在我们的实现中，我们默认采用Teor’s VGG网络。需要注意的是，应用这个脚本模块需要’PyTorch >= 1.6.0’。如果使用较低的 PyTorch 版本，我们将使用 PyTorch 官方 VGG 网络进行特征提取。要使用’P&R ‘进行评估，请在配置文件中添加以下配置:

```
metrics = [
    dict(type='PrecisionAndRecall', fake_nums=50000, prefix='PR-50K')
]
```

10.14 PPL

当在两个随机输入之间进行插值时，感知路径长度测量连续图像（其 VGG16 嵌入）之间的差异。剧烈的变化意味着多个特征一起发生了变化，它们可能会叠加在一起。通过实验表明，较小的 PPL 分数表明整体图像质量较高。作为该指标的基础，我们使用基于感知的成对图像距离，该距离被计算为两个 VGG16 嵌入之间的加权差，其中权重被拟合，从而评价指标与人类的感知相似性判断一致。如果我们将潜在空间插值路径细分为线性段，我们可以将该分段路径的总感知长度定义为每个段上感知差异的总和，并且感知路径长度的自然定义将是无限细分下的总和的极限，但在实践中，我们使用一个小的细分 $`\epsilon=10^{-4}`$ 来近似它。因此，潜在 spaceZ 中所有可能端点的平均感知路径长度为

$$`L_Z = E[\frac{1}{\epsilon^2}d(G(slerp(z_1,z_2;t))), G(slerp(z_1,z_2;t+\epsilon)))]`$$

以类似的方式计算潜在 space W 中的平均感知路径长度：：

$$`L_Z = E[\frac{1}{\epsilon^2}d(G(slerp(z_1,z_2;t))), G(slerp(z_1,z_2;t+\epsilon)))]`$$

当 $`z_1, z_2 \sim P(z)`$，如果我们设置 sampling 为 full，则 $` t \sim U(0,1)`$，如果设置 sampling 为 end，则 $` t \in \{0,1\}`$。$` G`$ 是生成器 (i.e. $` g \circ f`$ 用于 style-based 网络)，$` d(.,.)`$ 用于计算结果图像之间的感知距离。我们通过取 100,000 个样本来计算期望 (在代码中将’num_images ‘设置为 50,000)。

您可以在’metrics.py’中找到完整的实现，参考 https://github.com/rosinality/stylegan2-pytorch/blob/master/ppl.py。如果您想使用’PPL’指标评估模型，请在配置文件中添加以下配置:

```
# at the end of the configs/styleganv2/stylegan2_c2_ffhq_1024_b4x8.py
metrics = [
    xxx,
    dict(type='PerceptualPathLength', fake_nums=50000, prefix='ppl-w')
]
```

10.15 SWD

切片 Wasserstein 距离是概率分布的差异度量，距离越小表示生成的图像越真实。我们获得每个图像的拉普拉斯金字塔，并从拉普拉斯金字塔中提取小块作为描述符，然后可以通过获取真实和伪描述符切片的 Wasserstein 距离来计算 SWD。您可以在’metrics.py’中看到完整的实现，参考 https://github.com/tkarras/progressive_growing_of_gans/blob/master/metrics/sliced_wasserstein.py。如果您想使用’SWD’指标评估模型，请在配置文件中添加以下配置:

```
# at the end of the configs/dcgan/dcgan_1xb128-5epoches_lsun-bedroom-64x64.py
metrics = [
    dict(
```

(下页继续)

(续上页)

```
        type='SWD',
        prefix='swd',
        fake_nums=16384,
        sample_model='orig',
        image_shape=(3, 64, 64))
]
```

10.16 MS-SSIM

采用多尺度结构相似度来衡量两幅图像的相似度。我们在这里使用 MS-SSIM 来衡量生成图像的多样性，MS-SSIM 得分低表示生成图像的多样性高。您可以在’metrics.py’中看到完整的实现，参考 https://github.com/tkarras/progressive_growing_of_gans/blob/master/metrics/ms_ssim.py。如果您想使用’MS-SSIM’指标评估模型，请在配置文件中添加以下配置:

```
# at the end of the configs/dcgan/dcgan_1xb128-5epoches_lsun-bedroom-64x64.py
metrics = [
    dict(
        type='MS_SSIM', prefix='ms-ssim', fake_nums=10000,
        sample_model='orig')
]
```

10.17 Equivarience

生成模型的等价性是指模型正变换和几何变换的互换性。目前这个指标只针对 StyleGANv3 计算，您可以在’metrics.py’中看到完整的实现，参考 https://github.com/NVlabs/stylegan3/blob/main/metrics/equivariance.py。如果您想使用’Equivarience’指标评估模型，请在配置文件中添加以下配置:

```
# at the end of the configs/styleganv3/stylegan3-t_gamma2.0_8xb4-fp16-noaug_ffhq-
↪256x256.py
metrics = [
    dict(
        type='Equivariance',
        fake_nums=50000,
        sample_mode='ema',
        prefix='EQ',
        eq_cfg=dict(
            compute_eqt_int=True, compute_eqt_frac=True, compute_eqr=True))
]
```

CHAPTER 11

教程 6：可视化

图像的可视化是衡量图像处理、编辑和合成质量的重要手段。在配置文件中使用 `visualizer` 可以在训练或测试时保存可视化结果。您可以跟随MMEngine 文档学习可视化的用法。MMagic 提供了一套丰富的可视化功能。在本教程中，我们将介绍 MMagic 提供的可视化函数的用法。

- 教程 6: 可视化
  - 概述
    - *GAN* 的可视化配置
    - 图像翻译模型的可视化配置
    - 扩散模型的可视化配置
    - 图像补全模型的可视化配置
    - 图像抠图模型的可视化配置
    - *SISR/VSR/VFI* 等模型的可视化配置
  - 可视化钩子
  - 可视化器
  - 可视化后端
    - 在不同的存储后端可视化

11.1 概述

建议先学习 设计文档 里关于可视化的基本概念。

在 MMagic 中，训练或测试过程的可视化需要配置三个组件:VisualizationHook、Visualizer 和 VisBackend, 如下图表展示了 Visualizer 和 VisBackend 的关系。

VisualizationHook 在训练期间以固定的间隔获取模型输出的可视化结果，并将其传递给 **Visualizer**。**Visualizer** 负责将原始可视化结果转换为所需的类型 (png, gif 等)，然后将其传输到 **VisBackend** 进行存储或显示。

11.1.1 GAN 的可视化配置

对于像 StyleGAN 和 SAGAN 这样的 GAN 模型，通常的配置如下所示:

```
# 可视化钩子
custom_hooks = [
    dict(
        type='VisualizationHook',
        interval=5000,  # 配置可视化勾子的间隔数
        fixed_input=True,  # 是否固定噪声输入生成图像
        vis_kwargs_list=dict(type='GAN', name='fake_img')  #␣
↪对于GAN模型预先定义可视化参数
    )
]
# 可视化后端
vis_backends = [
    dict(type='VisBackend'),  # 可视化后端用于存储。
    dict(type='WandbVisBackend',  # 可以上传至Wandb的可视化后端
        init_kwargs=dict(
            project='MMagic',   # Wandb项目名
            name='GAN-Visualization-Demo'  # Wandb实验命名
        ))
]
# 可视化器
visualizer = dict(type='Visualizer', vis_backends=vis_backends)
```

如果您将指数移动平均 (EMA) 应用于生成器，并希望可视化 EMA 模型，您可以修改 VisualizationHook 的配置，如下所示:

```
custom_hooks = [
   dict(
       type='VisualizationHook',
       interval=5000,
       fixed_input=True,
```

(下页继续)

(续上页)

```
        # 同时在`fake_img`中可视化ema以及orig
        vis_kwargs_list=dict(
            type='Noise',
            name='fake_img',  # 使用`fake_img`保存图片
            sample_model='ema/orig',  # 对于`NoiseSampler`特别定义参数
            target_keys=['ema.fake_img', 'orig.fake_img']  # 指定的可视化的键值
        ))
]
```

11.1.2 图像翻译模型的可视化配置

对于 CycleGAN、Pix2Pix 等翻译模型，可以形成如下可视化配置:

```
# 可视化钩子
custom_hooks = [
    dict(
        type='VisualizationHook',
        interval=5000,
        fixed_input=True,
        vis_kwargs_list=[
            dict(
                type='Translation',  # 在训练集可视化结果
                name='trans'),  #  保存`trans`字段的图像
            dict(
                type='Translationval',  # 在验证集可视化结果
                name='trans_val'),  #  保存`trans_val`字段的图像
        ])
]
# 可视化后端
vis_backends = [
    dict(type='VisBackend'),  # 可视化后端用于存储。
    dict(type='WandbVisBackend',  # 可以上传至Wandb的可视化后端
        init_kwargs=dict(
            project='MMagic',   # Wandb项目名
            name='Translation-Visualization-Demo'  # Wandb实验命名
        ))
]
# 可视化器
visualizer = dict(type='Visualizer', vis_backends=vis_backends)
```

11.1.3 扩散模型的可视化配置

对于扩散模型，例如 Improved-DDPM，我们可以使用以下配置通过 gif 来可视化去噪过程:

```
# 可视化钩子
custom_hooks = [
    dict(
        type='VisualizationHook',
        interval=5000,
        fixed_input=True,
        vis_kwargs_list=dict(type='DDPMDenoising'))  # 对于DDPM模型预先定义可视化参数
]
# 可视化后端
vis_backends = [
    dict(type='VisBackend'),  # 可视化后端用于存储。
    dict(type='WandbVisBackend',  # 可以上传至Wandb的可视化后端
        init_kwargs=dict(
            project='MMagic',   # Wandb项目名
            name='Diffusion-Visualization-Demo'  # Wandb实验命名
        ))
]
# 可视化器
visualizer = dict(type='Visualizer', vis_backends=vis_backends)
```

11.1.4 图像补全模型的可视化配置

对于图像补全模型，如 AOT-GAN 和 Global&Local，通常的配置如下所示:

```
# 可视化后端
vis_backends = [dict(type='LocalVisBackend')]
# 可视化器
visualizer = dict(
    type='ConcatImageVisualizer',
    vis_backends=vis_backends,
    fn_key='gt_path',
    img_keys=['gt_img', 'input', 'pred_img'],
    bgr2rgb=True)
# 可视化钩子
custom_hooks = [dict(type='BasicVisualizationHook', interval=1)]
```

11.1.5 图像抠图模型的可视化配置

对于 DIM 和 GCA 等图像抠图模型，通常的配置如下所示:

```
# 可视化后端
vis_backends = [dict(type='LocalVisBackend')]
# 可视化器
visualizer = dict(
    type='ConcatImageVisualizer',
    vis_backends=vis_backends,
    fn_key='trimap_path',
    img_keys=['pred_alpha', 'trimap', 'gt_merged', 'gt_alpha'],
    bgr2rgb=True)
# 可视化钩子
custom_hooks = [dict(type='BasicVisualizationHook', interval=1)]
```

11.1.6 SISR/VSR/VFI 等模型的可视化配置

对于 SISR/VSR/VFI 等模型，如 EDSR, EDVR 和 CAIN，通常的配置如下所示:

```
# 可视化后端
vis_backends = [dict(type='LocalVisBackend')]
# 可视化器
visualizer = dict(
    type='ConcatImageVisualizer',
    vis_backends=vis_backends,
    fn_key='gt_path',
    img_keys=['gt_img', 'input', 'pred_img'],
    bgr2rgb=False)
# 可视化钩子
custom_hooks = [dict(type='BasicVisualizationHook', interval=1)]
```

可视化钩子、可视化器和可视化后端组件的具体配置如下所述.

11.2 可视化钩子

在 MMagic 中，我们使用 BasicVisualizationHook 和 VisualizationHook 作为可视化钩子。可视化钩子支持以下三种情况。

(1) 修改 vis_kwargs_list，实现特定输入下模型输出的可视化，适用于特定数据输入下 GAN 生成结果和图像翻译模型的翻译结果的可视化等。下面是两个典型的例子:

```
# input as dict
vis_kwargs_list = dict(
    type='Noise',  # 使用'Noise'采样生成模型输入
    name='fake_img',  # 定义保存图像的命名
)

# input as list of dict
vis_kwargs_list = [
    dict(type='Arguments',  # 使用'Arguments'采样生成模型输入
         name='arg_output',  # 定义保存图像的命名
         vis_mode='gif',  # 通过gif来可视化
         forward_kwargs=dict(forward_mode='sampling', sample_kwargs=dict(show_
↪pbar=True))  # 为'Arguments'采样定义参数
    ),
    dict(type='Data',  # 在dataloader使用'Data'采样提供数据作为可视化输入
         n_samples=36,  # 定义多少采样生成可视化结果
         fixed_input=False,  # 定义对于可视化过程不固定输入
    )
]
```

vis_kwargs_list 接受字典或字典的列表作为输入。每个字典必须包含一个 type 字段，指示用于生成模型输入的采样器类型，并且每个字典还必须包含采样器所需的关键字字段 (例如:ArgumentSampler 要求参数字典包含 forward_kwargs)。

需要注意的是，此内容由相应的采样器检查，不受 BasicVisualizationHook 的限制。

此外，其他字段是通用字段 (例如:n_samples、n_row,name,fixed_input 等等)。如果没有传入，则使用 BasicVisualizationHook 初始化的默认值。

为了方便用户使用，MMagic 为 **GAN**、**Translation models**、**SinGAN** 和 **Diffusion models** 预置了可视化参数，用户可以通过以下配置直接使用预定义的可视化方法:

```
vis_kwargs_list = dict(type='GAN')
vis_kwargs_list = dict(type='SinGAN')
vis_kwargs_list = dict(type='Translation')
vis_kwargs_list = dict(type='TranslationVal')
vis_kwargs_list = dict(type='TranslationTest')
vis_kwargs_list = dict(type='DDPMDenoising')
```

11.3 可视化器

在 MMagic 中，我们实现了 ConcatImageVisualizer 和 Visualizer，它们继承自 mmengine.Visualizer。Visualizer 的基类是 ManagerMixin，这使得 Visualizer 成为一个全局唯一的对象。在实例化之后，Visualizer 可以在代码的任何地方通过 Visualizer.get_current_instance() 调用，如下所示:

```
# configs
vis_backends = [dict(type='VisBackend')]
visualizer = dict(
    type='Visualizer', vis_backends=vis_backends, name='visualizer')
```

```
# `get_instance()` 是为全局唯一实例化调用
VISUALIZERS.build(cfg.visualizer)

# 通过上述代码实例化后，您可以在任何位置调用`get_current_instance`方法来获取可视化器
visualizer = Visualizer.get_current_instance()
```

Visualizer 的核心接口是 add_datasample。通过这个界面，该接口将根据相应的 vis_mode 调用相应的绘图函数，以获得 np.ndarray 类型的可视化结果。然后调用 show 或 add_image 来直接显示结果或将可视化结果传递给预定义的 vis_backend。

11.4 可视化后端

- MMEngine 的基本 VisBackend 包括 LocalVisBackend、TensorboardVisBackend 和 WandbVisBackend。您可以关注MMEngine Documents了解更多有关它们的信息。
- VisBackend: **File System** 的后端。将可视化结果保存到相应位置。
- TensorboardVisBackend: **Tensorboard** 的后端。将可视化结果发送到 Tensorboard。
- WandbVisBackend: **Wandb** 的后端。将可视化结果发送到 Tensorboard。

一个 Visualizer 对象可以访问任意数量的 visbackend，用户可以在代码中通过类名访问后端。

```
# 配置文件
vis_backends = [dict(type='Visualizer'), dict(type='WandbVisBackend')]
visualizer = dict(
    type='Visualizer', vis_backends=vis_backends, name='visualizer')
```

```
# 代码内
VISUALIZERS.build(cfg.visualizer)
visualizer = Visualizer.get_current_instance()
```

(下页继续)

(续上页)

```
# 通过类名访问后端
gen_vis_backend = visualizer.get_backend('VisBackend')
gen_wandb_vis_backend = visualizer.get_backend('GenWandbVisBackend')
```

当有多个 VisBackend 具有相同的类名时，用户必须为每个 VisBackend 指定名称。

```
# 配置文件
vis_backends = [
    dict(type='VisBackend', name='gen_vis_backend_1'),
    dict(type='VisBackend', name='gen_vis_backend_2')
]
visualizer = dict(
    type='Visualizer', vis_backends=vis_backends, name='visualizer')
```

```
# 代码内
VISUALIZERS.build(cfg.visualizer)
visualizer = Visualizer.get_current_instance()

local_vis_backend_1 = visualizer.get_backend('gen_vis_backend_1')
local_vis_backend_2 = visualizer.get_backend('gen_vis_backend_2')
```

11.4.1 在不同的存储后端可视化

如果想用不同的存储后端（Wandb, Tensorboard, 或者远程窗口里常规的后端），像以下这样改配置文件的 vis_backends 就行了：

Local

```
vis_backends = [dict(type='LocalVisBackend')]
```

Tensorboard

```
vis_backends = [dict(type='TensorboardVisBackend')]
visualizer = dict(
    type='ConcatImageVisualizer', vis_backends=vis_backends, name='visualizer')
```

Wandb

```
vis_backends = [dict(type='WandbVisBackend', init_kwargs=dict(project={PROJECTS},␣
↪name={EXPNAME}))]
visualizer = dict(
    type='ConcatImageVisualizer', vis_backends=vis_backends, name='visualizer')
```

CHAPTER 12

教程 7：实用工具（待更新）

我们在 tools/ 目录下提供了很多有用的工具。

12.1 获取 FLOP 和参数量（实验性）

我们提供了一个改编自 flops-counter.pytorch 的脚本来计算模型的 FLOP 和参数量。

```
python tools/get_flops.py ${CONFIG_FILE} [--shape ${INPUT_SHAPE}]
```

例如，

```
python tools/get_flops.py configs/resotorer/srresnet.py --shape 40 40
```

你会得到以下的结果。

```
==============================
Input shape: (3, 40, 40)
Flops: 4.07 GMac
Params: 1.52 M
==============================
```

注：此工具仍处于实验阶段，我们不保证数字正确。您可以将结果用于简单的比较，但在技术报告或论文中采用它之前，请仔细检查它。

(1) FLOPs 与输入形状有关，而参数量与输入形状无关。默认输入形状为 (1, 3, 250, 250)。(2) 一些运算符不计

入 FLOP，如 GN 和自定义运算符。你可以通过修改 `mmcv/cnn/utils/flops_counter.py` 来添加对新运算符的支持。

12.2 发布模型

在将模型上传到 AWS 之前，您可能需要 (1) 将模型权重转换为 CPU tensors, (2) 删除优化器状态，和 (3) 计算模型权重文件的哈希并将哈希 ID 附加到文件名。

```
python tools/publish_model.py ${INPUT_FILENAME} ${OUTPUT_FILENAME}
```

例如，

```
python tools/publish_model.py work_dirs/example_exp/latest.pth example_model_20200202.
↪pth
```

最终输出文件名将是 `example_model_20200202-{hash id}.pth`.

12.3 转换为 ONNX（实验性）

我们提供了一个脚本将模型转换为 ONNX 格式。转换后的模型可以通过 Netron 等工具进行可视化。此外，我们还支持比较 Pytorch 和 ONNX 模型之间的输出结果。

```
python tools/pytorch2onnx.py
    ${CFG_PATH} \
    ${CHECKPOINT_PATH} \
    ${MODEL_TYPE} \
    ${IMAGE_PATH} \
    --trimap-path ${TRIMAP_PATH} \
    --output-file ${OUTPUT_ONNX} \
    --show \
    --verify \
    --dynamic-export
```

参数说明：

- `config`：模型配置文件的路径。
- `checkpoint`：模型模型权重文件的路径。
- `model_type`：配置文件的模型类型，选项：`inpainting`, `mattor`, `restorer`, `synthesizer`。
- `image_path`：输入图像文件的路径。
- `--trimap-path`：输入三元图文件的路径，用于 mattor 模型。
- `--output-file`: 输出 ONNX 模型的路径。默认为 `tmp.onnx`。

- `--opset-version` : ONNX opset 版本。默认为 11。
- `--show`: 确定是否打印导出模型的架构。默认为 `False`。
- `--verify`: 确定是否验证导出模型的正确性。默认为 `False`。
- `--dynamic-export`: 确定是否导出具有动态输入和输出形状的 ONNX 模型。默认为 `False`。

注：此工具仍处于试验阶段。目前不支持某些自定义运算符。我们现在只支持 `mattor` 和 `restorer`。

12.3.1 支持导出到 ONNX 的模型列表

下表列出了保证可导出到 ONNX 并可在 ONNX Runtime 中运行的模型。

注:

- *以上所有模型均使用 Pytorch==1.6.0 和 onnxruntime==1.5.1*
- 如果您遇到上面列出的模型的任何问题，请创建一个 issue，我们会尽快处理。对于列表中未包含的型号，请尝试自行解决。
- 由于此功能是实验性的并且可能会快速更改，请始终尝试使用最新的 `mmcv` 和 `mmagic`。

12.4 将 ONNX 转换为 TensorRT（实验性）

我们还提供了将 ONNX 模型转换为 TensorRT 格式的脚本。此外，我们支持比较 ONNX 和 TensorRT 模型之间的输出结果。

```
python tools/onnx2tensorrt.py
    ${CFG_PATH} \
    ${MODEL_TYPE} \
    ${IMAGE_PATH} \
    ${INPUT_ONNX} \
    --trt-file ${OUT_TENSORRT} \
    --max-shape INT INT INT INT \
    --min-shape INT INT INT INT \
    --workspace-size INT \
    --fp16 \
    --show \
    --verify \
    --verbose
```

参数说明:

- `config` : 模型配置文件的路径。
- `model_type` : 配置文件的模型类型，选项: `inpainting`, `mattor`, `restorer`, `synthesizer`。
- `img_path` : 输入图像文件的路径。

- onnx_file：输入 ONNX 文件的路径。
- --trt-file：输出 TensorRT 模型的路径。默认为 tmp.trt。
- --max-shape：模型输入的最大形状。
- --min-shape：模型输入的最小形状。
- --workspace-size: 以 GiB 为单位的最大工作空间大小。默认为 1 GiB。
- --fp16: 确定是否以 fp16 模式导出 TensorRT。默认为 False。
- --show: 确定是否显示 ONNX 和 TensorRT 的输出。默认为 False。
- --verify: 确定是否验证导出模型的正确性。默认为 False。
- --verbose: 确定在创建 TensorRT 引擎时是否详细记录日志消息。默认为 False。

注：此工具仍处于试验阶段。目前不支持某些自定义运算符。我们现在只支持 restorer。在生成 SRCNN 的 ONNX 文件时，将 SCRNN 模型中的‘bicubic’替换为‘bilinear’[此处](https://github.com/open-mmlab/mmagic/blob/764e6065e315b7d0033762038fcbf0bb1c570d4d/mmagic.bones/modelsrnn py#L40)。因为 TensorRT 目前不支持 bicubic 插值，最终性能将下降约 4%。

12.4.1 支持导出到 TensorRT 的模型列表

下表列出了保证可导出到 TensorRT 引擎并可在 TensorRT 中运行的模型。

注：

- *以上所有模型均使用 Pytorch==1.8.1、onnxruntime==1.7.0 和 tensorrt==7.2.3.4 进行测试*
- 如果您遇到上面列出的模型的任何问题，请创建一个问题，我们会尽快处理。对于列表中未包含的型号，请尝试自行解决。
- 由于此功能是实验性的并且可能会快速更改，因此请始终尝试使用最新的 mmcv 和 mmagic。

12.5 评估 ONNX 和 TensorRT 模型（实验性）

我们在 tools/deploy_test.py 中提供了评估 TensorRT 和 ONNX 模型的方法。

12.5.1 先决条件

要评估 ONNX 和 TensorRT 模型，应先安装 onnx、onnxruntime 和 TensorRT。遵循 mmcv 中的 ONNXRuntime 和 [mmcv 中的 TensorRT 插件](https://github.com/open-mmlab/mmcv/blob/master/docs/tensorrt_plugin.md%EF%BC%89%E4%BD%BF%E7%94%A8 ONNXRuntime 自定义操作和 TensorRT 插件安装 mmcv-full。

12.5.2 用法

```
python tools/deploy_test.py \
    ${CONFIG_FILE} \
    ${MODEL_PATH} \
    ${BACKEND} \
    --out ${OUTPUT_FILE} \
    --save-path ${SAVE_PATH} \
    ----cfg-options ${CFG_OPTIONS} \
```

12.5.3 参数说明：

- `config`: 模型配置文件的路径。
- `model`: TensorRT 或 ONNX 模型文件的路径。
- `backend`: 用于测试的后端，选择 tensorrt 或 onnxruntime。
- `--out`: pickle 格式的输出结果文件的路径。
- `--save-path`: 存储图像的路径，如果没有给出，则不会保存图像。
- `--cfg-options`: 覆盖使用的配置文件中的一些设置，`xxx=yyy` 格式的键值对将被合并到配置文件中。

12.5.4 结果和模型

注：

- 所有 ONNX 和 TensorRT 模型都使用数据集上的动态形状进行评估，图像根据原始配置文件进行预处理。
- 此工具仍处于试验阶段，我们目前仅支持 `restorer`。

CHAPTER 13

教程 8：模型部署指南

MMDeploy 是 OpenMMLab 的部署仓库，负责包括 MMClassification、MMDetection、MMagic 等在内的各算法库的部署工作。你可以从这里获取 MMDeploy 对 MMClassification 部署支持的最新文档。

本文的结构如下：

- 安装
- 模型转换
- 模型规范
- 模型推理
  - 后端模型推理
  - *SDK* 模型推理
- 模型支持列表

13.1 安装

请参考此处安装 mmagic。然后，按照说明安装 mmdeploy。

备注： 如果安装的是 mmdeploy 预编译包，那么也请通过‘git clone https://github.com/open-mmlab/mmdeploy.git –depth=1’下载 mmdeploy 源码。因为它包含了部署时要用到的配置文件

13.2 模型转换

假设在安装步骤中，mmagic 和 mmdeploy 代码库在同级目录下，并且当前的工作目录为 mmagic 的根目录，那么以 ESRGAN 模型为例，你可以从此处下载对应的 checkpoint，并使用以下代码将之转换为 onnx 模型：

```
from mmdeploy.apis import torch2onnx
from mmdeploy.backend.sdk.export_info import export2SDK

img = 'tests/data/image/face/000001.png'
work_dir = 'mmdeploy_models/mmagic/onnx'
save_file = 'end2end.onnx'
deploy_cfg = '../mmdeploy/configs/mmagic/super-resolution/super-resolution_
↪onnxruntime_dynamic.py'
model_cfg = 'configs/esrgan/esrgan_psnr-x4c64b23g32_1xb16-1000k_div2k.py'
model_checkpoint = 'esrgan_psnr_x4c64b23g32_1x16_1000k_div2k_20200420-bf5c993c.pth'
device = 'cpu'

# 1. convert model to onnx
torch2onnx(img, work_dir, save_file, deploy_cfg, model_cfg,
  model_checkpoint, device)

# 2. extract pipeline info for inference by MMDeploy SDK
export2SDK(deploy_cfg, model_cfg, work_dir, pth=model_checkpoint, device=device)
```

转换的关键之一是使用正确的配置文件。项目中已内置了各后端部署配置文件。文件的命名模式是：

```
{task}/{task}_{backend}-{precision}_{static | dynamic}_{shape}.py
```

其中：

- **{task}:** mmagic 中的任务
- **{backend}:** 推理后端名称。比如，onnxruntime、tensorrt、pplnn、ncnn、openvino、coreml 等等
- **{precision}:** 推理精度。比如，fp16、int8。不填表示 fp32
- **{static | dynamic}:** 动态、静态 shape
- **{shape}:** 模型输入的 shape 或者 shape 范围

在上例中，你也可以把 ESRGAN 转为其他后端模型。比如使用 `super-resolution_tensorrt-fp16_dynamic-32x32-5` `py`，把模型转为 tensorrt-fp16 模型。

小技巧: 当转 tensorrt 模型时，–device 需要被设置为 “cuda”

13.3 模型规范

在使用转换后的模型进行推理之前，有必要了解转换结果的结构。它存放在 `--work-dir` 指定的路路径下。

上例中的 `mmdeploy_models/mmagic/onnx`，结构如下：

```
mmdeploy_models/mmagic/onnx
├── deploy.json
├── detail.json
├── end2end.onnx
└── pipeline.json
```

重要的是：

- **end2end.onnx**: 推理引擎文件。可用 ONNX Runtime 推理
- ***xxx*.json**: mmdeploy SDK 推理所需的 meta 信息

整个文件夹被定义为 **mmdeploy SDK model**。换言之，**mmdeploy SDK model** 既包括推理引擎，也包括推理 meta 信息。

13.4 模型推理

13.4.1 后端模型推理

以上述模型转换后的 `end2end.onnx` 为例，你可以使用如下代码进行推理：

```
from mmdeploy.apis.utils import build_task_processor
from mmdeploy.utils import get_input_shape, load_config
import torch

deploy_cfg = '../mmdeploy/configs/mmagic/super-resolution/super-resolution_
↪onnxruntime_dynamic.py'
model_cfg = 'configs/esrgan/esrgan_psnr-x4c64b23g32_1xb16-1000k_div2k.py'
device = 'cpu'
backend_model = ['mmdeploy_models/mmagic/onnx/end2end.onnx']
image = 'tests/data/image/lq/baboon_x4.png'

# read deploy_cfg and model_cfg
deploy_cfg, model_cfg = load_config(deploy_cfg, model_cfg)

# build task and backend model
task_processor = build_task_processor(model_cfg, deploy_cfg, device)
model = task_processor.build_backend_model(backend_model)
```

(下页继续)

(续上页)

```
# process input image
input_shape = get_input_shape(deploy_cfg)
model_inputs, _ = task_processor.create_input(image, input_shape)

# do model inference
with torch.no_grad():
    result = model.test_step(model_inputs)

# visualize results
task_processor.visualize(
    image=image,
    model=model,
    result=result[0],
    window_name='visualize',
    output_file='output_restorer.bmp')
```

13.4.2 SDK 模型推理

你也可以参考如下代码，对 SDK model 进行推理：

```
from mmdeploy_python import Restorer
import cv2

img = cv2.imread('tests/data/image/lq/baboon_x4.png')

# create a predictor
restorer = Restorer(model_path='mmdeploy_models/mmagic/onnx', device_name='cpu',␣
↪device_id=0)
# perform inference
result = restorer(img)

# visualize inference result
cv2.imwrite('output_restorer.bmp', result)
```

除了 python API，mmdeploy SDK 还提供了诸如 C、C++、C#、Java 等多语言接口。你可以参考样例学习其他语言接口的使用方法。

13.5 模型支持列表

请参考这里

CHAPTER 14

评估器

14.1 评测指标与评测器

在模型的验证和测试中，通常需要对模型的精度进行定量的评测。在 mmagic 中实现了评测指标 (metric) 和评测器 (evaluator) 来完成这一功能。

- 评测指标 (metric) 用于根据测试数据和模型预测结果，特定模型精度指标的计算。在 mmagic 中内置了多种 metric，详见评价指标。同时 metric 和数据集解耦，每种 metric 可以用于多个数据集。
- 评测器 (evaluator) 是评测指标的上层模块，通常需要包含一个或者多个指标。评测器的作用是在模型评测时完成必要的数据格式转换，并调用评测指标来计算模型精度。评测器通常由执行器或测试脚本构建，分别用于在线评测和离线评测。

mmagic 中的评测器继承自 mmengine 中的评测器，基本使用方法也与 mmengine 中的评测器类似，具体可以参见模型精度评测。但不同于其他上层视觉任务，生成模型的评估指标往往具有多种输入。例如 Inception Score（IS）指标的输入仅为虚假图片和任意数量的真实图片；Perceptual path length（PPL) 则需要从隐空间中进行采样。为了对不同的评测指标进行兼容，mmagic 设计了两个重要的方法 prepare_metrics 和 prepare_samplers 来实现上述要求。

14.2 prepare_metrics

```
class Evaluator(Evaluator):
        ...
    def prepare_metrics(self, module: BaseModel, dataloader: DataLoader):
        """Prepare for metrics before evaluation starts. Some metrics use
        pretrained model to extract feature. Some metrics use pretrained model
        to extract feature and input channel order may vary among those models.
        Therefore, we first parse the output color order from data
        preprocessor and set the color order for each metric. Then we pass the
        dataloader to each metrics to prepare pre-calculated items. (e.g.
        inception feature of the real images). If metric has no pre-calculated
        items, :meth:`metric.prepare` will be ignored. Once the function has
        been called, :attr:`self.is_ready` will be set as `True`. If
        :attr:`self.is_ready` is `True`, this function will directly return to
        avoid duplicate computation.

        Args:
            module (BaseModel): Model to evaluate.
            dataloader (DataLoader): The dataloader for real images.
        """
        if self.metrics is None:
            self.is_ready = True
            return

        if self.is_ready:
            return

        # prepare metrics
        for metric in self.metrics:
            metric.prepare(module, dataloader)
        self.is_ready = True
```

prepare_metrics 方法需要在评测开始之前调用。它被用于在每个评测指标开始评测之前进行预处理，会依次调用 evaluator 的所有评测指标的 prepare 方法来准备该评测指标的需要预先计算好的元素 (例如一些隐藏层的特征)。同时为了避免多次重复调用，在所有评测指标预处理完成之后，evaluator.is_ready 标志位会被设置为 True。

```
class GenMetric(BaseMetric):
        ...
    def prepare(self, module: nn.Module, dataloader: DataLoader) -> None:
        """Prepare for the pre-calculating items of the metric. Defaults to do
        nothing.
```

(下页继续)

(续上页)

```
        Args:
            module (nn.Module): Model to evaluate.
            dataloader (DataLoader): Dataloader for the real images.
        """
        if is_model_wrapper(module):
            module = module.module
        self.data_preprocessor = module.data_preprocessor
```

14.3 prepare_samplers

对于生成模型而言，不同的 metric 需要不同的输入。例如 FID, KID, IS 只需要生成的 fake images，而 PPL 则需要隐空间的向量。因此 mmagic 将不同的评估指标按照输入的类型进行了分组，属于同一个组的一个或者多个评测指标共享一个数据的采样器，每个评测指标的 sampler mode 由该评测指标的 SAMPLER_MODE 属性决定。

```
class GenMetric(BaseMetric):
        ...
    SAMPLER_MODE = 'normal'

class GenerativeMetric(GenMetric):
        ...
    SAMPLER_MODE = 'Generative'
```

而 evaluator 的 prepare_samplers 方法就是根据所有评测指标的 sampler mode 来准备好 data sampler。

```
class Evaluator(Evaluator):
        ...
    def prepare_samplers(self, module: BaseModel, dataloader: DataLoader
                         ) -> List[Tuple[List[BaseMetric], Iterator]]:
        """Prepare for the sampler for metrics whose sampling mode are
        different. For generative models, different metric need image
        generated with different inputs. For example, FID, KID and IS need
        images generated with random noise, and PPL need paired images on the
        specific noise interpolation path. Therefore, we first group metrics
        with respect to their sampler's mode (refers to
        :attr:~`GenMetrics.SAMPLER_MODE`), and build a shared sampler for each
        metric group. To be noted that, the length of the shared sampler
        depends on the metric of the most images required in each group.

        Args:
            module (BaseModel): Model to evaluate. Some metrics (e.g. PPL)
```

(下页继续)

(续上页)

```
                require `module` in their sampler.
            dataloader (DataLoader): The dataloader for real image.

        Returns:
            List[Tuple[List[BaseMetric], Iterator]]: A list of "metrics-shared
                sampler" pair.
        """
        if self.metrics is None:
            return [[[None], []]]

        # grouping metrics based on `SAMPLER_MODE` and `sample_mode`
        metric_mode_dict = defaultdict(list)
        for metric in self.metrics:  # 为每个metric指定sampler group
            metric_md5 = self._cal_metric_hash(metric)
            metric_mode_dict[metric_md5].append(metric)

        metrics_sampler_list = []
        for metrics in metric_mode_dict.values(): #为每个group生成sampler
            first_metric = metrics[0]
            metrics_sampler_list.append([
                metrics,
                first_metric.get_metric_sampler(module, dataloader, metrics)
            ])

        return metrics_sampler_list
```

该方法会首先检查自身是否有需要计算的评测指标：如果没有直接返回，如果有则会遍历所有评测指标，对所有采样指标根据 sampler_mode 和 sample_model 进行分组，具体实现方式为根据 sampler_mode 和 sample_model 计算 hash 码，将具有相同 hash 码的评测指标放入同一个列表里。

```
class Evaluator(Evaluator):
        ...
    @staticmethod
    def _cal_metric_hash(metric: GenMetric):
        """Calculate a unique hash value based on the `SAMPLER_MODE` and
        `sample_model`."""
        sampler_mode = metric.SAMPLER_MODE
        sample_model = metric.sample_model
        metric_dict = {
            'SAMPLER_MODE': sampler_mode,
            'sample_model': sample_model
        }
        if hasattr(metric, 'need_cond_input'):
```

(下页继续)

(续上页)

```
            metric_dict['need_cond_input'] = metric.need_cond_input
        md5 = hashlib.md5(repr(metric_dict).encode('utf-8')).hexdigest()
        return md5
```

最后该方法会为每一个评测指标组生成一个 sampler 采样器，添加到列表返回。

14.4 评测器评测流程

整个评测器的评测流程在方法 mmagic.engine.runner.MultiValLoop.run 和 mmagic.engine.runner.MultiTestLoop.run 中实现。以 mmagic.engine.runner.MultiTestLoop.run 为例：

```
class MultiValLoop(BaseLoop):
        ...
    def run(self):
        ...
        # 1. prepare all metrics and get the total length
        metrics_sampler_lists = []
        meta_info_list = []
        dataset_name_list = []
        for evaluator, dataloader in zip(self.evaluators, self.dataloaders):
            # 1.1 prepare for metrics
            evaluator.prepare_metrics(module, dataloader)
            # 1.2 prepare for metric-sampler pair
            metrics_sampler_list = evaluator.prepare_samplers(
                module, dataloader)
            metrics_sampler_lists.append(metrics_sampler_list)
            # 1.3 update total length
            self._total_length += sum([
                len(metrics_sampler[1])
                for metrics_sampler in metrics_sampler_list
            ])
            # 1.4 save metainfo and dataset's name
            meta_info_list.append(
                getattr(dataloader.dataset, 'metainfo', None))
            dataset_name_list.append(dataloader.dataset.__class__.__name__)
```

runner 首先会通过 evaluator.prepare_metrics 和 evaluator.prepare_samplers 两个方法来进行评测所需要的预处理工作和获取评测所需要的数据采样器；同时更新所有采样器的采样总长度。由于 mmagic 的评测指标和数据集进行了分离，因此一些在评测时所需要的 meta_info 也需要进行保存并传递给评测器。

```
class MultiValLoop(BaseLoop):
        ...
```

(下页继续)

(续上页)

```
    def run(self):
        ...
        # 2. run evaluation
        for idx in range(len(self.evaluators)):
            # 2.1 set self.evaluator for run_iter
            self.evaluator = self.evaluators[idx]
            self.dataloader = self.dataloaders[idx]

            # 2.2 update metainfo for evaluator and visualizer
            meta_info = meta_info_list[idx]
            dataset_name = dataset_name_list[idx]
            if meta_info:
                self.evaluator.dataset_meta = meta_info
                self._runner.visualizer.dataset_meta = meta_info
            else:
                warnings.warn(
                    f'Dataset {dataset_name} has no metainfo. `dataset_meta` '
                    'in evaluator, metric and visualizer will be None.')

            # 2.3 generate images
            metrics_sampler_list = metrics_sampler_lists[idx]
            for metrics, sampler in metrics_sampler_list:
                for data in sampler:
                    self.run_iter(idx_counter, data, metrics)
                    idx_counter += 1

            # 2.4 evaluate metrics and update multi_metric
            metrics = self.evaluator.evaluate()
            if multi_metric and metrics.keys() & multi_metric.keys():
                raise ValueError('Please set different prefix for different'
                                 ' datasets in `val_evaluator`')
            else:
                multi_metric.update(metrics)
        # 3. finish evaluation and call hooks
        self._runner.call_hook('after_val_epoch', metrics=multi_metric)
        self._runner.call_hook('after_val')
```

在完成了评测前的准备之后，runner 会遍历所有 evaluator，依次进行评估，每个 evaluator 需要对应一个 dataloader，完成一个数据集的评测工作。具体在对每个 evaluator 进行评测的过程中，首先需要将评测所需要的 meta_info 传递给评测器，随后遍历该 evaluator 的所有 metrics_sampler，生成评测所需要的图像，最后再完成评测。

CHAPTER 15

Data Structure

MMaigc 的数据结构接口 `DataSample` 继承自 MMEngine 的 `BaseDataElement`.MMEngine 的抽象数据接口实现了基础的增/删/改/查功能，且支持不同设备间的数据迁移，也支持了类字典和张量的操作，充分满足了数据的日常使用需求，这也使得不同算法的数据接口可以得到统一。

特别的，`BaseDataElement` 中存在两种类型的数据:

- `metainfo` 类型，包含数据的元信息以确保数据的完整性，如 `img_shape`, `img_id` 等数据所在图片的一些基本信息，方便可视化等情况下对数据进行恢复和使用。
- `data` 类型，如标注框、框的标签、和实例掩码等。

得益于统一的数据封装，算法库内的 `visualizer`, `evaluator`, `model` 等各个模块间的数据流通都得到了极大的简化。

`DataSample` 中的数据分为以下几个属性:

```
- ``gt_img``: 原始图像
- ``pred_img``: 模型预测图像
- ``ref_img``:参考图像
- ``mask``: 图像修复中的遮挡区域
- ``trimap``: 图像抠图中的三通道图
- ``gt_alpha``: 图像抠图中原始Alpha图
- ``pred_alpha``: 图像抠图中模型预测Alpha图
- ``gt_fg``: 图像抠图中原始前景图
- ``pred_fg``: 图像抠图中模型预测前景图
- ``gt_bg``:  图像抠图中原始背景图
```

(下页继续)

(续上页)

```
- ``pred_bg``: 图像抠图中模型预测背景图
- ``gt_merged``:  图像抠图中原始合并图
```

以下示例代码展示了 DataSample 的组成元素类型:

```
import torch
import numpy as np
from mmagic.structures import DataSample
img_meta = dict(img_shape=(800, 1196, 3))
img = torch.rand((3, 800, 1196))
data_sample = DataSample(gt_img=img, metainfo=img_meta)
assert 'img_shape' in data_sample.metainfo_keys()
data_sample
# `DataSample` 的组成元素类型
<DataSample(

        META INFORMATION
        img_shape: (800, 1196, 3)

        DATA FIELDS
        gt_img: tensor(3, 800, 1196)
    ) at 0x1f6a5a99a00>
```

DataSample 同样支持 stack 和 split 操作对数据进行批处理:

1. Stack

该函数用于将数据样本列表堆叠成一个。当数据样本堆叠时，所有张量字段都将堆叠在第一维度。如果数据样本中有非张量字段，例如列表或字典，则这些字段的值将保存在列表中。

```
    Args:
        data_samples (Sequence['DataSample']): 待堆叠的数据样本序列

    Returns:
        DataSample: 堆叠的数据样本
```

2. Split

该函数将在第一维度拆分数据样本序列。

```
        Args:
         allow_nonseq_value (bool): 是否允许在拆分操作中使用非顺序数据。如果为 "True
↪",                          将为所有拆分数据样本复制非序列数据；否则，将引发错误。默认为␣
↪"False"。
```

(下页继续)

(续上页)

```
    Returns:
        Sequence[DataSample]: 拆分后的数据样本列表。
```

以下示例代码展示了 stack 和 split 的使用方法:

```
import torch
import numpy as np
from mmagic.structures import DataSample
img_meta1 = img_meta2 = dict(img_shape=(800, 1196, 3))
img1 = torch.rand((3, 800, 1196))
img2 = torch.rand((3, 800, 1196))
data_sample1 = DataSample(gt_img=img1, metainfo=img_meta1)
data_sample2 = DataSample(gt_img=img2, metainfo=img_meta1)
```

```
# 堆叠stack
data_sample = DataSample.stack([data_sample1, data_sample2])
print(data_sample.gt_img.shape)
    torch.Size([2, 3, 800, 1196])
print(data_sample.metainfo)
    {'img_shape': [(800, 1196, 3), (800, 1196, 3)]}

# 拆分split
data_sample1_, data_sample2_ = data_sample.split()
assert (data_sample1_.gt_img == img1).all()
assert (data_sample2_.gt_img == img2).all()
```

CHAPTER 16

数据预处理器（待更新）

CHAPTER 17

数据流

- 数据流
  - 数据流概述
  - 数据集与模型之间的数据流
    - 数据加载器的数据处理
    - 数据预处理器的数据处理
  - 模型输出与可视化器之间的数据流

17.1 数据流概述

Runner 相当于 MMEngine 中的“集成器”。它覆盖了框架的所有方面，并肩负着组织和调度几乎所有模块的责任，这意味着各模块之间的数据流也由 `Runner` 控制。在本章节中，我们将介绍 Runner 管理的内部模块之间的数据流和数据格式约定。

在上图中，在训练迭代中，数据加载器（dataloader）从存储中加载图像并传输到数据预处理器（data preprocessor），数据预处理器会将图像放到特定的设备上，并将数据堆叠到批处理中，之后模型接受批处理数据作为输入，最后将模型的输出计算损失函数（loss）。在评估时模型参数会被冻结，模型的输出需要经由数据预处理器（data preprocessor）解构再被传递给 Evaluator 计算指标或者提供给*Visualizer*进行可视化。

17.2 数据集与模型之间的数据流

在本节中将介绍在 MMagic 中数据集中的数据流传递，关于数据集定义和数据处理管线相关的解读详见开发指南。数据集（dataset）和模型（model）之间的数据流传递一般可以分为如下四个步骤:

1. 读取 `XXDataset` 收集数据集的原始信息，并且通过数据处理管线对数据进行数据转换处理;
2. 使用 `PackInputs` 将转换完成的数据打包成为一个字典;
3. 使用 `collate_fn` 将各个张量集成为一个批处理张量;
4. 使用 `data_preprocessor` 把以上所有数据迁移到 GPUS 等目标设备，并在数据加载器中将之前打包的字典解压为一个元组，该元组包含输入图像与对应的元信息（`DataSample`）。

17.2.1 数据处理管线的数据处理

在 MMagic 中，经由不同类型的 `XXDataset`, 分别读取数据 (LQ) 以及标注 (GT)，并且在不同的数据预处理管道中进行数据转换，最后通过 `PackInputs` 将处理之后的数据打包为字典，此字典包含训练以及测试过程所需的所有数据。

```
@MODELS.register_module()
class BaseEditModel(BaseModel):
    """Base model for image and video editing.
    """
    def forward(self,
                inputs: torch.Tensor,
                data_samples: Optional[List[DataSample]] = None,
                mode: str = 'tensor',
                **kwargs) -> Union[torch.Tensor, List[DataSample], dict]:
        if isinstance(inputs, dict):
            inputs = inputs['img']
        if mode == 'tensor':
            return self.forward_tensor(inputs, data_samples, **kwargs)

        elif mode == 'predict':
            predictions = self.forward_inference(inputs, data_samples,
                                                 **kwargs)
            predictions = self.convert_to_datasample(predictions, data_samples,
                                                     inputs)
            return predictions

        elif mode == 'loss':
            return self.forward_train(inputs, data_samples, **kwargs)
```

```
@MODELS.register_module()
class BaseConditionalGAN(BaseGAN):
    """Base class for Conditional GAM models.
    """
    def forward(self,
                inputs: ForwardInputs,
                data_samples: Optional[list] = None,
                mode: Optional[str] = None) -> List[DataSample]:
        if isinstance(inputs, Tensor):
            noise = inputs
            sample_kwargs = {}
        else:
            noise = inputs.get('noise', None)
            num_batches = get_valid_num_batches(inputs, data_samples)
            noise = self.noise_fn(noise, num_batches=num_batches)
            sample_kwargs = inputs.get('sample_kwargs', dict())
        num_batches = noise.shape[0]

        pass
        ...
```

例如在 `BaseEditModel` 和 `BaseConditionalGAN` 模型中分别需要输入（input）包括 `img` 和 `noise` 的键值输入。同时，相应的字段也应该在配置文件中暴露，以 cyclegan_lsgan-id0-resnet-in_1xb1-80kiters_facades.py 为例，

```
domain_a = 'photo'
domain_b = 'mask'
pack_input = dict(
    type='PackInputs',
    keys=[f'img_{domain_a}', f'img_{domain_b}'],
    data_keys=[f'img_{domain_a}', f'img_{domain_b}'])
```

17.2.2 数据加载器的数据处理

以数据集中的获取字典列表作为输入，数据加载器（dataloader）中的 `collect_fn` 会提取每个字典的 `inputs` 并将其整合成一个批处理张量；此外，每个字典中的 `data_sample` 也会被整合为一个列表，从而输出一个与先前字典有相同键的字典；最终数据加载器会通过 `collect_fn` 输出这个字典。详细文档可见数据集与数据加载器。

17.2.3 数据预处理器的数据处理

数据预处理是数据输入模型之前，处理数据过程的最后一步。数据预处理过程会对图像进行归一处理，如把 BGR 模式转换为 RGB 模式，并将所有数据迁移至 GPU 等目标设备中。上述各步骤完成后，最终会得到一个元组，该元组包含一个批处理图像的列表，和一个数据样本的列表。详细文档可见数据预处理。

17.3 模型输出与可视化器之间的数据流

MMEngine 约定了抽象数据接口用于数据传递，其中数据样本(DataSample) 作为一层更加高级封装可以容纳更多类别的标签数据。在 MMagic 中，用于可视化对比的 `ConcatImageVisualizer` 同时也通过 `add_datasample` 方法控制可视化具体内容，具体配置如下。

```
visualizer = dict(
    type='ConcatImageVisualizer',
    vis_backends=[dict(type='LocalVisBackend')],
    fn_key='gt_path',
    img_keys=['gt_img', 'input', 'pred_img'],
    bgr2rgb=True)
```

CHAPTER 18

如何设计自己的模型

MMagic 建立在 MMEngine 和 MMCV 的基础上，使用户能够快速地设计新模型，轻松地地训练和评估它们。在本节中，您将学习如何设计自己的模型。

本指南的结构如下:

- 如何设计自己的模型
  - *MMagic* 中的模型概述
  - 一个 *SRCNN* 的例子
    - *Step 1:* 定义 *SRCNN* 网络
    - *Step 2:* 定义 *SRCNN* 的模型
    - *Step 3:* 开始训练 *SRCNN*
  - 一个 *DCGAN* 的例子
    - *Step 1:* 定义 *DCGAN* 的网络
    - *Step 2:* 设计 *DCGAN* 的模型
    - *Step 3:* 开始训练 *DCGAN*
  - 参考文献

18.1 MMagic 中的模型概述

在 MMagic 中，一个算法可以分为两部分: **Model** 和 **Module**.

- **Model** 是最顶层的包装，并且总是继承自 MMEngine 中提供的 `BaseModel` 。**Model** 负责网络前向、损耗计算、反向、参数更新等. 在 MMagic 中, **Model** 应该注册为 `MODELS`.
- **Module** 模块包括用于训练或推理的 **architectures** ，预定义的 **loss classes**, 以及对批量输入数据预处理的 **data preprocessors** 。**Module** 总是作为 **Model** 的元素呈现。在 MMagic 中, **Module** 应该注册为 **MODULES**。

以 DCGAN model 模型为例，生成器 和 判别器 是 **Module**, 分别用于生成图像和鉴别图像真伪。`DCGAN` 是 **Model**, 它从 dataloader 中获取数据，交替训练生成器和鉴别器。

您可以通过以下链接找到 **Model** 和 **Module** 的实现。

- **Model**:
  - Editors
- **Module**:
  - Layers
  - Losses
  - Data Preprocessor

18.2 一个 SRCNN 的例子

这里，我们以经典图像超分辨率模型 SRCNN[1] 的实现为例。

18.2.1 Step 1: 定义 SRCNN 网络

SRCNN 是第一个用于单幅图像超分辨率 [1] 的深度学习方法。为了实现 SRCNN 的网络架构，我们需要创建一个新文件 `mmagic/models/editors/srgan/sr_resnet.py` 并执行 `class MSRResNet`。

在这一步中，我们通过继承 `mmengine.models.BaseModule` 来实现 `class MSRResNet`，并在 `__init__` 函数中定义网络架构。特别地，我们需要使用 `@MODELS.register_module()` 将 `class MSRResNet` 的实现添加到 MMagic 的注册中。

```
import torch.nn as nn
from mmengine.model import BaseModule
from mmagic.registry import MODELS

from mmagic.models.utils import (PixelShufflePack, ResidualBlockNoBN,
                                 default_init_weights, make_layer)
```

(下页继续)

(续上页)

```
@MODELS.register_module()
class MSRResNet(BaseModule):
    """修改后的SRResNet。

    由 "使用生成对抗网络的照片-现实的单幅图像超级分辨率
↪"中的SRResNet修改而来的压缩版本。

    它使用无BN的残差块，类似于EDSR。
    目前支持x2、x3和x4上采样比例因子。

    Args:
        in_channels (int): Channel number of inputs.
        out_channels (int): Channel number of outputs.
        mid_channels (int): Channel number of intermediate features.
            Default: 64.
        num_blocks (int): Block number in the trunk network. Default: 16.
        upscale_factor (int): Upsampling factor. Support x2, x3 and x4.
            Default: 4.
    """
    _supported_upscale_factors = [2, 3, 4]

    def __init__(self,
                 in_channels,
                 out_channels,
                 mid_channels=64,
                 num_blocks=16,
                 upscale_factor=4):

        super().__init__()
        self.in_channels = in_channels
        self.out_channels = out_channels
        self.mid_channels = mid_channels
        self.num_blocks = num_blocks
        self.upscale_factor = upscale_factor

        self.conv_first = nn.Conv2d(
            in_channels, mid_channels, 3, 1, 1, bias=True)
        self.trunk_net = make_layer(
            ResidualBlockNoBN, num_blocks, mid_channels=mid_channels)

        # upsampling
```

(下页继续)

(续上页)

```
        if self.upscale_factor in [2, 3]:
            self.upsample1 = PixelShufflePack(
                mid_channels,
                mid_channels,
                self.upscale_factor,
                upsample_kernel=3)
        elif self.upscale_factor == 4:
            self.upsample1 = PixelShufflePack(
                mid_channels, mid_channels, 2, upsample_kernel=3)
            self.upsample2 = PixelShufflePack(
                mid_channels, mid_channels, 2, upsample_kernel=3)
        else:
            raise ValueError(
                f'Unsupported scale factor {self.upscale_factor}. '
                f'Currently supported ones are '
                f'{self._supported_upscale_factors}.')

        self.conv_hr = nn.Conv2d(
            mid_channels, mid_channels, 3, 1, 1, bias=True)
        self.conv_last = nn.Conv2d(
            mid_channels, out_channels, 3, 1, 1, bias=True)

        self.img_upsampler = nn.Upsample(
            scale_factor=self.upscale_factor,
            mode='bilinear',
            align_corners=False)

        # activation function
        self.lrelu = nn.LeakyReLU(negative_slope=0.1, inplace=True)

        self.init_weights()

    def init_weights(self):
        """Init weights for models.

        Args:
            pretrained (str, optional): Path for pretrained weights. If given
                None, pretrained weights will not be loaded. Defaults to None.
            strict (boo, optional): Whether strictly load the pretrained model.
                Defaults to True.
        """

        for m in [self.conv_first, self.conv_hr, self.conv_last]:
```

(下页继续)

(续上页)

```
            default_init_weights(m, 0.1)
```

然后，我们实现了 class MSRResNet 的 forward 函数，该函数将输入张量作为输入张量，然后返回 MSRResNet 的结果。

```
    def forward(self, x):
        """Forward function.

        Args:
            x (Tensor): Input tensor with shape (n, c, h, w).

        Returns:
            Tensor: Forward results.
        """

        feat = self.lrelu(self.conv_first(x))
        out = self.trunk_net(feat)

        if self.upscale_factor in [2, 3]:
            out = self.upsample1(out)
        elif self.upscale_factor == 4:
            out = self.upsample1(out)
            out = self.upsample2(out)

        out = self.conv_last(self.lrelu(self.conv_hr(out)))
        upsampled_img = self.img_upsampler(x)
        out += upsampled_img
        return out
```

在 class MSRResNet 实现后，我们需要更新 mmagic/models/editors/__init__.py 中的模型列表，以便我们可以通过 mmagic.models.editors 导入和使用 class MSRResNet。

```
from .srgan.sr_resnet import MSRResNet
```

18.2.2 Step 2: 定义 SRCNN 的模型

网络架构实现后，我们需要定义我们的模型 class BaseEditModel 并实现 class BaseEditModel 的前向循环。

为了实现 class BaseEditModel，我们创建一个新文件 mmagic/models/base_models/base_edit_model.py。具体来说，class BaseEditModel 继承自 mmengine.model.BaseModel. 在 __init__ 函数中，我们定义了 class BaseEditModel 的损失函数，训练，测试配置和网络。

```python
from typing import List, Optional

import torch
from mmengine.model import BaseModel

from mmagic.registry import MODELS
from mmagic.structures import DataSample


@MODELS.register_module()
class BaseEditModel(BaseModel):
    """用于图像和视频编辑的基本模型。

    它必须包含一个生成器，将帧作为输入并输出插值帧。它也有一个用于训练的pixel-
↪wise损失。

    Args:
        generator (dict): Config for the generator structure.
        pixel_loss (dict): Config for pixel-wise loss.
        train_cfg (dict): Config for training. Default: None.
        test_cfg (dict): Config for testing. Default: None.
        init_cfg (dict, optional): The weight initialized config for
            :class:`BaseModule`.
        data_preprocessor (dict, optional): The pre-process config of
            :class:`BaseDataPreprocessor`.

    Attributes:
        init_cfg (dict, optional): Initialization config dict.
        data_preprocessor (:obj:`BaseDataPreprocessor`): Used for
            pre-processing data sampled by dataloader to the format accepted by
            :meth:`forward`. Default: None.
    """

    def __init__(self,
                 generator,
                 pixel_loss,
                 train_cfg=None,
                 test_cfg=None,
                 init_cfg=None,
                 data_preprocessor=None):
        super().__init__(
            init_cfg=init_cfg, data_preprocessor=data_preprocessor)

        self.train_cfg = train_cfg
```

(下页继续)

(续上页)

```
        self.test_cfg = test_cfg

        # generator
        self.generator = MODELS.build(generator)

        # loss
        self.pixel_loss = MODELS.build(pixel_loss)
```

因为 mmengine.model.BaseModel 提供了算法模型的基本功能，例如权重初始化、批量输入预处理、解析损失和更新模型参数。因此，子类继承自 BaseModel，即本例中的 class BaseEditModel, 只需要实现 forward 方法，该方法实现了计算损失和预测的逻辑。

具体来说，class BaseEditModel 实现的 forward 函数将 batch_inputs 和 data_samples 作为输入，并根据模式参数返回结果。

```
    def forward(self,
                batch_inputs: torch.Tensor,
                data_samples: Optional[List[DataSample]] = None,
                mode: str = 'tensor',
                **kwargs):
        """返回训练、验证、测试和简单推理过程的损失或预测。

        BaseModel的``forward``方法是一个抽象方法，它的子类必须实现这个方法。

        接受由:attr:`data_preprocessor`处理的``batch_inputs`` 和 ``data_samples``,␣
↪并根据模式参数返回结果。.

        在非分布式训练、验证和测试过程中，
        ``forward``将被``BaseModel.train_step``,
        ``BaseModel.val_step``和``BaseModel.val_step``直接调用。

        在分布式数据并行训练过程中,``MMSeparateDistributedDataParallel.train_
↪step``将首先调用``DistributedDataParallel.
↪forward``以启用自动梯度同步，然后调用``forward``获得训练损失。

        Args:
            batch_inputs (torch.Tensor): batch input tensor collated by
                :attr:`data_preprocessor`.
            data_samples (List[BaseDataElement], optional):
                data samples collated by :attr:`data_preprocessor`.
            mode (str): mode should be one of ``loss``, ``predict`` and
                ``tensor``

                - ``loss``: Called by ``train_step`` and return loss ``dict``
```

(下页继续)

(续上页)

```
                used for logging
            - ``predict``: Called by ``val_step`` and ``test_step``
                and return list of ``BaseDataElement`` results used for
                computing metric.
            - ``tensor``: Called by custom use to get ``Tensor`` type
                results.

        Returns:
            ForwardResults:

            - If ``mode == loss``, return a ``dict`` of loss tensor used
                for backward and logging.
            - If ``mode == predict``, return a ``list`` of
                :obj:`BaseDataElement` for computing metric
                and getting inference result.
            - If ``mode == tensor``, return a tensor or ``tuple`` of tensor
                or ``dict or tensor for custom use.
        """

        if mode == 'tensor':
            return self.forward_tensor(batch_inputs, data_samples, **kwargs)

        elif mode == 'predict':
            return self.forward_inference(batch_inputs, data_samples, **kwargs)

        elif mode == 'loss':
            return self.forward_train(batch_inputs, data_samples, **kwargs)
```

具体来说，在 forward_tensor 中,class BaseEditModel 直接返回网络的前向张量。

```
    def forward_tensor(self, batch_inputs, data_samples=None, **kwargs):
        """Forward tensor.
            Returns result of simple forward.

        Args:
            batch_inputs (torch.Tensor): batch input tensor collated by
                :attr:`data_preprocessor`.
            data_samples (List[BaseDataElement], optional):
                data samples collated by :attr:`data_preprocessor`.

        Returns:
            Tensor: result of simple forward.
        """
```

(下页继续)

(续上页)

```
        feats = self.generator(batch_inputs, **kwargs)

        return feats
```

在 forward_inference 函数中，class BaseEditModel 首先将前向张量转换为图像，然后返回该图像作为输出。

```
    def forward_inference(self, batch_inputs, data_samples=None, **kwargs):
        """Forward inference.
            Returns predictions of validation, testing, and simple inference.

        Args:
            batch_inputs (torch.Tensor): batch input tensor collated by
                :attr:`data_preprocessor`.
            data_samples (List[BaseDataElement], optional):
                data samples collated by :attr:`data_preprocessor`.

        Returns:
            List[DataSample]: predictions.
        """

        feats = self.forward_tensor(batch_inputs, data_samples, **kwargs)
        feats = self.data_preprocessor.destructor(feats)
        predictions = []
        for idx in range(feats.shape[0]):
            predictions.append(
                DataSample(
                    pred_img=feats[idx].to('cpu'),
                    metainfo=data_samples[idx].metainfo))

        return predictions
```

在 forward_train 中,class BaseEditModel 计算损失函数，并返回一个包含损失的字典作为输出。

```
    def forward_train(self, batch_inputs, data_samples=None, **kwargs):
        """Forward training.
            Returns dict of losses of training.

        Args:
            batch_inputs (torch.Tensor): batch input tensor collated by
                :attr:`data_preprocessor`.
            data_samples (List[BaseDataElement], optional):
                data samples collated by :attr:`data_preprocessor`.
```

(下页继续)

(续上页)

```
        Returns:
            dict: Dict of losses.
        """

        feats = self.forward_tensor(batch_inputs, data_samples, **kwargs)
        gt_imgs = [data_sample.gt_img.data for data_sample in data_samples]
        batch_gt_data = torch.stack(gt_imgs)

        loss = self.pixel_loss(feats, batch_gt_data)

        return dict(loss=loss)
```

在实现了 class BaseEditModel 之后，我们需要更新 mmagic/models/__init__.py 中的模型列表，这样我们就可以通过 mmagic.models 导入和使用 class BaseEditModel。

```
from .base_models.base_edit_model import BaseEditModel
```

18.2.3 Step 3: 开始训练 SRCNN

在实现了网络结构和 SRCNN 的前向循环后、现在我们可以创建一个新的文件 configs/srcnn/srcnn_x4k915_g1_1000k_div2k.py 来设置训练 SRCNN 所需的配置。

在配置文件中，我们需要指定我们的模型 class BaseEditModel 的参数，包括生成器网络结构、损失函数、额外的训练和测试配置，以及输入张量的数据预处理器。请参考*MMagic 中的损失函数介绍*了解 MMagic 中损失函数的更多细节。

```
# model settings
model = dict(
    type='BaseEditModel',
    generator=dict(
        type='SRCNNNet',
        channels=(3, 64, 32, 3),
        kernel_sizes=(9, 1, 5),
        upscale_factor=scale),
    pixel_loss=dict(type='L1Loss', loss_weight=1.0, reduction='mean'),
    data_preprocessor=dict(
        type='DataPreprocessor',
        mean=[0., 0., 0.],
        std=[255., 255., 255.],
    ))
```

我们还需要根据创建自己的数据加载器来指定训练数据加载器和测试数据加载器。最后，我们可以开始训练

我们自己的模型:

```
python tools/train.py configs/srcnn/srcnn_x4k915_g1_1000k_div2k.py
```

18.3 一个 DCGAN 的例子

这里，我们以经典 gan 模型 DCGAN[2] 的实现为例。

18.3.1 Step 1: 定义 DCGAN 的网络

DCGAN 是一种经典的图像生成对抗网络 [2]。为了实现 DCGAN 的网络架构，我们需要创建两个新文件 `mmagic/models/editors/dcgan/dcgan_generator.py` 和 `mmagic/models/editors/dcgan/dcgan_discriminator.py`，并实现生成器 (`class DCGANGenerator`) 和鉴别器 (`class DCGANDiscriminator`)。

在这一步中，我们实现了 `class DCGANGenerator`,`class DCGANDiscriminator` 并在 `__init__` 函数中定义了网络架构。特别地，我们需要使用 `@MODULES.register_module()` 来将生成器和鉴别器添加到 MMagic 的注册中。

以下面的代码为例:

```
import torch.nn as nn
from mmcv.cnn import ConvModule
from mmcv.runner import load_checkpoint
from mmcv.utils.parrots_wrapper import _BatchNorm
from mmengine.logging import MMLogger
from mmengine.model.utils import normal_init

from mmagic.models.builder import MODULES
from ..common import get_module_device


@MODULES.register_module()
class DCGANGenerator(nn.Module):
    def __init__(self,
                 output_scale,
                 out_channels=3,
                 base_channels=1024,
                 input_scale=4,
                 noise_size=100,
                 default_norm_cfg=dict(type='BN'),
                 default_act_cfg=dict(type='ReLU'),
                 out_act_cfg=dict(type='Tanh'),
```

(下页继续)

(续上页)

```
                 pretrained=None):
        super().__init__()
        self.output_scale = output_scale
        self.base_channels = base_channels
        self.input_scale = input_scale
        self.noise_size = noise_size

        # 上采样的次数
        self.num_upsamples = int(np.log2(output_scale // input_scale))

        # 输出4x4的特征图
        self.noise2feat = ConvModule(
            noise_size,
            base_channels,
            kernel_size=4,
            stride=1,
            padding=0,
            conv_cfg=dict(type='ConvTranspose2d'),
            norm_cfg=default_norm_cfg,
            act_cfg=default_act_cfg)

        # 建立上采样骨干（不包括输出层）
        upsampling = []
        curr_channel = base_channels
        for _ in range(self.num_upsamples - 1):
            upsampling.append(
                ConvModule(
                    curr_channel,
                    curr_channel // 2,
                    kernel_size=4,
                    stride=2,
                    padding=1,
                    conv_cfg=dict(type='ConvTranspose2d'),
                    norm_cfg=default_norm_cfg,
                    act_cfg=default_act_cfg))

            curr_channel //= 2

        self.upsampling = nn.Sequential(*upsampling)

        # 输出层
        self.output_layer = ConvModule(
            curr_channel,
```

(下页继续)

(续上页)

```
            out_channels,
            kernel_size=4,
            stride=2,
            padding=1,
            conv_cfg=dict(type='ConvTranspose2d'),
            norm_cfg=None,
            act_cfg=out_act_cfg)
```

然后，我们实现了 DCGANGenerator 的 forward 函数，该函数接受 noise 张量或 num_batches，然后返回 DCGANGenerator 的结果。

```
    def forward(self, noise, num_batches=0, return_noise=False):
        noise_batch = noise_batch.to(get_module_device(self))
        x = self.noise2feat(noise_batch)
        x = self.upsampling(x)
        x = self.output_layer(x)
        return x
```

如果你想为你的网络实现特定的权重初始化方法，你需要自己添加 init_weights 函数。

```
    def init_weights(self, pretrained=None):
        if isinstance(pretrained, str):
            logger = MMLogger.get_current_instance()
            load_checkpoint(self, pretrained, strict=False, logger=logger)
        elif pretrained is None:
            for m in self.modules():
                if isinstance(m, (nn.Conv2d, nn.ConvTranspose2d)):
                    normal_init(m, 0, 0.02)
                elif isinstance(m, _BatchNorm):
                    nn.init.normal_(m.weight.data)
                    nn.init.constant_(m.bias.data, 0)
        else:
            raise TypeError('pretrained must be a str or None but'
                            f' got {type(pretrained)} instead.')
```

在实现 DCGANGenerator 类之后，我们需要更新 mmagic/models/editors/__init__.py 中的模型列表，以便我们可以通过 mmagic.models.editors 导入和使用 DCGANGenerator 类。

类 DCGANDiscriminator 的实现遵循类似的逻辑，你可以在这里找到实现。

18.3.2 Step 2: 设计 DCGAN 的模型

在实现了网络 **Module** 之后，我们需要定义我们的 **Model** 类 DCGAN。

你的 **Model** 应该继承自 MMEngine 提供的BaseModel，并实现三个函数，train_step, val_step 和 test_step。

- train_step: 这个函数负责更新网络的参数，由 MMEngine 的 Loop (IterBasedTrainLoop或 EpochBasedTrainLoop) 调用。train_step 将数据批处理和OptimWrapper作为输入并返回一个日志字典。
- val_step: 该函数负责在训练过程中获取用于验证的输出，由 MultiValLoop调用。
- test_step: 该函数负责在测试过程中获取输出，由MultiTestLoop调用。

  请注意，在 train_step, val_step 和 test_step 中，调用 DataPreprocessor 对输入数据进行预处理，然后再将它们提供给神经网络。要了解有关 DataPreprocessor 的更多信息，请参阅此文件 and 和本教程。

为了简化使用，我们在 MMagic 中提供了BaseGAN类，它为 GAN 模型实现了通用的 train_step, val_step 和 test_step 函数。使用 BaseGAN 作为基类，每个特定的 GAN 算法只需要实现 train_generator and train_discriminator.

在 train_step 中，我们支持数据预处理、梯度累积 (由OptimWrapper实现) 和指数滑动平均 (EMA) 通过(ExponentialMovingAverage)实现。使用 BaseGAN.train_step，每个特定的 GAN 算法只需要实现 train_generator 和 train_discriminator。

```
    def train_step(self, data: dict,
                   optim_wrapper: OptimWrapperDict) -> Dict[str, Tensor]:
        message_hub = MessageHub.get_current_instance()
        curr_iter = message_hub.get_info('iter')
        data = self.data_preprocessor(data, True)
        disc_optimizer_wrapper: OptimWrapper = optim_wrapper['discriminator']
        disc_accu_iters = disc_optimizer_wrapper._accumulative_counts

        # 训练判别器，使用MMEngine提供的上下文管理器
        with disc_optimizer_wrapper.optim_context(self.discriminator):
            # train_discriminator should be implemented!
            log_vars = self.train_discriminator(
                **data, optimizer_wrapper=disc_optimizer_wrapper)

        # add 1 to `curr_iter` because iter is updated in train loop.
        # Whether to update the generator. We update generator with
        # discriminator is fully updated for `self.n_discriminator_steps`
        # iterations. And one full updating for discriminator contains
        # `disc_accu_counts` times of grad accumulations.
        if (curr_iter + 1) % (self.discriminator_steps * disc_accu_iters) == 0:
```

(下页继续)

(续上页)

```
            set_requires_grad(self.discriminator, False)
            gen_optimizer_wrapper = optim_wrapper['generator']
            gen_accu_iters = gen_optimizer_wrapper._accumulative_counts

            log_vars_gen_list = []
            # init optimizer wrapper status for generator manually
            gen_optimizer_wrapper.initialize_count_status(
                self.generator, 0, self.generator_steps * gen_accu_iters)
            # update generator, use context manager provided by MMEngine
            for _ in range(self.generator_steps * gen_accu_iters):
                with gen_optimizer_wrapper.optim_context(self.generator):
                    # train_generator should be implemented!
                    log_vars_gen = self.train_generator(
                        **data, optimizer_wrapper=gen_optimizer_wrapper)

                log_vars_gen_list.append(log_vars_gen)
            log_vars_gen = gather_log_vars(log_vars_gen_list)
            log_vars_gen.pop('loss', None)  # remove 'loss' from gen logs

            set_requires_grad(self.discriminator, True)

            # only do ema after generator update
            if self.with_ema_gen and (curr_iter + 1) >= (
                    self.ema_start * self.discriminator_steps *
                    disc_accu_iters):
                self.generator_ema.update_parameters(
                    self.generator.module
                    if is_model_wrapper(self.generator) else self.generator)

            log_vars.update(log_vars_gen)

        # return the log dict
        return log_vars
```

在 val_step 和 test_step，我们渐进地调用数据预处理和 BaseGAN.forward。

```
    def val_step(self, data: dict) -> SampleList:
        data = self.data_preprocessor(data)
        # call `forward`
        outputs = self(**data)
        return outputs

    def test_step(self, data: dict) -> SampleList:
        data = self.data_preprocessor(data)
```

(下页继续)

(续上页)

```
        # call `orward`
        outputs = self(**data)
        return outputs
```

然后，我们在 DCGAN 类中实现 train_generator 和 train_discriminator。

```
from typing import Dict, Tuple

import torch
import torch.nn.functional as F
from mmengine.optim import OptimWrapper
from torch import Tensor

from mmagic.registry import MODELS
from .base_gan import BaseGAN


@MODELS.register_module()
class DCGAN(BaseGAN):
    def disc_loss(self, disc_pred_fake: Tensor,
                  disc_pred_real: Tensor) -> Tuple:
        losses_dict = dict()
        losses_dict['loss_disc_fake'] = F.binary_cross_entropy_with_logits(
            disc_pred_fake, 0. * torch.ones_like(disc_pred_fake))
        losses_dict['loss_disc_real'] = F.binary_cross_entropy_with_logits(
            disc_pred_real, 1. * torch.ones_like(disc_pred_real))

        loss, log_var = self.parse_losses(losses_dict)
        return loss, log_var

    def gen_loss(self, disc_pred_fake: Tensor) -> Tuple:
        losses_dict = dict()
        losses_dict['loss_gen'] = F.binary_cross_entropy_with_logits(
            disc_pred_fake, 1. * torch.ones_like(disc_pred_fake))
        loss, log_var = self.parse_losses(losses_dict)
        return loss, log_var

    def train_discriminator(
            self, inputs, data_sample,
            optimizer_wrapper: OptimWrapper) -> Dict[str, Tensor]:
        real_imgs = inputs['img']

        num_batches = real_imgs.shape[0]
```

(下页继续)

(续上页)

```
        noise_batch = self.noise_fn(num_batches=num_batches)
        with torch.no_grad():
            fake_imgs = self.generator(noise=noise_batch, return_noise=False)

        disc_pred_fake = self.discriminator(fake_imgs)
        disc_pred_real = self.discriminator(real_imgs)

        parsed_losses, log_vars = self.disc_loss(disc_pred_fake,
                                                 disc_pred_real)
        optimizer_wrapper.update_params(parsed_losses)
        return log_vars

    def train_generator(self, inputs, data_sample,
                        optimizer_wrapper: OptimWrapper) -> Dict[str, Tensor]:
        num_batches = inputs['img'].shape[0]

        noise = self.noise_fn(num_batches=num_batches)
        fake_imgs = self.generator(noise=noise, return_noise=False)

        disc_pred_fake = self.discriminator(fake_imgs)
        parsed_loss, log_vars = self.gen_loss(disc_pred_fake)

        optimizer_wrapper.update_params(parsed_loss)
        return log_vars
```

在实现了 `class DCGAN` 之后，我们需要更新 `mmagic/models/__init__.py` 中的模型列表，以便我们可以通过 `mmagic.models` 导入和使用 `class DCGAN`。

18.3.3 Step 3: 开始训练 DCGAN

在实现了网络 **Module** 和 DCGAN 的 **Model** 之后，现在我们可以创建一个新文件 `configs/dcgan/dcgan_1xb128-5epoches_lsun-bedroom-64x64.py` 来设置训练 DCGAN 所需的配置。

在配置文件中，我们需要指定模型的参数，`class DCGAN`，包括生成器网络架构和输入张量的数据预处理器。

```
# model settings
model = dict(
    type='DCGAN',
    noise_size=100,
    data_preprocessor=dict(type='GANDataPreprocessor'),
    generator=dict(type='DCGANGenerator', output_scale=64, base_channels=1024),
    discriminator=dict(
```

(下页继续)

(续上页)

```
        type='DCGANDiscriminator',
        input_scale=64,
        output_scale=4,
        out_channels=1))
```

我们还需要根据创建自己的数据加载器指定训练数据加载器和测试数据加载器。最后，我们可以开始训练我们自己的模型:

```
python tools/train.py configs/dcgan/dcgan_1xb128-5epoches_lsun-bedroom-64x64.py
```

18.4 参考文献

1. Dong, Chao and Loy, Chen Change and He, Kaiming and Tang, Xiaoou. Image Super-Resolution Using Deep Convolutional Networks[J]. IEEE transactions on pattern analysis and machine intelligence, 2015.
2. Radford, Alec, Luke Metz, and Soumith Chintala. "Unsupervised representation learning with deep convolutional generative adversarial networks." arXiv preprint arXiv:1511.06434 (2015).

CHAPTER 19

如何自定义数据集

本文档将介绍 MMagic 中每一个数据集的设计方式，以及用户如何设计自定义数据集。

- 如何自定义数据集
  - 支持的数据集格式
    - *BasicImageDataset*
    - *BasicFramesDataset*
    - *BasicConditonalDataset*
      - 1. 逐行读取的标注文件格式（例如 txt 文件）
      - 2. 基于字典的标注文件格式（例如 json）
      - 3. 基于文件夹的标注格式（无需标注文件）
    - *ImageNet 和 CIFAR10 数据集*
    - *AdobeComp1kDataset*
    - *GrowScaleImgDataset*
    - *SinGANDataset*
    - *PairedImageDataset*
    - *UnpairedImageDataset*
  - 实现一个新的数据集
    - 重复数据集

19.1 支持的数据集格式

在 MMagic 中，所有的数据集都是从 `BaseDataset` 类继承而来的。每个数据集都通过 `load_data_list` 方法来加载数据信息列表（例如数据所在的路径）。在 `__getitem__` 方法中，调用 `prepare_data` 来获取前处理后的数据。在 `prepare_data` 方法中，数据加载流程包括如下步骤：

1. 通过传入的索引来获取数据信息，由 `get_data_info` 方法实现。
2. 对数据应用数据转换，由 `pipeline` 方法实现。

19.1.1 BasicImageDataset

BasicImageDataset `mmagic.datasets.BasicImageDataset` 是一个通用图片数据集，是为了底层视觉任务而设计的，比如图像超分辨率，图像修复和无条件图像生成。可以选择是否使用标注文件。

如使用标注文件，标注的格式可以如下所示：

```
  Case 1 (CelebA-HQ):

      000001.png
      000002.png

  Case 2 (DIV2K):

      0001_s001.png (480,480,3)
      0001_s002.png (480,480,3)
      0001_s003.png (480,480,3)
      0002_s001.png (480,480,3)
      0002_s002.png (480,480,3)

  Case 3 (Vimeo90k):

      00001/0266 (256, 448, 3)
      00001/0268 (256, 448, 3)
```

下面我们给出几个如何使用 `BasicImageDataset` 的示例。假定文件结构如下：

```
mmagic (root)
├── mmagic
├── tools
├── configs
├── data
|   ├── DIV2K
|   |   ├── DIV2K_train_HR
|   |   |   ├── image.png
```

(下页继续)

(续上页)

```
|   |   ├── DIV2K_train_LR_bicubic
|   |   |   ├── X2
|   |   |   ├── X3
|   |   |   ├── X4
|   |   |   |   ├── image_x4.png
|   |   ├── DIV2K_valid_HR
|   |   ├── DIV2K_valid_LR_bicubic
|   |   |   ├── X2
|   |   |   ├── X3
|   |   |   ├── X4
|   ├── places
|   |   ├── test_set
|   |   ├── train_set
|   |   ├── meta
|   |   |    ├── Places365_train.txt
|   |   |    ├── Places365_val.txt
|   ├── celebahq
|   |   ├── imgs_1024
```

按照以上的文件结构给出 3 个示例。

示例 1: 加载 DIV2K 数据集来训练一个 SISR 模型。

```
dataset = BasicImageDataset(
    ann_file='',
    metainfo=dict(
        dataset_type='div2k',
        task_name='sisr'),
    data_root='data/DIV2K',
    data_prefix=dict(
        gt='DIV2K_train_HR', img='DIV2K_train_LR_bicubic/X4'),
    filename_tmpl=dict(img='{}_x4', gt='{}'),
    pipeline=[])
```

示例 2: 加载 places 数据集来训练一个 inpainting 模型.

```
dataset = BasicImageDataset(
    ann_file='meta/Places365_train.txt',
    metainfo=dict(
        dataset_type='places365',
        task_name='inpainting'),
    data_root='data/places',
    data_prefix=dict(gt='train_set'),
    pipeline=[])
```

示例 3: 加载 CelebA-HQ 数据集来训练一个 PGGAN 模型.

```
dataset = BasicImageDataset(
        pipeline=[],
        data_root='./data/celebahq/imgs_1024')
```

19.1.2 BasicFramesDataset

BasicFramesDataset mmagic.datasets.BasicFramesDataset 也是一个通用图片数据集，为视频帧的底层视觉任务而设计的，比如视频的超分辨率和视频帧插值。可以选择是否使用标注文件。

如使用标注文件, 标注的格式示例所示：

```
Case 1 (Vid4):

   calendar 41
   city 34
   foliage 49
   walk 47

Case 2 (REDS):

   000/00000000.png (720, 1280, 3)
   000/00000001.png (720, 1280, 3)

Case 3 (Vimeo90k):

   00001/0266 (256, 448, 3)
   00001/0268 (256, 448, 3)
```

假定文件结构如下：

```
mmagic (root)
├── mmagic
├── tools
├── configs
├── data
|   ├── Vid4
|   |   ├── BIx4
|   |   |   ├── city
|   |   |   |   ├── img1.png
|   |   ├── GT
|   |   |   ├── city
|   |   |   |   ├── img1.png
```

(下页继续)

(续上页)

```
|   |   ├── meta_info_Vid4_GT.txt
|   ├── vimeo-triplet
|   |   ├── sequences
|   |   |   ├── 00001
|   |   |   |   ├── 0389
|   |   |   |   |   ├── img1.png
|   |   |   |   |   ├── img2.png
|   |   |   |   |   ├── img3.png
|   |   ├── tri_trainlist.txt
```

按照以上的文件结构给出两个示例。

示例 1: 加载 Vid4 数据集来训练一个 VSR 模型.

```
dataset = BasicFramesDataset(
    ann_file='meta_info_Vid4_GT.txt',
    metainfo=dict(dataset_type='vid4', task_name='vsr'),
    data_root='data/Vid4',
    data_prefix=dict(img='BIx4', gt='GT'),
    pipeline=[],
    depth=2,
    num_input_frames=5)
```

示例 2: 加载 Vimeo90k 数据集来训练一个 VFI 模型.

```
dataset = BasicFramesDataset(
    ann_file='tri_trainlist.txt',
    metainfo=dict(dataset_type='vimeo90k', task_name='vfi'),
    data_root='data/vimeo-triplet',
    data_prefix=dict(img='sequences', gt='sequences'),
    pipeline=[],
    depth=2,
    load_frames_list=dict(
        img=['img1.png', 'img3.png'], gt=['img2.png']))
```

19.1.3 BasicConditonalDataset

BasicConditonalDataset mmagic.datasets.BasicConditonalDataset 是为条件生成对抗网络而设计的（例如 SAGAN、BigGAN）。该数据集支持为标注文件加载标签。BasicConditonalDataset 支持如下 3 种标注格式。

1. 逐行读取的标注文件格式（例如 txt 文件）

样本文件结构：

```
data_prefix/
├── folder_1
│   ├── xxx.png
│   ├── xxy.png
│   └── ...
└── folder_2
    ├── 123.png
    ├── nsdf3.png
    └── ...
```

样本标注文件格式（第一列是图像的路径，第二列是类别的索引）

```
folder_1/xxx.png 0
folder_1/xxy.png 1
folder_2/123.png 5
folder_2/nsdf3.png 3
...
```

ImageNet 数据集的配置示例：

```
dataset=dict(
    type='BasicConditionalDataset,
    data_root='./data/imagenet/',
    ann_file='meta/train.txt',
    data_prefix='train',
    pipeline=train_pipeline),
```

2. 基于字典的标注文件格式（例如 json）

样本文件结构：

```
data_prefix/
├── folder_1
│   ├── xxx.png
│   ├── xxy.png
│   └── ...
└── folder_2
    ├── 123.png
    ├── nsdf3.png
    └── ...
```

样本标注文件格式（键为图像的路径，值为标签）。

```
    {
        "folder_1/xxx.png": [1, 2, 3, 4],
        "folder_1/xxy.png": [2, 4, 1, 0],
        "folder_2/123.png": [0, 9, 8, 1],
        "folder_2/nsdf3.png", [1, 0, 0, 2],
        ...
    }
```

EG3D (shapenet-car) 数据集的配置示例：

```
dataset = dict(
    type='BasicConditionalDataset',
    data_root='./data/eg3d/shapenet-car',
    ann_file='annotation.json',
    pipeline=train_pipeline)
```

在这种类型的注释中，标签可以是任何类型，不仅限于索引。

3. 基于文件夹的标注格式（无需标注文件）

样本文件结构：

```
    data_prefix/
    ├── class_x
    │   ├── xxx.png
    │   ├── xxy.png
    │   └── ...
    │       └── xxz.png
    └── class_y
        ├── 123.png
        ├── nsdf3.png
        ├── ...
        └── asd932_.png
```

如果在配置的 ann_file 中指定了标注文件，则将使用上面的前两种方式生成数据集，否则将尝试使用第三种方式。

19.1.4 ImageNet 和 CIFAR10 数据集

ImageNet 数据集 `mmagic.datasets.ImageNet` 和 **CIFAR10 数据集** `mmagic.datasets.CIFAR10` 是为 `ImageNet` 和 `CIFAR10` 这两个数据集而设计的。这两个数据集都是基于 `BasicConditionalDataset` 封装的。您可以使用它们来轻松加载这两个数据集的数据。

`ImageNet` 的配置示例:

```
pipeline = [
    dict(type='LoadImageFromFile', key='img'),
    dict(type='RandomCropLongEdge', keys=['img']),
    dict(type='Resize', scale=(128, 128), keys=['img'], backend='pillow'),
    dict(type='Flip', keys=['img'], flip_ratio=0.5, direction='horizontal'),
    dict(type='PackInputs')
]

dataset=dict(
    type='ImageNet',
    data_root='./data/imagenet/',
    ann_file='meta/train.txt',
    data_prefix='train',
    pipeline=pipeline),
```

`CIFAR10` 的配置示例:

```
pipeline = [dict(type='PackInputs')]

dataset = dict(
    type='CIFAR10',
    data_root='./data',
    data_prefix='cifar10',
    test_mode=False,
    pipeline=pipeline)
```

19.1.5 AdobeComp1kDataset

AdobeComp1kDataset `mmagic.datasets.AdobeComp1kDataset` 是为 `Adobe composition-1k` 数据集而设计的。

该数据集加载（alpha, fg, bg）数据，并对数据执行指定的变换。您可以在 `pipeline` 中指定在线合成图像或加载离线已合成的图像。

在线合成 `comp-1k` 数据集示例:

```
[
   {
        "alpha": 'alpha/000.png',
        "fg": 'fg/000.png',
        "bg": 'bg/000.png'
   },
   {
        "alpha": 'alpha/001.png',
        "fg": 'fg/001.png',
        "bg": 'bg/001.png'
   },
]
```

离线合成 comp-1k 数据集示例:

```
[
  {
      "alpha": 'alpha/000.png',
      "merged": 'merged/000.png',
      "fg": 'fg/000.png',
      "bg": 'bg/000.png'
  },
  {
      "alpha": 'alpha/001.png',
      "merged": 'merged/001.png',
      "fg": 'fg/001.png',
      "bg": 'bg/001.png'
  },
]
```

19.1.6 GrowScaleImgDataset

GrowScaleImgDataset 是为了动态 GAN 模型（例如 PGGAN 和 StyleGANv1）而设计的。在这个数据集中，我们支持在训练过程中切换数据根目录，来加载不同分辨率的训练图像。这个过程是通过 GrowScaleImgDataset.update_annotations 方法实现的，并在训练过程中由 PGGANFetchDataHook.before_train_iter 调用。

```
def update_annotations(self, curr_scale):
    # 确定是否需要更新数据根目录
    if curr_scale == self._actual_curr_scale:
        return False

    # 按图像分辨率（尺度）提取新的数据根目录
```

(下页继续)

(续上页)

```
    for scale in self._img_scales:
        if curr_scale <= scale:
            self._curr_scale = scale
            break
        if scale == self._img_scales[-1]:
            assert RuntimeError(
                f'Cannot find a suitable scale for {curr_scale}')
    self._actual_curr_scale = curr_scale
    self.data_root = self.data_roots[str(self._curr_scale)]

    # 使用新的数据根目录重新加载数据列表
    self.load_data_list()

    # print basic dataset information to check the validity
    print_log('Update Dataset: ' + repr(self), 'current')
    return True
```

19.1.7 SinGANDataset

SinGANDataset 是为 SinGAN 模型训练而设计的数据集。在 SinGAN 的训练中，我们不会去迭代数据集中的图像，而是返回一个一致的预处理图像字典。由于不需要根据给定的索引加载相应的图像数据，我们绕过了 BaseDataset 的默认数据加载逻辑。

```
def load_data_list(self, min_size, max_size, scale_factor_init):
    # 加载单张图像
    real = mmcv.imread(self.data_root)
    self.reals, self.scale_factor, self.stop_scale = create_real_pyramid(
        real, min_size, max_size, scale_factor_init)

    self.data_dict = {}

    # 生成多尺度图像
    for i, real in enumerate(self.reals):
        self.data_dict[f'real_scale{i}'] = real

    self.data_dict['input_sample'] = np.zeros_like(
        self.data_dict['real_scale0']).astype(np.float32)

def __getitem__(self, index):
    # 直接返回转换过的数据字典
    return self.pipeline(self.data_dict)
```

19.1.8 PairedImageDataset

PairedImageDataset 专为需要成对训练数据的图像转换模型（例如 Pix2Pix）设计。

目录结构如下所示，其中每个图像文件都是图像对的拼接。

```
./data/dataset_name/
├── test
|   └── XXX.jpg
└── train
    └── XXX.jpg
```

在 PairedImageDataset 中，我们在 load_data_list 方法中扫描文件列表，然后将路径保存在 pair_path 字段中，以适配 LoadPairedImageFromFile 中的转换。

```
def load_data_list(self):
    data_infos = []
    pair_paths = sorted(self.scan_folder(self.data_root))
    for pair_path in pair_paths:
        # save path in the specific field
        data_infos.append(dict(pair_path=pair_path))

    return data_infos
```

19.1.9 UnpairedImageDataset

UnpairedImageDataset 是专为不需要成对数据的图像转换模型（例如 CycleGAN）设计的数据集。

目录结构如下所示：

```
./data/dataset_name/
├── testA
|   └── XXX.jpg
├── testB
|   └── XXX.jpg
├── trainA
|   └── XXX.jpg
└── trainB
    └── XXX.jpg
```

在该数据集中，我们重载了 __getitem__ 方法，实现了在训练过程中加载随机的图像对。

```
def __getitem__(self, idx):
    if not self.test_mode:
```

(下页继续)

(续上页)

```
        return self.prepare_train_data(idx)

    return self.prepare_test_data(idx)

def prepare_train_data(self, idx):
    img_a_path = self.data_infos_a[idx % self.len_a]['path']
    idx_b = np.random.randint(0, self.len_b)
    img_b_path = self.data_infos_b[idx_b]['path']
    results = dict()
    results[f'img_{self.domain_a}_path'] = img_a_path
    results[f'img_{self.domain_b}_path'] = img_b_path
    return self.pipeline(results)

def prepare_test_data(self, idx):
    img_a_path = self.data_infos_a[idx % self.len_a]['path']
    img_b_path = self.data_infos_b[idx % self.len_b]['path']
    results = dict()
    results[f'img_{self.domain_a}_path'] = img_a_path
    results[f'img_{self.domain_b}_path'] = img_b_path
    return self.pipeline(results)
```

19.2 实现一个新的数据集

如果您需要为一个新的底层 CV 任务（例如去噪、去雨、去雾和去反射）创建一个数据集，或者现有的数据集格式不符合您的需求，您可以将新的数据格式重新组织成现有的格式，或者在 `mmagic/datasets` 中创建一个新的数据集中来加载数据。

从现有的数据集基类中继承（例如 `BasicImageDataset` 和 `BasicFramesDataset`）会比较容易创建一个新的数据集。

您也可以创建一个继承自 BaseDataset 的新数据集，它是定义在 MMEngine 中的数据集基类。

下面是创建一个用于视频帧插值的数据集的示例：

```
from .basic_frames_dataset import BasicFramesDataset
from mmagic.registry import DATASETS


@DATASETS.register_module()
class NewVFIDataset(BasicFramesDataset):
    """Introduce the dataset

    Examples of file structure.
```

(下页继续)

(续上页)

```
    Args:
        pipeline (list[dict | callable]): A sequence of data transformations.
        folder (str | :obj:`Path`): Path to the folder.
        ann_file (str | :obj:`Path`): Path to the annotation file.
        test_mode (bool): Store `True` when building test dataset.
            Default: `False`.
    """

    def __init__(self, ann_file, metainfo, data_root, data_prefix,
                 pipeline, test_mode=False):
        super().__init__(ann_file, metainfo, data_root, data_prefix,
                         pipeline, test_mode)
        self.data_infos = self.load_annotations()

    def load_annotations(self):
        """Load annoations for the dataset.

        Returns:
            list[dict]: A list of dicts for paired paths and other information.
        """
        data_infos = []
        ...
        return data_infos
```

欢迎提交新的数据集类到 MMagic

19.2.1 重复数据集

我们使用 RepeatDataset 作为包装器来重复数据集。例如，假设原始数据集是 `Dataset_A`，为了重复它，配置文件应该如下所示：

```
dataset_A_train = dict(
        type='RepeatDataset',
        times=N,
        dataset=dict(  # This is the original config of Dataset_A
            type='Dataset_A',
            ...
            pipeline=train_pipeline
        )
    )
```

您可以参考 MMEngine 中的教程。

CHAPTER 20

如何设计自己的数据变换

在本教程中，我们将介绍 MMagic 中变换流水线的设计。

The structure of this guide are as follows:

- 如何设计自己的数据变换
  - *MMagic* 中的数据流水线
    - 数据变换的一个简单示例
    - *BasicVSR* 的一个示例
    - *Pix2Pix* 的一个示例
  - *MMagic* 中支持的数据变换
    - 数据加载
    - 预处理
    - 格式化
  - 扩展和使用自定义流水线
    - 一个简单的 *MyTransform* 示例
    - 一个翻转变换的示例

20.1 MMagic 中的数据流水线

按照典型的惯例，我们使用 Dataset 和 DataLoader 来加载多个线程的数据。Dataset 返回一个与模型的 forward 方法的参数相对应的数据项的字典。

数据准备流水线和数据集是分开的。通常，一个数据集定义了如何处理标注，而一个数据管道定义了准备一个数据字典的所有步骤。

一个流水线由一连串的操作组成。每个操作都需要一个字典作为输入，并为下一个变换输出一个字典。

这些操作被分为数据加载、预处理和格式化。

在 MMagic 中，所有数据变换都继承自 BaseTransform。变换的输入和输出类型都是字典。

20.1.1 数据变换的一个简单示例

```
>>> from mmagic.transforms import LoadPairedImageFromFile
>>> transforms = LoadPairedImageFromFile(
>>>     key='pair',
>>>     domain_a='horse',
>>>     domain_b='zebra',
>>>     flag='color'),
>>> data_dict = {'pair_path': './data/pix2pix/facades/train/1.png'}
>>> data_dict = transforms(data_dict)
>>> print(data_dict.keys())
dict_keys(['pair_path', 'pair', 'pair_ori_shape', 'img_mask', 'img_photo', 'img_mask_
↪path', 'img_photo_path', 'img_mask_ori_shape', 'img_photo_ori_shape'])
```

一般来说，变换流水线的最后一步必须是 PackInputs. PackInputs 将把处理过的数据打包成一个包含两个字段的字典：inputs 和 data_samples. inputs 是你想用作模型输入的变量，它可以是 torch.Tensor 的类型，torch.Tensor 的字典，或者你想要的任何类型。data_samples 是一个 DataSample 的列表. 每个 DataSample 都包含真实值和对应输入的必要信息。

20.1.2 BasicVSR 的一个示例

下面是一个 BasicVSR 的流水线示例。

```
train_pipeline = [
    dict(type='LoadImageFromFile', key='img', channel_order='rgb'),
    dict(type='LoadImageFromFile', key='gt', channel_order='rgb'),
    dict(type='SetValues', dictionary=dict(scale=scale)),
    dict(type='PairedRandomCrop', gt_patch_size=256),
    dict(
        type='Flip',
```

(下页继续)

(续上页)

```
        keys=['img', 'gt'],
        flip_ratio=0.5,
        direction='horizontal'),
    dict(
        type='Flip', keys=['img', 'gt'], flip_ratio=0.5, direction='vertical'),
    dict(type='RandomTransposeHW', keys=['img', 'gt'], transpose_ratio=0.5),
    dict(type='MirrorSequence', keys=['img', 'gt']),
    dict(type='PackInputs')
]

val_pipeline = [
    dict(type='GenerateSegmentIndices', interval_list=[1]),
    dict(type='LoadImageFromFile', key='img', channel_order='rgb'),
    dict(type='LoadImageFromFile', key='gt', channel_order='rgb'),
    dict(type='PackInputs')
]

test_pipeline = [
    dict(type='LoadImageFromFile', key='img', channel_order='rgb'),
    dict(type='LoadImageFromFile', key='gt', channel_order='rgb'),
    dict(type='MirrorSequence', keys=['img']),
    dict(type='PackInputs')
]
```

对于每个操作，我们列出了添加/更新/删除的相关字典字段，标记为‘*’的字典字段是可选的。

20.1.3 Pix2Pix 的一个示例

下面是一个在 aerial2maps 数据集上 Pix2Pix 训练的流水线示例。

```
source_domain = 'aerial'
target_domain = 'map'

pipeline = [
    dict(
        type='LoadPairedImageFromFile',
        io_backend='disk',
        key='pair',
        domain_a=domain_a,
        domain_b=domain_b,
        flag='color'),
    dict(
        type='TransformBroadcaster',
```

(下页继续)

(续上页)

```
        mapping={'img': [f'img_{domain_a}', f'img_{domain_b}']},
        auto_remap=True,
        share_random_params=True,
        transforms=[
            dict(
                type='mmagic.Resize', scale=(286, 286),
                interpolation='bicubic'),
            dict(type='mmagic.FixedCrop', crop_size=(256, 256))
        ]),
    dict(
        type='Flip',
        keys=[f'img_{domain_a}', f'img_{domain_b}'],
        direction='horizontal'),
    dict(
        type='PackInputs',
        keys=[f'img_{domain_a}', f'img_{domain_b}', 'pair'])
```

20.2 MMagic 中支持的数据变换

20.2.1 数据加载

20.2.2 预处理

20.2.3 格式化

20.2.4 Albumentations

MMagic 支持添加 Albumentations 库中的 transformation，请浏览 https://albumentations.ai/docs/getting_started/transforms_and_targets 获取更多 transformation 的信息。

使用 Albumentations 的示例如下：

```
albu_transforms = [
   dict(
         type='Resize',
         height=100,
         width=100,
   ),
   dict(
         type='RandomFog',
         p=0.5,
```

(下页继续)

(续上页)

```
    ),
    dict(
            type='RandomRain',
            p=0.5
    ),
    dict(
            type='RandomSnow',
            p=0.5,
    ),
]
pipeline = [
    dict(
            type='LoadImageFromFile',
            key='img',
            color_type='color',
            channel_order='rgb',
            imdecode_backend='cv2'),
    dict(
            type='Albumentations',
            keys=['img'],
            transforms=albu_transforms),
    dict(type='PackInputs')
]
```

20.3 扩展和使用自定义流水线

20.3.1 一个简单的 MyTransform 示例

1. 在文件中写入一个新的流水线，例如在 `my_pipeline.py` 中。它接受一个字典作为输入，并返回一个字典。

```
import random
from mmcv.transforms import BaseTransform
from mmagic.registry import TRANSFORMS


@TRANSFORMS.register_module()
class MyTransform(BaseTransform):
    """Add your transform

    Args:
```

(下页继续)

(续上页)

```
        p (float): Probability of shifts. Default 0.5.
    """

    def __init__(self, p=0.5):
        self.p = p

    def transform(self, results):
        if random.random() > self.p:
            results['dummy'] = True
        return results

    def __repr__(self):

        repr_str = self.__class__.__name__
        repr_str += (f'(p={self.p})')

        return repr_str
```

2. 在你的配置文件中导入并使用该流水线。

确保导入相对于你的训练脚本所在的位置。

```
train_pipeline = [
    ...
    dict(type='MyTransform', p=0.2),
    ...
]
```

20.3.2 一个翻转变换的示例

这里我们以一个简单的翻转变换为例：

```
import random
import mmcv
from mmcv.transforms import BaseTransform, TRANSFORMS

@TRANSFORMS.register_module()
class MyFlip(BaseTransform):
    def __init__(self, direction: str):
        super().__init__()
        self.direction = direction

    def transform(self, results: dict) -> dict:
```

(下页继续)

(续上页)

```
        img = results['img']
        results['img'] = mmcv.imflip(img, direction=self.direction)
        return results
```

因此，我们可以实例化一个 MyFlip 对象，用它来处理数据字典。

```
import numpy as np

transform = MyFlip(direction='horizontal')
data_dict = {'img': np.random.rand(224, 224, 3)}
data_dict = transform(data_dict)
processed_img = data_dict['img']
```

或者，我们可以在配置文件的数据流水线中使用 MyFlip 变换。

```
pipeline = [
    ...
    dict(type='MyFlip', direction='horizontal'),
    ...
]
```

请注意，如果你想在配置中使用 MyFlip ，你必须确保在程序运行过程中导入包含 MyFlip 的文件。

CHAPTER 21

如何设计自己的损失函数

`losses` 在 `MMagic` 中注册为 `LOSSES`。在 MMagic 中设计自己的损失函数，步骤和在 MMagic 中自定义任何其他模型类似。本节主要具体介绍了如何在 MMagic 中实现自定义的损失函数。本教程建议您在实现自定义的损失函数时，应该遵循本教程相同的设计，这样在我们的框架中使用您新定义的损失函数，就不需要额外的工作。

本指南包括：

- 设计你自己的损失函数
  - 支持损失函数介绍
  - 设计一个新的损失函数
    * MSELoss 的一个例子
    * DiscShiftLoss 的一个例子
    * GANWithCustomizedLoss 的一个例子
  - 可用损失函数
    * 常规损失函数
    * 损失函数组件

21.1 支持的损失函数介绍

为了方便使用，您可以直接使用我们为具体算法设置的默认损失计算过程，如 lsgan、biggan、styleganv2 等。以 `stylegan2` 为例，我们使用 R1 梯度惩罚和生成器路径长度正则化作为可配置损失，用户可以调整相关参数，如 `r1_loss_weight` 和 `g_reg_weight`。

```
# stylegan2_base.py
loss_config =dict(
     r1_loss_weight=10。 / 2. * d_reg_interval,
     r1_interval=d_reg_interval,
     norm_mode='HWC',
     g_reg_interval=g_reg_interval,
     g_reg_weight=2。 * g_reg_interval,
     pl_batch_shrink=2)

model=dict(
     type='StyleGAN2',
     xxx,
     loss_config=loss_config)
```

21.2 设计一个新的损失函数

21.2.1 MSELoss 的一个例子

一般来说，要实现一个损失模块，我们会编写一个函数实现，然后用类实现包装它。以 MSELoss 为例：

```
@masked_loss
def mse_loss(pred, target)：
     return F.mse_loss (pred, target, reduction='none')

@LOSSES.register_module()
Class MSELoss(nn.Module)：

     def __init__(self, loss_weight=1.0, reduction='mean', sample_wise=False):
         # 代码可以在``mmagic/models/losses/pixelwise_loss.py``中找到

     def forward(self, pred, target, weight=None, **kwargs):
         # 代码可以在``mmagic/models/losses/pixelwise_loss.py``中找到
```

根据这个损失函数的定义，我们现在可以简单地通过在配置文件中定义它来使用：

```
pixel_loss=dict(type='MSELoss', loss_weight=1.0, reduction='mean')
```

请注意，上面的 pixel_loss 必须在模型中定义。详情请参考自定义模型。与自定义模型类似，为了使用您自己实现的损失函数，您需要在编写后在 mmagic/models/losses/__init__.py 中导入该损失函数。

21.2.2 DiscShiftLoss 的一个例子

一般来说，要实现一个损失模块，我们会编写一个函数实现，然后用类实现包装它。但是，在 MMagic 中，我们提供了另一个统一的接口 data_info 供用户定义输入参数和数据项之间的映射。

```
@weighted_loss
def disc_shift_loss(pred):
     return pred**2

@MODULES.register_module()
Class DiscShiftLoss(nn.Module):

     def __init__(self, loss_weight=1.0, data_info=None):
         super(DiscShiftLoss, self).__init__()
         # 代码可以在``mmagic/models/losses/disc_auxiliary_loss.py``中找到

     def forward(self, *args, **kwargs):
         # 代码可以在``mmagic/models/losses/disc_auxiliary_loss.py``中找到
```

这种损失模块设计的目标是允许在生成模型 (MODELS) 中自动使用它，而无需其他复杂代码来定义数据和关键字参数之间的映射。因此，与 OpenMMLab 中的其他框架不同，我们的损失模块包含一个特殊的关键字 data_info，它是一个定义输入参数与生成模型数据之间映射的字典。以 DiscShiftLoss 为例，用户在编写配置文件时，可能会用到这个 loss，如下：

```
dict(type='DiscShiftLoss',
    loss_weight=0.001 * 0.5,
    data_info=dict(pred='disc_pred_real'))
```

data_info 中的信息告诉模块使用 disc_pred_real 数据作为 pred 参数的输入张量。一旦 data_info 不为 None，我们的损失模块将自动构建计算图。

```
@MODULES.register_module()
class DiscShiftLoss(nn.Module):

    def __init__(self, loss_weight=1.0, data_info=None):
        super(DiscShiftLoss, self).__init__()
        self.loss_weight = loss_weight
        self.data_info = data_info
```

(下页继续)

(续上页)

```
    def forward(self, *args, **kwargs):
        # use data_info to build computational path
        if self.data_info is not None:
            # parse the args and kwargs
            if len(args) == 1:
                assert isinstance(args[0], dict), (
                    'You should offer a dictionary containing network outputs '
                    'for building up computational graph of this loss module.')
                outputs_dict = args[0]
            elif 'outputs_dict' in kwargs:
                assert len(args) == 0, (
                    'If the outputs dict is given in keyworded arguments, no'
                    ' further non-keyworded arguments should be offered.')
                outputs_dict = kwargs.pop('outputs_dict')
            else:
                raise NotImplementedError(
                    'Cannot parsing your arguments passed to this loss module.'
                    ' Please check the usage of this module')
            # link the outputs with loss input args according to self.data_info
            loss_input_dict = {
                k: outputs_dict[v]
                for k, v in self.data_info.items()
            }
            kwargs.update(loss_input_dict)
            kwargs.update(dict(weight=self.loss_weight))
            return disc_shift_loss(**kwargs)
        else:
            # if you have not define how to build computational graph, this
            # module will just directly return the loss as usual.
            return disc_shift_loss(*args, weight=self.loss_weight, **kwargs)

    @staticmethod
    def loss_name():
        return 'loss_disc_shift'
```

如这部分代码所示，一旦用户设置了 “data_info”，损失模块将收到一个包含所有必要数据和模块的字典，该字典由训练过程中的 “MODELS” 提供。如果此字典作为非关键字参数给出，则应将其作为第一个参数提供。如果您使用关键字参数，请将其命名为 `outputs_dict`。

21.2.3 GANWithCustomizedLoss 的一个例子

为了构建计算图，生成模型必须提供包含各种数据的字典。仔细观察任何生成模型，你会发现我们将各种特征和模块收集到字典中。我们在这里提供了一个自定义的 `GANWithCustomizedLoss` 来展示这个过程。

```
class GANWithCustomizedLoss(BaseModel):

    def __init__(self, gan_loss, disc_auxiliary_loss, gen_auxiliary_loss,
                 *args, **kwargs):
        # ...
        if gan_loss is not None:
            self.gan_loss = MODULES.build(gan_loss)
        else:
            self.gan_loss = None

        if disc_auxiliary_loss:
            self.disc_auxiliary_losses = MODULES.build(disc_auxiliary_loss)
            if not isinstance(self.disc_auxiliary_losses, nn.ModuleList):
                self.disc_auxiliary_losses = nn.ModuleList(
                    [self.disc_auxiliary_losses])
        else:
            self.disc_auxiliary_loss = None

        if gen_auxiliary_loss:
            self.gen_auxiliary_losses = MODULES.build(gen_auxiliary_loss)
            if not isinstance(self.gen_auxiliary_losses, nn.ModuleList):
                self.gen_auxiliary_losses = nn.ModuleList(
                    [self.gen_auxiliary_losses])
        else:
            self.gen_auxiliary_losses = None

    def train_step(self, data: dict,
                   optim_wrapper: OptimWrapperDict) -> Dict[str, Tensor]:
        # ...

        # get data dict to compute losses for disc
        data_dict_ = dict(
            iteration=curr_iter,
            gen=self.generator,
            disc=self.discriminator,
            disc_pred_fake=disc_pred_fake,
            disc_pred_real=disc_pred_real,
            fake_imgs=fake_imgs,
            real_imgs=real_imgs)
```

(下页继续)

(续上页)

```
        loss_disc, log_vars_disc = self._get_disc_loss(data_dict_)

        # ...

    def _get_disc_loss(self, outputs_dict):
        # Construct losses dict. If you hope some items to be included in the
        # computational graph, you have to add 'loss' in its name. Otherwise,
        # items without 'loss' in their name will just be used to print
        # information.
        losses_dict = {}
        # gan loss
        losses_dict['loss_disc_fake'] = self.gan_loss(
            outputs_dict['disc_pred_fake'], target_is_real=False, is_disc=True)
        losses_dict['loss_disc_real'] = self.gan_loss(
            outputs_dict['disc_pred_real'], target_is_real=True, is_disc=True)

        # disc auxiliary loss
        if self.with_disc_auxiliary_loss:
            for loss_module in self.disc_auxiliary_losses:
                loss_ = loss_module(outputs_dict)
                if loss_ is None:
                    continue

                # the `loss_name()` function return name as 'loss_xxx'
                if loss_module.loss_name() in losses_dict:
                    losses_dict[loss_module.loss_name(
                    )] = losses_dict[loss_module.loss_name()] + loss_
                else:
                    losses_dict[loss_module.loss_name()] = loss_
        loss, log_var = self.parse_losses(losses_dict)

        return loss, log_var
```

在这里，_get_disc_loss 将帮助自动组合各种损失函数。

因此，只要用户设计相同规则的损失模块，就可以在生成模型的训练中插入任何一种损失，无需对模型代码进行其他修改。您只需要在配置文件中定义 data_info 即可。

21.3 可用损失函数

我们在配置中列出了可用的损失示例，如下所示。

21.3.1 常规损失函数

```
# dic gan
loss_gan=dict(
    type='GANLoss',
    gan_type='vanilla',
    loss_weight=0.001,
)
```

```
# deepfillv1
loss_gan=dict(
    type='GANLoss',
    gan_type='wgan',
    loss_weight=0.0001,
)
```

```
# deepfillv2
loss_gan=dict(
    type='GANLoss',
    gan_type='hinge',
    loss_weight=0.1,
)
```

```
# aot-gan
loss_gan=dict(
    type='GANLoss',
    gan_type='smgan',
    loss_weight=0.01,
)
```

```
# deepfillv1
loss_gp=dict(type='GradientPenaltyLoss', loss_weight=10.)
```

```
# deepfillv1
loss_disc_shift=dict(type='DiscShiftLoss', loss_weight=0.001)
```

```
# dim
loss_comp=dict(type='CharbonnierCompLoss', loss_weight=0.5)
```

```
# dic gan
feature_loss=dict(
    type='LightCNNFeatureLoss',
    pretrained=pretrained_light_cnn,
    loss_weight=0.1,
    criterion='l1')
```

```
# dic gan
pixel_loss=dict(type='L1Loss', loss_weight=1.0, reduction='mean')
```

```
# dic gan
align_loss=dict(type='MSELoss', loss_weight=0.1, reduction='mean')
```

```
# dim
loss_alpha=dict(type='CharbonnierLoss', loss_weight=0.5)
```

```
# partial conv
loss_tv=dict(
    type='MaskedTVLoss',
    loss_weight=0.1
)
```

```
# real_basicvsr
perceptual_loss=dict(
    type='PerceptualLoss',
    layer_weights={
        '2': 0.1,
        '7': 0.1,
        '16': 1.0,
        '25': 1.0,
        '34': 1.0,
    },
    vgg_type='vgg19',
    perceptual_weight=1.0,
    style_weight=0,
    norm_img=False)
```

```
# ttsr
transferal_perceptual_loss=dict(
    type='TransferalPerceptualLoss',
    loss_weight=1e-2,
    use_attention=False,
    criterion='mse')
```

21.3.2 损失函数组件

对于“GANWithCustomizedLoss”，我们提供了几个组件来构建自定义损失。

CHAPTER 22

常见问题解答

我们在此列出了许多用户面临的一些常见问题及其相应的解决方案。如果您发现任何常见问题，并有办法帮助他人解决这些问题，请随时丰富列表内容。如果这里的内容没有涵盖您的问题，请使用提供的模板创建一个问题，并确保填写了模板中的所有必要信息。

22.1 常见问题

问题 1：“xxx: ‘yyy is not in the zzz registry’”.

回答 1：只有导入模块文件时，才会触发注册表机制。所以你需要在某个地方导入该文件。

问题 2：某个数据集的文件夹结构是什么?

回答 2：您可以根据数据集准备教程来确保文件夹结构的正确性。

问题 3：如何使用 LMDB 数据训练模型?

回答 3：您可以使用工具/数据中的脚本制作 LMDB 文件。更多详情请参见数据集准备教程。

问题 4：为什么使用了 MMCV==xxx，但在导入 mmagic 时却出现了不兼容?

回答 4：这是因为 MMCV 和 MMagic 的版本不兼容。兼容的 MMagic 和 MMCV 版本如下所示。请选择正确的 MMCV 版本以避免安装问题。

注意：如果已安装 mmcv，则需要先运行 pip uninstall mmcv。如果同时安装了 mmcv 和 mmcv-full，则会出现模块未找到错误（ModuleNotFoundError）。**问题 5：**如何忽略基本配置中的某些字段?

回答 5：有些时候您可以设置 *delete*=True 来忽略基本配置中的某些字段。您可以参考 MMEngine 的简单说明。您可以仔细阅读本教程，以便更好地理解这一功能。

问题 6：如何在配置中使用中间变量?

回答 6：有些中间变量会在配置文件中使用，比如数据集中的 train_pipeline/test_pipeline。值得注意的是，当修改子配置中的中间变量时，用户需要再次将中间变量传递到相应的字段中。

CHAPTER 23

概览

23.1 概览

- 预训练权重个数: 124
- 配置文件个数: 0
- 论文个数: 32
  - ALGORITHM: 34

CHAPTER 24

去模糊

24.1 概览

- 预训练权重个数: 2
- 配置文件个数: 0
- 论文个数: 1
  - ALGORITHM: 1

24.2 DeblurGAN-v2 (ICCV’2019)

DeblurGAN-v2: Deblurring (Orders-of-Magnitude) Faster and Better

任务: 去模糊

24.2.1 摘要

我们提出了一种新的端到端的用于单图像运动去模糊生成对抗网络（GAN），名为 DeblurGAN-v2，它显著提高了最高水平的去模糊效率、质量和灵活性。DeblurGAN-v2 是基于具有双尺度鉴别器的相对论条件 GAN。我们首次将特征金字塔网络引入去模糊来作为 DeblurGAN-v2 生成器的核心构建块。它可以灵活地与各种主干网络一起工作，以在性能和效率之间取得平衡。先进的主干网络的插件（例如，Inception-ResNet-v2）可以带来最先进的去模糊处理。同时，凭借轻量级主干网络（例如 MobileNet 及其变体），DeblurGAN-v2 的速度比最接近的竞争对手快 10-100 倍，同时保持接近最先进的结果，这意味着可用于实时视频去模糊。我们证明

了 DeblurGAN-v2 在几个流行的基准测试中获得了非常有竞争力的性能，包括去模糊质量（客观和主观）以及效率。此外，我们还展示了该架构对于一般图像恢复任务也依然有效。

24.2.2 结果与模型

24.2.3 快速开始

训练

您可以使用以下命令来训练模型。

```
## cpu train
CUDA_VISIBLE_DEVICES=-1 python tools/train.py configs/deblurganv2/deblurganv2_fpn-
↪inception_1xb1_gopro.py

## single-gpu train
python tools/train.py configs/deblurganv2/deblurganv2_fpn-inception_1xb1_gopro.py

## multi-gpu train
./tools/dist_train.sh configs/deblurganv2/deblurganv2_fpn-inception_1xb1_gopro.py 8
```

更多细节可以参考 train_test.md 中的 **Train a model** 部分。

测试

您可以使用以下命令来测试模型。

```
## cpu test
CUDA_VISIBLE_DEVICES=-1 python tools/test.py configs/deblurganv2/deblurganv2_fpn-
↪inception_1xb1_gopro.py https://download.openxlab.org.cn/models/xiaomile/
↪DeblurGANv2/weight/DeblurGANv2_fpn-inception.pth

## single-gpu test
python tools/test.py configs/deblurganv2/deblurganv2_fpn-inception_1xb1_gopro.py␣
↪https://download.openxlab.org.cn/models/xiaomile/DeblurGANv2/weight/DeblurGANv2_fpn-
↪inception.pth

## multi-gpu test
./tools/dist_test.sh configs/deblurganv2/deblurganv2_fpn-inception_1xb1_gopro.py␣
↪https://download.openxlab.org.cn/models/xiaomile/DeblurGANv2/weight/DeblurGANv2_fpn-
↪inception.pth 8
```

更多细节可以参考 train_test.md 中的 **Test a pre-trained model** 部分。

24.2.4 引用

```
@InProceedings{Kupyn_2019_ICCV,
author = {Orest Kupyn and Tetiana Martyniuk and Junru Wu and Zhangyang Wang},
title = {DeblurGAN-v2: Deblurring (Orders-of-Magnitude) Faster and Better},
booktitle = {The IEEE International Conference on Computer Vision (ICCV)},
month = {Oct},
year = {2019}
}
```

CHAPTER 25

图像修复

25.1 概览

- 预训练权重个数: 9
- 配置文件个数: 0
- 论文个数: 5
  - ALGORITHM: 5

25.2 AOT-GAN (TVCG’2021)

AOT-GAN: Aggregated Contextual Transformations for High-Resolution Image Inpainting

任务: 图像修复

25.2.1 摘要

25.2.2 结果与模型

Places365-Challenge

25.2.3 快速开始

训练

您可以使用以下命令来训练模型。

```
## CPU上训练
CUDA_VISIBLE_DEVICES=-1 python tools/train.py configs/aot_gan/aot-gan_smpgan_4xb4_
↪places-512x512.py

## 单个GPU上训练
python tools/train.py configs/aot_gan/aot-gan_smpgan_4xb4_places-512x512.py

## 多个GPU上训练
./tools/dist_train.sh configs/aot_gan/aot-gan_smpgan_4xb4_places-512x512.py 8
```

更多细节可以参考 train_test.md 中的 **Train a model** 部分。

测试

您可以使用以下命令来测试模型。

```
## CPU上测试
CUDA_VISIBLE_DEVICES=-1 python tools/test.py configs/aot_gan/aot-gan_smpgan_4xb4_
↪places-512x512.py https://download.openmmlab.com/mmediting/inpainting/aot_gan/AOT-
↪GAN_512x512_4x12_places_20220509-6641441b.pth

## 单个GPU上测试
python tools/test.py configs/aot_gan/aot-gan_smpgan_4xb4_places-512x512.py https://
↪download.openmmlab.com/mmediting/inpainting/aot_gan/AOT-GAN_512x512_4x12_places_
↪20220509-6641441b.pth

## 多个GPU上测试
./tools/dist_test.sh configs/aot_gan/aot-gan_smpgan_4xb4_places-512x512.py https://
↪download.openmmlab.com/mmediting/inpainting/aot_gan/AOT-GAN_512x512_4x12_places_
↪20220509-6641441b.pth 8
```

更多细节可以参考 train_test.md 中的 **Test a pre-trained model** 部分。

25.2.4 引用

```
@inproceedings{yan2021agg,
  author = {Zeng, Yanhong and Fu, Jianlong and Chao, Hongyang and Guo, Baining},
  title = {Aggregated Contextual Transformations for High-Resolution Image Inpainting}
↪,
  booktitle = {Arxiv},
  pages={-},
  year = {2020}
}
```

25.3 DeepFillv2 (CVPR’2019)

任务: 图像修复

```
@inproceedings{yu2019free,
  title={Free-form image inpainting with gated convolution},
  author={Yu, Jiahui and Lin, Zhe and Yang, Jimei and Shen, Xiaohui and Lu, Xin and␣
↪Huang, Thomas S},
  booktitle={Proceedings of the IEEE International Conference on Computer Vision},
  pages={4471--4480},
  year={2019}
}
```

CelebA-HQ

Places365-Challenge

25.3.1 快速开始

训练

您可以使用以下命令来训练模型。

```
## CPU上训练
CUDA_VISIBLE_DEVICES=-1 python tools/train.py configs/deepfillv2/deepfillv2_8xb2_
↪places-256x256.py

## 单个GPU上训练
python tools/train.py configs/deepfillv2/deepfillv2_8xb2_places-256x256.py

## 多个GPU上训练
./tools/dist_train.sh configs/deepfillv2/deepfillv2_8xb2_places-256x256.py 8
```

更多细节可以参考 train_test.md 中的 **Train a model** 部分。

测试

您可以使用以下命令来测试模型。

```
## CPU上测试
CUDA_VISIBLE_DEVICES=-1 python tools/test.py configs/deepfillv2/deepfillv2_8xb2_
↪places-256x256.py https://download.openmmlab.com/mmediting/inpainting/deepfillv2/
↪deepfillv2_256x256_8x2_places_20200619-10d15793.pth

## 单个GPU上测试
python tools/test.py configs/deepfillv2/deepfillv2_8xb2_places-256x256.py https://
↪download.openmmlab.com/mmediting/inpainting/deepfillv2/deepfillv2_256x256_8x2_
↪places_20200619-10d15793.pth

## 多个GPU上测试
./tools/dist_test.sh configs/deepfillv2/deepfillv2_8xb2_places-256x256.py https://
↪download.openmmlab.com/mmediting/inpainting/deepfillv2/deepfillv2_256x256_8x2_
↪places_20200619-10d15793.pth 8
```

更多细节可以参考 train_test.md 中的 **Test a pre-trained model** 部分。

25.4 PConv (ECCV'2018)

> **任务**: 图像修复

```
@inproceedings{liu2018image,
  title={Image inpainting for irregular holes using partial convolutions},
  author={Liu, Guilin and Reda, Fitsum A and Shih, Kevin J and Wang, Ting-Chun and␣
↪Tao, Andrew and Catanzaro, Bryan},
  booktitle={Proceedings of the European Conference on Computer Vision (ECCV)},
  pages={85--100},
  year={2018}
}
```

Places365-Challenge

CelebA-HQ

25.4.1 快速开始

训练

您可以使用以下命令来训练模型。

```
## CPU上训练
CUDA_VISIBLE_DEVICES=-1 python tools/train.py configs/partial_conv/pconv_stage2_4xb2_
↪places-256x256.py

## 单个GPU上训练
python tools/train.py configs/partial_conv/pconv_stage2_4xb2_places-256x256.py

## 多个GPU上训练
./tools/dist_train.sh configs/partial_conv/pconv_stage2_4xb2_places-256x256.py 8
```

更多细节可以参考 train_test.md 中的 **Train a model** 部分。

测试

您可以使用以下命令来测试模型。

```
## CPU上测试
CUDA_VISIBLE_DEVICES=-1 python tools/test.py configs/partial_conv/pconv_stage2_4xb2_
↪places-256x256.py https://download.openmmlab.com/mmediting/inpainting/pconv/pconv_
↪256x256_stage2_4x2_places_20200619-1ffed0e8.pth

## 单个GPU上测试
python tools/test.py configs/partial_conv/pconv_stage2_4xb2_places-256x256.py https://
↪download.openmmlab.com/mmediting/inpainting/pconv/pconv_256x256_stage2_4x2_places_
↪20200619-1ffed0e8.pth

## 多个GPU上测试
./tools/dist_test.sh configs/partial_conv/pconv_stage2_4xb2_places-256x256.py https://
↪download.openmmlab.com/mmediting/inpainting/pconv/pconv_256x256_stage2_4x2_places_
↪20200619-1ffed0e8.pth 8
```

更多细节可以参考 train_test.md 中的 **Test a pre-trained model** 部分。

25.5 DeepFillv1 (CVPR’2018)

任务: 图像修复

```
@inproceedings{yu2018generative,
  title={Generative image inpainting with contextual attention},
  author={Yu, Jiahui and Lin, Zhe and Yang, Jimei and Shen, Xiaohui and Lu, Xin and␣
↪Huang, Thomas S},
  booktitle={Proceedings of the IEEE conference on computer vision and pattern␣
↪recognition},
  pages={5505--5514},
  year={2018}
}
```

CelebA-HQ

Places365-Challenge

25.5.1 快速开始

训练

您可以使用以下命令来训练模型。

```
## CPU上训练
CUDA_VISIBLE_DEVICES=-1 python tools/train.py configs/deepfillv1/deepfillv1_8xb2_
↪places-256x256.py

## 单个GPU上训练
python tools/train.py configs/deepfillv1/deepfillv1_8xb2_places-256x256.py

## 多个GPU上训练
./tools/dist_train.sh configs/deepfillv1/deepfillv1_8xb2_places-256x256.py 8
```

更多细节可以参考 train_test.md 中的 **Train a model** 部分。

测试

您可以使用以下命令来测试模型。

```
## CPU上测试
CUDA_VISIBLE_DEVICES=-1 python tools/test.py configs/deepfillv1/deepfillv1_8xb2_
↪places-256x256.py https://download.openmmlab.com/mmediting/inpainting/deepfillv1/
↪deepfillv1_256x256_8x2_places_20200619-c00a0e21.pth

## 单个GPU上测试
```

(下页继续)

(续上页)

```
python tools/test.py configs/deepfillv1/deepfillv1_8xb2_places-256x256.py https://
↪download.openmmlab.com/mmediting/inpainting/deepfillv1/deepfillv1_256x256_8x2_
↪places_20200619-c00a0e21.pth

## 多个GPU上测试
./tools/dist_test.sh configs/deepfillv1/deepfillv1_8xb2_places-256x256.py https://
↪download.openmmlab.com/mmediting/inpainting/deepfillv1/deepfillv1_256x256_8x2_
↪places_20200619-c00a0e21.pth 8
```

更多细节可以参考 train_test.md 中的 **Test a pre-trained model** 部分。

25.6 Global&Local (ToG'2017)

任务: 图像修复

```
@article{iizuka2017globally,
  title={Globally and locally consistent image completion},
  author={Iizuka, Satoshi and Simo-Serra, Edgar and Ishikawa, Hiroshi},
  journal={ACM Transactions on Graphics (ToG)},
  volume={36},
  number={4},
  pages={1--14},
  year={2017},
  publisher={ACM New York, NY, USA}
}
```

请注意，为了与当前的深度图像修复方法进行公平比较，我们没有在 *Global&Local* 中使用后处理模块。

Places365-Challenge

CelebA-HQ

25.6.1 快速开始

训练

您可以使用以下命令来训练模型。

```
## CPU上训练
CUDA_VISIBLE_DEVICES=-1 python tools/train.py configs/global_local/gl_8xb12_places-
↪256x256.py

## 单个GPU上训练
```

(下页继续)

(续上页)

```
python tools/train.py configs/global_local/gl_8xb12_places-256x256.py

## 多个GPU上训练
./tools/dist_train.sh configs/global_local/gl_8xb12_places-256x256.py 8
```

更多细节可以参考 train_test.md 中的 **Train a model** 部分。

测试

您可以使用以下命令来测试模型。

```
## CPU上测试
CUDA_VISIBLE_DEVICES=-1 python tools/test.py configs/global_local/gl_8xb12_places-
↪256x256.py https://download.openmmlab.com/mmediting/inpainting/global_local/gl_
↪256x256_8x12_places_20200619-52a040a8.pth

## 单个GPU上测试
python tools/test.py configs/global_local/gl_8xb12_places-256x256.py https://download.
↪openmmlab.com/mmediting/inpainting/global_local/gl_256x256_8x12_places_20200619-
↪52a040a8.pth

## 多个GPU上测试
./tools/dist_test.sh configs/global_local/gl_8xb12_places-256x256.py https://download.
↪openmmlab.com/mmediting/inpainting/global_local/gl_256x256_8x12_places_20200619-
↪52a040a8.pth 8
```

更多细节可以参考 train_test.md 中的 **Test a pre-trained model** 部分。

CHAPTER 26

视频超分辨率

26.1 概览

- 预训练权重个数: 25
- 配置文件个数: 0
- 论文个数: 6
  - ALGORITHM: 7

26.2 RealBasicVSR (CVPR’2022)

任务: 视频超分辨率

```
@InProceedings{chan2022investigating,
  author = {Chan, Kelvin C.K. and Zhou, Shangchen and Xu, Xiangyu and Loy, Chen␣
↪Change},
  title = {RealBasicVSR: Investigating Tradeoffs in Real-World Video Super-Resolution}
↪,
  booktitle = {Proceedings of the IEEE conference on computer vision and pattern␣
↪recognition},
  year = {2022}
}
```

在 Y 通道上评估。计算 NRQM、NIQE 和 PI 的代码可以在这里找到。我们使用 MATLAB 官方代码计算 BRISQUE。

26.2.1 训练

训练分为两个阶段：

1. 使用 realbasicvsr_wogan-c64b20-2x30x8_8xb2-lr1e-4-300k_reds.py 训练一个没有感知损失和对抗性损失的模型。
2. 使用感知损失和对抗性损失 realbasicvsr_c64b20-1x30x8_8xb1-lr5e-5-150k_reds.py 微调模型。

注:

1. 您可能希望将图像裁剪为子图像以加快 IO。请参阅此处了解更多详情。

26.2.2 快速开始

训练

您可以使用以下命令来训练模型。

```
## CPU上训练
CUDA_VISIBLE_DEVICES=-1 python tools/train.py configs/real_basicvsr/realbasicvsr_
↪c64b20-1x30x8_8xb1-lr5e-5-150k_reds.py

## 单个GPU上训练
python tools/train.py configs/real_basicvsr/realbasicvsr_c64b20-1x30x8_8xb1-lr5e-5-
↪150k_reds.py

## 多个GPU上训练
./tools/dist_train.sh configs/real_basicvsr/realbasicvsr_c64b20-1x30x8_8xb1-lr5e-5-
↪150k_reds.py 8
```

更多细节可以参考 train_test.md 中的 **Train a model** 部分。

测试

您可以使用以下命令来测试模型。

```
## CPU上测试
CUDA_VISIBLE_DEVICES=-1 python tools/test.py configs/real_basicvsr/realbasicvsr_
↪c64b20-1x30x8_8xb1-lr5e-5-150k_reds.py https://download.openmmlab.com/mmediting/
↪restorers/real_basicvsr/realbasicvsr_c64b20_1x30x8_lr5e-5_150k_reds_20211104-
↪52f77c2c.pth

## 单个GPU上测试
```

(下页继续)

(续上页)

```
python tools/test.py configs/real_basicvsr/realbasicvsr_c64b20-1x30x8_8xb1-lr5e-5-
↪150k_reds.py https://download.openmmlab.com/mmediting/restorers/real_basicvsr/
↪realbasicvsr_c64b20_1x30x8_lr5e-5_150k_reds_20211104-52f77c2c.pth

## 多个GPU上测试
./tools/dist_test.sh configs/real_basicvsr/realbasicvsr_c64b20-1x30x8_8xb1-lr5e-5-
↪150k_reds.py https://download.openmmlab.com/mmediting/restorers/real_basicvsr/
↪realbasicvsr_c64b20_1x30x8_lr5e-5_150k_reds_20211104-52f77c2c.pth 8
```

更多细节可以参考 train_test.md 中的 **Test a pre-trained model** 部分。

26.3 BasicVSR++ (CVPR'2022)

任务: 视频超分辨率

```
@InProceedings{chan2022basicvsrplusplus,
  author = {Chan, Kelvin C.K. and Zhou, Shangchen and Xu, Xiangyu and Loy, Chen␣
↪Change},
  title = {BasicVSR++: Improving Video Super-Resolution with Enhanced Propagation and␣
↪Alignment},
  booktitle = {Proceedings of the IEEE conference on computer vision and pattern␣
↪recognition},
  year = {2022}
}
```

SPyNet 的预训练权重在这里。

请注意，以下模型是从较小的模型中微调而来的。这些模型的训练方案将在 MMagic 达到 5k star 时发布。我们在这里提供预训练的模型。

26.3.1 快速开始

训练

您可以使用以下命令来训练模型。

```
## CPU上训练
CUDA_VISIBLE_DEVICES=-1 python tools/train.py configs/basicvsr_pp/basicvsr-pp_c64n7_
↪8xb1-600k_reds4.py

## 单个GPU上训练
python tools/train.py configs/basicvsr_pp/basicvsr-pp_c64n7_8xb1-600k_reds4.py
```

(下页继续)

(续上页)

```
## 多个GPU上训练
./tools/dist_train.sh configs/basicvsr_pp/basicvsr-pp_c64n7_8xb1-600k_reds4.py 8
```

更多细节可以参考 train_test.md 中的 **Train a model** 部分。

测试

您可以使用以下命令来测试模型。

```
## CPU上测试
CUDA_VISIBLE_DEVICES=-1 python tools/test.py configs/basicvsr_pp/basicvsr-pp_c64n7_
↪8xb1-600k_reds4.py https://download.openmmlab.com/mmediting/restorers/basicvsr_
↪plusplus/basicvsr_plusplus_c64n7_8x1_600k_reds4_20210217-db622b2f.pth

## 单个GPU上测试
python tools/test.py configs/basicvsr_pp/basicvsr-pp_c64n7_8xb1-600k_reds4.py https://
↪download.openmmlab.com/mmediting/restorers/basicvsr_plusplus/basicvsr_plusplus_
↪c64n7_8x1_600k_reds4_20210217-db622b2f.pth

## 多个GPU上测试
./tools/dist_test.sh configs/basicvsr_pp/basicvsr-pp_c64n7_8xb1-600k_reds4.py https://
↪download.openmmlab.com/mmediting/restorers/basicvsr_plusplus/basicvsr_plusplus_
↪c64n7_8x1_600k_reds4_20210217-db622b2f.pth 8
```

更多细节可以参考 train_test.md 中的 **Test a pre-trained model** 部分。

26.4 IconVSR (CVPR'2021)

任务: 视频超分辨率

```
@InProceedings{chan2021basicvsr,
  author = {Chan, Kelvin CK and Wang, Xintao and Yu, Ke and Dong, Chao and Loy, Chen␣
↪Change},
  title = {BasicVSR: The Search for Essential Components in Video Super-Resolution␣
↪and Beyond},
  booktitle = {Proceedings of the IEEE conference on computer vision and pattern␣
↪recognition},
  year = {2021}
}
```

对于 REDS4，我们对 RGB 通道进行评估。对于其他数据集，我们对 Y 通道进行评估。我们使用 PSNR 和 SSIM 作为指标。IconVSR 组件的预训练权重可以在这里找到：SPyNet，用于 REDS 的 EDVR-M，以及 用于 Vimeo-90K 的 EDVR-M。

26.4.1 快速开始

训练

您可以使用以下命令来训练模型。

```
## CPU上训练
CUDA_VISIBLE_DEVICES=-1 python tools/train.py configs/iconvsr/iconvsr_2xb4_reds4.py

## 单个GPU上训练
python tools/train.py configs/iconvsr/iconvsr_2xb4_reds4.py

## 多个GPU上训练
./tools/dist_train.sh configs/iconvsr/iconvsr_2xb4_reds4.py 8
```

更多细节可以参考 train_test.md 中的 **Train a model** 部分。

测试

您可以使用以下命令来测试模型。

```
## CPU上测试
CUDA_VISIBLE_DEVICES=-1 python tools/test.py configs/iconvsr/iconvsr_2xb4_reds4.py␣
↪https://download.openmmlab.com/mmediting/restorers/iconvsr/iconvsr_reds4_20210413-
↪9e09d621.pth

## 单个GPU上测试
python tools/test.py configs/iconvsr/iconvsr_2xb4_reds4.py https://download.openmmlab.
↪com/mmediting/restorers/iconvsr/iconvsr_reds4_20210413-9e09d621.pth

## 多个GPU上测试
./tools/dist_test.sh configs/iconvsr/iconvsr_2xb4_reds4.py https://download.openmmlab.
↪com/mmediting/restorers/iconvsr/iconvsr_reds4_20210413-9e09d621.pth 8
```

更多细节可以参考 train_test.md 中的 **Test a pre-trained model** 部分。

26.5 BasicVSR (CVPR’2021)

> **任务**: 视频超分辨率

```
@InProceedings{chan2021basicvsr,
  author = {Chan, Kelvin CK and Wang, Xintao and Yu, Ke and Dong, Chao and Loy, Chen␣
↪Change},
  title = {BasicVSR: The Search for Essential Components in Video Super-Resolution␣
↪and Beyond},
```

(下页继续)

(续上页)

```
  booktitle = {Proceedings of the IEEE conference on computer vision and pattern␣
↪recognition},
  year = {2021}
}
```

对于 REDS4，我们对 RGB 通道进行评估。对于其他数据集，我们对 Y 通道进行评估。我们使用 PSNR 和 SSIM 作为指标。SPyNet 的预训练权重在这里。

26.5.1 快速开始

训练

您可以使用以下命令来训练模型。

```
## CPU上训练
CUDA_VISIBLE_DEVICES=-1 python tools/train.py configs/basicvsr/basicvsr_2xb4_reds4.py

## 单个GPU上训练
python tools/train.py configs/basicvsr/basicvsr_2xb4_reds4.py

## 多个GPU上训练
./tools/dist_train.sh configs/basicvsr/basicvsr_2xb4_reds4.py 8
```

更多细节可以参考 train_test.md 中的 **Train a model** 部分。

测试

您可以使用以下命令来测试模型。

```
## CPU上测试
CUDA_VISIBLE_DEVICES=-1 python tools/test.py configs/basicvsr/basicvsr_2xb4_reds4.py␣
↪https://download.openmmlab.com/mmediting/restorers/basicvsr/basicvsr_reds4_20120409-
↪0e599677.pth

## 单个GPU上测试
python tools/test.py configs/basicvsr/basicvsr_2xb4_reds4.py https://download.
↪openmmlab.com/mmediting/restorers/basicvsr/basicvsr_reds4_20120409-0e599677.pth

## 多个GPU上测试
./tools/dist_test.sh configs/basicvsr/basicvsr_2xb4_reds4.py https://download.
↪openmmlab.com/mmediting/restorers/basicvsr/basicvsr_reds4_20120409-0e599677.pth 8
```

更多细节可以参考 train_test.md 中的 **Test a pre-trained model** 部分。

26.6 TDAN (CVPR’2020)

任务: 视频超分辨率

```
@InProceedings{tian2020tdan,
  title={TDAN: Temporally-Deformable Alignment Network for Video Super-Resolution},
  author={Tian, Yapeng and Zhang, Yulun and Fu, Yun and Xu, Chenliang},
  booktitle = {Proceedings of the IEEE conference on Computer Vision and Pattern␣
↪Recognition},
  year = {2020}
}
```

在 RGB 通道上进行评估，在评估之前裁剪每个边界中的 8 像素。我们使用 PSNR 和 SSIM 作为指标。

26.6.1 快速开始

训练

您可以使用以下命令来训练模型。

TDAN 训练有两个阶段。

阶段 1: 以更大的学习率训练 (1e-4)

```
## CPU上训练
CUDA_VISIBLE_DEVICES=-1 python tools/train.py configs/tdan/tdan_x4_1xb16-lr1e-4-400k_
↪vimeo90k-bi.py

## 单个GPU上训练
python tools/train.py configs/tdan/tdan_x4_1xb16-lr1e-4-400k_vimeo90k-bi.py

## 多个GPU上训练
./tools/dist_train.sh configs/tdan/tdan_x4_1xb16-lr1e-4-400k_vimeo90k-bi.py 8
```

阶段 2: 以较小的学习率进行微调 (5e-5)

```
## CPU上训练
CUDA_VISIBLE_DEVICES=-1 python tools/train.py configs/tdan/tdan_x4ft_1xb16-lr5e-5-
↪400k_vimeo90k-bi.py

## 单个GPU上训练
python tools/train.py configs/tdan/tdan_x4ft_1xb16-lr5e-5-400k_vimeo90k-bi.py

## 多个GPU上训练
./tools/dist_train.sh configs/tdan/tdan_x4ft_1xb16-lr5e-5-400k_vimeo90k-bi.py 8
```

更多细节可以参考 train_test.md 中的 **Train a model** 部分。

测试

您可以使用以下命令来测试模型。

```
## CPU上测试
CUDA_VISIBLE_DEVICES=-1 python tools/test.py configs/tdan/tdan_x4ft_1xb16-lr5e-5-400k_
↪vimeo90k-bi.py https://download.openmmlab.com/mmediting/restorers/tdan/tdan_
↪vimeo90k_bix4_20210528-739979d9.pth

## 单个GPU上测试
python tools/test.py configs/tdan/tdan_x4ft_1xb16-lr5e-5-400k_vimeo90k-bi.py https://
↪download.openmmlab.com/mmediting/restorers/tdan/tdan_vimeo90k_bix4_20210528-
↪739979d9.pth

## 多个GPU上测试
./tools/dist_test.sh configs/tdan/tdan_x4ft_1xb16-lr5e-5-400k_vimeo90k-bi.py https://
↪download.openmmlab.com/mmediting/restorers/tdan/tdan_vimeo90k_bix4_20210528-
↪739979d9.pth 8
```

更多细节可以参考 train_test.md 中的 **Test a pre-trained model** 部分。

26.7 TOFlow (IJCV'2019)

> Video Enhancement with Task-Oriented Flow
>
> **任务**: 视频插帧, 视频超分辨率

26.7.1 预训练模型测试结果

在 RGB 通道上评估。评估指标 `PSNR / SSIM`。

注: 由于 `batch_size=1` 预训练的 SPyNet 不包含 BN 层，这与 `https://github.com/Coldog2333/pytoflow` 一致.

26.7.2 快速开始

训练

您可以使用以下命令来训练模型。

TOF 的训练仅支持视频插帧任务。

```
## CPU上训练
CUDA_VISIBLE_DEVICES=-1 python tools/train.py configs/tof/tof_spynet-chair-wobn_1xb1_
↪vimeo90k-triplet.py

## 单个GPU上训练
python tools/train.py configs/tof/tof_spynet-chair-wobn_1xb1_vimeo90k-triplet.py

## 多个GPU上训练
./tools/dist_train.sh configs/tof/tof_spynet-chair-wobn_1xb1_vimeo90k-triplet.py 8
```

更多细节可以参考 train_test.md 中的 **Train a model** 部分。

测试

您可以使用以下命令来测试模型。

TOF 的测试支持视频插帧和视频超分辨率两种任务。

任务 1: 视频插帧

```
## CPU上测试
CUDA_VISIBLE_DEVICES=-1 python tools/test.py configs/tof/tof_spynet-chair-wobn_1xb1_
↪vimeo90k-triplet.py https://download.openmmlab.com/mmediting/video_interpolators/
↪toflow/pretrained_spynet_chair_20220321-4d82e91b.pth

## 单个GPU上测试
python tools/test.py configs/tof/tof_spynet-chair-wobn_1xb1_vimeo90k-triplet.py␣
↪https://download.openmmlab.com/mmediting/video_interpolators/toflow/pretrained_
↪spynet_chair_20220321-4d82e91b.pth

## 多个GPU上测试
./tools/dist_test.sh configs/tof/tof_spynet-chair-wobn_1xb1_vimeo90k-triplet.py␣
↪https://download.openmmlab.com/mmediting/video_interpolators/toflow/pretrained_
↪spynet_chair_20220321-4d82e91b.pth 8
```

任务 2: 视频超分辨率

```
## CPU上测试
CUDA_VISIBLE_DEVICES=-1 python tools/test.py configs/tof/tof_x4_official_vimeo90k.py␣
↪https://download.openmmlab.com/mmediting/restorers/tof/tof_x4_vimeo90k_official-
↪a569ff50.pth

## 单个GPU上测试
python tools/test.py configs/tof/tof_x4_official_vimeo90k.py https://download.
↪openmmlab.com/mmediting/restorers/tof/tof_x4_vimeo90k_official-a569ff50.pth
```

(下页继续)

(续上页)

```
## 多个GPU上测试
./tools/dist_test.sh configs/tof/tof_x4_official_vimeo90k.py https://download.
↪openmmlab.com/mmediting/restorers/tof/tof_x4_vimeo90k_official-a569ff50.pth 8
```

更多细节可以参考 train_test.md 中的 **Test a pre-trained model** 部分。

26.7.3 Citation

```
@article{xue2019video,
  title={Video enhancement with task-oriented flow},
  author={Xue, Tianfan and Chen, Baian and Wu, Jiajun and Wei, Donglai and Freeman,␣
↪William T},
  journal={International Journal of Computer Vision},
  volume={127},
  number={8},
  pages={1106--1125},
  year={2019},
  publisher={Springer}
}
```

26.8 EDVR (CVPRW'2019)

任务: 视频超分辨率

```
@InProceedings{wang2019edvr,
  author    = {Wang, Xintao and Chan, Kelvin C.K. and Yu, Ke and Dong, Chao and Loy,␣
↪Chen Change},
  title     = {EDVR: Video restoration with enhanced deformable convolutional␣
↪networks},
  booktitle = {The IEEE Conference on Computer Vision and Pattern Recognition␣
↪Workshops (CVPRW)},
  month     = {June},
  year      = {2019},
}
```

在 RGB 通道上进行评估。我们使用 PSNR 和 SSIM 作为指标。

26.8.1 快速开始

训练

您可以使用以下命令来训练模型。

```
## CPU上训练
CUDA_VISIBLE_DEVICES=-1 python tools/train.py configs/edvr/edvrm_8xb4-600k_reds.py

## 单个GPU上训练
python tools/train.py configs/edvr/edvrm_8xb4-600k_reds.py

## 多个GPU上训练
./tools/dist_train.sh configs/edvr/edvrm_8xb4-600k_reds.py 8
```

更多细节可以参考 train_test.md 中的 **Train a model** 部分。

测试

您可以使用以下命令来测试模型。

```
## CPU上测试
CUDA_VISIBLE_DEVICES=-1 python tools/test.py configs/edvr/edvrm_8xb4-600k_reds.py␣
↪https://download.openmmlab.com/mmediting/restorers/edvr/edvrm_x4_8x4_600k_reds_
↪20210625-e29b71b5.pth

## 单个GPU上测试
python tools/test.py configs/edvr/edvrm_8xb4-600k_reds.py https://download.openmmlab.
↪com/mmediting/restorers/edvr/edvrm_x4_8x4_600k_reds_20210625-e29b71b5.pth

## 多个GPU上测试
./tools/dist_test.sh configs/edvr/edvrm_8xb4-600k_reds.py https://download.openmmlab.
↪com/mmediting/restorers/edvr/edvrm_x4_8x4_600k_reds_20210625-e29b71b5.pth 8
```

更多细节可以参考 train_test.md 中的 **Test a pre-trained model** 部分。

CHAPTER 27

扩散模型

27.1 概览

- 预训练权重个数: 2
- 配置文件个数: 0
- 论文个数: 1
  - ALGORITHM: 1

27.2 Disco Diffusion (2022)

Disco Diffusion

任务: 图文生成, 图像到图像的翻译, 扩散模型

27.2.1 摘要

Disco Diffusion(DD) 是一个 Google Colab 笔记本，它利用一种叫做 CLIP-Guided Diffusion 的人工智能图像生成技术，让你从文本输入中创造出引人注目的精美图像。

由 Somnai 创建，由 Gandamu 改进，并建立在 RiversHaveWings、nshepperd 和许多其他人的工作之上。更多细节见 Credits。

27.2.2 模型与结果

我们已经转换了几个 unet 的权重，并提供相关的配置文件。在 Tutorial 中可以看到更多关于不同 unet 的细节。

27.2.3 待办列表

- [x] 图文生成
- [x] 图像到图像的翻译
- [x] Imagenet, portrait 扩散模型
- [] 像素艺术，水彩，科幻小说的扩散模型
- [] 支持图像提示
- [] 支持视频生成
- [] 支持更快的采样器 (plms，dpm-solver 等)

我们很欢迎社区用户支持这些项目和任何其他有趣的工作!

27.2.4 Quick Start

运行以下代码，你可以使用文本生成图像。

```
from mmengine import Config, MODELS
from mmagic.utils import register_all_modules
from torchvision.utils import save_image

register_all_modules()

disco = MODELS.build(
    Config.fromfile('configs/disco_diffusion/disco-baseline.py').model).cuda().eval()
text_prompts = {
    0: [
        "A beautiful painting of a singular lighthouse, shining its light across a␣
↪tumultuous sea of blood by greg rutkowski and thomas kinkade, Trending on␣
↪artstation.",
        "yellow color scheme"
    ]
}
image = disco.infer(
    height=768,
    width=1280,
    text_prompts=text_prompts,
```

(下页继续)

(续上页)

```
    show_progress=True,
    num_inference_steps=250,
    eta=0.8)['samples']
save_image(image, "image.png")
```

27.2.5 教程

考虑到 `disco-diffusion` 包含许多可调整的参数，我们为用户提供了一个 jupyter-notebook/colab的教程，展示了不同参数的含义，并给出相应的调整结果。请参考Disco Sheet。

27.2.6 鸣谢

Since our adaptation of disco-diffusion are heavily influenced by disco colab, here we copy the credits below.

Modified by Daniel Russell (https://github.com/russelldc, https://twitter.com/danielrussruss) to include (hopefully) optimal params for quick generations in 15-100 timesteps rather than 1000, as well as more robust augmentations.

Further improvements from Dango233 and nshepperd helped improve the quality of diffusion in general, and especially so for shorter runs like this notebook aims to achieve.

Vark added code to load in multiple Clip models at once, which all prompts are evaluated against, which may greatly improve accuracy.

The latest zoom, pan, rotation, and keyframes features were taken from Chigozie Nri' s VQGAN Zoom Notebook (https://github.com/chigozienri, https://twitter.com/chigozienri)

Advanced DangoCutn Cutout method is also from Dango223.

–

Disco:

Somnai (https://twitter.com/Somnai_dreams) added Diffusion Animation techniques, QoL improvements and various implementations of tech and techniques, mostly listed in the changelog below.

3D animation implementation added by Adam Letts (https://twitter.com/gandamu_ml) in collaboration with Somnai. Creation of disco.py and ongoing maintenance.

Turbo feature by Chris Allen (https://twitter.com/zippy731)

Improvements to ability to run on local systems, Windows support, and dependency installation by HostsServer (https://twitter.com/HostsServer)

VR Mode by Tom Mason (https://twitter.com/nin_artificial)

Horizontal and Vertical symmetry functionality by nshepperd. Symmetry transformation_steps by huemin (https://twitter.com/huemin_art). Symmetry integration into Disco Diffusion by Dmitrii Tochilkin (https://twitter.com/cut_pow).

Warp and custom model support by Alex Spirin (https://twitter.com/devdef).

Pixel Art Diffusion, Watercolor Diffusion, and Pulp SciFi Diffusion models from KaliYuga (https://twitter.com/KaliYuga_ai). Follow KaliYuga's Twitter for the latest models and for notebooks with specialized settings.

Integration of OpenCLIP models and initiation of integration of KaliYuga models by Palmweaver / Chris Scalf (https://twitter.com/ChrisScalf11)

Integrated portrait_generator_v001 from Felipe3DArtist (https://twitter.com/Felipe3DArtist)

27.2.7 Citation

```
@misc{github,
  author={alembics},
  title={disco-diffusion},
  year={2022},
  url={https://github.com/alembics/disco-diffusion},
}
```

CHAPTER 28

图像到图像的翻译

28.1 概览

- 预训练权重个数: 2
- 配置文件个数: 0
- 论文个数: 1
  - ALGORITHM: 1

28.2 Disco Diffusion (2022)

Disco Diffusion

任务: 图文生成, 图像到图像的翻译, 扩散模型

28.2.1 摘要

Disco Diffusion(DD) 是一个 Google Colab 笔记本，它利用一种叫做 CLIP-Guided Diffusion 的人工智能图像生成技术，让你从文本输入中创造出引人注目的精美图像。

由 Somnai 创建，由 Gandamu 改进，并建立在 RiversHaveWings、nshepperd 和许多其他人的工作之上。更多细节见 Credits。

28.2.2 模型与结果

我们已经转换了几个 unet 的权重，并提供相关的配置文件。在 Tutorial 中可以看到更多关于不同 unet 的细节。

28.2.3 待办列表

- [x] 图文生成
- [x] 图像到图像的翻译
- [x] Imagenet, portrait 扩散模型
- [] 像素艺术，水彩，科幻小说的扩散模型
- [] 支持图像提示
- [] 支持视频生成
- [] 支持更快的采样器 (plms，dpm-solver 等)

我们很欢迎社区用户支持这些项目和任何其他有趣的工作!

28.2.4 Quick Start

运行以下代码，你可以使用文本生成图像。

```
from mmengine import Config, MODELS
from mmagic.utils import register_all_modules
from torchvision.utils import save_image

register_all_modules()

disco = MODELS.build(
    Config.fromfile('configs/disco_diffusion/disco-baseline.py').model).cuda().eval()
text_prompts = {
    0: [
        "A beautiful painting of a singular lighthouse, shining its light across a␣
↪tumultuous sea of blood by greg rutkowski and thomas kinkade, Trending on␣
↪artstation.",
        "yellow color scheme"
    ]
}
image = disco.infer(
    height=768,
    width=1280,
    text_prompts=text_prompts,
```

(下页继续)

(续上页)

```
    show_progress=True,
    num_inference_steps=250,
    eta=0.8)['samples']
save_image(image, "image.png")
```

28.2.5 教程

考虑到 `disco-diffusion` 包含许多可调整的参数，我们为用户提供了一个 jupyter-notebook/colab的教程，展示了不同参数的含义，并给出相应的调整结果。请参考Disco Sheet。

28.2.6 鸣谢

Since our adaptation of disco-diffusion are heavily influenced by disco colab, here we copy the credits below.

Modified by Daniel Russell (https://github.com/russelldc, https://twitter.com/danielrussruss) to include (hopefully) optimal params for quick generations in 15-100 timesteps rather than 1000, as well as more robust augmentations.

Further improvements from Dango233 and nshepperd helped improve the quality of diffusion in general, and especially so for shorter runs like this notebook aims to achieve.

Vark added code to load in multiple Clip models at once, which all prompts are evaluated against, which may greatly improve accuracy.

The latest zoom, pan, rotation, and keyframes features were taken from Chigozie Nri' s VQGAN Zoom Notebook (https://github.com/chigozienri, https://twitter.com/chigozienri)

Advanced DangoCutn Cutout method is also from Dango223.

–

Disco:

Somnai (https://twitter.com/Somnai_dreams) added Diffusion Animation techniques, QoL improvements and various implementations of tech and techniques, mostly listed in the changelog below.

3D animation implementation added by Adam Letts (https://twitter.com/gandamu_ml) in collaboration with Somnai. Creation of disco.py and ongoing maintenance.

Turbo feature by Chris Allen (https://twitter.com/zippy731)

Improvements to ability to run on local systems, Windows support, and dependency installation by HostsServer (https://twitter.com/HostsServer)

VR Mode by Tom Mason (https://twitter.com/nin_artificial)

Horizontal and Vertical symmetry functionality by nshepperd. Symmetry transformation_steps by huemin (https://twitter.com/huemin_art). Symmetry integration into Disco Diffusion by Dmitrii Tochilkin (https://twitter.com/cut_pow).

Warp and custom model support by Alex Spirin (https://twitter.com/devdef).

Pixel Art Diffusion, Watercolor Diffusion, and Pulp SciFi Diffusion models from KaliYuga (https://twitter.com/KaliYuga_ai). Follow KaliYuga's Twitter for the latest models and for notebooks with specialized settings.

Integration of OpenCLIP models and initiation of integration of KaliYuga models by Palmweaver / Chris Scalf (https://twitter.com/ChrisScalf11)

Integrated portrait_generator_v001 from Felipe3DArtist (https://twitter.com/Felipe3DArtist)

28.2.7 Citation

```
@misc{github,
  author={alembics},
  title={disco-diffusion},
  year={2022},
  url={https://github.com/alembics/disco-diffusion},
}
```

CHAPTER 29

图像恢复

29.1 概览

- 预训练权重个数: 2
- 配置文件个数: 0
- 论文个数: 1
  - ALGORITHM: 1

29.2 NAFNET (ECCV’2022)

任务: 图像恢复

```
@article{chen2022simple,
  title={Simple Baselines for Image Restoration},
  author={Chen, Liangyu and Chu, Xiaojie and Zhang, Xiangyu and Sun, Jian},
  journal={arXiv preprint arXiv:2204.04676},
  year={2022}
}
```

Note:

- 评估结果 a(b) 中，a 代表由 MMagic 测量，b 代表由原论文提供。
- PSNR 是在 RGB 通道评估。

- SSIM 是平均的分别在 RGB 通道评估的 SSIM, 而原论文使用了 3D 的 SSIM 卷积核做统一评估。

29.2.1 快速开始

训练

您可以使用以下命令来训练模型。

```
## CPU上训练
CUDA_VISIBLE_DEVICES=-1 python tools/train.py configs/nafnet/nafnet_
↪c64eb2248mb12db2222_8xb8-lr1e-3-400k_sidd.py

## 单个GPU上训练
python tools/train.py configs/nafnet/nafnet_c64eb2248mb12db2222_8xb8-lr1e-3-400k_sidd.
↪py

## 多个GPU上训练
./tools/dist_train.sh configs/nafnet/nafnet_c64eb2248mb12db2222_8xb8-lr1e-3-400k_sidd.
↪py 8
```

更多细节可以参考 train_test.md 中的 **Train a model** 部分。

测试

您可以使用以下命令来测试模型。

```
## CPU上测试
CUDA_VISIBLE_DEVICES=-1 python tools/test.py configs/nafnet/nafnet_
↪c64eb2248mb12db2222_8xb8-lr1e-3-400k_sidd.py /path/to/checkpoint

## 单个GPU上测试
python tools/test.py configs/nafnet/nafnet_c64eb2248mb12db2222_8xb8-lr1e-3-400k_sidd.
↪py /path/to/checkpoint

## 多个GPU上测试
./tools/dist_test.sh configs/nafnet/nafnet_c64eb2248mb12db2222_8xb8-lr1e-3-400k_sidd.
↪py /path/to/checkpoint 8
```

预训练模型未来将会上传，敬请等待。更多细节可以参考 train_test.md 中的 **Test a pre-trained model** 部分。

CHAPTER 30

jpeg 压缩伪影移除

30.1 概览

- 预训练权重个数: 29
- 配置文件个数: 0
- 论文个数: 1
  - ALGORITHM: 1

30.2 SwinIR (ICCVW’2021)

任务: 图像超分辨率, 图像去噪, JPEG 压缩伪影移除

```
@inproceedings{liang2021swinir,
  title={Swinir: Image restoration using swin transformer},
  author={Liang, Jingyun and Cao, Jiezhang and Sun, Guolei and Zhang, Kai and Van␣
↪Gool, Luc and Timofte, Radu},
  booktitle={Proceedings of the IEEE/CVF International Conference on Computer Vision},
  pages={1833--1844},
  year={2021}
}
```

30.2.1 Classical Image Super-Resolution

在 Y 通道上进行评估，在评估之前裁剪每个边界中的 scale 像素。我们使用 PSNR 和 SSIM 作为指标。

30.2.2 Lightweight Image Super-Resolution

在 Y 通道上进行评估，在评估之前裁剪每个边界中的 scale 像素。我们使用 PSNR 和 SSIM 作为指标。

30.2.3 Real-World Image Super-Resolution

在 Y 通道上进行评估。我们使用 NIQE 作为指标。

30.2.4 Grayscale Image Deoising

在灰度图上进行评估。我们使用 PSNR 作为指标。

30.2.5 Color Image Deoising

在 RGB 通道上进行评估。我们使用 PSNR 作为指标。

30.2.6 JPEG Compression Artifact Reduction (grayscale)

在灰度图上进行评估。我们使用 PSNR 和 SSIM 作为指标。

30.2.7 JPEG Compression Artifact Reduction (color)

在 RGB 通道上进行评估。我们使用 PSNR 和 SSIM 作为指标。

30.2.8 快速开始

训练

您可以使用以下命令来训练模型。

```
## CPU上训练
## 001 Classical Image Super-Resolution (middle size)
## (setting1: when model is trained on DIV2K and with training_patch_size=48)
CUDA_VISIBLE_DEVICES=-1 python tools/train.py configs/swinir/swinir_x2s48w8d6e180_
↪8xb4-lr2e-4-500k_div2k.py
CUDA_VISIBLE_DEVICES=-1 python tools/train.py configs/swinir/swinir_x3s48w8d6e180_
↪8xb4-lr2e-4-500k_div2k.py
```

(下页继续)

(续上页)

```
CUDA_VISIBLE_DEVICES=-1 python tools/train.py configs/swinir/swinir_x4s48w8d6e180_
↪8xb4-lr2e-4-500k_div2k.py

## (setting2: when model is trained on DIV2K+Flickr2K and with training_patch_size=64)
CUDA_VISIBLE_DEVICES=-1 python tools/train.py configs/swinir/swinir_x2s64w8d6e180_
↪8xb4-lr2e-4-500k_df2k.py
CUDA_VISIBLE_DEVICES=-1 python tools/train.py configs/swinir/swinir_x3s64w8d6e180_
↪8xb4-lr2e-4-500k_df2k.py
CUDA_VISIBLE_DEVICES=-1 python tools/train.py configs/swinir/swinir_x4s64w8d6e180_
↪8xb4-lr2e-4-500k_df2k.py

## 002 Lightweight Image Super-Resolution (small size)
CUDA_VISIBLE_DEVICES=-1 python tools/train.py configs/swinir/swinir_x2s64w8d4e60_8xb4-
↪lr2e-4-500k_div2k.py
CUDA_VISIBLE_DEVICES=-1 python tools/train.py configs/swinir/swinir_x3s64w8d4e60_8xb4-
↪lr2e-4-500k_div2k.py
CUDA_VISIBLE_DEVICES=-1 python tools/train.py configs/swinir/swinir_x4s64w8d4e60_8xb4-
↪lr2e-4-500k_div2k.py

## 003 Real-World Image Super-Resolution
CUDA_VISIBLE_DEVICES=-1 python tools/train.py configs/swinir/swinir_gan-x2s64w8d6e180_
↪8xb4-lr1e-4-600k_df2k-ost.py
CUDA_VISIBLE_DEVICES=-1 python tools/train.py configs/swinir/swinir_psnr-
↪x2s64w8d6e180_8xb4-lr1e-4-600k_df2k-ost.py
CUDA_VISIBLE_DEVICES=-1 python tools/train.py configs/swinir/swinir_gan-x4s64w8d6e180_
↪8xb4-lr1e-4-600k_df2k-ost.py
CUDA_VISIBLE_DEVICES=-1 python tools/train.py configs/swinir/swinir_psnr-
↪x4s64w8d6e180_8xb4-lr1e-4-600k_df2k-ost.py
CUDA_VISIBLE_DEVICES=-1 python tools/train.py configs/swinir/swinir_gan-x4s64w8d9e240_
↪8xb4-lr1e-4-600k_df2k-ost.py
CUDA_VISIBLE_DEVICES=-1 python tools/train.py configs/swinir/swinir_psnr-
↪x4s64w8d9e240_8xb4-lr1e-4-600k_df2k-ost.py

## 004 Grayscale Image Deoising (middle size)
CUDA_VISIBLE_DEVICES=-1 python tools/train.py configs/swinir/swinir_s128w8d6e180_8xb1-
↪lr2e-4-1600k_dfwb-grayDN15.py
CUDA_VISIBLE_DEVICES=-1 python tools/train.py configs/swinir/swinir_s128w8d6e180_8xb1-
↪lr2e-4-1600k_dfwb-grayDN25.py
CUDA_VISIBLE_DEVICES=-1 python tools/train.py configs/swinir/swinir_s128w8d6e180_8xb1-
↪lr2e-4-1600k_dfwb-grayDN50.py

## 005 Color Image Deoising (middle size)
CUDA_VISIBLE_DEVICES=-1 python tools/train.py configs/swinir/swinir_s128w8d6e180_8xb1-
↪lr2e-4-1600k_dfwb-colorDN15.py
```

(下页继续)

(续上页)

```
CUDA_VISIBLE_DEVICES=-1 python tools/train.py configs/swinir/swinir_s128w8d6e180_8xb1-
↪lr2e-4-1600k_dfwb-colorDN25.py
CUDA_VISIBLE_DEVICES=-1 python tools/train.py configs/swinir/swinir_s128w8d6e180_8xb1-
↪lr2e-4-1600k_dfwb-colorDN50.py

## 006 JPEG Compression Artifact Reduction (middle size, using window_size=7 because␣
↪JPEG encoding uses 8x8 blocks)
## grayscale
CUDA_VISIBLE_DEVICES=-1 python tools/train.py configs/swinir/swinir_s126w7d6e180_8xb1-
↪lr2e-4-1600k_dfwb-grayCAR10.py
CUDA_VISIBLE_DEVICES=-1 python tools/train.py configs/swinir/swinir_s126w7d6e180_8xb1-
↪lr2e-4-1600k_dfwb-grayCAR20.py
CUDA_VISIBLE_DEVICES=-1 python tools/train.py configs/swinir/swinir_s126w7d6e180_8xb1-
↪lr2e-4-1600k_dfwb-grayCAR30.py
CUDA_VISIBLE_DEVICES=-1 python tools/train.py configs/swinir/swinir_s126w7d6e180_8xb1-
↪lr2e-4-1600k_dfwb-grayCAR40.py

## color
CUDA_VISIBLE_DEVICES=-1 python tools/train.py configs/swinir/swinir_s126w7d6e180_8xb1-
↪lr2e-4-1600k_dfwb-colorCAR10.py
CUDA_VISIBLE_DEVICES=-1 python tools/train.py configs/swinir/swinir_s126w7d6e180_8xb1-
↪lr2e-4-1600k_dfwb-colorCAR20.py
CUDA_VISIBLE_DEVICES=-1 python tools/train.py configs/swinir/swinir_s126w7d6e180_8xb1-
↪lr2e-4-1600k_dfwb-colorCAR30.py
CUDA_VISIBLE_DEVICES=-1 python tools/train.py configs/swinir/swinir_s126w7d6e180_8xb1-
↪lr2e-4-1600k_dfwb-colorCAR40.py



## 单个GPU上训练
## 001 Classical Image Super-Resolution (middle size)
## (setting1: when model is trained on DIV2K and with training_patch_size=48)
python tools/train.py configs/swinir/swinir_x2s48w8d6e180_8xb4-lr2e-4-500k_div2k.py
python tools/train.py configs/swinir/swinir_x3s48w8d6e180_8xb4-lr2e-4-500k_div2k.py
python tools/train.py configs/swinir/swinir_x4s48w8d6e180_8xb4-lr2e-4-500k_div2k.py

## (setting2: when model is trained on DIV2K+Flickr2K and with training_patch_size=64)
python tools/train.py configs/swinir/swinir_x2s64w8d6e180_8xb4-lr2e-4-500k_df2k.py
python tools/train.py configs/swinir/swinir_x3s64w8d6e180_8xb4-lr2e-4-500k_df2k.py
python tools/train.py configs/swinir/swinir_x4s64w8d6e180_8xb4-lr2e-4-500k_df2k.py

## 002 Lightweight Image Super-Resolution (small size)
python tools/train.py configs/swinir/swinir_x2s64w8d4e60_8xb4-lr2e-4-500k_div2k.py
```

(下页继续)

(续上页)

```
python tools/train.py configs/swinir/swinir_x3s64w8d4e60_8xb4-lr2e-4-500k_div2k.py
python tools/train.py configs/swinir/swinir_x4s64w8d4e60_8xb4-lr2e-4-500k_div2k.py

## 003 Real-World Image Super-Resolution
python tools/train.py configs/swinir/swinir_gan-x2s64w8d6e180_8xb4-lr1e-4-600k_df2k-
↪ost.py
python tools/train.py configs/swinir/swinir_psnr-x2s64w8d6e180_8xb4-lr1e-4-600k_df2k-
↪ost.py
python tools/train.py configs/swinir/swinir_gan-x4s64w8d6e180_8xb4-lr1e-4-600k_df2k-
↪ost.py
python tools/train.py configs/swinir/swinir_psnr-x4s64w8d6e180_8xb4-lr1e-4-600k_df2k-
↪ost.py
python tools/train.py configs/swinir/swinir_gan-x4s64w8d9e240_8xb4-lr1e-4-600k_df2k-
↪ost.py
python tools/train.py configs/swinir/swinir_psnr-x4s64w8d9e240_8xb4-lr1e-4-600k_df2k-
↪ost.py

## 004 Grayscale Image Deoising (middle size)
python tools/train.py configs/swinir/swinir_s128w8d6e180_8xb1-lr2e-4-1600k_dfwb-
↪grayDN15.py
python tools/train.py configs/swinir/swinir_s128w8d6e180_8xb1-lr2e-4-1600k_dfwb-
↪grayDN25.py
python tools/train.py configs/swinir/swinir_s128w8d6e180_8xb1-lr2e-4-1600k_dfwb-
↪grayDN50.py

## 005 Color Image Deoising (middle size)
python tools/train.py configs/swinir/swinir_s128w8d6e180_8xb1-lr2e-4-1600k_dfwb-
↪colorDN15.py
python tools/train.py configs/swinir/swinir_s128w8d6e180_8xb1-lr2e-4-1600k_dfwb-
↪colorDN25.py
python tools/train.py configs/swinir/swinir_s128w8d6e180_8xb1-lr2e-4-1600k_dfwb-
↪colorDN50.py

## 006 JPEG Compression Artifact Reduction (middle size, using window_size=7 because␣
↪JPEG encoding uses 8x8 blocks)
## grayscale
python tools/train.py configs/swinir/swinir_s126w7d6e180_8xb1-lr2e-4-1600k_dfwb-
↪grayCAR10.py
python tools/train.py configs/swinir/swinir_s126w7d6e180_8xb1-lr2e-4-1600k_dfwb-
↪grayCAR20.py
python tools/train.py configs/swinir/swinir_s126w7d6e180_8xb1-lr2e-4-1600k_dfwb-
↪grayCAR30.py
python tools/train.py configs/swinir/swinir_s126w7d6e180_8xb1-lr2e-4-1600k_dfwb-
↪grayCAR40.py
```

(下页继续)

(续上页)

```
## color
python tools/train.py configs/swinir/swinir_s126w7d6e180_8xb1-lr2e-4-1600k_dfwb-
↪colorCAR10.py
python tools/train.py configs/swinir/swinir_s126w7d6e180_8xb1-lr2e-4-1600k_dfwb-
↪colorCAR20.py
python tools/train.py configs/swinir/swinir_s126w7d6e180_8xb1-lr2e-4-1600k_dfwb-
↪colorCAR30.py
python tools/train.py configs/swinir/swinir_s126w7d6e180_8xb1-lr2e-4-1600k_dfwb-
↪colorCAR40.py



## 多个GPU上训练
## 001 Classical Image Super-Resolution (middle size)
## (setting1: when model is trained on DIV2K and with training_patch_size=48)
./tools/dist_train.sh configs/swinir/swinir_x2s48w8d6e180_8xb4-lr2e-4-500k_div2k.py 8
./tools/dist_train.sh configs/swinir/swinir_x3s48w8d6e180_8xb4-lr2e-4-500k_div2k.py 8
./tools/dist_train.sh configs/swinir/swinir_x4s48w8d6e180_8xb4-lr2e-4-500k_div2k.py 8

## (setting2: when model is trained on DIV2K+Flickr2K and with training_patch_size=64)
./tools/dist_train.sh configs/swinir/swinir_x2s64w8d6e180_8xb4-lr2e-4-500k_df2k.py 8
./tools/dist_train.sh configs/swinir/swinir_x3s64w8d6e180_8xb4-lr2e-4-500k_df2k.py 8
./tools/dist_train.sh configs/swinir/swinir_x4s64w8d6e180_8xb4-lr2e-4-500k_df2k.py 8

## 002 Lightweight Image Super-Resolution (small size)
./tools/dist_train.sh configs/swinir/swinir_x2s64w8d4e60_8xb4-lr2e-4-500k_div2k.py 8
./tools/dist_train.sh configs/swinir/swinir_x3s64w8d4e60_8xb4-lr2e-4-500k_div2k.py 8
./tools/dist_train.sh configs/swinir/swinir_x4s64w8d4e60_8xb4-lr2e-4-500k_div2k.py 8

## 003 Real-World Image Super-Resolution
./tools/dist_train.sh configs/swinir/swinir_gan-x2s64w8d6e180_8xb4-lr1e-4-600k_df2k-
↪ost.py 8
./tools/dist_train.sh configs/swinir/swinir_psnr-x2s64w8d6e180_8xb4-lr1e-4-600k_df2k-
↪ost.py 8
./tools/dist_train.sh configs/swinir/swinir_gan-x4s64w8d6e180_8xb4-lr1e-4-600k_df2k-
↪ost.py 8
./tools/dist_train.sh configs/swinir/swinir_psnr-x4s64w8d6e180_8xb4-lr1e-4-600k_df2k-
↪ost.py 8
./tools/dist_train.sh configs/swinir/swinir_gan-x4s64w8d9e240_8xb4-lr1e-4-600k_df2k-
↪ost.py 8
./tools/dist_train.sh configs/swinir/swinir_psnr-x4s64w8d9e240_8xb4-lr1e-4-600k_df2k-
↪ost.py 8
```

(下页继续)

(续上页)

```
## 004 Grayscale Image Deoising (middle size)
./tools/dist_train.sh configs/swinir/swinir_s128w8d6e180_8xb1-lr2e-4-1600k_dfwb-
↪grayDN15.py 8
./tools/dist_train.sh configs/swinir/swinir_s128w8d6e180_8xb1-lr2e-4-1600k_dfwb-
↪grayDN25.py 8
./tools/dist_train.sh configs/swinir/swinir_s128w8d6e180_8xb1-lr2e-4-1600k_dfwb-
↪grayDN50.py 8

## 005 Color Image Deoising (middle size)
./tools/dist_train.sh configs/swinir/swinir_s128w8d6e180_8xb1-lr2e-4-1600k_dfwb-
↪colorDN15.py 8
./tools/dist_train.sh configs/swinir/swinir_s128w8d6e180_8xb1-lr2e-4-1600k_dfwb-
↪colorDN25.py 8
./tools/dist_train.sh configs/swinir/swinir_s128w8d6e180_8xb1-lr2e-4-1600k_dfwb-
↪colorDN50.py 8

## 006 JPEG Compression Artifact Reduction (middle size, using window_size=7 because␣
↪JPEG encoding uses 8x8 blocks)
## grayscale
./tools/dist_train.sh configs/swinir/swinir_s126w7d6e180_8xb1-lr2e-4-1600k_dfwb-
↪grayCAR10.py 8
./tools/dist_train.sh configs/swinir/swinir_s126w7d6e180_8xb1-lr2e-4-1600k_dfwb-
↪grayCAR20.py 8
./tools/dist_train.sh configs/swinir/swinir_s126w7d6e180_8xb1-lr2e-4-1600k_dfwb-
↪grayCAR30.py 8
./tools/dist_train.sh configs/swinir/swinir_s126w7d6e180_8xb1-lr2e-4-1600k_dfwb-
↪grayCAR40.py 8

## color
./tools/dist_train.sh configs/swinir/swinir_s126w7d6e180_8xb1-lr2e-4-1600k_dfwb-
↪colorCAR10.py 8
./tools/dist_train.sh configs/swinir/swinir_s126w7d6e180_8xb1-lr2e-4-1600k_dfwb-
↪colorCAR20.py 8
./tools/dist_train.sh configs/swinir/swinir_s126w7d6e180_8xb1-lr2e-4-1600k_dfwb-
↪colorCAR30.py 8
./tools/dist_train.sh configs/swinir/swinir_s126w7d6e180_8xb1-lr2e-4-1600k_dfwb-
↪colorCAR40.py 8
```

更多细节可以参考 train_test.md 中的 **Train a model** 部分。

测试

您可以使用以下命令来测试模型。

```
## CPU上测试
## 001 Classical Image Super-Resolution (middle size)
## (setting1: when model is trained on DIV2K and with training_patch_size=48)
CUDA_VISIBLE_DEVICES=-1 python tools/test.py configs/swinir/swinir_x2s48w8d6e180_8xb4-
↪lr2e-4-500k_div2k.py https://download.openmmlab.com/mmediting/swinir/swinir_
↪x2s48w8d6e180_8xb4-lr2e-4-500k_div2k-ed2d419e.pth

CUDA_VISIBLE_DEVICES=-1 python tools/test.py configs/swinir/swinir_x3s48w8d6e180_8xb4-
↪lr2e-4-500k_div2k.py https://download.openmmlab.com/mmediting/swinir/swinir_
↪x3s48w8d6e180_8xb4-lr2e-4-500k_div2k-926950f1.pth

CUDA_VISIBLE_DEVICES=-1 python tools/test.py configs/swinir/swinir_x4s48w8d6e180_8xb4-
↪lr2e-4-500k_div2k.py https://download.openmmlab.com/mmediting/swinir/swinir_
↪x4s48w8d6e180_8xb4-lr2e-4-500k_div2k-88e4903d.pth

## (setting2: when model is trained on DIV2K+Flickr2K and with training_patch_size=64)
CUDA_VISIBLE_DEVICES=-1 python tools/test.py configs/swinir/swinir_x2s64w8d6e180_8xb4-
↪lr2e-4-500k_df2k.py https://download.openmmlab.com/mmediting/swinir/swinir_
↪x2s64w8d6e180_8xb4-lr2e-4-500k_df2k-69e15fb6.pth

CUDA_VISIBLE_DEVICES=-1 python tools/test.py configs/swinir/swinir_x3s64w8d6e180_8xb4-
↪lr2e-4-500k_df2k.py https://download.openmmlab.com/mmediting/swinir/swinir_
↪x3s64w8d6e180_8xb4-lr2e-4-500k_df2k-d6982f7b.pth

CUDA_VISIBLE_DEVICES=-1 python tools/test.py configs/swinir/swinir_x4s64w8d6e180_8xb4-
↪lr2e-4-500k_df2k.py https://download.openmmlab.com/mmediting/swinir/swinir_
↪x4s64w8d6e180_8xb4-lr2e-4-500k_df2k-0502d775.pth


## 002 Lightweight Image Super-Resolution (small size)
CUDA_VISIBLE_DEVICES=-1 python tools/test.py configs/swinir/swinir_x2s64w8d4e60_8xb4-
↪lr2e-4-500k_div2k.py https://download.openmmlab.com/mmediting/swinir/swinir_
↪x2s64w8d4e60_8xb4-lr2e-4-500k_div2k-131d3f64.pth

CUDA_VISIBLE_DEVICES=-1 python tools/test.py configs/swinir/swinir_x3s64w8d4e60_8xb4-
↪lr2e-4-500k_div2k.py https://download.openmmlab.com/mmediting/swinir/swinir_
↪x3s64w8d4e60_8xb4-lr2e-4-500k_div2k-309cb239.pth

CUDA_VISIBLE_DEVICES=-1 python tools/test.py configs/swinir/swinir_x4s64w8d4e60_8xb4-
↪lr2e-4-500k_div2k.py https://download.openmmlab.com/mmediting/swinir/swinir_
↪x4s64w8d4e60_8xb4-lr2e-4-500k_div2k-d6622d03.pth

## 003 Real-World Image Super-Resolution
CUDA_VISIBLE_DEVICES=-1 python tools/test.py configs/swinir/swinir_gan-x2s64w8d6e180_
↪8xb4-lr1e-4-600k_df2k-ost.py https://download.openmmlab.com/mmediting/swinir/swinir_
↪gan-x2s64w8d6e180_8xb4-lr1e-4-600k_df2k-os-c6425057.pth
```

(下页继续)

(续上页)

```
CUDA_VISIBLE_DEVICES=-1 python tools/test.py configs/swinir/swinir_psnr-x2s64w8d6e180_
↪8xb4-lr1e-4-600k_df2k-ost.py https://download.openmmlab.com/mmediting/swinir/swinir_
↪psnr-x2s64w8d6e180_8xb4-lr1e-4-600k_df2k-os-6f0c425f.pth

CUDA_VISIBLE_DEVICES=-1 python tools/test.py configs/swinir/swinir_gan-x4s64w8d6e180_
↪8xb4-lr1e-4-600k_df2k-ost.py https://download.openmmlab.com/mmediting/swinir/swinir_
↪gan-x4s64w8d6e180_8xb4-lr1e-4-600k_df2k-os-36960d18.pth

CUDA_VISIBLE_DEVICES=-1 python tools/test.py configs/swinir/swinir_psnr-x4s64w8d6e180_
↪8xb4-lr1e-4-600k_df2k-ost.py https://download.openmmlab.com/mmediting/swinir/swinir_
↪psnr-x4s64w8d6e180_8xb4-lr1e-4-600k_df2k-os-a016a72f.pth

CUDA_VISIBLE_DEVICES=-1 python tools/test.py configs/swinir/swinir_gan-x4s64w8d9e240_
↪8xb4-lr1e-4-600k_df2k-ost.py https://download.openmmlab.com/mmediting/swinir/swinir_
↪gan-x4s64w8d9e240_8xb4-lr1e-4-600k_df2k-os-9f1599b5.pth

CUDA_VISIBLE_DEVICES=-1 python tools/test.py configs/swinir/swinir_psnr-x4s64w8d9e240_
↪8xb4-lr1e-4-600k_df2k-ost.py https://download.openmmlab.com/mmediting/swinir/swinir_
↪psnr-x4s64w8d9e240_8xb4-lr1e-4-600k_df2k-os-25f1722a.pth

## 004 Grayscale Image Deoising (middle size)
CUDA_VISIBLE_DEVICES=-1 python tools/test.py configs/swinir/swinir_s128w8d6e180_8xb1-
↪lr2e-4-1600k_dfwb-grayDN15.py https://download.openmmlab.com/mmediting/swinir/
↪swinir_s128w8d6e180_8xb1-lr2e-4-1600k_dfwb-grayDN15-6782691b.pth

CUDA_VISIBLE_DEVICES=-1 python tools/test.py configs/swinir/swinir_s128w8d6e180_8xb1-
↪lr2e-4-1600k_dfwb-grayDN25.py https://download.openmmlab.com/mmediting/swinir/
↪swinir_s128w8d6e180_8xb1-lr2e-4-1600k_dfwb-grayDN25-d0d8d4da.pth

CUDA_VISIBLE_DEVICES=-1 python tools/test.py configs/swinir/swinir_s128w8d6e180_8xb1-
↪lr2e-4-1600k_dfwb-grayDN50.py https://download.openmmlab.com/mmediting/swinir/
↪swinir_s128w8d6e180_8xb1-lr2e-4-1600k_dfwb-grayDN50-54c9968a.pth

## 005 Color Image Deoising (middle size)
CUDA_VISIBLE_DEVICES=-1 python tools/test.py configs/swinir/swinir_s128w8d6e180_8xb1-
↪lr2e-4-1600k_dfwb-colorDN15.py https://download.openmmlab.com/mmediting/swinir/
↪swinir_s128w8d6e180_8xb1-lr2e-4-1600k_dfwb-colorDN15-c74a2cee.pth

CUDA_VISIBLE_DEVICES=-1 python tools/test.py configs/swinir/swinir_s128w8d6e180_8xb1-
↪lr2e-4-1600k_dfwb-colorDN25.py https://download.openmmlab.com/mmediting/swinir/
↪swinir_s128w8d6e180_8xb1-lr2e-4-1600k_dfwb-colorDN25-df2b1c0c.pth
```

(下页继续)

(续上页)

```
CUDA_VISIBLE_DEVICES=-1 python tools/test.py configs/swinir/swinir_s128w8d6e180_8xb1-
↪lr2e-4-1600k_dfwb-colorDN50.py https://download.openmmlab.com/mmediting/swinir/
↪swinir_s128w8d6e180_8xb1-lr2e-4-1600k_dfwb-colorDN50-e369874c.pth

## 006 JPEG Compression Artifact Reduction (middle size, using window_size=7 because␣
↪JPEG encoding usesx8 blocks)
## grayscale
CUDA_VISIBLE_DEVICES=-1 python tools/test.py configs/swinir/swinir_s126w7d6e180_8xb1-
↪lr2e-4-1600k_dfwb-grayCAR10.py https://download.openmmlab.com/mmediting/swinir/
↪swinir_s126w7d6e180_8xb1-lr2e-4-1600k_dfwb-grayCAR10-da93c8e9.pth

CUDA_VISIBLE_DEVICES=-1 python tools/test.py configs/swinir/swinir_s126w7d6e180_8xb1-
↪lr2e-4-1600k_dfwb-grayCAR20.py https://download.openmmlab.com/mmediting/swinir/
↪swinir_s126w7d6e180_8xb1-lr2e-4-1600k_dfwb-grayCAR20-d47367b1.pth

CUDA_VISIBLE_DEVICES=-1 python tools/test.py configs/swinir/swinir_s126w7d6e180_8xb1-
↪lr2e-4-1600k_dfwb-grayCAR30.py https://download.openmmlab.com/mmediting/swinir/
↪swinir_s126w7d6e180_8xb1-lr2e-4-1600k_dfwb-grayCAR30-52c083cf.pth

CUDA_VISIBLE_DEVICES=-1 python tools/test.py configs/swinir/swinir_s126w7d6e180_8xb1-
↪lr2e-4-1600k_dfwb-grayCAR40.py https://download.openmmlab.com/mmediting/swinir/
↪swinir_s126w7d6e180_8xb1-lr2e-4-1600k_dfwb-grayCAR40-803e8d9b.pth


## color
CUDA_VISIBLE_DEVICES=-1 python tools/test.py configs/swinir/swinir_s126w7d6e180_8xb1-
↪lr2e-4-1600k_dfwb-colorCAR10.py https://download.openmmlab.com/mmediting/swinir/
↪swinir_s126w7d6e180_8xb1-lr2e-4-1600k_dfwb-colorCAR10-09aafadc.pth

CUDA_VISIBLE_DEVICES=-1 python tools/test.py configs/swinir/swinir_s126w7d6e180_8xb1-
↪lr2e-4-1600k_dfwb-colorCAR20.py https://download.openmmlab.com/mmediting/swinir/
↪swinir_s126w7d6e180_8xb1-lr2e-4-1600k_dfwb-colorCAR20-b8a42b5e.pth

CUDA_VISIBLE_DEVICES=-1 python tools/test.py configs/swinir/swinir_s126w7d6e180_8xb1-
↪lr2e-4-1600k_dfwb-colorCAR30.py https://download.openmmlab.com/mmediting/swinir/
↪swinir_s126w7d6e180_8xb1-lr2e-4-1600k_dfwb-colorCAR30-e9fe6859.pth

CUDA_VISIBLE_DEVICES=-1 python tools/test.py configs/swinir/swinir_s126w7d6e180_8xb1-
↪lr2e-4-1600k_dfwb-colorCAR40.py https://download.openmmlab.com/mmediting/swinir/
↪swinir_s126w7d6e180_8xb1-lr2e-4-1600k_dfwb-colorCAR40-5b77a6e6.pth
```

(下页继续)

(续上页)

```
## 单个GPU上测试
## 001 Classical Image Super-Resolution (middle size)
## (setting1: when model is trained on DIV2K and with training_patch_size=48)
python tools/test.py configs/swinir/swinir_x2s48w8d6e180_8xb4-lr2e-4-500k_div2k.py␣
↪https://download.openmmlab.com/mmediting/swinir/swinir_x2s48w8d6e180_8xb4-lr2e-4-
↪500k_div2k-ed2d419e.pth

python tools/test.py configs/swinir/swinir_x3s48w8d6e180_8xb4-lr2e-4-500k_div2k.py␣
↪https://download.openmmlab.com/mmediting/swinir/swinir_x3s48w8d6e180_8xb4-lr2e-4-
↪500k_div2k-926950f1.pth

python tools/test.py configs/swinir/swinir_x4s48w8d6e180_8xb4-lr2e-4-500k_div2k.py␣
↪https://download.openmmlab.com/mmediting/swinir/swinir_x4s48w8d6e180_8xb4-lr2e-4-
↪500k_div2k-88e4903d.pth

## (setting2: when model is trained on DIV2K+Flickr2K and with training_patch_size=64)
python tools/test.py configs/swinir/swinir_x2s64w8d6e180_8xb4-lr2e-4-500k_df2k.py␣
↪https://download.openmmlab.com/mmediting/swinir/swinir_x2s64w8d6e180_8xb4-lr2e-4-
↪500k_df2k-69e15fb6.pth

python tools/test.py configs/swinir/swinir_x3s64w8d6e180_8xb4-lr2e-4-500k_df2k.py␣
↪https://download.openmmlab.com/mmediting/swinir/swinir_x3s64w8d6e180_8xb4-lr2e-4-
↪500k_df2k-d6982f7b.pth

python tools/test.py configs/swinir/swinir_x4s64w8d6e180_8xb4-lr2e-4-500k_df2k.py␣
↪https://download.openmmlab.com/mmediting/swinir/swinir_x4s64w8d6e180_8xb4-lr2e-4-
↪500k_df2k-0502d775.pth


## 002 Lightweight Image Super-Resolution (small size)
python tools/test.py configs/swinir/swinir_x2s64w8d4e60_8xb4-lr2e-4-500k_div2k.py␣
↪https://download.openmmlab.com/mmediting/swinir/swinir_x2s64w8d4e60_8xb4-lr2e-4-
↪500k_div2k-131d3f64.pth

python tools/test.py configs/swinir/swinir_x3s64w8d4e60_8xb4-lr2e-4-500k_div2k.py␣
↪https://download.openmmlab.com/mmediting/swinir/swinir_x3s64w8d4e60_8xb4-lr2e-4-
↪500k_div2k-309cb239.pth

python tools/test.py configs/swinir/swinir_x4s64w8d4e60_8xb4-lr2e-4-500k_div2k.py␣
↪https://download.openmmlab.com/mmediting/swinir/swinir_x4s64w8d4e60_8xb4-lr2e-4-
↪500k_div2k-d6622d03.pth
```

(下页继续)

(续上页)

```
## 003 Real-World Image Super-Resolution
python tools/test.py configs/swinir/swinir_gan-x2s64w8d6e180_8xb4-lr1e-4-600k_df2k-
↪ost.py https://download.openmmlab.com/mmediting/swinir/swinir_gan-x2s64w8d6e180_
↪8xb4-lr1e-4-600k_df2k-os-c6425057.pth

python tools/test.py configs/swinir/swinir_psnr-x2s64w8d6e180_8xb4-lr1e-4-600k_df2k-
↪ost.py https://download.openmmlab.com/mmediting/swinir/swinir_psnr-x2s64w8d6e180_
↪8xb4-lr1e-4-600k_df2k-os-6f0c425f.pth

python tools/test.py configs/swinir/swinir_gan-x4s64w8d6e180_8xb4-lr1e-4-600k_df2k-
↪ost.py https://download.openmmlab.com/mmediting/swinir/swinir_gan-x4s64w8d6e180_
↪8xb4-lr1e-4-600k_df2k-os-36960d18.pth

python tools/test.py configs/swinir/swinir_psnr-x4s64w8d6e180_8xb4-lr1e-4-600k_df2k-
↪ost.py https://download.openmmlab.com/mmediting/swinir/swinir_psnr-x4s64w8d6e180_
↪8xb4-lr1e-4-600k_df2k-os-a016a72f.pth

python tools/test.py configs/swinir/swinir_gan-x4s64w8d9e240_8xb4-lr1e-4-600k_df2k-
↪ost.py https://download.openmmlab.com/mmediting/swinir/swinir_gan-x4s64w8d9e240_
↪8xb4-lr1e-4-600k_df2k-os-9f1599b5.pth

python tools/test.py configs/swinir/swinir_psnr-x4s64w8d9e240_8xb4-lr1e-4-600k_df2k-
↪ost.py https://download.openmmlab.com/mmediting/swinir/swinir_psnr-x4s64w8d9e240_
↪8xb4-lr1e-4-600k_df2k-os-25f1722a.pth


## 004 Grayscale Image Deoising (middle size)
python tools/test.py configs/swinir/swinir_s128w8d6e180_8xb1-lr2e-4-1600k_dfwb-
↪grayDN15.py https://download.openmmlab.com/mmediting/swinir/swinir_s128w8d6e180_
↪8xb1-lr2e-4-1600k_dfwb-grayDN15-6782691b.pth

python tools/test.py configs/swinir/swinir_s128w8d6e180_8xb1-lr2e-4-1600k_dfwb-
↪grayDN25.py https://download.openmmlab.com/mmediting/swinir/swinir_s128w8d6e180_
↪8xb1-lr2e-4-1600k_dfwb-grayDN25-d0d8d4da.pth

python tools/test.py configs/swinir/swinir_s128w8d6e180_8xb1-lr2e-4-1600k_dfwb-
↪grayDN50.py https://download.openmmlab.com/mmediting/swinir/swinir_s128w8d6e180_
↪8xb1-lr2e-4-1600k_dfwb-grayDN50-54c9968a.pth


## 005 Color Image Deoising (middle size)
python tools/test.py configs/swinir/swinir_s128w8d6e180_8xb1-lr2e-4-1600k_dfwb-
↪colorDN15.py https://download.openmmlab.com/mmediting/swinir/swinir_s128w8d6e180_
↪8xb1-lr2e-4-1600k_dfwb-colorDN15-c74a2cee.pth
```

(下页继续)

(续上页)

```
python tools/test.py configs/swinir/swinir_s128w8d6e180_8xb1-lr2e-4-1600k_dfwb-
↪colorDN25.py https://download.openmmlab.com/mmediting/swinir/swinir_s128w8d6e180_
↪8xb1-lr2e-4-1600k_dfwb-colorDN25-df2b1c0c.pth

python tools/test.py configs/swinir/swinir_s128w8d6e180_8xb1-lr2e-4-1600k_dfwb-
↪colorDN50.py https://download.openmmlab.com/mmediting/swinir/swinir_s128w8d6e180_
↪8xb1-lr2e-4-1600k_dfwb-colorDN50-e369874c.pth


## 006 JPEG Compression Artifact Reduction (middle size, using window_size=7 because␣
↪JPEG encoding usesx8 blocks)
## grayscale
python tools/test.py configs/swinir/swinir_s126w7d6e180_8xb1-lr2e-4-1600k_dfwb-
↪grayCAR10.py https://download.openmmlab.com/mmediting/swinir/swinir_s126w7d6e180_
↪8xb1-lr2e-4-1600k_dfwb-grayCAR10-da93c8e9.pth

python tools/test.py configs/swinir/swinir_s126w7d6e180_8xb1-lr2e-4-1600k_dfwb-
↪grayCAR20.py https://download.openmmlab.com/mmediting/swinir/swinir_s126w7d6e180_
↪8xb1-lr2e-4-1600k_dfwb-grayCAR20-d47367b1.pth

python tools/test.py configs/swinir/swinir_s126w7d6e180_8xb1-lr2e-4-1600k_dfwb-
↪grayCAR30.py https://download.openmmlab.com/mmediting/swinir/swinir_s126w7d6e180_
↪8xb1-lr2e-4-1600k_dfwb-grayCAR30-52c083cf.pth

python tools/test.py configs/swinir/swinir_s126w7d6e180_8xb1-lr2e-4-1600k_dfwb-
↪grayCAR40.py https://download.openmmlab.com/mmediting/swinir/swinir_s126w7d6e180_
↪8xb1-lr2e-4-1600k_dfwb-grayCAR40-803e8d9b.pth


## color
python tools/test.py configs/swinir/swinir_s126w7d6e180_8xb1-lr2e-4-1600k_dfwb-
↪colorCAR10.py https://download.openmmlab.com/mmediting/swinir/swinir_s126w7d6e180_
↪8xb1-lr2e-4-1600k_dfwb-colorCAR10-09aafadc.pth

python tools/test.py configs/swinir/swinir_s126w7d6e180_8xb1-lr2e-4-1600k_dfwb-
↪colorCAR20.py https://download.openmmlab.com/mmediting/swinir/swinir_s126w7d6e180_
↪8xb1-lr2e-4-1600k_dfwb-colorCAR20-b8a42b5e.pth

python tools/test.py configs/swinir/swinir_s126w7d6e180_8xb1-lr2e-4-1600k_dfwb-
↪colorCAR30.py https://download.openmmlab.com/mmediting/swinir/swinir_s126w7d6e180_
↪8xb1-lr2e-4-1600k_dfwb-colorCAR30-e9fe6859.pth
```

(下页继续)

(续上页)

```
python tools/test.py configs/swinir/swinir_s126w7d6e180_8xb1-lr2e-4-1600k_dfwb-
↪colorCAR40.py https://download.openmmlab.com/mmediting/swinir/swinir_s126w7d6e180_
↪8xb1-lr2e-4-1600k_dfwb-colorCAR40-5b77a6e6.pth



## 多GPU测试
## 001 Classical Image Super-Resolution (middle size)
## (setting1: when model is trained on DIV2K and with training_patch_size=48)
./tools/dist_test.sh configs/swinir/swinir_x2s48w8d6e180_8xb4-lr2e-4-500k_div2k.py␣
↪https://download.openmmlab.com/mmediting/swinir/swinir_x2s48w8d6e180_8xb4-lr2e-4-
↪500k_div2k-ed2d419e.pth

./tools/dist_test.sh configs/swinir/swinir_x3s48w8d6e180_8xb4-lr2e-4-500k_div2k.py␣
↪https://download.openmmlab.com/mmediting/swinir/swinir_x3s48w8d6e180_8xb4-lr2e-4-
↪500k_div2k-926950f1.pth

./tools/dist_test.sh configs/swinir/swinir_x4s48w8d6e180_8xb4-lr2e-4-500k_div2k.py␣
↪https://download.openmmlab.com/mmediting/swinir/swinir_x4s48w8d6e180_8xb4-lr2e-4-
↪500k_div2k-88e4903d.pth

## (setting2: when model is trained on DIV2K+Flickr2K and with training_patch_size=64)
./tools/dist_test.sh configs/swinir/swinir_x2s64w8d6e180_8xb4-lr2e-4-500k_df2k.py␣
↪https://download.openmmlab.com/mmediting/swinir/swinir_x2s64w8d6e180_8xb4-lr2e-4-
↪500k_df2k-69e15fb6.pth

./tools/dist_test.sh configs/swinir/swinir_x3s64w8d6e180_8xb4-lr2e-4-500k_df2k.py␣
↪https://download.openmmlab.com/mmediting/swinir/swinir_x3s64w8d6e180_8xb4-lr2e-4-
↪500k_df2k-d6982f7b.pth

./tools/dist_test.sh configs/swinir/swinir_x4s64w8d6e180_8xb4-lr2e-4-500k_df2k.py␣
↪https://download.openmmlab.com/mmediting/swinir/swinir_x4s64w8d6e180_8xb4-lr2e-4-
↪500k_df2k-0502d775.pth

## 002 Lightweight Image Super-Resolution (small size)
./tools/dist_test.sh configs/swinir/swinir_x2s64w8d4e60_8xb4-lr2e-4-500k_div2k.py␣
↪https://download.openmmlab.com/mmediting/swinir/swinir_x2s64w8d4e60_8xb4-lr2e-4-
↪500k_div2k-131d3f64.pth

./tools/dist_test.sh configs/swinir/swinir_x3s64w8d4e60_8xb4-lr2e-4-500k_div2k.py␣
↪https://download.openmmlab.com/mmediting/swinir/swinir_x3s64w8d4e60_8xb4-lr2e-4-
↪500k_div2k-309cb239.pth
```

(下页继续)

(续上页)

```
./tools/dist_test.sh configs/swinir/swinir_x4s64w8d4e60_8xb4-lr2e-4-500k_div2k.py␣
↪https://download.openmmlab.com/mmediting/swinir/swinir_x4s64w8d4e60_8xb4-lr2e-4-
↪500k_div2k-d6622d03.pth

## 003 Real-World Image Super-Resolution
./tools/dist_test.sh configs/swinir/swinir_gan-x2s64w8d6e180_8xb4-lr1e-4-600k_df2k-
↪ost.py https://download.openmmlab.com/mmediting/swinir/swinir_gan-x2s64w8d6e180_
↪8xb4-lr1e-4-600k_df2k-os-c6425057.pth

./tools/dist_test.sh configs/swinir/swinir_psnr-x2s64w8d6e180_8xb4-lr1e-4-600k_df2k-
↪ost.py https://download.openmmlab.com/mmediting/swinir/swinir_psnr-x2s64w8d6e180_
↪8xb4-lr1e-4-600k_df2k-os-6f0c425f.pth

./tools/dist_test.sh configs/swinir/swinir_gan-x4s64w8d6e180_8xb4-lr1e-4-600k_df2k-
↪ost.py https://download.openmmlab.com/mmediting/swinir/swinir_gan-x4s64w8d6e180_
↪8xb4-lr1e-4-600k_df2k-os-36960d18.pth

./tools/dist_test.sh configs/swinir/swinir_psnr-x4s64w8d6e180_8xb4-lr1e-4-600k_df2k-
↪ost.py https://download.openmmlab.com/mmediting/swinir/swinir_psnr-x4s64w8d6e180_
↪8xb4-lr1e-4-600k_df2k-os-a016a72f.pth

./tools/dist_test.sh configs/swinir/swinir_gan-x4s64w8d9e240_8xb4-lr1e-4-600k_df2k-
↪ost.py https://download.openmmlab.com/mmediting/swinir/swinir_gan-x4s64w8d9e240_
↪8xb4-lr1e-4-600k_df2k-os-9f1599b5.pth

./tools/dist_test.sh configs/swinir/swinir_psnr-x4s64w8d9e240_8xb4-lr1e-4-600k_df2k-
↪ost.py https://download.openmmlab.com/mmediting/swinir/swinir_psnr-x4s64w8d9e240_
↪8xb4-lr1e-4-600k_df2k-os-25f1722a.pth

## 004 Grayscale Image Deoising (middle size)
./tools/dist_test.sh configs/swinir/swinir_s128w8d6e180_8xb1-lr2e-4-1600k_dfwb-
↪grayDN15.py https://download.openmmlab.com/mmediting/swinir/swinir_s128w8d6e180_
↪8xb1-lr2e-4-1600k_dfwb-grayDN15-6782691b.pth

./tools/dist_test.sh configs/swinir/swinir_s128w8d6e180_8xb1-lr2e-4-1600k_dfwb-
↪grayDN25.py https://download.openmmlab.com/mmediting/swinir/swinir_s128w8d6e180_
↪8xb1-lr2e-4-1600k_dfwb-grayDN25-d0d8d4da.pth

./tools/dist_test.sh configs/swinir/swinir_s128w8d6e180_8xb1-lr2e-4-1600k_dfwb-
↪grayDN50.py https://download.openmmlab.com/mmediting/swinir/swinir_s128w8d6e180_
↪8xb1-lr2e-4-1600k_dfwb-grayDN50-54c9968a.pth

## 005 Color Image Deoising (middle size)
```

(下页继续)

(续上页)

```
./tools/dist_test.sh configs/swinir/swinir_s128w8d6e180_8xb1-lr2e-4-1600k_dfwb-
↪colorDN15.py https://download.openmmlab.com/mmediting/swinir/swinir_s128w8d6e180_
↪8xb1-lr2e-4-1600k_dfwb-colorDN15-c74a2cee.pth

./tools/dist_test.sh configs/swinir/swinir_s128w8d6e180_8xb1-lr2e-4-1600k_dfwb-
↪colorDN25.py https://download.openmmlab.com/mmediting/swinir/swinir_s128w8d6e180_
↪8xb1-lr2e-4-1600k_dfwb-colorDN25-df2b1c0c.pth

./tools/dist_test.sh configs/swinir/swinir_s128w8d6e180_8xb1-lr2e-4-1600k_dfwb-
↪colorDN50.py https://download.openmmlab.com/mmediting/swinir/swinir_s128w8d6e180_
↪8xb1-lr2e-4-1600k_dfwb-colorDN50-e369874c.pth

## 006 JPEG Compression Artifact Reduction (middle size, using window_size=7 because␣
↪JPEG encoding uses 8x8 blocks)
## grayscale
./tools/dist_test.sh configs/swinir/swinir_s126w7d6e180_8xb1-lr2e-4-1600k_dfwb-
↪grayCAR10.py https://download.openmmlab.com/mmediting/swinir/swinir_s126w7d6e180_
↪8xb1-lr2e-4-1600k_dfwb-grayCAR10-da93c8e9.pth

./tools/dist_test.sh configs/swinir/swinir_s126w7d6e180_8xb1-lr2e-4-1600k_dfwb-
↪grayCAR20.py https://download.openmmlab.com/mmediting/swinir/swinir_s126w7d6e180_
↪8xb1-lr2e-4-1600k_dfwb-grayCAR20-d47367b1.pth

./tools/dist_test.sh configs/swinir/swinir_s126w7d6e180_8xb1-lr2e-4-1600k_dfwb-
↪grayCAR30.py https://download.openmmlab.com/mmediting/swinir/swinir_s126w7d6e180_
↪8xb1-lr2e-4-1600k_dfwb-grayCAR30-52c083cf.pth

./tools/dist_test.sh configs/swinir/swinir_s126w7d6e180_8xb1-lr2e-4-1600k_dfwb-
↪grayCAR40.py https://download.openmmlab.com/mmediting/swinir/swinir_s126w7d6e180_
↪8xb1-lr2e-4-1600k_dfwb-grayCAR40-803e8d9b.pth

## color
./tools/dist_test.sh configs/swinir/swinir_s126w7d6e180_8xb1-lr2e-4-1600k_dfwb-
↪colorCAR10.py https://download.openmmlab.com/mmediting/swinir/swinir_s126w7d6e180_
↪8xb1-lr2e-4-1600k_dfwb-colorCAR10-09aafadc.pth

./tools/dist_test.sh configs/swinir/swinir_s126w7d6e180_8xb1-lr2e-4-1600k_dfwb-
↪colorCAR20.py https://download.openmmlab.com/mmediting/swinir/swinir_s126w7d6e180_
↪8xb1-lr2e-4-1600k_dfwb-colorCAR20-b8a42b5e.pth

./tools/dist_test.sh configs/swinir/swinir_s126w7d6e180_8xb1-lr2e-4-1600k_dfwb-
↪colorCAR30.py https://download.openmmlab.com/mmediting/swinir/swinir_s126w7d6e180_
↪8xb1-lr2e-4-1600k_dfwb-colorCAR30-e9fe6859.pth
```

(下页继续)

(续上页)

```
./tools/dist_test.sh configs/swinir/swinir_s126w7d6e180_8xb1-lr2e-4-1600k_dfwb-
↪colorCAR40.py https://download.openmmlab.com/mmediting/swinir/swinir_s126w7d6e180_
↪8xb1-lr2e-4-1600k_dfwb-colorCAR30-e9fe6859.pth
```

更多细节可以参考 train_test.md 中的 **Test a pre-trained model** 部分。

CHAPTER 31

条件生成对抗网络

31.1 概览

- 预训练权重个数: 7
- 配置文件个数: 0
- 论文个数: 1
  - ALGORITHM: 1

31.2 BigGAN (ICLR’2019)

Large Scale GAN Training for High Fidelity Natural Image Synthesis

任务: 条件生成对抗网络

31.2.1 Abstract

Despite recent progress in generative image modeling, successfully generating high-resolution, diverse samples from complex datasets such as ImageNet remains an elusive goal. To this end, we train Generative Adversarial Networks at the largest scale yet attempted, and study the instabilities specific to such scale. We find that applying orthogonal regularization to the generator renders it amenable to a simple “truncation trick,” allowing fine control over the trade-off between sample fidelity and variety by reducing the variance of the Generator’s input. Our modifications lead to models which set the new state of the art in class-conditional image synthesis. When trained on ImageNet at 128x128 resolution, our

models (BigGANs) achieve an Inception Score (IS) of 166.5 and Frechet Inception Distance (FID) of 7.4, improving over the previous best IS of 52.52 and FID of 18.6.

31.2.2 Introduction

BigGAN/BigGAN-Deep 是一个条件生成模型，通过扩大批次大小和模型参数的数量，可以生成高分辨率和高质量的图像。

我们已经在 Cifar10（32x32）中完成了 BigGAN 的训练，并在 ImageNet1k（128x128）上对齐了训练性能。下面是一些抽样的结果，供你参考。

对我们训练的 BigGAN 进行评估.

关于可复现性的说明

BigGAN 128x128 模型是用 V100 GPU 和 CUDA 10.1 训练的，用 A100 和 CUDA 11.3 很难再现结果。如果你对复现有任何想法，请随时与我们联系。

31.2.3 转换后的权重

由于我们还没有完成对模型的训练，我们为您提供了几个已经评估过的预训练权重。这里，我们指的是BigGAN-PyTorch和pytorch-pretrained-BigGAN。

下面提供了评估结果和下载链接

采样结果如下。

```
python demo/conditional_demo.py CONFIG_PATH CKPT_PATH --sample-cfg truncation=0.4 ##␣
↪set truncation value as you want
```

对于转换后的权重，我们在 configs/_base_/models 下提供模型配置，列举如下。

```
## biggan_cvt-BigGAN-PyTorch-rgb_imagenet1k-128x128.py
## biggan-deep_cvt-hugging-face-rgb_imagenet1k-128x128.py
## biggan-deep_cvt-hugging-face_rgb_imagenet1k-256x256.py
## biggan-deep_cvt-hugging-face_rgb_imagenet1k-512x512.py
```

31.2.4 Interpolation

要在 BigGAN（或其他条件模型）上执行图像插值，请运行

```
python apps/conditional_interpolate.py CONFIG_PATH  CKPT_PATH  --samples-path SAMPLES_
↪PATH
```

要在 BigGAN 上进行具有固定噪声的图像插值，请运行

```
python apps/conditional_interpolate.py CONFIG_PATH  CKPT_PATH  --samples-path SAMPLES_
↪PATH --fix-z
```

```
python apps/conditional_interpolate.py CONFIG_PATH  CKPT_PATH  --samples-path SAMPLES_
↪PATH --fix-y
```

31.2.5 Citation

```
@inproceedings{
    brock2018large,
    title={Large Scale {GAN} Training for High Fidelity Natural Image Synthesis},
    author={Andrew Brock and Jeff Donahue and Karen Simonyan},
    booktitle={International Conference on Learning Representations},
    year={2019},
    url={https://openreview.net/forum?id=B1xsqj09Fm},
}
```

CHAPTER 32

视频插帧

32.1 概览

- 预训练权重个数: 7
- 配置文件个数: 0
- 论文个数: 3
  - ALGORITHM: 3

32.2 FLAVR (arXiv'2020)

FLAVR: Flow-Agnostic Video Representations for Fast Frame Interpolation

任务: 视频插帧

32.2.1 预训练模型测试结果

在 RGB 通道上评估。评估指标 `PSNR / SSIM`。

注：FLAVR 中的 8 倍视频插帧算法将会在未来版本中支持。

32.2.2 快速开始

训练

您可以使用以下命令来训练模型。

```
## CPU上训练
CUDA_VISIBLE_DEVICES=-1 python tools/train.py configs/flavr/flavr_in4out1_8xb4_
↪vimeo90k-septuplet.py

## 单个GPU上训练
python tools/train.py configs/flavr/flavr_in4out1_8xb4_vimeo90k-septuplet.py

## 多个GPU上训练
./tools/dist_train.sh configs/flavr/flavr_in4out1_8xb4_vimeo90k-septuplet.py 8
```

更多细节可以参考 train_test.md 中的 **Train a model** 部分。

测试

您可以使用以下命令来测试模型。

```
## CPU上测试
CUDA_VISIBLE_DEVICES=-1 python tools/test.py configs/flavr/flavr_in4out1_8xb4_
↪vimeo90k-septuplet.py https://download.openmmlab.com/mmediting/video_interpolators/
↪flavr/flavr_in4out1_g8b4_vimeo90k_septuplet_20220509-c2468995.pth

## 单个GPU上测试
python tools/test.py configs/flavr/flavr_in4out1_8xb4_vimeo90k-septuplet.py https://
↪download.openmmlab.com/mmediting/video_interpolators/flavr/flavr_in4out1_g8b4_
↪vimeo90k_septuplet_20220509-c2468995.pth

## 多个GPU上测试
./tools/dist_test.sh configs/flavr/flavr_in4out1_8xb4_vimeo90k-septuplet.py https://
↪download.openmmlab.com/mmediting/video_interpolators/flavr/flavr_in4out1_g8b4_
↪vimeo90k_septuplet_20220509-c2468995.pth 8
```

更多细节可以参考 train_test.md 中的 **Test a pre-trained model** 部分。

32.2.3 Citation

```
@article{kalluri2020flavr,
  title={Flavr: Flow-agnostic video representations for fast frame interpolation},
  author={Kalluri, Tarun and Pathak, Deepak and Chandraker, Manmohan and Tran, Du},
  journal={arXiv preprint arXiv:2012.08512},
  year={2020}
}
```

32.3 CAIN (AAAI'2020)

任务: 视频插帧

```
@inproceedings{choi2020channel,
  title={Channel attention is all you need for video frame interpolation},
  author={Choi, Myungsub and Kim, Heewon and Han, Bohyung and Xu, Ning and Lee,␣
↪Kyoung Mu},
  booktitle={Proceedings of the AAAI Conference on Artificial Intelligence},
  volume={34},
  number={07},
  pages={10663--10671},
  year={2020}
}
```

在 RGB 通道上进行评估。我们使用 PSNR 和 SSIM 作为指标。学习率调整策略是等间隔调整策略。

32.3.1 快速开始

训练

您可以使用以下命令来训练模型。

```
## CPU上训练
CUDA_VISIBLE_DEVICES=-1 python tools/train.py configs/cain/cain_g1b32_1xb5_vimeo90k-
↪triplet.py

## 单个GPU上训练
python tools/train.py configs/cain/cain_g1b32_1xb5_vimeo90k-triplet.py

## 多个GPU上训练
./tools/dist_train.sh configs/cain/cain_g1b32_1xb5_vimeo90k-triplet.py 8
```

更多细节可以参考 train_test.md 中的 **Train a model** 部分。

测试

您可以使用以下命令来测试模型。

```
## CPU上测试
CUDA_VISIBLE_DEVICES=-1 python tools/test.py configs/cain/cain_g1b32_1xb5_vimeo90k-
↪triplet.py https://download.openmmlab.com/mmediting/video_interpolators/cain/cain_
↪b5_g1b32_vimeo90k_triplet_20220530-3520b00c.pth

## 单个GPU上测试
python tools/test.py configs/cain/cain_g1b32_1xb5_vimeo90k-triplet.py https://
↪download.openmmlab.com/mmediting/video_interpolators/cain/cain_b5_g1b32_vimeo90k_
↪triplet_20220530-3520b00c.pth

## 多个GPU上测试
./tools/dist_test.sh configs/cain/cain_g1b32_1xb5_vimeo90k-triplet.py https://
↪download.openmmlab.com/mmediting/video_interpolators/cain/cain_b5_g1b32_vimeo90k_
↪triplet_20220530-3520b00c.pth 8
```

更多细节可以参考 train_test.md 中的 **Test a pre-trained model** 部分。

32.4 TOFlow (IJCV'2019)

> Video Enhancement with Task-Oriented Flow
>
> **任务**: 视频插帧, 视频超分辨率

32.4.1 预训练模型测试结果

在 RGB 通道上评估。评估指标 `PSNR / SSIM`。

注: 由于 `batch_size=1` 预训练的 SPyNet 不包含 BN 层，这与 `https://github.com/Coldog2333/pytoflow` 一致.

32.4.2 快速开始

训练

您可以使用以下命令来训练模型。

TOF 的训练仅支持视频插帧任务。

```
## CPU上训练
CUDA_VISIBLE_DEVICES=-1 python tools/train.py configs/tof/tof_spynet-chair-wobn_1xb1_
↪vimeo90k-triplet.py
```

(下页继续)

(续上页)

```
## 单个GPU上训练
python tools/train.py configs/tof/tof_spynet-chair-wobn_1xb1_vimeo90k-triplet.py

## 多个GPU上训练
./tools/dist_train.sh configs/tof/tof_spynet-chair-wobn_1xb1_vimeo90k-triplet.py 8
```

更多细节可以参考 train_test.md 中的 **Train a model** 部分。

测试

您可以使用以下命令来测试模型。

TOF 的测试支持视频插帧和视频超分辨率两种任务。

任务 1: 视频插帧

```
## CPU上测试
CUDA_VISIBLE_DEVICES=-1 python tools/test.py configs/tof/tof_spynet-chair-wobn_1xb1_
↪vimeo90k-triplet.py https://download.openmmlab.com/mmediting/video_interpolators/
↪toflow/pretrained_spynet_chair_20220321-4d82e91b.pth

## 单个GPU上测试
python tools/test.py configs/tof/tof_spynet-chair-wobn_1xb1_vimeo90k-triplet.py␣
↪https://download.openmmlab.com/mmediting/video_interpolators/toflow/pretrained_
↪spynet_chair_20220321-4d82e91b.pth

## 多个GPU上测试
./tools/dist_test.sh configs/tof/tof_spynet-chair-wobn_1xb1_vimeo90k-triplet.py␣
↪https://download.openmmlab.com/mmediting/video_interpolators/toflow/pretrained_
↪spynet_chair_20220321-4d82e91b.pth 8
```

任务 2: 视频超分辨率

```
## CPU上测试
CUDA_VISIBLE_DEVICES=-1 python tools/test.py configs/tof/tof_x4_official_vimeo90k.py␣
↪https://download.openmmlab.com/mmediting/restorers/tof/tof_x4_vimeo90k_official-
↪a569ff50.pth

## 单个GPU上测试
python tools/test.py configs/tof/tof_x4_official_vimeo90k.py https://download.
↪openmmlab.com/mmediting/restorers/tof/tof_x4_vimeo90k_official-a569ff50.pth

## 多个GPU上测试
./tools/dist_test.sh configs/tof/tof_x4_official_vimeo90k.py https://download.
↪openmmlab.com/mmediting/restorers/tof/tof_x4_vimeo90k_official-a569ff50.pth 8
```

更多细节可以参考 train_test.md 中的 **Test a pre-trained model** 部分。

32.4.3 Citation

```
@article{xue2019video,
  title={Video enhancement with task-oriented flow},
  author={Xue, Tianfan and Chen, Baian and Wu, Jiajun and Wei, Donglai and Freeman,␣
↪William T},
  journal={International Journal of Computer Vision},
  volume={127},
  number={8},
  pages={1106--1125},
  year={2019},
  publisher={Springer}
}
```

CHAPTER 33

图文生成

33.1 概览

- 预训练权重个数: 2
- 配置文件个数: 0
- 论文个数: 1
  - ALGORITHM: 1

33.2 Disco Diffusion (2022)

Disco Diffusion

任务: 图文生成, 图像到图像的翻译, 扩散模型

33.2.1 摘要

Disco Diffusion(DD) 是一个 Google Colab 笔记本，它利用一种叫做 CLIP-Guided Diffusion 的人工智能图像生成技术，让你从文本输入中创造出引人注目的精美图像。

由 Somnai 创建，由 Gandamu 改进，并建立在 RiversHaveWings、nshepperd 和许多其他人的工作之上。更多细节见 Credits。

33.2.2 模型与结果

我们已经转换了几个 unet 的权重，并提供相关的配置文件。在 Tutorial 中可以看到更多关于不同 unet 的细节。

33.2.3 待办列表

- [x] 图文生成
- [x] 图像到图像的翻译
- [x] Imagenet, portrait 扩散模型
- [] 像素艺术，水彩，科幻小说的扩散模型
- [] 支持图像提示
- [] 支持视频生成
- [] 支持更快的采样器 (plms，dpm-solver 等)

我们很欢迎社区用户支持这些项目和任何其他有趣的工作!

33.2.4 Quick Start

运行以下代码，你可以使用文本生成图像。

```
from mmengine import Config, MODELS
from mmagic.utils import register_all_modules
from torchvision.utils import save_image

register_all_modules()

disco = MODELS.build(
    Config.fromfile('configs/disco_diffusion/disco-baseline.py').model).cuda().eval()
text_prompts = {
    0: [
        "A beautiful painting of a singular lighthouse, shining its light across a␣
↪tumultuous sea of blood by greg rutkowski and thomas kinkade, Trending on␣
↪artstation.",
        "yellow color scheme"
    ]
}
image = disco.infer(
    height=768,
    width=1280,
    text_prompts=text_prompts,
```

(下页继续)

(续上页)

```
    show_progress=True,
    num_inference_steps=250,
    eta=0.8)['samples']
save_image(image, "image.png")
```

33.2.5 教程

考虑到 `disco-diffusion` 包含许多可调整的参数，我们为用户提供了一个 jupyter-notebook/colab的教程，展示了不同参数的含义，并给出相应的调整结果。请参考Disco Sheet。

33.2.6 鸣谢

Since our adaptation of disco-diffusion are heavily influenced by disco colab, here we copy the credits below.

Modified by Daniel Russell (https://github.com/russelldc, https://twitter.com/danielrussruss) to include (hopefully) optimal params for quick generations in 15-100 timesteps rather than 1000, as well as more robust augmentations.

Further improvements from Dango233 and nshepperd helped improve the quality of diffusion in general, and especially so for shorter runs like this notebook aims to achieve.

Vark added code to load in multiple Clip models at once, which all prompts are evaluated against, which may greatly improve accuracy.

The latest zoom, pan, rotation, and keyframes features were taken from Chigozie Nri' s VQGAN Zoom Notebook (https://github.com/chigozienri, https://twitter.com/chigozienri)

Advanced DangoCutn Cutout method is also from Dango223.

–

Disco:

Somnai (https://twitter.com/Somnai_dreams) added Diffusion Animation techniques, QoL improvements and various implementations of tech and techniques, mostly listed in the changelog below.

3D animation implementation added by Adam Letts (https://twitter.com/gandamu_ml) in collaboration with Somnai. Creation of disco.py and ongoing maintenance.

Turbo feature by Chris Allen (https://twitter.com/zippy731)

Improvements to ability to run on local systems, Windows support, and dependency installation by HostsServer (https://twitter.com/HostsServer)

VR Mode by Tom Mason (https://twitter.com/nin_artificial)

Horizontal and Vertical symmetry functionality by nshepperd. Symmetry transformation_steps by huemin (https://twitter.com/huemin_art). Symmetry integration into Disco Diffusion by Dmitrii Tochilkin (https://twitter.com/cut_pow).

Warp and custom model support by Alex Spirin (https://twitter.com/devdef).

Pixel Art Diffusion, Watercolor Diffusion, and Pulp SciFi Diffusion models from KaliYuga (https://twitter.com/KaliYuga_ai). Follow KaliYuga's Twitter for the latest models and for notebooks with specialized settings.

Integration of OpenCLIP models and initiation of integration of KaliYuga models by Palmweaver / Chris Scalf (https://twitter.com/ChrisScalf11)

Integrated portrait_generator_v001 from Felipe3DArtist (https://twitter.com/Felipe3DArtist)

33.2.7 Citation

```
@misc{github,
  author={alembics},
  title={disco-diffusion},
  year={2022},
  url={https://github.com/alembics/disco-diffusion},
}
```

CHAPTER 34

图像上色

34.1 概览

- 预训练权重个数: 1
- 配置文件个数: 0
- 论文个数: 1
  - ALGORITHM: 1

34.2 Instance-aware Image Colorization (CVPR’2020)

Instance-Aware Image Colorization

任务: 图像上色

34.2.1 摘要

Image colorization is inherently an ill-posed problem with multi-modal uncertainty. Previous methods leverage the deep neural network to map input grayscale images to plausible color outputs directly. Although these learning-based methods have shown impressive performance, they usually fail on the input images that contain multiple objects. The leading cause is that existing models perform learning and colorization on the entire image. In the absence of a clear figure-ground separation, these models cannot effectively locate and learn meaningful object-level semantics. In this paper, we propose a method for achieving instance-aware colorization. Our network architecture leverages an off-the-shelf object

detector to obtain cropped object images and uses an instance colorization network to extract object-level features. We use a similar network to extract the full-image features and apply a fusion module to full object-level and image-level features to predict the final colors. Both colorization networks and fusion modules are learned from a large-scale dataset. Experimental results show that our work outperforms existing methods on different quality metrics and achieves state-of-the-art performance on image colorization.

34.2.2 结果和模型

34.2.3 快速开始

您可以使用以下命令来对一张图像进行上色。

```
python demo/colorization_demo.py configs/inst_colorization/inst-colorizatioon_full_
↪official_cocostuff-256x256.py https://download.openmmlab.com/mmediting/inst_
↪colorization/inst-colorizatioon_full_official_cocostuff-256x256-5b9d4eee.pth input.
↪jpg output.jpg
```

更多细节可以参考 Tutorial 3: inference with pre-trained models。

```
@inproceedings{Su-CVPR-2020,
  author = {Su, Jheng-Wei and Chu, Hung-Kuo and Huang, Jia-Bin},
  title = {Instance-aware Image Colorization},
  booktitle = {IEEE Conference on Computer Vision and Pattern Recognition (CVPR)},
  year = {2020}
}
```

CHAPTER 35

图像去噪，图像去模糊，图像去雨

35.1 概览

- 预训练权重个数: 13
- 配置文件个数: 0
- 论文个数: 0
  - ALGORITHM: 1

35.2 Restormer (CVPR’2022)

Restormer: Efficient Transformer for High-Resolution Image Restoration

任务: 图像去噪，图像去模糊，图像去雨

35.2.1 各个任务下模型的测试结果

图像去雨

所有数据集均在 Y 通道上进行测试，测试指标为 PSNR 和 SSIM。

图像去模糊

Gopro 和 HIDE 数据集上使用 RGB 通道测试，ReakBlur-J 和 ReakBlur-R 数据集使用 Y 通道测试。测试指标为 PSNR 和 SSIM。

图像去失焦模糊

所有指标均在 RGB 通道上进行测试。测试指标为 PSNR、SSIM、MAE 和 LPIPS.

图像高斯噪声去除

灰度图的高斯噪声

使用 PSNR 和 SSIM 指标对数据集上的灰度图进行测试。

> 上面三行代表每个噪声等级训练一个单独的模型，下面三行代表学习一个单一的模型来处理各种噪音水平。

彩色图像的高斯噪声

所有指标均在 RGB 通道上进行测试，测试指标为 PSNR 和 SSIM。

> 上面三行代表每个噪声等级训练一个单独的模型，下面三行代表学习一个单一的模型来处理各种噪音水平。

真实场景图像去噪

所有指标均在 RGB 通道上进行测试，测试指标为 PSNR 和 SSIM。

35.2.2 使用方法

训练

可以参考 train_test.md 中的 **Train a model** 部分。

测试

您可以使用以下命令来测试模型。

```
## cpu test
## Deraining
CUDA_VISIBLE_DEVICES=-1 python tools/test.py configs/restormer/restormer_official_
↪rain13k.py https://download.openmmlab.com/mmediting/restormer/restormer_official_
↪rain13k-2be7b550.pth

## Motion Deblurring
```

(下页继续)

(续上页)

```
CUDA_VISIBLE_DEVICES=-1 python tools/test.py configs/restormer/restormer_official_
↪gopro.py https://download.openmmlab.com/mmediting/restormer/restormer_official_
↪gopro-db7363a0.pth

## Defocus Deblurring
## Single
CUDA_VISIBLE_DEVICES=-1 python tools/test.py configs/restormer/restormer_official_
↪dpdd-dual.py https://download.openmmlab.com/mmediting/restormer/restormer_official_
↪dpdd-single-6bc31582.pth
## Dual
CUDA_VISIBLE_DEVICES=-1 python tools/test.py configs/restormer/restormer_official_
↪dpdd-single.py https://download.openmmlab.com/mmediting/restormer/restormer_
↪official_dpdd-dual-52c94c00.pth

## Gaussian Denoising
## Test Grayscale Gaussian Noise
## sigma15
CUDA_VISIBLE_DEVICES=-1 python tools/test.py configs/restormer/restormer_official_
↪dfwb-gray-sigma15.py https://download.openmmlab.com/mmediting/restormer/restormer_
↪official_dfwb-gray-sigma15-da74417f.pth

CUDA_VISIBLE_DEVICES=-1 python tools/test.py configs/restormer/restormer_official_
↪dfwb-gray-sigma15.py https://download.openmmlab.com/mmediting/restormer/restormer_
↪official_dfwb-gray-blind-5f094bcc.pth

## sigma25
CUDA_VISIBLE_DEVICES=-1 python tools/test.py configs/restormer/restormer_official_
↪dfwb-gray-sigma25.py https://download.openmmlab.com/mmediting/restormer/restormer_
↪official_dfwb-gray-sigma25-08010841.pth

CUDA_VISIBLE_DEVICES=-1 python tools/test.py configs/restormer/restormer_official_
↪dfwb-gray-sigma25.py https://download.openmmlab.com/mmediting/restormer/restormer_
↪official_dfwb-gray-blind-5f094bcc.pth

## sigma50
CUDA_VISIBLE_DEVICES=-1 python tools/test.py configs/restormer/restormer_official_
↪dfwb-gray-sigma50.py https://download.openmmlab.com/mmediting/restormer/restormer_
↪official_dfwb-gray-sigma50-ee852dfe.pth

CUDA_VISIBLE_DEVICES=-1 python tools/test.py configs/restormer/restormer_official_
↪dfwb-gray-sigma50.py https://download.openmmlab.com/mmediting/restormer/restormer_
↪official_dfwb-gray-blind-5f094bcc.pth
```

(下页继续)

(续上页)

```
## Test Color Gaussian Noise
## sigma15
CUDA_VISIBLE_DEVICES=-1 python tools/test.py configs/restormer/restormer_official_
↪dfwb-color-sigma15.py https://download.openmmlab.com/mmediting/restormer/restormer_
↪official_dfwb-color-sigma15-012ceb71.pth

CUDA_VISIBLE_DEVICES=-1 python tools/test.py configs/restormer/restormer_official_
↪dfwb-color-sigma15.py https://download.openmmlab.com/mmediting/restormer/restormer_
↪official_dfwb-color-blind-dfd03c9f.pth

## sigma25
CUDA_VISIBLE_DEVICES=-1 python tools/test.py configs/restormer/restormer_official_
↪dfwb-color-sigma25.py https://download.openmmlab.com/mmediting/restormer/restormer_
↪official_dfwb-color-sigma25-e307f222.pth

CUDA_VISIBLE_DEVICES=-1 python tools/test.py configs/restormer/restormer_official_
↪dfwb-color-sigma25.py https://download.openmmlab.com/mmediting/restormer/restormer_
↪official_dfwb-color-blind-dfd03c9f.pth

## sigma50
CUDA_VISIBLE_DEVICES=-1 python tools/test.py configs/restormer/restormer_official_
↪dfwb-color-sigma50.py https://download.openmmlab.com/mmediting/restormer/restormer_
↪official_dfwb-color-sigma50-a991983d.pth

CUDA_VISIBLE_DEVICES=-1 python tools/test.py configs/restormer/restormer_official_
↪dfwb-color-sigma50.py https://download.openmmlab.com/mmediting/restormer/restormer_
↪official_dfwb-color-blind-dfd03c9f.pth

## single-gpu test
## Deraining
python tools/test.py configs/restormer/restormer_official_rain13k.py https://download.
↪openmmlab.com/mmediting/restormer/restormer_official_rain13k-2be7b550.pth

## Motion Deblurring
python tools/test.py configs/restormer/restormer_official_gopro.py https://download.
↪openmmlab.com/mmediting/restormer/restormer_official_gopro-db7363a0.pth

## Defocus Deblurring
## Single
python tools/test.py configs/restormer/restormer_official_dpdd-dual.py https://
↪download.openmmlab.com/mmediting/restormer/restormer_official_dpdd-single-6bc31582.
↪pth
## Dual
```

(下页继续)

(续上页)

```
python tools/test.py configs/restormer/restormer_official_dpdd-single.py https://
↪download.openmmlab.com/mmediting/restormer/restormer_official_dpdd-dual-52c94c00.pth

## Gaussian Denoising
## Test Grayscale Gaussian Noise
## sigma15
python tools/test.py configs/restormer/restormer_official_dfwb-gray-sigma15.py https:/
↪/download.openmmlab.com/mmediting/restormer/restormer_official_dfwb-gray-sigma15-
↪da74417f.pth

python tools/test.py configs/restormer/restormer_official_dfwb-gray-sigma15.py https:/
↪/download.openmmlab.com/mmediting/restormer/restormer_official_dfwb-gray-blind-
↪5f094bcc.pth

## sigma25
python tools/test.py configs/restormer/restormer_official_dfwb-gray-sigma25.py https:/
↪/download.openmmlab.com/mmediting/restormer/restormer_official_dfwb-gray-sigma25-
↪08010841.pth

python tools/test.py configs/restormer/restormer_official_dfwb-gray-sigma25.py https:/
↪/download.openmmlab.com/mmediting/restormer/restormer_official_dfwb-gray-blind-
↪5f094bcc.pth

## sigma50
python tools/test.py configs/restormer/restormer_official_dfwb-gray-sigma50.py https:/
↪/download.openmmlab.com/mmediting/restormer/restormer_official_dfwb-gray-sigma50-
↪ee852dfe.pth

python tools/test.py configs/restormer/restormer_official_dfwb-gray-sigma50.py https:/
↪/download.openmmlab.com/mmediting/restormer/restormer_official_dfwb-gray-blind-
↪5f094bcc.pth

## Test Color Gaussian Noise
## sigma15
python tools/test.py configs/restormer/restormer_official_dfwb-color-sigma15.py␣
↪https://download.openmmlab.com/mmediting/restormer/restormer_official_dfwb-color-
↪sigma15-012ceb71.pth

python tools/test.py configs/restormer/restormer_official_dfwb-color-sigma15.py␣
↪https://download.openmmlab.com/mmediting/restormer/restormer_official_dfwb-color-
↪blind-dfd03c9f.pth

## sigma25
```

(下页继续)

(续上页)

```
python tools/test.py configs/restormer/restormer_official_dfwb-color-sigma25.py␣
↪https://download.openmmlab.com/mmediting/restormer/restormer_official_dfwb-color-
↪sigma25-e307f222.pth

python tools/test.py configs/restormer/restormer_official_dfwb-color-sigma25.py␣
↪https://download.openmmlab.com/mmediting/restormer/restormer_official_dfwb-color-
↪blind-dfd03c9f.pth

## sigma50
python tools/test.py configs/restormer/restormer_official_dfwb-color-sigma50.py␣
↪https://download.openmmlab.com/mmediting/restormer/restormer_official_dfwb-color-
↪sigma50-a991983d.pth

python tools/test.py configs/restormer/restormer_official_dfwb-color-sigma50.py␣
↪https://download.openmmlab.com/mmediting/restormer/restormer_official_dfwb-color-
↪blind-dfd03c9f.pth


## multi-gpu test
## Deraining
./tools/dist_test.sh configs/restormer/restormer_official_rain13k.py https://download.
↪openmmlab.com/mmediting/restormer/restormer_official_rain13k-2be7b550.pth

## Motion Deblurring
./tools/dist_test.sh configs/restormer/restormer_official_gopro.py https://download.
↪openmmlab.com/mmediting/restormer/restormer_official_gopro-db7363a0.pth

## Defocus Deblurring
## Single
./tools/dist_test.sh configs/restormer/restormer_official_dpdd-dual.py https://
↪download.openmmlab.com/mmediting/restormer/restormer_official_dpdd-single-6bc31582.
↪pth
## Dual
./tools/dist_test.sh configs/restormer/restormer_official_dpdd-single.py https://
↪download.openmmlab.com/mmediting/restormer/restormer_official_dpdd-dual-52c94c00.pth

## Gaussian Denoising
## Test Grayscale Gaussian Noise
## sigma15
./tools/dist_test.sh configs/restormer/restormer_official_dfwb-gray-sigma15.py https:/
↪/download.openmmlab.com/mmediting/restormer/restormer_official_dfwb-gray-sigma15-
↪da74417f.pth
```

(下页继续)

(续上页)

```
./tools/dist_test.sh configs/restormer/restormer_official_dfwb-gray-sigma15.py https:/
↪/download.openmmlab.com/mmediting/restormer/restormer_official_dfwb-gray-blind-
↪5f094bcc.pth

## sigma25
./tools/dist_test.sh configs/restormer/restormer_official_dfwb-gray-sigma25.py https:/
↪/download.openmmlab.com/mmediting/restormer/restormer_official_dfwb-gray-sigma25-
↪08010841.pth

./tools/dist_test.sh configs/restormer/restormer_official_dfwb-gray-sigma25.py https:/
↪/download.openmmlab.com/mmediting/restormer/restormer_official_dfwb-gray-blind-
↪5f094bcc.pth

## sigma50
./tools/dist_test.sh configs/restormer/restormer_official_dfwb-gray-sigma50.py https:/
↪/download.openmmlab.com/mmediting/restormer/restormer_official_dfwb-gray-sigma50-
↪ee852dfe.pth

./tools/dist_test.sh configs/restormer/restormer_official_dfwb-gray-sigma50.py https:/
↪/download.openmmlab.com/mmediting/restormer/restormer_official_dfwb-gray-blind-
↪5f094bcc.pth

## Test Color Gaussian Noise
## sigma15
./tools/dist_test.sh configs/restormer/restormer_official_dfwb-color-sigma15.py␣
↪https://download.openmmlab.com/mmediting/restormer/restormer_official_dfwb-color-
↪sigma15-012ceb71.pth

./tools/dist_test.sh configs/restormer/restormer_official_dfwb-color-sigma15.py␣
↪https://download.openmmlab.com/mmediting/restormer/restormer_official_dfwb-color-
↪blind-dfd03c9f.pth

## sigma25
./tools/dist_test.sh configs/restormer/restormer_official_dfwb-color-sigma25.py␣
↪https://download.openmmlab.com/mmediting/restormer/restormer_official_dfwb-color-
↪sigma25-e307f222.pth

./tools/dist_test.sh configs/restormer/restormer_official_dfwb-color-sigma25.py␣
↪https://download.openmmlab.com/mmediting/restormer/restormer_official_dfwb-color-
↪blind-dfd03c9f.pth

## sigma50
./tools/dist_test.sh configs/restormer/restormer_official_dfwb-color-sigma50.py␣
↪https://download.openmmlab.com/mmediting/restormer/restormer_official_dfwb-color-
↪sigma50-a991983d.pth
```

(下页继续)

(续上页)

```
./tools/dist_test.sh configs/restormer/restormer_official_dfwb-color-sigma50.py␣
↪https://download.openmmlab.com/mmediting/restormer/restormer_official_dfwb-color-
↪blind-dfd03c9f.pth
```

更多细节可以参考 train_test.md 中的 **Test a pre-trained model** 部分。

CHAPTER 36

图像去噪

36.1 概览

- 预训练权重个数: 29
- 配置文件个数: 0
- 论文个数: 1
  - ALGORITHM: 1

36.2 SwinIR (ICCVW’2021)

任务: 图像超分辨率, 图像去噪, JPEG 压缩伪影移除

```
@inproceedings{liang2021swinir,
  title={Swinir: Image restoration using swin transformer},
  author={Liang, Jingyun and Cao, Jiezhang and Sun, Guolei and Zhang, Kai and Van␣
↪Gool, Luc and Timofte, Radu},
  booktitle={Proceedings of the IEEE/CVF International Conference on Computer Vision},
  pages={1833--1844},
  year={2021}
}
```

36.2.1 Classical Image Super-Resolution

在 Y 通道上进行评估，在评估之前裁剪每个边界中的 scale 像素。我们使用 PSNR 和 SSIM 作为指标。

36.2.2 Lightweight Image Super-Resolution

在 Y 通道上进行评估，在评估之前裁剪每个边界中的 scale 像素。我们使用 PSNR 和 SSIM 作为指标。

36.2.3 Real-World Image Super-Resolution

在 Y 通道上进行评估。我们使用 NIQE 作为指标。

36.2.4 Grayscale Image Deoising

在灰度图上进行评估。我们使用 PSNR 作为指标。

36.2.5 Color Image Deoising

在 RGB 通道上进行评估。我们使用 PSNR 作为指标。

36.2.6 JPEG Compression Artifact Reduction (grayscale)

在灰度图上进行评估。我们使用 PSNR 和 SSIM 作为指标。

36.2.7 JPEG Compression Artifact Reduction (color)

在 RGB 通道上进行评估。我们使用 PSNR 和 SSIM 作为指标。

36.2.8 快速开始

训练

您可以使用以下命令来训练模型。

```
## CPU上训练
## 001 Classical Image Super-Resolution (middle size)
## (setting1: when model is trained on DIV2K and with training_patch_size=48)
CUDA_VISIBLE_DEVICES=-1 python tools/train.py configs/swinir/swinir_x2s48w8d6e180_
↪8xb4-lr2e-4-500k_div2k.py
CUDA_VISIBLE_DEVICES=-1 python tools/train.py configs/swinir/swinir_x3s48w8d6e180_
↪8xb4-lr2e-4-500k_div2k.py
```

(下页继续)

(续上页)

```
CUDA_VISIBLE_DEVICES=-1 python tools/train.py configs/swinir/swinir_x4s48w8d6e180_
↪8xb4-lr2e-4-500k_div2k.py

## (setting2: when model is trained on DIV2K+Flickr2K and with training_patch_size=64)
CUDA_VISIBLE_DEVICES=-1 python tools/train.py configs/swinir/swinir_x2s64w8d6e180_
↪8xb4-lr2e-4-500k_df2k.py
CUDA_VISIBLE_DEVICES=-1 python tools/train.py configs/swinir/swinir_x3s64w8d6e180_
↪8xb4-lr2e-4-500k_df2k.py
CUDA_VISIBLE_DEVICES=-1 python tools/train.py configs/swinir/swinir_x4s64w8d6e180_
↪8xb4-lr2e-4-500k_df2k.py

## 002 Lightweight Image Super-Resolution (small size)
CUDA_VISIBLE_DEVICES=-1 python tools/train.py configs/swinir/swinir_x2s64w8d4e60_8xb4-
↪lr2e-4-500k_div2k.py
CUDA_VISIBLE_DEVICES=-1 python tools/train.py configs/swinir/swinir_x3s64w8d4e60_8xb4-
↪lr2e-4-500k_div2k.py
CUDA_VISIBLE_DEVICES=-1 python tools/train.py configs/swinir/swinir_x4s64w8d4e60_8xb4-
↪lr2e-4-500k_div2k.py

## 003 Real-World Image Super-Resolution
CUDA_VISIBLE_DEVICES=-1 python tools/train.py configs/swinir/swinir_gan-x2s64w8d6e180_
↪8xb4-lr1e-4-600k_df2k-ost.py
CUDA_VISIBLE_DEVICES=-1 python tools/train.py configs/swinir/swinir_psnr-
↪x2s64w8d6e180_8xb4-lr1e-4-600k_df2k-ost.py
CUDA_VISIBLE_DEVICES=-1 python tools/train.py configs/swinir/swinir_gan-x4s64w8d6e180_
↪8xb4-lr1e-4-600k_df2k-ost.py
CUDA_VISIBLE_DEVICES=-1 python tools/train.py configs/swinir/swinir_psnr-
↪x4s64w8d6e180_8xb4-lr1e-4-600k_df2k-ost.py
CUDA_VISIBLE_DEVICES=-1 python tools/train.py configs/swinir/swinir_gan-x4s64w8d9e240_
↪8xb4-lr1e-4-600k_df2k-ost.py
CUDA_VISIBLE_DEVICES=-1 python tools/train.py configs/swinir/swinir_psnr-
↪x4s64w8d9e240_8xb4-lr1e-4-600k_df2k-ost.py

## 004 Grayscale Image Deoising (middle size)
CUDA_VISIBLE_DEVICES=-1 python tools/train.py configs/swinir/swinir_s128w8d6e180_8xb1-
↪lr2e-4-1600k_dfwb-grayDN15.py
CUDA_VISIBLE_DEVICES=-1 python tools/train.py configs/swinir/swinir_s128w8d6e180_8xb1-
↪lr2e-4-1600k_dfwb-grayDN25.py
CUDA_VISIBLE_DEVICES=-1 python tools/train.py configs/swinir/swinir_s128w8d6e180_8xb1-
↪lr2e-4-1600k_dfwb-grayDN50.py

## 005 Color Image Deoising (middle size)
CUDA_VISIBLE_DEVICES=-1 python tools/train.py configs/swinir/swinir_s128w8d6e180_8xb1-
↪lr2e-4-1600k_dfwb-colorDN15.py
```

(下页继续)

(续上页)

```
CUDA_VISIBLE_DEVICES=-1 python tools/train.py configs/swinir/swinir_s128w8d6e180_8xb1-
↪lr2e-4-1600k_dfwb-colorDN25.py
CUDA_VISIBLE_DEVICES=-1 python tools/train.py configs/swinir/swinir_s128w8d6e180_8xb1-
↪lr2e-4-1600k_dfwb-colorDN50.py

## 006 JPEG Compression Artifact Reduction (middle size, using window_size=7 because␣
↪JPEG encoding uses 8x8 blocks)
## grayscale
CUDA_VISIBLE_DEVICES=-1 python tools/train.py configs/swinir/swinir_s126w7d6e180_8xb1-
↪lr2e-4-1600k_dfwb-grayCAR10.py
CUDA_VISIBLE_DEVICES=-1 python tools/train.py configs/swinir/swinir_s126w7d6e180_8xb1-
↪lr2e-4-1600k_dfwb-grayCAR20.py
CUDA_VISIBLE_DEVICES=-1 python tools/train.py configs/swinir/swinir_s126w7d6e180_8xb1-
↪lr2e-4-1600k_dfwb-grayCAR30.py
CUDA_VISIBLE_DEVICES=-1 python tools/train.py configs/swinir/swinir_s126w7d6e180_8xb1-
↪lr2e-4-1600k_dfwb-grayCAR40.py

## color
CUDA_VISIBLE_DEVICES=-1 python tools/train.py configs/swinir/swinir_s126w7d6e180_8xb1-
↪lr2e-4-1600k_dfwb-colorCAR10.py
CUDA_VISIBLE_DEVICES=-1 python tools/train.py configs/swinir/swinir_s126w7d6e180_8xb1-
↪lr2e-4-1600k_dfwb-colorCAR20.py
CUDA_VISIBLE_DEVICES=-1 python tools/train.py configs/swinir/swinir_s126w7d6e180_8xb1-
↪lr2e-4-1600k_dfwb-colorCAR30.py
CUDA_VISIBLE_DEVICES=-1 python tools/train.py configs/swinir/swinir_s126w7d6e180_8xb1-
↪lr2e-4-1600k_dfwb-colorCAR40.py



## 单个GPU上训练
## 001 Classical Image Super-Resolution (middle size)
## (setting1: when model is trained on DIV2K and with training_patch_size=48)
python tools/train.py configs/swinir/swinir_x2s48w8d6e180_8xb4-lr2e-4-500k_div2k.py
python tools/train.py configs/swinir/swinir_x3s48w8d6e180_8xb4-lr2e-4-500k_div2k.py
python tools/train.py configs/swinir/swinir_x4s48w8d6e180_8xb4-lr2e-4-500k_div2k.py

## (setting2: when model is trained on DIV2K+Flickr2K and with training_patch_size=64)
python tools/train.py configs/swinir/swinir_x2s64w8d6e180_8xb4-lr2e-4-500k_df2k.py
python tools/train.py configs/swinir/swinir_x3s64w8d6e180_8xb4-lr2e-4-500k_df2k.py
python tools/train.py configs/swinir/swinir_x4s64w8d6e180_8xb4-lr2e-4-500k_df2k.py

## 002 Lightweight Image Super-Resolution (small size)
python tools/train.py configs/swinir/swinir_x2s64w8d4e60_8xb4-lr2e-4-500k_div2k.py
```

(下页继续)

(续上页)

```
python tools/train.py configs/swinir/swinir_x3s64w8d4e60_8xb4-lr2e-4-500k_div2k.py
python tools/train.py configs/swinir/swinir_x4s64w8d4e60_8xb4-lr2e-4-500k_div2k.py

## 003 Real-World Image Super-Resolution
python tools/train.py configs/swinir/swinir_gan-x2s64w8d6e180_8xb4-lr1e-4-600k_df2k-
↪ost.py
python tools/train.py configs/swinir/swinir_psnr-x2s64w8d6e180_8xb4-lr1e-4-600k_df2k-
↪ost.py
python tools/train.py configs/swinir/swinir_gan-x4s64w8d6e180_8xb4-lr1e-4-600k_df2k-
↪ost.py
python tools/train.py configs/swinir/swinir_psnr-x4s64w8d6e180_8xb4-lr1e-4-600k_df2k-
↪ost.py
python tools/train.py configs/swinir/swinir_gan-x4s64w8d9e240_8xb4-lr1e-4-600k_df2k-
↪ost.py
python tools/train.py configs/swinir/swinir_psnr-x4s64w8d9e240_8xb4-lr1e-4-600k_df2k-
↪ost.py

## 004 Grayscale Image Deoising (middle size)
python tools/train.py configs/swinir/swinir_s128w8d6e180_8xb1-lr2e-4-1600k_dfwb-
↪grayDN15.py
python tools/train.py configs/swinir/swinir_s128w8d6e180_8xb1-lr2e-4-1600k_dfwb-
↪grayDN25.py
python tools/train.py configs/swinir/swinir_s128w8d6e180_8xb1-lr2e-4-1600k_dfwb-
↪grayDN50.py

## 005 Color Image Deoising (middle size)
python tools/train.py configs/swinir/swinir_s128w8d6e180_8xb1-lr2e-4-1600k_dfwb-
↪colorDN15.py
python tools/train.py configs/swinir/swinir_s128w8d6e180_8xb1-lr2e-4-1600k_dfwb-
↪colorDN25.py
python tools/train.py configs/swinir/swinir_s128w8d6e180_8xb1-lr2e-4-1600k_dfwb-
↪colorDN50.py

## 006 JPEG Compression Artifact Reduction (middle size, using window_size=7 because␣
↪JPEG encoding uses 8x8 blocks)
## grayscale
python tools/train.py configs/swinir/swinir_s126w7d6e180_8xb1-lr2e-4-1600k_dfwb-
↪grayCAR10.py
python tools/train.py configs/swinir/swinir_s126w7d6e180_8xb1-lr2e-4-1600k_dfwb-
↪grayCAR20.py
python tools/train.py configs/swinir/swinir_s126w7d6e180_8xb1-lr2e-4-1600k_dfwb-
↪grayCAR30.py
python tools/train.py configs/swinir/swinir_s126w7d6e180_8xb1-lr2e-4-1600k_dfwb-
↪grayCAR40.py
```

(下页继续)

(续上页)

```
## color
python tools/train.py configs/swinir/swinir_s126w7d6e180_8xb1-lr2e-4-1600k_dfwb-
↪colorCAR10.py
python tools/train.py configs/swinir/swinir_s126w7d6e180_8xb1-lr2e-4-1600k_dfwb-
↪colorCAR20.py
python tools/train.py configs/swinir/swinir_s126w7d6e180_8xb1-lr2e-4-1600k_dfwb-
↪colorCAR30.py
python tools/train.py configs/swinir/swinir_s126w7d6e180_8xb1-lr2e-4-1600k_dfwb-
↪colorCAR40.py



## 多个GPU上训练
## 001 Classical Image Super-Resolution (middle size)
## (setting1: when model is trained on DIV2K and with training_patch_size=48)
./tools/dist_train.sh configs/swinir/swinir_x2s48w8d6e180_8xb4-lr2e-4-500k_div2k.py 8
./tools/dist_train.sh configs/swinir/swinir_x3s48w8d6e180_8xb4-lr2e-4-500k_div2k.py 8
./tools/dist_train.sh configs/swinir/swinir_x4s48w8d6e180_8xb4-lr2e-4-500k_div2k.py 8

## (setting2: when model is trained on DIV2K+Flickr2K and with training_patch_size=64)
./tools/dist_train.sh configs/swinir/swinir_x2s64w8d6e180_8xb4-lr2e-4-500k_df2k.py 8
./tools/dist_train.sh configs/swinir/swinir_x3s64w8d6e180_8xb4-lr2e-4-500k_df2k.py 8
./tools/dist_train.sh configs/swinir/swinir_x4s64w8d6e180_8xb4-lr2e-4-500k_df2k.py 8

## 002 Lightweight Image Super-Resolution (small size)
./tools/dist_train.sh configs/swinir/swinir_x2s64w8d4e60_8xb4-lr2e-4-500k_div2k.py 8
./tools/dist_train.sh configs/swinir/swinir_x3s64w8d4e60_8xb4-lr2e-4-500k_div2k.py 8
./tools/dist_train.sh configs/swinir/swinir_x4s64w8d4e60_8xb4-lr2e-4-500k_div2k.py 8

## 003 Real-World Image Super-Resolution
./tools/dist_train.sh configs/swinir/swinir_gan-x2s64w8d6e180_8xb4-lr1e-4-600k_df2k-
↪ost.py 8
./tools/dist_train.sh configs/swinir/swinir_psnr-x2s64w8d6e180_8xb4-lr1e-4-600k_df2k-
↪ost.py 8
./tools/dist_train.sh configs/swinir/swinir_gan-x4s64w8d6e180_8xb4-lr1e-4-600k_df2k-
↪ost.py 8
./tools/dist_train.sh configs/swinir/swinir_psnr-x4s64w8d6e180_8xb4-lr1e-4-600k_df2k-
↪ost.py 8
./tools/dist_train.sh configs/swinir/swinir_gan-x4s64w8d9e240_8xb4-lr1e-4-600k_df2k-
↪ost.py 8
./tools/dist_train.sh configs/swinir/swinir_psnr-x4s64w8d9e240_8xb4-lr1e-4-600k_df2k-
↪ost.py 8
```

(下页继续)

(续上页)

```
## 004 Grayscale Image Deoising (middle size)
./tools/dist_train.sh configs/swinir/swinir_s128w8d6e180_8xb1-lr2e-4-1600k_dfwb-
↪grayDN15.py 8
./tools/dist_train.sh configs/swinir/swinir_s128w8d6e180_8xb1-lr2e-4-1600k_dfwb-
↪grayDN25.py 8
./tools/dist_train.sh configs/swinir/swinir_s128w8d6e180_8xb1-lr2e-4-1600k_dfwb-
↪grayDN50.py 8

## 005 Color Image Deoising (middle size)
./tools/dist_train.sh configs/swinir/swinir_s128w8d6e180_8xb1-lr2e-4-1600k_dfwb-
↪colorDN15.py 8
./tools/dist_train.sh configs/swinir/swinir_s128w8d6e180_8xb1-lr2e-4-1600k_dfwb-
↪colorDN25.py 8
./tools/dist_train.sh configs/swinir/swinir_s128w8d6e180_8xb1-lr2e-4-1600k_dfwb-
↪colorDN50.py 8

## 006 JPEG Compression Artifact Reduction (middle size, using window_size=7 because␣
↪JPEG encoding uses 8x8 blocks)
## grayscale
./tools/dist_train.sh configs/swinir/swinir_s126w7d6e180_8xb1-lr2e-4-1600k_dfwb-
↪grayCAR10.py 8
./tools/dist_train.sh configs/swinir/swinir_s126w7d6e180_8xb1-lr2e-4-1600k_dfwb-
↪grayCAR20.py 8
./tools/dist_train.sh configs/swinir/swinir_s126w7d6e180_8xb1-lr2e-4-1600k_dfwb-
↪grayCAR30.py 8
./tools/dist_train.sh configs/swinir/swinir_s126w7d6e180_8xb1-lr2e-4-1600k_dfwb-
↪grayCAR40.py 8

## color
./tools/dist_train.sh configs/swinir/swinir_s126w7d6e180_8xb1-lr2e-4-1600k_dfwb-
↪colorCAR10.py 8
./tools/dist_train.sh configs/swinir/swinir_s126w7d6e180_8xb1-lr2e-4-1600k_dfwb-
↪colorCAR20.py 8
./tools/dist_train.sh configs/swinir/swinir_s126w7d6e180_8xb1-lr2e-4-1600k_dfwb-
↪colorCAR30.py 8
./tools/dist_train.sh configs/swinir/swinir_s126w7d6e180_8xb1-lr2e-4-1600k_dfwb-
↪colorCAR40.py 8
```

更多细节可以参考 train_test.md 中的 **Train a model** 部分。

测试

您可以使用以下命令来测试模型。

```
## CPU上测试
## 001 Classical Image Super-Resolution (middle size)
## (setting1: when model is trained on DIV2K and with training_patch_size=48)
CUDA_VISIBLE_DEVICES=-1 python tools/test.py configs/swinir/swinir_x2s48w8d6e180_8xb4-
↪lr2e-4-500k_div2k.py https://download.openmmlab.com/mmediting/swinir/swinir_
↪x2s48w8d6e180_8xb4-lr2e-4-500k_div2k-ed2d419e.pth

CUDA_VISIBLE_DEVICES=-1 python tools/test.py configs/swinir/swinir_x3s48w8d6e180_8xb4-
↪lr2e-4-500k_div2k.py https://download.openmmlab.com/mmediting/swinir/swinir_
↪x3s48w8d6e180_8xb4-lr2e-4-500k_div2k-926950f1.pth

CUDA_VISIBLE_DEVICES=-1 python tools/test.py configs/swinir/swinir_x4s48w8d6e180_8xb4-
↪lr2e-4-500k_div2k.py https://download.openmmlab.com/mmediting/swinir/swinir_
↪x4s48w8d6e180_8xb4-lr2e-4-500k_div2k-88e4903d.pth

## (setting2: when model is trained on DIV2K+Flickr2K and with training_patch_size=64)
CUDA_VISIBLE_DEVICES=-1 python tools/test.py configs/swinir/swinir_x2s64w8d6e180_8xb4-
↪lr2e-4-500k_df2k.py https://download.openmmlab.com/mmediting/swinir/swinir_
↪x2s64w8d6e180_8xb4-lr2e-4-500k_df2k-69e15fb6.pth

CUDA_VISIBLE_DEVICES=-1 python tools/test.py configs/swinir/swinir_x3s64w8d6e180_8xb4-
↪lr2e-4-500k_df2k.py https://download.openmmlab.com/mmediting/swinir/swinir_
↪x3s64w8d6e180_8xb4-lr2e-4-500k_df2k-d6982f7b.pth

CUDA_VISIBLE_DEVICES=-1 python tools/test.py configs/swinir/swinir_x4s64w8d6e180_8xb4-
↪lr2e-4-500k_df2k.py https://download.openmmlab.com/mmediting/swinir/swinir_
↪x4s64w8d6e180_8xb4-lr2e-4-500k_df2k-0502d775.pth



## 002 Lightweight Image Super-Resolution (small size)
CUDA_VISIBLE_DEVICES=-1 python tools/test.py configs/swinir/swinir_x2s64w8d4e60_8xb4-
↪lr2e-4-500k_div2k.py https://download.openmmlab.com/mmediting/swinir/swinir_
↪x2s64w8d4e60_8xb4-lr2e-4-500k_div2k-131d3f64.pth

CUDA_VISIBLE_DEVICES=-1 python tools/test.py configs/swinir/swinir_x3s64w8d4e60_8xb4-
↪lr2e-4-500k_div2k.py https://download.openmmlab.com/mmediting/swinir/swinir_
↪x3s64w8d4e60_8xb4-lr2e-4-500k_div2k-309cb239.pth

CUDA_VISIBLE_DEVICES=-1 python tools/test.py configs/swinir/swinir_x4s64w8d4e60_8xb4-
↪lr2e-4-500k_div2k.py https://download.openmmlab.com/mmediting/swinir/swinir_
↪x4s64w8d4e60_8xb4-lr2e-4-500k_div2k-d6622d03.pth

## 003 Real-World Image Super-Resolution
CUDA_VISIBLE_DEVICES=-1 python tools/test.py configs/swinir/swinir_gan-x2s64w8d6e180_
↪8xb4-lr1e-4-600k_df2k-ost.py https://download.openmmlab.com/mmediting/swinir/swinir_
↪gan-x2s64w8d6e180_8xb4-lr1e-4-600k_df2k-os-c6425057.pth
```

(下页继续)

(续上页)

```
CUDA_VISIBLE_DEVICES=-1 python tools/test.py configs/swinir/swinir_psnr-x2s64w8d6e180_
↪8xb4-lr1e-4-600k_df2k-ost.py https://download.openmmlab.com/mmediting/swinir/swinir_
↪psnr-x2s64w8d6e180_8xb4-lr1e-4-600k_df2k-os-6f0c425f.pth

CUDA_VISIBLE_DEVICES=-1 python tools/test.py configs/swinir/swinir_gan-x4s64w8d6e180_
↪8xb4-lr1e-4-600k_df2k-ost.py https://download.openmmlab.com/mmediting/swinir/swinir_
↪gan-x4s64w8d6e180_8xb4-lr1e-4-600k_df2k-os-36960d18.pth

CUDA_VISIBLE_DEVICES=-1 python tools/test.py configs/swinir/swinir_psnr-x4s64w8d6e180_
↪8xb4-lr1e-4-600k_df2k-ost.py https://download.openmmlab.com/mmediting/swinir/swinir_
↪psnr-x4s64w8d6e180_8xb4-lr1e-4-600k_df2k-os-a016a72f.pth

CUDA_VISIBLE_DEVICES=-1 python tools/test.py configs/swinir/swinir_gan-x4s64w8d9e240_
↪8xb4-lr1e-4-600k_df2k-ost.py https://download.openmmlab.com/mmediting/swinir/swinir_
↪gan-x4s64w8d9e240_8xb4-lr1e-4-600k_df2k-os-9f1599b5.pth

CUDA_VISIBLE_DEVICES=-1 python tools/test.py configs/swinir/swinir_psnr-x4s64w8d9e240_
↪8xb4-lr1e-4-600k_df2k-ost.py https://download.openmmlab.com/mmediting/swinir/swinir_
↪psnr-x4s64w8d9e240_8xb4-lr1e-4-600k_df2k-os-25f1722a.pth

## 004 Grayscale Image Deoising (middle size)
CUDA_VISIBLE_DEVICES=-1 python tools/test.py configs/swinir/swinir_s128w8d6e180_8xb1-
↪lr2e-4-1600k_dfwb-grayDN15.py https://download.openmmlab.com/mmediting/swinir/
↪swinir_s128w8d6e180_8xb1-lr2e-4-1600k_dfwb-grayDN15-6782691b.pth

CUDA_VISIBLE_DEVICES=-1 python tools/test.py configs/swinir/swinir_s128w8d6e180_8xb1-
↪lr2e-4-1600k_dfwb-grayDN25.py https://download.openmmlab.com/mmediting/swinir/
↪swinir_s128w8d6e180_8xb1-lr2e-4-1600k_dfwb-grayDN25-d0d8d4da.pth

CUDA_VISIBLE_DEVICES=-1 python tools/test.py configs/swinir/swinir_s128w8d6e180_8xb1-
↪lr2e-4-1600k_dfwb-grayDN50.py https://download.openmmlab.com/mmediting/swinir/
↪swinir_s128w8d6e180_8xb1-lr2e-4-1600k_dfwb-grayDN50-54c9968a.pth

## 005 Color Image Deoising (middle size)
CUDA_VISIBLE_DEVICES=-1 python tools/test.py configs/swinir/swinir_s128w8d6e180_8xb1-
↪lr2e-4-1600k_dfwb-colorDN15.py https://download.openmmlab.com/mmediting/swinir/
↪swinir_s128w8d6e180_8xb1-lr2e-4-1600k_dfwb-colorDN15-c74a2cee.pth

CUDA_VISIBLE_DEVICES=-1 python tools/test.py configs/swinir/swinir_s128w8d6e180_8xb1-
↪lr2e-4-1600k_dfwb-colorDN25.py https://download.openmmlab.com/mmediting/swinir/
↪swinir_s128w8d6e180_8xb1-lr2e-4-1600k_dfwb-colorDN25-df2b1c0c.pth
```

(下页继续)

(续上页)

```
CUDA_VISIBLE_DEVICES=-1 python tools/test.py configs/swinir/swinir_s128w8d6e180_8xb1-
↪lr2e-4-1600k_dfwb-colorDN50.py https://download.openmmlab.com/mmediting/swinir/
↪swinir_s128w8d6e180_8xb1-lr2e-4-1600k_dfwb-colorDN50-e369874c.pth

## 006 JPEG Compression Artifact Reduction (middle size, using window_size=7 because␣
↪JPEG encoding usesx8 blocks)
## grayscale
CUDA_VISIBLE_DEVICES=-1 python tools/test.py configs/swinir/swinir_s126w7d6e180_8xb1-
↪lr2e-4-1600k_dfwb-grayCAR10.py https://download.openmmlab.com/mmediting/swinir/
↪swinir_s126w7d6e180_8xb1-lr2e-4-1600k_dfwb-grayCAR10-da93c8e9.pth

CUDA_VISIBLE_DEVICES=-1 python tools/test.py configs/swinir/swinir_s126w7d6e180_8xb1-
↪lr2e-4-1600k_dfwb-grayCAR20.py https://download.openmmlab.com/mmediting/swinir/
↪swinir_s126w7d6e180_8xb1-lr2e-4-1600k_dfwb-grayCAR20-d47367b1.pth

CUDA_VISIBLE_DEVICES=-1 python tools/test.py configs/swinir/swinir_s126w7d6e180_8xb1-
↪lr2e-4-1600k_dfwb-grayCAR30.py https://download.openmmlab.com/mmediting/swinir/
↪swinir_s126w7d6e180_8xb1-lr2e-4-1600k_dfwb-grayCAR30-52c083cf.pth

CUDA_VISIBLE_DEVICES=-1 python tools/test.py configs/swinir/swinir_s126w7d6e180_8xb1-
↪lr2e-4-1600k_dfwb-grayCAR40.py https://download.openmmlab.com/mmediting/swinir/
↪swinir_s126w7d6e180_8xb1-lr2e-4-1600k_dfwb-grayCAR40-803e8d9b.pth


## color
CUDA_VISIBLE_DEVICES=-1 python tools/test.py configs/swinir/swinir_s126w7d6e180_8xb1-
↪lr2e-4-1600k_dfwb-colorCAR10.py https://download.openmmlab.com/mmediting/swinir/
↪swinir_s126w7d6e180_8xb1-lr2e-4-1600k_dfwb-colorCAR10-09aafadc.pth

CUDA_VISIBLE_DEVICES=-1 python tools/test.py configs/swinir/swinir_s126w7d6e180_8xb1-
↪lr2e-4-1600k_dfwb-colorCAR20.py https://download.openmmlab.com/mmediting/swinir/
↪swinir_s126w7d6e180_8xb1-lr2e-4-1600k_dfwb-colorCAR20-b8a42b5e.pth

CUDA_VISIBLE_DEVICES=-1 python tools/test.py configs/swinir/swinir_s126w7d6e180_8xb1-
↪lr2e-4-1600k_dfwb-colorCAR30.py https://download.openmmlab.com/mmediting/swinir/
↪swinir_s126w7d6e180_8xb1-lr2e-4-1600k_dfwb-colorCAR30-e9fe6859.pth

CUDA_VISIBLE_DEVICES=-1 python tools/test.py configs/swinir/swinir_s126w7d6e180_8xb1-
↪lr2e-4-1600k_dfwb-colorCAR40.py https://download.openmmlab.com/mmediting/swinir/
↪swinir_s126w7d6e180_8xb1-lr2e-4-1600k_dfwb-colorCAR40-5b77a6e6.pth
```

(下页继续)

(续上页)

```
## 单个GPU上测试
## 001 Classical Image Super-Resolution (middle size)
## (setting1: when model is trained on DIV2K and with training_patch_size=48)
python tools/test.py configs/swinir/swinir_x2s48w8d6e180_8xb4-lr2e-4-500k_div2k.py␣
↪https://download.openmmlab.com/mmediting/swinir/swinir_x2s48w8d6e180_8xb4-lr2e-4-
↪500k_div2k-ed2d419e.pth

python tools/test.py configs/swinir/swinir_x3s48w8d6e180_8xb4-lr2e-4-500k_div2k.py␣
↪https://download.openmmlab.com/mmediting/swinir/swinir_x3s48w8d6e180_8xb4-lr2e-4-
↪500k_div2k-926950f1.pth

python tools/test.py configs/swinir/swinir_x4s48w8d6e180_8xb4-lr2e-4-500k_div2k.py␣
↪https://download.openmmlab.com/mmediting/swinir/swinir_x4s48w8d6e180_8xb4-lr2e-4-
↪500k_div2k-88e4903d.pth

## (setting2: when model is trained on DIV2K+Flickr2K and with training_patch_size=64)
python tools/test.py configs/swinir/swinir_x2s64w8d6e180_8xb4-lr2e-4-500k_df2k.py␣
↪https://download.openmmlab.com/mmediting/swinir/swinir_x2s64w8d6e180_8xb4-lr2e-4-
↪500k_df2k-69e15fb6.pth

python tools/test.py configs/swinir/swinir_x3s64w8d6e180_8xb4-lr2e-4-500k_df2k.py␣
↪https://download.openmmlab.com/mmediting/swinir/swinir_x3s64w8d6e180_8xb4-lr2e-4-
↪500k_df2k-d6982f7b.pth

python tools/test.py configs/swinir/swinir_x4s64w8d6e180_8xb4-lr2e-4-500k_df2k.py␣
↪https://download.openmmlab.com/mmediting/swinir/swinir_x4s64w8d6e180_8xb4-lr2e-4-
↪500k_df2k-0502d775.pth


## 002 Lightweight Image Super-Resolution (small size)
python tools/test.py configs/swinir/swinir_x2s64w8d4e60_8xb4-lr2e-4-500k_div2k.py␣
↪https://download.openmmlab.com/mmediting/swinir/swinir_x2s64w8d4e60_8xb4-lr2e-4-
↪500k_div2k-131d3f64.pth

python tools/test.py configs/swinir/swinir_x3s64w8d4e60_8xb4-lr2e-4-500k_div2k.py␣
↪https://download.openmmlab.com/mmediting/swinir/swinir_x3s64w8d4e60_8xb4-lr2e-4-
↪500k_div2k-309cb239.pth

python tools/test.py configs/swinir/swinir_x4s64w8d4e60_8xb4-lr2e-4-500k_div2k.py␣
↪https://download.openmmlab.com/mmediting/swinir/swinir_x4s64w8d4e60_8xb4-lr2e-4-
↪500k_div2k-d6622d03.pth
```

(下页继续)

(续上页)

```
## 003 Real-World Image Super-Resolution
python tools/test.py configs/swinir/swinir_gan-x2s64w8d6e180_8xb4-lr1e-4-600k_df2k-
↪ost.py https://download.openmmlab.com/mmediting/swinir/swinir_gan-x2s64w8d6e180_
↪8xb4-lr1e-4-600k_df2k-os-c6425057.pth

python tools/test.py configs/swinir/swinir_psnr-x2s64w8d6e180_8xb4-lr1e-4-600k_df2k-
↪ost.py https://download.openmmlab.com/mmediting/swinir/swinir_psnr-x2s64w8d6e180_
↪8xb4-lr1e-4-600k_df2k-os-6f0c425f.pth

python tools/test.py configs/swinir/swinir_gan-x4s64w8d6e180_8xb4-lr1e-4-600k_df2k-
↪ost.py https://download.openmmlab.com/mmediting/swinir/swinir_gan-x4s64w8d6e180_
↪8xb4-lr1e-4-600k_df2k-os-36960d18.pth

python tools/test.py configs/swinir/swinir_psnr-x4s64w8d6e180_8xb4-lr1e-4-600k_df2k-
↪ost.py https://download.openmmlab.com/mmediting/swinir/swinir_psnr-x4s64w8d6e180_
↪8xb4-lr1e-4-600k_df2k-os-a016a72f.pth

python tools/test.py configs/swinir/swinir_gan-x4s64w8d9e240_8xb4-lr1e-4-600k_df2k-
↪ost.py https://download.openmmlab.com/mmediting/swinir/swinir_gan-x4s64w8d9e240_
↪8xb4-lr1e-4-600k_df2k-os-9f1599b5.pth

python tools/test.py configs/swinir/swinir_psnr-x4s64w8d9e240_8xb4-lr1e-4-600k_df2k-
↪ost.py https://download.openmmlab.com/mmediting/swinir/swinir_psnr-x4s64w8d9e240_
↪8xb4-lr1e-4-600k_df2k-os-25f1722a.pth


## 004 Grayscale Image Deoising (middle size)
python tools/test.py configs/swinir/swinir_s128w8d6e180_8xb1-lr2e-4-1600k_dfwb-
↪grayDN15.py https://download.openmmlab.com/mmediting/swinir/swinir_s128w8d6e180_
↪8xb1-lr2e-4-1600k_dfwb-grayDN15-6782691b.pth

python tools/test.py configs/swinir/swinir_s128w8d6e180_8xb1-lr2e-4-1600k_dfwb-
↪grayDN25.py https://download.openmmlab.com/mmediting/swinir/swinir_s128w8d6e180_
↪8xb1-lr2e-4-1600k_dfwb-grayDN25-d0d8d4da.pth

python tools/test.py configs/swinir/swinir_s128w8d6e180_8xb1-lr2e-4-1600k_dfwb-
↪grayDN50.py https://download.openmmlab.com/mmediting/swinir/swinir_s128w8d6e180_
↪8xb1-lr2e-4-1600k_dfwb-grayDN50-54c9968a.pth


## 005 Color Image Deoising (middle size)
python tools/test.py configs/swinir/swinir_s128w8d6e180_8xb1-lr2e-4-1600k_dfwb-
↪colorDN15.py https://download.openmmlab.com/mmediting/swinir/swinir_s128w8d6e180_
↪8xb1-lr2e-4-1600k_dfwb-colorDN15-c74a2cee.pth
```

(下页继续)

(续上页)

```
python tools/test.py configs/swinir/swinir_s128w8d6e180_8xb1-lr2e-4-1600k_dfwb-
↪colorDN25.py https://download.openmmlab.com/mmediting/swinir/swinir_s128w8d6e180_
↪8xb1-lr2e-4-1600k_dfwb-colorDN25-df2b1c0c.pth

python tools/test.py configs/swinir/swinir_s128w8d6e180_8xb1-lr2e-4-1600k_dfwb-
↪colorDN50.py https://download.openmmlab.com/mmediting/swinir/swinir_s128w8d6e180_
↪8xb1-lr2e-4-1600k_dfwb-colorDN50-e369874c.pth


## 006 JPEG Compression Artifact Reduction (middle size, using window_size=7 because␣
↪JPEG encoding usesx8 blocks)
## grayscale
python tools/test.py configs/swinir/swinir_s126w7d6e180_8xb1-lr2e-4-1600k_dfwb-
↪grayCAR10.py https://download.openmmlab.com/mmediting/swinir/swinir_s126w7d6e180_
↪8xb1-lr2e-4-1600k_dfwb-grayCAR10-da93c8e9.pth

python tools/test.py configs/swinir/swinir_s126w7d6e180_8xb1-lr2e-4-1600k_dfwb-
↪grayCAR20.py https://download.openmmlab.com/mmediting/swinir/swinir_s126w7d6e180_
↪8xb1-lr2e-4-1600k_dfwb-grayCAR20-d47367b1.pth

python tools/test.py configs/swinir/swinir_s126w7d6e180_8xb1-lr2e-4-1600k_dfwb-
↪grayCAR30.py https://download.openmmlab.com/mmediting/swinir/swinir_s126w7d6e180_
↪8xb1-lr2e-4-1600k_dfwb-grayCAR30-52c083cf.pth

python tools/test.py configs/swinir/swinir_s126w7d6e180_8xb1-lr2e-4-1600k_dfwb-
↪grayCAR40.py https://download.openmmlab.com/mmediting/swinir/swinir_s126w7d6e180_
↪8xb1-lr2e-4-1600k_dfwb-grayCAR40-803e8d9b.pth


## color
python tools/test.py configs/swinir/swinir_s126w7d6e180_8xb1-lr2e-4-1600k_dfwb-
↪colorCAR10.py https://download.openmmlab.com/mmediting/swinir/swinir_s126w7d6e180_
↪8xb1-lr2e-4-1600k_dfwb-colorCAR10-09aafadc.pth

python tools/test.py configs/swinir/swinir_s126w7d6e180_8xb1-lr2e-4-1600k_dfwb-
↪colorCAR20.py https://download.openmmlab.com/mmediting/swinir/swinir_s126w7d6e180_
↪8xb1-lr2e-4-1600k_dfwb-colorCAR20-b8a42b5e.pth

python tools/test.py configs/swinir/swinir_s126w7d6e180_8xb1-lr2e-4-1600k_dfwb-
↪colorCAR30.py https://download.openmmlab.com/mmediting/swinir/swinir_s126w7d6e180_
↪8xb1-lr2e-4-1600k_dfwb-colorCAR30-e9fe6859.pth
```

(下页继续)

(续上页)

```
python tools/test.py configs/swinir/swinir_s126w7d6e180_8xb1-lr2e-4-1600k_dfwb-
↪colorCAR40.py https://download.openmmlab.com/mmediting/swinir/swinir_s126w7d6e180_
↪8xb1-lr2e-4-1600k_dfwb-colorCAR40-5b77a6e6.pth



## 多GPU测试
## 001 Classical Image Super-Resolution (middle size)
## (setting1: when model is trained on DIV2K and with training_patch_size=48)
./tools/dist_test.sh configs/swinir/swinir_x2s48w8d6e180_8xb4-lr2e-4-500k_div2k.py␣
↪https://download.openmmlab.com/mmediting/swinir/swinir_x2s48w8d6e180_8xb4-lr2e-4-
↪500k_div2k-ed2d419e.pth

./tools/dist_test.sh configs/swinir/swinir_x3s48w8d6e180_8xb4-lr2e-4-500k_div2k.py␣
↪https://download.openmmlab.com/mmediting/swinir/swinir_x3s48w8d6e180_8xb4-lr2e-4-
↪500k_div2k-926950f1.pth

./tools/dist_test.sh configs/swinir/swinir_x4s48w8d6e180_8xb4-lr2e-4-500k_div2k.py␣
↪https://download.openmmlab.com/mmediting/swinir/swinir_x4s48w8d6e180_8xb4-lr2e-4-
↪500k_div2k-88e4903d.pth

## (setting2: when model is trained on DIV2K+Flickr2K and with training_patch_size=64)
./tools/dist_test.sh configs/swinir/swinir_x2s64w8d6e180_8xb4-lr2e-4-500k_df2k.py␣
↪https://download.openmmlab.com/mmediting/swinir/swinir_x2s64w8d6e180_8xb4-lr2e-4-
↪500k_df2k-69e15fb6.pth

./tools/dist_test.sh configs/swinir/swinir_x3s64w8d6e180_8xb4-lr2e-4-500k_df2k.py␣
↪https://download.openmmlab.com/mmediting/swinir/swinir_x3s64w8d6e180_8xb4-lr2e-4-
↪500k_df2k-d6982f7b.pth

./tools/dist_test.sh configs/swinir/swinir_x4s64w8d6e180_8xb4-lr2e-4-500k_df2k.py␣
↪https://download.openmmlab.com/mmediting/swinir/swinir_x4s64w8d6e180_8xb4-lr2e-4-
↪500k_df2k-0502d775.pth

## 002 Lightweight Image Super-Resolution (small size)
./tools/dist_test.sh configs/swinir/swinir_x2s64w8d4e60_8xb4-lr2e-4-500k_div2k.py␣
↪https://download.openmmlab.com/mmediting/swinir/swinir_x2s64w8d4e60_8xb4-lr2e-4-
↪500k_div2k-131d3f64.pth

./tools/dist_test.sh configs/swinir/swinir_x3s64w8d4e60_8xb4-lr2e-4-500k_div2k.py␣
↪https://download.openmmlab.com/mmediting/swinir/swinir_x3s64w8d4e60_8xb4-lr2e-4-
↪500k_div2k-309cb239.pth
```

(下页继续)

(续上页)

```
./tools/dist_test.sh configs/swinir/swinir_x4s64w8d4e60_8xb4-lr2e-4-500k_div2k.py␣
↪https://download.openmmlab.com/mmediting/swinir/swinir_x4s64w8d4e60_8xb4-lr2e-4-
↪500k_div2k-d6622d03.pth

## 003 Real-World Image Super-Resolution
./tools/dist_test.sh configs/swinir/swinir_gan-x2s64w8d6e180_8xb4-lr1e-4-600k_df2k-
↪ost.py https://download.openmmlab.com/mmediting/swinir/swinir_gan-x2s64w8d6e180_
↪8xb4-lr1e-4-600k_df2k-os-c6425057.pth

./tools/dist_test.sh configs/swinir/swinir_psnr-x2s64w8d6e180_8xb4-lr1e-4-600k_df2k-
↪ost.py https://download.openmmlab.com/mmediting/swinir/swinir_psnr-x2s64w8d6e180_
↪8xb4-lr1e-4-600k_df2k-os-6f0c425f.pth

./tools/dist_test.sh configs/swinir/swinir_gan-x4s64w8d6e180_8xb4-lr1e-4-600k_df2k-
↪ost.py https://download.openmmlab.com/mmediting/swinir/swinir_gan-x4s64w8d6e180_
↪8xb4-lr1e-4-600k_df2k-os-36960d18.pth

./tools/dist_test.sh configs/swinir/swinir_psnr-x4s64w8d6e180_8xb4-lr1e-4-600k_df2k-
↪ost.py https://download.openmmlab.com/mmediting/swinir/swinir_psnr-x4s64w8d6e180_
↪8xb4-lr1e-4-600k_df2k-os-a016a72f.pth

./tools/dist_test.sh configs/swinir/swinir_gan-x4s64w8d9e240_8xb4-lr1e-4-600k_df2k-
↪ost.py https://download.openmmlab.com/mmediting/swinir/swinir_gan-x4s64w8d9e240_
↪8xb4-lr1e-4-600k_df2k-os-9f1599b5.pth

./tools/dist_test.sh configs/swinir/swinir_psnr-x4s64w8d9e240_8xb4-lr1e-4-600k_df2k-
↪ost.py https://download.openmmlab.com/mmediting/swinir/swinir_psnr-x4s64w8d9e240_
↪8xb4-lr1e-4-600k_df2k-os-25f1722a.pth

## 004 Grayscale Image Deoising (middle size)
./tools/dist_test.sh configs/swinir/swinir_s128w8d6e180_8xb1-lr2e-4-1600k_dfwb-
↪grayDN15.py https://download.openmmlab.com/mmediting/swinir/swinir_s128w8d6e180_
↪8xb1-lr2e-4-1600k_dfwb-grayDN15-6782691b.pth

./tools/dist_test.sh configs/swinir/swinir_s128w8d6e180_8xb1-lr2e-4-1600k_dfwb-
↪grayDN25.py https://download.openmmlab.com/mmediting/swinir/swinir_s128w8d6e180_
↪8xb1-lr2e-4-1600k_dfwb-grayDN25-d0d8d4da.pth

./tools/dist_test.sh configs/swinir/swinir_s128w8d6e180_8xb1-lr2e-4-1600k_dfwb-
↪grayDN50.py https://download.openmmlab.com/mmediting/swinir/swinir_s128w8d6e180_
↪8xb1-lr2e-4-1600k_dfwb-grayDN50-54c9968a.pth

## 005 Color Image Deoising (middle size)
```

(下页继续)

(续上页)

```
./tools/dist_test.sh configs/swinir/swinir_s128w8d6e180_8xb1-lr2e-4-1600k_dfwb-
↪colorDN15.py https://download.openmmlab.com/mmediting/swinir/swinir_s128w8d6e180_
↪8xb1-lr2e-4-1600k_dfwb-colorDN15-c74a2cee.pth

./tools/dist_test.sh configs/swinir/swinir_s128w8d6e180_8xb1-lr2e-4-1600k_dfwb-
↪colorDN25.py https://download.openmmlab.com/mmediting/swinir/swinir_s128w8d6e180_
↪8xb1-lr2e-4-1600k_dfwb-colorDN25-df2b1c0c.pth

./tools/dist_test.sh configs/swinir/swinir_s128w8d6e180_8xb1-lr2e-4-1600k_dfwb-
↪colorDN50.py https://download.openmmlab.com/mmediting/swinir/swinir_s128w8d6e180_
↪8xb1-lr2e-4-1600k_dfwb-colorDN50-e369874c.pth

## 006 JPEG Compression Artifact Reduction (middle size, using window_size=7 because␣
↪JPEG encoding uses 8x8 blocks)
## grayscale
./tools/dist_test.sh configs/swinir/swinir_s126w7d6e180_8xb1-lr2e-4-1600k_dfwb-
↪grayCAR10.py https://download.openmmlab.com/mmediting/swinir/swinir_s126w7d6e180_
↪8xb1-lr2e-4-1600k_dfwb-grayCAR10-da93c8e9.pth

./tools/dist_test.sh configs/swinir/swinir_s126w7d6e180_8xb1-lr2e-4-1600k_dfwb-
↪grayCAR20.py https://download.openmmlab.com/mmediting/swinir/swinir_s126w7d6e180_
↪8xb1-lr2e-4-1600k_dfwb-grayCAR20-d47367b1.pth

./tools/dist_test.sh configs/swinir/swinir_s126w7d6e180_8xb1-lr2e-4-1600k_dfwb-
↪grayCAR30.py https://download.openmmlab.com/mmediting/swinir/swinir_s126w7d6e180_
↪8xb1-lr2e-4-1600k_dfwb-grayCAR30-52c083cf.pth

./tools/dist_test.sh configs/swinir/swinir_s126w7d6e180_8xb1-lr2e-4-1600k_dfwb-
↪grayCAR40.py https://download.openmmlab.com/mmediting/swinir/swinir_s126w7d6e180_
↪8xb1-lr2e-4-1600k_dfwb-grayCAR40-803e8d9b.pth

## color
./tools/dist_test.sh configs/swinir/swinir_s126w7d6e180_8xb1-lr2e-4-1600k_dfwb-
↪colorCAR10.py https://download.openmmlab.com/mmediting/swinir/swinir_s126w7d6e180_
↪8xb1-lr2e-4-1600k_dfwb-colorCAR10-09aafadc.pth

./tools/dist_test.sh configs/swinir/swinir_s126w7d6e180_8xb1-lr2e-4-1600k_dfwb-
↪colorCAR20.py https://download.openmmlab.com/mmediting/swinir/swinir_s126w7d6e180_
↪8xb1-lr2e-4-1600k_dfwb-colorCAR20-b8a42b5e.pth

./tools/dist_test.sh configs/swinir/swinir_s126w7d6e180_8xb1-lr2e-4-1600k_dfwb-
↪colorCAR30.py https://download.openmmlab.com/mmediting/swinir/swinir_s126w7d6e180_
↪8xb1-lr2e-4-1600k_dfwb-colorCAR30-e9fe6859.pth
```

(下页继续)

(续上页)

```
./tools/dist_test.sh configs/swinir/swinir_s126w7d6e180_8xb1-lr2e-4-1600k_dfwb-
↪colorCAR40.py https://download.openmmlab.com/mmediting/swinir/swinir_s126w7d6e180_
↪8xb1-lr2e-4-1600k_dfwb-colorCAR30-e9fe6859.pth
```

更多细节可以参考 train_test.md 中的 **Test a pre-trained model** 部分。

CHAPTER 37

图像超分辨率

37.1 概览

- 预训练权重个数: 52
- 配置文件个数: 0
- 论文个数: 11
  - ALGORITHM: 11

37.2 SwinIR (ICCVW’2021)

任务: 图像超分辨率, 图像去噪, JPEG 压缩伪影移除

```
@inproceedings{liang2021swinir,
  title={Swinir: Image restoration using swin transformer},
  author={Liang, Jingyun and Cao, Jiezhang and Sun, Guolei and Zhang, Kai and Van␣
↪Gool, Luc and Timofte, Radu},
  booktitle={Proceedings of the IEEE/CVF International Conference on Computer Vision},
  pages={1833--1844},
  year={2021}
}
```

37.2.1 Classical Image Super-Resolution

在 Y 通道上进行评估，在评估之前裁剪每个边界中的 scale 像素。我们使用 PSNR 和 SSIM 作为指标。

37.2.2 Lightweight Image Super-Resolution

在 Y 通道上进行评估，在评估之前裁剪每个边界中的 scale 像素。我们使用 PSNR 和 SSIM 作为指标。

37.2.3 Real-World Image Super-Resolution

在 Y 通道上进行评估。我们使用 NIQE 作为指标。

37.2.4 Grayscale Image Deoising

在灰度图上进行评估。我们使用 PSNR 作为指标。

37.2.5 Color Image Deoising

在 RGB 通道上进行评估。我们使用 PSNR 作为指标。

37.2.6 JPEG Compression Artifact Reduction (grayscale)

在灰度图上进行评估。我们使用 PSNR 和 SSIM 作为指标。

37.2.7 JPEG Compression Artifact Reduction (color)

在 RGB 通道上进行评估。我们使用 PSNR 和 SSIM 作为指标。

37.2.8 快速开始

训练

您可以使用以下命令来训练模型。

```
## CPU上训练
## 001 Classical Image Super-Resolution (middle size)
## (setting1: when model is trained on DIV2K and with training_patch_size=48)
CUDA_VISIBLE_DEVICES=-1 python tools/train.py configs/swinir/swinir_x2s48w8d6e180_
↪8xb4-lr2e-4-500k_div2k.py
CUDA_VISIBLE_DEVICES=-1 python tools/train.py configs/swinir/swinir_x3s48w8d6e180_
↪8xb4-lr2e-4-500k_div2k.py
```

(下页继续)

(续上页)

```
CUDA_VISIBLE_DEVICES=-1 python tools/train.py configs/swinir/swinir_x4s48w8d6e180_
↪8xb4-lr2e-4-500k_div2k.py

## (setting2: when model is trained on DIV2K+Flickr2K and with training_patch_size=64)
CUDA_VISIBLE_DEVICES=-1 python tools/train.py configs/swinir/swinir_x2s64w8d6e180_
↪8xb4-lr2e-4-500k_df2k.py
CUDA_VISIBLE_DEVICES=-1 python tools/train.py configs/swinir/swinir_x3s64w8d6e180_
↪8xb4-lr2e-4-500k_df2k.py
CUDA_VISIBLE_DEVICES=-1 python tools/train.py configs/swinir/swinir_x4s64w8d6e180_
↪8xb4-lr2e-4-500k_df2k.py

## 002 Lightweight Image Super-Resolution (small size)
CUDA_VISIBLE_DEVICES=-1 python tools/train.py configs/swinir/swinir_x2s64w8d4e60_8xb4-
↪lr2e-4-500k_div2k.py
CUDA_VISIBLE_DEVICES=-1 python tools/train.py configs/swinir/swinir_x3s64w8d4e60_8xb4-
↪lr2e-4-500k_div2k.py
CUDA_VISIBLE_DEVICES=-1 python tools/train.py configs/swinir/swinir_x4s64w8d4e60_8xb4-
↪lr2e-4-500k_div2k.py

## 003 Real-World Image Super-Resolution
CUDA_VISIBLE_DEVICES=-1 python tools/train.py configs/swinir/swinir_gan-x2s64w8d6e180_
↪8xb4-lr1e-4-600k_df2k-ost.py
CUDA_VISIBLE_DEVICES=-1 python tools/train.py configs/swinir/swinir_psnr-
↪x2s64w8d6e180_8xb4-lr1e-4-600k_df2k-ost.py
CUDA_VISIBLE_DEVICES=-1 python tools/train.py configs/swinir/swinir_gan-x4s64w8d6e180_
↪8xb4-lr1e-4-600k_df2k-ost.py
CUDA_VISIBLE_DEVICES=-1 python tools/train.py configs/swinir/swinir_psnr-
↪x4s64w8d6e180_8xb4-lr1e-4-600k_df2k-ost.py
CUDA_VISIBLE_DEVICES=-1 python tools/train.py configs/swinir/swinir_gan-x4s64w8d9e240_
↪8xb4-lr1e-4-600k_df2k-ost.py
CUDA_VISIBLE_DEVICES=-1 python tools/train.py configs/swinir/swinir_psnr-
↪x4s64w8d9e240_8xb4-lr1e-4-600k_df2k-ost.py

## 004 Grayscale Image Deoising (middle size)
CUDA_VISIBLE_DEVICES=-1 python tools/train.py configs/swinir/swinir_s128w8d6e180_8xb1-
↪lr2e-4-1600k_dfwb-grayDN15.py
CUDA_VISIBLE_DEVICES=-1 python tools/train.py configs/swinir/swinir_s128w8d6e180_8xb1-
↪lr2e-4-1600k_dfwb-grayDN25.py
CUDA_VISIBLE_DEVICES=-1 python tools/train.py configs/swinir/swinir_s128w8d6e180_8xb1-
↪lr2e-4-1600k_dfwb-grayDN50.py

## 005 Color Image Deoising (middle size)
CUDA_VISIBLE_DEVICES=-1 python tools/train.py configs/swinir/swinir_s128w8d6e180_8xb1-
↪lr2e-4-1600k_dfwb-colorDN15.py
```

(下页继续)

(续上页)

```
CUDA_VISIBLE_DEVICES=-1 python tools/train.py configs/swinir/swinir_s128w8d6e180_8xb1-
↪lr2e-4-1600k_dfwb-colorDN25.py
CUDA_VISIBLE_DEVICES=-1 python tools/train.py configs/swinir/swinir_s128w8d6e180_8xb1-
↪lr2e-4-1600k_dfwb-colorDN50.py

## 006 JPEG Compression Artifact Reduction (middle size, using window_size=7 because␣
↪JPEG encoding uses 8x8 blocks)
## grayscale
CUDA_VISIBLE_DEVICES=-1 python tools/train.py configs/swinir/swinir_s126w7d6e180_8xb1-
↪lr2e-4-1600k_dfwb-grayCAR10.py
CUDA_VISIBLE_DEVICES=-1 python tools/train.py configs/swinir/swinir_s126w7d6e180_8xb1-
↪lr2e-4-1600k_dfwb-grayCAR20.py
CUDA_VISIBLE_DEVICES=-1 python tools/train.py configs/swinir/swinir_s126w7d6e180_8xb1-
↪lr2e-4-1600k_dfwb-grayCAR30.py
CUDA_VISIBLE_DEVICES=-1 python tools/train.py configs/swinir/swinir_s126w7d6e180_8xb1-
↪lr2e-4-1600k_dfwb-grayCAR40.py

## color
CUDA_VISIBLE_DEVICES=-1 python tools/train.py configs/swinir/swinir_s126w7d6e180_8xb1-
↪lr2e-4-1600k_dfwb-colorCAR10.py
CUDA_VISIBLE_DEVICES=-1 python tools/train.py configs/swinir/swinir_s126w7d6e180_8xb1-
↪lr2e-4-1600k_dfwb-colorCAR20.py
CUDA_VISIBLE_DEVICES=-1 python tools/train.py configs/swinir/swinir_s126w7d6e180_8xb1-
↪lr2e-4-1600k_dfwb-colorCAR30.py
CUDA_VISIBLE_DEVICES=-1 python tools/train.py configs/swinir/swinir_s126w7d6e180_8xb1-
↪lr2e-4-1600k_dfwb-colorCAR40.py



## 单个GPU上训练
## 001 Classical Image Super-Resolution (middle size)
## (setting1: when model is trained on DIV2K and with training_patch_size=48)
python tools/train.py configs/swinir/swinir_x2s48w8d6e180_8xb4-lr2e-4-500k_div2k.py
python tools/train.py configs/swinir/swinir_x3s48w8d6e180_8xb4-lr2e-4-500k_div2k.py
python tools/train.py configs/swinir/swinir_x4s48w8d6e180_8xb4-lr2e-4-500k_div2k.py

## (setting2: when model is trained on DIV2K+Flickr2K and with training_patch_size=64)
python tools/train.py configs/swinir/swinir_x2s64w8d6e180_8xb4-lr2e-4-500k_df2k.py
python tools/train.py configs/swinir/swinir_x3s64w8d6e180_8xb4-lr2e-4-500k_df2k.py
python tools/train.py configs/swinir/swinir_x4s64w8d6e180_8xb4-lr2e-4-500k_df2k.py

## 002 Lightweight Image Super-Resolution (small size)
python tools/train.py configs/swinir/swinir_x2s64w8d4e60_8xb4-lr2e-4-500k_div2k.py
```

(下页继续)

(续上页)

```
python tools/train.py configs/swinir/swinir_x3s64w8d4e60_8xb4-lr2e-4-500k_div2k.py
python tools/train.py configs/swinir/swinir_x4s64w8d4e60_8xb4-lr2e-4-500k_div2k.py

## 003 Real-World Image Super-Resolution
python tools/train.py configs/swinir/swinir_gan-x2s64w8d6e180_8xb4-lr1e-4-600k_df2k-
↪ost.py
python tools/train.py configs/swinir/swinir_psnr-x2s64w8d6e180_8xb4-lr1e-4-600k_df2k-
↪ost.py
python tools/train.py configs/swinir/swinir_gan-x4s64w8d6e180_8xb4-lr1e-4-600k_df2k-
↪ost.py
python tools/train.py configs/swinir/swinir_psnr-x4s64w8d6e180_8xb4-lr1e-4-600k_df2k-
↪ost.py
python tools/train.py configs/swinir/swinir_gan-x4s64w8d9e240_8xb4-lr1e-4-600k_df2k-
↪ost.py
python tools/train.py configs/swinir/swinir_psnr-x4s64w8d9e240_8xb4-lr1e-4-600k_df2k-
↪ost.py

## 004 Grayscale Image Deoising (middle size)
python tools/train.py configs/swinir/swinir_s128w8d6e180_8xb1-lr2e-4-1600k_dfwb-
↪grayDN15.py
python tools/train.py configs/swinir/swinir_s128w8d6e180_8xb1-lr2e-4-1600k_dfwb-
↪grayDN25.py
python tools/train.py configs/swinir/swinir_s128w8d6e180_8xb1-lr2e-4-1600k_dfwb-
↪grayDN50.py

## 005 Color Image Deoising (middle size)
python tools/train.py configs/swinir/swinir_s128w8d6e180_8xb1-lr2e-4-1600k_dfwb-
↪colorDN15.py
python tools/train.py configs/swinir/swinir_s128w8d6e180_8xb1-lr2e-4-1600k_dfwb-
↪colorDN25.py
python tools/train.py configs/swinir/swinir_s128w8d6e180_8xb1-lr2e-4-1600k_dfwb-
↪colorDN50.py

## 006 JPEG Compression Artifact Reduction (middle size, using window_size=7 because␣
↪JPEG encoding uses 8x8 blocks)
## grayscale
python tools/train.py configs/swinir/swinir_s126w7d6e180_8xb1-lr2e-4-1600k_dfwb-
↪grayCAR10.py
python tools/train.py configs/swinir/swinir_s126w7d6e180_8xb1-lr2e-4-1600k_dfwb-
↪grayCAR20.py
python tools/train.py configs/swinir/swinir_s126w7d6e180_8xb1-lr2e-4-1600k_dfwb-
↪grayCAR30.py
python tools/train.py configs/swinir/swinir_s126w7d6e180_8xb1-lr2e-4-1600k_dfwb-
↪grayCAR40.py
```

(下页继续)

(续上页)

```
## color
python tools/train.py configs/swinir/swinir_s126w7d6e180_8xb1-lr2e-4-1600k_dfwb-
↪colorCAR10.py
python tools/train.py configs/swinir/swinir_s126w7d6e180_8xb1-lr2e-4-1600k_dfwb-
↪colorCAR20.py
python tools/train.py configs/swinir/swinir_s126w7d6e180_8xb1-lr2e-4-1600k_dfwb-
↪colorCAR30.py
python tools/train.py configs/swinir/swinir_s126w7d6e180_8xb1-lr2e-4-1600k_dfwb-
↪colorCAR40.py



## 多个GPU上训练
## 001 Classical Image Super-Resolution (middle size)
## (setting1: when model is trained on DIV2K and with training_patch_size=48)
./tools/dist_train.sh configs/swinir/swinir_x2s48w8d6e180_8xb4-lr2e-4-500k_div2k.py 8
./tools/dist_train.sh configs/swinir/swinir_x3s48w8d6e180_8xb4-lr2e-4-500k_div2k.py 8
./tools/dist_train.sh configs/swinir/swinir_x4s48w8d6e180_8xb4-lr2e-4-500k_div2k.py 8

## (setting2: when model is trained on DIV2K+Flickr2K and with training_patch_size=64)
./tools/dist_train.sh configs/swinir/swinir_x2s64w8d6e180_8xb4-lr2e-4-500k_df2k.py 8
./tools/dist_train.sh configs/swinir/swinir_x3s64w8d6e180_8xb4-lr2e-4-500k_df2k.py 8
./tools/dist_train.sh configs/swinir/swinir_x4s64w8d6e180_8xb4-lr2e-4-500k_df2k.py 8

## 002 Lightweight Image Super-Resolution (small size)
./tools/dist_train.sh configs/swinir/swinir_x2s64w8d4e60_8xb4-lr2e-4-500k_div2k.py 8
./tools/dist_train.sh configs/swinir/swinir_x3s64w8d4e60_8xb4-lr2e-4-500k_div2k.py 8
./tools/dist_train.sh configs/swinir/swinir_x4s64w8d4e60_8xb4-lr2e-4-500k_div2k.py 8

## 003 Real-World Image Super-Resolution
./tools/dist_train.sh configs/swinir/swinir_gan-x2s64w8d6e180_8xb4-lr1e-4-600k_df2k-
↪ost.py 8
./tools/dist_train.sh configs/swinir/swinir_psnr-x2s64w8d6e180_8xb4-lr1e-4-600k_df2k-
↪ost.py 8
./tools/dist_train.sh configs/swinir/swinir_gan-x4s64w8d6e180_8xb4-lr1e-4-600k_df2k-
↪ost.py 8
./tools/dist_train.sh configs/swinir/swinir_psnr-x4s64w8d6e180_8xb4-lr1e-4-600k_df2k-
↪ost.py 8
./tools/dist_train.sh configs/swinir/swinir_gan-x4s64w8d9e240_8xb4-lr1e-4-600k_df2k-
↪ost.py 8
./tools/dist_train.sh configs/swinir/swinir_psnr-x4s64w8d9e240_8xb4-lr1e-4-600k_df2k-
↪ost.py 8
```

(下页继续)

(续上页)

```
## 004 Grayscale Image Deoising (middle size)
./tools/dist_train.sh configs/swinir/swinir_s128w8d6e180_8xb1-lr2e-4-1600k_dfwb-
↪grayDN15.py 8
./tools/dist_train.sh configs/swinir/swinir_s128w8d6e180_8xb1-lr2e-4-1600k_dfwb-
↪grayDN25.py 8
./tools/dist_train.sh configs/swinir/swinir_s128w8d6e180_8xb1-lr2e-4-1600k_dfwb-
↪grayDN50.py 8

## 005 Color Image Deoising (middle size)
./tools/dist_train.sh configs/swinir/swinir_s128w8d6e180_8xb1-lr2e-4-1600k_dfwb-
↪colorDN15.py 8
./tools/dist_train.sh configs/swinir/swinir_s128w8d6e180_8xb1-lr2e-4-1600k_dfwb-
↪colorDN25.py 8
./tools/dist_train.sh configs/swinir/swinir_s128w8d6e180_8xb1-lr2e-4-1600k_dfwb-
↪colorDN50.py 8

## 006 JPEG Compression Artifact Reduction (middle size, using window_size=7 because␣
↪JPEG encoding uses 8x8 blocks)
## grayscale
./tools/dist_train.sh configs/swinir/swinir_s126w7d6e180_8xb1-lr2e-4-1600k_dfwb-
↪grayCAR10.py 8
./tools/dist_train.sh configs/swinir/swinir_s126w7d6e180_8xb1-lr2e-4-1600k_dfwb-
↪grayCAR20.py 8
./tools/dist_train.sh configs/swinir/swinir_s126w7d6e180_8xb1-lr2e-4-1600k_dfwb-
↪grayCAR30.py 8
./tools/dist_train.sh configs/swinir/swinir_s126w7d6e180_8xb1-lr2e-4-1600k_dfwb-
↪grayCAR40.py 8

## color
./tools/dist_train.sh configs/swinir/swinir_s126w7d6e180_8xb1-lr2e-4-1600k_dfwb-
↪colorCAR10.py 8
./tools/dist_train.sh configs/swinir/swinir_s126w7d6e180_8xb1-lr2e-4-1600k_dfwb-
↪colorCAR20.py 8
./tools/dist_train.sh configs/swinir/swinir_s126w7d6e180_8xb1-lr2e-4-1600k_dfwb-
↪colorCAR30.py 8
./tools/dist_train.sh configs/swinir/swinir_s126w7d6e180_8xb1-lr2e-4-1600k_dfwb-
↪colorCAR40.py 8
```

更多细节可以参考 train_test.md 中的 **Train a model** 部分。

测试

您可以使用以下命令来测试模型。

```
## CPU上测试
## 001 Classical Image Super-Resolution (middle size)
## (setting1: when model is trained on DIV2K and with training_patch_size=48)
CUDA_VISIBLE_DEVICES=-1 python tools/test.py configs/swinir/swinir_x2s48w8d6e180_8xb4-
↪lr2e-4-500k_div2k.py https://download.openmmlab.com/mmediting/swinir/swinir_
↪x2s48w8d6e180_8xb4-lr2e-4-500k_div2k-ed2d419e.pth

CUDA_VISIBLE_DEVICES=-1 python tools/test.py configs/swinir/swinir_x3s48w8d6e180_8xb4-
↪lr2e-4-500k_div2k.py https://download.openmmlab.com/mmediting/swinir/swinir_
↪x3s48w8d6e180_8xb4-lr2e-4-500k_div2k-926950f1.pth

CUDA_VISIBLE_DEVICES=-1 python tools/test.py configs/swinir/swinir_x4s48w8d6e180_8xb4-
↪lr2e-4-500k_div2k.py https://download.openmmlab.com/mmediting/swinir/swinir_
↪x4s48w8d6e180_8xb4-lr2e-4-500k_div2k-88e4903d.pth

## (setting2: when model is trained on DIV2K+Flickr2K and with training_patch_size=64)
CUDA_VISIBLE_DEVICES=-1 python tools/test.py configs/swinir/swinir_x2s64w8d6e180_8xb4-
↪lr2e-4-500k_df2k.py https://download.openmmlab.com/mmediting/swinir/swinir_
↪x2s64w8d6e180_8xb4-lr2e-4-500k_df2k-69e15fb6.pth

CUDA_VISIBLE_DEVICES=-1 python tools/test.py configs/swinir/swinir_x3s64w8d6e180_8xb4-
↪lr2e-4-500k_df2k.py https://download.openmmlab.com/mmediting/swinir/swinir_
↪x3s64w8d6e180_8xb4-lr2e-4-500k_df2k-d6982f7b.pth

CUDA_VISIBLE_DEVICES=-1 python tools/test.py configs/swinir/swinir_x4s64w8d6e180_8xb4-
↪lr2e-4-500k_df2k.py https://download.openmmlab.com/mmediting/swinir/swinir_
↪x4s64w8d6e180_8xb4-lr2e-4-500k_df2k-0502d775.pth


## 002 Lightweight Image Super-Resolution (small size)
CUDA_VISIBLE_DEVICES=-1 python tools/test.py configs/swinir/swinir_x2s64w8d4e60_8xb4-
↪lr2e-4-500k_div2k.py https://download.openmmlab.com/mmediting/swinir/swinir_
↪x2s64w8d4e60_8xb4-lr2e-4-500k_div2k-131d3f64.pth

CUDA_VISIBLE_DEVICES=-1 python tools/test.py configs/swinir/swinir_x3s64w8d4e60_8xb4-
↪lr2e-4-500k_div2k.py https://download.openmmlab.com/mmediting/swinir/swinir_
↪x3s64w8d4e60_8xb4-lr2e-4-500k_div2k-309cb239.pth

CUDA_VISIBLE_DEVICES=-1 python tools/test.py configs/swinir/swinir_x4s64w8d4e60_8xb4-
↪lr2e-4-500k_div2k.py https://download.openmmlab.com/mmediting/swinir/swinir_
↪x4s64w8d4e60_8xb4-lr2e-4-500k_div2k-d6622d03.pth

## 003 Real-World Image Super-Resolution
CUDA_VISIBLE_DEVICES=-1 python tools/test.py configs/swinir/swinir_gan-x2s64w8d6e180_
↪8xb4-lr1e-4-600k_df2k-ost.py https://download.openmmlab.com/mmediting/swinir/swinir_
↪gan-x2s64w8d6e180_8xb4-lr1e-4-600k_df2k-os-c6425057.pth
```

(下页继续)

(续上页)

```
CUDA_VISIBLE_DEVICES=-1 python tools/test.py configs/swinir/swinir_psnr-x2s64w8d6e180_
↪8xb4-lr1e-4-600k_df2k-ost.py https://download.openmmlab.com/mmediting/swinir/swinir_
↪psnr-x2s64w8d6e180_8xb4-lr1e-4-600k_df2k-os-6f0c425f.pth

CUDA_VISIBLE_DEVICES=-1 python tools/test.py configs/swinir/swinir_gan-x4s64w8d6e180_
↪8xb4-lr1e-4-600k_df2k-ost.py https://download.openmmlab.com/mmediting/swinir/swinir_
↪gan-x4s64w8d6e180_8xb4-lr1e-4-600k_df2k-os-36960d18.pth

CUDA_VISIBLE_DEVICES=-1 python tools/test.py configs/swinir/swinir_psnr-x4s64w8d6e180_
↪8xb4-lr1e-4-600k_df2k-ost.py https://download.openmmlab.com/mmediting/swinir/swinir_
↪psnr-x4s64w8d6e180_8xb4-lr1e-4-600k_df2k-os-a016a72f.pth

CUDA_VISIBLE_DEVICES=-1 python tools/test.py configs/swinir/swinir_gan-x4s64w8d9e240_
↪8xb4-lr1e-4-600k_df2k-ost.py https://download.openmmlab.com/mmediting/swinir/swinir_
↪gan-x4s64w8d9e240_8xb4-lr1e-4-600k_df2k-os-9f1599b5.pth

CUDA_VISIBLE_DEVICES=-1 python tools/test.py configs/swinir/swinir_psnr-x4s64w8d9e240_
↪8xb4-lr1e-4-600k_df2k-ost.py https://download.openmmlab.com/mmediting/swinir/swinir_
↪psnr-x4s64w8d9e240_8xb4-lr1e-4-600k_df2k-os-25f1722a.pth

## 004 Grayscale Image Deoising (middle size)
CUDA_VISIBLE_DEVICES=-1 python tools/test.py configs/swinir/swinir_s128w8d6e180_8xb1-
↪lr2e-4-1600k_dfwb-grayDN15.py https://download.openmmlab.com/mmediting/swinir/
↪swinir_s128w8d6e180_8xb1-lr2e-4-1600k_dfwb-grayDN15-6782691b.pth

CUDA_VISIBLE_DEVICES=-1 python tools/test.py configs/swinir/swinir_s128w8d6e180_8xb1-
↪lr2e-4-1600k_dfwb-grayDN25.py https://download.openmmlab.com/mmediting/swinir/
↪swinir_s128w8d6e180_8xb1-lr2e-4-1600k_dfwb-grayDN25-d0d8d4da.pth

CUDA_VISIBLE_DEVICES=-1 python tools/test.py configs/swinir/swinir_s128w8d6e180_8xb1-
↪lr2e-4-1600k_dfwb-grayDN50.py https://download.openmmlab.com/mmediting/swinir/
↪swinir_s128w8d6e180_8xb1-lr2e-4-1600k_dfwb-grayDN50-54c9968a.pth

## 005 Color Image Deoising (middle size)
CUDA_VISIBLE_DEVICES=-1 python tools/test.py configs/swinir/swinir_s128w8d6e180_8xb1-
↪lr2e-4-1600k_dfwb-colorDN15.py https://download.openmmlab.com/mmediting/swinir/
↪swinir_s128w8d6e180_8xb1-lr2e-4-1600k_dfwb-colorDN15-c74a2cee.pth

CUDA_VISIBLE_DEVICES=-1 python tools/test.py configs/swinir/swinir_s128w8d6e180_8xb1-
↪lr2e-4-1600k_dfwb-colorDN25.py https://download.openmmlab.com/mmediting/swinir/
↪swinir_s128w8d6e180_8xb1-lr2e-4-1600k_dfwb-colorDN25-df2b1c0c.pth
```

(下页继续)

(续上页)

```
CUDA_VISIBLE_DEVICES=-1 python tools/test.py configs/swinir/swinir_s128w8d6e180_8xb1-
↪lr2e-4-1600k_dfwb-colorDN50.py https://download.openmmlab.com/mmediting/swinir/
↪swinir_s128w8d6e180_8xb1-lr2e-4-1600k_dfwb-colorDN50-e369874c.pth

## 006 JPEG Compression Artifact Reduction (middle size, using window_size=7 because␣
↪JPEG encoding usesx8 blocks)
## grayscale
CUDA_VISIBLE_DEVICES=-1 python tools/test.py configs/swinir/swinir_s126w7d6e180_8xb1-
↪lr2e-4-1600k_dfwb-grayCAR10.py https://download.openmmlab.com/mmediting/swinir/
↪swinir_s126w7d6e180_8xb1-lr2e-4-1600k_dfwb-grayCAR10-da93c8e9.pth

CUDA_VISIBLE_DEVICES=-1 python tools/test.py configs/swinir/swinir_s126w7d6e180_8xb1-
↪lr2e-4-1600k_dfwb-grayCAR20.py https://download.openmmlab.com/mmediting/swinir/
↪swinir_s126w7d6e180_8xb1-lr2e-4-1600k_dfwb-grayCAR20-d47367b1.pth

CUDA_VISIBLE_DEVICES=-1 python tools/test.py configs/swinir/swinir_s126w7d6e180_8xb1-
↪lr2e-4-1600k_dfwb-grayCAR30.py https://download.openmmlab.com/mmediting/swinir/
↪swinir_s126w7d6e180_8xb1-lr2e-4-1600k_dfwb-grayCAR30-52c083cf.pth

CUDA_VISIBLE_DEVICES=-1 python tools/test.py configs/swinir/swinir_s126w7d6e180_8xb1-
↪lr2e-4-1600k_dfwb-grayCAR40.py https://download.openmmlab.com/mmediting/swinir/
↪swinir_s126w7d6e180_8xb1-lr2e-4-1600k_dfwb-grayCAR40-803e8d9b.pth


## color
CUDA_VISIBLE_DEVICES=-1 python tools/test.py configs/swinir/swinir_s126w7d6e180_8xb1-
↪lr2e-4-1600k_dfwb-colorCAR10.py https://download.openmmlab.com/mmediting/swinir/
↪swinir_s126w7d6e180_8xb1-lr2e-4-1600k_dfwb-colorCAR10-09aafadc.pth

CUDA_VISIBLE_DEVICES=-1 python tools/test.py configs/swinir/swinir_s126w7d6e180_8xb1-
↪lr2e-4-1600k_dfwb-colorCAR20.py https://download.openmmlab.com/mmediting/swinir/
↪swinir_s126w7d6e180_8xb1-lr2e-4-1600k_dfwb-colorCAR20-b8a42b5e.pth

CUDA_VISIBLE_DEVICES=-1 python tools/test.py configs/swinir/swinir_s126w7d6e180_8xb1-
↪lr2e-4-1600k_dfwb-colorCAR30.py https://download.openmmlab.com/mmediting/swinir/
↪swinir_s126w7d6e180_8xb1-lr2e-4-1600k_dfwb-colorCAR30-e9fe6859.pth

CUDA_VISIBLE_DEVICES=-1 python tools/test.py configs/swinir/swinir_s126w7d6e180_8xb1-
↪lr2e-4-1600k_dfwb-colorCAR40.py https://download.openmmlab.com/mmediting/swinir/
↪swinir_s126w7d6e180_8xb1-lr2e-4-1600k_dfwb-colorCAR40-5b77a6e6.pth
```

(下页继续)

(续上页)

```
## 单个GPU上测试
## 001 Classical Image Super-Resolution (middle size)
## (setting1: when model is trained on DIV2K and with training_patch_size=48)
python tools/test.py configs/swinir/swinir_x2s48w8d6e180_8xb4-lr2e-4-500k_div2k.py␣
↪https://download.openmmlab.com/mmediting/swinir/swinir_x2s48w8d6e180_8xb4-lr2e-4-
↪500k_div2k-ed2d419e.pth

python tools/test.py configs/swinir/swinir_x3s48w8d6e180_8xb4-lr2e-4-500k_div2k.py␣
↪https://download.openmmlab.com/mmediting/swinir/swinir_x3s48w8d6e180_8xb4-lr2e-4-
↪500k_div2k-926950f1.pth

python tools/test.py configs/swinir/swinir_x4s48w8d6e180_8xb4-lr2e-4-500k_div2k.py␣
↪https://download.openmmlab.com/mmediting/swinir/swinir_x4s48w8d6e180_8xb4-lr2e-4-
↪500k_div2k-88e4903d.pth

## (setting2: when model is trained on DIV2K+Flickr2K and with training_patch_size=64)
python tools/test.py configs/swinir/swinir_x2s64w8d6e180_8xb4-lr2e-4-500k_df2k.py␣
↪https://download.openmmlab.com/mmediting/swinir/swinir_x2s64w8d6e180_8xb4-lr2e-4-
↪500k_df2k-69e15fb6.pth

python tools/test.py configs/swinir/swinir_x3s64w8d6e180_8xb4-lr2e-4-500k_df2k.py␣
↪https://download.openmmlab.com/mmediting/swinir/swinir_x3s64w8d6e180_8xb4-lr2e-4-
↪500k_df2k-d6982f7b.pth

python tools/test.py configs/swinir/swinir_x4s64w8d6e180_8xb4-lr2e-4-500k_df2k.py␣
↪https://download.openmmlab.com/mmediting/swinir/swinir_x4s64w8d6e180_8xb4-lr2e-4-
↪500k_df2k-0502d775.pth


## 002 Lightweight Image Super-Resolution (small size)
python tools/test.py configs/swinir/swinir_x2s64w8d4e60_8xb4-lr2e-4-500k_div2k.py␣
↪https://download.openmmlab.com/mmediting/swinir/swinir_x2s64w8d4e60_8xb4-lr2e-4-
↪500k_div2k-131d3f64.pth

python tools/test.py configs/swinir/swinir_x3s64w8d4e60_8xb4-lr2e-4-500k_div2k.py␣
↪https://download.openmmlab.com/mmediting/swinir/swinir_x3s64w8d4e60_8xb4-lr2e-4-
↪500k_div2k-309cb239.pth

python tools/test.py configs/swinir/swinir_x4s64w8d4e60_8xb4-lr2e-4-500k_div2k.py␣
↪https://download.openmmlab.com/mmediting/swinir/swinir_x4s64w8d4e60_8xb4-lr2e-4-
↪500k_div2k-d6622d03.pth
```

(下页继续)

(续上页)

```
## 003 Real-World Image Super-Resolution
python tools/test.py configs/swinir/swinir_gan-x2s64w8d6e180_8xb4-lr1e-4-600k_df2k-
↪ost.py https://download.openmmlab.com/mmediting/swinir/swinir_gan-x2s64w8d6e180_
↪8xb4-lr1e-4-600k_df2k-os-c6425057.pth

python tools/test.py configs/swinir/swinir_psnr-x2s64w8d6e180_8xb4-lr1e-4-600k_df2k-
↪ost.py https://download.openmmlab.com/mmediting/swinir/swinir_psnr-x2s64w8d6e180_
↪8xb4-lr1e-4-600k_df2k-os-6f0c425f.pth

python tools/test.py configs/swinir/swinir_gan-x4s64w8d6e180_8xb4-lr1e-4-600k_df2k-
↪ost.py https://download.openmmlab.com/mmediting/swinir/swinir_gan-x4s64w8d6e180_
↪8xb4-lr1e-4-600k_df2k-os-36960d18.pth

python tools/test.py configs/swinir/swinir_psnr-x4s64w8d6e180_8xb4-lr1e-4-600k_df2k-
↪ost.py https://download.openmmlab.com/mmediting/swinir/swinir_psnr-x4s64w8d6e180_
↪8xb4-lr1e-4-600k_df2k-os-a016a72f.pth

python tools/test.py configs/swinir/swinir_gan-x4s64w8d9e240_8xb4-lr1e-4-600k_df2k-
↪ost.py https://download.openmmlab.com/mmediting/swinir/swinir_gan-x4s64w8d9e240_
↪8xb4-lr1e-4-600k_df2k-os-9f1599b5.pth

python tools/test.py configs/swinir/swinir_psnr-x4s64w8d9e240_8xb4-lr1e-4-600k_df2k-
↪ost.py https://download.openmmlab.com/mmediting/swinir/swinir_psnr-x4s64w8d9e240_
↪8xb4-lr1e-4-600k_df2k-os-25f1722a.pth


## 004 Grayscale Image Deoising (middle size)
python tools/test.py configs/swinir/swinir_s128w8d6e180_8xb1-lr2e-4-1600k_dfwb-
↪grayDN15.py https://download.openmmlab.com/mmediting/swinir/swinir_s128w8d6e180_
↪8xb1-lr2e-4-1600k_dfwb-grayDN15-6782691b.pth

python tools/test.py configs/swinir/swinir_s128w8d6e180_8xb1-lr2e-4-1600k_dfwb-
↪grayDN25.py https://download.openmmlab.com/mmediting/swinir/swinir_s128w8d6e180_
↪8xb1-lr2e-4-1600k_dfwb-grayDN25-d0d8d4da.pth

python tools/test.py configs/swinir/swinir_s128w8d6e180_8xb1-lr2e-4-1600k_dfwb-
↪grayDN50.py https://download.openmmlab.com/mmediting/swinir/swinir_s128w8d6e180_
↪8xb1-lr2e-4-1600k_dfwb-grayDN50-54c9968a.pth


## 005 Color Image Deoising (middle size)
python tools/test.py configs/swinir/swinir_s128w8d6e180_8xb1-lr2e-4-1600k_dfwb-
↪colorDN15.py https://download.openmmlab.com/mmediting/swinir/swinir_s128w8d6e180_
↪8xb1-lr2e-4-1600k_dfwb-colorDN15-c74a2cee.pth
```

(下页继续)

(续上页)

```
python tools/test.py configs/swinir/swinir_s128w8d6e180_8xb1-lr2e-4-1600k_dfwb-
↪colorDN25.py https://download.openmmlab.com/mmediting/swinir/swinir_s128w8d6e180_
↪8xb1-lr2e-4-1600k_dfwb-colorDN25-df2b1c0c.pth

python tools/test.py configs/swinir/swinir_s128w8d6e180_8xb1-lr2e-4-1600k_dfwb-
↪colorDN50.py https://download.openmmlab.com/mmediting/swinir/swinir_s128w8d6e180_
↪8xb1-lr2e-4-1600k_dfwb-colorDN50-e369874c.pth


## 006 JPEG Compression Artifact Reduction (middle size, using window_size=7 because␣
↪JPEG encoding usesx8 blocks)
## grayscale
python tools/test.py configs/swinir/swinir_s126w7d6e180_8xb1-lr2e-4-1600k_dfwb-
↪grayCAR10.py https://download.openmmlab.com/mmediting/swinir/swinir_s126w7d6e180_
↪8xb1-lr2e-4-1600k_dfwb-grayCAR10-da93c8e9.pth

python tools/test.py configs/swinir/swinir_s126w7d6e180_8xb1-lr2e-4-1600k_dfwb-
↪grayCAR20.py https://download.openmmlab.com/mmediting/swinir/swinir_s126w7d6e180_
↪8xb1-lr2e-4-1600k_dfwb-grayCAR20-d47367b1.pth

python tools/test.py configs/swinir/swinir_s126w7d6e180_8xb1-lr2e-4-1600k_dfwb-
↪grayCAR30.py https://download.openmmlab.com/mmediting/swinir/swinir_s126w7d6e180_
↪8xb1-lr2e-4-1600k_dfwb-grayCAR30-52c083cf.pth

python tools/test.py configs/swinir/swinir_s126w7d6e180_8xb1-lr2e-4-1600k_dfwb-
↪grayCAR40.py https://download.openmmlab.com/mmediting/swinir/swinir_s126w7d6e180_
↪8xb1-lr2e-4-1600k_dfwb-grayCAR40-803e8d9b.pth


## color
python tools/test.py configs/swinir/swinir_s126w7d6e180_8xb1-lr2e-4-1600k_dfwb-
↪colorCAR10.py https://download.openmmlab.com/mmediting/swinir/swinir_s126w7d6e180_
↪8xb1-lr2e-4-1600k_dfwb-colorCAR10-09aafadc.pth

python tools/test.py configs/swinir/swinir_s126w7d6e180_8xb1-lr2e-4-1600k_dfwb-
↪colorCAR20.py https://download.openmmlab.com/mmediting/swinir/swinir_s126w7d6e180_
↪8xb1-lr2e-4-1600k_dfwb-colorCAR20-b8a42b5e.pth

python tools/test.py configs/swinir/swinir_s126w7d6e180_8xb1-lr2e-4-1600k_dfwb-
↪colorCAR30.py https://download.openmmlab.com/mmediting/swinir/swinir_s126w7d6e180_
↪8xb1-lr2e-4-1600k_dfwb-colorCAR30-e9fe6859.pth
```

(下页继续)

(续上页)

```
python tools/test.py configs/swinir/swinir_s126w7d6e180_8xb1-lr2e-4-1600k_dfwb-
↪colorCAR40.py https://download.openmmlab.com/mmediting/swinir/swinir_s126w7d6e180_
↪8xb1-lr2e-4-1600k_dfwb-colorCAR40-5b77a6e6.pth



## 多GPU测试
## 001 Classical Image Super-Resolution (middle size)
## (setting1: when model is trained on DIV2K and with training_patch_size=48)
./tools/dist_test.sh configs/swinir/swinir_x2s48w8d6e180_8xb4-lr2e-4-500k_div2k.py␣
↪https://download.openmmlab.com/mmediting/swinir/swinir_x2s48w8d6e180_8xb4-lr2e-4-
↪500k_div2k-ed2d419e.pth

./tools/dist_test.sh configs/swinir/swinir_x3s48w8d6e180_8xb4-lr2e-4-500k_div2k.py␣
↪https://download.openmmlab.com/mmediting/swinir/swinir_x3s48w8d6e180_8xb4-lr2e-4-
↪500k_div2k-926950f1.pth

./tools/dist_test.sh configs/swinir/swinir_x4s48w8d6e180_8xb4-lr2e-4-500k_div2k.py␣
↪https://download.openmmlab.com/mmediting/swinir/swinir_x4s48w8d6e180_8xb4-lr2e-4-
↪500k_div2k-88e4903d.pth

## (setting2: when model is trained on DIV2K+Flickr2K and with training_patch_size=64)
./tools/dist_test.sh configs/swinir/swinir_x2s64w8d6e180_8xb4-lr2e-4-500k_df2k.py␣
↪https://download.openmmlab.com/mmediting/swinir/swinir_x2s64w8d6e180_8xb4-lr2e-4-
↪500k_df2k-69e15fb6.pth

./tools/dist_test.sh configs/swinir/swinir_x3s64w8d6e180_8xb4-lr2e-4-500k_df2k.py␣
↪https://download.openmmlab.com/mmediting/swinir/swinir_x3s64w8d6e180_8xb4-lr2e-4-
↪500k_df2k-d6982f7b.pth

./tools/dist_test.sh configs/swinir/swinir_x4s64w8d6e180_8xb4-lr2e-4-500k_df2k.py␣
↪https://download.openmmlab.com/mmediting/swinir/swinir_x4s64w8d6e180_8xb4-lr2e-4-
↪500k_df2k-0502d775.pth

## 002 Lightweight Image Super-Resolution (small size)
./tools/dist_test.sh configs/swinir/swinir_x2s64w8d4e60_8xb4-lr2e-4-500k_div2k.py␣
↪https://download.openmmlab.com/mmediting/swinir/swinir_x2s64w8d4e60_8xb4-lr2e-4-
↪500k_div2k-131d3f64.pth

./tools/dist_test.sh configs/swinir/swinir_x3s64w8d4e60_8xb4-lr2e-4-500k_div2k.py␣
↪https://download.openmmlab.com/mmediting/swinir/swinir_x3s64w8d4e60_8xb4-lr2e-4-
↪500k_div2k-309cb239.pth
```

(下页继续)

(续上页)

```
./tools/dist_test.sh configs/swinir/swinir_x4s64w8d4e60_8xb4-lr2e-4-500k_div2k.py␣
↪https://download.openmmlab.com/mmediting/swinir/swinir_x4s64w8d4e60_8xb4-lr2e-4-
↪500k_div2k-d6622d03.pth

## 003 Real-World Image Super-Resolution
./tools/dist_test.sh configs/swinir/swinir_gan-x2s64w8d6e180_8xb4-lr1e-4-600k_df2k-
↪ost.py https://download.openmmlab.com/mmediting/swinir/swinir_gan-x2s64w8d6e180_
↪8xb4-lr1e-4-600k_df2k-os-c6425057.pth

./tools/dist_test.sh configs/swinir/swinir_psnr-x2s64w8d6e180_8xb4-lr1e-4-600k_df2k-
↪ost.py https://download.openmmlab.com/mmediting/swinir/swinir_psnr-x2s64w8d6e180_
↪8xb4-lr1e-4-600k_df2k-os-6f0c425f.pth

./tools/dist_test.sh configs/swinir/swinir_gan-x4s64w8d6e180_8xb4-lr1e-4-600k_df2k-
↪ost.py https://download.openmmlab.com/mmediting/swinir/swinir_gan-x4s64w8d6e180_
↪8xb4-lr1e-4-600k_df2k-os-36960d18.pth

./tools/dist_test.sh configs/swinir/swinir_psnr-x4s64w8d6e180_8xb4-lr1e-4-600k_df2k-
↪ost.py https://download.openmmlab.com/mmediting/swinir/swinir_psnr-x4s64w8d6e180_
↪8xb4-lr1e-4-600k_df2k-os-a016a72f.pth

./tools/dist_test.sh configs/swinir/swinir_gan-x4s64w8d9e240_8xb4-lr1e-4-600k_df2k-
↪ost.py https://download.openmmlab.com/mmediting/swinir/swinir_gan-x4s64w8d9e240_
↪8xb4-lr1e-4-600k_df2k-os-9f1599b5.pth

./tools/dist_test.sh configs/swinir/swinir_psnr-x4s64w8d9e240_8xb4-lr1e-4-600k_df2k-
↪ost.py https://download.openmmlab.com/mmediting/swinir/swinir_psnr-x4s64w8d9e240_
↪8xb4-lr1e-4-600k_df2k-os-25f1722a.pth

## 004 Grayscale Image Deoising (middle size)
./tools/dist_test.sh configs/swinir/swinir_s128w8d6e180_8xb1-lr2e-4-1600k_dfwb-
↪grayDN15.py https://download.openmmlab.com/mmediting/swinir/swinir_s128w8d6e180_
↪8xb1-lr2e-4-1600k_dfwb-grayDN15-6782691b.pth

./tools/dist_test.sh configs/swinir/swinir_s128w8d6e180_8xb1-lr2e-4-1600k_dfwb-
↪grayDN25.py https://download.openmmlab.com/mmediting/swinir/swinir_s128w8d6e180_
↪8xb1-lr2e-4-1600k_dfwb-grayDN25-d0d8d4da.pth

./tools/dist_test.sh configs/swinir/swinir_s128w8d6e180_8xb1-lr2e-4-1600k_dfwb-
↪grayDN50.py https://download.openmmlab.com/mmediting/swinir/swinir_s128w8d6e180_
↪8xb1-lr2e-4-1600k_dfwb-grayDN50-54c9968a.pth

## 005 Color Image Deoising (middle size)
```

(下页继续)

(续上页)

```
./tools/dist_test.sh configs/swinir/swinir_s128w8d6e180_8xb1-lr2e-4-1600k_dfwb-
↪colorDN15.py https://download.openmmlab.com/mmediting/swinir/swinir_s128w8d6e180_
↪8xb1-lr2e-4-1600k_dfwb-colorDN15-c74a2cee.pth

./tools/dist_test.sh configs/swinir/swinir_s128w8d6e180_8xb1-lr2e-4-1600k_dfwb-
↪colorDN25.py https://download.openmmlab.com/mmediting/swinir/swinir_s128w8d6e180_
↪8xb1-lr2e-4-1600k_dfwb-colorDN25-df2b1c0c.pth

./tools/dist_test.sh configs/swinir/swinir_s128w8d6e180_8xb1-lr2e-4-1600k_dfwb-
↪colorDN50.py https://download.openmmlab.com/mmediting/swinir/swinir_s128w8d6e180_
↪8xb1-lr2e-4-1600k_dfwb-colorDN50-e369874c.pth

## 006 JPEG Compression Artifact Reduction (middle size, using window_size=7 because␣
↪JPEG encoding uses 8x8 blocks)
## grayscale
./tools/dist_test.sh configs/swinir/swinir_s126w7d6e180_8xb1-lr2e-4-1600k_dfwb-
↪grayCAR10.py https://download.openmmlab.com/mmediting/swinir/swinir_s126w7d6e180_
↪8xb1-lr2e-4-1600k_dfwb-grayCAR10-da93c8e9.pth

./tools/dist_test.sh configs/swinir/swinir_s126w7d6e180_8xb1-lr2e-4-1600k_dfwb-
↪grayCAR20.py https://download.openmmlab.com/mmediting/swinir/swinir_s126w7d6e180_
↪8xb1-lr2e-4-1600k_dfwb-grayCAR20-d47367b1.pth

./tools/dist_test.sh configs/swinir/swinir_s126w7d6e180_8xb1-lr2e-4-1600k_dfwb-
↪grayCAR30.py https://download.openmmlab.com/mmediting/swinir/swinir_s126w7d6e180_
↪8xb1-lr2e-4-1600k_dfwb-grayCAR30-52c083cf.pth

./tools/dist_test.sh configs/swinir/swinir_s126w7d6e180_8xb1-lr2e-4-1600k_dfwb-
↪grayCAR40.py https://download.openmmlab.com/mmediting/swinir/swinir_s126w7d6e180_
↪8xb1-lr2e-4-1600k_dfwb-grayCAR40-803e8d9b.pth

## color
./tools/dist_test.sh configs/swinir/swinir_s126w7d6e180_8xb1-lr2e-4-1600k_dfwb-
↪colorCAR10.py https://download.openmmlab.com/mmediting/swinir/swinir_s126w7d6e180_
↪8xb1-lr2e-4-1600k_dfwb-colorCAR10-09aafadc.pth

./tools/dist_test.sh configs/swinir/swinir_s126w7d6e180_8xb1-lr2e-4-1600k_dfwb-
↪colorCAR20.py https://download.openmmlab.com/mmediting/swinir/swinir_s126w7d6e180_
↪8xb1-lr2e-4-1600k_dfwb-colorCAR20-b8a42b5e.pth

./tools/dist_test.sh configs/swinir/swinir_s126w7d6e180_8xb1-lr2e-4-1600k_dfwb-
↪colorCAR30.py https://download.openmmlab.com/mmediting/swinir/swinir_s126w7d6e180_
↪8xb1-lr2e-4-1600k_dfwb-colorCAR30-e9fe6859.pth
```

(下页继续)

(续上页)

```
./tools/dist_test.sh configs/swinir/swinir_s126w7d6e180_8xb1-lr2e-4-1600k_dfwb-
↪colorCAR40.py https://download.openmmlab.com/mmediting/swinir/swinir_s126w7d6e180_
↪8xb1-lr2e-4-1600k_dfwb-colorCAR30-e9fe6859.pth
```

更多细节可以参考 train_test.md 中的 **Test a pre-trained model** 部分。

37.3 Real-ESRGAN (ICCVW'2021)

任务: 图像超分辨率

```
@inproceedings{wang2021real,
  title={Real-ESRGAN: Training Real-World Blind Super-Resolution with Pure Synthetic␣
↪data},
  author={Wang, Xintao and Xie, Liangbin and Dong, Chao and Shan, Ying},
  booktitle={Proceedings of the IEEE/CVF International Conference on Computer Vision␣
↪Workshop (ICCVW)},
  pages={1905--1914},
  year={2021}
}
```

在 RGB 通道上进行评估，指标为 `PSNR/SSIM`。

37.3.1 快速开始

训练

您可以使用以下命令来训练模型。

```
## CPU上训练
CUDA_VISIBLE_DEVICES=-1 python tools/train.py configs/real_esrgan/realesrgan_
↪c64b23g32_4xb12-lr1e-4-400k_df2k-ost.py

## 单个GPU上训练
python tools/train.py configs/real_esrgan/realesrgan_c64b23g32_4xb12-lr1e-4-400k_df2k-
↪ost.py

## 多个GPU上训练
./tools/dist_train.sh configs/real_esrgan/realesrgan_c64b23g32_4xb12-lr1e-4-400k_df2k-
↪ost.py 8
```

更多细节可以参考 train_test.md 中的 **Train a model** 部分。

测试

您可以使用以下命令来测试模型。

```
## CPU上测试
CUDA_VISIBLE_DEVICES=-1 python tools/test.py configs/real_esrgan/realesrgan_c64b23g32_
↪4xb12-lr1e-4-400k_df2k-ost.py https://download.openmmlab.com/mmediting/restorers/
↪real_esrgan/realesrgan_c64b23g32_12x4_lr1e-4_400k_df2k_ost_20211010-34798885.pth

## 单个GPU上测试
python tools/test.py configs/real_esrgan/realesrgan_c64b23g32_4xb12-lr1e-4-400k_df2k-
↪ost.py https://download.openmmlab.com/mmediting/restorers/real_esrgan/realesrgan_
↪c64b23g32_12x4_lr1e-4_400k_df2k_ost_20211010-34798885.pth

## 多个GPU上测试
./tools/dist_test.sh configs/real_esrgan/realesrgan_c64b23g32_4xb12-lr1e-4-400k_df2k-
↪ost.py https://download.openmmlab.com/mmediting/restorers/real_esrgan/realesrgan_
↪c64b23g32_12x4_lr1e-4_400k_df2k_ost_20211010-34798885.pth 8
```

更多细节可以参考 train_test.md 中的 **Test a pre-trained model** 部分。

37.4 LIIF (CVPR’2021)

任务: 图像超分辨率

```
@inproceedings{chen2021learning,
  title={Learning continuous image representation with local implicit image function},
  author={Chen, Yinbo and Liu, Sifei and Wang, Xiaolong},
  booktitle={Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern␣
↪Recognition},
  pages={8628--8638},
  year={2021}
}
```

注:

- △ 指同上。
- 这两个配置仅在 *testing pipeline* 上有所不同。所以他们使用相同的检查点。
- 数据根据 EDSR 进行正则化。
- 在 RGB 通道上进行评估，在评估之前裁剪每个边界中的 `scale` 像素。

37.4.1 快速开始

训练

您可以使用以下命令来训练模型。

```
## CPU上训练
CUDA_VISIBLE_DEVICES=-1 python tools/train.py configs/liif/liif-edsr-norm_c64b16_
↪1xb16-1000k_div2k.py

## 单个GPU上训练
python tools/train.py configs/liif/liif-edsr-norm_c64b16_1xb16-1000k_div2k.py

## 多个GPU上训练
./tools/dist_train.sh configs/liif/liif-edsr-norm_c64b16_1xb16-1000k_div2k.py 8
```

更多细节可以参考 train_test.md 中的 **Train a model** 部分。

测试

您可以使用以下命令来测试模型。

```
## CPU上测试
CUDA_VISIBLE_DEVICES=-1 python tools/test.py configs/liif/liif-edsr-norm_c64b16_1xb16-
↪1000k_div2k.py https://download.openmmlab.com/mmediting/restorers/liif/liif_edsr_
↪norm_c64b16_g1_1000k_div2k_20210715-ab7ce3fc.pth

## 单个GPU上测试
python tools/test.py configs/liif/liif-edsr-norm_c64b16_1xb16-1000k_div2k.py https://
↪download.openmmlab.com/mmediting/restorers/liif/liif_edsr_norm_c64b16_g1_1000k_
↪div2k_20210715-ab7ce3fc.pth

## 多个GPU上测试
./tools/dist_test.sh configs/liif/liif-edsr-norm_c64b16_1xb16-1000k_div2k.py https://
↪download.openmmlab.com/mmediting/restorers/liif/liif_edsr_norm_c64b16_g1_1000k_
↪div2k_20210715-ab7ce3fc.pth 8
```

更多细节可以参考 train_test.md 中的 **Test a pre-trained model** 部分。

37.5 GLEAN (CVPR'2021)

任务: 图像超分辨率

```
@InProceedings{chan2021glean,
  author = {Chan, Kelvin CK and Wang, Xintao and Xu, Xiangyu and Gu, Jinwei and Loy,␣
↪Chen Change},
  title = {GLEAN: Generative Latent Bank for Large-Factor Image Super-Resolution},
  booktitle = {Proceedings of the IEEE conference on computer vision and pattern␣
↪recognition},
  year = {2021}
}
```

有关训练和测试中使用的元信息，请参阅此处。结果在 RGB 通道上进行评估。

37.5.1 快速开始

训练

您可以使用以下命令来训练模型。

```
## CPU上训练
CUDA_VISIBLE_DEVICES=-1 python tools/train.py configs/glean/glean_x8_2xb8_cat.py

## 单个GPU上训练
python tools/train.py configs/glean/glean_x8_2xb8_cat.py

## 多个GPU上训练
./tools/dist_train.sh configs/glean/glean_x8_2xb8_cat.py 8
```

更多细节可以参考 train_test.md 中的 **Train a model** 部分。

测试

您可以使用以下命令来测试模型。

```
## CPU上测试
CUDA_VISIBLE_DEVICES=-1 python tools/test.py configs/glean/glean_x8_2xb8_cat.py␣
↪https://download.openmmlab.com/mmediting/restorers/glean/glean_cat_8x_20210614-
↪d3ac8683.pth

## 单个GPU上测试
python tools/test.py configs/glean/glean_x8_2xb8_cat.py https://download.openmmlab.
↪com/mmediting/restorers/glean/glean_cat_8x_20210614-d3ac8683.pth
```

(下页继续)

(续上页)

```
## 多个GPU上测试
./tools/dist_test.sh configs/glean/glean_x8_2xb8_cat.py https://download.openmmlab.
↪com/mmediting/restorers/glean/glean_cat_8x_20210614-d3ac8683.pth 8
```

更多细节可以参考 train_test.md 中的 **Test a pre-trained model** 部分。

37.6 TTSR (CVPR'2020)

任务: 图像超分辨率

```
@inproceedings{yang2020learning,
  title={Learning texture transformer network for image super-resolution},
  author={Yang, Fuzhi and Yang, Huan and Fu, Jianlong and Lu, Hongtao and Guo,␣
↪Baining},
  booktitle={Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern␣
↪Recognition},
  pages={5791--5800},
  year={2020}
}
```

在 RGB 通道上进行评估，在评估之前裁剪每个边界中的 scale 像素。我们使用 PSNR 和 SSIM 作为指标。

37.6.1 快速开始

训练

您可以使用以下命令来训练模型。

```
## CPU上训练
CUDA_VISIBLE_DEVICES=-1 python tools/train.py configs/ttsr/ttsr-gan_x4c64b16_1xb9-
↪500k_CUFED.py

## 单个GPU上训练
python tools/train.py configs/ttsr/ttsr-gan_x4c64b16_1xb9-500k_CUFED.py

## 多个GPU上训练
./tools/dist_train.sh configs/ttsr/ttsr-gan_x4c64b16_1xb9-500k_CUFED.py 8
```

更多细节可以参考 train_test.md 中的 **Train a model** 部分。

测试

您可以使用以下命令来测试模型。

```
## CPU上测试
CUDA_VISIBLE_DEVICES=-1 python tools/test.py configs/ttsr/ttsr-gan_x4c64b16_1xb9-500k_
↪CUFED.py https://download.openmmlab.com/mmediting/restorers/ttsr/ttsr-gan_x4_c64b16_
↪g1_500k_CUFED_20210626-2ab28ca0.pth

## 单个GPU上测试
python tools/test.py configs/ttsr/ttsr-gan_x4c64b16_1xb9-500k_CUFED.py https://
↪download.openmmlab.com/mmediting/restorers/ttsr/ttsr-gan_x4_c64b16_g1_500k_CUFED_
↪20210626-2ab28ca0.pth

## 多个GPU上测试
./tools/dist_test.sh configs/ttsr/ttsr-gan_x4c64b16_1xb9-500k_CUFED.py https://
↪download.openmmlab.com/mmediting/restorers/ttsr/ttsr-gan_x4_c64b16_g1_500k_CUFED_
↪20210626-2ab28ca0.pth 8
```

更多细节可以参考 train_test.md 中的 **Test a pre-trained model** 部分。

37.7 DIC (CVPR'2020)

任务: 图像超分辨率

```
@inproceedings{ma2020deep,
  title={Deep face super-resolution with iterative collaboration between attentive␣
↪recovery and landmark estimation},
  author={Ma, Cheng and Jiang, Zhenyu and Rao, Yongming and Lu, Jiwen and Zhou, Jie},
  booktitle={Proceedings of the IEEE/CVF conference on computer vision and pattern␣
↪recognition},
  pages={5569--5578},
  year={2020}
}
```

在 RGB 通道上进行评估，在评估之前裁剪每个边界中的 `scale` 像素。我们使用 `PSNR` 和 `SSIM` 作为指标。

在 `dic_gan_x8c48b6_g4_150k_CelebAHQ` 的日志中，DICGAN 在 CelebA-HQ 测试集的前 9 张图片上进行了验证，因此下表中的 `PSNR/SSIM` 与日志数据不同。

`GPU 信息`: 训练过程中的 GPU 信息.

37.7.1 快速开始

训练

您可以使用以下命令来训练模型。

```
## CPU上训练
CUDA_VISIBLE_DEVICES=-1 python tools/train.py configs/dic/dic_gan-x8c48b6_4xb2-500k_
↪celeba-hq.py

## 单个GPU上训练
python tools/train.py configs/dic/dic_gan-x8c48b6_4xb2-500k_celeba-hq.py

## 多个GPU上训练
./tools/dist_train.sh configs/dic/dic_gan-x8c48b6_4xb2-500k_celeba-hq.py 8
```

更多细节可以参考 train_test.md 中的 **Train a model** 部分。

测试

您可以使用以下命令来测试模型。

```
## CPU上测试
CUDA_VISIBLE_DEVICES=-1 python tools/test.py configs/dic/dic_gan-x8c48b6_4xb2-500k_
↪celeba-hq.py https://download.openmmlab.com/mmediting/restorers/dic/dic_gan_x8c48b6_
↪g4_500k_CelebAHQ_20210625-3b89a358.pth

## 单个GPU上测试
python tools/test.py configs/dic/dic_gan-x8c48b6_4xb2-500k_celeba-hq.py https://
↪download.openmmlab.com/mmediting/restorers/dic/dic_gan_x8c48b6_g4_500k_CelebAHQ_
↪20210625-3b89a358.pth

## 多个GPU上测试
./tools/dist_test.sh configs/dic/dic_gan-x8c48b6_4xb2-500k_celeba-hq.py https://
↪download.openmmlab.com/mmediting/restorers/dic/dic_gan_x8c48b6_g4_500k_CelebAHQ_
↪20210625-3b89a358.pth 8
```

更多细节可以参考 train_test.md 中的 **Test a pre-trained model** 部分。

37.8 RDN (CVPR'2018)

任务: 图像超分辨率

```
@inproceedings{zhang2018residual,
  title={Residual dense network for image super-resolution},
  author={Zhang, Yulun and Tian, Yapeng and Kong, Yu and Zhong, Bineng and Fu, Yun},
  booktitle={Proceedings of the IEEE conference on computer vision and pattern
↪recognition},
  pages={2472--2481},
  year={2018}
}
```

在 RGB 通道上进行评估，在评估之前裁剪每个边界中的 scale 像素。我们使用 PSNR 和 SSIM 作为指标。

37.8.1 快速开始

训练

您可以使用以下命令来训练模型。

```
## CPU上训练
CUDA_VISIBLE_DEVICES=-1 python tools/train.py configs/rdn/rdn_x4c64b16_1xb16-1000k_
↪div2k.py

## 单个GPU上训练
python tools/train.py configs/rdn/rdn_x4c64b16_1xb16-1000k_div2k.py

## 多个GPU上训练
./tools/dist_train.sh configs/rdn/rdn_x4c64b16_1xb16-1000k_div2k.py 8
```

更多细节可以参考 train_test.md 中的 **Train a model** 部分。

测试

您可以使用以下命令来测试模型。

```
## CPU上测试
CUDA_VISIBLE_DEVICES=-1 python tools/test.py configs/rdn/rdn_x4c64b16_1xb16-1000k_
↪div2k.py https://download.openmmlab.com/mmediting/restorers/rdn/rdn_x4c64b16_g1_
↪1000k_div2k_20210419-3577d44f.pth

## 单个GPU上测试
python tools/test.py configs/rdn/rdn_x4c64b16_1xb16-1000k_div2k.py https://download.
↪openmmlab.com/mmediting/restorers/rdn/rdn_x4c64b16_g1_1000k_div2k_20210419-3577d44f.
↪pth
```

(下页继续)

(续上页)

```
## 多个GPU上测试
./tools/dist_test.sh configs/rdn/rdn_x4c64b16_1xb16-1000k_div2k.py https://download.
↪openmmlab.com/mmediting/restorers/rdn/rdn_x4c64b16_g1_1000k_div2k_20210419-3577d44f.
↪pth 8
```

更多细节可以参考 train_test.md 中的 **Test a pre-trained model** 部分。

37.9 ESRGAN (ECCVW'2018)

任务: 图像超分辨率

```
@inproceedings{wang2018esrgan,
  title={Esrgan: Enhanced super-resolution generative adversarial networks},
  author={Wang, Xintao and Yu, Ke and Wu, Shixiang and Gu, Jinjin and Liu, Yihao and␣
↪Dong, Chao and Qiao, Yu and Change Loy, Chen},
  booktitle={Proceedings of the European Conference on Computer Vision␣
↪Workshops(ECCVW)},
  pages={0--0},
  year={2018}
}
```

在 RGB 通道上进行评估，在评估之前裁剪每个边界中的 `scale` 像素。我们使用 `PSNR` 和 `SSIM` 作为指标。

37.9.1 快速开始

训练

您可以使用以下命令来训练模型。

```
## CPU上训练
CUDA_VISIBLE_DEVICES=-1 python tools/train.py configs/esrgan/esrgan_x4c64b23g32_1xb16-
↪400k_div2k.py

## 单个GPU上训练
python tools/train.py configs/esrgan/esrgan_x4c64b23g32_1xb16-400k_div2k.py

## 多个GPU上训练
./tools/dist_train.sh configs/esrgan/esrgan_x4c64b23g32_1xb16-400k_div2k.py 8
```

更多细节可以参考 train_test.md 中的 **Train a model** 部分。

测试

您可以使用以下命令来测试模型。

```
## CPU上测试
CUDA_VISIBLE_DEVICES=-1 python tools/test.py configs/esrgan/esrgan_x4c64b23g32_1xb16-
↪400k_div2k.py https://download.openmmlab.com/mmediting/restorers/esrgan/esrgan_
↪x4c64b23g32_1x16_400k_div2k_20200508-f8ccaf3b.pth

## 单个GPU上测试
python tools/test.py configs/esrgan/esrgan_x4c64b23g32_1xb16-400k_div2k.py https://
↪download.openmmlab.com/mmediting/restorers/esrgan/esrgan_x4c64b23g32_1x16_400k_
↪div2k_20200508-f8ccaf3b.pth

## 多个GPU上测试
./tools/dist_test.sh configs/esrgan/esrgan_x4c64b23g32_1xb16-400k_div2k.py https://
↪download.openmmlab.com/mmediting/restorers/esrgan/esrgan_x4c64b23g32_1x16_400k_
↪div2k_20200508-f8ccaf3b.pth 8
```

更多细节可以参考 train_test.md 中的 **Test a pre-trained model** 部分。

37.10 EDSR (CVPR'2017)

任务: 图像超分辨率

```
@inproceedings{lim2017enhanced,
  title={Enhanced deep residual networks for single image super-resolution},
  author={Lim, Bee and Son, Sanghyun and Kim, Heewon and Nah, Seungjun and Mu Lee,␣
↪Kyoung},
  booktitle={Proceedings of the IEEE conference on computer vision and pattern␣
↪recognition workshops},
  pages={136--144},
  year={2017}
}
```

在 RGB 通道上进行评估，在评估之前裁剪每个边界中的 `scale` 像素。我们使用 `PSNR` 和 `SSIM` 作为指标。

37.10.1 快速开始

训练

您可以使用以下命令来训练模型。

```
## CPU上训练
CUDA_VISIBLE_DEVICES=-1 python tools/train.py configs/edsr/edsr_x4c64b16_1xb16-300k_
↪div2k.py
```

(下页继续)

(续上页)

```
## 单个GPU上训练
python tools/train.py configs/edsr/edsr_x4c64b16_1xb16-300k_div2k.py

## 多个GPU上训练
./tools/dist_train.sh configs/edsr/edsr_x4c64b16_1xb16-300k_div2k.py 8
```

更多细节可以参考 train_test.md 中的 **Train a model** 部分。

测试

您可以使用以下命令来测试模型。

```
## CPU上测试
CUDA_VISIBLE_DEVICES=-1 python tools/test.py configs/edsr/edsr_x4c64b16_1xb16-300k_
↪div2k.py https://download.openmmlab.com/mmediting/restorers/edsr/edsr_x4c64b16_1x16_
↪300k_div2k_20200608-3c2af8a3.pth

## 单个GPU上测试
python tools/test.py configs/edsr/edsr_x4c64b16_1xb16-300k_div2k.py https://download.
↪openmmlab.com/mmediting/restorers/edsr/edsr_x4c64b16_1x16_300k_div2k_20200608-
↪3c2af8a3.pth

## 多个GPU上测试
./tools/dist_test.sh configs/edsr/edsr_x4c64b16_1xb16-300k_div2k.py https://download.
↪openmmlab.com/mmediting/restorers/edsr/edsr_x4c64b16_1x16_300k_div2k_20200608-
↪3c2af8a3.pth 8
```

更多细节可以参考 train_test.md 中的 **Test a pre-trained model** 部分。

37.11 SRGAN (CVPR'2016)

任务: 图像超分辨率

```
@inproceedings{ledig2016photo,
  title={Photo-realistic single image super-resolution using a generative adversarial␣
↪network},
  author={Ledig, Christian and Theis, Lucas and Husz{\'a}r, Ferenc and Caballero,␣
↪Jose and Cunningham, Andrew and Acosta, Alejandro and Aitken, Andrew and Tejani,␣
↪Alykhan and Totz, Johannes and Wang, Zehan},
  booktitle={Proceedings of the IEEE conference on computer vision and pattern␣
↪recognition workshops},
  year={2016}
}
```

在 RGB 通道上进行评估，在评估之前裁剪每个边界中的 scale 像素。我们使用 PSNR 和 SSIM 作为指标。

37.11.1 快速开始

训练

您可以使用以下命令来训练模型。

```
## CPU上训练
CUDA_VISIBLE_DEVICES=-1 python tools/train.py configs/srgan_resnet/srgan_x4c64b16_
↪1xb16-1000k_div2k.py

## 单个GPU上训练
python tools/train.py configs/srgan_resnet/srgan_x4c64b16_1xb16-1000k_div2k.py

## 多个GPU上训练
./tools/dist_train.sh configs/srgan_resnet/srgan_x4c64b16_1xb16-1000k_div2k.py 8
```

更多细节可以参考 train_test.md 中的 **Train a model** 部分。

测试

您可以使用以下命令来测试模型。

```
## CPU上测试
CUDA_VISIBLE_DEVICES=-1 python tools/test.py configs/srgan_resnet/srgan_x4c64b16_
↪1xb16-1000k_div2k.py https://download.openmmlab.com/mmediting/restorers/srresnet_
↪srgan/srgan_x4c64b16_1x16_1000k_div2k_20200606-a1f0810e.pth

## 单个GPU上测试
python tools/test.py configs/srgan_resnet/srgan_x4c64b16_1xb16-1000k_div2k.py https://
↪download.openmmlab.com/mmediting/restorers/srresnet_srgan/srgan_x4c64b16_1x16_1000k_
↪div2k_20200606-a1f0810e.pth

## 多个GPU上测试
./tools/dist_test.sh configs/srgan_resnet/srgan_x4c64b16_1xb16-1000k_div2k.py https://
↪download.openmmlab.com/mmediting/restorers/srresnet_srgan/srgan_x4c64b16_1x16_1000k_
↪div2k_20200606-a1f0810e.pth 8
```

更多细节可以参考 train_test.md 中的 **Test a pre-trained model** 部分。

37.12 SRCNN (TPAMI'2015)

任务: 图像超分辨率

```
@article{dong2015image,
  title={Image super-resolution using deep convolutional networks},
  author={Dong, Chao and Loy, Chen Change and He, Kaiming and Tang, Xiaoou},
  journal={IEEE transactions on pattern analysis and machine intelligence},
  volume={38},
  number={2},
  pages={295--307},
  year={2015},
  publisher={IEEE}
}
```

在 RGB 通道上进行评估，在评估之前裁剪每个边界中的 scale 像素。我们使用 PSNR 和 SSIM 作为指标。

37.12.1 快速开始

训练

您可以使用以下命令来训练模型。

```
## CPU上训练
CUDA_VISIBLE_DEVICES=-1 python tools/train.py configs/srcnn/srcnn_x4k915_1xb16-1000k_
↪div2k.py

## 单个GPU上训练
python tools/train.py configs/srcnn/srcnn_x4k915_1xb16-1000k_div2k.py

## 多个GPU上训练
./tools/dist_train.sh configs/srcnn/srcnn_x4k915_1xb16-1000k_div2k.py 8
```

更多细节可以参考 train_test.md 中的 **Train a model** 部分。

测试

您可以使用以下命令来测试模型。

```
## CPU上测试
CUDA_VISIBLE_DEVICES=-1 python tools/test.py configs/srcnn/srcnn_x4k915_1xb16-1000k_
↪div2k.py https://download.openmmlab.com/mmediting/restorers/srcnn/srcnn_x4k915_1x16_
↪1000k_div2k_20200608-4186f232.pth

## 单个GPU上测试
```

(下页继续)

(续上页)

```
python tools/test.py configs/srcnn/srcnn_x4k915_1xb16-1000k_div2k.py https://download.
↪openmmlab.com/mmediting/restorers/srcnn/srcnn_x4k915_1x16_1000k_div2k_20200608-
↪4186f232.pth

## 多个GPU上测试
./tools/dist_test.sh configs/srcnn/srcnn_x4k915_1xb16-1000k_div2k.py https://download.
↪openmmlab.com/mmediting/restorers/srcnn/srcnn_x4k915_1x16_1000k_div2k_20200608-
↪4186f232.pth 8
```

更多细节可以参考 train_test.md 中的 **Test a pre-trained model** 部分。

CHAPTER 38

图像抠图

38.1 概览

- 预训练权重个数: 9
- 配置文件个数: 0
- 论文个数: 3
  - ALGORITHM: 3

38.2 GCA (AAAI’2020)

任务: 图像抠图

```
@inproceedings{li2020natural,
  title={Natural Image Matting via Guided Contextual Attention},
  author={Li, Yaoyi and Lu, Hongtao},
  booktitle={Association for the Advancement of Artificial Intelligence (AAAI)},
  year={2020}
}
```

其他结果

38.2.1 快速开始

训练

您可以使用以下命令来训练模型。

```
## CPU上训练
CUDA_VISIBLE_DEVICES=-1 python tools/train.py configs/gca/gca_r34_4xb10-200k_comp1k.py

## 单个GPU上训练
python tools/train.py configs/gca/gca_r34_4xb10-200k_comp1k.py

## 多个GPU上训练
./tools/dist_train.sh configs/gca/gca_r34_4xb10-200k_comp1k.py 8
```

更多细节可以参考 train_test.md 中的 **Train a model** 部分。

测试

您可以使用以下命令来测试模型。

```
## CPU上测试
CUDA_VISIBLE_DEVICES=-1 python tools/test.py configs/gca/gca_r34_4xb10-200k_comp1k.py␣
↪https://download.openmmlab.com/mmediting/mattors/gca/gca_r34_4x10_200k_comp1k_SAD-
↪33.38_20220615-65595f39.pth

## 单个GPU上测试
python tools/test.py configs/gca/gca_r34_4xb10-200k_comp1k.py https://download.
↪openmmlab.com/mmediting/mattors/gca/gca_r34_4x10_200k_comp1k_SAD-33.38_20220615-
↪65595f39.pth

## 多个GPU上测试
./tools/dist_test.sh configs/gca/gca_r34_4xb10-200k_comp1k.py https://download.
↪openmmlab.com/mmediting/mattors/gca/gca_r34_4x10_200k_comp1k_SAD-33.38_20220615-
↪65595f39.pth 8
```

更多细节可以参考 train_test.md 中的 **Test a pre-trained model** 部分。

38.3 IndexNet (ICCV’2019)

任务: 图像抠图

```
@inproceedings{hao2019indexnet,
  title={Indices Matter: Learning to Index for Deep Image Matting},
  author={Lu, Hao and Dai, Yutong and Shen, Chunhua and Xu, Songcen},
```

(下页继续)

(续上页)

```
  booktitle={Proc. IEEE/CVF International Conference on Computer Vision (ICCV)},
  year={2019}
}
```

The performance of training (best performance) with different random seeds diverges in a large range. You may need to run several experiments for each setting to obtain the above performance.

其他结果

38.3.1 快速开始

训练

您可以使用以下命令来训练模型。

```
## CPU上训练
CUDA_VISIBLE_DEVICES=-1 python tools/train.py configs/indexnet/indexnet_mobv2_1xb16-
↪78k_comp1k.py

## 单个GPU上训练
python tools/train.py configs/indexnet/indexnet_mobv2_1xb16-78k_comp1k.py

## 多个GPU上训练
./tools/dist_train.sh configs/indexnet/indexnet_mobv2_1xb16-78k_comp1k.py 8
```

更多细节可以参考 train_test.md 中的 **Train a model** 部分。

测试

您可以使用以下命令来测试模型。

```
## CPU上测试
CUDA_VISIBLE_DEVICES=-1 python tools/test.py configs/indexnet/indexnet_mobv2_1xb16-
↪78k_comp1k.py https://download.openmmlab.com/mmediting/mattors/indexnet/indexnet_
↪mobv2_1x16_78k_comp1k_SAD-45.6_20200618_173817-26dd258d.pth

## 单个GPU上测试
python tools/test.py configs/indexnet/indexnet_mobv2_1xb16-78k_comp1k.py https://
↪download.openmmlab.com/mmediting/mattors/indexnet/indexnet_mobv2_1x16_78k_comp1k_
↪SAD-45.6_20200618_173817-26dd258d.pth

## 多个GPU上测试
./tools/dist_test.sh configs/indexnet/indexnet_mobv2_1xb16-78k_comp1k.py https://
↪download.openmmlab.com/mmediting/mattors/indexnet/indexnet_mobv2_1x16_78k_comp1k_
↪SAD-45.6_20200618_173817-26dd258d.pth 8
```

更多细节可以参考 train_test.md 中的 **Test a pre-trained model** 部分。

38.4 DIM (CVPR'2017)

> **任务**: 图像抠图

```
@inproceedings{xu2017deep,
  title={Deep image matting},
  author={Xu, Ning and Price, Brian and Cohen, Scott and Huang, Thomas},
  booktitle={Proceedings of the IEEE Conference on Computer Vision and Pattern␣
↪Recognition},
  pages={2970--2979},
  year={2017}
}
```

注

- 第一阶段：训练不带精炼器的编码器-解码器部分。\
- 第二阶段：固定编码器-解码器部分，训练精炼器部分。\
- 第三阶段：微调整个网络模型。

  模型在训练过程中的性能不稳定。因此，展示的性能并非来自最后一个模型权重文件，而是训练期间在验证集上取得的最佳性能。

  不同随机种子的训练性能（最佳性能）的发散程度很大，您可能需要为每个设置运行多个实验以获得上述性能。

38.4.1 快速开始

训练

您可以使用以下命令来训练模型。

DIM 训练有三个阶段。

阶段 1: 训练不带精炼器的编码器-解码器部分。

```
## CPU上训练
CUDA_VISIBLE_DEVICES=-1 python tools/train.py configs/dim/dim_stage1-v16_1xb1-1000k_
↪comp1k.py

## 单个GPU上训练
python tools/train.py configs/dim/dim_stage1-v16_1xb1-1000k_comp1k.py
```

(下页继续)

(续上页)

```
## 多个GPU上训练
./tools/dist_train.sh configs/dim/dim_stage1-v16_1xb1-1000k_comp1k.py 8
```

阶段 2: 固定编码器-解码器部分，训练精炼器部分。

```
## CPU上训练
CUDA_VISIBLE_DEVICES=-1 python tools/train.py configs/dim/dim_stage2-v16-pln_1xb1-
↪1000k_comp1k.py

## 单个GPU上训练
python tools/train.py configs/dim/dim_stage2-v16-pln_1xb1-1000k_comp1k.py

## 多个GPU上训练
./tools/dist_train.sh configs/dim/dim_stage2-v16-pln_1xb1-1000k_comp1k.py 8
```

阶段 3: 微调整个网络模型。

```
## CPU上训练
CUDA_VISIBLE_DEVICES=-1 python tools/train.py configs/dim/dim_stage3-v16-pln_1xb1-
↪1000k_comp1k.py

## 单个GPU上训练
python tools/train.py configs/dim/dim_stage3-v16-pln_1xb1-1000k_comp1k.py

## 多个GPU上训练
./tools/dist_train.sh configs/dim/dim_stage3-v16-pln_1xb1-1000k_comp1k.py 8
```

更多细节可以参考 train_test.md 中的 **Train a model** 部分。

测试

您可以使用以下命令来测试模型。

```
## CPU上测试
CUDA_VISIBLE_DEVICES=-1 python tools/test.py onfigs/dim/dim_stage3-v16-pln_1xb1-1000k_
↪comp1k.py https://download.openmmlab.com/mmediting/mattors/dim/dim_stage3_v16_pln_
↪1x1_1000k_comp1k_SAD-50.6_20200609_111851-647f24b6.pth

## 单个GPU上测试
python tools/test.py onfigs/dim/dim_stage3-v16-pln_1xb1-1000k_comp1k.py https://
↪download.openmmlab.com/mmediting/mattors/dim/dim_stage3_v16_pln_1x1_1000k_comp1k_
↪SAD-50.6_20200609_111851-647f24b6.pth

## 多个GPU上测试
./tools/dist_test.sh onfigs/dim/dim_stage3-v16-pln_1xb1-1000k_comp1k.py https://
↪download.openmmlab.com/mmediting/mattors/dim/dim_stage3_v16_pln_1x1_1000k_comp1k_
↪SAD-50.6_20200609_111851-647f24b6.pth 8
```

(下页继续)

(续上页)

更多细节可以参考 train_test.md 中的 **Test a pre-trained model** 部分。

CHAPTER 39

概览

- *df2k_ost*
- *realsrset*
- *videolq*
- *glean*
- *celeba-hq*
- *gopro*
- *vimeo90k*
- *paired-pix2pix*
- *sidd*
- *dpdd*
- *udm10*
- *unpaired-cyclegan*
- *hide*
- *paris-street-view*
- *vid4*
- *ntire21_decompression*
- *deraining*

- *reds*
- *denoising*
- *comp1k*
- *realblur*
- *spmcs*
- *places365*
- *live1*
- *vimeo90k-triplet*
- *classic5*
- *div2k*

CHAPTER 40

准备 DF2K_OST 数据集

```
@inproceedings{wang2021real,
  title={Real-ESRGAN: Training Real-World Blind Super-Resolution with Pure Synthetic
↪Data},
  author={Wang, Xintao and Xie, Liangbin and Dong, Chao and Shan, Ying},
  booktitle={Proceedings of the IEEE/CVF International Conference on Computer Vision},
  pages={1905--1914},
  year={2021}
}
```

- DIV2K 数据集可以在 这里 下载 (我们只使用训练集)。
- Flickr2K 数据集可以在 这里 下载 (我们只使用训练集)。
- OST 数据集可以在 这里 下载 (我们只使用训练集)。

请先将所有图片放入 GT 文件夹（命名不需要按顺序）:

```
mmagic
├── mmagic
├── tools
├── configs
├── data
|   ├── df2k_ost
|   |   ├── GT
|   |   |   ├── 0001.png
|   |   |   ├── 0002.png
```

(下页继续)

(续上页)

```
|   |   |   ├── ...
...
```

40.1 裁剪子图像

为了更快的 IO，我们建议将图像裁剪为子图像。我们提供了这样一个脚本：

```
python tools/dataset_converters/df2k_ost/preprocess_df2k_ost_dataset.py --data-root ./
↪data/df2k_ost
```

生成的数据存放在 `df2k_ost` 下，数据结构如下，其中 `_sub` 表示子图像。

```
mmagic
├── mmagic
├── tools
├── configs
├── data
|   ├── df2k_ost
|   |   ├── GT
|   |   ├── GT_sub
|   |   ├── meta_info_df2k_ost.txt
...
```

40.2 准备标注列表文件

如果您想使用标注模式来处理数据集，需要先准备一个 `txt` 格式的标注文件。

标注文件中的每一行包含了图片名以及图片尺寸（这些通常是 ground-truth 图片），这两个字段用空格间隔开。

标注文件示例:

```
0001_s001.png (480,480,3)
0001_s002.png (480,480,3)
```

请注意，`preprocess_df2k_ost_dataset.py` 脚本默认生成一份标注文件。

40.3 Prepare LMDB dataset for DF2K_OST

如果你想使用 LMDB 数据集来获得更快的 IO 速度，你可以通过以下方式制作 LMDB 文件：

```
python tools/dataset_converters/df2k_ost/preprocess_df2k_ost_dataset.py --data-root ./
↪data/df2k_ost --make-lmdb
```

CHAPTER 41

准备 RealSRSet 数据集

```
@inproceedings{zhang2021designing,
    title={Designing a Practical Degradation Model for Deep Blind Image Super-
↪Resolution},
    author={Zhang, Kai and Liang, Jingyun and Van Gool, Luc and Timofte, Radu},
    booktitle={IEEE International Conference on Computer Vision},
    pages={4791--4800},
    year={2021}
}
```

数据集 RealSRSet 可以从 此处 下载。

数据集 RealSRSet+5images 可以从 此处 下载。

文件目录结构应如下所示：

```
mmagic
├── mmagic
├── tools
├── configs
├── data
|   ├── realsrset
|   ├── RealSRSet+5images
```

CHAPTER 42

准备 VideoLQ 数据集

```
@inproceedings{chan2022investigating,
  author = {Chan, Kelvin C.K. and Zhou, Shangchen and Xu, Xiangyu and Loy, Chen␣
↪Change},
  title = {Investigating Tradeoffs in Real-World Video Super-Resolution},
  booktitle = {IEEE Conference on Computer Vision and Pattern Recognition},
  year = {2022}
}
```

数据集可以从 Dropbox / Google Drive / OneDrive 下载。

文件目录结构应如下所示：

```
mmagic
├── mmagic
├── tools
├── configs
├── data
|   ├── VideoLQ
|   |   ├── 000
|   |   |   ├── 00000000.png
|   |   |   ├── 00000001.png
|   |   |   ├── ...
|   |   ├── 001
|   |   ├── 002
|   |   ├── ...
```

CHAPTER 43

准备 GLEAN 数据集

```
@InProceedings{chan2021glean,
  author = {Chan, Kelvin CK and Wang, Xintao and Xu, Xiangyu and Gu, Jinwei and Loy,␣
↪Chen Change},
  title = {GLEAN: Generative Latent Bank for Large-Factor Image Super-Resolution},
  booktitle = {Proceedings of the IEEE conference on computer vision and pattern␣
↪recognition},
  year = {2021}
}
```

43.1 准备 cat_train 数据集

1. 从LSUN 主页下载cat 数据集。
2. 从GLEAN 主页下载cat_train/meta_info_LSUNcat_GT.txt。
3. 导出图像并下采样

从 lmdb 文件中导出图像，并下采样到所需尺寸。为此，我们提供了一个脚本：

```
python tools/dataset_converters/glean/preprocess_cat_train_dataset.py --lmdb-path .
↪data/cat --meta-file-path ./data/cat_train/meta_info_LSUNcat_GT.txt --out-dir ./
↪data/cat_train
```

生成的数据存储在 `cat_train` 目录下，目录结构应如下所示：

```
mmagic
├── mmagic
├── tools
├── configs
├── data
|   ├── cat_train
|   |   ├── GT
|   |   ├── BIx8_down
|   |   ├── BIx16_down
|   |   ├── meta_info_LSUNcat_GT.txt
...
```

43.2 准备 cat_test 数据集

1. 从数据集主页下载CAT 数据集。
2. 从GLEAN 主页下载cat_test/meta_info_Cat100_GT.txt。
3. 下采样

将图像下采样到所需尺寸。为此，我们提供了一个脚本：

```
python tools/dataset_converters/glean/preprocess_cat_test_dataset.py --data-path ./
↪data/CAT_03 --meta-file-path ./data/cat_test/meta_info_Cat100_GT.txt --out-dir ./
↪data/cat_test
```

生成的数据存储在 cat_test 目录下，目录结构应如下所示：

```
mmagic
├── mmagic
├── tools
├── configs
├── data
|   ├── cat_test
|   |   ├── GT
|   |   ├── BIx8_down
|   |   ├── BIx16_down
|   |   ├── meta_info_Cat100_GT.txt
...
```

43.3 准备 FFHQ 数据集

1. 从数据集主页下载FFHQ 数据集 (images1024x1024)。

将文件目录重构为如下所示：

```
ffhq
├── images
|   ├── 00000.png
|   ├── 00001.png
|   ├── ...
|   ├── 69999.png
```

2. 从GLEAN 主页下载ffhq/meta_info_FFHQ_GT.txt。

3. 下采样

将图像下采样到所需尺寸。为此，我们提供了一个脚本：

```
python tools/dataset_converters/glean/preprocess_ffhq_celebahq_dataset.py --data-root␣
↪./data/ffhq/images
```

生成的数据存储在 `ffhq` 目录下，目录结构应如下所示：

```
mmagic
├── mmagic
├── tools
├── configs
├── data
|   ├── ffhq
|   |   ├── images
|   |   ├── BIx8_down
|   |   ├── BIx16_down
|   |   ├── meta_info_FFHQ_GT.txt
...
```

43.4 准备 CelebA-HQ 数据集

1. 根据数据集主页文档准备数据集。

将文件目录重构为如下所示：

```
CelebA-HQ
├── GT
|   ├── 00000.png
```

(下页继续)

(续上页)

```
|   ├── 00001.png
|   ├── ...
|   ├── 30000.png
```

2. 从GLEAN 主页下载CelebA-HQ/meta_info_CelebAHQ_val100_GT.txt。

3. 下采样

将图像下采样到所需尺寸。为此，我们提供了一个脚本：

```
python tools/dataset_converters/glean/preprocess_ffhq_celebahq_dataset.py --data-root␣
↪./data/CelebA-HQ/GT
```

生成的数据存储在 CelebA-HQ 目录下，目录结构应如下所示：

```
mmagic
├── mmagic
├── tools
├── configsdata
├── data
|   ├── CelebA-HQ
|   |   ├── GT
|   |   ├── BIx8_down
|   |   ├── BIx16_down
|   |   ├── meta_info_CelebAHQ_val100_GT.txt
...
```

43.5 准备 FFHQ_CelebAHQ 数据集

将 FFHQ(ffhq/images) 和 CelebA-HQ(CelebA-HQ/GT) 合并，生成 FFHQ_CelebAHQ 数据集。

文件目录重构应如下所示：

```
mmagic
├── mmagic
├── tools
├── configs
├── data
|   ├── FFHQ_CelebAHQ
|   |   ├── GT
...
```

CHAPTER 44

准备 CelebA-HQ 数据集

```
@article{karras2017progressive,
  title={Progressive growing of gans for improved quality, stability, and variation},
  author={Karras, Tero and Aila, Timo and Laine, Samuli and Lehtinen, Jaakko},
  journal={arXiv preprint arXiv:1710.10196},
  year={2017}
}
```

请按照此处准备数据集。

```
mmagic
├── mmagic
├── tools
├── configs
├── data
|   ├── CelebA-HQ
|   |   ├── train_256
|   |   ├── test_256
|   |   ├── train_celeba_img_list.txt
|   |   ├── val_celeba_img_list.txt
```

CHAPTER 45

准备 GoPro 数据集

```
@inproceedings{Zamir2021Restormer,
  title={Restormer: Efficient Transformer for High-Resolution Image Restoration},
  author={Syed Waqas Zamir and Aditya Arora and Salman Khan and Munawar Hayat and␣
↪Fahad Shahbaz Khan and Ming-Hsuan Yang},
  booktitle={CVPR},
  year={2022}
}
```

训练数据集可以从 此处 下载。测试数据集可以从 此处 下载。

文件目录结构应如下所示：

```
mmagic
├── mmagic
├── tools
├── configs
├── data
|   ├── GoPro
|   |   ├── train
|   |   |   ├── blur
|   |   |   ├── sharp
|   |   ├── test
|   |   |   ├── blur
|   |   |   ├── sharp
```

CHAPTER 46

准备 Vimeo90K 数据集

```
@article{xue2019video,
  title={Video Enhancement with Task-Oriented Flow},
  author={Xue, Tianfan and Chen, Baian and Wu, Jiajun and Wei, Donglai and Freeman,␣
↪William T},
  journal={International Journal of Computer Vision (IJCV)},
  volume={127},
  number={8},
  pages={1106--1125},
  year={2019},
  publisher={Springer}
}
```

训练集和测试集可以从 此处 下载。

将数据集路径 `vimeo_septuplet/sequences` 重命名为 `vimeo90k/GT`。Vimeo90K 数据集包含了如下所示的 `clip/sequence/img` 目录结构：

```
vimeo90k
├── GT
|   ├── 00001
|   |   ├── 0001
|   |   |   ├── im1.png
|   |   |   ├── im2.png
|   |   |   ├── ...
|   |   ├── 0002
```

(下页继续)

(续上页)

```
|   |   ├── 0003
|   |   ├── ...
|   ├── 00002
|   ├── ...
├── sep_trainlist.txt
├── sep_testlist.txt
```

为了生成下采样图像 BIx4 和 BDx4，以及准备所需的标注文件，需要执行如下命令：

```
python tools/dataset_converters/vimeo90k/preprocess_vimeo90k_dataset.py --data-root ./
↪data/vimeo90k
```

文件目录结构应如下所示：

```
mmagic
├── mmagic
├── tools
├── configs
├── data
|   ├── vimeo_triplet
|   |   ├── GT
|   |   |   ├── 00001
|   |   |   |   ├── 0001
|   |   |   |   |   ├── im1.png
|   |   |   |   |   ├── im2.png
|   |   |   |   |   ├── ...
|   |   |   |   ├── 0002
|   |   |   |   ├── 0003
|   |   |   |   ├── ...
|   |   |   ├── 00002
|   |   |   ├── ...
|   |   ├── BIx4
|   |   ├── BDx4
|   |   ├── meta_info_Vimeo90K_test_GT.txt
|   |   ├── meta_info_Vimeo90K_train_GT.txt
```

46.1 准备 LMDB 格式的 Vimeo90K 数据集

如果您想使用 LMDB 以获得更快的 IO 速度，可以通过以下脚本来构建 LMDB 文件

```
python tools/dataset_converters/vimeo90k/preprocess_vimeo90k_dataset.py --data-root ./
↪data/vimeo90k --train_list ./data/vimeo90k/sep_trainlist.txt --gt-path ./data/
↪vimeo90k/GT --lq-path ./data/Vimeo90k/BIx4  --make-lmdb
```

CHAPTER 47

为 Pix2pix 准备配对数据集

```
@inproceedings{isola2017image,
  title={Image-to-image translation with conditional adversarial networks},
  author={Isola, Phillip and Zhu, Jun-Yan and Zhou, Tinghui and Efros, Alexei A},
  booktitle={Proceedings of the IEEE conference on computer vision and pattern␣
↪recognition},
  pages={1125--1134},
  year={2017}
}
```

您可以从此处下载配对数据集。然后，您需要解压缩并移动相应的数据集以遵循如下所示的文件夹结构。数据集已经由原作者准备好了。

```
mmagic
├── mmagic
├── tools
├── configs
├── data
|   ├── paired
|   |   ├── facades
|   |   ├── maps
|   |   ├── edges2shoes
|   |   |   ├── train
|   |   |   ├── test
```

CHAPTER 48

准备 SIDD 数据集

```
@inproceedings{Zamir2021Restormer,
  title={Restormer: Efficient Transformer for High-Resolution Image Restoration},
  author={Syed Waqas Zamir and Aditya Arora and Salman Khan and Munawar Hayat and␣
↪Fahad Shahbaz Khan and Ming-Hsuan Yang},
  booktitle={CVPR},
  year={2022}
}
```

训练数据集可以从 此处 下载。验证数据集可以从 此处 下载。

验证数据集需要从 mat 文件中导出，为此，我们提供了一个脚本：

```
python tools/dataset_converters/sidd/preprocess_sidd_test_dataset.py --data-root ./
↪data/SIDD/val --out-dir ./data/SIDD/val
```

文件目录结构应如下所示：

```
mmagic
├── mmagic
├── tools
├── configs
├── data
|   ├── SIDD
|   |   ├── train
|   |   |   ├── gt
```

(下页继续)

(续上页)

```
|   |   |   ├── noisy
|   |   ├── val
|   |   |   ├── gt
|   |   |   ├── noisy
|   |   |   ├── ValidationNoisyBlocksSrgb.mat
|   |   |   ├── ValidationGtBlocksSrgb.mat
```

CHAPTER 49

准备 DPDD 数据集

```
@inproceedings{Zamir2021Restormer,
  title={Restormer: Efficient Transformer for High-Resolution Image Restoration},
  author={Syed Waqas Zamir and Aditya Arora and Salman Khan and Munawar Hayat and␣
↪Fahad Shahbaz Khan and Ming-Hsuan Yang},
  booktitle={CVPR},
  year={2022}
}
```

测试数据集可以从 此处 下载。

文件目录结构应如下所示：

```
mmagic
├── mmagic
├── tools
├── configs
├── data
|   ├── DPDD
|   |   ├── inputL
|   |   ├── inputR
|   |   ├── inputC
|   |   ├── target
|   |   ├── indoor_labels.npy
|   |   ├── indoor_labels.txt
```

(下页继续)

(续上页)

```
|   |   ├── outdoor_labels.npy
|   |   ├── outdoor_labels.txt
```

CHAPTER 50

准备 UDM10 数据集

```
@inproceedings{PFNL,
    title={Progressive Fusion Video Super-Resolution Network via Exploiting Non-Local␣
↪Spatio-Temporal Correlations},
    author={Yi, Peng and Wang, Zhongyuan and Jiang, Kui and Jiang, Junjun and Ma,␣
↪Jiayi},
    booktitle={IEEE International Conference on Computer Vision (ICCV)},
    pages={3106-3115},
    year={2019},
}
```

数据集可以从 此处 下载。

文件目录结构应如下所示：

```
mmagic
├── mmagic
├── tools
├── configs
├── data
|   ├── UDM10
|   |   ├── GT
|   |   ├── BIx4
|   |   ├── BDx4
```

CHAPTER 51

为 CycleGAN 准备未配对数据集

```
@inproceedings{zhu2017unpaired,
  title={Unpaired image-to-image translation using cycle-consistent adversarial␣
↪networks},
  author={Zhu, Jun-Yan and Park, Taesung and Isola, Phillip and Efros, Alexei A},
  booktitle={Proceedings of the IEEE international conference on computer vision},
  pages={2223--2232},
  year={2017}
}
```

您可以从此处下载未配对的数据集。然后，您需要解压缩并移动相应的数据集以遵循如上所示的文件夹结构。数据集已经由原作者准备好了。

```
mmagic
├── mmagic
├── tools
├── configs
├── data
|   ├── unpaired
|   |   ├── facades
|   |   ├── horse2zebra
|   |   ├── summer2winter_yosemite
|   |   |   ├── trainA
|   |   |   ├── trainB
|   |   |   ├── testA
```

(下页继续)

(续上页)

```
|   |   |       ├── testB
```

CHAPTER 52

准备 HIDE 数据集

```
@inproceedings{Zamir2021Restormer,
  title={Restormer: Efficient Transformer for High-Resolution Image Restoration},
  author={Syed Waqas Zamir and Aditya Arora and Salman Khan and Munawar Hayat and␣
↪Fahad Shahbaz Khan and Ming-Hsuan Yang},
  booktitle={CVPR},
  year={2022}
}
```

测试数据集可以从 此处 下载。

文件目录结构应如下所示：

```
mmagic
├── mmagic
├── tools
├── configs
├── data
|   ├── HIDE
|   |   ├── input
|   |   ├── target
```

CHAPTER 53

准备 Paris Street View 数据集

```
@inproceedings{pathak2016context,
  title={Context encoders: Feature learning by inpainting},
  author={Pathak, Deepak and Krahenbuhl, Philipp and Donahue, Jeff and Darrell,␣
↪Trevor and Efros, Alexei A},
  booktitle={Proceedings of the IEEE conference on computer vision and pattern␣
↪recognition},
  pages={2536--2544},
  year={2016}
}
```

请从此处获取数据集。

```
mmagic
├── mmagic
├── tools
├── configs
├── data
|   ├── paris_street_view
|   |   ├── train
|   |   ├── val
```

CHAPTER 54

准备 Vid4 数据集

```
@article{xue2019video,
  title={On Bayesian adaptive video super resolution},
  author={Liu, Ce and Sun, Deqing},
  journal={IEEE Transactions on Pattern Analysis and Machine Intelligence},
  volume={36},
  number={2},
  pages={346--360},
  year={2013},
  publisher={IEEE}
}
```

可以从 此处 下载 Vid4 数据集，其中包含了由两种下采样方法得到的图片：

1. BIx4 包含了由双线性插值下采样得到的图片
2. BDx4 包含了由 σ=1.6 的高斯核模糊，然后每 4 个像素进行一次采样得到的图片

请注意，应为 Vid4 数据集准备一个如下所列的标注文件（例如 meta_info_Vid4_GT.txt）。

```
calendar 41 (576,720,3)
city 34 (576,704,3)
foliage 49 (480,720,3)
walk 47 (480,720,3)
```

对于 ToFlow，应准备直接上采样的数据集，为此，我们提供了一个脚本：

```
python tools/dataset_converters/vid4/preprocess_vid4_dataset.py --data-root ./data/
↪Vid4/BIx4
```

文件目录结构应如下所示：

```
mmagic
├── mmagic
├── tools
├── configs
├── data
|   ├── Vid4
|   |   ├── GT
|   |   |   ├── calendar
|   |   |   ├── city
|   |   |   ├── foliage
|   |   |   ├── walk
|   |   ├── BDx4
|   |   ├── BIx4
|   |   ├── BIx4up_direct
|   |   ├── meta_info_Vid4_GT.txt
```

CHAPTER 55

准备 NTIRE21 decompression 数据集

```
@inproceedings{yang2021dataset,
  title={{NTIRE 2021} Challenge on Quality Enhancement of Compressed Video: Dataset␣
↪and Study},
  author={Ren Yang and Radu Timofte},
  booktitle={IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops}
↪,
  year={2021}
}
```

测试数据集可以从其主页下载。

请按照主页教程生成数据集。

文件目录结构应如下所示：

```
mmagic
├── mmagic
├── tools
├── configs
├── data
|   ├── NTIRE21_decompression_track1
|   |   ├── GT
|   |   |   ├── 001
|   |   |   |   ├── 001.png
|   |   |   |   ├── ...
```

(下页继续)

(续上页)

```
|   |   |   ├── ...
|   |   |   ├── 010
|   |   ├── LQ
|   |   |   ├── 001
|   |   |   |   ├── 001.png
|   |   |   |   ├── ...
|   |   |   ├── ...
|   |   |   ├── 010
|   ├── NTIRE21_decompression_track2
|   |   ├── GT
|   |   ├── LQ
|   ├── NTIRE21_decompression_track3
|   |   ├── GT
|   |   ├── LQ
```

CHAPTER 56

准备 Deraining 数据集

```
@inproceedings{Zamir2021Restormer,
  title={Restormer: Efficient Transformer for High-Resolution Image Restoration},
  author={Syed Waqas Zamir and Aditya Arora and Salman Khan and Munawar Hayat and␣
↪Fahad Shahbaz Khan and Ming-Hsuan Yang},
  booktitle={CVPR},
  year={2022}
}
```

测试数据集 (Rain100H, Rain100L, Test100, Test1200, Test2800) 可以从 此处 下载。

文件目录结构应如下所示：

```
mmagic
├── mmagic
├── tools
├── configs
├── data
|   ├── Rain100H
|   |   ├── input
|   |   ├── target
|   ├── Rain100L
|   |   ├── input
|   |   ├── target
|   ├── Test100
|   |   ├── input
```

(下页继续)

(续上页)

```
|   |   ├── target
|   ├── Test1200
|   |   ├── input
|   |   ├── target
|   ├── Test2800
|   |   ├── input
|   |   ├── target
```

CHAPTER 57

准备 REDS 数据集

```
@InProceedings{Nah_2019_CVPR_Workshops_REDS,
  author = {Nah, Seungjun and Baik, Sungyong and Hong, Seokil and Moon, Gyeongsik and␣
↪Son, Sanghyun and Timofte, Radu and Lee, Kyoung Mu},
  title = {NTIRE 2019 Challenge on Video Deblurring and Super-Resolution: Dataset and␣
↪Study},
  booktitle = {The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)␣
↪Workshops},
  month = {June},
  year = {2019}
}
```

- 训练集: REDS 数据集.
- 验证集: REDS 数据集 和 Vid4.

请注意，我们合并了 REDS 的训练集和验证集，以便在 REDS4 划分（在 EDVR 中会使用到）和官方验证集划分之间切换。

原始验证集的名称被修改了（clip 000 到 029），以避免与训练集发生冲突（总共 240 个 clip）。具体而言，验证集中的 clips 被改名为 240、241、…269。

可通过运行以下命令来准备 REDS 数据集:

```
python tools/dataset_converters/reds/preprocess_reds_dataset.py ./data/REDS
```

```
mmagic
├── mmagic
├── tools
├── configs
├── data
|   ├── REDS
|   |   ├── train_sharp
|   |   |   ├── 000
|   |   |   ├── 001
|   |   |   ├── ...
|   |   ├── train_sharp_bicubic
|   |   |   ├── X4
|   |   |   |   ├── 000
|   |   |   |   ├── 001
|   |   |   |   ├── ...
|   |   ├── meta_info_reds4_train.txt
|   |   ├── meta_info_reds4_val.txt
|   |   ├── meta_info_official_train.txt
|   |   ├── meta_info_official_val.txt
|   |   ├── meta_info_REDS_GT.txt
|   ├── REDS4
|   |   ├── GT
|   |   ├── sharp_bicubic
```

57.1 准备 LMDB 格式的 REDS 数据集

如果您想使用 LMDB 以获得更快的 IO 速度，可以通过以下脚本来构建 LMDB 文件：

```
python tools/dataset_converters/reds/preprocess_reds_dataset.py --root-path ./data/
↪REDS --make-lmdb
```

57.2 裁剪为子图

MMagic 支持将 REDS 图像裁剪为子图像以加快 IO。我们提供了这样一个脚本：

```
python tools/dataset_converters/reds/crop_sub_images.py --data-root ./data/REDS  -
↪scales 4
```

生成的数据存储在 REDS 下，数据结构如下，其中 _sub 表示子图像。

```
mmagic
├── mmagic
├── tools
├── configs
├── data
|   ├── REDS
|   |   ├── train_sharp
|   |   |   ├── 000
|   |   |   ├── 001
|   |   |   ├── ...
|   |   ├── train_sharp_sub
|   |   |   ├── 000_s001
|   |   |   ├── 000_s002
|   |   |   ├── ...
|   |   |   ├── 001_s001
|   |   |   ├── ...
|   |   ├── train_sharp_bicubic
|   |   |   ├── X4
|   |   |   |   ├── 000
|   |   |   |   ├── 001
|   |   |   |   ├── ...
|   |   |   ├── X4_sub
|   |   |   ├── 000_s001
|   |   |   ├── 000_s002
|   |   |   ├── ...
|   |   |   ├── 001_s001
|   |   |   ├── ...
```

请注意，默认情况下，`preprocess_reds_dataset.py` 不会为裁剪后的数据集制作 lmdb 和注释文件。您可能需要为此类操作稍微修改脚本。

CHAPTER 58

准备 Denoising 数据集

```
@inproceedings{Zamir2021Restormer,
  title={Restormer: Efficient Transformer for High-Resolution Image Restoration},
  author={Syed Waqas Zamir and Aditya Arora and Salman Khan and Munawar Hayat and␣
↪Fahad Shahbaz Khan and Ming-Hsuan Yang},
  booktitle={CVPR},
  year={2022}
}
```

测试数据集 (Set12, BSD68, CBSD68, Kodak, McMaster, Urban100) 可以从 此处 下载。

文件目录结构应如下所示：

```
mmagic
├── mmagic
├── tools
├── configs
├── data
|   ├── denoising_gaussian_test
|   |   ├── Set12
|   |   ├── BSD68
|   |   ├── CBSD68
|   |   ├── Kodak
|   |   ├── McMaster
|   |   ├── Urban100
```

CHAPTER 59

准备 Composition-1k 数据集

59.1 介绍

```
@inproceedings{xu2017deep,
  title={Deep image matting},
  author={Xu, Ning and Price, Brian and Cohen, Scott and Huang, Thomas},
  booktitle={Proceedings of the IEEE conference on computer vision and pattern␣
↪recognition},
  pages={2970--2979},
  year={2017}
}
```

Adobe Composition-1k 数据集由前景图像及其相应的 alpha 图像组成。要获得完整的数据集，需要将前景与来自 COCO 数据集和 Pascal VOC 数据集的选定背景进行合成。

59.2 获取和提取

请按照 论文作者 的说明获取 Composition-1k (comp1k) 数据集。

59.3 合成完整数据集

Adobe composition-1k 数据集仅包含 alpha 和 fg（以及测试集中的 trimap）。在训练或评估之前，需要将 fg 与 COCO 数据（训练）或 VOC 数据（测试）合并。使用以下脚本执行图像合成并生成用于训练或测试的注释文件：

```
# 在 MMagic 的根文件夹下运行脚本
python tools/dataset_converters/matting/comp1k/preprocess_comp1k_dataset.py data/
↪adobe_composition-1k data/coco data/VOCdevkit --composite
```

生成的数据分别存储在 “adobe_composition-1k/Training_set” 和 “adobe_composition-1k/Test_set” 下。如果你只想合成测试数据（因为合成训练数据很耗时），你可以通过删除 --composite 选项来跳过合成训练集：

```
# 跳过合成训练集
python tools/dataset_converters/matting/comp1k/preprocess_comp1k_dataset.py data/
↪adobe_composition-1k data/coco data/VOCdevkit
```

如果您只想预处理测试数据，即对于 FBA，您可以通过添加 --skip-train 选项来跳过训练集：

```
# 跳过预处理训练集
python tools/dataset_converters/matting/comp1k/preprocess_comp1k_dataset.py data/
↪adobe_composition-1k data/coco data/VOCdevkit --skip-train
```

> 目前，GCA 和 FBA 支持在线合成训练数据。但是你可以修改其他模型的数据管道来执行在线合成，而不是加载合成图像（我们在数据管道中称之为 “合并”）。

59.4 检查 DIM 的目录结构

最终的文件夹结构应如下所示：

```
mmagic
├── mmagic
├── tools
├── configs
├── data
|   ├── adobe_composition-1k
|   |   ├── Test_set
|   |   |   ├── Adobe-licensed images
|   |   |   |   ├── alpha
|   |   |   |   ├── fg
|   |   |   |   ├── trimaps
|   |   |   ├── merged  (generated by tools/dataset_converters/matting/comp1k/
↪preprocess_comp1k_dataset.py)
```

(下页继续)

(续上页)

```
|   |   |   ├── bg      (generated by tools/dataset_converters/matting/comp1k/
↪preprocess_comp1k_dataset.py)
|   |   ├── Training_set
|   |   |   ├── Adobe-licensed images
|   |   |   |   ├── alpha
|   |   |   |   ├── fg
|   |   |   ├── Other
|   |   |   |   ├── alpha
|   |   |   |   ├── fg
|   |   |   ├── merged  (generated by tools/dataset_converters/matting/comp1k/
↪preprocess_comp1k_dataset.py)
|   |   |   ├── bg      (generated by tools/dataset_converters/matting/comp1k/
↪preprocess_comp1k_dataset.py)
|   |   ├── test_list.json     (generated by tools/dataset_converters/matting/comp1k/
↪preprocess_comp1k_dataset.py)
|   |   ├── training_list.json (generated by tools/dataset_converters/matting/comp1k/
↪preprocess_comp1k_dataset.py)
|   ├── coco
|   |   ├── train2014   (or train2017)
|   ├── VOCdevkit
|   |   ├── VOC2012
```

59.5 为 FBA 准备数据集

FBA 采用 Learning-base Sampling for Natural Image Matting 中提出的动态数据集增强。此外，为了减少增强过程中的伪影，它使用前景的扩展版本作为前景。我们提供脚本来估计前景。

准备测试集：

```
# 跳过预处理训练集，因为它在训练期间在线合成
python tools/dataset_converters/matting/comp1k/preprocess_comp1k_dataset.py data/
↪adobe_composition-1k data/coco data/VOCdevkit --skip-train
```

扩展训练集的前景：

```
python tools/dataset_converters/matting/comp1k/extend_fg.py data/adobe_composition-1k
```

59.6 检查 DIM 的目录结构

最终的文件夹结构应如下所示：

```
mmagic
├── mmagic
├── tools
├── configs
├── data
│   ├── adobe_composition-1k
│   │   ├── Test_set
│   │   │   ├── Adobe-licensed images
│   │   │   │   ├── alpha
│   │   │   │   ├── fg
│   │   │   │   ├── trimaps
│   │   │   ├── merged  (generated by tools/data/matting/comp1k/preprocess_comp1k_
↪dataset.py)
│   │   │   ├── bg      (generated by tools/data/matting/comp1k/preprocess_comp1k_
↪dataset.py)
│   │   ├── Training_set
│   │   │   ├── Adobe-licensed images
│   │   │   │   ├── alpha
│   │   │   │   ├── fg
│   │   │   │   ├── fg_extended (generated by tools/data/matting/comp1k/extend_fg.py)
│   │   │   ├── Other
│   │   │   │   ├── alpha
│   │   │   │   ├── fg
│   │   │   │   ├── fg_extended (generated by tools/data/matting/comp1k/extend_fg.py)
│   │   ├── test_list.json         (generated by tools/data/matting/comp1k/preprocess_
↪comp1k_dataset.py)
│   │   ├── training_list_fba.json (generated by tools/data/matting/comp1k/extend_fg.
↪py)
│   ├── coco
│   │   ├── train2014   (or train2017)
│   ├── VOCdevkit
│   │   ├── VOC2012
```

CHAPTER 60

准备 RealBlur 数据集

```
@inproceedings{Zamir2021Restormer,
  title={Restormer: Efficient Transformer for High-Resolution Image Restoration},
  author={Syed Waqas Zamir and Aditya Arora and Salman Khan and Munawar Hayat and␣
↪Fahad Shahbaz Khan and Ming-Hsuan Yang},
  booktitle={CVPR},
  year={2022}
}
```

测试数据集 RealBlurR 可以从 此处 下载。测试数据集 RealBlurJ 可以从 此处 下载。

文件目录结构应如下所示：

```
mmagic
├── mmagic
├── tools
├── configs
├── data
|   ├── RealBlur_R
|   |   ├── input
|   |   ├── target
|   ├── RealBlur_J
|   |   ├── input
|   |   ├── target
```

CHAPTER 61

准备 SPMCS 数据集

```
@InProceedings{tao2017spmc,
  author={Xin Tao and Hongyun Gao and Renjie Liao and Jue Wang and Jiaya Jia},
  title = {Detail-Revealing Deep Video Super-Resolution},
  booktitle = {The IEEE International Conference on Computer Vision (ICCV)},
  month = {Oct},
  year = {2017}
}
```

数据集可以从 此处 下载。

文件目录结构应如下所示：

```
mmagic
├── mmagic
├── tools
├── configs
├── data
|   ├── SPMCS
|   |   ├── GT
|   |   ├── BIx4
|   |   ├── BDx4
|   |   ├── meta_info_SPMCS_GT.txt
```

CHAPTER 62

准备 Places365 数据集

```
@article{zhou2017places,
  title={Places: A 10 million Image Database for Scene Recognition},
  author={Zhou, Bolei and Lapedriza, Agata and Khosla, Aditya and Oliva, Aude and
↪Torralba, Antonio},
  journal={IEEE Transactions on Pattern Analysis and Machine Intelligence},
  year={2017},
  publisher={IEEE}
}
```

请从 Places365 下载并准备数据。

```
mmagic
├── mmagic
├── tools
├── configs
├── data
|   ├── Places
|   |   ├── data_large
|   |   ├── val_large
|   |   ├── meta
|   |   |   ├── places365_train_challenge.txt
|   |   |   ├── places365_val.txt
```

CHAPTER 63

准备 LIVE1 数据集

```
@article{zhang2017beyond,
  title={Beyond a {Gaussian} denoiser: Residual learning of deep {CNN} for image␣
↪denoising},
  author={Zhang, Kai and Zuo, Wangmeng and Chen, Yunjin and Meng, Deyu and Zhang, Lei}
↪,
  journal={IEEE Transactions on Image Processing},
  year={2017},
  volume={26},
  number={7},
  pages={3142-3155},
}
```

测试数据集可以从 此处 下载。

文件目录结构应如下所示：

```
mmagic
├── mmagic
├── tools
├── configs
├── data
|   ├── LIVE1
```

CHAPTER 64

准备 Vimeo90K-triplet 数据集

```
@article{xue2019video,
  title={Video Enhancement with Task-Oriented Flow},
  author={Xue, Tianfan and Chen, Baian and Wu, Jiajun and Wei, Donglai and Freeman,␣
↪William T},
  journal={International Journal of Computer Vision (IJCV)},
  volume={127},
  number={8},
  pages={1106--1125},
  year={2019},
  publisher={Springer}
}
```

训练集和测试集可以从 此处 下载。

Vimeo90K-triplet 数据集包含了如下所示的 `clip/sequence/img` 目录结构：

```
mmagic
├── mmagic
├── tools
├── configs
├── data
|   ├── vimeo_triplet
|   |   ├── tri_testlist.txt
|   |   ├── tri_trainlist.txt
|   |   ├── sequences
```

(下页继续)

(续上页)

```
|   |   |   ├── 00001
|   |   |   |   ├── 0001
|   |   |   |   |   ├── im1.png
|   |   |   |   |   ├── im2.png
|   |   |   |   |   └── im3.png
|   |   |   |   ├── 0002
|   |   |   |   ├── 0003
|   |   |   |   ├── ...
|   |   |   ├── 00002
|   |   |   ├── ...
```

CHAPTER 65

准备 Classic5 数据集

```
@article{zhang2017beyond,
  title={Beyond a {Gaussian} denoiser: Residual learning of deep {CNN} for image␣
↪denoising},
  author={Zhang, Kai and Zuo, Wangmeng and Chen, Yunjin and Meng, Deyu and Zhang, Lei}
↪,
  journal={IEEE Transactions on Image Processing},
  year={2017},
  volume={26},
  number={7},
  pages={3142-3155},
}
```

测试数据集可以从 此处 下载。

文件目录结构应如下所示：

```
mmagic
├── mmagic
├── tools
├── configs
├── data
|   ├── Classic5
```

CHAPTER 66

准备 DIV2K 数据集

```
@InProceedings{Agustsson_2017_CVPR_Workshops,
    author = {Agustsson, Eirikur and Timofte, Radu},
    title = {NTIRE 2017 Challenge on Single Image Super-Resolution: Dataset and Study}
↪,
    booktitle = {The IEEE Conference on Computer Vision and Pattern Recognition␣
↪(CVPR) Workshops},
    month = {July},
    year = {2017}
}
```

- 训练集: DIV2K dataset.
- 验证集: Set5 和 Set14.

请注意，我们将原始的验证集（文件名 0801 到 0900）合并进了原始的训练集（文件名 0001 到 0800）。文件目录结构应如下所示：

```
mmagic
├── mmagic
├── tools
├── configs
├── data
│   ├── DIV2K
│   │   ├── DIV2K_train_HR
│   │   │   ├── 0001.png
```

(下页继续)

(续上页)

```
|   |   |   ├── 0002.png
|   |   |   ├── ...
|   |   |   ├── 0800.png
|   |   |   ├── 0801.png
|   |   |   ├── ...
|   |   |   ├── 0900.png
|   |   ├── DIV2K_train_LR_bicubic
|   |   |   ├── X2
|   |   |   ├── X3
|   |   |   ├── X4
|   |   ├── DIV2K_valid_HR
|   |   ├── DIV2K_valid_LR_bicubic
|   |   |   ├── X2
|   |   |   ├── X3
|   |   |   ├── X4
|   ├── Set5
|   |   ├── GTmod12
|   |   ├── LRbicx2
|   |   ├── LRbicx3
|   |   ├── LRbicx4
|   ├── Set14
|   |   ├── GTmod12
|   |   ├── LRbicx2
|   |   ├── LRbicx3
|   |   ├── LRbicx4
```

66.1 裁剪子图

为了加快 IO，建议将 DIV2K 中的图片裁剪为一系列子图，为此，我们提供了一个脚本：

```
python tools/dataset_converters/div2k/preprocess_div2k_dataset.py --data-root ./data/
↪DIV2K
```

生成的数据保存在 DIV2K 目录下，其文件结构如下所示，其中 _sub 表示子图:

```
mmagic
├── mmagic
├── tools
├── configs
├── data
|   ├── DIV2K
|   |   ├── DIV2K_train_HR
```

(下页继续)

(续上页)

```
|   |   ├── DIV2K_train_HR_sub
|   |   ├── DIV2K_train_LR_bicubic
|   |   |   ├── X2
|   |   |   ├── X3
|   |   |   ├── X4
|   |   |   ├── X2_sub
|   |   |   ├── X3_sub
|   |   |   ├── X4_sub
|   |   ├── DIV2K_valid_HR
|   |   ├── ...
|   |   ├── meta_info_DIV2K800sub_GT.txt
|   |   ├── meta_info_DIV2K100sub_GT.txt
...
```

66.2 准备标注列表文件

如果您想使用标注模式来处理数据集，需要先准备一个 `txt` 格式的标注文件。

标注文件中的每一行包含了图片名以及图片尺寸（这些通常是 ground-truth 图片），这两个字段用空格间隔开。

标注文件示例:

```
0001_s001.png (480,480,3)
0001_s002.png (480,480,3)
```

请注意，`preprocess_div2k_dataset` 脚本默认生成一份标注文件。

66.3 准备 LMDB 格式的 DIV2K 数据集

如果您想使用 `LMDB` 以获得更快的 IO 速度，可以通过以下脚本来构建 LMDB 文件

```
python tools/dataset_converters/div2k/preprocess_div2k_dataset.py --data-root ./data/
↪DIV2K --make-lmdb
```

CHAPTER 67

变更日志

67.1 v1.0.2 (24/08/2023)

亮点

1. 更详尽的文档

感谢社区的贡献者们帮助我们改进文档。我们已经改进了许多文档，包括中文和英文版本。更多详细信息请参考文档。

2. 新的算法

- 支持了 Prompt-to-prompt, DDIM Inversion 和 Null-text Inversion. 点击查看.

从左到右: origin image, DDIM inversion, Null-text inversion

Prompt-to-prompt 编辑

- 支持了 Textual Inversion. 点击查看.
- 支持了 Attention Injection 以便使用 controlnet 生成更稳定的视频. 点击查看.
- 支持了 Stable Diffusion Inpainting. 点击查看.

新功能和改进

- [增强] 支持在稳定扩散训练中的噪声偏移，由 @LeoXing1996 提交于 https://github.com/open-mmlab/mmagic/pull/1880 ↗
- [社区] 支持 Glide Upsampler，由 @Taited 提交于 https://github.com/open-mmlab/mmagic/pull/1663 ↗

- [增强] 支持 controlnet 推理器，由 @Z-Fran 提交于 https://github.com/open-mmlab/mmagic/pull/1891 ↗
- [功能] 支持 Albumentations 增强变换和流水线，由 @Z-Fran 提交于 https://github.com/open-mmlab/mmagic/pull/1894 ↗
- [功能] 为 unet 添加注意力注入，由 @liuwenran 提交于 https://github.com/open-mmlab/mmagic/pull/1895 ↗
- [增强] 更新基准测试脚本，由 @Z-Fran 提交于 https://github.com/open-mmlab/mmagic/pull/1907 ↗
- [增强] 更新 mmagic 文档，由 @crazysteeaam 提交于 https://github.com/open-mmlab/mmagic/pull/1920 ↗
- [增强] 支持 Prompt-to-prompt、ddim inversion 和 null-text inversion，由 @FerryHuang 提交于 https://github.com/open-mmlab/mmagic/pull/1908 ↗
- [CodeCamp2023-302] 支持 MMagic 可视化并编写用户指南，由 @aptsunny 提交于 https://github.com/open-mmlab/mmagic/pull/1939 ↗
- [功能] 支持 textual inversion，由 @LeoXing1996 提交于 https://github.com/open-mmlab/mmagic/pull/1822 ↗
- [增强] 对一些模型采用 BaseModule，由 @LeoXing1996 提交于 https://github.com/open-mmlab/mmagic/pull/1543 ↗
- [MMSIG] 支持 DeblurGANv2 推理，由 @xiaomile 提交于 https://github.com/open-mmlab/mmagic/pull/1955 ↗
- [CodeCamp2023-647] 添加 EG3D 的新配置，由 @RangeKing 提交于 https://github.com/open-mmlab/mmagic/pull/1985 ↗

Bug 修复

- 修复了 StableDiffusion 和 DreamBooth 训练中的 dtype 错误。by @LeoXing1996 in https://github.com/open-mmlab/mmagic/pull/1879
- 修复了 gui VideoSlider 的 bug。by @Z-Fran in https://github.com/open-mmlab/mmagic/pull/1885
- 修复了 init_model 和 glide demo 的问题。by @Z-Fran in https://github.com/open-mmlab/mmagic/pull/1888
- 修复了当 dim=3 时的 InstColorization bug。by @Z-Fran in https://github.com/open-mmlab/mmagic/pull/1901
- 修复了 sd 和 controlnet 的 fp16 bug。by @Z-Fran in https://github.com/open-mmlab/mmagic/pull/1914
- 修复了 controlnet 中的 num_images_per_prompt。by @LeoXing1996 in https://github.com/open-mmlab/mmagic/pull/1936
- 修正了 sd-inpainting 的 metafile 以修复推理器的初始化。by @LeoXing1996 in https://github.com/open-mmlab/mmagic/pull/1995

新贡献者

- @wyyang23 首次贡献于 https://github.com/open-mmlab/mmagic/pull/1886
- @yehuixie 首次贡献于 https://github.com/open-mmlab/mmagic/pull/1912
- @crazysteeaam 首次贡献于 https://github.com/open-mmlab/mmagic/pull/1920
- @BUPT-NingXinyu 首次贡献于 https://github.com/open-mmlab/mmagic/pull/1921

- @zhjunqin 首次贡献于 https://github.com/open-mmlab/mmagic/pull/1918
- @xuesheng1031 首次贡献于 https://github.com/open-mmlab/mmagic/pull/1923
- @wslgqq277g 首次贡献于 https://github.com/open-mmlab/mmagic/pull/1934
- @LYMDLUT 首次贡献于 https://github.com/open-mmlab/mmagic/pull/1933
- @RangeKing 首次贡献于 https://github.com/open-mmlab/mmagic/pull/1930
- @xin-li-67 首次贡献于 https://github.com/open-mmlab/mmagic/pull/1932
- @chg0901 首次贡献于 https://github.com/open-mmlab/mmagic/pull/1931
- @aptsunny 首次贡献于 https://github.com/open-mmlab/mmagic/pull/1939
- @YanxingLiu 首次贡献于 https://github.com/open-mmlab/mmagic/pull/1943
- @tackhwa 首次贡献于 https://github.com/open-mmlab/mmagic/pull/1937
- @Geo-Chou 首次贡献于 https://github.com/open-mmlab/mmagic/pull/1940
- @qsun1 首次贡献于 https://github.com/open-mmlab/mmagic/pull/1956
- @ththth888 首次贡献于 https://github.com/open-mmlab/mmagic/pull/1961
- @sijiua 首次贡献于 https://github.com/open-mmlab/mmagic/pull/1967
- @MING-ZCH 首次贡献于 https://github.com/open-mmlab/mmagic/pull/1982
- @AllYoung 首次贡献于 https://github.com/open-mmlab/mmagic/pull/1996

67.2 v1.0.1 (26/05/2023)

新功能和改进

- 支持 StableDiffusion tomesd 加速. #1801
- 支持所有 inpainting/matting/image restoration 模型的 inferencer. #1833, #1873
- 支持 animated drawings. #1837
- 支持 Style-Based Global Appearance Flow for Virtual Try-On at projects. #1786
- 支持 tokenizer wrapper 和 EmbeddingLayerWithFixe. #1846

Bug 修复

- 修复安装依赖. #1819
- 修复 inst-colorization PackInputs. #1828, #1827
- 修复 pip install 时 inferencer 无法使用的问题. #1875

67.3 v1.0.0 (25/04/2023)

我们正式发布 MMagic v1.0.0 版本，源自 MMEditing 和 MMGeneration。

自从 MMEditing 诞生以来，它一直是许多图像超分辨率、编辑和生成任务的首选算法库，帮助多个研究团队取得 10 余项国际顶级赛事的胜利，支撑了 100 多个 GitHub 生态项目。经过 OpenMMLab 2.0 框架的迭代更新以及与 MMGeneration 的合并，MMEditing 已经成为了一个支持基于 GAN 和 CNN 的底层视觉算法的强大工具。

而今天，MMEditing 将更加拥抱生成式 AI（Generative AI），正式更名为 **MMagic**（**M**ultimodal **A**dvanced, **G**enerative, and **I**ntelligent **C**reation），致力于打造更先进、更全面的 AIGC 开源算法库。

在 MMagic 中，我们已经支持了 53+ 模型，分布于 Stable Diffusion 的微调、图文生成、图像及视频修复、超分辨率、编辑和生成等多种任务。配合 MMEngine 出色的训练与实验管理支持，MMagic 将为广大研究者与 AIGC 爱好者们提供更加快捷灵活的实验支持，助力你的 AIGC 探索之旅。使用 MMagic，体验更多生成的魔力！让我们一起开启超越编辑的新纪元！More than Editing, Unlock the Magic!

主要更新

1. 新算法

我们支持了 4 个新任务以及 11 个新算法。

- Text2Image / Diffusion
  - ControlNet
  - DreamBooth
  - Stable Diffusion
  - Disco Diffusion
  - GLIDE
  - Guided Diffusion
- 3D-aware Generation

  - EG3D

- Image Restoration
  - NAFNet
  - Restormer
  - SwinIR
- Image Colorization
  - InstColorization

https://user-images.githubusercontent.com/49083766/233564593-7d3d48ed-e843-4432-b610-35e3d257765c.mp4

2. Magic Diffusion Model

针对 Diffusion Model，我们提供了以下“魔法”

- 支持基于 Stable Diffusion 与 Disco Diffusion 的图像生成.
- 支持 Dreambooth 以及 DreamBooth LoRA 等 Finetune 方法.

- 支持 ControlNet 进行可控性的文本到图像生成.
- 支持 xFormers 加速和优化策略，提高训练与推理效率.
- 支持基于 MultiFrame Render 的视频生成. MMagic 支持通过 ControlNet 与多帧渲染法实现长视频的生成。prompt keywords: a handsome man, silver hair, smiling, play basketball

  https://user-images.githubusercontent.com/12782558/227149757-fd054d32-554f-45d5-9f09-319184866d85.mp4

  prompt keywords: a girl, black hair, white pants, smiling, play basketball

  https://user-images.githubusercontent.com/49083766/233559964-bd5127bd-52f6-44b6-a089-9d7adfbc2430.mp4

  prompt keywords: a handsome man

https://user-images.githubusercontent.com/12782558/227152129-d70d5f76-a6fc-4d23-97d1-a94abd08f95a.mp4

- 支持通过 Wrapper 调用 Diffusers 的基础模型以及采样策略.
- SAM + MMagic = Generate Anything！当下流行的 SAM（Segment Anything Model）也可以为 MMagic 提供更多加持！想制作自己的动画，可以移步至 OpenMMLab PlayGround！

https://user-images.githubusercontent.com/49083766/233562228-f39fc675-326c-4ae8-986a-c942059effd0.mp4

3. 框架升级

为了提升你的“施法”效率，我们对“魔术回路”做了以下升级:

- 通过 OpenMMLab 2.0 框架的 MMEngine 和 MMCV，MMagic 将编辑框架分解为不同的组件，并且可以通过组合不同的模块轻松地构建自定义的编辑器模型。我们可以像搭建“乐高”一样定义训练流程，提供丰富的组件和策略。在 MMagic 中，你可以使用不同的 APIs 完全控制训练流程.
- 支持 33+ 算法 Pytorch 2.0 加速.
- 重构 DataSample，支持 batch 维度的组合与拆分.
- 重构 DataPreprocessor，并统一各种任务在训练与推理时的数据格式.
- 重构 MultiValLoop 与 MultiTestLoop，同时支持生成类型指标（e.g. FID）与重建类型指标（e.g. SSIM）的评测，同时支持一次性评测多个数据集
- 支持本地可视化以及使用 tensorboard 或 wandb 的可视化.

新功能和改进

- 支持 53+ 算法，232+ 配置，213+ 模型权重，26+ 损失函数，and 20+ 评价指标.
- 支持 controlnet 动画生成以及 Gradio gui. 点击查看.
- 支持 Inferencer 和 Demo，使用 High-level Inference APIs. 点击查看.
- 支持 Inpainting 推理的 Gradio gui. 点击查看.
- 支持可视化图像/视频质量比较工具. 点击查看.
- 开启 projects，助力社区更快向算法库中添加新算法. 点击查看.
- 完善数据集的预处理脚本和使用说明文档. 点击查看.

67.4 v1.0.0rc7 (07/04/2023)

主要更新

我们很高兴发布 MMEditing 1.0.0rc7 版本。此版本支持了 MMEditing 和 MMGeneration 的 51+ 模型，226+ configs 和 212+ checkpoints。以下是此次版本发布的重点新功能

- 支持了 DiffuserWrapper.

- 支持了 ControlNet 的推理与训练.
- 支持了 PyTorch 2.0.

新功能和改进

- 支持了 DiffuserWrapper. #1692
- 支持了 ControlNet 的推理与训练. #1744
- 支持了 PyTorch 2.0 (使用‘inductor’后端成功编译 33+ 模型) #1742.
- 支持了图像超分和视频超分的 inferencer. #1662, #1720
- 重构 get_flops 脚本. #1675
- 重构数据集的 dataset_converters 脚本和使用文档. #1690
- 迁移 stylegan 算子到 MMCV 中. #1383

Bug 修复

- 修复 disco inferencer. #1673
- 修复 nafnet optimizer 配置. #1716
- 修复 tof typo. #1711

贡献者

@LeoXing1996, @Z-Fran, @plyfager, @zengyh1900, @liuwenran, @ryanxingql, @HAOCHENYE, @VongolaWu

67.5 v1.0.0rc6 (02/03/2023)

主要更新

我们很高兴发布 MMEditing 1.0.0rc6 版本。此版本支持了 MMEditing 和 MMGeneration 的 50+ 模型，222+ configs 和 209+ checkpoints。以下是此次版本发布的重点新功能

- 支持了 Inpainting 任务推理的 Gradio gui.
- 支持了图像上色、图像翻译和 GAN 模型的 inferencer.

新功能和改进

- 重构了 FileIO. #1572
- 重构了 registry. #1621
- 重构了 Random degradations. #1583
- 重构了 DataSample, DataPreprocessor, Metric 和 Loop. #1656
- 使用 mmengine.basemodule 替换 nn.module. #1491
- 重构了算法库主页. #1609

- 支持了 Inpainting 任务推理的 Gradio gui. #1601
- 支持了图像上色的 inferencer. #1588
- 支持了图像翻译和所有 GAN 模型的 inferencer. #1650
- 支持了 GAN 模型的 inferencer. #1653, #1659
- 新增 Print config 工具. #1590
- 改进 type hints. #1604
- 更新 metrics 和 datasets 的中文文档. #1568, #1638
- 更新 BigGAN 和 Disco-Diffusion 的中文文档. #1620
- 更新 Guided-Diffusion 的 Evaluation 和 README. #1547

Bug 修复

- 修复 EMA `momentum`. #1581
- 修复 RandomNoise 的输出类型. #1585
- 修复 pytorch2onnx 工具. #1629
- 修复 API 文档. #1641, #1642
- 修复 RealESRGAN 加载 EMA 参数. #1647
- 修复 dataset_converters 脚本的 arg passing bug. #1648

贡献者

@plyfager, @LeoXing1996, @Z-Fran, @zengyh1900, @VongolaWu, @liuwenran, @austinmw, @dienachtderwelt, @liangzelong, @i-aki-y, @xiaomile, @Li-Qingyun, @vansin, @Luo-Yihang, @ydengbi, @ruoningYu, @triple-Mu

67.6 v1.0.0rc5 (04/01/2023)

主要更新

我们很高兴发布 MMEditing 1.0.0rc5 版本。此版本支持了 MMEditing 和 MMGeneration 的 49+ 模型，180+ configs 和 177+ checkpoints。以下是此次版本发布的重点新功能

- 支持了 Restormer 算法.
- 支持了 GLIDE 算法.
- 支持了 SwinIR 算法.
- 支持了 Stable Diffusion 算法.

新功能和改进

- 新增 Disco notebook. (#1507)

- 优化测试 requirements 和 CI. (#1514)
- 自动生成文档 summary 和 API docstring. (#1517)
- 开启 projects. (#1526)
- 支持 mscoco dataset. (#1520)
- 改进中文文档. (#1532)
- 添加 Type hints. (#1481)
- 更新模型权重下载链接. (#1554)
- 更新部署指南. (#1551)

Bug 修复

- 修复文档链接检查. (#1522)
- 修复 ssim bug. (#1515)
- 修复 realesrgan 的 `extract_gt_data`. (#1542)
- 修复算法索引. (#1559)
- F 修复 disco-diffusion 的 config 路径. (#1553)
- Fix text2image inferencer. (#1523)

贡献者

@plyfager, @LeoXing1996, @Z-Fran, @zengyh1900, @VongolaWu, @liuwenran, @AlexZou14, @lvhan028, @xiaomile, @ldr426, @austin273, @whu-lee, @willaty, @curiosity654, @Zdafeng, @Taited

67.7 v1.0.0rc4 (05/12/2022)

主要更新

我们很高兴发布 MMEditing 1.0.0rc4 版本。此版本支持了 MMEditing 和 MMGeneration 的 45+ 模型，176+ configs 和 175+ checkpoints。以下是此次版本发布的重点新功能

- 支持了 High-level APIs.
- 支持了 diffusion 算法.
- 支持了 Text2Image 任务.
- 支持了 3D-Aware Generation.

新功能和改进

- 支持和重构了 High-level APIs. (#1410)
- 支持了 disco-diffusion 文生图算法. (#1234, #1504)

- 支持了 EG3D 算法. (#1482, #1493, #1494, #1499)
- 支持了 NAFNet 算法. (#1369)

Bug 修复

- 修复 srgan 的训练配置. (#1441)
- 修复 cain 的 config. (#1404)
- 修复 rdn 和 srcnn 的训练配置. (#1392)

贡献者

@plyfager, @LeoXing1996, @Z-Fran, @zengyh1900, @VongolaWu, @gaoyang07, @ChangjianZhao, @zxczrx123, @jackghosts, @liuwenran, @CCODING04, @RoseZhao929, @shaocongliu, @liangzelong.

67.8 v1.0.0rc3 (10/11/2022)

主要更新

我们很高兴发布 MMEditing 1.0.0rc3 版本。此版本支持了 MMEditing 和 MMGeneration 的 43+ 模型，170+ configs 和 169+ checkpoints。以下是此次版本发布的重点新功能

- 将 `mmdet` 和 `clip` 改为可选安装需求.

新功能和改进

- 支持 `mmdet` 的 `try_import`. (#1408)
- 支持 `flip` 的 `try_import`. (#1420)
- 更新 `.gitignore`. ($1416)
- 设置 `inception_utils` 的 `real_feat` 为 cpu 变量. (#1415)
- 更新 StyleGAN2 和 PEGAN 的 README 和 configs. (#1418)
- 改进 API 文档的渲染. (#1373)

Bug 修复

- 修复 ESRGAN 的 config 和预训练模型加载. (#1407)
- 修复 LSGAN 的 config. (#1409)
- 修复 CAIN 的 config. (#1404)

贡献者

@Z-Fran, @zengyh1900, @plyfager, @LeoXing1996, @ruoningYu.

67.9 v1.0.0rc2 (02/11/2022)

主要更新

我们很高兴发布 MMEditing 1.0.0rc2 版本。此版本支持了 MMEditing 和 MMGeneration 的 43+ 模型，170+ configs 和 169+ checkpoints。以下是此次版本发布的重点新功能

- 基于 patch 和 slider 的图像和视频可视化质量比较工具.
- 支持了图像上色算法.

新功能和改进

- 支持了质量比较工具. (#1303)
- 支持了 instance aware colorization 上色算法. (#1370)
- 支持使用不同采样模型的 multi-metrics. (#1171)
- 改进代码实现
  - 重构 evaluation metrics. (#1164)
  - 在 PGGAN 的 `forward` 中保存 gt 图像. (#1332)
  - 改进 `preprocess_div2k_dataset.py` 脚本的默认参数. (#1380)
  - 支持在 visualizer 中裁剪像素值. (#1365)
  - 支持了 SinGAN 数据集和 SinGAN demo. (#1363)
  - 支持 GenDataPreprocessor 中返回 int 和 float 数据类型. (#1385)
- 改进文档
  - 更新菜单切换. (#1162)
  - 修复 TTSR README. (#1325)

Bug 修复

- 修复 PPL bug. (#1172)
- 修复 RDN `number of channels` 参数. (#1328)
- 修复 demo 的 exceptions 类型. (#1372)
- 修复 realesrgan ema. (#1341)
- 改进 assertion 检查 `GenerateFacialHeatmap` 的数据类型为 `np.float32`. (#1310)
- 修复 `unpaired_dataset.py` 的采样方式. (#1308)
- 修复视频超分模型的 pytorch2onnx 脚本. (#1300)
- 修复错误的 config 配置. (#1167,#1200,#1236,#1293,#1302,#1304,#1319,#1331,#1336,#1349,#1352,#1353,#1358,#1364,#1367,

贡献者

@LeoXing1996, @Z-Fran, @zengyh1900, @plyfager, @ryanxingql, @ruoningYu, @gaoyang07.

67.10 v1.0.0rc1(23/9/2022)

MMEditing 1.0.0rc1 已经合并了 MMGeneration 1.x。

- 支持 42+ 算法, 169+ 配置文件 and 168+ 预训练模型参数文件.
- 支持 26+ loss functions, 20+ metrics.
- 支持 tensorboard, wandb.
- 支持 unconditional GANs, conditional GANs, image2image translation 以及 internal learning.

67.11 v1.0.0rc0(31/8/2022)

MMEditing 1.0.0rc0 是 MMEditing 1.x 的第一个版本，是 OpenMMLab 2.0 项目的一部分。

基于新的训练引擎, MMEditing 1.x 统一了数据、模型、评测和可视化的接口。

该版本存在有一些 BC-breaking 的修改。请在迁移指南中查看更多细节。

CHAPTER 68

mmagic.apis.inferencers

68.1 Package Contents

68.1.1 Classes

ColorizationInferencer	inferencer that predicts with colorization models.
ConditionalInferencer	inferencer that predicts with conditional models.
ControlnetAnimationInferencer	Base inferencer.
EG3DInferencer	Base inferencer.
ImageSuperResolutionInferencer	inferencer that predicts with restoration models.
InpaintingInferencer	inferencer that predicts with inpainting models.
MattingInferencer	inferencer that predicts with matting models.
Text2ImageInferencer	inferencer that predicts with text2image models.
TranslationInferencer	inferencer that predicts with translation models.
UnconditionalInferencer	inferencer that predicts with unconditional models.
VideoInterpolationInferencer	inferencer that predicts with video interpolation models.
VideoRestorationInferencer	inferencer that predicts with video restoration models.

class mmagic.apis.inferencers.**ColorizationInferencer**(*config: Union[mmagic.utils.ConfigType, str]*, *ckpt: Optional[str]*, *device: Optional[str] = None*, *extra_parameters: Optional[Dict] = None*, *seed: int = 2022*, ***kwargs*)

Bases: mmagic.apis.inferencers.base_mmagic_inferencer.BaseMMagicInferencer

inferencer that predicts with colorization models.

func_kwargs

preprocess(*img: mmagic.apis.inferencers.base_mmagic_inferencer.InputsType*) → Dict

Process the inputs into a model-feedable format.

参数 **img** (*InputsType*) –Image to be translated by models.

返回 Results of preprocess.

返回类型 results(Dict)

forward(*inputs: mmagic.apis.inferencers.base_mmagic_inferencer.InputsType*) → mmagic.apis.inferencers.base_mmagic_inferencer.PredType

Forward the inputs to the model.

visualize(*preds: mmagic.apis.inferencers.base_mmagic_inferencer.PredType*, *result_out_dir: str = None*) → List[numpy.ndarray]

Visualize predictions.

参数

- **preds** (*List[Union[str, np.ndarray]]*) –Forward results by the inferencer.
- **data** (*List[Dict]*) –Not needed by this kind of inferencer.
- **result_out_dir** (*str*) –Output directory of image. Defaults to `''`.

返回 Result of visualize

返回类型 List[np.ndarray]

class mmagic.apis.inferencers.**ConditionalInferencer**(*config: Union[mmagic.utils.ConfigType, str]*, *ckpt: Optional[str]*, *device: Optional[str] = None*, *extra_parameters: Optional[Dict] = None*, *seed: int = 2022*, ***kwargs*)

Bases: mmagic.apis.inferencers.base_mmagic_inferencer.BaseMMagicInferencer

inferencer that predicts with conditional models.

func_kwargs

extra_parameters

preprocess(*label: mmagic.apis.inferencers.base_mmagic_inferencer.InputsType*) → Dict

Process the inputs into a model-feedable format.

参数 **label** (*InputsType*) –Input label for condition models.

返回 Results of preprocess.

返回类型 results(Dict)

forward(*inputs: mmagic.apis.inferencers.base_mmagic_inferencer.InputsType*) → mmagic.apis.inferencers.base_mmagic_inferencer.PredType

Forward the inputs to the model.

visualize(*preds: mmagic.apis.inferencers.base_mmagic_inferencer.PredType*, *result_out_dir: str = None*) → List[numpy.ndarray]

Visualize predictions.

参数

- **preds** (*List[Union[str, np.ndarray]]*) –Forward results by the inferencer.
- **data** (*List[Dict]*) –Not needed by this kind of inferencer.
- **result_out_dir** (*str*) –Output directory of image. Defaults to ''.

返回 Result of visualize

返回类型 List[np.ndarray]

_pred2dict(*data_sample:* mmagic.structures.DataSample) → Dict

Extract elements necessary to represent a prediction into a dictionary. It's better to contain only basic data elements such as strings and numbers in order to guarantee it's json-serializable.

参数 **data_sample** (*DataSample*) –The data sample to be converted.

返回 The output dictionary.

返回类型 dict

class mmagic.apis.inferencers.**ControlnetAnimationInferencer**(*config: Union[mmagic.utils.ConfigType, str]*, *device: Optional[str] = None*, *extra_parameters: Optional[Dict] = None*, *dtype=torch.float32*, ***kwargs*)

Bases: mmagic.apis.inferencers.base_mmagic_inferencer.BaseMMagicInferencer

Base inferencer.

参数

- **config** (*str or ConfigType*) –Model config or the path to it.
- **ckpt** (*str, optional*) –Path to the checkpoint.
- **device** (*str, optional*) –Device to run inference. If None, the best device will be automatically used.
- **result_out_dir** (*str*) –Output directory of images. Defaults to ''.

func_kwargs

func_order

extra_parameters

__call__(*prompt=None*, *video=None*, *negative_prompt=None*, *controlnet_conditioning_scale=0.7*, *image_width=512*, *image_height=512*, *save_path=None*, *strength=0.75*, *num_inference_steps=20*, *seed=1*, *output_fps=None*, *reference_img=None*, ***kwargs*) → Union[Dict, List[Dict]]

Call the inferencer.

参数 **kwargs** –Keyword arguments for the inferencer.

返回 Results of inference pipeline.

返回类型 Union[Dict, List[Dict]]

class mmagic.apis.inferencers.**EG3DInferencer**(*config: Union[mmagic.utils.ConfigType, str]*, *ckpt: Optional[str]*, *device: Optional[str] = None*, *extra_parameters: Optional[Dict] = None*, *seed: int = 2022*, ***kwargs*)

Bases: mmagic.apis.inferencers.base_mmagic_inferencer.BaseMMagicInferencer

Base inferencer.

参数

- **config** (*str or ConfigType*) –Model config or the path to it.
- **ckpt** (*str, optional*) –Path to the checkpoint.
- **device** (*str, optional*) –Device to run inference. If None, the best device will be automatically used.
- **result_out_dir** (*str*) –Output directory of images. Defaults to ''.

func_kwargs

extra_parameters

preprocess(*inputs: mmagic.apis.inferencers.base_mmagic_inferencer.InputsType = None*) → mmagic.utils.ForwardInputs

Process the inputs into a model-feedable format.

参数 **inputs** (*List[Union[str, np.ndarray]]*) –The conditional inputs for the inferencer. Defaults to None.

返回 The preprocessed inputs and data samples.

返回类型 ForwardInputs

forward(*inputs: mmagic.utils.ForwardInputs*, *interpolation: Optional[str] = 'both'*, *num_images: int = 100*) → Union[dict, List[dict]]

Forward the inputs to the model.

参数

- **inputs** (*ForwardInputs*) –Model inputs. If data sample (the second element of *inputs*) is not passed, will generate a sequence of images corresponding to passed *interpolation* mode.
- **interpolation** (*str*) –The interpolation mode. Supported choices are 'both', 'conditioning', and 'camera'. Defaults to 'both'.
- **num_images** (*int*) –The number of frames of interpolation. Defaults to 500.

返回

Output dict corresponds to the input condition or the list of output dict of each frame during the interpolation process.

返回类型 Union[dict, List[dict]]

visualize(*preds: Union[mmagic.apis.inferencers.base_mmagic_inferencer.PredType, List[mmagic.apis.inferencers.base_mmagic_inferencer.PredType]]*, *vis_mode: str = 'both'*, *save_img: bool = True*, *save_video: bool = True*, *img_suffix: str = '.png'*, *video_suffix: str = '.mp4'*, *result_out_dir: str = 'eg3d_output'*) → None

Visualize predictions.

参数

- **preds** (*Union[PredType, List[PredType]]*) –Prediction os model.
- **vis_mode** (*str, optional*) –Which output to visualize. Supported choices are 'both', 'depth', and 'img'. Defaults to 'all'.
- **save_img** (*bool, optional*) –Whether save images. Defaults to True.
- **save_video** (*bool, optional*) –Whether save videos. Defaults to True.
- **img_suffix** (*str, optional*) –The suffix of saved images. Defaults to '.png'.
- **video_suffix** (*str, optional*) –The suffix of saved videos. Defaults to '.mp4'.

- **result_out_dir** (*str, optional*) –The save director of image and videos. Defaults to 'eg3d_output'.

preprocess_img(*preds: List[dict]*) → torch.Tensor

Preprocess images in the predictions.

参数 preds (*List[dict]*) –List of prediction dict of each frame.

返回

Preprocessed image tensor shape like [num_frame * bz, 3, H, W].

返回类型 torch.Tensor

preprocess_depth(*preds: List[dict]*) → torch.Tensor

Preprocess depth in the predictions.

参数 preds (*List[dict]*) –List of prediction dict of each frame.

返回

Preprocessed depth tensor shape like [num_frame * bz, 3, H, W].

返回类型 torch.Tensor

postprocess(*preds: mmagic.apis.inferencers.base_mmagic_inferencer.PredType*, *imgs: Optional[List[numpy.ndarray]] = None*, *is_batch: bool = False*, *get_datasample: bool = False*) → Dict[str, torch.tensor]

Postprocess predictions.

参数

- **preds** (*List[Dict]*) –Predictions of the model.
- **imgs** (*Optional[np.ndarray]*) –Visualized predictions.
- **is_batch** (*bool*) –Whether the inputs are in a batch. Defaults to False.
- **get_datasample** (*bool*) –Whether to use Datasample to store inference results. If False, dict will be used.

返回 Inference results as a dict.

返回类型 Dict[str, torch.Tensor]

class mmagic.apis.inferencers.**ImageSuperResolutionInferencer**(*config: Union[mmagic.utils.ConfigType, str]*, *ckpt: Optional[str]*, *device: Optional[str] = None*, *extra_parameters: Optional[Dict] = None*, *seed: int = 2022*, ***kwargs*)

Bases: mmagic.apis.inferencers.base_mmagic_inferencer.BaseMMagicInferencer

inferencer that predicts with restoration models.

func_kwargs

preprocess(*img: mmagic.apis.inferencers.base_mmagic_inferencer.InputsType*, *ref: mmagic.apis.inferencers.base_mmagic_inferencer.InputsType = None*) → Dict

Process the inputs into a model-feedable format.

参数

- **img** (*InputsType*) –Image to be restored by models.
- **ref** (*InputsType*) –Reference image for restoration models. Defaults to None.

返回 Results of preprocess.

返回类型 data(Dict)

forward(*inputs: mmagic.apis.inferencers.base_mmagic_inferencer.InputsType*) → mmagic.apis.inferencers.base_mmagic_inferencer.PredType

Forward the inputs to the model.

visualize(*preds: mmagic.apis.inferencers.base_mmagic_inferencer.PredType*, *result_out_dir: str = None*) → List[numpy.ndarray]

Visualize predictions.

参数

- **preds** (*List[Union[str, np.ndarray]]*) –Forward results by the inferencer.
- **data** (*List[Dict]*) –Not needed by this kind of inferencer.
- **result_out_dir** (*str*) –Output directory of image. Defaults to ''.

返回 Result of visualize

返回类型 List[np.ndarray]

class mmagic.apis.inferencers.**InpaintingInferencer**(*config: Union[mmagic.utils.ConfigType, str]*, *ckpt: Optional[str]*, *device: Optional[str] = None*, *extra_parameters: Optional[Dict] = None*, *seed: int = 2022*, ***kwargs*)

Bases: mmagic.apis.inferencers.base_mmagic_inferencer.BaseMMagicInferencer

inferencer that predicts with inpainting models.

func_kwargs

_init_pipeline(*cfg*) → mmengine.dataset.Compose

Initialize the test pipeline.

preprocess(*img: mmagic.apis.inferencers.base_mmagic_inferencer.InputsType*, *mask: mmagic.apis.inferencers.base_mmagic_inferencer.InputsType*) → Dict

Process the inputs into a model-feedable format.

参数

- **img** (*InputsType*) –Image to be inpainted by models.
- **mask** (*InputsType*) –Image mask for inpainting models.

返回 Results of preprocess.

返回类型 results(Dict)

forward(*inputs: mmagic.apis.inferencers.base_mmagic_inferencer.InputsType*) → mmagic.apis.inferencers.base_mmagic_inferencer.PredType

Forward the inputs to the model.

visualize(*preds: mmagic.apis.inferencers.base_mmagic_inferencer.PredType*, *result_out_dir: str = None*) → List[numpy.ndarray]

Visualize predictions.

参数

- **preds** (*List[Union[str, np.ndarray]]*) –Forward results by the inferencer.
- **data** (*List[Dict]*) –Mask of input image.
- **result_out_dir** (*str*) –Output directory of image. Defaults to `''`.

返回 Result of visualize

返回类型 List[np.ndarray]

class mmagic.apis.inferencers.**MattingInferencer**(*config: Union[mmagic.utils.ConfigType, str]*, *ckpt: Optional[str]*, *device: Optional[str] = None*, *extra_parameters: Optional[Dict] = None*, *seed: int = 2022*, ***kwargs*)

Bases: mmagic.apis.inferencers.base_mmagic_inferencer.BaseMMagicInferencer

inferencer that predicts with matting models.

func_kwargs

preprocess(*img: mmagic.apis.inferencers.base_mmagic_inferencer.InputsType*, *trimap: mmagic.apis.inferencers.base_mmagic_inferencer.InputsType*) → Dict

Process the inputs into a model-feedable format.

参数

- **img** (*InputsType*) –Image to be processed by models.
- **mask** (*InputsType*) –Mask corresponding to the input image.

返回 Results of preprocess.

返回类型 results(Dict)

forward(*inputs: mmagic.apis.inferencers.base_mmagic_inferencer.InputsType*) → mmagic.apis.inferencers.base_mmagic_inferencer.PredType

Forward the inputs to the model.

visualize(*preds: mmagic.apis.inferencers.base_mmagic_inferencer.PredType*, *result_out_dir: str = None*) → List[numpy.ndarray]

Visualize predictions.

参数

- **preds** (*List[Union[str, np.ndarray]]*) –Forward results by the inferencer.
- **data** (*List[Dict]*) –Not needed by this kind of inferencer.
- **result_out_dir** (*str*) –Output directory of image. Defaults to ''.

返回 Result of visualize

返回类型 List[np.ndarray]

_pred2dict(*data_sample:* mmagic.structures.DataSample) → Dict

Extract elements necessary to represent a prediction into a dictionary. It's better to contain only basic data elements such as strings and numbers in order to guarantee it's json-serializable.

参数 **data_sample** (*DataSample*) –The data sample to be converted.

返回 The output dictionary.

返回类型 dict

class mmagic.apis.inferencers.**Text2ImageInferencer**(*config: Union[mmagic.utils.ConfigType, str]*, *ckpt: Optional[str]*, *device: Optional[str] = None*, *extra_parameters: Optional[Dict] = None*, *seed: int = 2022*, ***kwargs*)

Bases: mmagic.apis.inferencers.base_mmagic_inferencer.BaseMMagicInferencer

inferencer that predicts with text2image models.

func_kwargs

extra_parameters

preprocess(*text: mmagic.apis.inferencers.base_mmagic_inferencer.InputsType*, *control: str = None*) → Dict

Process the inputs into a model-feedable format.

参数 **text** (*InputsType*) –text input for text-to-image model.

返回 Results of preprocess.

返回类型 result(Dict)

forward(*inputs: mmagic.apis.inferencers.base_mmagic_inferencer.InputsType*) → mmagic.apis.inferencers.base_mmagic_inferencer.PredType

Forward the inputs to the model.

visualize(*preds: mmagic.apis.inferencers.base_mmagic_inferencer.PredType*, *result_out_dir: str = None*) → List[numpy.ndarray]

Visualize predictions.

参数

- **preds** (*List[Union[str, np.ndarray]]*) –Forward results by the inferencer.
- **result_out_dir** (*str*) –Output directory of image. Defaults to ''.

返回 Result of visualize

返回类型 List[np.ndarray]

class mmagic.apis.inferencers.**TranslationInferencer**(*config: Union[mmagic.utils.ConfigType, str]*, *ckpt: Optional[str]*, *device: Optional[str] = None*, *extra_parameters: Optional[Dict] = None*, *seed: int = 2022*, ***kwargs*)

Bases: mmagic.apis.inferencers.base_mmagic_inferencer.BaseMMagicInferencer

inferencer that predicts with translation models.

func_kwargs

preprocess(*img: mmagic.apis.inferencers.base_mmagic_inferencer.InputsType*) → Dict

Process the inputs into a model-feedable format.

参数 **img** (*InputsType*) –Image to be translated by models.

返回 Results of preprocess.

返回类型 results(Dict)

forward(*inputs: mmagic.apis.inferencers.base_mmagic_inferencer.InputsType*) → mmagic.apis.inferencers.base_mmagic_inferencer.PredType

Forward the inputs to the model.

visualize(*preds: mmagic.apis.inferencers.base_mmagic_inferencer.PredType*, *result_out_dir: str = None*) → List[numpy.ndarray]

Visualize predictions.

参数

- **preds** (*List[Union[str, np.ndarray]]*) –Forward results by the inferencer.
- **data** (*List[Dict]*) –Not needed by this kind of inferencer.
- **result_out_dir** (*str*) –Output directory of image. Defaults to ''.

返回 Result of visualize

返回类型 List[np.ndarray]

class mmagic.apis.inferencers.**UnconditionalInferencer**(*config: Union[mmagic.utils.ConfigType, str]*, *ckpt: Optional[str]*, *device: Optional[str] = None*, *extra_parameters: Optional[Dict] = None*, *seed: int = 2022*, ***kwargs*)

Bases: mmagic.apis.inferencers.base_mmagic_inferencer.BaseMMagicInferencer

inferencer that predicts with unconditional models.

func_kwargs

extra_parameters

preprocess() → Dict

Process the inputs into a model-feedable format.

返回 Results of preprocess.

返回类型 results(Dict)

forward(*inputs: mmagic.apis.inferencers.base_mmagic_inferencer.InputsType*) → mmagic.apis.inferencers.base_mmagic_inferencer.PredType

Forward the inputs to the model.

visualize(*preds: mmagic.apis.inferencers.base_mmagic_inferencer.PredType*, *result_out_dir: str = ''*) → List[numpy.ndarray]

Visualize predictions.

参数

- **preds** (*List[Union[str, np.ndarray]]*) –Forward results by the inferencer.
- **data** (*List[Dict]*) –Not needed by this kind of inferencer.
- **result_out_dir** (*str*) –Output directory of image. Defaults to ''.

返回 Result of visualize

返回类型 List[np.ndarray]

_pred2dict(*data_sample:* mmagic.structures.DataSample) → Dict

Extract elements necessary to represent a prediction into a dictionary. It's better to contain only basic data elements such as strings and numbers in order to guarantee it's json-serializable.

参数 **data_sample** (*DataSample*) –The data sample to be converted.

返回 The output dictionary.

返回类型 dict

class mmagic.apis.inferencers.**VideoInterpolationInferencer**(*config: Union[mmagic.utils.ConfigType, str]*, *ckpt: Optional[str]*, *device: Optional[str] = None*, *extra_parameters: Optional[Dict] = None*, *seed: int = 2022*, ***kwargs*)

Bases: mmagic.apis.inferencers.base_mmagic_inferencer.BaseMMagicInferencer

inferencer that predicts with video interpolation models.

func_kwargs

extra_parameters

preprocess(*video: mmagic.apis.inferencers.base_mmagic_inferencer.InputsType*) → Dict

Process the inputs into a model-feedable format.

参数 **video** (*InputsType*) –Video to be interpolated by models.

返回 Video to be interpolated by models.

返回类型 video(InputsType)

forward(*inputs: mmagic.apis.inferencers.base_mmagic_inferencer.InputsType*, *result_out_dir: mmagic.apis.inferencers.base_mmagic_inferencer.InputsType = ''*) → mmagic.apis.inferencers.base_mmagic_inferencer.PredType

Forward the inputs to the model.

参数

- **inputs** (*InputsType*) –Input video directory.
- **result_out_dir** (*str*) –Output directory of video. Defaults to ''.

返回 Result of forwarding

返回类型 PredType

visualize(*preds: mmagic.apis.inferencers.base_mmagic_inferencer.PredType*, *result_out_dir: str = ''*) → List[numpy.ndarray]

Visualize is not needed in this inferencer.

postprocess(*preds: mmagic.apis.inferencers.base_mmagic_inferencer.PredType*, *imgs: Optional[List[numpy.ndarray]] = None*) → Union[mmagic.apis.inferencers.base_mmagic_inferencer.ResType, Tuple[mmagic.apis.inferencers.base_mmagic_inferencer.ResType, numpy.ndarray]]

Postprocess is not needed in this inferencer.

class mmagic.apis.inferencers.**VideoRestorationInferencer**(*config: Union[mmagic.utils.ConfigType, str]*, *ckpt: Optional[str]*, *device: Optional[str] = None*, *extra_parameters: Optional[Dict] = None*, *seed: int = 2022*, ***kwargs*)

Bases: mmagic.apis.inferencers.base_mmagic_inferencer.BaseMMagicInferencer

inferencer that predicts with video restoration models.

func_kwargs

extra_parameters

preprocess(*video: mmagic.apis.inferencers.base_mmagic_inferencer.InputsType*) → Dict

Process the inputs into a model-feedable format.

参数 **video** (*InputsType*) –Video to be restored by models.

返回 Results of preprocess.

返回类型 results(InputsType)

forward(*inputs: mmagic.apis.inferencers.base_mmagic_inferencer.InputsType*) → mmagic.apis.inferencers.base_mmagic_inferencer.PredType

Forward the inputs to the model.

参数 **inputs** (*InputsType*) –Images array of input video.

返回 Results of forwarding

返回类型 PredType

visualize(*preds: mmagic.apis.inferencers.base_mmagic_inferencer.PredType*, *result_out_dir: str = ''*) → List[numpy.ndarray]

Visualize predictions.

参数

- **preds** (*List[Union[str, np.ndarray]]*) –Forward results by the inferencer.
- **data** (*List[Dict]*) –Not needed by this kind of inferencer.
- **result_out_dir** (*str*) –Output directory of image. Defaults to ''.

返回 Result of visualize

返回类型 List[np.ndarray]

postprocess(*preds: mmagic.apis.inferencers.base_mmagic_inferencer.PredType*, *imgs: Optional[List[numpy.ndarray]] = None*) → Union[mmagic.apis.inferencers.base_mmagic_inferencer.ResType, Tuple[mmagic.apis.inferencers.base_mmagic_inferencer.ResType, numpy.ndarray]]

Postprocess is not needed in this inferencer.

CHAPTER 69

mmagic.structures

69.1 Package Contents

69.1.1 Classes

DataSample	A data structure interface of MMagic. They are used as interfaces

class mmagic.structures.**DataSample**(*, *metainfo: Optional[dict] = None*, ***kwargs*)

Bases: mmengine.structures.BaseDataElement

A data structure interface of MMagic. They are used as interfaces between different components, e.g., model, visualizer, evaluator, etc. Typically, DataSample contains all the information and data from ground- truth and predictions.

***DataSample* inherits from *BaseDataElement*. See more details in:** https://mmengine.readthedocs.io/en/latest/advanced_tutorials/data_element.html Specifically, an instance of BaseDataElement consists of two components, - `metainfo`, which contains some meta information,

e.g., *img_shape*, *img_id*, *color_order*, etc.

- `data`, which contains the data used in the loop.

The attributes in `DataSample` are divided into several parts:

- `gt_img`: Ground truth image(s).
- `pred_img`: Image(s) of model predictions.
- `ref_img`: Reference image(s).
- `mask`: Mask in Inpainting.
- `trimap`: Trimap in Matting.
- `gt_alpha`: Ground truth alpha image in Matting.
- `pred_alpha`: Predicted alpha image in Matting.
- `gt_fg`: Ground truth foreground image in Matting.
- `pred_fg`: Predicted foreground image in Matting.
- `gt_bg`: Ground truth background image in Matting.
- `pred_bg`: Predicted background image in Matting.
- `gt_merged`: Ground truth merged image in Matting.

Examples:

```
 >>> import torch
 >>> import numpy as np
 >>> from mmagic.structures import DataSample
 >>> img_meta = dict(img_shape=(800, 1196, 3))
 >>> img = torch.rand((3, 800, 1196))
 >>> data_sample = DataSample(gt_img=img, metainfo=img_meta)
 >>> assert 'img_shape' in data_sample.metainfo_keys()
 >>> data_sample
<DataSample(

    META INFORMATION
    img_shape: (800, 1196, 3)

    DATA FIELDS
    gt_img: tensor(...)
) at 0x1f6a5a99a00>
```

We also support *stack* and *split* operation to handle a batch of data samples:

```
>>> import torch
>>> import numpy as np
>>> from mmagic.structures import DataSample
>>> img_meta1 = img_meta2 = dict(img_shape=(800, 1196, 3))
>>> img1 = torch.rand((3, 800, 1196))
>>> img2 = torch.rand((3, 800, 1196))
```

(下页继续)

(续上页)

```
>>> data_sample1 = DataSample(gt_img=img1, metainfo=img_meta1)
>>> data_sample2 = DataSample(gt_img=img2, metainfo=img_meta1)
```

```
>>> # stack them and then use as batched-tensor!
>>> data_sample = DataSample.stack([data_sample1, data_sample2])
>>> print(data_sample.gt_img.shape)
torch.Size([2, 3, 800, 1196])
>>> print(data_sample.metainfo)
{'img_shape': [(800, 1196, 3), (800, 1196, 3)]}
```

```
>>> # split them if you want
>>> data_sample1_, data_sample2_ = data_sample.split()
>>> assert (data_sample1_.gt_img == img1).all()
>>> assert (data_sample2_.gt_img == img2).all()
```

property gt_label

This the function to fetch gt label.

返回 gt label.

返回类型 LabelData

META_KEYS

DATA_KEYS

set_predefined_data(*data: dict*) → None

set or change pre-defined key-value pairs in `data_field` by parameter `data`.

参数 **data** (*dict*) –A dict contains annotations of image or model predictions.

set_tensor_data(*data: dict*) → None

convert input data to tensor, and then set or change key-value pairs in `data_field` by parameter `data`.

参数 **data** (*dict*) –A dict contains annotations of image or model predictions.

set_gt_label(*value: Union[numpy.ndarray, torch.Tensor, Sequence[numbers.Number], numbers.Number]*) → *DataSample*

Set label of `gt_label`.

gt_label()

Delete gt label.

classmethod stack(*data_samples: Sequence[*DataSample*]*) → *DataSample*

Stack a list of data samples to one. All tensor fields will be stacked at first dimension. Otherwise the values will be saved in a list.

参数 **data_samples** (*Sequence['DataSample']*) –A sequence of *DataSample* to stack.

返回 The stacked data sample.

返回类型 *DataSample*

split(*allow_nonseq_value: bool = False*) → Sequence[*DataSample*]

Split a sequence of data sample in the first dimension.

参数 **allow_nonseq_value** (*bool*) –Whether allow non-sequential data in split operation. If True, non-sequential data will be copied for all split data samples. Otherwise, an error will be raised. Defaults to False.

返回 The list of data samples after splitting.

返回类型 Sequence[*DataSample*]

__len__()

Get the length of the data sample.

CHAPTER 70

mmagic.datasets

70.1 Package Contents

70.1.1 Classes

BasicConditionalDataset	Custom dataset for conditional GAN. This class is based on the
BasicFramesDataset	BasicFramesDataset for open source projects in OpenMMLab/MMagic.
BasicImageDataset	BasicImageDataset for open source projects in OpenMMLab/MMagic.
CIFAR10	CIFAR10 Dataset.
AdobeComp1kDataset	Adobe composition-1k dataset.
ControlNetDataset	Demo dataset to test ControlNet. Modified from https://github.com/lllyas
DreamBoothDataset	Dataset for DreamBooth.
GrowScaleImgDataset	Grow Scale Unconditional Image Dataset.
ImageNet	ImageNet Dataset.
MSCoCoDataset	MSCoCo 2014 dataset.
PairedImageDataset	General paired image folder dataset for image generation.
SinGANDataset	SinGAN Dataset.
TextualInversionDataset	Dataset for DreamBooth.
UnpairedImageDataset	General unpaired image folder dataset for image generation.

class mmagic.datasets.**BasicConditionalDataset**(*ann_file: str = ''*, *metainfo: Optional[dict] = None*, *data_root: str = ''*, *data_prefix: Union[str, dict] = ''*, *extensions: Sequence[str] = ('.jpg', '.jpeg', '.png', '.ppm', '.bmp', '.pgm', '.tif')*, *lazy_init: bool = False*, *classes: Union[str, Sequence[str], None] = None*, ***kwargs*)

Bases: mmengine.dataset.BaseDataset

Custom dataset for conditional GAN. This class is based on the combination of *BaseDataset* (https://github.com/open- mmlab/mmclassification/blob/main/mmcls/datasets/base_dataset.py) # noqa and *CustomDataset* (https://github.com/open- mmlab/mmclassification/blob/main/mmcls/datasets/custom.py). # noqa.

The dataset supports two kinds of annotation format.

1. A annotation file read by line (e.g., txt) is provided, and each line indicates a sample:

   The sample files:

   ```
   data_prefix/
   ├── folder_1
   │   ├── xxx.png
   │   ├── xxy.png
   │   └── ...
   └── folder_2
       ├── 123.png
       ├── nsdf3.png
       └── ...
   ```

   The annotation file (the first column is the image path and the second column is the index of category):

   ```
   folder_1/xxx.png 0
   folder_1/xxy.png 1
   folder_2/123.png 5
   folder_2/nsdf3.png 3
   ...
   ```

   Please specify the name of categories by the argument `classes` or `metainfo`.

2. A dict-based annotation file (e.g., json) is provided, key and value indicate the path and label of the sample:

   The sample files:

   ```
   data_prefix/
   ├── folder_1
   │   ├── xxx.png
   │   ├── xxy.png
   │   └── ...
   ```

(下页继续)

(续上页)

```
└── folder_2
    ├── 123.png
    ├── nsdf3.png
    └── ...
```

The annotation file (the key is the image path and the value column is the label):

```
{
    "folder_1/xxx.png": [1, 2, 3, 4],
    "folder_1/xxy.png": [2, 4, 1, 0],
    "folder_2/123.png": [0, 9, 8, 1],
    "folder_2/nsdf3.png", [1, 0, 0, 2],
    ...
}
```

In this kind of annotation, labels can be any type and not restricted to an index.

3. The samples are arranged in the specific way:

```
data_prefix/
├── class_x
│   ├── xxx.png
│   ├── xxy.png
│   └── ...
│       └── xxz.png
└── class_y
    ├── 123.png
    ├── nsdf3.png
    ├── ...
    └── asd932_.png
```

If the `ann_file` is specified, the dataset will be generated by the first two ways, otherwise, try the third way.

参数

- **ann_file** (*str*) –Annotation file path. Defaults to ‘’.
- **metainfo** (*dict, optional*) –Meta information for dataset, such as class information. Defaults to None.
- **data_root** (*str*) –The root directory for `data_prefix` and `ann_file`. Defaults to ‘’.
- **data_prefix** (*str | dict*) –Prefix for the data. Defaults to ‘’.
- **extensions** (*Sequence[str]*) –A sequence of allowed extensions. Defaults to (‘.jpg’, ‘.jpeg’, ‘.png’, ‘.ppm’, ‘.bmp’, ‘.pgm’, ‘.tif’).

- **lazy_init** (*bool*) –Whether to load annotation during instantiation. In some cases, such as visualization, only the meta information of the dataset is needed, which is not necessary to load annotation file. `Basedataset` can skip load annotations to save time by set `lazy_init=False`. Defaults to False.
- ****kwargs** –Other keyword arguments in `BaseDataset`.

property img_prefix

The prefix of images.

property CLASSES

Return all categories names.

property class_to_idx

Map mapping class name to class index.

返回 mapping from class name to class index.

返回类型 dict

_find_samples(*file_backend*)

find samples from `data_prefix`.

load_data_list()

Load image paths and gt_labels.

is_valid_file(*filename: str*) → bool

Check if a file is a valid sample.

get_gt_labels()

Get all ground-truth labels (categories).

返回 categories for all images.

返回类型 np.ndarray

get_cat_ids(*idx: int*) → List[int]

Get category id by index.

参数 **idx** (*int*) –Index of data.

返回 Image category of specified index.

返回类型 cat_ids (List[int])

_compat_classes(*metainfo*, *classes*)

Merge the old style `classes` arguments to `metainfo`.

full_init()

Load annotation file and set `BaseDataset._fully_initialized` to True.

__repr__()

Print the basic information of the dataset.

返回 Formatted string.

返回类型 str

extra_repr() → List[str]

The extra repr information of the dataset.

class mmagic.datasets.**BasicFramesDataset**(*ann_file: str = ''*, *metainfo: Optional[dict] = None*, *data_root: Optional[str] = None*, *data_prefix: dict = dict(img='')*, *pipeline: List[Union[dict, Callable]] = []*, *test_mode: bool = False*, *filename_tmpl: dict = dict()*, *search_key: Optional[str] = None*, *backend_args: Optional[dict] = None*, *depth: int = 1*, *num_input_frames: Optional[int] = None*, *num_output_frames: Optional[int] = None*, *fixed_seq_len: Optional[int] = None*, *load_frames_list: dict = dict()*, ***kwargs*)

Bases: mmengine.dataset.BaseDataset

BasicFramesDataset for open source projects in OpenMMLab/MMagic.

This dataset is designed for low-level vision tasks with frames, such as video super-resolution and video frame interpolation.

The annotation file is optional.

If use annotation file, the annotation format can be shown as follows.

```
Case 1 (Vid4):

    calendar 41
    city 34
    foliage 49
    walk 47

Case 2 (REDS):

    000/00000000.png (720, 1280, 3)
    000/00000001.png (720, 1280, 3)

Case 3 (Vimeo90k):

    00001/0266 (256, 448, 3)
    00001/0268 (256, 448, 3)
```

参数

- **ann_file** (*str*) –Annotation file path. Defaults to ‘’.
- **metainfo** (*dict, optional*) –Meta information for dataset, such as class information. Defaults to None.
- **data_root** (*str, optional*) –The root directory for `data_prefix` and `ann_file`. Defaults to None.
- **data_prefix** (*dict, optional*) –Prefix for training data. Defaults to dict(img=’’, gt=’’).
- **pipeline** (*list, optional*) –Processing pipeline. Defaults to [].
- **test_mode** (*bool, optional*) –`test_mode=True` means in test phase. Defaults to False.
- **filename_tmpl** (*str*) –Template for each filename. Note that the template excludes the file extension. Default: ‘{}’.
- **search_key** (*str*) –The key used for searching the folder to get data_list. Default: ‘gt’.
- **backend_args** (*dict, optional*) –Arguments to instantiate the prefix of uri corresponding backend. Defaults to None.
- **depth** (*int*) –The depth of path. Default: 1
- **num_input_frames** (*None | int*) –Number of input frames. Default: None.
- **num_output_frames** (*None | int*) –Number of output frames. Default: None.
- **fixed_seq_len** (*None | int*) –The fixed sequence length. If None, BasicFramesDataset will obtain the length of each sequence. Default: None.
- **load_frames_list** (*dict*) –Load frames list for each key. Default: dict().

实际案例

Assume the file structure as the following:

mmagic (root) ├── mmagic ├── tools ├── configs ├── data │ ├── Vid4 │ │ ├── BIx4 │ │ │ ├── city │ │ │ │ ├── img1.png │ │ ├── GT │ │ │ ├── city │ │ │ │ ├── img1.png │ │ ├── meta_info_Vid4_GT.txt │ ├── places │ │ ├── sequences ||| ├── 00001 │ │ │ ├── 0389 │ │ │ │ ├── img1.png │ │ │ │ ├── img2.png │ │ │ │ ├── img3.png │ │ ├── tri_trainlist.txt

Case 1: Loading Vid4 dataset for training a VSR model.

```
dataset = BasicFramesDataset(
    ann_file='meta_info_Vid4_GT.txt',
```

(下页继续)

(续上页)

```
    metainfo=dict(dataset_type='vid4', task_name='vsr'),
    data_root='data/Vid4',
    data_prefix=dict(img='BIx4', gt='GT'),
    pipeline=[],
    depth=2,
    num_input_frames=5)
```

Case 2: Loading Vimeo90k dataset for training a VFI model.

```
dataset = BasicFramesDataset(
    ann_file='tri_trainlist.txt',
    metainfo=dict(dataset_type='vimeo90k', task_name='vfi'),
    data_root='data/vimeo-triplet',
    data_prefix=dict(img='sequences', gt='sequences'),
    pipeline=[],
    depth=2,
    load_frames_list=dict(
        img=['img1.png', 'img3.png'], gt=['img2.png']))
```

See more details in unittest

tests/test_datasets/test_base_frames_dataset.py TestFramesDatasets().test_version_1_method()

METAINFO

load_data_list() → List[dict]

Load data list from folder or annotation file.

返回 A list of annotation.

返回类型 list[dict]

_get_path_list()

Get list of paths from annotation file or folder of dataset.

返回 A list of paths.

返回类型 list[str]

_get_path_list_from_ann()

Get list of paths from annotation file.

返回 A list of paths.

返回类型 list[str]

_get_path_list_from_folder(*sub_folder=None*, *need_ext=True*, *depth=1*)

Get list of paths from folder.

参数

- **sub_folder** (*None | str*) –The path of sub_folder. Default: None.
- **need_ext** (*bool*) –Whether need ext. Default: True.
- **depth** (*int*) –Residual depth of path, recursively called to `depth == 1`. Default: 1

返回 A list of paths.

返回类型 list[str]

_set_seq_lens()

Get sequence lengths.

_get_frames_list(*key*, *folder*)

Obtain list of frames.

参数

- **key** (*str*) –The key of frames list, e.g. `img`, `gt`.
- **folder** (*str*) –Folder of frames.

返回 The paths list of frames.

返回类型 list[str]

class mmagic.datasets.**BasicImageDataset**(*ann_file: str = ''*, *metainfo: Optional[dict] = None*, *data_root: Optional[str] = None*, *data_prefix: dict = dict(img='')*, *pipeline: List[Union[dict, Callable]] = []*, *test_mode: bool = False*, *filename_tmpl: dict = dict()*, *search_key: Optional[str] = None*, *backend_args: Optional[dict] = None*, *img_suffix: Optional[Union[str, Tuple[str]]] = IMG_EXTENSIONS*, *recursive: bool = False*, ***kwards*)

Bases: `mmengine.dataset.BaseDataset`

BasicImageDataset for open source projects in OpenMMLab/MMagic.

This dataset is designed for low-level vision tasks with image, such as super-resolution and inpainting.

The annotation file is optional.

If use annotation file, the annotation format can be shown as follows.

```
Case 1 (CelebA-HQ):

    000001.png
    000002.png
```

(下页继续)

(续上页)

```
Case 2 (DIV2K):

    0001_s001.png (480,480,3)
    0001_s002.png (480,480,3)
    0001_s003.png (480,480,3)
    0002_s001.png (480,480,3)
    0002_s002.png (480,480,3)

Case 3 (Vimeo90k):

    00001/0266 (256, 448, 3)
    00001/0268 (256, 448, 3)
```

参数

- **ann_file** (*str*) –Annotation file path. Defaults to ‘’.
- **metainfo** (*dict, optional*) –Meta information for dataset, such as class information. Defaults to None.
- **data_root** (*str, optional*) –The root directory for `data_prefix` and `ann_file`. Defaults to None.
- **data_prefix** (*dict, optional*) –Prefix for training data. Defaults to dict(img=None, ann=None).
- **pipeline** (*list, optional*) –Processing pipeline. Defaults to [].
- **test_mode** (*bool, optional*) –`test_mode=True` means in test phase. Defaults to False.
- **filename_tmpl** (*dict*) –Template for each filename. Note that the template excludes the file extension. Default: dict().
- **search_key** (*str*) –The key used for searching the folder to get data_list. Default: ‘gt’.
- **backend_args** (*dict, optional*) –Arguments to instantiate the prefix of uri corresponding backend. Defaults to None.
- **suffix** (*str or tuple[str], optional*) –File suffix that we are interested in. Default: None.
- **recursive** (*bool*) –If set to True, recursively scan the directory. Default: False.

备注： Assume the file structure as the following:

```
mmagic (root)
├── mmagic
├── tools
├── configs
├── data
|   ├── DIV2K
|   |   ├── DIV2K_train_HR
|   |   |   ├── image.png
|   |   ├── DIV2K_train_LR_bicubic
|   |   |   ├── X2
|   |   |   ├── X3
|   |   |   ├── X4
|   |   |   |   ├── image_x4.png
|   |   ├── DIV2K_valid_HR
|   |   ├── DIV2K_valid_LR_bicubic
|   |   |   ├── X2
|   |   |   ├── X3
|   |   |   ├── X4
|   ├── places
|   |   ├── test_set
|   |   ├── train_set
|   |   ├── meta
|   |   |    ├── Places365_train.txt
|   |   |    ├── Places365_val.txt
```

实际案例

Case 1: Loading DIV2K dataset for training a SISR model.

```
dataset = BasicImageDataset(
    ann_file='',
    metainfo=dict(
        dataset_type='div2k',
        task_name='sisr'),
    data_root='data/DIV2K',
    data_prefix=dict(
        gt='DIV2K_train_HR', img='DIV2K_train_LR_bicubic/X4'),
    filename_tmpl=dict(img='{}_x4', gt='{}'),
    pipeline=[])
```

Case 2: Loading places dataset for training an inpainting model.

```
dataset = BasicImageDataset(
    ann_file='meta/Places365_train.txt',
    metainfo=dict(
        dataset_type='places365',
        task_name='inpainting'),
    data_root='data/places',
    data_prefix=dict(gt='train_set'),
    pipeline=[])
```

METAINFO

load_data_list() → List[dict]

Load data list from folder or annotation file.

返回 A list of annotation.

返回类型 list[dict]

_get_path_list()

Get list of paths from annotation file or folder of dataset.

返回 A list of paths.

返回类型 list[dict]

_get_path_list_from_ann()

Get list of paths from annotation file.

返回 List of paths.

返回类型 List

_get_path_list_from_folder()

Get list of paths from folder.

返回 List of paths.

返回类型 List

class mmagic.datasets.**CIFAR10**(*data_prefix: str*, *test_mode: bool*, *metainfo: Optional[dict] = None*, *data_root: str = ''*, *download: bool = True*, ***kwargs*)

Bases: mmagic.datasets.basic_conditional_dataset.BasicConditionalDataset

CIFAR10 Dataset.

This implementation is modified from https://github.com/pytorch/vision/blob/master/torchvision/datasets/cifar.py

参数

- **data_prefix** (*str*) –Prefix for data.

- **test_mode** (*bool*) –`test_mode=True` means in test phase. It determines to use the training set or test set.
- **metainfo** (*dict, optional*) –Meta information for dataset, such as categories information. Defaults to None.
- **data_root** (*str*) –The root directory for `data_prefix`. Defaults to ''.
- **download** (*bool*) –Whether to download the dataset if not exists. Defaults to True.
- ****kwargs** –Other keyword arguments in `BaseDataset`.

base_folder = 'cifar-10-batches-py'

url = 'https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz'

filename = 'cifar-10-python.tar.gz'

tgz_md5 = 'c58f30108f718f92721af3b95e74349a'

train_list = [['data_batch_1', 'c99cafc152244af753f735de768cd75f'], ['data_batch_2',...

test_list = [['test_batch', '40351d587109b95175f43aff81a1287e']]

meta

METAINFO

load_data_list()

Load images and ground truth labels.

_load_meta()

Load categories information from metafile.

_check_integrity()

Check the integrity of data files.

extra_repr() → List[str]

The extra repr information of the dataset.

class mmagic.datasets.**AdobeComp1kDataset**(*ann_file: Optional[str] = ''*, *metainfo: Optional[dict] = None*, *data_root: Optional[str] = ''*, *data_prefix: dict = dict(img_path='')*, *filter_cfg: Optional[dict] = None*, *indices: Optional[Union[int, Sequence[int]]] = None*, *serialize_data: bool = True*, *pipeline: List[Union[dict, Callable]] = []*, *test_mode: bool = False*, *lazy_init: bool = False*, *max_refetch: int = 1000*)

Bases: `mmengine.dataset.BaseDataset`

Adobe composition-1k dataset.

The dataset loads (alpha, fg, bg) data and apply specified transforms to the data. You could specify whether composite merged image online or load composited merged image in pipeline.

Example for online comp-1k dataset:

```
[
    {
        "alpha": 'alpha/000.png',
        "fg": 'fg/000.png',
        "bg": 'bg/000.png'
    },
    {
        "alpha": 'alpha/001.png',
        "fg": 'fg/001.png',
        "bg": 'bg/001.png'
    },
]
```

Example for offline comp-1k dataset:

```
[
    {
        "alpha": 'alpha/000.png',
        "merged": 'merged/000.png',
        "fg": 'fg/000.png',
        "bg": 'bg/000.png'
    },
    {
        "alpha": 'alpha/001.png',
        "merged": 'merged/001.png',
        "fg": 'fg/001.png',
        "bg": 'bg/001.png'
    },
]
```

参数

- **ann_file** (*str*) –Annotation file path. Defaults to ''.
- **data_root** (*str, optional*) –The root directory for `data_prefix` and `ann_file`. Defaults to None.
- **pipeline** (*list, optional*) –Processing pipeline. Defaults to [].
- **test_mode** (*bool, optional*) –`test_mode=True` means in test phase. Defaults to False.

- ****kwargs** –Other arguments passed to `mmengine.dataset.BaseDataset`.

实际案例

See unit-tests TODO: Move some codes in unittest here

METAINFO

load_data_list() → List[dict]

Load annotations from an annotation file named as `self.ann_file`

In order to be compatible to both new and old annotation format, we copy implementations from mmengine and do some modifications.

返回 A list of annotation.

返回类型 list[dict]

parse_data_info(*raw_data_info: dict*) → Union[dict, List[dict]]

Join data_root to each path in data_info.

class mmagic.datasets.**ControlNetDataset**(*ann_file: str = 'prompt.json'*, *data_root: str = './data/fill50k'*, *control_key='source'*, *image_key='target'*, *pipeline: List[Union[dict, Callable]] = []*)

Bases: `mmengine.dataset.BaseDataset`

Demo dataset to test ControlNet. Modified from https://github.com/lllyasviel/ControlNet/blob/16ea3b5379c1e78a4bc8e3fc9cae8d65c42511b1/tutorial_data set.py # noqa.

You can download the demo data from https://huggingface.co/lllyasviel/ControlNet/blob/main/training/fill50k.zip # noqa and then unzip the file to the `data` folder.

参数

- **ann_file** (*str*) –Path to the annotation file. Defaults to 'prompt.json' as ControlNet's default.
- **data_root** (*str*) –Path to the data root. Defaults to './data/fill50k'.
- **pipeline** (*list[dict | callable]*) –A sequence of data transforms.

load_data_list() → List[dict]

Load annotations from an annotation file named as `self.ann_file`

返回 A list of annotation.

返回类型 list[dict]

class mmagic.datasets.**DreamBoothDataset**(*data_root: str*, *concept_dir: str*, *prompt: str*, *pipeline: List[Union[dict, Callable]] = []*)

Bases: mmengine.dataset.BaseDataset

Dataset for DreamBooth.

参数

- **data_root** (*str*) –Path to the data root.
- **concept_dir** (*str*) –Path to the concept images.
- **prompt** (*str*) –Prompt of the concept.
- **pipeline** (*list[dict | callable]*) –A sequence of data transforms.

load_data_list() → list

Load data list from concept_dir and class_dir.

class mmagic.datasets.**GrowScaleImgDataset**(*data_roots: dict*, *pipeline*, *len_per_stage=int(1000000.0)*, *gpu_samples_per_scale=None*, *gpu_samples_base=32*, *io_backend: Optional[str] = None*, *file_lists: Optional[Union[str, dict]] = None*, *test_mode=False*)

Bases: mmengine.dataset.BaseDataset

Grow Scale Unconditional Image Dataset.

This dataset is similar with UnconditionalImageDataset, but offer more dynamic functionalities for the supporting complex algorithms, like PGGAN.

Highlight functionalities:

1. Support growing scale dataset. The motivation is to decrease data pre-processing load in CPU. In this dataset, you can provide imgs_roots like:

```
{'64': 'path_to_64x64_imgs',
 '512': 'path_to_512x512_imgs'}
```

   Then, in training scales lower than 64x64, this dataset will set self.imgs_root as 'path_to_64x64_imgs' ;

2. Offer samples_per_gpu according to different scales. In this dataset, self.samples_per_gpu will help runner to know the updated batch size.

Basically, This dataset contains raw images for training unconditional GANs. Given a root dir, we will recursively find all images in this root. The transformation on data is defined by the pipeline.

参数

- **imgs_root** (*str*) –Root path for unconditional images.
- **pipeline** (*list[dict | callable]*) –A sequence of data transforms.

- **len_per_stage** (*int, optional*) –The length of dataset for each scale. This args change the length dataset by concatenating or extracting subset. If given a value less than 0., the original length will be kept. Defaults to 1e6.
- **gpu_samples_per_scale** (*dict | None, optional*) –Dict contains `samples_per_gpu` for each scale. For example, `{'32': 4}` will set the scale of 32 with `samples_per_gpu=4`, despite other scale with `samples_per_gpu=self.gpu_samples_base`.
- **gpu_samples_base** (*int, optional*) –Set default `samples_per_gpu` for each scale. Defaults to 32.
- **io_backend** (*str, optional*) –The storage backend type. Options are "disk", "ceph", "memcached", "lmdb", "http" and "petrel". Default: None.
- **test_mode** (*bool, optional*) –If True, the dataset will work in test mode. Otherwise, in train mode. Default to False.

_VALID_IMG_SUFFIX = ('.jpg', '.png', '.jpeg', '.JPEG')

load_data_list()

Load annotations.

update_annotations(*curr_scale*)

Update annotations.

参数 **curr_scale** (*int*) –Current image scale.

返回 Whether to update.

返回类型 bool

concat_imgs_list_to(*num*)

Concat image list to specified length.

参数 **num** (*int*) –The length of the concatenated image list.

prepare_train_data(*idx*)

Prepare training data.

参数 **idx** (*int*) –Index of current batch.

返回 Prepared training data batch.

返回类型 dict

prepare_test_data(*idx*)

Prepare testing data.

参数 **idx** (*int*) –Index of current batch.

返回 Prepared training data batch.

返回类型 dict

__getitem__(*idx*)

Get the idx-th image and data information of dataset after `self.pipeline`, and `full_init` will be called if the dataset has not been fully initialized.

During training phase, if `self.pipeline` get `None`, `self._rand_another` will be called until a valid image is fetched or

the maximum limit of refetch is reached.

参数 **idx** (*int*) –The index of self.data_list.

返回 The idx-th image and data information of dataset after `self.pipeline`.

返回类型 dict

__repr__()

Print `self.transforms` in sequence.

返回 Formatted string.

返回类型 str

class mmagic.datasets.**ImageNet**(*ann_file: str = ''*, *metainfo: Optional[dict] = None*, *data_root: str = ''*, *data_prefix: Union[str, dict] = ''*, ***kwargs*)

Bases: `mmagic.datasets.basic_conditional_dataset.BasicConditionalDataset`

ImageNet Dataset.

The dataset supports two kinds of annotation format. More details can be found in `CustomDataset`.

参数

- **ann_file** (*str*) –Annotation file path. Defaults to ''.
- **metainfo** (*dict, optional*) –Meta information for dataset, such as class information. Defaults to None.
- **data_root** (*str*) –The root directory for `data_prefix` and `ann_file`. Defaults to ''.
- **data_prefix** (*str | dict*) –Prefix for training data. Defaults to ''.
- ****kwargs** –Other keyword arguments in `CustomDataset` and `BaseDataset`.

IMG_EXTENSIONS = ('.jpg', '.jpeg', '.png', '.ppm', '.bmp', '.pgm', '.tif')

METAINFO

class mmagic.datasets.**MSCoCoDataset**(*ann_file: str = ''*, *metainfo: Optional[dict] = None*, *data_root: str = ''*, *drop_caption_rate=0.0*, *phase='train'*, *year=2014*, *data_prefix: Union[str, dict] = ''*, *extensions: Sequence[str] = ('.jpg', '.jpeg', '.png', '.ppm', '.bmp', '.pgm', '.tif')*, *lazy_init: bool = False*, *classes: Union[str, Sequence[str], None] = None*, *caption_style: str = ''*, ***kwargs*)

Bases: mmagic.datasets.basic_conditional_dataset.BasicConditionalDataset

MSCoCo 2014 dataset.

参数

- **ann_file** (*str*) –Annotation file path. Defaults to ''.
- **metainfo** (*dict, optional*) –Meta information for dataset, such as class information. Defaults to None.
- **data_root** (*str*) –The root directory for data_prefix and ann_file. Defaults to ''.
- **drop_caption_rate** (*float, optional*) –Rate of dropping caption, used for training. Defaults to 0.0.
- **phase** (*str, optional*) –Subdataset used for certain phase, can be set to *train*, *test* and *val*. Defaults to 'train'.
- **year** (*int, optional*) –Version of CoCo dataset, can be set to 2014 and 2017. Defaults to 2014.
- **data_prefix** (*str | dict*) –Prefix for the data. Defaults to ''.
- **extensions** (*Sequence[str]*) –A sequence of allowed extensions. Defaults to ('.jpg', '.jpeg', '.png', '.ppm', '.bmp', '.pgm', '.tif').
- **lazy_init** (*bool*) –Whether to load annotation during instantiation. In some cases, such as visualization, only the meta information of the dataset is needed, which is not necessary to load annotation file. Basedataset can skip load annotations to save time by set lazy_init=False. Defaults to False.
- **caption_style** (*str*) –If you want to add a style description for each caption, you can set caption_style to your style prompt. For example, 'realistic style'. Defaults to empty str.
- ****kwargs** –Other keyword arguments in BaseDataset.

METAINFO

load_data_list()

Load image paths and gt_labels.

class mmagic.datasets.**PairedImageDataset**(*data_root*, *pipeline*, *io_backend: Optional[str] = None*, *test_mode=False*, *test_dir='test'*)

Bases: mmengine.dataset.BaseDataset

General paired image folder dataset for image generation.

It assumes that the training directory is '/path/to/data/train'. During test time, the directory is '/path/to/data/test'. '/path/to/data' can be initialized by args 'dataroot'. Each sample contains a pair of images concatenated in the w dimension (A|B).

参数

- **dataroot** (str | Path) –Path to the folder root of paired images.
- **pipeline** (*List[dict | callable]*) –A sequence of data transformations.
- **test_mode** (*bool*) –Store *True* when building test dataset. Default: *False*.
- **test_dir** (*str*) –Subfolder of dataroot which contain test images. Default: 'test'.

load_data_list()

Load paired image paths.

返回 List that contains paired image paths.

返回类型 list[dict]

scan_folder(*path*)

Obtain image path list (including sub-folders) from a given folder.

参数 **path** (str | Path) –Folder path.

返回 Image list obtained from the given folder.

返回类型 list[str]

class mmagic.datasets.**SinGANDataset**(*data_root*, *min_size*, *max_size*, *scale_factor_init*, *pipeline*, *num_samples=-1*)

Bases: mmengine.dataset.BaseDataset

SinGAN Dataset.

In this dataset, we create an image pyramid and save it in the cache.

参数

- **img_path** (*str*) –Path to the single image file.
- **min_size** (*int*) –Min size of the image pyramid. Here, the number will be set to the min(H, W).
- **max_size** (*int*) –Max size of the image pyramid. Here, the number will be set to the max(H, W).
- **scale_factor_init** (*float*) –Rescale factor. Note that the actual factor we use may be a little bit different from this value.

- **num_samples** (*int, optional*) –The number of samples (length) in this dataset. Defaults to -1.

full_init()

Skip the full init process for SinGANDataset.

load_data_list(*min_size*, *max_size*, *scale_factor_init*)

Load annotations for SinGAN Dataset.

参数

- **min_size** (*int*) –The minimum size for the image pyramid.
- **max_size** (*int*) –The maximum size for the image pyramid.
- **scale_factor_init** (*float*) –The initial scale factor.

__getitem__(*index*)

Get *:attr:self.data_dict*. For SinGAN, we use single image with different resolution to train the model.

参数 **idx** (*int*) –This will be ignored in *SinGANDataset*.

返回 Dict contains input image in different resolution. `self.pipeline`.

返回类型 dict

__len__()

Get the length of filtered dataset and automatically call `full_init` if the dataset has not been fully init.

返回 The length of filtered dataset.

返回类型 int

class mmagic.datasets.**TextualInversionDataset**(*data_root: str*, *concept_dir: str*, *placeholder: str*, *is_style: bool = False*, *pipeline: List[Union[dict, Callable]] = []*)

Bases: `mmengine.dataset.BaseDataset`

Dataset for DreamBooth.

参数

- **data_root** (*str*) –Path to the data root.
- **concept_dir** (*str*) –Path to the concept images.
- **is_style** (*bool*) –
- **prompt** (*str*) –Prompt of the concept.
- **pipeline** (*list[dict | callable]*) –A sequence of data transforms.

load_data_list() → list

Load data list from concept_dir and class_dir.

prepare_data(*idx*)

Get data processed by `self.pipeline`.

参数 **idx** (*int*) –The index of `data_info`.

返回 Depends on `self.pipeline`.

返回类型 Any

class mmagic.datasets.**UnpairedImageDataset**(*data_root*, *pipeline*, *io_backend: Optional[str] = None*, *test_mode=False*, *domain_a='A'*, *domain_b='B'*)

Bases: `mmengine.dataset.BaseDataset`

General unpaired image folder dataset for image generation.

It assumes that the training directory of images from domain A is '/path/to/data/trainA', and that from domain B is '/path/to/data/trainB', respectively. '/path/to/data' can be initialized by args 'dataroot'. During test time, the directory is '/path/to/data/testA' and '/path/to/data/testB', respectively.

参数

- **dataroot** (str | `Path`) –Path to the folder root of unpaired images.
- **pipeline** (*List[dict | callable]*) –A sequence of data transformations.
- **io_backend** (*str, optional*) –The storage backend type. Options are "disk", "ceph", "memcached", "lmdb", "http" and "petrel". Default: None.
- **test_mode** (*bool*) –Store *True* when building test dataset. Default: *False*.
- **domain_a** (*str, optional*) –Domain of images in trainA / testA. Defaults to 'A'.
- **domain_b** (*str, optional*) –Domain of images in trainB / testB. Defaults to 'B'.

load_data_list()

Load the data list.

返回 The data info list of source and target domain.

返回类型 list

_load_domain_data_list(*dataroot*)

Load unpaired image paths of one domain.

参数 **dataroot** (*str*) –Path to the folder root for unpaired images of one domain.

返回 List that contains unpaired image paths of one domain.

返回类型 list[dict]

get_data_info(*idx*) → dict

Get annotation by index and automatically call `full_init` if the dataset has not been fully initialized.

参数 **idx** (*int*) –The index of data.

返回 The idx-th annotation of the dataset.

返回类型 dict

__len__()

The length of the dataset.

scan_folder(*path*)

Obtain image path list (including sub-folders) from a given folder.

参数 **path** (str | Path) –Folder path.

返回 Image list obtained from the given folder.

返回类型 list[str]

CHAPTER 71

mmagic.datasets.transforms

71.1 Package Contents

71.1.1 Classes

AlbuCorruptFunction	AlbuCorruptFunction augmentation.
PairedAlbuTransForms	PairedAlbuTransForms augmentation.
Albumentations	Albumentation augmentation.
GenerateSeg	Generate segmentation mask from alpha matte.
GenerateSoftSeg	Generate soft segmentation mask from input segmentation mask.
MirrorSequence	Extend short sequences (e.g. Vimeo-90K) by mirroring the sequences.
TemporalReverse	Reverse frame lists for temporal augmentation.
BinarizeImage	Binarize image.
Clip	Clip the pixels.
ColorJitter	An interface for torch color jitter so that it can be invoked in mmagic
RandomAffine	Apply random affine to input images.
RandomMaskDilation	Randomly dilate binary masks.

下页继续

表 1 – 续上页

UnsharpMasking	Apply unsharp masking to an image or a sequence of images.
Flip	Flip the input data with a probability.
NumpyPad	Numpy Padding.
RandomRotation	Rotate the image by a randomly-chosen angle, measured in degree.
RandomTransposeHW	Randomly transpose images in H and W dimensions with a probability.
Resize	Resize data to a specific size for training or resize the images to fit
CenterCropLongEdge	Center crop the given image by the long edge.
Crop	Crop data to specific size for training.
CropAroundCenter	Randomly crop the images around unknown area in the center 1/4 images.
CropAroundFg	Crop around the whole foreground in the segmentation mask.
CropAroundUnknown	Crop around unknown area with a randomly selected scale.
CropLike	Crop/pad the image in the target_key according to the size of image in
FixedCrop	Crop paired data (at a specific position) to specific size for training.
InstanceCrop	Use maskrcnn to detect instances on image.
ModCrop	Mod crop images, used during testing.
PairedRandomCrop	Paired random crop.
RandomCropLongEdge	Random crop the given image by the long edge.
RandomResizedCrop	Crop data to random size and aspect ratio.
CompositeFg	Composite foreground with a random foreground.
MergeFgAndBg	Composite foreground image and background image with alpha.
PerturbBg	Randomly add gaussian noise or gamma change to background image.
RandomJitter	Randomly jitter the foreground in hsv space.
RandomLoadResizeBg	Randomly load a background image and resize it.
PackInputs	Pack data into DataSample for training, evaluation and testing.
GenerateCoordinateAndCell	Generate coordinate and cell. Generate coordinate from the desired size
GenerateFacialHeatmap	Generate heatmap from keypoint.

下页继续

表 1 – 续上页

GenerateFrameIndices	Generate frame index for REDS datasets. It also performs temporal
GenerateFrameIndiceswithPadding	Generate frame index with padding for REDS dataset and Vid4 dataset
GenerateSegmentIndices	Generate frame indices for a segment. It also performs temporal
GetMaskedImage	Get masked image.
GetSpatialDiscountMask	Get spatial discounting mask constant.
LoadImageFromFile	Load a single image or image frames from corresponding paths. Required
LoadMask	Load Mask for multiple types.
LoadPairedImageFromFile	Load a pair of images from file.
MATLABLikeResize	Resize the input image using MATLAB-like downsampling.
Normalize	Normalize images with the given mean and std value.
RescaleToZeroOne	Transform the images into a range between 0 and 1.
DegradationsWithShuffle	Apply random degradations to input, with degradations being shuffled.
RandomBlur	Apply random blur to the input.
RandomJPEGCompression	Apply random JPEG compression to the input.
RandomNoise	Apply random noise to the input.
RandomResize	Randomly resize the input.
RandomVideoCompression	Apply random video compression to the input.
RandomDownSampling	Generate LQ image from GT (and crop), which will randomly pick a scale.
FormatTrimap	Convert trimap (tensor) to one-hot representation.
GenerateTrimap	Using random erode/dilate to generate trimap from alpha matte.
GenerateTrimapWithDistTransform	Generate trimap with distance transform function.
TransformTrimap	Transform trimap into two-channel and six-channel.
CopyValues	Copy the value of source keys to destination keys.
SetValues	Set value to destination keys.

class mmagic.datasets.transforms.**AlbuCorruptFunction**(*keys: List[str]*, *config: List[dict]*, *p: float = 1.0*)

Bases: mmcv.transforms.BaseTransform

AlbuCorruptFunction augmentation.

Apply the same AlbuCorruptFunction augmentation to the input images.

transform(*results*)

processing input results according to *self.augs*.

参数

- **results** (*dict*) –contains the processed data
- **pipeline.** (*through the transform*) –

返回 the processed data.

返回类型 results

__repr__()

Return repr(self).

class mmagic.datasets.transforms.**PairedAlbuTransForms**(*size: int*, *lq_key: str = 'img'*, *gt_key: str = 'gt'*, *scope: str = 'geometric'*, *crop: str = 'random'*, *p: float = 0.5*)

Bases: mmcv.transforms.BaseTransform

PairedAlbuTransForms augmentation.

Apply the same AlbuTransforms augmentation to paired images.

transform(*results*)

processing input results according to *self.pipeline*.

参数

- **results** (*dict*) –contains the processed data
- **pipeline.** (*through the transform*) –

返回 the processed data.

返回类型 results

__repr__()

Return repr(self).

class mmagic.datasets.transforms.**Albumentations**(*keys: List[str]*, *transforms: List[dict]*)

Bases: mmcv.transforms.BaseTransform

Albumentation augmentation.

Adds custom transformations from Albumentations library. Please, visit *https://github.com/albumentations-team/albumentations* and *https://albumentations.ai/docs/getting_started/transforms_and_targets* to get more information.

An example of transforms is as followed:

```
albu_transforms = [
    dict(
        type='Resize',
        height=100,
        width=100,
    ),
    dict(
        type='RandomFog',
        p=0.5,
    ),
    dict(
        type='RandomRain',
        p=0.5
    ),
    dict(
        type='RandomSnow',
        p=0.5,
    ),
]
pipeline = [
    dict(
        type='LoadImageFromFile',
        key='img',
        color_type='color',
        channel_order='rgb',
        imdecode_backend='cv2'),
    dict(
        type='Albumentations',
        keys=['img'],
        transforms=albu_transforms),
    dict(type='PackInputs')
]
```

参数

- **keys** (*list[str]*) –A list specifying the keys whose values are modified.
- **transforms** (*list[dict]*) –A list of albu transformations.

albu_builder(*cfg: dict*) → albumentations

Import a module from albumentations.

It inherits some of `build_from_cfg()` logic.

参数 **cfg** (*dict*) –Config dict. It should at least contain the key “type”.

返回 The constructed object.

返回类型 obj

_apply_albu(*imgs*)

transform(*results*)

Transform function of Albumentations.

__repr__()

Return repr(self).

class mmagic.datasets.transforms.**GenerateSeg**(*kernel_size=5*, *erode_iter_range=(10, 20)*, *dilate_iter_range=(15, 30)*, *num_holes_range=(0, 3)*, *hole_sizes=[(15, 15), (25, 25), (35, 35), (45, 45)]*, *blur_ksizes=[(21, 21), (31, 31), (41, 41)]*)

Bases: mmcv.transforms.BaseTransform

Generate segmentation mask from alpha matte.

参数

- **kernel_size** (*int, optional*) –Kernel size for both erosion and dilation. The kernel will have the same height and width. Defaults to 5.
- **erode_iter_range** (*tuple, optional*) –Iteration of erosion. Defaults to (10, 20).
- **dilate_iter_range** (*tuple, optional*) –Iteration of dilation. Defaults to (15, 30).
- **num_holes_range** (*tuple, optional*) –Range of number of holes to randomly select from. Defaults to (0, 3).
- **hole_sizes** (*list, optional*) –List of (h, w) to be selected as the size of the rectangle hole. Defaults to [(15, 15), (25, 25), (35, 35), (45, 45)].
- **blur_ksizes** (*list, optional*) –List of (h, w) to be selected as the kernel_size of the gaussian blur. Defaults to [(21, 21), (31, 31), (41, 41)].

static _crop_hole(*img*, *start_point*, *hole_size*)

Create a all-zero rectangle hole in the image.

参数

- **img** (*np.ndarray*) –Source image.
- **start_point** (*tuple[int]*) –The top-left point of the rectangle.
- **hole_size** (*tuple[int]*) –The height and width of the rectangle hole.

返回 The cropped image.

返回类型 np.ndarray

transform(*results: dict*) → dict

Transform function.

参数 **results** (*dict*) –A dict containing the necessary information and data for augmentation.

返回 A dict containing the processed data and information.

返回类型 dict

__repr__()

Return repr(self).

class mmagic.datasets.transforms.**GenerateSoftSeg**(*fg_thr=0.2*, *border_width=25*, *erode_ksize=3*, *dilate_ksize=5*, *erode_iter_range=(10, 20)*, *dilate_iter_range=(3, 7)*, *blur_ksizes=[(21, 21), (31, 31), (41, 41)]*)

Bases: mmcv.transforms.BaseTransform

Generate soft segmentation mask from input segmentation mask.

Required key is "seg", added key is "soft_seg".

参数

- **fg_thr** (*float, optional*) –Threshold of the foreground in the normalized input segmentation mask. Defaults to 0.2.
- **border_width** (*int, optional*) –Width of border to be padded to the bottom of the mask. Defaults to 25.
- **erode_ksize** (*int, optional*) –Fixed kernel size of the erosion. Defaults to 5.
- **dilate_ksize** (*int, optional*) –Fixed kernel size of the dilation. Defaults to 5.
- **erode_iter_range** (*tuple, optional*) –Iteration of erosion. Defaults to (10, 20).
- **dilate_iter_range** (*tuple, optional*) –Iteration of dilation. Defaults to (3, 7).
- **blur_ksizes** (*list, optional*) –List of (h, w) to be selected as the kernel_size of the gaussian blur. Defaults to [(21, 21), (31, 31), (41, 41)].

transform(*results: dict*) → dict

Transform function.

参数 **results** (*dict*) –A dict containing the necessary information and data for augmentation.

返回 A dict containing the processed data and information.

返回类型 dict

__repr__()

Return repr(self).

class mmagic.datasets.transforms.**MirrorSequence**(*keys*)

Bases: mmcv.transforms.BaseTransform

Extend short sequences (e.g. Vimeo-90K) by mirroring the sequences.

Given a sequence with N frames (x1, ⋯, xN), extend the sequence to (x1, ⋯, xN, xN, ⋯, x1).

Required Keys:

- [KEYS]

Modified Keys:

- [KEYS]

参数 **keys** (*list[str]*) –The frame lists to be extended.

transform(*results*)

transform function.

参数 **results** (*dict*) –A dict containing the necessary information and data for augmentation.

返回 A dict containing the processed data and information.

返回类型 dict

__repr__()

Return repr(self).

class mmagic.datasets.transforms.**TemporalReverse**(*keys*, *reverse_ratio=0.5*)

Bases: mmcv.transforms.BaseTransform

Reverse frame lists for temporal augmentation.

Required keys are the keys in attributes “lq” and “gt”, added or modified keys are “lq”, “gt” and “reverse”.

参数

- **keys** (*list[str]*) –The frame lists to be reversed.
- **reverse_ratio** (*float*) –The probability to reverse the frame lists. Default: 0.5.

transform(*results*)

transform function.

参数 **results** (*dict*) –A dict containing the necessary information and data for augmentation.

返回 A dict containing the processed data and information.

返回类型 dict

__repr__()

Return repr(self).

class mmagic.datasets.transforms.**BinarizeImage**(*keys*, *binary_thr*, *a_min=0*, *a_max=1*, *dtype=np.uint8*)

Bases: mmcv.transforms.BaseTransform

Binarize image.

参数

- **keys** (*Sequence[str]*) –The images to be binarized.
- **binary_thr** (*float*) –Threshold for binarization.
- **a_min** (*int*) –Lower limits of pixel value.
- **a_max** (*int*) –Upper limits of pixel value.
- **dtype** (*np.dtype*) –Set the data type of the output. Default: np.uint8

_binarize(*img*)

Binarize image.

参数 **img** (*np.ndarray*) –Input image.

返回 Output image.

返回类型 img (np.ndarray)

transform(*results*)

The transform function of BinarizeImage.

参数 **results** (*dict*) –A dict containing the necessary information and data for augmentation.

返回 A dict containing the processed data and information.

返回类型 dict

__repr__()

Return repr(self).

class mmagic.datasets.transforms.**Clip**(*keys*, *a_min=0*, *a_max=255*)

Bases: mmcv.transforms.BaseTransform

Clip the pixels.

Modified keys are the attributes specified in "keys".

参数

- **keys** (*list[str]*) –The keys whose values are clipped.
- **a_min** (*int*) –Lower limits of pixel value.
- **a_max** (*int*) –Upper limits of pixel value.

_clip(*input_*)

Clip the pixels.

参数 **input** (*Union[List, np.ndarray]*) –Pixels to clip.

返回 Clipped pixels.

返回类型 Union[List, np.ndarray]

transform(*results*)

transform function.

参数 **results** (*dict*) –A dict containing the necessary information and data for augmentation.

返回

A dict with the values of the specified keys are rounded and clipped.

返回类型 dict

__repr__()

Return repr(self).

class mmagic.datasets.transforms.**ColorJitter**(*keys*, *channel_order='rgb'*, ***kwargs*)

Bases: mmcv.transforms.BaseTransform

An interface for torch color jitter so that it can be invoked in mmagic pipeline.

Randomly change the brightness, contrast and saturation of an image. Modified keys are the attributes specified in "keys".

Required Keys:

- [KEYS]

Modified Keys:

- [KEYS]

参数

- **keys** (*list[str]*) –The images to be resized.
- **channel_order** (*str*) –Order of channel, candidates are 'bgr' and 'rgb'. Default: 'rgb'.

提示

`**kwards` follows the args list of `torchvision.transforms.ColorJitter`.

brightness (float or tuple of float (min, max)): How much to jitter brightness. brightness_factor is chosen uniformly from [max(0, 1 - brightness), 1 + brightness] or the given [min, max]. Should be non negative numbers.

contrast (float or tuple of float (min, max)): How much to jitter contrast. contrast_factor is chosen uniformly from [max(0, 1 - contrast), 1 + contrast] or the given [min, max]. Should be non negative numbers.

saturation (float or tuple of float (min, max)): How much to jitter saturation. saturation_factor is chosen uniformly from [max(0, 1 - saturation), 1 + saturation] or the given [min, max]. Should be non negative numbers.

hue (float or tuple of float (min, max)): How much to jitter hue. hue_factor is chosen uniformly from [-hue, hue] or the given [min, max]. Should have 0<= hue <= 0.5 or -0.5 <= min <= max <= 0.5.

_color_jitter(*image*, *this_seed*)

Color Jitter Function.

参数

- **image** (*np.ndarray*) –Image.
- **this_seed** (*int*) –Seed of torch.

返回 The output image.

返回类型 image (np.ndarray)

transform(*results: Dict*) → Dict

The transform function of ColorJitter.

参数 **results** (*dict*) –The result dict.

返回 The result dict.

返回类型 dict

__repr__()

Return repr(self).

class mmagic.datasets.transforms.**RandomAffine**(*keys*, *degrees*, *translate=None*, *scale=None*, *shear=None*, *flip_ratio=None*)

Bases: `mmcv.transforms.BaseTransform`

Apply random affine to input images.

This class is adopted from https://github.com/pytorch/vision/blob/v0.5.0/torchvision/transforms/ transforms.py#L1015 It should be noted that in https://github.com/Yaoyi-Li/GCA-Matting/blob/master/dataloader/

data_generator.py#L70 random flip is added. See explanation of *flip_ratio* below. Required keys are the keys in attribute "keys", modified keys are keys in attribute "keys".

参数

- **keys** (*Sequence[str]*) –The images to be affined.
- **degrees** (*float | tuple[float]*) –Range of degrees to select from. If it is a float instead of a tuple like (min, max), the range of degrees will be (-degrees, +degrees). Set to 0 to deactivate rotations.
- **translate** (*tuple, optional*) –Tuple of maximum absolute fraction for horizontal and vertical translations. For example translate=(a, b), then horizontal shift is randomly sampled in the range -img_width * a < dx < img_width * a and vertical shift is randomly sampled in the range -img_height * b < dy < img_height * b. Default: None.
- **scale** (*tuple, optional*) –Scaling factor interval, e.g (a, b), then scale is randomly sampled from the range a <= scale <= b. Default: None.
- **shear** (*float | tuple[float], optional*) –Range of shear degrees to select from. If shear is a float, a shear parallel to the x axis and a shear parallel to the y axis in the range (-shear, +shear) will be applied. Else if shear is a tuple of 2 values, a x-axis shear and a y-axis shear in (shear[0], shear[1]) will be applied. Default: None.
- **flip_ratio** (*float, optional*) –Probability of the image being flipped. The flips in horizontal direction and vertical direction are independent. The image may be flipped in both directions. Default: None.

static _get_params(*degrees*, *translate*, *scale_ranges*, *shears*, *flip_ratio*, *img_size*)

Get parameters for affine transformation.

返回 Params to be passed to the affine transformation.

返回类型 paras (tuple)

static _get_inverse_affine_matrix(*center*, *angle*, *translate*, *scale*, *shear*, *flip*)

Helper method to compute inverse matrix for affine transformation.

As it is explained in PIL.Image.rotate, we need compute INVERSE of affine transformation matrix: M = T * C * RSS * C^-1 where T is translation matrix:

[1, 0, tx | 0, 1, ty | 0, 0, 1];

C is translation matrix to keep center: [1, 0, cx | 0, 1, cy | 0, 0, 1];

RSS is rotation with scale and shear matrix.

It is different from the original function in torchvision. 1. The order are changed to flip -> scale -> rotation -> shear. 2. x and y have different scale factors. RSS(shear, a, scale, f) =

[cos(a + shear)*scale_x*f -sin(a + shear)*scale_y 0] [sin(a)*scale_x*f cos(a)*scale_y 0] [0 0 1]

Thus, the inverse is M^-1 = C * RSS^-1 * C^-1 * T^-1.

transform(*results*)

transform function.

参数 **results** (*dict*) –A dict containing the necessary information and data for augmentation.

返回 A dict containing the processed data and information.

返回类型 dict

__repr__()

Return repr(self).

class mmagic.datasets.transforms.**RandomMaskDilation**(*keys*, *binary_thr=0.0*, *kernel_min=9*, *kernel_max=49*)

Bases: mmcv.transforms.BaseTransform

Randomly dilate binary masks.

参数

- **keys** (*Sequence[str]*) –The images to be resized.
- **binary_thr** (*float*) –Threshold for obtaining binary mask. Default: 0.
- **kernel_min** (*int*) –Min size of dilation kernel. Default: 9.
- **kernel_max** (*int*) –Max size of dilation kernel. Default: 49.

_random_dilate(*img*)

transform(*results*)

transform function.

参数 **results** (*dict*) –A dict containing the necessary information and data for augmentation.

返回 A dict containing the processed data and information.

返回类型 dict

__repr__()

Return repr(self).

class mmagic.datasets.transforms.**UnsharpMasking**(*kernel_size*, *sigma*, *weight*, *threshold*, *keys*)

Bases: mmcv.transforms.BaseTransform

Apply unsharp masking to an image or a sequence of images.

参数

- **kernel_size** (*int*) –The kernel_size of the Gaussian kernel.
- **sigma** (*float*) –The standard deviation of the Gaussian.

- **weight** (*float*) –The weight of the "details" in the final output.
- **threshold** (*float*) –Pixel differences larger than this value are regarded as "details".
- **keys** (*list[str]*) –The keys whose values are processed.

Added keys are "xxx_unsharp", where "xxx" are the attributes specified in "keys".

_unsharp_masking(*imgs*)

Unsharp masking function.

transform(*results*)

transform function.

参数 **results** (*dict*) –A dict containing the necessary information and data for augmentation.

返回 A dict containing the processed data and information.

返回类型 dict

__repr__()

Return repr(self).

class mmagic.datasets.transforms.**Flip**(*keys*, *flip_ratio=0.5*, *direction='horizontal'*)

Bases: mmcv.transforms.BaseTransform

Flip the input data with a probability.

Reverse the order of elements in the given data with a specific direction. The shape of the data is preserved, but the elements are reordered. Required keys are the keys in attributes "keys", added or modified keys are "flip", "flip_direction" and the keys in attributes "keys". It also supports flipping a list of images with the same flip.

Required Keys:

- [KEYS]

Modified Keys:

- [KEYS]

参数

- **keys** (*Union[str, List[str]]*) –The images to be flipped.
- **flip_ratio** (*float*) –The probability to flip the images. Default: 0.5.
- **direction** (*str*) –Flip images horizontally or vertically. Options are "horizontal" | "vertical". Default: "horizontal".

_directions = ['horizontal', 'vertical']

transform(*results*)

transform function.

参数 **results** (*dict*) –A dict containing the necessary information and data for augmentation.

返回 A dict containing the processed data and information.

返回类型 dict

__repr__()

Return repr(self).

class mmagic.datasets.transforms.**NumpyPad**(*keys*, *padding*, ***kwargs*)

Bases: mmcv.transforms.BaseTransform

Numpy Padding.

In this augmentation, numpy padding is adopted to customize padding augmentation. Please carefully read the numpy manual in: https://numpy.org/doc/stable/reference/generated/numpy.pad.html

If you just hope a single dimension to be padded, you must set `padding` like this:

```
padding = ((2, 2), (0, 0), (0, 0))
```

In this case, if you adopt an input with three dimension, only the first dimension will be padded.

参数

- **keys** (*Union[str, List[str]]*) –The images to be padded.
- **padding** (*int | tuple(int)*) –Please refer to the args `pad_width` in `numpy.pad`.

transform(*results*)

Call function.

参数 **results** (*dict*) –A dict containing the necessary information and data for augmentation.

返回 A dict containing the processed data and information.

返回类型 dict

__repr__() → str

Return repr(self).

class mmagic.datasets.transforms.**RandomRotation**(*keys*, *degrees*)

Bases: mmcv.transforms.BaseTransform

Rotate the image by a randomly-chosen angle, measured in degree.

参数

- **keys** (*list[str]*) –The images to be rotated.

- **degrees** (*tuple[float] | tuple[int] | float | int*) –If it is a tuple, it represents a range (min, max). If it is a float or int, the range is constructed as (-degrees, degrees).

transform(*results*)

transform function.

参数 **results** (*dict*) –A dict containing the necessary information and data for augmentation.

返回 A dict containing the processed data and information.

返回类型 dict

__repr__()

Return repr(self).

class mmagic.datasets.transforms.**RandomTransposeHW**(*keys*, *transpose_ratio=0.5*)

Bases: mmcv.transforms.BaseTransform

Randomly transpose images in H and W dimensions with a probability.

(TransposeHW = horizontal flip + anti-clockwise rotation by 90 degrees) When used with horizontal/vertical flips, it serves as a way of rotation augmentation. It also supports randomly transposing a list of images.

Required keys are the keys in attributes "keys", added or modified keys are "transpose" and the keys in attributes "keys".

参数

- **keys** (*list[str]*) –The images to be transposed.
- **transpose_ratio** (*float*) –The probability to transpose the images. Default: 0.5.

transform(*results*)

transform function.

参数 **results** (*dict*) –A dict containing the necessary information and data for augmentation.

返回 A dict containing the processed data and information.

返回类型 dict

__repr__()

Return repr(self).

class mmagic.datasets.transforms.**Resize**(*keys: Union[str, List[str]] = 'img'*, *scale=None*, *keep_ratio=False*, *size_factor=None*, *max_size=None*, *interpolation='bilinear'*, *backend=None*, *output_keys=None*)

Bases: mmcv.transforms.BaseTransform

Resize data to a specific size for training or resize the images to fit the network input regulation for testing.

When used for resizing images to fit network input regulation, the case is that a network may have several downsample and then upsample operation, then the input height and width should be divisible by the downsample factor of the network. For example, the network would downsample the input for 5 times with stride 2, then the downsample factor is 2^5 = 32 and the height and width should be divisible by 32.

Required keys are the keys in attribute "keys", added or modified keys are "keep_ratio", "scale_factor", "interpolation" and the keys in attribute "keys".

Required Keys:

- Required keys are the keys in attribute "keys"

Modified Keys:

- Modified the keys in attribute "keys" or save as new key ([OUT_KEY])

Added Keys:

- [OUT_KEY]_shape
- keep_ratio
- scale_factor
- interpolation

All keys in "keys" should have the same shape. "test_trans" is used to record the test transformation to align the input's shape.

参数

- **keys** (*str | list[str]*) –The image(s) to be resized.
- **scale** (*float | tuple[int]*) –If scale is tuple[int], target spatial size (h, w). Otherwise, target spatial size is scaled by input size. Note that when it is used, *size_factor* and *max_size* are useless. Default: None
- **keep_ratio** (*bool*) –If set to True, images will be resized without changing the aspect ratio. Otherwise, it will resize images to a given size. Default: False. Note that it is used together with *scale*.
- **size_factor** (*int*) –Let the output shape be a multiple of size_factor. Default:None. Note that when it is used, *scale* should be set to None and *keep_ratio* should be set to False.
- **max_size** (*int*) –The maximum size of the longest side of the output. Default:None. Note that it is used together with *size_factor*.
- **interpolation** (*str*) –Algorithm used for interpolation: "nearest"| "bilinear"| "bicubic" | "area" | "lanczos". Default: "bilinear".
- **backend** (*str | None*) –The image resize backend type. Options are *cv2*, *pillow*, *None*. If backend is None, the global imread_backend specified by `mmcv.use_backend()` will be used. Default: None.

- **output_keys** (*list[str] | None*) –The resized images. Default: None Note that if it is not *None*, its length should be equal to keys.

_resize(*img*)

Resize function.

参数 **img** (*np.ndarray*) –Image.

返回 Resized image.

返回类型 img (np.ndarray)

transform(*results: Dict*) → Dict

Transform function to resize images.

参数 **results** (*dict*) –A dict containing the necessary information and data for augmentation.

返回 A dict containing the processed data and information.

返回类型 dict

__repr__()

Return repr(self).

class mmagic.datasets.transforms.**CenterCropLongEdge**(*keys='img'*)

Bases: mmcv.transforms.BaseTransform

Center crop the given image by the long edge.

参数 **keys** (*list[str]*) –The images to be cropped.

transform(*results*)

Call function.

参数 **results** (*dict*) –A dict containing the necessary information and data for augmentation.

返回 A dict containing the processed data and information.

返回类型 dict

__repr__()

Return repr(self).

class mmagic.datasets.transforms.**Crop**(*keys*, *crop_size*, *random_crop=True*, *is_pad_zeros=False*)

Bases: mmcv.transforms.BaseTransform

Crop data to specific size for training.

参数

- **keys** (*Sequence[str]*) –The images to be cropped.
- **crop_size** (*Tuple[int]*) –Target spatial size (h, w).

- **random_crop** (*bool*) –If set to True, it will random crop image. Otherwise, it will work as center crop. Default: True.
- **is_pad_zeros** (*bool, optional*) –Whether to pad the image with 0 if crop_size is greater than image size. Default: False.

_crop(*data*)

Crop the data.

参数 **data** (*Union[List, np.ndarray]*) –Input data to crop.

返回 cropped data and corresponding crop box.

返回类型 tuple

transform(*results*)

Transform function.

参数 **results** (*dict*) –A dict containing the necessary information and data for augmentation.

返回 A dict containing the processed data and information.

返回类型 dict

__repr__()

Return repr(self).

class mmagic.datasets.transforms.**CropAroundCenter**(*crop_size*)

Bases: mmcv.transforms.BaseTransform

Randomly crop the images around unknown area in the center 1/4 images.

This cropping strategy is adopted in GCA matting. The *unknown area* is the same as *semi-transparent area*. https://arxiv.org/pdf/2001.04069.pdf

It retains the center 1/4 images and resizes the images to 'crop_size'. Required keys are "fg", "bg", "trimap" and "alpha", added or modified keys are "crop_bbox", "fg", "bg", "trimap" and "alpha".

参数 **crop_size** (*int | tuple*) –Desired output size. If int, square crop is applied.

transform(*results*)

Transform function.

参数 **results** (*dict*) –A dict containing the necessary information and data for augmentation.

返回 A dict containing the processed data and information.

返回类型 dict

__repr__()

Return repr(self).

class mmagic.datasets.transforms.**CropAroundFg**(*keys*, *bd_ratio_range=(0.1, 0.4)*, *test_mode=False*)

Bases: mmcv.transforms.BaseTransform

Crop around the whole foreground in the segmentation mask.

Required keys are "seg" and the keys in argument *keys*. Meanwhile, "seg" must be in argument *keys*. Added or modified keys are "crop_bbox" and the keys in argument *keys*.

参数

- **keys** (*Sequence[str]*) –The images to be cropped. It must contain 'seg'.
- **bd_ratio_range** (*tuple, optional*) –The range of the boundary (bd) ratio to select from. The boundary ratio is the ratio of the boundary to the minimal bbox that contains the whole foreground given by segmentation. Default to (0.1, 0.4).
- **test_mode** (*bool*) –Whether use test mode. In test mode, the tight crop area of foreground will be extended to the a square. Default to False.

transform(*results*)

Transform function.

参数 **results** (*dict*) –A dict containing the necessary information and data for augmentation.

返回 A dict containing the processed data and information.

返回类型 dict

class mmagic.datasets.transforms.**CropAroundUnknown**(*keys*, *crop_sizes*, *unknown_source='alpha'*, *interpolations='bilinear'*)

Bases: mmcv.transforms.BaseTransform

Crop around unknown area with a randomly selected scale.

Randomly select the w and h from a list of (w, h). Required keys are the keys in argument *keys*, added or modified keys are "crop_bbox" and the keys in argument *keys*. This class assumes value of "alpha" ranges from 0 to 255.

参数

- **keys** (*Sequence[str]*) –The images to be cropped. It must contain 'alpha'. If unknown_source is set to 'trimap', then it must also contain 'trimap'.
- **crop_sizes** (*list[int | tuple[int]]*) –List of (w, h) to be selected.
- **unknown_source** (*str, optional*) –Unknown area to select from. It must be 'alpha' or 'trimap'. Default to 'alpha'.
- **interpolations** (*str | list[str], optional*) –Interpolation method of mmcv.imresize. The interpolation operation will be applied when image size is smaller than

the crop_size. If given as a list of str, it should have the same length as *keys*. Or if given as a str all the keys will be resized with the same method. Default to 'bilinear'.

transform(*results*)

Transform function.

参数 **results** (*dict*) –A dict containing the necessary information and data for augmentation.

返回 A dict containing the processed data and information.

返回类型 dict

__repr__()

Return repr(self).

class mmagic.datasets.transforms.**CropLike**(*target_key*, *reference_key=None*)

Bases: mmcv.transforms.BaseTransform

Crop/pad the image in the target_key according to the size of image in the reference_key .

参数

- **target_key** (*str*) –The key needs to be cropped.
- **reference_key** (*str | None*) –The reference key, need its size. Default: None.

transform(*results*)

Transform function.

参数 **results** (*dict*) –A dict containing the necessary information and data for augmentation. Require self.target_key and self.reference_key.

返回

A dict containing the processed data and information. Modify self.target_key.

返回类型 dict

__repr__()

Return repr(self).

class mmagic.datasets.transforms.**FixedCrop**(*keys*, *crop_size*, *crop_pos=None*)

Bases: mmcv.transforms.BaseTransform

Crop paired data (at a specific position) to specific size for training.

参数

- **keys** (*Sequence[str]*) –The images to be cropped.
- **crop_size** (*Tuple[int]*) –Target spatial size (h, w).
- **crop_pos** (*Tuple[int]*) –Specific position (x, y). If set to None, random initialize the position to crop paired data batch. Default: None.

_crop(*data*, *x_offset*, *y_offset*, *crop_w*, *crop_h*)

Crop the data.

参数

- **data** (*Union[List, np.ndarray]*) –Input data to crop.
- **x_offset** (*int*) –The offset of x axis.
- **y_offset** (*int*) –The offset of y axis.
- **crop_w** (*int*) –The width of crop bbox.
- **crop_h** (*int*) –The height of crop bbox.

返回 cropped data and corresponding crop box.

返回类型 tuple

transform(*results*)

Transform function.

参数 **results** (*dict*) –A dict containing the necessary information and data for augmentation.

返回 A dict containing the processed data and information.

返回类型 dict

__repr__()

Return repr(self).

class mmagic.datasets.transforms.**InstanceCrop**(*config_file*, *key='img'*, *box_num_upbound=-1*, *finesize=256*)

Bases: mmcv.transforms.BaseTransform

Use maskrcnn to detect instances on image.

Mask R-CNN is used to detect the instance on the image pred_bbox is used to segment the instance on the image

参数

- **config_file** (*str*) –config file name relative to detectron2' s "configs/"
- **key** (*str*) –Unused
- **box_num_upbound** (*int*) –The upper limit on the number of instances in the figure

transform(*results: dict*) → dict

The transform function of InstanceCrop.

参数 **results** (*dict*) –A dict containing the necessary information and data for Conversion

返回

A dict containing the processed data and information.

返回类型 results (dict)

predict_bbox(*image*)

class mmagic.datasets.transforms.**ModCrop**(*key='gt'*)

Bases: mmcv.transforms.BaseTransform

Mod crop images, used during testing.

Required keys are "scale" and "KEY", added or modified keys are "KEY".

参数 **key** (*str*) –The key of image. Default: 'gt'

transform(*results*)

Transform function.

参数 **results** (*dict*) –A dict containing the necessary information and data for augmentation.

返回 A dict containing the processed data and information.

返回类型 dict

__repr__()

Return repr(self).

class mmagic.datasets.transforms.**PairedRandomCrop**(*gt_patch_size*, *lq_key='img'*, *gt_key='gt'*)

Bases: mmcv.transforms.BaseTransform

Paired random crop.

It crops a pair of img and gt images with corresponding locations. It also supports accepting img list and gt list. Required keys are "scale", "lq_key", and "gt_key", added or modified keys are "lq_key" and "gt_key".

参数

- **gt_patch_size** (*int*) –cropped gt patch size.
- **lq_key** (*str*) –Key of LQ img. Default: 'img'.
- **gt_key** (*str*) –Key of GT img. Default: 'gt'.

transform(*results*)

Transform function.

参数 **results** (*dict*) –A dict containing the necessary information and data for augmentation.

返回 A dict containing the processed data and information.

返回类型 dict

__repr__()

Return repr(self).

class mmagic.datasets.transforms.**RandomCropLongEdge**(*keys='img'*)

Bases: mmcv.transforms.BaseTransform

Random crop the given image by the long edge.

参数 keys (*list[str]*) –The images to be cropped.

transform(*results*)

Call function.

参数 results (*dict*) –A dict containing the necessary information and data for augmentation.

返回 A dict containing the processed data and information.

返回类型 dict

__repr__()

Return repr(self).

class mmagic.datasets.transforms.**RandomResizedCrop**(*keys*, *crop_size*, *scale=(0.08, 1.0)*, *ratio=(3.0 / 4.0, 4.0 / 3.0)*, *interpolation='bilinear'*)

Bases: mmcv.transforms.BaseTransform

Crop data to random size and aspect ratio.

A crop of a random proportion of the original image and a random aspect ratio of the original aspect ratio is made. The cropped image is finally resized to a given size specified by 'crop_size'. Modified keys are the attributes specified in "keys".

This code is partially adopted from torchvision.transforms.RandomResizedCrop: [https://pytorch.org/vision/stable/_modules/torchvision/transforms/ transforms.html#RandomResizedCrop].

参数

- **keys** (*list[str]*) –The images to be resized and random-cropped.
- **crop_size** (*int | tuple[int]*) –Target spatial size (h, w).
- **scale** (*tuple[float], optional*) –Range of the proportion of the original image to be cropped. Default: (0.08, 1.0).
- **ratio** (*tuple[float], optional*) –Range of aspect ratio of the crop. Default: (3. / 4., 4. / 3.).
- **interpolation** (*str, optional*) –Algorithm used for interpolation. It can be only either one of the following: "nearest" | "bilinear" | "bicubic" | "area" | "lanczos". Default: "bilinear".

get_params(*data*)

Get parameters for a random sized crop.

参数 **data** (*np.ndarray*) –Image of type numpy array to be cropped.

返回 A tuple containing the coordinates of the top left corner and the chosen crop size.

transform(*results*)

Transform function.

参数 **results** (*dict*) –A dict containing the necessary information and data for augmentation.

返回 A dict containing the processed data and information.

返回类型 dict

__repr__()

Return repr(self).

class mmagic.datasets.transforms.**CompositeFg**(*fg_dirs*, *alpha_dirs*, *interpolation='nearest'*)

Bases: mmcv.transforms.BaseTransform

Composite foreground with a random foreground.

This class composites the current training sample with additional data randomly (could be from the same dataset). With probability 0.5, the sample will be composited with a random sample from the specified directory. The composition is performed as:

$$fg_{new} = \alpha_1 * fg_1 + (1 - \alpha_1) * fg_2$$
$$\alpha_{new} = 1 - (1 - \alpha_1) * (1 - \alpha_2)$$

where (fg_1, α_1) is from the current sample and (fg_2, α_2) is the randomly loaded sample. With the above composition, α_{new} is still in *[0, 1]*.

Required keys are "alpha" and "fg". Modified keys are "alpha" and "fg".

参数

- **fg_dirs** (*str | list[str]*) –Path of directories to load foreground images from.
- **alpha_dirs** (*str | list[str]*) –Path of directories to load alpha mattes from.
- **interpolation** (*str*) –Interpolation method of *mmcv.imresize* to resize the randomly loaded images. Default: 'nearest'.

transform(*results: dict*) → dict

Transform function.

参数 **results** (*dict*) –A dict containing the necessary information and data for augmentation.

返回 A dict containing the processed data and information.

返回类型 dict

_get_file_list(*fg_dirs*, *alpha_dirs*)

__repr__()

Return repr(self).

class mmagic.datasets.transforms.**MergeFgAndBg**

Bases: mmcv.transforms.BaseTransform

Composite foreground image and background image with alpha.

Required keys are "alpha", "fg" and "bg", added key is "merged".

transform(*results: dict*) → dict

Transform function.

参数 results (*dict*) –A dict containing the necessary information and data for augmentation.

返回 A dict containing the processed data and information.

返回类型 dict

__repr__() → str

Return repr(self).

class mmagic.datasets.transforms.**PerturbBg**(*gamma_ratio=0.6*)

Bases: mmcv.transforms.BaseTransform

Randomly add gaussian noise or gamma change to background image.

Required key is "bg", added key is "noisy_bg".

参数 gamma_ratio (*float, optional*) –The probability to use gamma correction instead of gaussian noise. Defaults to 0.6.

transform(*results: dict*) → dict

Transform function.

参数 results (*dict*) –A dict containing the necessary information and data for augmentation.

返回 A dict containing the processed data and information.

返回类型 dict

__repr__()

Return repr(self).

class mmagic.datasets.transforms.**RandomJitter**(*hue_range=40*)

Bases: mmcv.transforms.BaseTransform

Randomly jitter the foreground in hsv space.

The jitter range of hue is adjustable while the jitter ranges of saturation and value are adaptive to the images. Side effect: the "fg" image will be converted to *np.float32*. Required keys are "fg" and "alpha", modified key is "fg".

参数 **hue_range** (*float | tuple[float]*) –Range of hue jittering. If it is a float instead of a tuple like (min, max), the range of hue jittering will be (-hue_range, +hue_range). Default: 40.

transform(*results*)

transform function.

参数 **results** (*dict*) –A dict containing the necessary information and data for augmentation.

返回 A dict containing the processed data and information.

返回类型 dict

__repr__()

Return repr(self).

class mmagic.datasets.transforms.**RandomLoadResizeBg**(*bg_dir*, *flag='color'*, *channel_order='bgr'*)

Bases: mmcv.transforms.BaseTransform

Randomly load a background image and resize it.

Required key is "fg", added key is "bg".

参数

- **bg_dir** (*str*) –Path of directory to load background images from.
- **flag** (*str*) –Loading flag for images. Default: 'color'.
- **channel_order** (*str*) –Order of channel, candidates are 'bgr' and 'rgb'. Default: 'bgr'.
- **kwargs** (*dict*) –Args for file client.

transform(*results: dict*) → dict

Transform function.

参数 **results** (*dict*) –A dict containing the necessary information and data for augmentation.

返回 A dict containing the processed data and information.

返回类型 dict

__repr__()

Return repr(self).

class mmagic.datasets.transforms.**PackInputs**(*keys: Tuple[List[str], str] = ['merged', 'img']*, *meta_keys: Tuple[List[str], str] = []*, *data_keys: Tuple[List[str], str] = []*)

Bases: mmcv.transforms.base.BaseTransform

Pack data into DataSample for training, evaluation and testing.

MMagic follows the design of data structure from MMEngine. Data from the loader will be packed into data field of DataSample. More details of DataSample refer to the documentation of MMEngine: https://mmengine.readthedocs.io/en/latest/advanced_tutorials/data_element.html

参数

- **Tuple[List[str]]** (*meta_keys*) –The keys to saved in returned inputs, which are used as the input of models, default to ['img', 'noise', 'merged'].
- **str** –The keys to saved in returned inputs, which are used as the input of models, default to ['img', 'noise', 'merged'].
- **None]** –The keys to saved in returned inputs, which are used as the input of models, default to ['img', 'noise', 'merged'].
- **Tuple[List[str]]** –The keys to saved in *data_field* of the *data_samples*.
- **str** –The keys to saved in *data_field* of the *data_samples*.
- **None]** –The keys to saved in *data_field* of the *data_samples*.
- **Tuple[List[str]]** –The meta keys to saved in *metainfo* of the *data_samples*. All the other data will be packed into the data of the *data_samples*
- **str** –The meta keys to saved in *metainfo* of the *data_samples*. All the other data will be packed into the data of the *data_samples*
- **None]** –The meta keys to saved in *metainfo* of the *data_samples*. All the other data will be packed into the data of the *data_samples*

transform(*results: dict*) → dict

Method to pack the input data.

参数 **results** (*dict*) –Result dict from the data pipeline.

返回

A dict contains

- ' inputs' (obj:*dict*): The forward data of models. According to different tasks, the *inputs* may contain images, videos, labels, text, etc.
- **' data_samples' (obj:*DataSample*): The annotation info of the** sample.

返回类型 dict

__repr__() → str

Return repr(self).

class mmagic.datasets.transforms.**GenerateCoordinateAndCell**(*sample_quantity=None*, *scale=None*, *target_size=None*, *reshape_gt=True*)

Bases: mmcv.transforms.base.BaseTransform

Generate coordinate and cell. Generate coordinate from the desired size of SR image.

Train or val:

1. Generate coordinate from GT.

#. Reshape GT image to (HgWg, 3) and transpose to (3, HgWg). where *Hg* and *Wg* represent the height and width of GT.

Test:

1. Generate coordinate from LQ and scale or target_size.
2. Then generate cell from coordinate.

参数

- **sample_quantity** (*int | None*) –The quantity of samples in coordinates. To ensure that the GT tensors in a batch have the same dimensions. Default: None.
- **scale** (*float*) –Scale of upsampling. Default: None.
- **target_size** (*tuple[int]*) –Size of target image. Default: None.
- **reshape_gt** (*bool*) –Whether reshape gt to (-1, 3). Default: True If sample_quantity is not None, reshape_gt = True.

The priority of getting ‘size of target image’ is:

1. results[‘gt’].shape[-2:]
2. results[‘lq’].shape[-2:] * scale
3. target_size

transform(*results*)

Call function.

参数

- **results** (*Require either in*) –A dict containing the necessary information
- **augmentation.** (*and data for*) –
- **results** –
- **'lq'** (*1.*) –
- **'gt'** (*2.*) –
- **None** (*3.*) –
- **and** (*the premise is self.target_size*) –

- **len** (*self.target_size*) –

返回 A dict containing the processed data and information. Reshape ‘gt’ to (-1, 3) and transpose to (3, -1) if ‘gt’ in results. Add ‘coord’ and ‘cell’.

返回类型 dict

__repr__()

Return repr(self).

class mmagic.datasets.transforms.**GenerateFacialHeatmap**(*image_key*, *ori_size*, *target_size*, *sigma=1.0*, *use_cache=True*)

Bases: mmcv.transforms.base.BaseTransform

Generate heatmap from keypoint.

参数

- **image_key** (*str*) –Key of facial image in dict.
- **ori_size** (*int | Tuple[int]*) –Original image size of keypoint.
- **target_size** (*int | Tuple[int]*) –Target size of heatmap.
- **sigma** (*float*) –Sigma parameter of heatmap. Default: 1.0
- **use_cache** (*bool*) –If True, load all heatmap at once. Default: True.

transform(*results*)

transform function.

参数 **results** (*dict*) –A dict containing the necessary information and data for augmentation. Require keypoint.

返回

A dict containing the processed data and information. Add ‘heatmap’.

返回类型 dict

generate_heatmap_from_img(*image*)

Generate heatmap from img.

参数 **image** (*np.ndarray*) –Face image.

results: heatmap (np.ndarray): Heatmap the face image.

_face_alignment_detector(*image*)

Generate face landmark by face_alignment.

参数 **image** (*np.ndarray*) –Face image.

返回 Location of landmark.

返回类型 landmark (Tuple[float])

_generate_one_heatmap(*keypoint*)

Generate One Heatmap.

参数 **keypoint** (*Tuple[float]*) –Location of a landmark.

results: heatmap (np.ndarray): A heatmap of landmark.

__repr__()

Return repr(self).

class mmagic.datasets.transforms.**GenerateFrameIndices**(*interval_list*, *frames_per_clip=99*)

Bases: mmcv.transforms.BaseTransform

Generate frame index for REDS datasets. It also performs temporal augmentation with random interval.

Required Keys:

- img_path
- gt_path
- key
- num_input_frames

Modified Keys:

- img_path
- gt_path

Added Keys:

- interval
- reverse

参数

- **interval_list** (*list[int]*) –Interval list for temporal augmentation. It will randomly pick an interval from interval_list and sample frame index with the interval.
- **frames_per_clip** (*int*) –Number of frames per clips. Default: 99 for REDS dataset.

transform(*results*)

transform function.

参数 **results** (*dict*) –A dict containing the necessary information and data for augmentation.

返回 A dict containing the processed data and information.

返回类型 dict

__repr__()

Return repr(self).

class mmagic.datasets.transforms.**GenerateFrameIndiceswithPadding**(*padding*, *filename_tmpl='{:08d}'*)

Bases: mmcv.transforms.BaseTransform

Generate frame index with padding for REDS dataset and Vid4 dataset during testing.

Required Keys:

- img_path
- gt_path
- key
- num_input_frames
- sequence_length

Modified Keys:

- img_path
- gt_path

参数 padding –padding mode, one of ‘replicate’ | ‘reflection’ | ‘reflection_circle’ | ‘circle’.

Examples: current_idx = 0, num_input_frames = 5 The generated frame indices under different padding mode:

replicate: [0, 0, 0, 1, 2] reflection: [2, 1, 0, 1, 2] reflection_circle: [4, 3, 0, 1, 2] circle: [3, 4, 0, 1, 2]

transform(*results*)

transform function.

参数 results (*dict*) –A dict containing the necessary information and data for augmentation.

返回 A dict containing the processed data and information.

返回类型 dict

__repr__()

Return repr(self).

class mmagic.datasets.transforms.**GenerateSegmentIndices**(*interval_list*, *start_idx=0*, *filename_tmpl='{:08d}.png'*)

Bases: mmcv.transforms.BaseTransform

Generate frame indices for a segment. It also performs temporal augmentation with random interval.

Required Keys:

- img_path
- gt_path
- key
- num_input_frames
- sequence_length

Modified Keys:

- img_path
- gt_path

Added Keys:

- interval
- reverse

参数

- **interval_list** (*list[int]*) –Interval list for temporal augmentation. It will randomly pick an interval from interval_list and sample frame index with the interval.
- **start_idx** (*int*) –The index corresponds to the first frame in the sequence. Default: 0.
- **filename_tmpl** (*str*) –Template for file name. Default: '{:08d}.png'.

transform(*results*)

transform function.

参数 **results** (*dict*) –A dict containing the necessary information and data for augmentation.

返回 A dict containing the processed data and information.

返回类型 dict

__repr__()

Return repr(self).

class mmagic.datasets.transforms.**GetMaskedImage**(*img_key='gt', mask_key='mask', out_key='img', zero_value=127.5*)

Bases: mmcv.transforms.base.BaseTransform

Get masked image.

参数

- **img_key** (*str*) –Key for clean image. Default: 'gt'.

- **mask_key** (*str*) –Key for mask image. The mask shape should be (h, w, 1) while '1' indicate holes and '0' indicate valid regions. Default: 'mask'.
- **img_key** –Key for output image. Default: 'img'.
- **zero_value** (*float*) –Pixel value of masked area.

transform(*results*)

transform function.

参数 **results** (*dict*) –A dict containing the necessary information and data for augmentation.

返回 A dict containing the processed data and information.

返回类型 dict

__repr__()

Return repr(self).

class mmagic.datasets.transforms.**GetSpatialDiscountMask**(*gamma=0.99*, *beta=1.5*)

Bases: mmcv.transforms.BaseTransform

Get spatial discounting mask constant.

Spatial discounting mask is first introduced in: Generative Image Inpainting with Contextual Attention.

参数

- **gamma** (*float, optional*) –Gamma for computing spatial discounting. Defaults to 0.99.
- **beta** (*float, optional*) –Beta for computing spatial discounting. Defaults to 1.5.

spatial_discount_mask(*mask_width*, *mask_height*)

Generate spatial discounting mask constant.

参数

- **mask_width** (*int*) –The width of bbox hole.
- **mask_height** (*int*) –The height of bbox height.

返回 Spatial discounting mask.

返回类型 np.ndarray

transform(*results*)

transform function.

参数 **results** (*dict*) –A dict containing the necessary information and data for augmentation.

返回 A dict containing the processed data and information.

返回类型 dict

__repr__()

Return repr(self).

class mmagic.datasets.transforms.**LoadImageFromFile**(*key: str*, *color_type: str = 'color'*, *channel_order: str = 'bgr'*, *imdecode_backend: Optional[str] = None*, *use_cache: bool = False*, *to_float32: bool = False*, *to_y_channel: bool = False*, *save_original_img: bool = False*, *backend_args: Optional[dict] = None*)

Bases: mmcv.transforms.BaseTransform

Load a single image or image frames from corresponding paths. Required Keys: - [Key]_path

New Keys: - [KEY] - ori_[KEY]_shape - ori_[KEY]

参数

- **key** (*str*) –Keys in results to find corresponding path.
- **color_type** (*str*) –The flag argument for :func:mmcv.imfrombytes. Defaults to 'color'.
- **channel_order** (*str*) –Order of channel, candidates are 'bgr' and 'rgb'. Default: 'bgr'.
- **imdecode_backend** (*str*) –The image decoding backend type. The backend argument for :func:mmcv.imfrombytes. See :func:mmcv.imfrombytes for details. candidates are 'cv2', 'turbojpeg', 'pillow', and 'tifffile'. Defaults to None.
- **use_cache** (*bool*) –If True, load all images at once. Default: False.
- **to_float32** (*bool*) –Whether to convert the loaded image to a float32 numpy array. If set to False, the loaded image is an uint8 array. Defaults to False.
- **to_y_channel** (*bool*) –Whether to convert the loaded image to y channel. Only support 'rgb2ycbcr' and 'rgb2ycbcr' Defaults to False.
- **backend_args** (*dict, optional*) –Arguments to instantiate the prefix of uri corresponding backend. Defaults to None.

transform(*results: dict*) → dict

Functions to load image or frames.

参数 **results** (*dict*) –Result dict from :obj:mmcv.BaseDataset.

返回 The dict contains loaded image and meta information.

返回类型 dict

_load_image(*filename*)

Load an image from file.

参数 **filename**(*str*) –Path of image file.

返回 Image.

返回类型 np.ndarray

_convert(*img: numpy.ndarray*)

Convert an image to the require format.

参数 **img**(*np.ndarray*) –The original image.

返回 The converted image.

返回类型 np.ndarray

__repr__()

Return repr(self).

class mmagic.datasets.transforms.**LoadMask**(*mask_mode='bbox'*, *mask_config=None*)

Bases: mmcv.transforms.BaseTransform

Load Mask for multiple types.

For different types of mask, users need to provide the corresponding config dict.

Example config for bbox:

```
config = dict(img_shape=(256, 256), max_bbox_shape=128)
```

Example config for irregular:

```
config = dict(
    img_shape=(256, 256),
    num_vertices=(4, 12),
    max_angle=4.,
    length_range=(10, 100),
    brush_width=(10, 40),
    area_ratio_range=(0.15, 0.5))
```

Example config for ff:

```
config = dict(
    img_shape=(256, 256),
    num_vertices=(4, 12),
    mean_angle=1.2,
    angle_range=0.4,
    brush_width=(12, 40))
```

Example config for set:

```
config = dict(
    mask_list_file='xxx/xxx/ooxx.txt',
    prefix='/xxx/xxx/ooxx/',
    io_backend='local',
    color_type='unchanged',
    file_client_kwargs=dict()
)

The mask_list_file contains the list of mask file name like this:
    test1.jpeg
    test2.jpeg
    ...
    ...

The prefix gives the data path.
```

参数

- **mask_mode** (*str*) –Mask mode in [‘bbox’ , ‘irregular’ , ‘ff’ , ‘set’ , ‘file’]. Default: ‘bbox’ . * bbox: square bounding box masks. * irregular: irregular holes. * ff: free-form holes from DeepFillv2. * set: randomly get a mask from a mask set. * file: get mask from ‘mask_path’ in results.
- **mask_config** (*dict*) –Params for creating masks. Each type of mask needs different configs. Default: None.

_init_info()

_get_random_mask_from_set()

_get_mask_from_file(*path*)

transform(*results*)

transform function.

参数 **results** (*dict*) –A dict containing the necessary information and data for augmentation.

返回 A dict containing the processed data and information.

返回类型 dict

__repr__()

Return repr(self).

class mmagic.datasets.transforms.**LoadPairedImageFromFile**(*key: str*, *domain_a: str = 'A'*, *domain_b: str = 'B'*, *color_type: str = 'color'*, *channel_order: str = 'bgr'*, *imdecode_backend: Optional[str] = None*, *use_cache: bool = False*, *to_float32: bool = False*, *to_y_channel: bool = False*, *save_original_img: bool = False*, *backend_args: Optional[dict] = None*)

Bases: *LoadImageFromFile*

Load a pair of images from file.

Each sample contains a pair of images, which are concatenated in the w dimension (a|b). This is a special loading class for generation paired dataset. It loads a pair of images as the common loader does and crops it into two images with the same shape in different domains.

Required key is "pair_path". Added or modified keys are "pair", "pair_ori_shape", "ori_pair", "img_{domain_a}", "img_{domain_b}", "img_{domain_a}_path", "img_{domain_b}_path", "img_{domain_a}_ori_shape", "img_{domain_b}_ori_shape", "ori_img_{domain_a}" and "ori_img_{domain_b}".

参数

- **key** (*str*) –Keys in results to find corresponding path.
- **domain_a** (*str, Optional*) –One of the paired image domain. Defaults to 'A'.
- **domain_b** (*str, Optional*) –The other of the paired image domain. Defaults to 'B'.
- **color_type** (*str*) –The flag argument for :func:mmcv.imfrombytes. Defaults to 'color'.
- **channel_order** (*str*) –Order of channel, candidates are 'bgr' and 'rgb'. Default: 'bgr'.
- **imdecode_backend** (*str*) –The image decoding backend type. The backend argument for :func:mmcv.imfrombytes. See :func:mmcv.imfrombytes for details. candidates are 'cv2', 'turbojpeg', 'pillow', and 'tifffile'. Defaults to None.
- **use_cache** (*bool*) –If True, load all images at once. Default: False.
- **to_float32** (*bool*) –Whether to convert the loaded image to a float32 numpy array. If set to False, the loaded image is an uint8 array. Defaults to False.

- **to_y_channel** (*bool*) –Whether to convert the loaded image to y channel. Only support 'rgb2ycbcr' and 'rgb2ycbcr' Defaults to False.
- **backend_args** (*dict, optional*) –Arguments to instantiate the prefix of uri corresponding backend. Defaults to None.
- **io_backend** (*str, optional*) –io backend where images are store. Defaults to None.

transform(*results: dict*) → dict

Functions to load paired images.

参数 **results** (*dict*) –A dict containing the necessary information and data for augmentation.

返回 A dict containing the processed data and information.

返回类型 dict

class mmagic.datasets.transforms.**MATLABLikeResize**(*keys*, *scale=None*, *output_shape=None*, *kernel='bicubic'*, *kernel_width=4.0*)

Bases: mmcv.transforms.BaseTransform

Resize the input image using MATLAB-like downsampling.

Currently support bicubic interpolation only. Note that the output of this function is slightly different from the official MATLAB function.

Required keys are the keys in attribute "keys". Added or modified keys are "scale" and "output_shape", and the keys in attribute "keys".

参数

- **keys** (*list[str]*) –A list of keys whose values are modified.
- **scale** (*float | None, optional*) –The scale factor of the resize operation. If None, it will be determined by output_shape. Default: None.
- **output_shape** (*tuple(int) | None, optional*) –The size of the output image. If None, it will be determined by scale. Note that if scale is provided, output_shape will not be used. Default: None.
- **kernel** (*str, optional*) –The kernel for the resize operation. Currently support 'bicubic' only. Default: 'bicubic'.
- **kernel_width** (*float*) –The kernel width. Currently support 4.0 only. Default: 4.0.

_resize(*img*)

resize an image to the require size.

参数 **img** (*np.ndarray*) –The original image.

返回 The resized image.

返回类型 output (np.ndarray)

transform(*results*)

transform function.

参数 **results** (*dict*) –A dict containing the necessary information and data for augmentation.

返回 A dict containing the processed data and information.

返回类型 dict

__repr__()

Return repr(self).

class mmagic.datasets.transforms.**Normalize**(*keys*, *mean*, *std*, *to_rgb=False*, *save_original=False*)

Bases: mmcv.transforms.BaseTransform

Normalize images with the given mean and std value.

Required keys are the keys in attribute "keys", added or modified keys are the keys in attribute "keys" and these keys with postfix '_norm_cfg'. It also supports normalizing a list of images.

参数

- **keys** (*Sequence[str]*) –The images to be normalized.
- **mean** (*np.ndarray*) –Mean values of different channels.
- **std** (*np.ndarray*) –Std values of different channels.
- **to_rgb** (*bool*) –Whether to convert channels from BGR to RGB. Default: False.
- **save_original** (*bool*) –Whether to save original images. Default: False.

transform(*results*)

transform function.

参数 **results** (*dict*) –A dict containing the necessary information and data for augmentation.

返回 A dict containing the processed data and information.

返回类型 dict

__repr__()

Return repr(self).

class mmagic.datasets.transforms.**RescaleToZeroOne**(*keys*)

Bases: mmcv.transforms.BaseTransform

Transform the images into a range between 0 and 1.

Required keys are the keys in attribute "keys", added or modified keys are the keys in attribute "keys". It also supports rescaling a list of images.

参数 **keys** (*Sequence[str]*) –The images to be transformed.

transform(*results*)

transform function.

参数 results (*dict*) –A dict containing the necessary information and data for augmentation.

返回 A dict containing the processed data and information.

返回类型 dict

__repr__()

Return repr(self).

class mmagic.datasets.transforms.**DegradationsWithShuffle**(*degradations*, *keys*, *shuffle_idx=None*)

Apply random degradations to input, with degradations being shuffled.

Degradation groups are supported. The order of degradations within the same group is preserved. For example, if we have degradations = [a, b, [c, d]] and shuffle_idx = None, then the possible orders are

```
[a, b, [c, d]]
[a, [c, d], b]
[b, a, [c, d]]
[b, [c, d], a]
[[c, d], a, b]
[[c, d], b, a]
```

Modified keys are the attributed specified in "keys".

参数

- **degradations** (*list[dict]*) –The list of degradations.
- **keys** (*list[str]*) –A list specifying the keys whose values are modified.
- **shuffle_idx** (*list | None, optional*) –The degradations corresponding to these indices are shuffled. If None, all degradations are shuffled. Default: None.

_build_degradations(*degradations*)

__call__(*results*)

__repr__()

Return repr(self).

class mmagic.datasets.transforms.**RandomBlur**(*params*, *keys*)

Apply random blur to the input.

Modified keys are the attributed specified in "keys".

参数

- **params** (*dict*) –A dictionary specifying the degradation settings.

- **keys** (*list[str]*) –A list specifying the keys whose values are modified.

get_kernel(*num_kernels: int*)

This is the function to create kernel.

参数 **num_kernels** (*int*) –the number of kernels

返回 _description_

返回类型 _type_

_apply_random_blur(*imgs*)

This is the function to apply blur operation on images.

参数 **imgs** (*Tensor*) –images

返回 Images applied blur

返回类型 Tensor

__call__(*results*)

__repr__()

Return repr(self).

class mmagic.datasets.transforms.**RandomJPEGCompression**(*params*, *keys*, *color_type='color'*, *bgr2rgb=False*)

Apply random JPEG compression to the input.

Modified keys are the attributed specified in "keys".

参数

- **params** (*dict*) –A dictionary specifying the degradation settings.
- **keys** (*list[str]*) –A list specifying the keys whose values are modified.
- **bgr2rgb** (*str*) –Whether change channel order. Default: False.

_apply_random_compression(*imgs*)

__call__(*results*)

__repr__()

Return repr(self).

class mmagic.datasets.transforms.**RandomNoise**(*params*, *keys*)

Apply random noise to the input.

Currently support Gaussian noise and Poisson noise.

Modified keys are the attributed specified in "keys".

参数

- **params** (*dict*) –A dictionary specifying the degradation settings.
- **keys** (*list[str]*) –A list specifying the keys whose values are modified.

_apply_gaussian_noise(*imgs*)

This is the function used to apply gaussian noise on images.

参数 **imgs** (*Tensor*) –images

返回 images applied gaussian noise

返回类型 Tensor

_apply_poisson_noise(*imgs*)

_apply_random_noise(*imgs*)

This is the function used to apply random noise on images.

参数 **imgs** (*Tensor*) –training images

返回 _description_

返回类型 _type_

__call__(*results*)

__repr__()

Return repr(self).

class mmagic.datasets.transforms.**RandomResize**(*params*, *keys*)

Randomly resize the input.

Modified keys are the attributed specified in "keys".

参数

- **params** (*dict*) –A dictionary specifying the degradation settings.
- **keys** (*list[str]*) –A list specifying the keys whose values are modified.

_random_resize(*imgs*)

This is the function used to randomly resize images for training augmentation.

参数 **imgs** (*Tensor*) –training images.

返回 images after randomly resized

返回类型 Tensor

__call__(*results*)

__repr__()

Return repr(self).

class mmagic.datasets.transforms.**RandomVideoCompression**(*params*, *keys*)

Apply random video compression to the input.

Modified keys are the attributed specified in "keys".

参数

- **params** (*dict*) –A dictionary specifying the degradation settings.
- **keys** (*list[str]*) –A list specifying the keys whose values are modified.

_apply_random_compression(*imgs*)

This is the function to apply random compression on images.

参数 **imgs** (*Tensor*) –training images

返回 images after randomly compressed

返回类型 Tensor

__call__(*results*)

__repr__()

Return repr(self).

class mmagic.datasets.transforms.**RandomDownSampling**(*scale_min=1.0*, *scale_max=4.0*, *patch_size=None*, *interpolation='bicubic'*, *backend='pillow'*)

Bases: mmcv.transforms.BaseTransform

Generate LQ image from GT (and crop), which will randomly pick a scale.

参数

- **scale_min** (*float*) –The minimum of upsampling scale, inclusive. Default: 1.0.
- **scale_max** (*float*) –The maximum of upsampling scale, exclusive. Default: 4.0.
- **patch_size** (*int*) –The cropped lr patch size. Default: None, means no crop.
- **interpolation** (*str*) –Interpolation method, accepted values are "nearest", "bilinear", "bicubic", "area", "lanczos" for 'cv2' backend, "nearest", "bilinear", "bicubic", "box", "lanczos", "hamming" for 'pillow' backend. Default: "bicubic".
- **backend** (*str | None*) –The image resize backend type. Options are *cv2*, *pillow*, *None*. If backend is None, the global imread_backend specified by mmcv.use_backend() will be used. Default: "pillow".
- **[scale_min** (*Scale will be picked in the range of*) –
- **scale_max).** –

transform(*results*)

transform function.

参数 results (*dict*) –A dict containing the necessary information and data for augmentation. 'gt' is required.

返回

A dict containing the processed data and information. modified 'gt', supplement 'lq' and 'scale' to keys.

返回类型 dict

__repr__()

Return repr(self).

class mmagic.datasets.transforms.**FormatTrimap**(*to_onehot=False*)

Bases: mmcv.transforms.BaseTransform

Convert trimap (tensor) to one-hot representation.

It transforms the trimap label from (0, 128, 255) to (0, 1, 2). If to_onehot is set to True, the trimap will convert to one-hot tensor of shape (3, H, W). Required key is "trimap", added or modified key are "trimap" and "format_trimap_to_onehot".

参数 to_onehot (*bool*) –whether convert trimap to one-hot tensor. Default: False.

transform(*results*)

Transform function.

参数 results (*dict*) –A dict containing the necessary information and data for augmentation.

返回 A dict containing the processed data and information.

返回类型 dict

__repr__()

Return repr(self).

class mmagic.datasets.transforms.**GenerateTrimap**(*kernel_size*, *iterations=1*, *random=True*)

Bases: mmcv.transforms.BaseTransform

Using random erode/dilate to generate trimap from alpha matte.

Required key is "alpha", added key is "trimap".

参数

- **kernel_size** (*int | tuple[int]*) –The range of random kernel_size of erode/dilate; int indicates a fixed kernel_size. If *random* is set to False and kernel_size is a tuple of length 2, then it will be interpreted as (erode kernel_size, dilate kernel_size). It should be noted that the kernel of the erosion and dilation has the same height and width.

- **iterations** (*int | tuple[int], optional*) –The range of random iterations of erode/dilate; int indicates a fixed iterations. If *random* is set to False and iterations is a tuple of length 2, then it will be interpreted as (erode iterations, dilate iterations). Default to 1.
- **random** (*bool, optional*) –Whether use random kernel_size and iterations when generating trimap. See *kernel_size* and *iterations* for more information. Default to True.

transform(*results: dict*) → dict

Transform function.

参数 results (*dict*) –A dict containing the necessary information and data for augmentation.

返回 A dict containing the processed data and information.

返回类型 dict

__repr__()

Return repr(self).

class mmagic.datasets.transforms.**GenerateTrimapWithDistTransform**(*dist_thr=20*, *random=True*)

Bases: mmcv.transforms.BaseTransform

Generate trimap with distance transform function.

参数

- **dist_thr** (*int, optional*) –Distance threshold. Area with alpha value between (0, 255) will be considered as initial unknown area. Then area with distance to unknown area smaller than the distance threshold will also be consider as unknown area. Defaults to 20.
- **random** (*bool, optional*) –If True, use random distance threshold from [1, dist_thr). If False, use *dist_thr* as the distance threshold directly. Defaults to True.

transform(*results: dict*) → dict

Transform function.

参数 results (*dict*) –A dict containing the necessary information and data for augmentation.

返回 A dict containing the processed data and information.

返回类型 dict

__repr__()

Return repr(self).

class mmagic.datasets.transforms.**TransformTrimap**

Bases: mmcv.transforms.BaseTransform

Transform trimap into two-channel and six-channel.

This class will generate a two-channel trimap composed of definite foreground and background masks and encode it into a six-channel trimap using Gaussian blurs of the generated two-channel trimap at three different scales. The transformed trimap has 6 channels.

Required key is "trimap", added key is "transformed_trimap" and "two_channel_trimap".

Adopted from the following repository: https://github.com/MarcoForte/FBA_Matting/blob/master/networks/transforms.py.

transform(*results: dict*) → dict

Transform function.

参数 results (*dict*) –A dict containing the necessary information and data for augmentation.

返回 A dict containing the processed data and information.

返回类型 dict

__repr__()

Return repr(self).

class mmagic.datasets.transforms.**CopyValues**(*src_keys*, *dst_keys*)

Bases: mmcv.transforms.BaseTransform

Copy the value of source keys to destination keys.

TODO Change to dict(dst=src)

It does the following: results[dst_key] = results[src_key] for (src_key, dst_key) in zip(src_keys, dst_keys).

Added keys are the keys in the attribute "dst_keys".

Required Keys:

- [SRC_KEYS]

Added Keys:

- [DST_KEYS]

参数

- **src_keys** (*list[str]*) –The source keys.
- **dst_keys** (*list[str]*) –The destination keys.

transform(*results*)

transform function.

参数 results (*dict*) –A dict containing the necessary information and data for augmentation.

返回 A dict with a key added/modified.

返回类型 dict

__repr__()

Return repr(self).

class mmagic.datasets.transforms.**SetValues**(*dictionary*)

Bases: mmcv.transforms.BaseTransform

Set value to destination keys.

It does the following: results[key] = value

Added keys are the keys in the dictionary.

Required Keys:

- None

Added or Modified Keys:

- keys in the dictionary

参数 dictionary (*dict*) –The dictionary to update.

transform(*results: Dict*)

transform function.

参数 results (*dict*) –A dict containing the necessary information and data for augmentation.

返回 A dict with a key added/modified.

返回类型 dict

__repr__()

Return repr(self).

CHAPTER 72

mmagic.evaluation

72.1 Package Contents

72.1.1 Classes

Evaluator	Evaluator for generative models. Unlike high-level vision tasks, metrics
MAE	Mean Absolute Error metric for image.
MSE	Mean Squared Error metric for image.
NIQE	Calculate NIQE (Natural Image Quality Evaluator) metric.
PSNR	Peak Signal-to-Noise Ratio.
SAD	Sum of Absolute Differences metric for image matting.
SNR	Signal-to-Noise Ratio.
SSIM	Calculate SSIM (structural similarity).
ConnectivityError	Connectivity error for evaluating alpha matte prediction.
Equivariance	Metric for generative metrics. Except for the preparation phase
FrechetInceptionDistance	FID metric. In this metric, we calculate the distance between real
GradientError	Gradient error for evaluating alpha matte prediction.
InceptionScore	IS (Inception Score) metric. The images are split into groups, and the
MattingMSE	Mean Squared Error metric for image matting.
MultiScaleStructureSimilarity	MS-SSIM (Multi-Scale Structure Similarity) metric.
PerceptualPathLength	Perceptual path length.
PrecisionAndRecall	Improved Precision and recall metric.

72.1.2 Functions

gauss_gradient(img, sigma)	Gaussian gradient.

class mmagic.evaluation.**Evaluator**(*metrics: Union[dict, mmengine.evaluator.BaseMetric, Sequence]*)

Bases: mmengine.evaluator.Evaluator

Evaluator for generative models. Unlike high-level vision tasks, metrics for generative models have various input types. For example, Inception Score (IS, *InceptionScore*) only needs to take fake images as input. However, Frechet Inception Distance (FID, *FrechetInceptionDistance*) needs to take both real images and fake images as input, and the numbers of real images and fake images can be set arbitrarily. For Perceptual path length (PPL, *PerceptualPathLength*), generator need to sample images along a latent path.

In order to be compatible with different metrics, we designed two critical functions, *prepare_metrics()* and *prepare_samplers()* to support those requirements.

- *prepare_metrics()* set the image images' color order and pass the dataloader to all metrics. Therefore metrics need pre-processing to prepare the corresponding feature.
- *prepare_samplers()* pass the dataloader and model to the metrics, and get the corresponding sampler of each kind of metrics. Metrics with same sample mode can share the sampler.

The whole evaluation process can be found in *mmagic.engine.runner.MultiValLoop.run()* and *mmagic.engine.runner.MultiTestLoop.run()*.

参数 **metrics**(*dict or BaseMetric or Sequence*) –The config of metrics.

prepare_metrics(*module: mmengine.model.BaseModel*, *dataloader: torch.utils.data.dataloader.DataLoader*)

Prepare for metrics before evaluation starts. Some metrics use pretrained model to extract feature. Some metrics use pretrained model to extract feature and input channel order may vary among those models. Therefore, we first parse the output color order from data preprocessor and set the color order for each metric. Then we pass the dataloader to each metrics to prepare pre-calculated items. (e.g. inception feature of the real images). If metric has no pre-calculated items, `metric.prepare()` will be ignored. Once the function has been called, `self.is_ready` will be set as *True*. If `self.is_ready` is *True*, this function will directly return to avoid duplicate computation.

参数

- **module**(*BaseModel*) –Model to evaluate.
- **dataloader**(*DataLoader*) –The dataloader for real images.

static _cal_metric_hash(*metric: mmagic.evaluation.metrics.base_gen_metric.GenMetric*)

Calculate a unique hash value based on the *SAMPLER_MODE* and *sample_model*.

prepare_samplers(*module: mmengine.model.BaseModel*, *dataloader: torch.utils.data.dataloader.DataLoader*) → List[Tuple[List[mmengine.evaluator.BaseMetric], Iterator]]

Prepare for the sampler for metrics whose sampling mode are different. For generative models, different metric need image generated with different inputs. For example, FID, KID and IS need images generated with random noise, and PPL need paired images on the specific noise interpolation path. Therefore, we first group metrics with respect to their sampler's mode (refers to :attr:~'GenMetrics.SAMPLER_MODE'), and build a shared sampler for each metric group. To be noted that, the length of the shared sampler depends on the metric of the most images required in each group.

参数

- **module** (*BaseModel*) –Model to evaluate. Some metrics (e.g. PPL) require *module* in their sampler.
- **dataloader** (*DataLoader*) –The dataloader for real image.

返回

A list of "metrics-shared sampler" pair.

返回类型 List[Tuple[List[BaseMetric], Iterator]]

process(*data_samples: Sequence[*mmagic.structures.DataSample*]*, *data_batch: Optional[Any]*, *metrics: Sequence[mmengine.evaluator.BaseMetric]*) → None

Pass *data_batch* from dataloader and *predictions* (generated results) to corresponding *metrics*.

参数

- **data_samples** (*Sequence[DataSample]*) –A batch of generated results from model.
- **data_batch** (*Optional[Any]*) –A batch of data from the metrics specific sampler or the dataloader.
- **metrics** (*Optional[Sequence[BaseMetric]]*) –Metrics to evaluate.

evaluate() → dict

Invoke `evaluate` method of each metric and collect the metrics dictionary. Different from *Evaluator.evaluate*, this function does not take *size* as input, and elements in *self.metrics* will call their own *evaluate* method to calculate the metric.

返回

Evaluation results of all metrics. The keys are the names of the metrics, and the values are corresponding results.

返回类型 dict

mmagic.evaluation.**gauss_gradient**(*img*, *sigma*)

Gaussian gradient.

From https://www.mathworks.com/matlabcentral/mlc-downloads/downloads/ submissions/8060/versions/2/previews/gaussgradient/gaussgradient.m/ index.html

参数

- **img** (*np.ndarray*) –Input image.
- **sigma** (*float*) –Standard deviation of the gaussian kernel.

返回 Gaussian gradient of input *img*.

返回类型 np.ndarray

class mmagic.evaluation.**MAE**(*gt_key: str = 'gt_img', pred_key: str = 'pred_img', mask_key: Optional[str] = None, scaling=1, device='cpu', collect_device: str = 'cpu', prefix: Optional[str] = None*)

Bases: mmagic.evaluation.metrics.base_sample_wise_metric.BaseSampleWiseMetric

Mean Absolute Error metric for image.

mean(abs(a-b))

参数

- **gt_key** (*str*) –Key of ground-truth. Default: 'gt_img'
- **pred_key** (*str*) –Key of prediction. Default: 'pred_img'
- **mask_key** (*str, optional*) –Key of mask, if mask_key is None, calculate all regions. Default: None
- **collect_device** (*str*) –Device name used for collecting results from different ranks during distributed training. Must be 'cpu' or 'gpu'. Defaults to 'cpu'.
- **prefix** (*str, optional*) –The prefix that will be added in the metric names to disambiguate homonymous metrics of different evaluators. If prefix is not provided in the argument, self.default_prefix will be used instead. Default: None

Metrics:

- MAE (float): Mean of Absolute Error

metric = 'MAE'

process_image(*gt*, *pred*, *mask*)

Process an image.

参数

- **gt** (*Tensor | np.ndarray*) –GT image.
- **pred** (*Tensor | np.ndarray*) –Pred image.
- **mask** (*Tensor | np.ndarray*) –Mask of evaluation.

返回 MAE result.

返回类型 result (np.ndarray)

class mmagic.evaluation.**MSE**(*gt_key: str = 'gt_img'*, *pred_key: str = 'pred_img'*, *mask_key: Optional[str] = None*, *scaling=1*, *device='cpu'*, *collect_device: str = 'cpu'*, *prefix: Optional[str] = None*)

Bases: mmagic.evaluation.metrics.base_sample_wise_metric.BaseSampleWiseMetric

Mean Squared Error metric for image.

mean((a-b)^2)

参数

- **gt_key** (*str*) –Key of ground-truth. Default: 'gt_img'
- **pred_key** (*str*) –Key of prediction. Default: 'pred_img'
- **mask_key** (*str, optional*) –Key of mask, if mask_key is None, calculate all regions. Default: None
- **collect_device** (*str*) –Device name used for collecting results from different ranks during distributed training. Must be 'cpu' or 'gpu'. Defaults to 'cpu'.
- **prefix** (*str, optional*) –The prefix that will be added in the metric names to disambiguate homonymous metrics of different evaluators. If prefix is not provided in the argument, self.default_prefix will be used instead. Default: None

Metrics:

- MSE (float): Mean of Squared Error

metric = 'MSE'

process_image(*gt*, *pred*, *mask*)

Process an image.

参数

- **gt** (*Torch | np.ndarray*) –GT image.
- **pred** (*Torch | np.ndarray*) –Pred image.
- **mask** (*Torch | np.ndarray*) –Mask of evaluation.

返回 MSE result.

返回类型 result (np.ndarray)

class mmagic.evaluation.**NIQE**(*key: str = 'pred_img'*, *is_predicted: bool = True*, *collect_device: str = 'cpu'*, *prefix: Optional[str] = None*, *crop_border=0*, *input_order='HWC'*, *convert_to='gray'*)

Bases: mmagic.evaluation.metrics.base_sample_wise_metric.BaseSampleWiseMetric

Calculate NIQE (Natural Image Quality Evaluator) metric.

Ref: Making a "Completely Blind" Image Quality Analyzer. This implementation could produce almost the same results as the official MATLAB codes: http://live.ece.utexas.edu/research/quality/niqe_release.zip

We use the official params estimated from the pristine dataset. We use the recommended block size (96, 96) without overlaps.

参数

- **key** (*str*) –Key of image. Default: 'pred_img'
- **is_predicted** (*bool*) –If the image is predicted, it will be picked from predictions; otherwise, it will be picked from data_batch. Default: True
- **collect_device** (*str*) –Device name used for collecting results from different ranks during distributed training. Must be 'cpu' or 'gpu'. Defaults to 'cpu'.
- **prefix** (*str, optional*) –The prefix that will be added in the metric names to disambiguate homonymous metrics of different evaluators. If prefix is not provided in the argument, self.default_prefix will be used instead. Default: None
- **crop_border** (*int*) –Cropped pixels in each edges of an image. These pixels are not involved in the PSNR calculation. Default: 0.
- **input_order** (*str*) –Whether the input order is 'HWC' or 'CHW'. Default: 'HWC'.
- **convert_to** (*str*) –Whether to convert the images to other color models. If None, the images are not altered. When computing for 'Y', the images are assumed to be in BGR order. Options are 'Y' and None. Default: 'gray'.

Metrics:

- NIQE (float): Natural Image Quality Evaluator

metric = 'NIQE'

process_image(*gt*, *pred*, *mask*) → None

Process an image.

参数

- **gt** (*np.ndarray*) –GT image.
- **pred** (*np.ndarray*) –Pred image.
- **mask** (*np.ndarray*) –Mask of evaluation.

返回 NIQE result.

返回类型 result (np.ndarray)

class mmagic.evaluation.**PSNR**(*gt_key: str = 'gt_img'*, *pred_key: str = 'pred_img'*, *collect_device: str = 'cpu'*, *prefix: Optional[str] = None*, *crop_border=0*, *input_order='CHW'*, *convert_to=None*)

Bases: mmagic.evaluation.metrics.base_sample_wise_metric.BaseSampleWiseMetric

Peak Signal-to-Noise Ratio.

Ref: https://en.wikipedia.org/wiki/Peak_signal-to-noise_ratio

参数

- **gt_key** (*str*) –Key of ground-truth. Default: 'gt_img'
- **pred_key** (*str*) –Key of prediction. Default: 'pred_img'
- **collect_device** (*str*) –Device name used for collecting results from different ranks during distributed training. Must be 'cpu' or 'gpu'. Defaults to 'cpu'.
- **prefix** (*str, optional*) –The prefix that will be added in the metric names to disambiguate homonymous metrics of different evaluators. If prefix is not provided in the argument, self.default_prefix will be used instead. Default: None
- **crop_border** (*int*) –Cropped pixels in each edges of an image. These pixels are not involved in the PSNR calculation. Default: 0.
- **input_order** (*str*) –Whether the input order is 'HWC' or 'CHW'. Default: 'CHW'.
- **convert_to** (*str*) –Whether to convert the images to other color models. If None, the images are not altered. When computing for 'Y', the images are assumed to be in BGR order. Options are 'Y' and None. Default: None.

Metrics:

- PSNR (float): Peak Signal-to-Noise Ratio

metric = 'PSNR'

process_image(*gt*, *pred*, *mask*)

Process an image.

参数

- **gt** (*Torch | np.ndarray*) –GT image.
- **pred** (*Torch | np.ndarray*) –Pred image.
- **mask** (*Torch | np.ndarray*) –Mask of evaluation.

返回 PSNR result.

返回类型 np.ndarray

class mmagic.evaluation.**SAD**(*norm_const=1000*, ***kwargs*)

Bases: mmagic.evaluation.metrics.base_sample_wise_metric.BaseSampleWiseMetric

Sum of Absolute Differences metric for image matting.

This metric compute per-pixel absolute difference and sum across all pixels. i.e. sum(abs(a-b)) / norm_const

---

备注： Current implementation assume image / alpha / trimap array in numpy format and with pixel value ranging from 0 to 255.

备注： pred_alpha should be masked by trimap before passing into this metric

---

Default prefix: ‘’

参数 **norm_const** (*int*) –Divide the result to reduce its magnitude. Default to 1000.

Metrics:

- SAD (float): Sum of Absolute Differences

default_prefix = ''

metric = 'SAD'

prepare(*module: torch.nn.Module*, *dataloader: torch.utils.data.dataloader.DataLoader*)

process(*data_batch: Sequence[dict]*, *data_samples: Sequence[dict]*) → None

Process one batch of data and predictions.

参数

- **data_batch** (*Sequence[Tuple[Any, dict]]*) –A batch of data from the dataloader.
- **predictions** (*Sequence[dict]*) –A batch of outputs from the model.

compute_metrics(*results: List*)

Compute the metrics from processed results.

参数 **results** (*dict*) –The processed results of each batch.

返回 The computed metrics. The keys are the names of the metrics, and the values are corresponding results.

返回类型 Dict

class mmagic.evaluation.**SNR**(*gt_key: str = 'gt_img'*, *pred_key: str = 'pred_img'*, *collect_device: str = 'cpu'*, *prefix: Optional[str] = None*, *crop_border=0*, *input_order='CHW'*, *convert_to=None*)

Bases: mmagic.evaluation.metrics.base_sample_wise_metric.BaseSampleWiseMetric

Signal-to-Noise Ratio.

Ref: https://en.wikipedia.org/wiki/Signal-to-noise_ratio

参数

- **gt_key** (*str*) –Key of ground-truth. Default: 'gt_img'
- **pred_key** (*str*) –Key of prediction. Default: 'pred_img'
- **collect_device** (*str*) –Device name used for collecting results from different ranks during distributed training. Must be 'cpu' or 'gpu'. Defaults to 'cpu'.
- **prefix** (*str, optional*) –The prefix that will be added in the metric names to disambiguate homonymous metrics of different evaluators. If prefix is not provided in the argument, self.default_prefix will be used instead. Default: None
- **crop_border** (*int*) –Cropped pixels in each edges of an image. These pixels are not involved in the SNR calculation. Default: 0.
- **input_order** (*str*) –Whether the input order is 'HWC' or 'CHW'. Default: 'CHW'.
- **convert_to** (*str*) –Whether to convert the images to other color models. If None, the images are not altered. When computing for 'Y', the images are assumed to be in BGR order. Options are 'Y' and None. Default: None.

Metrics:

- SNR (float): Signal-to-Noise Ratio

metric = 'SNR'

process_image(*gt*, *pred*, *mask*)

Process an image.

参数

- **gt** (*Torch | np.ndarray*) –GT image.
- **pred** (*Torch | np.ndarray*) –Pred image.
- **mask** (*Torch | np.ndarray*) –Mask of evaluation.

返回 SNR result.

返回类型 np.ndarray

class mmagic.evaluation.**SSIM**(*gt_key: str = 'gt_img'*, *pred_key: str = 'pred_img'*, *collect_device: str = 'cpu'*, *prefix: Optional[str] = None*, *crop_border=0*, *input_order='CHW'*, *convert_to=None*)

Bases: mmagic.evaluation.metrics.base_sample_wise_metric.BaseSampleWiseMetric

Calculate SSIM (structural similarity).

Ref: Image quality assessment: From error visibility to structural similarity

The results are the same as that of the official released MATLAB code in https://ece.uwaterloo.ca/~z70wang/research/ssim/.

For three-channel images, SSIM is calculated for each channel and then averaged.

参数

- **gt_key** (*str*) –Key of ground-truth. Default: 'gt_img'
- **pred_key** (*str*) –Key of prediction. Default: 'pred_img'
- **collect_device** (*str*) –Device name used for collecting results from different ranks during distributed training. Must be 'cpu' or 'gpu'. Defaults to 'cpu'.
- **prefix** (*str, optional*) –The prefix that will be added in the metric names to disambiguate homonymous metrics of different evaluators. If prefix is not provided in the argument, self.default_prefix will be used instead. Default: None
- **crop_border** (*int*) –Cropped pixels in each edges of an image. These pixels are not involved in the PSNR calculation. Default: 0.
- **input_order** (*str*) –Whether the input order is 'HWC' or 'CHW'. Default: 'HWC'.
- **convert_to** (*str*) –Whether to convert the images to other color models. If None, the images are not altered. When computing for 'Y', the images are assumed to be in BGR order. Options are 'Y' and None. Default: None.

Metrics:

- SSIM (float): Structural similarity

metric = 'SSIM'

process_image(*gt*, *pred*, *mask*)

Process an image.

参数

- **gt** (*Torch | np.ndarray*) –GT image.
- **pred** (*Torch | np.ndarray*) –Pred image.
- **mask** (*Torch | np.ndarray*) –Mask of evaluation.

返回 SSIM result.

返回类型 np.ndarray

class mmagic.evaluation.**ConnectivityError**(*step=0.1*, *norm_constant=1000*, ***kwargs*)

Bases: mmagic.evaluation.metrics.base_sample_wise_metric.BaseSampleWiseMetric

Connectivity error for evaluating alpha matte prediction.

备注: Current implementation assume image / alpha / trimap array in numpy format and with pixel value ranging from 0 to 255.

备注: pred_alpha should be masked by trimap before passing into this metric

---

参数

- **step** (*float*) –Step of threshold when computing intersection between *alpha* and *pred_alpha*. Default to 0.1 .
- **norm_const** (*int*) –Divide the result to reduce its magnitude. Default to 1000.

Default prefix:

Metrics:

- ConnectivityError (float): Connectivity Error

metric = 'ConnectivityError'

prepare(*module: torch.nn.Module*, *dataloader: torch.utils.data.dataloader.DataLoader*)

process(*data_batch: Sequence[dict]*, *data_samples: Sequence[dict]*) → None

Process one batch of data samples and predictions. The processed results should be stored in self.results, which will be used to compute the metrics when all batches have been processed.

参数

- **data_batch** (*Sequence[dict]*) –A batch of data from the dataloader.
- **predictions** (*Sequence[dict]*) –A batch of outputs from the model.

compute_metrics (*results: List*)

Compute the metrics from processed results.

参数 **results** (*dict*) –The processed results of each batch.

返回 The computed metrics. The keys are the names of the metrics, and the values are corresponding results.

返回类型 Dict

class mmagic.evaluation.**Equivariance**(*fake_nums: int*, *real_nums: int = 0*, *fake_key: Optional[str] = None*, *real_key: Optional[str] = 'gt_img'*, *need_cond_input: bool = False*, *sample_mode: str = 'ema'*, *sample_kwargs: dict = dict()*, *collect_device: str = 'cpu'*, *prefix: Optional[str] = None*, *eq_cfg=dict()*)

Bases: mmagic.evaluation.metrics.base_gen_metric.GenerativeMetric

Metric for generative metrics. Except for the preparation phase (prepare()), generative metrics do not need extra real images.

参数

- **fake_nums** (*int*) –Numbers of the generated image need for the metric.
- **real_nums** (*int*) –Numbers of the real image need for the metric. If *-1* is passed means all images from the dataset is need. Defaults to 0.
- **fake_key** (*Optional[str]*) –Key for get fake images of the output dict. Defaults to None.
- **real_key** (*Optional[str]*) –Key for get real images from the input dict. Defaults to 'img'.
- **need_cond_input** (*bool*) –If true, the sampler will return the conditional input randomly sampled from the original dataset. This require the dataset implement *get_data_info* and field *gt_label* must be contained in the return value of *get_data_info*. Noted that, for unconditional models, set *need_cond_input* as True may influence the result of evaluation results since the conditional inputs are sampled from the dataset distribution; otherwise will be sampled from the uniform distribution. Defaults to False.
- **sample_model** (*str*) –Sampling mode for the generative model. Support 'orig' and 'ema'. Defaults to 'ema'.
- **collect_device** (*str*) –Device name used for collecting results from different ranks during distributed training. Must be 'cpu' or 'gpu'. Defaults to 'cpu'.

- **prefix** (*str, optional*) –The prefix that will be added in the metric names to disambiguate homonymous metrics of different evaluators. If prefix is not provided in the argument, self.default_prefix will be used instead. Default to None.
- **sample_kwargs** (*dict*) –Sampling arguments for model test.

name = 'Equivariance'

process(*data_batch: dict*, *data_samples: Sequence[dict]*) → None

Process one batch of data samples and predictions. The processed results should be stored in `self.fake_results`, which will be used to compute the metrics when all batches have been processed.

参数

- **data_batch** (*dict*) –A batch of data from the dataloader.
- **data_samples** (*Sequence[dict]*) –A batch of outputs from the model.

get_metric_sampler(*model: torch.nn.Module*, *dataloader: torch.utils.data.dataloader.DataLoader*, *metrics: List[mmagic.evaluation.metrics.base_gen_metric.GenerativeMetric]*)

Get sampler for generative metrics. Returns a dummy iterator, whose return value of each iteration is a dict containing batch size and sample mode to generate images.

参数

- **model** (*nn.Module*) –Model to evaluate.
- **dataloader** (*DataLoader*) –Dataloader for real images. Used to get batch size during generate fake images.
- **metrics** (*List['GenerativeMetric']*) –Metrics with the same sampler mode.

返回 Sampler for generative metrics.

返回类型 dummy_iterator

compute_metrics(*results*) → dict

Compute the metrics from processed results.

参数 **results** (*list*) –The processed results of each batch.

返回 The computed metrics. The keys are the names of the metrics, and the values are corresponding results.

返回类型 dict

_collect_target_results(*target: str*) → Optional[list]

Collect function for Eq metric. This function support collect results typing as Dict[List[Tensor]]‘.

参数 **target** (*str*) –Target results to collect.

返回 The collected results.

返回类型 Optional[list]

class mmagic.evaluation.**FrechetInceptionDistance**(*fake_nums: int*, *real_nums: int = - 1*, *inception_style='StyleGAN'*, *inception_path: Optional[str] = None*, *inception_pkl: Optional[str] = None*, *fake_key: Optional[str] = None*, *real_key: Optional[str] = 'gt_img'*, *need_cond_input: bool = False*, *sample_model: str = 'orig'*, *collect_device: str = 'cpu'*, *prefix: Optional[str] = None*, *sample_kwargs: dict = dict()*)

Bases: mmagic.evaluation.metrics.base_gen_metric.GenerativeMetric

FID metric. In this metric, we calculate the distance between real distributions and fake distributions. The distributions are modeled by the real samples and fake samples, respectively. *Inception_v3* is adopted as the feature extractor, which is widely used in StyleGAN and BigGAN.

参数

- **fake_nums** (*int*) –Numbers of the generated image need for the metric.
- **real_nums** (*int*) –Numbers of the real images need for the metric. If -1 is passed, means all real images in the dataset will be used. Defaults to -1.
- **inception_style** (*str*) –The target inception style want to load. If the given style cannot be loaded successful, will attempt to load a valid one. Defaults to ‘StyleGAN’.
- **inception_path** (*str, optional*) –Path the the pretrain Inception network. Defaults to None.
- **inception_pkl** (*str, optional*) –Path to reference inception pickle file. If *None*, the statistical value of real distribution will be calculated at running time. Defaults to None.
- **fake_key** (*Optional[str]*) –Key for get fake images of the output dict. Defaults to None.
- **real_key** (*Optional[str]*) –Key for get real images from the input dict. Defaults to ‘img’.
- **need_cond_input** (*bool*) –If true, the sampler will return the conditional input randomly sampled from the original dataset. This require the dataset implement *get_data_info* and field *gt_label* must be contained in the return value of *get_data_info*. Noted that, for unconditional models, set *need_cond_input* as True may influence the result of evaluation results since the conditional inputs are sampled from the dataset distribution; otherwise will be sampled from the uniform distribution. Defaults to False.
- **sample_model** (*str*) –Sampling mode for the generative model. Support ‘orig’ and ‘ema’. Defaults to ‘orig’.
- **collect_device** (*str, optional*) –Device name used for collecting results from different ranks during distributed training. Must be ‘cpu’ or ‘gpu’. Defaults to ‘cpu’.

- **prefix** (*str, optional*) –The prefix that will be added in the metric names to disambiguate homonymous metrics of different evaluators. If prefix is not provided in the argument, self.default_prefix will be used instead. Defaults to None.

name = 'FID'

prepare(*module: torch.nn.Module*, *dataloader: torch.utils.data.dataloader.DataLoader*) → None

Preparing inception feature for the real images.

参数

- **module** (*nn.Module*) –The model to evaluate.
- **dataloader** (*DataLoader*) –The dataloader for real images.

_load_inception(*inception_style: str*, *inception_path: Optional[str]*) → Tuple[torch.nn.Module, str]

Load inception and return the successful loaded style.

参数

- **inception_style** (*str*) –Target style of Inception network want to load.
- **inception_path** (*Optional[str]*) –The path to the inception.

返回

The actually loaded inception network and corresponding style.

返回类型 Tuple[nn.Module, str]

forward_inception(*image: torch.Tensor*) → torch.Tensor

Feed image to inception network and get the output feature.

参数 **data_samples** (*Sequence[dict]*) –A batch of data sample dict used to extract inception feature.

返回 Image feature extracted from inception.

返回类型 Tensor

process(*data_batch: dict*, *data_samples: Sequence[dict]*) → None

Process one batch of data samples and predictions. The processed results should be stored in `self.fake_results`, which will be used to compute the metrics when all batches have been processed.

参数

- **data_batch** (*dict*) –A batch of data from the dataloader.
- **data_samples** (*Sequence[dict]*) –A batch of outputs from the model.

static _calc_fid(*sample_mean: numpy.ndarray*, *sample_cov: numpy.ndarray*, *real_mean: numpy.ndarray*, *real_cov: numpy.ndarray*, *eps: float = 1e-06*) → Tuple[float]

Refer to the implementation from:

https://github.com/rosinality/stylegan2-pytorch/blob/master/fid.py#L34

compute_metrics(*fake_results: list*) → dict

Compute the result of FID metric.

参数 **fake_results** (*list*) –List of image feature of fake images.

返回

A dict of the computed FID metric and its mean and covariance.

返回类型 dict

class mmagic.evaluation.**GradientError**(*sigma=1.4*, *norm_constant=1000*, ***kwargs*)

Bases: mmagic.evaluation.metrics.base_sample_wise_metric.BaseSampleWiseMetric

Gradient error for evaluating alpha matte prediction.

备注： Current implementation assume image / alpha / trimap array in numpy format and with pixel value ranging from 0 to 255.

备注： pred_alpha should be masked by trimap before passing into this metric

---

参数

- **sigma** (*float*) –Standard deviation of the gaussian kernel. Defaults to 1.4 .
- **norm_const** (*int*) –Divide the result to reduce its magnitude. Defaults to 1000 .

Default prefix:

Metrics:

- GradientError (float): Gradient Error

metric = 'GradientError'

prepare(*module: torch.nn.Module*, *dataloader: torch.utils.data.dataloader.DataLoader*)

process(*data_batch: Sequence[dict]*, *data_samples: Sequence[dict]*) → None

Process one batch of data samples and predictions. The processed results should be stored in self.results, which will be used to compute the metrics when all batches have been processed.

参数

- **data_batch** (*Sequence[dict]*) –A batch of data from the dataloader.
- **predictions** (*Sequence[dict]*) –A batch of outputs from the model.

compute_metrics(*results: List*)

Compute the metrics from processed results.

参数 **results**(*dict*) –The processed results of each batch.

返回 The computed metrics. The keys are the names of the metrics, and the values are corresponding results.

返回类型 Dict

class mmagic.evaluation.**InceptionScore**(*fake_nums: int = 50000.0*, *resize: bool = True*, *splits: int = 10*, *inception_style: str = 'StyleGAN'*, *inception_path: Optional[str] = None*, *resize_method='bicubic'*, *use_pillow_resize: bool = True*, *fake_key: Optional[str] = None*, *need_cond_input: bool = False*, *sample_model='orig'*, *collect_device: str = 'cpu'*, *prefix: str = None*)

Bases: mmagic.evaluation.metrics.base_gen_metric.GenerativeMetric

IS (Inception Score) metric. The images are split into groups, and the inception score is calculated on each group of images, then the mean and standard deviation of the score is reported. The calculation of the inception score on a group of images involves first using the inception v3 model to calculate the conditional probability for each image (p(ylx)). The marginal probability is then calculated as the average of the conditional probabilities for the images in the group (p(y)). The KL divergence is then calculated for each image as the conditional probability multiplied by the log of the conditional probability minus the log of the marginal probability. The KL divergence is then summed over all images and averaged over all classes and the exponent of the result is calculated to give the final score.

Ref: https://github.com/sbarratt/inception-score-pytorch/blob/master/inception_score.py # noqa

Note that we highly recommend that users should download the Inception V3 script module from the following address. Then, the *inception_pkl* can be set with user's local path. If not given, we will use the Inception V3 from pytorch model zoo. However, this may bring significant different in the final results.

Tero's Inception V3: https://nvlabs-fi-cdn.nvidia.com/stylegan2-ada-pytorch/pretrained/metrics/inception-2015-12-05.pt # noqa

参数

- **fake_nums**(*int*) –Numbers of the generated image need for the metric.
- **resize**(*bool, optional*) –Whether resize image to 299x299. Defaults to True.
- **splits**(*int, optional*) –The number of groups. Defaults to 10.
- **inception_style**(*str*) –The target inception style want to load. If the given style cannot be loaded successful, will attempt to load a valid one. Defaults to 'StyleGAN'.
- **inception_path**(*str, optional*) –Path the the pretrain Inception network. Defaults to None.

- **resize_method** (*str*) –Resize method. If *resize* is False, this will be ignored. Defaults to 'bicubic'.
- **use_pil_resize** (*bool*) –Whether use Bicubic interpolation with Pillow's backend. If set as True, the evaluation process may be a little bit slow, but achieve a more accurate IS result. Defaults to False.
- **fake_key** (*Optional[str]*) –Key for get fake images of the output dict. Defaults to None.
- **need_cond_input** (*bool*) –If true, the sampler will return the conditional input randomly sampled from the original dataset. This require the dataset implement *get_data_info* and field *gt_label* must be contained in the return value of *get_data_info*. Noted that, for unconditional models, set *need_cond_input* as True may influence the result of evaluation results since the conditional inputs are sampled from the dataset distribution; otherwise will be sampled from the uniform distribution. Defaults to False.
- **sample_model** (*str*) –Sampling mode for the generative model. Support 'orig' and 'ema'. Defaults to 'orig'.
- **collect_device** (*str, optional*) –Device name used for collecting results from different ranks during distributed training. Must be 'cpu' or 'gpu'. Defaults to 'cpu'.
- **prefix** (*str, optional*) –The prefix that will be added in the metric names to disambiguate homonymous metrics of different evaluators. If prefix is not provided in the argument, self.default_prefix will be used instead. Defaults to None.

name = 'IS'

pil_resize_method_mapping

prepare(*module: torch.nn.Module*, *dataloader: torch.utils.data.dataloader.DataLoader*) → None

Prepare for the pre-calculating items of the metric. Defaults to do nothing.

参数

- **module** (*nn.Module*) –Model to evaluate.
- **dataloader** (*DataLoader*) –Dataloader for the real images.

_load_inception(*inception_style: str*, *inception_path: Optional[str]*) → Tuple[torch.nn.Module, str]

Load pretrain model of inception network. :param inception_style: Target style of Inception network want to

load.

参数 **inception_path** (*Optional[str]*) –The path to the inception.

返回

The actually loaded inception network and corresponding style.

返回类型 Tuple[nn.Module, str]

_preprocess(*image: torch.Tensor*) → torch.Tensor

Preprocess image before pass to the Inception. Preprocess operations contain channel conversion and resize.

参数 **image** (*Tensor*) –Image tensor before preprocess.

返回

Image tensor after resize and channel conversion (if need.)

返回类型 Tensor

process(*data_batch: dict*, *data_samples: Sequence[dict]*) → None

Process one batch of data samples and predictions. The processed results should be stored in `self.fake_results`, which will be used to compute the metrics when all batches have been processed.

参数

- **data_batch** (*dict*) –A batch of data from the dataloader.
- **data_samples** (*Sequence[dict]*) –A batch of outputs from the model.

compute_metrics(*fake_results: list*) → dict

Compute the results of Inception Score metric.

参数 **fake_results** (*list*) –List of image feature of fake images.

返回 A dict of the computed IS metric and its standard error

返回类型 dict

class mmagic.evaluation.**MattingMSE**(*norm_const=1000*, ***kwargs*)

Bases: mmagic.evaluation.metrics.base_sample_wise_metric.BaseSampleWiseMetric

Mean Squared Error metric for image matting.

This metric compute per-pixel squared error average across all pixels. i.e. mean((a-b)^2) / norm_const

---

备注: Current implementation assume image / alpha / trimap array in numpy format and with pixel value ranging from 0 to 255.

备注: pred_alpha should be masked by trimap before passing into this metric

---

Default prefix:

参数 **norm_const** (*int*) –Divide the result to reduce its magnitude. Default to 1000.

Metrics:

- MattingMSE (float): Mean of Squared Error

default_prefix = ''

metric = 'MattingMSE'

prepare(*module: torch.nn.Module*, *dataloader: torch.utils.data.dataloader.DataLoader*)

process(*data_batch: Sequence[dict]*, *data_samples: Sequence[dict]*) → None

Process one batch of data and predictions.

参数

- **data_batch** (*Sequence[dict]*) –A batch of data from the dataloader.
- **data_samples** (*Sequence[dict]*) –A batch of outputs from the model.

compute_metrics(*results: List*)

Compute the metrics from processed results.

参数 **results** (*dict*) –The processed results of each batch.

返回 The computed metrics. The keys are the names of the metrics, and the values are corresponding results.

返回类型 Dict

class mmagic.evaluation.**MultiScaleStructureSimilarity**(*fake_nums: int*, *fake_key: Optional[str] = None*, *need_cond_input: bool = False*, *sample_model: str = 'ema'*, *collect_device: str = 'cpu'*, *prefix: Optional[str] = None*)

Bases: mmagic.evaluation.metrics.base_gen_metric.GenerativeMetric

MS-SSIM (Multi-Scale Structure Similarity) metric.

Ref: https://github.com/tkarras/progressive_growing_of_gans/blob/master/metrics/ms_ssim.py # noqa

参数

- **fake_nums** (*int*) –Numbers of the generated image need for the metric.
- **fake_key** (*Optional[str]*) –Key for get fake images of the output dict. Defaults to None.
- **real_key** (*Optional[str]*) –Key for get real images from the input dict. Defaults to 'img'.

- **need_cond_input** (*bool*) –If true, the sampler will return the conditional input randomly sampled from the original dataset. This require the dataset implement *get_data_info* and field *gt_label* must be contained in the return value of *get_data_info*. Noted that, for unconditional models, set *need_cond_input* as True may influence the result of evaluation results since the conditional inputs are sampled from the dataset distribution; otherwise will be sampled from the uniform distribution. Defaults to False.
- **sample_model** (*str*) –Sampling mode for the generative model. Support 'orig' and 'ema'. Defaults to 'ema'.
- **collect_device** (*str, optional*) –Device name used for collecting results from different ranks during distributed training. Must be 'cpu' or 'gpu'. Defaults to 'cpu'.
- **prefix** (*str, optional*) –The prefix that will be added in the metric names to disambiguate homonymous metrics of different evaluators. If prefix is not provided in the argument, self.default_prefix will be used instead. Defaults to None.

name = 'MS-SSIM'

process(*data_batch: dict*, *data_samples: Sequence[dict]*) → None

Feed data to the metric.

参数

- **data_batch** (*dict*) –Real images from dataloader. Do not be used in this metric.
- **data_samples** (*Sequence[dict]*) –Generated images.

_collect_target_results(*target: str*) → Optional[list]

Collected results for MS-SSIM metric. Size of *self.fake_results* in MS-SSIM does not relay on *self.fake_nums* but *self.num_pairs*.

参数 **target** (*str*) –Target results to collect.

返回 The collected results.

返回类型 Optional[list]

compute_metrics(*results_fake: List*)

Computed the result of MS-SSIM.

返回 Calculated MS-SSIM result.

返回类型 dict

class mmagic.evaluation.**PerceptualPathLength**(*fake_nums: int*, *real_nums: int = 0*, *fake_key: Optional[str] = None*, *real_key: Optional[str] = 'gt_img'*, *need_cond_input: bool = False*, *sample_model: str = 'ema'*, *collect_device: str = 'cpu'*, *prefix: Optional[str] = None*, *crop=True*, *epsilon=0.0001*, *space='W'*, *sampling='end'*, *latent_dim=512*)

Bases: mmagic.evaluation.metrics.base_gen_metric.GenerativeMetric

Perceptual path length.

Measure the difference between consecutive images (their VGG16 embeddings) when interpolating between two random inputs. Drastic changes mean that multiple features have changed together and that they might be entangled.

Ref: https://github.com/rosinality/stylegan2-pytorch/blob/master/ppl.py # noqa

参数

- **num_images** (*int*) –The number of evaluated generated samples.
- **image_shape** (*tuple, optional*) –Image shape in order "CHW". Defaults to None.
- **crop** (*bool, optional*) –Whether crop images. Defaults to True.
- **epsilon** (*float, optional*) –Epsilon parameter for path sampling. Defaults to 1e-4.
- **space** (*str, optional*) –Latent space. Defaults to 'W'.
- **sampling** (*str, optional*) –Sampling mode, whether sampling in full path or end-points. Defaults to 'end'.
- **latent_dim** (*int, optional*) –Latent dimension of input noise. Defaults to 512.
- **need_cond_input** (*bool*) –If true, the sampler will return the conditional input randomly sampled from the original dataset. This require the dataset implement *get_data_info* and field *gt_label* must be contained in the return value of *get_data_info*. Noted that, for unconditional models, set *need_cond_input* as True may influence the result of evaluation results since the conditional inputs are sampled from the dataset distribution; otherwise will be sampled from the uniform distribution. Defaults to False.

SAMPLER_MODE = 'path'

process(*data_batch: dict*, *data_samples: Sequence[dict]*) → None

Process one batch of data samples and predictions. The processed results should be stored in self.fake_results, which will be used to compute the metrics when all batches have been processed.

参数

- **data_batch** (*dict*) –A batch of data from the dataloader.
- **data_samples** (*Sequence[dict]*) –A batch of outputs from the model.

_compute_distance(*images*)

Feed data to the metric.

参数 **images** (*Tensor*) –Input tensor.

compute_metrics(*fake_results: list*) → dict

Summarize the results.

返回 Summarized results.

返回类型 dict | list

get_metric_sampler(*model: torch.nn.Module*, *dataloader: torch.utils.data.dataloader.DataLoader*, *metrics: list*)

Get sampler for generative metrics. Returns a dummy iterator, whose return value of each iteration is a dict containing batch size and sample mode to generate images.

参数

- **model** (*nn.Module*) –Model to evaluate.
- **dataloader** (*DataLoader*) –Dataloader for real images. Used to get batch size during generate fake images.
- **metrics** (*list*) –Metrics with the same sampler mode.

返回 Sampler for generative metrics.

返回类型 dummy_iterator

class mmagic.evaluation.**PrecisionAndRecall**(*fake_nums*, *real_nums=- 1*, *k=3*, *fake_key: Optional[str] = None*, *real_key: Optional[str] = 'gt_img'*, *need_cond_input: bool = False*, *sample_model: str = 'ema'*, *collect_device: str = 'cpu'*, *prefix: Optional[str] = None*, *vgg16_script='work_dirs/cache/vgg16.pt'*, *vgg16_pkl=None*, *row_batch_size=10000*, *col_batch_size=10000*, *auto_save=True*)

Bases: mmagic.evaluation.metrics.base_gen_metric.GenerativeMetric

Improved Precision and recall metric.

In this metric, we draw real and generated samples respectively, and embed them into a high-dimensional feature space using a pre-trained classifier network. We use these features to estimate the corresponding manifold. We obtain the estimation by calculating pairwise Euclidean distances between all feature vectors in the set and, for each feature vector, construct a hypersphere with radius equal to the distance to its kth nearest neighbor. Together, these hyperspheres define a volume in the feature space that serves as an estimate of the true manifold. Precision is quantified by querying for each generated image whether the image is within the estimated manifold of real images. Symmetrically, recall is calculated by querying for each real image whether the image is within estimated manifold of generated image.

Ref: https://github.com/NVlabs/stylegan2-ada-pytorch/blob/main/metrics/precision_recall.py # noqa

Note that we highly recommend that users should download the vgg16 script module from the following address. Then, the *vgg16_script* can be set with user's local path. If not given, we will use the vgg16 from pytorch model zoo. However, this may bring significant different in the final results.

Tero's vgg16: https://nvlabs-fi-cdn.nvidia.com/stylegan2-ada-pytorch/pretrained/metrics/vgg16.pt

参数

- **num_images** (*int*) –The number of evaluated generated samples.
- **image_shape** (*tuple*) –Image shape in order "CHW". Defaults to None.
- **num_real_need** (*int | None, optional*) –The number of real images. Defaults to None.
- **full_dataset** (*bool, optional*) –Whether to use full dataset for evaluation. Defaults to False.
- **k** (*int, optional*) –Kth nearest parameter. Defaults to 3.
- **bgr2rgb** (*bool, optional*) –Whether to change the order of image channel. Defaults to True.
- **vgg16_script** (*str, optional*) –Path for the Tero's vgg16 module. Defaults to 'work_dirs/cache/vgg16.pt'.
- **row_batch_size** (*int, optional*) –The batch size of row data. Defaults to 10000.
- **col_batch_size** (*int, optional*) –The batch size of col data. Defaults to 10000.
- **auto_save** (*bool, optional*) –Whether save vgg feature automatically.
- **need_cond_input** (*bool*) –If true, the sampler will return the conditional input randomly sampled from the original dataset. This require the dataset implement *get_data_info* and field *gt_label* must be contained in the return value of *get_data_info*. Noted that, for unconditional models, set *need_cond_input* as True may influence the result of evaluation results since the conditional inputs are sampled from the dataset distribution; otherwise will be sampled from the uniform distribution. Defaults to False.

name = 'PR'

_load_vgg(*vgg16_script: Optional[str]*) → Tuple[torch.nn.Module, bool]

Load VGG network from the given path.

参数 **vgg16_script** –The path of script model of VGG network. If None, will load the pytorch version.

返回

The actually loaded VGG network and corresponding style.

返回类型 Tuple[nn.Module, str]

extract_features(*images: torch.Tensor*) → torch.Tensor

Extracting image features.

参数 **images** (*torch.Tensor*) –Images tensor.

返回 Vgg16 features of input images.

返回类型 torch.Tensor

compute_metrics(*results_fake*) → dict

compute_metrics.

返回 Summarized results.

返回类型 dict

process(*data_batch: dict*, *data_samples: Sequence[dict]*) → None

Process one batch of data samples and predictions. The processed results should be stored in `self.fake_results`, which will be used to compute the metrics when all batches have been processed.

参数

- **data_batch** (*dict*) –A batch of data from the dataloader.
- **data_samples** (*Sequence[dict]*) –A batch of outputs from the model.

prepare(*module: torch.nn.Module*, *dataloader: torch.utils.data.dataloader.DataLoader*) → None

Prepare for the pre-calculating items of the metric. Defaults to do nothing.

参数

- **module** (*nn.Module*) –Model to evaluate.
- **dataloader** (*DataLoader*) –Dataloader for the real images.

class mmagic.evaluation.**SlicedWassersteinDistance**(*fake_nums: int*, *image_shape: tuple*, *fake_key: Optional[str] = None*, *real_key: Optional[str] = 'gt_img'*, *sample_model: str = 'ema'*, *collect_device: str = 'cpu'*, *prefix: Optional[str] = None*)

Bases: mmagic.evaluation.metrics.base_gen_metric.GenMetric

SWD (Sliced Wasserstein distance) metric. We calculate the SWD of two sets of images in the following way. In every ‘feed’, we obtain the Laplacian pyramids of every images and extract patches from the Laplacian pyramids as descriptors. In ‘summary’, we normalize these descriptors along channel, and reshape them so that we can use these descriptors to represent the distribution of real/fake images. And we can calculate the sliced Wasserstein distance of the real and fake descriptors as the SWD of the real and fake images.

Ref: https://github.com/tkarras/progressive_growing_of_gans/blob/master/metrics/sliced_wasserstein.py # noqa

参数

- **fake_nums** (*int*) –Numbers of the generated image need for the metric.
- **image_shape** (*tuple*) –Image shape in order “CHW”.
- **fake_key** (*Optional[str]*) –Key for get fake images of the output dict. Defaults to None.

- **real_key** (*Optional[str]*) –Key for get real images from the input dict. Defaults to 'gt_img'.
- **sample_model** (*str*) –Sampling mode for the generative model. Support 'orig' and 'ema'. Defaults to 'ema'.
- **collect_device** (*str*) –Device name used for collecting results from different ranks during distributed training. Must be 'cpu' or 'gpu'. Defaults to 'cpu'.
- **prefix** (*str, optional*) –The prefix that will be added in the metric names to disambiguate homonymous metrics of different evaluators. If prefix is not provided in the argument, self.default_prefix will be used instead. Defaults to None.

name = 'SWD'

process(*data_batch: dict*, *data_samples: Sequence[dict]*) → None

Process one batch of data samples and predictions. The processed results should be stored in `self.fake_results` and `self.real_results`, which will be used to compute the metrics when all batches have been processed.

参数

- **data_batch** (*dict*) –A batch of data from the dataloader.
- **data_samples** (*Sequence[dict]*) –A batch of outputs from the model.

_collect_target_results(*target: str*) → Optional[list]

Collect function for SWD metric. This function support collect results typing as *List[List[Tensor]]*.

参数 **target** (*str*) –Target results to collect.

返回 The collected results.

返回类型 Optional[list]

compute_metrics(*results_fake*, *results_real*) → dict

Compute the result of SWD metric.

参数

- **fake_results** (*list*) –List of image feature of fake images.
- **real_results** (*list*) –List of image feature of real images.

返回 A dict of the computed SWD metric.

返回类型 dict

class mmagic.evaluation.**TransFID**(*fake_nums: int*, *real_nums: int = - 1*, *inception_style='StyleGAN'*, *inception_path: Optional[str] = None*, *inception_pkl: Optional[str] = None*, *fake_key: Optional[str] = None*, *real_key: Optional[str] = 'img'*, *sample_model: str = 'ema'*, *collect_device: str = 'cpu'*, *prefix: Optional[str] = None*)

Bases: *FrechetInceptionDistance*

FID metric. In this metric, we calculate the distance between real distributions and fake distributions. The distributions are modeled by the real samples and fake samples, respectively. *Inception_v3* is adopted as the feature extractor, which is widely used in StyleGAN and BigGAN.

参数

- **fake_nums** (*int*) –Numbers of the generated image need for the metric.
- **real_nums** (*int*) –Numbers of the real images need for the metric. If -1 is passed, means all real images in the dataset will be used. Defaults to -1.
- **inception_style** (*str*) –The target inception style want to load. If the given style cannot be loaded successful, will attempt to load a valid one. Defaults to 'StyleGAN'.
- **inception_path** (*str, optional*) –Path the the pretrain Inception network. Defaults to None.
- **inception_pkl** (*str, optional*) –Path to reference inception pickle file. If *None*, the statistical value of real distribution will be calculated at running time. Defaults to None.
- **fake_key** (*Optional[str]*) –Key for get fake images of the output dict. Defaults to None.
- **real_key** (*Optional[str]*) –Key for get real images from the input dict. Defaults to 'img'.
- **need_cond_input** (*bool*) –If true, the sampler will return the conditional input randomly sampled from the original dataset. This require the dataset implement *get_data_info* and field *gt_label* must be contained in the return value of *get_data_info*. Noted that, for unconditional models, set *need_cond_input* as True may influence the result of evaluation results since the conditional inputs are sampled from the dataset distribution; otherwise will be sampled from the uniform distribution. Defaults to False.
- **sample_model** (*str*) –Sampling mode for the generative model. Support 'orig' and 'ema'. Defaults to 'orig'.
- **collect_device** (*str, optional*) –Device name used for collecting results from different ranks during distributed training. Must be 'cpu' or 'gpu'. Defaults to 'cpu'.
- **prefix** (*str, optional*) –The prefix that will be added in the metric names to disambiguate homonymous metrics of different evaluators. If prefix is not provided in the argument, self.default_prefix will be used instead. Defaults to None.

get_metric_sampler(*model: torch.nn.Module*, *dataloader: torch.utils.data.dataloader.DataLoader*, *metrics: List[mmagic.evaluation.metrics.base_gen_metric.GenerativeMetric]*) → torch.utils.data.dataloader.DataLoader

Get sampler for normal metrics. Directly returns the dataloader.

参数

- **model** (*nn.Module*) –Model to evaluate.
- **dataloader** (*DataLoader*) –Dataloader for real images.
- **metrics** (*List['GenMetric']*) –Metrics with the same sample mode.

返回 Default sampler for normal metrics.

返回类型 DataLoader

class mmagic.evaluation.**TransIS**(*fake_nums: int = 50000*, *resize: bool = True*, *splits: int = 10*, *inception_style: str = 'StyleGAN'*, *inception_path: Optional[str] = None*, *resize_method='bicubic'*, *use_pillow_resize: bool = True*, *fake_key: Optional[str] = None*, *sample_model='ema'*, *collect_device: str = 'cpu'*, *prefix: str = None*)

Bases: *InceptionScore*

IS (Inception Score) metric. The images are split into groups, and the inception score is calculated on each group of images, then the mean and standard deviation of the score is reported. The calculation of the inception score on a group of images involves first using the inception v3 model to calculate the conditional probability for each image (p(y|x)). The marginal probability is then calculated as the average of the conditional probabilities for the images in the group (p(y)). The KL divergence is then calculated for each image as the conditional probability multiplied by the log of the conditional probability minus the log of the marginal probability. The KL divergence is then summed over all images and averaged over all classes and the exponent of the result is calculated to give the final score.

Ref: https://github.com/sbarratt/inception-score-pytorch/blob/master/inception_score.py # noqa

Note that we highly recommend that users should download the Inception V3 script module from the following address. Then, the *inception_pkl* can be set with user's local path. If not given, we will use the Inception V3 from pytorch model zoo. However, this may bring significant different in the final results.

Tero's Inception V3: https://nvlabs-fi-cdn.nvidia.com/stylegan2-ada-pytorch/pretrained/metrics/inception-2015-12-05.pt # noqa

参数

- **fake_nums** (*int*) –Numbers of the generated image need for the metric.
- **resize** (*bool, optional*) –Whether resize image to 299x299. Defaults to True.
- **splits** (*int, optional*) –The number of groups. Defaults to 10.
- **inception_style** (*str*) –The target inception style want to load. If the given style cannot be loaded successful, will attempt to load a valid one. Defaults to 'StyleGAN'.
- **inception_path** (*str, optional*) –Path the the pretrain Inception network. Defaults to None.
- **resize_method** (*str*) –Resize method. If *resize* is False, this will be ignored. Defaults to 'bicubic'.

- **use_pil_resize** (*bool*) –Whether use Bicubic interpolation with Pillow's backend. If set as True, the evaluation process may be a little bit slow, but achieve a more accurate IS result. Defaults to False.
- **fake_key** (*Optional[str]*) –Key for get fake images of the output dict. Defaults to None.
- **sample_model** (*str*) –Sampling mode for the generative model. Support 'orig' and 'ema'. Defaults to 'ema'.
- **collect_device** (*str, optional*) –Device name used for collecting results from different ranks during distributed training. Must be 'cpu' or 'gpu'. Defaults to 'cpu'.
- **prefix** (*str, optional*) –The prefix that will be added in the metric names to disambiguate homonymous metrics of different evaluators. If prefix is not provided in the argument, self.default_prefix will be used instead. Defaults to None.

get_metric_sampler(*model: torch.nn.Module*, *dataloader: torch.utils.data.dataloader.DataLoader*, *metrics: List[mmagic.evaluation.metrics.base_gen_metric.GenerativeMetric]*) → torch.utils.data.dataloader.DataLoader

Get sampler for normal metrics. Directly returns the dataloader.

参数

- **model** (*nn.Module*) –Model to evaluate.
- **dataloader** (*DataLoader*) –Dataloader for real images.
- **metrics** (*List['GenMetric']*) –Metrics with the same sample mode.

返回 Default sampler for normal metrics.

返回类型 DataLoader

CHAPTER 73

mmagic.visualization

73.1 Package Contents

73.1.1 Classes

ConcatImageVisualizer	Visualize multiple images by concatenation.
PaviVisBackend	Visualization backend for Pavi.
TensorboardVisBackend	Tensorboard visualization backend class.
VisBackend	MMagic visualization backend class. It can write image, config, scalars,
WandbVisBackend	Wandb visualization backend for MMagic.
Visualizer	MMagic Visualizer.

class mmagic.visualization.**ConcatImageVisualizer**(*fn_key: str*, *img_keys: Sequence[str]*, *pixel_range={}*, *bgr2rgb=False*, *name: str = 'visualizer'*, **args*, ***kwargs*)

Bases: mmengine.visualization.Visualizer

Visualize multiple images by concatenation.

This visualizer will horizontally concatenate images belongs to different keys and vertically concatenate images belongs to different frames to visualize.

Image to be visualized can be:

- torch.Tensor or np.array
- Image sequences of shape (T, C, H, W)
- Multi-channel image of shape (1/3, H, W)
- Single-channel image of shape (C, H, W)

参数

- **fn_key** (*str*) –key used to determine file name for saving image. Usually it is the path of some input image. If the value is *dir/basename.ext*, the name used for saving will be basename.
- **img_keys** (*str*) –keys, values of which are images to visualize.
- **pixel_range** (*dict*) –min and max pixel value used to denormalize images, note that only float array or tensor will be denormalized, uint8 arrays are assumed to be unnormalized.
- **bgr2rgb** (*bool*) –whether to convert the image from BGR to RGB.
- **name** (*str*) –name of visualizer. Default: 'visualizer'.
- ****kwargs** (**args and*) –Other arguments are passed to *Visualizer*. # noqa

add_datasample(*data_sample:* mmagic.structures.DataSample, *step=0*) → None

Concatenate image and draw.

参数

- **input** (*torch.Tensor*) –Single input tensor from data_batch.
- **data_sample** (*DataSample*) –Single data_sample from data_batch.
- **output** (*DataSample*) –Single prediction output by model.
- **step** (*int*) –Global step value to record. Default: 0.

class mmagic.visualization.**PaviVisBackend**(*save_dir: str*, *exp_name: Optional[str] = None*, *labels: Optional[str] = None*, *project: Optional[str] = None*, *model: Optional[str] = None*, *description: Optional[str] = None*)

Bases: mmengine.visualization.BaseVisBackend

Visualization backend for Pavi.

property experiment: *VisBackend*

Return the experiment object associated with this visualization backend.

_init_env()

Init save dir.

add_image(*name: str*, *image: numpy.array*, *step: int = 0*, ***kwargs*) → None

Record the image to Pavi.

参数

- **name** (*str*) –The image identifier.
- **image** (*np.ndarray*) –The image to be saved. The format should be RGB. Default to None.
- **step** (*int*) –Global step value to record. Default to 0.

add_scalar(*name: str*, *value: Union[int, float, torch.Tensor, numpy.ndarray]*, *step: int = 0*, ***kwargs*) → None

Record the scalar data to Pavi.

参数

- **name** (*str*) –The scalar identifier.
- **value** (*int, float, torch.Tensor, np.ndarray*) –Value to save.
- **step** (*int*) –Global step value to record. Default to 0.

add_scalars(*scalar_dict: dict*, *step: int = 0*, *file_path: Optional[str] = None*, ***kwargs*) → None

Record the scalars to Pavi.

The scalar dict will be written to the default and specified files if `file_path` is specified.

参数

- **scalar_dict** (*dict*) –Key-value pair storing the tag and corresponding values. The value must be dumped into json format.
- **step** (*int*) –Global step value to record. Default to 0.
- **file_path** (*str, optional*) –The scalar's data will be saved to the `file_path` file at the same time if the `file_path` parameter is specified. Default to None.

class mmagic.visualization.**TensorboardVisBackend**(*save_dir: str*)

Bases: `mmengine.visualization.TensorboardVisBackend`

Tensorboard visualization backend class.

It can write images, config, scalars, etc. to a tensorboard file.

实际案例

```
>>> from mmengine.visualization import TensorboardVisBackend
>>> import numpy as np
>>> vis_backend = TensorboardVisBackend(save_dir='temp_dir')
>>> img = np.random.randint(0, 256, size=(10, 10, 3))
>>> vis_backend.add_image('img', img)
>>> vis_backend.add_scaler('mAP', 0.6)
>>> vis_backend.add_scalars({'loss': 0.1,'acc':0.8})
>>> cfg = Config(dict(a=1, b=dict(b1=[0, 1])))
>>> vis_backend.add_config(cfg)
```

参数 **save_dir** (*str*) –The root directory to save the files produced by the backend.

add_image(*name: str*, *image: numpy.array*, *step: int = 0*, ***kwargs*)

Record the image to Tensorboard. Additional support upload gif files.

参数

- **name** (*str*) –The image identifier.
- **image** (*np.ndarray*) –The image to be saved. The format should be RGB.
- **step** (*int*) –Useless parameter. Wandb does not need this parameter. Default to 0.

class mmagic.visualization.**VisBackend**(*save_dir: str*, *img_save_dir: str = 'vis_image'*, *config_save_file: str = 'config.py'*, *scalar_save_file: str = 'scalars.json'*, *ceph_path: Optional[str] = None*, *delete_local_image: bool = True*)

Bases: mmengine.visualization.BaseVisBackend

MMagic visualization backend class. It can write image, config, scalars, etc. to the local hard disk and ceph path. You can get the drawing backend through the experiment property for custom drawing.

实际案例

```
>>> from mmagic.visualization import VisBackend
>>> import numpy as np
>>> vis_backend = VisBackend(save_dir='temp_dir',
>>>                          ceph_path='s3://temp-bucket')
>>> img = np.random.randint(0, 256, size=(10, 10, 3))
>>> vis_backend.add_image('img', img)
>>> vis_backend.add_scalar('mAP', 0.6)
>>> vis_backend.add_scalars({'loss': [1, 2, 3], 'acc': 0.8})
```

(下页继续)

(续上页)

```
>>> cfg = Config(dict(a=1, b=dict(b1=[0, 1])))
>>> vis_backend.add_config(cfg)
```

参数

- **save_dir** (*str*) –The root directory to save the files produced by the visualizer.
- **img_save_dir** (*str*) –The directory to save images. Default to ‘vis_image’.
- **config_save_file** (*str*) –The file name to save config. Default to ‘config.py’.
- **scalar_save_file** (*str*) –The file name to save scalar values. Default to ‘scalars.json’.
- **ceph_path** (*Optional[str]*) –The remote path of Ceph cloud storage. Defaults to None.
- **delete_local** (*bool*) –Whether delete local after uploading to ceph or not. If `ceph_path` is None, this will be ignored. Defaults to True.

property experiment: *VisBackend*

Return the experiment object associated with this visualization backend.

_init_env()

Setup env for VisBackend.

add_config(*config: mmengine.config.Config*, ***kwargs*) → None

Record the config to disk.

参数 **config** (*Config*) –The Config object

add_image(*name: str*, *image: numpy.array*, *step: int = 0*, ***kwargs*) → None

Record the image to disk.

参数

- **name** (*str*) –The image identifier.
- **image** (*np.ndarray*) –The image to be saved. The format should be RGB. Default to None.
- **step** (*int*) –Global step value to record. Default to 0.

add_scalar(*name: str*, *value: Union[int, float, torch.Tensor, numpy.ndarray]*, *step: int = 0*, ***kwargs*) → None

Record the scalar data to disk.

参数

- **name** (*str*) –The scalar identifier.

- **value** (*int, float, torch.Tensor, np.ndarray*) –Value to save.
- **step** (*int*) –Global step value to record. Default to 0.

add_scalars(*scalar_dict: dict*, *step: int = 0*, *file_path: Optional[str] = None*, ***kwargs*) → None

Record the scalars to disk.

The scalar dict will be written to the default and specified files if `file_path` is specified.

参数

- **scalar_dict** (*dict*) –Key-value pair storing the tag and corresponding values. The value must be dumped into json format.
- **step** (*int*) –Global step value to record. Default to 0.
- **file_path** (*str, optional*) –The scalar's data will be saved to the `file_path` file at the same time if the `file_path` parameter is specified. Default to None.

_dump(*value_dict: dict*, *file_path: str*, *file_format: str*) → None

dump dict to file.

参数

- **value_dict** (*dict*) –The dict data to saved.
- **file_path** (*str*) –The file path to save data.
- **file_format** (*str*) –The file format to save data.

_upload(*path: str*, *delete_local=False*) → None

Upload file at path to remote.

参数 **path** (*str*) –Path of file to upload.

class mmagic.visualization.**WandbVisBackend**(*save_dir: str*, *init_kwargs: Optional[dict] = None*, *define_metric_cfg: Union[dict, list, None] = None*, *commit: Optional[bool] = True*, *log_code_name: Optional[str] = None*, *watch_kwargs: Optional[dict] = None*)

Bases: `mmengine.visualization.WandbVisBackend`

Wandb visualization backend for MMagic.

_init_env()

Setup env for wandb.

add_image(*name: str*, *image: numpy.array*, *step: int = 0*, ***kwargs*)

Record the image to wandb. Additional support upload gif files.

参数

- **name** (*str*) –The image identifier.

- **image** (*np.ndarray*) –The image to be saved. The format should be RGB.
- **step** (*int*) –Useless parameter. Wandb does not need this parameter. Default to 0.

class mmagic.visualization.**Visualizer**(*name='visualizer'*, *vis_backends: Optional[List[Dict]] = None*, *save_dir: Optional[str] = None*)

Bases: mmengine.visualization.Visualizer

MMagic Visualizer.

参数

- **name** (*str*) –Name of the instance. Defaults to 'visualizer'.
- **vis_backends** (*list, optional*) –Visual backend config list. Defaults to None.
- **save_dir** (*str, optional*) –Save file dir for all storage backends. If it is None, the backend storage will not save any data.

Examples:

```
>>> # Draw image
>>> vis = Visualizer()
>>> vis.add_datasample(
>>>     'random_noise',
>>>     gen_samples=torch.rand(2, 3, 10, 10),
>>>     gt_samples=dict(imgs=torch.randn(2, 3, 10, 10)),
>>>     gt_keys='imgs',
>>>     vis_mode='image',
>>>     n_rows=2,
>>>     step=10)
```

static _post_process_image(*image: torch.Tensor*) → torch.Tensor

Post process images.

参数 **image** (*Tensor*) –Image to post process. The value range of image should be in [0, 255] and the channel order should be BGR.

返回 Image in RGB color order.

返回类型 Tensor

static _get_n_row_and_padding(*samples: Tuple[dict, torch.Tensor]*, *n_row: Optional[int] = None*) → Tuple[int, Optional[torch.Tensor]]

Get number of sample in each row and tensor for padding the empty position.

参数

- **samples** (*Tuple[dict, Tensor]*) –Samples to visualize.
- **n_row** (*int, optional*) –Number of images displayed in each row of. If not passed, n_row will be set as int(sqrt(batch_size)).

返回

Number of sample in each row and tensor for padding the empty position.

返回类型 Tuple[int, Optional[int]]

_vis_gif_sample(*gen_samples: mmagic.utils.typing.SampleList*, *target_keys: Union[str, List[str], None]*, *n_row: int*) → numpy.ndarray

Visualize gif samples.

参数

- **gen_samples** (*SampleList*) –List of data samples to visualize
- **target_keys** (*Union[str, List[str], None]*) –Keys of the visualization target in data samples.
- **n_rows** (*int, optional*) –Number of images in one row.

返回 The visualization results.

返回类型 np.ndarray

_vis_image_sample(*gen_samples: mmagic.utils.typing.SampleList*, *target_keys: Union[str, List[str], None]*, *n_row: int*) → numpy.ndarray

Visualize image samples.

参数

- **gen_samples** (*SampleList*) –List of data samples to visualize
- **target_keys** (*Union[str, List[str], None]*) –Keys of the visualization target in data samples.
- **color_order** (*str*) –The color order of the passed images.
- **target_mean** (*Sequence[Union[float, int]]*) –The target mean of the visualization results.
- **target_std** (*Sequence[Union[float, int]]*) –The target std of the visualization results.
- **n_rows** (*int, optional*) –Number of images in one row.

返回 The visualization results.

返回类型 np.ndarray

_get_vis_data_by_key(*sample:* mmagic.structures.DataSample, *key: str*) → torch.Tensor

Get tensor in `DataSample` by the given key.

参数

- **sample** (*DataSample*) –Input data sample.

- **key** (*str*) –Name of the target tensor.

返回 Tensor from the data sample.

返回类型 Tensor

add_datasample(*name: str*, ***, *gen_samples: Sequence[*mmagic.structures.DataSample*]*, *target_keys: Optional[Tuple[str, List[str]]] = None*, *vis_mode: Optional[str] = None*, *n_row: Optional[int] = None*, *show: bool = False*, *wait_time: int = 0*, *step: int = 0*, ***kwargs*) → None

Draw datasample and save to all backends.

If GT and prediction are plotted at the same time, they are displayed in a stitched image where the left image is the ground truth and the right image is the prediction.

If `show` is True, all storage backends are ignored, and the images will be displayed in a local window.

参数

- **name** (*str*) –The image identifier.
- **gen_samples** (*List[DataSample]*) –Data samples to visualize.
- **vis_mode** (*str, optional*) –Visualization mode. If not passed, will visualize results as image. Defaults to None.
- **n_rows** (*int, optional*) –Number of images in one row. Defaults to None.
- **color_order** (*str*) –The color order of the passed images. Defaults to 'bgr'.
- **target_mean** (*Sequence[Union[float, int]]*) –The target mean of the visualization results. Defaults to 127.5.
- **target_std** (*Sequence[Union[float, int]]*) –The target std of the visualization results. Defaults to 127.5.
- **show** (*bool*) –Whether to display the drawn image. Default to False.
- **wait_time** (*float*) –The interval of show (s). Defaults to 0.
- **step** (*int*) –Global step value to record. Defaults to 0.

add_image(*name: str*, *image: numpy.ndarray*, *step: int = 0*, ***kwargs*) → None

Record the image. Support input kwargs.

参数

- **name** (*str*) –The image identifier.
- **image** (*np.ndarray, optional*) –The image to be saved. The format should be RGB. Default to None.
- **step** (*int*) –Global step value to record. Default to 0.

CHAPTER 74

mmagic.engine.hooks

74.1 Package Contents

74.1.1 Classes

ExponentialMovingAverageHook	Exponential Moving Average Hook.
IterTimerHook	IterTimerHooks inherits from `mmengine.hooks.IterTimerHook` and
PGGANFetchDataHook	PGGAN Fetch Data Hook.
PickleDataHook	Pickle Useful Data Hook.
ReduceLRSchedulerHook	A hook to update learning rate.
BasicVisualizationHook	Basic hook that invoke visualizers during validation and test.
VisualizationHook	MMagic Visualization Hook. Used to visual output samples in training,

class mmagic.engine.hooks.**ExponentialMovingAverageHook**(*module_keys*, *interp_mode='lerp'*, *interp_cfg=None*, *interval=-1*, *start_iter=0*)

Bases: `mmengine.hooks.Hook`

Exponential Moving Average Hook.

Exponential moving average is a trick that widely used in current GAN literature, e.g., PGGAN, StyleGAN, and BigGAN. This general idea of it is maintaining a model with the same architecture, but its parameters are updated as a moving average of the trained weights in the original model. In general, the model with moving averaged weights achieves better performance.

参数

- **module_keys** (*str | tuple[str]*) –The name of the ema model. Note that we require these keys are followed by '_ema' so that we can easily find the original model by discarding the last four characters.
- **interp_mode** (*str, optional*) –Mode of the interpolation method. Defaults to 'lerp'.
- **interp_cfg** (*dict | None, optional*) –Set arguments of the interpolation function. Defaults to None.
- **interval** (*int, optional*) –Evaluation interval (by iterations). Default: -1.
- **start_iter** (*int, optional*) –Start iteration for ema. If the start iteration is not reached, the weights of ema model will maintain the same as the original one. Otherwise, its parameters are updated as a moving average of the trained weights in the original model. Default: 0.

static lerp(*a*, *b*, *momentum=0.001*, *momentum_nontrainable=1.0*, *trainable=True*)

Does a linear interpolation of two parameters/ buffers.

参数

- **a** (*torch.Tensor*) –Interpolation start point, refer to orig state.
- **b** (*torch.Tensor*) –Interpolation end point, refer to ema state.
- **momentum** (*float, optional*) –The weight for the interpolation formula. Defaults to 0.001.
- **momentum_nontrainable** (*float, optional*) –The weight for the interpolation formula used for nontrainable parameters. Defaults to 1..
- **trainable** (*bool, optional*) –Whether input parameters is trainable. If set to False, momentum_nontrainable will be used. Defaults to True.

返回 Interpolation result.

返回类型 torch.Tensor

every_n_iters(*runner: mmengine.runner.Runner*, *n: int*)

This is the function to perform every n iterations.

参数

- **runner** (*Runner*) –runner used to drive the whole pipeline

- **n** (*int*) –the number of iterations

返回 the latest iterations

返回类型 int

after_train_iter(*runner: mmengine.runner.Runner*, *batch_idx: int*, *data_batch: DATA_BATCH = None*, *outputs: Optional[dict] = None*) → None

This is the function to perform after each training iteration.

参数

- **runner** (*Runner*) –runner to drive the pipeline
- **batch_idx** (*int*) –the id of batch
- **data_batch** (*DATA_BATCH, optional*) –data batch. Defaults to None.
- **outputs** (*Optional[dict], optional*) –output. Defaults to None.

before_run(*runner: mmengine.runner.Runner*)

This is the function perform before each run.

参数 **runner** (*Runner*) –runner used to drive the whole pipeline

引发 **RuntimeError** –error message

class mmagic.engine.hooks.**IterTimerHook**

Bases: mmengine.hooks.IterTimerHook

IterTimerHooks inherits from mmengine.hooks.IterTimerHook and overwrites self._after_iter().

This hooks should be used along with *mmagic.engine.runner.MultiValLoop* and *mmagic.engine.runner.MultiTestLoop*.

_after_iter(*runner*, *batch_idx: int*, *data_batch: DATA_BATCH = None*, *outputs: Optional[Union[dict, Sequence[mmengine.structures.BaseDataElement]]] = None*, *mode: str = 'train'*) → None

Calculating time for an iteration and updating "time" HistoryBuffer of runner.message_hub. If *mode* is 'train', we take *runner.max_iters* as the total iterations and calculate the rest time. If *mode* in *val* or *test*, we use *runner.val_loop.total_length* or *runner.test_loop.total_length* as total number of iterations. If you want to know how *total_length* is calculated, please refers to *mmagic.engine.runner.MultiValLoop.run()* and *mmagic.engine.runner.MultiTestLoop.run()*.

参数

- **runner** (*Runner*) –The runner of the training validation and testing process.
- **batch_idx** (*int*) –The index of the current batch in the loop.
- **data_batch** (*Sequence[dict], optional*) –Data from dataloader. Defaults to None.

- **outputs** (*dict or sequence, optional*) –Outputs from model. Defaults to None.
- **mode** (*str*) –Current mode of runner. Defaults to 'train'.

class mmagic.engine.hooks.**PGGANFetchDataHook**

Bases: mmengine.hooks.Hook

PGGAN Fetch Data Hook.

参数 **interval** (*int, optional*) –The interval of calling this hook. If set to -1, the visualization hook will not be called. Defaults to 1.

before_train_iter(*runner*, *batch_idx: int*, *data_batch: DATA_BATCH = None*) → None

All subclasses should override this method, if they need any operations before each training iteration.

参数

- **runner** (*Runner*) –The runner of the training process.
- **batch_idx** (*int*) –The index of the current batch in the train loop.
- **data_batch** (*dict or tuple or list, optional*) –Data from dataloader.

update_dataloader(*dataloader: torch.utils.data.dataloader.DataLoader*, *curr_scale: int*) → Optional[torch.utils.data.dataloader.DataLoader]

Update the data loader.

参数

- **dataloader** (*DataLoader*) –The dataloader to be updated.
- **curr_scale** (*int*) –The current scale of the generated image.

返回

The updated dataloader. If the dataloader do not need to update, return None.

返回类型 Optional[DataLoader]

class mmagic.engine.hooks.**PickleDataHook**(*output_dir*, *data_name_list*, *interval=- 1*, *before_run=False*, *after_run=False*, *filename_tmpl='iter_{}.pkl'*)

Bases: mmengine.hooks.Hook

Pickle Useful Data Hook.

This hook will be used in SinGAN training for saving some important data that will be used in testing or inference.

参数

- **output_dir** (*str*) –The output path for saving pickled data.
- **data_name_list** (*list[str]*) –The list contains the name of results in outputs dict.

- **interval** (*int*) –The interval of calling this hook. If set to -1, the PickleDataHook will not be called during training. Default: -1.
- **before_run** (*bool, optional*) –Whether to save before running. Defaults to False.
- **after_run** (*bool, optional*) –Whether to save after running. Defaults to False.
- **filename_tmpl** (*str, optional*) –Format string used to save images. The output file name will be formatted as this args. Defaults to 'iter_{}.pkl'.

after_run(*runner*)

The behavior after each train iteration.

参数 **runner** (*object*) –The runner.

before_run(*runner*)

The behavior after each train iteration.

参数 **runner** (*object*) –The runner.

after_train_iter(*runner*, *batch_idx: int*, *data_batch: DATA_BATCH = None*, *outputs: Optional[dict] = None*)

The behavior after each train iteration.

参数

- **runner** (*Runner*) –The runner of the training process.
- **batch_idx** (*int*) –The index of the current batch in the train loop.
- **data_batch** (*Sequence[dict], optional*) –Data from dataloader. Defaults to None.
- **outputs** (*dict, optional*) –Outputs from model. Defaults to None.

_pickle_data(*runner: mmengine.runner.Runner*)

Save target data to pickle file.

参数 **runner** (*Runner*) –The runner of the training process.

_get_numpy_data(*data: Tuple[List[torch.Tensor], torch.Tensor, int]*) → Tuple[List[numpy.ndarray], numpy.ndarray, int]

Convert tensor or list of tensor to numpy or list of numpy.

参数 **data** (*Tuple[List[Tensor], Tensor, int]*) –Data to be converted.

返回 Converted data.

返回类型 Tuple[List[np.ndarray], np.ndarray, int]

class mmagic.engine.hooks.**ReduceLRSchedulerHook**(*val_metric: str = None*, *by_epoch=True*, *interval=1*)

Bases: mmengine.hooks.ParamSchedulerHook

A hook to update learning rate.

参数

- **val_metric** (*str*) –The metric of validation. If val_metric is not None, we check val_metric to reduce learning. Default: None.
- **by_epoch** (*bool*) –Whether to update by epoch. Default: True.
- **interval** (*int*) –The interval of iterations to update. Default: 1.

_calculate_average_value()

after_train_epoch(*runner: mmengine.runner.Runner*)

Call step function for each scheduler after each train epoch.

参数 **runner** (*Runner*) –The runner of the training process.

after_train_iter(*runner: mmengine.runner.Runner*, *batch_idx: int*, *data_batch: DATA_BATCH = None*, *outputs: Optional[dict] = None*) → None

Call step function for each scheduler after each iteration.

参数

- **runner** (*Runner*) –The runner of the training process.
- **batch_idx** (*int*) –The index of the current batch in the train loop.
- **data_batch** (*Sequence[dict], optional*) –Data from dataloader. In order to keep this interface consistent with other hooks, we keep data_batch here. Defaults to None.
- **outputs** (*dict, optional*) –Outputs from model. In order to keep this interface consistent with other hooks, we keep data_batch here. Defaults to None.

after_val_epoch(*runner*, *metrics: Optional[Dict[str, float]] = None*)

Call step function for each scheduler after each validation epoch.

参数

- **runner** (*Runner*) –The runner of the training process.
- **metrics** (*dict, optional*) –The metrics of validation. Default: None.

class mmagic.engine.hooks.**BasicVisualizationHook**(*interval: dict = {}*, *on_train=False*, *on_val=True*, *on_test=True*)

Bases: mmengine.hooks.Hook

Basic hook that invoke visualizers during validation and test.

参数

- **interval** (*int | dict*) –Visualization interval. Default: {}.
- **on_train** (*bool*) –Whether to call hook during train. Default to False.
- **on_val** (*bool*) –Whether to call hook during validation. Default to True.
- **on_test** (*bool*) –Whether to call hook during test. Default to True.

priority = 'NORMAL'

_after_iter(*runner*, *batch_idx: int*, *data_batch: Optional[Sequence[dict]]*, *outputs: Optional[Sequence[mmengine.structures.BaseDataElement]]*, *mode=None*) → None

Show or Write the predicted results.

参数

- **runner** (*Runner*) –The runner of the training process.
- **batch_idx** (*int*) –The index of the current batch in the test loop.
- **data_batch** (*Sequence[dict], optional*) –Data from dataloader. Defaults to None.
- **outputs** (*Sequence[BaseDataElement], optional*) –Outputs from model. Defaults to None.

class mmagic.engine.hooks.**VisualizationHook**(*interval: int = 1000*, *vis_kwargs_list: Tuple[List[dict], dict] = None*, *fixed_input: bool = True*, *n_samples: Optional[int] = 64*, *n_row: Optional[int] = None*, *message_hub_vis_kwargs: Optional[Tuple[str, dict, List[str], List[Dict]]] = None*, *save_at_test: bool = True*, *max_save_at_test: int = 100*, *test_vis_keys: Optional[Union[str, List[str]]] = None*, *show: bool = False*, *wait_time: float = 0*)

Bases: mmengine.hooks.Hook

MMagic Visualization Hook. Used to visual output samples in training, validation and testing. In this hook, we use a list called *sample_kwargs_list* to control how to generate samples and how to visualize them. Each element in *sample_kwargs_list*, called *sample_kwargs*, may contains the following keywords:

- **Required key words:**
  - **'type' : Value must be string. Denotes what kind of sampler is used to** generate image. Refers to *get_sampler()*.
- **Optional key words (If not passed, will use the default value):**
  - 'n_row' : Value must be int. The number of images in one row.

- ‘num_samples’ : Value must be int. The number of samples to visualize.
- **‘vis_mode’ : Value must be string. How to visualize the generated** samples (e.g. image, gif).
- **‘fixed_input’ : Value must be bool. Whether use the fixed input** during the loop.
- ‘draw_gt’ : Value must be bool. Whether save the real images.
- **‘target_keys’ : Value must be string or list of string. The keys of** the target image to visualize.
- **‘name’ : Value must be string. If not passed, will use** *sample_kwargs[‘type’]* as default.

For convenience, we also define a group of alias of samplers’ type for models supported in MMagic. Refers to *:attr:self.SAMPLER_TYPE_MAPPING*.

示例

```
>>> # for GAN models
>>> custom_hooks = [
>>>     dict(
>>>         type='VisualizationHook',
>>>         interval=1000,
>>>         fixed_input=True,
>>>         vis_kwargs_list=dict(type='GAN', name='fake_img'))]
>>> # for Translation models
>>> custom_hooks = [
>>>     dict(
>>>         type='VisualizationHook',
>>>         interval=10,
>>>         fixed_input=False,
>>>         vis_kwargs_list=[dict(type='Translation',
>>>                               name='translation_train',
>>>                               n_samples=6, draw_gt=True,
>>>                               n_row=3),
>>>                          dict(type='TranslationVal',
>>>                               name='translation_val',
>>>                               n_samples=16, draw_gt=True,
>>>                               n_row=4)])]
```

NOTE: user-defined vis_kwargs > vis_kwargs_mapping > hook init args

参数

- **interval** (*int*) –Visualization interval. Default: 1000.
- **sampler_kwargs_list** (*Tuple[List[dict], dict]*) –The list of sampling behavior to generate images.

- **fixed_input** (*bool*) –The default action of whether use fixed input to generate samples during the loop. Defaults to True.
- **n_samples** (*Optional[int]*) –The default value of number of samples to visualize. Defaults to 64.
- **n_row** (*Optional[int]*) –The default value of number of images in each row in the visualization results. Defaults to None.
- **(Optional[Tuple[str** (*message_hub_vis_kwargs*) –List[Dict]]]): Key arguments visualize images in message hub. Defaults to None.
- **dict** –List[Dict]]]): Key arguments visualize images in message hub. Defaults to None.
- **List[str]** –List[Dict]]]): Key arguments visualize images in message hub. Defaults to None.

:param [List[Dict]]]): Key arguments visualize images in message hub.] Defaults to None.

参数

- **save_at_test** (*bool*) –Whether save images during test. Defaults to True.
- **max_save_at_test** (*int*) –Maximum number of samples saved at test time. If None is passed, all samples will be saved. Defaults to 100.
- **show** (*bool*) –Whether to display the drawn image. Default to False.
- **wait_time** (*float*) –The interval of show (s). Defaults to 0.

priority = 'NORMAL'

VIS_KWARGS_MAPPING

after_val_iter(*runner: mmengine.runner.Runner*, *batch_idx: int*, *data_batch: dict*, *outputs*) → None

VisualizationHook do not support visualize during validation.

参数

- **runner** (*Runner*) –The runner of the training process.
- **batch_idx** (*int*) –The index of the current batch in the test loop.
- **data_batch** (*Sequence[dict], optional*) –Data from dataloader. Defaults to None.
- **outputs** –outputs of the generation model

after_test_iter(*runner: mmengine.runner.Runner*, *batch_idx: int*, *data_batch: dict*, *outputs*)

Visualize samples after test iteration.

参数

- **runner** (*Runner*) –The runner of the training process.
- **batch_idx** (*int*) –The index of the current batch in the test loop.
- **data_batch** (*dict, optional*) –Data from dataloader. Defaults to None.
- **outputs** –outputs of the generation model Defaults to None.

after_train_iter(*runner: mmengine.runner.Runner*, *batch_idx: int*, *data_batch: dict = None*, *outputs: Optional[dict] = None*) → None

Visualize samples after train iteration.

参数

- **runner** (*Runner*) –The runner of the training process.
- **batch_idx** (*int*) –The index of the current batch in the train loop.
- **data_batch** (*dict*) –Data from dataloader. Defaults to None.
- **outputs** (*dict, optional*) –Outputs from model. Defaults to None.

vis_sample(*runner: mmengine.runner.Runner*, *batch_idx: int*, *data_batch: dict*, *outputs: Optional[dict] = None*) → None

Visualize samples.

参数

- **runner** (*Runner*) –The runner contains model to visualize.
- **batch_idx** (*int*) –The index of the current batch in loop.
- **data_batch** (*dict*) –Data from dataloader. Defaults to None.
- **outputs** (*dict, optional*) –Outputs from model. Defaults to None.

vis_from_message_hub(*batch_idx: int*)

Visualize samples from message hub.

参数

- **batch_idx** (*int*) –The index of the current batch in the test loop.
- **color_order** (*str*) –The color order of generated images.
- **target_mean** (*Sequence[Union[float, int]]*) –The original mean of the image tensor before preprocessing. Image will be re-shifted to `target_mean` before visualizing.
- **target_std** (*Sequence[Union[float, int]]*) –The original std of the image tensor before preprocessing. Image will be re-scaled to `target_std` before visualizing.

CHAPTER 75

mmagic.engine.optimizers

75.1 Package Contents

75.1.1 Classes

MultiOptimWrapperConstructor	OptimizerConstructor for GAN models. This class construct optimizer for
PGGANOptimWrapperConstructor	OptimizerConstructor for PGGAN models. Set optimizers for each
SinGANOptimWrapperConstructor	OptimizerConstructor for SinGAN models. Set optimizers for each

class mmagic.engine.optimizers.**MultiOptimWrapperConstructor**(*optim_wrapper_cfg: dict*, *paramwise_cfg=None*)

OptimizerConstructor for GAN models. This class construct optimizer for the submodules of the model separately, and return a `mmengine.optim.OptimWrapperDict` or `mmengine.optim.OptimWrapper`.

Example 1: Build multi optimizers (e.g., GANs):

```
>>> # build GAN model
>>> model = dict(
>>>     type='GANModel',
>>>     num_classes=10,
```

(下页继续)

(续上页)

```
>>>     generator=dict(type='Generator'),
>>>     discriminator=dict(type='Discriminator'))
>>> gan_model = MODELS.build(model)
>>> # build constructor
>>> optim_wrapper = dict(
>>>     generator=dict(
>>>         type='OptimWrapper',
>>>         accumulative_counts=1,
>>>         optimizer=dict(type='Adam', lr=0.0002,
>>>                        betas=(0.5, 0.999))),
>>>     discriminator=dict(
>>>         type='OptimWrapper',
>>>         accumulative_counts=1,
>>>         optimizer=dict(type='Adam', lr=0.0002,
>>>                            betas=(0.5, 0.999))))
>>> optim_dict_builder = MultiOptimWrapperConstructor(optim_wrapper)
>>> # build optim wrapper dict
>>> optim_wrapper_dict = optim_dict_builder(gan_model)
```

Example 2: Build multi optimizers for specific submodules:

```
>>> # build model
>>> class GAN(nn.Module):
>>>     def __init__(self) -> None:
>>>         super().__init__()
>>>         self.generator = nn.Conv2d(3, 3, 1)
>>>         self.discriminator = nn.Conv2d(3, 3, 1)
>>> class TextEncoder(nn.Module):
>>>     def __init__(self):
>>>         super().__init__()
>>>         self.embedding = nn.Embedding(100, 100)
>>> class ToyModel(nn.Module):
>>>     def __init__(self) -> None:
>>>         super().__init__()
>>>         self.m1 = GAN()
>>>         self.m2 = nn.Conv2d(3, 3, 1)
>>>         self.m3 = nn.Linear(2, 2)
>>>         self.text_encoder = TextEncoder()
>>> model = ToyModel()
>>> # build constructor
>>> optim_wrapper = {
>>>     '.*embedding': {
>>>         'type': 'OptimWrapper',
>>>         'optimizer': {
```

(下页继续)

(续上页)

```
>>>             'type': 'Adam',
>>>             'lr': 1e-4,
>>>             'betas': (0.9, 0.99)
>>>         }
>>>     },
>>>     'm1.generator': {
>>>         'type': 'OptimWrapper',
>>>         'optimizer': {
>>>             'type': 'Adam',
>>>             'lr': 1e-5,
>>>             'betas': (0.9, 0.99)
>>>         }
>>>     },
>>>     'm2': {
>>>         'type': 'OptimWrapper',
>>>         'optimizer': {
>>>             'type': 'Adam',
>>>             'lr': 1e-5,
>>>         }
>>>     }
>>> }
>>> optim_dict_builder = MultiOptimWrapperConstructor(optim_wrapper)
>>> # build optim wrapper dict
>>> optim_wrapper_dict = optim_dict_builder(model)
```

Example 3: Build a single optimizer for multi modules (e.g., DreamBooth):

```
>>> # build StableDiffusion model
>>> model = dict(
>>>     type='StableDiffusion',
>>>     unet=dict(type='unet'),
>>>     vae=dict(type='vae'),
        text_encoder=dict(type='text_encoder'))
>>> diffusion_model = MODELS.build(model)
>>> # build constructor
>>> optim_wrapper = dict(
>>>     modules=['unet', 'text_encoder']
>>>     optimizer=dict(type='Adam', lr=0.0002),
>>>     accumulative_counts=1)
>>> optim_dict_builder = MultiOptimWrapperConstructor(optim_wrapper)
>>> # build optim wrapper dict
>>> optim_wrapper_dict = optim_dict_builder(diffusion_model)
```

参数

- **optim_wrapper_cfg_dict** (*dict*) –Config of the optimizer wrapper.
- **paramwise_cfg** (*dict*) –Config of parameter-wise settings. Default: None.

__call__(*module: torch.nn.Module*) → Union[mmengine.optim.OptimWrapperDict, mmengine.optim.OptimWrapper]

Build optimizer and return a optimizer_wrapper_dict.

class mmagic.engine.optimizers.**PGGANOptimWrapperConstructor**(*optim_wrapper_cfg: dict*, *paramwise_cfg: Optional[dict] = None*)

OptimizerConstructor for PGGAN models. Set optimizers for each stage of PGGAN. All submodule must be contained in a `torch.nn.ModuleList` named 'blocks'. And we access each submodule by *MODEL.blocks[SCALE]*, where *MODEL* is generator or discriminator, and the scale is the index of the resolution scale.

More detail about the resolution scale and naming rule please refers to `PGGANGenerator` and `PGGANDiscriminator`.

示例

```
>>> # build PGGAN model
>>> model = dict(
>>>     type='ProgressiveGrowingGAN',
>>>     data_preprocessor=dict(type='GANDataPreprocessor'),
>>>     noise_size=512,
>>>     generator=dict(type='PGGANGenerator', out_scale=1024,
>>>                    noise_size=512),
>>>     discriminator=dict(type='PGGANDiscriminator', in_scale=1024),
>>>     nkimgs_per_scale={
>>>         '4': 600,
>>>         '8': 1200,
>>>         '16': 1200,
>>>         '32': 1200,
>>>         '64': 1200,
>>>         '128': 1200,
>>>         '256': 1200,
>>>         '512': 1200,
>>>         '1024': 12000,
>>>     },
>>>     transition_kimgs=600,
>>>     ema_config=dict(interval=1))
>>> pggan = MODELS.build(model)
>>> # build constructor
```

(下页继续)

(续上页)

```
>>> optim_wrapper = dict(
>>>     generator=dict(optimizer=dict(type='Adam', lr=0.001,
>>>                                   betas=(0., 0.99))),
>>>     discriminator=dict(
>>>         optimizer=dict(type='Adam', lr=0.001, betas=(0., 0.99))),
>>>     lr_schedule=dict(
>>>         generator={
>>>             '128': 0.0015,
>>>             '256': 0.002,
>>>             '512': 0.003,
>>>             '1024': 0.003
>>>         },
>>>         discriminator={
>>>             '128': 0.0015,
>>>             '256': 0.002,
>>>             '512': 0.003,
>>>             '1024': 0.003
>>>         }))
>>> optim_wrapper_dict_builder = PGGANOptimWrapperConstructor(
>>>     optim_wrapper)
>>> # build optim wrapper dict
>>> optim_wrapper_dict = optim_wrapper_dict_builder(pggan)
```

参数

- **optim_wrapper_cfg** (*dict*) –Config of the optimizer wrapper.
- **paramwise_cfg** (*Optional[dict]*) –Parameter-wise options.

__call__(*module: torch.nn.Module*) → mmengine.optim.OptimWrapperDict

Build optimizer and return a optimizerwrapperdict.

class mmagic.engine.optimizers.**SinGANOptimWrapperConstructor**(*optim_wrapper_cfg: dict*, *paramwise_cfg: Optional[dict] = None*)

OptimizerConstructor for SinGAN models. Set optimizers for each submodule of SinGAN. All submodule must be contained in a `torch.nn.ModuleList` named 'blocks'. And we access each submodule by *MODEL.blocks[SCALE]*, where *MODEL* is generator or discriminator, and the scale is the index of the resolution scale.

More detail about the resolution scale and naming rule please refers to `SinGANMultiScaleGenerator` and `SinGANMultiScaleDiscriminator`.

示例

```
>>> # build SinGAN model
>>> model = dict(
>>>     type='SinGAN',
>>>     data_preprocessor=dict(
>>>         type='GANDataPreprocessor',
>>>         non_image_keys=['input_sample']),
>>>     generator=dict(
>>>         type='SinGANMultiScaleGenerator',
>>>         in_channels=3,
>>>         out_channels=3,
>>>         num_scales=2),
>>>     discriminator=dict(
>>>         type='SinGANMultiScaleDiscriminator',
>>>         in_channels=3,
>>>         num_scales=3))
>>> singan = MODELS.build(model)
>>> # build constructor
>>> optim_wrapper = dict(
>>>     generator=dict(optimizer=dict(type='Adam', lr=0.0005,
>>>                                   betas=(0.5, 0.999))),
>>>     discriminator=dict(
>>>         optimizer=dict(type='Adam', lr=0.0005,
>>>                        betas=(0.5, 0.999))))
>>> optim_wrapper_dict_builder = SinGANOptimWrapperConstructor(
>>>     optim_wrapper)
>>> # build optim wrapper dict
>>> optim_wrapper_dict = optim_wrapper_dict_builder(singan)
```

参数

- **optim_wrapper_cfg** (*dict*) –Config of the optimizer wrapper.
- **paramwise_cfg** (*Optional[dict]*) –Parameter-wise options.

__call__ (*module: torch.nn.Module*) → mmengine.optim.OptimWrapperDict

Build optimizer and return a optimizerwrapperdict.

CHAPTER 76

mmagic.engine.runner

76.1 Package Contents

76.1.1 Classes

LogProcessor	LogProcessor inherits from mmengine.runner.LogProcessor and
MultiTestLoop	Test loop for MMagic models which support evaluate multiply dataset at
MultiValLoop	Validation loop for MMagic models which support evaluate multiply

class mmagic.engine.runner.**LogProcessor**(*window_size=10*, *by_epoch=True*, *custom_cfg: Optional[List[dict]] = None*, *num_digits: int = 4*, *log_with_hierarchy: bool = False*, *mean_pattern='.*(loss|time|data_time|grad_norm).*'*)

Bases: mmengine.runner.LogProcessor

LogProcessor inherits from mmengine.runner.LogProcessor and overwrites self.get_log_after_iter().

This log processor should be used along with *mmagic.engine.runner.MultiValLoop* and *mmagic.engine.runner.MultiTestLoop*.

_get_dataloader_size(*runner*, *mode*) → int

Get dataloader size of current loop. In *MultiValLoop* and *MultiTestLoop*, we use *total_length* instead of *len(dataloader)* to denote the total number of iterations.

参数

- **runner** (*Runner*) –The runner of the training/validation/testing
- **mode** (*str*) –Current mode of runner.

返回 The dataloader size of current loop.

返回类型 int

class mmagic.engine.runner.**MultiTestLoop**(*runner*, *dataloader*, *evaluator*, *fp16=False*)

Bases: mmengine.runner.base_loop.BaseLoop

Test loop for MMagic models which support evaluate multiply dataset at the same time. This class support evaluate:

1. Metrics (metric) on a single dataset (e.g. PSNR and SSIM on DIV2K dataset)
2. Different metrics on different datasets (e.g. PSNR on DIV2K and SSIM and PSNR on SET5)

Use cases:

Case 1: metrics on a single dataset

```
>>> # add the following lines in your config
>>> # 1. use `MultiTestLoop` instead of `TestLoop` in MMEngine
>>> val_cfg = dict(type='MultiTestLoop')
>>> # 2. specific MultiEvaluator instead of Evaluator in MMEngine
>>> test_evaluator = dict(
>>>     type='MultiEvaluator',
>>>     metrics=[
>>>         dict(type='PSNR', crop_border=2, prefix='Set5'),
>>>         dict(type='SSIM', crop_border=2, prefix='Set5'),
>>>     ])
>>> # 3. define dataloader
>>> test_dataloader = dict(...)
```

Case 2: different metrics on different datasets

```
>>> # add the following lines in your config
>>> # 1. use `MultiTestLoop` instead of `TestLoop` in MMEngine
>>> Test_cfg = dict(type='MultiTestLoop')
>>> # 2. specific a list MultiEvaluator
>>> # do not forget to add prefix for each metric group
>>> div2k_evaluator = dict(
>>>     type='MultiEvaluator',
>>>     metrics=dict(type='SSIM', crop_border=2, prefix='DIV2K'))
```

(下页继续)

(续上页)

```
>>> set5_evaluator = dict(
>>>     type='MultiEvaluator',
>>>     metrics=[
>>>         dict(type='PSNR', crop_border=2, prefix='Set5'),
>>>         dict(type='SSIM', crop_border=2, prefix='Set5'),
>>>     ])
>>> # define evaluator config
>>> test_evaluator = [div2k_evaluator, set5_evaluator]
>>> # 3. specific a list dataloader for each metric groups
>>> div2k_dataloader = dict(...)
>>> set5_dataloader = dict(...)
>>> # define dataloader config
>>> test_dataloader = [div2k_dataloader, set5_dataloader]
```

参数

- **runner** (*Runner*) –A reference of runner.
- **dataloader** (*Dataloader or dict or list*) –A dataloader object or a dict to build a dataloader a list of dataloader object or a list of config dicts.
- **evaluator** (*Evaluator or dict or list*) –A evaluator object or a dict to build the evaluator or a list of evaluator object or a list of config dicts.

property total_length: int

_build_dataloaders(*dataloader: DATALOADER_TYPE*) → List[torch.utils.data.DataLoader]

Build dataloaders.

参数 **dataloader** (*Dataloader or dict or list*) –A dataloader object or a dict to build a dataloader a list of dataloader object or a list of config dict.

返回 List of dataloaders for compute metrics.

返回类型 List[Dataloader]

_build_evaluators(*evaluator: EVALUATOR_TYPE*) → List[mmengine.evaluator.Evaluator]

Build evaluators.

参数 **evaluator** (*Evaluator or dict or list*) –A evaluator object or a dict to build the evaluator or a list of evaluator object or a list of config dicts.

返回 List of evaluators for compute metrics.

返回类型 List[Evaluator]

run()

Launch validation. The evaluation process consists of four steps.

1. Prepare pre-calculated items for all metrics by calling `self.evaluator.prepare_metrics()`.
2. Get a list of metrics-sampler pair. Each pair contains a list of metrics with the same sampler mode and a shared sampler.
3. Generate images for the each metrics group. Loop for elements in each sampler and feed to the model as input by calling `self.run_iter()`.
4. Evaluate all metrics by calling `self.evaluator.evaluate()`.

run_iter(*idx*, *data_batch: dict*, *metrics: Sequence[mmengine.evaluator.BaseMetric]*)

Iterate one mini-batch and feed the output to corresponding *metrics*.

参数

- **idx** (*int*) –Current idx for the input data.
- **data_batch** (*dict*) –Batch of data from dataloader.
- **metrics** (*Sequence[BaseMetric]*) –Specific metrics to evaluate.

class mmagic.engine.runner.**MultiValLoop**(*runner*, *dataloader: DATALOADER_TYPE*, *evaluator: EVALUATOR_TYPE*, *fp16: bool = False*)

Bases: mmengine.runner.base_loop.BaseLoop

Validation loop for MMagic models which support evaluate multiply dataset at the same time. This class support evaluate:

1. Metrics (metric) on a single dataset (e.g. PSNR and SSIM on DIV2K dataset)
2. Different metrics on different datasets (e.g. PSNR on DIV2K and SSIM and PSNR on SET5)

Use cases:

Case 1: metrics on a single dataset

```
>>> # add the following lines in your config
>>> # 1. use `MultiValLoop` instead of `ValLoop` in MMEngine
>>> val_cfg = dict(type='MultiValLoop')
>>> # 2. specific MultiEvaluator instead of Evaluator in MMEngine
>>> val_evaluator = dict(
>>>     type='MultiEvaluator',
>>>     metrics=[
>>>         dict(type='PSNR', crop_border=2, prefix='Set5'),
>>>         dict(type='SSIM', crop_border=2, prefix='Set5'),
>>>     ])
>>> # 3. define dataloader
>>> val_dataloader = dict(...)
```

Case 2: different metrics on different datasets

```
>>> # add the following lines in your config
>>> # 1. use `MultiValLoop` instead of `ValLoop` in MMEngine
>>> val_cfg = dict(type='MultiValLoop')
>>> # 2. specific a list MultiEvaluator
>>> # do not forget to add prefix for each metric group
>>> div2k_evaluator = dict(
>>>     type='MultiEvaluator',
>>>     metrics=dict(type='SSIM', crop_border=2, prefix='DIV2K'))
>>> set5_evaluator = dict(
>>>     type='MultiEvaluator',
>>>     metrics=[
>>>         dict(type='PSNR', crop_border=2, prefix='Set5'),
>>>         dict(type='SSIM', crop_border=2, prefix='Set5'),
>>>     ])
>>> # define evaluator config
>>> val_evaluator = [div2k_evaluator, set5_evaluator]
>>> # 3. specific a list dataloader for each metric groups
>>> div2k_dataloader = dict(...)
>>> set5_dataloader = dict(...)
>>> # define dataloader config
>>> val_dataloader = [div2k_dataloader, set5_dataloader]
```

参数

- **runner** (*Runner*) –A reference of runner.
- **dataloader** (*Dataloader or dict or list*) –A dataloader object or a dict to build a dataloader a list of dataloader object or a list of config dicts.
- **evaluator** (*Evaluator or dict or list*) –A evaluator object or a dict to build the evaluator or a list of evaluator object or a list of config dicts.

property total_length: int

_build_dataloaders(*dataloader: DATALOADER_TYPE*) → List[torch.utils.data.DataLoader]

Build dataloaders.

参数 **dataloader** (*Dataloader or dict or list*) –A dataloader object or a dict to build a dataloader a list of dataloader object or a list of config dict.

返回 List of dataloaders for compute metrics.

返回类型 List[Dataloader]

_build_evaluators(*evaluator: EVALUATOR_TYPE*) → List[mmengine.evaluator.Evaluator]

Build evaluators.

参数 **evaluator** (*Evaluator or dict or list*) –A evaluator object or a dict to build the evaluator or a list of evaluator object or a list of config dicts.

返回 List of evaluators for compute metrics.

返回类型 List[Evaluator]

run()

Launch validation. The evaluation process consists of four steps.

1. Prepare pre-calculated items for all metrics by calling `self.evaluator.prepare_metrics()`.
2. Get a list of metrics-sampler pair. Each pair contains a list of metrics with the same sampler mode and a shared sampler.
3. Generate images for the each metrics group. Loop for elements in each sampler and feed to the model as input by calling `self.run_iter()`.
4. Evaluate all metrics by calling `self.evaluator.evaluate()`.

run_iter(*idx*, *data_batch: dict*, *metrics: Sequence[mmengine.evaluator.BaseMetric]*)

Iterate one mini-batch and feed the output to corresponding *metrics*.

参数

- **idx** (*int*) –Current idx for the input data.
- **data_batch** (*dict*) –Batch of data from dataloader.
- **metrics** (*Sequence[BaseMetric]*) –Specific metrics to evaluate.

CHAPTER 77

mmagic.engine.schedulers

77.1 Package Contents

77.1.1 Classes

LinearLrInterval	Linear learning rate scheduler for image generation.
ReduceLR	Decays the learning rate of each parameter group by linearly changing

class mmagic.engine.schedulers.**LinearLrInterval**(**args*, *interval=1*, ***kwargs*)

Bases: mmengine.optim.LinearLR

Linear learning rate scheduler for image generation.

In the beginning, the learning rate is ‘start_factor’ defined in mmengine. We give a target learning rate ‘end_factor’ and a start point ‘begin’. If :attr:self.by_epoch is True, ‘begin’ is calculated by epoch, otherwise, calculated by iteration.” Before ‘begin’, we fix learning rate as ‘start_factor’; After ‘begin’, we linearly update learning rate to ‘end_factor’.

参数 **interval** (*int*) –The interval to update the learning rate. Default: 1.

_get_value()

Compute value using chainable form of the scheduler.

class mmagic.engine.schedulers.**ReduceLR**(*optimizer*, *mode: str = 'min'*, *factor: float = 0.1*, *patience: int = 10*, *threshold: float = 0.0001*, *threshold_mode: str = 'rel'*, *cooldown: int = 0*, *min_lr: float = 0.0*, *eps: float = 1e-08*, ***kwargs*)

Bases: mmengine.optim._ParamScheduler

Decays the learning rate of each parameter group by linearly changing small multiplicative factor until the number of epoch reaches a pre-defined milestone: end.

Notice that such decay can happen simultaneously with other changes to the learning rate from outside this scheduler.

备注:

The learning rate of each parameter group will be update at regular intervals.

参数

- **optimizer** (*Optimizer or OptimWrapper*) –Wrapped optimizer.
- **mode** (*str, optional*) –One of *min*, *max*. In *min* mode, lr will be reduced when the quantity monitored has stopped decreasing; in *max* mode it will be reduced when the quantity monitored has stopped increasing. Default: 'min'.
- **factor** (*float, optional*) –Factor by which the learning rate will be reduced. new_lr = lr * factor. Default: 0.1.
- **patience** (*int, optional*) –Number of epochs with no improvement after which learning rate will be reduced. For example, if *patience = 2*, then we will ignore the first 2 epochs with no improvement, and will only decrease the LR after the 3rd epoch if the loss still hasn't improved then. Default: 10.
- **threshold** (*float, optional*) –Threshold for measuring the new optimum, to only focus on significant changes. Default: 1e-4.
- **threshold_mode** (*str, optional*) –One of *rel*, *abs*. In *rel* mode, dynamic_threshold = best * (1 + threshold) in 'max' mode or best * (1 - threshold) in *min* mode. In *abs* mode, dynamic_threshold = best + threshold in *max* mode or best - threshold in *min* mode. Default: 'rel'.
- **cooldown** (*int, optional*) –Number of epochs to wait before resuming normal operation after lr has been reduced. Default: 0.
- **min_lr** (*float, optional*) –Minimum LR value to keep. If LR after decay is lower than *min_lr*, it will be clipped to this value. Default: 0.
- **eps** (*float, optional*) –Minimal decay applied to lr. If the difference between new and old lr is smaller than eps, the update is ignored. Default: 1e-8.

- **begin** (*int*) –Step at which to start updating the learning rate. Defaults to 0.
- **end** (*int*) –Step at which to stop updating the learning rate.
- **last_step** (*int*) –The index of last step. Used for resume without state dict. Defaults to -1.
- **by_epoch** (*bool*) –Whether the scheduled learning rate is updated by epochs. Defaults to True.

property in_cooldown

_get_value()

Compute value using chainable form of the scheduler.

_init_is_better(*mode*)

_reset()

is_better(*a*, *best*)

CHAPTER 78

mmagic.models.archs

78.1 Package Contents

78.1.1 Classes

AllGatherLayer	All gather layer with backward propagation path.
ASPP	ASPP module from DeepLabV3.
AttentionInjection	Wrapper for stable diffusion unet.
SpatialTemporalEnsemble	Apply spatial and temporal ensemble and compute outputs.
SimpleGatedConvModule	Simple Gated Convolutional Module.
ImgNormalize	Normalize images with the given mean and std value.
LinearModule	A linear block that contains linear/norm/activation layers.
LoRAWrapper	Wrapper for LoRA layer.
MultiLayerDiscriminator	Multilayer Discriminator.
PatchDiscriminator	A PatchGAN discriminator.
ResNet	General ResNet.
DepthwiseSeparableConvModule	Depthwise separable convolution module.
SimpleEncoderDecoder	Simple encoder-decoder model from matting.
SoftMaskPatchDiscriminator	A Soft Mask-Guided PatchGAN discriminator.
ResidualBlockNoBN	Residual block without BN.
TokenizerWrapper	Tokenizer wrapper for CLIPTokenizer. Only support CLIPTokenizer
PixelShufflePack	Pixel Shuffle upsample layer.
VGG16	Customized VGG16 Encoder.

78.1.2 Functions

pixel_unshuffle(→ torch.Tensor)	Down-sample by pixel unshuffle.
set_lora(→ torch.nn.Module)	Set LoRA for module.
set_lora_disable(→ torch.nn.Module)	Disable LoRA modules.
set_lora_enable(→ torch.nn.Module)	Enable LoRA modules.
set_only_lora_trainable(→ torch.nn.Module)	Set only LoRA modules trainable.

class mmagic.models.archs.**AllGatherLayer**(**args*, ***kwargs*)

Bases: torch.autograd.Function

All gather layer with backward propagation path.

Indeed, this module is to make dist.all_gather() in the backward graph. Such kind of operation has been widely used in Moco and other contrastive learning algorithms.

static forward(*ctx*, *x*)

Forward function.

static backward(*ctx*, **grad_outputs*)

Backward function.

class mmagic.models.archs.**ASPP**(*in_channels: int*, *out_channels: int = 256*, *mid_channels: int = 256*, *dilations: Sequence[int] = (12, 24, 36)*, *conv_cfg: Optional[dict] = None*, *norm_cfg: Optional[dict] = dict(type='BN')*, *act_cfg: Optional[dict] = dict(type='ReLU')*, *separable_conv: bool = False*)

Bases: torch.nn.Module

ASPP module from DeepLabV3.

The code is adopted from https://github.com/pytorch/vision/blob/master/torchvision/models/segmentation/deeplabv3.py

For more information about the module: "Rethinking Atrous Convolution for Semantic Image Segmentation".

参数

- **in_channels** (*int*) –Input channels of the module.
- **out_channels** (*int*) –Output channels of the module. Default: 256.
- **mid_channels** (*int*) –Output channels of the intermediate ASPP conv modules. Default: 256.
- **dilations** (*Sequence[int]*) –Dilation rate of three ASPP conv module. Default: [12, 24, 36].
- **conv_cfg** (*dict*) –Config dict for convolution layer. If "None", nn.Conv2d will be applied. Default: None.

- **norm_cfg** (*dict*) –Config dict for normalization layer. Default: dict(type='BN').
- **act_cfg** (*dict*) –Config dict for activation layer. Default: dict(type='ReLU').
- **separable_conv** (*bool*) –Whether replace normal conv with depthwise separable conv which is faster. Default: False.

forward(*x: torch.Tensor*) → torch.Tensor

Forward function for ASPP module.

参数 **x** (*Tensor*) –Input tensor with shape (N, C, H, W).

返回 Output tensor.

返回类型 Tensor

class mmagic.models.archs.**AttentionInjection**(*module: torch.nn.Module*, *injection_weight=5*)

Bases: torch.nn.Module

Wrapper for stable diffusion unet.

参数 **module** (*nn.Module*) –The module to be wrapped.

forward(*x: torch.Tensor*, *t*, *encoder_hidden_states=None*, *down_block_additional_residuals=None*, *mid_block_additional_residual=None*, *ref_x=None*) → torch.Tensor

Forward and add LoRA mapping.

参数 **x** (*Tensor*) –The input tensor.

返回 The output tensor.

返回类型 Tensor

mmagic.models.archs.**pixel_unshuffle**(*x: torch.Tensor*, *scale: int*) → torch.Tensor

Down-sample by pixel unshuffle.

参数

- **x** (*Tensor*) –Input tensor.
- **scale** (*int*) –Scale factor.

返回 Output tensor.

返回类型 Tensor

class mmagic.models.archs.**SpatialTemporalEnsemble**(*is_temporal_ensemble: Optional[bool] = False*)

Bases: torch.nn.Module

Apply spatial and temporal ensemble and compute outputs.

参数 **is_temporal_ensemble** (*bool, optional*) –Whether to apply ensemble temporally. If True, the sequence will also be flipped temporally. If the input is an image, this argument must be set to False. Default: False.

_transform(*imgs: torch.Tensor*, *mode: str*) → torch.Tensor

Apply spatial transform (flip, rotate) to the images.

参数

- **imgs** (*torch.Tensor*) –The images to be transformed/
- **mode** (*str*) –The mode of transform. Supported values are ‘vertical’, ‘horizontal’, and ‘transpose’, corresponding to vertical flip, horizontal flip, and rotation, respectively.

返回 Output of the model with spatial ensemble applied.

返回类型 torch.Tensor

spatial_ensemble(*imgs: torch.Tensor*, *model: torch.nn.Module*) → torch.Tensor

Apply spatial ensemble.

参数

- **imgs** (*torch.Tensor*) –The images to be processed by the model. Its size should be either (n, t, c, h, w) or (n, c, h, w).
- **model** (*nn.Module*) –The model to process the images.

返回 Output of the model with spatial ensemble applied.

返回类型 torch.Tensor

forward(*imgs: torch.Tensor*, *model: torch.nn.Module*) → torch.Tensor

Apply spatial and temporal ensemble.

参数

- **imgs** (*torch.Tensor*) –The images to be processed by the model. Its size should be either (n, t, c, h, w) or (n, c, h, w).
- **model** (*nn.Module*) –The model to process the images.

返回 Output of the model with spatial ensemble applied.

返回类型 torch.Tensor

class mmagic.models.archs.**SimpleGatedConvModule**(*in_channels: int*, *out_channels: int*, *kernel_size: Union[int, Tuple[int, int]]*, *feat_act_cfg: Optional[dict] = dict(type='ELU')*, *gate_act_cfg: Optional[dict] = dict(type='Sigmoid')*, ***kwargs*)

Bases: torch.nn.Module

Simple Gated Convolutional Module.

This module is a simple gated convolutional module. The detailed formula is:

$$y = \phi(conv1(x)) * \sigma(conv2(x)),$$

where *phi* is the feature activation function and *sigma* is the gate activation function. In default, the gate activation function is sigmoid.

参数

- **in_channels** (*int*) –Same as nn.Conv2d.
- **out_channels** (*int*) –The number of channels of the output feature. Note that *out_channels* in the conv module is doubled since this module contains two convolutions for feature and gate separately.
- **kernel_size** (*int or tuple[int]*) –Same as nn.Conv2d.
- **feat_act_cfg** (*dict*) –Config dict for feature activation layer. Default: dict(type='ELU').
- **gate_act_cfg** (*dict*) –Config dict for gate activation layer. Default: dict(type='Sigmoid').
- **kwargs** (*keyword arguments*) –Same as *ConvModule*.

forward (*x: torch.Tensor*) → torch.Tensor

Forward Function.

参数 **x** (*torch.Tensor*) –Input tensor with shape of (n, c, h, w).

返回 Output tensor with shape of (n, c, h', w').

返回类型 torch.Tensor

class mmagic.models.archs.**ImgNormalize** (*pixel_range: float*, *img_mean: Tuple[float, float, float]*, *img_std: Tuple[float, float, float]*, *sign: int = - 1*)

Bases: torch.nn.Conv2d

Normalize images with the given mean and std value.

Based on Conv2d layer, can work in GPU.

参数

- **pixel_range** (*float*) –Pixel range of feature.
- **img_mean** (*Tuple[float]*) –Image mean of each channel.
- **img_std** (*Tuple[float]*) –Image std of each channel.
- **sign** (*int*) –Sign of bias. Default -1.

class mmagic.models.archs.**LinearModule**(*in_features: int*, *out_features: int*, *bias: bool = True*, *act_cfg: Optional[dict] = dict(type='ReLU')*, *inplace: bool = True*, *with_spectral_norm: bool = False*, *order: Tuple[str, str] = ('linear', 'act')*)

Bases: torch.nn.Module

A linear block that contains linear/norm/activation layers.

For low level vision, we add spectral norm and padding layer.

参数

- **in_features** (*int*) –Same as nn.Linear.
- **out_features** (*int*) –Same as nn.Linear.
- **bias** (*bool*) –Same as nn.Linear. Default: True.
- **act_cfg** (*dict*) –Config dict for activation layer, "relu" by default.
- **inplace** (*bool*) –Whether to use inplace mode for activation. Default: True.
- **with_spectral_norm** (*bool*) –Whether use spectral norm in linear module. Default: False.
- **order** (*tuple[str]*) –The order of linear/activation layers. It is a sequence of "linear", "norm" and "act". Examples are ("linear", "act") and ("act", "linear").

init_weights() → None

Init weights for the model.

forward(*x: torch.Tensor*, *activate: Optional[bool] = True*) → torch.Tensor

Forward Function.

参数

- **x** (*torch.Tensor*) –Input tensor with shape of $(n, *, c)$. Same as torch.nn.Linear.
- **activate** (*bool, optional*) –Whether to use activation layer. Defaults to True.

返回 Same as torch.nn.Linear.

返回类型 torch.Tensor

class mmagic.models.archs.**LoRAWrapper**(*module: torch.nn.Module*, *in_feat: int*, *out_feat: int*, *rank: int*, *scale: float = 1*, *names: Optional[Union[str, List[str]]] = None*)

Bases: torch.nn.Module

Wrapper for LoRA layer.

参数

- **module** (*nn.Module*) –The module to be wrapped.

- **in_feat** (*int*) –Number of input features.
- **out_feat** (*int*) –Number of output features.
- **rank** (*int*) –The rank of LoRA.
- **scale** (*float*) –The scale of LoRA feature.
- **names** (*Union[str, List[str]], optional*) –The name of LoRA layers. If you want to add multi LoRA for one module, names for each LoRA mapping must be defined.

add_lora(*name: str*, *rank: int*, *scale: float = 1*, *state_dict: Optional[dict] = None*)

Add LoRA mapping.

参数

- **name** (*str*) –The name of added LoRA.
- **rank** (*int*) –The rank of added LoRA.
- **scale** (*float, optional*) –The scale of added LoRA. Defaults to 1.
- **state_dict** (*dict, optional*) –The state dict of added LoRA. Defaults to None.

_set_value(*attr_name: str*, *value: Any*, *name: Optional[str] = None*)

Set value of attribute.

参数

- **attr_name** (*str*) –The name of attribute to be set value.
- **value** (*Any*) –The value to be set.
- **name** (*str, optional*) –The name of field in *attr_name*. If passed, will set value to *attr_name[name]*. Defaults to None.

set_scale(*scale: float*, *name: Optional[str] = None*)

Set LoRA scale.

参数

- **scale** (*float*) –The scale to be set.
- **name** (*str, optional*) –The name of LoRA to be set. Defaults to None.

set_enable(*name: Optional[str] = None*)

Enable LoRA for the current layer.

参数 **name** (*str, optional*) –The name of LoRA to be set. Defaults to None.

set_disable(*name: Optional[str] = None*)

Disable LoRA for the current layer.

参数 **name** (*str, optional*) –The name of LoRA to be set. Defaults to None.

forward_lora_mapping(*x: torch.Tensor*) → torch.Tensor

Forward LoRA mapping.

参数 **x** (*Tensor*) –The input tensor.

返回 The output tensor.

返回类型 Tensor

forward(*x: torch.Tensor*) → torch.Tensor

Forward and add LoRA mapping.

参数 **x** (*Tensor*) –The input tensor.

返回 The output tensor.

返回类型 Tensor

classmethod wrap_lora(*module*, *rank=4*, *scale=1*, *names=None*, *state_dict=None*)

Wrap LoRA.

Use case: »> linear = nn.Linear(2, 4) »> lora_linear = LoRAWrapper.wrap_lora(linear, 4, 1)

参数

- **module** (*nn.Module*) –The module to add LoRA.
- **rank** (*int*) –The rank for LoRA.
- **scale** (*float*) –

返回类型 *LoRAWrapper*

mmagic.models.archs.**set_lora**(*module: torch.nn.Module*, *config: dict*, *verbose: bool = True*) → torch.nn.Module

Set LoRA for module.

Use case: »> 1. set all lora with same parameters »> lora_config = dict(»> rank=4, »> scale=1, »> target_modules=['to_q', 'to_k', 'to_v'])

```
>>> 2. set lora with different parameters
>>> lora_config = dict(
>>>     rank=4,
>>>     scale=1,
>>>     target_modules=[
>>>         # set `to_q` the default parameters
>>>         'to_q',
>>>         # set `to_k` the defined parameters
>>>         dict(target_module='to_k', rank=8, scale=1),
>>>         # set `to_v` the defined `rank` and default `scale`
>>>         dict(target_module='to_v', rank=16)
>>>     ])
```

参数

- **module** (*nn.Module*) –The module to set LoRA.
- **config** (*dict*) –The config dict.
- **verbose** (*bool*) –Whether to print log. Defaults to True.

mmagic.models.archs.**set_lora_disable**(*module: torch.nn.Module*) → torch.nn.Module

Disable LoRA modules.

mmagic.models.archs.**set_lora_enable**(*module: torch.nn.Module*) → torch.nn.Module

Enable LoRA modules.

mmagic.models.archs.**set_only_lora_trainable**(*module: torch.nn.Module*) → torch.nn.Module

Set only LoRA modules trainable.

class mmagic.models.archs.**MultiLayerDiscriminator**(*in_channels: int*, *max_channels: int*, *num_convs: int = 5*, *fc_in_channels: Optional[int] = None*, *fc_out_channels: int = 1024*, *kernel_size: int = 5*, *conv_cfg: Optional[dict] = None*, *norm_cfg: Optional[dict] = None*, *act_cfg: Optional[dict] = dict(type='ReLU')*, *out_act_cfg: Optional[dict] = dict(type='ReLU')*, *with_input_norm: bool = True*, *with_out_convs: bool = False*, *with_spectral_norm: bool = False*, ***kwargs*)

Bases: torch.nn.Module

Multilayer Discriminator.

This is a commonly used structure with stacked multiply convolution layers.

参数

- **in_channels** (*int*) –Input channel of the first input convolution.
- **max_channels** (*int*) –The maximum channel number in this structure.
- **num_conv** (*int*) –Number of stacked intermediate convs (including input conv but excluding output conv). Default to 5.
- **fc_in_channels** (*int | None*) –Input dimension of the fully connected layer. If *fc_in_channels* is None, the fully connected layer will be removed. Default to None.
- **fc_out_channels** (*int*) –Output dimension of the fully connected layer. Default to 1024.

- **kernel_size** (*int*) –Kernel size of the conv modules. Default to 5.
- **conv_cfg** (*dict*) –Config dict to build conv layer.
- **norm_cfg** (*dict*) –Config dict to build norm layer.
- **act_cfg** (*dict*) –Config dict for activation layer, "relu" by default.
- **out_act_cfg** (*dict*) –Config dict for output activation, "relu" by default.
- **with_input_norm** (*bool*) –Whether add normalization after the input conv. Default to True.
- **with_out_convs** (*bool*) –Whether add output convs to the discriminator. The output convs contain two convs. The first out conv has the same setting as the intermediate convs but a stride of 1 instead of 2. The second out conv is a conv similar to the first out conv but reduces the number of channels to 1 and has no activation layer. Default to False.
- **with_spectral_norm** (*bool*) –Whether use spectral norm after the conv layers. Default to False.
- **kwargs** (*keyword arguments*) –

forward(*x: torch.Tensor*) → torch.Tensor

Forward Function.

参数 **x** (*torch.Tensor*) –Input tensor with shape of (n, c, h, w).

返回 Output tensor with shape of (n, c, h', w') or (n, c).

返回类型 torch.Tensor

init_weights(*pretrained: Optional[str] = None*) → None

Init weights for models.

参数 **pretrained** (*str, optional*) –Path for pretrained weights. If given None, pretrained weights will not be loaded. Defaults to None.

class mmagic.models.archs.**PatchDiscriminator**(*in_channels: int*, *base_channels: int = 64*, *num_conv: int = 3*, *norm_cfg: dict = dict(type='BN')*, *init_cfg: Optional[dict] = dict(type='normal', gain=0.02)*)

Bases: mmengine.model.BaseModule

A PatchGAN discriminator.

参数

- **in_channels** (*int*) –Number of channels in input images.
- **base_channels** (*int*) –Number of channels at the first conv layer. Default: 64.

- **num_conv** (*int*) –Number of stacked intermediate convs (excluding input and output conv). Default: 3.
- **norm_cfg** (*dict*) –Config dict to build norm layer. Default: *dict(type='BN')*.
- **init_cfg** (*dict*) –Config dict for initialization. *type*: The name of our initialization method. Default: 'normal'. *gain*: Scaling factor for normal, xavier and orthogonal. Default: 0.02.

forward(*x: torch.Tensor*) → torch.Tensor

Forward function.

参数 **x** (*Tensor*) –Input tensor with shape (n, c, h, w).

返回 Forward results.

返回类型 Tensor

init_weights() → None

Initialize weights for the model.

参数 **pretrained**(*str, optional*) –Path for pretrained weights. If given None, pretrained weights will not be loaded. Default: None.

class mmagic.models.archs.**ResNet**(*depth: int*, *in_channels: int = 3*, *stem_channels: int = 64*, *base_channels: int = 64*, *num_stages: int = 4*, *strides: Sequence[int] = (1, 2, 2, 2)*, *dilations: Sequence[int] = (1, 1, 2, 4)*, *deep_stem: bool = False*, *avg_down: bool = False*, *frozen_stages: int = - 1*, *act_cfg: dict = dict(type='ReLU')*, *conv_cfg: Optional[dict] = None*, *norm_cfg: dict = dict(type='BN')*, *with_cp: bool = False*, *multi_grid: Optional[Sequence[int]] = None*, *contract_dilation: bool = False*, *zero_init_residual: bool = True*)

Bases: torch.nn.Module

General ResNet.

This class is adopted from https://github.com/open-mmlab/mmsegmentation/blob/master/mmseg/models/backbones/resnet.py.

参数

- **depth** (*int*) –Depth of resnet, from {18, 34, 50, 101, 152}.
- **in_channels** (*int*) –Number of input image channels. Default" 3.
- **stem_channels** (*int*) –Number of stem channels. Default: 64.
- **base_channels** (*int*) –Number of base channels of res layer. Default: 64.
- **num_stages** (*int*) –Resnet stages, normally 4.
- **strides** (*Sequence[int]*) –Strides of the first block of each stage. Default: (1, 2, 2, 2).

- **dilations** (*Sequence[int]*) –Dilation of each stage. Default: (1, 1, 2, 4).
- **deep_stem** (*bool*) –Replace 7x7 conv in input stem with 3 3x3 conv. Default: False.
- **avg_down** (*bool*) –Use AvgPool instead of stride conv when downsampling in the bottleneck. Default: False.
- **frozen_stages** (*int*) –Stages to be frozen (stop grad and set eval mode). -1 means not freezing any parameters. Default: -1.
- **act_cfg** (*dict*) –Dictionary to construct and config activation layer. Default: dict(type='ReLU').
- **conv_cfg** (*dict*) –Dictionary to construct and config convolution layer. Default: None.
- **norm_cfg** (*dict*) –Dictionary to construct and config norm layer. Default: dict(type='BN').
- **with_cp** (*bool*) –Use checkpoint or not. Using checkpoint will save some memory while slowing down the training speed. Default: False.
- **multi_grid** (*Sequence[int]|None*) –Multi grid dilation rates of last stage. Default: None.
- **contract_dilation** (*bool*) –Whether contract first dilation of each layer Default: False.
- **zero_init_residual** (*bool*) –Whether to use zero init for last norm layer in resblocks to let them behave as identity. Default: True.

property norm1: torch.nn.Module

normalization layer after the second convolution layer

Type nn.Module

arch_settings

_make_stem_layer(*in_channels: int*, *stem_channels: int*) → None

Make stem layer for ResNet.

_make_layer(*block: BasicBlock*, *planes: int*, *blocks: int*, *stride: int = 1*, *dilation: int = 1*) → torch.nn.Module

_nostride_dilate(*m: torch.nn.Module*, *dilate: int*) → None

init_weights(*pretrained: Optional[str] = None*) → None

Init weights for the model.

参数 **pretrained**(*str, optional*) –Path for pretrained weights. If given None, pretrained weights will not be loaded. Defaults to None.

_freeze_stages() → None

Freeze stages param and norm stats.

forward(*x: torch.Tensor*) → List[torch.Tensor]

Forward function.

参数 **x** (*Tensor*) –Input tensor with shape (N, C, H, W).

返回 Output tensor.

返回类型 Tensor

class mmagic.models.archs.**DepthwiseSeparableConvModule**(*in_channels: int*, *out_channels: int*, *kernel_size: Union[int, Tuple[int, int]]*, *stride: Union[int, Tuple[int, int]] = 1*, *padding: Union[int, Tuple[int, int]] = 0*, *dilation: Union[int, Tuple[int, int]] = 1*, *norm_cfg: Optional[dict] = None*, *act_cfg: Optional[dict] = dict(type='ReLU')*, *dw_norm_cfg: Union[dict, str] = 'default'*, *dw_act_cfg: Union[dict, str] = 'default'*, *pw_norm_cfg: Union[dict, str] = 'default'*, *pw_act_cfg: Union[dict, str] = 'default'*, ***kwargs*)

Bases: torch.nn.Module

Depthwise separable convolution module.

See https://arxiv.org/pdf/1704.04861.pdf for details.

This module can replace a ConvModule with the conv block replaced by two conv block: depthwise conv block and pointwise conv block. The depthwise conv block contains depthwise-conv/norm/activation layers. The pointwise conv block contains pointwise-conv/norm/activation layers. It should be noted that there will be norm/activation layer in the depthwise conv block if norm_cfg and act_cfg are specified.

参数

- **in_channels** (*int*) –Same as nn.Conv2d.
- **out_channels** (*int*) –Same as nn.Conv2d.
- **kernel_size** (*int or tuple[int]*) –Same as nn.Conv2d.
- **stride** (*int or tuple[int]*) –Same as nn.Conv2d. Default: 1.
- **padding** (*int or tuple[int]*) –Same as nn.Conv2d. Default: 0.
- **dilation** (*int or tuple[int]*) –Same as nn.Conv2d. Default: 1.

- **norm_cfg** (*dict*) –Default norm config for both depthwise ConvModule and pointwise ConvModule. Default: None.
- **act_cfg** (*dict*) –Default activation config for both depthwise ConvModule and pointwise ConvModule. Default: dict(type=’ ReLU’).
- **dw_norm_cfg** (*dict*) –Norm config of depthwise ConvModule. If it is ‘default’, it will be the same as `norm_cfg`. Default: ‘default’.
- **dw_act_cfg** (*dict*) –Activation config of depthwise ConvModule. If it is ‘default’, it will be the same as `act_cfg`. Default: ‘default’.
- **pw_norm_cfg** (*dict*) –Norm config of pointwise ConvModule. If it is ‘default’, it will be the same as *norm_cfg*. Default: ‘default’.
- **pw_act_cfg** (*dict*) –Activation config of pointwise ConvModule. If it is ‘default’, it will be the same as `act_cfg`. Default: ‘default’.
- **kwargs** (*optional*) –Other shared arguments for depthwise and pointwise ConvModule. See ConvModule for ref.

forward(*x: torch.Tensor*) → torch.Tensor

Forward function.

参数 **x** (*Tensor*) –Input tensor with shape (N, C, H, W).

返回 Output tensor.

返回类型 Tensor

class mmagic.models.archs.**SimpleEncoderDecoder**(*encoder: dict*, *decoder: dict*, *init_cfg: Optional[dict] = None*)

Bases: `mmengine.model.BaseModule`

Simple encoder-decoder model from matting.

参数

- **encoder** (*dict*) –Config of the encoder.
- **decoder** (*dict*) –Config of the decoder.
- **init_cfg** (*dict, optional*) –Initialization config dict.

forward(**args*, ***kwargs*) → torch.Tensor

Forward function.

返回 The output tensor of the decoder.

返回类型 Tensor

class mmagic.models.archs.**SoftMaskPatchDiscriminator**(*in_channels: int*, *base_channels: Optional[int] = 64*, *num_conv: Optional[int] = 3*, *norm_cfg: Optional[dict] = None*, *init_cfg: Optional[dict] = dict(type='normal', gain=0.02)*, *with_spectral_norm: Optional[bool] = False*)

Bases: mmengine.model.BaseModule

A Soft Mask-Guided PatchGAN discriminator.

参数

- **in_channels** (*int*) –Number of channels in input images.
- **base_channels** (*int, optional*) –Number of channels at the first conv layer. Default: 64.
- **num_conv** (*int, optional*) –Number of stacked intermediate convs (excluding input and output conv). Default: 3.
- **norm_cfg** (*dict, optional*) –Config dict to build norm layer. Default: None.
- **init_cfg** (*dict, optional*) –Config dict for initialization. *type*: The name of our initialization method. Default: 'normal'. *gain*: Scaling factor for normal, xavier and orthogonal. Default: 0.02.
- **with_spectral_norm** (*bool, optional*) –Whether use spectral norm after the conv layers. Default: False.

forward(*x: torch.Tensor*) → torch.Tensor

Forward function.

参数 **x** (*Tensor*) –Input tensor with shape (n, c, h, w).

返回 Forward results.

返回类型 Tensor

init_weights() → None

Initialize weights for the model.

class mmagic.models.archs.**ResidualBlockNoBN**(*mid_channels: int = 64*, *res_scale: float = 1.0*)

Bases: torch.nn.Module

Residual block without BN.

It has a style of:

```
---Conv-ReLU-Conv-+-
 |________________|
```

参数

- **mid_channels** (*int*) –Channel number of intermediate features. Default: 64.
- **res_scale** (*float*) –Used to scale the residual before addition. Default: 1.0.

init_weights() → None

Initialize weights for ResidualBlockNoBN.

Initialization methods like *kaiming_init* are for VGG-style modules. For modules with residual paths, using smaller std is better for stability and performance. We empirically use 0.1. See more details in "ESRGAN: Enhanced Super-Resolution Generative Adversarial Networks"

forward(*x: torch.Tensor*) → torch.Tensor

Forward function.

参数 **x** (*Tensor*) –Input tensor with shape (n, c, h, w).

返回 Forward results.

返回类型 Tensor

class mmagic.models.archs.**TokenizerWrapper**(*from_pretrained: Optional[Union[str, os.PathLike]] = None*, *from_config: Optional[Union[str, os.PathLike]] = None*, **args*, ***kwargs*)

Tokenizer wrapper for CLIPTokenizer. Only support CLIPTokenizer currently. This wrapper is modified from https://github.com/huggingface/diffusers/blob/e51f19aee82c8dd874b715a09dbc521d88835d68/src/diffusers/loaders.py#L358 # noqa.

参数

- **from_pretrained** (*Union[str, os.PathLike], optional*) –The *model id* of a pretrained model or a path to a *directory* containing model weights and config. Defaults to None.
- **from_config** (*Union[str, os.PathLike], optional*) –The *model id* of a pretrained model or a path to a *directory* containing model weights and config. Defaults to None.
- ***args** –If *from_pretrained* is passed, *args and **kwargs will be passed to *from_pretrained* function. Otherwise, *args and **kwargs will be used to initialize the model by *self._module_cls(*args, **kwargs)*.
- ****kwargs** –If *from_pretrained* is passed, *args and **kwargs will be passed to *from_pretrained* function. Otherwise, *args and **kwargs will be used to initialize the model by *self._module_cls(*args, **kwargs)*.

__getattr__(*name: str*) → Any

try_adding_tokens(*tokens: Union[str, List[str]]*, **args*, ***kwargs*)

Attempt to add tokens to the tokenizer.

参数 **tokens** (*Union[str, List[str]]*) –The tokens to be added.

get_token_info (*token: str*) → dict

Get the information of a token, including its start and end index in the current tokenizer.

参数 **token** (*str*) –The token to be queried.

返回

The information of the token, including its start and end index in current tokenizer.

返回类型 dict

add_placeholder_token (*placeholder_token: str*, **args*, *num_vec_per_token: int = 1*, ***kwargs*)

Add placeholder tokens to the tokenizer.

参数

- **placeholder_token** (*str*) –The placeholder token to be added.
- **num_vec_per_token** (*int, optional*) –The number of vectors of the added placeholder token.
- ***args** –The arguments for *self.wrapped.add_tokens*.
- ****kwargs** –The arguments for *self.wrapped.add_tokens*.

replace_placeholder_tokens_in_text (*text: Union[str, List[str]]*, *vector_shuffle: bool = False*, *prop_tokens_to_load: float = 1.0*) → Union[str, List[str]]

Replace the keywords in text with placeholder tokens. This function will be called in *self.__call__* and *self.encode*.

参数

- **text** (*Union[str, List[str]]*) –The text to be processed.
- **vector_shuffle** (*bool, optional*) –Whether to shuffle the vectors. Defaults to False.
- **prop_tokens_to_load** (*float, optional*) –The proportion of tokens to be loaded. If 1.0, all tokens will be loaded. Defaults to 1.0.

返回 The processed text.

返回类型 Union[str, List[str]]

replace_text_with_placeholder_tokens (*text: Union[str, List[str]]*) → Union[str, List[str]]

Replace the placeholder tokens in text with the original keywords. This function will be called in *self.decode*.

参数 **text** (*Union[str, List[str]]*) –The text to be processed.

返回 The processed text.

返回类型 Union[str, List[str]]

__call__(*text: Union[str, List[str]]*, **args*, *vector_shuffle: bool = False*, *prop_tokens_to_load: float = 1.0*, ***kwargs*)

The call function of the wrapper.

参数

- **text** (*Union[str, List[str]]*) –The text to be tokenized.
- **vector_shuffle** (*bool, optional*) –Whether to shuffle the vectors. Defaults to False.
- **prop_tokens_to_load** (*float, optional*) –The proportion of tokens to be loaded. If 1.0, all tokens will be loaded. Defaults to 1.0
- ***args** –The arguments for *self.wrapped.__call__*.
- ****kwargs** –The arguments for *self.wrapped.__call__*.

encode(*text: Union[str, List[str]]*, **args*, ***kwargs*)

Encode the passed text to token index.

参数

- **text** (*Union[str, List[str]]*) –The text to be encode.
- ***args** –The arguments for *self.wrapped.__call__*.
- ****kwargs** –The arguments for *self.wrapped.__call__*.

decode(*token_ids*, *return_raw: bool = False*, **args*, ***kwargs*) → Union[str, List[str]]

Decode the token index to text.

参数

- **token_ids** –The token index to be decoded.
- **return_raw** –Whether keep the placeholder token in the text. Defaults to False.
- ***args** –The arguments for *self.wrapped.decode*.
- ****kwargs** –The arguments for *self.wrapped.decode*.

返回 The decoded text.

返回类型 Union[str, List[str]]

__repr__()

The representation of the wrapper.

class mmagic.models.archs.**PixelShufflePack**(*in_channels: int*, *out_channels: int*, *scale_factor: int*, *upsample_kernel: int*)

Bases: torch.nn.Module

Pixel Shuffle upsample layer.

参数

- **in_channels** (*int*) –Number of input channels.
- **out_channels** (*int*) –Number of output channels.
- **scale_factor** (*int*) –Upsample ratio.
- **upsample_kernel** (*int*) –Kernel size of Conv layer to expand channels.

返回 Upsampled feature map.

init_weights() → None

Initialize weights for PixelShufflePack.

forward(*x: torch.Tensor*) → torch.Tensor

Forward function for PixelShufflePack.

参数 **x** (*Tensor*) –Input tensor with shape (n, c, h, w).

返回 Forward results.

返回类型 Tensor

class mmagic.models.archs.**VGG16**(*in_channels: int*, *batch_norm: Optional[bool] = False*, *aspp: Optional[bool] = False*, *dilations: Optional[List[int]] = None*, *init_cfg: Optional[dict] = None*)

Bases: mmengine.model.BaseModule

Customized VGG16 Encoder.

A 1x1 conv is added after the original VGG16 conv layers. The indices of max pooling layers are returned for unpooling layers in decoders.

参数

- **in_channels** (*int*) –Number of input channels.
- **batch_norm** (*bool, optional*) –Whether use nn.BatchNorm2d. Default to False.
- **aspp** (*bool, optional*) –Whether use ASPP module after the last conv layer. Default to False.
- **dilations** (*list[int], optional*) –Atrous rates of ASPP module. Default to None.
- **init_cfg** (*dict, optional*) –Initialization config dict.

_make_layer(*inplanes: int*, *planes: int*, *convs_layers: int*) → torch.nn.Module

init_weights() → None

Init weights for the model.

forward(*x: torch.Tensor*) → Dict[str, torch.Tensor]

Forward function for ASPP module.

参数 **x** (*Tensor*) –Input tensor with shape (N, C, H, W).

返回 Dict containing output tensor and maxpooling indices.

返回类型 dict

CHAPTER 79

mmagic.models.base_models

79.1 Package Contents

79.1.1 Classes

ExponentialMovingAverage	Implements the exponential moving average (EMA) of the model.
RampUpEMA	Implements the exponential moving average with ramping up momentum.
BaseConditionalGAN	Base class for Conditional GAM models.
BaseEditModel	Base model for image and video editing.
BaseGAN	Base class for GAN models.
BaseMattor	Base class for trimap-based matting models.
BaseTranslationModel	Base Translation Model.
BasicInterpolator	Basic model for video interpolation.
OneStageInpaintor	Standard one-stage inpaintor with commonly used losses.
TwoStageInpaintor	Standard two-stage inpaintor with commonly used losses. A two-stage

class mmagic.models.base_models.**ExponentialMovingAverage**(*model: torch.nn.Module*, *momentum: float = 0.0002*, *interval: int = 1*, *device: Optional[torch.device] = None*, *update_buffers: bool = False*)

Bases: mmengine.model.BaseAveragedModel

Implements the exponential moving average (EMA) of the model.

All parameters are updated by the formula as below:

$$Xema_{t+1} = (1 - momentum) * Xema_t + momentum * X_t$$

参数

- **model** (*nn.Module*) –The model to be averaged.
- **momentum** (*float*) –The momentum used for updating ema parameter. Defaults to 0.0002. Ema's parameter are updated with the formula $averaged\_param = (1 - momentum) * averaged\_param + momentum * source\_param$.
- **interval** (*int*) –Interval between two updates. Defaults to 1.
- **device** (*torch.device, optional*) –If provided, the averaged model will be stored on the device. Defaults to None.
- **update_buffers** (*bool*) –if True, it will compute running averages for both the parameters and the buffers of the model. Defaults to False.

avg_func(*averaged_param: torch.Tensor*, *source_param: torch.Tensor*, *steps: int*) → None

Compute the moving average of the parameters using exponential moving average.

参数

- **averaged_param** (*Tensor*) –The averaged parameters.
- **source_param** (*Tensor*) –The source parameters.
- **steps** (*int*) –The number of times the parameters have been updated.

_load_from_state_dict(*state_dict: dict*, *prefix: str*, *local_metadata: dict*, *strict: bool*, *missing_keys: list*, *unexpected_keys: list*, *error_msgs: List[str]*) → None

Overrides nn.Module._load_from_state_dict to support loading state_dict without wrap ema module with BaseAveragedModel.

In OpenMMLab 1.0, model will not wrap ema submodule with BaseAveragedModel, and the ema weight key in *state_dict* will miss *module* prefix. Therefore, BaseAveragedModel need to automatically add the module prefix if the corresponding key in state_dict misses it.

参数

- **state_dict** (*dict*) –A dict containing parameters and persistent buffers.
- **prefix** (*str*) –The prefix for parameters and buffers used in this module
- **local_metadata** (*dict*) –a dict containing the metadata for this module.
- **strict** (*bool*) –Whether to strictly enforce that the keys in `state_dict` with `prefix` match the names of parameters and buffers in this module
- **missing_keys** (*List[str]*) –if `strict=True`, add missing keys to this list
- **unexpected_keys** (*List[str]*) –if `strict=True`, add unexpected keys to this list
- **error_msgs** (*List[str]*) –error messages should be added to this list, and will be reported together in `load_state_dict()`.

sync_buffers(*model: torch.nn.Module*) → None

Copy buffer from model to averaged model.

参数 **model** (*nn.Module*) –The model whose parameters will be averaged.

sync_parameters(*model: torch.nn.Module*) → None

Copy buffer and parameters from model to averaged model.

参数 **model** (*nn.Module*) –The model whose parameters will be averaged.

class mmagic.models.base_models.**RampUpEMA**(*model: torch.nn.Module*, *interval: int = 1*, *ema_kimg: int = 10*, *ema_rampup: float = 0.05*, *batch_size: int = 32*, *eps: float = 1e-08*, *start_iter: int = 0*, *device: Optional[torch.device] = None*, *update_buffers: bool = False*)

Bases: `mmengine.model.BaseAveragedModel`

Implements the exponential moving average with ramping up momentum.

Ref: https://github.com/NVlabs/stylegan3/blob/master/training/training_loop.py # noqa

参数

- **model** (*nn.Module*) –The model to be averaged.
- **interval** (*int*) –Interval between two updates. Defaults to 1.
- **ema_kimg** (*int, optional*) –EMA kimgs. Defaults to 10.
- **ema_rampup** (*float, optional*) –Ramp up rate. Defaults to 0.05.
- **batch_size** (*int, optional*) –Global batch size. Defaults to 32.
- **eps** (*float, optional*) –Ramp up epsilon. Defaults to 1e-8.
- **start_iter** (*int, optional*) –EMA start iter. Defaults to 0.

- **device** (*torch.device, optional*) –If provided, the averaged model will be stored on the `device`. Defaults to None.
- **update_buffers** (*bool*) –if True, it will compute running averages for both the parameters and the buffers of the model. Defaults to False.

static rampup(*steps*, *ema_kimg=10*, *ema_rampup=0.05*, *batch_size=4*, *eps=1e-08*)

Ramp up ema momentum.

Ref: https://github.com/NVlabs/stylegan3/blob/a5a69f58294509598714d1e88c9646c3d7c6ec94/training/training_loop.py#L300-L308 # noqa

参数

- **steps** –
- **ema_kimg** (*int, optional*) –Half-life of the exponential moving average of generator weights. Defaults to 10.
- **ema_rampup** (*float, optional*) –EMA ramp-up coefficient.If set to None, then rampup will be disabled. Defaults to 0.05.
- **batch_size** (*int, optional*) –Total batch size for one training iteration. Defaults to 4.
- **eps** (*float, optional*) –Epsiolon to avoid `batch_size` divided by zero. Defaults to 1e-8.

返回 Updated momentum.

返回类型 dict

avg_func(*averaged_param: torch.Tensor*, *source_param: torch.Tensor*, *steps: int*) → None

Compute the moving average of the parameters using exponential moving average.

参数

- **averaged_param** (*Tensor*) –The averaged parameters.
- **source_param** (*Tensor*) –The source parameters.
- **steps** (*int*) –The number of times the parameters have been updated.

_load_from_state_dict(*state_dict: dict*, *prefix: str*, *local_metadata: dict*, *strict: bool*, *missing_keys: list*, *unexpected_keys: list*, *error_msgs: List[str]*) → None

Overrides `nn.Module._load_from_state_dict` to support loading `state_dict` without wrap ema module with `BaseAveragedModel`.

In OpenMMLab 1.0, model will not wrap ema submodule with `BaseAveragedModel`, and the ema weight key in *state_dict* will miss *module* prefix. Therefore, `BaseAveragedModel` need to automatically add the `module` prefix if the corresponding key in `state_dict` misses it.

参数

- **state_dict** (*dict*) –A dict containing parameters and persistent buffers.
- **prefix** (*str*) –The prefix for parameters and buffers used in this module
- **local_metadata** (*dict*) –a dict containing the metadata for this module.
- **strict** (*bool*) –Whether to strictly enforce that the keys in `state_dict` with `prefix` match the names of parameters and buffers in this module
- **missing_keys** (*List[str]*) –if `strict=True`, add missing keys to this list
- **unexpected_keys** (*List[str]*) –if `strict=True`, add unexpected keys to this list
- **error_msgs** (*List[str]*) –error messages should be added to this list, and will be reported together in `load_state_dict()`.

sync_buffers(*model: torch.nn.Module*) → None

Copy buffer from model to averaged model.

参数 **model** (*nn.Module*) –The model whose parameters will be averaged.

sync_parameters(*model: torch.nn.Module*) → None

Copy buffer and parameters from model to averaged model.

参数 **model** (*nn.Module*) –The model whose parameters will be averaged.

class mmagic.models.base_models.**BaseConditionalGAN**(*generator: ModelType*, *discriminator: Optional[ModelType] = None*, *data_preprocessor: Optional[Union[dict, mmengine.Config]] = None*, *generator_steps: int = 1*, *discriminator_steps: int = 1*, *noise_size: Optional[int] = None*, *num_classes: Optional[int] = None*, *ema_config: Optional[Dict] = None*, *loss_config: Optional[Dict] = None*)

Bases: `mmagic.models.base_models.base_gan.BaseGAN`

Base class for Conditional GAM models.

参数

- **generator** (*ModelType*) –The config or model of the generator.
- **discriminator** (*Optional[ModelType]*) –The config or model of the discriminator. Defaults to None.
- **data_preprocessor** (*Optional[Union[dict, Config]]*) –The pre-process config or `DataPreprocessor`.
- **generator_steps** (*int*) –The number of times the generator is completely updated before the discriminator is updated. Defaults to 1.

- **discriminator_steps** (*int*) –The number of times the discriminator is completely updated before the generator is updated. Defaults to 1.
- **noise_size** (*Optional[int]*) –Size of the input noise vector. Default to None.
- **num_classes** (*Optional[int]*) –The number classes you would like to generate. Defaults to None.
- **ema_config** (*Optional[Dict]*) –The config for generator's exponential moving average setting. Defaults to None.

label_fn(*label: mmagic.utils.typing.LabelVar = None*, *num_batches: int = 1*) → torch.Tensor

Sampling function for label. There are three scenarios in this function:

- If *label* is a callable function, sample *num_batches* of labels with passed *label*.
- If *label* is *None*, sample *num_batches* of labels in range of *[0, self.num_classes-1]* uniformly.
- If *label* is a *torch.Tensor*, check the range of the tensor is in *[0, self.num_classes-1]*. If all values are in valid range, directly return *label*.

参数

- **label** (*Union[Tensor, Callable, List[int], None]*) –You can directly give a batch of label through a `torch.Tensor` or offer a callable function to sample a batch of label data. Otherwise, the `None` indicates to use the default label sampler. Defaults to *None*.
- **num_batches** (*int, optional*) –The number of batches label want to sample. If *label* is a Tensor, this will be ignored. Defaults to 1.

返回 Sampled label tensor.

返回类型 Tensor

data_sample_to_label(*data_sample:* mmagic.structures.DataSample) → Optional[torch.Tensor]

Get labels from input *data_sample* and pack to *torch.Tensor*. If no label is found in the passed *data_sample*, *None* would be returned.

参数 **data_sample** (*DataSample*) –Input data samples.

返回 Packed label tensor.

返回类型 Optional[torch.Tensor]

static _get_valid_num_classes(*num_classes: Optional[int]*, *generator: ModelType*, *discriminator: Optional[ModelType]*) → int

Try to get the value of *num_classes* from input, *generator* and *discriminator* and check the consistency of these values. If no conflict is found, return the *num_classes*.

参数

- **num_classes** (*Optional[int]*) –*num_classes* passed to *BaseConditionalGAN_refactor*'s initialize function.
- **generator** (*ModelType*) –The config or the model of generator.
- **discriminator** (*Optional[ModelType]*) –The config or model of discriminator.

返回 The number of classes to be generated.

返回类型 int

forward(*inputs: mmagic.utils.typing.ForwardInputs*, *data_samples: Optional[list] = None*, *mode: Optional[str] = None*) → List[*mmagic.structures.DataSample*]

Sample images with the given inputs. If forward mode is 'ema' or 'orig', the image generated by corresponding generator will be returned. If forward mode is 'ema/orig', images generated by original generator and EMA generator will both be returned in a dict.

参数

- **inputs** (*ForwardInputs*) –Dict containing the necessary information (e.g. noise, num_batches, mode) to generate image.
- **data_samples** (*Optional[list]*) –Data samples collated by `data_preprocessor`. Defaults to None.
- **mode** (*Optional[str]*) –*mode* is not used in *BaseConditionalGAN*. Defaults to None.

返回 Generated images or image dict.

返回类型 List[DataSample]

train_generator(*inputs: dict*, *data_samples: List[*mmagic.structures.DataSample*]*, *optimizer_wrapper: mmengine.optim.OptimWrapper*) → Dict[str, torch.Tensor]

Training function for discriminator. All GANs should implement this function by themselves.

参数

- **inputs** (*dict*) –Inputs from dataloader.
- **data_samples** (*List[DataSample]*) –Data samples from dataloader.
- **optim_wrapper** (*OptimWrapper*) –OptimWrapper instance used to update model parameters.

返回 A `dict` of tensor for logging.

返回类型 Dict[str, Tensor]

train_discriminator(*inputs: dict*, *data_samples: List[*mmagic.structures.DataSample*]*, *optimizer_wrapper: mmengine.optim.OptimWrapper*) → Dict[str, torch.Tensor]

Training function for discriminator. All GANs should implement this function by themselves.

参数

- **inputs** (*dict*) –Inputs from dataloader.
- **data_samples** (*List[DataSample]*) –Data samples from dataloader.
- **optim_wrapper** (*OptimWrapper*) –OptimWrapper instance used to update model parameters.

返回 A `dict` of tensor for logging.

返回类型 Dict[str, Tensor]

class mmagic.models.base_models.**BaseEditModel**(*generator: dict*, *pixel_loss: dict*, *train_cfg: Optional[dict] = None*, *test_cfg: Optional[dict] = None*, *init_cfg: Optional[dict] = None*, *data_preprocessor: Optional[dict] = None*)

Bases: `mmengine.model.BaseModel`

Base model for image and video editing.

It must contain a generator that takes frames as inputs and outputs an interpolated frame. It also has a pixel-wise loss for training.

参数

- **generator** (*dict*) –Config for the generator structure.
- **pixel_loss** (*dict*) –Config for pixel-wise loss.
- **train_cfg** (*dict*) –Config for training. Default: None.
- **test_cfg** (*dict*) –Config for testing. Default: None.
- **init_cfg** (*dict, optional*) –The weight initialized config for `BaseModule`.
- **data_preprocessor** (*dict, optional*) –The pre-process config of `BaseDataPreprocessor`.

init_cfg

Initialization config dict.

Type dict, optional

data_preprocessor

Used for pre-processing data sampled by dataloader to the format accepted by *forward()*. Default: None.

Type `BaseDataPreprocessor`

forward(*inputs: torch.Tensor*, *data_samples: Optional[List[*mmagic.structures.DataSample*]] = None*, *mode: str = 'tensor'*, ***kwargs*) → Union[torch.Tensor, List[*mmagic.structures.DataSample*], dict]

Returns losses or predictions of training, validation, testing, and simple inference process.

`forward` method of BaseModel is an abstract method, its subclasses must implement this method.

Accepts `inputs` and `data_samples` processed by *data_preprocessor*, and returns results according to mode arguments.

During non-distributed training, validation, and testing process, `forward` will be called by `BaseModel.train_step`, `BaseModel.val_step` and `BaseModel.val_step` directly.

During distributed data parallel training process, `MMSeparateDistributedDataParallel.train_step` will first call `DistributedDataParallel.forward` to enable automatic gradient synchronization, and then call `forward` to get training loss.

参数

- **inputs** (*torch.Tensor*) –batch input tensor collated by *data_preprocessor*.
- **data_samples** (*List[BaseDataElement], optional*) –data samples collated by *data_preprocessor*.
- **mode** (*str*) –mode should be one of `loss`, `predict` and `tensor`. Default: 'tensor'.
  - `loss`: Called by `train_step` and return loss `dict` used for logging
  - `predict`: Called by `val_step` and `test_step` and return list of `BaseDataElement` results used for computing metric.
  - `tensor`: Called by custom use to get `Tensor` type results.

返回

- If `mode == loss`, return a `dict` of loss tensor used for backward and logging.
- If `mode == predict`, return a `list` of `BaseDataElement` for computing metric and getting inference result.
- If `mode == tensor`, return a tensor or `tuple` of tensor or `dict` or tensor for custom use.

返回类型 ForwardResults

convert_to_datasample(*predictions:* mmagic.structures.DataSample, *data_samples:* mmagic.structures.DataSample, *inputs: Optional[torch.Tensor]*) → List[*mmagic.structures.DataSample*]

Add predictions and destructed inputs (if passed) to data samples.

参数

- **predictions** (*DataSample*) –The predictions of the model.
- **data_samples** (*DataSample*) –The data samples loaded from dataloader.
- **inputs** (*Optional[torch.Tensor]*) –The input of model. Defaults to None.

返回 Modified data samples.

返回类型 List[DataSample]

forward_tensor(*inputs: torch.Tensor*, *data_samples: Optional[List[*mmagic.structures.DataSample*]] = None*, ***kwargs*) → torch.Tensor

Forward tensor. Returns result of simple forward.

参数

- **inputs** (*torch.Tensor*) –batch input tensor collated by *data_preprocessor*.
- **data_samples** (*List[BaseDataElement], optional*) –data samples collated by *data_preprocessor*.

返回 result of simple forward.

返回类型 Tensor

forward_inference(*inputs: torch.Tensor*, *data_samples: Optional[List[*mmagic.structures.DataSample*]] = None*, ***kwargs*) → *mmagic.structures.DataSample*

Forward inference. Returns predictions of validation, testing, and simple inference.

参数

- **inputs** (*torch.Tensor*) –batch input tensor collated by *data_preprocessor*.
- **data_samples** (*List[BaseDataElement], optional*) –data samples collated by *data_preprocessor*.

返回 predictions.

返回类型 DataSample

forward_train(*inputs: torch.Tensor*, *data_samples: Optional[List[*mmagic.structures.DataSample*]] = None*, ***kwargs*) → Dict[str, torch.Tensor]

Forward training. Returns dict of losses of training.

参数

- **inputs** (*torch.Tensor*) –batch input tensor collated by *data_preprocessor*.
- **data_samples** (*List[BaseDataElement], optional*) –data samples collated by *data_preprocessor*.

返回 Dict of losses.

返回类型 dict

class mmagic.models.base_models.**BaseGAN**(*generator: ModelType*, *discriminator: Optional[ModelType] = None*, *data_preprocessor: Optional[Union[dict, mmengine.Config]] = None*, *generator_steps: int = 1*, *discriminator_steps: int = 1*, *noise_size: Optional[int] = None*, *ema_config: Optional[Dict] = None*, *loss_config: Optional[Dict] = None*)

Bases: mmengine.model.BaseModel

Base class for GAN models.

参数

- **generator** (*ModelType*) –The config or model of the generator.
- **discriminator** (*Optional[ModelType]*) –The config or model of the discriminator. Defaults to None.
- **data_preprocessor** (*Optional[Union[dict, Config]]*) –The pre-process config or DataPreprocessor.
- **generator_steps** (*int*) –The number of times the generator is completely updated before the discriminator is updated. Defaults to 1.
- **discriminator_steps** (*int*) –The number of times the discriminator is completely updated before the generator is updated. Defaults to 1.
- **ema_config** (*Optional[Dict]*) –The config for generator's exponential moving average setting. Defaults to None.

property generator_steps: int

The number of times the generator is completely updated before the discriminator is updated.

Type int

property discriminator_steps: int

The number of times the discriminator is completely updated before the generator is updated.

Type int

property device: torch.device

Get current device of the model.

返回 The current device of the model.

返回类型 torch.device

property with_ema_gen: bool

Whether the GAN adopts exponential moving average.

返回

If *True*, means this GAN model is adopted to exponential moving average and vice versa.

返回类型 bool

static gather_log_vars(*log_vars_list: List[Dict[str, torch.Tensor]]*) → Dict[str, torch.Tensor]

Gather a list of log_vars. :param log_vars_list: List[Dict[str, Tensor]]

返回 Dict[str, Tensor]

_init_loss(*loss_config: Optional[Dict] = None*) → None

Initialize customized loss modules.

If loss_config is a dict, we allow kinds of value for each field.

1. ***loss_config* is None: Users will implement all loss calculations** in their own function. Weights for each loss terms are hard coded.
2. ***loss_config* is dict of scalar or string: Users will implement all** loss calculations and use passed *loss_config* to control the weight or behavior of the loss calculation. Users will unpack and use each field in this dict by themselves.

   loss_config = dict(gp_norm_mode='HWC', gp_loss_weight=10)
3. ***loss_config* is dict of dict: Each field in *loss_config* will** used to build a corresponding loss module. And use loss calculation function predefined by *BaseGAN* to calculate the loss.

   loss_config = dict()

示例

loss_config = dict(# *BaseGAN* pre-defined fields gan_loss=dict(type='GANLoss', gan_type='wgan-logistic-ns'), disc_auxiliary_loss=dict(

type='R1GradientPenalty', loss_weight=10. / 2., interval=2, norm_mode='HWC', data_info=dict(

real_data='real_imgs', discriminator='disc')),

gen_auxiliary_loss=dict(type='GeneratorPathRegularizer', loss_weight=2, pl_batch_shrink=2, interval=g_reg_interval, data_info=dict(

generator='gen', num_batches='batch_size')),

user-defined field for loss weights or loss calculation my_loss_2=dict(weight=2, norm_mode='L1'), my_loss_3=2, my_loss_4_norm_type='L2')

参数 **loss_config**(*Optional[Dict], optional*) – Loss config used to build loss modules or define the loss weights. Defaults to None.

noise_fn(*noise: mmagic.utils.typing.NoiseVar = None*, *num_batches: int = 1*)

Sampling function for noise. There are three scenarios in this function:

- If *noise* is a callable function, sample *num_batches* of noise with passed *noise*.
- If *noise* is *None*, sample *num_batches* of noise from gaussian distribution.
- If *noise* is a *torch.Tensor*, directly return *noise*.

参数

- **noise** (*Union[Tensor, Callable, List[int], None]*) –You can directly give a batch of label through a `torch.Tensor` or offer a callable function to sample a batch of label data. Otherwise, the `None` indicates to use the default noise sampler. Defaults to *None*.
- **num_batches** (*int, optional*) –The number of batches label want to sample. If *label* is a Tensor, this will be ignored. Defaults to 1.

返回 Sampled noise tensor.

返回类型 Tensor

_init_ema_model(*ema_config: dict*)

Initialize a EMA model corresponding to the given *ema_config*. If *ema_config* is an empty dict or *None*, EMA model will not be initialized.

参数 **ema_config** (*dict*) –Config to initialize the EMA model.

_get_valid_model(*batch_inputs: mmagic.utils.typing.ForwardInputs*) → str

Try to get the valid forward model from inputs.

- If forward model is defined in *batch_inputs*, it will be used as forward model.
- If forward model is not defined in *batch_inputs*, 'ema' will returned if **:property:'with_ema_gen'** is true. Otherwise, 'orig' will be returned.

参数 **batch_inputs** (*ForwardInputs*) –Inputs passed to *forward()*.

返回

Forward model to generate image. ('orig', 'ema' or ' ema/orig').

返回类型 str

forward(*inputs: mmagic.utils.typing.ForwardInputs*, *data_samples: Optional[list] = None*, *mode: Optional[str] = None*) → mmagic.utils.typing.SampleList

Sample images with the given inputs. If forward mode is 'ema' or 'orig', the image generated by corresponding generator will be returned. If forward mode is 'ema/orig', images generated by original generator and EMA generator will both be returned in a dict.

参数

- **batch_inputs** (*ForwardInputs*) –Dict containing the necessary information (e.g. noise, num_batches, mode) to generate image.
- **data_samples** (*Optional[list]*) –Data samples collated by `data_preprocessor`. Defaults to None.
- **mode** (*Optional[str]*) –*mode* is not used in *BaseGAN*. Defaults to None.

返回 A list of `DataSample` contain generated results.

返回类型 SampleList

val_step(*data: dict*) → mmagic.utils.typing.SampleList

Gets the generated image of given data.

Calls `self.data_preprocessor(data)` and `self(inputs, data_sample, mode=None)` in order. Return the generated results which will be passed to evaluator.

参数 **data** (*dict*) –Data sampled from metric specific sampler. More details in *Metrics* and *Evaluator*.

返回 Generated image or image dict.

返回类型 SampleList

test_step(*data: dict*) → mmagic.utils.typing.SampleList

Gets the generated image of given data. Same as *val_step()*.

参数 **data** (*dict*) –Data sampled from metric specific sampler. More details in *Metrics* and *Evaluator*.

返回 Generated image or image dict.

返回类型 List[DataSample]

train_step(*data: dict*, *optim_wrapper: mmengine.optim.OptimWrapperDict*) → Dict[str, torch.Tensor]

Train GAN model. In the training of GAN models, generator and discriminator are updated alternatively. In MMagic's design, *self.train_step* is called with data input. Therefore we always update discriminator, whose updating is relay on real data, and then determine if the generator needs to be updated based on the current number of iterations. More details about whether to update generator can be found in `should_gen_update()`.

参数

- **data** (*dict*) –Data sampled from dataloader.
- **optim_wrapper** (*OptimWrapperDict*) –OptimWrapperDict instance contains OptimWrapper of generator and discriminator.

返回 A `dict` of tensor for logging.

返回类型 Dict[str, torch.Tensor]

_get_gen_loss(*out_dict*)

_get_disc_loss(*out_dict*)

train_generator(*inputs: dict*, *data_samples: List[*mmagic.structures.DataSample*]*, *optimizer_wrapper: mmengine.optim.OptimWrapper*) → Dict[str, torch.Tensor]

Training function for discriminator. All GANs should implement this function by themselves.

参数

- **inputs** (*dict*) –Inputs from dataloader.
- **data_samples** (*List[DataSample]*) –Data samples from dataloader.
- **optim_wrapper** (*OptimWrapper*) –OptimWrapper instance used to update model parameters.

返回 A dict of tensor for logging.

返回类型 Dict[str, Tensor]

train_discriminator(*inputs: dict*, *data_samples: List[*mmagic.structures.DataSample*]*, *optimizer_wrapper: mmengine.optim.OptimWrapper*) → Dict[str, torch.Tensor]

Training function for discriminator. All GANs should implement this function by themselves.

参数

- **inputs** (*dict*) –Inputs from dataloader.
- **data_samples** (*List[DataSample]*) –Data samples from dataloader.
- **optim_wrapper** (*OptimWrapper*) –OptimWrapper instance used to update model parameters.

返回 A dict of tensor for logging.

返回类型 Dict[str, Tensor]

class mmagic.models.base_models.**BaseMattor**(*data_preprocessor: Union[dict, mmengine.config.Config]*, *backbone: dict*, *init_cfg: Optional[dict] = None*, *train_cfg: Optional[dict] = None*, *test_cfg: Optional[dict] = None*)

Bases: mmengine.model.BaseModel

Base class for trimap-based matting models.

A matting model must contain a backbone which produces *pred_alpha*, a dense prediction with the same height and width of input image. In some cases (such as DIM), the model has a refiner which refines the prediction of the backbone.

Subclasses should overwrite the following functions:

- _forward_train(), to return a loss
- _forward_test(), to return a prediction
- _forward(), to return raw tensors

For test, this base class provides functions to resize inputs and post-process pred_alphas to get predictions

参数

- **backbone** (*dict*) –Config of backbone.

- **data_preprocessor** (*dict*) –Config of data_preprocessor. See `MattorPreprocessor` for details.
- **init_cfg** (*dict, optional*) –Initialization config dict.
- **train_cfg** (*dict*) –Config of training. Customized by subclassesCustomized bu In `train_cfg`, `train_backbone` should be specified. If the model has a refiner, `train_refiner` should be specified.
- **test_cfg** (*dict*) –Config of testing. In `test_cfg`, If the model has a refiner, `train_refiner` should be specified.

resize_inputs(*batch_inputs: torch.Tensor*) → torch.Tensor

Pad or interpolate images and trimaps to multiple of given factor.

restore_size(*pred_alpha: torch.Tensor*, *data_sample:* mmagic.structures.DataSample) → torch.Tensor

Restore the predicted alpha to the original shape.

The shape of the predicted alpha may not be the same as the shape of original input image. This function restores the shape of the predicted alpha.

参数

- **pred_alpha** (*torch.Tensor*) –A single predicted alpha of shape (1, H, W).
- **data_sample** (*DataSample*) –Data sample containing original shape as meta data.

返回 The reshaped predicted alpha.

返回类型 torch.Tensor

postprocess(*batch_pred_alpha: torch.Tensor*, *data_samples:* mmagic.structures.DataSample) → List[*mmagic.structures.DataSample*]

Post-process alpha predictions.

This function contains the following steps:

1. Restore padding or interpolation
2. Mask alpha prediction with trimap
3. Clamp alpha prediction to 0-1
4. Convert alpha prediction to uint8
5. Pack alpha prediction into DataSample

Currently only batch_size 1 is actually supported.

参数

- **batch_pred_alpha** (*torch.Tensor*) –A batch of predicted alpha of shape (N, 1, H, W).
- **data_samples** (*List[DataSample]*) –List of data samples.

返回

A list of predictions. Each data sample contains a pred_alpha, which is a torch.Tensor with dtype=uint8, device=cuda:0

返回类型 List[DataSample]

forward(*inputs: torch.Tensor*, *data_samples: DataSamples = None*, *mode: str = 'tensor'*) → List[*mmagic.structures.DataSample*]

General forward function.

参数

- **inputs** (*torch.Tensor*) –A batch of inputs. with image and trimap concatenated alone channel dimension.
- **data_samples** (*List[DataSample], optional*) –A list of data samples, containing: - Ground-truth alpha / foreground / background to compute loss - other meta information
- **mode** (*str*) –mode should be one of `loss`, `predict` and `tensor`. Default: 'tensor'.
  - `loss`: Called by `train_step` and return loss `dict` used for logging
  - `predict`: Called by `val_step` and `test_step` and return list of `BaseDataElement` results used for computing metric.
  - `tensor`: Called by custom use to get `Tensor` type results.

返回 Sequence of predictions packed into DataElement

返回类型 List[DataElement]

convert_to_datasample(*predictions: List[*mmagic.structures.DataSample*]*, *data_samples:* mmagic.structures.DataSample) → List[*mmagic.structures.DataSample*]

Add predictions to data samples.

参数

- **predictions** (*List[DataSample]*) –The predictions of the model.
- **data_samples** (*DataSample*) –The data samples loaded from dataloader.

返回 Modified data samples.

返回类型 List[DataSample]

class mmagic.models.base_models.**BaseTranslationModel**(*generator*, *discriminator*, *default_domain: str*, *reachable_domains: List[str]*, *related_domains: List[str]*, *data_preprocessor*, *discriminator_steps: int = 1*, *disc_init_steps: int = 0*, *real_img_key: str = 'real_img'*, *loss_config: Optional[dict] = None*)

Bases: mmengine.model.BaseModel

Base Translation Model.

Translation models can transfer images from one domain to another. Domain information like *default_domain*, *reachable_domains* are needed to initialize the class. And we also provide query functions like *is_domain_reachable*, *get_other_domains*.

You can get a specific generator based on the domain, and by specifying *target_domain* in the forward function, you can decide the domain of generated images. Considering the difference among different image translation models, we only provide the external interfaces mentioned above. When you implement image translation with a specific method, you can inherit both *BaseTranslationModel* and the method (e.g BaseGAN) and implement abstract methods.

参数

- **default_domain** (*str*) –Default output domain.
- **reachable_domains** (*list[str]*) –Domains that can be generated by the model.
- **related_domains** (*list[str]*) –Domains involved in training and testing. *reachable_domains* must be contained in *related_domains*. However, related_domains may contain source domains that are used to retrieve source images from data_batch but not in reachable_domains.
- **discriminator_steps** (*int*) –The number of times the discriminator is completely updated before the generator is updated. Defaults to 1.
- **disc_init_steps** (*int*) –The number of initial steps used only to train discriminators.

init_weights()

Initialize weights for the module dict.

参数 **pretrained**(*str, optional*) –Path for pretrained weights. If given None, pretrained weights will not be loaded. Default: None.

get_module(*module*)

Get *nn.ModuleDict* to fit the *MMDistributedDataParallel* interface.

参数 **module** (*MMDistributedDataParallel | nn.ModuleDict*) –The input module that needs processing.

返回 The ModuleDict of multiple networks.

返回类型 nn.ModuleDict

forward(*img*, *test_mode=False*, ***kwargs*)

Forward function.

参数

- **img** (*tensor*) –Input image tensor.
- **test_mode** (*bool*) –Whether in test mode or not. Default: False.
- **kwargs** (*dict*) –Other arguments.

forward_train(*img*, *target_domain*, ***kwargs*)

Forward function for training.

参数

- **img** (*tensor*) –Input image tensor.
- **target_domain** (*str*) –Target domain of output image.
- **kwargs** (*dict*) –Other arguments.

返回 Forward results.

返回类型 dict

forward_test(*img*, *target_domain*, ***kwargs*)

Forward function for testing.

参数

- **img** (*tensor*) –Input image tensor.
- **target_domain** (*str*) –Target domain of output image.
- **kwargs** (*dict*) –Other arguments.

返回 Forward results.

返回类型 dict

is_domain_reachable(*domain*)

Whether image of this domain can be generated.

get_other_domains(*domain*)

get other domains.

_get_target_generator(*domain*)

get target generator.

_get_target_discriminator(*domain*)

get target discriminator.

translation(*image*, *target_domain=None*, ***kwargs*)

Translation Image to target style.

参数

- **image** (*tensor*) –Image tensor with a shape of (N, C, H, W).
- **target_domain** (*str, optional*) –Target domain of output image. Default to None.

返回 Image tensor of target style.

返回类型 dict

class mmagic.models.base_models.**BasicInterpolator**(*generator: dict*, *pixel_loss: dict*, *train_cfg: Optional[dict] = None*, *test_cfg: Optional[dict] = None*, *required_frames: int = 2*, *step_frames: int = 1*, *init_cfg: Optional[dict] = None*, *data_preprocessor: Optional[dict] = None*)

Bases: mmagic.models.base_models.base_edit_model.BaseEditModel

Basic model for video interpolation.

It must contain a generator that takes frames as inputs and outputs an interpolated frame. It also has a pixel-wise loss for training.

参数

- **generator** (*dict*) –Config for the generator structure.
- **pixel_loss** (*dict*) –Config for pixel-wise loss.
- **train_cfg** (*dict*) –Config for training. Default: None.
- **test_cfg** (*dict*) –Config for testing. Default: None.
- **required_frames** (*int*) –Required frames in each process. Default: 2
- **step_frames** (*int*) –Step size of video frame interpolation. Default: 1
- **init_cfg** (*dict, optional*) –The weight initialized config for BaseModule.
- **data_preprocessor** (*dict, optional*) –The pre-process config of BaseDataPreprocessor.

init_cfg

Initialization config dict.

Type dict, optional

data_preprocessor

Used for pre-processing data sampled by dataloader to the format accepted by `forward()`.

Type `BaseDataPreprocessor`

split_frames(*input_tensors: torch.Tensor*) → torch.Tensor

split input tensors for inference.

参数 **input_tensors** (*Tensor*) –Tensor of input frames with shape [1, t, c, h, w]

返回 Split tensor with shape [t-1, 2, c, h, w]

返回类型 Tensor

static merge_frames(*input_tensors: torch.Tensor*, *output_tensors: torch.Tensor*) → list

merge input frames and output frames.

Interpolate a frame between the given two frames.

Merged from [[in1, in2], [in2, in3], [in3, in4], ···] [[out1], [out2], [out3], ···]

to [in1, out1, in2, out2, in3, out3, in4, ···]

参数

- **input_tensors** (*Tensor*) –The input frames with shape [n, 2, c, h, w]
- **output_tensors** (*Tensor*) –The output frames with shape [n, 1, c, h, w].

返回 The final frames.

返回类型 list[np.array]

class mmagic.models.base_models.**OneStageInpaintor**(*data_preprocessor: Union[dict, mmengine.config.Config]*, *encdec: dict*, *disc: Optional[dict] = None*, *loss_gan: Optional[dict] = None*, *loss_gp: Optional[dict] = None*, *loss_disc_shift: Optional[dict] = None*, *loss_composed_percep: Optional[dict] = None*, *loss_out_percep: bool = False*, *loss_l1_hole: Optional[dict] = None*, *loss_l1_valid: Optional[dict] = None*, *loss_tv: Optional[dict] = None*, *train_cfg: Optional[dict] = None*, *test_cfg: Optional[dict] = None*, *init_cfg: Optional[dict] = None*)

Bases: mmengine.model.BaseModel

Standard one-stage inpaintor with commonly used losses.

An inpaintor must contain an encoder-decoder style generator to inpaint masked regions. A discriminator will be adopted when adversarial training is needed.

In this class, we provide a common interface for inpaintors. For other inpaintors, only some funcs may be modified to fit the input style or training schedule.

参数

- **data_preprocessor** (*dict*) –Config of data_preprocessor.
- **encdec** (*dict*) –Config for encoder-decoder style generator.
- **disc** (*dict*) –Config for discriminator.
- **loss_gan** (*dict*) –Config for adversarial loss.
- **loss_gp** (*dict*) –Config for gradient penalty loss.
- **loss_disc_shift** (*dict*) –Config for discriminator shift loss.
- **loss_composed_percep** (*dict*) –Config for perceptual and style loss with composed image as input.
- **loss_out_percep** (*dict*) –Config for perceptual and style loss with direct output as input.
- **loss_l1_hole** (*dict*) –Config for l1 loss in the hole.
- **loss_l1_valid** (*dict*) –Config for l1 loss in the valid region.
- **loss_tv** (*dict*) –Config for total variation loss.

- **train_cfg** (*dict*) –Configs for training scheduler. *disc_step* must be contained for indicates the discriminator updating steps in each training step.
- **test_cfg** (*dict*) –Configs for testing scheduler.
- **init_cfg** (*dict, optional*) –Initialization config dict.

forward(*inputs: torch.Tensor*, *data_samples: Optional[mmagic.utils.SampleList]*, *mode: str = 'tensor'*) → FORWARD_RETURN_TYPE

Forward function.

参数

- **inputs** (*torch.Tensor*) –batch input tensor collated by `data_preprocessor`.
- **data_samples** (*List[BaseDataElement], optional*) –data samples collated by `data_preprocessor`.
- **mode** (*str*) –mode should be one of `loss`, `predict` and `tensor`. Default: 'tensor'.
  - `loss`: Called by `train_step` and return loss `dict` used for logging
  - `predict`: Called by `val_step` and `test_step` and return list of `BaseDataElement` results used for computing metric.
  - `tensor`: Called by custom use to get `Tensor` type results.

返回

- If `mode == loss`, return a `dict` of loss tensor used for backward and logging.
- If `mode == predict`, return a `list` of `BaseDataElement` for computing metric and getting inference result.
- If `mode == tensor`, return a tensor or `tuple` of tensor or `dict` or tensor for custom use.

返回类型 ForwardResults

train_step(*data: List[dict]*, *optim_wrapper: mmengine.optim.OptimWrapperDict*) → dict

Train step function.

In this function, the inpaintor will finish the train step following the pipeline:

1. get fake res/image
2. optimize discriminator (if have)
3. optimize generator

If *self.train_cfg.disc_step > 1*, the train step will contain multiple iterations for optimizing discriminator with different input data and only one iteration for optimizing generator after *disc_step* iterations for discriminator.

参数

- **data** (*List[dict]*) –Batch of data as input.
- **optim_wrapper** (*dict[torch.optim.Optimizer]*) –Dict with optimizers for generator and discriminator (if have).

返回

Dict with loss, information for logger, the number of samples and results for visualization.

返回类型 dict

abstract forward_train(**args*, ***kwargs*) → None

Forward function for training.

In this version, we do not use this interface.

forward_train_d(*data_batch: torch.Tensor*, *is_real: bool*, *is_disc: bool*) → dict

Forward function in discriminator training step.

In this function, we compute the prediction for each data batch (real or fake). Meanwhile, the standard gan loss will be computed with several proposed losses for stable training.

参数

- **data_batch** (*torch.Tensor*) –Batch of real data or fake data.
- **is_real** (*bool*) –If True, the gan loss will regard this batch as real data. Otherwise, the gan loss will regard this batch as fake data.
- **is_disc** (*bool*) –If True, this function is called in discriminator training step. Otherwise, this function is called in generator training step. This will help us to compute different types of adversarial loss, like LSGAN.

返回 Contains the loss items computed in this function.

返回类型 dict

generator_loss(*fake_res: torch.Tensor*, *fake_img: torch.Tensor*, *gt: torch.Tensor*, *mask: torch.Tensor*, *masked_img: torch.Tensor*) → Tuple[dict, dict]

Forward function in generator training step.

In this function, we mainly compute the loss items for generator with the given (fake_res, fake_img). In general, the *fake_res* is the direct output of the generator and the *fake_img* is the composition of direct output and ground-truth image.

参数

- **fake_res** (*torch.Tensor*) –Direct output of the generator.
- **fake_img** (*torch.Tensor*) –Composition of *fake_res* and ground-truth image.
- **gt** (*torch.Tensor*) –Ground-truth image.
- **mask** (*torch.Tensor*) –Mask image.

- **masked_img** (*torch.Tensor*) –Composition of mask image and ground-truth image.

返回 Dict contains the results computed within this function for visualization and dict contains the loss items computed in this function.

返回类型 tuple(dict)

forward_tensor(*inputs: torch.Tensor*, *data_samples: mmagic.utils.SampleList*) → Tuple[torch.Tensor, torch.Tensor]

Forward function in tensor mode.

参数

- **inputs** (*torch.Tensor*) –Input tensor.
- **data_samples** (*List[dict]*) –List of data sample dict.

返回

Direct output of the generator and composition of *fake_res* and ground-truth image.

返回类型 tuple

forward_test(*inputs: torch.Tensor*, *data_samples: mmagic.utils.SampleList*) → *mmagic.structures.DataSample*

Forward function for testing.

参数

- **inputs** (*torch.Tensor*) –Input tensor.
- **data_samples** (*List[dict]*) –List of data sample dict.

返回

List of prediction saved in DataSample.

返回类型 predictions (List[DataSample])

convert_to_datasample(*predictions:* mmagic.structures.DataSample, *data_samples:* mmagic.structures.DataSample, *inputs: Optional[torch.Tensor]*) → List[*mmagic.structures.DataSample*]

Add predictions and destructed inputs (if passed) to data samples.

参数

- **predictions** (*DataSample*) –The predictions of the model.
- **data_samples** (*DataSample*) –The data samples loaded from dataloader.
- **inputs** (*Optional[torch.Tensor]*) –The input of model. Defaults to None.

返回 Modified data samples.

返回类型 List[DataSample]

forward_dummy(*x: torch.Tensor*) → torch.Tensor

Forward dummy function for getting flops.

参数 **x** (*torch.Tensor*) –Input tensor with shape of (n, c, h, w).

返回 Results tensor with shape of (n, 3, h, w).

返回类型 torch.Tensor

class mmagic.models.base_models.**TwoStageInpaintor**(*data_preprocessor: Union[dict, mmengine.config.Config]*, *encdec: dict*, *disc: Optional[dict] = None*, *loss_gan: Optional[dict] = None*, *loss_gp: Optional[dict] = None*, *loss_disc_shift: Optional[dict] = None*, *loss_composed_percep: Optional[dict] = None*, *loss_out_percep: bool = False*, *loss_l1_hole: Optional[dict] = None*, *loss_l1_valid: Optional[dict] = None*, *loss_tv: Optional[dict] = None*, *train_cfg: Optional[dict] = None*, *test_cfg: Optional[dict] = None*, *init_cfg: Optional[dict] = None*, *stage1_loss_type: Optional[Sequence[str]] = ('loss_l1_hole',)*, *stage2_loss_type: Optional[Sequence[str]] = ('loss_l1_hole', 'loss_gan')*, *input_with_ones: bool = True*, *disc_input_with_mask: bool = False*)

Bases: mmagic.models.base_models.one_stage.OneStageInpaintor

Standard two-stage inpaintor with commonly used losses. A two-stage inpaintor contains two encoder-decoder style generators to inpaint masked regions. Currently, we support these loss types in each of two stage inpaintors:

['loss_gan' , 'loss_l1_hole' , 'loss_l1_valid' , 'loss_composed_percep' , 'loss_out_percep' , 'loss_tv'] The *stage1_loss_type* and *stage2_loss_type* should be chosen from these loss types.

参数

- **data_preprocessor** (*dict*) –Config of data_preprocessor.
- **encdec** (*dict*) –Config for encoder-decoder style generator.
- **disc** (*dict*) –Config for discriminator.
- **loss_gan** (*dict*) –Config for adversarial loss.
- **loss_gp** (*dict*) –Config for gradient penalty loss.
- **loss_disc_shift** (*dict*) –Config for discriminator shift loss.

- **loss_composed_percep** (*dict*) –Config for perceptual and style loss with composed image as input.
- **loss_out_percep** (*dict*) –Config for perceptual and style loss with direct output as input.
- **loss_l1_hole** (*dict*) –Config for l1 loss in the hole.
- **loss_l1_valid** (*dict*) –Config for l1 loss in the valid region.
- **loss_tv** (*dict*) –Config for total variation loss.
- **train_cfg** (*dict*) –Configs for training scheduler. *disc_step* must be contained for indicates the discriminator updating steps in each training step.
- **test_cfg** (*dict*) –Configs for testing scheduler.
- **init_cfg** (*dict, optional*) –Initialization config dict.
- **stage1_loss_type** (*tuple[str]*) –Contains the loss names used in the first stage model. Default: (‘loss_l1_hole’).
- **stage2_loss_type** (*tuple[str]*) –Contains the loss names used in the second stage model. Default: (‘loss_l1_hole’, ‘loss_gan’).
- **input_with_ones** (*bool*) –Whether to concatenate an extra ones tensor in input. Default: True.
- **disc_input_with_mask** (*bool*) –Whether to add mask as input in discriminator. Default: False.

forward_tensor(*inputs: torch.Tensor*, *data_samples: mmagic.utils.SampleList*) → Tuple[torch.Tensor, torch.Tensor]

Forward function in tensor mode.

参数

- **inputs** (*torch.Tensor*) –Input tensor.
- **data_samples** (*List[dict]*) –List of data sample dict.

返回 Dict contains output results.

返回类型 dict

two_stage_loss(*stage1_data: dict*, *stage2_data: dict*, *gt: torch.Tensor*, *mask: torch.Tensor*, *masked_img: torch.Tensor*) → Tuple[dict, dict]

Calculate two-stage loss.

参数

- **stage1_data** (*dict*) –Contain stage1 results.
- **stage2_data** (*dict*) –Contain stage2 results..

- **gt** (*torch.Tensor*) –Ground-truth image.
- **mask** (*torch.Tensor*) –Mask image.
- **masked_img** (*torch.Tensor*) –Composition of mask image and ground-truth image.

返回 Dict contains the results computed within this function for visualization and dict contains the loss items computed in this function.

返回类型 tuple(dict)

calculate_loss_with_type(*loss_type: str*, *fake_res: torch.Tensor*, *fake_img: torch.Tensor*, *gt: torch.Tensor*, *mask: torch.Tensor*, *prefix: Optional[str] = 'stage1_'*) → dict

Calculate multiple types of losses.

参数

- **loss_type** (*str*) –Type of the loss.
- **fake_res** (*torch.Tensor*) –Direct results from model.
- **fake_img** (*torch.Tensor*) –Composited results from model.
- **gt** (*torch.Tensor*) –Ground-truth tensor.
- **mask** (*torch.Tensor*) –Mask tensor.
- **prefix** (*str, optional*) –Prefix for loss name. Defaults to ‘stage1_’. # noqa

返回 Contain loss value with its name.

返回类型 dict

train_step(*data: List[dict]*, *optim_wrapper: mmengine.optim.OptimWrapperDict*) → dict

Train step function.

In this function, the inpaintor will finish the train step following the pipeline: 1. get fake res/image 2. optimize discriminator (if have) 3. optimize generator

If *self.train_cfg.disc_step > 1*, the train step will contain multiple iterations for optimizing discriminator with different input data and only one iteration for optimizing generator after *disc_step* iterations for discriminator.

参数

- **data** (*List[dict]*) –Batch of data as input.
- **optim_wrapper** (*dict[torch.optim.Optimizer]*) –Dict with optimizers for generator and discriminator (if have).

返回 Dict with loss, information for logger, the number of samples and results for visualization.

返回类型 dict

CHAPTER 80

mmagic.models.losses

80.1 Package Contents

80.1.1 Classes

AdvLoss	Returns the loss of discriminator with the specified type for
CLIPLoss	Clip loss. In styleclip, this loss is used to optimize the latent code
CharbonnierCompLoss	Charbonnier composition loss.
L1CompositionLoss	L1 composition loss.
MSECompositionLoss	MSE (L2) composition loss.
FaceIdLoss	Face similarity loss. Generally this loss is used to keep the id
LightCNNFeatureLoss	Feature loss of DICGAN, based on LightCNN.
DiscShiftLoss	Disc shift loss.
GANLoss	Define GAN loss.
GaussianBlur	A Gaussian filter which blurs a given tensor with a two-dimensional
GradientPenaltyLoss	Gradient penalty loss for wgan-gp.
GradientLoss	Gradient loss.
CLIPLossComps	Clip loss. In styleclip, this loss is used to optimize the latent code
DiscShiftLossComps	Disc Shift Loss.
FaceIdLossComps	Face similarity loss. Generally this loss is used to keep the id
GANLossComps	Define GAN loss.

80.1.2 Functions

disc_shift_loss(→ torch.Tensor)	Disc Shift loss.
gen_path_regularizer(→ Tuple[torch.Tensor])	Generator Path Regularization.
gradient_penalty_loss(→ torch.Tensor)	Calculate gradient penalty for wgan-gp.
r1_gradient_penalty_loss(→ torch.Tensor)	Calculate R1 gradient penalty for WGAN-GP.
mask_reduce_loss(→ torch.Tensor)	Apply element-wise weight and reduce loss.
reduce_loss(→ torch.Tensor)	Reduce loss as specified.
tv_loss(→ torch.Tensor)	L2 total variation loss, as in Mahendran et al.

class mmagic.models.losses.**AdvLoss**

Returns the loss of discriminator with the specified type for DeblurGanv2.

参数 **loss_type** (*Str*) –One of value in [wgan-gp,lsgan,gan,ragan,ragan-ls].

class mmagic.models.losses.**CLIPLoss**(*loss_weight: float = 1.0*, *data_info: Optional[dict] = None*, *clip_model: dict = dict()*, *loss_name: str = 'loss_clip'*)

Bases: torch.nn.Module

Clip loss. In styleclip, this loss is used to optimize the latent code to generate image that match the text.

In this loss, we may need to provide image, text. Thus, an example of the data_info is:

```
data_info = dict(
    image='fake_imgs',
    text='descriptions')
```

Then, the module will automatically construct this mapping from the input data dictionary.

参数

- **loss_weight** (*float, optional*) –Weight of this loss item. Defaults to 1..
- **data_info** (*dict, optional*) –Dictionary contains the mapping between loss input args and data dictionary. If None, this module will directly pass the input data to the loss function. Defaults to None.
- **clip_model** (*dict, optional*) –Kwargs for clip loss model. Defaults to dict().
- **loss_name** (*str, optional*) –Name of the loss item. If you want this loss item to be included into the backward graph, *loss_* must be the prefix of the name. Defaults to 'loss_clip'.

forward(*image: torch.Tensor*, *text: torch.Tensor*) → torch.Tensor

Forward function.

If `self.data_info` is not `None`, a dictionary containing all of the data and necessary modules should be passed into this function. If this dictionary is given as a non-keyword argument, it should be offered as the first argument. If you are using keyword argument, please name it as *outputs_dict*.

If `self.data_info` is `None`, the input argument or key-word argument will be directly passed to loss function, `third_party_net_loss`.

class mmagic.models.losses.**CharbonnierCompLoss**(*loss_weight: float = 1.0*, *reduction: str = 'mean'*, *sample_wise: bool = False*, *eps: bool = 1e-12*)

Bases: `torch.nn.Module`

Charbonnier composition loss.

参数

- **loss_weight** (*float*) –Loss weight for L1 loss. Default: 1.0.
- **reduction** (*str*) –Specifies the reduction to apply to the output. Supported choices are 'none' | 'mean' | 'sum'. Default: 'mean'.
- **sample_wise** (*bool*) –Whether calculate the loss sample-wise. This argument only takes effect when *reduction* is 'mean' and *weight* (argument of *forward()*) is not None. It will first reduces loss with 'mean' per-sample, and then it means over all the samples. Default: False.
- **eps** (*float*) –A value used to control the curvature near zero. Default: 1e-12.

forward(*pred_alpha: torch.Tensor*, *fg: torch.Tensor*, *bg: torch.Tensor*, *ori_merged: torch.Tensor*, *weight: Optional[torch.Tensor] = None*, ***kwargs*) → torch.Tensor

参数

- **pred_alpha** (*Tensor*) –of shape (N, 1, H, W). Predicted alpha matte.
- **fg** (*Tensor*) –of shape (N, 3, H, W). Tensor of foreground object.
- **bg** (*Tensor*) –of shape (N, 3, H, W). Tensor of background object.
- **ori_merged** (*Tensor*) –of shape (N, 3, H, W). Tensor of origin merged image before normalized by ImageNet mean and std.
- **weight** (*Tensor, optional*) –of shape (N, 1, H, W). It is an indicating matrix: weight[trimap == 128] = 1. Default: None.

class mmagic.models.losses.**L1CompositionLoss**(*loss_weight: float = 1.0*, *reduction: str = 'mean'*, *sample_wise: bool = False*)

Bases: `torch.nn.Module`

L1 composition loss.

参数

- **loss_weight** (*float*) –Loss weight for L1 loss. Default: 1.0.

- **reduction** (*str*) –Specifies the reduction to apply to the output. Supported choices are 'none' | 'mean' | 'sum'. Default: 'mean'.
- **sample_wise** (*bool*) –Whether calculate the loss sample-wise. This argument only takes effect when *reduction* is 'mean' and *weight* (argument of *forward()*) is not None. It will first reduces loss with 'mean' per-sample, and then it means over all the samples. Default: False.

forward(*pred_alpha: torch.Tensor*, *fg: torch.Tensor*, *bg: torch.Tensor*, *ori_merged: torch.Tensor*, *weight: Optional[torch.Tensor] = None*, ***kwargs*) → torch.Tensor

参数

- **pred_alpha** (*Tensor*) –of shape (N, 1, H, W). Predicted alpha matte.
- **fg** (*Tensor*) –of shape (N, 3, H, W). Tensor of foreground object.
- **bg** (*Tensor*) –of shape (N, 3, H, W). Tensor of background object.
- **ori_merged** (*Tensor*) –of shape (N, 3, H, W). Tensor of origin merged image before normalized by ImageNet mean and std.
- **weight** (*Tensor, optional*) –of shape (N, 1, H, W). It is an indicating matrix: weight[trimap == 128] = 1. Default: None.

class mmagic.models.losses.**MSECompositionLoss**(*loss_weight: float = 1.0*, *reduction: str = 'mean'*, *sample_wise: bool = False*)

Bases: torch.nn.Module

MSE (L2) composition loss.

参数

- **loss_weight** (*float*) –Loss weight for MSE loss. Default: 1.0.
- **reduction** (*str*) –Specifies the reduction to apply to the output. Supported choices are 'none' | 'mean' | 'sum'. Default: 'mean'.
- **sample_wise** (*bool*) –Whether calculate the loss sample-wise. This argument only takes effect when *reduction* is 'mean' and *weight* (argument of *forward()*) is not None. It will first reduces loss with 'mean' per-sample, and then it means over all the samples. Default: False.

forward(*pred_alpha: torch.Tensor*, *fg: torch.Tensor*, *bg: torch.Tensor*, *ori_merged: torch.Tensor*, *weight: Optional[torch.Tensor] = None*, ***kwargs*) → torch.Tensor

参数

- **pred_alpha** (*Tensor*) –of shape (N, 1, H, W). Predicted alpha matte.
- **fg** (*Tensor*) –of shape (N, 3, H, W). Tensor of foreground object.
- **bg** (*Tensor*) –of shape (N, 3, H, W). Tensor of background object.

- **ori_merged** (*Tensor*) –of shape (N, 3, H, W). Tensor of origin merged image before normalized by ImageNet mean and std.
- **weight** (*Tensor, optional*) –of shape (N, 1, H, W). It is an indicating matrix: weight[trimap == 128] = 1. Default: None.

class mmagic.models.losses.**FaceIdLoss**(*loss_weight: float = 1.0*, *data_info: Optional[dict] = None*, *facenet: dict = dict(type='ArcFace', ir_se50_weights=None)*, *loss_name: str = 'loss_id'*)

Bases: torch.nn.Module

Face similarity loss. Generally this loss is used to keep the id consistency of the input face image and output face image.

In this loss, we may need to provide `gt`, `pred` and `x`. Thus, an example of the `data_info` is:

```
data_info = dict(
    gt='real_imgs',
    pred='fake_imgs')
```

Then, the module will automatically construct this mapping from the input data dictionary.

参数

- **loss_weight** (*float, optional*) –Weight of this loss item. Defaults to 1..
- **data_info** (*dict, optional*) –Dictionary contains the mapping between loss input args and data dictionary. If None, this module will directly pass the input data to the loss function. Defaults to None.
- **facenet** (*dict, optional*) –Config dict for facenet. Defaults to dict(type='ArcFace', ir_se50_weights=None, device='cuda').
- **loss_name** (*str, optional*) –Name of the loss item. If you want this loss item to be included into the backward graph, *loss_* must be the prefix of the name. Defaults to 'loss_id'.

forward(*pred: torch.Tensor*, *gt: torch.Tensor*) → torch.Tensor

Forward function.

class mmagic.models.losses.**LightCNNFeatureLoss**(*pretrained: str*, *loss_weight: float = 1.0*, *criterion: str = 'l1'*)

Bases: torch.nn.Module

Feature loss of DICGAN, based on LightCNN.

参数

- **pretrained** (*str*) –Path for pretrained weights.
- **loss_weight** (*float*) –Loss weight. Default: 1.0.

- **criterion** (*str*) –Criterion type. Options are 'l1' and 'mse'. Default: 'l1'.

forward(*pred: torch.Tensor*, *gt: torch.Tensor*) → torch.Tensor

Forward function.

参数

- **pred** (*Tensor*) –Predicted tensor.
- **gt** (*Tensor*) –GT tensor.

返回 Forward results.

返回类型 Tensor

class mmagic.models.losses.**DiscShiftLoss**(*loss_weight: float = 0.1*)

Bases: torch.nn.Module

Disc shift loss.

参数 **loss_weight** (*float, optional*) –Loss weight. Defaults to 1.0.

forward(*x: torch.Tensor*) → torch.Tensor

Forward function.

参数 **x** (*Tensor*) –Tensor with shape (n, c, h, w)

返回 Loss.

返回类型 Tensor

class mmagic.models.losses.**GANLoss**(*gan_type: str*, *real_label_val: float = 1.0*, *fake_label_val: float = 0.0*, *loss_weight: float = 1.0*)

Bases: torch.nn.Module

Define GAN loss.

参数

- **gan_type** (*str*) –Support 'vanilla', 'lsgan', 'wgan', 'hinge', 'l1'.
- **real_label_val** (*float*) –The value for real label. Default: 1.0.
- **fake_label_val** (*float*) –The value for fake label. Default: 0.0.
- **loss_weight** (*float*) –Loss weight. Default: 1.0. Note that loss_weight is only for generators; and it is always 1.0 for discriminators.

_wgan_loss(*input: torch.Tensor*, *target: bool*) → torch.Tensor

wgan loss.

参数

- **input** (*Tensor*) –Input tensor.

- **target** (*bool*) –Target label.

返回 wgan loss.

返回类型 Tensor

get_target_label(*input: torch.Tensor*, *target_is_real: bool*) → Union[bool, torch.Tensor]

Get target label.

参数

- **input** (*Tensor*) –Input tensor.
- **target_is_real** (*bool*) –Whether the target is real or fake.

返回

Target tensor. Return bool for wgan, otherwise, return Tensor.

返回类型 (bool | Tensor)

forward(*input: torch.Tensor*, *target_is_real: bool*, *is_disc: bool = False*, *mask: Optional[torch.Tensor] = None*) → torch.Tensor

参数

- **input** (*Tensor*) –The input for the loss module, i.e., the network prediction.
- **target_is_real** (*bool*) –Whether the target is real or fake.
- **is_disc** (*bool*) –Whether the loss for discriminators or not. Default: False.
- **mask** (*Tensor*) –The mask tensor. Default: None.

返回 GAN loss value.

返回类型 Tensor

class mmagic.models.losses.**GaussianBlur**(*kernel_size: Tuple[int, int] = (71, 71)*, *sigma: Tuple[float, float] = (10.0, 10.0)*)

Bases: torch.nn.Module

A Gaussian filter which blurs a given tensor with a two-dimensional gaussian kernel by convolving it along each channel. Batch operation is supported.

This function is modified from kornia.filters.gaussian: *<https://kornia.readthedocs.io/en/latest/_modules/kornia/filters/gaussian.html>*.

参数

- **kernel_size** (*tuple[int]*) –The size of the kernel. Default: (71, 71).
- **sigma** (*tuple[float]*) –The standard deviation of the kernel.
- **Default** (*10.0, 10.0*) –

返回 The Gaussian-blurred tensor.

返回类型 Tensor

Shape:

- input: Tensor with shape of (n, c, h, w)
- output: Tensor with shape of (n, c, h, w)

static compute_zero_padding(*kernel_size: Tuple[int, int]*) → tuple

Compute zero padding tuple.

参数 **kernel_size** (*tuple[int]*) –The size of the kernel.

返回 Padding of height and weight.

返回类型 tuple

get_2d_gaussian_kernel(*kernel_size: Tuple[int, int]*, *sigma: Tuple[float, float]*) → torch.Tensor

Get the two-dimensional Gaussian filter matrix coefficients.

参数

- **kernel_size** (*tuple[int]*) –Kernel filter size in the x and y direction. The kernel sizes should be odd and positive.
- **sigma** (*tuple[int]*) –Gaussian standard deviation in the x and y direction.

返回

A 2D torch tensor with gaussian filter matrix coefficients.

返回类型 kernel_2d (Tensor)

get_1d_gaussian_kernel(*kernel_size: int*, *sigma: float*) → torch.Tensor

Get the Gaussian filter coefficients in one dimension (x or y direction).

参数

- **kernel_size** (*int*) –Kernel filter size in x or y direction. Should be odd and positive.
- **sigma** (*float*) –Gaussian standard deviation in x or y direction.

返回

A 1D torch tensor with gaussian filter coefficients in x or y direction.

返回类型 kernel_1d (Tensor)

gaussian(*kernel_size: int*, *sigma: float*) → torch.Tensor

Gaussian function.

参数

- **kernel_size** (*int*) –Kernel filter size in x or y direction. Should be odd and positive.

- **sigma** (*float*) –Gaussian standard deviation in x or y direction.

返回

A 1D torch tensor with gaussian filter coefficients in x or y direction.

返回类型 Tensor

forward(*x: torch.Tensor*) → torch.Tensor

Forward function.

参数 **x** (*Tensor*) –Tensor with shape (n, c, h, w)

返回 The Gaussian-blurred tensor.

返回类型 Tensor

class mmagic.models.losses.**GradientPenaltyLoss**(*loss_weight: float = 1.0*)

Bases: torch.nn.Module

Gradient penalty loss for wgan-gp.

参数 **loss_weight** (*float*) –Loss weight. Default: 1.0.

forward(*discriminator: torch.nn.Module*, *real_data: torch.Tensor*, *fake_data: torch.Tensor*, *mask: Optional[torch.Tensor] = None*) → torch.Tensor

Forward function.

参数

- **discriminator** (*nn.Module*) –Network for the discriminator.
- **real_data** (*Tensor*) –Real input data.
- **fake_data** (*Tensor*) –Fake input data.
- **mask** (*Tensor*) –Masks for inpainting. Default: None.

返回 Loss.

返回类型 Tensor

mmagic.models.losses.**disc_shift_loss**(*pred: torch.Tensor*) → torch.Tensor

Disc Shift loss.

This loss is proposed in PGGAN as an auxiliary loss for discriminator.

参数 **pred** (*Tensor*) –Input tensor.

返回 loss tensor.

返回类型 torch.Tensor

mmagic.models.losses.**gen_path_regularizer**(*generator: torch.nn.Module*, *num_batches: int*, *mean_path_length: torch.Tensor*, *pl_batch_shrink: int = 1*, *decay: float = 0.01*, *weight: float = 1.0*, *pl_batch_size: Optional[int] = None*, *sync_mean_buffer: bool = False*, *loss_scaler: Optional[torch.cuda.amp.grad_scaler.GradScaler] = None*, *use_apex_amp: bool = False*) → Tuple[torch.Tensor]

Generator Path Regularization.

Path regularization is proposed in StyleGAN2, which can help the improve the continuity of the latent space. More details can be found in: Analyzing and Improving the Image Quality of StyleGAN, CVPR2020.

参数

- **generator** (*nn.Module*) –The generator module. Note that this loss requires that the generator contains `return_latents` interface, with which we can get the latent code of the current sample.
- **num_batches** (*int*) –The number of samples used in calculating this loss.
- **mean_path_length** (*Tensor*) –The mean path length, calculated by moving average.
- **pl_batch_shrink** (*int, optional*) –The factor of shrinking the batch size for saving GPU memory. Defaults to 1.
- **decay** (*float, optional*) –Decay for moving average of mean path length. Defaults to 0.01.
- **weight** (*float, optional*) –Weight of this loss item. Defaults to `1.`.
- **pl_batch_size** (*int | None, optional*) –The batch size in calculating generator path. Once this argument is set, the `num_batches` will be overridden with this argument and won't be affected by `pl_batch_shrink`. Defaults to None.
- **sync_mean_buffer** (*bool, optional*) –Whether to sync mean path length across all of GPUs. Defaults to False.

返回 The penalty loss, detached mean path tensor, and current path length.

返回类型 tuple[Tensor]

mmagic.models.losses.**gradient_penalty_loss**(*discriminator: torch.nn.Module*, *real_data: torch.Tensor*, *fake_data: torch.Tensor*, *mask: Optional[torch.Tensor] = None*, *norm_mode: str = 'pixel'*) → torch.Tensor

Calculate gradient penalty for wgan-gp.

参数

- **discriminator** (*nn.Module*) –Network for the discriminator.
- **real_data** (*Tensor*) –Real input data.
- **fake_data** (*Tensor*) –Fake input data.
- **mask** (*Tensor*) –Masks for inpainting. Default: None.

返回 A tensor for gradient penalty.

返回类型 Tensor

mmagic.models.losses.**r1_gradient_penalty_loss**(*discriminator: torch.nn.Module*, *real_data: torch.Tensor*, *mask: Optional[torch.Tensor] = None*, *norm_mode: str = 'pixel'*, *loss_scaler: Optional[torch.cuda.amp.grad_scaler.GradScaler] = None*, *use_apex_amp: bool = False*) → torch.Tensor

Calculate R1 gradient penalty for WGAN-GP.

R1 regularizer comes from: "Which Training Methods for GANs do actually Converge?" ICML'2018

Different from original gradient penalty, this regularizer only penalized gradient w.r.t. real data.

参数

- **discriminator** (*nn.Module*) –Network for the discriminator.
- **real_data** (*Tensor*) –Real input data.
- **mask** (*Tensor*) –Masks for inpainting. Default: None.
- **norm_mode** (*str*) –This argument decides along which dimension the norm of the gradients will be calculated. Currently, we support ["pixel", "HWC"]. Defaults to "pixel".

返回 A tensor for gradient penalty.

返回类型 Tensor

class mmagic.models.losses.**GradientLoss**(*loss_weight: float = 1.0*, *reduction: str = 'mean'*)

Bases: torch.nn.Module

Gradient loss.

参数

- **loss_weight** (*float*) –Loss weight for L1 loss. Default: 1.0.
- **reduction** (*str*) –Specifies the reduction to apply to the output. Supported choices are 'none' | 'mean' | 'sum'. Default: 'mean'.

forward(*pred: torch.Tensor*, *target: torch.Tensor*, *weight: Optional[torch.Tensor] = None*) → torch.Tensor

参数

- **pred** (*Tensor*) –of shape (N, C, H, W). Predicted tensor.
- **target** (*Tensor*) –of shape (N, C, H, W). Ground truth tensor.
- **weight** (*Tensor, optional*) –of shape (N, C, H, W). Element-wise weights. Default: None.

class mmagic.models.losses.**CLIPLossComps**(*loss_weight: float = 1.0*, *data_info: Optional[dict] = None*, *clip_model: dict = dict()*, *loss_name: str = 'loss_clip'*)

Bases: torch.nn.Module

Clip loss. In styleclip, this loss is used to optimize the latent code to generate image that match the text.

In this loss, we may need to provide image, text. Thus, an example of the data_info is:

```
data_info = dict(
    image='fake_imgs',
    text='descriptions')
```

Then, the module will automatically construct this mapping from the input data dictionary.

参数

- **loss_weight** (*float, optional*) –Weight of this loss item. Defaults to 1..
- **data_info** (*dict, optional*) –Dictionary contains the mapping between loss input args and data dictionary. If None, this module will directly pass the input data to the loss function. Defaults to None.
- **clip_model** (*dict, optional*) –Kwargs for clip loss model. Defaults to dict().
- **loss_name** (*str, optional*) –Name of the loss item. If you want this loss item to be included into the backward graph, *loss_* must be the prefix of the name. Defaults to 'loss_clip' .

forward(**args*, ***kwargs*) → torch.Tensor

Forward function.

If self.data_info is not None, a dictionary containing all of the data and necessary modules should be passed into this function. If this dictionary is given as a non-keyword argument, it should be offered as the first argument. If you are using keyword argument, please name it as *outputs_dict*.

If self.data_info is None, the input argument or key-word argument will be directly passed to loss function, third_party_net_loss.

static loss_name() → str

Loss Name.

This function must be implemented and will return the name of this loss function. This name will be used to combine different loss items by simple sum operation. In addition, if you want this loss item to be included

into the backward graph, *loss_* must be the prefix of the name.

返回 The name of this loss item.

返回类型 str

class mmagic.models.losses.**DiscShiftLossComps**(*loss_weight: float = 1.0*, *data_info: Optional[dict] = None*, *loss_name: str = 'loss_disc_shift'*)

Bases: `torch.nn.Module`

Disc Shift Loss.

This loss is proposed in PGGAN as an auxiliary loss for discriminator.

Note for the design of "data_info": In `MMagic`, almost all of loss modules contain the argument `data_info`, which can be used for constructing the link between the input items (needed in loss calculation) and the data from the generative model. For example, in the training of GAN model, we will collect all of important data/modules into a dictionary:

列表 1: Code from StaticUnconditionalGAN, train_step

```
data_dict_ = dict(
    gen=self.generator,
    disc=self.discriminator,
    disc_pred_fake=disc_pred_fake,
    disc_pred_real=disc_pred_real,
    fake_imgs=fake_imgs,
    real_imgs=real_imgs,
    iteration=curr_iter,
    batch_size=batch_size)
```

But in this loss, we will need to provide `pred` as input. Thus, an example of the `data_info` is:

```
data_info = dict(
    pred='disc_pred_fake')
```

Then, the module will automatically construct this mapping from the input data dictionary.

In addition, in general, `disc_shift_loss` will be applied over real and fake data. In this case, users just need to add this loss module twice, but with different `data_info`. Our model will automatically add these two items.

参数

- **loss_weight** (*float, optional*) –Weight of this loss item. Defaults to `1.`.
- **data_info** (*dict, optional*) –Dictionary contains the mapping between loss input args and data dictionary. If `None`, this module will directly pass the input data to the loss function. Defaults to None.

- **loss_name** (*str, optional*) –Name of the loss item. If you want this loss item to be included into the backward graph, *loss_* must be the prefix of the name. Defaults to 'loss_disc_shift' .

forward(**args*, ***kwargs*) → torch.Tensor

Forward function.

If `self.data_info` is not `None`, a dictionary containing all of the data and necessary modules should be passed into this function. If this dictionary is given as a non-keyword argument, it should be offered as the first argument. If you are using keyword argument, please name it as *outputs_dict*.

If `self.data_info` is `None`, the input argument or key-word argument will be directly passed to loss function, `disc_shift_loss`.

loss_name() → str

Loss Name.

This function must be implemented and will return the name of this loss function. This name will be used to combine different loss items by simple sum operation. In addition, if you want this loss item to be included into the backward graph, *loss_* must be the prefix of the name.

返回 The name of this loss item.

返回类型 str

class mmagic.models.losses.**FaceIdLossComps**(*loss_weight: float = 1.0*, *data_info: Optional[dict] = None*, *facenet: dict = dict(type='ArcFace', ir_se50_weights=None)*, *loss_name: str = 'loss_id'*)

Bases: `torch.nn.Module`

Face similarity loss. Generally this loss is used to keep the id consistency of the input face image and output face image.

In this loss, we may need to provide `gt`, `pred` and `x`. Thus, an example of the `data_info` is:

```
data_info = dict(
    gt='real_imgs',
    pred='fake_imgs')
```

Then, the module will automatically construct this mapping from the input data dictionary.

参数

- **loss_weight** (*float, optional*) –Weight of this loss item. Defaults to `1.`.
- **data_info** (*dict, optional*) –Dictionary contains the mapping between loss input args and data dictionary. If `None`, this module will directly pass the input data to the loss function. Defaults to None.

- **facenet** (*dict, optional*) –Config dict for facenet. Defaults to dict(type='ArcFace', ir_se50_weights=None).
- **loss_name** (*str, optional*) –Name of the loss item. If you want this loss item to be included into the backward graph, *loss_* must be the prefix of the name. Defaults to 'loss_id'.

forward(**args*, ***kwargs*) → torch.Tensor

Forward function.

If `self.data_info` is not `None`, a dictionary containing all of the data and necessary modules should be passed into this function. If this dictionary is given as a non-keyword argument, it should be offered as the first argument. If you are using keyword argument, please name it as *outputs_dict*.

If `self.data_info` is `None`, the input argument or key-word argument will be directly passed to loss function, `third_party_net_loss`.

loss_name() → str

Loss Name.

This function must be implemented and will return the name of this loss function. This name will be used to combine different loss items by simple sum operation. In addition, if you want this loss item to be included into the backward graph, *loss_* must be the prefix of the name.

返回 The name of this loss item.

返回类型 str

class mmagic.models.losses.**GANLossComps**(*gan_type: str*, *real_label_val: float = 1.0*, *fake_label_val: float = 0.0*, *loss_weight: float = 1.0*)

Bases: `torch.nn.Module`

Define GAN loss.

参数

- **gan_type** (*str*) –Support 'vanilla', 'lsgan', 'wgan', 'hinge', 'wgan-logistic-ns'.
- **real_label_val** (*float*) –The value for real label. Default: 1.0.
- **fake_label_val** (*float*) –The value for fake label. Default: 0.0.
- **loss_weight** (*float*) –Loss weight. Default: 1.0. Note that loss_weight is only for generators; and it is always 1.0 for discriminators.

_wgan_loss(*input: torch.Tensor*, *target: bool*) → torch.Tensor

wgan loss.

参数

- **input** (*Tensor*) –Input tensor.

- **target** (*bool*) –Target label.

返回 wgan loss.

返回类型 Tensor

_wgan_logistic_ns_loss(*input: torch.Tensor*, *target: bool*) → torch.Tensor

WGAN loss in logistically non-saturating mode.

This loss is widely used in StyleGANv2.

参数

- **input** (*Tensor*) –Input tensor.
- **target** (*bool*) –Target label.

返回 wgan loss.

返回类型 Tensor

get_target_label(*input: torch.Tensor*, *target_is_real: bool*) → Union[bool, torch.Tensor]

Get target label.

参数

- **input** (*Tensor*) –Input tensor.
- **target_is_real** (*bool*) –Whether the target is real or fake.

返回 Target tensor. Return bool for wgan, otherwise, return Tensor.

返回类型 (bool | Tensor)

forward(*input: torch.Tensor*, *target_is_real: bool*, *is_disc: bool = False*) → torch.Tensor

参数

- **input** (*Tensor*) –The input for the loss module, i.e., the network prediction.
- **target_is_real** (*bool*) –Whether the targe is real or fake.
- **is_disc** (*bool*) –Whether the loss for discriminators or not. Default: False.

返回 GAN loss value.

返回类型 Tensor

class mmagic.models.losses.**GeneratorPathRegularizerComps**(*loss_weight: float = 1.0*, *pl_batch_shrink: int = 1*, *decay: float = 0.01*, *pl_batch_size: Optional[int] = None*, *sync_mean_buffer: bool = False*, *interval: int = 1*, *data_info: Optional[dict] = None*, *use_apex_amp: bool = False*, *loss_name: str = 'loss_path_regular'*)

Bases: torch.nn.Module

Generator Path Regularizer.

Path regularization is proposed in StyleGAN2, which can help the improve the continuity of the latent space. More details can be found in: Analyzing and Improving the Image Quality of StyleGAN, CVPR2020.

Users can achieve lazy regularization by setting interval arguments here.

Note for the design of "data_info": In MMagic, almost all of loss modules contain the argument data_info, which can be used for constructing the link between the input items (needed in loss calculation) and the data from the generative model. For example, in the training of GAN model, we will collect all of important data/modules into a dictionary:

列表 2: Code from StaticUnconditionalGAN, train_step

```
data_dict_ = dict(
    gen=self.generator,
    disc=self.discriminator,
    fake_imgs=fake_imgs,
    disc_pred_fake_g=disc_pred_fake_g,
    iteration=curr_iter,
    batch_size=batch_size)
```

But in this loss, we will need to provide generator and num_batches as input. Thus an example of the data_info is:

```
data_info = dict(
    generator='gen',
    num_batches='batch_size')
```

Then, the module will automatically construct this mapping from the input data dictionary.

参数

- **loss_weight** (*float, optional*) –Weight of this loss item. Defaults to 1..

- **pl_batch_shrink** (*int, optional*) – The factor of shrinking the batch size for saving GPU memory. Defaults to 1.
- **decay** (*float, optional*) – Decay for moving average of mean path length. Defaults to 0.01.
- **pl_batch_size** (*int | None, optional*) – The batch size in calculating generator path. Once this argument is set, the `num_batches` will be overridden with this argument and won't be affected by `pl_batch_shrink`. Defaults to None.
- **sync_mean_buffer** (*bool, optional*) – Whether to sync mean path length across all of GPUs. Defaults to False.
- **interval** (*int, optional*) – The interval of calculating this loss. This argument is used to support lazy regularization. Defaults to 1.
- **data_info** (*dict, optional*) – Dictionary contains the mapping between loss input args and data dictionary. If `None`, this module will directly pass the input data to the loss function. Defaults to None.
- **loss_name** (*str, optional*) – Name of the loss item. If you want this loss item to be included into the backward graph, *loss_* must be the prefix of the name. Defaults to 'loss_path_regular'.

forward(**args*, ***kwargs*) → torch.Tensor

Forward function.

If `self.data_info` is not `None`, a dictionary containing all of the data and necessary modules should be passed into this function. If this dictionary is given as a non-keyword argument, it should be offered as the first argument. If you are using keyword argument, please name it as *outputs_dict*.

If `self.data_info` is `None`, the input argument or key-word argument will be directly passed to loss function, `gen_path_regularizer`.

loss_name() → str

Loss Name.

This function must be implemented and will return the name of this loss function. This name will be used to combine different loss items by simple sum operation. In addition, if you want this loss item to be included into the backward graph, *loss_* must be the prefix of the name.

返回 The name of this loss item.

返回类型 str

class mmagic.models.losses.**GradientPenaltyLossComps**(*loss_weight: float = 1.0*, *norm_mode: str = 'pixel'*, *data_info: Optional[dict] = None*, *loss_name: str = 'loss_gp'*)

Bases: `torch.nn.Module`

Gradient Penalty for WGAN-GP.

In the detailed implementation, there are two streams where one uses the pixel-wise gradient norm, but the other adopts normalization along instance (HWC) dimensions. Thus, `norm_mode` are offered to define which mode you want.

Note for the design of "data_info": In `MMagic`, almost all of loss modules contain the argument `data_info`, which can be used for constructing the link between the input items (needed in loss calculation) and the data from the generative model. For example, in the training of GAN model, we will collect all of important data/modules into a dictionary:

列表 3: Code from StaticUnconditionalGAN, train_step

```
data_dict_ = dict(
    gen=self.generator,
    disc=self.discriminator,
    disc_pred_fake=disc_pred_fake,
    disc_pred_real=disc_pred_real,
    fake_imgs=fake_imgs,
    real_imgs=real_imgs,
    iteration=curr_iter,
    batch_size=batch_size)
```

But in this loss, we will need to provide `discriminator`, `real_data`, and `fake_data` as input. Thus, an example of the `data_info` is:

```
data_info = dict(
    discriminator='disc',
    real_data='real_imgs',
    fake_data='fake_imgs')
```

Then, the module will automatically construct this mapping from the input data dictionary.

参数

- **loss_weight** (*float, optional*) –Weight of this loss item. Defaults to `1.`.
- **data_info** (*dict, optional*) –Dictionary contains the mapping between loss input args and data dictionary. If `None`, this module will directly pass the input data to the loss function. Defaults to None.
- **norm_mode** (*str*) –This argument decides along which dimension the norm of the gradients will be calculated. Currently, we support ["pixel" , "HWC"]. Defaults to "pixel" .
- **loss_name** (*str, optional*) –Name of the loss item. If you want this loss item to be included into the backward graph, *loss_* must be the prefix of the name. Defaults to 'loss_gp' .

forward(**args*, ***kwargs*) → torch.Tensor

Forward function.

If `self.data_info` is not `None`, a dictionary containing all of the data and necessary modules should be passed into this function. If this dictionary is given as a non-keyword argument, it should be offered as the first argument. If you are using keyword argument, please name it as *outputs_dict*.

If `self.data_info` is `None`, the input argument or key-word argument will be directly passed to loss function, `gradient_penalty_loss`.

loss_name() → str

Loss Name.

This function must be implemented and will return the name of this loss function. This name will be used to combine different loss items by simple sum operation. In addition, if you want this loss item to be included into the backward graph, *loss_* must be the prefix of the name.

返回 The name of this loss item.

返回类型 str

class mmagic.models.losses.**R1GradientPenaltyComps**(*loss_weight: float = 1.0*, *norm_mode: str = 'pixel'*, *interval: int = 1*, *data_info: Optional[dict] = None*, *use_apex_amp: bool = False*, *loss_name: str = 'loss_r1_gp'*)

Bases: `torch.nn.Module`

R1 gradient penalty for WGAN-GP.

R1 regularizer comes from: "Which Training Methods for GANs do actually Converge?" ICML'2018

Different from original gradient penalty, this regularizer only penalized gradient w.r.t. real data.

Note for the design of "data_info": In `MMagic`, almost all of loss modules contain the argument `data_info`, which can be used for constructing the link between the input items (needed in loss calculation) and the data from the generative model. For example, in the training of GAN model, we will collect all of important data/modules into a dictionary:

列表 4: Code from StaticUnconditionalGAN, train_step

```
data_dict_ = dict(
    gen=self.generator,
    disc=self.discriminator,
    disc_pred_fake=disc_pred_fake,
    disc_pred_real=disc_pred_real,
    fake_imgs=fake_imgs,
    real_imgs=real_imgs,
```

(下页继续)

(续上页)

```
    iteration=curr_iter,
    batch_size=batch_size)
```

But in this loss, we will need to provide `discriminator` and `real_data` as input. Thus, an example of the `data_info` is:

```
data_info = dict(
    discriminator='disc',
    real_data='real_imgs')
```

Then, the module will automatically construct this mapping from the input data dictionary.

参数

- **loss_weight** (*float, optional*) –Weight of this loss item. Defaults to `1.`.
- **data_info** (*dict, optional*) –Dictionary contains the mapping between loss input args and data dictionary. If `None`, this module will directly pass the input data to the loss function. Defaults to None.
- **norm_mode** (*str*) –This argument decides along which dimension the norm of the gradients will be calculated. Currently, we support ["pixel" , "HWC"]. Defaults to "pixel" .
- **interval** (*int, optional*) –The interval of calculating this loss. Defaults to 1.
- **loss_name** (*str, optional*) –Name of the loss item. If you want this loss item to be included into the backward graph, *loss_* must be the prefix of the name. Defaults to 'loss_r1_gp' .

forward(**args*, ***kwargs*) → torch.Tensor

Forward function.

If `self.data_info` is not `None`, a dictionary containing all of the data and necessary modules should be passed into this function. If this dictionary is given as a non-keyword argument, it should be offered as the first argument. If you are using keyword argument, please name it as *outputs_dict*.

If `self.data_info` is `None`, the input argument or key-word argument will be directly passed to loss function, `r1_gradient_penalty_loss`.

loss_name() → str

Loss Name.

This function must be implemented and will return the name of this loss function. This name will be used to combine different loss items by simple sum operation. In addition, if you want this loss item to be included into the backward graph, *loss_* must be the prefix of the name.

返回 The name of this loss item.

返回类型 str

mmagic.models.losses.**mask_reduce_loss**(*loss: torch.Tensor*, *weight: Optional[torch.Tensor] = None*, *reduction: str = 'mean'*, *sample_wise: bool = False*) → torch.Tensor

Apply element-wise weight and reduce loss.

参数

- **loss** (*Tensor*) –Element-wise loss.
- **weight** (*Tensor*) –Element-wise weights. Default: None.
- **reduction** (*str*) –Same as built-in losses of PyTorch. Options are "none", "mean" and "sum". Default: 'mean'.
- **sample_wise** (*bool*) –Whether calculate the loss sample-wise. This argument only takes effect when *reduction* is 'mean' and *weight* (argument of *forward()*) is not None. It will first reduces loss with 'mean' per-sample, and then it means over all the samples. Default: False.

返回 Processed loss values.

返回类型 Tensor

mmagic.models.losses.**reduce_loss**(*loss: torch.Tensor*, *reduction: str*) → torch.Tensor

Reduce loss as specified.

参数

- **loss** (*Tensor*) –Elementwise loss tensor.
- **reduction** (*str*) –Options are "none", "mean" and "sum".

返回 Reduced loss tensor.

返回类型 Tensor

class mmagic.models.losses.**PerceptualLoss**(*layer_weights: dict*, *layer_weights_style: Optional[dict] = None*, *vgg_type: str = 'vgg19'*, *use_input_norm: bool = True*, *perceptual_weight: float = 1.0*, *style_weight: float = 1.0*, *norm_img: bool = True*, *pretrained: str = 'torchvision://vgg19'*, *criterion: str = 'l1'*)

Bases: torch.nn.Module

Perceptual loss with commonly used style loss.

参数

- **layers_weights** (*dict*) –The weight for each layer of vgg feature for perceptual loss. Here is an example: {'4': 1., '9': 1., '18': 1.}, which means the 5th, 10th and 18th feature layer will be extracted with weight 1.0 in calculating losses.
- **layers_weights_style** (*dict*) –The weight for each layer of vgg feature for style loss. If set to 'None', the weights are set equal to the weights for perceptual loss. Default: None.

- **vgg_type** (*str*) –The type of vgg network used as feature extractor. Default: 'vgg19'.
- **use_input_norm** (*bool*) –If True, normalize the input image in vgg. Default: True.
- **perceptual_weight** (*float*) –If *perceptual_weight > 0*, the perceptual loss will be calculated and the loss will multiplied by the weight. Default: 1.0.
- **style_weight** (*float*) –If *style_weight > 0*, the style loss will be calculated and the loss will multiplied by the weight. Default: 1.0.
- **norm_img** (*bool*) –If True, the image will be normed to [0, 1]. Note that this is different from the *use_input_norm* which norm the input in in forward function of vgg according to the statistics of dataset. Importantly, the input image must be in range [-1, 1].
- **pretrained** (*str*) –Path for pretrained weights. Default: 'torchvision://vgg19'.
- **criterion** (*str*) –Criterion type. Options are 'l1' and 'mse'. Default: 'l1'.

forward(*x: torch.Tensor*, *gt: torch.Tensor*) → Tuple[torch.Tensor]

Forward function.

参数

- **x** (*Tensor*) –Input tensor with shape (n, c, h, w).
- **gt** (*Tensor*) –Ground-truth tensor with shape (n, c, h, w).

返回 Forward results.

返回类型 Tensor

_gram_mat(*x: torch.Tensor*) → torch.Tensor

Calculate Gram matrix.

参数 **x** (*torch.Tensor*) –Tensor with shape of (n, c, h, w).

返回 Gram matrix.

返回类型 torch.Tensor

class mmagic.models.losses.**PerceptualVGG**(*layer_name_list: List[str]*, *vgg_type: str = 'vgg19'*, *use_input_norm: bool = True*, *pretrained: str = 'torchvision://vgg19'*)

Bases: torch.nn.Module

VGG network used in calculating perceptual loss.

In this implementation, we allow users to choose whether use normalization in the input feature and the type of vgg network. Note that the pretrained path must fit the vgg type.

参数

- **layer_name_list** (*list[str]*) –According to the name in this list, forward function will return the corresponding features. This list contains the name each layer in *vgg.feature*. An example of this list is [‘4’, ‘10’].
- **vgg_type** (*str*) –Set the type of vgg network. Default: ‘vgg19’.
- **use_input_norm** (*bool*) –If True, normalize the input image. Importantly, the input feature must in the range [0, 1]. Default: True.
- **pretrained** (*str*) –Path for pretrained weights. Default: ‘torchvision://vgg19’

forward(*x: torch.Tensor*) → torch.Tensor

Forward function.

参数 **x** (*Tensor*) –Input tensor with shape (n, c, h, w).

返回 Forward results.

返回类型 Tensor

init_weights(*model: torch.nn.Module*, *pretrained: str*) → None

Init weights.

参数

- **model** (*nn.Module*) –Models to be inited.
- **pretrained** (*str*) –Path for pretrained weights.

class mmagic.models.losses.**TransferalPerceptualLoss**(*loss_weight: float = 1.0*, *use_attention: bool = True*, *criterion: str = 'mse'*)

Bases: torch.nn.Module

Transferal perceptual loss.

参数

- **loss_weight** (*float*) –Loss weight. Default: 1.0.
- **use_attention** (*bool*) –If True, use soft-attention tensor. Default: True
- **criterion** (*str*) –Criterion type. Options are ‘l1’ and ‘mse’. Default: ‘mse’.

forward(*maps: Tuple[torch.Tensor]*, *soft_attention: torch.Tensor*, *textures: Tuple[torch.Tensor]*) → torch.Tensor

Forward function.

参数

- **maps** (*Tuple[Tensor]*) –Input tensors.
- **soft_attention** (*Tensor*) –Soft-attention tensor.
- **textures** (*Tuple[Tensor]*) –Ground-truth tensors.

返回 Forward results.

返回类型 Tensor

class mmagic.models.losses.**CharbonnierLoss**(*loss_weight: float = 1.0*, *reduction: str = 'mean'*, *sample_wise: bool = False*, *eps: float = 1e-12*)

Bases: torch.nn.Module

Charbonnier loss (one variant of Robust L1Loss, a differentiable variant of L1Loss).

Described in "Deep Laplacian Pyramid Networks for Fast and Accurate Super-Resolution".

参数

- **loss_weight** (*float*) –Loss weight for L1 loss. Default: 1.0.
- **reduction** (*str*) –Specifies the reduction to apply to the output. Supported choices are 'none' | 'mean' | 'sum'. Default: 'mean'.
- **sample_wise** (*bool*) –Whether calculate the loss sample-wise. This argument only takes effect when *reduction* is 'mean' and *weight* (argument of *forward()*) is not None. It will first reduces loss with 'mean' per-sample, and then it means over all the samples. Default: False.
- **eps** (*float*) –A value used to control the curvature near zero. Default: 1e-12.

forward(*pred: torch.Tensor*, *target: torch.Tensor*, *weight: Optional[torch.Tensor] = None*, ***kwargs*) → torch.Tensor

Forward Function.

参数

- **pred** (*Tensor*) –of shape (N, C, H, W). Predicted tensor.
- **target** (*Tensor*) –of shape (N, C, H, W). Ground truth tensor.
- **weight** (*Tensor, optional*) –of shape (N, C, H, W). Element-wise weights. Default: None.

class mmagic.models.losses.**L1Loss**(*loss_weight: float = 1.0*, *reduction: str = 'mean'*, *sample_wise: bool = False*)

Bases: torch.nn.Module

L1 (mean absolute error, MAE) loss.

参数

- **loss_weight** (*float*) –Loss weight for L1 loss. Default: 1.0.
- **reduction** (*str*) –Specifies the reduction to apply to the output. Supported choices are 'none' | 'mean' | 'sum'. Default: 'mean'.

- **sample_wise** (*bool*) –Whether calculate the loss sample-wise. This argument only takes effect when *reduction* is ‘mean’ and *weight* (argument of *forward()*) is not None. It will first reduce loss with ‘mean’ per-sample, and then it means over all the samples. Default: False.

forward(*pred: torch.Tensor*, *target: torch.Tensor*, *weight: Optional[torch.Tensor] = None*, ***kwargs*) → torch.Tensor

Forward Function.

参数

- **pred** (*Tensor*) –of shape (N, C, H, W). Predicted tensor.
- **target** (*Tensor*) –of shape (N, C, H, W). Ground truth tensor.
- **weight** (*Tensor, optional*) –of shape (N, C, H, W). Element-wise weights. Default: None.

class mmagic.models.losses.**MaskedTVLoss**(*loss_weight: float = 1.0*)

Bases: *L1Loss*

Masked TV loss.

参数 **loss_weight** (*float, optional*) –Loss weight. Defaults to 1.0.

forward(*pred: torch.Tensor*, *mask: Optional[torch.Tensor] = None*) → torch.Tensor

Forward function.

参数

- **pred** (*torch.Tensor*) –Tensor with shape of (n, c, h, w).
- **mask** (*torch.Tensor, optional*) –Tensor with shape of (n, 1, h, w). Defaults to None.

返回 [description]

返回类型 [type]

class mmagic.models.losses.**MSELoss**(*loss_weight: float = 1.0*, *reduction: str = 'mean'*, *sample_wise: bool = False*)

Bases: torch.nn.Module

MSE (L2) loss.

参数

- **loss_weight** (*float*) –Loss weight for MSE loss. Default: 1.0.
- **reduction** (*str*) –Specifies the reduction to apply to the output. Supported choices are ‘none’ | ‘mean’ | ‘sum’. Default: ‘mean’.

- **sample_wise** (*bool*) –Whether calculate the loss sample-wise. This argument only takes effect when *reduction* is ‘mean’ and *weight* (argument of *forward()*) is not None. It will first reduces loss with ‘mean’ per-sample, and then it means over all the samples. Default: False.

forward(*pred: torch.Tensor*, *target: torch.Tensor*, *weight: Optional[torch.Tensor] = None*, ***kwargs*) → torch.Tensor

Forward Function.

参数

- **pred** (*Tensor*) –of shape (N, C, H, W). Predicted tensor.
- **target** (*Tensor*) –of shape (N, C, H, W). Ground truth tensor.
- **weight** (*Tensor, optional*) –of shape (N, C, H, W). Element-wise weights. Default: None.

class mmagic.models.losses.**PSNRLoss**(*loss_weight: float = 1.0*, *toY: bool = False*)

Bases: torch.nn.Module

PSNR Loss in “HINet: Half Instance Normalization Network for Image Restoration”.

参数

- **loss_weight** (*float, optional*) –Loss weight. Defaults to 1.0.
- **reduction** –reduction for PSNR. Can only be mean here.
- **toY** –change to calculate the PSNR of Y channel in YCbCr format

forward(*pred: torch.Tensor*, *target: torch.Tensor*) → torch.Tensor

mmagic.models.losses.**tv_loss**(*input: torch.Tensor*) → torch.Tensor

L2 total variation loss, as in Mahendran et al.

CHAPTER 81

mmagic.models.data_preprocessors

81.1 Package Contents

81.1.1 Classes

DataPreprocessor	Image pre-processor for generative models. This class provide
MattorPreprocessor	DataPreprocessor for matting models.

class mmagic.models.data_preprocessors.**DataPreprocessor**(*mean: Union[Sequence[Union[float, int]], float, int] = 127.5*, *std: Union[Sequence[Union[float, int]], float, int] = 127.5*, *pad_size_divisor: int = 1*, *pad_value: Union[float, int] = 0*, *pad_mode: str = 'constant'*, *non_image_keys: Optional[Tuple[str, List[str]]] = None*, *non_concentate_keys: Optional[Tuple[str, List[str]]] = None*, *output_channel_order: Optional[str] = None*, *data_keys: Union[List[str], str] = 'gt_img'*, *input_view: Optional[tuple] = None*, *output_view: Optional[tuple] = None*, *stack_data_sample=True*)

Bases: mmengine.model.ImgDataPreprocessor

Image pre-processor for generative models. This class provide normalization and bgr to rgb conversion for image tensor inputs. The input of this classes should be dict which keys are *inputs* and *data_samples*.

Besides to process tensor *inputs*, this class support dict as *inputs*. - If the value is *Tensor* and the corresponding key is not contained in *_NON_IMAGE_KEYS*, it will be processed as image tensor. - If the value is *Tensor* and the corresponding key belongs to *_NON_IMAGE_KEYS*, it will not remains unchanged. - If value is string or integer, it will not remains unchanged.

参数

- **mean** (*Sequence[float or int], float or int, optional*) –The pixel mean of image channels. Noted that normalization operation is performed *after channel order conversion*. If it is not specified, images will not be normalized. Defaults None.
- **std** (*Sequence[float or int], float or int, optional*) –The pixel standard deviation of image channels. Noted that normalization operation is performed *after channel order conversion*. If it is not specified, images will not be normalized. Defaults None.
- **pad_size_divisor** (*int*) –The size of padded image should be divisible by pad_size_divisor. Defaults to 1.
- **pad_value** (*float or int*) –The padded pixel value. Defaults to 0.
- **pad_mode** (*str*) –Padding mode for torch.nn.functional.pad. Defaults to ‘constant’.

- **non_image_keys** (*List[str] or str*) –Keys for fields that not need to be processed (padding, channel conversion and normalization) as images. If not passed, the keys in *_NON_IMAGE_KEYS* will be used. This argument will only work when *inputs* is dict or list of dict. Defaults to None.
- **non_concatenate_keys** (*List[str] or str*) –Keys for fields that not need to be concatenated. If not passed, the keys in *_NON_CONCATENATE_KEYS* will be used. This argument will only work when *inputs* is dict or list of dict. Defaults to None.
- **output_channel_order** (*str, optional*) –The desired image channel order of output the data preprocessor. This is also the desired input channel order of model (and this most likely to be the output order of model). If not passed, no channel order conversion will be performed. Defaults to None.
- **data_keys** (*List[str] or str*) –Keys to preprocess in data samples. Defaults to 'gt_img' .
- **input_view** (*tuple, optional*) –The view of input tensor. This argument maybe deleted in the future. Defaults to None.
- **output_view** (*tuple, optional*) –The view of output tensor. This argument maybe deleted in the future. Defaults to None.
- **stack_data_sample** (*bool*) –Whether stack a list of data samples to one data sample. Only support with input data samples are *DataSamples*. Defaults to True.

_NON_IMAGE_KEYS = ['noise']

_NON_CONCATENATE_KEYS = ['num_batches', 'mode', 'sample_kwargs', 'eq_cfg']

cast_data(*data: CastData*) → CastData

Copying data to the target device.

参数 **data** (*dict*) –Data returned by `DataLoader`.

返回 Inputs and data sample at target device.

返回类型 CollatedResult

static _parse_channel_index(*inputs*) → int

Parse channel index of inputs.

_parse_channel_order(*key: str*, *inputs: torch.Tensor*, *data_sample: Optional[*mmagic.structures.DataSample*] = None*) → str

_parse_batch_channel_order(*key: str*, *inputs: Sequence*, *data_samples: Optional[Sequence[*mmagic.structures.DataSample*]]*) → str

Parse channel order of inputs in batch.

_update_metainfo(*padding_info: torch.Tensor*, *channel_order_info: Optional[dict] = None*, *data_samples: Optional[mmagic.utils.typing.SampleList] = None*) → mmagic.utils.typing.SampleList

Update *padding_info* and *channel_order* to metainfo of.

a batch of 'data_samples'. For channel order, we consider same field among data samples share the same channel order. Therefore *channel_order* is passed as a dict, which key and value are field name and corresponding channel order. For padding info, we consider padding info is same among all field of a sample, but can vary between samples. Therefore, we pass *padding_info* as Tensor shape like (B, 1, 1).

参数

- **padding_info** (*Tensor*) –The padding info of each sample. Shape like (B, 1, 1).
- **channel_order** (*dict, Optional*) –The channel order of target field. Key and value are field name and corresponding channel order respectively.
- **data_samples** (*List[DataSample], optional*) –The data samples to be updated. If not passed, will initialize a list of empty data samples. Defaults to None.

返回 The updated data samples.

返回类型 List[DataSample]

_do_conversion(*inputs: torch.Tensor*, *inputs_order: str = 'BGR'*, *target_order: Optional[str] = None*) → Tuple[torch.Tensor, str]

Conduct channel order conversion for *a batch of inputs*, and return the converted inputs and order after conversion.

inputs_order:

- RGB / RGB: Convert to target order.
- SINGLE: Do not change

_do_norm(*inputs: torch.Tensor*, *do_norm: Optional[bool] = None*) → torch.Tensor

_preprocess_image_tensor(*inputs: torch.Tensor*, *data_samples: Optional[mmagic.utils.typing.SampleList] = None*, *key: str = 'img'*) → Tuple[torch.Tensor, mmagic.utils.typing.SampleList]

Preprocess a batch of image tensor and update metainfo to corresponding data samples.

参数

- **inputs** (*Tensor*) –Image tensor with shape (C, H, W), (N, C, H, W) or (N, t, C, H, W) to preprocess.
- **data_samples** (*List[DataSample], optional*) –The data samples of corresponding inputs. If not passed, a list of empty data samples will be initialized to save metainfo. Defaults to None.
- **key** (*str*) –The key of image tensor in data samples. Defaults to 'img'.

返回

The preprocessed image tensor and updated data samples.

返回类型 Tuple[Tensor, List[DataSample]]

_preprocess_image_list(*tensor_list: List[torch.Tensor]*, *data_samples: Optional[mmagic.utils.typing.SampleList]*, *key: str = 'img'*) → Tuple[torch.Tensor, mmagic.utils.typing.SampleList]

Preprocess a list of image tensor and update metainfo to corresponding data samples.

参数

- **tensor_list** (*List[Tensor]*) –Image tensor list to be preprocess.
- **data_samples** (*List[DataSample], optional*) –The data samples of corresponding inputs. If not passed, a list of empty data samples will be initialized to save metainfo. Defaults to None.
- **key** (*str*) –The key of tensor list in data samples. Defaults to 'img'.

返回

The preprocessed image tensor and updated data samples.

返回类型 Tuple[Tensor, List[DataSample]]

_preprocess_dict_inputs(*batch_inputs: dict*, *data_samples: Optional[mmagic.utils.typing.SampleList] = None*) → Tuple[dict, mmagic.utils.typing.SampleList]

Preprocess dict type inputs.

参数

- **batch_inputs** (*dict*) –Input dict.
- **data_samples** (*List[DataSample], optional*) –The data samples of corresponding inputs. If not passed, a list of empty data samples will be initialized to save metainfo. Defaults to None.

返回

The preprocessed dict and updated data samples.

返回类型 Tuple[dict, List[DataSample]]

_preprocess_data_sample(*data_samples: mmagic.utils.typing.SampleList*, *training: bool*) → *mmagic.structures.DataSample*

Preprocess data samples. When *training* is True, fields belong to `self.data_keys` will be converted to `self.output_channel_order` and then normalized by *self.mean* and *self.std*. When *training* is False, fields belongs to `self.data_keys` will be attempted to convert to 'BGR' without normalization. The corresponding metainfo related to normalization, channel order conversion will be updated to data sample as well.

参数

- **data_samples** (*List[DataSample]*) –A list of data samples to preprocess.
- **training** (*bool*) –Whether in training mode.

返回 The list of processed data samples.

返回类型 list

forward(*data: dict*, *training: bool = False*) → dict

Performs normalization、padding and channel order conversion.

参数

- **data** (*dict*) –Input data to process.
- **training** (*bool*) –Whether to in training mode. Default: False.

返回 Data in the same format as the model input.

返回类型 dict

destruct(*outputs: torch.Tensor*, *data_samples: Union[mmagic.utils.typing.SampleList,*
mmagic.structures.DataSample, *None] = None*, *key: str = 'img'*) → Union[list, torch.Tensor]

Destruct padding, normalization and convert channel order to BGR if could. If *data_samples* is a list, outputs will be destructed as a batch of tensor. If *data_samples* is a *DataSample*, *outputs* will be destructed as a single tensor.

Before feed model outputs to visualizer and evaluator, users should call this function for model outputs and inputs.

Use cases:

```
>>> # destruct model outputs.
>>> # model outputs share the same preprocess information with inputs
>>> # ('img') therefore use 'img' as key
>>> feats = self.forward_tensor(inputs, data_samples, **kwargs)
>>> feats = self.data_preprocessor.destruct(feats, data_samples, 'img')
```

```
>>> # destruct model inputs for visualization
>>> for idx, data_sample in enumerate(data_samples):
>>>     destructed_input = self.data_preprocessor.destruct(
>>>         inputs[idx], data_sample, key='img')
>>>     data_sample.set_data({'input': destructed_input})
```

参数

- **outputs** (*Tensor*) –Tensor to destruct.

- **data_samples** (*Union[SampleList, DataSample], optional*) –Data samples (or data sample) corresponding to *outputs*. Defaults to None
- **key** (*str*) –The key of field in data sample. Defaults to 'img'.

返回 Destructed outputs.

返回类型 Union[list, Tensor]

_destruct_norm_and_conversion(*batch_tensor: torch.Tensor*, *data_samples: Union[mmagic.utils.typing.SampleList, mmagic.structures.DataSample, None]*, *key: str*) → torch.Tensor

De-norm and de-convert channel order. Noted that, we de-norm first, and then de-conversion, since mean and std used in normalization is based on channel order after conversion.

参数

- **batch_tensor** (*Tensor*) –Tensor to destruct.
- **data_samples** (*Union[SampleList, DataSample], optional*) –Data samples (or data sample) corresponding to *outputs*.
- **key** (*str*) –The key of field in data sample.

返回 Destructed tensor.

返回类型 Tensor

_destruct_padding(*batch_tensor: torch.Tensor*, *data_samples: Union[mmagic.utils.typing.SampleList, mmagic.structures.DataSample, None]*, *same_padding: bool = True*) → Union[list, torch.Tensor]

Destruct padding of the input tensor.

参数

- **batch_tensor** (*Tensor*) –Tensor to destruct.
- **data_samples** (*Union[SampleList, DataSample], optional*) –Data samples (or data sample) corresponding to *outputs*. If
- **same_padding** (*bool*) –Whether all samples will un-padded with the padding info of the first sample, and return a stacked un-padded tensor. Otherwise each sample will be unpadded with padding info saved in corresponding data samples, and return a list of un-padded tensor, since each un-padded tensor may have the different shape. Defaults to True.

返回 Destructed outputs.

返回类型 Union[list, Tensor]

class mmagic.models.data_preprocessors.**MattorPreprocessor**(*mean: MEAN_STD_TYPE = [123.675, 116.28, 103.53]*, *std: MEAN_STD_TYPE = [58.395, 57.12, 57.375]*, *output_channel_order: str = 'RGB'*, *proc_trimap: str = 'rescale_to_zero_one'*, *stack_data_sample=True*)

Bases: mmagic.models.data_preprocessors.data_preprocessor.DataPreprocessor

DataPreprocessor for matting models.

See base class DataPreprocessor for detailed information.

Workflow as follow :

- Collate and move data to the target device.
- Convert inputs from bgr to rgb if the shape of input is (3, H, W).
- Normalize image with defined std and mean.
- Stack inputs to batch_inputs.

参数

- **mean** (*Sequence[float or int], float or int, optional*) –The pixel mean of image channels. Noted that normalization operation is performed *after channel order conversion*. If it is not specified, images will not be normalized. Defaults None.
- **std** (*Sequence[float or int], float or int, optional*) –The pixel standard deviation of image channels. Noted that normalization operation is performed *after channel order conversion*. If it is not specified, images will not be normalized. Defaults None.
- **proc_trimap** (*str*) –Methods to process gt tensors. Default: 'rescale_to_zero_one'. Available options are rescale_to_zero_one and as-is.
- **stack_data_sample** (*bool*) –Whether stack a list of data samples to one data sample. Only support with input data samples are *DataSamples*. Defaults to True.

_proc_batch_trimap(*batch_trimaps: torch.Tensor*)

_preprocess_data_sample(*data_samples: mmagic.utils.typing.SampleList*, *training: bool*) → list

Preprocess data samples. When *training* is True, fields belong to self.data_keys will be converted to self.output_channel_order and *divided by 255*. When *training* is False, fields belongs to self.data_keys will be attempted to convert to 'BGR' without normalization. The corresponding metainfo related to normalization, channel order conversion will be updated to data sample as well.

参数

- **data_samples** (*List[DataSample]*) –A list of data samples to preprocess.
- **training** (*bool*) –Whether in training mode.

返回 The list of processed data samples.

返回类型 list

forward(*data: Sequence[dict]*, *training: bool = False*) → Tuple[torch.Tensor, list]

Pre-process input images, trimaps, ground-truth as configured.

参数

- **data** (*Sequence[dict]*) –data sampled from dataloader.
- **training** (*bool*) –Whether to enable training time augmentation. Default: False.

返回 Batched inputs and list of data samples.

返回类型 Tuple[torch.Tensor, list]

CHAPTER 82

mmagic.models.editors

82.1 Package Contents

82.1.1 Classes

AOTBlockNeck	Dilation backbone used in AOT-GAN model.
AOTEncoderDecoder	Encoder-Decoder used in AOT-GAN model.
AOTInpaintor	Inpaintor for AOT-GAN method.
IDLossModel	Face id loss model.
BasicVSR	BasicVSR model for video super-resolution.
BasicVSRNet	BasicVSR network structure for video super-resolution.
BasicVSRPlusPlusNet	BasicVSR++ network structure.
BigGAN	Implementation of ‘Large Scale GAN Training for High Fidelity Natural
CAIN	CAIN model for Video Interpolation.
CAINNet	CAIN network structure.
ControlStableDiffusion	Implementation of ‘ControlNet with Stable Diffusion.
CycleGAN	CycleGAN model for unpaired image-to-image translation.
DCGAN	Implementation of ‘Unsupervised Representation Learning with Deep

下页继续

表 1 – 续上页

DenoisingUnet	Denoising Unet. This network receives a diffused image x_t and
DeblurGanV2	Base class for all algorithmic models.
DeblurGanV2Discriminator	Defines the discriminator for DeblurGanv2 with the specified arguments..
DeblurGanV2Generator	Defines the generator for DeblurGanv2 with the specified arguments..
ContextualAttentionModule	Contexture attention module.
ContextualAttentionNeck	Neck with contextual attention module.
DeepFillDecoder	Decoder used in DeepFill model.
DeepFillEncoder	Encoder used in DeepFill model.
DeepFillRefiner	Refiner used in DeepFill model.
DeepFillv1Discriminators	Discriminators used in DeepFillv1 model.
DeepFillv1Inpaintor	Inpaintor for deepfillv1 method.
DeepFillEncoderDecoder	Two-stage encoder-decoder structure used in DeepFill model.
DIC	DIC model for Face Super-Resolution.
DICNet	DIC network structure for face super-resolution.
FeedbackBlock	Feedback Block of DIC.
FeedbackBlockCustom	Custom feedback block, will be used as the first feedback block.
FeedbackBlockHeatmapAttention	Feedback block with HeatmapAttention.
LightCNN	LightCNN discriminator with input size 128 x 128.
MaxFeature	Conv2d or Linear layer with max feature selector.
DIM	Deep Image Matting model.
ClipWrapper	Clip Models wrapper.
DiscoDiffusion	Disco Diffusion (DD) is a Google Colab Notebook which leverages an AI
DreamBooth	Implementation of 'DreamBooth with Stable Diffusion.
EDSRNet	EDSR network structure.
EDVR	EDVR model for video super-resolution.
EDVRNet	EDVR network structure for video super-resolution.
EG3D	Implementation of 'Efficient Geometry-aware 3D Generative Adversarial
ESRGAN	Enhanced SRGAN model for single image super-resolution.
RRDBNet	Networks consisting of Residual in Residual Dense Block, which is used
FBADecoder	Decoder for FBA matting.

下页继续

表 1 – 续上页

FBAResnetDilated	ResNet-based encoder for FBA image matting.
FLAVR	FLAVR model for video interpolation.
FLAVRNet	PyTorch implementation of FLAVR for video frame interpolation.
GCA	Guided Contextual Attention image matting model.
GGAN	Implementation of *Geometric GAN*.
GLEANStyleGANv2	GLEAN (using StyleGANv2) architecture for super-resolution.
GLDecoder	Decoder used in Global&Local model.
GLDilationNeck	Dilation Backbone used in Global&Local model.
GLEncoder	Encoder used in Global&Local model.
GLEncoderDecoder	Encoder-Decoder used in Global&Local model.
AblatedDiffusionModel	Guided diffusion Model.
IconVSRNet	IconVSR network structure for video super-resolution.
DepthwiseIndexBlock	Depthwise index block.
HolisticIndexBlock	Holistic Index Block.
IndexedUpsample	Indexed upsample module.
IndexNet	IndexNet matting model.
IndexNetDecoder	Decoder for IndexNet.
IndexNetEncoder	Encoder for IndexNet.
InstColorization	Colorization InstColorization method.
LIIF	LIIF model for single image super-resolution.
MLPRefiner	Multilayer perceptrons (MLPs), refiner used in LIIF.
LSGAN	Implementation of *Least Squares Generative Adversarial Networks*.
MSPIEStyleGAN2	MS-PIE StyleGAN2.
PESinGAN	Positional Encoding in SinGAN.
NAFBaseline	The original version of Baseline model in ”Simple Baseline for Image
NAFBaselineLocal	The original version of Baseline model in ”Simple Baseline for Image
NAFNet	NAFNet.
NAFNetLocal	The original version of NAFNetLocal in ”Simple Baseline for Image
MaskConvModule	Mask convolution module.
PartialConv2d	Implementation for partial convolution.
PConvDecoder	Decoder with partial conv.
PConvEncoder	Encoder with partial conv.
PConvEncoderDecoder	Encoder-Decoder with partial conv module.

下页继续

表 1 – 续上页

PConvInpaintor	Inpaintor for Partial Convolution method.
ProgressiveGrowingGAN	Progressive Growing Unconditional GAN.
Pix2Pix	Pix2Pix model for paired image-to-image translation.
PlainDecoder	Simple decoder from Deep Image Matting.
PlainRefiner	Simple refiner from Deep Image Matting.
RDNNet	RDN model for single image super-resolution.
RealBasicVSR	RealBasicVSR model for real-world video super-resolution.
RealBasicVSRNet	RealBasicVSR network structure for real-world video super-resolution.
RealESRGAN	Real-ESRGAN model for single image super-resolution.
UNetDiscriminatorWithSpectralNorm	A U-Net discriminator with spectral normalization.
Restormer	Restormer A PyTorch impl of: ‘Restormer: Efficient Transformer for High-
SAGAN	Implementation of *Self-Attention Generative Adversarial Networks*.
SinGAN	SinGAN.
SRCNNNet	SRCNN network structure for image super resolution.
SRGAN	SRGAN model for single image super-resolution.
ModifiedVGG	A modified VGG discriminator with input size 128 x 128.
MSRResNet	Modified SRResNet.
StableDiffusion	Class for Stable Diffusion. Refers to https://github.com/Stability-
StableDiffusionInpaint	Class for Stable Diffusion. Refers to https://github.com/Stability-
StyleGAN1	Implementation of ‘A Style-Based Generator Architecture for Generative
StyleGAN2	Implementation of ‘Analyzing and Improving the Image Quality of
StyleGAN3	Implementation of *Alias-Free Generative Adversarial Networks*. # noqa.
StyleGAN3Generator	StyleGAN3 Generator.
SwinIRNet	SwinIR
TDAN	TDAN model for video super-resolution.
TDANNet	TDAN network structure for video super-resolution.
TextualInversion	Implementation of ‘An Image is Worth One Word: Personalizing Text-to-
TOFlowVFINet	PyTorch implementation of TOFlow for video frame interpolation.

下页继续

表 1 – 续上页

TOFlowVSRNet	PyTorch implementation of TOFlow.
ToFResBlock	ResNet architecture.
LTE	Learnable Texture Extractor.
TTSR	TTSR model for Reference-based Image Super-Resolution.
SearchTransformer	Search texture reference by transformer.
TTSRDiscriminator	A discriminator for TTSR.
TTSRNet	TTSR network structure (main-net) for reference-based super-resolution.
WGANGP	Implementation of *Improved Training of Wasserstein GANs*.

class mmagic.models.editors.**AOTBlockNeck**(*in_channels=256*, *dilation_rates=(1, 2, 4, 8)*, *num_aotblock=8*, *act_cfg=dict(type='ReLU')*, ***kwargs*)

Bases: mmengine.model.BaseModule

Dilation backbone used in AOT-GAN model.

This implementation follows: Aggregated Contextual Transformations for High-Resolution Image Inpainting

参数

- **in_channels** (*int, optional*) –Channel number of input feature. Default: 256.
- **dilation_rates** (*Tuple[int], optional*) –The dilation rates used
- **Default** (*for AOT block.*) –(1, 2, 4, 8).
- **num_aotblock** (*int, optional*) –Number of AOT blocks. Default: 8.
- **act_cfg** (*dict, optional*) –Config dict for activation layer, "relu" by default.
- **kwargs** (*keyword arguments*) –

forward(*x*)

class mmagic.models.editors.**AOTEncoderDecoder**(*encoder=dict(type='AOTEncoder')*, *decoder=dict(type='AOTDecoder')*, *dilation_neck=dict(type='AOTBlockNeck')*)

Bases: mmagic.models.editors.global_local.GLEncoderDecoder

Encoder-Decoder used in AOT-GAN model.

This implementation follows: Aggregated Contextual Transformations for High-Resolution Image Inpainting The architecture of the encoder-decoder is: (conv2d x 3) –> (dilated conv2d x 8) –> (conv2d or deconv2d x 3).

参数

- **encoder** (*dict*) –Config dict to encoder.

- **decoder** (*dict*) –Config dict to build decoder.
- **dilation_neck** (*dict*) –Config dict to build dilation neck.

class mmagic.models.editors.**AOTInpaintor**(*data_preprocessor: Union[dict, mmengine.config.Config]*, *encdec: dict*, *disc: Optional[dict] = None*, *loss_gan: Optional[dict] = None*, *loss_gp: Optional[dict] = None*, *loss_disc_shift: Optional[dict] = None*, *loss_composed_percep: Optional[dict] = None*, *loss_out_percep: bool = False*, *loss_l1_hole: Optional[dict] = None*, *loss_l1_valid: Optional[dict] = None*, *loss_tv: Optional[dict] = None*, *train_cfg: Optional[dict] = None*, *test_cfg: Optional[dict] = None*, *init_cfg: Optional[dict] = None*)

Bases: *mmagic.models.base_models.OneStageInpaintor*

Inpaintor for AOT-GAN method.

This inpaintor is implemented according to the paper: Aggregated Contextual Transformations for High-Resolution Image Inpainting

forward_train_d(*data_batch*, *is_real*, *is_disc*, *mask*)

Forward function in discriminator training step.

In this function, we compute the prediction for each data batch (real or fake). Meanwhile, the standard gan loss will be computed with several proposed losses for stable training.

参数

- **data_batch** (*torch.Tensor*) –Batch of real data or fake data.
- **is_real** (*bool*) –If True, the gan loss will regard this batch as real data. Otherwise, the gan loss will regard this batch as fake data.
- **is_disc** (*bool*) –If True, this function is called in discriminator training step. Otherwise, this function is called in generator training step. This will help us to compute different types of adversarial loss, like LSGAN.
- **mask** (*torch.Tensor*) –Mask of data.

返回 Contains the loss items computed in this function.

返回类型 dict

generator_loss(*fake_res*, *fake_img*, *gt*, *mask*, *masked_img*)

Forward function in generator training step.

In this function, we mainly compute the loss items for generator with the given (fake_res, fake_img). In general, the *fake_res* is the direct output of the generator and the *fake_img* is the composition of direct output and ground-truth image.

参数

- **fake_res** (*torch.Tensor*) –Direct output of the generator.
- **fake_img** (*torch.Tensor*) –Composition of *fake_res* and ground-truth image.
- **gt** (*torch.Tensor*) –Ground-truth image.
- **mask** (*torch.Tensor*) –Mask image.
- **masked_img** (*torch.Tensor*) –Composition of mask image and ground-truth image.

返回

Dict contains the results computed within this function for visualization and dict contains the loss items computed in this function.

返回类型 tuple(dict)

forward_tensor(*inputs*, *data_samples*)

Forward function in tensor mode.

参数

- **inputs** (*torch.Tensor*) –Input tensor.
- **data_samples** (*List[dict]*) –List of data sample dict.

返回

Direct output of the generator and composition of *fake_res* and ground-truth image.

返回类型 tuple

train_step(*data: List[dict]*, *optim_wrapper*)

Train step function.

In this function, the inpaintor will finish the train step following the pipeline: 1. get fake res/image 2. compute reconstruction losses for generator 3. compute adversarial loss for discriminator 4. optimize generator 5. optimize discriminator

参数

- **data** (*List[dict]*) –Batch of data as input.
- **optim_wrapper** (*dict[torch.optim.Optimizer]*) –Dict with optimizers for generator and discriminator (if have).

返回

Dict with loss, information for logger, the number of samples and results for visualization.

返回类型 dict

class mmagic.models.editors.**IDLossModel**(*ir_se50_weights=None*)

Bases: torch.nn.Module

Face id loss model.

参数 ir_se50_weights (*str, optional*) –Url of ir-se50 weights. Defaults to None.

_ir_se50_url = 'https://gg0ltg.by.files.1drv.com/
y4m3fNNszG03z9n8JQ7EhdtQKW8tQVQMFBisPVRgoXi_UfP8pKSSqv8RJNmHy2Ja...'

extract_feats(*x*)

Extracting face features.

参数 x (*torch.Tensor*) –Image tensor of faces.

返回 Face features.

返回类型 torch.Tensor

forward(*pred=None*, *gt=None*)

Calculate face loss.

参数

- **pred** (*torch.Tensor, optional*) –Predictions of face images. Defaults to None.
- **gt** (*torch.Tensor, optional*) –Ground truth of face images. Defaults to None.

返回

A tuple contain face similarity loss and improvement.

返回类型 Tuple(float, float)

class mmagic.models.editors.**BasicVSR**(*generator*, *pixel_loss*, *ensemble=None*, *train_cfg=None*, *test_cfg=None*, *init_cfg=None*, *data_preprocessor=None*)

Bases: mmagic.models.BaseEditModel

BasicVSR model for video super-resolution.

Note that this model is used for IconVSR.

Paper: BasicVSR: The Search for Essential Components in Video Super-Resolution and Beyond, CVPR, 2021

参数

- **generator** (*dict*) –Config for the generator structure.
- **pixel_loss** (*dict*) –Config for pixel-wise loss.
- **ensemble** (*dict*) –Config for ensemble. Default: None.
- **train_cfg** (*dict*) –Config for training. Default: None.
- **test_cfg** (*dict*) –Config for testing. Default: None.

- **init_cfg** (*dict, optional*) –The weight initialized config for BaseModule.
- **data_preprocessor** (*dict, optional*) –The pre-process config of BaseDataPreprocessor.

check_if_mirror_extended(*lrs*)

Check whether the input is a mirror-extended sequence.

If mirror-extended, the i-th (i=0, ⋯, t-1) frame is equal to the (t-1-i)-th frame.

参数 **lrs** (*tensor*) –Input LR images with shape (n, t, c, h, w)

forward_train(*inputs*, *data_samples=None*, ***kwargs*)

Forward training. Returns dict of losses of training.

参数

- **inputs** (*torch.Tensor*) –batch input tensor collated by data_preprocessor.
- **data_samples** (*List[BaseDataElement], optional*) –data samples collated by data_preprocessor.

返回 Dict of losses.

返回类型 dict

forward_inference(*inputs*, *data_samples=None*, ***kwargs*)

Forward inference. Returns predictions of validation, testing.

参数

- **inputs** (*torch.Tensor*) –batch input tensor collated by data_preprocessor.
- **data_samples** (*List[BaseDataElement], optional*) –data samples collated by data_preprocessor.

返回 predictions.

返回类型 DataSample

class mmagic.models.editors.**BasicVSRNet**(*mid_channels=64*, *num_blocks=30*, *spynet_pretrained=None*)

Bases: mmengine.model.BaseModule

BasicVSR network structure for video super-resolution.

Support only x4 upsampling.

Paper: BasicVSR: The Search for Essential Components in Video Super-Resolution and Beyond, CVPR, 2021

参数

- **mid_channels** (*int*) –Channel number of the intermediate features. Default: 64.

- **num_blocks** (*int*) –Number of residual blocks in each propagation branch. Default: 30.
- **spynet_pretrained** (*str*) –Pre-trained model path of SPyNet. Default: None.

check_if_mirror_extended(*lrs*)

Check whether the input is a mirror-extended sequence.

If mirror-extended, the i-th (i=0, ⋯, t-1) frame is equal to the (t-1-i)-th frame.

参数 **lrs** (*tensor*) –Input LR images with shape (n, t, c, h, w)

compute_flow(*lrs*)

Compute optical flow using SPyNet for feature warping.

Note that if the input is an mirror-extended sequence, 'flows_forward' is not needed, since it is equal to 'flows_backward.flip(1)'.

参数 **lrs** (*tensor*) –Input LR images with shape (n, t, c, h, w)

返回

Optical flow. 'flows_forward' corresponds to the flows used for forward-time propagation (current to previous). 'flows_backward' corresponds to the flows used for backward-time propagation (current to next).

返回类型 tuple(Tensor)

forward(*lrs*)

Forward function for BasicVSR.

参数 **lrs** (*Tensor*) –Input LR sequence with shape (n, t, c, h, w).

返回 Output HR sequence with shape (n, t, c, 4h, 4w).

返回类型 Tensor

class mmagic.models.editors.**BasicVSRPlusPlusNet**(*mid_channels=64*, *num_blocks=7*, *max_residue_magnitude=10*, *is_low_res_input=True*, *spynet_pretrained=None*, *cpu_cache_length=100*)

Bases: mmengine.model.BaseModule

BasicVSR++ network structure.

Support either x4 upsampling or same size output.

Paper: BasicVSR++: Improving Video Super-Resolution with Enhanced Propagation and Alignment

参数

- **mid_channels** (*int, optional*) –Channel number of the intermediate features. Default: 64.
- **num_blocks** (*int, optional*) –The number of residual blocks in each propagation branch. Default: 7.
- **max_residue_magnitude** (*int*) –The maximum magnitude of the offset residue (Eq. 6 in paper). Default: 10.
- **is_low_res_input** (*bool, optional*) –Whether the input is low-resolution or not. If False, the output resolution is equal to the input resolution. Default: True.
- **spynet_pretrained** (*str, optional*) –Pre-trained model path of SPyNet. Default: None.
- **cpu_cache_length** (*int, optional*) –When the length of sequence is larger than this value, the intermediate features are sent to CPU. This saves GPU memory, but slows down the inference speed. You can increase this number if you have a GPU with large memory. Default: 100.

check_if_mirror_extended(*lqs*)

Check whether the input is a mirror-extended sequence.

If mirror-extended, the i-th (i=0, ⋯, t-1) frame is equal to the (t-1-i)-th frame.

参数 **lqs** (*tensor*) –Input low quality (LQ) sequence with shape (n, t, c, h, w).

compute_flow(*lqs*)

Compute optical flow using SPyNet for feature alignment.

Note that if the input is an mirror-extended sequence, ‘flows_forward’ is not needed, since it is equal to ‘flows_backward.flip(1)’.

参数 **lqs** (*tensor*) –Input low quality (LQ) sequence with shape (n, t, c, h, w).

返回

Optical flow. ‘flows_forward’ corresponds to the flows used for forward-time propagation (current to previous). ‘flows_backward’ corresponds to the flows used for backward-time propagation (current to next).

返回类型 tuple(Tensor)

propagate(*feats*, *flows*, *module_name*)

Propagate the latent features throughout the sequence.

参数

- **dict** (*feats*) –Features from previous branches. Each component is a list of tensors with shape (n, c, h, w).
- **flows** (*tensor*) –Optical flows with shape (n, t - 1, 2, h, w).

- **module_name** (*str*) –The name of the propagation branches. Can either be 'backward_1', 'forward_1', 'backward_2', 'forward_2'.

返回

A dictionary containing all the propagated features. Each key in the dictionary corresponds to a propagation branch, which is represented by a list of tensors.

返回类型 dict(list[tensor])

upsample(*lqs*, *feats*)

Compute the output image given the features.

参数

- **lqs** (*tensor*) –Input low quality (LQ) sequence with shape (n, t, c, h, w).
- **feats** (*dict*) –The features from the propagation branches.

返回 Output HR sequence with shape (n, t, c, 4h, 4w).

返回类型 Tensor

forward(*lqs*)

Forward function for BasicVSR++.

参数 **lqs** (*tensor*) –Input low quality (LQ) sequence with shape (n, t, c, h, w).

返回 Output HR sequence with shape (n, t, c, 4h, 4w).

返回类型 Tensor

class mmagic.models.editors.**BigGAN**(*generator: ModelType*, *discriminator: Optional[ModelType] = None*, *data_preprocessor: Optional[Union[dict, mmengine.Config]] = None*, *generator_steps: int = 1*, *discriminator_steps: int = 1*, *noise_size: Optional[int] = None*, *num_classes: Optional[int] = None*, *ema_config: Optional[Dict] = None*)

Bases: *mmagic.models.base_models.BaseConditionalGAN*

Implementation of Large Scale GAN Training for High Fidelity Natural Image Synthesis (BigGAN).

Detailed architecture can be found in `BigGANGenerator` and `BigGANDiscriminator`

参数

- **generator** (*ModelType*) –The config or model of the generator.
- **discriminator** (*Optional[ModelType]*) –The config or model of the discriminator. Defaults to None.
- **data_preprocessor** (*Optional[Union[dict, Config]]*) –The pre-process config or `DataPreprocessor`.

- **generator_steps** (*int*) –Number of times the generator was completely updated before the discriminator is updated. Defaults to 1.
- **discriminator_steps** (*int*) –Number of times the discriminator was completely updated before the generator is updated. Defaults to 1.
- **noise_size** (*Optional[int]*) –Size of the input noise vector. Default to 128.
- **num_classes** (*Optional[int]*) –The number classes you would like to generate. Defaults to None.
- **ema_config** (*Optional[Dict]*) –The config for generator's exponential moving average setting. Defaults to None.

disc_loss(*disc_pred_fake: torch.Tensor*, *disc_pred_real: torch.Tensor*) → Tuple

Get disc loss. BigGAN use hinge loss to train the discriminator.

参数

- **disc_pred_fake** (*Tensor*) –Discriminator's prediction of the fake images.
- **disc_pred_real** (*Tensor*) –Discriminator's prediction of the real images.

返回 Loss value and a dict of log variables.

返回类型 tuple[Tensor, dict]

gen_loss(*disc_pred_fake*)

Get disc loss. BigGAN use hinge loss to train the generator.

参数 **disc_pred_fake** (*Tensor*) –Discriminator's prediction of the fake images.

返回 Loss value and a dict of log variables.

返回类型 tuple[Tensor, dict]

train_discriminator(*inputs: dict*, *data_samples:* mmagic.structures.DataSample, *optimizer_wrapper: mmengine.optim.OptimWrapper*) → Dict[str, torch.Tensor]

Train discriminator.

参数

- **inputs** (*dict*) –Inputs from dataloader.
- **data_samples** (*DataSample*) –Data samples from dataloader.
- **optim_wrapper** (*OptimWrapper*) –OptimWrapper instance used to update model parameters.

返回 A `dict` of tensor for logging.

返回类型 Dict[str, Tensor]

train_generator(*inputs: dict*, *data_samples:* mmagic.structures.DataSample, *optimizer_wrapper: mmengine.optim.OptimWrapper*) → Dict[str, torch.Tensor]

Train generator.

参数

- **inputs** (*dict*) –Inputs from dataloader.
- **data_samples** (*DataSample*) –Data samples from dataloader. Do not used in generator's training.
- **optim_wrapper** (*OptimWrapper*) –OptimWrapper instance used to update model parameters.

返回 A `dict` of tensor for logging.

返回类型 Dict[str, Tensor]

class mmagic.models.editors.**CAIN**(*generator: dict*, *pixel_loss: dict*, *train_cfg: Optional[dict] = None*, *test_cfg: Optional[dict] = None*, *required_frames: int = 2*, *step_frames: int = 1*, *init_cfg: Optional[dict] = None*, *data_preprocessor: Optional[dict] = None*)

Bases: *mmagic.models.base_models.BasicInterpolator*

CAIN model for Video Interpolation.

Paper: Channel Attention Is All You Need for Video Frame Interpolation Ref repo: https://github.com/myungsub/CAIN

参数

- **generator** (*dict*) –Config for the generator structure.
- **pixel_loss** (*dict*) –Config for pixel-wise loss.
- **train_cfg** (*dict*) –Config for training. Default: None.
- **test_cfg** (*dict*) –Config for testing. Default: None.
- **required_frames** (*int*) –Required frames in each process. Default: 2
- **step_frames** (*int*) –Step size of video frame interpolation. Default: 1
- **init_cfg** (*dict, optional*) –The weight initialized config for `BaseModule`.
- **data_preprocessor** (*dict, optional*) –The pre-process config of `BaseDataPreprocessor`.

init_cfg

Initialization config dict.

Type dict, optional

data_preprocessor

Used for pre-processing data sampled by dataloader to the format accepted by `forward()`.

Type `BaseDataPreprocessor`

forward_inference(*inputs*, *data_samples=None*)

Forward inference. Returns predictions of validation, testing, and simple inference.

参数

- **inputs** (*torch.Tensor*) –batch input tensor collated by *data_preprocessor*.
- **data_samples** (*List[BaseDataElement], optional*) –data samples collated by *data_preprocessor*.

返回 predictions.

返回类型 List[DataSample]

class mmagic.models.editors.**CAINNet**(*in_channels=3*, *kernel_size=3*, *num_block_groups=5*, *num_block_layers=12*, *depth=3*, *reduction=16*, *norm=None*, *padding=7*, *act=nn.LeakyReLU(0.2, True)*, *init_cfg=None*)

Bases: `mmengine.model.BaseModule`

CAIN network structure.

Paper: Channel Attention Is All You Need for Video Frame Interpolation. Ref repo: https://github.com/myungsub/CAIN

参数

- **in_channels** (*int*) –Channel number of inputs. Default: 3.
- **kernel_size** (*int*) –Kernel size of CAINNet. Default: 3.
- **num_block_groups** (*int*) –Number of block groups. Default: 5.
- **num_block_layers** (*int*) –Number of blocks in a group. Default: 12.
- **depth** (*int*) –Down scale depth, scale = 2**depth. Default: 3.
- **reduction** (*int*) –Channel reduction of CA. Default: 16.
- **norm** (*str | None*) –Normalization layer. If it is None, no normalization is performed. Default: None.
- **padding** (*int*) –Padding of CAINNet. Default: 7.
- **act** (*function*) –activate function. Default: nn.LeakyReLU(0.2, True).
- **init_cfg** (*dict, optional*) –Initialization config dict. Default: None.

forward(*imgs*, *padding_flag=False*)

Forward function.

参数

- **imgs** (*Tensor*) –Input tensor with shape (n, 2, c, h, w).
- **padding_flag** (*bool*) –Padding or not. Default: False.

返回 Forward results.

返回类型 Tensor

class mmagic.models.editors.**ControlStableDiffusion**(*vae: ModelType*, *text_encoder: ModelType*, *tokenizer: str*, *unet: ModelType*, *controlnet: ModelType*, *scheduler: ModelType*, *test_scheduler: Optional[ModelType] = None*, *dtype: str = 'fp32'*, *enable_xformers: bool = True*, *noise_offset_weight: float = 0*, *tomesd_cfg: Optional[dict] = None*, *data_preprocessor=dict(type='DataPreprocessor')*, *init_cfg: Optional[dict] = None*, *attention_injection=False*)

Bases: mmagic.models.editors.stable_diffusion.StableDiffusion

Implementation of ‘ControlNet with Stable Diffusion.

<https://arxiv.org/abs/2302.05543>‘_ (ControlNet).

参数

- **vae** (*Union[dict, nn.Module]*) –The config or module for VAE model.
- **text_encoder** (*Union[dict, nn.Module]*) –The config or module for text encoder.
- **tokenizer** (*str*) –The **name** for CLIP tokenizer.
- **unet** (*Union[dict, nn.Module]*) –The config or module for Unet model.
- **controlnet** (*Union[dict, nn.Module]*) –The config or module for ControlNet.
- **schedule** (*Union[dict, nn.Module]*) –The config or module for diffusion scheduler.
- **test_scheduler** (*Union[dict, nn.Module], optional*) –The config or module for diffusion scheduler in test stage (*self.infer*). If not passed, will use the same scheduler as *schedule*. Defaults to None.
- **dtype** (*str, optional*) –The dtype for the model. Defaults to ‘fp16’.
- **enable_xformers** (*bool, optional*) –Whether to use xformers. Defaults to True.
- **noise_offset_weight** (*bool, optional*) –The weight of noise offset introduced in https://www.crosslabs.org/blog/diffusion-with-offset-noise # noqa Defaults to 0.

- **data_preprocessor** (*dict, optional*) –The pre-process config of `BaseDataPreprocessor`. Defaults to

  dict(type=’DataPreprocessor’).

- **init_cfg** (*dict, optional*) –The weight initialized config for `BaseModule`. Defaults to None/

init_weights()

Initialize the weights. Noted that this function will only be called at train. If you want to inference with a different unet model, you can call this function manually or use *mmagic.models.editors.controlnet.controlnet_utils.change_base_model* to convert the weight of ControlNet manually.

Example: »> 1. init controlnet from unet »> init_cfg = dict(type=’init_from_unet’)

```
>>> 2. switch controlnet weight from unet
>>> # base model is not defined, use `runwayml/stable-diffusion-v1-5`
>>> # as default
>>> init_cfg = dict(type='convert_from_unet')
>>> # base model is defined
>>> init_cfg = dict(
>>>     type='convert_from_unet',
>>>     base_model=dict(
>>>         type='UNet2DConditionModel',
>>>         from_pretrained='REPO_ID',
>>>         subfolder='unet'))
```

train_step(*data: dict*, *optim_wrapper: mmengine.optim.OptimWrapperDict*) → Dict[str, torch.Tensor]

Train step for ControlNet model. :param data: Data sampled from dataloader. :type data: dict :param optim_wrapper: OptimWrapperDict instance

contains OptimWrapper of generator and discriminator.

返回 A `dict` of tensor for logging.

返回类型 Dict[str, torch.Tensor]

val_step(*data: dict*) → mmagic.utils.typing.SampleList

Gets the generated image of given data. Calls `self.data_preprocessor` and `self.infer` in order. Return the generated results which will be passed to evaluator or visualizer.

参数 **data** (*dict or tuple or list*) –Data sampled from dataset.

返回 Generated image or image dict.

返回类型 SampleList

test_step(*data: dict*) → mmagic.utils.typing.SampleList

Gets the generated image of given data. Calls `self.data_preprocessor` and `self.infer` in order. Return the generated results which will be passed to evaluator or visualizer.

参数 **data**(*dict or tuple or list*) –Data sampled from dataset.

返回 Generated image or image dict.

返回类型 SampleList

static prepare_control(*image: Tuple[PIL.Image.Image, List[PIL.Image.Image], torch.Tensor, List[torch.Tensor]]*, *width: int*, *height: int*, *batch_size: int*, *num_images_per_prompt: int*, *device: str*, *dtype: str*) → torch.Tensor

A helper function to prepare single control images.

参数

- **image** (*Tuple[Image.Image, List[Image.Image], Tensor, List[Tensor]]*) –# noqa The input image for control.
- **batch_size** (*int*) –The number of the prompt. The control will be repeated for *batch_size* times.
- **num_images_per_prompt** (*int*) –The number images generate for one prompt.
- **device** (*str*) –The device of the control.
- **dtype** (*str*) –The dtype of the control.

返回 The control in torch.tensor.

返回类型 Tensor

train(*mode: bool = True*)

Set train/eval mode.

参数 **mode**(*bool, optional*) –Whether set train mode. Defaults to True.

infer(*prompt: Union[str, List[str]]*, *height: Optional[int] = None*, *width: Optional[int] = None*, *control: Optional[Union[str, numpy.ndarray, torch.Tensor]] = None*, *controlnet_conditioning_scale: float = 1.0*, *num_inference_steps: int = 20*, *guidance_scale: float = 7.5*, *negative_prompt: Optional[Union[str, List[str]]] = None*, *num_images_per_prompt: Optional[int] = 1*, *eta: float = 0.0*, *generator: Optional[torch.Generator] = None*, *latents: Optional[torch.FloatTensor] = None*, *return_type='image'*, *show_progress=True*)

Function invoked when calling the pipeline for generation.

参数

- **prompt** (*str or List[str]*) –The prompt or prompts to guide the image generation.
- **height** (*int, Optional*) –The height in pixels of the generated image. If not passed, the height will be *self.unet_sample_size * self.vae_scale_factor* Defaults to None.

- **width** (*int, Optional*) –The width in pixels of the generated image. If not passed, the width will be *self.unet_sample_size * self.vae_scale_factor* Defaults to None.
- **num_inference_steps** (*int*) –The number of denoising steps. More denoising steps usually lead to a higher quality image at the expense of slower inference. Defaults to 50.
- **guidance_scale** (*float*) –Guidance scale as defined in Classifier- Free Diffusion Guidance (https://arxiv.org/abs/2207.12598). Defaults to 7.5
- **negative_prompt** (*str or List[str], optional*) –The prompt or prompts not to guide the image generation. Ignored when not using guidance (i.e., ignored if *guidance_scale* is less than 1). Defaults to None.
- **num_images_per_prompt** (*int*) –The number of images to generate per prompt. Defaults to 1.
- **eta** (*float*) –Corresponds to parameter eta (η) in the DDIM paper: https://arxiv.org/abs/2010.02502. Only applies to DDIMScheduler, will be ignored for others. Defaults to 0.0.
- **generator** (*torch.Generator, optional*) –A torch generator to make generation deterministic. Defaults to None.
- **latents** (*torch.FloatTensor, optional*) –Pre-generated noisy latents, sampled from a Gaussian distribution, to be used as inputs for image generation. Can be used to tweak the same generation with different prompts. If not provided, a latents tensor will be generated by sampling using the supplied random *generator*. Defaults to None.
- **return_type** (*str*) –The return type of the inference results. Supported types are 'image', 'numpy', 'tensor'. If 'image' is passed, a list of PIL images will be returned. If 'numpy' is passed, a numpy array with shape [N, C, H, W] will be returned, and the value range will be same as decoder's output range. If 'tensor' is passed, the decoder's output will be returned. Defaults to 'image'.

返回 A dict containing the generated images and Control image.

返回类型 dict

abstract forward(**args*, ***kwargs*)

forward is not implemented now.

class mmagic.models.editors.**CycleGAN**(**args*, *buffer_size=50*, *loss_config=dict(cycle_loss_weight=10.0, id_loss_weight=0.5)*, ***kwargs*)

Bases: *mmagic.models.base_models.BaseTranslationModel*

CycleGAN model for unpaired image-to-image translation.

Ref: Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks

forward_test(*img*, *target_domain*, ***kwargs*)

Forward function for testing.

参数

- **img** (*tensor*) –Input image tensor.
- **target_domain** (*str*) –Target domain of output image.
- **kwargs** (*dict*) –Other arguments.

返回 Forward results.

返回类型 dict

_get_disc_loss(*outputs*)

Backward function for the discriminators.

参数 **outputs** (*dict*) –Dict of forward results.

返回 Discriminators' loss and loss dict.

返回类型 dict

_get_gen_loss(*outputs*)

Backward function for the generators.

参数 **outputs** (*dict*) –Dict of forward results.

返回 Generators' loss and loss dict.

返回类型 dict

_get_opposite_domain(*domain*)

Get the opposite domain respect to the input domain.

参数 **domain** (*str*) –The input domain.

返回 The opposite domain.

返回类型 str

train_step(*data: dict*, *optim_wrapper: mmengine.optim.OptimWrapperDict*)

Training step function.

参数

- **data_batch** (*dict*) –Dict of the input data batch.
- **optimizer** (*dict[torch.optim.Optimizer]*) –Dict of optimizers for the generators and discriminators.
- **ddp_reducer** (Reducer | None, optional) –Reducer from ddp. It is used to prepare for `backward()` in ddp. Defaults to None.

- **running_status** (*dict | None, optional*) –Contains necessary basic information for training, e.g., iteration number. Defaults to None.

返回 Dict of loss, information for logger, the number of samples and results for visualization.

返回类型 dict

test_step(*data: dict*) → mmagic.utils.typing.SampleList

Gets the generated image of given data. Same as *val_step()*.

参数 **data** (*dict*) –Data sampled from metric specific sampler. More details in *Metrics* and *Evaluator*.

返回 A list of `DataSample` contain generated results.

返回类型 SampleList

val_step(*data: dict*) → mmagic.utils.typing.SampleList

Gets the generated image of given data. Same as *val_step()*.

参数 **data** (*dict*) –Data sampled from metric specific sampler. More details in *Metrics* and *Evaluator*.

返回 A list of `DataSample` contain generated results.

返回类型 SampleList

class mmagic.models.editors.**DCGAN**(*generator: ModelType*, *discriminator: Optional[ModelType] = None*, *data_preprocessor: Optional[Union[dict, mmengine.Config]] = None*, *generator_steps: int = 1*, *discriminator_steps: int = 1*, *noise_size: Optional[int] = None*, *ema_config: Optional[Dict] = None*, *loss_config: Optional[Dict] = None*)

Bases: *mmagic.models.base_models.BaseGAN*

Implementation of *Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks*.

Paper link: <https://arxiv.org/abs/1511.06434>‘_ (DCGAN).

Detailed architecture can be found in `DCGANGenerator` # noqa and `DCGANDiscriminator` # noqa

disc_loss(*disc_pred_fake: torch.Tensor*, *disc_pred_real: torch.Tensor*) → Tuple

Get disc loss. DCGAN use the vanilla gan loss to train the discriminator.

参数

- **disc_pred_fake** (*Tensor*) –Discriminator’s prediction of the fake images.
- **disc_pred_real** (*Tensor*) –Discriminator’s prediction of the real images.

返回 Loss value and a dict of log variables.

返回类型 tuple[Tensor, dict]

gen_loss(*disc_pred_fake: torch.Tensor*) → Tuple

Get gen loss. DCGAN use the vanilla gan loss to train the generator.

参数 **disc_pred_fake** (*Tensor*) –Discriminator's prediction of the fake images.

返回 Loss value and a dict of log variables.

返回类型 tuple[Tensor, dict]

train_discriminator(*inputs: dict*, *data_samples:* mmagic.structures.DataSample, *optimizer_wrapper: mmengine.optim.OptimWrapper*) → Dict[str, torch.Tensor]

Train discriminator.

参数

- **inputs** (*dict*) –Inputs from dataloader.
- **data_samples** (*DataSample*) –Data samples from dataloader.
- **optim_wrapper** (*OptimWrapper*) –OptimWrapper instance used to update model parameters.

返回 A `dict` of tensor for logging.

返回类型 Dict[str, Tensor]

train_generator(*inputs: dict*, *data_samples:* mmagic.structures.DataSample, *optimizer_wrapper: mmengine.optim.OptimWrapper*) → Dict[str, torch.Tensor]

Train generator.

参数

- **inputs** (*dict*) –Inputs from dataloader.
- **data_samples** (*DataSample*) –Data samples from dataloader. Do not used in generator's training.
- **optim_wrapper** (*OptimWrapper*) –OptimWrapper instance used to update model parameters.

返回 A `dict` of tensor for logging.

返回类型 Dict[str, Tensor]

class mmagic.models.editors.**DenoisingUnet**(*image_size*, *in_channels=3*, *out_channels=None*, *base_channels=128*, *resblocks_per_downsample=3*, *num_timesteps=1000*, *use_rescale_timesteps=False*, *dropout=0*, *embedding_channels=- 1*, *num_classes=0*, *use_fp16=False*, *channels_cfg=None*, *output_cfg=dict(mean='eps', var='learned_range')*, *norm_cfg=dict(type='GN', num_groups=32)*, *act_cfg=dict(type='SiLU', inplace=False)*, *shortcut_kernel_size=1*, *use_scale_shift_norm=False*, *resblock_updown=False*, *num_heads=4*, *time_embedding_mode='sin'*, *time_embedding_cfg=None*, *resblock_cfg=dict(type='DenoisingResBlock')*, *attention_cfg=dict(type='MultiHeadAttention')*, *encoder_channels=None*, *downsample_conv=True*, *upsample_conv=True*, *downsample_cfg=dict(type='DenoisingDownsample')*, *upsample_cfg=dict(type='DenoisingUpsample')*, *attention_res=[16, 8]*, *pretrained=None*, *unet_type=''*, *down_block_types: Tuple[str] = ()*, *up_block_types: Tuple[str] = ()*, *cross_attention_dim=768*, *layers_per_block: int = 2*)

Bases: mmengine.model.BaseModule

Denoising Unet. This network receives a diffused image `x_t` and current timestep `t`, and returns a `output_dict` corresponding to the passed `output_cfg`.

`output_cfg` defines the number of channels and the meaning of the output. `output_cfg` mainly contains keys of `mean` and `var`, denoting how the network outputs mean and variance required for the denoising process. For `mean`: 1. `dict(mean='EPS')`: Model will predict noise added in the

diffusion process, and the `output_dict` will contain a key named `eps_t_pred`.

2. **`dict(mean='START_X')`: Model will direct predict the mean of the** original image *x_0*, and the `output_dict` will contain a key named `x_0_pred`.
3. **`dict(mean='X_TM1_PRED')`: Model will predict the mean of diffused** image at *t-1* timestep, and the `output_dict` will contain a key named `x_tm1_pred`.

For `var`: 1. `dict(var='FIXED_SMALL')` or `dict(var='FIXED_LARGE')`: Variance in

the denoising process is regarded as a fixed value. Therefore only ‘mean’ will be predicted, and the output channels will equal to the input image (e.g., three channels for RGB image.)

2. **`dict(var='LEARNED')`: Model will predict *log_variance* in the** denoising process, and the

`output_dict` will contain a key named `log_var`.

3. **`dict(var='LEARNED_RANGE')`: Model will predict an interpolation** factor and the *log_variance* will be calculated as *factor * upper_bound + (1-factor) * lower_bound*. The `output_dict` will contain a key named `factor`.

If `var` is not `FIXED_SMALL` or `FIXED_LARGE`, the number of output channels will be the double of input channels, where the first half part contains predicted mean values and the other part is the predicted variance values. Otherwise, the number of output channels equals to the input channels, only containing the predicted mean values.

参数

- **`image_size`** (*`int | list[int]`*) –The size of image to denoise.
- **`in_channels`** (*`int, optional`*) –The input channels of the input image. Defaults as `3`.
- **`out_channels`** (*`int, optional`*) –The output channels of the output prediction. Defaults as `None` for automaticaaly assigned by `var_mode`.
- **`base_channels`** (*`int, optional`*) –The basic channel number of the generator. The other layers contain channels based on this number. Defaults to `128`.
- **`resblocks_per_downsample`** (*`int, optional`*) –Number of ResBlock used between two downsample operations. The number of ResBlock between upsample operations will be the same value to keep symmetry. Defaults to 3.
- **`num_timesteps`** (*`int, optional`*) –The total timestep of the denoising process and the diffusion process. Defaults to `1000`.
- **`use_rescale_timesteps`** (*`bool, optional`*) –Whether rescale the input timesteps in range of [0, 1000]. Defaults to `True`.
- **`dropout`** (*`float, optional`*) –The probability of dropout operation of each ResBlock. Pass `0` to do not use dropout. Defaults as 0.
- **`embedding_channels`** (*`int, optional`*) –The output channels of time embedding layer and label embedding layer. If not passed (or passed `-1`), output channels of the embedding layers will set as four times of `base_channels`. Defaults to `-1`.
- **`num_classes`** (*`int, optional`*) –The number of conditional classes. If set to 0, this model will be degraded to an unconditional model. Defaults to 0.
- **`channels_cfg`** (*`list | dict[list], optional`*) –Config for input channels of the intermediate blocks. If list is passed, each element of the list indicates the scale factor for the input channels of the current block with regard to the `base_channels`. For block `i`, the input and output channels should be `channels_cfg[i] * base_channels` and `channels_cfg[i+1] * base_channels` If dict is provided, the key of the dict should be the output scale and corresponding value should be a list to define channels. Default: Please refer to `_default_channels_cfg`.

- **output_cfg** (*dict, optional*) –Config for output variables. Defaults to `dict(mean='eps', var='learned_range')`.
- **norm_cfg** (*dict, optional*) –The config for normalization layers. Defaults to `dict(type='GN', num_groups=32)`.
- **act_cfg** (*dict, optional*) –The config for activation layers. Defaults to `dict(type='SiLU', inplace=False)`.
- **shortcut_kernel_size** (*int, optional*) –The kernel size for shortcut conv in ResBlocks. The value of this argument will overwrite the default value of *resblock_cfg*. Defaults to *3*.
- **use_scale_shift_norm** (*bool, optional*) –Whether perform scale and shift after normalization operation. Defaults to True.
- **num_heads** (*int, optional*) –The number of attention heads. Defaults to 4.
- **time_embedding_mode** (*str, optional*) –Embedding method of `time_embedding`. Defaults to 'sin'.
- **time_embedding_cfg** (*dict, optional*) –Config for `time_embedding`. Defaults to None.
- **resblock_cfg** (*dict, optional*) –Config for ResBlock. Defaults to `dict(type='DenoisingResBlock')`.
- **attention_cfg** (*dict, optional*) –Config for attention operation. Defaults to `dict(type='MultiHeadAttention')`.
- **upsample_conv** (*bool, optional*) –Whether use conv in upsample block. Defaults to `True`.
- **downsample_conv** (*bool, optional*) –Whether use conv operation in downsample block. Defaults to `True`.
- **upsample_cfg** (*dict, optional*) –Config for upsample blocks. Defaults to `dict(type='DenoisingDownsample')`.
- **downsample_cfg** (*dict, optional*) –Config for downsample blocks. Defaults to `dict(type='DenoisingUpsample')`.
- **attention_res** (*int | list[int], optional*) –Resolution of feature maps to apply attention operation. Defaults to `[16, 8]`.
- **pretrained** (*str | dict, optional*) –Path for the pretrained model or dict containing information for pretrained models whose necessary key is 'ckpt_path'. Besides, you can also provide 'prefix' to load the generator part from the whole state dict. Defaults to None.

_default_channels_cfg

forward(*x_t*, *t*, *encoder_hidden_states=None*, *label=None*, *return_noise=False*)

Forward function. :param x_t: Diffused image at timestep *t* to denoise. :type x_t: torch.Tensor :param t: Current timestep. :type t: torch.Tensor :param label: You can directly give a

batch of label through a `torch.Tensor` or offer a callable function to sample a batch of label data. Otherwise, the `None` indicates to use the default label sampler.

参数 return_noise (*bool, optional*) –If True, inputted `x_t` and `t` will be returned in a dict with output desired by `output_cfg`. Defaults to False.

返回 If not `return_noise`

返回类型 torch.Tensor | dict

init_weights(*pretrained=None*)

Init weights for models.

We just use the initialization method proposed in the original paper.

参数 pretrained (*str, optional*) –Path for pretrained weights. If given None, pretrained weights will not be loaded. Defaults to None.

convert_to_fp16()

Convert the precision of the model to float16.

convert_to_fp32()

Convert the precision of the model to float32.

class mmagic.models.editors.**DeblurGanV2**(*generator: ModelType*, *discriminator: Optional[ModelType] = None*, *pixel_loss: Optional[Union[dict, str]] = None*, *disc_loss: Optional[Union[dict, str]] = None*, *adv_lambda: float = 0.001*, *warmup_num: int = 3*, *train_cfg: Optional[dict] = None*, *test_cfg: Optional[dict] = None*, *init_cfg: Optional[dict] = None*, *data_preprocessor: Optional[dict] = None*)

Bases: `mmengine.model.BaseModel`

Base class for all algorithmic models.

BaseModel implements the basic functions of the algorithmic model, such as weights initialize, batch inputs preprocess(see more information in `BaseDataPreprocessor`), parse losses, and update model parameters.

Subclasses inherit from BaseModel only need to implement the forward method, which implements the logic to calculate loss and predictions, then can be trained in the runner.

实际案例

```
>>> @MODELS.register_module()
>>> class ToyModel(BaseModel):
>>>
>>>     def __init__(self):
>>>         super().__init__()
>>>         self.backbone = nn.Sequential()
>>>         self.backbone.add_module('conv1', nn.Conv2d(3, 6, 5))
>>>         self.backbone.add_module('pool', nn.MaxPool2d(2, 2))
>>>         self.backbone.add_module('conv2', nn.Conv2d(6, 16, 5))
>>>         self.backbone.add_module('fc1', nn.Linear(16 * 5 * 5, 120))
>>>         self.backbone.add_module('fc2', nn.Linear(120, 84))
>>>         self.backbone.add_module('fc3', nn.Linear(84, 10))
>>>
>>>         self.criterion = nn.CrossEntropyLoss()
>>>
>>>     def forward(self, batch_inputs, data_samples, mode='tensor'):
>>>         data_samples = torch.stack(data_samples)
>>>         if mode == 'tensor':
>>>             return self.backbone(batch_inputs)
>>>         elif mode == 'predict':
>>>             feats = self.backbone(batch_inputs)
>>>             predictions = torch.argmax(feats, 1)
>>>             return predictions
>>>         elif mode == 'loss':
>>>             feats = self.backbone(batch_inputs)
>>>             loss = self.criterion(feats, data_samples)
>>>             return dict(loss=loss)
```

参数

- **data_preprocessor** (*dict, optional*) –The pre-process config of `BaseDataPreprocessor`.
- **init_cfg** (*dict, optional*) –The weight initialized config for `BaseModule`.

data_preprocessor

Used for pre-processing data sampled by dataloader to the format accepted by *`forward()`*.

Type `BaseDataPreprocessor`

init_cfg

Initialization config dict.

Type dict, optional

forward(*inputs: torch.Tensor*, *data_samples: Optional[List[*mmagic.structures.DataSample*]] = None*, *mode: str = 'tensor'*, ***kwargs*) → Union[torch.Tensor, List[*mmagic.structures.DataSample*], dict]

Returns losses or predictions of training, validation, testing, and simple inference process.

`forward` method of BaseModel is an abstract method, its subclasses must implement this method.

Accepts `inputs` and `data_samples` processed by *`data_preprocessor`*, and returns results according to mode arguments.

During non-distributed training, validation, and testing process, `forward` will be called by `BaseModel.train_step`, `BaseModel.val_step` and `BaseModel.val_step` directly.

During distributed data parallel training process, `MMSeparateDistributedDataParallel.train_step` will first call `DistributedDataParallel.forward` to enable automatic gradient synchronization, and then call `forward` to get training loss.

参数

- **inputs** (*torch.Tensor*) –batch input tensor collated by *`data_preprocessor`*.
- **data_samples** (*List[BaseDataElement], optional*) –data samples collated by *`data_preprocessor`*.
- **mode** (*str*) –mode should be one of `loss`, `predict` and `tensor`. Default: 'tensor'.
  - `loss`: Called by `train_step` and return loss `dict` used for logging
  - `predict`: Called by `val_step` and `test_step` and return list of `BaseDataElement` results used for computing metric.
  - `tensor`: Called by custom use to get `Tensor` type results.

返回

- If `mode == loss`, return a `dict` of loss tensor used for backward and logging.
- If `mode == val`, return a `list` of `BaseDataElement` for computing metric and getting inference result.
- If `mode == predict`, return a `list` of `BaseDataElement` for computing metric and getting inference result.
- If `mode == tensor`, return a tensor or `tuple` of tensor or `dict` or tensor for custom use.

返回类型 ForwardResults

convert_to_datasample(*predictions:* mmagic.structures.DataSample, *data_samples:* mmagic.structures.DataSample, *inputs: Optional[torch.Tensor]*) → List[*mmagic.structures.DataSample*]

Add predictions and destructed inputs (if passed) to data samples.

参数

- **predictions** (*DataSample*) –The predictions of the model.
- **data_samples** (*DataSample*) –The data samples loaded from dataloader.
- **inputs** (*Optional[torch.Tensor]*) –The input of model. Defaults to None.

返回 Modified data samples.

返回类型 List[DataSample]

forward_tensor(*inputs: torch.Tensor*, *data_samples: Optional[List[*mmagic.structures.DataSample*]] = None*, ***kwargs*) → torch.Tensor

Forward tensor. Returns result of simple forward.

参数

- **inputs** (*torch.Tensor*) –batch input tensor collated by *data_preprocessor*.
- **data_samples** (*List[BaseDataElement], optional*) –data samples collated by *data_preprocessor*.

返回 result of simple forward.

返回类型 Tensor

forward_inference(*inputs: torch.Tensor*, *data_samples: Optional[List[*mmagic.structures.DataSample*]] = None*, ***kwargs*) → List[*mmagic.structures.DataSample*]

Forward inference. Returns predictions of validation, testing, and simple inference.

参数

- **inputs** (*torch.Tensor*) –batch input tensor collated by *data_preprocessor*.
- **data_samples** (*List[BaseDataElement], optional*) –data samples collated by *data_preprocessor*.

返回 predictions.

返回类型 List[EditDataSample]

forward_train(*inputs*, *data_samples=None*, ***kwargs*)

Forward training. Losses of training is calculated in train_step.

参数

- **inputs** (*torch.Tensor*) –batch input tensor collated by *data_preprocessor*.
- **data_samples** (*List[BaseDataElement], optional*) –data samples collated by *data_preprocessor*.

返回 Result of `forward_tensor` with `training=True`.

返回类型 Tensor

val_step(*data: Union[tuple, dict, list]*) → list

Gets the predictions of given data.

Calls `self.data_preprocessor(data, False)` and `self(inputs, data_sample, mode='predict')` in order. Return the predictions which will be passed to evaluator.

参数 **data** (*dict or tuple or list*) –Data sampled from dataset.

返回 The predictions of given data.

返回类型 list

test_step(*data: Union[dict, tuple, list]*) → list

`BaseModel` implements `test_step` the same as `val_step`.

参数 **data** (*dict or tuple or list*) –Data sampled from dataset.

返回 The predictions of given data.

返回类型 list

_run_forward(*data: Union[dict, tuple, list]*, *mode: str*) → Union[Dict[str, torch.Tensor], list]

Unpacks data for *forward()*

参数

- **data** (*dict or tuple or list*) –Data sampled from dataset.
- **mode** (*str*) –Mode of forward.

返回 Results of training or testing mode.

返回类型 dict or list

train_step(*data: List[dict]*, *optim_wrapper: mmengine.optim.OptimWrapperDict*) → Dict[str, torch.Tensor]

Train step of GAN-based method.

参数

- **data** (*List[dict]*) –Data sampled from dataloader.
- **optim_wrapper** (*OptimWrapper*) –OptimWrapper instance used to update model parameters.

返回 A `dict` of tensor for logging.

返回类型 Dict[str, torch.Tensor]

g_step(*batch_outputs: torch.Tensor*, *batch_gt_data: torch.Tensor*)

G step of DobuleGAN: Calculate losses of generator.

参数

- **batch_outputs** (*Tensor*) –Batch output of generator.

- **batch_gt_data** (*Tensor*) –Batch GT data.

返回 Dict of losses.

返回类型 dict

d_step(*batch_outputs: torch.Tensor*, *batch_gt_data: torch.Tensor*)

D step of DobuleGAN: Calculate losses of generator.

参数

- **batch_outputs** (*Tensor*) –Batch output of generator.
- **batch_gt_data** (*Tensor*) –Batch GT data.

返回 Dict of losses.

返回类型 dict

g_step_with_optim(*batch_outputs: torch.Tensor*, *batch_gt_data: torch.Tensor*, *optim_wrapper: mmengine.optim.OptimWrapperDict*)

G step with optim of GAN: Calculate losses of generator and run optim.

参数

- **batch_outputs** (*Tensor*) –Batch output of generator.
- **batch_gt_data** (*Tensor*) –Batch GT data.
- **optim_wrapper** (*OptimWrapperDict*) –Optim wrapper dict.

返回 Dict of parsed losses.

返回类型 dict

d_step_with_optim(*batch_outputs: torch.Tensor*, *batch_gt_data: torch.Tensor*, *optim_wrapper: mmengine.optim.OptimWrapperDict*)

D step with optim of GAN: Calculate losses of discriminator and run optim.

参数

- **batch_outputs** (*Tensor*) –Batch output of generator.
- **batch_gt_data** (*Tensor*) –Batch GT data.
- **optim_wrapper** (*OptimWrapperDict*) –Optim wrapper dict.

返回 Dict of parsed losses.

返回类型 dict

extract_gt_data(*data_samples*)

extract gt data from data samples.

参数 **data_samples** (*list*) –List of DataSample.

返回 Extract gt data.

返回类型 Tensor

class mmagic.models.editors.**DeblurGanV2Discriminator**

Defines the discriminator for DeblurGanv2 with the specified arguments..

参数 **model** (*Str*) –Type of the discriminator model

class mmagic.models.editors.**DeblurGanV2Generator**

Defines the generator for DeblurGanv2 with the specified arguments..

参数 **model** (*Str*) –Type of the generator model

class mmagic.models.editors.**ContextualAttentionModule**(*unfold_raw_kernel_size=4*, *unfold_raw_stride=2*, *unfold_raw_padding=1*, *unfold_corr_kernel_size=3*, *unfold_corr_stride=1*, *unfold_corr_dilation=1*, *unfold_corr_padding=1*, *scale=0.5*, *fuse_kernel_size=3*, *softmax_scale=10*, *return_attention_score=True*)

Bases: mmengine.model.BaseModule

Contexture attention module.

The details of this module can be found in: Generative Image Inpainting with Contextual Attention

参数

- **unfold_raw_kernel_size** (*int*) –Kernel size used in unfolding raw feature. Default: 4.
- **unfold_raw_stride** (*int*) –Stride used in unfolding raw feature. Default: 2.
- **unfold_raw_padding** (*int*) –Padding used in unfolding raw feature. Default: 1.
- **unfold_corr_kernel_size** (*int*) –Kernel size used in unfolding context for computing correlation maps. Default: 3.
- **unfold_corr_stride** (*int*) –Stride used in unfolding context for computing correlation maps. Default: 1.
- **unfold_corr_dilation** (*int*) –Dilation used in unfolding context for computing correlation maps. Default: 1.
- **unfold_corr_padding** (*int*) –Padding used in unfolding context for computing correlation maps. Default: 1.

- **scale** (*float*) –The resale factor used in resize input features. Default: 0.5.
- **fuse_kernel_size** (*int*) –The kernel size used in fusion module. Default: 3.
- **softmax_scale** (*float*) –The scale factor for softmax function. Default: 10.
- **return_attention_score** (*bool*) –If True, the attention score will be returned. Default: True.

forward(*x*, *context*, *mask=None*)

Forward Function.

参数

- **x** (*torch.Tensor*) –Tensor with shape (n, c, h, w).
- **context** (*torch.Tensor*) –Tensor with shape (n, c, h, w).
- **mask** (*torch.Tensor*) –Tensor with shape (n, 1, h, w). Default: None.

返回 Features after contextural attention.

返回类型 tuple(torch.Tensor)

patch_correlation(*x*, *kernel*)

Calculate patch correlation.

参数

- **x** (*torch.Tensor*) –Input tensor.
- **kernel** (*torch.Tensor*) –Kernel tensor.

返回 Tensor with shape of (n, l, h, w).

返回类型 torch.Tensor

patch_copy_deconv(*attention_score*, *context_filter*)

Copy patches using deconv.

参数

- **attention_score** (*torch.Tensor*) –Tensor with shape of (n, l , h, w).
- **context_filter** (*torch.Tensor*) –Filter kernel.

返回 Tensor with shape of (n, c, h, w).

返回类型 torch.Tensor

fuse_correlation_map(*correlation_map*, *h_unfold*, *w_unfold*)

Fuse correlation map.

This operation is to fuse correlation map for increasing large consistent correlation regions.

The mechanism behind this op is simple and easy to understand. A standard ‘Eye’ matrix will be applied as a filter on the correlation map in horizontal and vertical direction.

The shape of input correlation map is (n, h_unfold*w_unfold, h, w). When adopting fusing, we will apply convolutional filter in the reshaped feature map with shape of (n, 1, h_unfold*w_fold, h*w).

A simple specification for horizontal direction is shown below:

```
       (h, (h, (h, (h,
        0)  1)  2)  3)  ...
(h, 0)
(h, 1)      1
(h, 2)          1
(h, 3)              1
...
```

calculate_unfold_hw(*input_size*, *kernel_size=3*, *stride=1*, *dilation=1*, *padding=0*)

Calculate (h, w) after unfolding.

The official implementation of *unfold* in pytorch will put the dimension (h, w) into *L*. Thus, this function is just to calculate the (h, w) according to the equation in: https://pytorch.org/docs/stable/nn.html#torch.nn.Unfold

calculate_overlap_factor(*attention_score*)

Calculate the overlap factor after applying deconv.

参数 **attention_score** (*torch.Tensor*) –The attention score with shape of (n, c, h, w).

返回 The overlap factor will be returned.

返回类型 torch.Tensor

mask_correlation_map(*correlation_map*, *mask*)

Add mask weight for correlation map.

Add a negative infinity number to the masked regions so that softmax function will result in ‘zero’ in those regions.

参数

- **correlation_map** (*torch.Tensor*) –Correlation map with shape of (n, h_unfold*w_unfold, h_map, w_map).
- **mask** (*torch.Tensor*) –Mask tensor with shape of (n, c, h, w). ‘1’ in the mask indicates masked region while ‘0’ indicates valid region.

返回 Updated correlation map with mask.

返回类型 torch.Tensor

im2col(*img*, *kernel_size*, *stride=1*, *padding=0*, *dilation=1*, *normalize=False*, *return_cols=False*)

Reshape image-style feature to columns.

This function is used for unfold feature maps to columns. The details of this function can be found in: https://pytorch.org/docs/1.1.0/nn.html?highlight=unfold#torch.nn.Unfold

参数

- **img** (*torch.Tensor*) –Features to be unfolded. The shape of this feature should be (n, c, h, w).
- **kernel_size** (*int*) –In this function, we only support square kernel with same height and width.
- **stride** (*int*) –Stride number in unfolding. Default: 1.
- **padding** (*int*) –Padding number in unfolding. Default: 0.
- **dilation** (*int*) –Dilation number in unfolding. Default: 1.
- **normalize** (*bool*) –If True, the unfolded feature will be normalized. Default: False.
- **return_cols** (*bool*) –The official implementation in PyTorch of unfolding will return features with shape of (n, c*$prod{kernel_size}$, L). If True, the features will be reshaped to (n, L, c, kernel_size, kernel_size). Otherwise, the results will maintain the shape as the official implementation.

返回 Unfolded columns. If *return_cols* is True, the shape of output tensor is *(n, L, c, kernel_size, kernel_size)*. Otherwise, the shape will be *(n, c*$prod{kernel_size}$, L)*.

返回类型 torch.Tensor

class mmagic.models.editors.**ContextualAttentionNeck**(*in_channels*, *conv_type='conv'*, *conv_cfg=None*, *norm_cfg=None*, *act_cfg=dict(type='ELU')*, *contextual_attention_args=dict(softmax_scale=10.0)*, ***kwargs*)

Bases: mmengine.model.BaseModule

Neck with contextual attention module.

参数

- **in_channels** (*int*) –The number of input channels.
- **conv_type** (*str*) –The type of conv module. In DeepFillv1 model, the *conv_type* should be ‘conv’. In DeepFillv2 model, the *conv_type* should be ‘gated_conv’.
- **conv_cfg** (*dict | None*) –Config of conv module. Default: None.
- **norm_cfg** (*dict | None*) –Config of norm module. Default: None.
- **act_cfg** (*dict | None*) –Config of activation layer. Default: dict(type=’ELU’).
- **contextual_attention_args** (*dict*) –Config of contextual attention module. Default: dict(softmax_scale=10.).

- **kwargs** (*keyword arguments*) –

_conv_type

forward(*x*, *mask*)

Forward Function.

参数

- **x** (*torch.Tensor*) –Input tensor with shape of (n, c, h, w).
- **mask** (*torch.Tensor*) –Input tensor with shape of (n, 1, h, w).

返回 Output tensor with shape of (n, c, h', w').

返回类型 torch.Tensor

class mmagic.models.editors.**DeepFillDecoder**(*in_channels*, *conv_type='conv'*, *norm_cfg=None*, *act_cfg=dict(type='ELU')*, *out_act_cfg=dict(type='clip', min=- 1.0, max=1.0)*, *channel_factor=1.0*, ***kwargs*)

Bases: mmengine.model.BaseModule

Decoder used in DeepFill model.

This implementation follows: Generative Image Inpainting with Contextual Attention

参数

- **in_channels** (*int*) –The number of input channels.
- **conv_type** (*str*) –The type of conv module. In DeepFillv1 model, the *conv_type* should be 'conv'. In DeepFillv2 model, the *conv_type* should be 'gated_conv'.
- **norm_cfg** (*dict*) –Config dict to build norm layer. Default: None.
- **act_cfg** (*dict*) –Config dict for activation layer, "elu" by default.
- **out_act_cfg** (*dict*) –Config dict for output activation layer. Here, we provide commonly used *clamp* or *clip* operation.
- **channel_factor** (*float*) –The scale factor for channel size. Default: 1.
- **kwargs** (*keyword arguments*) –

_conv_type

forward(*input_dict*)

Forward Function.

参数 **input_dict** (*dict | torch.Tensor*) –Input dict with middle features or torch.Tensor.

返回 Output tensor with shape of (n, c, h, w).

返回类型 torch.Tensor

class mmagic.models.editors.**DeepFillEncoder**(*in_channels=5*, *conv_type='conv'*, *norm_cfg=None*, *act_cfg=dict(type='ELU')*, *encoder_type='stage1'*, *channel_factor=1.0*, ***kwargs*)

Bases: mmengine.model.BaseModule

Encoder used in DeepFill model.

This implementation follows: Generative Image Inpainting with Contextual Attention

参数

- **in_channels** (*int*) –The number of input channels. Default: 5.
- **conv_type** (*str*) –The type of conv module. In DeepFillv1 model, the *conv_type* should be 'conv'. In DeepFillv2 model, the *conv_type* should be 'gated_conv'.
- **norm_cfg** (*dict*) –Config dict to build norm layer. Default: None.
- **act_cfg** (*dict*) –Config dict for activation layer, "elu" by default.
- **encoder_type** (*str*) –Type of the encoder. Should be one of ['stage1', 'stage2_conv', 'stage2_attention']. Default: 'stage1'.
- **channel_factor** (*float*) –The scale factor for channel size. Default: 1.
- **kwargs** (*keyword arguments*) –

_conv_type

forward(*x*)

Forward Function.

参数 **x** (*torch.Tensor*) –Input tensor with shape of (n, c, h, w).

返回 Output tensor with shape of (n, c, h', w').

返回类型 torch.Tensor

class mmagic.models.editors.**DeepFillRefiner**(*encoder_attention=dict(type='DeepFillEncoder', encoder_type='stage2_attention')*, *encoder_conv=dict(type='DeepFillEncoder', encoder_type='stage2_conv')*, *dilation_neck=dict(type='GLDilationNeck', in_channels=128, act_cfg=dict(type='ELU'))*, *contextual_attention=dict(type='ContextualAttentionNeck', in_channels=128)*, *decoder=dict(type='DeepFillDecoder', in_channels=256)*)

Bases: mmengine.model.BaseModule

Refiner used in DeepFill model.

This implementation follows: Generative Image Inpainting with Contextual Attention.

参数

- **encoder_attention** (*dict*) –Config dict for encoder used in branch with contextual attention module.
- **encoder_conv** (*dict*) –Config dict for encoder used in branch with just convolutional operation.
- **dilation_neck** (*dict*) –Config dict for dilation neck in branch with just convolutional operation.
- **contextual_attention** (*dict*) –Config dict for contextual attention neck.
- **decoder** (*dict*) –Config dict for decoder used to fuse and decode features.

forward(*x*, *mask*)

Forward Function.

参数

- **x** (*torch.Tensor*) –Input tensor with shape of (n, c, h, w).
- **mask** (*torch.Tensor*) –Input tensor with shape of (n, 1, h, w).

返回 Output tensor with shape of (n, c, h’, w’).

返回类型 torch.Tensor

class mmagic.models.editors.**DeepFillv1Discriminators**(*global_disc_cfg*, *local_disc_cfg*)

Bases: mmengine.model.BaseModule

Discriminators used in DeepFillv1 model.

In DeepFillv1 model, the discriminators are independent without any concatenation like Global&Local model. Thus, we call this model *DeepFillv1Discriminators*. There exist a global discriminator and a local discriminator with global and local input respectively.

The details can be found in: Generative Image Inpainting with Contextual Attention.

参数

- **global_disc_cfg** (*dict*) –Config dict for global discriminator.
- **local_disc_cfg** (*dict*) –Config dict for local discriminator.

forward(*x*)

Forward function.

参数 **x** (*tuple[torch.Tensor]*) –Contains global image and the local image patch.

返回 Contains the prediction from discriminators in global image and local image patch.

返回类型 tuple[torch.Tensor]

init_weights()

Init weights for models.

class mmagic.models.editors.**DeepFillv1Inpaintor**(*data_preprocessor: dict*, *encdec: dict*, *disc=None*, *loss_gan=None*, *loss_gp=None*, *loss_disc_shift=None*, *loss_composed_percep=None*, *loss_out_percep=False*, *loss_l1_hole=None*, *loss_l1_valid=None*, *loss_tv=None*, *stage1_loss_type=None*, *stage2_loss_type=None*, *train_cfg=None*, *test_cfg=None*, *init_cfg: Optional[dict] = None*)

Bases: *mmagic.models.base_models.TwoStageInpaintor*

Inpaintor for deepfillv1 method.

This inpaintor is implemented according to the paper: Generative image inpainting with contextual attention

Importantly, this inpaintor is an example for using custom training schedule based on *TwoStageInpaintor*.

The training pipeline of deepfillv1 is as following:

```
if cur_iter < iter_tc:
    update generator with only l1 loss
else:
    update discriminator
    if cur_iter > iter_td:
        update generator with l1 loss and adversarial loss
```

The new attribute *cur_iter* is added for recording current number of iteration. The *train_cfg* contains the setting of the training schedule:

```
train_cfg = dict(
    start_iter=0,
    disc_step=1,
    iter_tc=90000,
    iter_td=100000
)
```

iter_tc and *iter_td* correspond to the notation T_C and T_D of the original paper.

参数

- **generator** (*dict*) –Config for encoder-decoder style generator.
- **disc** (*dict*) –Config for discriminator.
- **loss_gan** (*dict*) –Config for adversarial loss.
- **loss_gp** (*dict*) –Config for gradient penalty loss.
- **loss_disc_shift** (*dict*) –Config for discriminator shift loss.
- **loss_composed_percep** (*dict*) –Config for perceptual and style loss with composed image as input.
- **loss_out_percep** (*dict*) –Config for perceptual and style loss with direct output as input.
- **loss_l1_hole** (*dict*) –Config for l1 loss in the hole.
- **loss_l1_valid** (*dict*) –Config for l1 loss in the valid region.
- **loss_tv** (*dict*) –Config for total variation loss.
- **train_cfg** (*dict*) –Configs for training scheduler. *disc_step* must be contained for indicates the discriminator updating steps in each training step.
- **test_cfg** (*dict*) –Configs for testing scheduler.
- **init_cfg** (*dict, optional*) –Initialization config dict.

forward_train_d(*data_batch*, *is_real*, *is_disc*)

Forward function in discriminator training step.

In this function, we modify the default implementation with only one discriminator. In DeepFillv1 model, they use two separated discriminators for global and local consistency.

参数

- **data_batch** (*torch.Tensor*) –Batch of real data or fake data.
- **is_real** (*bool*) –If True, the gan loss will regard this batch as real data. Otherwise, the gan loss will regard this batch as fake data.
- **is_disc** (*bool*) –If True, this function is called in discriminator training step. Otherwise, this function is called in generator training step. This will help us to compute different types of adversarial loss, like LSGAN.

返回 Contains the loss items computed in this function.

返回类型 dict

two_stage_loss(*stage1_data*, *stage2_data*, *gt*, *mask*, *masked_img*)

Calculate two-stage loss.

参数

- **stage1_data** (*dict*) –Contain stage1 results.
- **stage2_data** (*dict*) –Contain stage2 results.
- **gt** (*torch.Tensor*) –Ground-truth image.
- **mask** (*torch.Tensor*) –Mask image.
- **masked_img** (*torch.Tensor*) –Composition of mask image and ground-truth image.

返回 Dict contains the results computed within this function for visualization and dict contains the loss items computed in this function.

返回类型 tuple(dict)

calculate_loss_with_type(*loss_type*, *fake_res*, *fake_img*, *gt*, *mask*, *prefix='stage1_'*, *fake_local=None*)

Calculate multiple types of losses.

参数

- **loss_type** (*str*) –Type of the loss.
- **fake_res** (*torch.Tensor*) –Direct results from model.
- **fake_img** (*torch.Tensor*) –Composited results from model.
- **gt** (*torch.Tensor*) –Ground-truth tensor.
- **mask** (*torch.Tensor*) –Mask tensor.
- **prefix** (*str, optional*) –Prefix for loss name. Defaults to ‘stage1_’. # noqa
- **fake_local** (*torch.Tensor, optional*) –Local results from model. Defaults to None.

返回 Contain loss value with its name.

返回类型 dict

train_step(*data: List[dict]*, *optim_wrapper*)

Train step function.

In this function, the inpaintor will finish the train step following the pipeline:

1. get fake res/image
2. optimize discriminator (if have)
3. optimize generator

If *self.train_cfg.disc_step > 1*, the train step will contain multiple iterations for optimizing discriminator with different input data and only one iteration for optimizing generator after *disc_step* iterations for discriminator.

参数

- **data** (*List[dict]*) –Batch of data as input.

- **optim_wrapper** (*dict[torch.optim.Optimizer]*) –Dict with optimizers for generator and discriminator (if have).

返回 Dict with loss, information for logger, the number of samples and results for visualization.

返回类型 dict

class mmagic.models.editors.**DeepFillEncoderDecoder**(*stage1=dict(type='GLEncoderDecoder', encoder=dict(type='DeepFillEncoder'), decoder=dict(type='DeepFillDecoder', in_channels=128), dilation_neck=dict(type='GLDilationNeck', in_channels=128, act_cfg=dict(type='ELU'))), stage2=dict(type='DeepFillRefiner')*, *return_offset=False*)

Bases: mmengine.model.BaseModule

Two-stage encoder-decoder structure used in DeepFill model.

The details are in: Generative Image Inpainting with Contextual Attention

参数

- **stage1** (*dict*) –Config dict for building stage1 model. As DeepFill model uses Global&Local model as baseline in first stage, the stage1 model can be easily built with *GLEncoderDecoder*.
- **stage2** (*dict*) –Config dict for building stage2 model.
- **return_offset** (*bool*) –Whether to return offset feature in contextual attention module. Default: False.

forward(*x*)

Forward function.

参数 **x** (*torch.Tensor*) –This input tensor has the shape of (n, 5, h, w). In channel dimension, we concatenate [masked_img, ones, mask] as DeepFillv1 models do.

返回 The first two item is the results from first and second stage. If set *return_offset* as True, the offset will be returned as the third item.

返回类型 tuple[torch.Tensor]

init_weights()

Init weights for models.

class mmagic.models.editors.**DIC**(*generator*, *pixel_loss*, *align_loss*, *discriminator=None*, *gan_loss=None*, *feature_loss=None*, *train_cfg=None*, *test_cfg=None*, *init_cfg=None*, *data_preprocessor=None*)

Bases: mmagic.models.editors.srgan.SRGAN

DIC model for Face Super-Resolution.

Paper: Deep Face Super-Resolution with Iterative Collaboration between Attentive Recovery and Landmark Estimation.

参数

- **generator** (*dict*) –Config for the generator.
- **pixel_loss** (*dict*) –Config for the pixel loss.
- **align_loss** (*dict*) –Config for the align loss.
- **discriminator** (*dict*) –Config for the discriminator. Default: None.
- **gan_loss** (*dict*) –Config for the gan loss. Default: None.
- **feature_loss** (*dict*) –Config for the feature loss. Default: None.
- **train_cfg** (*dict*) –Config for train. Default: None.
- **test_cfg** (*dict*) –Config for testing. Default: None.
- **init_cfg** (*dict, optional*) –The weight initialized config for BaseModule. Default: None.
- **data_preprocessor** (*dict, optional*) –The pre-process config of BaseDataPreprocessor. Default: None.

forward_tensor(*inputs*, *data_samples=None*, *training=False*)

Forward tensor. Returns result of simple forward.

参数

- **inputs** (*torch.Tensor*) –batch input tensor collated by data_preprocessor.
- **data_samples** (*List[BaseDataElement], optional*) –data samples collated by data_preprocessor.
- **training** (*bool*) –Whether is training. Default: False.

返回

results of forward inference and forward train.

返回类型 (Tensor | Tuple[List[Tensor]])

if_run_g()

Calculates whether need to run the generator step.

if_run_d()

Calculates whether need to run the discriminator step.

g_step(*batch_outputs*, *batch_gt_data*)

G step of GAN: Calculate losses of generator.

参数

- **batch_outputs** (*Tensor*) –Batch output of generator.
- **batch_gt_data** (*Tensor*) –Batch GT data.

返回 Dict of losses.

返回类型 dict

train_step(*data: List[dict]*, *optim_wrapper: mmengine.optim.OptimWrapperDict*) → Dict[str, torch.Tensor]

Train step of GAN-based method.

参数

- **data** (*List[dict]*) –Data sampled from dataloader.
- **optim_wrapper** (*OptimWrapper*) –OptimWrapper instance used to update model parameters.

返回 A `dict` of tensor for logging.

返回类型 Dict[str, torch.Tensor]

static extract_gt_data(*data_samples*)

extract gt data from data samples.

参数 **data_samples** (*list*) –List of DataSample.

返回 Extract gt data.

返回类型 Tensor

class mmagic.models.editors.**DICNet**(*in_channels*, *out_channels*, *mid_channels*, *num_blocks=6*, *hg_mid_channels=256*, *hg_num_keypoints=68*, *num_steps=4*, *upscale_factor=8*, *detach_attention=False*, *prelu_init=0.2*, *num_heatmaps=5*, *num_fusion_blocks=7*, *init_cfg=None*)

Bases: `mmengine.model.BaseModule`

DIC network structure for face super-resolution.

Paper: Deep Face Super-Resolution with Iterative Collaboration between Attentive Recovery and Landmark Estimation

参数

- **in_channels** (*int*) –Number of channels in the input image

- **out_channels** (*int*) –Number of channels in the output image
- **mid_channels** (*int*) –Channel number of intermediate features. Default: 64
- **num_blocks** (*tuple[int]*) –Block numbers in the trunk network. Default: 6
- **hg_mid_channels** (*int*) –Channel number of intermediate features of HourGlass. Default: 256
- **hg_num_keypoints** (*int*) –Keypoint number of HourGlass. Default: 68
- **num_steps** (*int*) –Number of iterative steps. Default: 4
- **upscale_factor** (*int*) –Upsampling factor. Default: 8
- **detach_attention** (*bool*) –Detached from the current tensor for heatmap or not.
- **prelu_init** (*float*) –*init* of PReLU. Default: 0.2
- **num_heatmaps** (*int*) –Number of heatmaps. Default: 5
- **num_fusion_blocks** (*int*) –Number of fusion blocks. Default: 7
- **init_cfg** (*dict, optional*) –Initialization config dict. Default: None.

forward(*x*)

Forward function.

参数 **x** (*Tensor*) –Input tensor.

返回 Forward results. sr_outputs (list[Tensor]): forward sr results. heatmap_outputs (list[Tensor]): forward heatmap results.

返回类型 Tensor

class mmagic.models.editors.**FeedbackBlock**(*mid_channels*, *num_blocks*, *upscale_factor*, *padding=2*, *prelu_init=0.2*)

Bases: torch.nn.Module

Feedback Block of DIC.

It has a style of:

```
----- Module ----->
  ^             |
  |_____________|
```

参数

- **mid_channels** (*int*) –Number of channels in the intermediate features.
- **num_blocks** (*int*) –Number of blocks.
- **upscale_factor** (*int*) –upscale factor.

- **padding** (*int*) –Padding size. Default: 2.
- **prelu_init** (*float*) –*init* of PReLU. Default: 0.2

forward(*x*)

Forward function.

参数 **x** (*Tensor*) –Input tensor with shape (n, c, h, w).

返回 Forward results.

返回类型 Tensor

class mmagic.models.editors.**FeedbackBlockCustom**(*in_channels*, *mid_channels*, *num_blocks*, *upscale_factor*)

Bases: *FeedbackBlock*

Custom feedback block, will be used as the first feedback block.

参数

- **in_channels** (*int*) –Number of channels in the input features.
- **mid_channels** (*int*) –Number of channels in the intermediate features.
- **num_blocks** (*int*) –Number of blocks.
- **upscale_factor** (*int*) –upscale factor.

forward(*x*)

Forward function.

参数 **x** (*Tensor*) –Input tensor with shape (n, c, h, w).

返回 Forward results.

返回类型 Tensor

class mmagic.models.editors.**FeedbackBlockHeatmapAttention**(*mid_channels*, *num_blocks*, *upscale_factor*, *num_heatmaps*, *num_fusion_blocks*, *padding=2*, *prelu_init=0.2*)

Bases: *FeedbackBlock*

Feedback block with HeatmapAttention.

参数

- **in_channels** (*int*) –Number of channels in the input features.
- **mid_channels** (*int*) –Number of channels in the intermediate features.
- **num_blocks** (*int*) –Number of blocks.

- **upscale_factor** (*int*) –upscale factor.
- **padding** (*int*) –Padding size. Default: 2.
- **prelu_init** (*float*) –*init* of PReLU. Default: 0.2

forward(*x*, *heatmap*)

Forward function.

参数

- **x** (*Tensor*) –Input feature tensor.
- **heatmap** (*Tensor*) –Input heatmap tensor.

返回 Forward results.

返回类型 Tensor

class mmagic.models.editors.**LightCNN**(*in_channels*)

Bases: mmengine.model.BaseModule

LightCNN discriminator with input size 128 x 128.

It is used to train DICGAN.

参数 **in_channels** (*int*) –Channel number of inputs.

forward(*x*)

Forward function.

参数 **x** (*Tensor*) –Input tensor.

返回 Forward results.

返回类型 Tensor

init_weights(*pretrained=None*, *strict=True*)

Init weights for models.

参数

- **pretrained** (*str, optional*) –Path for pretrained weights. If given None, pretrained weights will not be loaded. Defaults to None.
- **strict** (*boo, optional*) –Whether strictly load the pretrained model. Defaults to True.

class mmagic.models.editors.**MaxFeature**(*in_channels*, *out_channels*, *kernel_size=3*, *stride=1*, *padding=1*, *filter_type='conv2d'*)

Bases: torch.nn.Module

Conv2d or Linear layer with max feature selector.

Generate feature maps with double channels, split them and select the max feature.

参数

- **in_channels** (*int*) –Channel number of inputs.
- **out_channels** (*int*) –Channel number of outputs.
- **kernel_size** (*int or tuple*) –Size of the convolving kernel.
- **stride** (*int or tuple, optional*) –Stride of the convolution. Default: 1
- **padding** (*int or tuple, optional*) –Zero-padding added to both sides of the input. Default: 1
- **filter_type** (*str*) –Type of filter. Options are 'conv2d' and 'linear'. Default: 'conv2d'.

forward(*x*)

Forward function.

参数 **x** (*Tensor*) –Input tensor.

返回 Forward results.

返回类型 Tensor

class mmagic.models.editors.**DIM**(*data_preprocessor*, *backbone*, *refiner=None*, *train_cfg=None*, *test_cfg=None*, *loss_alpha=None*, *loss_comp=None*, *loss_refine=None*, *init_cfg: Optional[dict] = None*)

Bases: *mmagic.models.base_models.BaseMattor*

Deep Image Matting model.

https://arxiv.org/abs/1703.03872

备注: For `(self.train_cfg.train_backbone, self.train_cfg.train_refiner)`:

- `(True, False)` corresponds to the encoder-decoder stage in the paper.
- `(False, True)` corresponds to the refinement stage in the paper.
- `(True, True)` corresponds to the fine-tune stage in the paper.

参数

- **data_preprocessor** (*dict, optional*) –Config of data pre-processor.
- **backbone** (*dict*) –Config of backbone.
- **refiner** (*dict*) –Config of refiner.
- **loss_alpha** (*dict*) –Config of the alpha prediction loss. Default: None.
- **loss_comp** (*dict*) –Config of the composition loss. Default: None.

- **loss_refine** (*dict*) –Config of the loss of the refiner. Default: None.
- **train_cfg** (*dict*) –Config of training. In `train_cfg`, `train_backbone` should be specified. If the model has a refiner, `train_refiner` should be specified.
- **test_cfg** (*dict*) –Config of testing. In `test_cfg`, If the model has a refiner, `train_refiner` should be specified.
- **init_cfg** (*dict, optional*) –The weight initialized config for `BaseModule`. Default: None.

property with_refiner

Whether the matting model has a refiner.

init_weights()

Initialize the model network weights.

train(*mode=True*)

Mode switcher.

参数 **mode** (*bool*) –whether to set training mode (`True`) or evaluation mode (`False`). Default: `True`.

freeze_backbone()

Freeze the backbone and only train the refiner.

_forward(*x: torch.Tensor*, *, *refine: bool = True*) → Tuple[torch.Tensor, torch.Tensor]

Raw forward function.

参数

- **x** (*torch.Tensor*) –Concatenation of merged image and trimap with shape (N, 4, H, W)
- **refine** (*bool*) –if forward through refiner

返回 pred_alpha, with shape (N, 1, H, W) torch.Tensor: pred_refine, with shape (N, 4, H, W)

返回类型 torch.Tensor

_forward_test(*inputs*)

Forward to get alpha prediction.

_forward_train(*inputs*, *data_samples*)

Defines the computation performed at every training call.

参数

- **inputs** (*torch.Tensor*) –Concatenation of normalized image and trimap shape (N, 4, H, W)
- **data_samples** (*list[DataSample]*) –Data samples containing: - gt_alpha (Tensor): Ground-truth of alpha

shape (N, 1, H, W), normalized to 0 to 1.

- **gt_fg (Tensor): Ground-truth of foreground** shape (N, C, H, W), normalized to 0 to 1.
- **gt_bg (Tensor): Ground-truth of background** shape (N, C, H, W), normalized to 0 to 1.

返回 Contains the loss items and batch information.

返回类型 dict

class mmagic.models.editors.**ClipWrapper**(*clip_type*, **args*, ***kwargs*)

Bases: torch.nn.Module

Clip Models wrapper.

We provide wrappers for the clip models of openai and mlfoundations, where the user can specify clip_type as clip or open_clip, and then initialize a clip model using the same arguments as in the original codebase. The following clip models settings are provided in the official repo of disco diffusion: | Setting | Source | Arguments | # noqa |:----------:|:----:|:----:| # noqa | ViTB32 | clip | name='ViT-B/32', jit=False | # noqa | ViTB16 | clip | name='ViT-B/16', jit=False | # noqa | ViTL14 | clip | name='ViT-L/14', jit=False | # noqa | ViTL14_336px | clip | name='ViT-L/14@336px', jit=False | # noqa | RN50 | clip | name='RN50', jit=False | # noqa | RN50x4 | clip | name='RN50x4', jit=False | # noqa | RN50x16 | clip | name='RN50x16', jit=False | # noqa | RN50x64 | clip | name='RN50x64', jit=False | # noqa | RN101 | clip | name='RN101', jit=False | # noqa | ViTB32_laion2b_e16 | open_clip | name='ViT-B-32', pretrained='laion2b_e16' | # noqa | ViTB32_laion400m_e31 | open_clip | model_name='ViT-B-32', pretrained='laion400m_e31' | # noqa | ViTB32_laion400m_32 | open_clip | model_name='ViT-B-32', pretrained='laion400m_e32' | # noqa | ViTB32quickgelu_laion400m_e31 | open_clip | model_name='ViT-B-32-quickgelu', pretrained='laion400m_e31' | # noqa | ViTB32quickgelu_laion400m_e32 | open_clip | model_name='ViT-B-32-quickgelu', pretrained='laion400m_e32' | # noqa | ViTB16_laion400m_e31 | open_clip | model_name='ViT-B-16', pretrained='laion400m_e31' | # noqa | ViTB16_laion400m_e32 | open_clip | model_name='ViT-B-16', pretrained='laion400m_e32' | # noqa | RN50_yffcc15m | open_clip | model_name='RN50', pretrained='yfcc15m' | # noqa | RN50_cc12m | open_clip | model_name='RN50', pretrained='cc12m' | # noqa | RN50_quickgelu_yfcc15m | open_clip | model_name='RN50-quickgelu', pretrained='yfcc15m' | # noqa | RN50_quickgelu_cc12m | open_clip | model_name='RN50-quickgelu', pretrained='cc12m' | # noqa | RN101_yfcc15m | open_clip | model_name='RN101', pretrained='yfcc15m' | # noqa | RN101_quickgelu_yfcc15m | open_clip | model_name='RN101-quickgelu', pretrained='yfcc15m' | # noqa

An example of a clip_modes_cfg is as follows:

Examples:

```
>>> # Use OpenAI's CLIP
>>> config = dict(
```

(下页继续)

(续上页)

```
>>>     type='ClipWrapper',
>>>     clip_type='clip',
>>>     name='ViT-B/32',
>>>     jit=False)
```

```
>>> # Use OpenCLIP
>>> config = dict(
>>>     type='ClipWrapper',
>>>     clip_type='open_clip',
>>>     model_name='RN50',
>>>     pretrained='yfcc15m')
```

```
>>> # Use CLIP from Hugging Face Transformers
>>> config = dict(
>>>     type='ClipWrapper',
>>>     clip_type='huggingface',
>>>     pretrained_model_name_or_path='runwayml/stable-diffusion-v1-5',
>>>     subfolder='text_encoder')
```

参数

- **clip_type** (*List[Dict]*) –The original source of the clip model. Whether be `clip`, `open_clip` or `hugging_face`.
- ***args** –Arguments to initialize corresponding clip model.
- ****kwargs** –Arguments to initialize corresponding clip model.

get_embedding_layer()

Function to get embedding layer of the clip model.

Only support for CLIPTextModel currently.

add_embedding(*embeddings: Union[dict, List[dict]]*)

set_only_embedding_trainable()

set_embedding_layer()

unset_embedding_layer()

forward(**args*, ***kwargs*)

Forward function.

class mmagic.models.editors.**DiscoDiffusion**(*unet*, *diffusion_scheduler*, *secondary_model=None*, *clip_models=[]*, *use_fp16=False*, *pretrained_cfgs=None*)

Bases: `torch.nn.Module`

Disco Diffusion (DD) is a Google Colab Notebook which leverages an AI Image generating technique called CLIP-Guided Diffusion to allow you to create compelling and beautiful images from just text inputs. Created by Somnai, augmented by Gandamu, and building on the work of RiversHaveWings, nshepperd, and many others.

Ref: Github Repo: https://github.com/alembics/disco-diffusion Colab: https://colab.research.google.com/github/alembics/disco-diffusion/blob/main/Disco_Diffusion.ipynb # noqa

参数

- **unet** (*ModelType*) –Config of denoising Unet.
- **diffusion_scheduler** (*ModelType*) –Config of diffusion_scheduler scheduler.
- **secondary_model** (*ModelType*) –A smaller secondary diffusion model trained by Katherine Crowson to remove noise from intermediate timesteps to prepare them for CLIP. Ref: https://twitter.com/rivershavewings/status/1462859669454536711 # noqa Defaults to None.
- **clip_models** (*list*) –Config of clip models. Defaults to [].
- **use_fp16** (*bool*) –Whether to use fp16 for unet model. Defaults to False.
- **pretrained_cfgs** (*dict*) –Path Config for pretrained weights. Usually this is a dict contains module name and the corresponding ckpt path. Defaults to None.

property device

Get current device of the model.

返回 The current device of the model.

返回类型 torch.device

load_pretrained_models(*pretrained_cfgs*)

Loading pretrained weights to model. `pretrained_cfgs` is a dict consist of module name as key and checkpoint path as value.

参数

- **pretrained_cfgs** (*dict*) –Path Config for pretrained weights.
- **the** (*Usually this is a dict contains module name and*) –
- **None.** (*corresponding ckpt path. Defaults to*) –

infer(*scheduler_kwargs=None*, *height=None*, *width=None*, *init_image=None*, *batch_size=1*, *num_inference_steps=100*, *skip_steps=0*, *show_progress=True*, *text_prompts=[]*, *image_prompts=[]*, *eta=0.8*, *clip_guidance_scale=5000*, *init_scale=1000*, *tv_scale=0.0*, *sat_scale=0.0*, *range_scale=150*, *cut_overview=[12] * 400 + [4] * 600*, *cut_innercut=[4] * 400 + [12] * 600*, *cut_ic_pow=[1] * 1000*, *cut_icgray_p=[0.2] * 400 + [0] * 600*, *cutn_batches=4*, *seed=None*)

Inference API for disco diffusion.

参数

- **scheduler_kwargs** (*dict*) –Args for infer time diffusion scheduler. Defaults to None.
- **height** (*int*) –Height of output image. Defaults to None.
- **width** (*int*) –Width of output image. Defaults to None.
- **init_image** (*str*) –Initial image at the start point of denoising. Defaults to None.
- **batch_size** (*int*) –Batch size. Defaults to 1.
- **num_inference_steps** (*int*) –Number of inference steps. Defaults to 1000.
- **skip_steps** (*int*) –Denoising steps to skip, usually set with `init_image`. Defaults to 0.
- **show_progress** (*bool*) –Whether to show progress. Defaults to False.
- **text_prompts** (*list*) –Text prompts. Defaults to [].
- **image_prompts** (*list*) –Image prompts, this is not the same as `init_image`, they works the same way with `text_prompts`. Defaults to [].
- **eta** (*float*) –Eta for ddim sampling. Defaults to 0.8.
- **clip_guidance_scale** (*int*) –The Scale of influence of prompts on output image. Defaults to 1000.
- **seed** (*int*) –Sampling seed. Defaults to None.

class mmagic.models.editors.**DreamBooth**(*vae: ModelType*, *text_encoder: ModelType*, *tokenizer: str*, *unet: ModelType*, *scheduler: ModelType*, *test_scheduler: Optional[ModelType] = None*, *lora_config: Optional[dict] = None*, *val_prompts: Union[str, List[str]] = None*, *class_prior_prompt: Optional[str] = None*, *num_class_images: Optional[int] = 3*, *prior_loss_weight: float = 0*, *finetune_text_encoder: bool = False*, *dtype: str = 'fp16'*, *enable_xformers: bool = True*, *noise_offset_weight: float = 0*, *tomesd_cfg: Optional[dict] = None*, *data_preprocessor: Optional[ModelType] = dict(type='DataPreprocessor')*, *init_cfg: Optional[dict] = None*)

Bases: mmagic.models.editors.stable_diffusion.stable_diffusion.StableDiffusion

Implementation of 'DreamBooth with Stable Diffusion.

<https://arxiv.org/abs/2208.12242>‘_ (DreamBooth).

参数

- **vae** (*Union[dict, nn.Module]*) –The config or module for VAE model.
- **text_encoder** (*Union[dict, nn.Module]*) –The config or module for text encoder.
- **tokenizer** (*str*) –The **name** for CLIP tokenizer.
- **unet** (*Union[dict, nn.Module]*) –The config or module for Unet model.
- **schedule** (*Union[dict, nn.Module]*) –The config or module for diffusion scheduler.
- **test_scheduler** (*Union[dict, nn.Module], optional*) –The config or module for diffusion scheduler in test stage (*self.infer*). If not passed, will use the same scheduler as *schedule*. Defaults to None.
- **lora_config** (*dict, optional*) –The config for LoRA finetuning. Defaults to None.
- **val_prompts** (*Union[str, List[str]], optional*) –The prompts for validation. Defaults to None.
- **class_prior_prompt** (*str, optional*) –The prompt for class prior loss.
- **num_class_images** (*int, optional*) –The number of images for class prior. Defaults to 3.
- **prior_loss_weight** (*float, optional*) –The weight for class prior loss. Defaults to 0.
- **fine_tune_text_encoder** (*bool, optional*) –Whether to fine-tune text encoder. Defaults to False.
- **dtype** (*str, optional*) –The dtype for the model. Defaults to ‘fp16’.
- **enable_xformers** (*bool, optional*) –Whether to use xformers. Defaults to True.
- **noise_offset_weight** (*bool, optional*) –The weight of noise offset introduced in https://www.crosslabs.org/blog/diffusion-with-offset-noise # noqa Defaults to 0.
- **tomesd_cfg** (*dict, optional*) –The config for TOMESD. Please refers to https://github.com/dbolya/tomesd and https://github.com/open-mmlab/mmagic/blob/main/mmagic/models/utils/tome_utils.py for detail. # noqa Defaults to None.
- **data_preprocessor** (*dict, optional*) –The pre-process config of `BaseDataPreprocessor`. Defaults to

  dict(type=’DataPreprocessor’).

- **init_cfg** (*dict, optional*) –The weight initialized config for `BaseModule`. Defaults to None/

generate_class_prior_images(*num_batches=None*)

Generate images for class prior loss.

参数 **num_batches** (*int*) –Number of batches to generate images. If not passed, all images will be generated in one forward. Defaults to None.

prepare_model()

Prepare model for training.

Move model to target dtype and disable gradient for some models.

set_lora()

Set LORA for model.

val_step(*data: dict*) → mmagic.utils.typing.SampleList

Gets the generated image of given data. Calls `self.data_preprocessor` and `self.infer` in order. Return the generated results which will be passed to evaluator or visualizer.

参数 **data** (*dict or tuple or list*) –Data sampled from dataset.

返回 Generated image or image dict.

返回类型 SampleList

test_step(*data: dict*) → mmagic.utils.typing.SampleList

Gets the generated image of given data. Calls `self.data_preprocessor` and `self.infer` in order. Return the generated results which will be passed to evaluator or visualizer.

参数 **data** (*dict or tuple or list*) –Data sampled from dataset.

返回 Generated image or image dict.

返回类型 SampleList

train_step(*data*, *optim_wrapper*)

Implements the default model training process including preprocessing, model forward propagation, loss calculation, optimization, and back-propagation.

During non-distributed training. If subclasses do not override the *train_step()*, `EpochBasedTrainLoop` or `IterBasedTrainLoop` will call this method to update model parameters. The default parameter update process is as follows:

1. Calls `self.data_processor(data, training=False)` to collect batch_inputs and corresponding data_samples(labels).
2. Calls `self(batch_inputs, data_samples, mode='loss')` to get raw loss
3. Calls `self.parse_losses` to get `parsed_losses` tensor used to backward and dict of loss tensor used to log messages.

4. Calls `optim_wrapper.update_params(loss)` to update model.

参数

- **data** (*dict or tuple or list*) –Data sampled from dataset.
- **optim_wrapper** (*OptimWrapper*) –OptimWrapper instance used to update model parameters.

返回 A `dict` of tensor for logging.

返回类型 Dict[str, torch.Tensor]

abstract forward(*inputs: torch.Tensor*, *data_samples: Optional[list] = None*, *mode: str = 'tensor'*) → Union[Dict[str, torch.Tensor], list]

forward is not implemented now.

class mmagic.models.editors.**EDSRNet**(*in_channels*, *out_channels*, *mid_channels=64*, *num_blocks=16*, *upscale_factor=4*, *res_scale=1*, *rgb_mean=[0.4488, 0.4371, 0.404]*, *rgb_std=[1.0, 1.0, 1.0]*)

Bases: `mmengine.model.BaseModule`

EDSR network structure.

Paper: Enhanced Deep Residual Networks for Single Image Super-Resolution. Ref repo: https://github.com/thstkdgus35/EDSR-PyTorch

参数

- **in_channels** (*int*) –Channel number of inputs.
- **out_channels** (*int*) –Channel number of outputs.
- **mid_channels** (*int*) –Channel number of intermediate features. Default: 64.
- **num_blocks** (*int*) –Block number in the trunk network. Default: 16.
- **upscale_factor** (*int*) –Upsampling factor. Support 2^n and 3. Default: 4.
- **res_scale** (*float*) –Used to scale the residual in residual block. Default: 1.
- **rgb_mean** (*list[float]*) –Image mean in RGB orders. Default: [0.4488, 0.4371, 0.4040], calculated from DIV2K dataset.
- **rgb_std** (*list[float]*) –Image std in RGB orders. In EDSR, it uses [1.0, 1.0, 1.0]. Default: [1.0, 1.0, 1.0].

forward(*x*)

Forward function.

参数 **x** (*Tensor*) –Input tensor with shape (n, c, h, w).

返回 Forward results.

返回类型 Tensor

class mmagic.models.editors.**EDVR**(*generator*, *pixel_loss*, *train_cfg=None*, *test_cfg=None*, *init_cfg=None*, *data_preprocessor=None*)

Bases: mmagic.models.BaseEditModel

EDVR model for video super-resolution.

EDVR: Video Restoration with Enhanced Deformable Convolutional Networks.

参数

- **generator** (*dict*) –Config for the generator structure.
- **pixel_loss** (*dict*) –Config for pixel-wise loss.
- **train_cfg** (*dict*) –Config for training. Default: None.
- **test_cfg** (*dict*) –Config for testing. Default: None.
- **init_cfg** (*dict, optional*) –The weight initialized config for BaseModule.
- **data_preprocessor** (*dict, optional*) –The pre-process config of BaseDataPreprocessor.

forward_train(*inputs*, *data_samples=None*)

Forward training. Returns dict of losses of training.

参数

- **inputs** (*torch.Tensor*) –batch input tensor collated by data_preprocessor.
- **data_samples** (*List[BaseDataElement], optional*) –data samples collated by data_preprocessor.

返回 Dict of losses.

返回类型 dict

class mmagic.models.editors.**EDVRNet**(*in_channels*, *out_channels*, *mid_channels=64*, *num_frames=5*, *deform_groups=8*, *num_blocks_extraction=5*, *num_blocks_reconstruction=10*, *center_frame_idx=2*, *with_tsa=True*, *init_cfg=None*)

Bases: mmengine.model.BaseModule

EDVR network structure for video super-resolution.

Now only support X4 upsampling factor. Paper: EDVR: Video Restoration with Enhanced Deformable Convolutional Networks.

参数

- **in_channels** (*int*) –Channel number of inputs.

- **out_channels** (*int*) –Channel number of outputs.
- **mid_channels** (*int*) –Channel number of intermediate features. Default: 64.
- **num_frames** (*int*) –Number of input frames. Default: 5.
- **deform_groups** (*int*) –Deformable groups. Defaults: 8.
- **num_blocks_extraction** (*int*) –Number of blocks for feature extraction. Default: 5.
- **num_blocks_reconstruction** (*int*) –Number of blocks for reconstruction. Default: 10.
- **center_frame_idx** (*int*) –The index of center frame. Frame counting from 0. Default: 2.
- **with_tsa** (*bool*) –Whether to use TSA module. Default: True.
- **init_cfg** (*dict, optional*) –Initialization config dict. Default: None.

forward(*x*)

Forward function for EDVRNet.

参数 **x** (*Tensor*) –Input tensor with shape (n, t, c, h, w).

返回 SR center frame with shape (n, c, h, w).

返回类型 Tensor

init_weights()

Init weights for models.

class mmagic.models.editors.**EG3D**(*generator: ModelType*, *discriminator: Optional[ModelType] = None*, *camera: Optional[ModelType] = None*, *data_preprocessor: Optional[Union[dict, mmengine.Config]] = None*, *generator_steps: int = 1*, *discriminator_steps: int = 1*, *noise_size: Optional[int] = None*, *ema_config: Optional[Dict] = None*, *loss_config: Optional[Dict] = None*)

Bases: *mmagic.models.base_models.BaseConditionalGAN*

Implementation of *Efficient Geometry-aware 3D Generative Adversarial Networks*

<https://openaccess.thecvf.com/content/CVPR2022/papers/Chan_Efficient_Geometry-Aware_3D_Generative_Adversarial_Networks_CVPR_2022_paper.pdf>_ (EG3D). # noqa

Detailed architecture can be found in `TriplaneGenerator` and `DualDiscriminator`

参数

- **generator** (*ModelType*) –The config or model of the generator.

- **discriminator** (*Optional[ModelType]*) –The config or model of the discriminator. Defaults to None.
- **camera** (*Optional[ModelType]*) –The pre-defined camera to sample random camera position. If you want to generate images or videos via high-level API, you must set this argument. Defaults to None.
- **data_preprocessor** (*Optional[Union[dict, Config]]*) –The pre-process config or `DataPreprocessor`.
- **generator_steps** (*int*) –Number of times the generator was completely updated before the discriminator is updated. Defaults to 1.
- **discriminator_steps** (*int*) –Number of times the discriminator was completely updated before the generator is updated. Defaults to 1.
- **noise_size** (*Optional[int]*) –Size of the input noise vector. Default to 128.
- **num_classes** (*Optional[int]*) –The number classes you would like to generate. Defaults to None.
- **ema_config** (*Optional[Dict]*) –The config for generator's exponential moving average setting. Defaults to None.
- **loss_config** (*Optional[Dict]*) –The config for training losses. Defaults to None.

label_fn(*label: Optional[torch.Tensor] = None*, *num_batches: int = 1*) → torch.Tensor

Label sampling function for EG3D model.

参数 **label** (*Optional[Tensor]*) –Conditional for EG3D model. If not passed, `self.camera` will be used to sample random camera-to-world and intrinsics matrix. Defaults to None.

返回 Conditional input for EG3D model.

返回类型 torch.Tensor

data_sample_to_label(*data_sample: mmagic.utils.typing.SampleList*) → Optional[torch.Tensor]

Get labels from input *data_sample* and pack to *torch.Tensor*. If no label is found in the passed *data_sample*, *None* would be returned.

参数 **data_sample** (*List[DataSample]*) –Input data samples.

返回 Packed label tensor.

返回类型 Optional[torch.Tensor]

pack_to_data_sample(*output: Dict[str, torch.Tensor]*, *data_sample: Optional[*mmagic.structures.DataSample*] = None*) → *mmagic.structures.DataSample*

Pack output to data sample. If `data_sample` is not passed, a new DataSample will be instantiated. Otherwise, outputs will be added to the passed datasample.

参数

- **output** (*Dict[Tensor]*) –Output of the model.
- **index** (*int*) –The index to save.
- **data_sample** (*DataSample, optional*) –Data sample to save outputs. Defaults to None.

返回 Data sample with packed outputs.

返回类型 DataSample

forward(*inputs: mmagic.utils.typing.ForwardInputs*, *data_samples: Optional[list] = None*, *mode: Optional[str] = None*) → List[*mmagic.structures.DataSample*]

Sample images with the given inputs. If forward mode is 'ema' or 'orig', the image generated by corresponding generator will be returned. If forward mode is 'ema/orig', images generated by original generator and EMA generator will both be returned in a dict.

参数

- **inputs** (*ForwardInputs*) –Dict containing the necessary information (e.g. noise, num_batches, mode) to generate image.
- **data_samples** (*Optional[list]*) –Data samples collated by `data_preprocessor`. Defaults to None.
- **mode** (*Optional[str]*) –*mode* is not used in `BaseConditionalGAN`. Defaults to None.

返回 Generated images or image dict.

返回类型 List[DataSample]

interpolation(*num_images: int*, *num_batches: int = 4*, *mode: str = 'both'*, *sample_model: str = 'orig'*, *show_pbar: bool = True*) → List[dict]

Interpolation input and return a list of output results. We support three kinds of interpolation mode:

- **'camera': First generate style code with random noise and forward** camera. Then synthesis images with interpolated camera position and fixed style code.
- **'conditioning': First generate style code with fixed noise and** interpolated camera. Then synthesis images with style codes and forward camera.
- 'both': Generate images with interpolated camera position.

参数

- **num_images** (*int*) –The number of images want to generate.
- **num_batches** (*int, optional*) –The number of batches to generate at one time. Defaults to 4.

- **mode** (*str, optional*) –The interpolation mode. Supported choices are 'both', 'camera', and 'conditioning'. Defaults to 'both'.
- **sample_model** (*str, optional*) –The model used to generate images, support 'orig' and 'ema'. Defaults to 'orig'.
- **show_pbar** (*bool, optional*) –Whether display a progress bar during interpolation. Defaults to True.

返回 The list of output dict of each frame.

返回类型 List[dict]

class mmagic.models.editors.**ESRGAN**(*generator*, *discriminator=None*, *gan_loss=None*, *pixel_loss=None*, *perceptual_loss=None*, *train_cfg=None*, *test_cfg=None*, *init_cfg=None*, *data_preprocessor=None*)

Bases: mmagic.models.editors.srgan.SRGAN

Enhanced SRGAN model for single image super-resolution.

Ref: ESRGAN: Enhanced Super-Resolution Generative Adversarial Networks. It uses RaGAN for GAN updates: The relativistic discriminator: a key element missing from standard GAN.

参数

- **generator** (*dict*) –Config for the generator.
- **discriminator** (*dict*) –Config for the discriminator. Default: None.
- **gan_loss** (*dict*) –Config for the gan loss. Note that the loss weight in gan loss is only for the generator.
- **pixel_loss** (*dict*) –Config for the pixel loss. Default: None.
- **perceptual_loss** (*dict*) –Config for the perceptual loss. Default: None.
- **train_cfg** (*dict*) –Config for training. Default: None. You may change the training of gan by setting: *disc_steps*: how many discriminator updates after one generate update; *disc_init_steps*: how many discriminator updates at the start of the training. These two keys are useful when training with WGAN.
- **test_cfg** (*dict*) –Config for testing. Default: None.
- **init_cfg** (*dict, optional*) –The weight initialized config for BaseModule. Default: None.

g_step(*batch_outputs: torch.Tensor*, *batch_gt_data: torch.Tensor*)

G step of GAN: Calculate losses of generator.

参数

- **batch_outputs** (*Tensor*) –Batch output of generator.

- **batch_gt_data** (*Tensor*) –Batch GT data.

返回 Dict of losses.

返回类型 dict

d_step_real(*batch_outputs: torch.Tensor*, *batch_gt_data: torch.Tensor*)

D step of real data.

参数

- **batch_outputs** (*Tensor*) –Batch output of generator.
- **batch_gt_data** (*Tensor*) –Batch GT data.

返回 Dict of losses.

返回类型 dict

d_step_fake(*batch_outputs: torch.Tensor*, *batch_gt_data*)

D step of fake data.

参数

- **batch_outputs** (*Tensor*) –Batch output of generator.
- **batch_gt_data** (*Tensor*) –Batch GT data.

返回 Dict of losses.

返回类型 dict

class mmagic.models.editors.**RRDBNet**(*in_channels*, *out_channels*, *mid_channels=64*, *num_blocks=23*, *growth_channels=32*, *upscale_factor=4*, *init_cfg=None*)

Bases: mmengine.model.BaseModule

Networks consisting of Residual in Residual Dense Block, which is used in ESRGAN and Real-ESRGAN.

ESRGAN: Enhanced Super-Resolution Generative Adversarial Networks. Real-ESRGAN: Training Real-World Blind Super-Resolution with Pure Synthetic Data. # noqa: E501 Currently, it supports [x1/x2/x4] upsampling scale factor.

参数

- **in_channels** (*int*) –Channel number of inputs.
- **out_channels** (*int*) –Channel number of outputs.
- **mid_channels** (*int*) –Channel number of intermediate features. Default: 64
- **num_blocks** (*int*) –Block number in the trunk network. Defaults: 23
- **growth_channels** (*int*) –Channels for each growth. Default: 32.
- **upscale_factor** (*int*) –Upsampling factor. Support x1, x2 and x4. Default: 4.

- **init_cfg** (*dict, optional*) –Initialization config dict. Default: None.

_supported_upscale_factors = [1, 2, 4]

forward(*x*)

Forward function.

参数 **x** (*Tensor*) –Input tensor with shape (n, c, h, w).

返回 Forward results.

返回类型 Tensor

init_weights()

Init weights for models.

class mmagic.models.editors.**FBADecoder**(*pool_scales*, *in_channels*, *channels*, *conv_cfg=None*, *norm_cfg=dict(type='BN')*, *act_cfg=dict(type='ReLU')*, *align_corners=False*)

Bases: torch.nn.Module

Decoder for FBA matting.

参数

- **pool_scales** (*tuple[int]*) –Pooling scales used in
- **Module.** (*Pooling Pyramid*) –
- **in_channels** (*int*) –Input channels.
- **channels** (*int*) –Channels after modules, before conv_seg.
- **conv_cfg** (*dict|None*) –Config of conv layers.
- **norm_cfg** (*dict|None*) –Config of norm layers.
- **act_cfg** (*dict*) –Config of activation layers.
- **align_corners** (*bool*) –align_corners argument of F.interpolate.

init_weights(*pretrained=None*)

Init weights for the model.

参数 **pretrained** (*str, optional*) –Path for pretrained weights. If given None, pretrained weights will not be loaded. Defaults to None.

forward(*inputs*)

Forward function.

参数 **inputs** (*dict*) –Output dict of FbaEncoder.

返回 Predicted alpha, fg and bg of the current batch.

返回类型 tuple(Tensor)

class mmagic.models.editors.**FBAResnetDilated**(*depth: int*, *in_channels: int = 3*, *stem_channels: int = 64*, *base_channels: int = 64*, *num_stages: int = 4*, *strides: Sequence[int] = (1, 2, 2, 2)*, *dilations: Sequence[int] = (1, 1, 2, 4)*, *deep_stem: bool = False*, *avg_down: bool = False*, *frozen_stages: int = - 1*, *act_cfg: dict = dict(type='ReLU')*, *conv_cfg: Optional[dict] = None*, *norm_cfg: dict = dict(type='BN')*, *with_cp: bool = False*, *multi_grid: Optional[Sequence[int]] = None*, *contract_dilation: bool = False*, *zero_init_residual: bool = True*)

Bases: *mmagic.models.archs.ResNet*

ResNet-based encoder for FBA image matting.

forward(*x*)

Forward function.

参数 **x** (*Tensor*) –Input tensor with shape (N, C, H, W).

返回 Output tensor.

返回类型 Tensor

class mmagic.models.editors.**FLAVR**(*generator: dict*, *pixel_loss: dict*, *train_cfg: Optional[dict] = None*, *test_cfg: Optional[dict] = None*, *required_frames: int = 2*, *step_frames: int = 1*, *init_cfg: Optional[dict] = None*, *data_preprocessor: Optional[dict] = None*)

Bases: *mmagic.models.base_models.BasicInterpolator*

FLAVR model for video interpolation.

Paper: FLAVR: Flow-Agnostic Video Representations for Fast Frame Interpolation

Ref repo: https://github.com/tarun005/FLAVR

参数

- **generator** (*dict*) –Config for the generator structure.
- **pixel_loss** (*dict*) –Config for pixel-wise loss.
- **train_cfg** (*dict*) –Config for training. Default: None.
- **test_cfg** (*dict*) –Config for testing. Default: None.
- **required_frames** (*int*) –Required frames in each process. Default: 2
- **step_frames** (*int*) –Step size of video frame interpolation. Default: 1
- **init_cfg** (*dict, optional*) –The weight initialized config for BaseModule.

- **data_preprocessor** (*dict, optional*) –The pre-process config of `BaseDataPreprocessor`.

init_cfg

Initialization config dict.

Type dict, optional

data_preprocessor

Used for pre-processing data sampled by dataloader to the format accepted by `forward()`.

Type `BaseDataPreprocessor`

static merge_frames(*input_tensors*, *output_tensors*)

merge input frames and output frames.

Interpolate a frame between the given two frames.

Merged from [[in1, in2, in3, in4], [in2, in3, in4, in5], ···] [[out1], [out2], [out3], ···]

to [in1, in2, out1, in3, out2, ···, in(-3), out(-1), in(-2), in(-1)]

参数

- **input_tensors** (*Tensor*) –The input frames with shape [n, 4, c, h, w]
- **output_tensors** (*Tensor*) –The output frames with shape [n, 1, c, h, w].

返回 The final frames.

返回类型 list[np.array]

class mmagic.models.editors.**FLAVRNet**(*num_input_frames*, *num_output_frames*, *mid_channels_list=[512, 256, 128, 64]*, *encoder_layers_list=[2, 2, 2, 2]*, *bias=False*, *norm_cfg=None*, *join_type='concat'*, *up_mode='transpose'*, *init_cfg=None*)

Bases: `mmengine.model.BaseModule`

PyTorch implementation of FLAVR for video frame interpolation.

Paper: FLAVR: Flow-Agnostic Video Representations for Fast Frame Interpolation

Ref repo: https://github.com/tarun005/FLAVR

参数

- **num_input_frames** (*int*) –Number of input frames.
- **num_output_frames** (*int*) –Number of output frames.
- **mid_channels_list** (*list[int]*) –List of number of mid channels. Default: [512, 256, 128, 64]

- **encoder_layers_list** (*list[int]*) –List of number of layers in encoder. Default: [2, 2, 2, 2]
- **bias** (*bool*) –If `True`, adds a learnable bias to the conv layers. Default: `True`
- **norm_cfg** (*dict | None*) –Config dict for normalization layer. Default: None
- **join_type** (*str*) –Join type of tensors from decoder and encoder. Candidates are `concat` and `add`. Default: `concat`
- **up_mode** (*str*) –Up-mode UpConv3d, candidates are `transpose` and `trilinear`. Default: `transpose`
- **init_cfg** (*dict, optional*) –Initialization config dict. Default: None.

forward(*images: torch.Tensor*)

Forward function.

参数 **images** (*Tensor*) –Input frames tensor with shape (N, T, C, H, W).

返回 Output tensor.

返回类型 out (Tensor)

class mmagic.models.editors.**GCA**(*data_preprocessor*, *backbone*, *loss_alpha=None*, *init_cfg: Optional[dict] = None*, *train_cfg=None*, *test_cfg=None*)

Bases: *mmagic.models.base_models.BaseMattor*

Guided Contextual Attention image matting model.

https://arxiv.org/abs/2001.04069

参数

- **data_preprocessor** (*dict, optional*) –The pre-process config of `BaseDataPreprocessor`.
- **backbone** (*dict*) –Config of backbone.
- **loss_alpha** (*dict*) –Config of the alpha prediction loss. Default: None.
- **init_cfg** (*dict, optional*) –Initialization config dict. Default: None.
- **train_cfg** (*dict*) –Config of training. In `train_cfg`, `train_backbone` should be specified. If the model has a refiner, `train_refiner` should be specified.
- **test_cfg** (*dict*) –Config of testing. In `test_cfg`, If the model has a refiner, `train_refiner` should be specified.

_forward(*inputs*)

Forward function.

参数 **inputs** (*torch.Tensor*) –Input tensor.

返回 Output tensor.

返回类型 Tensor

_forward_test(*inputs*)

Forward function for testing GCA model.

参数 **inputs** (*torch.Tensor*) –batch input tensor.

返回 Output tensor of model.

返回类型 Tensor

_forward_train(*inputs*, *data_samples*)

Forward function for training GCA model.

参数

- **inputs** (*torch.Tensor*) –batch input tensor collated by data_preprocessor.
- **data_samples** (*List[BaseDataElement]*) –data samples collated by data_preprocessor.

返回 Contains the loss items and batch information.

返回类型 dict

class mmagic.models.editors.**GGAN**(*generator: ModelType*, *discriminator: Optional[ModelType] = None*, *data_preprocessor: Optional[Union[dict, mmengine.Config]] = None*, *generator_steps: int = 1*, *discriminator_steps: int = 1*, *noise_size: Optional[int] = None*, *ema_config: Optional[Dict] = None*, *loss_config: Optional[Dict] = None*)

Bases: *mmagic.models.base_models.BaseGAN*

Implementation of *Geometric GAN*.

<https://arxiv.org/abs/1705.02894>‘_(GGAN).

disc_loss(*disc_pred_fake: torch.Tensor*, *disc_pred_real: torch.Tensor*) → Tuple

Get disc loss. GGAN use hinge loss to train the discriminator.

参数

- **disc_pred_fake** (*Tensor*) –Discriminator’s prediction of the fake images.
- **disc_pred_real** (*Tensor*) –Discriminator’s prediction of the real images.

返回 Loss value and a dict of log variables.

返回类型 tuple[Tensor, dict]

gen_loss(*disc_pred_fake*)

Get disc loss. GGAN use hinge loss to train the generator.

参数 **disc_pred_fake** (*Tensor*) –Discriminator’s prediction of the fake images.

返回 Loss value and a dict of log variables.

返回类型 tuple[Tensor, dict]

train_discriminator(*inputs: dict*, *data_samples: List[*mmagic.structures.DataSample*]*, *optimizer_wrapper: mmengine.optim.OptimWrapper*) → Dict[str, torch.Tensor]

Train discriminator.

参数

- **inputs** (*dict*) –Inputs from dataloader.
- **data_samples** (*List[DataSample]*) –Data samples from dataloader.
- **optim_wrapper** (*OptimWrapper*) –OptimWrapper instance used to update model parameters.

返回 A dict of tensor for logging.

返回类型 Dict[str, Tensor]

train_generator(*inputs: dict*, *data_samples: List[*mmagic.structures.DataSample*]*, *optimizer_wrapper: mmengine.optim.OptimWrapper*) → Dict[str, torch.Tensor]

Train generator.

参数

- **inputs** (*dict*) –Inputs from dataloader.
- **data_samples** (*List[DataSample]*) –Data samples from dataloader. Do not used in generator' s training.
- **optim_wrapper** (*OptimWrapper*) –OptimWrapper instance used to update model parameters.

返回 A dict of tensor for logging.

返回类型 Dict[str, Tensor]

class mmagic.models.editors.**GLEANStyleGANv2**(*in_size*, *out_size*, *img_channels=3*, *rrdb_channels=64*, *num_rrdbs=23*, *style_channels=512*, *num_mlps=8*, *channel_multiplier=2*, *blur_kernel=[1, 3, 3, 1]*, *lr_mlp=0.01*, *default_style_mode='mix'*, *eval_style_mode='single'*, *mix_prob=0.9*, *init_cfg=None*, *fp16_enabled=False*, *bgr2rgb=False*)

Bases: mmengine.model.BaseModule

GLEAN (using StyleGANv2) architecture for super-resolution.

Paper: GLEAN: Generative Latent Bank for Large-Factor Image Super-Resolution, CVPR, 2021

This method makes use of StyleGAN2 and hence the arguments mostly follow that in ‘StyleGAN2v2Generator’.

In StyleGAN2, we use a static architecture composing of a style mapping module and number of convolutional style blocks. More details can be found in: Analyzing and Improving the Image Quality of StyleGAN CVPR2020.

You can load pretrained model through passing information into `pretrained` argument. We have already offered official weights as follows:

- stylegan2-ffhq-config-f: http://download.openmmlab.com/mmediting/stylegan2/official_weights/stylegan2-ffhq-config-f-official_20210327_171224-bce9310c.pth # noqa
- stylegan2-horse-config-f: http://download.openmmlab.com/mmediting/stylegan2/official_weights/stylegan2-horse-config-f-official_20210327_173203-ef3e69ca.pth # noqa
- stylegan2-car-config-f: http://download.openmmlab.com/mmediting/stylegan2/official_weights/stylegan2-car-config-f-official_20210327_172340-8cfe053c.pth # noqa
- stylegan2-cat-config-f: http://download.openmmlab.com/mmediting/stylegan2/official_weights/stylegan2-cat-config-f-official_20210327_172444-15bc485b.pth # noqa
- stylegan2-church-config-f: http://download.openmmlab.com/mmediting/stylegan2/official_weights/stylegan2-church-config-f-official_20210327_172657-1d42b7d1.pth # noqa

If you want to load the ema model, you can just use following codes:

```
# ckpt_http is one of the valid path from http source
generator = StyleGANv2Generator(1024, 512,
                                pretrained=dict(
                                    ckpt_path=ckpt_http,
                                    prefix='generator_ema'))
```

Of course, you can also download the checkpoint in advance and set `ckpt_path` with local path. If you just want to load the original generator (not the ema model), please set the prefix with ‘generator’.

Note that our implementation allows to generate BGR image, while the original StyleGAN2 outputs RGB images by default. Thus, we provide `bgr2rgb` argument to convert the image space.

参数

- **in_size** (*int*) –The size of the input image.
- **out_size** (*int*) –The output size of the StyleGAN2 generator.
- **img_channels** (*int*) –Number of channels of the input images. 3 for RGB image and 1 for grayscale image. Default: 3.
- **rrdb_channels** (*int*) –Number of channels of the RRDB features. Default: 64.
- **num_rrdbs** (*int*) –Number of RRDB blocks in the encoder. Default: 23.
- **style_channels** (*int*) –The number of channels for style code. Default: 512.

- **num_mlps** (*int, optional*) –The number of MLP layers. Defaults to 8.
- **channel_multiplier** (*int, optional*) –The multiplier factor for the channel number. Defaults to 2.
- **blur_kernel** (*list, optional*) –The blurry kernel. Defaults to [1, 3, 3, 1].
- **lr_mlp** (*float, optional*) –The learning rate for the style mapping layer. Defaults to 0.01.
- **default_style_mode** (*str, optional*) –The default mode of style mixing. In training, we adopt mixing style mode in default. However, in the evaluation, we use ‘single’ style mode. *[‘mix’, ‘single’]* are currently supported. Defaults to ‘mix’.
- **eval_style_mode** (*str, optional*) –The evaluation mode of style mixing. Defaults to ‘single’.
- **mix_prob** (*float, optional*) –Mixing probability. The value should be in range of [0, 1]. Defaults to 0.9.
- **init_cfg** (*dict, optional*) –Initialization config dict. Default: None.
- **fp16_enabled** (*bool, optional*) –Whether to use fp16 training in this module. Defaults to False.
- **bgr2rgb** (*bool, optional*) –Whether to flip the image channel dimension. Defaults to False.

forward(*lq*)

Forward function.

参数 **lq** (*Tensor*) –Input LR image with shape (n, c, h, w).

返回 Output HR image.

返回类型 Tensor

class mmagic.models.editors.**GLDecoder**(*in_channels=256*, *norm_cfg=None*, *act_cfg=dict(type='ReLU')*, *out_act='clip'*)

Bases: mmengine.model.BaseModule

Decoder used in Global&Local model.

This implementation follows: Globally and locally Consistent Image Completion

参数

- **in_channels** (*int*) –Channel number of input feature.
- **norm_cfg** (*dict*) –Config dict to build norm layer.
- **act_cfg** (*dict*) –Config dict for activation layer, “relu” by default.

- **out_act** (*str*) –Output activation type, “clip” by default. Noted that in our implementation, we clip the output with range [-1, 1].

forward(*x*)

Forward Function.

参数 **x** (*torch.Tensor*) –Input tensor with shape of (n, c, h, w).

返回 Output tensor with shape of (n, c, h’, w’).

返回类型 torch.Tensor

class mmagic.models.editors.**GLDilationNeck**(*in_channels=256*, *conv_type='conv'*, *norm_cfg=None*, *act_cfg=dict(type='ReLU')*, ***kwargs*)

Bases: mmengine.model.BaseModule

Dilation Backbone used in Global&Local model.

This implementation follows: Globally and locally Consistent Image Completion

参数

- **in_channels** (*int*) –Channel number of input feature.
- **conv_type** (*str*) –The type of conv module. In DeepFillv1 model, the *conv_type* should be ‘conv’. In DeepFillv2 model, the *conv_type* should be ‘gated_conv’.
- **norm_cfg** (*dict*) –Config dict to build norm layer.
- **act_cfg** (*dict*) –Config dict for activation layer, “relu” by default.
- **kwargs** (*keyword arguments*) –

_conv_type

forward(*x*)

Forward Function.

参数 **x** (*torch.Tensor*) –Input tensor with shape of (n, c, h, w).

返回 Output tensor with shape of (n, c, h’, w’).

返回类型 torch.Tensor

class mmagic.models.editors.**GLEncoder**(*norm_cfg=None*, *act_cfg=dict(type='ReLU')*)

Bases: mmengine.model.BaseModule

Encoder used in Global&Local model.

This implementation follows: Globally and locally Consistent Image Completion

参数

- **norm_cfg** (*dict*) –Config dict to build norm layer.

- **act_cfg** (*dict*) –Config dict for activation layer, "relu" by default.

forward(*x*)

Forward Function.

参数 **x** (*torch.Tensor*) –Input tensor with shape of (n, c, h, w).

返回 Output tensor with shape of (n, c, h', w').

返回类型 torch.Tensor

class mmagic.models.editors.**GLEncoderDecoder**(*encoder=dict(type='GLEncoder')*, *decoder=dict(type='GLDecoder')*, *dilation_neck=dict(type='GLDilationNeck')*)

Bases: mmengine.model.BaseModule

Encoder-Decoder used in Global&Local model.

This implementation follows: Globally and locally Consistent Image Completion

The architecture of the encoder-decoder is: (conv2d x 6) -> (dilated conv2d x 4) -> (conv2d or deconv2d x 7)

参数

- **encoder** (*dict*) –Config dict to encoder.
- **decoder** (*dict*) –Config dict to build decoder.
- **dilation_neck** (*dict*) –Config dict to build dilation neck.

forward(*x*)

Forward Function.

参数 **x** (*torch.Tensor*) –Input tensor with shape of (n, c, h, w).

返回 Output tensor with shape of (n, c, h', w').

返回类型 torch.Tensor

class mmagic.models.editors.**AblatedDiffusionModel**(*data_preprocessor*, *unet*, *diffusion_scheduler*, *use_fp16=False*, *classifier=None*, *classifier_scale=1.0*, *rgb2bgr=False*, *pretrained_cfgs=None*)

Bases: mmengine.model.BaseModel

Guided diffusion Model.

参数

- **data_preprocessor** (*dict, optional*) –The pre-process config of BaseDataPreprocessor.
- **unet** (*ModelType*) –Config of denoising Unet.

- **diffusion_scheduler** (*ModelType*) –Config of diffusion_scheduler scheduler.
- **use_fp16** (*bool*) –Whether to use fp16 for unet model. Defaults to False.
- **classifier** (*ModelType*) –Config of classifier. Defaults to None.
- **pretrained_cfgs** (*dict*) –Path Config for pretrained weights. Usually this is a dict contains module name and the corresponding ckpt path.Defaults to None.

property device

Get current device of the model.

返回 The current device of the model.

返回类型 torch.device

load_pretrained_models(*pretrained_cfgs*)

summary

参数 **pretrained_cfgs** (*_type_*) –_description_

infer(*scheduler_kwargs=None*, *init_image=None*, *batch_size=1*, *num_inference_steps=1000*, *labels=None*, *classifier_scale=0.0*, *show_progress=False*)

summary

参数

- **init_image** (*_type_, optional*) –_description_. Defaults to None.
- **batch_size** (*int, optional*) –_description_. Defaults to 1.
- **num_inference_steps** (*int, optional*) –_description_. Defaults to 1000.
- **labels** (*_type_, optional*) –_description_. Defaults to None.
- **show_progress** (*bool, optional*) –_description_. Defaults to False.

返回 _description_

返回类型 _type_

forward(*inputs: mmagic.utils.typing.ForwardInputs*, *data_samples: Optional[list] = None*, *mode: Optional[str] = None*) → List[*mmagic.structures.DataSample*]

summary

参数

- **inputs** (*ForwardInputs*) –_description_
- **data_samples** (*Optional[list], optional*) –_description_. Defaults to None.
- **mode** (*Optional[str], optional*) –_description_. Defaults to None.

返回 _description_

返回类型 List[DataSample]

val_step(*data: dict*) → mmagic.utils.typing.SampleList

Gets the generated image of given data.

Calls `self.data_preprocessor(data)` and `self(inputs, data_sample, mode=None)` in order. Return the generated results which will be passed to evaluator.

参数 **data** (*dict*) –Data sampled from metric specific sampler. More details in *Metrics* and *Evaluator*.

返回 Generated image or image dict.

返回类型 SampleList

test_step(*data: dict*) → mmagic.utils.typing.SampleList

Gets the generated image of given data. Same as *val_step()*.

参数 **data** (*dict*) –Data sampled from metric specific sampler. More details in *Metrics* and *Evaluator*.

返回 Generated image or image dict.

返回类型 List[DataSample]

train_step(*data: dict*, *optim_wrapper: mmengine.optim.OptimWrapperDict*)

summary

参数

- **data** (*dict*) –_description_
- **optim_wrapper** (*OptimWrapperDict*) –_description_

返回 _description_

返回类型 _type_

get_module(*model: torch.nn.Module*, *module_name: str*) → torch.nn.Module

Get an inner module from model.

Since we will wrapper DDP for some model, we have to judge whether the module can be indexed directly.

参数

- **model** (*nn.Module*) –This model may wrapped with DDP or not.
- **module_name** (*str*) –The name of specific module.

返回 Returned sub module.

返回类型 nn.Module

class mmagic.models.editors.**IconVSRNet**(*mid_channels=64*, *num_blocks=30*, *keyframe_stride=5*, *padding=2*, *spynet_pretrained=None*, *edvr_pretrained=None*)

Bases: mmengine.model.BaseModule

IconVSR network structure for video super-resolution.

Support only x4 upsampling.

Paper: BasicVSR: The Search for Essential Components in Video Super-Resolution and Beyond, CVPR, 2021

参数

- **mid_channels** (*int*) –Channel number of the intermediate features. Default: 64.
- **num_blocks** (*int*) –Number of residual blocks in each propagation branch. Default: 30.
- **keyframe_stride** (*int*) –Number determining the keyframes. If stride=5, then the (0, 5, 10, 15, ⋯)-th frame will be the keyframes. Default: 5.
- **padding** (*int*) –Number of frames to be padded at two ends of the sequence. 2 for REDS and 3 for Vimeo-90K. Default: 2.
- **spynet_pretrained** (*str*) –Pre-trained model path of SPyNet. Default: None.
- **edvr_pretrained** (*str*) –Pre-trained model path of EDVR (for refill). Default: None.

spatial_padding(*lrs*)

Apply padding spatially.

Since the PCD module in EDVR requires that the resolution is a multiple of 4, we apply padding to the input LR images if their resolution is not divisible by 4.

参数 **lrs** (*Tensor*) –Input LR sequence with shape (n, t, c, h, w).

返回 Padded LR sequence with shape (n, t, c, h_pad, w_pad).

返回类型 Tensor

check_if_mirror_extended(*lrs*)

Check whether the input is a mirror-extended sequence.

If mirror-extended, the i-th (i=0, ⋯, t-1) frame is equal to the (t-1-i)-th frame.

参数 **lrs** (*tensor*) –Input LR images with shape (n, t, c, h, w)

compute_refill_features(*lrs*, *keyframe_idx*)

Compute keyframe features for information-refill.

Since EDVR-M is used, padding is performed before feature computation. :param lrs: Input LR images with shape (n, t, c, h, w) :type lrs: Tensor :param keyframe_idx: The indices specifying the keyframes. :type keyframe_idx: list(int)

返回

The keyframe features. Each key corresponds to the indices in keyframe_idx.

返回类型 dict(Tensor)

compute_flow(*lrs*)

Compute optical flow using SPyNet for feature warping.

Note that if the input is an mirror-extended sequence, 'flows_forward' is not needed, since it is equal to 'flows_backward.flip(1)'.

参数 **lrs** (*tensor*) –Input LR images with shape (n, t, c, h, w)

返回

Optical flow. 'flows_forward' corresponds to the flows used for forward-time propagation (current to previous). 'flows_backward' corresponds to the flows used for backward-time propagation (current to next).

返回类型 tuple(Tensor)

forward(*lrs*)

Forward function for IconVSR.

参数 **lrs** (*Tensor*) –Input LR tensor with shape (n, t, c, h, w).

返回 Output HR tensor with shape (n, t, c, 4h, 4w).

返回类型 Tensor

class mmagic.models.editors.**DepthwiseIndexBlock**(*in_channels*, *norm_cfg=dict(type='BN')*, *use_context=False*, *use_nonlinear=False*, *mode='o2o'*)

Bases: mmengine.model.BaseModule

Depthwise index block.

From https://arxiv.org/abs/1908.00672.

参数

- **in_channels** (*int*) –Input channels of the holistic index block.
- **norm_cfg** (*dict*) –Config dict for normalization layer. Default: dict(type='BN').
- **use_context** (*bool, optional*) –Whether use larger kernel size in index block. Refer to the paper for more information. Defaults to False.

- **use_nonlinear** (*bool*) –Whether add a non-linear conv layer in the index blocks. Default: False.
- **mode** (*str*) –Mode of index block. Should be ‘o2o’ or ‘m2o’. In ‘o2o’ mode, the group of the conv layers is 1; In ‘m2o’ mode, the group of the conv layer is *in_channels*.

forward(*x*)

Forward function.

参数 **x** (*Tensor*) –Input feature map with shape (N, C, H, W).

返回 Encoder index feature and decoder index feature.

返回类型 tuple(Tensor)

class mmagic.models.editors.**HolisticIndexBlock**(*in_channels*, *norm_cfg=dict(type='BN')*, *use_context=False*, *use_nonlinear=False*)

Bases: mmengine.model.BaseModule

Holistic Index Block.

From https://arxiv.org/abs/1908.00672.

参数

- **in_channels** (*int*) –Input channels of the holistic index block.
- **norm_cfg** (*dict*) –Config dict for normalization layer. Default: dict(type=’BN’).
- **use_context** (*bool, optional*) –Whether use larger kernel size in index block. Refer to the paper for more information. Defaults to False.
- **use_nonlinear** (*bool*) –Whether add a non-linear conv layer in the index block. Default: False.

forward(*x*)

Forward function.

参数 **x** (*Tensor*) –Input feature map with shape (N, C, H, W).

返回 Encoder index feature and decoder index feature.

返回类型 tuple(Tensor)

class mmagic.models.editors.**IndexedUpsample**(*in_channels*, *out_channels*, *kernel_size=5*, *norm_cfg=dict(type='BN')*, *conv_module=ConvModule*, *init_cfg: Optional[dict] = None*)

Bases: mmengine.model.BaseModule

Indexed upsample module.

参数

- **in_channels** (*int*) –Input channels.
- **out_channels** (*int*) –Output channels.
- **kernel_size** (*int, optional*) –Kernel size of the convolution layer. Defaults to 5.
- **norm_cfg** (*dict, optional*) –Config dict for normalization layer. Defaults to dict(type='BN').
- **conv_module** (*ConvModule | DepthwiseSeparableConvModule, optional*) –Conv module. Defaults to ConvModule.
- **init_cfg** (*dict, optional*) –Initialization config dict. Default: None.

init_weights()

Init weights for the module.

forward(*x*, *shortcut*, *dec_idx_feat=None*)

Forward function.

参数

- **x** (*Tensor*) –Input feature map with shape (N, C, H, W).
- **shortcut** (*Tensor*) –The shortcut connection with shape (N, C, H', W').
- **dec_idx_feat** (*Tensor, optional*) –The decode index feature map with shape (N, C, H', W'). Defaults to None.

返回 Output tensor with shape (N, C, H', W').

返回类型 Tensor

class mmagic.models.editors.**IndexNet**(*data_preprocessor*, *backbone*, *loss_alpha=None*, *loss_comp=None*, *init_cfg=None*, *train_cfg=None*, *test_cfg=None*)

Bases: *mmagic.models.base_models.BaseMattor*

IndexNet matting model.

This implementation follows: Indices Matter: Learning to Index for Deep Image Matting

参数

- **data_preprocessor** (*dict, optional*) –The pre-process config of BaseDataPreprocessor.
- **backbone** (*dict*) –Config of backbone.
- **train_cfg** (*dict*) –Config of training. In 'train_cfg', 'train_backbone' should be specified.
- **test_cfg** (*dict*) –Config of testing.

- **init_cfg** (*dict, optional*) –The weight initialized config for BaseModule.
- **loss_alpha** (*dict*) –Config of the alpha prediction loss. Default: None.
- **loss_comp** (*dict*) –Config of the composition loss. Default: None.

_forward(*inputs*)

Forward function.

参数 **inputs** (*torch.Tensor*) –Input tensor.

返回 Output tensor.

返回类型 Tensor

_forward_test(*inputs*)

Forward function for testing IndexNet model.

参数 **inputs** (*torch.Tensor*) –batch input tensor.

返回 Output tensor of model.

返回类型 Tensor

_forward_train(*inputs*, *data_samples*)

Forward function for training IndexNet model.

参数

- **inputs** (*torch.Tensor*) –batch input tensor collated by data_preprocessor.
- **data_samples** (*List[BaseDataElement]*) –data samples collated by data_preprocessor.

返回 Contains the loss items and batch information.

返回类型 dict

class mmagic.models.editors.**IndexNetDecoder**(*in_channels*, *kernel_size=5*, *norm_cfg=dict(type='BN')*, *separable_conv=False*, *init_cfg: Optional[dict] = None*)

Bases: mmengine.model.BaseModule

Decoder for IndexNet.

Please refer to https://arxiv.org/abs/1908.00672.

参数

- **in_channels** (*int*) –Input channels of the decoder.
- **kernel_size** (*int, optional*) –Kernel size of the convolution layer. Defaults to 5.

- **norm_cfg** (*None | dict, optional*) –Config dict for normalization layer. Defaults to dict(type=’BN’).
- **separable_conv** (*bool*) –Whether to use separable conv. Default: False.
- **init_cfg** (*dict, optional*) –Initialization config dict. Default: None.

init_weights()

Init weights for the module.

forward(*inputs*)

Forward function.

参数 **inputs** (*dict*) –Output dict of IndexNetEncoder.

返回 Predicted alpha matte of the current batch.

返回类型 Tensor

class mmagic.models.editors.**IndexNetEncoder**(*in_channels*, *out_stride=32*, *width_mult=1*, *index_mode='m2o'*, *aspp=True*, *norm_cfg=dict(type='BN')*, *freeze_bn=False*, *use_nonlinear=True*, *use_context=True*, *init_cfg: Optional[dict] = None*)

Bases: mmengine.model.BaseModule

Encoder for IndexNet.

Please refer to https://arxiv.org/abs/1908.00672.

参数

- **in_channels** (*int, optional*) –Input channels of the encoder.
- **out_stride** (*int, optional*) –Output stride of the encoder. For example, if *out_stride* is 32, the input feature map or image will be downsample to the 1/32 of original size. Defaults to 32.
- **width_mult** (*int, optional*) –Width multiplication factor of channel dimension in MobileNetV2. Defaults to 1.
- **index_mode** (*str, optional*) –Index mode of the index network. It must be one of {‘holistic’, ‘o2o’, ‘m2o’}. If it is set to ‘holistic’, then Holistic index network will be used as the index network. If it is set to ‘o2o’ (or ‘m2o’), when O2O (or M2O) Depthwise index network will be used as the index network. Defaults to ‘m2o’.
- **aspp** (*bool, optional*) –Whether use ASPP module to augment output feature. Defaults to True.
- **norm_cfg** (*None | dict, optional*) –Config dict for normalization layer. Defaults to dict(type=’BN’).

- **freeze_bn** (*bool, optional*) –Whether freeze batch norm layer. Defaults to False.
- **use_nonlinear** (*bool, optional*) –Whether use nonlinearity in index network. Refer to the paper for more information. Defaults to True.
- **use_context** (*bool, optional*) –Whether use larger kernel size in index network. Refer to the paper for more information. Defaults to True.
- **init_cfg** (*dict, optional*) –Initialization config dict. Default: None.

引发

- **ValueError** –out_stride must 16 or 32.
- **NameError** –Supported index_mode are {‘holistic’, ‘o2o’, ‘m2o’}.

_make_layer(*layer_setting*, *norm_cfg*)

train(*mode=True*)

Set BatchNorm modules in the model to evaluation mode.

init_weights()

Init weights for the model.

Initialization is based on self._init_cfg

参数 **pretrained** (*str, optional*) –Path for pretrained weights. If given None, pretrained weights will not be loaded. Defaults to None.

forward(*x*)

Forward function.

参数 **x** (*Tensor*) –Input feature map with shape (N, C, H, W).

返回 Output tensor, shortcut feature and decoder index feature.

返回类型 dict

class mmagic.models.editors.**InstColorization**(*data_preprocessor: Union[dict, mmengine.config.Config]*, *image_model*, *instance_model*, *fusion_model*, *color_data_opt*, *which_direction='AtoB'*, *loss=None*, *init_cfg=None*, *train_cfg=None*, *test_cfg=None*)

Bases: mmengine.model.BaseModel

Colorization InstColorization method.

This Colorization is implemented according to the paper: Instance-aware Image Colorization, CVPR 2020

Adapted from ‘https://github.com/ericsujw/InstColorization.git’‘InstColorization/models/train_model’ Copyright (c) 2020, Su, under MIT License.

参数

- **data_preprocessor** (*dict, optional*) –The pre-process config of BaseDataPreprocessor.
- **image_model** (*dict*) –Config for single image model
- **instance_model** (*dict*) –Config for instance model
- **fusion_model** (*dict*) –Config for fusion model
- **color_data_opt** (*dict*) –Option for colorspace conversion
- **which_direction** (*str*) –AtoB or BtoA
- **loss** (*dict*) –Config for loss.
- **init_cfg** (*str*) –Initialization config dict. Default: None.
- **train_cfg** (*dict*) –Config for training. Default: None.
- **test_cfg** (*dict*) –Config for testing. Default: None.

forward(*inputs: torch.Tensor*, *data_samples: Optional[List[*mmagic.structures.DataSample*]] = None*, *mode: str = 'tensor'*, ***kwargs*)

Returns losses or predictions of training, validation, testing, and simple inference process.

`forward` method of BaseModel is an abstract method, its subclasses must implement this method.

Accepts `inputs` and `data_samples` processed by `data_preprocessor`, and returns results according to mode arguments.

During non-distributed training, validation, and testing process, `forward` will be called by `BaseModel.train_step`, `BaseModel.val_step` and `BaseModel.val_step` directly.

During distributed data parallel training process, `MMSeparateDistributedDataParallel.train_step` will first call `DistributedDataParallel.forward` to enable automatic gradient synchronization, and then call `forward` to get training loss.

参数

- **inputs** (*torch.Tensor*) –batch input tensor collated by `data_preprocessor`.
- **data_samples** (*List[BaseDataElement], optional*) –data samples collated by `data_preprocessor`.
- **mode** (*str*) –mode should be one of `loss`, `predict` and `tensor`. Default: 'tensor'.
  - `loss`: Called by `train_step` and return loss `dict` used for logging
  - `predict`: Called by `val_step` and `test_step` and return list of `BaseDataElement` results used for computing metric.
  - `tensor`: Called by custom use to get `Tensor` type results.

返回

- If `mode == loss`, return a `dict` of loss tensor used for backward and logging.
- If `mode == predict`, return a `list` of `BaseDataElement` for computing metric and getting inference result.
- If `mode == tensor`, return a tensor or `tuple` of tensor or `dict` or tensor for custom use.

返回类型 ForwardResults

convert_to_datasample(*inputs*, *data_samples*)

Add predictions and destructed inputs (if passed) to data samples.

参数

- **inputs** (*Optional[torch.Tensor]*) –The input of model. Defaults to None.
- **data_samples** (*List[DataSample]*) –The data samples loaded from dataloader.

返回 Modified data samples.

返回类型 List[DataSample]

abstract forward_train(*inputs*, *data_samples=None*, ***kwargs*)

Forward function for training.

abstract train_step(*data: List[dict]*, *optim_wrapper: mmengine.optim.OptimWrapperDict*) → Dict[str, torch.Tensor]

Train step function.

参数

- **data** (*List[dict]*) –Batch of data as input.
- **optim_wrapper** (*dict[torch.optim.Optimizer]*) –Dict with optimizers for generator and discriminator (if have).

返回

Dict with loss, information for logger, the number of samples and results for visualization.

返回类型 dict

forward_inference(*inputs*, *data_samples=None*, ***kwargs*)

Forward inference. Returns predictions of validation, testing.

参数

- **inputs** (*torch.Tensor*) –batch input tensor collated by `data_preprocessor`.

- **data_samples** (*List[BaseDataElement], optional*) –data samples collated by data_preprocessor.

返回 predictions.

返回类型 List[DataSample]

forward_tensor(*inputs*, *data_samples*)

Forward function in tensor mode.

参数

- **inputs** (*torch.Tensor*) –Input tensor.
- **data_sample** (*dict*) –Dict contains data sample.

返回 Dict contains output results.

返回类型 dict

class mmagic.models.editors.**LIIF**(*generator: dict*, *pixel_loss: dict*, *train_cfg: Optional[dict] = None*, *test_cfg: Optional[dict] = None*, *init_cfg: Optional[dict] = None*, *data_preprocessor: Optional[dict] = None*)

Bases: *mmagic.models.base_models.BaseEditModel*

LIIF model for single image super-resolution.

Paper: Learning Continuous Image Representation with Local Implicit Image Function

参数

- **generator** (*dict*) –Config for the generator.
- **pixel_loss** (*dict*) –Config for the pixel loss.
- **pretrained** (*str*) –Path for pretrained model. Default: None.
- **data_preprocessor** (*dict, optional*) –The pre-process config of BaseDataPreprocessor.

forward_tensor(*inputs*, *data_samples=None*, ***kwargs*)

Forward tensor. Returns result of simple forward.

参数

- **inputs** (*torch.Tensor*) –batch input tensor collated by data_preprocessor.
- **data_samples** (*List[BaseDataElement], optional*) –data samples collated by data_preprocessor.

返回 result of simple forward.

返回类型 Tensor

forward_inference(*inputs*, *data_samples=None*, ***kwargs*)

Forward inference. Returns predictions of validation, testing, and simple inference.

参数

- **inputs** (*torch.Tensor*) –batch input tensor collated by data_preprocessor.
- **data_samples** (*BaseDataElement, optional*) –data samples collated by data_preprocessor.

返回 predictions.

返回类型 List[DataSample]

class mmagic.models.editors.**MLPRefiner**(*in_dim*, *out_dim*, *hidden_list*)

Bases: mmengine.model.BaseModule

Multilayer perceptrons (MLPs), refiner used in LIIF.

参数

- **in_dim** (*int*) –Input dimension.
- **out_dim** (*int*) –Output dimension.
- **hidden_list** (*list[int]*) –List of hidden dimensions.

forward(*x*)

Forward function.

参数 **x** (*Tensor*) –The input of MLP.

返回 The output of MLP.

返回类型 Tensor

class mmagic.models.editors.**LSGAN**(*generator: ModelType*, *discriminator: Optional[ModelType] = None*, *data_preprocessor: Optional[Union[dict, mmengine.Config]] = None*, *generator_steps: int = 1*, *discriminator_steps: int = 1*, *noise_size: Optional[int] = None*, *ema_config: Optional[Dict] = None*, *loss_config: Optional[Dict] = None*)

Bases: *mmagic.models.base_models.BaseGAN*

Implementation of *Least Squares Generative Adversarial Networks*.

Paper link: https://arxiv.org/pdf/1611.04076.pdf

Detailed architecture can be found in LSGANGenerator and LSGANDiscriminator

disc_loss(*disc_pred_fake: torch.Tensor*, *disc_pred_real: torch.Tensor*) → Tuple

Get disc loss. LSGAN use the least squares loss to train the discriminator.

$$L_D = \left(D\left(X_{\text{data}}\right) - 1\right)^2 + \left(D(G(z))\right)^2$$

参数

- **disc_pred_fake** (*Tensor*) –Discriminator's prediction of the fake images.
- **disc_pred_real** (*Tensor*) –Discriminator's prediction of the real images.

返回 Loss value and a dict of log variables.

返回类型 tuple[Tensor, dict]

gen_loss(*disc_pred_fake: torch.Tensor*) → Tuple

Get gen loss. LSGAN use the least squares loss to train the generator.

$$L_G = (D(G(z)) - 1)^2$$

参数 **disc_pred_fake** (*Tensor*) –Discriminator's prediction of the fake images.

返回 Loss value and a dict of log variables.

返回类型 tuple[Tensor, dict]

train_discriminator(*inputs: dict*, *data_samples:* mmagic.structures.DataSample, *optimizer_wrapper: mmengine.optim.OptimWrapper*) → Dict[str, torch.Tensor]

Train discriminator.

参数

- **inputs** (*dict*) –Inputs from dataloader.
- **data_samples** (*DataSample*) –Data samples from dataloader.
- **optim_wrapper** (*OptimWrapper*) –OptimWrapper instance used to update model parameters.

返回 A `dict` of tensor for logging.

返回类型 Dict[str, Tensor]

train_generator(*inputs: dict*, *data_samples: List[*mmagic.structures.DataSample*]*, *optimizer_wrapper: mmengine.optim.OptimWrapper*) → Dict[str, torch.Tensor]

Train generator.

参数

- **inputs** (*dict*) –Inputs from dataloader.
- **data_samples** (*List[DataSample]*) –Data samples from dataloader. Do not used in generator's training.
- **optim_wrapper** (*OptimWrapper*) –OptimWrapper instance used to update model parameters.

返回 A `dict` of tensor for logging.

返回类型 Dict[str, Tensor]

class mmagic.models.editors.**MSPIEStyleGAN2**(**args*, *train_settings=dict()*, ***kwargs*)

Bases: mmagic.models.editors.stylegan2.StyleGAN2

MS-PIE StyleGAN2.

In this GAN, we adopt the MS-PIE training schedule so that multi-scale images can be generated with a single generator. Details can be found in: Positional Encoding as Spatial Inductive Bias in GANs, CVPR2021.

参数 **train_settings** (*dict*) –Config for training settings. Defaults to *dict()*.

train_step(*data: dict*, *optim_wrapper: mmengine.optim.OptimWrapperDict*) → Dict[str, torch.Tensor]

Train GAN model. In the training of GAN models, generator and discriminator are updated alternatively. In MMagic's design, *self.train_step* is called with data input. Therefore we always update discriminator, whose updating is relay on real data, and then determine if the generator needs to be updated based on the current number of iterations. More details about whether to update generator can be found in should_gen_update().

参数

- **data** (*dict*) –Data sampled from dataloader.
- **optim_wrapper** (*OptimWrapperDict*) –OptimWrapperDict instance contains OptimWrapper of generator and discriminator.

返回 A dict of tensor for logging.

返回类型 Dict[str, torch.Tensor]

train_generator(*inputs: dict*, *data_samples:* mmagic.structures.DataSample, *optimizer_wrapper: mmengine.optim.OptimWrapper*) → Dict[str, torch.Tensor]

Train generator.

参数

- **inputs** (*TrainInput*) –Inputs from dataloader.
- **data_samples** (*DataSample*) –Data samples from dataloader. Do not used in generator's training.
- **optim_wrapper** (*OptimWrapper*) –OptimWrapper instance used to update model parameters.

返回 A dict of tensor for logging.

返回类型 Dict[str, Tensor]

train_discriminator(*inputs: dict*, *data_samples:* mmagic.structures.DataSample, *optimizer_wrapper: mmengine.optim.OptimWrapper*) → Dict[str, torch.Tensor]

Train discriminator.

参数

- **inputs** (*TrainInput*) –Inputs from dataloader.
- **data_samples** (*DataSample*) –Data samples from dataloader.
- **optim_wrapper** (*OptimWrapper*) –OptimWrapper instance used to update model parameters.

返回 A dict of tensor for logging.

返回类型 Dict[str, Tensor]

class mmagic.models.editors.**PESinGAN**(*generator: ModelType*, *discriminator: Optional[ModelType]*, *data_preprocessor: Optional[Union[dict, mmengine.Config]] = None*, *generator_steps: int = 1*, *discriminator_steps: int = 1*, *num_scales: Optional[int] = None*, *fixed_noise_with_pad: bool = False*, *first_fixed_noises_ch: int = 1*, *iters_per_scale: int = 200*, *noise_weight_init: int = 0.1*, *lr_scheduler_args: Optional[dict] = None*, *test_pkl_data: Optional[str] = None*, *ema_confg: Optional[dict] = None*)

Bases: mmagic.models.editors.singan.SinGAN

Positional Encoding in SinGAN.

This modified SinGAN is used to reimplement the experiments in: Positional Encoding as Spatial Inductive Bias in GANs, CVPR2021.

construct_fixed_noises()

Construct the fixed noises list used in SinGAN.

class mmagic.models.editors.**NAFBaseline**(*img_channel=3*, *mid_channels=16*, *middle_blk_num=1*, *enc_blk_nums=[1, 1, 1, 28]*, *dec_blk_nums=[1, 1, 1, 1]*, *dw_expand=1*, *ffn_expand=2*)

Bases: mmengine.model.BaseModule

The original version of Baseline model in “Simple Baseline for Image Restoration”.

参数

- **img_channels** (*int*) –Channel number of inputs.
- **mid_channels** (*int*) –Channel number of intermediate features.
- **middle_blk_num** (*int*) –Number of middle blocks.
- **enc_blk_nums** (*List of int*) –Number of blocks for each encoder.
- **dec_blk_nums** (*List of int*) –Number of blocks for each decoder.

forward(*inp*)

Forward function.

参数 **inp** –input tensor image with (B, C, H, W) shape

check_image_size(*x*)

Check image size and pad images so that it has enough dimension do downsample.

参数 **x** –input tensor image with (B, C, H, W) shape.

class mmagic.models.editors.**NAFBaselineLocal**(**args*, *train_size=(1, 3, 256, 256)*, *fast_imp=False*, ***kwargs*)

Bases: mmagic.models.editors.nafnet.naf_avgpool2d.Local_Base, *NAFBaseline*

The original version of Baseline model in "Simple Baseline for Image Restoration".

参数

- **img_channels** (*int*) –Channel number of inputs.
- **mid_channels** (*int*) –Channel number of intermediate features.
- **middle_blk_num** (*int*) –Number of middle blocks.
- **enc_blk_nums** (*List of int*) –Number of blocks for each encoder.
- **dec_blk_nums** (*L`ist of int*) –Number of blocks for each decoder.

class mmagic.models.editors.**NAFNet**(*img_channels=3*, *mid_channels=16*, *middle_blk_num=1*, *enc_blk_nums=[]*, *dec_blk_nums=[]*)

Bases: mmengine.model.BaseModule

NAFNet.

The original version of NAFNet in "Simple Baseline for Image Restoration".

参数

- **img_channels** (*int*) –Channel number of inputs.
- **mid_channels** (*int*) –Channel number of intermediate features.
- **middle_blk_num** (*int*) –Number of middle blocks.
- **enc_blk_nums** (*List of int*) –Number of blocks for each encoder.
- **dec_blk_nums** (*List of int*) –Number of blocks for each decoder.

forward(*inp*)

Forward function.

参数 **inp** –input tensor image with (B, C, H, W) shape

check_image_size(*x*)

Check image size and pad images so that it has enough dimension do downsample.

参数 **x** –input tensor image with (B, C, H, W) shape.

class mmagic.models.editors.**NAFNetLocal**(**args*, *train_size=(1, 3, 256, 256)*, *fast_imp=False*, ***kwargs*)

Bases: mmagic.models.editors.nafnet.naf_avgpool2d.Local_Base, *NAFNet*

The original version of NAFNetLocal in "Simple Baseline for Image Restoration".

NAFNetLocal uses local average pooling modules than NAFNet.

参数

- **img_channels** (*int*) –Channel number of inputs.
- **mid_channels** (*int*) –Channel number of intermediate features.
- **middle_blk_num** (*int*) –Number of middle blocks.
- **enc_blk_nums** (*List of int*) –Number of blocks for each encoder.
- **dec_blk_nums** (*List of int*) –Number of blocks for each decoder.

class mmagic.models.editors.**MaskConvModule**(**args*, ***kwargs*)

Bases: mmcv.cnn.ConvModule

Mask convolution module.

This is a simple wrapper for mask convolution like: 'partial conv'. Convolutions in this module always need a mask as extra input.

参数

- **in_channels** (*int*) –Same as nn.Conv2d.
- **out_channels** (*int*) –Same as nn.Conv2d.
- **kernel_size** (*int or tuple[int]*) –Same as nn.Conv2d.
- **stride** (*int or tuple[int]*) –Same as nn.Conv2d.
- **padding** (*int or tuple[int]*) –Same as nn.Conv2d.
- **dilation** (*int or tuple[int]*) –Same as nn.Conv2d.
- **groups** (*int*) –Same as nn.Conv2d.
- **bias** (*bool or str*) –If specified as *auto*, it will be decided by the norm_cfg. Bias will be set as True if norm_cfg is None, otherwise False.
- **conv_cfg** (*dict*) –Config dict for convolution layer.
- **norm_cfg** (*dict*) –Config dict for normalization layer.
- **act_cfg** (*dict*) –Config dict for activation layer, "relu" by default.
- **inplace** (*bool*) –Whether to use inplace mode for activation.
- **with_spectral_norm** (*bool*) –Whether use spectral norm in conv module.

- **padding_mode** (*str*) –If the *padding_mode* has not been supported by current *Conv2d* in Pytorch, we will use our own padding layer instead. Currently, we support [‘zeros’, ‘circular’] with official implementation and [‘reflect’] with our own implementation. Default: ‘zeros’.
- **order** (*tuple[str]*) –The order of conv/norm/activation layers. It is a sequence of “conv”, “norm” and “act”. Examples are (“conv”, “norm”, “act”) and (“act”, “conv”, “norm”).

supported_conv_list = ['PConv']

forward(*x*, *mask=None*, *activate=True*, *norm=True*, *return_mask=True*)

Forward function for partial conv2d.

参数

- **x** (*torch.Tensor*) –Tensor with shape of (n, c, h, w).
- **mask** (*torch.Tensor*) –Tensor with shape of (n, c, h, w) or (n, 1, h, w). If mask is not given, the function will work as standard conv2d. Default: None.
- **activate** (*bool*) –Whether use activation layer.
- **norm** (*bool*) –Whether use norm layer.
- **return_mask** (*bool*) –If True and mask is not None, the updated mask will be returned. Default: True.

返回

Result Tensor or 2-tuple of

Tensor: Results after partial conv.

Tensor: Updated mask will be returned if mask is given and *return_mask* is True.

返回类型 Tensor or tuple

class mmagic.models.editors.**PartialConv2d**(**args*, *multi_channel=False*, *eps=1e-08*, ***kwargs*)

Bases: torch.nn.Conv2d

Implementation for partial convolution.

Image Inpainting for Irregular Holes Using Partial Convolutions [https://arxiv.org/abs/1804.07723]

参数

- **multi_channel** (*bool*) –If True, the mask is multi-channel. Otherwise, the mask is single-channel.
- **eps** (*float*) –Need to be changed for mixed precision training. For mixed precision training, you need change 1e-8 to 1e-6.

forward(*input*, *mask=None*, *return_mask=True*)

Forward function for partial conv2d.

参数

- **input** (*torch.Tensor*) –Tensor with shape of (n, c, h, w).
- **mask** (*torch.Tensor*) –Tensor with shape of (n, c, h, w) or (n, 1, h, w). If mask is not given, the function will work as standard conv2d. Default: None.
- **return_mask** (*bool*) –If True and mask is not None, the updated mask will be returned. Default: True.

返回 Results after partial conv. torch.Tensor : Updated mask will be returned if mask is given and `return_mask` is True.

返回类型 torch.Tensor

class mmagic.models.editors.**PConvDecoder**(*num_layers=7*, *interpolation='nearest'*, *conv_cfg=dict(type='PConv', multi_channel=True)*, *norm_cfg=dict(type='BN')*)

Bases: `mmengine.model.BaseModule`

Decoder with partial conv.

About the details for this architecture, pls see: Image Inpainting for Irregular Holes Using Partial Convolutions

参数

- **num_layers** (*int*) –The number of convolutional layers. Default: 7.
- **interpolation** (*str*) –The upsample mode. Default: 'nearest'.
- **conv_cfg** (*dict*) –Config for convolution module. Default: {'type': 'PConv', 'multi_channel': True}.
- **norm_cfg** (*dict*) –Config for norm layer. Default: {'type': 'BN'}.

forward(*input_dict*)

Forward Function.

参数 **input_dict** (*dict | torch.Tensor*) –Input dict with middle features or torch.Tensor.

返回 Output tensor with shape of (n, c, h, w).

返回类型 torch.Tensor

class mmagic.models.editors.**PConvEncoder**(*in_channels=3*, *num_layers=7*, *conv_cfg=dict(type='PConv', multi_channel=True)*, *norm_cfg=dict(type='BN', requires_grad=True)*, *norm_eval=False*)

Bases: mmengine.model.BaseModule

Encoder with partial conv.

About the details for this architecture, pls see: Image Inpainting for Irregular Holes Using Partial Convolutions

参数

- **in_channels** (*int*) –The number of input channels. Default: 3.
- **num_layers** (*int*) –The number of convolutional layers. Default: 7.
- **conv_cfg** (*dict*) –Config for convolution module. Default: {‘type’: ‘PConv’, ‘multi_channel’: True}.
- **norm_cfg** (*dict*) –Config for norm layer. Default: {‘type’: ‘BN’}.
- **norm_eval** (*bool*) –Whether to set norm layers to eval mode, namely, freeze running stats (mean and var). Note: Effective on Batch Norm and its variants only. Default: False.

train(*mode=True*)

Set BatchNorm modules in the model to evaluation mode.

forward(*x*, *mask*)

Forward function for partial conv encoder.

参数

- **x** (*torch.Tensor*) –Masked image with shape (n, c, h, w).
- **mask** (*torch.Tensor*) –Mask tensor with shape (n, c, h, w).

返回 Contains the results and middle level features in this module. *hidden_feats* contain the middle feature maps and *hidden_masks* store updated masks.

返回类型 dict

class mmagic.models.editors.**PConvEncoderDecoder**(*encoder*, *decoder*)

Bases: mmengine.model.BaseModule

Encoder-Decoder with partial conv module.

参数

- **encoder** (*dict*) –Config of the encoder.
- **decoder** (*dict*) –Config of the decoder.

forward(*x*, *mask_in*)

Forward Function.

参数

- **x** (*torch.Tensor*) –Input tensor with shape of (n, c, h, w).

- **mask_in** (*torch.Tensor*) –Input tensor with shape of (n, c, h, w).

返回 Output tensor with shape of (n, c, h', w').

返回类型 torch.Tensor

class mmagic.models.editors.**PConvInpaintor**(*data_preprocessor: Union[dict, mmengine.config.Config]*, *encdec: dict*, *disc: Optional[dict] = None*, *loss_gan: Optional[dict] = None*, *loss_gp: Optional[dict] = None*, *loss_disc_shift: Optional[dict] = None*, *loss_composed_percep: Optional[dict] = None*, *loss_out_percep: bool = False*, *loss_l1_hole: Optional[dict] = None*, *loss_l1_valid: Optional[dict] = None*, *loss_tv: Optional[dict] = None*, *train_cfg: Optional[dict] = None*, *test_cfg: Optional[dict] = None*, *init_cfg: Optional[dict] = None*)

Bases: *mmagic.models.base_models.OneStageInpaintor*

Inpaintor for Partial Convolution method.

This inpaintor is implemented according to the paper: Image inpainting for irregular holes using partial convolutions

forward_tensor(*inputs*, *data_samples*)

Forward function in tensor mode.

参数

- **inputs** (*torch.Tensor*) –Input tensor.
- **data_sample** (*dict*) –Dict contains data sample.

返回 Dict contains output results.

返回类型 dict

train_step(*data: List[dict]*, *optim_wrapper*)

Train step function.

In this function, the inpaintor will finish the train step following the pipeline:

1. get fake res/image
2. optimize discriminator (if have)
3. optimize generator

If *self.train_cfg.disc_step > 1*, the train step will contain multiple iterations for optimizing discriminator with different input data and only one iteration for optimizing generator after *disc_step* iterations for discriminator.

参数

- **data** (*List[dict]*) –Batch of data as input.
- **optim_wrapper** (*dict[torch.optim.Optimizer]*) –Dict with optimizers for generator and discriminator (if have).

返回 Dict with loss, information for logger, the number of samples and results for visualization.

返回类型 dict

class mmagic.models.editors.**ProgressiveGrowingGAN**(*generator*, *discriminator*, *data_preprocessor*, *nkimgs_per_scale*, *noise_size=None*, *interp_real=None*, *transition_kimgs: int = 600*, *prev_stage: int = 0*, *ema_config: Optional[Dict] = None*)

Bases: *mmagic.models.base_models.BaseGAN*

Progressive Growing Unconditional GAN.

In this GAN model, we implement progressive growing training schedule, which is proposed in Progressive Growing of GANs for improved Quality, Stability and Variation, ICLR 2018.

We highly recommend to use `GrowScaleImgDataset` for saving computational load in data pre-processing.

Notes for **using PGGAN**:

1. In official implementation, Tero uses gradient penalty with `norm_mode="HWC"`
2. We do not implement `minibatch_repeats` where has been used in official Tensorflow implementation.

Notes for resuming progressive growing GANs: Users should specify the `prev_stage` in `train_cfg`. Otherwise, the model is possible to reset the optimizer status, which will bring inferior performance. For example, if your model is resumed from the *256* stage, you should set `train_cfg=dict(prev_stage=256)`.

参数

- **generator** (*dict*) –Config for generator.
- **discriminator** (*dict*) –Config for discriminator.

forward(*inputs: mmagic.utils.typing.ForwardInputs*, *data_samples: Optional[list] = None*, *mode: Optional[str] = None*) → mmagic.utils.typing.SampleList

Sample images from noises by using the generator.

参数

- **batch_inputs** (*ForwardInputs*) –Dict containing the necessary information (e.g. noise, num_batches, mode) to generate image.
- **data_samples** (*Optional[list]*) –Data samples collated by `data_preprocessor`. Defaults to None.
- **mode** (*Optional[str]*) –*mode* is not used in *ProgressiveGrowingGAN*. Defaults to None.

返回 A list of `DataSample` contain generated results.

返回类型 SampleList

train_discriminator(*inputs: torch.Tensor*, *data_samples: List[*mmagic.structures.DataSample*]*, *optimizer_wrapper: mmengine.optim.OptimWrapper*) → Dict[str, torch.Tensor]

Train discriminator.

参数

- **inputs** (*Tensor*) –Inputs from current resolution training.
- **data_samples** (*List[DataSample]*) –Data samples from dataloader. Do not used in generator's training.
- **optim_wrapper** (*OptimWrapper*) –OptimWrapper instance used to update model parameters.

返回 A `dict` of tensor for logging.

返回类型 Dict[str, Tensor]

disc_loss(*disc_pred_fake: torch.Tensor*, *disc_pred_real: torch.Tensor*, *fake_data: torch.Tensor*, *real_data: torch.Tensor*) → Tuple[torch.Tensor, dict]

Get disc loss. PGGAN use WGAN-GP's loss and discriminator shift loss to train the discriminator.

参数

- **disc_pred_fake** (*Tensor*) –Discriminator's prediction of the fake images.
- **disc_pred_real** (*Tensor*) –Discriminator's prediction of the real images.
- **fake_data** (*Tensor*) –Generated images, used to calculate gradient penalty.
- **real_data** (*Tensor*) –Real images, used to calculate gradient penalty.

返回 Loss value and a dict of log variables.

返回类型 Tuple[Tensor, dict]

train_generator(*inputs: torch.Tensor*, *data_samples: List[*mmagic.structures.DataSample*]*, *optimizer_wrapper: mmengine.optim.OptimWrapper*) → Dict[str, torch.Tensor]

Train generator.

参数

- **inputs** (*Tensor*) –Inputs from current resolution training.
- **data_samples** (*List[DataSample]*) –Data samples from dataloader. Do not used in generator's training.
- **optim_wrapper** (*OptimWrapper*) –OptimWrapper instance used to update model parameters.

返回 A `dict` of tensor for logging.

返回类型 Dict[str, Tensor]

gen_loss(*disc_pred_fake: torch.Tensor*) → Tuple[torch.Tensor, dict]

Generator loss for PGGAN. PGGAN use WGAN's loss to train the generator.

参数

- **disc_pred_fake** (*Tensor*) –Discriminator's prediction of the fake images.
- **recon_imgs** (*Tensor*) –Reconstructive images.

返回 Loss value and a dict of log variables.

返回类型 Tuple[Tensor, dict]

train_step(*data: dict*, *optim_wrapper: mmengine.optim.OptimWrapperDict*)

Train step function.

This function implements the standard training iteration for asynchronous adversarial training. Namely, in each iteration, we first update discriminator and then compute loss for generator with the newly updated discriminator.

As for distributed training, we use the `reducer` from ddp to synchronize the necessary params in current computational graph.

参数

- **data_batch** (*dict*) –Input data from dataloader.
- **optimizer** (*dict*) –Dict contains optimizer for generator and discriminator.
- **ddp_reducer** (`Reducer` | None, optional) –Reducer from ddp. It is used to prepare for `backward()` in ddp. Defaults to None.
- **running_status** (*dict | None, optional*) –Contains necessary basic information for training, e.g., iteration number. Defaults to None.

返回 Contains 'log_vars', 'num_samples', and 'results'.

返回类型 dict

class mmagic.models.editors.**Pix2Pix**(**args*, ***kwargs*)

Bases: *mmagic.models.base_models.BaseTranslationModel*

Pix2Pix model for paired image-to-image translation.

Ref: Image-to-Image Translation with Conditional Adversarial Networks

forward_test(*img*, *target_domain*, ***kwargs*)

Forward function for testing.

参数

- **img** (*tensor*) –Input image tensor.
- **target_domain** (*str*) –Target domain of output image.
- **kwargs** (*dict*) –Other arguments.

返回 Forward results.

返回类型 dict

_get_disc_loss(*outputs*)

Get the loss of discriminator.

参数 **outputs** (*dict*) –A dict of output.

返回 Loss and a dict of log of loss terms.

返回类型 Tuple

_get_gen_loss(*outputs*)

Get the loss of generator.

参数 **outputs** (*dict*) –A dict of output.

返回 Loss and a dict of log of loss terms.

返回类型 Tuple

train_step(*data*, *optim_wrapper=None*)

Training step function.

参数

- **data_batch** (*dict*) –Dict of the input data batch.
- **optimizer** (*dict[torch.optim.Optimizer]*) –Dict of optimizers for the generator and discriminator.
- **ddp_reducer** (Reducer | None, optional) –Reducer from ddp. It is used to prepare for `backward()` in ddp. Defaults to None.
- **running_status** (*dict | None, optional*) –Contains necessary basic information for training, e.g., iteration number. Defaults to None.

返回 Dict of loss, information for logger, the number of samples and results for visualization.

返回类型 dict

test_step(*data: dict*) → mmagic.utils.typing.SampleList

Gets the generated image of given data. Same as *val_step()*.

参数 **data** (*dict*) –Data sampled from metric specific sampler. More details in *Metrics* and *Evaluator*.

返回 Generated image or image dict.

返回类型 List[DataSample]

val_step(*data: dict*) → mmagic.utils.typing.SampleList

Gets the generated image of given data. Same as `val_step()`.

参数 **data** (*dict*) –Data sampled from metric specific sampler. More details in *Metrics* and *Evaluator*.

返回 Generated image or image dict.

返回类型 List[DataSample]

class mmagic.models.editors.**PlainDecoder**(*in_channels*, *init_cfg: Optional[dict] = None*)

Bases: mmengine.model.BaseModule

Simple decoder from Deep Image Matting.

参数

- **in_channels** (*int*) –Channel num of input features.
- **init_cfg** (*dict, optional*) –Initialization config dict. Default: None.

init_weights()

Init weights for the module.

forward(*inputs*)

Forward function of PlainDecoder.

参数 **inputs** (*dict*) –Output dictionary of the VGG encoder containing:

- out (Tensor): Output of the VGG encoder.
- max_idx_1 (Tensor): Index of the first maxpooling layer in the VGG encoder.
- max_idx_2 (Tensor): Index of the second maxpooling layer in the VGG encoder.
- max_idx_3 (Tensor): Index of the third maxpooling layer in the VGG encoder.
- max_idx_4 (Tensor): Index of the fourth maxpooling layer in the VGG encoder.
- max_idx_5 (Tensor): Index of the fifth maxpooling layer in the VGG encoder.

返回 Output tensor.

返回类型 Tensor

class mmagic.models.editors.**PlainRefiner**(*conv_channels=64*, *init_cfg=None*)

Bases: mmengine.model.BaseModule

Simple refiner from Deep Image Matting.

参数

- **conv_channels** (*int*) –Number of channels produced by the three main convolutional layer. Default: 64.

- **pretrained** (*str*) –Name of pretrained model. Default: None.

init_weights()

Init weights for the module.

forward(*x*, *raw_alpha*)

Forward function.

参数

- **x** (*Tensor*) –The input feature map of refiner.
- **raw_alpha** (*Tensor*) –The raw predicted alpha matte.

返回 The refined alpha matte.

返回类型 Tensor

class mmagic.models.editors.**RDNNet**(*in_channels*, *out_channels*, *mid_channels=64*, *num_blocks=16*, *upscale_factor=4*, *num_layers=8*, *channel_growth=64*)

Bases: mmengine.model.BaseModule

RDN model for single image super-resolution.

Paper: Residual Dense Network for Image Super-Resolution

Adapted from 'https://github.com/yjn870/RDN-pytorch.git' 'RDN-pytorch/blob/master/models.py' Copyright (c) 2021, JaeYun Yeo, under MIT License.

Most of the implementation follows the implementation in: 'https://github.com/sanghyun-son/EDSR-PyTorch.git' 'EDSR-PyTorch/blob/master/src/model/rdn.py' Copyright (c) 2017, sanghyun-son, under MIT license.

参数

- **in_channels** (*int*) –Channel number of inputs.
- **out_channels** (*int*) –Channel number of outputs.
- **mid_channels** (*int*) –Channel number of intermediate features. Default: 64.
- **num_blocks** (*int*) –Block number in the trunk network. Default: 16.
- **upscale_factor** (*int*) –Upsampling factor. Support 2^n and 3. Default: 4.
- **num_layer** (*int*) –Layer number in the Residual Dense Block. Default: 8.
- **channel_growth** (*int*) –Channels growth in each layer of RDB. Default: 64.

forward(*x*)

Forward function.

参数 **x** (*Tensor*) –Input tensor with shape (n, c, h, w).

返回 Forward results.

返回类型 Tensor

class mmagic.models.editors.**RealBasicVSR**(*generator*, *discriminator=None*, *gan_loss=None*, *pixel_loss=None*, *cleaning_loss=None*, *perceptual_loss=None*, *is_use_sharpened_gt_in_pixel=False*, *is_use_sharpened_gt_in_percep=False*, *is_use_sharpened_gt_in_gan=False*, *is_use_ema=False*, *train_cfg=None*, *test_cfg=None*, *init_cfg=None*, *data_preprocessor=None*)

Bases: mmagic.models.editors.real_esrgan.RealESRGAN

RealBasicVSR model for real-world video super-resolution.

Ref: Investigating Tradeoffs in Real-World Video Super-Resolution, arXiv

参数

- **generator** (*dict*) –Config for the generator.
- **discriminator** (*dict, optional*) –Config for the discriminator. Default: None.
- **gan_loss** (*dict, optional*) –Config for the gan loss. Note that the loss weight in gan loss is only for the generator.
- **pixel_loss** (*dict, optional*) –Config for the pixel loss. Default: None.
- **cleaning_loss** (*dict, optional*) –Config for the image cleaning loss. Default: None.
- **perceptual_loss** (*dict, optional*) –Config for the perceptual loss. Default: None.
- **is_use_sharpened_gt_in_pixel** (*bool, optional*) –Whether to use the image sharpened by unsharp masking as the GT for pixel loss. Default: False.
- **is_use_sharpened_gt_in_percep** (*bool, optional*) –Whether to use the image sharpened by unsharp masking as the GT for perceptual loss. Default: False.
- **is_use_sharpened_gt_in_gan** (*bool, optional*) –Whether to use the image sharpened by unsharp masking as the GT for adversarial loss. Default: False.
- **train_cfg** (*dict*) –Config for training. Default: None. You may change the training of gan by setting: *disc_steps*: how many discriminator updates after one generate update; *disc_init_steps*: how many discriminator updates at the start of the training. These two keys are useful when training with WGAN.
- **test_cfg** (*dict*) –Config for testing. Default: None.
- **init_cfg** (*dict, optional*) –The weight initialized config for BaseModule. Default: None.

- **data_preprocessor** (*dict, optional*) –The pre-process config of `BaseDataPreprocessor`. Default: None.

extract_gt_data(*data_samples*)

extract gt data from data samples.

参数 **data_samples** (*list*) –List of DataSample.

返回 Extract gt data.

返回类型 Tensor

g_step(*batch_outputs*, *batch_gt_data*)

G step of GAN: Calculate losses of generator.

参数

- **batch_outputs** (*Tensor*) –Batch output of generator.
- **batch_gt_data** (*Tuple[Tensor]*) –Batch GT data.

返回 Dict of losses.

返回类型 dict

train_step(*data: List[dict]*, *optim_wrapper: mmengine.optim.OptimWrapperDict*) → Dict[str, torch.Tensor]

Train step of GAN-based method.

参数

- **data** (*List[dict]*) –Data sampled from dataloader.
- **optim_wrapper** (*OptimWrapper*) –OptimWrapper instance used to update model parameters.

返回 A `dict` of tensor for logging.

返回类型 Dict[str, torch.Tensor]

forward_train(*batch_inputs*, *data_samples=None*)

Forward Train.

Run forward of generator with `return_lqs=True`

参数

- **batch_inputs** (*Tensor*) –Batch inputs.
- **data_samples** (*List[DataSample]*) –Data samples of Editing. Default:None

返回

Result of generator. (outputs, lqs)

返回类型 Tuple[Tensor]

class mmagic.models.editors.**RealBasicVSRNet**(*mid_channels=64*, *num_propagation_blocks=20*, *num_cleaning_blocks=20*, *dynamic_refine_thres=255*, *spynet_pretrained=None*, *is_fix_cleaning=False*, *is_sequential_cleaning=False*)

Bases: mmengine.model.BaseModule

RealBasicVSR network structure for real-world video super-resolution.

Support only x4 upsampling.

Paper: Investigating Tradeoffs in Real-World Video Super-Resolution, arXiv

参数

- **mid_channels** (*int, optional*) –Channel number of the intermediate features. Default: 64.
- **num_propagation_blocks** (*int, optional*) –Number of residual blocks in each propagation branch. Default: 20.
- **num_cleaning_blocks** (*int, optional*) –Number of residual blocks in the image cleaning module. Default: 20.
- **dynamic_refine_thres** (*int, optional*) –Stop cleaning the images when the residue is smaller than this value. Default: 255.
- **spynet_pretrained** (*str, optional*) –Pre-trained model path of SPyNet. Default: None.
- **is_fix_cleaning** (*bool, optional*) –Whether to fix the weights of the image cleaning module during training. Default: False.
- **is_sequential_cleaning** (*bool, optional*) –Whether to clean the images sequentially. This is used to save GPU memory, but the speed is slightly slower. Default: False.

forward(*lqs*, *return_lqs=False*)

Forward function for BasicVSR++.

参数

- **lqs** (*tensor*) –Input low quality (LQ) sequence with shape (n, t, c, h, w).
- **return_lqs** (*bool*) –Whether to return LQ sequence. Default: False.

返回 Output HR sequence.

返回类型 Tensor

class mmagic.models.editors.**RealESRGAN**(*generator*, *discriminator=None*, *gan_loss=None*, *pixel_loss=None*, *perceptual_loss=None*, *is_use_sharpened_gt_in_pixel=False*, *is_use_sharpened_gt_in_percep=False*, *is_use_sharpened_gt_in_gan=False*, *is_use_ema=True*, *train_cfg=None*, *test_cfg=None*, *init_cfg=None*, *data_preprocessor=None*)

Bases: mmagic.models.editors.srgan.SRGAN

Real-ESRGAN model for single image super-resolution.

Ref: Real-ESRGAN: Training Real-World Blind Super-Resolution with Pure Synthetic Data, 2021.

Note: generator_ema is realized in EMA_HOOK

参数

- **generator** (*dict*) –Config for the generator.
- **discriminator** (*dict, optional*) –Config for the discriminator. Default: None.
- **gan_loss** (*dict, optional*) –Config for the gan loss. Note that the loss weight in gan loss is only for the generator.
- **pixel_loss** (*dict, optional*) –Config for the pixel loss. Default: None.
- **perceptual_loss** (*dict, optional*) –Config for the perceptual loss. Default: None.
- **is_use_sharpened_gt_in_pixel** (*bool, optional*) –Whether to use the image sharpened by unsharp masking as the GT for pixel loss. Default: False.
- **is_use_sharpened_gt_in_percep** (*bool, optional*) –Whether to use the image sharpened by unsharp masking as the GT for perceptual loss. Default: False.
- **is_use_sharpened_gt_in_gan** (*bool, optional*) –Whether to use the image sharpened by unsharp masking as the GT for adversarial loss. Default: False.
- **is_use_ema** (*bool, optional*) –When to apply exponential moving average on the network weights. Default: True.
- **train_cfg** (*dict*) –Config for training. Default: None. You may change the training of gan by setting: *disc_steps*: how many discriminator updates after one generate update; *disc_init_steps*: how many discriminator updates at the start of the training. These two keys are useful when training with WGAN.
- **test_cfg** (*dict*) –Config for testing. Default: None.
- **init_cfg** (*dict, optional*) –The weight initialized config for BaseModule. Default: None.

- **data_preprocessor** (*dict, optional*) –The pre-process config of `BaseDataPreprocessor`. Default: None.

forward_tensor(*inputs*, *data_samples=None*, *training=False*)

Forward tensor. Returns result of simple forward.

参数

- **inputs** (*torch.Tensor*) –batch input tensor collated by `data_preprocessor`.
- **data_samples** (*List[BaseDataElement], optional*) –data samples collated by `data_preprocessor`.
- **training** (*bool*) –Whether is training. Default: False.

返回 result of simple forward.

返回类型 Tensor

g_step(*batch_outputs*, *batch_gt_data*)

G step of GAN: Calculate losses of generator.

参数

- **batch_outputs** (*Tensor*) –Batch output of generator.
- **batch_gt_data** (*Tuple[Tensor]*) –Batch GT data.

返回 Dict of losses.

返回类型 dict

d_step_real(*batch_outputs*, *batch_gt_data: torch.Tensor*)

Real part of D step.

参数

- **batch_outputs** (*Tensor*) –Batch output of generator.
- **batch_gt_data** (*Tuple[Tensor]*) –Batch GT data.

返回 Real part of gan_loss for discriminator.

返回类型 Tensor

d_step_fake(*batch_outputs*, *batch_gt_data*)

Fake part of D step.

参数

- **batch_outputs** (*Tensor*) –Output of generator.
- **batch_gt_data** (*Tuple[Tensor]*) –Batch GT data.

返回 Fake part of gan_loss for discriminator.

返回类型 Tensor

extract_gt_data(*data_samples*)

extract gt data from data samples.

参数 **data_samples** (*list*) –List of DataSample.

返回 Extract gt data.

返回类型 Tensor

class mmagic.models.editors.**UNetDiscriminatorWithSpectralNorm**(*in_channels*, *mid_channels=64*, *skip_connection=True*)

Bases: mmengine.model.BaseModule

A U-Net discriminator with spectral normalization.

参数

- **in_channels** (*int*) –Channel number of the input.
- **mid_channels** (*int, optional*) –Channel number of the intermediate features. Default: 64.
- **skip_connection** (*bool, optional*) –Whether to use skip connection. Default: True.

forward(*img*)

Forward function.

参数 **img** (*Tensor*) –Input tensor with shape (n, c, h, w).

返回 Forward results.

返回类型 Tensor

class mmagic.models.editors.**Restormer**(*inp_channels=3*, *out_channels=3*, *dim=48*, *num_blocks=[4, 6, 6, 8]*, *num_refinement_blocks=4*, *heads=[1, 2, 4, 8]*, *ffn_expansion_factor=2.66*, *bias=False*, *LayerNorm_type='WithBias'*, *dual_pixel_task=False*, *dual_keys=['imgL', 'imgR']*)

Bases: mmengine.model.BaseModule

Restormer A PyTorch impl of: *Restormer: Efficient Transformer for High- Resolution Image Restoration*. Ref repo: https://github.com/swz30/Restormer.

参数

- **inp_channels** (*int*) –Number of input image channels. Default: 3.
- **out_channels** (*int*) –Number of output image channels: 3.

- **dim** (*int*) –Number of feature dimension. Default: 48.
- **num_blocks** (*List (int)*) –Depth of each Transformer layer. Default: [4, 6, 6, 8].
- **num_refinement_blocks** (*int*) –Number of refinement blocks. Default: 4.
- **heads** (*List (int)*) –Number of attention heads in different layers. Default: 7.
- **ffn_expansion_factor** (*float*) –Ratio of feed forward network expansion. Default: 2.66.
- **bias** (*bool*) –The bias of convolution. Default: False
- **LayerNorm_type** (*str|optional*) –Select layer Normalization type. Optional: 'WithBias','BiasFree' Default: 'WithBias'.
- **dual_pixel_task** (*bool*) –True for dual-pixel defocus deblurring only. Also set inp_channels=6. Default: False.
- **dual_keys** (*List*) –Keys of dual images in inputs. Default: ['imgL', 'imgR'].

forward(*inp_img*)

Forward function.

参数 **inp_img** (*Tensor*) –Input tensor with shape (B, C, H, W).

返回 Forward results.

返回类型 Tensor

class mmagic.models.editors.**SAGAN**(*generator: ModelType*, *discriminator: Optional[ModelType] = None*, *data_preprocessor: Optional[Union[dict, mmengine.Config]] = None*, *generator_steps: int = 1*, *discriminator_steps: int = 1*, *noise_size: Optional[int] = 128*, *num_classes: Optional[int] = None*, *ema_config: Optional[Dict] = None*)

Bases: *mmagic.models.base_models.BaseConditionalGAN*

Implementation of *Self-Attention Generative Adversarial Networks*.

<https://arxiv.org/abs/1805.08318>'_ (SAGAN), Spectral Normalization for Generative Adversarial Networks (SNGAN), and cGANs with Projection Discriminator (Proj-GAN).

Detailed architecture can be found in `SNGANGenerator` and `ProjDiscriminator`

参数

- **generator** (*ModelType*) –The config or model of the generator.
- **discriminator** (*Optional[ModelType]*) –The config or model of the discriminator. Defaults to None.
- **data_preprocessor** (*Optional[Union[dict, Config]]*) –The pre-process config or `DataPreprocessor`.

- **generator_steps** (*int*) –Number of times the generator was completely updated before the discriminator is updated. Defaults to 1.
- **discriminator_steps** (*int*) –Number of times the discriminator was completely updated before the generator is updated. Defaults to 1.
- **noise_size** (*Optional[int]*) –Size of the input noise vector. Default to 128.
- **num_classes** (*Optional[int]*) –The number classes you would like to generate. Defaults to None.
- **ema_config** (*Optional[Dict]*) –The config for generator's exponential moving average setting. Defaults to None.

disc_loss(*disc_pred_fake: torch.Tensor*, *disc_pred_real: torch.Tensor*) → Tuple[torch.Tensor, dict]

Get disc loss. SAGAN, SNGAN and Proj-GAN use hinge loss to train the discriminator.

参数

- **disc_pred_fake** (*Tensor*) –Discriminator's prediction of the fake images.
- **disc_pred_real** (*Tensor*) –Discriminator's prediction of the real images.

返回 Loss value and a dict of log variables.

返回类型 Tuple[Tensor, dict]

gen_loss(*disc_pred_fake: torch.Tensor*) → Tuple[torch.Tensor, dict]

Get disc loss. SAGAN, SNGAN and Proj-GAN use hinge loss to train the generator.

参数 **disc_pred_fake** (*Tensor*) –Discriminator's prediction of the fake images.

返回 Loss value and a dict of log variables.

返回类型 Tuple[Tensor, dict]

train_discriminator(*inputs: dict*, *data_samples:* mmagic.structures.DataSample, *optimizer_wrapper: mmengine.optim.OptimWrapper*) → Dict[str, torch.Tensor]

Train discriminator.

参数

- **inputs** (*dict*) –Inputs from dataloader.
- **data_samples** (*DataSample*) –Data samples from dataloader.
- **optim_wrapper** (*OptimWrapper*) –OptimWrapper instance used to update model parameters.

返回 A `dict` of tensor for logging.

返回类型 Dict[str, Tensor]

train_generator(*inputs: dict*, *data_samples:* mmagic.structures.DataSample, *optimizer_wrapper: mmengine.optim.OptimWrapper*) → Dict[str, torch.Tensor]

Train generator.

参数

- **inputs** (*dict*) –Inputs from dataloader.
- **data_samples** (*DataSample*) –Data samples from dataloader. Do not used in generator's training.
- **optim_wrapper** (*OptimWrapper*) –OptimWrapper instance used to update model parameters.

返回 A `dict` of tensor for logging.

返回类型 Dict[str, Tensor]

class mmagic.models.editors.**SinGAN**(*generator: ModelType*, *discriminator: Optional[ModelType] = None*, *data_preprocessor: Optional[Union[dict, mmengine.Config]] = None*, *generator_steps: int = 1*, *discriminator_steps: int = 1*, *num_scales: Optional[int] = None*, *iters_per_scale: int = 2000*, *noise_weight_init: int = 0.1*, *lr_scheduler_args: Optional[dict] = None*, *test_pkl_data: Optional[str] = None*, *ema_confg: Optional[dict] = None*)

Bases: *mmagic.models.base_models.BaseGAN*

SinGAN.

This model implement the single image generative adversarial model proposed in: Singan: Learning a Generative Model from a Single Natural Image, ICCV'19.

Notes for training:

- This model should be trained with our dataset `SinGANDataset`.
- In training, the `total_iters` arguments is related to the number of scales in the image pyramid and `iters_per_scale` in the `train_cfg`. You should set it carefully in the training config file.

Notes for model architectures:

- The generator and discriminator need `num_scales` in initialization. However, this arguments is generated by `create_real_pyramid` function from the `singan_dataset.py`. The last element in the returned list (`stop_scale`) is the value for `num_scales`. Pay attention that this scale is counted from zero. Please see our tutorial for SinGAN to obtain more details or our standard config for reference.

参数

- **generator** (*ModelType*) –The config or model of the generator.

- **discriminator** (*Optional[ModelType]*) –The config or model of the discriminator. Defaults to None.
- **data_preprocessor** (*Optional[Union[dict, Config]]*) –The pre-process config or `DataPreprocessor`.
- **generator_steps** (*int*) –The number of times the generator is completely updated before the discriminator is updated. Defaults to 1.
- **discriminator_steps** (*int*) –The number of times the discriminator is completely updated before the generator is updated. Defaults to 1.
- **num_scales** (*int*) –The number of scales/stages in generator/ discriminator. Note that this number is counted from zero, which is the same as the original paper. Defaults to None.
- **iters_per_scale** (*int*) –The training iteration for each resolution scale. Defaults to 2000.
- **noise_weight_init** (*float*) –The initialize weight of fixed noise. Defaults to 0.1
- **lr_scheduler_args** (*Optional[dict]*) –Arguments for learning schedulers. Note that in SinGAN, we use MultiStepLR, which is the same as the original paper. If not passed, no learning schedule will be used. Defaults to None.
- **test_pkl_data** (*Optional[str]*) –The path of pickle file which contains fixed noise and noise weight. This is must for test. Defaults to None.
- **ema_config** (*Optional[Dict]*) –The config for generator's exponential moving average setting. Defaults to None.

load_test_pkl()

Load pickle for test.

_from_numpy(*data: Tuple[list, numpy.ndarray]*) → Tuple[torch.Tensor, List[torch.Tensor]]

Convert input numpy array or list of numpy array to Tensor or list of Tensor.

参数 **data** (*Tuple[list, np.ndarray]*) –Input data to convert.

返回 Converted Tensor or list of tensor.

返回类型 Tuple[Tensor, List[Tensor]]

get_module(*model: torch.nn.Module*, *module_name: str*) → torch.nn.Module

Get an inner module from model.

Since we will wrapper DDP for some model, we have to judge whether the module can be indexed directly.

参数

- **model** (*nn.Module*) –This model may wrapped with DDP or not.
- **module_name** (*str*) –The name of specific module.

返回 Returned sub module.

返回类型 nn.Module

construct_fixed_noises()

Construct the fixed noises list used in SinGAN.

forward(*inputs: mmagic.utils.ForwardInputs*, *data_samples: Optional[list] = None*, *mode=None*) → List[*mmagic.structures.DataSample*]

Forward function for SinGAN. For SinGAN, *inputs* should be a dict contains 'num_batches', 'mode' and other input arguments for the generator.

参数

- **inputs** (*dict*) –Dict containing the necessary information (e.g., noise, num_batches, mode) to generate image.
- **data_samples** (*Optional[list]*) –Data samples collated by `data_preprocessor`. Defaults to None.
- **mode** (*Optional[str]*) –*mode* is not used in `BaseConditionalGAN`. Defaults to None.

gen_loss(*disc_pred_fake: torch.Tensor*, *recon_imgs: torch.Tensor*) → Tuple[torch.Tensor, dict]

Generator loss for SinGAN. SinGAN use WGAN's loss and MSE loss to train the generator.

参数

- **disc_pred_fake** (*Tensor*) –Discriminator's prediction of the fake images.
- **recon_imgs** (*Tensor*) –Reconstructive images.

返回 Loss value and a dict of log variables.

返回类型 Tuple[Tensor, dict]

disc_loss(*disc_pred_fake: torch.Tensor*, *disc_pred_real: torch.Tensor*, *fake_data: torch.Tensor*, *real_data: torch.Tensor*) → Tuple[torch.Tensor, dict]

Get disc loss. SAGAN, SNGAN and Proj-GAN use hinge loss to train the generator.

参数

- **disc_pred_fake** (*Tensor*) –Discriminator's prediction of the fake images.
- **disc_pred_real** (*Tensor*) –Discriminator's prediction of the real images.
- **fake_data** (*Tensor*) –Generated images, used to calculate gradient penalty.
- **real_data** (*Tensor*) –Real images, used to calculate gradient penalty.

返回 Loss value and a dict of log variables.

返回类型 Tuple[Tensor, dict]

train_generator(*inputs: dict*, *data_samples: List[*mmagic.structures.DataSample*]*, *optimizer_wrapper: mmengine.optim.OptimWrapper*) → Dict[str, torch.Tensor]

Train generator.

参数

- **inputs** (*dict*) –Inputs from dataloader.
- **data_samples** (*List[DataSample]*) –Data samples from dataloader. Do not used in generator's training.
- **optim_wrapper** (*OptimWrapper*) –OptimWrapper instance used to update model parameters.

返回 A `dict` of tensor for logging.

返回类型 Dict[str, Tensor]

train_discriminator(*inputs: dict*, *data_samples: List[*mmagic.structures.DataSample*]*, *optimizer_wrapper: mmengine.optim.OptimWrapper*) → Dict[str, torch.Tensor]

Train discriminator.

参数

- **inputs** (*dict*) –Inputs from dataloader.
- **data_samples** (*List[DataSample]*) –Data samples from dataloader.
- **optim_wrapper** (*OptimWrapper*) –OptimWrapper instance used to update model parameters.

返回 A `dict` of tensor for logging.

返回类型 Dict[str, Tensor]

train_gan(*inputs_dict: dict*, *data_sample: List[*mmagic.structures.DataSample*]*, *optim_wrapper: mmengine.optim.OptimWrapperDict*) → Dict[str, torch.Tensor]

Train GAN model. In the training of GAN models, generator and discriminator are updated alternatively. In MMagic's design, *self.train_step* is called with data input. Therefore we always update discriminator, whose updating is relay on real data, and then determine if the generator needs to be updated based on the current number of iterations. More details about whether to update generator can be found in `should_gen_update()`.

参数

- **data** (*dict*) –Data sampled from dataloader.
- **data_sample** (*List[DataSample]*) –List of data sample contains GT and meta information.
- **optim_wrapper** (*OptimWrapperDict*) –OptimWrapperDict instance contains OptimWrapper of generator and discriminator.

返回 A `dict` of tensor for logging.

返回类型 Dict[str, torch.Tensor]

train_step(*data: dict*, *optim_wrapper: mmengine.optim.OptimWrapperDict*) → Dict[str, torch.Tensor]

Train step for SinGAN model. SinGAN is trained with multi-resolution images, and each resolution is trained for *:attr:self.iters_per_scale* times.

We initialize the weight and learning rate scheduler of the corresponding module at the start of each resolution's training. At the end of each resolution's training, we update the weight of the noise of current resolution by mse loss between reconstructed image and real image.

参数

- **data** (*dict*) –Data sampled from dataloader.
- **optim_wrapper** (*OptimWrapperDict*) –OptimWrapperDict instance contains OptimWrapper of generator and discriminator.

返回 A `dict` of tensor for logging.

返回类型 Dict[str, torch.Tensor]

test_step(*data: dict*) → mmagic.utils.SampleList

Gets the generated image of given data in test progress. Before generate images, we call *:meth:self.load_test_pkl* to load the fixed noise and current stage of the model from the pickle file.

参数 **data** (*dict*) –Data sampled from metric specific sampler. More details in *Metrics* and *Evaluator*.

返回 A list of `DataSample` contain generated results.

返回类型 SampleList

class mmagic.models.editors.**SRCNNNet**(*channels=(3, 64, 32, 3)*, *kernel_sizes=(9, 1, 5)*, *upscale_factor=4*)

Bases: `mmengine.model.BaseModule`

SRCNN network structure for image super resolution.

SRCNN has three conv layers. For each layer, we can define the *in_channels*, *out_channels* and *kernel_size*. The input image will first be upsampled with a bicubic upsampler, and then super-resolved in the HR spatial size.

Paper: Learning a Deep Convolutional Network for Image Super-Resolution.

参数

- **channels** (*tuple[int]*) –A tuple of channel numbers for each layer including channels of input and output . Default: (3, 64, 32, 3).
- **kernel_sizes** (*tuple[int]*) –A tuple of kernel sizes for each conv layer. Default: (9, 1, 5).

- **upscale_factor** (*int*) –Upsampling factor. Default: 4.

forward(*x*)

Forward function.

参数 **x** (*Tensor*) –Input tensor with shape (n, c, h, w).

返回 Forward results.

返回类型 Tensor

class mmagic.models.editors.**SRGAN**(*generator*, *discriminator=None*, *gan_loss=None*, *pixel_loss=None*, *perceptual_loss=None*, *train_cfg=None*, *test_cfg=None*, *init_cfg=None*, *data_preprocessor=None*)

Bases: *mmagic.models.base_models.BaseEditModel*

SRGAN model for single image super-resolution.

Ref: Photo-Realistic Single Image Super-Resolution Using a Generative Adversarial Network.

参数

- **generator** (*dict*) –Config for the generator.
- **discriminator** (*dict*) –Config for the discriminator. Default: None.
- **gan_loss** (*dict*) –Config for the gan loss. Note that the loss weight in gan loss is only for the generator.
- **pixel_loss** (*dict*) –Config for the pixel loss. Default: None.
- **perceptual_loss** (*dict*) –Config for the perceptual loss. Default: None.
- **train_cfg** (*dict*) –Config for training. Default: None.
- **test_cfg** (*dict*) –Config for testing. Default: None.
- **init_cfg** (*dict, optional*) –The weight initialized config for BaseModule. Default: None.
- **data_preprocessor** (*dict, optional*) –The pre-process config of BaseDataPreprocessor. Default: None.

forward_train(*inputs*, *data_samples=None*, ***kwargs*)

Forward training. Losses of training is calculated in train_step.

参数

- **inputs** (*torch.Tensor*) –batch input tensor collated by data_preprocessor.
- **data_samples** (*List[BaseDataElement], optional*) –data samples collated by data_preprocessor.

返回 Result of forward_tensor with training=True.

返回类型 Tensor

forward_tensor(*inputs*, *data_samples=None*, *training=False*)

Forward tensor. Returns result of simple forward.

参数

- **inputs** (*torch.Tensor*) –batch input tensor collated by data_preprocessor.
- **data_samples** (*List[BaseDataElement], optional*) –data samples collated by data_preprocessor.
- **training** (*bool*) –Whether is training. Default: False.

返回 result of simple forward.

返回类型 Tensor

if_run_g()

Calculates whether need to run the generator step.

if_run_d()

Calculates whether need to run the discriminator step.

g_step(*batch_outputs: torch.Tensor*, *batch_gt_data: torch.Tensor*)

G step of GAN: Calculate losses of generator.

参数

- **batch_outputs** (*Tensor*) –Batch output of generator.
- **batch_gt_data** (*Tensor*) –Batch GT data.

返回 Dict of losses.

返回类型 dict

d_step_real(*batch_outputs*, *batch_gt_data: torch.Tensor*)

Real part of D step.

参数

- **batch_outputs** (*Tensor*) –Batch output of generator.
- **batch_gt_data** (*Tensor*) –Batch GT data.

返回 Real part of gan_loss for discriminator.

返回类型 Tensor

d_step_fake(*batch_outputs: torch.Tensor*, *batch_gt_data*)

Fake part of D step.

参数

- **batch_outputs** (*Tensor*) –Batch output of generator.
- **batch_gt_data** (*Tensor*) –Batch GT data.

返回 Fake part of gan_loss for discriminator.

返回类型 Tensor

g_step_with_optim(*batch_outputs: torch.Tensor*, *batch_gt_data: torch.Tensor*, *optim_wrapper: mmengine.optim.OptimWrapperDict*)

G step with optim of GAN: Calculate losses of generator and run optim.

参数

- **batch_outputs** (*Tensor*) –Batch output of generator.
- **batch_gt_data** (*Tensor*) –Batch GT data.
- **optim_wrapper** (*OptimWrapperDict*) –Optim wrapper dict.

返回 Dict of parsed losses.

返回类型 dict

d_step_with_optim(*batch_outputs: torch.Tensor*, *batch_gt_data: torch.Tensor*, *optim_wrapper: mmengine.optim.OptimWrapperDict*)

D step with optim of GAN: Calculate losses of discriminator and run optim.

参数

- **batch_outputs** (*Tensor*) –Batch output of generator.
- **batch_gt_data** (*Tensor*) –Batch GT data.
- **optim_wrapper** (*OptimWrapperDict*) –Optim wrapper dict.

返回 Dict of parsed losses.

返回类型 dict

extract_gt_data(*data_samples*)

extract gt data from data samples.

参数 **data_samples** (*list*) –List of DataSample.

返回 Extract gt data.

返回类型 Tensor

train_step(*data: List[dict]*, *optim_wrapper: mmengine.optim.OptimWrapperDict*) → Dict[str, torch.Tensor]

Train step of GAN-based method.

参数

- **data** (*List[dict]*) –Data sampled from dataloader.

- **optim_wrapper** (*OptimWrapper*) –OptimWrapper instance used to update model parameters.

返回 A dict of tensor for logging.

返回类型 Dict[str, torch.Tensor]

class mmagic.models.editors.**ModifiedVGG**(*in_channels*, *mid_channels*)

Bases: mmengine.model.BaseModule

A modified VGG discriminator with input size 128 x 128.

It is used to train SRGAN and ESRGAN.

参数

- **in_channels** (*int*) –Channel number of inputs. Default: 3.
- **mid_channels** (*int*) –Channel number of base intermediate features. Default: 64.

forward(*x*)

Forward function.

参数 **x** (*Tensor*) –Input tensor with shape (n, c, h, w).

返回 Forward results.

返回类型 Tensor

class mmagic.models.editors.**MSRResNet**(*in_channels*, *out_channels*, *mid_channels=64*, *num_blocks=16*, *upscale_factor=4*)

Bases: mmengine.model.BaseModule

Modified SRResNet.

A compacted version modified from SRResNet in "Photo-Realistic Single Image Super-Resolution Using a Generative Adversarial Network".

It uses residual blocks without BN, similar to EDSR. Currently, it supports x2, x3 and x4 upsampling scale factor.

参数

- **in_channels** (*int*) –Channel number of inputs.
- **out_channels** (*int*) –Channel number of outputs.
- **mid_channels** (*int*) –Channel number of intermediate features. Default: 64.
- **num_blocks** (*int*) –Block number in the trunk network. Default: 16.
- **upscale_factor** (*int*) –Upsampling factor. Support x2, x3 and x4. Default: 4.

_supported_upscale_factors = [2, 3, 4]

forward(*x*)

Forward function.

参数 **x** (*Tensor*) –Input tensor with shape (n, c, h, w).

返回 Forward results.

返回类型 Tensor

init_weights()

Init weights for models.

class mmagic.models.editors.**StableDiffusion**(*vae: ModelType*, *text_encoder: ModelType*, *tokenizer: str*, *unet: ModelType*, *scheduler: ModelType*, *test_scheduler: Optional[ModelType] = None*, *dtype: Optional[str] = None*, *enable_xformers: bool = True*, *noise_offset_weight: float = 0*, *tomesd_cfg: Optional[dict] = None*, *data_preprocessor: Optional[ModelType] = dict(type='DataPreprocessor')*, *init_cfg: Optional[dict] = None*)

Bases: mmengine.model.BaseModel

Class for Stable Diffusion. Refers to https://github.com/Stability- AI/stablediffusion and https://github.com/huggingface/diffusers/blob/main/s rc/diffusers/pipelines/stable_diffusion/pipeline_stable_diffusion_attend_and_excite.py # noqa.

参数

- **unet** (*Union[dict, nn.Module]*) –The config or module for Unet model.
- **text_encoder** (*Union[dict, nn.Module]*) –The config or module for text encoder.
- **vae** (*Union[dict, nn.Module]*) –The config or module for VAE model.
- **tokenizer** (*str*) –The **name** for CLIP tokenizer.
- **schedule** (*Union[dict, nn.Module]*) –The config or module for diffusion scheduler.
- **test_scheduler** (*Union[dict, nn.Module], optional*) –The config or module for diffusion scheduler in test stage (*self.infer*). If not passed, will use the same scheduler as *schedule*. Defaults to None.
- **dtype** (*str, optional*) –The dtype for the model This argument will not work when dtype is defined for submodels. Defaults to None.
- **enable_xformers** (*bool, optional*) –Whether to use xformers. Defaults to True.

- **noise_offset_weight** (*bool, optional*) –The weight of noise offset introduced in https://www.crosslabs.org/blog/diffusion-with-offset-noise Defaults to 0.
- **data_preprocessor** (*dict, optional*) –The pre-process config of `BaseDataPreprocessor`.
- **init_cfg** (*dict, optional*) –The weight initialized config for `BaseModule`.

property device

set_xformers(*module: Optional[torch.nn.Module] = None*) → torch.nn.Module

Set xformers for the model.

返回 The model with xformers.

返回类型 nn.Module

set_tomesd() → torch.nn.Module

Set ToMe for the stable diffusion model.

返回 The model with ToMe.

返回类型 nn.Module

train(*mode: bool = True*)

Set train/eval mode.

参数 **mode** (*bool, optional*) –Whether set train mode. Defaults to True.

infer(*prompt: Union[str, List[str]]*, *height: Optional[int] = None*, *width: Optional[int] = None*, *num_inference_steps: int = 50*, *guidance_scale: float = 7.5*, *negative_prompt: Optional[Union[str, List[str]]] = None*, *num_images_per_prompt: Optional[int] = 1*, *eta: float = 0.0*, *generator: Optional[torch.Generator] = None*, *latents: Optional[torch.FloatTensor] = None*, *show_progress=True*, *seed=1*, *return_type='image'*)

Function invoked when calling the pipeline for generation.

参数

- **prompt** (*str* or *List[str]*) –The prompt or prompts to guide the image generation.
- **(int** (*height*) –defaults to self.unet_sample_size * self.vae_scale_factor): The height in pixels of the generated image.
- **optional** –defaults to self.unet_sample_size * self.vae_scale_factor): The height in pixels of the generated image.

:param [defaults to self.unet_sample_size * self.vae_scale_factor):] The height in pixels of the generated image.

参数

- **(int** (*width*) –defaults to self.unet_sample_size * self.vae_scale_factor): The width in pixels of the generated image.
- **optional** –defaults to self.unet_sample_size * self.vae_scale_factor): The width in pixels of the generated image.

:param [defaults to self.unet_sample_size * self.vae_scale_factor):] The width in pixels of the generated image.

参数

- **num_inference_steps** (*int*, *optional*, defaults to 50) –The number of denoising steps. More denoising steps usually lead to a higher quality image at the expense of slower inference.
- **guidance_scale** (*float*, *optional*, defaults to 7.5) –Guidance scale as defined in [Classifier-Free Diffusion Guidance] (https://arxiv.org/abs/2207.12598).
- **negative_prompt** (*str* or *List[str]*, *optional*) –The prompt or prompts not to guide the image generation. Ignored when not using guidance (i.e., ignored if *guidance_scale* is less than *1*).
- **num_images_per_prompt** (*int*, *optional*, defaults to 1) –The number of images to generate per prompt.
- **eta** (*float*, *optional*, defaults to 0.0) –Corresponds to parameter eta (η) in the DDIM paper: https://arxiv.org/abs/2010.02502. Only applies to [*schedulers.DDIMScheduler*], will be ignored for others.
- **generator** (*torch.Generator*, *optional*) –A [torch generator] to make generation deterministic.
- **latents** (*torch.FloatTensor*, *optional*) –Pre-generated noisy latents, sampled from a Gaussian distribution, to be used as inputs for image generation. Can be used to tweak the same generation with different prompts. If not provided, a latents tensor will be generated by sampling using the supplied random *generator*.
- **return_type** (*str*) –The return type of the inference results. Supported types are ‘image’, ‘numpy’, ‘tensor’. If ‘image’ is passed, a list of PIL images will be returned. If ‘numpy’ is passed, a numpy array with shape [N, C, H, W] will be returned, and the value range will be same as decoder’s output range. If ‘tensor’ is passed, the decoder’s output will be returned. Defaults to ‘image’.

返回 A dict containing the generated images.

返回类型 dict

output_to_pil(*image*) → List[PIL.Image.Image]

Convert output tensor to PIL image. Output tensor will be de-normed to [0, 255] by *DataPreprocessor.destruct*. Due to no *data_samples* is passed, color order conversion will not be performed.

参数 **image** (*torch.Tensor*) –The output tensor of the decoder.

返回 The list of processed PIL images.

返回类型 List[Image.Image]

_encode_prompt(*prompt*, *device*, *num_images_per_prompt*, *do_classifier_free_guidance*, *negative_prompt*)

Encodes the prompt into text encoder hidden states.

参数

- **prompt** (*str or list(int)*) –prompt to be encoded.
- **device** –(torch.device): torch device.
- **num_images_per_prompt** (*int*) –number of images that should be generated per prompt.
- **do_classifier_free_guidance** (*bool*) –whether to use classifier free guidance or not.
- **negative_prompt** (*str or List[str]*) –The prompt or prompts not to guide the image generation. Ignored when not using guidance (i.e., ignored if *guidance_scale* is less than *1*).

返回

text embeddings generated by clip text encoder.

返回类型 text_embeddings (torch.Tensor)

decode_latents(*latents*)

use vae to decode latents.

参数 **latents** (*torch.Tensor*) –latents to decode.

返回 image result.

返回类型 image (torch.Tensor)

prepare_extra_step_kwargs(*generator*, *eta*)

prepare extra kwargs for the scheduler step.

参数

- **generator** (*torch.Generator*) –generator for random functions.
- **eta** (*float*) –eta (η) is only used with the DDIMScheduler, it will be ignored for other schedulers. eta corresponds to η in DDIM paper: https://arxiv.org/abs/2010.02502 and should be between [0, 1]

返回 dict contains ‘generator’ and ‘eta’

返回类型 extra_step_kwargs (dict)

prepare_test_scheduler_extra_step_kwargs(*generator*, *eta*)

prepare extra kwargs for the scheduler step.

参数

- **generator** (*torch.Generator*) –generator for random functions.
- **eta** (*float*) –eta (η) is only used with the DDIMScheduler, it will be ignored for other schedulers. eta corresponds to η in DDIM paper: https://arxiv.org/abs/2010.02502 and should be between [0, 1]

返回 dict contains ‘generator’ and ‘eta’

返回类型 extra_step_kwargs (dict)

check_inputs(*prompt*, *height*, *width*)

check whether inputs are in suitable format or not.

prepare_latents(*batch_size*, *num_channels_latents*, *height*, *width*, *dtype*, *device*, *generator*, *latents=None*)

prepare latents for diffusion to run in latent space.

参数

- **batch_size** (*int*) –batch size.
- **num_channels_latents** (*int*) –latent channel nums.
- **height** (*int*) –image height.
- **width** (*int*) –image width.
- **dtype** (*torch.dtype*) –float type.
- **device** (*torch.device*) –torch device.
- **generator** (*torch.Generator*) –generator for random functions, defaults to None.
- **latents** (*torch.Tensor*) –Pre-generated noisy latents, defaults to None.

返回 prepared latents.

返回类型 latents (torch.Tensor)

val_step(*data: dict*) → mmagic.utils.typing.SampleList

Gets the predictions of given data.

Calls `self.data_preprocessor(data, False)` and `self(inputs, data_sample, mode='predict')` in order. Return the predictions which will be passed to evaluator.

参数 **data** (*dict or tuple or list*) –Data sampled from dataset.

返回 The predictions of given data.

返回类型 list

test_step(*data: dict*) → mmagic.utils.typing.SampleList

`BaseModel` implements `test_step` the same as `val_step`.

参数 **data** (*dict or tuple or list*) –Data sampled from dataset.

返回 The predictions of given data.

返回类型 list

train_step(*data*, *optim_wrapper_dict*)

Implements the default model training process including preprocessing, model forward propagation, loss calculation, optimization, and back-propagation.

During non-distributed training. If subclasses do not override the *train_step()*, `EpochBasedTrainLoop` or `IterBasedTrainLoop` will call this method to update model parameters. The default parameter update process is as follows:

1. Calls `self.data_processor(data, training=False)` to collect batch_inputs and corresponding data_samples(labels).
2. Calls `self(batch_inputs, data_samples, mode='loss')` to get raw loss
3. Calls `self.parse_losses` to get `parsed_losses` tensor used to backward and dict of loss tensor used to log messages.
4. Calls `optim_wrapper.update_params(loss)` to update model.

参数

- **data** (*dict or tuple or list*) –Data sampled from dataset.
- **optim_wrapper** (*OptimWrapper*) –OptimWrapper instance used to update model parameters.

返回 A `dict` of tensor for logging.

返回类型 Dict[str, torch.Tensor]

abstract forward(*inputs: torch.Tensor*, *data_samples: Optional[list] = None*, *mode: str = 'tensor'*) → Union[Dict[str, torch.Tensor], list]

forward is not implemented now.

class mmagic.models.editors.**StableDiffusionInpaint**(**args*, ***kwargs*)

Bases: `mmagic.models.editors.stable_diffusion.stable_diffusion.StableDiffusion`

Class for Stable Diffusion. Refers to https://github.com/Stability- AI/stablediffusion and https://github.com/huggingface/diffusers/blob/main/s rc/diffusers/pipelines/stable_diffusion/pipeline_stable_diffusion_attend_and_excite.py # noqa.

参数

- **unet** (*Union[dict, nn.Module]*) –The config or module for Unet model.
- **text_encoder** (*Union[dict, nn.Module]*) –The config or module for text encoder.
- **vae** (*Union[dict, nn.Module]*) –The config or module for VAE model.
- **tokenizer** (*str*) –The **name** for CLIP tokenizer.
- **schedule** (*Union[dict, nn.Module]*) –The config or module for diffusion scheduler.
- **test_scheduler** (*Union[dict, nn.Module], optional*) –The config or module for diffusion scheduler in test stage (*self.infer*). If not passed, will use the same scheduler as *schedule*. Defaults to None.
- **dtype** (*str, optional*) –The dtype for the model This argument will not work when dtype is defined for submodels. Defaults to None.
- **enable_xformers** (*bool, optional*) –Whether to use xformers. Defaults to True.
- **noise_offset_weight** (*bool, optional*) –The weight of noise offset introduced in https://www.crosslabs.org/blog/diffusion-with-offset-noise Defaults to 0.
- **data_preprocessor** (*dict, optional*) –The pre-process config of `BaseDataPreprocessor`.
- **init_cfg** (*dict, optional*) –The weight initialized config for `BaseModule`.

infer (*prompt: Union[str, List[str]]*, *image: Union[torch.FloatTensor, PIL.Image.Image] = None*, *mask_image: Union[torch.FloatTensor, PIL.Image.Image] = None*, *height: Optional[int] = None*, *width: Optional[int] = None*, *num_inference_steps: int = 50*, *guidance_scale: float = 7.5*, *negative_prompt: Optional[Union[str, List[str]]] = None*, *num_images_per_prompt: Optional[int] = 1*, *eta: float = 0.0*, *generator: Optional[torch.Generator] = None*, *latents: Optional[torch.FloatTensor] = None*, *show_progress=True*, *seed=1*, *return_type='image'*)

Function invoked when calling the pipeline for generation.

参数

- **prompt** (*str* or *List[str]*) –The prompt or prompts to guide the image generation.
- **image** (*Union[torch.FloatTensor, Image.Image]*) –The image to inpaint.
- **mask_image** (*Union[torch.FloatTensor, Image.Image]*) –The mask to apply to the image, i.e. regions to inpaint.

- **(int** (*height*) –defaults to self.unet_sample_size * self.vae_scale_factor): The height in pixels of the generated image.
- **optional** –defaults to self.unet_sample_size * self.vae_scale_factor): The height in pixels of the generated image.

:param [defaults to self.unet_sample_size * self.vae_scale_factor):] The height in pixels of the generated image.

参数

- **(int** (*width*) –defaults to self.unet_sample_size * self.vae_scale_factor): The width in pixels of the generated image.
- **optional** –defaults to self.unet_sample_size * self.vae_scale_factor): The width in pixels of the generated image.

:param [defaults to self.unet_sample_size * self.vae_scale_factor):] The width in pixels of the generated image.

参数

- **num_inference_steps** (*int*, *optional*, defaults to 50) –The number of denoising steps. More denoising steps usually lead to a higher quality image at the expense of slower inference.
- **guidance_scale** (*float*, *optional*, defaults to 7.5) –Guidance scale as defined in [Classifier-Free Diffusion Guidance] (https://arxiv.org/abs/2207.12598).
- **negative_prompt** (*str* or *List[str]*, *optional*) –The prompt or prompts not to guide the image generation. Ignored when not using guidance (i.e., ignored if *guidance_scale* is less than *1*).
- **num_images_per_prompt** (*int*, *optional*, defaults to 1) –The number of images to generate per prompt.
- **eta** (*float*, *optional*, defaults to 0.0) –Corresponds to parameter eta (η) in the DDIM paper: https://arxiv.org/abs/2010.02502. Only applies to [*schedulers.DDIMScheduler*], will be ignored for others.
- **generator** (*torch.Generator*, *optional*) –A [torch generator] to make generation deterministic.
- **latents** (*torch.FloatTensor*, *optional*) –Pre-generated noisy latents, sampled from a Gaussian distribution, to be used as inputs for image generation. Can be used to tweak the same generation with different prompts. If not provided, a latents tensor will be generated by sampling using the supplied random *generator*.

- **return_type** (*str*) –The return type of the inference results. Supported types are 'image', 'numpy', 'tensor'. If 'image' is passed, a list of PIL images will be returned. If 'numpy' is passed, a numpy array with shape [N, C, H, W] will be returned, and the value range will be same as decoder's output range. If 'tensor' is passed, the decoder's output will be returned. Defaults to 'image'.

返回 A dict containing the generated images.

返回类型 dict

prepare_mask_latents(*mask*, *masked_image*, *batch_size*, *num_channels_latents*, *height*, *width*, *dtype*, *device*, *generator*, *do_classifier_free_guidance*)

prepare latents for diffusion to run in latent space.

参数

- **mask** (*torch.Tensor*) –The mask to apply to the image, i.e. regions to inpaint.
- **image** (*torch.Tensor*) –The image to be masked.
- **batch_size** (*int*) –batch size.
- **num_channels_latents** (*int*) –latent channel nums.
- **height** (*int*) –image height.
- **width** (*int*) –image width.
- **dtype** (*torch.dtype*) –float type.
- **device** (*torch.device*) –torch device.
- **generator** (*torch.Generator*) –generator for random functions, defaults to None.
- **latents** (*torch.Tensor*) –Pre-generated noisy latents, defaults to None.
- **do_classifier_free_guidance** (*bool*) –Whether to apply classifier-free guidance.

返回 prepared latents.

返回类型 latents (torch.Tensor)

abstract val_step(*data: dict*) → mmagic.utils.typing.SampleList

Performs a validation step on the provided data.

This method is decorated with *torch.no_grad()* which indicates no gradients will be computed during the operations. This ensures efficient memory usage during testing.

参数 **data** (*dict*) –Dictionary containing input data for testing.

返回 List of samples processed during the testing step.

返回类型 SampleList

引发 **NotImplementedError** –This method has not been implemented.

abstract test_step(*data: dict*) → mmagic.utils.typing.SampleList

Performs a testing step on the provided data.

This method is decorated with *torch.no_grad()* which indicates no gradients will be computed during the operations. This ensures efficient memory usage during testing.

参数 **data** (*dict*) –Dictionary containing input data for testing.

返回 List of samples processed during the testing step.

返回类型 SampleList

引发 **NotImplementedError** –This method has not been implemented.

abstract train_step(*data*, *optim_wrapper_dict*)

Performs a training step on the provided data.

参数

- **data** –Input data for training.
- **optim_wrapper_dict** –Dictionary containing optimizer wrappers which may contain optimizers, schedulers, etc. required for the training step.

引发 **NotImplementedError** –This method has not been implemented.

class mmagic.models.editors.**StyleGAN1**(*generator: ModelType*, *discriminator: Optional[ModelType] = None*, *data_preprocessor: Optional[Union[dict, mmengine.Config]] = None*, *style_channels: int = 512*, *nkimgs_per_scale: dict = {}*, *interp_real: Optional[dict] = None*, *transition_kimgs: int = 600*, *prev_stage: int = 0*, *ema_config: Optional[Dict] = None*)

Bases: mmagic.models.editors.pggan.ProgressiveGrowingGAN

Implementation of *A Style-Based Generator Architecture for Generative Adversarial Networks.*

<https://openaccess.thecvf.com/content_CVPR_2019/html/Karras_A_Style-Based_Generator_Architecture_for_Generative_Adversarial_Networks_CVPR_2019_paper.html>‘_ # noqa (StyleGANv1). This class is inherited from *ProgressiveGrowingGAN* to support progressive training.

Detailed architecture can be found in StyleGAN1Generator and StyleGAN1Discriminator

参数

- **generator** (*ModelType*) –The config or model of the generator.
- **discriminator** (*Optional[ModelType]*) –The config or model of the discriminator. Defaults to None.

- **data_preprocessor** (*Optional[Union[dict, Config]]*) –The pre-process config or DataPreprocessor.
- **style_channels** (*int*) –The number of channels for style code. Defaults to 128.
- **nkimgs_per_scale** (*dict*) –The number of images need for each resolution's training. Defaults to *{}*.
- **intep_real** (*dict, optional*) –The config of interpolation method for real images. If not passed, bilinear interpolation with align_corners will be used. Defaults to None.
- **transition_kimgs** (*int, optional*) –The number of images during used to transit from the previous torgb layer to newer torgb layer. Defaults to 600.
- **prev_stage** (*int, optional*) –The resolution of previous stage. Used for resume training. Defaults to 0.
- **ema_config** (*Optional[Dict]*) –The config for generator's exponential moving average setting. Defaults to None.

disc_loss (*disc_pred_fake: torch.Tensor*, *disc_pred_real: torch.Tensor*, *fake_data: torch.Tensor*, *real_data: torch.Tensor*) → Tuple[torch.Tensor, dict]

Get disc loss. StyleGANv1 use non-saturating gan loss and R1 gradient penalty. loss to train the discriminator.

参数

- **disc_pred_fake** (*Tensor*) –Discriminator's prediction of the fake images.
- **disc_pred_real** (*Tensor*) –Discriminator's prediction of the real images.
- **fake_data** (*Tensor*) –Generated images, used to calculate gradient penalty.
- **real_data** (*Tensor*) –Real images, used to calculate gradient penalty.

返回 Loss value and a dict of log variables.

返回类型 Tuple[Tensor, dict]

gen_loss (*disc_pred_fake: torch.Tensor*) → Tuple[torch.Tensor, dict]

Generator loss for PGGAN. PGGAN use WGAN's loss to train the generator.

参数 **disc_pred_fake** (*Tensor*) –Discriminator's prediction of the fake images.

返回 Loss value and a dict of log variables.

返回类型 Tuple[Tensor, dict]

class mmagic.models.editors.**StyleGAN2** (*generator: ModelType*, *discriminator: Optional[ModelType] = None*, *data_preprocessor: Optional[Union[dict, mmengine.Config]] = None*, *generator_steps: int = 1*, *discriminator_steps: int = 1*, *ema_config: Optional[Dict] = None*, *loss_config=dict()*)

Bases: *mmagic.models.base_models.BaseGAN*

Implementation of *Analyzing and Improving the Image Quality of Stylegan*. # noqa.

Paper link: https://openaccess.thecvf.com/content_CVPR_2020/html/Karras_Analyzing_and_Improving_the_Image_Quality_of_StyleGAN_CVPR_2020_paper.html. # noqa

`StyleGAN2Generator` and `StyleGAN2Discriminator`

参数

- **generator** (*ModelType*) –The config or model of the generator.
- **discriminator** (*Optional[ModelType]*) –The config or model of the discriminator. Defaults to None.
- **data_preprocessor** (*Optional[Union[dict, Config]]*) –The pre-process config or `DataPreprocessor`.
- **generator_steps** (*int*) –The number of times the generator is completely updated before the discriminator is updated. Defaults to 1.
- **discriminator_steps** (*int*) –The number of times the discriminator is completely updated before the generator is updated. Defaults to 1.
- **ema_config** (*Optional[Dict]*) –The config for generator's exponential moving average setting. Defaults to None.

disc_loss(*disc_pred_fake: torch.Tensor*, *disc_pred_real: torch.Tensor*, *real_imgs: torch.Tensor*) → Tuple

Get disc loss. StyleGANv2 use the non-saturating loss and R1 gradient penalty to train the discriminator.

参数

- **disc_pred_fake** (*Tensor*) –Discriminator's prediction of the fake images.
- **disc_pred_real** (*Tensor*) –Discriminator's prediction of the real images.
- **real_imgs** (*Tensor*) –Input real images.

返回 Loss value and a dict of log variables.

返回类型 tuple[Tensor, dict]

gen_loss(*disc_pred_fake: torch.Tensor*, *batch_size: int*) → Tuple

Get gen loss. StyleGANv2 use the non-saturating loss and generator path regularization to train the generator.

参数

- **disc_pred_fake** (*Tensor*) –Discriminator's prediction of the fake images.
- **batch_size** (*int*) –Batch size for generating fake images.

返回 Loss value and a dict of log variables.

返回类型 tuple[Tensor, dict]

train_discriminator(*inputs: dict*, *data_samples:* mmagic.structures.DataSample, *optimizer_wrapper: mmengine.optim.OptimWrapper*) → Dict[str, torch.Tensor]

Train discriminator.

参数

- **inputs** (*dict*) –Inputs from dataloader.
- **data_samples** (*DataSample*) –Data samples from dataloader.
- **optim_wrapper** (*OptimWrapper*) –OptimWrapper instance used to update model parameters.

返回 A `dict` of tensor for logging.

返回类型 Dict[str, Tensor]

train_generator(*inputs: dict*, *data_samples:* mmagic.structures.DataSample, *optimizer_wrapper: mmengine.optim.OptimWrapper*) → Dict[str, torch.Tensor]

Train generator.

参数

- **inputs** (*dict*) –Inputs from dataloader.
- **data_samples** (*DataSample*) –Data samples from dataloader. Do not used in generator's training.
- **optim_wrapper** (*OptimWrapper*) –OptimWrapper instance used to update model parameters.

返回 A `dict` of tensor for logging.

返回类型 Dict[str, Tensor]

train_step(*data: dict*, *optim_wrapper: mmengine.optim.OptimWrapperDict*) → Dict[str, torch.Tensor]

Train GAN model. In the training of GAN models, generator and discriminator are updated alternatively. In MMagic's design, *self.train_step* is called with data input. Therefore we always update discriminator, whose updating is relay on real data, and then determine if the generator needs to be updated based on the current number of iterations. More details about whether to update generator can be found in `should_gen_update()`.

参数

- **data** (*dict*) –Data sampled from dataloader.
- **optim_wrapper** (*OptimWrapperDict*) –OptimWrapperDict instance contains OptimWrapper of generator and discriminator.

返回 A `dict` of tensor for logging.

返回类型 Dict[str, torch.Tensor]

class mmagic.models.editors.**StyleGAN3**(*generator: ModelType*, *discriminator: Optional[ModelType] = None*, *data_preprocessor: Optional[Union[dict, mmengine.Config]] = None*, *generator_steps: int = 1*, *discriminator_steps: int = 1*, *forward_kwargs: Optional[Dict] = None*, *ema_config: Optional[Dict] = None*, *loss_config=dict()*)

Bases: `mmagic.models.editors.stylegan2.StyleGAN2`

Implementation of *Alias-Free Generative Adversarial Networks*. # noqa.

Paper link: https://nvlabs-fi-cdn.nvidia.com/stylegan3/stylegan3-paper.pdf # noqa

Detailed architecture can be found in

`StyleGAN3Generator` and `StyleGAN2Discriminator`

test_step(*data: dict*) → mmagic.utils.typing.SampleList

Gets the generated image of given data. Same as *val_step()*.

参数 **data** (*dict*) –Data sampled from metric specific sampler. More details in *Metrics* and *Evaluator*.

返回 A list of `DataSample` contain generated results.

返回类型 SampleList

val_step(*data: dict*) → mmagic.utils.typing.SampleList

Gets the generated image of given data. Same as *val_step()*.

参数 **data** (*dict*) –Data sampled from metric specific sampler. More details in *Metrics* and *Evaluator*.

返回 A list of `DataSample` contain generated results.

返回类型 SampleList

train_discriminator(*inputs: dict*, *data_samples:* mmagic.structures.DataSample, *optimizer_wrapper: mmengine.optim.OptimWrapper*) → Dict[str, torch.Tensor]

Train discriminator.

参数

- **inputs** (*dict*) –Inputs from dataloader.
- **data_samples** (*DataSample*) –Data samples from dataloader.
- **optim_wrapper** (*OptimWrapper*) –OptimWrapper instance used to update model parameters.

返回 A `dict` of tensor for logging.

返回类型 Dict[str, Tensor]

train_generator(*inputs: dict*, *data_samples:* mmagic.structures.DataSample, *optimizer_wrapper: mmengine.optim.OptimWrapper*) → Dict[str, torch.Tensor]

Train generator.

参数

- **inputs** (*dict*) –Inputs from dataloader.
- **data_samples** (*DataSample*) –Data samples from dataloader. Do not used in generator's training.
- **optim_wrapper** (*OptimWrapper*) –OptimWrapper instance used to update model parameters.

返回 A `dict` of tensor for logging.

返回类型 Dict[str, Tensor]

sample_equivarience_pairs(*batch_size*, *sample_mode='ema'*, *eq_cfg=dict(compute_eqt_int=False, compute_eqt_frac=False, compute_eqr=False, translate_max=0.125, rotate_max=1)*, *sample_kwargs=dict()*)

class mmagic.models.editors.**StyleGAN3Generator**(*out_size*, *style_channels*, *img_channels*, *noise_size=512*, *rgb2bgr=False*, *pretrained=None*, *synthesis_cfg=dict(type='SynthesisNetwork')*, *mapping_cfg=dict(type='MappingNetwork')*)

Bases: `mmengine.model.BaseModule`

StyleGAN3 Generator.

In StyleGAN3, we make several changes to StyleGANv2's generator which include transformed fourier features, filtered nonlinearity and non-critical sampling, etc. More details can be found in: Alias-Free Generative Adversarial Networks NeurIPS'2021.

Ref: https://github.com/NVlabs/stylegan3

参数

- **out_size** (*int*) –The output size of the StyleGAN3 generator.
- **style_channels** (*int*) –The number of channels for style code.
- **img_channels** (*int*) –The number of output's channels.
- **noise_size** (*int, optional*) –Size of the input noise vector. Defaults to 512.

- **rgb2bgr** (*bool, optional*) –Whether to reformat the output channels with order *bgr*. We provide several pre-trained StyleGAN3 weights whose output channels order is *rgb*. You can set this argument to True to use the weights.
- **pretrained** (*str | dict, optional*) –Path for the pretrained model or dict containing information for pretrained models whose necessary key is 'ckpt_path'. Besides, you can also provide 'prefix' to load the generator part from the whole state dict. Defaults to None.
- **synthesis_cfg** (*dict, optional*) –Config for synthesis network. Defaults to dict(type='SynthesisNetwork').
- **mapping_cfg** (*dict, optional*) –Config for mapping network. Defaults to dict(type='MappingNetwork').

_load_pretrained_model(*ckpt_path*, *prefix=''*, *map_location='cpu'*, *strict=True*)

forward(*noise*, *num_batches=0*, *input_is_latent=False*, *truncation=1*, *num_truncation_layer=None*, *update_emas=False*, *force_fp32=True*, *return_noise=False*, *return_latents=False*)

Forward Function for stylegan3.

参数

- **noise** (*torch.Tensor | callable | None*) –You can directly give a batch of noise through a `torch.Tensor` or offer a callable function to sample a batch of noise data. Otherwise, the `None` indicates to use the default noise sampler.
- **num_batches** (*int, optional*) –The number of batch size. Defaults to 0.
- **input_is_latent** (*bool, optional*) –If *True*, the input tensor is the latent tensor. Defaults to False.
- **truncation** (*float, optional*) –Truncation factor. Give value less than 1., the truncation trick will be adopted. Defaults to 1.
- **num_truncation_layer** (*int, optional*) –Number of layers use truncated latent. Defaults to None.
- **update_emas** (*bool, optional*) –Whether update moving average of mean latent. Defaults to False.
- **force_fp32** (*bool, optional*) –Force fp32 ignore the weights. Defaults to True.
- **return_noise** (*bool, optional*) –If True, `noise_batch` will be returned in a dict with `fake_img`. Defaults to False.
- **return_latents** (*bool, optional*) –If True, `latent` will be returned in a dict with `fake_img`. Defaults to False.

返回 Generated image tensor or dictionary containing more data.

返回类型 torch.Tensor | dict

get_mean_latent(*num_samples=4096*, ***kwargs*)

Get mean latent of W space in this generator.

参数 **num_samples** (*int, optional*) –Number of sample times. Defaults to 4096.

返回 Mean latent of this generator.

返回类型 Tensor

get_training_kwargs(*phase*)

Get training kwargs. In StyleGANv3, we enable fp16, and update magnitude ema during training of discriminator. This function is used to pass related arguments.

参数 **phase** (*str*) –Current training phase.

返回 Training kwargs.

返回类型 dict

class mmagic.models.editors.**SwinIRNet**(*img_size=64*, *patch_size=1*, *in_chans=3*, *embed_dim=96*, *depths=[6, 6, 6, 6]*, *num_heads=[6, 6, 6, 6]*, *window_size=7*, *mlp_ratio=4.0*, *qkv_bias=True*, *qk_scale=None*, *drop_rate=0.0*, *attn_drop_rate=0.0*, *drop_path_rate=0.1*, *norm_layer=nn.LayerNorm*, *ape=False*, *patch_norm=True*, *use_checkpoint=False*, *upscale=2*, *img_range=1.0*, *upsampler=''*, *resi_connection='1conv'*, ***kwargs*)

Bases: mmengine.model.BaseModule

SwinIR A PyTorch impl of: *SwinIR: Image Restoration Using Swin Transformer*, based on Swin Transformer. Ref repo: https://github.com/JingyunLiang/SwinIR

参数

- **img_size** (*int | tuple(int)*) –Input image size. Default 64
- **patch_size** (*int | tuple(int)*) –Patch size. Default: 1
- **in_chans** (*int*) –Number of input image channels. Default: 3
- **embed_dim** (*int*) –Patch embedding dimension. Default: 96
- **depths** (*tuple(int)*) –Depth of each Swin Transformer layer. Default: [6, 6, 6, 6]
- **num_heads** (*tuple(int)*) –Number of attention heads in different layers. Default: [6, 6, 6, 6]
- **window_size** (*int*) –Window size. Default: 7
- **mlp_ratio** (*float*) –Ratio of mlp hidden dim to embedding dim. Default: 4
- **qkv_bias** (*bool*) –If True, add a learnable bias to query, key, value. Default: True

- **qk_scale** (*float*) –Override default qk scale of head_dim ** -0.5 if set. Default: None
- **drop_rate** (*float*) –Dropout rate. Default: 0
- **attn_drop_rate** (*float*) –Attention dropout rate. Default: 0
- **drop_path_rate** (*float*) –Stochastic depth rate. Default: 0.1
- **norm_layer** (*nn.Module*) –Normalization layer. Default: nn.LayerNorm.
- **ape** (*bool*) –If True, add absolute position embedding to the patch embedding. Default: False
- **patch_norm** (*bool*) –If True, add normalization after patch embedding. Default: True
- **use_checkpoint** (*bool*) –Whether to use checkpointing to save memory. Default: False
- **upscale** (*int*) –Upscale factor. 2/3/4/8 for image SR, 1 for denoising and compress artifact reduction. Default: 2
- **img_range** (*float*) –Image range. 1. or 255. Default: 1.0
- **upsampler** (*string, optional*) –The reconstruction module. 'pixelshuffle'/ 'pixelshuffledirect'/'nearest+conv'/None. Default: ''
- **resi_connection** (*string*) –The convolutional block before residual connection. '1conv'/'3conv'. Default: '1conv'

_init_weights(*m*)

no_weight_decay()

no_weight_decay_keywords()

check_image_size(*x*)

Check image size and pad images so that it has enough dimension do window size.

参数 **x** –input tensor image with (B, C, H, W) shape.

forward_features(*x*)

Forward function of Deep Feature Extraction.

参数 **x** (*Tensor*) –Input tensor with shape (B, C, H, W).

返回 Forward results.

返回类型 Tensor

forward(*x*)

Forward function.

参数 **x** (*Tensor*) –Input tensor with shape (B, C, H, W).

返回 Forward results.

返回类型 Tensor

class mmagic.models.editors.**TDAN**(*generator*, *pixel_loss*, *lq_pixel_loss*, *train_cfg=None*, *test_cfg=None*, *init_cfg=None*, *data_preprocessor=None*)

Bases: mmagic.models.BaseEditModel

TDAN model for video super-resolution.

Paper: TDAN: Temporally-Deformable Alignment Network for Video Super- Resolution, CVPR, 2020

参数

- **generator** (*dict*) –Config for the generator structure.
- **pixel_loss** (*dict*) –Config for pixel-wise loss.
- **lq_pixel_loss** (*dict*) –Config for pixel-wise loss for the LQ images.
- **train_cfg** (*dict*) –Config for training. Default: None.
- **test_cfg** (*dict*) –Config for testing. Default: None.
- **init_cfg** (*dict, optional*) –The weight initialized config for BaseModule.
- **data_preprocessor** (*dict, optional*) –The pre-process config of BaseDataPreprocessor.

forward_train(*inputs*, *data_samples=None*, ***kwargs*)

Forward training. Returns dict of losses of training.

参数

- **inputs** (*torch.Tensor*) –batch input tensor collated by data_preprocessor.
- **data_samples** (*List[BaseDataElement], optional*) –data samples collated by data_preprocessor.

返回 Dict of losses.

返回类型 dict

forward_tensor(*inputs*, *data_samples=None*, *training=False*, ***kwargs*)

Forward tensor. Returns result of simple forward.

参数

- **inputs** (*torch.Tensor*) –batch input tensor collated by data_preprocessor.
- **data_samples** (*List[BaseDataElement], optional*) –data samples collated by data_preprocessor.

- **training** (*bool*) –Whether is training. Default: False.

返回

results of forward inference and forward train.

返回类型 (Tensor | List[Tensor])

class mmagic.models.editors.**TDANNet**(*in_channels=3*, *mid_channels=64*, *out_channels=3*, *num_blocks_before_align=5*, *num_blocks_after_align=10*)

Bases: mmengine.model.BaseModule

TDAN network structure for video super-resolution.

Support only x4 upsampling.

Paper: TDAN: Temporally-Deformable Alignment Network for Video Super- Resolution, CVPR, 2020

参数

- **in_channels** (*int*) –Number of channels of the input image. Default: 3.
- **mid_channels** (*int*) –Number of channels of the intermediate features. Default: 64.
- **out_channels** (*int*) –Number of channels of the output image. Default: 3.
- **num_blocks_before_align** (*int*) –Number of residual blocks before temporal alignment. Default: 5.
- **num_blocks_after_align** (*int*) –Number of residual blocks after temporal alignment. Default: 10.

forward(*lrs*)

Forward function for TDANNet.

参数 **lrs** (*Tensor*) –Input LR sequence with shape (n, t, c, h, w).

返回 Output HR image with shape (n, c, 4h, 4w) and aligned LR images with shape (n, t, c, h, w).

返回类型 tuple[Tensor]

class mmagic.models.editors.**TextualInversion**(*placeholder_token: str*, *vae: ModelType*, *text_encoder: ModelType*, *tokenizer: str*, *unet: ModelType*, *scheduler: ModelType*, *test_scheduler: Optional[ModelType] = None*, *dtype: Optional[str] = None*, *enable_xformers: bool = True*, *noise_offset_weight: float = 0*, *tomesd_cfg: Optional[dict] = None*, *initialize_token: Optional[str] = None*, *num_vectors_per_token: int = 1*, *val_prompts=None*, *data_preprocessor: Optional[ModelType] = dict(type='DataPreprocessor')*, *init_cfg: Optional[dict] = None*)

Bases: mmagic.models.editors.stable_diffusion.stable_diffusion.StableDiffusion

Implementation of 'An Image is Worth One Word: Personalizing Text-to- Image Generation using Textual Inversion.

<https://arxiv.org/abs/2208.01618>'_ (Textual Inversion).

参数

- **vae** (*Union[dict, nn.Module]*) –The config or module for VAE model.
- **text_encoder** (*Union[dict, nn.Module]*) –The config or module for text encoder.
- **tokenizer** (*str*) –The **name** for CLIP tokenizer.
- **unet** (*Union[dict, nn.Module]*) –The config or module for Unet model.
- **schedule** (*Union[dict, nn.Module]*) –The config or module for diffusion scheduler.
- **test_scheduler** (*Union[dict, nn.Module], optional*) –The config or module for diffusion scheduler in test stage (*self.infer*). If not passed, will use the same scheduler as *schedule*. Defaults to None.
- **dtype** (*str, optional*) –The dtype for the model. Defaults to 'fp16'.
- **enable_xformers** (*bool, optional*) –Whether to use xformers. Defaults to True.
- **noise_offset_weight** (*bool, optional*) –The weight of noise offset introduced in https://www.crosslabs.org/blog/diffusion-with-offset-noise # noqa Defaults to 0.
- **tomesd_cfg** (*dict, optional*) –The config for TOMESD. Please refers to https://github.com/dbolya/tomesd and https://github.com/open-mmlab/mmagic/blob/main/mmagic/models/utils/tome_utils.py for detail. # noqa Defaults to None.

- **initialize_token** (*str, optional*) –The initialization token for textual embedding to train. Defaults to None.
- **num_vefctor_per_token** (*int*) –The length of the learnable embedding. Defaults to 1.
- **val_prompts** (*Union[str, List[str]], optional*) –The prompts for validation. Defaults to None.
- **data_preprocessor** (*dict, optional*) –The pre-process config of `BaseDataPreprocessor`. Defaults to

  dict(type='DataPreprocessor').
- **init_cfg** (*dict, optional*) –The weight initialized config for `BaseModule`. Defaults to None/

prepare_models()

Disable gradient for untrainable modules to save memory.

val_step(*data: dict*) → mmagic.utils.typing.SampleList

Gets the generated image of given data. Calls `self.data_preprocessor` and `self.infer` in order. Return the generated results which will be passed to evaluator or visualizer.

参数 **data** (*dict or tuple or list*) –Data sampled from dataset.

返回 Generated image or image dict.

返回类型 SampleList

test_step(*data: dict*) → mmagic.utils.typing.SampleList

Gets the generated image of given data. Calls `self.data_preprocessor` and `self.infer` in order. Return the generated results which will be passed to evaluator or visualizer.

参数 **data** (*dict or tuple or list*) –Data sampled from dataset.

返回 Generated image or image dict.

返回类型 SampleList

add_tokens(*placeholder_token: str*, *initialize_token: str = None*, *num_vectors_per_token: int = 1*)

Add token for training.

TODO: support add tokens as dict, then we can load pretrained tokens.

train_step(*data*, *optim_wrapper*)

Training step.

class mmagic.models.editors.**TOFlowVFINet**(*rgb_mean=[0.485, 0.456, 0.406]*, *rgb_std=[0.229, 0.224, 0.225]*, *flow_cfg=dict(norm_cfg=None, pretrained=None)*, *init_cfg=None*)

Bases: mmengine.model.BaseModule

PyTorch implementation of TOFlow for video frame interpolation.

Paper: Xue et al., Video Enhancement with Task-Oriented Flow, IJCV 2018 Code reference:

1. https://github.com/anchen1011/toflow
2. https://github.com/Coldog2333/pytoflow

参数

- **rgb_mean** (*list[float]*) –Image mean in RGB orders. Default: [0.485, 0.456, 0.406]
- **rgb_std** (*list[float]*) –Image std in RGB orders. Default: [0.229, 0.224, 0.225]
- **flow_cfg** (*dict*) –Config of SPyNet. Default: dict(norm_cfg=None, pretrained=None)
- **init_cfg** (*dict, optional*) –Initialization config dict. Default: None.

forward(*imgs*)

参数 **imgs** –Input frames with shape of (b, 2, 3, h, w).

返回 Interpolated frame with shape of (b, 3, h, w).

返回类型 Tensor

class mmagic.models.editors.**TOFlowVSRNet**(*adapt_official_weights=False*, *init_cfg=None*)

Bases: mmengine.model.BaseModule

PyTorch implementation of TOFlow.

In TOFlow, the LR frames are pre-upsampled and have the same size with the GT frames.

Paper: Xue et al., Video Enhancement with Task-Oriented Flow, IJCV 2018 Code reference:

1. https://github.com/anchen1011/toflow
2. https://github.com/Coldog2333/pytoflow

参数 **adapt_official_weights** (*bool*) –Whether to adapt the weights translated from the official implementation. Set to false if you want to train from scratch. Default: False

forward(*lrs*)

参数 **lrs** –Input lr frames: (b, 7, 3, h, w).

返回 SR frame: (b, 3, h, w).

返回类型 Tensor

class mmagic.models.editors.**ToFResBlock**

Bases: torch.nn.Module

ResNet architecture.

Three-layers ResNet/ResBlock

forward(*frames*)

参数 **frames** (*Tensor*) –Tensor with shape of (b, 2, 3, h, w).

返回 Interpolated frame with shape of (b, 3, h, w).

返回类型 Tensor

class mmagic.models.editors.**LTE**(*requires_grad=True*, *pixel_range=1.0*, *load_pretrained_vgg=True*, *init_cfg=None*)

Bases: mmengine.model.BaseModule

Learnable Texture Extractor.

Based on pretrained VGG19. Generate features in 3 levels.

参数

- **requires_grad** (*bool*) –Require grad or not. Default: True.
- **pixel_range** (*float*) –Pixel range of feature. Default: 1.
- **load_pretrained_vgg** (*bool*) –Load pretrained VGG from torchvision. Default: True. Train: must load pretrained VGG. Eval: needn't load pretrained VGG, because we will load pretrained LTE.
- **init_cfg** (*dict, optional*) –Initialization config dict.

forward(*x*)

Forward function.

参数 **x** (*Tensor*) –Input tensor with shape (n, 3, h, w).

返回

Forward results in 3 levels. x_level3: Forward results in level 3 (n, 256, h/4, w/4). x_level2: Forward results in level 2 (n, 128, h/2, w/2). x_level1: Forward results in level 1 (n, 64, h, w).

返回类型 Tuple[Tensor]

class mmagic.models.editors.**TTSR**(*generator*, *extractor*, *transformer*, *pixel_loss*, *discriminator=None*, *perceptual_loss=None*, *transferal_perceptual_loss=None*, *gan_loss=None*, *train_cfg=None*, *test_cfg=None*, *init_cfg=None*, *data_preprocessor=None*)

Bases: mmagic.models.editors.srgan.SRGAN

TTSR model for Reference-based Image Super-Resolution.

Paper: Learning Texture Transformer Network for Image Super-Resolution.

参数

- **generator** (*dict*) –Config for the generator.
- **extractor** (*dict*) –Config for the extractor.
- **transformer** (*dict*) –Config for the transformer.
- **pixel_loss** (*dict*) –Config for the pixel loss.
- **discriminator** (*dict*) –Config for the discriminator. Default: None.
- **perceptual_loss** (*dict*) –Config for the perceptual loss. Default: None.
- **transferal_perceptual_loss** (*dict*) –Config for the transferal perceptual loss. Default: None.
- **gan_loss** (*dict*) –Config for the GAN loss. Default: None
- **train_cfg** (*dict*) –Config for train. Default: None.
- **test_cfg** (*dict*) –Config for testing. Default: None.
- **init_cfg** (*dict, optional*) –The weight initialized config for BaseModule. Default: None.
- **data_preprocessor** (*dict, optional*) –The pre-process config of BaseDataPreprocessor. Default: None.

forward_tensor(*inputs*, *data_samples=None*, *training=False*)

Forward tensor. Returns result of simple forward.

参数

- **inputs** (*torch.Tensor*) –batch input tensor collated by data_preprocessor.
- **data_samples** (*List[BaseDataElement], optional*) –data samples collated by data_preprocessor.
- **training** (*bool*) –Whether is training. Default: False.

返回

results of forward inference and forward train.

返回类型 (Tensor | Tuple[List[Tensor]])

if_run_g()

Calculates whether need to run the generator step.

if_run_d()

Calculates whether need to run the discriminator step.

g_step(*batch_outputs*, *batch_gt_data:* mmagic.structures.DataSample)

G step of GAN: Calculate losses of generator.

参数

- **batch_outputs** (*Tensor*) –Batch output of generator.
- **batch_gt_data** (*Tensor*) –Batch GT data.

返回 Dict of losses.

返回类型 dict

g_step_with_optim(*batch_outputs: torch.Tensor*, *batch_gt_data: torch.Tensor*, *optim_wrapper: mmengine.optim.OptimWrapperDict*)

G step with optim of GAN: Calculate losses of generator and run optim.

参数

- **batch_outputs** (*Tensor*) –Batch output of generator.
- **batch_gt_data** (*Tensor*) –Batch GT data.
- **optim_wrapper** (*OptimWrapperDict*) –Optim wrapper dict.

返回 Dict of parsed losses.

返回类型 dict

train_step(*data: List[dict]*, *optim_wrapper: mmengine.optim.OptimWrapperDict*) → Dict[str, torch.Tensor]

Train step of GAN-based method.

参数

- **data** (*List[dict]*) –Data sampled from dataloader.
- **optim_wrapper** (*OptimWrapper*) –OptimWrapper instance used to update model parameters.

返回 A `dict` of tensor for logging.

返回类型 Dict[str, torch.Tensor]

class mmagic.models.editors.**SearchTransformer**(*init_cfg: Union[dict, List[dict], None] = None*)

Bases: `mmengine.model.BaseModule`

Search texture reference by transformer.

Include relevance embedding, hard-attention and soft-attention.

gather(*inputs*, *dim*, *index*)

Hard Attention. Gathers values along an axis specified by dim.

参数

- **inputs** (*Tensor*) –The source tensor. (N, C*k*k, H*W)
- **dim** (*int*) –The axis along which to index.
- **index** (*Tensor*) –The indices of elements to gather. (N, H*W)

results: outputs (Tensor): The result tensor. (N, C*k*k, H*W)

forward(*img_lq*, *ref_lq*, *refs*)

Texture transformer.

Q = LTE(img_lq) K = LTE(ref_lq) V = LTE(ref), from V_level_n to V_level_1

Relevance embedding aims to embed the relevance between the LQ and Ref image by estimating the similarity between Q and K.

Hard-Attention: Only transfer features from the most relevant position in V for each query.

Soft-Attention: synthesize features from the transferred GT texture features T and the LQ features F from the backbone.

参数

- **extractor** (*All args are features come from*) –These features contain 3 levels. When upscale_factor=4, the size ratio of these features is level3:level2:level1 = 1:2:4.
- **img_lq** (*Tensor*) –Tensor of 4x bicubic-upsampled lq image. (N, C, H, W)
- **ref_lq** (*Tensor*) –Tensor of ref_lq. ref_lq is obtained by applying bicubic down-sampling and up-sampling with factor 4x on ref. (N, C, H, W)
- **refs** (*Tuple[Tensor]*) –Tuple of ref tensors. [(N, C, H, W), (N, C/2, 2H, 2W), …]

返回

tuple contains: soft_attention (Tensor): Soft-Attention tensor. (N, 1, H, W)

textures (Tuple[Tensor]): Transferred GT textures. [(N, C, H, W), (N, C/2, 2H, 2W), …]

返回类型 tuple

class mmagic.models.editors.**TTSRDiscriminator**(*in_channels=3*, *in_size=160*, *init_cfg=None*)

Bases: mmengine.model.BaseModule

A discriminator for TTSR.

参数

- **in_channels** (*int*) –Channel number of inputs. Default: 3.
- **in_size** (*int*) –Size of input image. Default: 160.
- **init_cfg** (*dict, optional*) –Initialization config dict.

forward(*x*)

Forward function.

参数 **x** (*Tensor*) –Input tensor with shape (n, c, h, w).

返回 Forward results.

返回类型 Tensor

class mmagic.models.editors.**TTSRNet**(*in_channels*, *out_channels*, *mid_channels=64*, *texture_channels=64*, *num_blocks=(16, 16, 8, 4)*, *res_scale=1.0*, *init_cfg=None*)

Bases: mmengine.model.BaseModule

TTSR network structure (main-net) for reference-based super-resolution.

Paper: Learning Texture Transformer Network for Image Super-Resolution

Adapted from 'https://github.com/researchmm/TTSR.git' 'https://github.com/researchmm/TTSR' Copyright permission at 'https://github.com/researchmm/TTSR/issues/38'.

参数

- **in_channels** (*int*) –Number of channels in the input image
- **out_channels** (*int*) –Number of channels in the output image
- **mid_channels** (*int*) –Channel number of intermediate features. Default: 64
- **texture_channels** (*int*) –Number of texture channels. Default: 64.
- **num_blocks** (*tuple[int]*) –Block numbers in the trunk network. Default: (16, 16, 8, 4)
- **res_scale** (*float*) –Used to scale the residual in residual block. Default: 1.
- **init_cfg** (*dict, optional*) –Initialization config dict.

forward(*x*, *soft_attention*, *textures*)

Forward function.

参数

- **x** (*Tensor*) –Input tensor with shape (n, c, h, w).
- **soft_attention** (*Tensor*) –Soft-Attention tensor with shape (n, 1, h, w).
- **textures** (*Tuple[Tensor]*) –Transferred HR texture tensors. [(N, C, H, W), (N, C/2, 2H, 2W), ···]

返回 Forward results.

返回类型 Tensor

class mmagic.models.editors.**WGANGP**(**args*, ***kwargs*)

Bases: *mmagic.models.base_models.BaseGAN*

Implementation of *Improved Training of Wasserstein GANs*.

Paper link: https://arxiv.org/pdf/1704.00028

Detailed architecture can be found in `WGANGPGenerator` and `WGANGPDiscriminator`

disc_loss(*real_data: torch.Tensor*, *fake_data: torch.Tensor*, *disc_pred_fake: torch.Tensor*, *disc_pred_real: torch.Tensor*) → Tuple

Get disc loss. WGAN-GP use the wgan loss and gradient penalty to train the discriminator.

参数

- **real_data** (*Tensor*) –Real input data.
- **fake_data** (*Tensor*) –Fake input data.
- **disc_pred_fake** (*Tensor*) –Discriminator's prediction of the fake images.
- **disc_pred_real** (*Tensor*) –Discriminator's prediction of the real images.

返回 Loss value and a dict of log variables.

返回类型 tuple[Tensor, dict]

gen_loss(*disc_pred_fake: torch.Tensor*) → Tuple

Get gen loss. DCGAN use the wgan loss to train the generator.

参数 **disc_pred_fake** (*Tensor*) –Discriminator's prediction of the fake images.

返回 Loss value and a dict of log variables.

返回类型 tuple[Tensor, dict]

train_discriminator(*inputs: dict*, *data_samples:* mmagic.structures.DataSample, *optimizer_wrapper: mmengine.optim.OptimWrapper*) → Dict[str, torch.Tensor]

Train discriminator.

参数

- **inputs** (*dict*) –Inputs from dataloader.
- **data_samples** (*DataSample*) –Data samples from dataloader.

- **optim_wrapper** (*OptimWrapper*) –OptimWrapper instance used to update model parameters.

返回 A `dict` of tensor for logging.

返回类型 Dict[str, Tensor]

train_generator(*inputs: dict*, *data_samples:* mmagic.structures.DataSample, *optimizer_wrapper: mmengine.optim.OptimWrapper*) → Dict[str, torch.Tensor]

Train generator.

参数

- **inputs** (*dict*) –Inputs from dataloader.
- **data_samples** (*DataSample*) –Data samples from dataloader. Do not used in generator's training.
- **optim_wrapper** (*OptimWrapper*) –OptimWrapper instance used to update model parameters.

返回 A `dict` of tensor for logging.

返回类型 Dict[str, Tensor]

CHAPTER 83

mmagic.utils

83.1 Package Contents

83.1.1 Functions

modify_args()	Modify args of argparse.ArgumentParser.
all_to_tensor(value)	Trans image and sequence of frames to tensor.
can_convert_to_image(value)	Judge whether the input value can be converted to image tensor via
get_box_info(pred_bbox, original_shape, final_size)	**param pred_bbox** The bounding box for the instance
reorder_image(img[, input_order])	Reorder images to 'HWC' order.
tensor2img(tensor[, out_type, min_max])	Convert torch Tensors into image numpy arrays.
to_numpy(img[, dtype])	Convert data into numpy arrays of dtype.
download_from_url(url[, dest_path, dest_dir, hash_prefix])	Download object at the given URL to a local path.
print_colored_log(msg[, level, color])	Print colored log with default logger.
get_sampler(sample_kwargs, runner)	Get a sampler to loop input data.
register_all_modules(→ None)	Register all modules in mmagic into the registries.
try_import(→ Optional[types.ModuleType])	Try to import a module.
add_gaussian_noise(img, mu, sigma)	Add Gaussian Noise on the input image.
~~*adjust_gamma*(image[, gamma, gain])~~	~~Performs Gamma Correction on the input image.~~
bbox2mask(img_shape, bbox[, dtype])	Generate mask in np.ndarray from bbox.
brush_stroke_mask(img_shape[, num_vertices, ...])	Generate free-form mask.

83.1.2 Attributes

MMAGIC_CACHE_DIR
ConfigType
ForwardInputs
LabelVar
NoiseVar
SampleList

mmagic.utils.**modify_args**()

Modify args of argparse.ArgumentParser.

mmagic.utils.**all_to_tensor**(*value*)

Trans image and sequence of frames to tensor.

参数 **value** (*np.ndarray | list[np.ndarray] | Tuple[np.ndarray]*) –The original image or list of frames.

返回 The output tensor.

返回类型 Tensor

mmagic.utils.**can_convert_to_image**(*value*)

Judge whether the input value can be converted to image tensor via `images_to_tensor()` function.

参数 **value** (*any*) –The input value.

返回

If true, the input value can convert to image with `images_to_tensor()`, and vice versa.

返回类型 bool

mmagic.utils.**get_box_info**(*pred_bbox*, *original_shape*, *final_size*)

参数

- **pred_bbox** –The bounding box for the instance
- **original_shape** –Original image shape
- **final_size** –Size of the final output

返回 [L_pad, R_pad, T_pad, B_pad, rh, rw]

返回类型 List

mmagic.utils.**reorder_image**(*img*, *input_order='HWC'*)

Reorder images to 'HWC' order.

If the input_order is (h, w), return (h, w, 1); If the input_order is (c, h, w), return (h, w, c); If the input_order is (h, w, c), return as it is.

参数

- **img** (*np.ndarray*) –Input image.
- **input_order** (*str*) –Whether the input order is 'HWC' or 'CHW'. If the input image shape is (h, w), input_order will not have effects. Default: 'HWC'.

返回 Reordered image.

返回类型 np.ndarray

mmagic.utils.**tensor2img**(*tensor*, *out_type=np.uint8*, *min_max=(0, 1)*)

Convert torch Tensors into image numpy arrays.

After clamping to (min, max), image values will be normalized to [0, 1].

For different tensor shapes, this function will have different behaviors:

1. **4D mini-batch Tensor of shape (N x 3/1 x H x W):** Use *make_grid* to stitch images in the batch dimension, and then convert it to numpy array.
2. **3D Tensor of shape (3/1 x H x W) and 2D Tensor of shape (H x W):** Directly change to numpy array.

Note that the image channel in input tensors should be RGB order. This function will convert it to cv2 convention, i.e., (H x W x C) with BGR order.

参数

- **tensor** (*Tensor | list[Tensor]*) –Input tensors.
- **out_type** (*numpy type*) –Output types. If np.uint8, transform outputs to uint8 type with range [0, 255]; otherwise, float type with range [0, 1]. Default: np.uint8.
- **min_max** (*tuple*) –min and max values for clamp.

返回 3D ndarray of shape (H x W x C) or 2D ndarray of shape (H x W).

返回类型 (Tensor | list[Tensor])

mmagic.utils.**to_numpy**(*img*, *dtype=np.float64*)

Convert data into numpy arrays of dtype.

参数

- **img** (*Tensor | np.ndarray*) –Input data.

- **dtype** (*np.dtype*) –Set the data type of the output. Default: np.float64

返回 Converted numpy arrays data.

返回类型 img (np.ndarray)

mmagic.utils.**MMAGIC_CACHE_DIR**

mmagic.utils.**download_from_url**(*url*, *dest_path=None*, *dest_dir=MMAGIC_CACHE_DIR*, *hash_prefix=None*)

Download object at the given URL to a local path.

参数

- **url** (*str*) –URL of the object to download.
- **dest_path** (*str*) –Path where object will be saved.
- **dest_dir** (*str*) –The directory of the destination. Defaults to `'~/.cache/openmmlab/mmagic/'`.
- **hash_prefix** (*string, optional*) –If not None, the SHA256 downloaded file should start with *hash_prefix*. Default: None.

返回 path for the downloaded file.

返回类型 str

mmagic.utils.**print_colored_log**(*msg*, *level=logging.INFO*, *color='magenta'*)

Print colored log with default logger.

参数

- **msg** (*str*) –Message to log.
- **level** (*int*) –The root logger level. Note that only the process of rank 0 is affected, while other processes will set the level to "Error" and be silent most of the time.Log level, default to 'info'.
- **color** (*str, optional*) –Color 'magenta'.

mmagic.utils.**get_sampler**(*sample_kwargs: dict*, *runner: Optional[mmengine.runner.Runner]*)

Get a sampler to loop input data.

参数

- **sample_kwargs** (*dict*) –_description_
- **runner** (*Optional[Runner]*) –_description_

返回 _description_

返回类型 _type_

mmagic.utils.**register_all_modules**(*init_default_scope: bool = True*) → None

Register all modules in mmagic into the registries.

参数 **init_default_scope** (*bool*) –Whether initialize the mmagic default scope. When *init_default_scope=True*, the global default scope will be set to *mmagic*, and all registries will build modules from mmagic's registry node. To understand more about the registry, please refer to https://mmengine.readthedocs.io/en/latest/advanced_tutorials/registry.html Defaults to True.

mmagic.utils.**try_import**(*name: str*) → Optional[types.ModuleType]

Try to import a module.

参数 **name** (*str*) –Specifies what module to import in absolute or relative terms (e.g. either pkg.mod or ..mod).

返回 If importing successfully, returns the imported module, otherwise returns None.

返回类型 ModuleType or None

mmagic.utils.**add_gaussian_noise**(*img: numpy.ndarray*, *mu*, *sigma*)

Add Gaussian Noise on the input image.

参数

- **img** (*np.ndarray*) –Input image.
- **mu** (*float*) –The mu value of the Gaussian function.
- **sigma** (*float*) –The sigma value of the Gaussian function.

返回 Gaussian noisy output image.

返回类型 noisy_img (np.ndarray)

mmagic.utils.**adjust_gamma**(*image*, *gamma=1*, *gain=1*)

Performs Gamma Correction on the input image.

This function is adopted from skimage: https://github.com/scikit-image/scikit-image/blob/7e4840bd9439d1dfb6beaf549998452c99f97fdd/skimage/exposure/ exposure.py#L439-L494

Also known as Power Law Transform. This function transforms the input image pixelwise according to the equation `O = I**gamma` after scaling each pixel to the range 0 to 1.

参数

- **image** (*np.ndarray*) –Input image.
- **gamma** (*float, optional*) –Non negative real number. Defaults to 1.
- **gain** (*float, optional*) –The constant multiplier. Defaults to 1.

返回 Gamma corrected output image.

返回类型 np.ndarray

mmagic.utils.**bbox2mask**(*img_shape*, *bbox*, *dtype='uint8'*)

Generate mask in np.ndarray from bbox.

The returned mask has the shape of (h, w, 1). '1' indicates the hole and '0' indicates the valid regions.

We prefer to use *uint8* as the data type of masks, which may be different from other codes in the community.

参数

- **img_shape** (*tuple[int]*) –The size of the image.
- **bbox** (*tuple[int]*) –Configuration tuple, (top, left, height, width)
- **np.dtype** (*str*) –Indicate the data type of returned masks. Default: 'uint8'

返回 Mask in the shape of (h, w, 1).

返回类型 mask (np.ndarray)

mmagic.utils.**brush_stroke_mask**(*img_shape*, *num_vertices=(4, 12)*, *mean_angle=2 * math.pi / 5*, *angle_range=2 * math.pi / 15*, *brush_width=(12, 40)*, *max_loops=4*, *dtype='uint8'*)

Generate free-form mask.

The method of generating free-form mask is in the following paper: Free-Form Image Inpainting with Gated Convolution.

When you set the config of this type of mask. You may note the usage of *np.random.randint* and the range of *np.random.randint* is [left, right).

We prefer to use *uint8* as the data type of masks, which may be different from other codes in the community.

TODO: Rewrite the implementation of this function.

参数

- **img_shape** (*tuple[int]*) –Size of the image.
- **num_vertices** (*int | tuple[int]*) –Min and max number of vertices. If only give an integer, we will fix the number of vertices. Default: (4, 12).
- **mean_angle** (*float*) –Mean value of the angle in each vertex. The angle is measured in radians. Default: 2 * math.pi / 5.
- **angle_range** (*float*) –Range of the random angle. Default: 2 * math.pi / 15.
- **brush_width** (*int | tuple[int]*) –(min_width, max_width). If only give an integer, we will fix the width of brush. Default: (12, 40).
- **max_loops** (*int*) –The max number of for loops of drawing strokes. Default: 4.
- **np.dtype** (*str*) –Indicate the data type of returned masks. Default: 'uint8'.

返回 Mask in the shape of (h, w, 1).

返回类型 mask (np.ndarray)

mmagic.utils.**get_irregular_mask**(*img_shape*, *area_ratio_range=(0.15, 0.5)*, ***kwargs*)

Get irregular mask with the constraints in mask ratio.

参数

- **img_shape** (*tuple[int]*) –Size of the image.
- **area_ratio_range** (*tuple(float)*) –Contain the minimum and maximum area
- **Default** (*ratio.*) –(0.15, 0.5).

返回 Mask in the shape of (h, w, 1).

返回类型 mask (np.ndarray)

mmagic.utils.**make_coord**(*shape*, *ranges=None*, *flatten=True*)

Make coordinates at grid centers.

参数

- **shape** (*tuple*) –shape of image.
- **ranges** (*tuple*) –range of coordinate value. Default: None.
- **flatten** (*bool*) –flatten to (n, 2) or Not. Default: True.

返回 coordinates.

返回类型 coord (Tensor)

mmagic.utils.**random_bbox**(*img_shape*, *max_bbox_shape*, *max_bbox_delta=40*, *min_margin=20*)

Generate a random bbox for the mask on a given image.

In our implementation, the max value cannot be obtained since we use *np.random.randint*. And this may be different with other standard scripts in the community.

参数

- **img_shape** (*tuple[int]*) –The size of a image, in the form of (h, w).
- **max_bbox_shape** (*int | tuple[int]*) –Maximum shape of the mask box, in the form of (h, w). If it is an integer, the mask box will be square.
- **max_bbox_delta** (*int | tuple[int]*) –Maximum delta of the mask box, in the form of (delta_h, delta_w). If it is an integer, delta_h and delta_w will be the same. Mask shape will be randomly sampled from the range of *max_bbox_shape - max_bbox_delta* and *max_bbox_shape*. Default: (40, 40).
- **min_margin** (*int | tuple[int]*) –The minimum margin size from the edges of mask box to the image boarder, in the form of (margin_h, margin_w). If it is an integer, margin_h and margin_w will be the same. Default: (20, 20).

返回 The generated box, (top, left, h, w).

返回类型 tuple[int]

mmagic.utils.**random_choose_unknown**(*unknown*, *crop_size*)

Randomly choose an unknown start (top-left) point for a given crop_size.

参数

- **unknown** (*np.ndarray*) –The binary unknown mask.
- **crop_size** (*tuple[int]*) –The given crop size.

返回 The top-left point of the chosen bbox.

返回类型 tuple[int]

mmagic.utils.**ConfigType**

mmagic.utils.**ForwardInputs**

mmagic.utils.**LabelVar**

mmagic.utils.**NoiseVar**

mmagic.utils.**SampleList**

CHAPTER 84

概览

本节将从以下几个方面介绍如何从 MMEditing 0.x 迁移至 MMagic 1.x：

- 概览
  - 新依赖项
  - 总体结构
  - 其他配置设置

84.1 新依赖项

MMagic 1.x 依赖于一些新的包，您可以按照安装教程准备一个新的干净环境并重新安装。

84.2 总体结构

我们在 MMagic 1.x 中对总体结构进行了重构，具体如下：

- 旧版本 MMEdit 中的 `core` 被拆分为 `engine`、`evaluation`、`structures` 和 `visualization`
- 旧版本 MMEdit 中 `datasets` 的 `pipelines` 被重构为 `transforms`
- MMagic 1.x 中的 `models` 被重构为六个部分：`archs`、`base_models`、`data_preprocessors`、`editors`、`diffusion_schedulers` 和 `losses`。

84.3 其他配置设置

我们将配置文件重命名为新模板:`{model_settings}_{module_setting}_{training_setting}_{datasets_inf`

更多配置细节请参见配置指南。

CHAPTER 85

运行设置的迁移

我们更新了 MMagic 1.x 中的运行设置，重要修改如下：

- `checkpoint_config` 被移动到 `default_hooks.checkpoint`，`log_config` 被移动到 `default_hooks.logger`。我们将许多 hooks 设置从脚本代码移动到运行配置的 `default_hooks` 字段中。
- `resume_from` 被移除，使用 `resume` 替代它。
  - 如果 resume=True 并且 load_from 不是 None, 则从 load_from 中的检查点恢复训练。
  - 如果 resume=True 且 load_from 为 None，则尝试从工作目录中的最新检查点恢复。
  - 如果 resume=False 且 load_from 不为 None，则仅加载检查点，不恢复训练。
  - 如果 resume=False 且 load_from 为 None，则不加载也不恢复。
- `dist_params` 字段现在是 `env_cfg` 的一个子字段。并且在 `env_cfg` 还有一些新的配置。
- `workflow` 相关功能已被删除。
- 新字段 `visualizer`: 可视化工具是一个新设计。在 runner 中使用可视化器实例来处理结果和日志可视化并保存到不同的后端，例如 Local、TensorBoard 和 Wandb。
- 新字段 `default_scope`: 所有注册器搜索 module 的起点。

```
checkpoint_config = dict(  # 设置检查点 hook 的配置, 参考 https://github.com/open-
↪mmlab/mmcv/blob/master/mmcv/runner/hooks/checkpoint.py 完成的
    interval=5000,  # 保存间隔为 5000 次迭代
    save_optimizer=True,  # 也保存优化器
```

(下页继续)

(续上页)

```
    by_epoch=False)  # 通过 iterations 计数
log_config = dict(  # 注册日志 hook 的配置
    interval=100,  # 打印日志的间隔
    hooks=[
        dict(type='TextLoggerHook', by_epoch=False),  # logger 用来记录训练过程
        dict(type='TensorboardLoggerHook'),  # 也支持 Tensorboard logger
    ])
visual_config = None  # 可视化配置，我们不使用它。
# runtime settings
dist_params = dict(backend='nccl')  # 设置分布式训练的参数，还可以设置端口
log_level = 'INFO'  # 日志等级
load_from = None # 从指定路径加载预训练模型，这不会恢复训练
resume_from = None # 从给定路径恢复检查点，训练将从保存检查点的epoch开始恢复
workflow = [('train', 1)]  # Runner 的工作流程. [('train', 1)]␣
↪意味着只有一个工作流，并且名为 “train” 的工作流执行一次。␣
↪在训练当前的抠图模型时，请保持此项不变
```

```
default_hooks = dict(  # 用来创建默认 hooks
    checkpoint=dict(  # 设置 checkpoint hook 的配置
        type='CheckpointHook',
        interval=5000,  # 保存间隔为5000次迭代
        save_optimizer=True,
        by_epoch=False,  # 通过 iterations 计数
        out_dir=save_dir,
    ),
    timer=dict(type='IterTimerHook'),
    logger=dict(type='LoggerHook', interval=100),  # 注册 logger hook 的配置
    param_scheduler=dict(type='ParamSchedulerHook'),
    sampler_seed=dict(type='DistSamplerSeedHook'),
)
default_scope = 'mmedit' # 用来设置注册位置
env_cfg = dict(  # 设置分布式训练的参数，还可以设置端口
    cudnn_benchmark=False,
    mp_cfg=dict(mp_start_method='fork', opencv_num_threads=4),
    dist_cfg=dict(backend='nccl'),
)
log_level = 'INFO'  # 日志等级
log_processor = dict(type='LogProcessor', window_size=100, by_epoch=False)  #␣
↪用来创建日志处理器
load_from = None  # 从指定路径加载预训练模型，这不会恢复训练
resume = False  # 从给定路径恢复检查点，训练将从保存检查点的epoch开始恢复
```

CHAPTER 86

模型的迁移

我们在 MMagic 1.x. 版本更新了模型设定，其中重要的改动如下所示：

- 删除 `pretrained` 字段.
- 在模型设定中添加 `train_cfg` 和 `test_cfg` 字段.
- 添加 `data_preprocessor` 字段. 这里主要是将归一化和颜色空间转换操作从 `dataset transform` 流程中移动到 `data_preprocessor` 中. 我们接下来会介绍 `data_preprocessor`.

```
model = dict(
    type='BasicRestorer',  # Name of the model
    generator=dict(  # Config of the generator
        type='EDSR',  # Type of the generator
        in_channels=3,  # Channel number of inputs
        out_channels=3,  # Channel number of outputs
        mid_channels=64,  # Channel number of intermediate features
        num_blocks=16,  # Block number in the trunk network
        upscale_factor=scale, # Upsampling factor
        res_scale=1,  # Used to scale the residual in residual block
        rgb_mean=(0.4488, 0.4371, 0.4040),  # Image mean in RGB orders
        rgb_std=(1.0, 1.0, 1.0)),  # Image std in RGB orders
    pretrained=None,
    pixel_loss=dict(type='L1Loss', loss_weight=1.0, reduction='mean'))  # Config for␣
↪pixel loss model training and testing settings
```

```
model = dict(
    type='BaseEditModel',  # Name of the model
    generator=dict(  # Config of the generator
        type='EDSRNet',  # Type of the generator
        in_channels=3,  # Channel number of inputs
        out_channels=3,  # Channel number of outputs
        mid_channels=64,  # Channel number of intermediate features
        num_blocks=16,  # Block number in the trunk network
        upscale_factor=scale, # Upsampling factor
        res_scale=1,  # Used to scale the residual in residual block
        rgb_mean=(0.4488, 0.4371, 0.4040),  # Image mean in RGB orders
        rgb_std=(1.0, 1.0, 1.0)),  # Image std in RGB orders
    pixel_loss=dict(type='L1Loss', loss_weight=1.0, reduction='mean')  # Config for␣
↪pixel loss
    train_cfg=dict(),  # Config of training model.
    test_cfg=dict(),  # Config of testing model.
    data_preprocessor=dict(  # The Config to build data preprocessor
        type='DataPreprocessor', mean=[0., 0., 0.], std=[255., 255.,
                                                         255.]))
```

我们在 MMagic 1.x. 版本中对模型进行了重构，其中重要的改动如下所示：

- MMagic 1.x 中的 `models` 被重构为六个部分：`archs`、`base_models`、`data_preprocessors`、`editors`、`diffusion_schedulers` 和 `losses`.
- 在 `models` 中添加了 `data_preprocessor` 模块。这里主要是将归一化和颜色空间转换操作从 `dataset transform` 流程中移动到 `data_preprocessor` 中. 此时，数据流经过数据预处理后，会先经过 `data_preprocessor` 模块的转换，然后再输入到模型中.

模型的更多详细信息请参见模型指南.

CHAPTER 87

评估与测试设置的迁移

我们更新了 MMagic 1.x 中的评估设置，重要修改如下：

- 评估字段被分为 `val_evaluator` 和 `test_evaluator` ，`interval` 被移动到 `train_cfg.val_interval` 。
- 评估指标从 `test_cfg` 移至 `val_evaluator` 和 `test_evaluator`

```
train_cfg = None  # 训练配置字典变量设为 None
test_cfg = dict(  # 测试配置字典变量
    metrics=['PSNR'],  # 测试期间使用的指标 PSNR （峰值信噪比）
    crop_border=scale)  # 评估期间裁剪边框

evaluation = dict(  # 构建评估钩子的配置字典变量
    interval=5000,  # 评价间隔
    save_image=True,  # 评估期间保存图像
    gpu_collect=True)  # 使用 GPU 收集
```

```
val_evaluator = [
    dict(type='PSNR', crop_border=scale),  # 要评估的指标名称
]
test_evaluator = val_evaluator

train_cfg = dict(
    type='IterBasedTrainLoop', max_iters=300000, val_interval=5000)  #␣
↪训练循环类型配置
```

(下页继续)

(续上页)

```
val_cfg = dict(type='ValLoop')  # 验证循环类型的名称
test_cfg = dict(type='TestLoop')  # 测试循环类型的名称
```

我们已将MMGeneration 1.x合并到 MMagic 中. 这里是关于 MMGeneration 的评估和测试设置的迁移。

评估字段分为 `val_evaluator` 和 `test_evaluator` ，并且评估字段不再支持 `interval` 和 `save_best` 参数。

- `interval` 移至 `train_cfg.val_interval`，请参阅调度设置。
- `save_best` 移至 `default_hooks.checkpoint.save_best`。

```
evaluation = dict(
    type='GenerativeEvalHook',
    interval=10000,
    metrics=[
        dict(
            type='FID',
            num_images=50000,
            bgr2rgb=True,
            inception_args=dict(type='StyleGAN')),
        dict(type='IS', num_images=50000)
    ],
    best_metric=['fid', 'is'],
    sample_kwargs=dict(sample_model='ema'))
```

```
val_evaluator = dict(
    type='Evaluator',
    metrics=[
        dict(
            type='FID',
            prefix='FID-Full-50k',
            fake_nums=50000,
            inception_style='StyleGAN',
            sample_model='orig'),
        dict(
            type='IS',
            prefix='IS-50k',
            fake_nums=50000)])
# 设置最佳配置
default_hooks = dict(
    checkpoint=dict(
        type='CheckpointHook',
        interval=10000,
        by_epoch=False,
```

(下页继续)

(续上页)

```
        less_keys=['FID-Full-50k/fid'],
        greater_keys=['IS-50k/is'],
        save_optimizer=True,
        save_best=['FID-Full-50k/fid', 'IS-50k/is'],
        rule=['less', 'greater']))
test_evaluator = val_evaluator
```

为了正确评估和测试模型，我们需要在 `val_cfg` 和 `test_cfg` 中设置特定的循环。

```
total_iters = 1000000

runner = dict(
    type='DynamicIterBasedRunner',
    is_dynamic_ddp=False,
    pass_training_status=True)
```

```
train_cfg = dict(
    by_epoch=False,  # 使用基于迭代的训练
    max_iters=1000000,  # 最大训练迭代次数
    val_begin=1,
    val_interval=10000)  # 评价间隔
val_cfg = dict(type='MultiValLoop')  # 验证中的特定循环
test_cfg = dict(type='MultiTestLoop')  # 测试中的特定循环
```

CHAPTER 88

调度器的迁移

我们更新了 MMagic 1.x 中的调度器设置，重要修改如下:

- 现在我们使用 `optim_wrapper` 字段来指定关于优化过程的所有配置。`optimizer` 字段现在是 `optim_wrapper` 的一个子字段。
- `lr_config` 字段被移除，我们使用新的 `param_scheduler` 来代替它。
- `total_iters` 字段已移至 `train_cfg`，作为 `max_iters`, `val_cfg` 和 `test_cfg`，用于配置训练、验证和测试中的循环。

```
optimizers = dict(generator=dict(type='Adam', lr=1e-4, betas=(0.9, 0.999)))    #␣
↪用于构建优化器的配置，支持 PyTorch 中的所有优化器，其参数与 PyTorch 中的参数相同。
total_iters = 300000 # 总训练迭代次数
lr_config = dict( # 用于注册 LrUpdater hook 的学习率调度器配置
    policy='Step', by_epoch=False, step=[200000], gamma=0.5)  # 调度器的策略
```

```
optim_wrapper = dict(
    dict(
        type='OptimWrapper',
        optimizer=dict(type='Adam', lr=1e-4),
    )
)  # 用于构建优化器的配置，支持 PyTorch 中的所有优化器，其参数与 PyTorch␣
↪中的参数相同。
param_scheduler = dict(  # 学习策略的配置
    type='MultiStepLR', by_epoch=False, milestones=[200000], gamma=0.5)  #␣
↪调度器的策略
```

(下页继续)

(续上页)

```
train_cfg = dict(
    type='IterBasedTrainLoop', max_iters=300000, val_interval=5000)  #␣
↪训练循环类型的配置
val_cfg = dict(type='ValLoop')  # 验证循环类型的名称
test_cfg = dict(type='TestLoop')  # 测试循环类型的名称
```

有关调度器设置的更多详细信息可在 MMEngine Documents 中找到。

CHAPTER 89

Data Settings 的迁移

本篇文档负责介绍 data settings 的迁移方式：

- [Data Settings 的迁移](#Data Settings 的迁移)
  - *Data Pipelines*
  - *Dataloader*

89.1 Data Pipelines

在 MMagic 1.x 中我们更新了 data pipeline 的设置，有以下几个重要的修改：

- 去除了 `normalization` 和 `color space` 两种数据变换操作，并将它们移动到了 `data_preprocessor` 部分。
- 原版本中格式化数据变换 pipeline 的 `Collect` 和 `ToTensor` 在新版本中被整合为 `PackInputs`。更多的细节可以在数据变换文档 中查看。

```
train_pipeline = [  # Train pipeline
    dict(type='LoadImageFromFile',  # 从文件读取图片
        io_backend='disk',  # io backend
        key='lq',  # 找到结果对应路径的 keys
        flag='unchanged'),  # 读取图片的 flag
    dict(type='LoadImageFromFile',  # 从文件读取图片
        io_backend='disk',  # io backend
```

(下页继续)

(续上页)

```
        key='gt',  # 找到结果对应路径的 keys
        flag='unchanged'),  # 读取图片的 flag
    dict(type='RescaleToZeroOne', keys=['lq', 'gt']),  # 将图片从 [0, 255] 缩放到 [0,
↪1]
    dict(type='Normalize',  # normalize 图片的 augmentation pipeline
        keys=['lq', 'gt'],  # 需要 normalized 的图片
        mean=[0, 0, 0],  # 平均值
        std=[1, 1, 1],  # 标准差
        to_rgb=True),  # 是否转换到 rgb 通道
    dict(type='PairedRandomCrop', gt_patch_size=96),  # PairedRandomCrop
    dict(type='Flip',  # 翻转图片
        keys=['lq', 'gt'],  # 需要翻转的图片
        flip_ratio=0.5,  # 翻转概率
        direction='horizontal'),  # Flip 方向
    dict(type='Flip',  # Flip 图片
        keys=['lq', 'gt'],  # 需要翻转的图片
        flip_ratio=0.5,  # Flip ratio
        direction='vertical'),  # Flip 方向
    dict(type='RandomTransposeHW',  # 随即对图片的高和宽转置
        keys=['lq', 'gt'],  # 需要 transpose 的图片
        transpose_ratio=0.5  # Transpose ratio
        ),
    dict(type='Collect',  # Pipeline that decides which keys in the data should be
↪passed to the model
        keys=['lq', 'gt'],  # Keys to pass to the model
        meta_keys=['lq_path', 'gt_path']), # Meta information keys. 训练时 meta
↪information 不是必须的
    dict(type='ToTensor',  # 图片转为 tensor
        keys=['lq', 'gt'])  # 需要转换为 tensor 的图片
]
test_pipeline = [  # Test pipeline
    dict(
        type='LoadImageFromFile',  # 从文件读取图片
        io_backend='disk',  # io backend
        key='lq', # 找到结果对应路径的 keys
        flag='unchanged'),  # flag for reading images
    dict(
        type='LoadImageFromFile',  # 从文件读取图片
        io_backend='disk',  # io backend
        key='gt', # 找到结果对应路径的 keys
        flag='unchanged'),  # flag for reading images
    dict(type='RescaleToZeroOne', keys=['lq', 'gt']),  # 将图片从 [0, 255] 缩放到 [0,
↪1]
```

(下页继续)

(续上页)

```
    dict(
        type='Normalize',  # 对输入图片执行 normalization 的数据增强 pipeline
        keys=['lq', 'gt'],  # 需要 normalized 图片
        mean=[0, 0, 0],  # Mean values
        std=[1, 1, 1],  # Standard variance
        to_rgb=True),  # 是否转为 rgb 格式
    dict(type='Collect',  # Pipeline that decides which keys in the data should be␣
↪passed to the model
        keys=['lq', 'gt'],  # Keys to pass to the model
        meta_keys=['lq_path', 'gt_path']),  # Meta information keys
    dict(type='ToTensor',  # 图片转为 tensor
        keys=['lq', 'gt'])  # 需要转换为 tensor 的图片
]
```

```
train_pipeline = [  # train pipeline
    dict(type='LoadImageFromFile',  # 从文件读取图片
        key='img',   # 找到结果对应路径的 keys
        color_type='color',  # 图片的 color type
        channel_order='rgb',  # 图片的 channel 顺序
        imdecode_backend='cv2'),  # decode backend
    dict(type='LoadImageFromFile',  # 从文件读取图片
        key='gt',   # 找到结果对应路径的 keys
        color_type='color',  # 图片的 color type
        channel_order='rgb',  # 图片的 channel 顺序
        imdecode_backend='cv2'),  # decode backend
    dict(type='SetValues', dictionary=dict(scale=scale)),  # 设置 destination keys
    dict(type='PairedRandomCrop', gt_patch_size=96),  # PairedRandomCrop
    dict(type='Flip',  # 翻转图片
        keys=['lq', 'gt'],  # 需要翻转的图片
        flip_ratio=0.5,  # Flip ratio
        direction='horizontal'),  # Flip 方向
    dict(type='Flip',  # Flip images
        keys=['lq', 'gt'],  # 需要翻转的图片
        flip_ratio=0.5,  # Flip ratio
        direction='vertical'),  # Flip 方向
    dict(type='RandomTransposeHW',  # 随即对图片的高和宽进行转置
        keys=['lq', 'gt'], # 需要转置的图片
        transpose_ratio=0.5  # Transpose ratio
        ),
    dict(type='PackInputs')  # 在当前 pipeline 中收集数据的设置
]
test_pipeline = [  # Test pipeline
    dict(type='LoadImageFromFile',   # 从文件读取图片
```

(下页继续)

(续上页)

```
        key='img',   # 找到结果对应路径的 keys
        color_type='color',  # 图片的 color type
        channel_order='rgb',  # 图片的 channel order
        imdecode_backend='cv2'),  # decode backend
    dict(type='LoadImageFromFile',   # 从文件读取图片
        key='gt',  # 找到结果对应路径的 keys
        color_type='color', # 图片的 color type
        channel_order='rgb',  # 图片的 channel order
        imdecode_backend='cv2'),  # decode backend
    dict(type='PackInputs')  # 在当前 pipeline 中收集数据的设置
]
```

89.2 Dataloader

在 MMagic 1.x 中我们更新了 dataloader 的设置方式，有以下几个重要的修改：

- 原版本中的 `data` 字段分为了 `train_dataloader` ， `val_dataloader` 和 `test_dataloader` 三个独立的部分。这样我们就可以细粒度的对各部分进行配置。例如用户就可以针对训练和测试制定不同的 sampler 和 batch size 。
- `samples_per_gpu` 更名为 `batch_size` 。
- `workers_per_gpu` 更名为 `num_workers` 。

```
data = dict(
    # train
    samples_per_gpu=16,  # 每个 GPU 上的 batch_size
    workers_per_gpu=4,  # 每个 GPU 上做 pre-fetch 的 worker 数
    drop_last=True,  # 在 data_loader 中使用 drop_last
    train=dict(  # Train dataset 配置
        type='RepeatDataset',  # 对 iter-based 模型设置为 RepeatDataset
        times=1000,  # RepeatDataset 的 repeated times 参数
        dataset=dict(
            type=train_dataset_type,  # 数据集类型
            lq_folder='data/DIV2K/DIV2K_train_LR_bicubic/X2_sub',  # lq 的文件路径
            gt_folder='data/DIV2K/DIV2K_train_HR_sub',  # ground truth 的文件路径
            ann_file='data/DIV2K/meta_info_DIV2K800sub_GT.txt',  # 标注文件的路径
            pipeline=train_pipeline,  # 参照 train_pipeline
            scale=scale)),  # Upsampling 的 scale factor
    # validation
    val_samples_per_gpu=1,  # validation 时每个 GPU 上的 batch_size
    val_workers_per_gpu=4,  # validation 是每个 GPU 上做 pre-fetch 的 worker 数
    val=dict(
```

(下页继续)

(续上页)

```
        type=val_dataset_type,  # 数据集类型
        lq_folder='data/val_set5/Set5_bicLRx2',  # lq 的文件路径
        gt_folder='data/val_set5/Set5_mod12',  # ground truth 的文件路径
        pipeline=test_pipeline,  # 参照 test_pipeline
        scale=scale,  # Upsampling 的 scale factor
        filename_tmpl='{}'),  # filename 模板
    # test
    test=dict(
        type=val_dataset_type,  # 数据集类型
        lq_folder='data/val_set5/Set5_bicLRx2', # lq 的文件路径
        gt_folder='data/val_set5/Set5_mod12',  # ground truth 的文件路径
        pipeline=test_pipeline,  # 参照 test_pipeline
        scale=scale,  # Upsampling 的 scale factor
        filename_tmpl='{}'),  # filename 模板
)
```

```
dataset_type = 'BasicImageDataset'  # 数据集类型
data_root = 'data'  # 数据集根目录
train_dataloader = dict(
    batch_size=16,
    num_workers=4,  # 每个 GPU 上做 pre-fetch 的 worker 数
    persistent_workers=False,  # 是否保持 workers instance 存活
    sampler=dict(type='InfiniteSampler', shuffle=True),  # data sampler 类型
    dataset=dict(  # 训练数据集 config
        type=dataset_type,  # 数据集类型
        ann_file='meta_info_DIV2K800sub_GT.txt',  # 标注文件路径
        metainfo=dict(dataset_type='div2k', task_name='sisr'),
        data_root=data_root + '/DIV2K',  # 数据根目录
        data_prefix=dict(  # 图像文件前缀
            img='DIV2K_train_LR_bicubic/X2_sub', gt='DIV2K_train_HR_sub'),
        filename_tmpl=dict(img='{}', gt='{}'),  # Filename 模板
        pipeline=train_pipeline))
val_dataloader = dict(
    batch_size=1,
    num_workers=4,  # 每个 GPU 上做 pre-fetch 的 worker 数
    persistent_workers=False,  # 是否保持 workers instance 存活
    drop_last=False,  # 是否丢弃最后未完成的 batch
    sampler=dict(type='DefaultSampler', shuffle=False),  # data sampler 类型
    dataset=dict(  # Validation 数据集设置
        type=dataset_type,  # 数据集类型
        metainfo=dict(dataset_type='set5', task_name='sisr'),
        data_root=data_root + '/Set5',  # 数据根目录
        data_prefix=dict(img='LRbicx2', gt='GTmod12'),  # 图像文件前缀
```

(下页继续)

(续上页)

```
        pipeline=test_pipeline))
test_dataloader = val_dataloader
```

CHAPTER 90

分布式训练的迁移

我们已经将MMGeneration 1.x合并至 MMagic。以下是针对 MMGeneration 中分布式训练的迁移事项。

在 0.x 版中，MMGeneration 使用 `DDPWrapper` 和 `DynamicRunner` 来训练对应的静态和动态模型（例如 PGGAN 和 StyleGANv2），但在 1.x 版中，我们使用 MMEngine 提供的 `MMSeparateDistributedDataParallel` 来实现分布式训练。

如下是配置前后对比：

```
# 使用DDPWrapper
use_ddp_wrapper = True
find_unused_parameters = False

runner = dict(
    type='DynamicIterBasedRunner',
    is_dynamic_ddp=False)
```

```
model_wrapper_cfg = dict(
    type='MMSeparateDistributedDataParallel',
    broadcast_buffers=False,
    find_unused_parameters=False)
```

```
use_ddp_wrapper = False
find_unused_parameters = False

# 使用DynamicRunner
```

(下页继续)

(续上页)

```
runner = dict(
    type='DynamicIterBasedRunner',
    is_dynamic_ddp=True)
```

```
model_wrapper_cfg = dict(
    type='MMSeparateDistributedDataParallel',
    broadcast_buffers=False,
    find_unused_parameters=True) # 针对动态模型，设置`find_unused_
↪parameters`标志为True
```

CHAPTER 91

优化器的迁移

我们已经将MMGeneration 1.x合并至 MMagic。以下是针对 MMGeneration 中优化器的迁移事项。

在 0.x 版中，MMGeneration 使用 PyTorch 自带的优化器，其只提供了通用参数优化，而在 1.x 版中，我们则使用了 MMEngine 提供的 `OptimizerWrapper`。

对比 PyTorch 自带的 `Optimizer`，`OptimizerWrapper` 可以支持如下功能：

- `OptimizerWrapper.update_params` 在一个单一的函数中就实现了 `zero_grad`，`backward` 和 `step`
- 支持梯度自动累积
- 提供一个名为 `OptimizerWrapper.optim_context` 的上下文管理器来封装前向进程，`optim_context` 会根据当前更新迭代数目来自动调用 `torch.no_sync`，在 AMP(Auto Mixed Precision) 训练中，`autocast` 也会在 `optim_context` 中被调用。

对 GAN 模型，生成器和鉴别器采用不同的优化器和训练策略。要使 GAN 模型的 `train_step` 函数签名和其它模型的保持一致，我们使用从 `OptimizerWrapper` 继承下来的 `OptimWrapperDict` 来封装生成器和鉴别器的优化器，为了便于该流程的自动化 MMagic 实现了 `MultiOptimWrapperContructor` 构造器。如你想训练 GAN 模型，那么应该在你的配置中指定该构造器。

如下是 0.x 版和 1.x 版的配置对比

```
optimizer = dict(
    generator=dict(type='Adam', lr=0.0001, betas=(0.0, 0.999), eps=1e-6),
    discriminator=dict(type='Adam', lr=0.0004, betas=(0.0, 0.999), eps=1e-6))
```

```
optim_wrapper = dict(
    constructor='MultiOptimWrapperConstructor',
    generator=dict(optimizer=dict(type='Adam', lr=0.0002, betas=(0.0, 0.999), eps=1e-
↪6)),
    discriminator=dict(
        optimizer=dict(type='Adam', lr=0.0004, betas=(0.0, 0.999), eps=1e-6)))
```

注意，在 1.x 版中，MMGeneration 使用 `OptimWrapper` 来实现梯度累加，这就会导致在 0.x 版和 1.x 版之间，`discriminator_steps` 配置（用于在多次更新鉴别器之后更新一次生成器的训练技巧）与梯度累加均出现不一致问题。

- 在 0.x 版中，我们在配置里使用 `disc_steps`，`gen_steps` 和 `batch_accumulation_steps`。`disc_steps` 和 `batch_accumulation_steps` 会根据 `train_step` 的调用次数来进行统计（亦即 dataloader 中数据的读取次数）。因此鉴别器的一段连续性更新次数为 `disc_steps // batch_accumulation_steps`。且对于生成器，`gen_steps` 是生成器实际的一段连续性更新次数
- 但在 1.x 版中，我们在配置里则使用了 `discriminator_steps`，`generator_steps` 和 `accumulative_counts`。`discriminator_steps` 和 `generator_steps` 指的是自身在更新其它模型之前的一段连续性的更新次数以 BigGAN-128 配置为例。

```
model = dict(
    type='BasiccGAN',
    generator=dict(
        type='BigGANGenerator',
        output_scale=128,
        noise_size=120,
        num_classes=1000,
        base_channels=96,
        shared_dim=128,
        with_shared_embedding=True,
        sn_eps=1e-6,
        init_type='ortho',
        act_cfg=dict(type='ReLU', inplace=True),
        split_noise=True,
        auto_sync_bn=False),
    discriminator=dict(
        type='BigGANDiscriminator',
        input_scale=128,
        num_classes=1000,
        base_channels=96,
        sn_eps=1e-6,
        init_type='ortho',
        act_cfg=dict(type='ReLU', inplace=True),
        with_spectral_norm=True),
    gan_loss=dict(type='GANLoss', gan_type='hinge'))
```

(下页继续)

(续上页)

```
# 连续性更新鉴别器`disc_steps // batch_accumulation_steps = 8 // 8 = 1`次
# 连续性更新生成器`gen_steps = 1`次
# 生成器与鉴别器在每次更新之前执行`batch_accumulation_steps = 8`次梯度累加
train_cfg = dict(
    disc_steps=8, gen_steps=1, batch_accumulation_steps=8, use_ema=True)
```

```
model = dict(
    type='BigGAN',
    num_classes=1000,
    data_preprocessor=dict(type='DataPreprocessor'),
    generator=dict(
        type='BigGANGenerator',
        output_scale=128,
        noise_size=120,
        num_classes=1000,
        base_channels=96,
        shared_dim=128,
        with_shared_embedding=True,
        sn_eps=1e-6,
        init_type='ortho',
        act_cfg=dict(type='ReLU', inplace=True),
        split_noise=True,
        auto_sync_bn=False),
    discriminator=dict(
        type='BigGANDiscriminator',
        input_scale=128,
        num_classes=1000,
        base_channels=96,
        sn_eps=1e-6,
        init_type='ortho',
        act_cfg=dict(type='ReLU', inplace=True),
        with_spectral_norm=True),
    # 连续性更新鉴别器`discriminator_steps = 1`次
    # 连续性更新生成器`generator_steps = 1`次
    generator_steps=1,
    discriminator_steps=1)

optim_wrapper = dict(
    constructor='MultiOptimWrapperConstructor',
    generator=dict(
        # 生成器在每次更新之前执行`accumulative_counts = 8`次梯度累加
        accumulative_counts=8,
```

(下页继续)

(续上页)

```
        optimizer=dict(type='Adam', lr=0.0001, betas=(0.0, 0.999), eps=1e-6)),
    discriminator=dict(
        # 鉴别器在每次更新之前执行`accumulative_counts = 8`次梯度累加
        accumulative_counts=8,
        optimizer=dict(type='Adam', lr=0.0004, betas=(0.0, 0.999), eps=1e-6)))
```

CHAPTER 92

可视化的迁移

在 0.x 版中，MMEditing 使用 `VisualizationHook` 来对训练过程中生成的结果进行可视化，在 1.x 版中，我们将该功能整合到 `BasicVisualizationHook` / `VisualizationHook` 中，而且遵循 MMEngine 的设计，我们实现了 `ConcatImageVisualizer` / `Visualizer` 和一系列 `VisBackend` 来绘制和保存可视化结果。

```
visual_config = dict(
    type='VisualizationHook',
    output_dir='visual',
    interval=1000,
    res_name_list=['gt_img', 'masked_img', 'fake_res', 'fake_img'],
)
```

```
vis_backends = [dict(type='LocalVisBackend')]
visualizer = dict(
    type='ConcatImageVisualizer',
    vis_backends=vis_backends,
    fn_key='gt_path',
    img_keys=['gt_img', 'input', 'pred_img'],
    bgr2rgb=True)
custom_hooks = [dict(type='BasicVisualizationHook', interval=1)]
```

要了解更多关于可视化的功能，请参阅这个教程。

CHAPTER 93

混合精度训练的迁移

在 0.x 版中，MMEditing 并不支持对整体前向过程的混合精度训练。相反，用户必须使用 `auto_fp16` 装饰器来适配特定子模块，然后再将子模块的参数转化成 fp16。这样就可以拥有对模型参数的更细粒度的控制，但是该方法使用起来很繁琐，而且用户需要自己处理一些操作，比如训练过程中损失函数的缩放

MMagic 1.x 版使用了 MMEngine 提供的 `AmpOptimWrapper`，在 `AmpOptimWrapper.update_params` 中，梯度缩放和 `GradScaler` 更新将被自动执行，且在 `optim_context` 上下文管理其中，`auto_cast` 被应用到整个前向过程中。

具体来说，0.x 版和 1.x 版之间的差异如下所示：

```
# 配置
runner = dict(fp16_loss_scaler=dict(init_scale=512))
```

```
# 代码
import torch.nn as nn
from mmedit.models.builder import build_model
from mmedit.core.runners.fp16_utils import auto_fp16


class DemoModule(nn.Module):
    def __init__(self, cfg):
        self.net = build_model(cfg)

    @auto_fp16
    def forward(self, x):
```

(下页继续)

(续上页)

```
        return self.net(x)

class DemoModel(nn.Module):

    def __init__(self, cfg):
        super().__init__(self)
        self.demo_network = DemoModule(cfg)

    def train_step(self,
                   data_batch,
                   optimizer,
                   ddp_reducer=None,
                   loss_scaler=None,
                   use_apex_amp=False,
                   running_status=None):
        # 从 data_batch 中获取数据
        inputs = data_batch['img']
        output = self.demo_network(inputs)

        optimizer.zero_grad()
        loss, log_vars = self.get_loss(data_dict_)

        if ddp_reducer is not None:
            ddp_reducer.prepare_for_backward(_find_tensors(loss_disc))

        if loss_scaler:
            # 添加 fp16 支持
            loss_scaler.scale(loss_disc).backward()
        elif use_apex_amp:
            from apex import amp
            with amp.scale_loss(loss_disc, optimizer,
                    loss_id=0) as scaled_loss_disc:
                scaled_loss_disc.backward()
        else:
            loss_disc.backward()

        if loss_scaler:
            loss_scaler.unscale_(optimizer)
            loss_scaler.step(optimizer)
        else:
            optimizer.step()
```

```
# 配置
```

(下页继续)

(续上页)

```
optim_wrapper = dict(
    constructor='OptimWrapperConstructor',
    generator=dict(
        accumulative_counts=8,
        optimizer=dict(type='Adam', lr=0.0001, betas=(0.0, 0.999), eps=1e-06),
        type='AmpOptimWrapper',  # 使用 amp 封装器
        loss_scale='dynamic'),
    discriminator=dict(
        accumulative_counts=8,
        optimizer=dict(type='Adam', lr=0.0004, betas=(0.0, 0.999), eps=1e-06),
        type='AmpOptimWrapper',  # 使用 amp 封装器
        loss_scale='dynamic'))
```

```
# 代码
import torch.nn as nn
from mmagic.registry import MODULES
from mmengine.model import BaseModel


class DemoModule(nn.Module):
    def __init__(self, cfg):
        self.net = MODULES.build(cfg)

    def forward(self, x):
        return self.net(x)

class DemoModel(BaseModel):
    def __init__(self, cfg):
        super().__init__(self)
        self.demo_network = DemoModule(cfg)

    def train_step(self, data, optim_wrapper):
        # 从 data_batch 中获取数据
        data = self.data_preprocessor(data, True)
        inputs = data['inputs']

        with optim_wrapper.optim_context(self.discriminator):
            output = self.demo_network(inputs)
        loss_dict = self.get_loss(output)
        # 使用 `BaseModel` 提供的 parse_loss
        loss, log_vars = self.parse_loss(loss_dict)
        optimizer_wrapper.update_params(loss)
```

(下页继续)

(续上页)

```
        return log_vars
```

若要避免用户操作配置文件，MMagic 在 `train.py` 里提供了 `--amp` 选项，其可以让用户在不修改配置文件的情况下启动混合精度训练，用户可以使用以下命令启动混合精度训练：

```
bash tools/dist_train.sh CONFIG GPUS --amp

# 对 slurm 用户
bash tools/slurm_train.sh PARTITION JOB_NAME CONFIG WORK_DIR --amp
```

CHAPTER 94

NPU (华为🅵腾)

94.1 使用方法

首先，请参考MMCV 安装带有 NPU 支持的 MMCV 与 mmengine 。

使用如下命令，可以利用 8 个 NPU 训练模型（以 edsr 为例）：

```
bash tools/dist_train.sh configs/edsr/edsr_x2c64b16_1xb16-300k_div2k.py 8
```

或者，使用如下命令，在一个 NPU 上训练模型（以 edsr 为例）：

```
python tools/train.py configs/edsr/edsr_x2c64b16_1xb16-300k_div2k.py
```

94.2 经过验证的模型

注意:

- 如果没有特别标记，NPU 上的结果与使用 FP32 的 GPU 上的结果相同。

以上所有模型权重及训练日志均由华为🅵腾团队提供

CHAPTER 95

English

CHAPTER 96

简体中文

CHAPTER 97

Indices and tables

- genindex
- modindex
- search

Python 模块索引

m

mmagic.apis.inferencers, 389
mmagic.datasets, 408
mmagic.datasets.transforms, 431
mmagic.engine.hooks, 519
mmagic.engine.optimizers, 529
mmagic.engine.runner, 535
mmagic.engine.schedulers, 541
mmagic.evaluation, 480
mmagic.models.archs, 546
mmagic.models.base_models, 567
mmagic.models.data_preprocessors, 623
mmagic.models.editors, 633
mmagic.models.losses, 596
mmagic.structures, 403
mmagic.utils, 782
mmagic.visualization, 509

索引

符号

_NON_CONCATENATE_KEYS (*mmagic.models.data_preprocessors.DataPreprocessor* 属性), 625

_NON_IMAGE_KEYS (*mmagic.models.data_preprocessors.DataPreprocessor* 属性), 625

_VALID_IMG_SUFFIX (*mmagic.datasets.GrowScaleImgDataset* 属性), 423

__call__() (*mmagic.apis.inferencers.ControlnetAnimationInferencer* 方法), 392

__call__() (*mmagic.datasets.transforms.DegradationsWithShuffle* 方法), 471

__call__() (*mmagic.datasets.transforms.RandomBlur* 方法), 472

__call__() (*mmagic.datasets.transforms.RandomJPEGCompression* 方法), 472

__call__() (*mmagic.datasets.transforms.RandomNoise* 方法), 473

__call__() (*mmagic.datasets.transforms.RandomResize* 方法), 473

__call__() (*mmagic.datasets.transforms.RandomVideoCompression* 方法), 474

__call__() (*mmagic.engine.optimizers.MultiOptimWrapperConstructor* 方法), 532

__call__() (*mmagic.engine.optimizers.PGGANOptimWrapperConstructor* 方法), 533

__call__() (*mmagic.engine.optimizers.SinGANOptimWrapperConstructor* 方法), 534

__call__() (*mmagic.models.archs.TokenizerWrapper* 方法), 562

__getattr__() (*mmagic.models.archs.TokenizerWrapper* 方法), 561

__getitem__() (*mmagic.datasets.GrowScaleImgDataset* 方法), 424

__getitem__() (*mmagic.datasets.SinGANDataset* 方法), 427

__len__() (*mmagic.datasets.SinGANDataset* 方法), 427

__len__() (*mmagic.datasets.UnpairedImageDataset* 方法), 429

__len__() (*mmagic.structures.DataSample* 方法), 406

__repr__() (*mmagic.datasets.BasicConditionalDataset* 方法), 411

__repr__() (*mmagic.datasets.GrowScaleImgDataset* 方法), 424

__repr__() (*mmagic.datasets.transforms.AlbuCorruptFunction* 方法), 434

__repr__() (*mmagic.datasets.transforms.Albumentations* 方法), 436

__repr__() (*mmagic.datasets.transforms.BinarizeImage* 方法), 439

__repr__() (*mmagic.datasets.transforms.CenterCropLongEdge* 方法), 448

__repr__() (*mmagic.datasets.transforms.Clip* 方法), 440

__repr__() (*mmagic.datasets.transforms.ColorJitter* 方法), 441

__repr__() (*mmagic.datasets.transforms.CompositeFg* 方法), 455

__repr__() (mmagic.datasets.transforms.CopyValues 方法), 477
__repr__() (mmagic.datasets.transforms.Crop 方法), 449
__repr__() (mmagic.datasets.transforms.CropAroundCenter 方法), 449
__repr__() (mmagic.datasets.transforms.CropAroundUnknown 方法), 451
__repr__() (mmagic.datasets.transforms.CropLike 方法), 451
__repr__() (mmagic.datasets.transforms.DegradationsWithShuffle 方法), 471
__repr__() (mmagic.datasets.transforms.FixedCrop 方法), 452
__repr__() (mmagic.datasets.transforms.Flip 方法), 445
__repr__() (mmagic.datasets.transforms.FormatTrimap 方法), 475
__repr__() (mmagic.datasets.transforms.GenerateCoordinateAndCell 方法), 460
__repr__() (mmagic.datasets.transforms.GenerateFacialHeatmap 方法), 461
__repr__() (mmagic.datasets.transforms.GenerateFrameIndices 方法), 461
__repr__() (mmagic.datasets.transforms.GenerateFrameIndicesWithPadding 方法), 462
__repr__() (mmagic.datasets.transforms.GenerateSeg 方法), 437
__repr__() (mmagic.datasets.transforms.GenerateSegmentIndices 方法), 463
__repr__() (mmagic.datasets.transforms.GenerateSoftSeg 方法), 437
__repr__() (mmagic.datasets.transforms.GenerateTrimap 方法), 476
__repr__() (mmagic.datasets.transforms.GenerateTrimapWithDistTransform 方法), 476
__repr__() (mmagic.datasets.transforms.GetMaskedImage 方法), 464
__repr__() (mmagic.datasets.transforms.GetSpatialDiscountMask 方法), 464
__repr__() (mmagic.datasets.transforms.LoadImageFromFile 方法), 466
__repr__() (mmagic.datasets.transforms.LoadMask 方法), 467
__repr__() (mmagic.datasets.transforms.MATLABLikeResize 方法), 470
__repr__() (mmagic.datasets.transforms.MergeFgAndBg 方法), 456
__repr__() (mmagic.datasets.transforms.MirrorSequence 方法), 438
__repr__() (mmagic.datasets.transforms.ModCrop 方法), 453
__repr__() (mmagic.datasets.transforms.Normalize 方法), 470
__repr__() (mmagic.datasets.transforms.NumpyPad 方法), 445
__repr__() (mmagic.datasets.transforms.PackInputs 方法), 458
__repr__() (mmagic.datasets.transforms.PairedAlbuTransForms 方法), 434
__repr__() (mmagic.datasets.transforms.PairedRandomCrop 方法), 453
__repr__() (mmagic.datasets.transforms.PerturbBg 方法), 456
__repr__() (mmagic.datasets.transforms.RandomAffine 方法), 443
__repr__() (mmagic.datasets.transforms.RandomBlur 方法), 472
__repr__() (mmagic.datasets.transforms.RandomCropLongEdge 方法), 454
__repr__() (mmagic.datasets.transforms.RandomDownSampling 方法), 475
__repr__() (mmagic.datasets.transforms.RandomJPEGCompression 方法), 472
__repr__() (mmagic.datasets.transforms.RandomJitter 方法), 457
__repr__() (mmagic.datasets.transforms.RandomLoadResizeBg 方法), 457
__repr__() (mmagic.datasets.transforms.RandomMaskDilation 方法), 443
__repr__() (mmagic.datasets.transforms.RandomNoise 方法), 473
__repr__() (mmagic.datasets.transforms.RandomResize 方法), 473

__repr__() (*mmagic.datasets.transforms.RandomResizedCrop* 方法), 455
__repr__() (*mmagic.datasets.transforms.RandomRotation* 方法), 446
__repr__() (*mmagic.datasets.transforms.RandomTransposeHW* 方法), 446
__repr__() (*mmagic.datasets.transforms.RandomVideoCompression* 方法), 474
__repr__() (*mmagic.datasets.transforms.RescaleToZeroOne* 方法), 471
__repr__() (*mmagic.datasets.transforms.Resize* 方法), 448
__repr__() (*mmagic.datasets.transforms.SetValues* 方法), 478
__repr__() (*mmagic.datasets.transforms.TemporalReverse* 方法), 438
__repr__() (*mmagic.datasets.transforms.TransformTrimap* 方法), 477
__repr__() (*mmagic.datasets.transforms.UnsharpMasking* 方法), 444
__repr__() (*mmagic.models.archs.TokenizerWrapper* 方法), 563
_after_iter() (*mmagic.engine.hooks.BasicVisualizationHook* 方法), 525
_after_iter() (*mmagic.engine.hooks.IterTimerHook* 方法), 521
_apply_albu() (*mmagic.datasets.transforms.Albumentations* 方法), 436
_apply_gaussian_noise() (*mmagic.datasets.transforms.RandomNoise* 方法), 473
_apply_poisson_noise() (*mmagic.datasets.transforms.RandomNoise* 方法), 473
_apply_random_blur() (*mmagic.datasets.transforms.RandomBlur* 方法), 472
_apply_random_compression() (*mmagic.datasets.transforms.RandomJPEGCompression* 方法), 472
_apply_random_compression() (*mmagic.datasets.transforms.RandomVideoCompression* 方法), 474
_apply_random_noise() (*mmagic.datasets.transforms.RandomNoise* 方法), 473
_binarize() (*mmagic.datasets.transforms.BinarizeImage* 方法), 439
_build_dataloaders() (*mmagic.engine.runner.MultiTestLoop* 方法), 537
_build_dataloaders() (*mmagic.engine.runner.MultiValLoop* 方法), 539
_build_degradations() (*mmagic.datasets.transforms.DegradationsWithShuffle* 方法), 471
_build_evaluators() (*mmagic.engine.runner.MultiTestLoop* 方法), 537
_build_evaluators() (*mmagic.engine.runner.MultiValLoop* 方法), 539
_cal_metric_hash() (*mmagic.evaluation.Evaluator* 静态方法), 481
_calc_fid() (*mmagic.evaluation.FrechetInceptionDistance* 静态方法), 494
_calculate_average_value() (*mmagic.engine.hooks.ReduceLRSchedulerHook* 方法), 524
_check_integrity() (*mmagic.datasets.CIFAR10* 方法), 419
_clip() (*mmagic.datasets.transforms.Clip* 方法), 439
_collect_target_results() (*mmagic.evaluation.Equivariance* 方法), 492
_collect_target_results() (*mmagic.evaluation.MultiScaleStructureSimilarity* 方法), 500
_collect_target_results() (*mmagic.evaluation.SlicedWassersteinDistance* 方法), 505
_color_jitter() (*mmagic.datasets.transforms.ColorJitter* 方法), 441
_compat_classes()

(*mmagic.datasets.BasicConditionalDataset* 方法), 411
_compute_distance() (*mmagic.evaluation.PerceptualPathLength* 方法), 501
_conv_type (*mmagic.models.editors.ContextualAttentionNeck* 属性), 668
_conv_type (*mmagic.models.editors.DeepFillDecoder* 属性), 668
_conv_type (*mmagic.models.editors.DeepFillEncoder* 属性), 669
_conv_type (*mmagic.models.editors.GLDilationNeck* 属性), 703
_convert() (*mmagic.datasets.transforms.LoadImageFromFile* 方法), 466
_crop() (*mmagic.datasets.transforms.Crop* 方法), 449
_crop() (*mmagic.datasets.transforms.FixedCrop* 方法), 451
_crop_hole() (*mmagic.datasets.transforms.GenerateSeg* 静态方法), 436
_default_channels_cfg (*mmagic.models.editors.DenoisingUnet* 属性), 657
_destruct_norm_and_conversion() (*mmagic.models.data_preprocessors.DataPreprocessor* 方法), 629
_destruct_padding() (*mmagic.models.data_preprocessors.DataPreprocessor* 方法), 629
_directions (*mmagic.datasets.transforms.Flip* 属性), 444
_do_conversion() (*mmagic.models.data_preprocessors.DataPreprocessor* 方法), 626
_do_norm() (*mmagic.models.data_preprocessors.DataPreprocessor* 方法), 626
_dump() (*mmagic.visualization.VisBackend* 方法), 514
_encode_prompt() (*mmagic.models.editors.StableDiffusion* 方法), 753
_face_alignment_detector() (*mmagic.datasets.transforms.GenerateFacialHeatmap* 方法), 460
_find_samples() (*mmagic.datasets.BasicConditionalDataset* 方法), 411
_forward() (*mmagic.models.editors.DIM* 方法), 681
_forward() (*mmagic.models.editors.GCA* 方法), 698
_forward() (*mmagic.models.editors.IndexNet* 方法), 711
_forward_test() (*mmagic.models.editors.DIM* 方法), 681
_forward_test() (*mmagic.models.editors.GCA* 方法), 699
_forward_test() (*mmagic.models.editors.IndexNet* 方法), 711
_forward_train() (*mmagic.models.editors.DIM* 方法), 681
_forward_train() (*mmagic.models.editors.GCA* 方法), 699
_forward_train() (*mmagic.models.editors.IndexNet* 方法), 711
_freeze_stages() (*mmagic.models.archs.ResNet* 方法), 557
_from_numpy() (*mmagic.models.editors.SinGAN* 方法), 742
_generate_one_heatmap() (*mmagic.datasets.transforms.GenerateFacialHeatmap* 方法), 461
_get_dataloader_size() (*mmagic.engine.runner.LogProcessor* 方法), 535
_get_disc_loss() (*mmagic.models.base_models.BaseGAN* 方法), 580
_get_disc_loss() (*mmagic.models.editors.CycleGAN* 方法), 652
_get_disc_loss() (*mmagic.models.editors.Pix2Pix* 方法), 730
_get_file_list() (*mmagic.datasets.transforms.CompositeFg* 方法), 455
_get_frames_list() (*mmagic.datasets.BasicFramesDataset* 方法), 415
_get_gen_loss() (*mmagic.models.base_models.BaseGAN* 方法), 580
_get_gen_loss() (*mmagic.models.editors.CycleGAN* 方法), 652

_get_gen_loss() (*mmagic.models.editors.Pix2Pix 方法*), 730
_get_inverse_affine_matrix() (*mmagic.datasets.transforms.RandomAffine 静态方法*), 442
_get_mask_from_file() (*mmagic.datasets.transforms.LoadMask 方法*), 467
_get_n_row_and_padding() (*mmagic.visualization.Visualizer 静态方法*), 515
_get_numpy_data() (*mmagic.engine.hooks.PickleDataHook 方法*), 523
_get_opposite_domain() (*mmagic.models.editors.CycleGAN 方法*), 652
_get_params() (*mmagic.datasets.transforms.RandomAffine 静态方法*), 442
_get_path_list() (*mmagic.datasets.BasicFramesDataset 方法*), 414
_get_path_list() (*mmagic.datasets.BasicImageDataset 方法*), 418
_get_path_list_from_ann() (*mmagic.datasets.BasicFramesDataset 方法*), 414
_get_path_list_from_ann() (*mmagic.datasets.BasicImageDataset 方法*), 418
_get_path_list_from_folder() (*mmagic.datasets.BasicFramesDataset 方法*), 414
_get_path_list_from_folder() (*mmagic.datasets.BasicImageDataset 方法*), 418
_get_random_mask_from_set() (*mmagic.datasets.transforms.LoadMask 方法*), 467
_get_target_discriminator() (*mmagic.models.base_models.BaseTranslationModel 方法*), 586
_get_target_generator() (*mmagic.models.base_models.BaseTranslationModel 方法*), 585
_get_valid_model() (*mmagic.models.base_models.BaseGAN 方法*), 579
_get_valid_num_classes() (*mmagic.models.base_models.BaseConditionalGAN 静态方法*), 572
_get_value() (*mmagic.engine.schedulers.LinearLrInterval 方法*), 541
_get_value() (*mmagic.engine.schedulers.ReduceLR 方法*), 543
_get_vis_data_by_key() (*mmagic.visualization.Visualizer 方法*), 516
_gram_mat() (*mmagic.models.losses.PerceptualLoss 方法*), 618
_init_ema_model() (*mmagic.models.base_models.BaseGAN 方法*), 579
_init_env() (*mmagic.visualization.PaviVisBackend 方法*), 510
_init_env() (*mmagic.visualization.VisBackend 方法*), 513
_init_env() (*mmagic.visualization.WandbVisBackend 方法*), 514
_init_info() (*mmagic.datasets.transforms.LoadMask 方法*), 467
_init_is_better() (*mmagic.engine.schedulers.ReduceLR 方法*), 543
_init_loss() (*mmagic.models.base_models.BaseGAN 方法*), 578
_init_pipeline() (*mmagic.apis.inferencers.InpaintingInferencer 方法*), 395
_init_weights() (*mmagic.models.editors.SwinIRNet 方法*), 767
_ir_se50_url (*mmagic.models.editors.IDLossModel 属性*), 640
_load_domain_data_list() (*mmagic.datasets.UnpairedImageDataset 方法*), 428
_load_from_state_dict()

(*mmagic.models.base_models.ExponentialMovingAverage* 方法), 568
_load_from_state_dict() (*mmagic.models.base_models.RampUpEMA* 方法), 570
_load_image() (*mmagic.datasets.transforms.LoadImageFromFile* 方法), 465
_load_inception() (*mmagic.evaluation.FrechetInceptionDistance* 方法), 494
_load_inception() (*mmagic.evaluation.InceptionScore* 方法), 497
_load_meta() (*mmagic.datasets.CIFAR10* 方法), 419
_load_pretrained_model() (*mmagic.models.editors.StyleGAN3Generator* 方法), 765
_load_vgg() (*mmagic.evaluation.PrecisionAndRecall* 方法), 503
_make_layer() (*mmagic.models.archs.ResNet* 方法), 557
_make_layer() (*mmagic.models.archs.VGG16* 方法), 564
_make_layer() (*mmagic.models.editors.IndexNetEncoder* 方法), 713
_make_stem_layer() (*mmagic.models.archs.ResNet* 方法), 557
_nostride_dilate() (*mmagic.models.archs.ResNet* 方法), 557
_parse_batch_channel_order() (*mmagic.models.data_preprocessors.DataPreprocessor* 方法), 625
_parse_channel_index() (*mmagic.models.data_preprocessors.DataPreprocessor* 静态方法), 625
_parse_channel_order() (*mmagic.models.data_preprocessors.DataPreprocessor* 方法), 625
_pickle_data() (*mmagic.engine.hooks.PickleDataHook* 方法), 523
_post_process_image() (*mmagic.visualization.Visualizer* 静态方法), 515
_pred2dict() (*mmagic.apis.inferencers.ConditionalInferencer* 方法), 391
_pred2dict() (*mmagic.apis.inferencers.MattingInferencer* 方法), 397
_pred2dict() (*mmagic.apis.inferencers.UnconditionalInferencer* 方法), 400
_preprocess() (*mmagic.evaluation.InceptionScore* 方法), 498
_preprocess_data_sample() (*mmagic.models.data_preprocessors.DataPreprocessor* 方法), 627
_preprocess_data_sample() (*mmagic.models.data_preprocessors.MattorPreprocessor* 方法), 630
_preprocess_dict_inputs() (*mmagic.models.data_preprocessors.DataPreprocessor* 方法), 627
_preprocess_image_list() (*mmagic.models.data_preprocessors.DataPreprocessor* 方法), 627
_preprocess_image_tensor() (*mmagic.models.data_preprocessors.DataPreprocessor* 方法), 626
_proc_batch_trimap() (*mmagic.models.data_preprocessors.MattorPreprocessor* 方法), 630
_random_dilate() (*mmagic.datasets.transforms.RandomMaskDilation* 方法), 443
_random_resize() (*mmagic.datasets.transforms.RandomResize* 方法), 473
_reset() (*mmagic.engine.schedulers.ReduceLR* 方法), 543
_resize() (*mmagic.datasets.transforms.MATLABLikeResize* 方法), 469
_resize() (*mmagic.datasets.transforms.Resize* 方法), 448
_run_forward() (*mmagic.models.editors.DeblurGanV2* 方法), 662
_set_seq_lens() (*mmagic.datasets.BasicFramesDataset* 方法), 415
_set_value() (*mmagic.models.archs.LoRAWrapper*

方法), 552
_supported_upscale_factors (*mmagic.models.editors.MSRResNet* 属性), 749
_supported_upscale_factors (*mmagic.models.editors.RRDBNet* 属性), 695
_transform() (*mmagic.models.archs.SpatialTemporalEnsemble* 方法), 549
_unsharp_masking() (*mmagic.datasets.transforms.UnsharpMasking* 方法), 444
_update_metainfo() (*mmagic.models.data_preprocessors.DataPreprocessor* 方法), 625
_upload() (*mmagic.visualization.VisBackend* 方法), 514
_vis_gif_sample() (*mmagic.visualization.Visualizer* 方法), 516
_vis_image_sample() (*mmagic.visualization.Visualizer* 方法), 516
_wgan_logistic_ns_loss() (*mmagic.models.losses.GANLossComps* 方法), 611
_wgan_loss() (*mmagic.models.losses.GANLoss* 方法), 601
_wgan_loss() (*mmagic.models.losses.GANLossComps* 方法), 610

A

AblatedDiffusionModel (*mmagic.models.editors* 中的类), 704
add_config() (*mmagic.visualization.VisBackend* 方法), 513
add_datasample() (*mmagic.visualization.ConcatImageVisualizer* 方法), 510
add_datasample() (*mmagic.visualization.Visualizer* 方法), 517
add_embedding() (*mmagic.models.editors.ClipWrapper* 方法), 683
add_gaussian_noise() (在 *mmagic.utils* 模块中), 786
add_image() (*mmagic.visualization.PaviVisBackend* 方法), 510
add_image() (*mmagic.visualization.TensorboardVisBackend* 方法), 512
add_image() (*mmagic.visualization.VisBackend* 方法), 513
add_image() (*mmagic.visualization.Visualizer* 方法), 517
add_image() (*mmagic.visualization.WandbVisBackend* 方法), 514
add_lora() (*mmagic.models.archs.LoRAWrapper* 方法), 552
add_placeholder_token() (*mmagic.models.archs.TokenizerWrapper* 方法), 562
add_scalar() (*mmagic.visualization.PaviVisBackend* 方法), 511
add_scalar() (*mmagic.visualization.VisBackend* 方法), 513
add_scalars() (*mmagic.visualization.PaviVisBackend* 方法), 511
add_scalars() (*mmagic.visualization.VisBackend* 方法), 514
add_tokens() (*mmagic.models.editors.TextualInversion* 方法), 771
adjust_gamma() (在 *mmagic.utils* 模块中), 786
AdobeComp1kDataset (*mmagic.datasets* 中的类), 419
AdvLoss (*mmagic.models.losses* 中的类), 597
after_run() (*mmagic.engine.hooks.PickleDataHook* 方法), 523
after_test_iter() (*mmagic.engine.hooks.VisualizationHook* 方法), 527
after_train_epoch() (*mmagic.engine.hooks.ReduceLRSchedulerHook* 方法), 524
after_train_iter() (*mmagic.engine.hooks.ExponentialMovingAverageHook* 方法), 521
after_train_iter() (*mmagic.engine.hooks.PickleDataHook* 方

法), 523
after_train_iter() (*mmagic.engine.hooks.ReduceLRSchedulerHook* 方法), 524
after_train_iter() (*mmagic.engine.hooks.VisualizationHook* 方法), 528
after_val_epoch() (*mmagic.engine.hooks.ReduceLRSchedulerHook* 方法), 524
after_val_iter() (*mmagic.engine.hooks.VisualizationHook* 方法), 527
albu_builder() (*mmagic.datasets.transforms.Albumentations* 方法), 435
AlbuCorruptFunction (*mmagic.datasets.transforms* 中的类), 433
Albumentations (*mmagic.datasets.transforms* 中的类), 434
all_to_tensor() (在 *mmagic.utils* 模块中), 783
AllGatherLayer (*mmagic.models.archs* 中的类), 547
AOTBlockNeck (*mmagic.models.editors* 中的类), 637
AOTEncoderDecoder (*mmagic.models.editors* 中的类), 637
AOTInpaintor (*mmagic.models.editors* 中的类), 638
arch_settings (*mmagic.models.archs.ResNet* 属性), 557
ASPP (*mmagic.models.archs* 中的类), 547
AttentionInjection (*mmagic.models.archs* 中的类), 548
avg_func() (*mmagic.models.base_models.ExponentialMovingAverage* 方法), 568
avg_func() (*mmagic.models.base_models.RampUpEMA* 方法), 570

B

backward() (*mmagic.models.archs.AllGatherLayer* 静态方法), 547
base_folder (*mmagic.datasets.CIFAR10* 属性), 419
BaseConditionalGAN (*mmagic.models.base_models* 中的类), 571
BaseEditModel (*mmagic.models.base_models* 中的类), 574
BaseGAN (*mmagic.models.base_models* 中的类), 576
BaseMattor (*mmagic.models.base_models* 中的类), 581
BaseTranslationModel (*mmagic.models.base_models* 中的类), 583
BasicConditionalDataset (*mmagic.datasets* 中的类), 409
BasicFramesDataset (*mmagic.datasets* 中的类), 412
BasicImageDataset (*mmagic.datasets* 中的类), 415
BasicInterpolator (*mmagic.models.base_models* 中的类), 586
BasicVisualizationHook (*mmagic.engine.hooks* 中的类), 524
BasicVSR (*mmagic.models.editors* 中的类), 640
BasicVSRNet (*mmagic.models.editors* 中的类), 641
BasicVSRPlusPlusNet (*mmagic.models.editors* 中的类), 642
bbox2mask() (在 *mmagic.utils* 模块中), 786
before_run() (*mmagic.engine.hooks.ExponentialMovingAverageHook* 方法), 521
before_run() (*mmagic.engine.hooks.PickleDataHook* 方法), 523
before_train_iter() (*mmagic.engine.hooks.PGGANFetchDataHook* 方法), 522
BigGAN (*mmagic.models.editors* 中的类), 644
BinarizeImage (*mmagic.datasets.transforms* 中的类), 438
bring_stroke_mask() (在 *mmagic.utils* 模块中), 787

C

CAIN (*mmagic.models.editors* 中的类), 646
CAINNet (*mmagic.models.editors* 中的类), 647
calculate_loss_with_type() (*mmagic.models.base_models.TwoStageInpaintor* 方法), 594
calculate_loss_with_type() (*mmagic.models.editors.DeepFillv1Inpaintor* 方法), 673
calculate_overlap_factor()

(*mmagic.models.editors.ContextualAttentionModule* 方法), 666
calculate_unfold_hw() (*mmagic.models.editors.ContextualAttentionModule* 方法), 666
can_convert_to_image() (在 *mmagic.utils* 模块中), 783
cast_data() (*mmagic.models.data_preprocessors.DataPreprocessor* 方法), 625
CenterCropLongEdge (*mmagic.datasets.transforms* 中的类), 448
CharbonnierCompLoss (*mmagic.models.losses* 中的类), 598
CharbonnierLoss (*mmagic.models.losses* 中的类), 620
check_if_mirror_extended() (*mmagic.models.editors.BasicVSR* 方法), 641
check_if_mirror_extended() (*mmagic.models.editors.BasicVSRNet* 方法), 642
check_if_mirror_extended() (*mmagic.models.editors.BasicVSRPlusPlusNet* 方法), 643
check_if_mirror_extended() (*mmagic.models.editors.IconVSRNet* 方法), 707
check_image_size() (*mmagic.models.editors.NAFBaseline* 方法), 720
check_image_size() (*mmagic.models.editors.NAFNet* 方法), 721
check_image_size() (*mmagic.models.editors.SwinIRNet* 方法), 767
check_inputs() (*mmagic.models.editors.StableDiffusion* 方法), 754
CIFAR10 (*mmagic.datasets* 中的类), 418
class_to_idx (*mmagic.datasets.BasicConditionalDataset* *property*), 411
CLASSES (*mmagic.datasets.BasicConditionalDataset* *property*), 411
Clip (*mmagic.datasets.transforms* 中的类), 439
CLIPLoss (*mmagic.models.losses* 中的类), 597
CLIPLossComps (*mmagic.models.losses* 中的类), 607
ClipWrapper (*mmagic.models.editors* 中的类), 682
ColorizationInferencer (*mmagic.apis.inferencers* 中的类), 389
ColorJitter (*mmagic.datasets.transforms* 中的类), 440
CompositeFg (*mmagic.datasets.transforms* 中的类), 455
compute_flow() (*mmagic.models.editors.BasicVSRNet* 方法), 642
compute_flow() (*mmagic.models.editors.BasicVSRPlusPlusNet* 方法), 643
compute_flow() (*mmagic.models.editors.IconVSRNet* 方法), 708
compute_metrics() (*mmagic.evaluation.ConnectivityError* 方法), 491
compute_metrics() (*mmagic.evaluation.Equivariance* 方法), 492
compute_metrics() (*mmagic.evaluation.FrechetInceptionDistance* 方法), 495
compute_metrics() (*mmagic.evaluation.GradientError* 方法), 495
compute_metrics() (*mmagic.evaluation.InceptionScore* 方法), 498
compute_metrics() (*mmagic.evaluation.MattingMSE* 方法), 499
compute_metrics() (*mmagic.evaluation.MultiScaleStructureSimilarity* 方法), 500
compute_metrics() (*mmagic.evaluation.PerceptualPathLength* 方法), 501
compute_metrics() (*mmagic.evaluation.PrecisionAndRecall* 方法), 504
compute_metrics() (*mmagic.evaluation.SAD* 方法),

487
compute_metrics() (*mmagic.evaluation.SlicedWassersteinDistance* 方法), 505
compute_refill_features() (*mmagic.models.editors.IconVSRNet* 方法), 707
compute_zero_padding() (*mmagic.models.losses.GaussianBlur* 静态方法), 603
concat_imgs_list_to() (*mmagic.datasets.GrowScaleImgDataset* 方法), 423
ConcatImageVisualizer (*mmagic.visualization* 中的类), 509
ConditionalInferencer (*mmagic.apis.inferencers* 中的类), 390
ConfigType() (在 *mmagic.utils* 模块中), 789
ConnectivityError (*mmagic.evaluation* 中的类), 490
construct_fixed_noises() (*mmagic.models.editors.PESinGAN* 方法), 720
construct_fixed_noises() (*mmagic.models.editors.SinGAN* 方法), 743
ContextualAttentionModule (*mmagic.models.editors* 中的类), 664
ContextualAttentionNeck (*mmagic.models.editors* 中的类), 667
ControlnetAnimationInferencer (*mmagic.apis.inferencers* 中的类), 391
ControlNetDataset (*mmagic.datasets* 中的类), 421
ControlStableDiffusion (*mmagic.models.editors* 中的类), 648
convert_to_datasample() (*mmagic.models.base_models.BaseEditModel* 方法), 575
convert_to_datasample() (*mmagic.models.base_models.BaseMattor* 方法), 583
convert_to_datasample() (*mmagic.models.base_models.OneStageInpaintor* 方法), 591
convert_to_datasample() (*mmagic.models.editors.DeblurGanV2* 方法), 660
convert_to_datasample() (*mmagic.models.editors.InstColorization* 方法), 715
convert_to_fp16() (*mmagic.models.editors.DenoisingUnet* 方法), 658
convert_to_fp32() (*mmagic.models.editors.DenoisingUnet* 方法), 658
CopyValues (*mmagic.datasets.transforms* 中的类), 477
Crop (*mmagic.datasets.transforms* 中的类), 448
CropAroundCenter (*mmagic.datasets.transforms* 中的类), 449
CropAroundFg (*mmagic.datasets.transforms* 中的类), 449
CropAroundUnknown (*mmagic.datasets.transforms* 中的类), 450
CropLike (*mmagic.datasets.transforms* 中的类), 451
CycleGAN (*mmagic.models.editors* 中的类), 651

D

d_step() (*mmagic.models.editors.DeblurGanV2* 方法), 663
d_step_fake() (*mmagic.models.editors.ESRGAN* 方法), 694
d_step_fake() (*mmagic.models.editors.RealESRGAN* 方法), 737
d_step_fake() (*mmagic.models.editors.SRGAN* 方法), 747
d_step_real() (*mmagic.models.editors.ESRGAN* 方法), 694
d_step_real() (*mmagic.models.editors.RealESRGAN* 方法), 737
d_step_real() (*mmagic.models.editors.SRGAN* 方法), 747
d_step_with_optim() (*mmagic.models.editors.DeblurGanV2* 方法), 663

d_step_with_optim() (*mmagic.models.editors.SRGAN 方法*), 748
DATA_KEYS (*mmagic.structures.DataSample 属性*), 405
data_preprocessor (*mmagic.models.base_models.BaseEditModel 属性*), 574
data_preprocessor (*mmagic.models.base_models.BasicInterpolator 属性*), 587
data_preprocessor (*mmagic.models.editors.CAIN 属性*), 646
data_preprocessor (*mmagic.models.editors.DeblurGanV2 属性*), 659
data_preprocessor (*mmagic.models.editors.FLAVR 属性*), 697
data_sample_to_label() (*mmagic.models.base_models.BaseConditionalGAN 方法*), 572
data_sample_to_label() (*mmagic.models.editors.EG3D 方法*), 691
DataPreprocessor (*mmagic.models.data_preprocessors 中的类*), 623
DataSample (*mmagic.structures 中的类*), 403
DCGAN (*mmagic.models.editors 中的类*), 653
DeblurGanV2 (*mmagic.models.editors 中的类*), 658
DeblurGanV2Discriminator (*mmagic.models.editors 中的类*), 664
DeblurGanV2Generator (*mmagic.models.editors 中的类*), 664
decode() (*mmagic.models.archs.TokenizerWrapper 方法*), 563
decode_latents() (*mmagic.models.editors.StableDiffusion 方法*), 753
DeepFillDecoder (*mmagic.models.editors 中的类*), 668
DeepFillEncoder (*mmagic.models.editors 中的类*), 669
DeepFillEncoderDecoder (*mmagic.models.editors 中的类*), 674
DeepFillRefiner (*mmagic.models.editors 中的类*), 669
DeepFillv1Discriminators (*mmagic.models.editors 中的类*), 670
DeepFillv1Inpaintor (*mmagic.models.editors 中的类*), 671
default_prefix (*mmagic.evaluation.MattingMSE 属性*), 499
default_prefix (*mmagic.evaluation.SAD 属性*), 487
DegradationsWithShuffle (*mmagic.datasets.transforms 中的类*), 471
DenoisingUnet (*mmagic.models.editors 中的类*), 654
DepthwiseIndexBlock (*mmagic.models.editors 中的类*), 708
DepthwiseSeparableConvModule (*mmagic.models.archs 中的类*), 558
destruct() (*mmagic.models.data_preprocessors.DataPreprocessor 方法*), 628
device (*mmagic.models.base_models.BaseGAN property*), 577
device (*mmagic.models.editors.AblatedDiffusionModel property*), 705
device (*mmagic.models.editors.DiscoDiffusion property*), 684
device (*mmagic.models.editors.StableDiffusion property*), 751
DIC (*mmagic.models.editors 中的类*), 674
DICNet (*mmagic.models.editors 中的类*), 676
DIM (*mmagic.models.editors 中的类*), 680
disc_loss() (*mmagic.models.editors.BigGAN 方法*), 645
disc_loss() (*mmagic.models.editors.DCGAN 方法*), 653
disc_loss() (*mmagic.models.editors.GGAN 方法*), 699
disc_loss() (*mmagic.models.editors.LSGAN 方法*), 717
disc_loss() (*mmagic.models.editors.ProgressiveGrowingGAN 方法*), 728
disc_loss() (*mmagic.models.editors.SAGAN 方法*), 740
disc_loss() (*mmagic.models.editors.SinGAN 方法*), 743
disc_loss() (*mmagic.models.editors.StyleGAN1 方*

法), 760
disc_loss() (*mmagic.models.editors.StyleGAN2* 方法), 761
disc_loss() (*mmagic.models.editors.WGANGP* 方法), 778
disc_shift_loss() (在 *mmagic.models.losses* 模块中), 604
DiscoDiffusion (*mmagic.models.editors* 中的类), 683
discriminator_steps (*mmagic.models.base_models.BaseGAN* property), 577
DiscShiftLoss (*mmagic.models.losses* 中的类), 601
DiscShiftLossComps (*mmagic.models.losses* 中的类), 608
download_from_url() (在 *mmagic.utils* 模块中), 785
DreamBooth (*mmagic.models.editors* 中的类), 685
DreamBoothDataset (*mmagic.datasets* 中的类), 421

E

EDSRNet (*mmagic.models.editors* 中的类), 688
EDVR (*mmagic.models.editors* 中的类), 689
EDVRNet (*mmagic.models.editors* 中的类), 689
EG3D (*mmagic.models.editors* 中的类), 690
EG3DInferencer (*mmagic.apis.inferencers* 中的类), 392
encode() (*mmagic.models.archs.TokenizerWrapper* 方法), 563
Equivariance (*mmagic.evaluation* 中的类), 491
ESRGAN (*mmagic.models.editors* 中的类), 693
evaluate() (*mmagic.evaluation.Evaluator* 方法), 482
Evaluator (*mmagic.evaluation* 中的类), 481
every_n_iters() (*mmagic.engine.hooks.ExponentialMovingAverageHook* 方法), 520
experiment (*mmagic.visualization.PaviVisBackend* property), 510
experiment (*mmagic.visualization.VisBackend* property), 513
ExponentialMovingAverage (*mmagic.models.base_models* 中的类), 567
ExponentialMovingAverageHook (*mmagic.engine.hooks* 中的类), 519
extra_parameters (*mmagic.apis.inferencers.ConditionalInferencer* 属性), 391
extra_parameters (*mmagic.apis.inferencers.ControlnetAnimationInferen* 属性), 392
extra_parameters (*mmagic.apis.inferencers.EG3DInferencer* 属性), 392
extra_parameters (*mmagic.apis.inferencers.Text2ImageInferencer* 属性), 397
extra_parameters (*mmagic.apis.inferencers.UnconditionalInferencer* 属性), 399
extra_parameters (*mmagic.apis.inferencers.VideoInterpolationInference* 属性), 400
extra_parameters (*mmagic.apis.inferencers.VideoRestorationInferencer* 属性), 401
extra_repr() (*mmagic.datasets.BasicConditionalDataset* 方法), 412
extra_repr() (*mmagic.datasets.CIFAR10* 方法), 419
extract_feats() (*mmagic.models.editors.IDLossModel* 方法), 640
extract_features() (*mmagic.evaluation.PrecisionAndRecall* 方法), 503
extract_gt_data() (*mmagic.models.editors.DeblurGanV2* 方法), 663
extract_gt_data() (*mmagic.models.editors.DIC* 静态方法), 676
extract_gt_data() (*mmagic.models.editors.RealBasicVSR* 方法), 734
extract_gt_data() (*mmagic.models.editors.RealESRGAN* 方法), 738
extract_gt_data() (*mmagic.models.editors.SRGAN* 方法), 748

F

FaceIdLoss (*mmagic.models.losses* 中的类), 600
FaceIdLossComps (*mmagic.models.losses* 中的类), 609
FBADecoder (*mmagic.models.editors* 中的类), 695

FBAResnetDilated (*mmagic.models.editors* 中的类), 696
FeedbackBlock (*mmagic.models.editors* 中的类), 677
FeedbackBlockCustom (*mmagic.models.editors* 中的类), 678
FeedbackBlockHeatmapAttention (*mmagic.models.editors* 中的类), 678
filename (*mmagic.datasets.CIFAR10* 属性), 419
FixedCrop (*mmagic.datasets.transforms* 中的类), 451
FLAVR (*mmagic.models.editors* 中的类), 696
FLAVRNet (*mmagic.models.editors* 中的类), 697
Flip (*mmagic.datasets.transforms* 中的类), 444
FormatTrimap (*mmagic.datasets.transforms* 中的类), 475
forward() (*mmagic.apis.inferencers.ColorizationInferencer* 方法), 390
forward() (*mmagic.apis.inferencers.ConditionalInferencer* 方法), 391
forward() (*mmagic.apis.inferencers.EG3DInferencer* 方法), 393
forward() (*mmagic.apis.inferencers.ImageSuperResolutionInferencer* 方法), 395
forward() (*mmagic.apis.inferencers.InpaintingInferencer* 方法), 396
forward() (*mmagic.apis.inferencers.MattingInferencer* 方法), 397
forward() (*mmagic.apis.inferencers.Text2ImageInferencer* 方法), 398
forward() (*mmagic.apis.inferencers.TranslationInferencer* 方法), 398
forward() (*mmagic.apis.inferencers.UnconditionalInferencer* 方法), 399
forward() (*mmagic.apis.inferencers.VideoInterpolationInferencer* 方法), 400
forward() (*mmagic.apis.inferencers.VideoRestorationInferencer* 方法), 401
forward() (*mmagic.models.archs.AllGatherLayer* 静态方法), 547
forward() (*mmagic.models.archs.ASPP* 方法), 548
forward() (*mmagic.models.archs.AttentionInjection* 方法), 548
forward() (*mmagic.models.archs.DepthwiseSeparableConvModule* 方法), 559
forward() (*mmagic.models.archs.LinearModule* 方法), 551
forward() (*mmagic.models.archs.LoRAWrapper* 方法), 553
forward() (*mmagic.models.archs.MultiLayerDiscriminator* 方法), 555
forward() (*mmagic.models.archs.PatchDiscriminator* 方法), 556
forward() (*mmagic.models.archs.PixelShufflePack* 方法), 564
forward() (*mmagic.models.archs.ResidualBlockNoBN* 方法), 561
forward() (*mmagic.models.archs.ResNet* 方法), 557
forward() (*mmagic.models.archs.SimpleEncoderDecoder* 方法), 559
forward() (*mmagic.models.archs.SimpleGatedConvModule* 方法), 550
forward() (*mmagic.models.archs.SoftMaskPatchDiscriminator* 方法), 560
forward() (*mmagic.models.archs.SpatialTemporalEnsemble* 方法), 549
forward() (*mmagic.models.archs.VGG16* 方法), 564
forward() (*mmagic.models.base_models.BaseConditionalGAN* 方法), 573
forward() (*mmagic.models.base_models.BaseEditModel* 方法), 574
forward() (*mmagic.models.base_models.BaseGAN* 方法), 579
forward() (*mmagic.models.base_models.BaseMattor* 方法), 583
forward() (*mmagic.models.base_models.BaseTranslationModel* 方法), 585
forward() (*mmagic.models.base_models.OneStageInpaintor* 方法), 589
forward() (*mmagic.models.data_preprocessors.DataPreprocessor* 方法), 628
forward() (*mmagic.models.data_preprocessors.MattorPreprocessor* 方法), 631
forward() (*mmagic.models.editors.AblatedDiffusionModel* 方法), 705
forward() (*mmagic.models.editors.AOTBlockNeck* 方

法), 637
forward() (*mmagic.models.editors.BasicVSRNet 方法*), 642
forward() (*mmagic.models.editors.BasicVSRPlusPlusNet 方法*), 644
forward() (*mmagic.models.editors.CAINNet 方法*), 647
forward() (*mmagic.models.editors.ClipWrapper 方法*), 683
forward() (*mmagic.models.editors.ContextualAttentionModule 方法*), 665
forward() (*mmagic.models.editors.ContextualAttentionNeck 方法*), 668
forward() (*mmagic.models.editors.ControlStableDiffusion 方法*), 651
forward() (*mmagic.models.editors.DeblurGanV2 方法*), 659
forward() (*mmagic.models.editors.DeepFillDecoder 方法*), 668
forward() (*mmagic.models.editors.DeepFillEncoder 方法*), 669
forward() (*mmagic.models.editors.DeepFillEncoderDecoder 方法*), 674
forward() (*mmagic.models.editors.DeepFillRefiner 方法*), 670
forward() (*mmagic.models.editors.DeepFillv1Discriminators 方法*), 670
forward() (*mmagic.models.editors.DenoisingUnet 方法*), 657
forward() (*mmagic.models.editors.DepthwiseIndexBlock 方法*), 709
forward() (*mmagic.models.editors.DICNet 方法*), 677
forward() (*mmagic.models.editors.DreamBooth 方法*), 688
forward() (*mmagic.models.editors.EDSRNet 方法*), 688
forward() (*mmagic.models.editors.EDVRNet 方法*), 690
forward() (*mmagic.models.editors.EG3D 方法*), 692
forward() (*mmagic.models.editors.FBADecoder 方法*), 695
forward() (*mmagic.models.editors.FBAResnetDilated 方法*), 696
forward() (*mmagic.models.editors.FeedbackBlock 方法*), 678
forward() (*mmagic.models.editors.FeedbackBlockCustom 方法*), 678
forward() (*mmagic.models.editors.FeedbackBlockHeatmapAttention 方法*), 679
forward() (*mmagic.models.editors.FLAVRNet 方法*), 698
forward() (*mmagic.models.editors.GLDecoder 方法*), 703
forward() (*mmagic.models.editors.GLDilationNeck 方法*), 703
forward() (*mmagic.models.editors.GLEANStyleGANv2 方法*), 702
forward() (*mmagic.models.editors.GLEncoder 方法*), 704
forward() (*mmagic.models.editors.GLEncoderDecoder 方法*), 704
forward() (*mmagic.models.editors.HolisticIndexBlock 方法*), 709
forward() (*mmagic.models.editors.IconVSRNet 方法*), 708
forward() (*mmagic.models.editors.IDLossModel 方法*), 640
forward() (*mmagic.models.editors.IndexedUpsample 方法*), 710
forward() (*mmagic.models.editors.IndexNetDecoder 方法*), 712
forward() (*mmagic.models.editors.IndexNetEncoder 方法*), 713
forward() (*mmagic.models.editors.InstColorization 方法*), 714
forward() (*mmagic.models.editors.LightCNN 方法*), 679
forward() (*mmagic.models.editors.LTE 方法*), 773
forward() (*mmagic.models.editors.MaskConvModule 方法*), 723
forward() (*mmagic.models.editors.MaxFeature 方法*), 680
forward() (*mmagic.models.editors.MLPRefiner 方法*), 717
forward() (*mmagic.models.editors.ModifiedVGG 方法*), 749

forward() (*mmagic.models.editors.MSRResNet 方法*), 749
forward() (*mmagic.models.editors.NAFBaseline 方法*), 720
forward() (*mmagic.models.editors.NAFNet 方法*), 721
forward() (*mmagic.models.editors.PartialConv2d 方法*), 723
forward() (*mmagic.models.editors.PConvDecoder 方法*), 724
forward() (*mmagic.models.editors.PConvEncoder 方法*), 725
forward() (*mmagic.models.editors.PConvEncoderDecoder 方法*), 725
forward() (*mmagic.models.editors.PlainDecoder 方法*), 731
forward() (*mmagic.models.editors.PlainRefiner 方法*), 732
forward() (*mmagic.models.editors.ProgressiveGrowingGAN 方法*), 727
forward() (*mmagic.models.editors.RDNNet 方法*), 732
forward() (*mmagic.models.editors.RealBasicVSRNet 方法*), 735
forward() (*mmagic.models.editors.Restormer 方法*), 739
forward() (*mmagic.models.editors.RRDBNet 方法*), 695
forward() (*mmagic.models.editors.SearchTransformer 方法*), 776
forward() (*mmagic.models.editors.SinGAN 方法*), 743
forward() (*mmagic.models.editors.SRCNNNet 方法*), 746
forward() (*mmagic.models.editors.StableDiffusion 方法*), 755
forward() (*mmagic.models.editors.StyleGAN3Generator 方法*), 765
forward() (*mmagic.models.editors.SwinIRNet 方法*), 767
forward() (*mmagic.models.editors.TDANNet 方法*), 769
forward() (*mmagic.models.editors.TOFlowVFINet 方法*), 772
forward() (*mmagic.models.editors.TOFlowVSRNet 方法*), 772
forward() (*mmagic.models.editors.ToFResBlock 方法*), 773
forward() (*mmagic.models.editors.TTSRDiscriminator 方法*), 777
forward() (*mmagic.models.editors.TTSRNet 方法*), 777
forward() (*mmagic.models.editors.UNetDiscriminatorWithSpectralNorm 方法*), 738
forward() (*mmagic.models.losses.CharbonnierCompLoss 方法*), 598
forward() (*mmagic.models.losses.CharbonnierLoss 方法*), 620
forward() (*mmagic.models.losses.CLIPLoss 方法*), 597
forward() (*mmagic.models.losses.CLIPLossComps 方法*), 607
forward() (*mmagic.models.losses.DiscShiftLoss 方法*), 601
forward() (*mmagic.models.losses.DiscShiftLossComps 方法*), 609
forward() (*mmagic.models.losses.FaceIdLoss 方法*), 600
forward() (*mmagic.models.losses.FaceIdLossComps 方法*), 610
forward() (*mmagic.models.losses.GANLoss 方法*), 602
forward() (*mmagic.models.losses.GANLossComps 方法*), 611
forward() (*mmagic.models.losses.GaussianBlur 方法*), 604
forward() (*mmagic.models.losses.GeneratorPathRegularizerComps 方法*), 613
forward() (*mmagic.models.losses.GradientLoss 方法*), 606
forward() (*mmagic.models.losses.GradientPenaltyLoss 方法*), 604
forward() (*mmagic.models.losses.GradientPenaltyLossComps 方法*), 614
forward() (*mmagic.models.losses.L1CompositionLoss 方法*), 599
forward() (*mmagic.models.losses.L1Loss 方法*), 621
forward() (*mmagic.models.losses.LightCNNFeatureLoss 方法*), 601
forward() (*mmagic.models.losses.MaskedTVLoss 方*

法), 621
forward() (mmagic.models.losses.MSECompositionLoss 方法), 599
forward() (mmagic.models.losses.MSELoss 方法), 622
forward() (mmagic.models.losses.PerceptualLoss 方法), 618
forward() (mmagic.models.losses.PerceptualVGG 方法), 619
forward() (mmagic.models.losses.PSNRLoss 方法), 622
forward() (mmagic.models.losses.R1GradientPenaltyComps 方法), 616
forward() (mmagic.models.losses.TransferalPerceptualLoss 方法), 619
forward_dummy() (mmagic.models.base_models.OneStageInpaintor方法), 591
forward_features() (mmagic.models.editors.SwinIRNet 方法), 767
forward_inception() (mmagic.evaluation.FrechetInceptionDistance 方法), 494
forward_inference() (mmagic.models.base_models.BaseEditModel 方法), 576
forward_inference() (mmagic.models.editors.BasicVSR 方法), 641
forward_inference() (mmagic.models.editors.CAIN 方法), 647
forward_inference() (mmagic.models.editors.DeblurGanV2 方法), 661
forward_inference() (mmagic.models.editors.InstColorization 方法), 715
forward_inference() (mmagic.models.editors.LIIF 方法), 716
forward_lora_mapping() (mmagic.models.archs.LoRAWrapper 方法), 552
forward_tensor() (mmagic.models.base_models.BaseEditModel 方法), 575
forward_tensor() (mmagic.models.base_models.OneStageInpaintor 方法), 591
forward_tensor() (mmagic.models.base_models.TwoStageInpaintor 方法), 593
forward_tensor() (mmagic.models.editors.AOTInpaintor 方法), 639
forward_tensor() (mmagic.models.editors.DeblurGanV2 方法), 661
forward_tensor() (mmagic.models.editors.DIC 方法), 675
forward_tensor() (mmagic.models.editors.InstColorization 方法), 716
forward_tensor() (mmagic.models.editors.LIIF 方法), 716
forward_tensor() (mmagic.models.editors.PConvInpaintor 方法), 726
forward_tensor() (mmagic.models.editors.RealESRGAN 方法), 737
forward_tensor() (mmagic.models.editors.SRGAN 方法), 747
forward_tensor() (mmagic.models.editors.TDAN 方法), 768
forward_tensor() (mmagic.models.editors.TTSR 方法), 774
forward_test() (mmagic.models.base_models.BaseTranslationModel 方法), 585
forward_test() (mmagic.models.base_models.OneStageInpaintor 方法), 591
forward_test() (mmagic.models.editors.CycleGAN 方法), 651
forward_test() (mmagic.models.editors.Pix2Pix 方法), 729
forward_train() (mmagic.models.base_models.BaseEditModel 方法), 576
forward_train() (mmagic.models.base_models.BaseTranslationModel 方法), 585
forward_train() (mmagic.models.base_models.OneStageInpaintor 方法), 590
forward_train() (mmagic.models.editors.BasicVSR 方法), 641
forward_train() (mmagic.models.editors.DeblurGanV2 方法), 661

forward_train() (*mmagic.models.editors.EDVR* 方法), 689
forward_train() (*mmagic.models.editors.InstColorization* 方法), 715
forward_train() (*mmagic.models.editors.RealBasicVSR* 方法), 734
forward_train() (*mmagic.models.editors.SRGAN* 方法), 746
forward_train() (*mmagic.models.editors.TDAN* 方法), 768
forward_train_d() (*mmagic.models.base_models.OneStageInpaintor* 方法), 590
forward_train_d() (*mmagic.models.editors.AOTInpaintor* 方法), 638
forward_train_d() (*mmagic.models.editors.DeepFillv1Inpaintor* 方法), 672
ForwardInputs() (在 *mmagic.utils* 模块中), 789
FrechetInceptionDistance (*mmagic.evaluation* 中的类), 493
freeze_backbone() (*mmagic.models.editors.DIM* 方法), 681
full_init() (*mmagic.datasets.BasicConditionalDataset* 方法), 411
full_init() (*mmagic.datasets.SinGANDataset* 方法), 427
func_kwargs (*mmagic.apis.inferencers.ColorizationInferencer* 属性), 390
func_kwargs (*mmagic.apis.inferencers.ConditionalInferencer* 属性), 390
func_kwargs (*mmagic.apis.inferencers.ControlnetAnimationInferencer* 属性), 392
func_kwargs (*mmagic.apis.inferencers.EG3DInferencer* 属性), 392
func_kwargs (*mmagic.apis.inferencers.ImageSuperResolutionInferencer* 属性), 395
func_kwargs (*mmagic.apis.inferencers.InpaintingInferencer* 属性), 395
func_kwargs (*mmagic.apis.inferencers.MattingInferencer* 属性), 396
func_kwargs (*mmagic.apis.inferencers.Text2ImageInferencer* 属性), 397
func_kwargs (*mmagic.apis.inferencers.TranslationInferencer* 属性), 398
func_kwargs (*mmagic.apis.inferencers.UnconditionalInferencer* 属性), 399
func_kwargs (*mmagic.apis.inferencers.VideoInterpolationInferencer* 属性), 400
func_kwargs (*mmagic.apis.inferencers.VideoRestorationInferencer* 属性), 401
func_order (*mmagic.apis.inferencers.ControlnetAnimationInferencer* 属性), 392
fuse_correlation_map() (*mmagic.models.editors.ContextualAttentionModule* 方法), 665

G

g_step() (*mmagic.models.editors.DeblurGanV2* 方法), 662
g_step() (*mmagic.models.editors.DIC* 方法), 676
g_step() (*mmagic.models.editors.ESRGAN* 方法), 693
g_step() (*mmagic.models.editors.RealBasicVSR* 方法), 734
g_step() (*mmagic.models.editors.RealESRGAN* 方法), 737
g_step() (*mmagic.models.editors.SRGAN* 方法), 747
g_step() (*mmagic.models.editors.TTSR* 方法), 775
g_step_with_optim() (*mmagic.models.editors.DeblurGanV2* 方法), 663
g_step_with_optim() (*mmagic.models.editors.SRGAN* 方法), 748
g_step_with_optim() (*mmagic.models.editors.TTSR* 方法), 775
GANLoss (*mmagic.models.losses* 中的类), 601
GANLossComps (*mmagic.models.losses* 中的类), 610
gather() (*mmagic.models.editors.SearchTransformer* 方法), 775
gather_log_vars() (*mmagic.models.base_models.BaseGAN* 静态方法), 577
gauss_gradient() (在 *mmagic.evaluation* 模块中),

482
gaussian() (*mmagic.models.losses.GaussianBlur* 方法), 603
GaussianBlur (*mmagic.models.losses* 中的类), 602
GCA (*mmagic.models.editors* 中的类), 698
gen_loss() (*mmagic.models.editors.BigGAN* 方法), 645
gen_loss() (*mmagic.models.editors.DCGAN* 方法), 653
gen_loss() (*mmagic.models.editors.GGAN* 方法), 699
gen_loss() (*mmagic.models.editors.LSGAN* 方法), 718
gen_loss() (*mmagic.models.editors.ProgressiveGrowingGAN* 方法), 729
gen_loss() (*mmagic.models.editors.SAGAN* 方法), 740
gen_loss() (*mmagic.models.editors.SinGAN* 方法), 743
gen_loss() (*mmagic.models.editors.StyleGAN1* 方法), 760
gen_loss() (*mmagic.models.editors.StyleGAN2* 方法), 761
gen_loss() (*mmagic.models.editors.WGANGP* 方法), 778
gen_path_regularizer() (在 *mmagic.models.losses* 模块中), 604
generate_class_prior_images() (*mmagic.models.editors.DreamBooth* 方法), 687
generate_heatmap_from_img() (*mmagic.datasets.transforms.GenerateFacialHeatmap* 方法), 460
GenerateCoordinateAndCell (*mmagic.datasets.transforms* 中的类), 458
GenerateFacialHeatmap (*mmagic.datasets.transforms* 中的类), 460
GenerateFrameIndices (*mmagic.datasets.transforms* 中的类), 461
GenerateFrameIndiceswithPadding (*mmagic.datasets.transforms* 中的类), 462
GenerateSeg (*mmagic.datasets.transforms* 中的类), 436
GenerateSegmentIndices (*mmagic.datasets.transforms* 中的类), 462
GenerateSoftSeg (*mmagic.datasets.transforms* 中的类), 437
GenerateTrimap (*mmagic.datasets.transforms* 中的类), 475
GenerateTrimapWithDistTransform (*mmagic.datasets.transforms* 中的类), 476
generator_loss() (*mmagic.models.base_models.OneStageInpaintor* 方法), 590
generator_loss() (*mmagic.models.editors.AOTInpaintor* 方法), 638
generator_steps (*mmagic.models.base_models.BaseGAN* *property*), 577
GeneratorPathRegularizerComps (*mmagic.models.losses* 中的类), 611
get_1d_gaussian_kernel() (*mmagic.models.losses.GaussianBlur* 方法), 603
get_2d_gaussian_kernel() (*mmagic.models.losses.GaussianBlur* 方法), 603
get_box_info() (在 *mmagic.utils* 模块中), 783
get_cat_ids() (*mmagic.datasets.BasicConditionalDataset* 方法), 411
get_data_info() (*mmagic.datasets.UnpairedImageDataset* 方法), 428
get_embedding_layer() (*mmagic.models.editors.ClipWrapper* 方法), 683
get_gt_labels() (*mmagic.datasets.BasicConditionalDataset* 方法), 411
get_irregular_mask() (在 *mmagic.utils* 模块中), 788
get_kernel() (*mmagic.datasets.transforms.RandomBlur* 方法), 472
get_mean_latent() (*mmagic.models.editors.StyleGAN3Generator* 方法), 766
get_metric_sampler() (*mmagic.evaluation.Equivariance* 方法), 492
get_metric_sampler() (*mmagic.evaluation.PerceptualPathLength* 方法), 502

get_metric_sampler() (*mmagic.evaluation.TransFID 方法*), 506
get_metric_sampler() (*mmagic.evaluation.TransIS 方法*), 508
get_module() (*mmagic.models.base_models.BaseTranslationModel 方法*), 584
get_module() (*mmagic.models.editors.AblatedDiffusionModel 方法*), 706
get_module() (*mmagic.models.editors.SinGAN 方法*), 742
get_other_domains() (*mmagic.models.base_models.BaseTranslationModel 方法*), 585
get_params() (*mmagic.datasets.transforms.RandomResizedCrop 方法*), 454
get_sampler() (*在 mmagic.utils 模块中*), 785
get_target_label() (*mmagic.models.losses.GANLoss 方法*), 602
get_target_label() (*mmagic.models.losses.GANLossComps 方法*), 611
get_token_info() (*mmagic.models.archs.TokenizerWrapper 方法*), 562
get_training_kwargs() (*mmagic.models.editors.StyleGAN3Generator 方法*), 766
GetMaskedImage (*mmagic.datasets.transforms 中的类*), 463
GetSpatialDiscountMask (*mmagic.datasets.transforms 中的类*), 464
GGAN (*mmagic.models.editors 中的类*), 699
GLDecoder (*mmagic.models.editors 中的类*), 702
GLDilationNeck (*mmagic.models.editors 中的类*), 703
GLEANStyleGANv2 (*mmagic.models.editors 中的类*), 700
GLEncoder (*mmagic.models.editors 中的类*), 703
GLEncoderDecoder (*mmagic.models.editors 中的类*), 704
gradient_penalty_loss() (*在 mmagic.models.losses 模块中*), 605
GradientError (*mmagic.evaluation 中的类*), 495
GradientLoss (*mmagic.models.losses 中的类*), 606
GradientPenaltyLoss (*mmagic.models.losses 中的类*), 604
GradientPenaltyLossComps (*mmagic.models.losses 中的类*), 613
GrowScaleImgDataset (*mmagic.datasets 中的类*), 422
gt_label (*mmagic.structures.DataSample property*), 405
gt_label() (*mmagic.structures.DataSample 方法*), 405

H

HolisticIndexBlock (*mmagic.models.editors 中的类*), 709

I

IconVSRNet (*mmagic.models.editors 中的类*), 706
IDLossModel (*mmagic.models.editors 中的类*), 639
if_run_d() (*mmagic.models.editors.DIC 方法*), 675
if_run_d() (*mmagic.models.editors.SRGAN 方法*), 747
if_run_d() (*mmagic.models.editors.TTSR 方法*), 775
if_run_g() (*mmagic.models.editors.DIC 方法*), 675
if_run_g() (*mmagic.models.editors.SRGAN 方法*), 747
if_run_g() (*mmagic.models.editors.TTSR 方法*), 774
im2col() (*mmagic.models.editors.ContextualAttentionModule 方法*), 666
ImageNet (*mmagic.datasets 中的类*), 424
ImageSuperResolutionInferencer (*mmagic.apis.inferencers 中的类*), 394
IMG_EXTENSIONS (*mmagic.datasets.ImageNet 属性*), 424
img_prefix (*mmagic.datasets.BasicConditionalDataset property*), 411
ImgNormalize (*mmagic.models.archs 中的类*), 550
in_cooldown (*mmagic.engine.schedulers.ReduceLR property*), 543
InceptionScore (*mmagic.evaluation 中的类*), 496
IndexedUpsample (*mmagic.models.editors 中的类*), 709
IndexNet (*mmagic.models.editors 中的类*), 710
IndexNetDecoder (*mmagic.models.editors 中的类*), 711

IndexNetEncoder (*mmagic.models.editors* 中的类), 712
infer() (*mmagic.models.editors.AblatedDiffusionModel* 方法), 705
infer() (*mmagic.models.editors.ControlStableDiffusion* 方法), 650
infer() (*mmagic.models.editors.DiscoDiffusion* 方法), 684
infer() (*mmagic.models.editors.StableDiffusion* 方法), 751
infer() (*mmagic.models.editors.StableDiffusionInpaint* 方法), 756
init_cfg (*mmagic.models.base_models.BaseEditModel* 属性), 574
init_cfg (*mmagic.models.base_models.BasicInterpolator* 属性), 586
init_cfg (*mmagic.models.editors.CAIN* 属性), 646
init_cfg (*mmagic.models.editors.DeblurGanV2* 属性), 659
init_cfg (*mmagic.models.editors.FLAVR* 属性), 697
init_weights() (*mmagic.models.archs.LinearModule* 方法), 551
init_weights() (*mmagic.models.archs.MultiLayerDiscriminator* 方法), 555
init_weights() (*mmagic.models.archs.PatchDiscriminator* 方法), 556
init_weights() (*mmagic.models.archs.PixelShufflePack* 方法), 564
init_weights() (*mmagic.models.archs.ResidualBlockNoBN* 方法), 561
init_weights() (*mmagic.models.archs.ResNet* 方法), 557
init_weights() (*mmagic.models.archs.SoftMaskPatchDiscriminator* 方法), 560
init_weights() (*mmagic.models.archs.VGG16* 方法), 564
init_weights() (*mmagic.models.base_models.BaseTranslationModel* 方法), 584
init_weights() (*mmagic.models.editors.ControlStableDiffusion* 方法), 649
init_weights() (*mmagic.models.editors.DeepFillEncoderDecoder* 方法), 674
init_weights() (*mmagic.models.editors.DeepFillv1Discriminators* 方法), 671
init_weights() (*mmagic.models.editors.DenoisingUnet* 方法), 658
init_weights() (*mmagic.models.editors.DIM* 方法), 681
init_weights() (*mmagic.models.editors.EDVRNet* 方法), 690
init_weights() (*mmagic.models.editors.FBADecoder* 方法), 695
init_weights() (*mmagic.models.editors.IndexedUpsample* 方法), 710
init_weights() (*mmagic.models.editors.IndexNetDecoder* 方法), 712
init_weights() (*mmagic.models.editors.IndexNetEncoder* 方法), 713
init_weights() (*mmagic.models.editors.LightCNN* 方法), 679
init_weights() (*mmagic.models.editors.MSRResNet* 方法), 750
init_weights() (*mmagic.models.editors.PlainDecoder* 方法), 731
init_weights() (*mmagic.models.editors.PlainRefiner* 方法), 732
init_weights() (*mmagic.models.editors.RRDBNet* 方法), 695
init_weights() (*mmagic.models.losses.PerceptualVGG* 方法), 619
InpaintingInferencer (*mmagic.apis.inferencers* 中的类), 395
InstanceCrop (*mmagic.datasets.transforms* 中的类), 452
InstanceColorization (*mmagic.models.editors* 中的类), 713
interpolation() (*mmagic.models.editors.EG3D* 方法), 692
is_better() (*mmagic.engine.schedulers.ReduceLR* 方法), 543
is_domain_reachable() (*mmagic.models.base_models.BaseTranslationModel* 方法), 585
is_valid_file() (*mmagic.datasets.BasicConditionalDataset*

方法), 411
IterTimerHook (*mmagic.engine.hooks* 中的类), 521

L

L1CompositionLoss (*mmagic.models.losses* 中的类), 598
L1Loss (*mmagic.models.losses* 中的类), 620
label_fn() (*mmagic.models.base_models.BaseConditionalGAN* 方法), 572
label_fn() (*mmagic.models.editors.EG3D* 方法), 691
LabelVar() (在 *mmagic.utils* 模块中), 789
lerp() (*mmagic.engine.hooks.ExponentialMovingAverageHook* 静态方法), 520
LightCNN (*mmagic.models.editors* 中的类), 679
LightCNNFeatureLoss (*mmagic.models.losses* 中的类), 600
LIIF (*mmagic.models.editors* 中的类), 716
LinearLrInterval (*mmagic.engine.schedulers* 中的类), 541
LinearModule (*mmagic.models.archs* 中的类), 550
load_data_list() (*mmagic.datasets.AdobeComp1kDataset* 方法), 421
load_data_list() (*mmagic.datasets.BasicConditionalDataset* 方法), 411
load_data_list() (*mmagic.datasets.BasicFramesDataset* 方法), 414
load_data_list() (*mmagic.datasets.BasicImageDataset* 方法), 418
load_data_list() (*mmagic.datasets.CIFAR10* 方法), 419
load_data_list() (*mmagic.datasets.ControlNetDataset* 方法), 421
load_data_list() (*mmagic.datasets.DreamBoothDataset* 方法), 422
load_data_list() (*mmagic.datasets.GrowScaleImgDataset* 方法), 423
load_data_list() (*mmagic.datasets.MSCoCoDataset* 方法), 425
load_data_list() (*mmagic.datasets.PairedImageDataset* 方法), 426
load_data_list() (*mmagic.datasets.SinGANDataset* 方法), 427
load_data_list() (*mmagic.datasets.TextualInversionDataset* 方法), 427
load_data_list() (*mmagic.datasets.UnpairedImageDataset* 方法), 428
load_pretrained_models() (*mmagic.models.editors.AblatedDiffusionModel* 方法), 705
load_pretrained_models() (*mmagic.models.editors.DiscoDiffusion* 方法), 684
load_test_pkl() (*mmagic.models.editors.SinGAN* 方法), 742
LoadImageFromFile (*mmagic.datasets.transforms* 中的类), 465
LoadMask (*mmagic.datasets.transforms* 中的类), 466
LoadPairedImageFromFile (*mmagic.datasets.transforms* 中的类), 467
LogProcessor (*mmagic.engine.runner* 中的类), 535
LoRAWrapper (*mmagic.models.archs* 中的类), 551
loss_name() (*mmagic.models.losses.CLIPLossComps* 静态方法), 607
loss_name() (*mmagic.models.losses.DiscShiftLossComps* 方法), 609
loss_name() (*mmagic.models.losses.FaceIdLossComps* 方法), 610
loss_name() (*mmagic.models.losses.GeneratorPathRegularizerComps* 方法), 613
loss_name() (*mmagic.models.losses.GradientPenaltyLossComps* 方法), 615
loss_name() (*mmagic.models.losses.R1GradientPenaltyComps* 方法), 616
LSGAN (*mmagic.models.editors* 中的类), 717
LTE (*mmagic.models.editors* 中的类), 773

M

MAE (*mmagic.evaluation* 中的类), 483
make_coord() (在 *mmagic.utils* 模块中), 788
mask_correlation_map() (*mmagic.models.editors.ContextualAttentionModule* 方法), 666
mask_reduce_loss() (在 *mmagic.models.losses* 模块中), 616

MaskConvModule (*mmagic.models.editors* 中的类), 722
MaskedTVLoss (*mmagic.models.losses* 中的类), 621
MATLABLikeResize (*mmagic.datasets.transforms* 中的类), 469
MattingInferencer (*mmagic.apis.inferencers* 中的类), 396
MattingMSE (*mmagic.evaluation* 中的类), 498
MattorPreprocessor (*mmagic.models.data_preprocessors* 中的类), 629
MaxFeature (*mmagic.models.editors* 中的类), 679
merge_frames() (*mmagic.models.base_models.BasicInterpolator* 静态方法), 587
merge_frames() (*mmagic.models.editors.FLAVR* 静态方法), 697
MergeFgAndBg (*mmagic.datasets.transforms* 中的类), 456
meta (*mmagic.datasets.CIFAR10* 属性), 419
META_KEYS (*mmagic.structures.DataSample* 属性), 405
METAINFO (*mmagic.datasets.AdobeComp1kDataset* 属性), 421
METAINFO (*mmagic.datasets.BasicFramesDataset* 属性), 414
METAINFO (*mmagic.datasets.BasicImageDataset* 属性), 418
METAINFO (*mmagic.datasets.CIFAR10* 属性), 419
METAINFO (*mmagic.datasets.ImageNet* 属性), 424
METAINFO (*mmagic.datasets.MSCoCoDataset* 属性), 425
metric (*mmagic.evaluation.ConnectivityError* 属性), 490
metric (*mmagic.evaluation.GradientError* 属性), 495
metric (*mmagic.evaluation.MAE* 属性), 483
metric (*mmagic.evaluation.MattingMSE* 属性), 499
metric (*mmagic.evaluation.MSE* 属性), 484
metric (*mmagic.evaluation.NIQE* 属性), 485
metric (*mmagic.evaluation.PSNR* 属性), 486
metric (*mmagic.evaluation.SAD* 属性), 487
metric (*mmagic.evaluation.SNR* 属性), 488
metric (*mmagic.evaluation.SSIM* 属性), 489
MirrorSequence (*mmagic.datasets.transforms* 中的类), 437
MLPRefiner (*mmagic.models.editors* 中的类), 717
mmagic.apis.inferencers 模块, 389
mmagic.datasets 模块, 408
mmagic.datasets.transforms 模块, 431
mmagic.engine.hooks 模块, 519
mmagic.engine.optimizers 模块, 529
mmagic.engine.runner 模块, 535
mmagic.engine.schedulers 模块, 541
mmagic.evaluation 模块, 480
mmagic.models.archs 模块, 546
mmagic.models.base_models 模块, 567
mmagic.models.data_preprocessors 模块, 623
mmagic.models.editors 模块, 633
mmagic.models.losses 模块, 596
mmagic.structures 模块, 403
mmagic.utils 模块, 782
mmagic.visualization 模块, 509
MMAGIC_CACHE_DIR() (在 *mmagic.utils* 模块中), 785
ModCrop (*mmagic.datasets.transforms* 中的类), 453
ModifiedVGG (*mmagic.models.editors* 中的类), 749
modify_args() (在 *mmagic.utils* 模块中), 783
MSCoCoDataset (*mmagic.datasets* 中的类), 424
MSE (*mmagic.evaluation* 中的类), 484
MSECompositionLoss (*mmagic.models.losses* 中的类), 599
MSELoss (*mmagic.models.losses* 中的类), 621

MSPIEStyleGAN2 (*mmagic.models.editors* 中的类), 719
MSRResNet (*mmagic.models.editors* 中的类), 749
MultiLayerDiscriminator (*mmagic.models.archs* 中的类), 554
MultiOptimWrapperConstructor (*mmagic.engine.optimizers* 中的类), 529
MultiScaleStructureSimilarity (*mmagic.evaluation* 中的类), 499
MultiTestLoop (*mmagic.engine.runner* 中的类), 536
MultiValLoop (*mmagic.engine.runner* 中的类), 538

N

NAFBaseline (*mmagic.models.editors* 中的类), 720
NAFBaselineLocal (*mmagic.models.editors* 中的类), 721
NAFNet (*mmagic.models.editors* 中的类), 721
NAFNetLocal (*mmagic.models.editors* 中的类), 721
name (*mmagic.evaluation.Equivariance* 属性), 492
name (*mmagic.evaluation.FrechetInceptionDistance* 属性), 494
name (*mmagic.evaluation.InceptionScore* 属性), 497
name (*mmagic.evaluation.MultiScaleStructureSimilarity* 属性), 500
name (*mmagic.evaluation.PrecisionAndRecall* 属性), 503
name (*mmagic.evaluation.SlicedWassersteinDistance* 属性), 505
NIQE (*mmagic.evaluation* 中的类), 485
no_weight_decay() (*mmagic.models.editors.SwinIRNet* 方法), 767
no_weight_decay_keywords() (*mmagic.models.editors.SwinIRNet* 方法), 767
noise_fn() (*mmagic.models.base_models.BaseGAN* 方法), 578
NoiseVar() (在 *mmagic.utils* 模块中), 789
norm1 (*mmagic.models.archs.ResNet property*), 557
Normalize (*mmagic.datasets.transforms* 中的类), 470
NumpyPad (*mmagic.datasets.transforms* 中的类), 445

O

OneStageInpaintor (*mmagic.models.base_models* 中的类), 587
output_to_pil() (*mmagic.models.editors.StableDiffusion* 方法), 752

P

pack_to_data_sample() (*mmagic.models.editors.EG3D* 方法), 691
PackInputs (*mmagic.datasets.transforms* 中的类), 457
PairedAlbuTransForms (*mmagic.datasets.transforms* 中的类), 434
PairedImageDataset (*mmagic.datasets* 中的类), 425
PairedRandomCrop (*mmagic.datasets.transforms* 中的类), 453
parse_data_info() (*mmagic.datasets.AdobeComp1kDataset* 方法), 421
PartialConv2d (*mmagic.models.editors* 中的类), 723
patch_copy_deconv() (*mmagic.models.editors.ContextualAttentionModule* 方法), 665
patch_correlation() (*mmagic.models.editors.ContextualAttentionModule* 方法), 665
PatchDiscriminator (*mmagic.models.archs* 中的类), 555
PaviVisBackend (*mmagic.visualization* 中的类), 510
PConvDecoder (*mmagic.models.editors* 中的类), 724
PConvEncoder (*mmagic.models.editors* 中的类), 724
PConvEncoderDecoder (*mmagic.models.editors* 中的类), 725
PConvInpaintor (*mmagic.models.editors* 中的类), 726
PerceptualLoss (*mmagic.models.losses* 中的类), 617
PerceptualPathLength (*mmagic.evaluation* 中的类), 500
PerceptualVGG (*mmagic.models.losses* 中的类), 618
PerturbBg (*mmagic.datasets.transforms* 中的类), 456
PESinGAN (*mmagic.models.editors* 中的类), 720

PGGANFetchDataHook (*mmagic.engine.hooks* 中的类), 522
PGGANOptimWrapperConstructor (*mmagic.engine.optimizers* 中的类), 532
PickleDataHook (*mmagic.engine.hooks* 中的类), 522
pil_resize_method_mapping (*mmagic.evaluation.InceptionScore* 属性), 497
Pix2Pix (*mmagic.models.editors* 中的类), 729
pixel_unshuffle() (在 *mmagic.models.archs* 模块中), 548
PixelShufflePack (*mmagic.models.archs* 中的类), 563
PlainDecoder (*mmagic.models.editors* 中的类), 731
PlainRefiner (*mmagic.models.editors* 中的类), 731
postprocess() (*mmagic.apis.inferencers.EG3DInferencer* 方法), 394
postprocess() (*mmagic.apis.inferencers.VideoInterpolationInferencer* 方法), 401
postprocess() (*mmagic.apis.inferencers.VideoRestorationInferencer* 方法), 402
postprocess() (*mmagic.models.base_models.BaseMattor* 方法), 582
PrecisionAndRecall (*mmagic.evaluation* 中的类), 502
predict_bbox() (*mmagic.datasets.transforms.InstanceCrop* 方法), 453
prepare() (*mmagic.evaluation.ConnectivityError* 方法), 490
prepare() (*mmagic.evaluation.FrechetInceptionDistance* 方法), 494
prepare() (*mmagic.evaluation.GradientError* 方法), 495
prepare() (*mmagic.evaluation.InceptionScore* 方法), 497
prepare() (*mmagic.evaluation.MattingMSE* 方法), 499
prepare() (*mmagic.evaluation.PrecisionAndRecall* 方法), 504
prepare() (*mmagic.evaluation.SAD* 方法), 487
prepare_control() (*mmagic.models.editors.ControlStableDiffusion* 静态方法), 650
prepare_data() (*mmagic.datasets.TextualInversionDataset* 方法), 427
prepare_extra_step_kwargs() (*mmagic.models.editors.StableDiffusion* 方法), 753
prepare_latents() (*mmagic.models.editors.StableDiffusion* 方法), 754
prepare_mask_latents() (*mmagic.models.editors.StableDiffusionInpaint* 方法), 758
prepare_metrics() (*mmagic.evaluation.Evaluator* 方法), 481
prepare_model() (*mmagic.models.editors.DreamBooth* 方法), 687
prepare_models() (*mmagic.models.editors.TextualInversion* 方法), 771
prepare_samplers() (*mmagic.evaluation.Evaluator* 方法), 481
prepare_test_data() (*mmagic.datasets.GrowScaleImgDataset* 方法), 423
prepare_test_scheduler_extra_step_kwargs() (*mmagic.models.editors.StableDiffusion* 方法), 754
prepare_train_data() (*mmagic.datasets.GrowScaleImgDataset* 方法), 423
preprocess() (*mmagic.apis.inferencers.ColorizationInferencer* 方法), 390
preprocess() (*mmagic.apis.inferencers.ConditionalInferencer* 方法), 391
preprocess() (*mmagic.apis.inferencers.EG3DInferencer* 方法), 392
preprocess() (*mmagic.apis.inferencers.ImageSuperResolutionInferencer* 方法), 395
preprocess() (*mmagic.apis.inferencers.InpaintingInferencer* 方法), 396
preprocess() (*mmagic.apis.inferencers.MattingInferencer* 方法), 396
preprocess() (*mmagic.apis.inferencers.Text2ImageInferencer* 方法), 397

preprocess() (*mmagic.apis.inferencers.TranslationInferencer* 方法), 398
preprocess() (*mmagic.apis.inferencers.UnconditionalInferencer* 方法), 399
preprocess() (*mmagic.apis.inferencers.VideoInterpolationInferencer* 方法), 400
preprocess() (*mmagic.apis.inferencers.VideoRestorationInferencer* 方法), 401
preprocess_depth() (*mmagic.apis.inferencers.EG3DInferencer* 方法), 394
preprocess_img() (*mmagic.apis.inferencers.EG3DInferencer* 方法), 394
print_colored_log() (在 *mmagic.utils* 模块中), 785
priority (*mmagic.engine.hooks.BasicVisualizationHook* 属性), 525
priority (*mmagic.engine.hooks.VisualizationHook* 属性), 527
process() (*mmagic.evaluation.ConnectivityError* 方法), 490
process() (*mmagic.evaluation.Equivariance* 方法), 492
process() (*mmagic.evaluation.Evaluator* 方法), 482
process() (*mmagic.evaluation.FrechetInceptionDistance* 方法), 494
process() (*mmagic.evaluation.GradientError* 方法), 495
process() (*mmagic.evaluation.InceptionScore* 方法), 498
process() (*mmagic.evaluation.MattingMSE* 方法), 499
process() (*mmagic.evaluation.MultiScaleStructureSimilarity* 方法), 500
process() (*mmagic.evaluation.PerceptualPathLength* 方法), 501
process() (*mmagic.evaluation.PrecisionAndRecall* 方法), 504
process() (*mmagic.evaluation.SAD* 方法), 487
process() (*mmagic.evaluation.SlicedWassersteinDistance* 方法), 505
process_image() (*mmagic.evaluation.MAE* 方法), 483
process_image() (*mmagic.evaluation.MSE* 方法), 484
process_image() (*mmagic.evaluation.NIQE* 方法), 485
process_image() (*mmagic.evaluation.PSNR* 方法), 486
process_image() (*mmagic.evaluation.SNR* 方法), 488
process_image() (*mmagic.evaluation.SSIM* 方法), 490
ProgressiveGrowingGAN (*mmagic.models.editors* 中的类), 727
propagate() (*mmagic.models.editors.BasicVSRPlusPlusNet* 方法), 643
PSNR (*mmagic.evaluation* 中的类), 486
PSNRLoss (*mmagic.models.losses* 中的类), 622

R

r1_gradient_penalty_loss() (在 *mmagic.models.losses* 模块中), 606
R1GradientPenaltyComps (*mmagic.models.losses* 中的类), 615
rampup() (*mmagic.models.base_models.RampUpEMA* 静态方法), 570
RampUpEMA (*mmagic.models.base_models* 中的类), 569
random_bbox() (在 *mmagic.utils* 模块中), 788
random_choose_unknown() (在 *mmagic.utils* 模块中), 789
RandomAffine (*mmagic.datasets.transforms* 中的类), 441
RandomBlur (*mmagic.datasets.transforms* 中的类), 471
RandomCropLongEdge (*mmagic.datasets.transforms* 中的类), 453
RandomDownSampling (*mmagic.datasets.transforms* 中的类), 474
RandomJitter (*mmagic.datasets.transforms* 中的类), 456
RandomJPEGCompression (*mmagic.datasets.transforms* 中的类), 472
RandomLoadResizeBg (*mmagic.datasets.transforms* 中的类), 457
RandomMaskDilation (*mmagic.datasets.transforms*

中的类), 443
RandomNoise (*mmagic.datasets.transforms* 中的类), 472
RandomResize (*mmagic.datasets.transforms* 中的类), 473
RandomResizedCrop (*mmagic.datasets.transforms* 中的类), 454
RandomRotation (*mmagic.datasets.transforms* 中的类), 445
RandomTransposeHW (*mmagic.datasets.transforms* 中的类), 446
RandomVideoCompression (*mmagic.datasets.transforms* 中的类), 473
RDNNet (*mmagic.models.editors* 中的类), 732
RealBasicVSR (*mmagic.models.editors* 中的类), 733
RealBasicVSRNet (*mmagic.models.editors* 中的类), 734
RealESRGAN (*mmagic.models.editors* 中的类), 735
reduce_loss() (在 *mmagic.models.losses* 模块中), 617
ReduceLR (*mmagic.engine.schedulers* 中的类), 541
ReduceLRSchedulerHook (*mmagic.engine.hooks* 中的类), 523
register_all_modules() (在 *mmagic.utils* 模块中), 785
reorder_image() (在 *mmagic.utils* 模块中), 784
replace_placeholder_tokens_in_text() (*mmagic.models.archs.TokenizerWrapper* 方法), 562
replace_text_with_placeholder_tokens() (*mmagic.models.archs.TokenizerWrapper* 方法), 562
RescaleToZeroOne (*mmagic.datasets.transforms* 中的类), 470
ResidualBlockNoBN (*mmagic.models.archs* 中的类), 560
Resize (*mmagic.datasets.transforms* 中的类), 446
resize_inputs() (*mmagic.models.base_models.BaseMattor* 方法), 582
ResNet (*mmagic.models.archs* 中的类), 556
restore_size() (*mmagic.models.base_models.BaseMattor* 方法), 582
Restormer (*mmagic.models.editors* 中的类), 738
RRDBNet (*mmagic.models.editors* 中的类), 694
run() (*mmagic.engine.runner.MultiTestLoop* 方法), 537
run() (*mmagic.engine.runner.MultiValLoop* 方法), 540
run_iter() (*mmagic.engine.runner.MultiTestLoop* 方法), 538
run_iter() (*mmagic.engine.runner.MultiValLoop* 方法), 540

S

SAD (*mmagic.evaluation* 中的类), 487
SAGAN (*mmagic.models.editors* 中的类), 739
sample_equivarience_pairs() (*mmagic.models.editors.StyleGAN3* 方法), 764
SampleList() (在 *mmagic.utils* 模块中), 789
SAMPLER_MODE (*mmagic.evaluation.PerceptualPathLength* 属性), 501
scan_folder() (*mmagic.datasets.PairedImageDataset* 方法), 426
scan_folder() (*mmagic.datasets.UnpairedImageDataset* 方法), 429
SearchTransformer (*mmagic.models.editors* 中的类), 775
set_disable() (*mmagic.models.archs.LoRAWrapper* 方法), 552
set_embedding_layer() (*mmagic.models.editors.ClipWrapper* 方法), 683
set_enable() (*mmagic.models.archs.LoRAWrapper* 方法), 552
set_gt_label() (*mmagic.structures.DataSample* 方法), 405
set_lora() (*mmagic.models.editors.DreamBooth* 方法), 687
set_lora() (在 *mmagic.models.archs* 模块中), 553
set_lora_disable() (在 *mmagic.models.archs* 模块中), 554
set_lora_enable() (在 *mmagic.models.archs* 模块中), 554
set_only_embedding_trainable() (*mmagic.models.editors.ClipWrapper* 方法),

683
set_only_lora_trainable() (在 mmagic.models.archs 模块中), 554
set_predefined_data() (mmagic.structures.DataSample 方法), 405
set_scale() (mmagic.models.archs.LoRAWrapper 方法), 552
set_tensor_data() (mmagic.structures.DataSample 方法), 405
set_tomesd() (mmagic.models.editors.StableDiffusion 方法), 751
set_xformers() (mmagic.models.editors.StableDiffusion 方法), 751
SetValues (mmagic.datasets.transforms 中的类), 478
SimpleEncoderDecoder (mmagic.models.archs 中的类), 559
SimpleGatedConvModule (mmagic.models.archs 中的类), 549
SinGAN (mmagic.models.editors 中的类), 741
SinGANDataset (mmagic.datasets 中的类), 426
SinGANOptimWrapperConstructor (mmagic.engine.optimizers 中的类), 533
SlicedWassersteinDistance (mmagic.evaluation 中的类), 504
SNR (mmagic.evaluation 中的类), 488
SoftMaskPatchDiscriminator (mmagic.models.archs 中的类), 559
spatial_discount_mask() (mmagic.datasets.transforms.GetSpatialDiscountMask 方法), 464
spatial_ensemble() (mmagic.models.archs.SpatialTemporalEnsemble 方法), 549
spatial_padding() (mmagic.models.editors.IconVSRNet 方法), 707
SpatialTemporalEnsemble (mmagic.models.archs 中的类), 548
split() (mmagic.structures.DataSample 方法), 406
split_frames() (mmagic.models.base_models.BasicInterpolator 方法), 587
SRCNNNet (mmagic.models.editors 中的类), 745
SRGAN (mmagic.models.editors 中的类), 746
SSIM (mmagic.evaluation 中的类), 489
StableDiffusion (mmagic.models.editors 中的类), 750
StableDiffusionInpaint (mmagic.models.editors 中的类), 755
stack() (mmagic.structures.DataSample 类方法), 405
StyleGAN1 (mmagic.models.editors 中的类), 759
StyleGAN2 (mmagic.models.editors 中的类), 760
StyleGAN3 (mmagic.models.editors 中的类), 763
StyleGAN3Generator (mmagic.models.editors 中的类), 764
supported_conv_list (mmagic.models.editors.MaskConvModule 属性), 723
SwinIRNet (mmagic.models.editors 中的类), 766
sync_buffers() (mmagic.models.base_models.ExponentialMovingAverag 方法), 569
sync_buffers() (mmagic.models.base_models.RampUpEMA 方法), 571
sync_parameters() (mmagic.models.base_models.ExponentialMovingAverage 方法), 569
sync_parameters() (mmagic.models.base_models.RampUpEMA 方法), 571

T

TDAN (mmagic.models.editors 中的类), 768
TDANNet (mmagic.models.editors 中的类), 769
TemporalReverse (mmagic.datasets.transforms 中的类), 438
tensor2img() (在 mmagic.utils 模块中), 784
TensorboardVisBackend (mmagic.visualization 中的类), 511
test_list (mmagic.datasets.CIFAR10 属性), 419
test_step() (mmagic.models.base_models.BaseGAN 方法), 580
test_step() (mmagic.models.editors.AblatedDiffusionModel 方法), 706
test_step() (mmagic.models.editors.ControlStableDiffusion 方法), 649

test_step() (*mmagic.models.editors.CycleGAN 方法*), 653
test_step() (*mmagic.models.editors.DeblurGanV2 方法*), 662
test_step() (*mmagic.models.editors.DreamBooth 方法*), 687
test_step() (*mmagic.models.editors.Pix2Pix 方法*), 730
test_step() (*mmagic.models.editors.SinGAN 方法*), 745
test_step() (*mmagic.models.editors.StableDiffusion 方法*), 755
test_step() (*mmagic.models.editors.StableDiffusionInpaint 方法*), 759
test_step() (*mmagic.models.editors.StyleGAN3 方法*), 763
test_step() (*mmagic.models.editors.TextualInversion 方法*), 771
Text2ImageInferencer (*mmagic.apis.inferencers 中的类*), 397
TextualInversion (*mmagic.models.editors 中的类*), 769
TextualInversionDataset (*mmagic.datasets 中的类*), 427
tgz_md5 (*mmagic.datasets.CIFAR10 属性*), 419
to_numpy() (*在 mmagic.utils 模块中*), 784
TOFlowVFINet (*mmagic.models.editors 中的类*), 771
TOFlowVSRNet (*mmagic.models.editors 中的类*), 772
ToFResBlock (*mmagic.models.editors 中的类*), 772
TokenizerWrapper (*mmagic.models.archs 中的类*), 561
total_length (*mmagic.engine.runner.MultiTestLoop property*), 537
total_length (*mmagic.engine.runner.MultiValLoop property*), 539
train() (*mmagic.models.editors.ControlStableDiffusion 方法*), 650
train() (*mmagic.models.editors.DIM 方法*), 681
train() (*mmagic.models.editors.IndexNetEncoder 方法*), 713
train() (*mmagic.models.editors.PConvEncoder 方法*), 725
train() (*mmagic.models.editors.StableDiffusion 方法*), 751
train_discriminator() (*mmagic.models.base_models.BaseConditionalGAN 方法*), 573
train_discriminator() (*mmagic.models.base_models.BaseGAN 方法*), 581
train_discriminator() (*mmagic.models.editors.BigGAN 方法*), 645
train_discriminator() (*mmagic.models.editors.DCGAN 方法*), 654
train_discriminator() (*mmagic.models.editors.GGAN 方法*), 700
train_discriminator() (*mmagic.models.editors.LSGAN 方法*), 718
train_discriminator() (*mmagic.models.editors.MSPIEStyleGAN2 方法*), 719
train_discriminator() (*mmagic.models.editors.ProgressiveGrowingGAN 方法*), 728
train_discriminator() (*mmagic.models.editors.SAGAN 方法*), 740
train_discriminator() (*mmagic.models.editors.SinGAN 方法*), 744
train_discriminator() (*mmagic.models.editors.StyleGAN2 方法*), 762
train_discriminator() (*mmagic.models.editors.StyleGAN3 方法*), 763
train_discriminator() (*mmagic.models.editors.WGANGP 方法*), 778
train_gan() (*mmagic.models.editors.SinGAN 方法*), 744
train_generator() (*mmagic.models.base_models.BaseConditionalGAN 方法*), 573
train_generator() (*mmagic.models.base_models.BaseGAN 方*

法), 580
train_generator() (*mmagic.models.editors.BigGAN* 方法), 645
train_generator() (*mmagic.models.editors.DCGAN* 方法), 654
train_generator() (*mmagic.models.editors.GGAN* 方法), 700
train_generator() (*mmagic.models.editors.LSGAN* 方法), 718
train_generator() (*mmagic.models.editors.MSPIEStyleGAN2* 方法), 719
train_generator() (*mmagic.models.editors.ProgressiveGrowingGAN* 方法), 728
train_generator() (*mmagic.models.editors.SAGAN* 方法), 740
train_generator() (*mmagic.models.editors.SinGAN* 方法), 743
train_generator() (*mmagic.models.editors.StyleGAN2* 方法), 762
train_generator() (*mmagic.models.editors.StyleGAN3* 方法), 764
train_generator() (*mmagic.models.editors.WGANGP* 方法), 779
train_list (*mmagic.datasets.CIFAR10* 属性), 419
train_step() (*mmagic.models.base_models.BaseGAN* 方法), 580
train_step() (*mmagic.models.base_models.OneStageInpaintor* 方法), 589
train_step() (*mmagic.models.base_models.TwoStageInpaintor* 方法), 594
train_step() (*mmagic.models.editors.AblatedDiffusionModel* 方法), 706
train_step() (*mmagic.models.editors.AOTInpaintor* 方法), 639
train_step() (*mmagic.models.editors.ControlStableDiffusion* 方法), 649
train_step() (*mmagic.models.editors.CycleGAN* 方法), 652
train_step() (*mmagic.models.editors.DeblurGanV2* 方法), 662
train_step() (*mmagic.models.editors.DeepFillv1Inpaintor* 方法), 673
train_step() (*mmagic.models.editors.DIC* 方法), 676
train_step() (*mmagic.models.editors.DreamBooth* 方法), 687
train_step() (*mmagic.models.editors.InstColorization* 方法), 715
train_step() (*mmagic.models.editors.MSPIEStyleGAN2* 方法), 719
train_step() (*mmagic.models.editors.PConvInpaintor* 方法), 726
train_step() (*mmagic.models.editors.Pix2Pix* 方法), 730
train_step() (*mmagic.models.editors.ProgressiveGrowingGAN* 方法), 729
train_step() (*mmagic.models.editors.RealBasicVSR* 方法), 734
train_step() (*mmagic.models.editors.SinGAN* 方法), 745
train_step() (*mmagic.models.editors.SRGAN* 方法), 748
train_step() (*mmagic.models.editors.StableDiffusion* 方法), 755
train_step() (*mmagic.models.editors.StableDiffusionInpaint* 方法), 759
train_step() (*mmagic.models.editors.StyleGAN2* 方法), 762
train_step() (*mmagic.models.editors.TextualInversion* 方法), 771
train_step() (*mmagic.models.editors.TTSR* 方法), 775
TransferalPerceptualLoss (*mmagic.models.losses* 中的类), 619
TransFID (*mmagic.evaluation* 中的类), 505
transform() (*mmagic.datasets.transforms.AlbuCorruptFunction* 方法), 433
transform() (*mmagic.datasets.transforms.Albumentations* 方法), 436
transform() (*mmagic.datasets.transforms.BinarizeImage*

方法), 439
transform() (mmagic.datasets.transforms.CenterCropLongEdge 方法), 448
transform() (mmagic.datasets.transforms.Clip 方法), 440
transform() (mmagic.datasets.transforms.ColorJitter 方法), 441
transform() (mmagic.datasets.transforms.CompositeFg 方法), 455
transform() (mmagic.datasets.transforms.CopyValues 方法), 477
transform() (mmagic.datasets.transforms.Crop 方法), 449
transform() (mmagic.datasets.transforms.CropAroundCenter 方法), 449
transform() (mmagic.datasets.transforms.CropAroundFg 方法), 450
transform() (mmagic.datasets.transforms.CropAroundUnknown 方法), 451
transform() (mmagic.datasets.transforms.CropLike 方法), 451
transform() (mmagic.datasets.transforms.FixedCrop 方法), 452
transform() (mmagic.datasets.transforms.Flip 方法), 444
transform() (mmagic.datasets.transforms.FormatTrimap 方法), 475
transform() (mmagic.datasets.transforms.GenerateCoordinateAndCell 方法), 459
transform() (mmagic.datasets.transforms.GenerateFacialHeatmap 方法), 460
transform() (mmagic.datasets.transforms.GenerateFrameIndices 方法), 461
transform() (mmagic.datasets.transforms.GenerateFrameIndicesWithPadding 方法), 462
transform() (mmagic.datasets.transforms.GenerateSeg 方法), 436
transform() (mmagic.datasets.transforms.GenerateSegmentIndices 方法), 463
transform() (mmagic.datasets.transforms.GenerateSoftSeg 方法), 437
transform() (mmagic.datasets.transforms.GenerateTrimap 方法), 476
transform() (mmagic.datasets.transforms.GenerateTrimapWithDistTrans 方法), 476
transform() (mmagic.datasets.transforms.GetMaskedImage 方法), 464
transform() (mmagic.datasets.transforms.GetSpatialDiscountMask 方法), 464
transform() (mmagic.datasets.transforms.InstanceCrop 方法), 452
transform() (mmagic.datasets.transforms.LoadImageFromFile 方法), 465
transform() (mmagic.datasets.transforms.LoadMask 方法), 467
transform() (mmagic.datasets.transforms.LoadPairedImageFromFile 方法), 469
transform() (mmagic.datasets.transforms.MATLABLikeResize 方法), 469
transform() (mmagic.datasets.transforms.MergeFgAndBg 方法), 456
transform() (mmagic.datasets.transforms.MirrorSequence 方法), 438
transform() (mmagic.datasets.transforms.ModCrop 方法), 453
transform() (mmagic.datasets.transforms.Normalize 方法), 470
transform() (mmagic.datasets.transforms.NumpyPad 方法), 445
transform() (mmagic.datasets.transforms.PackInputs 方法), 458
transform() (mmagic.datasets.transforms.PairedAlbuTransForms 方法), 434
transform() (mmagic.datasets.transforms.PairedRandomCrop 方法), 453
transform() (mmagic.datasets.transforms.PerturbBg 方法), 456
transform() (mmagic.datasets.transforms.RandomAffine 方法), 443
transform() (mmagic.datasets.transforms.RandomCropLongEdge 方法), 454
transform() (mmagic.datasets.transforms.RandomDownSampling 方法), 474
transform() (mmagic.datasets.transforms.RandomJitter

方法), 457
transform() (*mmagic.datasets.transforms.RandomLoadResizeBg* 方法), 457
transform() (*mmagic.datasets.transforms.RandomMaskDilation* 方法), 443
transform() (*mmagic.datasets.transforms.RandomResizedCrop* 方法), 455
transform() (*mmagic.datasets.transforms.RandomRotation* 方法), 446
transform() (*mmagic.datasets.transforms.RandomTranspose* 方法), 446
transform() (*mmagic.datasets.transforms.RescaleToZeroOne* 方法), 470
transform() (*mmagic.datasets.transforms.Resize* 方法), 448
transform() (*mmagic.datasets.transforms.SetValues* 方法), 478
transform() (*mmagic.datasets.transforms.TemporalReverse* 方法), 438
transform() (*mmagic.datasets.transforms.TransformTrimap* 方法), 477
transform() (*mmagic.datasets.transforms.UnsharpMasking* 方法), 444
TransformTrimap (*mmagic.datasets.transforms* 中的类), 476
TransIS (*mmagic.evaluation* 中的类), 507
translation() (*mmagic.models.base_models.BaseTranslationModel* 方法), 586
TranslationInferencer (*mmagic.apis.inferencers* 中的类), 398
try_adding_tokens() (*mmagic.models.archs.TokenizerWrapper* 方法), 561
try_import() (在 *mmagic.utils* 模块中), 786
TTSR (*mmagic.models.editors* 中的类), 773
TTSRDiscriminator (*mmagic.models.editors* 中的类), 776
TTSRNet (*mmagic.models.editors* 中的类), 777
tv_loss() (在 *mmagic.models.losses* 模块中), 622
two_stage_loss() (*mmagic.models.base_models.TwoStageInpaintor* 方法), 593
two_stage_loss() (*mmagic.models.editors.DeepFillv1Inpaintor* 方法), 672
TwoStageInpaintor (*mmagic.models.base_models* 中的类), 592

U

UnconditionalInferencer (*mmagic.apis.inferencers* 中的类), 399
UNetDiscriminatorWithSpectralNorm (*mmagic.models.editors* 中的类), 738
UnpairedImageDataset (*mmagic.datasets* 中的类), 428
unset_embedding_layer() (*mmagic.models.editors.ClipWrapper* 方法), 683
UnsharpMasking (*mmagic.datasets.transforms* 中的类), 443
update_annotations() (*mmagic.datasets.GrowScaleImgDataset* 方法), 423
update_dataloader() (*mmagic.engine.hooks.PGGANFetchDataHook* 方法), 522
upsample() (*mmagic.models.editors.BasicVSRPlusPlusNet* 方法), 644
url (*mmagic.datasets.CIFAR10* 属性), 419

V

val_step() (*mmagic.models.base_models.BaseGAN* 方法), 580
val_step() (*mmagic.models.editors.AblatedDiffusionModel* 方法), 706
val_step() (*mmagic.models.editors.ControlStableDiffusion* 方法), 649
val_step() (*mmagic.models.editors.CycleGAN* 方法), 653
val_step() (*mmagic.models.editors.DeblurGanV2* 方法), 661
val_step() (*mmagic.models.editors.DreamBooth* 方法), 687
val_step() (*mmagic.models.editors.Pix2Pix* 方法), 731
val_step() (*mmagic.models.editors.StableDiffusion* 方法), 754

val_step() (*mmagic.models.editors.StableDiffusionInpaint* 方法), 758
val_step() (*mmagic.models.editors.StyleGAN3* 方法), 763
val_step() (*mmagic.models.editors.TextualInversion* 方法), 771
VGG16 (*mmagic.models.archs* 中的类), 564
VideoInterpolationInferencer (*mmagic.apis.inferencers* 中的类), 400
VideoRestorationInferencer (*mmagic.apis.inferencers* 中的类), 401
vis_from_message_hub() (*mmagic.engine.hooks.VisualizationHook* 方法), 528
VIS_KWARGS_MAPPING (*mmagic.engine.hooks.VisualizationHook* 属性), 527
vis_sample() (*mmagic.engine.hooks.VisualizationHook* 方法), 528
VisBackend (*mmagic.visualization* 中的类), 512
VisualizationHook (*mmagic.engine.hooks* 中的类), 525
visualize() (*mmagic.apis.inferencers.ColorizationInferencer* 方法), 390
visualize() (*mmagic.apis.inferencers.ConditionalInferencer* 方法), 391
visualize() (*mmagic.apis.inferencers.EG3DInferencer* 方法), 393
visualize() (*mmagic.apis.inferencers.ImageSuperResolutionInferencer* 方法), 395
visualize() (*mmagic.apis.inferencers.InpaintingInferencer* 方法), 396
visualize() (*mmagic.apis.inferencers.MattingInferencer* 方法), 397
visualize() (*mmagic.apis.inferencers.Text2ImageInferencer* 方法), 398
visualize() (*mmagic.apis.inferencers.TranslationInferencer* 方法), 398
visualize() (*mmagic.apis.inferencers.UnconditionalInferencer* 方法), 399
visualize() (*mmagic.apis.inferencers.VideoInterpolationInferencer* 方法), 400
visualize() (*mmagic.apis.inferencers.VideoRestorationInferencer* 方法), 401
Visualizer (*mmagic.visualization* 中的类), 515

W

WandbVisBackend (*mmagic.visualization* 中的类), 514
WGANGP (*mmagic.models.editors* 中的类), 778
with_ema_gen (*mmagic.models.base_models.BaseGAN property*), 577
with_refiner (*mmagic.models.editors.DIM property*), 681
wrap_lora() (*mmagic.models.archs.LoRAWrapper* 类方法), 553

模块

mmagic.apis.inferencers, 389
mmagic.datasets, 408
mmagic.datasets.transforms, 431
mmagic.engine.hooks, 519
mmagic.engine.optimizers, 529
mmagic.engine.runner, 535
mmagic.engine.schedulers, 541
mmagic.evaluation, 480
mmagic.models.archs, 546
mmagic.models.base_models, 567
mmagic.models.data_preprocessors, 623
mmagic.models.editors, 633
mmagic.models.losses, 596
mmagic.structures, 403
mmagic.utils, 782
mmagic.visualization, 509