MMEditing

MMEditing Authors

2022 年 02 月 16 日

开始你的第一步

您可以在页面左下角切换中英文文档。

CHAPTER 1

依赖

- Linux (目前 Windows 暂无官方支持)
- Python 3.6+
- PyTorch 1.3 或更高
- CUDA 9.0 或更高
- NCCL 2
- GCC 5.4 或更高
- mmcv

CHAPTER 2

安装

a. 创建并激活 conda 虚拟环境，如：

```
conda create -n open-mmlab python=3.7 -y
conda activate open-mmlab
```

b. 按照 PyTorch 官方文档 安装 PyTorch 和 torchvision，如：

```
conda install pytorch torchvision -c pytorch
```

注：确保 CUDA 编译版本和 CUDA 运行版本相匹配。用户可以参照 PyTorch 官网 对预编译包所支持的 CUDA 版本进行核对。

例 1：如果 `/usr/local/cuda` 文件夹下已安装了 CUDA 10.1 版本，则需要安装 CUDA 10.1 下预编译的 PyTorch。

```
conda install pytorch cudatoolkit=10.1 torchvision -c pytorch
```

例 2：如果你在 `/usr/local/cuda` 文件夹下已安装 CUDA 9.2 版本，并且想要安装 PyTorch 1.3.1 版本，则需要安装 CUDA 9.2 下预编译的 PyTorch。

```
conda install pytorch=1.3.1 cudatoolkit=9.2 torchvision=0.4.2 -c pytorch
```

如果你从源码编译 PyTorch 而不是安装的预编译版本的话，你可以使用更多 CUDA 版本（例如 9.0）。

c. 克隆 MMEditing 仓库

```
git clone https://github.com/open-mmlab/mmediting.git
cd mmediting
```

d. 安装相关依赖和 MMEditing

```
pip install -r requirements.txt
pip install -v -e .  # or "python setup.py develop"
```

如果你是在 macOS 环境下安装，则需将上述命令的最后一行替换为：

```
CC=clang CXX=clang++ CFLAGS='-stdlib=libc++' pip install -e .
```

注：

1. git commit 的 id 将会被写到版本号中，如 0.6.0+2e7045c。这个版本号也会被保存到训练好的模型中。推荐用户每次在对本地代码和 github 上的源码进行同步后，执行一次步骤 b。如果 C++/CUDA 代码被修改，就必须进行这一步骤。

   重要：如果你在一个新的 CUDA/PyTorch 版本下重新安装了 mmedit，请确保删除了 `./build` 文件夹

   ```
   pip uninstall mmedit
   rm -rf ./build
   find . -name "*.so" | xargs rm
   ```

2. 根据上述步骤，MMEditing 就会以 `dev` 模式被安装，任何本地的代码修改都会立刻生效，不需要再重新安装一遍（除非用户提交了 commits，并且想更新版本号）。

3. 如果用户想使用 `opencv-python-headless` 而不是 `opencv-python`，可在安装 `MMCV` 前安装 `opencv-python-headless`。

4. 有些模型（例如图像修复任务中的 `EDVR`）依赖于 `mmcv-full` 中的一些 CUDA 算子，具体的依赖可在 `requirements.txt` 中找到。如需要，请通过 `pip install -r requirements.txt` 命令来安装 `mmcv-full`，安装过程中会在本地编译 CUDA 算子，这个过程大概需要 10 分钟。另一种方案是安装预编译版本的 `mmcv-full`，请参考 MMCV 主页 获取具体的安装指令。此外，如果你要使用的模型不依赖于 CUDA 算子，那么也可以使用 `pip install mmcv` 来安装轻量版本的 mmcv，其中 CUDA 算子被移除了。

2.1 CPU 环境下的安装步骤

MMEditing 也可以在只有 CPU 的环境下安装（即无法使用 GPU 的环境）。

然而在该环境下，有些功能将被移除，例如：

- Deformable Convolution（可变形卷积）

因此，如果您尝试使用一些包含可变形卷积的模型进行推理，则会出现错误。

2.2 利用 Docker 镜像安装 MMEditing

MMEditing 提供了一个 Dockerfile 来创建 docker 镜像

```
# build an image with PyTorch 1.5, CUDA 10.1
docker build -t mmediting docker/
```

运行以下命令：

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

2.3 完整的安装脚本

```
conda create -n open-mmlab python=3.7 -y
conda activate open-mmlab

# install latest pytorch prebuilt with the default prebuilt CUDA version (usually the␣
↪latest)
conda install -c pytorch pytorch torchvision -y
git clone https://github.com/open-mmlab/mmediting.git
cd mmediting
pip install -r requirements.txt
pip install -v -e .
```

CHAPTER 3

演示

我们针对特定任务提供了一些脚本，可以对单张图像进行推理。

3.1 图像补全

您可以使用以下命令，输入一张测试图像以及缺损部位的遮罩图像，实现对测试图像的补全。

```
python demo/inpainting_demo.py ${CONFIG_FILE} ${CHECKPOINT_FILE} ${MASKED_IMAGE_FILE}
↪${MASK_FILE} ${SAVE_FILE} [--imshow] [--device ${GPU_ID}]
```

如果指定了–imshow ，演示程序将使用 opencv 显示图像。例子：

```
python demo/inpainting_demo.py configs/inpainting/global_local/gl_256x256_8x12_celeba.
↪py work_dirs/inpainting/global_local/gl_256x256_8x12_celeba_20200619-5af0493f.pth␣
↪tests/data/image/celeba_test.png tests/data/image/bbox_mask.png tests/data/pred/
↪inpainting_celeba.png
```

补全结果将保存在 `tests/data/pred/inpainting_celeba.png` 中。

3.2 抠图

您可以使用以下命令，输入一张测试图像以及对应的三元图（trimap），实现对测试图像的抠图。

```
python demo/matting_demo.py ${CONFIG_FILE} ${CHECKPOINT_FILE} ${IMAGE_FILE} ${TRIMAP_
↪FILE} ${SAVE_FILE} [--imshow] [--device ${GPU_ID}]
```

如果指定了–imshow，演示程序将使用 opencv 显示图像。例子：

```
python demo/matting_demo.py configs/mattors/dim/dim_stage3_v16_pln_1x1_1000k_comp1k.
↪py work_dirs/dim_stage3/latest.pth tests/data/merged/GT05.jpg tests/data/trimap/
↪GT05.png tests/data/pred/GT05.png
```

预测的 alpha 遮罩将保存在 `tests/data/pred/GT05.png` 中。

3.3 图像超分辨率

您可以使用以下命令来测试要恢复的图像。

```
python demo/restoration_demo.py ${CONFIG_FILE} ${CHECKPOINT_FILE} ${IMAGE_FILE} $
↪{SAVE_FILE} [--imshow] [--device ${GPU_ID}]
```

如果指定了–imshow，演示程序将使用 opencv 显示图像。例子：

```
python demo/restoration_demo.py configs/restorers/esrgan/esrgan_x4c64b23g32_g1_400k_
↪div2k.py work_dirs/esrgan_x4c64b23g32_g1_400k_div2k/latest.pth tests/data/lq/baboon_
↪x4.png demo/demo_out_baboon.png
```

3.4 人脸图像超分辨率

您可以使用以下命令来测试要恢复的人脸图像。

```
python demo/restoration_face_demo.py ${CONFIG_FILE} ${CHECKPOINT_FILE} ${IMAGE_FILE} $
↪{SAVE_FILE} [--upscale_factor] [--face_size] [--imshow] [--device ${GPU_ID}]
```

如果指定了–imshow，演示程序将使用 opencv 显示图像。例子：

```
python demo/restoration_face_demo.py configs/restorers/glean/glean_in128out1024_2x4_
↪300k_ffhq_celebahq.py https://download.openmmlab.com/mmediting/restorers/glean/
↪glean_in128out1024_4x2_300k_ffhq_celebahq_20210812-acbcb04f.pth tests/data/face/
↪000001.png results/000001.png --upscale_factor 4
```

3.5 视频超分辨率

您可以使用以下命令来测试视频以进行恢复。

```
python demo/restoration_video_demo.py ${CONFIG_FILE} ${CHECKPOINT_FILE} ${INPUT_DIR} $
↪{OUTPUT_DIR} [--window_size=$WINDOW_SIZE] [--device ${GPU_ID}]
```

它同时支持滑动窗口框架和循环框架。例子：

EDVR:

```
python demo/restoration_video_demo.py ./configs/restorers/edvr/edvrm_wotsa_x4_g8_600k_
↪reds.py https://download.openmmlab.com/mmediting/restorers/edvr/edvrm_wotsa_x4_8x4_
↪600k_reds_20200522-0570e567.pth data/Vid4/BIx4/calendar/ ./output --window_size=5
```

BasicVSR:

```
python demo/restoration_video_demo.py ./configs/restorers/basicvsr/basicvsr_reds4.py␣
↪https://download.openmmlab.com/mmediting/restorers/basicvsr/basicvsr_reds4_20120409-
↪0e599677.pth data/Vid4/BIx4/calendar/ ./output
```

复原的视频将保存在 `output/` 中。

3.6 图像生成

```
python demo/generation_demo.py ${CONFIG_FILE} ${CHECKPOINT_FILE} ${IMAGE_FILE} ${SAVE_
↪FILE} [--unpaired_path ${UNPAIRED_IMAGE_FILE}] [--imshow] [--device ${GPU_ID}]
```

如果指定了 `--unpaired_path` （用于 CycleGAN），模型将执行未配对的图像到图像的转换。如果指定了 `--imshow` ，演示也将使用 opencv 显示图像。例子：

针对配对数据：

```
python demo/generation_demo.py configs/example_config.py work_dirs/example_exp/
↪example_model_20200202.pth demo/demo.jpg demo/demo_out.jpg
```

针对未配对数据（用 opencv 显示图像）：

```
python demo/generation_demo.py configs/example_config.py work_dirs/example_exp/
↪example_model_20200202.pth demo/demo.jpg demo/demo_out.jpg --unpaired_path demo/
↪demo_unpaired.jpg --imshow
```

CHAPTER 4

总览

- 模型权重文件数量: 68
- 配置文件数量: 63
- 论文数量: 25
  - ALGORITHM: 25

有关支持的数据集，请参阅数据集总览。

4.1 补全模型

- 模型权重文件数量: 8
- 配置文件数量: 8
- 论文数量: 4
  - [ALGORITHM] Free-Form Image Inpainting With Gated Convolution (⇨)
  - [ALGORITHM] Generative Image Inpainting With Contextual Attention (⇨)
  - [ALGORITHM] Globally and Locally Consistent Image Completion (⇨)
  - [ALGORITHM] Image Inpainting for Irregular Holes Using Partial Convolutions (⇨)

4.2 抠图模型

- 模型权重文件数量: 9
- 配置文件数量: 9
- 论文数量: 3
  - [ALGORITHM] Deep Image Matting (⇨)
  - [ALGORITHM] Indices Matter: Learning to Index for Deep Image Matting (⇨)
  - [ALGORITHM] Natural Image Matting via Guided Contextual Attention (⇨)

4.3 超分辨率模型

- 模型权重文件数量: 41
- 配置文件数量: 36
- 论文数量: 16
  - [ALGORITHM] Basicvsr: The Search for Essential Components in Video Super-Resolution and Beyond (⇨⇨)
  - [ALGORITHM] Basicvsr++: Improving Video Super-Resolution With Enhanced Propagation and Alignment (⇨)
  - [ALGORITHM] Deep Face Super-Resolution With Iterative Collaboration Between Attentive Recovery and Landmark Estimation (⇨)
  - [ALGORITHM] Edvr: Video Restoration With Enhanced Deformable Convolutional Networks (⇨)
  - [ALGORITHM] Enhanced Deep Residual Networks for Single Image Super-Resolution (⇨)
  - [ALGORITHM] Esrgan: Enhanced Super-Resolution Generative Adversarial Networks (⇨)
  - [ALGORITHM] Glean: Generative Latent Bank for Large-Factor Image Super-Resolution (⇨)
  - [ALGORITHM] Image Super-Resolution Using Deep Convolutional Networks (⇨)
  - [ALGORITHM] Investigating Tradeoffs in Real-World Video Super-Resolution (⇨)
  - [ALGORITHM] Learning Continuous Image Representation With Local Implicit Image Function (⇨)
  - [ALGORITHM] Learning Texture Transformer Network for Image Super-Resolution (⇨)
  - [ALGORITHM] Photo-Realistic Single Image Super-Resolution Using a Generative Adversarial Network (⇨)
  - [ALGORITHM] Real-Esrgan: Training Real-World Blind Super-Resolution With Pure Synthetic Data (⇨)
  - [ALGORITHM] Residual Dense Network for Image Super-Resolution (⇨)

  - [ALGORITHM] Tdan: Temporally-Deformable Alignment Network for Video Super-Resolution (⇨)
  - [ALGORITHM] Video Enhancement With Task-Oriented Flow (⇨)

4.4 生成模型

- 模型权重文件数量: 10
- 配置文件数量: 10
- 论文数量: 2
  - [ALGORITHM] Image-to-Image Translation With Conditional Adversarial Networks (⇨)
  - [ALGORITHM] Unpaired Image-to-Image Translation Using Cycle-Consistent Adversarial Networks (⇨)

CHAPTER 5

补全模型

5.1 DeepFillv1 (CVPR’2018)

```
@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}
}
```

Places365-Challenge

CelebA-HQ

5.2 DeepFillv2 (CVPR’2019)

```
@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}
}
```

Places365-Challenge

CelebA-HQ

5.3 Global&Local (ToG’2017)

```
@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

5.4 PConv (ECCV' 2018)

```
@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

CHAPTER 6

抠图模型

6.1 DIM (CVPR’2017)

```
@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。

6.2 GCA (AAAI' 2020)

```
@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

6.3 IndexNet (ICCV' 2019)

```
@inproceedings{hao2019indexnet,
  title={Indices Matter: Learning to Index for Deep Image Matting},
  author={Lu, Hao and Dai, Yutong and Shen, Chunhua and Xu, Songcen},
  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

CHAPTER 7

超分辨率模型

7.1 BasicVSR (CVPR’2021)

```
@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。SPyNet 的预训练权重在这里。

7.2 BasicVSR++

```
@article{chan2021basicvsrplusplus,
  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 = {arXiv preprint arXiv:2104.13371},
  year = {2021}
}
```

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 MMEditing 达到 5k star 时发布。我们在这里提供预训练的模型。

NTIRE 2021 Video Super-Resolution

NTIRE 2021 Quality Enhancement of Compressed Video - Track 1

NTIRE 2021 Quality Enhancement of Compressed Video - Track 2

NTIRE 2021 Quality Enhancement of Compressed Video - Track 3

7.3 DIC (CVPR’2020)

```
@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。

7.4 EDSR (CVPR' 2017)

```
@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。

7.5 EDVR (CVPRW' 2019)

```
@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。

7.6 ESRGAN (ECCVW' 2018)

```
@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。

7.7 GLEAN (CVPR' 2021)

```
@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。

7.8 IconVSR (CVPR' 2021)

```
@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。

7.9 LIIF (CVPR' 2021)

```
@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。

7.10 RDN (CVPR’2018)

```
@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。

7.11 RealBasicVSR (arXiv’2021)

```
@article{chan2021investigating,
  author = {Chan, Kelvin C.K. and Zhou, Shangchen and Xu, Xiangyu and Loy, Chen␣
↪Change},
  title = {Investigating Tradeoffs in Real-World Video Super-Resolution},
  journal = {arXiv preprint arXiv:2111.12704},
  year = {2021}
}
```

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。

7.11.1 训练

训练分为两个阶段：

1. 使用 realbasicvsr_wogan_c64b20_2x30x8_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_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。

7.12 Real-ESRGAN (ICCVW' 2021)

```
@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`。

7.13 SRCNN (TPAMI' 2015)

```
@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。

7.14 SRGAN (CVPR’2016)

```
@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。

7.15 TDAN (CVPR’2020)

```
@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。

训练

您可以使用以下命令来训练模型。

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

TDAN 训练有两个阶段。

阶段 1: 以更大的学习率训练 (1e-4)

```
./tools/dist_train.sh configs/restorers/tdan/tdan_vimeo90k_bix4_lr1e-4_400k.py 8
```

阶段 2: 以较小的学习率进行微调 (5e-5)

```
./tools/dist_train.sh configs/restorers/tdan/tdan_vimeo90k_bix4_ft_lr5e-5_400k.py 8
```

更多细节可以参考 getting_started 中的 **Train a model** 部分。

测试

您可以使用以下命令来测试模型。

```
python tools/test.py ${CONFIG_FILE} ${CHECKPOINT_FILE} [--out ${RESULT_FILE}] [--save-
↪path ${IMAGE_SAVE_PATH}]
```

示例：使用 `bicubic` 下采样在 SPMCS-30 上测试 TDAN。

```
python tools/test.py configs/restorers/tdan/tdan_vimeo90k_bix4_ft_lr5e-5_400k.py ␣
↪checkpoints/SOME_CHECKPOINT.pth --save_path outputs/
```

更多细节可以参考 getting_started 中的 **Inference with pretrained models** 部分。

7.16 TOFlow (IJCV' 2019)

```
@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}
}
```

在 RGB 通道上进行评估。我们使用 `PSNR` 和 `SSIM` 作为指标。

7.17 TTSR (CVPR’2020)

```
@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。

CHAPTER 8

生成模型

8.1 CycleGAN (ICCV’2017)

```
@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}
}
```

我们使用“FID”和“IS”指标来评估 CycleGAN 的生成性能。

注：随着更大的身份损失，图像到图像的转换会变得更加保守，这导致转换较少。原作者没有提及身份损失的最佳权重。因此，除了默认设置之外，我们还将身份损失的权重设置为 0（表示 `id0`）以进行更全面的比较。

8.2 Pix2Pix (CVPR' 2017)

```
@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}
}
```

我们使用 `FID` 和 `IS` 指标来评估 pix2pix 的生成表现。

注：我们严格遵守论文第 3.3 节中的设置：

"*At inference time, we run the generator net in exactly the same manner as during the training phase. This differs from the usual protocol in that we apply dropout at test time, and we apply batch normalization using the statistics of the test batch, rather than aggregated statistics of the training batch.*"

即使用 `model.train()` 模式，因此可能会导致每次推理结果略有不同。

CHAPTER 9

总览

- 论文数量: 11
  - DATASET: 11

有关支持的算法, 可参见模型总览.

9.1 图像生成数据集

- 论文数量: 2
  - [DATASET] Image-to-Image Translation With Conditional Adversarial Networks (为 Pix2pix 准备配对数据集 ⇨)
  - [DATASET] Unpaired Image-to-Image Translation Using Cycle-Consistent Adversarial Networks (为 CycleGAN 准备未配对数据集 ⇨)

9.2 图像补全数据集

- 论文数量: 3
  - [DATASET] Context Encoders: Feature Learning by Inpainting (准备 Paris Street View 数据集 ⇨)
  - [DATASET] Places: A 10 Million Image Database for Scene Recognition (准备 Places365 数据集 ⇨)
  - [DATASET] Progressive Growing of Gans for Improved Quality, Stability, and Variation (准备 CelebA-HQ 数据集 ⇨)

9.3 抠图数据集

- 论文数量: 1
  - [DATASET] Deep Image Matting (准备 Composition-1k 数据集 ⇨)

9.4 超分辨率数据集

- 论文数量: 5
  - [DATASET] Ntire 2017 Challenge on Single Image Super-Resolution: Dataset and Study (准备 DIV2K 数据集 ⇨)
  - [DATASET] Ntire 2019 Challenge on Video Deblurring and Super-Resolution: Dataset and Study (准备 REDS 数据集 ⇨)
  - [DATASET] On Bayesian Adaptive Video Super Resolution (准备 Vid4 数据集 ⇨)
  - [DATASET] Real-Esrgan: Training Real-World Blind Super-Resolution With Pure Synthetic Data (准备 DF2K_OST 数据集 ⇨)
  - [DATASET] Video Enhancement With Task-Oriented Flow (准备 Vimeo90K 数据集 ⇨)

CHAPTER 10

图像补全数据集

建议将数据集软链接到 `$MMEDITING/data` 。如果您的文件夹结构不同，您可能需要更改配置文件中的相应路径。

MMEditing 支持的补全数据集：

- Paris Street View [主页]
- CelebA-HQ [主页]
- Places365 [主页]

由于在图像补全任务中，我们只需要使用图像，因此我们不需要对数据集进行额外的预处理操作，文件目录的结构也可以和本例有所不同。您可以利用原始数据集提供的信息，如 `Place365` （例如 `meta`）。或者，您可以直接遍历数据集文件夹，并将所有图像文件的路径罗列在一个文本文件中。下面的例子节选自 Places365 数据集中的 `Places365_val.txt`，针对图像补全任务，我们只需要使用其中的文件名信息。

```
Places365_val_00000001.jpg 165
Places365_val_00000002.jpg 358
Places365_val_00000003.jpg 93
Places365_val_00000004.jpg 164
Places365_val_00000005.jpg 289
Places365_val_00000006.jpg 106
Places365_val_00000007.jpg 81
Places365_val_00000008.jpg 121
Places365_val_00000009.jpg 150
Places365_val_00000010.jpg 302
Places365_val_00000011.jpg 42
```

10.1 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。

```
mmediting
├── mmedit
├── tools
├── configs
├── data
|   ├── celeba-hq
|   |   ├── train
|   |   ├── val
```

10.2 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。

```
mmediting
├── mmedit
├── tools
├── configs
├── data
|   ├── paris_street_view
|   |   ├── train
|   |   ├── val
```

10.3 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。

```
mmediting
├── mmedit
├── tools
├── configs
├── data
|   ├── places
|   |   ├── test_set
|   |   ├── train_set
|   |   ├── meta
|   |   |   ├── Places365_train.txt
|   |   |   ├── Places365_val.txt
```

CHAPTER 11

抠图数据集

建议将数据集软链接到 `$MMEDITING/data` 。如果您的文件夹结构不同，您可能需要更改配置文件中的相应路径。

MMEditing 支持的抠图数据集：

- Composition-1k [Homepage]

11.1 准备 Composition-1k 数据集

11.1.1 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。

11.1.2 获取和提取

请按照 论文作者 的说明获取 Composition-1k (comp1k) 数据集。

11.1.3 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 MMEditing 的根文件夹下运行脚本
python tools/data/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/data/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/data/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"）。

11.1.4 检查 DIM 的目录结构

最终的文件夹结构应如下所示：

```
mmediting
├── mmedit
├── tools
├── configs
├── data
│   ├── adobe_composition-1k
│   │   ├── Test_set
│   │   │   ├── Adobe-licensed images
│   │   │   │   ├── alpha
```

(下页继续)

(续上页)

```
|   |   |   |   ├── 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
|   |   |   ├── Other
|   |   |   |   ├── alpha
|   |   |   |   ├── fg
|   |   |   ├── merged  (generated by tools/data/matting/comp1k/preprocess_comp1k_
↪dataset.py)
|   |   |   ├── bg      (generated by tools/data/matting/comp1k/preprocess_comp1k_
↪dataset.py)
|   |   ├── test_list.json     (generated by tools/data/matting/comp1k/preprocess_
↪comp1k_dataset.py)
|   |   ├── training_list.json (generated by tools/data/matting/comp1k/preprocess_
↪comp1k_dataset.py)
|   ├── coco
|   |   ├── train2014   (or train2017)
|   ├── VOCdevkit
|   |   ├── VOC2012
```

11.1.5 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/data/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/data/matting/comp1k/extend_fg.py data/adobe_composition-1k
```

11.1.6 检查 DIM 的目录结构

最终的文件夹结构应如下所示：

```
mmediting
├── mmedit
├── 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 12

超分辨率数据集

建议将数据集的根目录链接到 `$MMEDITING/data` 下，如果您的文件目录结构不一致，那么可能需要在配置文件中修改对应的文件路径。

MMEditing 支持下列超分辨率数据集：

- 图像超分辨率
  - DIV2K [Homepage]
- 视频超分辨率
  - REDS [Homepage]
  - Vimeo90K [Homepage]

12.1 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）：

```
mmediting
├── mmedit
├── tools
├── configs
├── data
|   ├── df2k_ost
|   |   ├── GT
|   |   |   ├── 0001.png
|   |   |   ├── 0002.png
|   |   |   ├── ...
...
```

12.1.1 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/data/super-resolution/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。

```
mmediting
├── mmedit
├── tools
├── configs
├── data
|   ├── df2k_ost
|   |   ├── GT
|   |   ├── GT_sub
...
```

12.1.2 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/data/super-resolution/df2k_ost/preprocess_df2k_ost_dataset.py --data-
↪root ./data/df2k_ost --make-lmdb
```

12.2 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 and Set14.

```
mmediting
├── mmedit
├── tools
├── configs
├── data
│   ├── DIV2K
│   │   ├── DIV2K_train_HR
│   │   ├── DIV2K_train_LR_bicubic
│   │   │   ├── X2
│   │   │   ├── X3
│   │   │   ├── X4
│   │   ├── DIV2K_valid_HR
│   │   ├── DIV2K_valid_LR_bicubic
│   │   │   ├── X2
│   │   │   ├── X3
│   │   │   ├── X4
│   ├── val_set5
│   │   ├── Set5_bicLRx2
│   │   ├── Set5_bicLRx3
│   │   ├── Set5_bicLRx4
```

(下页继续)

(续上页)

```
|   ├── val_set14
|   |   ├── Set14_bicLRx2
|   |   ├── Set14_bicLRx3
|   |   ├── Set14_bicLRx4
```

12.2.1 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/data/super-resolution/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:

```
mmediting
├── mmedit
├── tools
├── configs
├── data
|   ├── DIV2K
|   |   ├── DIV2K_train_HR
|   |   ├── DIV2K_train_HR_sub
|   |   ├── DIV2K_train_LR_bicubic
|   |   |   ├── X2
|   |   |   ├── X3
|   |   |   ├── X4
|   |   |   ├── X2_sub
|   |   |   ├── X3_sub
|   |   |   ├── X4_sub
|   |   ├── DIV2K_valid_HR
|   |   ├── ...
...
```

12.2.2 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)
```

12.2.3 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/data/super-resolution/div2k/preprocess_div2k_dataset.py --data-root ./
↪data/DIV2K --make-lmdb
```

12.3 准备 REDS 数据集

```bibtex
@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/data/super-resolution/reds/preprocess_reds_dataset.py ./data/REDS
```

```
mmediting
├── mmedit
├── tools
├── configs
├── data
│   ├── REDS
│   │   ├── train_sharp
│   │   │   ├── 000
│   │   │   ├── 001
│   │   │   ├── ...
```

(下页继续)

(续上页)

```
|   |   ├── train_sharp_bicubic
|   |   |   ├── 000
|   |   |   ├── 001
|   |   |   ├── ...
|   ├── REDS4
|   |   ├── GT
|   |   ├── sharp_bicubic
```

12.3.1 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/data/super-resolution/reds/preprocess_reds_dataset.py --root-path ./data/
↪REDS --make-lmdb
```

12.3.2 裁剪为子图

MMEditing 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/data/super-resolution/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。

```
mmediting
├── mmedit
├── tools
├── configs
├── data
|   ├── REDS
|   |   ├── train_sharp
|   |   |   ├── 000
|   |   |   ├── 001
|   |   |   ├── ...
|   |   ├── train_sharp_sub
|   |   |   ├── 000_s001
|   |   |   ├── 000_s002
|   |   |   ├── ...
|   |   |   ├── 001_s001
|   |   |   ├── ...
|   |   ├── train_sharp_bicubic
|   |   |   ├── X4
```

(下页继续)

(续上页)

```
|   |   |   |   ├── 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。

12.4 准备 Vid4 数据集

```bibtex
@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

12.5 准备 Vimeo90K 数据集

```bibtex
@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},
```

(下页继续)

(续上页)

```
  pages={1106--1125},
  year={2019},
  publisher={Springer}
}
```

训练集和测试集可以从 此处 下载。

Vimeo90K 数据集包含了如下所示的 `clip/sequence/img` 目录结构：

```
├── GT/LQ
|   ├── 00001
|   |   ├── 0001
|   |   |   ├── im1.png
|   |   |   ├── im2.png
|   |   |   ├── ...
|   |   ├── 0002
|   |   ├── 0003
|   |   ├── ...
|   ├── 00002
|   ├── ...
```

12.5.1 准备 Vimeo90K 数据集的标注文件

为了准备好训练所需的标注文件，请先从 Vimeo90K 数据集官网下载训练路径列表，随后执行如下命令：

```
python tools/data/super-resolution/vimeo90k/preprocess_vimeo90k_dataset.py ./data/
↪Vimeo90K/official_train_list.txt
```

测试集的标注文件可通过类似方式生成.

12.5.2 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/data/super-resolution/vimeo90k/preprocess_vimeo90k_dataset.py ./data/
↪Vimeo90K/official_train_list.txt --gt_path ./data/Vimeo90K/GT --lq_path ./data/
↪Vimeo90K/LQ  --make-lmdb
```

CHAPTER 13

图像生成数据集

建议将数据集软链接到 `$MMEDITING/data` 。如果您的文件夹结构不同，您可能需要更改配置文件中的相应路径。

MMEditing 支持的生成数据集：

- Pix2Pix 的配对数据集 [主页]
- CycleGAN 的未配对数据集 [主页]

13.1 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。

```
mmediting
├── mmedit
├── tools
├── configs
├── data
|   ├── paired
|   |   ├── facades
|   |   ├── maps
|   |   ├── edges2shoes
|   |   |   ├── train
|   |   |   ├── test
```

13.2 为 CycleGAN 准备未配对数据集

```bibtex
@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。

```
mmediting
├── mmedit
├── tools
├── configs
├── data
|   ├── unpaired
|   |   ├── facades
|   |   ├── horse2zebra
|   |   ├── summer2winter_yosemite
|   |   |   ├── trainA
|   |   |   ├── trainB
|   |   |   ├── testA
|   |   |   ├── testB
```

CHAPTER 14

使用预训练模型进行推理

我们提供用于在完整数据集上进行预训练模型评估和特定任务图像演示的测试脚本。

14.1 测试一个预训练模型

MMEditing 使用 `MMDistributedDataParallel` 实现 **分布式**测试。

14.1.1 在单/多个 GPU 上进行测试

您可以使用以下命令在单/多个 GPU 上测试预训练模型。

```
# 单 GPU 测试
python tools/test.py ${CONFIG_FILE} ${CHECKPOINT_FILE} [--out ${RESULT_FILE}] [--save-
↪path ${IMAGE_SAVE_PATH}]

# 多 GPU 测试
./tools/dist_test.sh ${CONFIG_FILE} ${CHECKPOINT_FILE} ${GPU_NUM} [--out ${RESULT_
↪FILE}] [--save-path ${IMAGE_SAVE_PATH}]
```

例如

```
# 单 GPU 测试
python tools/test.py configs/example_config.py work_dirs/example_exp/example_model_
↪20200202.pth --out work_dirs/example_exp/results.pkl
```

(下页继续)

(续上页)

```
# 多 GPU 测试
./tools/dist_test.sh configs/example_config.py work_dirs/example_exp/example_model_
↪20200202.pth --save-path work_dirs/example_exp/results/
```

14.1.2 在 slurm 上测试

如果您在使用 slurm 管理的集群上运行 MMEditing，则可以使用脚本 `slurm_test.sh`。(此脚本也支持单机测试。)

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

以下是使用 8 个 GPU 在作业名称为 `test` 的 `dev` 分区上测试示例模型的例子。

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

您可以查看 slurm_test.sh 以获取完整的参数和环境变量。

14.1.3 可选参数

- `--out`: 以 pickle 格式指定输出结果的文件名。如果没有给出，结果将不会保存到文件中。
- `--save-path`: 指定存储编辑图像的路径。如果没有给出，图像将不会被保存。
- `--seed`: 测试期间的随机种子。此参数用于固定某些任务中的结果，例如修复。
- `--deterministic`: 与 `--seed` 相关，此参数决定是否为 CUDNN 后端设置确定性的选项。如果指定该参数，会将 `torch.backends.cudnn.deterministic` 设置为 `True`，将 `torch.backends.cudnn.benchmark` 设置为 `False`。

注：目前，我们不使用像 MMDetection 那样的 `--eval` 参数来指定评估指标。评估指标在配置文件中给出(参见*config.md*)。

CHAPTER 15

训练一个模型

MMEditing 使用 `MMDistributedDataParallel` 实现 **分布式**测试。

所有输出（日志文件和模型权重文件）都将保存到工作目录中，工作目录由配置文件中的 `work_dir` 指定。

默认情况下，我们在多次迭代后评估验证集上的模型，您可以通过在训练配置中添加 `interval` 参数来更改评估间隔。

```
evaluation = dict(interval=1e4, by_epoch=False)  # 每一万次迭代进行一次评估。
```

15.1 在单/多个 GPU 上训练

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

可选参数是：

- `--no-validate` (**不建议**): 默认情况下，代码库将在训练期间每 k 次迭代执行一次评估。若要禁用此行为，请使用 `--no-validate`。
- `--work-dir ${WORK_DIR}`: 覆盖配置文件中指定的工作目录。
- `--resume-from ${CHECKPOINT_FILE}`: 从已有的模型权重文件恢复。

`resume-from` 和 `load-from` 之间的区别：`resume-from` 加载模型权重和优化器状态，迭代也从指定的检查点继承。它通常用于恢复意外中断的训练过程。`load-from` 只加载模型权重，训练迭代从 0 开始，通常用于微调。

15.2 在 slurm 上训练

如果您在使用 slurm 管理的集群上运行 MMEditing，则可以使用脚本 `slurm_train.sh`。（此脚本也支持单机训练。）

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

以下是使用 8 个 GPU 在 `dev` 分区上训练修复模型的示例。

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

您可以查看 slurm_train.sh 以获取完整的参数和环境变量。

15.3 在一台机器上启动多个作业

如果您在一台机器上启动多个作业，例如，在具有 8 个 GPU 的机器上进行 2 个 4-GPU 训练的作业，您需要为每个作业指定不同的端口（默认为 29500）以避免通信冲突。

```
CUDA_VISIBLE_DEVICES=0,1,2,3 PORT=29500 ./tools/dist_train.sh ${CONFIG_FILE} 4
CUDA_VISIBLE_DEVICES=4,5,6,7 PORT=29501 ./tools/dist_train.sh ${CONFIG_FILE} 4
```

如果您使用 Slurm 启动训练作业，则需要修改配置文件（通常是配置文件的倒数第 6 行）以设置不同的通信端口。

在 `config1.py` 中，

```
dist_params = dict(backend='nccl', port=29500)
```

在 `config2.py` 中，

```
dist_params = dict(backend='nccl', port=29501)
```

然后您可以使用 `config1.py` 和 `config2.py` 启动两个作业。

```
CUDA_VISIBLE_DEVICES=0,1,2,3 GPUS=4 ./tools/slurm_train.sh ${PARTITION} ${JOB_NAME}␣
↪config1.py ${WORK_DIR}
CUDA_VISIBLE_DEVICES=4,5,6,7 GPUS=4 ./tools/slurm_train.sh ${PARTITION} ${JOB_NAME}␣
↪config2.py ${WORK_DIR}
```

CHAPTER 16

教程 1: 了解配置文件

mmedit 采用基于 python 文件的配置系统，您可以在 `$MMEditing/configs` 下查看预置的配置文件。

16.1 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}_[model setting]_{backbone}_[refiner]_[norm setting]_[misc]_[gpu x batch_per_
↪gpu]_{schedule}_{dataset}
```

`{xxx}` 是必填字段，`[yyy]` 是可选的。

- `{model}`: 模型种类，例如 `srcnn`, `dim` 等等。
- `[model setting]`: 特定设置一些模型，例如，输入图像 `resolution` , 训练 `stage name`。
- `{backbone}`: 主干网络种类，例如 `r50` (ResNet-50)、`x101` (ResNeXt-101)。
- `{refiner}`: 精炼器种类，例如 `pln` 简单精炼器模型
- `[norm_setting]`: 指定归一化设置，默认为批归一化，其他归一化可以设为: `bn`(批归一化), `gn` (组归一化), `syncbn` (同步批归一化)。
- `[misc]`: 模型中各式各样的设置/插件，例如 `dconv`, `gcb`, `attention`, `mstrain`。
- `[gpu x batch_per_gpu]`: GPU 数目和每个 GPU 的样本数，默认为 `8x2` 。
- `{schedule}`: 训练策略，如 `20k`, `100k` 等，意思是 `20k` 或 `100k` 迭代轮数。

- {dataset}: 数据集，如 places（图像补全）、comp1k（抠图）、div2k（图像恢复）和 paired（图像生成）。

16.2 配置文件 - 生成

与 MMDetection 一样，我们将模块化和继承设计融入我们的配置系统，以方便进行各种实验。

16.2.1 示例 - pix2pix

为了帮助用户对完整的配置和生成系统中的模块有一个基本的了解，我们对 pix2pix 的配置做如下简要说明。更详细的用法和各个模块对应的替代方案，请参考 API 文档。

```
## 模型设置
model = dict(
    type='Pix2Pix',  ## 合成器名称
    generator=dict(
        type='UnetGenerator',  ## 生成器名称
        in_channels=3,  ## 生成器的输入通道数
        out_channels=3,  ## 生成器的输出通道数
        num_down=8,  ## # 生成器中下采样的次数
        base_channels=64,  ## 生成器最后卷积层的通道数
        norm_cfg=dict(type='BN'),  ## 归一化层的配置
        use_dropout=True,  ## 是否在生成器中使用 dropout
        init_cfg=dict(type='normal', gain=0.02)),  ## 初始化配置
    discriminator=dict(
        type='PatchDiscriminator',  ## 判别器的名称
        in_channels=6,  ## 判别器的输入通道数
        base_channels=64,  ## 判别器第一卷积层的通道数
        num_conv=3,  ## 判别器中堆叠的中间卷积层（不包括输入和输出卷积层）的数量
        norm_cfg=dict(type='BN'),  ## 归一化层的配置
        init_cfg=dict(type='normal', gain=0.02)),  ## 初始化配置
    gan_loss=dict(
        type='GANLoss',  ## GAN 损失的名称
        gan_type='vanilla',  ## GAN 损失的类型
        real_label_val=1.0,  ## GAN 损失函数中真实标签的值
        fake_label_val=0.0,  ## GAN 损失函数中伪造标签的值
        loss_weight=1.0),  ## GAN 损失函数的权重
    pixel_loss=dict(type='L1Loss', loss_weight=100.0, reduction='mean'))
## 模型训练和测试设置
train_cfg = dict(
    direction='b2a')  ## pix2pix 的图像到图像的转换方向 (模型训练的方向，和测试方向一致)。模型默认: a2b
test_cfg = dict(
```

(下页继续)

(续上页)

```
    direction='b2a',   ## pix2pix 的图像到图像的转换方向 (模型测试的方向，和训练方向一致)。模型默认: a2b
    show_input=True)  ## 保存 pix2pix 的测试图像时是否显示输入的真实图像

## 数据设置
train_dataset_type = 'GenerationPairedDataset'  ## 训练数据集的类型
val_dataset_type = 'GenerationPairedDataset'  ## 验证/测试数据集类型
img_norm_cfg = dict(mean=[0.5, 0.5, 0.5], std=[0.5, 0.5, 0.5])  ## 输入图像归一化配置
train_pipeline = [
    dict(
        type='LoadPairedImageFromFile',  ## 从文件路径加载图像对
        io_backend='disk',  ## 存储图像的 IO 后端
        key='pair',  ## 查找对应路径的关键词
        flag='color'),  ## 加载图像标志
    dict(
        type='Resize',  ## 图像大小调整
        keys=['img_a', 'img_b'],  ## 要调整大小的图像的关键词
        scale=(286, 286),  ## 调整图像大小的比例
        interpolation='bicubic'),  ## 调整图像大小时用于插值的算法
    dict(
        type='FixedCrop',  ## 固定裁剪，在特定位置将配对图像裁剪为特定大小以训练 pix2pix
        keys=['img_a', 'img_b'],  ## 要裁剪的图像的关键词
        crop_size=(256, 256)),  ## 裁剪图像的大小
    dict(
        type='Flip',  ## 翻转图像
        keys=['img_a', 'img_b'],  ## 要翻转的图像的关键词
        direction='horizontal'),  ## 水平或垂直翻转图像
    dict(
        type='RescaleToZeroOne',  ## 将图像从 [0, 255] 缩放到 [0, 1]
        keys=['img_a', 'img_b']),  ## 要重新缩放的图像的关键词
    dict(
        type='Normalize',  ## 图像归一化
        keys=['img_a', 'img_b'],  ## 要归一化的图像的关键词
        to_rgb=True,  ## 是否将图像通道从 BGR 转换为 RGB
        **img_norm_cfg),  ## 图像归一化配置 (`img_norm_cfg` 的定义见上文)
    dict(
       type='ImageToTensor',  ## 将图像转化为 Tensor
       keys=['img_a', 'img_b']),  ## 要从图像转换为 Tensor 的图像的关键词
    dict(
        type='Collect',  ## 决定数据中哪些键应该传递给合成器
        keys=['img_a', 'img_b'],  ## 图像的关键词
        meta_keys=['img_a_path', 'img_b_path'])  ## 图片的元关键词
]
```

(下页继续)

(续上页)

```
test_pipeline = [
    dict(
        type='LoadPairedImageFromFile',  ## 从文件路径加载图像对
        io_backend='disk',  ## 存储图像的 IO 后端
        key='pair',  ## 查找对应路径的关键词
        flag='color'),  ## 加载图像标志
    dict(
        type='Resize',  ## 图像大小调整
        keys=['img_a', 'img_b'],  ## 要调整大小的图像的关键词
        scale=(256, 256),  ## 调整图像大小的比例
        interpolation='bicubic'),  ## 调整图像大小时用于插值的算法
    dict(
        type='RescaleToZeroOne',  ## 将图像从 [0, 255] 缩放到 [0, 1]
        keys=['img_a', 'img_b']),  ## 要重新缩放的图像的关键词
    dict(
        type='Normalize',  ## 图像归一化
        keys=['img_a', 'img_b'],  ## 要归一化的图像的关键词
        to_rgb=True,  ## 是否将图像通道从 BGR 转换为 RGB
        **img_norm_cfg),  ## 图像归一化配置 (`img_norm_cfg` 的定义见上文)
    dict(
       type='ImageToTensor',  ## 将图像转化为 Tensor
       keys=['img_a', 'img_b']),  ## 要从图像转换为 Tensor 的图像的关键词
    dict(
        type='Collect',  ## 决定数据中哪些键应该传递给合成器
        keys=['img_a', 'img_b'],  ## 图像的关键词
        meta_keys=['img_a_path', 'img_b_path'])  ## 图片的元关键词
]
data_root = 'data/pix2pix/facades'  ## 数据的根路径
data = dict(
    samples_per_gpu=1,  ## 单个 GPU 的批量大小
    workers_per_gpu=4,  ## 为每个 GPU 预取数据的 Worker 数
    drop_last=True,  ## 是否丢弃训练中的最后一批数据
    val_samples_per_gpu=1,  ## 验证中单个 GPU 的批量大小
    val_workers_per_gpu=0,  ## 在验证中为每个 GPU 预取数据的 Worker 数
    train=dict(  ## 训练数据集配置
        type=train_dataset_type,
        dataroot=data_root,
        pipeline=train_pipeline,
        test_mode=False),
    val=dict(  ## 验证数据集配置
        type=val_dataset_type,
        dataroot=data_root,
        pipeline=test_pipeline,
```

(下页继续)

(续上页)

```
        test_mode=True),
    test=dict(  ## 测试数据集配置
        type=val_dataset_type,
        dataroot=data_root,
        pipeline=test_pipeline,
        test_mode=True))

## 优化器
optimizers = dict(  ## 用于构建优化器的配置，支持 PyTorch 中所有优化器，且参数与 PyTorch 中对应优化器相同
    generator=dict(type='Adam', lr=2e-4, betas=(0.5, 0.999)),
    discriminator=dict(type='Adam', lr=2e-4, betas=(0.5, 0.999)))

## 学习策略
lr_config = dict(policy='Fixed', by_epoch=False)  ## 用于注册 LrUpdater 钩子的学习率调度程序配置

## 检查点保存
checkpoint_config = dict(interval=4000, save_optimizer=True, by_epoch=False)  ## 配置检查点钩子，实现参考 https://github.com/open-mmlab/mmcv/blob/master/mmcv/runner/hooks/
↪checkpoint.py
evaluation = dict(  ## 构建验证钩子的配置
    interval=4000,  ## 验证区间
    save_image=True)  ## 是否保存图片
log_config = dict(  ## 配置注册记录器钩子
    interval=100,  ## 打印日志的时间间隔
    hooks=[
        dict(type='TextLoggerHook', by_epoch=False),  ## 用于记录训练过程的记录器
        ## dict(type='TensorboardLoggerHook')  # 还支持 Tensorboard 记录器
    ])
visual_config = None  ## 构建可视化钩子的配置

## 运行设置
total_iters = 80000  ## 训练模型的总迭代次数
cudnn_benchmark = True  ## 设置 cudnn_benchmark
dist_params = dict(backend='nccl')  ## 设置分布式训练的参数，端口也可以设置
log_level = 'INFO'  ## 日志级别
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_from = None  ## 从给定路径恢复检查点，当检查点被保存时，训练将从该 epoch 恢复
workflow = [('train', 1)]  ## runner 的工作流程。 [('train', 1)] 表示只有一个工作流程，名为
↪'train' 的工作流程执行一次。 训练当前生成模型时保持不变
exp_name = 'pix2pix_facades'  ## 实验名称
work_dir = f'./work_dirs/{exp_name}'  ## 保存当前实验的模型检查点和日志的目录
```

16.3 配置文件 - 补全

16.3.1 配置名称样式

与 MMDetection 一样，我们将模块化和继承设计融入我们的配置系统，以方便进行各种实验。

16.3.2 配置字段说明

为了帮助用户对完整的配置和修复系统中的模块有一个基本的了解，我们对 Global&Local 的配置作如下简要说明。更详细的用法和各个模块对应的替代方案，请参考 API 文档。

```
model = dict(
    type='GLInpaintor', ## 补全器的名称
    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'))), ## 扩颈的配置
    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, ## 判别器中的最大通道数
            fc_in_channels=512 * 4 * 4, ## 最后一个全连接层的输入通道
            fc_out_channels=1024, ## 最后一个全连接层的输出通道
            num_convs=6, ## 判别器中使用的卷积数量
            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, ## 判别器中的最大通道数
            fc_in_channels=512 * 4 * 4, ## 最后一个全连接层的输入通道
            fc_out_channels=1024, ## 最后一个全连接层的输出通道
            num_convs=5, ## 判别器中使用的卷积数量
            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 损失函数的权重
    ),
```

(下页继续)

(续上页)

```
    loss_l1_hole=dict(
        type='L1Loss', ## L1 损失的类型
        loss_weight=1.0 ## L1 损失函数的权重
    ),
    pretrained=None) ## 预训练权重的路径

train_cfg = dict(
    disc_step=1, ## 训练生成器之前训练判别器的迭代次数
    iter_tc=90000, ## 预热生成器的迭代次数
    iter_td=100000, ## 预热判别器的迭代次数
    start_iter=0, ## 开始的迭代
    local_size=(128, 128)) ## 图像块的大小
test_cfg = dict(metrics=['l1']) ## 测试的配置

dataset_type = 'ImgInpaintingDataset' ## 数据集类型
input_shape = (256, 256) ## 输入图像的形状

train_pipeline = [
    dict(type='LoadImageFromFile', key='gt_img'), ## 加载图片的配置
    dict(
        type='LoadMask', ## 加载掩码
        mask_mode='bbox', ## 掩码的类型
        mask_config=dict(
            max_bbox_shape=(128, 128), ## 检测框的形状
            max_bbox_delta=40, ## 检测框高宽的变化
            min_margin=20,  ## 检测框到图片边界的最小距离
            img_shape=input_shape)),  ## 输入图像的形状
    dict(
        type='Crop', ## 裁剪
        keys=['gt_img'],  ## 要裁剪的图像的关键词
        crop_size=(384, 384),  ## 裁剪图像块的大小
        random_crop=True,  ## 是否使用随机裁剪
    ),
    dict(
        type='Resize',  ## 图像大小调整
        keys=['gt_img'],  ## 要调整大小的图像的关键词
        scale=input_shape,  ## 调整图像大小的比例
        keep_ratio=False,  ## 调整大小时是否保持比例
    ),
    dict(
        type='Normalize',  ## 图像归一化
        keys=['gt_img'],  ## 要归一化的图像的关键词
        mean=[127.5] * 3,  ## 归一化中使用的均值
```

(下页继续)

(续上页)

```
        std=[127.5] * 3,  ## 归一化中使用的标准差
        to_rgb=False),  ## 是否将图像通道从 BGR 转换为 RGB
    dict(type='GetMaskedImage'),  ## 获取被掩盖的图像
    dict(
        type='Collect',  ## 决定数据中哪些键应该传递给合成器
        keys=['gt_img', 'masked_img', 'mask', 'mask_bbox'],  ## 要收集的数据的关键词
        meta_keys=['gt_img_path']),  ## 要收集的数据的元关键词
    dict(type='ImageToTensor', keys=['gt_img', 'masked_img', 'mask']),  ## 将图像转化为␣
↪Tensor
    dict(type='ToTensor', keys=['mask_bbox'])  ## 转化为 Tensor
]

test_pipeline = train_pipeline  ## 构建测试/验证流程

data_root = 'data/places365'  ## 数据根目录

data = dict(
    samples_per_gpu=12,  ## 单个 GPU 的批量大小
    workers_per_gpu=8,  ## 为每个 GPU 预取数据的 Worker 数
    val_samples_per_gpu=1,  ## 验证中单个 GPU 的批量大小
    val_workers_per_gpu=8,  ## 在验证中为每个 GPU 预取数据的 Worker 数
    drop_last=True,  ## 是否丢弃训练中的最后一批数据
    train=dict(  ## 训练数据集配置
        type=dataset_type,
        ann_file=f'{data_root}/train_places_img_list_total.txt',
        data_prefix=data_root,
        pipeline=train_pipeline,
        test_mode=False),
    val=dict(  ## 验证数据集配置
        type=dataset_type,
        ann_file=f'{data_root}/val_places_img_list.txt',
        data_prefix=data_root,
        pipeline=test_pipeline,
        test_mode=True))

optimizers = dict(  ## 用于构建优化器的配置，支持 PyTorch 中所有优化器，且参数与 PyTorch 中对应
优化器相同
    generator=dict(type='Adam', lr=0.0004), disc=dict(type='Adam', lr=0.0004))

lr_config = dict(policy='Fixed', by_epoch=False)  ## 用于注册 LrUpdater 钩子的学习率调度程
序配置

checkpoint_config = dict(by_epoch=False, interval=50000)  ## 配置检查点钩子，实现参考␣
↪https://github.com/open-mmlab/mmcv/blob/master/mmcv/runner/hooks/checkpoint.py
```

(下页继续)

(续上页)

```
log_config = dict(  ## 配置注册记录器钩子
    interval=100,  ## 打印日志的时间间隔
    hooks=[
        dict(type='TextLoggerHook', by_epoch=False),
        ## dict(type='TensorboardLoggerHook'),  # 支持 Tensorboard 记录器
        ## dict(type='PaviLoggerHook', init_kwargs=dict(project='mmedit'))
    ])  ## 用于记录训练过程的记录器

visual_config = dict(  ## 构建可视化钩子的配置
    type='VisualizationHook',
    output_dir='visual',
    interval=1000,
    res_name_list=[
        'gt_img', 'masked_img', 'fake_res', 'fake_img', 'fake_gt_local'
    ],
)  ## 用于可视化训练过程的记录器。

evaluation = dict(interval=50000)  ## 构建验证钩子的配置

total_iters = 500002
dist_params = dict(backend='nccl')  ## 设置分布式训练的参数，端口也可以设置
log_level = 'INFO'  ## 日志级别
work_dir = None  ## 保存当前实验的模型检查点和日志的目录
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_from = None  ## 从给定路径恢复检查点，当检查点被保存时，训练将从该 epoch 恢复
workflow = [('train', 10000)]  ## runner 的工作流程。 [('train', 1)] 表示只有一个工作流程，名
为 'train' 的工作流程执行一次。 训练当前生成模型时保持不变
exp_name = 'gl_places'  ## 实验名称
find_unused_parameters = False  ## 是否在分布式训练中查找未使用的参数
```

16.4 配置文件 - 抠图

与 MMDetection 一样，我们将模块化和继承设计融入我们的配置系统，以方便进行各种实验。

16.4.1 例子 - Deep Image Matting Model

为了帮助用户对一个完整的配置有一个基本的了解，我们对我们实现的原始 DIM 模型的配置做一个简短的评论，如下所示。更详细的用法和各个模块对应的替代方案，请参考 API 文档。

```
## 模型配置
model = dict(
    type='DIM',  ## 模型的名称（我们称之为抠图器）
    backbone=dict(  ## 主干网络的配置
        type='SimpleEncoderDecoder',  ## 主干网络的类型
        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',  ## 编码器的预训练权重
    loss_alpha=dict(  ## alpha 损失的配置
        type='CharbonnierLoss',  ## 预测的 alpha 遮罩的损失类型
        loss_weight=0.5),  ## alpha 损失的权重
    loss_comp=dict(  ## 组合损失的配置
        type='CharbonnierCompLoss',  ## 组合损失的类型
        loss_weight=0.5))  ## 组合损失的权重
train_cfg = dict(  ## 训练 DIM 模型的配置
    train_backbone=True,  ## 在 DIM stage 1 中，会对主干网络进行训练
    train_refiner=False)  ## 在 DIM stage 1 中，不会对精炼器进行训练
test_cfg = dict(  ## 测试 DIM 模型的配置
    refine=False,  ## 是否使用精炼器输出作为输出，在 stage 1 中，我们不使用它
    metrics=['SAD', 'MSE', 'GRAD', 'CONN'])  ## 测试时使用的指标

## 数据配置
dataset_type = 'AdobeComp1kDataset'  ## 数据集类型，这将用于定义数据集
data_root = 'data/adobe_composition-1k'  ## 数据的根目录
img_norm_cfg = dict(  ## 归一化输入图像的配置
    mean=[0.485, 0.456, 0.406],  ## 归一化中使用的均值
    std=[0.229, 0.224, 0.225],  ## 归一化中使用的标准差
    to_rgb=True)  ## 是否将图像通道从 BGR 转换为 RGB
train_pipeline = [  ## 训练数据处理流程
    dict(
        type='LoadImageFromFile',  ## 从文件加载 alpha 遮罩
        key='alpha',  ## 注释文件中 alpha 遮罩的键关键词。流程将从路径 "alpha_path" 中读取␣
↪alpha 遮罩
        flag='grayscale'),  ## 加载灰度图像，形状为（高度、宽度）
    dict(
        type='LoadImageFromFile',  ## 从文件中加载图像
        key='fg'),  ## 要加载的图像的关键词。流程将从路径 "fg_path" 读取 fg
```

（下页继续）

(续上页)

```
    dict(
        type='LoadImageFromFile',  ## 从文件中加载图像
        key='bg'),  ## 要加载的图像的关键词。流程将从路径 “bg_path” 读取 bg
    dict(
        type='LoadImageFromFile',  ## 从文件中加载图像
        key='merged'),  ## 要加载的图像的关键词。流程将从路径 “merged_path” 读取并合并
    dict(
        type='CropAroundUnknown',  ## 在未知区域 (半透明区域) 周围裁剪图像
        keys=['alpha', 'merged', 'ori_merged', 'fg', 'bg'],  ## 要裁剪的图像
        crop_sizes=[320, 480, 640]),  ## 裁剪大小
    dict(
        type='Flip',  ## 翻转图像
        keys=['alpha', 'merged', 'ori_merged', 'fg', 'bg']),  ## 要翻转的图像
    dict(
        type='Resize',  ## 图像大小调整
        keys=['alpha', 'merged', 'ori_merged', 'fg', 'bg'],  ## 图像调整大小的图像
        scale=(320, 320),  ## 目标大小
        keep_ratio=False),  ## 是否保持高宽比例
    dict(
        type='GenerateTrimap',  ## 从 alpha 遮罩生成三元图。
        kernel_size=(1, 30)),  ## 腐蚀/扩张内核大小的范围
    dict(
        type='RescaleToZeroOne', ## 将图像从 [0, 255] 缩放到 [0, 1]
        keys=['merged', 'alpha', 'ori_merged', 'fg', 'bg']),  ## 要重新缩放的图像
    dict(
        type='Normalize',  ## 图像归一化
        keys=['merged'],  ## 要归一化的图像
        **img_norm_cfg),  ## 图像归一化配置 (`img_norm_cfg` 的定义见上文)
    dict(
        type='Collect',  ## 决定数据中哪些键应该传递给合成器
        keys=['merged', 'alpha', 'trimap', 'ori_merged', 'fg', 'bg'],  ## 图像的关键词
        meta_keys=[]),  ## 图片的元关键词, 这里不需要元信息。
    dict(
        type='ImageToTensor',  ## 将图像转化为 Tensor
        keys=['merged', 'alpha', 'trimap', 'ori_merged', 'fg', 'bg']),  ## 要转换为␣
↪Tensor 的图像
]
test_pipeline = [
    dict(
        type='LoadImageFromFile', ## 从文件加载 alpha 遮罩
        key='alpha',  ## 注释文件中 alpha 遮罩的键关键词。流程将从路径 “alpha_path” 中读取␣
↪alpha 遮罩
        flag='grayscale',
```

(下页继续)

(续上页)

```
        save_original_img=True),
    dict(
        type='LoadImageFromFile',  ## 从文件中加载图像
        key='trimap',  ## 要加载的图像的关键词。流程将从路径 “trimap_path” 读取三元图
        flag='grayscale', ## 加载灰度图像，形状为（高度、宽度）
        save_original_img=True), ## 保存三元图用于计算指标。 它将与 “ori_trimap” 一起保存
    dict(
        type='LoadImageFromFile',  ## 从文件中加载图像
        key='merged'),  ## 要加载的图像的关键词。流程将从路径 “merged_path” 读取并合并
    dict(
        type='Pad',  ## 填充图像以与模型的下采样因子对齐
        keys=['trimap', 'merged'],  ## 要填充的图像
        mode='reflect'),  ## 填充模式
    dict(
        type='RescaleToZeroOne',  ## 与 train_pipeline 相同
        keys=['merged', 'ori_alpha']),  ## 要缩放的图像
    dict(
        type='Normalize',  ## 与 train_pipeline 相同
        keys=['merged'],
        **img_norm_cfg),
    dict(
        type='Collect',  ## 与 train_pipeline 相同
        keys=['merged', 'trimap'],
        meta_keys=[
            'merged_path', 'pad', 'merged_ori_shape', 'ori_alpha',
            'ori_trimap'
        ]),
    dict(
        type='ImageToTensor',  ## 与 train_pipeline 相同
        keys=['merged', 'trimap']),
]
data = dict(
    samples_per_gpu=1,  ## 单个 GPU 的批量大小
    workers_per_gpu=4,  ## 为每个 GPU 预取数据的 Worker 数
    drop_last=True,  ## 是否丢弃训练中的最后一批数据
    train=dict(  ## 训练数据集配置
        type=dataset_type,  ## 数据集的类型
        ann_file=f'{data_root}/training_list.json',  ## 注解文件路径
        data_prefix=data_root,  ## 图像路径的前缀
        pipeline=train_pipeline),  ## 见上文 train_pipeline
    val=dict(  ## 验证数据集配置
        type=dataset_type,
        ann_file=f'{data_root}/test_list.json',
```

(下页继续)

(续上页)

```
        data_prefix=data_root,
        pipeline=test_pipeline),  ## 见上文 test_pipeline
    test=dict(  ## 测试数据集配置
        type=dataset_type,
        ann_file=f'{data_root}/test_list.json',
        data_prefix=data_root,
        pipeline=test_pipeline))  ## 见上文 test_pipeline

## 优化器
optimizers = 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，且参数与 PyTorch 中对应优化器相同
## 学习策略
lr_config = dict(  ## 用于注册 LrUpdater 钩子的学习率调度程序配置
    policy='Fixed')  ## 调度器的策略，支持 CosineAnnealing、Cyclic 等。支持的 LrUpdater 详情请参考 https://github.com/open-mmlab/mmcv/blob/master/mmcv/runner/hooks/lr_updater.py
↪#L9。

## 检查点保存
checkpoint_config = dict(  ## 配置检查点钩子，实现参考 https://github.com/open-mmlab/mmcv/
↪blob/master/mmcv/runner/hooks/checkpoint.py
    interval=40000,  ## 保存间隔为 40000 次迭代
    by_epoch=False)  ## 按迭代计数
evaluation = dict(  ## # 构建验证钩子的配置
    interval=40000)  ## 验证区间
log_config = dict(  ## 配置注册记录器钩子
    interval=10,  ## 打印日志的时间间隔
    hooks=[
        dict(type='TextLoggerHook', by_epoch=False),  ## 用于记录训练过程的记录器
        ## dict(type='TensorboardLoggerHook')  # 支持 Tensorboard 记录器
    ])

## runtime settings
total_iters = 1000000  ## 训练模型的总迭代次数
dist_params = dict(backend='nccl')  ## 设置分布式训练的参数，端口也可以设置
log_level = 'INFO'  ## 日志级别
work_dir = './work_dirs/dim_stage1'  ## 保存当前实验的模型检查点和日志的目录
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_from = None  ## 从给定路径恢复检查点，当检查点被保存时，训练将从该 epoch 恢复
workflow = [('train', 1)]  ## runner 的工作流程。 [('train', 1)] 表示只有一个工作流程，名为
↪'train' 的工作流程执行一次。 训练当前抠图模型时保持不变
```

16.5 配置文件 - 复原

16.5.1 示例-EDSR

为了帮助用户理解 mmediting 的配置文件结构，这里以 EDSR 为例，给出其配置文件的注释。对于每个模块的详细用法以及对应参数的选择，请参照 API 文档。

```
exp_name = 'edsr_x2c64b16_1x16_300k_div2k'  ## 实验名称

scale = 2  ## 上采样放大因子

## 模型设置
model = dict(
    type='BasicRestorer',  ## 图像恢复模型类型
    generator=dict(  ## 生成器配置
        type='EDSR',  ## 生成器类型
        in_channels=3,  ## 输入通道数
        out_channels=3,  ## 输出通道数
        mid_channels=64,  ## 中间特征通道数
        num_blocks=16,  ## 残差块数目
        upscale_factor=scale, ## 上采样因子
        res_scale=1,  ## 残差缩放因子
        rgb_mean=(0.4488, 0.4371, 0.4040),  ## 输入图像 RGB 通道的平均值
        rgb_std=(1.0, 1.0, 1.0)),  ## 输入图像 RGB 通道的方差
    pixel_loss=dict(type='L1Loss', loss_weight=1.0, reduction='mean'))  ## 像素损失函数的
配置

## 模型训练和测试设置
train_cfg = None  ## 训练的配置
test_cfg = dict(  ## 测试的配置
    metrics=['PSNR'],  ## 测试时使用的评价指标
    crop_border=scale)  ## 测试时裁剪的边界尺寸

## 数据集设置
train_dataset_type = 'SRAnnotationDataset'  ## 用于训练的数据集类型
val_dataset_type = 'SRFolderDataset'  ##  用于验证的数据集类型
train_pipeline = [  ## 训练数据前处理流水线步骤组成的列表
    dict(type='LoadImageFromFile',  ## 从文件加载图像
        io_backend='disk',  ## 读取图像时使用的 io 类型
        key='lq',  ## 设置 LR 图像的键来找到相应的路径
        flag='unchanged'),  ## 读取图像的标识
    dict(type='LoadImageFromFile',  ## 从文件加载图像
        io_backend='disk',  ## 读取图像时使用的 io 类型
        key='gt',  ## 设置 HR 图像的键来找到相应的路径
```

(下页继续)

(续上页)

```
        flag='unchanged'),  ## 读取图像的标识
    dict(type='RescaleToZeroOne', keys=['lq', 'gt']),  ## 将图像从 [0, 255] 重缩放到 [0,
1]
    dict(type='Normalize',  ## 正则化图像
        keys=['lq', 'gt'],  ## 执行正则化图像的键
        mean=[0, 0, 0],  ## 平均值
        std=[1, 1, 1],  ## 标准差
        to_rgb=True),  ## 更改为 RGB 通道
    dict(type='PairedRandomCrop', gt_patch_size=96),  ## LR 和 HR 成对随机裁剪
    dict(type='Flip',  ## 图像翻转
        keys=['lq', 'gt'],  ## 执行翻转图像的键
        flip_ratio=0.5,  ## 执行翻转的几率
        direction='horizontal'),  ## 翻转方向
    dict(type='Flip',  ## 图像翻转
        keys=['lq', 'gt'],  ## 执行翻转图像的键
        flip_ratio=0.5,  ## 执行翻转几率
        direction='vertical'),  ## 翻转方向
    dict(type='RandomTransposeHW',  ## 图像的随机的转置
        keys=['lq', 'gt'],  ## 执行转置图像的键
        transpose_ratio=0.5  ## 执行转置的几率
        ),
    dict(type='Collect',  ## Collect 类决定哪些键会被传递到生成器中
        keys=['lq', 'gt'],  ## 传入模型的键
        meta_keys=['lq_path', 'gt_path']), ## 元信息键。在训练中，不需要元信息
    dict(type='ImageToTensor',  ## 将图像转换为张量
        keys=['lq', 'gt'])  ## 执行图像转换为张量的键
]
test_pipeline = [  ## 测试数据前处理流水线步骤组成的列表
    dict(
        type='LoadImageFromFile',  ## 从文件加载图像
        io_backend='disk',  ## 读取图像时使用的 io 类型
        key='lq',  ## 设置 LR 图像的键来找到相应的路径
        flag='unchanged'),  ## 读取图像的标识
    dict(
        type='LoadImageFromFile',  ## 从文件加载图像
        io_backend='disk',  ## 读取图像时使用的 io 类型
        key='gt',  ## 设置 HR 图像的键来找到相应的路径
        flag='unchanged'),  ## 读取图像的标识
    dict(type='RescaleToZeroOne', keys=['lq', 'gt']),  ## 将图像从 [0, 255] 重缩放到 [0,
1]
    dict(
        type='Normalize',  ## 正则化图像
        keys=['lq', 'gt'],  ## 执行正则化图像的键
```

(下页继续)

(续上页)

```
        mean=[0, 0, 0],  ## 平均值
        std=[1, 1, 1],  ## 标准差
        to_rgb=True),  ## 更改为 RGB 通道
    dict(type='Collect',  ## Collect 类决定哪些键会被传递到生成器中
        keys=['lq', 'gt'],  ## 传入模型的键
        meta_keys=['lq_path', 'lq_path']),  ## 元信息键
    dict(type='ImageToTensor',  ## 将图像转换为张量
        keys=['lq', 'gt'])  ## 执行图像转换为张量的键
]

data = dict(
    ## 训练
    samples_per_gpu=16,  ## 单个 GPU 的批大小
    workers_per_gpu=6,  ## 单个 GPU 的 dataloader 的进程
    drop_last=True,  ## 在训练过程中丢弃最后一个批次
    train=dict(  ## 训练数据集的设置
        type='RepeatDataset',  ## 基于迭代的重复数据集
        times=1000,  ## 重复数据集的重复次数
        dataset=dict(
            type=train_dataset_type,  ## 数据集类型
            lq_folder='data/DIV2K/DIV2K_train_LR_bicubic/X2_sub',  ## lq 文件夹的路径
            gt_folder='data/DIV2K/DIV2K_train_HR_sub',  ## gt 文件夹的路径
            ann_file='data/DIV2K/meta_info_DIV2K800sub_GT.txt',  ## 批注文件的路径
            pipeline=train_pipeline,  ## 训练流水线，如上所示
            scale=scale)),  ## 上采样放大因子

    ## 验证
    val_samples_per_gpu=1,  ## 验证时单个 GPU 的批大小
    val_workers_per_gpu=1,  ## 验证时单个 GPU 的 dataloader 的进程
    val=dict(
        type=val_dataset_type,  ## 数据集类型
        lq_folder='data/val_set5/Set5_bicLRx2',  ## lq 文件夹的路径
        gt_folder='data/val_set5/Set5_mod12',  ## gt 文件夹的路径
        pipeline=test_pipeline,  ## 测试流水线，如上所示
        scale=scale,  ## 上采样放大因子
        filename_tmpl='{}'),  ## 文件名模板

    ## 测试
    test=dict(
        type=val_dataset_type,  ## 数据集类型
        lq_folder='data/val_set5/Set5_bicLRx2',  ## lq 文件夹的路径
        gt_folder='data/val_set5/Set5_mod12',  ## gt 文件夹的路径
        pipeline=test_pipeline,  ## 测试流水线，如上所示
```

(下页继续)

(续上页)

```
        scale=scale,  ## 上采样放大因子
        filename_tmpl='{}'))  ## 文件名模板

## 优化器设置
optimizers = dict(generator=dict(type='Adam', lr=1e-4, betas=(0.9, 0.999)))  ## 用于构建
优化器的设置，支持 PyTorch 中所有参数与 PyTorch 中参数相同的优化器

## 学习策略
total_iters = 300000  ## 训练模型的总迭代数
lr_config = dict( ## 用于注册 LrUpdater 钩子的学习率调度程序配置
    policy='Step', by_epoch=False, step=[200000], gamma=0.5)  ## 调度器的策略，还支持余弦、
循环等

checkpoint_config = dict(  ## 模型权重钩子设置，更多细节可参考 https://github.com/open-
↪mmlab/mmcv/blob/master/mmcv/runner/hooks/checkpoint.py
    interval=5000,  ## 模型权重文件保存间隔为 5000 次迭代
    save_optimizer=True,  ## 保存优化器
    by_epoch=False)  ## 按迭代次数计数
evaluation = dict(  ## 构建验证钩子的配置
    interval=5000,  ## 执行验证的间隔为 5000 次迭代
    save_image=True,  ## 验证期间保存图像
    gpu_collect=True)  ## 使用 gpu 收集
log_config = dict(  ## 注册日志钩子的设置
    interval=100,  ## 打印日志间隔
    hooks=[
        dict(type='TextLoggerHook', by_epoch=False),  ## 记录训练过程信息的日志
        dict(type='TensorboardLoggerHook'),  ## 同时支持 Tensorboard 日志
    ])
visual_config = None  ## 可视化的设置

## 运行设置
dist_params = dict(backend='nccl')  ## 建立分布式训练的设置，其中端口号也可以设置
log_level = 'INFO'  ## 日志等级
work_dir = f'./work_dirs/{exp_name}'  ## 记录当前实验日志和模型权重文件的文件夹
load_from = None ## 从给定路径加载模型作为预训练模型. 这个选项不会用于断点恢复训练
resume_from = None ## 加载给定路径的模型权重文件作为断点续连的模型，训练将从该时间点保存的周期点继
续进行
workflow = [('train', 1)]  ## runner 的执行流. [('train', 1)] 代表只有一个执行流，并且这个名
为 train 的执行流只执行一次
```

16.6 配置文件 - 生成

与 MMDetection 一样，我们将模块化和继承设计融入我们的配置系统，以方便进行各种实验。

16.6.1 示例 - pix2pix

为了帮助用户对完整的配置和生成系统中的模块有一个基本的了解，我们对 pix2pix 的配置做如下简要说明。更详细的用法和各个模块对应的替代方案，请参考 API 文档。

```
## 模型设置
model = dict(
    type='Pix2Pix',  ## 合成器名称
    generator=dict(
        type='UnetGenerator',  ## 生成器名称
        in_channels=3,  ## 生成器的输入通道数
        out_channels=3,  ## 生成器的输出通道数
        num_down=8,  ## # 生成器中下采样的次数
        base_channels=64,  ## 生成器最后卷积层的通道数
        norm_cfg=dict(type='BN'),  ## 归一化层的配置
        use_dropout=True,  ## 是否在生成器中使用 dropout
        init_cfg=dict(type='normal', gain=0.02)),  ## 初始化配置
    discriminator=dict(
        type='PatchDiscriminator',  ## 判别器的名称
        in_channels=6,  ## 判别器的输入通道数
        base_channels=64,  ## 判别器第一卷积层的通道数
        num_conv=3,  ## 判别器中堆叠的中间卷积层（不包括输入和输出卷积层）的数量
        norm_cfg=dict(type='BN'),  ## 归一化层的配置
        init_cfg=dict(type='normal', gain=0.02)),  ## 初始化配置
    gan_loss=dict(
        type='GANLoss',  ## GAN 损失的名称
        gan_type='vanilla',  ## GAN 损失的类型
        real_label_val=1.0,  ## GAN 损失函数中真实标签的值
        fake_label_val=0.0,  ## GAN 损失函数中伪造标签的值
        loss_weight=1.0),  ## GAN 损失函数的权重
    pixel_loss=dict(type='L1Loss', loss_weight=100.0, reduction='mean'))
## 模型训练和测试设置
train_cfg = dict(
    direction='b2a')  ## pix2pix 的图像到图像的转换方向 (模型训练的方向，和测试方向一致)。模型默
认: a2b
test_cfg = dict(
    direction='b2a',   ## pix2pix 的图像到图像的转换方向 (模型测试的方向，和训练方向一致)。模型默
认: a2b
    show_input=True)  ## 保存 pix2pix 的测试图像时是否显示输入的真实图像
```

(下页继续)

(续上页)

```
## 数据设置
train_dataset_type = 'GenerationPairedDataset'  ## 训练数据集的类型
val_dataset_type = 'GenerationPairedDataset'  ## 验证/测试数据集类型
img_norm_cfg = dict(mean=[0.5, 0.5, 0.5], std=[0.5, 0.5, 0.5])  ## 输入图像归一化配置
train_pipeline = [
    dict(
        type='LoadPairedImageFromFile',  ## 从文件路径加载图像对
        io_backend='disk',  ## 存储图像的 IO 后端
        key='pair',  ## 查找对应路径的关键词
        flag='color'),  ## 加载图像标志
    dict(
        type='Resize',  ## 图像大小调整
        keys=['img_a', 'img_b'],  ## 要调整大小的图像的关键词
        scale=(286, 286),  ## 调整图像大小的比例
        interpolation='bicubic'),  ## 调整图像大小时用于插值的算法
    dict(
        type='FixedCrop',  ## 固定裁剪，在特定位置将配对图像裁剪为特定大小以训练 pix2pix
        keys=['img_a', 'img_b'],  ## 要裁剪的图像的关键词
        crop_size=(256, 256)),  ## 裁剪图像的大小
    dict(
        type='Flip',  ## 翻转图像
        keys=['img_a', 'img_b'],  ## 要翻转的图像的关键词
        direction='horizontal'),  ## 水平或垂直翻转图像
    dict(
        type='RescaleToZeroOne',  ## 将图像从 [0, 255] 缩放到 [0, 1]
        keys=['img_a', 'img_b']),  ## 要重新缩放的图像的关键词
    dict(
        type='Normalize',  ## 图像归一化
        keys=['img_a', 'img_b'],  ## 要归一化的图像的关键词
        to_rgb=True,  ## 是否将图像通道从 BGR 转换为 RGB
        **img_norm_cfg),  ## 图像归一化配置 (`img_norm_cfg` 的定义见上文)
    dict(
       type='ImageToTensor',  ## 将图像转化为 Tensor
       keys=['img_a', 'img_b']),  ## 要从图像转换为 Tensor 的图像的关键词
    dict(
        type='Collect',  ## 决定数据中哪些键应该传递给合成器
        keys=['img_a', 'img_b'],  ## 图像的关键词
        meta_keys=['img_a_path', 'img_b_path'])  ## 图片的元关键词
]
test_pipeline = [
    dict(
        type='LoadPairedImageFromFile',  ## 从文件路径加载图像对
```

(下页继续)

(续上页)

```
        io_backend='disk',  ## 存储图像的 IO 后端
        key='pair',  ## 查找对应路径的关键词
        flag='color'),  ## 加载图像标志
    dict(
        type='Resize',  ## 图像大小调整
        keys=['img_a', 'img_b'],  ## 要调整大小的图像的关键词
        scale=(256, 256),  ## 调整图像大小的比例
        interpolation='bicubic'),  ## 调整图像大小时用于插值的算法
    dict(
        type='RescaleToZeroOne',  ## 将图像从 [0, 255] 缩放到 [0, 1]
        keys=['img_a', 'img_b']),  ## 要重新缩放的图像的关键词
    dict(
        type='Normalize',  ## 图像归一化
        keys=['img_a', 'img_b'],  ## 要归一化的图像的关键词
        to_rgb=True,  ## 是否将图像通道从 BGR 转换为 RGB
        **img_norm_cfg),  ## 图像归一化配置 (`img_norm_cfg` 的定义见上文)
    dict(
       type='ImageToTensor',  ## 将图像转化为 Tensor
       keys=['img_a', 'img_b']),  ## 要从图像转换为 Tensor 的图像的关键词
    dict(
        type='Collect',  ## 决定数据中哪些键应该传递给合成器
        keys=['img_a', 'img_b'],  ## 图像的关键词
        meta_keys=['img_a_path', 'img_b_path'])  ## 图片的元关键词
]
data_root = 'data/pix2pix/facades'  ## 数据的根路径
data = dict(
    samples_per_gpu=1,  ## 单个 GPU 的批量大小
    workers_per_gpu=4,  ## 为每个 GPU 预取数据的 Worker 数
    drop_last=True,  ## 是否丢弃训练中的最后一批数据
    val_samples_per_gpu=1,  ## 验证中单个 GPU 的批量大小
    val_workers_per_gpu=0,  ## 在验证中为每个 GPU 预取数据的 Worker 数
    train=dict(  ## 训练数据集配置
        type=train_dataset_type,
        dataroot=data_root,
        pipeline=train_pipeline,
        test_mode=False),
    val=dict(  ## 验证数据集配置
        type=val_dataset_type,
        dataroot=data_root,
        pipeline=test_pipeline,
        test_mode=True),
    test=dict(  ## 测试数据集配置
        type=val_dataset_type,
```

(下页继续)

(续上页)

```
        dataroot=data_root,
        pipeline=test_pipeline,
        test_mode=True))

## 优化器
optimizers = dict(  ## 用于构建优化器的配置，支持 PyTorch 中所有优化器，且参数与 PyTorch 中对应优化器相同
    generator=dict(type='Adam', lr=2e-4, betas=(0.5, 0.999)),
    discriminator=dict(type='Adam', lr=2e-4, betas=(0.5, 0.999)))

## 学习策略
lr_config = dict(policy='Fixed', by_epoch=False)  ## 用于注册 LrUpdater 钩子的学习率调度程序配置

## 检查点保存
checkpoint_config = dict(interval=4000, save_optimizer=True, by_epoch=False)  ## 配置检查点钩子，实现参考 https://github.com/open-mmlab/mmcv/blob/master/mmcv/runner/hooks/
↪checkpoint.py
evaluation = dict(  ## 构建验证钩子的配置
    interval=4000,  ## 验证区间
    save_image=True)  ## 是否保存图片
log_config = dict(  ## 配置注册记录器钩子
    interval=100,  ## 打印日志的时间间隔
    hooks=[
        dict(type='TextLoggerHook', by_epoch=False),  ## 用于记录训练过程的记录器
        ## dict(type='TensorboardLoggerHook')  # 还支持 Tensorboard 记录器
    ])
visual_config = None  ## 构建可视化钩子的配置

## 运行设置
total_iters = 80000  ## 训练模型的总迭代次数
cudnn_benchmark = True  ## 设置 cudnn_benchmark
dist_params = dict(backend='nccl')  ## 设置分布式训练的参数，端口也可以设置
log_level = 'INFO'  ## 日志级别
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_from = None  ## 从给定路径恢复检查点，当检查点被保存时，训练将从该 epoch 恢复
workflow = [('train', 1)]  ## runner 的工作流程。 [('train', 1)] 表示只有一个工作流程，名为
↪'train' 的工作流程执行一次。 训练当前生成模型时保持不变
exp_name = 'pix2pix_facades'  ## 实验名称
work_dir = f'./work_dirs/{exp_name}'  ## 保存当前实验的模型检查点和日志的目录
```

16.7 配置文件 - 补全

16.7.1 配置名称样式

与 MMDetection 一样，我们将模块化和继承设计融入我们的配置系统，以方便进行各种实验。

16.7.2 配置字段说明

为了帮助用户对完整的配置和修复系统中的模块有一个基本的了解，我们对 Global&Local 的配置作如下简要说明。更详细的用法和各个模块对应的替代方案，请参考 API 文档。

```
model = dict(
    type='GLInpaintor', ## 补全器的名称
    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'))), ## 扩颈的配置
    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, ## 判别器中的最大通道数
            fc_in_channels=512 * 4 * 4, ## 最后一个全连接层的输入通道
            fc_out_channels=1024, ## 最后一个全连接层的输出通道
            num_convs=6, ## 判别器中使用的卷积数量
            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, ## 判别器中的最大通道数
            fc_in_channels=512 * 4 * 4, ## 最后一个全连接层的输入通道
            fc_out_channels=1024, ## 最后一个全连接层的输出通道
            num_convs=5, ## 判别器中使用的卷积数量
            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 损失函数的权重
    ),
```

(下页继续)

(续上页)

```
    loss_l1_hole=dict(
        type='L1Loss', ## L1 损失的类型
        loss_weight=1.0 ## L1 损失函数的权重
    ),
    pretrained=None) ## 预训练权重的路径

train_cfg = dict(
    disc_step=1, ## 训练生成器之前训练判别器的迭代次数
    iter_tc=90000, ## 预热生成器的迭代次数
    iter_td=100000, ## 预热判别器的迭代次数
    start_iter=0, ## 开始的迭代
    local_size=(128, 128)) ## 图像块的大小
test_cfg = dict(metrics=['l1']) ## 测试的配置

dataset_type = 'ImgInpaintingDataset' ## 数据集类型
input_shape = (256, 256) ## 输入图像的形状

train_pipeline = [
    dict(type='LoadImageFromFile', key='gt_img'), ## 加载图片的配置
    dict(
        type='LoadMask', ## 加载掩码
        mask_mode='bbox', ## 掩码的类型
        mask_config=dict(
            max_bbox_shape=(128, 128), ## 检测框的形状
            max_bbox_delta=40, ## 检测框高宽的变化
            min_margin=20,  ## 检测框到图片边界的最小距离
            img_shape=input_shape)),  ## 输入图像的形状
    dict(
        type='Crop', ## 裁剪
        keys=['gt_img'],  ## 要裁剪的图像的关键词
        crop_size=(384, 384),  ## 裁剪图像块的大小
        random_crop=True,  ## 是否使用随机裁剪
    ),
    dict(
        type='Resize',  ## 图像大小调整
        keys=['gt_img'],  ## 要调整大小的图像的关键词
        scale=input_shape,  ## 调整图像大小的比例
        keep_ratio=False,  ## 调整大小时是否保持比例
    ),
    dict(
        type='Normalize',  ## 图像归一化
        keys=['gt_img'],  ## 要归一化的图像的关键词
        mean=[127.5] * 3,  ## 归一化中使用的均值
```

(下页继续)

(续上页)

```
        std=[127.5] * 3,  ## 归一化中使用的标准差
        to_rgb=False),  ## 是否将图像通道从 BGR 转换为 RGB
    dict(type='GetMaskedImage'),  ## 获取被掩盖的图像
    dict(
        type='Collect',  ## 决定数据中哪些键应该传递给合成器
        keys=['gt_img', 'masked_img', 'mask', 'mask_bbox'],  ## 要收集的数据的关键词
        meta_keys=['gt_img_path']),  ## 要收集的数据的元关键词
    dict(type='ImageToTensor', keys=['gt_img', 'masked_img', 'mask']),  ## 将图像转化为␣
↪Tensor
    dict(type='ToTensor', keys=['mask_bbox'])  ## 转化为 Tensor
]

test_pipeline = train_pipeline  ## 构建测试/验证流程

data_root = 'data/places365'  ## 数据根目录

data = dict(
    samples_per_gpu=12,  ## 单个 GPU 的批量大小
    workers_per_gpu=8,  ## 为每个 GPU 预取数据的 Worker 数
    val_samples_per_gpu=1,  ## 验证中单个 GPU 的批量大小
    val_workers_per_gpu=8,  ## 在验证中为每个 GPU 预取数据的 Worker 数
    drop_last=True,  ## 是否丢弃训练中的最后一批数据
    train=dict(  ## 训练数据集配置
        type=dataset_type,
        ann_file=f'{data_root}/train_places_img_list_total.txt',
        data_prefix=data_root,
        pipeline=train_pipeline,
        test_mode=False),
    val=dict(  ## 验证数据集配置
        type=dataset_type,
        ann_file=f'{data_root}/val_places_img_list.txt',
        data_prefix=data_root,
        pipeline=test_pipeline,
        test_mode=True))

optimizers = dict(  ## 用于构建优化器的配置，支持 PyTorch 中所有优化器，且参数与 PyTorch 中对应
优化器相同
    generator=dict(type='Adam', lr=0.0004), disc=dict(type='Adam', lr=0.0004))

lr_config = dict(policy='Fixed', by_epoch=False)  ## 用于注册 LrUpdater 钩子的学习率调度程
序配置

checkpoint_config = dict(by_epoch=False, interval=50000)  ## 配置检查点钩子，实现参考␣
↪https://github.com/open-mmlab/mmcv/blob/master/mmcv/runner/hooks/checkpoint.py
```

(下页继续)

(续上页)

```
log_config = dict(  ## 配置注册记录器钩子
    interval=100,  ## 打印日志的时间间隔
    hooks=[
        dict(type='TextLoggerHook', by_epoch=False),
        ## dict(type='TensorboardLoggerHook'),  # 支持 Tensorboard 记录器
        ## dict(type='PaviLoggerHook', init_kwargs=dict(project='mmedit'))
    ])  ## 用于记录训练过程的记录器

visual_config = dict(  ## 构建可视化钩子的配置
    type='VisualizationHook',
    output_dir='visual',
    interval=1000,
    res_name_list=[
        'gt_img', 'masked_img', 'fake_res', 'fake_img', 'fake_gt_local'
    ],
)  ## 用于可视化训练过程的记录器。

evaluation = dict(interval=50000)  ## 构建验证钩子的配置

total_iters = 500002
dist_params = dict(backend='nccl')  ## 设置分布式训练的参数，端口也可以设置
log_level = 'INFO'  ## 日志级别
work_dir = None  ## 保存当前实验的模型检查点和日志的目录
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_from = None  ## 从给定路径恢复检查点，当检查点被保存时，训练将从该 epoch 恢复
workflow = [('train', 10000)]  ## runner 的工作流程。 [('train', 1)] 表示只有一个工作流程，名为 'train' 的工作流程执行一次。 训练当前生成模型时保持不变
exp_name = 'gl_places'  ## 实验名称
find_unused_parameters = False  ## 是否在分布式训练中查找未使用的参数
```

16.8 配置文件 - 抠图

与 MMDetection 一样，我们将模块化和继承设计融入我们的配置系统，以方便进行各种实验。

16.8.1 例子 - Deep Image Matting Model

为了帮助用户对一个完整的配置有一个基本的了解，我们对我们实现的原始 DIM 模型的配置做一个简短的评论，如下所示。更详细的用法和各个模块对应的替代方案，请参考 API 文档。

```
## 模型配置
model = dict(
    type='DIM',  ## 模型的名称（我们称之为抠图器）
    backbone=dict(  ## 主干网络的配置
        type='SimpleEncoderDecoder',  ## 主干网络的类型
        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',  ## 编码器的预训练权重
    loss_alpha=dict(  ## alpha 损失的配置
        type='CharbonnierLoss',  ## 预测的 alpha 遮罩的损失类型
        loss_weight=0.5),  ## alpha 损失的权重
    loss_comp=dict(  ## 组合损失的配置
        type='CharbonnierCompLoss',  ## 组合损失的类型
        loss_weight=0.5))  ## 组合损失的权重
train_cfg = dict(  ## 训练 DIM 模型的配置
    train_backbone=True,  ## 在 DIM stage 1 中，会对主干网络进行训练
    train_refiner=False)  ## 在 DIM stage 1 中，不会对精炼器进行训练
test_cfg = dict(  ## 测试 DIM 模型的配置
    refine=False,  ## 是否使用精炼器输出作为输出，在 stage 1 中，我们不使用它
    metrics=['SAD', 'MSE', 'GRAD', 'CONN'])  ## 测试时使用的指标

## 数据配置
dataset_type = 'AdobeComp1kDataset'  ## 数据集类型，这将用于定义数据集
data_root = 'data/adobe_composition-1k'  ## 数据的根目录
img_norm_cfg = dict(  ## 归一化输入图像的配置
    mean=[0.485, 0.456, 0.406],  ## 归一化中使用的均值
    std=[0.229, 0.224, 0.225],  ## 归一化中使用的标准差
    to_rgb=True)  ## 是否将图像通道从 BGR 转换为 RGB
train_pipeline = [  ## 训练数据处理流程
    dict(
        type='LoadImageFromFile',  ## 从文件加载 alpha 遮罩
        key='alpha',  ## 注释文件中 alpha 遮罩的键关键词。流程将从路径 “alpha_path” 中读取␣
↪alpha 遮罩
        flag='grayscale'),  ## 加载灰度图像，形状为（高度、宽度）
    dict(
        type='LoadImageFromFile',  ## 从文件中加载图像
        key='fg'),  ## 要加载的图像的关键词。流程将从路径 “fg_path” 读取 fg
```

（下页继续）

(续上页)

```
    dict(
        type='LoadImageFromFile',  ## 从文件中加载图像
        key='bg'),  ## 要加载的图像的关键词。流程将从路径 “bg_path” 读取 bg
    dict(
        type='LoadImageFromFile',  ## 从文件中加载图像
        key='merged'),  ## 要加载的图像的关键词。流程将从路径 “merged_path” 读取并合并
    dict(
        type='CropAroundUnknown',  ## 在未知区域 (半透明区域) 周围裁剪图像
        keys=['alpha', 'merged', 'ori_merged', 'fg', 'bg'],  ## 要裁剪的图像
        crop_sizes=[320, 480, 640]),  ## 裁剪大小
    dict(
        type='Flip',  ## 翻转图像
        keys=['alpha', 'merged', 'ori_merged', 'fg', 'bg']),  ## 要翻转的图像
    dict(
        type='Resize',  ## 图像大小调整
        keys=['alpha', 'merged', 'ori_merged', 'fg', 'bg'],  ## 图像调整大小的图像
        scale=(320, 320),  ## 目标大小
        keep_ratio=False),  ## 是否保持高宽比例
    dict(
        type='GenerateTrimap',  ## 从 alpha 遮罩生成三元图。
        kernel_size=(1, 30)),  ## 腐蚀/扩张内核大小的范围
    dict(
        type='RescaleToZeroOne', ## 将图像从 [0, 255] 缩放到 [0, 1]
        keys=['merged', 'alpha', 'ori_merged', 'fg', 'bg']),  ## 要重新缩放的图像
    dict(
        type='Normalize',  ## 图像归一化
        keys=['merged'],  ## 要归一化的图像
        **img_norm_cfg),  ## 图像归一化配置 (`img_norm_cfg` 的定义见上文)
    dict(
        type='Collect',  ## 决定数据中哪些键应该传递给合成器
        keys=['merged', 'alpha', 'trimap', 'ori_merged', 'fg', 'bg'],  ## 图像的关键词
        meta_keys=[]),  ## 图片的元关键词，这里不需要元信息。
    dict(
        type='ImageToTensor',  ## 将图像转化为 Tensor
        keys=['merged', 'alpha', 'trimap', 'ori_merged', 'fg', 'bg']),  ## 要转换为␣
↪Tensor 的图像
]
test_pipeline = [
    dict(
        type='LoadImageFromFile', ## 从文件加载 alpha 遮罩
        key='alpha',  ## 注释文件中 alpha 遮罩的键关键词。流程将从路径 “alpha_path” 中读取␣
↪alpha 遮罩
        flag='grayscale',
```

(下页继续)

(续上页)

```
        save_original_img=True),
    dict(
        type='LoadImageFromFile',  ## 从文件中加载图像
        key='trimap',  ## 要加载的图像的关键词。流程将从路径 "trimap_path" 读取三元图
        flag='grayscale', ## 加载灰度图像，形状为（高度、宽度）
        save_original_img=True), ## 保存三元图用于计算指标。 它将与 "ori_trimap" 一起保存
    dict(
        type='LoadImageFromFile',  ## 从文件中加载图像
        key='merged'),  ## 要加载的图像的关键词。流程将从路径 "merged_path" 读取并合并
    dict(
        type='Pad',  ## 填充图像以与模型的下采样因子对齐
        keys=['trimap', 'merged'],  ## 要填充的图像
        mode='reflect'),  ## 填充模式
    dict(
        type='RescaleToZeroOne',  ## 与 train_pipeline 相同
        keys=['merged', 'ori_alpha']),  ## 要缩放的图像
    dict(
        type='Normalize',  ## 与 train_pipeline 相同
        keys=['merged'],
        **img_norm_cfg),
    dict(
        type='Collect',  ## 与 train_pipeline 相同
        keys=['merged', 'trimap'],
        meta_keys=[
            'merged_path', 'pad', 'merged_ori_shape', 'ori_alpha',
            'ori_trimap'
        ]),
    dict(
        type='ImageToTensor',  ## 与 train_pipeline 相同
        keys=['merged', 'trimap']),
]
data = dict(
    samples_per_gpu=1,  ## 单个 GPU 的批量大小
    workers_per_gpu=4,  ## 为每个 GPU 预取数据的 Worker 数
    drop_last=True,  ## 是否丢弃训练中的最后一批数据
    train=dict(  ## 训练数据集配置
        type=dataset_type,  ## 数据集的类型
        ann_file=f'{data_root}/training_list.json',  ## 注解文件路径
        data_prefix=data_root,  ## 图像路径的前缀
        pipeline=train_pipeline),  ## 见上文 train_pipeline
    val=dict(  ## 验证数据集配置
        type=dataset_type,
        ann_file=f'{data_root}/test_list.json',
```

(下页继续)

(续上页)

```
        data_prefix=data_root,
        pipeline=test_pipeline),  ## 见上文 test_pipeline
    test=dict(  ## 测试数据集配置
        type=dataset_type,
        ann_file=f'{data_root}/test_list.json',
        data_prefix=data_root,
        pipeline=test_pipeline))  ## 见上文 test_pipeline

## 优化器
optimizers = 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，且参数与 PyTorch 中对应优化器相同
## 学习策略
lr_config = dict(  ## 用于注册 LrUpdater 钩子的学习率调度程序配置
    policy='Fixed')  ## 调度器的策略，支持 CosineAnnealing、Cyclic 等。支持的 LrUpdater 详情请参考 https://github.com/open-mmlab/mmcv/blob/master/mmcv/runner/hooks/lr_updater.py
↪#L9。

## 检查点保存
checkpoint_config = dict(  ## 配置检查点钩子，实现参考 https://github.com/open-mmlab/mmcv/
↪blob/master/mmcv/runner/hooks/checkpoint.py
    interval=40000,  ## 保存间隔为 40000 次迭代
    by_epoch=False)  ## 按迭代计数
evaluation = dict(  ## # 构建验证钩子的配置
    interval=40000)  ## 验证区间
log_config = dict(  ## 配置注册记录器钩子
    interval=10,  ## 打印日志的时间间隔
    hooks=[
        dict(type='TextLoggerHook', by_epoch=False),  ## 用于记录训练过程的记录器
        ## dict(type='TensorboardLoggerHook')  # 支持 Tensorboard 记录器
    ])

## runtime settings
total_iters = 1000000  ## 训练模型的总迭代次数
dist_params = dict(backend='nccl')  ## 设置分布式训练的参数，端口也可以设置
log_level = 'INFO'  ## 日志级别
work_dir = './work_dirs/dim_stage1'  ## 保存当前实验的模型检查点和日志的目录
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_from = None  ## 从给定路径恢复检查点，当检查点被保存时，训练将从该 epoch 恢复
workflow = [('train', 1)]  ## runner 的工作流程。 [('train', 1)] 表示只有一个工作流程，名为
↪'train' 的工作流程执行一次。 训练当前抠图模型时保持不变
```

16.9 配置文件 - 复原

16.9.1 示例-EDSR

为了帮助用户理解 mmediting 的配置文件结构，这里以 EDSR 为例，给出其配置文件的注释。对于每个模块的详细用法以及对应参数的选择，请参照 API 文档。

```
exp_name = 'edsr_x2c64b16_1x16_300k_div2k'  ## 实验名称

scale = 2  ## 上采样放大因子

## 模型设置
model = dict(
    type='BasicRestorer',  ## 图像恢复模型类型
    generator=dict(  ## 生成器配置
        type='EDSR',  ## 生成器类型
        in_channels=3,  ## 输入通道数
        out_channels=3,  ## 输出通道数
        mid_channels=64,  ## 中间特征通道数
        num_blocks=16,  ## 残差块数目
        upscale_factor=scale, ## 上采样因子
        res_scale=1,  ## 残差缩放因子
        rgb_mean=(0.4488, 0.4371, 0.4040),  ## 输入图像 RGB 通道的平均值
        rgb_std=(1.0, 1.0, 1.0)),  ## 输入图像 RGB 通道的方差
    pixel_loss=dict(type='L1Loss', loss_weight=1.0, reduction='mean'))  ## 像素损失函数的
配置

## 模型训练和测试设置
train_cfg = None  ## 训练的配置
test_cfg = dict(  ## 测试的配置
    metrics=['PSNR'],  ## 测试时使用的评价指标
    crop_border=scale)  ## 测试时裁剪的边界尺寸

## 数据集设置
train_dataset_type = 'SRAnnotationDataset'  ## 用于训练的数据集类型
val_dataset_type = 'SRFolderDataset'  ##  用于验证的数据集类型
train_pipeline = [  ## 训练数据前处理流水线步骤组成的列表
    dict(type='LoadImageFromFile',  ## 从文件加载图像
        io_backend='disk',  ## 读取图像时使用的 io 类型
        key='lq',  ## 设置 LR 图像的键来找到相应的路径
        flag='unchanged'),  ## 读取图像的标识
    dict(type='LoadImageFromFile',  ## 从文件加载图像
        io_backend='disk',  ## 读取图像时使用的 io 类型
        key='gt',  ## 设置 HR 图像的键来找到相应的路径
```

(下页继续)

(续上页)

```
        flag='unchanged'),  ## 读取图像的标识
    dict(type='RescaleToZeroOne', keys=['lq', 'gt']),  ## 将图像从 [0, 255] 重缩放到 [0,
1]
    dict(type='Normalize',  ## 正则化图像
        keys=['lq', 'gt'],  ## 执行正则化图像的键
        mean=[0, 0, 0],  ## 平均值
        std=[1, 1, 1],  ## 标准差
        to_rgb=True),  ## 更改为 RGB 通道
    dict(type='PairedRandomCrop', gt_patch_size=96),  ## LR 和 HR 成对随机裁剪
    dict(type='Flip',  ## 图像翻转
        keys=['lq', 'gt'],  ## 执行翻转图像的键
        flip_ratio=0.5,  ## 执行翻转的几率
        direction='horizontal'),  ## 翻转方向
    dict(type='Flip',  ## 图像翻转
        keys=['lq', 'gt'],  ## 执行翻转图像的键
        flip_ratio=0.5,  ## 执行翻转几率
        direction='vertical'),  ## 翻转方向
    dict(type='RandomTransposeHW',  ## 图像的随机的转置
        keys=['lq', 'gt'],  ## 执行转置图像的键
        transpose_ratio=0.5  ## 执行转置的几率
        ),
    dict(type='Collect',  ## Collect 类决定哪些键会被传递到生成器中
        keys=['lq', 'gt'],  ## 传入模型的键
        meta_keys=['lq_path', 'gt_path']), ## 元信息键。在训练中，不需要元信息
    dict(type='ImageToTensor',  ## 将图像转换为张量
        keys=['lq', 'gt'])  ## 执行图像转换为张量的键
]
test_pipeline = [  ## 测试数据前处理流水线步骤组成的列表
    dict(
        type='LoadImageFromFile',  ## 从文件加载图像
        io_backend='disk',  ## 读取图像时使用的 io 类型
        key='lq',  ## 设置 LR 图像的键来找到相应的路径
        flag='unchanged'),  ## 读取图像的标识
    dict(
        type='LoadImageFromFile',  ## 从文件加载图像
        io_backend='disk',  ## 读取图像时使用的 io 类型
        key='gt',  ## 设置 HR 图像的键来找到相应的路径
        flag='unchanged'),  ## 读取图像的标识
    dict(type='RescaleToZeroOne', keys=['lq', 'gt']),  ## 将图像从 [0, 255] 重缩放到 [0,
1]
    dict(
        type='Normalize',  ## 正则化图像
        keys=['lq', 'gt'],  ## 执行正则化图像的键
```

(下页继续)

(续上页)

```
        mean=[0, 0, 0],  ## 平均值
        std=[1, 1, 1],  ## 标准差
        to_rgb=True),  ## 更改为 RGB 通道
    dict(type='Collect',  ## Collect 类决定哪些键会被传递到生成器中
        keys=['lq', 'gt'],  ## 传入模型的键
        meta_keys=['lq_path', 'lq_path']),  ## 元信息键
    dict(type='ImageToTensor',  ## 将图像转换为张量
        keys=['lq', 'gt'])  ## 执行图像转换为张量的键
]

data = dict(
    ## 训练
    samples_per_gpu=16,  ## 单个 GPU 的批大小
    workers_per_gpu=6,  ## 单个 GPU 的 dataloader 的进程
    drop_last=True,  ## 在训练过程中丢弃最后一个批次
    train=dict(  ## 训练数据集的设置
        type='RepeatDataset',  ## 基于迭代的重复数据集
        times=1000,  ## 重复数据集的重复次数
        dataset=dict(
            type=train_dataset_type,  ## 数据集类型
            lq_folder='data/DIV2K/DIV2K_train_LR_bicubic/X2_sub',  ## lq 文件夹的路径
            gt_folder='data/DIV2K/DIV2K_train_HR_sub',  ## gt 文件夹的路径
            ann_file='data/DIV2K/meta_info_DIV2K800sub_GT.txt',  ## 批注文件的路径
            pipeline=train_pipeline,  ## 训练流水线，如上所示
            scale=scale)),  ## 上采样放大因子

    ## 验证
    val_samples_per_gpu=1,  ## 验证时单个 GPU 的批大小
    val_workers_per_gpu=1,  ## 验证时单个 GPU 的 dataloader 的进程
    val=dict(
        type=val_dataset_type,  ## 数据集类型
        lq_folder='data/val_set5/Set5_bicLRx2',  ## lq 文件夹的路径
        gt_folder='data/val_set5/Set5_mod12',  ## gt 文件夹的路径
        pipeline=test_pipeline,  ## 测试流水线，如上所示
        scale=scale,  ## 上采样放大因子
        filename_tmpl='{}'),  ## 文件名模板

    ## 测试
    test=dict(
        type=val_dataset_type,  ## 数据集类型
        lq_folder='data/val_set5/Set5_bicLRx2',  ## lq 文件夹的路径
        gt_folder='data/val_set5/Set5_mod12',  ## gt 文件夹的路径
        pipeline=test_pipeline,  ## 测试流水线，如上所示
```

(下页继续)

(续上页)

```
        scale=scale,  ## 上采样放大因子
        filename_tmpl='{}'))  ## 文件名模板

## 优化器设置
optimizers = dict(generator=dict(type='Adam', lr=1e-4, betas=(0.9, 0.999)))  ## 用于构建
优化器的设置，支持 PyTorch 中所有参数与 PyTorch 中参数相同的优化器

## 学习策略
total_iters = 300000  ## 训练模型的总迭代数
lr_config = dict( ## 用于注册 LrUpdater 钩子的学习率调度程序配置
    policy='Step', by_epoch=False, step=[200000], gamma=0.5)  ## 调度器的策略，还支持余弦、
循环等

checkpoint_config = dict(  ## 模型权重钩子设置，更多细节可参考 https://github.com/open-
↪mmlab/mmcv/blob/master/mmcv/runner/hooks/checkpoint.py
    interval=5000,  ## 模型权重文件保存间隔为 5000 次迭代
    save_optimizer=True,  ## 保存优化器
    by_epoch=False)  ## 按迭代次数计数
evaluation = dict(  ## 构建验证钩子的配置
    interval=5000,  ## 执行验证的间隔为 5000 次迭代
    save_image=True,  ## 验证期间保存图像
    gpu_collect=True)  ## 使用 gpu 收集
log_config = dict(  ## 注册日志钩子的设置
    interval=100,  ## 打印日志间隔
    hooks=[
        dict(type='TextLoggerHook', by_epoch=False),  ## 记录训练过程信息的日志
        dict(type='TensorboardLoggerHook'),  ## 同时支持 Tensorboard 日志
    ])
visual_config = None  ## 可视化的设置

## 运行设置
dist_params = dict(backend='nccl')  ## 建立分布式训练的设置，其中端口号也可以设置
log_level = 'INFO'  ## 日志等级
work_dir = f'./work_dirs/{exp_name}'  ## 记录当前实验日志和模型权重文件的文件夹
load_from = None ## 从给定路径加载模型作为预训练模型. 这个选项不会用于断点恢复训练
resume_from = None ## 加载给定路径的模型权重文件作为断点续连的模型，训练将从该时间点保存的周期点继
续进行
workflow = [('train', 1)]  ## runner 的执行流. [('train', 1)] 代表只有一个执行流，并且这个名
为 train 的执行流只执行一次
```

16.10 配置文件 - 生成

与 MMDetection 一样，我们将模块化和继承设计融入我们的配置系统，以方便进行各种实验。

16.10.1 示例 - pix2pix

为了帮助用户对完整的配置和生成系统中的模块有一个基本的了解，我们对 pix2pix 的配置做如下简要说明。更详细的用法和各个模块对应的替代方案，请参考 API 文档。

```
## 模型设置
model = dict(
    type='Pix2Pix',  ## 合成器名称
    generator=dict(
        type='UnetGenerator',  ## 生成器名称
        in_channels=3,  ## 生成器的输入通道数
        out_channels=3,  ## 生成器的输出通道数
        num_down=8,  ## # 生成器中下采样的次数
        base_channels=64,  ## 生成器最后卷积层的通道数
        norm_cfg=dict(type='BN'),  ## 归一化层的配置
        use_dropout=True,  ## 是否在生成器中使用 dropout
        init_cfg=dict(type='normal', gain=0.02)),  ## 初始化配置
    discriminator=dict(
        type='PatchDiscriminator',  ## 判别器的名称
        in_channels=6,  ## 判别器的输入通道数
        base_channels=64,  ## 判别器第一卷积层的通道数
        num_conv=3,  ## 判别器中堆叠的中间卷积层（不包括输入和输出卷积层）的数量
        norm_cfg=dict(type='BN'),  ## 归一化层的配置
        init_cfg=dict(type='normal', gain=0.02)),  ## 初始化配置
    gan_loss=dict(
        type='GANLoss',  ## GAN 损失的名称
        gan_type='vanilla',  ## GAN 损失的类型
        real_label_val=1.0,  ## GAN 损失函数中真实标签的值
        fake_label_val=0.0,  ## GAN 损失函数中伪造标签的值
        loss_weight=1.0),  ## GAN 损失函数的权重
    pixel_loss=dict(type='L1Loss', loss_weight=100.0, reduction='mean'))
## 模型训练和测试设置
train_cfg = dict(
    direction='b2a')  ## pix2pix 的图像到图像的转换方向 (模型训练的方向，和测试方向一致)。模型默
认: a2b
test_cfg = dict(
    direction='b2a',   ## pix2pix 的图像到图像的转换方向 (模型测试的方向，和训练方向一致)。模型默
认: a2b
    show_input=True)  ## 保存 pix2pix 的测试图像时是否显示输入的真实图像
```

(下页继续)

(续上页)

```
## 数据设置
train_dataset_type = 'GenerationPairedDataset'  ## 训练数据集的类型
val_dataset_type = 'GenerationPairedDataset'  ## 验证/测试数据集类型
img_norm_cfg = dict(mean=[0.5, 0.5, 0.5], std=[0.5, 0.5, 0.5])  ## 输入图像归一化配置
train_pipeline = [
    dict(
        type='LoadPairedImageFromFile',  ## 从文件路径加载图像对
        io_backend='disk',  ## 存储图像的 IO 后端
        key='pair',  ## 查找对应路径的关键词
        flag='color'),  ## 加载图像标志
    dict(
        type='Resize',  ## 图像大小调整
        keys=['img_a', 'img_b'],  ## 要调整大小的图像的关键词
        scale=(286, 286),  ## 调整图像大小的比例
        interpolation='bicubic'),  ## 调整图像大小时用于插值的算法
    dict(
        type='FixedCrop',  ## 固定裁剪，在特定位置将配对图像裁剪为特定大小以训练 pix2pix
        keys=['img_a', 'img_b'],  ## 要裁剪的图像的关键词
        crop_size=(256, 256)),  ## 裁剪图像的大小
    dict(
        type='Flip',  ## 翻转图像
        keys=['img_a', 'img_b'],  ## 要翻转的图像的关键词
        direction='horizontal'),  ## 水平或垂直翻转图像
    dict(
        type='RescaleToZeroOne',  ## 将图像从 [0, 255] 缩放到 [0, 1]
        keys=['img_a', 'img_b']),  ## 要重新缩放的图像的关键词
    dict(
        type='Normalize',  ## 图像归一化
        keys=['img_a', 'img_b'],  ## 要归一化的图像的关键词
        to_rgb=True,  ## 是否将图像通道从 BGR 转换为 RGB
        **img_norm_cfg),  ## 图像归一化配置 (`img_norm_cfg` 的定义见上文)
    dict(
       type='ImageToTensor',  ## 将图像转化为 Tensor
       keys=['img_a', 'img_b']),  ## 要从图像转换为 Tensor 的图像的关键词
    dict(
        type='Collect',  ## 决定数据中哪些键应该传递给合成器
        keys=['img_a', 'img_b'],  ## 图像的关键词
        meta_keys=['img_a_path', 'img_b_path'])  ## 图片的元关键词
]
test_pipeline = [
    dict(
        type='LoadPairedImageFromFile',  ## 从文件路径加载图像对
```

(下页继续)

(续上页)

```
        io_backend='disk',  ## 存储图像的 IO 后端
        key='pair',  ## 查找对应路径的关键词
        flag='color'),  ## 加载图像标志
    dict(
        type='Resize',  ## 图像大小调整
        keys=['img_a', 'img_b'],  ## 要调整大小的图像的关键词
        scale=(256, 256),  ## 调整图像大小的比例
        interpolation='bicubic'),  ## 调整图像大小时用于插值的算法
    dict(
        type='RescaleToZeroOne',  ## 将图像从 [0, 255] 缩放到 [0, 1]
        keys=['img_a', 'img_b']),  ## 要重新缩放的图像的关键词
    dict(
        type='Normalize',  ## 图像归一化
        keys=['img_a', 'img_b'],  ## 要归一化的图像的关键词
        to_rgb=True,  ## 是否将图像通道从 BGR 转换为 RGB
        **img_norm_cfg),  ## 图像归一化配置 (`img_norm_cfg` 的定义见上文)
    dict(
       type='ImageToTensor',  ## 将图像转化为 Tensor
       keys=['img_a', 'img_b']),  ## 要从图像转换为 Tensor 的图像的关键词
    dict(
        type='Collect',  ## 决定数据中哪些键应该传递给合成器
        keys=['img_a', 'img_b'],  ## 图像的关键词
        meta_keys=['img_a_path', 'img_b_path'])  ## 图片的元关键词
]
data_root = 'data/pix2pix/facades'  ## 数据的根路径
data = dict(
    samples_per_gpu=1,  ## 单个 GPU 的批量大小
    workers_per_gpu=4,  ## 为每个 GPU 预取数据的 Worker 数
    drop_last=True,  ## 是否丢弃训练中的最后一批数据
    val_samples_per_gpu=1,  ## 验证中单个 GPU 的批量大小
    val_workers_per_gpu=0,  ## 在验证中为每个 GPU 预取数据的 Worker 数
    train=dict(  ## 训练数据集配置
        type=train_dataset_type,
        dataroot=data_root,
        pipeline=train_pipeline,
        test_mode=False),
    val=dict(  ## 验证数据集配置
        type=val_dataset_type,
        dataroot=data_root,
        pipeline=test_pipeline,
        test_mode=True),
    test=dict(  ## 测试数据集配置
        type=val_dataset_type,
```

(下页继续)

(续上页)

```
        dataroot=data_root,
        pipeline=test_pipeline,
        test_mode=True))

## 优化器
optimizers = dict(  ## 用于构建优化器的配置，支持 PyTorch 中所有优化器，且参数与 PyTorch 中对应优化器相同
    generator=dict(type='Adam', lr=2e-4, betas=(0.5, 0.999)),
    discriminator=dict(type='Adam', lr=2e-4, betas=(0.5, 0.999)))

## 学习策略
lr_config = dict(policy='Fixed', by_epoch=False)  ## 用于注册 LrUpdater 钩子的学习率调度程序配置

## 检查点保存
checkpoint_config = dict(interval=4000, save_optimizer=True, by_epoch=False)  ## 配置检查点钩子，实现参考 https://github.com/open-mmlab/mmcv/blob/master/mmcv/runner/hooks/
↪checkpoint.py
evaluation = dict(  ## 构建验证钩子的配置
    interval=4000,  ## 验证区间
    save_image=True)  ## 是否保存图片
log_config = dict(  ## 配置注册记录器钩子
    interval=100,  ## 打印日志的时间间隔
    hooks=[
        dict(type='TextLoggerHook', by_epoch=False),  ## 用于记录训练过程的记录器
        ## dict(type='TensorboardLoggerHook')  # 还支持 Tensorboard 记录器
    ])
visual_config = None  ## 构建可视化钩子的配置

## 运行设置
total_iters = 80000  ## 训练模型的总迭代次数
cudnn_benchmark = True  ## 设置 cudnn_benchmark
dist_params = dict(backend='nccl')  ## 设置分布式训练的参数，端口也可以设置
log_level = 'INFO'  ## 日志级别
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_from = None  ## 从给定路径恢复检查点，当检查点被保存时，训练将从该 epoch 恢复
workflow = [('train', 1)]  ## runner 的工作流程。 [('train', 1)] 表示只有一个工作流程，名为
↪'train' 的工作流程执行一次。 训练当前生成模型时保持不变
exp_name = 'pix2pix_facades'  ## 实验名称
work_dir = f'./work_dirs/{exp_name}'  ## 保存当前实验的模型检查点和日志的目录
```

16.11 配置文件 - 补全

16.11.1 配置名称样式

与 MMDetection 一样，我们将模块化和继承设计融入我们的配置系统，以方便进行各种实验。

16.11.2 配置字段说明

为了帮助用户对完整的配置和修复系统中的模块有一个基本的了解，我们对 Global&Local 的配置作如下简要说明。更详细的用法和各个模块对应的替代方案，请参考 API 文档。

```
model = dict(
    type='GLInpaintor', ## 补全器的名称
    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'))), ## 扩颈的配置
    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, ## 判别器中的最大通道数
            fc_in_channels=512 * 4 * 4, ## 最后一个全连接层的输入通道
            fc_out_channels=1024, ## 最后一个全连接层的输出通道
            num_convs=6, ## 判别器中使用的卷积数量
            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, ## 判别器中的最大通道数
            fc_in_channels=512 * 4 * 4, ## 最后一个全连接层的输入通道
            fc_out_channels=1024, ## 最后一个全连接层的输出通道
            num_convs=5, ## 判别器中使用的卷积数量
            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 损失函数的权重
    ),
```

(下页继续)

(续上页)

```
    loss_l1_hole=dict(
        type='L1Loss', ## L1 损失的类型
        loss_weight=1.0 ## L1 损失函数的权重
    ),
    pretrained=None) ## 预训练权重的路径

train_cfg = dict(
    disc_step=1, ## 训练生成器之前训练判别器的迭代次数
    iter_tc=90000, ## 预热生成器的迭代次数
    iter_td=100000, ## 预热判别器的迭代次数
    start_iter=0, ## 开始的迭代
    local_size=(128, 128)) ## 图像块的大小
test_cfg = dict(metrics=['l1']) ## 测试的配置

dataset_type = 'ImgInpaintingDataset' ## 数据集类型
input_shape = (256, 256) ## 输入图像的形状

train_pipeline = [
    dict(type='LoadImageFromFile', key='gt_img'), ## 加载图片的配置
    dict(
        type='LoadMask', ## 加载掩码
        mask_mode='bbox', ## 掩码的类型
        mask_config=dict(
            max_bbox_shape=(128, 128), ## 检测框的形状
            max_bbox_delta=40, ## 检测框高宽的变化
            min_margin=20,  ## 检测框到图片边界的最小距离
            img_shape=input_shape)),  ## 输入图像的形状
    dict(
        type='Crop', ## 裁剪
        keys=['gt_img'],  ## 要裁剪的图像的关键词
        crop_size=(384, 384),  ## 裁剪图像块的大小
        random_crop=True,  ## 是否使用随机裁剪
    ),
    dict(
        type='Resize',  ## 图像大小调整
        keys=['gt_img'],  ## 要调整大小的图像的关键词
        scale=input_shape,  ## 调整图像大小的比例
        keep_ratio=False,  ## 调整大小时是否保持比例
    ),
    dict(
        type='Normalize',  ## 图像归一化
        keys=['gt_img'],  ## 要归一化的图像的关键词
        mean=[127.5] * 3,  ## 归一化中使用的均值
```

(下页继续)

(续上页)

```
        std=[127.5] * 3,  ## 归一化中使用的标准差
        to_rgb=False),  ## 是否将图像通道从 BGR 转换为 RGB
    dict(type='GetMaskedImage'),  ## 获取被掩盖的图像
    dict(
        type='Collect',  ## 决定数据中哪些键应该传递给合成器
        keys=['gt_img', 'masked_img', 'mask', 'mask_bbox'],  ## 要收集的数据的关键词
        meta_keys=['gt_img_path']),  ## 要收集的数据的元关键词
    dict(type='ImageToTensor', keys=['gt_img', 'masked_img', 'mask']),  ## 将图像转化为␣
↪Tensor
    dict(type='ToTensor', keys=['mask_bbox'])  ## 转化为 Tensor
]

test_pipeline = train_pipeline  ## 构建测试/验证流程

data_root = 'data/places365'  ## 数据根目录

data = dict(
    samples_per_gpu=12,  ## 单个 GPU 的批量大小
    workers_per_gpu=8,  ## 为每个 GPU 预取数据的 Worker 数
    val_samples_per_gpu=1,  ## 验证中单个 GPU 的批量大小
    val_workers_per_gpu=8,  ## 在验证中为每个 GPU 预取数据的 Worker 数
    drop_last=True,  ## 是否丢弃训练中的最后一批数据
    train=dict(  ## 训练数据集配置
        type=dataset_type,
        ann_file=f'{data_root}/train_places_img_list_total.txt',
        data_prefix=data_root,
        pipeline=train_pipeline,
        test_mode=False),
    val=dict(  ## 验证数据集配置
        type=dataset_type,
        ann_file=f'{data_root}/val_places_img_list.txt',
        data_prefix=data_root,
        pipeline=test_pipeline,
        test_mode=True))

optimizers = dict(  ## 用于构建优化器的配置，支持 PyTorch 中所有优化器，且参数与 PyTorch 中对应
优化器相同
    generator=dict(type='Adam', lr=0.0004), disc=dict(type='Adam', lr=0.0004))

lr_config = dict(policy='Fixed', by_epoch=False)  ## 用于注册 LrUpdater 钩子的学习率调度程
序配置

checkpoint_config = dict(by_epoch=False, interval=50000)  ## 配置检查点钩子，实现参考␣
↪https://github.com/open-mmlab/mmcv/blob/master/mmcv/runner/hooks/checkpoint.py
```

(下页继续)

(续上页)

```
log_config = dict(  ## 配置注册记录器钩子
    interval=100,  ## 打印日志的时间间隔
    hooks=[
        dict(type='TextLoggerHook', by_epoch=False),
        ## dict(type='TensorboardLoggerHook'),  # 支持 Tensorboard 记录器
        ## dict(type='PaviLoggerHook', init_kwargs=dict(project='mmedit'))
    ])  ## 用于记录训练过程的记录器

visual_config = dict(  ## 构建可视化钩子的配置
    type='VisualizationHook',
    output_dir='visual',
    interval=1000,
    res_name_list=[
        'gt_img', 'masked_img', 'fake_res', 'fake_img', 'fake_gt_local'
    ],
)  ## 用于可视化训练过程的记录器。

evaluation = dict(interval=50000)  ## 构建验证钩子的配置

total_iters = 500002
dist_params = dict(backend='nccl')  ## 设置分布式训练的参数，端口也可以设置
log_level = 'INFO'  ## 日志级别
work_dir = None  ## 保存当前实验的模型检查点和日志的目录
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_from = None  ## 从给定路径恢复检查点，当检查点被保存时，训练将从该 epoch 恢复
workflow = [('train', 10000)]  ## runner 的工作流程。 [('train', 1)] 表示只有一个工作流程，名为 'train' 的工作流程执行一次。 训练当前生成模型时保持不变
exp_name = 'gl_places'  ## 实验名称
find_unused_parameters = False  ## 是否在分布式训练中查找未使用的参数
```

16.12 配置文件 - 抠图

与 MMDetection 一样，我们将模块化和继承设计融入我们的配置系统，以方便进行各种实验。

16.12.1 例子 - Deep Image Matting Model

为了帮助用户对一个完整的配置有一个基本的了解，我们对我们实现的原始 DIM 模型的配置做一个简短的评论，如下所示。更详细的用法和各个模块对应的替代方案，请参考 API 文档。

```
## 模型配置
model = dict(
    type='DIM',  ## 模型的名称（我们称之为抠图器）
    backbone=dict(  ## 主干网络的配置
        type='SimpleEncoderDecoder',  ## 主干网络的类型
        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',  ## 编码器的预训练权重
    loss_alpha=dict(  ## alpha 损失的配置
        type='CharbonnierLoss',  ## 预测的 alpha 遮罩的损失类型
        loss_weight=0.5),  ## alpha 损失的权重
    loss_comp=dict(  ## 组合损失的配置
        type='CharbonnierCompLoss',  ## 组合损失的类型
        loss_weight=0.5))  ## 组合损失的权重
train_cfg = dict(  ## 训练 DIM 模型的配置
    train_backbone=True,  ## 在 DIM stage 1 中，会对主干网络进行训练
    train_refiner=False)  ## 在 DIM stage 1 中，不会对精炼器进行训练
test_cfg = dict(  ## 测试 DIM 模型的配置
    refine=False,  ## 是否使用精炼器输出作为输出，在 stage 1 中，我们不使用它
    metrics=['SAD', 'MSE', 'GRAD', 'CONN'])  ## 测试时使用的指标

## 数据配置
dataset_type = 'AdobeComp1kDataset'  ## 数据集类型，这将用于定义数据集
data_root = 'data/adobe_composition-1k'  ## 数据的根目录
img_norm_cfg = dict(  ## 归一化输入图像的配置
    mean=[0.485, 0.456, 0.406],  ## 归一化中使用的均值
    std=[0.229, 0.224, 0.225],  ## 归一化中使用的标准差
    to_rgb=True)  ## 是否将图像通道从 BGR 转换为 RGB
train_pipeline = [  ## 训练数据处理流程
    dict(
        type='LoadImageFromFile',  ## 从文件加载 alpha 遮罩
        key='alpha',  ## 注释文件中 alpha 遮罩的键关键词。流程将从路径 "alpha_path" 中读取␣
↪alpha 遮罩
        flag='grayscale'),  ## 加载灰度图像，形状为（高度、宽度）
    dict(
        type='LoadImageFromFile',  ## 从文件中加载图像
        key='fg'),  ## 要加载的图像的关键词。流程将从路径 "fg_path" 读取 fg
```

（下页继续）

(续上页)

```
    dict(
        type='LoadImageFromFile',  ## 从文件中加载图像
        key='bg'),  ## 要加载的图像的关键词。流程将从路径 “bg_path” 读取 bg
    dict(
        type='LoadImageFromFile',  ## 从文件中加载图像
        key='merged'),  ## 要加载的图像的关键词。流程将从路径 “merged_path” 读取并合并
    dict(
        type='CropAroundUnknown',  ## 在未知区域（半透明区域）周围裁剪图像
        keys=['alpha', 'merged', 'ori_merged', 'fg', 'bg'],  ## 要裁剪的图像
        crop_sizes=[320, 480, 640]),  ## 裁剪大小
    dict(
        type='Flip',  ## 翻转图像
        keys=['alpha', 'merged', 'ori_merged', 'fg', 'bg']),  ## 要翻转的图像
    dict(
        type='Resize',  ## 图像大小调整
        keys=['alpha', 'merged', 'ori_merged', 'fg', 'bg'],  ## 图像调整大小的图像
        scale=(320, 320),  ## 目标大小
        keep_ratio=False),  ## 是否保持高宽比例
    dict(
        type='GenerateTrimap',  ## 从 alpha 遮罩生成三元图。
        kernel_size=(1, 30)),  ## 腐蚀/扩张内核大小的范围
    dict(
        type='RescaleToZeroOne', ## 将图像从 [0, 255] 缩放到 [0, 1]
        keys=['merged', 'alpha', 'ori_merged', 'fg', 'bg']),  ## 要重新缩放的图像
    dict(
        type='Normalize',  ## 图像归一化
        keys=['merged'],  ## 要归一化的图像
        **img_norm_cfg),  ## 图像归一化配置（`img_norm_cfg` 的定义见上文）
    dict(
        type='Collect',  ## 决定数据中哪些键应该传递给合成器
        keys=['merged', 'alpha', 'trimap', 'ori_merged', 'fg', 'bg'],  ## 图像的关键词
        meta_keys=[]),  ## 图片的元关键词，这里不需要元信息。
    dict(
        type='ImageToTensor',  ## 将图像转化为 Tensor
        keys=['merged', 'alpha', 'trimap', 'ori_merged', 'fg', 'bg']),  ## 要转换为␣
↪Tensor 的图像
]
test_pipeline = [
    dict(
        type='LoadImageFromFile', ## 从文件加载 alpha 遮罩
        key='alpha',  ## 注释文件中 alpha 遮罩的键关键词。流程将从路径 “alpha_path” 中读取␣
↪alpha 遮罩
        flag='grayscale',
```

(下页继续)

(续上页)

```
        save_original_img=True),
    dict(
        type='LoadImageFromFile',  ## 从文件中加载图像
        key='trimap',  ## 要加载的图像的关键词。流程将从路径 “trimap_path” 读取三元图
        flag='grayscale', ## 加载灰度图像，形状为（高度、宽度）
        save_original_img=True), ## 保存三元图用于计算指标。 它将与 “ori_trimap” 一起保存
    dict(
        type='LoadImageFromFile',  ## 从文件中加载图像
        key='merged'),  ## 要加载的图像的关键词。流程将从路径 “merged_path” 读取并合并
    dict(
        type='Pad',  ## 填充图像以与模型的下采样因子对齐
        keys=['trimap', 'merged'],  ## 要填充的图像
        mode='reflect'),  ## 填充模式
    dict(
        type='RescaleToZeroOne',  ## 与 train_pipeline 相同
        keys=['merged', 'ori_alpha']),  ## 要缩放的图像
    dict(
        type='Normalize',  ## 与 train_pipeline 相同
        keys=['merged'],
        **img_norm_cfg),
    dict(
        type='Collect',  ## 与 train_pipeline 相同
        keys=['merged', 'trimap'],
        meta_keys=[
            'merged_path', 'pad', 'merged_ori_shape', 'ori_alpha',
            'ori_trimap'
        ]),
    dict(
        type='ImageToTensor',  ## 与 train_pipeline 相同
        keys=['merged', 'trimap']),
]
data = dict(
    samples_per_gpu=1,  ## 单个 GPU 的批量大小
    workers_per_gpu=4,  ## 为每个 GPU 预取数据的 Worker 数
    drop_last=True,  ## 是否丢弃训练中的最后一批数据
    train=dict(  ## 训练数据集配置
        type=dataset_type,  ## 数据集的类型
        ann_file=f'{data_root}/training_list.json',  ## 注解文件路径
        data_prefix=data_root,  ## 图像路径的前缀
        pipeline=train_pipeline),  ## 见上文 train_pipeline
    val=dict(  ## 验证数据集配置
        type=dataset_type,
        ann_file=f'{data_root}/test_list.json',
```

(下页继续)

(续上页)

```
        data_prefix=data_root,
        pipeline=test_pipeline),  ## 见上文 test_pipeline
    test=dict(  ## 测试数据集配置
        type=dataset_type,
        ann_file=f'{data_root}/test_list.json',
        data_prefix=data_root,
        pipeline=test_pipeline))  ## 见上文 test_pipeline

## 优化器
optimizers = 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，且参数与 PyTorch 中对应优化器相同
## 学习策略
lr_config = dict(  ## 用于注册 LrUpdater 钩子的学习率调度程序配置
    policy='Fixed')  ## 调度器的策略，支持 CosineAnnealing、Cyclic 等。支持的 LrUpdater 详情请参考 https://github.com/open-mmlab/mmcv/blob/master/mmcv/runner/hooks/lr_updater.py
↪#L9。

## 检查点保存
checkpoint_config = dict(  ## 配置检查点钩子，实现参考 https://github.com/open-mmlab/mmcv/
↪blob/master/mmcv/runner/hooks/checkpoint.py
    interval=40000,  ## 保存间隔为 40000 次迭代
    by_epoch=False)  ## 按迭代计数
evaluation = dict(  ## # 构建验证钩子的配置
    interval=40000)  ## 验证区间
log_config = dict(  ## 配置注册记录器钩子
    interval=10,  ## 打印日志的时间间隔
    hooks=[
        dict(type='TextLoggerHook', by_epoch=False),  ## 用于记录训练过程的记录器
        ## dict(type='TensorboardLoggerHook')  # 支持 Tensorboard 记录器
    ])

## runtime settings
total_iters = 1000000  ## 训练模型的总迭代次数
dist_params = dict(backend='nccl')  ## 设置分布式训练的参数，端口也可以设置
log_level = 'INFO'  ## 日志级别
work_dir = './work_dirs/dim_stage1'  ## 保存当前实验的模型检查点和日志的目录
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_from = None  ## 从给定路径恢复检查点，当检查点被保存时，训练将从该 epoch 恢复
workflow = [('train', 1)]  ## runner 的工作流程。 [('train', 1)] 表示只有一个工作流程，名为
↪'train' 的工作流程执行一次。 训练当前抠图模型时保持不变
```

16.13 配置文件 - 复原

16.13.1 示例-EDSR

为了帮助用户理解 mmediting 的配置文件结构，这里以 EDSR 为例，给出其配置文件的注释。对于每个模块的详细用法以及对应参数的选择，请参照 API 文档。

```
exp_name = 'edsr_x2c64b16_1x16_300k_div2k'  ## 实验名称

scale = 2  ## 上采样放大因子

## 模型设置
model = dict(
    type='BasicRestorer',  ## 图像恢复模型类型
    generator=dict(  ## 生成器配置
        type='EDSR',  ## 生成器类型
        in_channels=3,  ## 输入通道数
        out_channels=3,  ## 输出通道数
        mid_channels=64,  ## 中间特征通道数
        num_blocks=16,  ## 残差块数目
        upscale_factor=scale, ## 上采样因子
        res_scale=1,  ## 残差缩放因子
        rgb_mean=(0.4488, 0.4371, 0.4040),  ## 输入图像 RGB 通道的平均值
        rgb_std=(1.0, 1.0, 1.0)),  ## 输入图像 RGB 通道的方差
    pixel_loss=dict(type='L1Loss', loss_weight=1.0, reduction='mean'))  ## 像素损失函数的
配置

## 模型训练和测试设置
train_cfg = None  ## 训练的配置
test_cfg = dict(  ## 测试的配置
    metrics=['PSNR'],  ## 测试时使用的评价指标
    crop_border=scale)  ## 测试时裁剪的边界尺寸

## 数据集设置
train_dataset_type = 'SRAnnotationDataset'  ## 用于训练的数据集类型
val_dataset_type = 'SRFolderDataset'  ##  用于验证的数据集类型
train_pipeline = [  ## 训练数据前处理流水线步骤组成的列表
    dict(type='LoadImageFromFile',  ## 从文件加载图像
        io_backend='disk',  ## 读取图像时使用的 io 类型
        key='lq',  ## 设置 LR 图像的键来找到相应的路径
        flag='unchanged'),  ## 读取图像的标识
    dict(type='LoadImageFromFile',  ## 从文件加载图像
        io_backend='disk',  ## 读取图像时使用的 io 类型
        key='gt',  ## 设置 HR 图像的键来找到相应的路径
```

(下页继续)

(续上页)

```
        flag='unchanged'),  ## 读取图像的标识
    dict(type='RescaleToZeroOne', keys=['lq', 'gt']),  ## 将图像从 [0, 255] 重缩放到 [0,
1]
    dict(type='Normalize',  ## 正则化图像
        keys=['lq', 'gt'],  ## 执行正则化图像的键
        mean=[0, 0, 0],  ## 平均值
        std=[1, 1, 1],  ## 标准差
        to_rgb=True),  ## 更改为 RGB 通道
    dict(type='PairedRandomCrop', gt_patch_size=96),  ## LR 和 HR 成对随机裁剪
    dict(type='Flip',  ## 图像翻转
        keys=['lq', 'gt'],  ## 执行翻转图像的键
        flip_ratio=0.5,  ## 执行翻转的几率
        direction='horizontal'),  ## 翻转方向
    dict(type='Flip',  ## 图像翻转
        keys=['lq', 'gt'],  ## 执行翻转图像的键
        flip_ratio=0.5,  ## 执行翻转几率
        direction='vertical'),  ## 翻转方向
    dict(type='RandomTransposeHW',  ## 图像的随机的转置
        keys=['lq', 'gt'],  ## 执行转置图像的键
        transpose_ratio=0.5  ## 执行转置的几率
        ),
    dict(type='Collect',  ## Collect 类决定哪些键会被传递到生成器中
        keys=['lq', 'gt'],  ## 传入模型的键
        meta_keys=['lq_path', 'gt_path']), ## 元信息键。在训练中，不需要元信息
    dict(type='ImageToTensor',  ## 将图像转换为张量
        keys=['lq', 'gt'])  ## 执行图像转换为张量的键
]
test_pipeline = [  ## 测试数据前处理流水线步骤组成的列表
    dict(
        type='LoadImageFromFile',  ## 从文件加载图像
        io_backend='disk',  ## 读取图像时使用的 io 类型
        key='lq',  ## 设置 LR 图像的键来找到相应的路径
        flag='unchanged'),  ## 读取图像的标识
    dict(
        type='LoadImageFromFile',  ## 从文件加载图像
        io_backend='disk',  ## 读取图像时使用的 io 类型
        key='gt',  ## 设置 HR 图像的键来找到相应的路径
        flag='unchanged'),  ## 读取图像的标识
    dict(type='RescaleToZeroOne', keys=['lq', 'gt']),  ## 将图像从 [0, 255] 重缩放到 [0,
1]
    dict(
        type='Normalize',  ## 正则化图像
        keys=['lq', 'gt'],  ## 执行正则化图像的键
```

(下页继续)

(续上页)

```
        mean=[0, 0, 0],  ## 平均值
        std=[1, 1, 1],  ## 标准差
        to_rgb=True),  ## 更改为 RGB 通道
    dict(type='Collect',  ## Collect 类决定哪些键会被传递到生成器中
        keys=['lq', 'gt'],  ## 传入模型的键
        meta_keys=['lq_path', 'lq_path']),  ## 元信息键
    dict(type='ImageToTensor',  ## 将图像转换为张量
        keys=['lq', 'gt'])  ## 执行图像转换为张量的键
]

data = dict(
    ## 训练
    samples_per_gpu=16,  ## 单个 GPU 的批大小
    workers_per_gpu=6,  ## 单个 GPU 的 dataloader 的进程
    drop_last=True,  ## 在训练过程中丢弃最后一个批次
    train=dict(  ## 训练数据集的设置
        type='RepeatDataset',  ## 基于迭代的重复数据集
        times=1000,  ## 重复数据集的重复次数
        dataset=dict(
            type=train_dataset_type,  ## 数据集类型
            lq_folder='data/DIV2K/DIV2K_train_LR_bicubic/X2_sub',  ## lq 文件夹的路径
            gt_folder='data/DIV2K/DIV2K_train_HR_sub',  ## gt 文件夹的路径
            ann_file='data/DIV2K/meta_info_DIV2K800sub_GT.txt',  ## 批注文件的路径
            pipeline=train_pipeline,  ## 训练流水线，如上所示
            scale=scale)),  ## 上采样放大因子

    ## 验证
    val_samples_per_gpu=1,  ## 验证时单个 GPU 的批大小
    val_workers_per_gpu=1,  ## 验证时单个 GPU 的 dataloader 的进程
    val=dict(
        type=val_dataset_type,  ## 数据集类型
        lq_folder='data/val_set5/Set5_bicLRx2',  ## lq 文件夹的路径
        gt_folder='data/val_set5/Set5_mod12',  ## gt 文件夹的路径
        pipeline=test_pipeline,  ## 测试流水线，如上所示
        scale=scale,  ## 上采样放大因子
        filename_tmpl='{}'),  ## 文件名模板

    ## 测试
    test=dict(
        type=val_dataset_type,  ## 数据集类型
        lq_folder='data/val_set5/Set5_bicLRx2',  ## lq 文件夹的路径
        gt_folder='data/val_set5/Set5_mod12',  ## gt 文件夹的路径
        pipeline=test_pipeline,  ## 测试流水线，如上所示
```

(下页继续)

(续上页)

```
        scale=scale,  ## 上采样放大因子
        filename_tmpl='{}'))  ## 文件名模板

## 优化器设置
optimizers = dict(generator=dict(type='Adam', lr=1e-4, betas=(0.9, 0.999)))  ## 用于构建
优化器的设置，支持 PyTorch 中所有参数与 PyTorch 中参数相同的优化器

## 学习策略
total_iters = 300000  ## 训练模型的总迭代数
lr_config = dict( ## 用于注册 LrUpdater 钩子的学习率调度程序配置
    policy='Step', by_epoch=False, step=[200000], gamma=0.5)  ## 调度器的策略，还支持余弦、
循环等

checkpoint_config = dict(  ## 模型权重钩子设置，更多细节可参考 https://github.com/open-
↪mmlab/mmcv/blob/master/mmcv/runner/hooks/checkpoint.py
    interval=5000,  ## 模型权重文件保存间隔为 5000 次迭代
    save_optimizer=True,  ## 保存优化器
    by_epoch=False)  ## 按迭代次数计数
evaluation = dict(  ## 构建验证钩子的配置
    interval=5000,  ## 执行验证的间隔为 5000 次迭代
    save_image=True,  ## 验证期间保存图像
    gpu_collect=True)  ## 使用 gpu 收集
log_config = dict(  ## 注册日志钩子的设置
    interval=100,  ## 打印日志间隔
    hooks=[
        dict(type='TextLoggerHook', by_epoch=False),  ## 记录训练过程信息的日志
        dict(type='TensorboardLoggerHook'),  ## 同时支持 Tensorboard 日志
    ])
visual_config = None  ## 可视化的设置

## 运行设置
dist_params = dict(backend='nccl')  ## 建立分布式训练的设置，其中端口号也可以设置
log_level = 'INFO'  ## 日志等级
work_dir = f'./work_dirs/{exp_name}'  ## 记录当前实验日志和模型权重文件的文件夹
load_from = None ## 从给定路径加载模型作为预训练模型. 这个选项不会用于断点恢复训练
resume_from = None ## 加载给定路径的模型权重文件作为断点续连的模型，训练将从该时间点保存的周期点继
续进行
workflow = [('train', 1)]  ## runner 的执行流. [('train', 1)] 代表只有一个执行流，并且这个名
为 train 的执行流只执行一次
```

CHAPTER 17

实用工具

我们在 tools/ 目录下提供了很多有用的工具。

17.1 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。

17.2 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`.

17.3 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`。

17.3.1 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 `mmedit`。

17.4 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/mmediting/blob/764e6065e315b7d0033762038fcbf0bb1c570d4d/mmedit.bones/modelsrnn py#L40）。因为 TensorRT 目前不支持 bicubic 插值，最终性能将下降约 4%。

17.4.1 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 `mmedit`。

17.5 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。

17.5.1 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）使用 ONNXRuntime 自定义操作和 TensorRT 插件安装 `mmcv-full`。

17.5.2 用法

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

17.5.3 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。

17.5.4 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 18

基准

详情请参阅 model_zoo。

CHAPTER 19

常见问题解答

19.1 在配置中使用中间变量

配置文件中使用了一些中间变量，如数据集中的 train_pipeline 和 test_pipeline。

例如，我们通常先定义 train_pipeline 和 test_pipeline，再将它们传递到 data 中。因此，train_pipeline 和 test_pipeline 是中间变量。

```
...
train_dataset_type = 'SRAnnotationDataset'
val_dataset_type = 'SRFolderDataset'
train_pipeline = [
    dict(
        type='LoadImageFromFile',
        io_backend='disk',
        key='lq',
        flag='unchanged'),
    ...
    dict(type='Collect', keys=['lq', 'gt'], meta_keys=['lq_path', 'gt_path']),
    dict(type='ImageToTensor', keys=['lq', 'gt'])
]
test_pipeline = [
    dict(
        type='LoadImageFromFile',
        io_backend='disk',
```

(下页继续)

(续上页)

```
        key='lq',
        flag='unchanged'),
    ...
    dict(type='Collect', keys=['lq', 'gt'], meta_keys=['lq_path', 'lq_path']),
    dict(type='ImageToTensor', keys=['lq', 'gt'])
]

data = dict(
    # 训练
    train_dataloader = dict(
        samples_per_gpu=16,
        workers_per_gpu=6,
        drop_last=True),
    train=dict(
        type='RepeatDataset',
        times=1000,
        dataset=dict(
            type=train_dataset_type,
            lq_folder='data/DIV2K/DIV2K_train_LR_bicubic/X2_sub',
            gt_folder='data/DIV2K/DIV2K_train_HR_sub',
            ann_file='data/DIV2K/meta_info_DIV2K800sub_GT.txt',
            pipeline=train_pipeline,
            scale=scale)),
    # 验证
    val_dataloader = dict(samples_per_gpu=1, workers_per_gpu=1),
    val=dict(
        type=val_dataset_type,
        lq_folder='data/val_set5/Set5_bicLRx2',
        gt_folder='data/val_set5/Set5_mod12',
        pipeline=test_pipeline,
        scale=scale,
        filename_tmpl='{}')

empty_cache = True  # 在每次迭代后清空缓存。
```

CHAPTER 20

mmedit.core

class mmedit.core.**DistEvalIterHook**(*dataloader*, *interval=1*, *gpu_collect=False*, ***eval_kwargs*)

Distributed evaluation hook.

参数

- **dataloader** (*DataLoader*) –A PyTorch dataloader.
- **interval** (*int*) –Evaluation interval. Default: 1.
- **tmpdir** (*str | None*) –Temporary directory to save the results of all processes. Default: None.
- **gpu_collect** (*bool*) –Whether to use gpu or cpu to collect results. Default: False.
- **eval_kwargs** (*dict*) –Other eval kwargs. It may contain: save_image (bool): Whether save image. save_path (str): The path to save image.

after_train_iter(*runner*)

The behavior after each train iteration.

参数 **runner** (mmcv.runner.BaseRunner) –The runner.

class mmedit.core.**EvalIterHook**(*dataloader*, *interval=1*, ***eval_kwargs*)

Non-Distributed evaluation hook for iteration-based runner.

This hook will regularly perform evaluation in a given interval when performing in non-distributed environment.

参数

- **dataloader** (*DataLoader*) –A PyTorch dataloader.

- **interval** (*int*) –Evaluation interval. Default: 1.
- **eval_kwargs** (*dict*) –Other eval kwargs. It contains: save_image (bool): Whether to save image. save_path (str): The path to save image.

after_train_iter(*runner*)

The behavior after each train iteration.

参数 **runner** (mmcv.runner.BaseRunner) –The runner.

evaluate(*runner*, *results*)

Evaluation function.

参数

- **runner** (mmcv.runner.BaseRunner) –The runner.
- **results** (*dict*) –Model forward results.

class mmedit.core.**L1Evaluation**

L1 evaluation metric.

参数 **data_dict** (*dict*) –Must contain keys of ‘gt_img’ and ‘fake_res’. If ‘mask’ is given, the results will be computed with mask as weight.

class mmedit.core.**LinearLrUpdaterHook**(*target_lr=0*, *start=0*, *interval=1*, ***kwargs*)

Linear learning rate scheduler for image generation.

In the beginning, the learning rate is ‘base_lr’ defined in mmcv. We give a target learning rate ‘target_lr’ and a start point ‘start’ (iteration / epoch). Before ‘start’, we fix learning rate as ‘base_lr’; After ‘start’, we linearly update learning rate to ‘target_lr’.

参数

- **target_lr** (*float*) –The target learning rate. Default: 0.
- **start** (*int*) –The start point (iteration / epoch, specified by args ‘by_epoch’ in its parent class in mmcv) to update learning rate. Default: 0.
- **interval** (*int*) –The interval to update the learning rate. Default: 1.

get_lr(*runner*, *base_lr*)

Calculates the learning rate.

参数

- **runner** (*object*) –The passed runner.
- **base_lr** (*float*) –Base learning rate.

返回 Current learning rate.

返回类型 float

class mmedit.core.**VisualizationHook**(*output_dir*, *res_name_list*, *interval=- 1*, *filename_tmpl='iter_{}.png'*, *rerange=True*, *bgr2rgb=True*, *nrow=1*, *padding=4*)

Visualization hook.

In this hook, we use the official api *save_image* in torchvision to save the visualization results.

参数

- **output_dir** (*str*) –The file path to store visualizations.
- **res_name_list** (*str*) –The list contains the name of results in outputs dict. The results in outputs dict must be a torch.Tensor with shape (n, c, h, w).
- **interval** (*int*) –The interval of calling this hook. If set to -1, the visualization hook will not be called. Default: -1.
- **filename_tmpl** (*str*) –Format string used to save images. The output file name will be formatted as this args. Default: 'iter_{}.png'.
- **rerange** (*bool*) –Whether to rerange the output value from [-1, 1] to [0, 1]. We highly recommend users should preprocess the visualization results on their own. Here, we just provide a simple interface. Default: True.
- **bgr2rgb** (*bool*) –Whether to reformat the channel dimension from BGR to RGB. The final image we will save is following RGB style. Default: True.
- **nrow** (*int*) –The number of samples in a row. Default: 1.
- **padding** (*int*) –The number of padding pixels between each samples. Default: 4.

after_train_iter(*runner*)

The behavior after each train iteration.

参数 **runner** (*object*) –The runner.

mmedit.core.**build_optimizers**(*model*, *cfgs*)

Build multiple optimizers from configs.

If *cfgs* contains several dicts for optimizers, then a dict for each constructed optimizers will be returned. If *cfgs* only contains one optimizer config, the constructed optimizer itself will be returned.

For example,

1) Multiple optimizer configs:

```
optimizer_cfg = dict(
    model1=dict(type='SGD', lr=lr),
    model2=dict(type='SGD', lr=lr))
```

The return dict is `dict('model1': torch.optim.Optimizer, 'model2': torch.optim.Optimizer)`

2) Single optimizer config:

```
optimizer_cfg = dict(type='SGD', lr=lr)
```

The return is `torch.optim.Optimizer`.

参数

- **model** (`nn.Module`) –The model with parameters to be optimized.
- **cfgs** (*dict*) –The config dict of the optimizer.

返回 The initialized optimizers.

返回类型 dict[`torch.optim.Optimizer`]|`torch.optim.Optimizer`

`mmedit.core.`**psnr**(*img1*, *img2*, *crop_border=0*, *input_order='HWC'*, *convert_to=None*)

Calculate PSNR (Peak Signal-to-Noise Ratio).

Ref: https://en.wikipedia.org/wiki/Peak_signal-to-noise_ratio

参数

- **img1** (*ndarray*) –Images with range [0, 255].
- **img2** (*ndarray*) –Images with range [0, 255].
- **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.

返回 psnr result.

返回类型 float

`mmedit.core.`**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** (*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 ndarray

`mmedit.core.ssim`(*img1*, *img2*, *crop_border=0*, *input_order='HWC'*, *convert_to=None*)

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

- **img1** (*ndarray*) –Images with range [0, 255].
- **img2** (*ndarray*) –Images with range [0, 255].
- **crop_border** (*int*) –Cropped pixels in each edges of an image. These pixels are not involved in the SSIM 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.

返回 ssim result.

返回类型 float

`mmedit.core.tensor2img`(*tensor*, *out_type=<class 'numpy.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])

CHAPTER 21

mmedit.datasets

21.1 datasets

class mmedit.datasets.**AdobeComp1kDataset**(*ann_file*, *pipeline*, *data_prefix=None*, *test_mode=False*)

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'
    },
]
```

load_annotations()

Load annoations for Adobe Composition-1k dataset.

It loads image paths from json file.

返回 Loaded dict.

返回类型 dict

class mmedit.datasets.**BaseDataset**(*pipeline*, *test_mode=False*)

Base class for datasets.

All datasets should subclass it. All subclasses should overwrite:

`load_annotations`, supporting to load information and generate image lists.

参数

- **pipeline** (*list[dict | callable]*) –A sequence of data transforms.
- **test_mode** (*bool*) –If True, the dataset will work in test mode. Otherwise, in train mode.

abstract load_annotations()

Abstract function for loading annotation.

All subclasses should overwrite this function

prepare_test_data(*idx*)

Prepare testing data.

参数 **idx** (*int*) –Index for getting each testing batch.

返回 Returned testing batch.

返回类型 Tensor

prepare_train_data(*idx*)

Prepare training data.

参数 **idx** (*int*) –Index of the training batch data.

返回 Returned training batch.

返回类型 dict

class mmedit.datasets.**BaseGenerationDataset**(*pipeline*, *test_mode=False*)

Base class for generation datasets.

evaluate(*results*, *logger=None*)

Evaluating with saving generated images. (needs no metrics)

参数 **results** (*list[tuple]*) –The output of forward_test() of the model.

返回 Evaluation results dict.

返回类型 dict

static 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 mmedit.datasets.**BaseMattingDataset**(*ann_file*, *pipeline*, *data_prefix=None*, *test_mode=False*)

Base image matting dataset.

evaluate(*results*, *logger=None*)

Evaluating with different metrics.

参数 **results** (*list[tuple]*) –The output of forward_test() of the model.

返回 Evaluation results dict.

返回类型 dict

class mmedit.datasets.**BaseSRDataset**(*pipeline*, *scale*, *test_mode=False*)

Base class for super resolution datasets.

evaluate(*results*, *logger=None*)

Evaluate with different metrics.

参数 **results** (*list[tuple]*) –The output of forward_test() of the model.

返回 Evaluation results dict.

返回类型 dict

static 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 form given folder.

返回类型 list[str]

class mmedit.datasets.**GenerationPairedDataset**(*dataroot*, *pipeline*, *test_mode=False*)

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*.

load_annotations()

Load paired image paths.

返回 List that contains paired image paths.

返回类型 list[dict]

class mmedit.datasets.**GenerationUnpairedDataset**(*dataroot*, *pipeline*, *test_mode=False*)

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.
- **test_mode** (*bool*) –Store *True* when building test dataset. Default: *False*.

load_annotations(*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]

prepare_test_data(*idx*)

Prepare unpaired test data.

参数 **idx** (*int*) –Index of current batch.

返回 Prepared test data batch.

返回类型 list[dict]

prepare_train_data(*idx*)

Prepare unpaired training data.

参数 **idx** (*int*) –Index of current batch.

返回 Prepared training data batch.

返回类型 dict

class mmedit.datasets.**ImgInpaintingDataset**(*ann_file*, *pipeline*, *data_prefix=None*, *test_mode=False*)

Image dataset for inpainting.

load_annotations()

Load annotations for dataset.

返回 Contain dataset annotations.

返回类型 list[dict]

class mmedit.datasets.**RepeatDataset**(*dataset*, *times*)

A wrapper of repeated dataset.

The length of repeated dataset will be *times* larger than the original dataset. This is useful when the data loading time is long but the dataset is small. Using RepeatDataset can reduce the data loading time between epochs.

参数

- **dataset** (Dataset) –The dataset to be repeated.
- **times** (*int*) –Repeat times.

class mmedit.datasets.**SRAnnotationDataset**(*lq_folder*, *gt_folder*, *ann_file*, *pipeline*, *scale*, *test_mode=False*, *filename_tmpl='{}'*)

General paired image dataset with an annotation file for image restoration.

The dataset loads lq (Low Quality) and gt (Ground-Truth) image pairs, applies specified transforms and finally returns a dict containing paired data and other information.

This is the "annotation file mode": Each line in the annotation file contains the image names and image shape (usually for gt), separated by a white space.

Example of an annotation file:

```
0001_s001.png (480,480,3)
0001_s002.png (480,480,3)
```

参数

- **lq_folder** (str | Path) –Path to a lq folder.
- **gt_folder** (str | Path) –Path to a gt folder.
- **ann_file** (str | Path) –Path to the annotation file.
- **pipeline** (*list[dict | callable]*) –A sequence of data transformations.
- **scale** (*int*) –Upsampling scale ratio.
- **test_mode** (*bool*) –Store *True* when building test dataset. Default: *False*.
- **filename_tmpl** (*str*) –Template for each filename. Note that the template excludes the file extension. Default: '{}'.

load_annotations()

Load annoations for SR dataset.

It loads the LQ and GT image path from the annotation file. Each line in the annotation file contains the image names and image shape (usually for gt), separated by a white space.

返回 Returned dict for LQ and GT pairs.

返回类型 dict

class mmedit.datasets.**SRFacialLandmarkDataset**(*gt_folder*, *ann_file*, *pipeline*, *scale*, *test_mode=False*)

Facial image and landmark dataset with an annotation file for image restoration.

The dataset loads gt (Ground-Truth) image, shape of image, face box, and landmark. Applies specified transforms and finally returns a dict containing paired data and other information.

This is the "annotation file mode": Each dict in the annotation list contains the image names, image shape, face box, and landmark.

Annotation file is a *npy* file, which contains a list of dict. Example of an annotation file:

```
dict1(file=*, bbox=*, shape=*, landmark=*)
dict2(file=*, bbox=*, shape=*, landmark=*)
```

参数

- **gt_folder** (str | Path) –Path to a gt folder.
- **ann_file** (str | Path) –Path to the annotation file.
- **pipeline** (*list[dict | callable]*) –A sequence of data transformations.
- **scale** (*int*) –Upsampling scale ratio.
- **test_mode** (*bool*) –Store *True* when building test dataset. Default: *False*.

load_annotations()

Load annoations for SR dataset.

Annotation file is a *npy* file, which contains a list of dict.

It loads the GT image path and landmark from the annotation file. Each dict in the annotation file contains the image names, image shape (usually for gt), bbox and landmark.

返回

Returned dict for GT and landmark. Contains: gt_path, bbox, shape, landmark.

返回类型 dict

class mmedit.datasets.**SRFolderDataset**(*lq_folder*, *gt_folder*, *pipeline*, *scale*, *test_mode=False*, *filename_tmpl='{}'*)

General paired image folder dataset for image restoration.

The dataset loads lq (Low Quality) and gt (Ground-Truth) image pairs, applies specified transforms and finally returns a dict containing paired data and other information.

This is the "folder mode", which needs to specify the lq folder path and gt folder path, each folder containing the corresponding images. Image lists will be generated automatically. You can also specify the filename template to match the lq and gt pairs.

For example, we have two folders with the following structures:

```
data_root
├── lq
|   ├── 0001_x4.png
|   ├── 0002_x4.png
├── gt
|   ├── 0001.png
|   ├── 0002.png
```

then, you need to set:

```
lq_folder = data_root/lq
gt_folder = data_root/gt
filename_tmpl = '{}_x4'
```

参数

- **lq_folder** (str | Path) –Path to a lq folder.
- **gt_folder** (str | Path) –Path to a gt folder.
- **pipeline** (*List[dict | callable]*) –A sequence of data transformations.
- **scale** (*int*) –Upsampling scale ratio.

- **test_mode** (*bool*) –Store *True* when building test dataset. Default: *False*.
- **filename_tmpl** (*str*) –Template for each filename. Note that the template excludes the file extension. Default: '{}'.

load_annotations()

Load annoations for SR dataset.

It loads the LQ and GT image path from folders.

返回 Returned dict for LQ and GT pairs.

返回类型 dict

class mmedit.datasets.**SRFolderGTDataset**(*gt_folder*, *pipeline*, *scale*, *test_mode=False*, *filename_tmpl='{}'*)

General ground-truth image folder dataset for image restoration.

The dataset loads gt (Ground-Truth) image only, applies specified transforms and finally returns a dict containing paired data and other information.

This is the "gt folder mode", which needs to specify the gt folder path, each folder containing the corresponding images. Image lists will be generated automatically.

For example, we have a folder with the following structure:

```
data_root
├── gt
|   ├── 0001.png
|   ├── 0002.png
```

then, you need to set:

```
gt_folder = data_root/gt
```

参数

- **gt_folder** (str | Path) –Path to a gt folder.
- **pipeline** (*List[dict | callable]*) –A sequence of data transformations.
- **scale** (*int | tuple*) –Upsampling scale or upsampling scale range.
- **test_mode** (*bool*) –Store *True* when building test dataset. Default: *False*.

load_annotations()

Load annoations for SR dataset.

It loads the GT image path from folder.

返回 Returned dict for GT.

返回类型 dict

class mmedit.datasets.**SRFolderMultipleGTDataset**(*lq_folder*, *gt_folder*, *pipeline*, *scale*, *ann_file=None*, *num_input_frames=None*, *test_mode=True*)

General dataset for video super resolution, used for recurrent networks.

The dataset loads several LQ (Low-Quality) frames and GT (Ground-Truth) frames. Then it applies specified transforms and finally returns a dict containing paired data and other information.

This dataset takes an annotation file specifying the sequences used in training or test. If no annotation file is provided, it assumes all video sequences under the root directory is used for training or test.

In the annotation file (.txt), each line contains:

1. folder name;
2. number of frames in this sequence (in the same folder)

Examples:

```
calendar 41
city 34
foliage 49
walk 47
```

参数

- **lq_folder** (str | Path) –Path to a lq folder.
- **gt_folder** (str | Path) –Path to a gt folder.
- **pipeline** (*list[dict | callable]*) –A sequence of data transformations.
- **scale** (*int*) –Upsampling scale ratio.
- **ann_file** (*str*) –The path to the annotation file. If None, we assume that all sequences in the folder is used. Default: None
- **num_input_frames** (*None | int*) –The number of frames per iteration. If None, the whole clip is extracted. If it is a positive integer, a sequence of 'num_input_frames' frames is extracted from the clip. Note that non-positive integers are not accepted. Default: None.
- **test_mode** (*bool*) –Store *True* when building test dataset. Default: *True*.

load_annotations()

Load annoations for the dataset.

返回 Returned dict for LQ and GT pairs.

返回类型 dict

class mmedit.datasets.**SRFolderRefDataset**(*pipeline*, *scale*, *ref_folder*, *gt_folder=None*, *lq_folder=None*, *test_mode=False*, *filename_tmpl_gt='{}'*, *filename_tmpl_lq='{}'*)

General paired image folder dataset for reference-based image restoration.

The dataset loads ref (reference) image pairs Must contain: ref (reference) Optional: GT (Ground-Truth), LQ (Low Quality), or both

Cannot only contain ref.

Applies specified transforms and finally returns a dict containing paired data and other information.

This is the "folder mode", which needs to specify the ref folder path and gt folder path, each folder containing the corresponding images. Image lists will be generated automatically. You can also specify the filename template to match the image pairs.

For example, we have three folders with the following structures:

```
data_root
├── ref
│   ├── 0001.png
│   ├── 0002.png
├── gt
│   ├── 0001.png
│   ├── 0002.png
├── lq
│   ├── 0001_x4.png
│   ├── 0002_x4.png
```

then, you need to set:

```
ref_folder = 'data_root/ref'
gt_folder = 'data_root/gt'
lq_folder = 'data_root/lq'
filename_tmpl_gt='{}'
filename_tmpl_lq='{}_x4'
```

参数

- **pipeline** (*List[dict | callable]*) –A sequence of data transformations.
- **scale** (*int*) –Upsampling scale ratio.
- **ref_folder** (str | Path) –Path to a ref folder.
- **gt_folder** (str | Path | None) –Path to a gt folder. Default: None.
- **lq_folder** (str | Path | None) –Path to a gt folder. Default: None.
- **test_mode** (*bool*) –Store *True* when building test dataset. Default: *False*.

- **filename_tmpl_gt** (*str*) –Template for gt filename. Note that the template excludes the file extension. Default: '{}'.
- **filename_tmpl_lq** (*str*) –Template for lq filename. Note that the template excludes the file extension. Default: '{}'.

load_annotations()

Load annoations for SR dataset.

It loads the ref, LQ and GT image path from folders.

返回 Returned dict for ref, LQ and GT pairs.

返回类型 dict

class mmedit.datasets.**SRFolderVideoDataset**(*lq_folder*, *gt_folder*, *num_input_frames*, *pipeline*, *scale*, *ann_file=None*, *filename_tmpl='{:08d}'*, *metric_average_mode='clip'*, *test_mode=True*)

General dataset for video SR, used for sliding-window framework.

The dataset loads several LQ (Low-Quality) frames and one GT (Ground-Truth) frames. Then it applies specified transforms and finally returns a dict containing paired data and other information.

This dataset takes an annotation file specifying the sequences used in training or test. If no annotation file is provided, it assumes all video sequences under the root directory are used for training or test.

In the annotation file (.txt), each line contains:

1. image name (no file extension);
2. number of frames in the sequence (in the same folder)

Examples:

```
calendar/00000000 41
calendar/00000001 41
...
calendar/00000040 41
city/00000000 34
...
```

参数

- **lq_folder** (str | Path) –Path to a lq folder.
- **gt_folder** (str | Path) –Path to a gt folder.
- **num_input_frames** (*int*) –Window size for input frames.
- **pipeline** (*list[dict | callable]*) –A sequence of data transformations.
- **scale** (*int*) –Upsampling scale ratio.

- **ann_file** (*str*) –The path to the annotation file. If None, we assume that all sequences in the folder is used. Default: None.
- **filename_tmpl** (*str*) –Template for each filename. Note that the template excludes the file extension. Default: '{:08d}'.
- **metric_average_mode** (*str*) –The way to compute the average metric. If 'clip', we first compute an average value for each clip, and then average the values from different clips. If 'all', we compute the average of all frames. Default: 'clip'.
- **test_mode** (*bool*) –Store *True* when building test dataset. Default: *True*.

evaluate(*results*, *logger=None*)

Evaluate with different metrics.

参数 **results** (*list[tuple]*) –The output of forward_test() of the model.

返回 Evaluation results dict.

返回类型 dict

load_annotations()

Load annoations for the dataset.

返回 Returned dict for LQ and GT pairs.

返回类型 dict

class mmedit.datasets.**SRLmdbDataset**(*lq_folder*, *gt_folder*, *pipeline*, *scale*, *test_mode=False*)

General paired image lmdb dataset for image restoration.

The dataset loads lq (Low Quality) and gt (Ground-Truth) image pairs, applies specified transforms and finally returns a dict containing paired data and other information.

This is the "lmdb mode". In order to speed up IO, you are recommended to use lmdb. First, you need to make lmdb files. Suppose the lmdb files are path_to_lq/lq.lmdb and path_to_gt/gt.lmdb, then you can just set:

```
lq_folder = path_to_lq/lq.lmdb
gt_folder = path_to_gt/gt.lmdb
```

Contents of lmdb. Taking the lq.lmdb for example, the file structure is:

```
lq.lmdb
├── data.mdb
├── lock.mdb
├── meta_info.txt
```

The data.mdb and lock.mdb are standard lmdb files and you can refer to https://lmdb.readthedocs.io/en/release/ for more details.

The meta_info.txt is a specified txt file to record the meta information of our datasets. It will be automatically created when preparing datasets by our provided dataset tools. Each line in the txt file records

1. image name (with extension);
2. image shape;
3. compression level, separated by a white space.

For example, the meta information of the lq.lmdb is: *baboon.png (120,125,3) 1*, which means: 1) image name (with extension): baboon.png; 2) image shape: (120,125,3); and 3) compression level: 1

We use the image name without extension as the lmdb key. Note that we use the same key for the corresponding lq and gt images.

参数

- **lq_folder** (str | `Path`) –Path to a lq lmdb file.
- **gt_folder** (str | `Path`) –Path to a gt lmdb file.
- **pipeline** (*`list[dict | callable]`*) –A sequence of data transformations.
- **scale** (*`int`*) –Upsampling scale ratio.
- **test_mode** (*`bool`*) –Store *True* when building test dataset. Default: *False*.

load_annotations()

Load annoations for SR dataset.

It loads the LQ and GT image path from the `meta_info.txt` in the LMDB files.

返回 Returned dict for LQ and GT pairs.

返回类型 dict

class mmedit.datasets.**SRREDSDataset**(*lq_folder*, *gt_folder*, *ann_file*, *num_input_frames*, *pipeline*, *scale*, *val_partition='official'*, *test_mode=False*)

REDS dataset for video super resolution.

The dataset loads several LQ (Low-Quality) frames and a center GT (Ground-Truth) frame. Then it applies specified transforms and finally returns a dict containing paired data and other information.

It reads REDS keys from the txt file. Each line contains: 1. image name; 2, image shape, separated by a white space. Examples:

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

参数

- **lq_folder** (str | `Path`) –Path to a lq folder.
- **gt_folder** (str | `Path`) –Path to a gt folder.

- **ann_file** (str | Path) –Path to the annotation file.
- **num_input_frames** (*int*) –Window size for input frames.
- **pipeline** (*list[dict | callable]*) –A sequence of data transformations.
- **scale** (*int*) –Upsampling scale ratio.
- **val_partition** (*str*) –Validation partition mode. Choices ['official' or
- **Default** (*'REDS4'*]) – 'official' .
- **test_mode** (*bool*) –Store *True* when building test dataset. Default: *False*.

load_annotations()

Load annoations for REDS dataset.

返回 Returned dict for LQ and GT pairs.

返回类型 dict

class mmedit.datasets.**SRREDSMultipleGTDataset**(*lq_folder*, *gt_folder*, *num_input_frames*, *pipeline*, *scale*, *val_partition='official'*, *test_mode=False*)

REDS dataset for video super resolution for recurrent networks.

The dataset loads several LQ (Low-Quality) frames and GT (Ground-Truth) frames. Then it applies specified transforms and finally returns a dict containing paired data and other information.

参数

- **lq_folder** (str | Path) –Path to a lq folder.
- **gt_folder** (str | Path) –Path to a gt folder.
- **num_input_frames** (*int*) –Number of input frames.
- **pipeline** (*list[dict | callable]*) –A sequence of data transformations.
- **scale** (*int*) –Upsampling scale ratio.
- **val_partition** (*str*) –Validation partition mode. Choices ['official' or
- **Default** (*'REDS4'*]) – 'official' .
- **test_mode** (*bool*) –Store *True* when building test dataset. Default: *False*.

load_annotations()

Load annoations for REDS dataset.

返回 Returned dict for LQ and GT pairs.

返回类型 dict

class mmedit.datasets.**SRTestMultipleGTDataset**(*lq_folder*, *gt_folder*, *pipeline*, *scale*, *test_mode=True*)

Test dataset for video super resolution for recurrent networks.

It assumes all video sequences under the root directory is used for test.

The dataset loads several LQ (Low-Quality) frames and GT (Ground-Truth) frames. Then it applies specified transforms and finally returns a dict containing paired data and other information.

参数

- **lq_folder** (str | Path) –Path to a lq folder.
- **gt_folder** (str | Path) –Path to a gt folder.
- **pipeline** (*list[dict | callable]*) –A sequence of data transformations.
- **scale** (*int*) –Upsampling scale ratio.
- **test_mode** (*bool*) –Store *True* when building test dataset. Default: *True*.

load_annotations()

Load annoations for the test dataset.

返回 Returned dict for LQ and GT pairs.

返回类型 dict

class mmedit.datasets.**SRVid4Dataset**(*lq_folder*, *gt_folder*, *ann_file*, *num_input_frames*, *pipeline*, *scale*, *filename_tmpl='{:08d}'*, *metric_average_mode='clip'*, *test_mode=False*)

Vid4 dataset for video super resolution.

The dataset loads several LQ (Low-Quality) frames and a center GT (Ground-Truth) frame. Then it applies specified transforms and finally returns a dict containing paired data and other information.

It reads Vid4 keys from the txt file. Each line contains:

1. folder name;
2. number of frames in this clip (in the same folder);
3. image shape, separated by a white space.

Examples:

```
calendar 40 (320,480,3)
city 34 (320,480,3)
```

参数

- **lq_folder** (str | Path) –Path to a lq folder.
- **gt_folder** (str | Path) –Path to a gt folder.
- **ann_file** (str | Path) –Path to the annotation file.
- **num_input_frames** (*int*) –Window size for input frames.

- **pipeline** (*list[dict | callable]*) –A sequence of data transformations.
- **scale** (*int*) –Upsampling scale ratio.
- **filename_tmpl** (*str*) –Template for each filename. Note that the template excludes the file extension. Default: '{:08d}'.
- **metric_average_mode** (*str*) –The way to compute the average metric. If 'clip', we first compute an average value for each clip, and then average the values from different clips. If 'all', we compute the average of all frames. Default: 'clip'.
- **test_mode** (*bool*) –Store *True* when building test dataset. Default: *False*.

evaluate(*results*, *logger=None*)

Evaluate with different metrics. :param results: The output of forward_test() of the model. :type results: list[tuple]

返回 Evaluation results dict.

返回类型 dict

load_annotations()

Load annoations for Vid4 dataset. :returns: Returned dict for LQ and GT pairs. :rtype: dict

class mmedit.datasets.**SRVimeo90KDataset**(*lq_folder*, *gt_folder*, *ann_file*, *num_input_frames*, *pipeline*, *scale*, *test_mode=False*)

Vimeo90K dataset for video super resolution.

The dataset loads several LQ (Low-Quality) frames and a center GT (Ground-Truth) frame. Then it applies specified transforms and finally returns a dict containing paired data and other information.

It reads Vimeo90K keys from the txt file. Each line contains: 1. image name; 2, image shape, separated by a white space. Examples:

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

参数

- **lq_folder** (str | Path) –Path to a lq folder.
- **gt_folder** (str | Path) –Path to a gt folder.
- **ann_file** (str | Path) –Path to the annotation file.
- **num_input_frames** (*int*) –Window size for input frames.
- **pipeline** (*list[dict | callable]*) –A sequence of data transformations.
- **scale** (*int*) –Upsampling scale ratio.
- **test_mode** (*bool*) –Store *True* when building test dataset. Default: *False*.

load_annotations()

Load annoations for VimeoK dataset.

返回 Returned dict for LQ and GT pairs.

返回类型 dict

class mmedit.datasets.**SRVimeo90KMultipleGTDataset**(*lq_folder*, *gt_folder*, *ann_file*, *pipeline*, *scale*, *num_input_frames=7*, *test_mode=False*)

Vimeo90K dataset for video super resolution for recurrent networks.

The dataset loads several LQ (Low-Quality) frames and GT (Ground-Truth) frames. Then it applies specified transforms and finally returns a dict containing paired data and other information.

It reads Vimeo90K keys from the txt file. Each line contains:

1. video frame folder
2. image shape

Examples:

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

参数

- **lq_folder** (str | Path) –Path to a lq folder.
- **gt_folder** (str | Path) –Path to a gt folder.
- **ann_file** (str | Path) –Path to the annotation file.
- **pipeline** (*list[dict | callable]*) –A sequence of data transformations.
- **scale** –Upsampling scale ratio.

load_annotations()

Load annoations for Vimeo-90K dataset.

返回 Returned dict for LQ and GT pairs.

返回类型 dict

mmedit.datasets.**build_dataloader**(*dataset*, *samples_per_gpu*, *workers_per_gpu*, *num_gpus=1*, *dist=True*, *shuffle=True*, *seed=None*, *drop_last=False*, *pin_memory=True*, *persistent_workers=True*, ***kwargs*)

Build PyTorch DataLoader.

In distributed training, each GPU/process has a dataloader. In non-distributed training, there is only one dataloader for all GPUs.

参数

- **dataset** (`Dataset`) –A PyTorch dataset.
- **samples_per_gpu** (*int*) –Number of samples on each GPU, i.e., batch size of each GPU.
- **workers_per_gpu** (*int*) –How many subprocesses to use for data loading for each GPU.
- **num_gpus** (*int*) –Number of GPUs. Only used in non-distributed training. Default: 1.
- **dist** (*bool*) –Distributed training/test or not. Default: True.
- **shuffle** (*bool*) –Whether to shuffle the data at every epoch. Default: True.
- **seed** (*int | None*) –Seed to be used. Default: None.
- **drop_last** (*bool*) –Whether to drop the last incomplete batch in epoch. Default: False
- **pin_memory** (*bool*) –Whether to use pin_memory in DataLoader. Default: True
- **persistent_workers** (*bool*) –If True, the data loader will not shutdown the worker processes after a dataset has been consumed once. This allows to maintain the workers Dataset instances alive. The argument also has effect in PyTorch>=1.7.0. Default: True
- **kwargs** (*dict, optional*) –Any keyword argument to be used to initialize DataLoader.

返回 A PyTorch dataloader.

返回类型 DataLoader

`mmedit.datasets.`**build_dataset**(*cfg*, *default_args=None*)

Build a dataset from config dict.

It supports a variety of dataset config. If `cfg` is a Sequential (list or dict), it will be a concatenated dataset of the datasets specified by the Sequential. If it is a `RepeatDataset`, then it will repeat the dataset `cfg['dataset']` for `cfg['times']` times. If the `ann_file` of the dataset is a Sequential, then it will build a concatenated dataset with the same dataset type but different `ann_file`.

参数

- **cfg** (*dict*) –Config dict. It should at least contain the key "type".
- **default_args** (*dict, optional*) –Default initialization arguments. Default: None.

返回 The constructed dataset.

返回类型 Dataset

21.2 pipelines

class mmedit.datasets.pipelines.**BinarizeImage**(*keys*, *binary_thr*, *to_int=False*)

Binarize image.

参数

- **keys** (*Sequence[str]*) –The images to be binarized.
- **binary_thr** (*float*) –Threshold for binarization.
- **to_int** (*bool*) –If True, return image as int32, otherwise return image as float32.

class mmedit.datasets.pipelines.**Collect**(*keys*, *meta_keys=None*)

Collect data from the loader relevant to the specific task.

This is usually the last stage of the data loader pipeline. Typically keys is set to some subset of "img", "gt_labels". .

The "img_meta" item is always populated. The contents of the "meta" dictionary depends on "meta_keys".

参数

- **keys** (*Sequence[str]*) –Required keys to be collected.
- **meta_keys** (*Sequence[str]*) –Required keys to be collected to "meta". Default: None.

class mmedit.datasets.pipelines.**Compose**(*transforms*)

Compose a data pipeline with a sequence of transforms.

参数 **transforms** (*list[dict | callable]*) –Either config dicts of transforms or transform objects.

class mmedit.datasets.pipelines.**CompositeFg**(*fg_dirs*, *alpha_dirs*, *interpolation='nearest'*, *io_backend='disk'*, ***kwargs*)

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.

class mmedit.datasets.pipelines.**CopyValues**(*src_keys*, *dst_keys*)

Copy the value of a source key to a destination key.

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".

参数

- **src_keys** (*list[str]*) –The source keys.
- **dst_keys** (*list[str]*) –The destination keys.

class mmedit.datasets.pipelines.**Crop**(*keys*, *crop_size*, *random_crop=True*)

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.

class mmedit.datasets.pipelines.**CropAroundCenter**(*crop_size*)

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.

class mmedit.datasets.pipelines.**CropAroundFg**(*keys*, *bd_ratio_range=(0.1, 0.4)*, *test_mode=False*)

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.

class mmedit.datasets.pipelines.**CropAroundUnknown** (*keys*, *crop_sizes*, *unknown_source='alpha'*, *interpolations='bilinear'*)

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 'tirmap'. 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'.

class mmedit.datasets.pipelines.**CropLike** (*target_key*, *reference_key=None*)

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.

class mmedit.datasets.pipelines.**CropSequence** (*keys*, *crop_size*, *random_crop=True*)

Crop a sequence to specific size for training.

The main difference to 'Crop' is that the region to be cropped is the same for every images in the sequence.

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.

class mmedit.datasets.pipelines.**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.

class mmedit.datasets.pipelines.**FixedCrop**(*keys*, *crop_size*, *crop_pos=None*)

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.

class mmedit.datasets.pipelines.**Flip**(*keys*, *flip_ratio=0.5*, *direction='horizontal'*)

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.

参数

- **keys** (*list[str]*) –The images to be flipped.
- **flip_ratio** (*float*) –The propability to flip the images.

- **direction** (*str*) –Flip images horizontally or vertically. Options are "horizontal" | "vertical". Default: "horizontal".

class mmedit.datasets.pipelines.**FormatTrimap**(*to_onehot=False*)

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 "to_onehot".

参数 **to_onehot** (*bool*) –whether convert trimap to one-hot tensor. Default: `False`.

class mmedit.datasets.pipelines.**GenerateCoordinateAndCell**(*sample_quantity=None*, *scale=None*, *target_size=None*)

Generate coordinate and cell.

Generate coordinate from the desired size of SR image.

Train or val:

1. Generate coordinate from GT.
2. **Reshape GT image to (HgWg, 3) and transpose to (3, HgWg).** where *Hg* and *Wg* represent the height and width of GT.

Test: Generate coordinate from LQ and scale or target_size.

Then generate cell from coordinate.

参数

- **sample_quantity** (*int*) –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.

The priority of getting 'size of target image' is: 1, results['gt'].shape[-2:] 2, results['lq'].shape[-2:] * scale 3, target_size

class mmedit.datasets.pipelines.**GenerateFrameIndices**(*interval_list*, *frames_per_clip=99*)

Generate frame index for REDS datasets. It also performs temporal augmention with random interval.

Required keys: lq_path, gt_path, key, num_input_frames Added or modified keys: lq_path, gt_path, 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.

class mmedit.datasets.pipelines.**GenerateFrameIndiceswithPadding**(*padding*, *filename_tmpl='{:08d}'*)

Generate frame index with padding for REDS dataset and Vid4 dataset during testing.

Required keys: lq_path, gt_path, key, num_input_frames, max_frame_num Added or modified keys: lq_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]

class mmedit.datasets.pipelines.**GenerateHeatmap**(*keypoint*, *ori_size*, *target_size*, *sigma=1.0*)

Generate heatmap from keypoint.

参数

- **keypoint** (*str*) –Key of keypoint 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

class mmedit.datasets.pipelines.**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)]*)

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)].

class mmedit.datasets.pipelines.**GenerateSegmentIndices**(*interval_list*, *start_idx=0*, *filename_tmpl='{:08d}.png'*)

Generate frame indices for a segment. It also performs temporal augmention with random interval.

Required keys: lq_path, gt_path, key, num_input_frames, sequence_length Added or modified keys: lq_path, gt_path, 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'.

class mmedit.datasets.pipelines.**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)]*)

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)].

class mmedit.datasets.pipelines.**GenerateTrimap**(*kernel_size*, *iterations=1*, *random=True*)

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.

class mmedit.datasets.pipelines.**GenerateTrimapWithDistTransform**(*dist_thr=20*, *random=True*)

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.

class mmedit.datasets.pipelines.**GetMaskedImage**(*img_name='gt_img', mask_name='mask'*)

Get masked image.

参数

- **img_name** (*str*) –Key for clean image.
- **mask_name** (*str*) –Key for mask image. The mask shape should be (h, w, 1) while '1' indicate holes and '0' indicate valid regions.

class mmedit.datasets.pipelines.**GetSpatialDiscountMask**(*gamma=0.99, beta=1.5*)

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

class mmedit.datasets.pipelines.**ImageToTensor**(*keys*, *to_float32=True*)

Convert image type to *torch.Tensor* type.

参数

- **keys** (*Sequence[str]*) –Required keys to be converted.
- **to_float32** (*bool*) –Whether convert numpy image array to np.float32 before converted to tensor. Default: True.

class mmedit.datasets.pipelines.**LoadImageFromFile**(*io_backend='disk'*, *key='gt'*, *flag='color'*, *channel_order='bgr'*, *convert_to=None*, *save_original_img=False*, *use_cache=False*, *backend=None*, ***kwargs*)

Load image from file.

参数

- **io_backend** (*str*) –io backend where images are store. Default: 'disk'.
- **key** (*str*) –Keys in results to find corresponding path. Default: 'gt'.
- **flag** (*str*) –Loading flag for images. Default: 'color'.
- **channel_order** (*str*) –Order of channel, candidates are 'bgr' and 'rgb'. Default: 'bgr'.
- **convert_to** (*str | None*) –The color space of the output image. If None, no conversion is conducted. Default: None.
- **save_original_img** (*bool*) –If True, maintain a copy of the image in *results* dict with name of *f' ori_{key}'*. Default: False.
- **use_cache** (*bool*) –If True, load all images at once. Default: False.
- **backend** (*str*) –The image loading backend type. Options are *cv2*, *pillow*, and 'turbojpeg'. Default: None.
- **kwargs** (*dict*) –Args for file client.

class mmedit.datasets.pipelines.**LoadImageFromFileList**(*io_backend='disk'*, *key='gt'*, *flag='color'*, *channel_order='bgr'*, *convert_to=None*, *save_original_img=False*, *use_cache=False*, *backend=None*, ***kwargs*)

Load image from file list.

It accepts a list of path and read each frame from each path. A list of frames will be returned.

参数

- **io_backend** (*str*) –io backend where images are store. Default: ‘disk’.
- **key** (*str*) –Keys in results to find corresponding path. Default: ‘gt’.
- **flag** (*str*) –Loading flag for images. Default: ‘color’.
- **channel_order** (*str*) –Order of channel, candidates are ‘bgr’ and ‘rgb’. Default: ‘bgr’.
- **convert_to** (*str | None*) –The color space of the output image. If None, no conversion is conducted. Default: None.
- **save_original_img** (*bool*) –If True, maintain a copy of the image in *results* dict with name of *f’ ori_{key}’*. Default: False.
- **kwargs** (*dict*) –Args for file client.

class mmedit.datasets.pipelines.**LoadMask**(*mask_mode='bbox', mask_config=None*)

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='disk',
    flag='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']. * 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.

class mmedit.datasets.pipelines.**LoadPairedImageFromFile**(*io_backend='disk', key='gt', flag='color', channel_order='bgr', convert_to=None, save_original_img=False, use_cache=False, backend=None, **kwargs*)

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_a" , "img_b" , "img_a_path" , "img_b_path" , "img_a_ori_shape" , "img_b_ori_shape" , "ori_img_a" and "ori_img_b" .

参数

- **io_backend** (*str*) –io backend where images are store. Default: 'disk' .
- **key** (*str*) –Keys in results to find corresponding path. Default: 'gt' .

- **flag** (*str*) –Loading flag for images. Default: 'color'.
- **channel_order** (*str*) –Order of channel, candidates are 'bgr' and 'rgb'. Default: 'bgr'.
- **save_original_img** (*bool*) –If True, maintain a copy of the image in *results* dict with name of *f'ori_{key}'*. Default: False.
- **kwargs** (*dict*) –Args for file client.

class mmedit.datasets.pipelines.**MATLABLikeResize**(*keys*, *scale=None*, *output_shape=None*, *kernel='bicubic'*, *kernel_width=4.0*)

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.

class mmedit.datasets.pipelines.**MergeFgAndBg**

Composite foreground image and background image with alpha.

Required keys are "alpha", "fg" and "bg", added key is "merged".

class mmedit.datasets.pipelines.**MirrorSequence**(*keys*)

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).

参数 **keys** (*list[str]*) –The frame lists to be extended.

class mmedit.datasets.pipelines.**ModCrop**

Mod crop gt images, used during testing.

Required keys are "scale" and "gt", added or modified keys are "gt".

class mmedit.datasets.pipelines.**Normalize**(*keys*, *mean*, *std*, *to_rgb=False*, *save_original=False*)

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.

class mmedit.datasets.pipelines.**Pad**(*keys*, *ds_factor=32*, ***kwargs*)

Pad the images to align with network downsample factor for testing.

See *Reshape* for more explanation. *numpy.pad* is used for the pad operation. Required keys are the keys in attribute "keys", added or modified keys are "test_trans" and the keys in attribute "keys". 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** (*list[str]*) –The images to be padded.
- **ds_factor** (*int*) –Downsample factor of the network. The height and weight will be padded to a multiple of ds_factor. Default: 32.
- **kwargs** (*option*) –any keyword arguments to be passed to *numpy.pad*.

class mmedit.datasets.pipelines.**PairedRandomCrop**(*gt_patch_size*)

Paried random crop.

It crops a pair of lq and gt images with corresponding locations. It also supports accepting lq list and gt list. Required keys are "scale", "lq", and "gt", added or modified keys are "lq" and "gt".

参数 **gt_patch_size** (*int*) –cropped gt patch size.

class mmedit.datasets.pipelines.**PerturbBg**(*gamma_ratio=0.6*)

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.

class mmedit.datasets.pipelines.**Quantize**(*keys*)

Quantize and clip the image to [0, 1].

It is assumed that the the input has range [0, 1].

Modified keys are the attributes specified in "keys".

参数 **keys** (*list[str]*) –The keys whose values are clipped.

class mmedit.datasets.pipelines.**RandomAffine**(*keys*, *degrees*, *translate=None*, *scale=None*, *shear=None*, *flip_ratio=None*)

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.

class mmedit.datasets.pipelines.**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.

class mmedit.datasets.pipelines.**RandomDownSampling**(*scale_min=1.0*, *scale_max=4.0*, *patch_size=None*, *interpolation='bicubic'*, *backend='pillow'*)

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".
- **will be picked in the range of [scale_min** (*Scale*) –
- **scale_max)** –

class mmedit.datasets.pipelines.**RandomJPEGCompression**(*params*, *keys*)

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.

class mmedit.datasets.pipelines.**RandomJitter**(*hue_range=40*)

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.

class mmedit.datasets.pipelines.**RandomLoadResizeBg**(*bg_dir*, *io_backend='disk'*, *flag='color'*, *channel_order='bgr'*, ***kwargs*)

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.
- **io_backend** (*str*) –io backend where images are store. Default: 'disk'.
- **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.

class mmedit.datasets.pipelines.**RandomMaskDilation**(*keys*, *binary_thr=0.0*, *kernel_min=9*, *kernel_max=49*)

Randomly dilate binary masks.

参数

- **keys** (*Sequence[str]*) –The images to be resized.
- **get_binary** (*bool*) –If True, according to binary_thr, reset final output as binary mask. Otherwise, return masks directly.
- **binary_thr** (*float*) –Threshold for obtaining binary mask.
- **kernel_min** (*int*) –Min size of dilation kernel.
- **kernel_max** (*int*) –Max size of dilation kernel.

class mmedit.datasets.pipelines.**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.

class mmedit.datasets.pipelines.**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.

class mmedit.datasets.pipelines.**RandomTransposeHW**(*keys*, *transpose_ratio=0.5*)

Randomly transpose images in H and W dimensions with a probability.

(TransposeHW = horizontal flip + anti-clockwise rotatation 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 propability to transpose the images.

class mmedit.datasets.pipelines.**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.

class mmedit.datasets.pipelines.**RescaleToZeroOne**(*keys*)

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.

class mmedit.datasets.pipelines.**Resize**(*keys*, *scale=None*, *keep_ratio=False*, *size_factor=None*, *max_size=None*, *interpolation='bilinear'*, *backend=None*, *output_keys=None*)

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".

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** (*list[str]*) –The images 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 togher 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 togher 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.

class mmedit.datasets.pipelines.**TemporalReverse**(*keys*, *reverse_ratio=0.5*)

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 propability to reverse the frame lists. Default: 0.5.

class mmedit.datasets.pipelines.**ToTensor**(*keys*)

Convert some values in results dict to *torch.Tensor* type in data loader pipeline.

参数 **keys** (*Sequence[str]*) –Required keys to be converted.

class mmedit.datasets.pipelines.**TransformTrimap**

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.

class mmedit.datasets.pipelines.**UnsharpMasking**(*kernel_size*, *sigma*, *weight*, *threshold*, *keys*)

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".

CHAPTER 22

mmedit.models

22.1 models

class mmedit.models.**BaseMattor**(*backbone*, *refiner=None*, *train_cfg=None*, *test_cfg=None*, *pretrained=None*)

Base class for matting model.

A matting model must contain a backbone which produces *alpha*, a dense prediction with the same height and width of input image. In some cases, the model will has a refiner which refines the prediction of the backbone.

The subclasses should overwrite the function `forward_train` and `forward_test` which define the output of the model and maybe the connection between the backbone and the refiner.

参数

- **backbone** (*dict*) –Config of backbone.
- **refiner** (*dict*) –Config of refiner.
- **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.
- **pretrained** (*str*) –Path of pretrained model.

evaluate(*pred_alpha*, *meta*)

Evaluate predicted alpha matte.

The evaluation metrics are determined by `self.test_cfg.metrics`.

参数

- **pred_alpha** (*np.ndarray*) –The predicted alpha matte of shape (H, W).
- **meta** (*list[dict]*) –Meta data about the current data batch. Currently only batch_size 1 is supported. Required keys in the meta dict are `ori_alpha` and `ori_trimap`.

返回 The evaluation result.

返回类型 dict

forward(*merged*, *trimap*, *meta*, *alpha=None*, *test_mode=False*, ***kwargs*)

Defines the computation performed at every call.

参数

- **merged** (*Tensor*) –Image to predict alpha matte.
- **trimap** (*Tensor*) –Trimap of the input image.
- **meta** (*list[dict]*) –Meta data about the current data batch. Defaults to None.
- **alpha** (*Tensor, optional*) –Ground-truth alpha matte. Defaults to None.
- **test_mode** (*bool, optional*) –Whether in test mode. If `True`, it will call `forward_test` of the model. Otherwise, it will call `forward_train` of the model. Defaults to False.

返回 Return the output of `self.forward_test` if `test_mode` are set to `True`. Otherwise return the output of `self.forward_train`.

返回类型 dict

abstract forward_test(*merged*, *trimap*, *meta*, ***kwargs*)

Defines the computation performed at every test call.

abstract forward_train(*merged*, *trimap*, *alpha*, ***kwargs*)

Defines the computation performed at every training call.

参数

- **merged** (*Tensor*) –Image to predict alpha matte.
- **trimap** (*Tensor*) –Trimap of the input image.
- **alpha** (*Tensor*) –Ground-truth alpha matte.

freeze_backbone()

Freeze the backbone and only train the refiner.

init_weights(*pretrained=None*)

Initialize the model network weights.

参数 **pretrained** (*str, optional*) –Path to the pretrained weight. Defaults to None.

restore_shape (*pred_alpha*, *meta*)

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** (*np.ndarray*) –The predicted alpha.
- **meta** (*list[dict]*) –Meta data about the current data batch. Currently only batch_size 1 is supported.

返回 The reshaped predicted alpha.

返回类型 np.ndarray

save_image (*pred_alpha*, *meta*, *save_path*, *iteration*)

Save predicted alpha to file.

参数

- **pred_alpha** (*np.ndarray*) –The predicted alpha matte of shape (H, W).
- **meta** (*list[dict]*) –Meta data about the current data batch. Currently only batch_size 1 is supported. Required keys in the meta dict are `merged_path`.
- **save_path** (*str*) –The directory to save predicted alpha matte.
- **iteration** (*int | None*) –If given as None, the saved alpha matte will have the same file name with `merged_path` in meta dict. If given as an int, the saved alpha matte would named with postfix `_{iteration}.png`.

train_step (*data_batch*, *optimizer*)

Defines the computation and network update at every training call.

参数

- **data_batch** (*torch.Tensor*) –Batch of data as input.
- **optimizer** (*torch.optim.Optimizer*) –Optimizer of the model.

返回 Output of `train_step` containing the logging variables of the current data batch.

返回类型 dict

property with_refiner

Whether the matting model has a refiner.

class mmedit.models.**BaseModel**

Base model.

All models should subclass it. All subclass should overwrite:

`init_weights`, supporting to initialize models.

`forward_train`, supporting to forward when training.

`forward_test`, supporting to forward when testing.

`train_step`, supporting to train one step when training.

forward(*imgs*, *labels*, *test_mode*, ***kwargs*)

Forward function for base model.

参数

- **imgs** (*Tensor*) –Input image(s).
- **labels** (*Tensor*) –Ground-truth label(s).
- **test_mode** (*bool*) –Whether in test mode.
- **kwargs** (*dict*) –Other arguments.

返回 Forward results.

返回类型 Tensor

abstract forward_test(*imgs*)

Abstract method for testing forward.

All subclass should overwrite it.

abstract forward_train(*imgs*, *labels*)

Abstract method for training forward.

All subclass should overwrite it.

abstract init_weights()

Abstract method for initializing weight.

All subclass should overwrite it.

parse_losses(*losses*)

Parse losses dict for different loss variants.

参数 **losses** (*dict*) –Loss dict.

返回 Sum of the total loss. log_vars (dict): loss dict for different variants.

返回类型 loss (float)

abstract train_step(*data_batch*, *optimizer*)

Abstract method for one training step.

All subclass should overwrite it.

val_step(*data_batch*, ***kwargs*)

Abstract method for one validation step.

All subclass should overwrite it.

class mmedit.models.**BasicRestorer**(*generator*, *pixel_loss*, *train_cfg=None*, *test_cfg=None*, *pretrained=None*)

Basic model for image restoration.

It must contain a generator that takes an image as inputs and outputs a restored image. It also has a pixel-wise loss for training.

The subclasses should overwrite the function *forward_train*, *forward_test* and *train_step*.

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.
- **pretrained** (*str*) –Path for pretrained model. Default: None.

evaluate(*output*, *gt*)

Evaluation function.

参数

- **output** (*Tensor*) –Model output with shape (n, c, h, w).
- **gt** (*Tensor*) –GT Tensor with shape (n, c, h, w).

返回 Evaluation results.

返回类型 dict

forward(*lq*, *gt=None*, *test_mode=False*, ***kwargs*)

Forward function.

参数

- **lq** (*Tensor*) –Input lq images.
- **gt** (*Tensor*) –Ground-truth image. Default: None.
- **test_mode** (*bool*) –Whether in test mode or not. Default: False.
- **kwargs** (*dict*) –Other arguments.

forward_dummy(*img*)

Used for computing network FLOPs.

参数 **img** (*Tensor*) –Input image.

返回 Output image.

返回类型 Tensor

forward_test (*lq*, *gt=None*, *meta=None*, *save_image=False*, *save_path=None*, *iteration=None*)

Testing forward function.

参数

- **lq** (*Tensor*) –LQ Tensor with shape (n, c, h, w).
- **gt** (*Tensor*) –GT Tensor with shape (n, c, h, w). Default: None.
- **save_image** (*bool*) –Whether to save image. Default: False.
- **save_path** (*str*) –Path to save image. Default: None.
- **iteration** (*int*) –Iteration for the saving image name. Default: None.

返回 Output results.

返回类型 dict

forward_train (*lq*, *gt*)

Training forward function.

参数

- **lq** (*Tensor*) –LQ Tensor with shape (n, c, h, w).
- **gt** (*Tensor*) –GT Tensor with shape (n, c, h, w).

返回 Output tensor.

返回类型 Tensor

init_weights (*pretrained=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.

train_step (*data_batch*, *optimizer*)

Train step.

参数

- **data_batch** (*dict*) –A batch of data.
- **optimizer** (*obj*) –Optimizer.

返回 Returned output.

返回类型 dict

val_step (*data_batch*, ***kwargs*)

Validation step.

参数

- **data_batch** (*dict*) –A batch of data.

- **kwargs** (*dict*) –Other arguments for `val_step`.

返回 Returned output.

返回类型 dict

class `mmedit.models.`**CycleGAN**(*generator*, *discriminator*, *gan_loss*, *cycle_loss*, *id_loss=None*, *train_cfg=None*, *test_cfg=None*, *pretrained=None*)

CycleGAN model for unpaired image-to-image translation.

Ref: Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks

参数

- **generator** (*dict*) –Config for the generator.
- **discriminator** (*dict*) –Config for the discriminator.
- **gan_loss** (*dict*) –Config for the gan loss.
- **cycle_loss** (*dict*) –Config for the cycle-consistency loss.
- **id_loss** (*dict*) –Config for the identity 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 generator update. *disc_init_steps*: how many discriminator updates at the start of the training. These two keys are useful when training with WGAN. *direction*: image-to-image translation direction (the model training direction): a2b | b2a. *buffer_size*: GAN image buffer size.
- **test_cfg** (*dict*) –Config for testing. Default: None. You may change the testing of gan by setting: *direction*: image-to-image translation direction (the model training direction): a2b | b2a. *show_input*: whether to show input real images. *test_direction*: direction in the test mode (the model testing direction). CycleGAN has two generators. It decides whether to perform forward or backward translation with respect to *direction* during testing: a2b | b2a.
- **pretrained** (*str*) –Path for pretrained model. Default: None.

backward_discriminators(*outputs*)

Backward function for the discriminators.

参数 **outputs** (*dict*) –Dict of forward results.

返回 Loss dict.

返回类型 dict

backward_generators(*outputs*)

Backward function for the generators.

参数 **outputs** (*dict*) –Dict of forward results.

返回 Loss dict.

返回类型 dict

forward(*img_a*, *img_b*, *meta*, *test_mode=False*, ***kwargs*)

Forward function.

参数

- **img_a** (*Tensor*) –Input image from domain A.
- **img_b** (*Tensor*) –Input image from domain B.
- **meta** (*list[dict]*) –Input meta data.
- **test_mode** (*bool*) –Whether in test mode or not. Default: False.
- **kwargs** (*dict*) –Other arguments.

forward_dummy(*img*)

Used for computing network FLOPs.

参数 **img** (*Tensor*) –Dummy input used to compute FLOPs.

返回 Dummy output produced by forwarding the dummy input.

返回类型 Tensor

forward_test(*img_a*, *img_b*, *meta*, *save_image=False*, *save_path=None*, *iteration=None*)

Forward function for testing.

参数

- **img_a** (*Tensor*) –Input image from domain A.
- **img_b** (*Tensor*) –Input image from domain B.
- **meta** (*list[dict]*) –Input meta data.
- **save_image** (*bool, optional*) –If True, results will be saved as images. Default: False.
- **save_path** (*str, optional*) –If given a valid str path, the results will be saved in this path. Default: None.
- **iteration** (*int, optional*) –Iteration number. Default: None.

返回 Dict of forward and evaluation results for testing.

返回类型 dict

forward_train(*img_a*, *img_b*, *meta*)

Forward function for training.

参数

- **img_a** (*Tensor*) –Input image from domain A.
- **img_b** (*Tensor*) –Input image from domain B.

- **meta** (*list[dict]*) –Input meta data.

返回 Dict of forward results for training.

返回类型 dict

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

init_weights (*pretrained=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.

setup (*img_a*, *img_b*, *meta*)

Perform necessary pre-processing steps.

参数

- **img_a** (*Tensor*) –Input image from domain A.
- **img_b** (*Tensor*) –Input image from domain B.
- **meta** (*list[dict]*) –Input meta data.

返回 The real images from domain A/B, and the image path as the metadata.

返回类型 Tensor, Tensor, list[str]

train_step (*data_batch*, *optimizer*)

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.

返回 Dict of loss, information for logger, the number of samples and results for visualization.

返回类型 dict

val_step (*data_batch*, ***kwargs*)

Validation step function.

参数

- **data_batch** (*dict*) –Dict of the input data batch.
- **kwargs** (*dict*) –Other arguments.

返回 Dict of evaluation results for validation.

返回类型 dict

class mmedit.models.**DIM**(*backbone*, *refiner=None*, *train_cfg=None*, *test_cfg=None*, *pretrained=None*, *loss_alpha=None*, *loss_comp=None*, *loss_refine=None*)

Deep Image Matting model.

https://arxiv.org/abs/1703.03872

注解: 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

- **backbone** (*dict*) –Config of backbone.
- **refiner** (*dict*) –Config of refiner.
- **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.
- **pretrained** (*str*) –Path of pretrained model.
- **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.

forward_test(*merged*, *trimap*, *meta*, *save_image=False*, *save_path=None*, *iteration=None*)

Defines the computation performed at every test call.

参数

- **merged** (*Tensor*) –Image to predict alpha matte.
- **trimap** (*Tensor*) –Trimap of the input image.

- **meta** (*list[dict]*) –Meta data about the current data batch. Currently only batch_size 1 is supported. It may contain information needed to calculate metrics (`ori_alpha` and `ori_trimap`) or save predicted alpha matte (`merged_path`).
- **save_image** (*bool, optional*) –Whether save predicted alpha matte. Defaults to False.
- **save_path** (*str, optional*) –The directory to save predicted alpha matte. Defaults to None.
- **iteration** (*int, optional*) –If given as None, the saved alpha matte will have the same file name with `merged_path` in meta dict. If given as an int, the saved alpha matte would named with postfix `_{iteration}.png`. Defaults to None.

返回 Contains the predicted alpha and evaluation result.

返回类型 dict

forward_train(*merged*, *trimap*, *meta*, *alpha*, *ori_merged*, *fg*, *bg*)

Defines the computation performed at every training call.

参数

- **merged** (*Tensor*) –of shape (N, C, H, W) encoding input images. Typically these should be mean centered and std scaled.
- **trimap** (*Tensor*) –of shape (N, 1, H, W). Tensor of trimap read by opencv.
- **meta** (*list[dict]*) –Meta data about the current data batch.
- **alpha** (*Tensor*) –of shape (N, 1, H, W). Tensor of alpha read by opencv.
- **ori_merged** (*Tensor*) –of shape (N, C, H, W). Tensor of origin merged image read by opencv (not normalized).
- **fg** (*Tensor*) –of shape (N, C, H, W). Tensor of fg read by opencv.
- **bg** (*Tensor*) –of shape (N, C, H, W). Tensor of bg read by opencv.

返回 Contains the loss items and batch information.

返回类型 dict

class mmedit.models.**DeepFillv1Inpaintor**(**args*, *stage1_loss_type=('loss_l1_hole')*, *stage2_loss_type=('loss_l1_hole', 'loss_gan')*, *input_with_ones=True*, *disc_input_with_mask=False*, ***kwargs*)

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_'.
- **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

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** (*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

get_module(*model*, *module_name*)

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

train_step(*data_batch*, *optimizer*)

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 gerator after *disc_step* iterations for discriminator.

参数

- **data_batch** (*torch.Tensor*) –Batch of data as input.
- **optimizer** (*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

two_stage_loss(*stage1_data*, *stage2_data*, *data_batch*)

Calculate two-stage loss.

参数

- **stage1_data** (*dict*) –Contain stage1 results.
- **stage2_data** (*dict*) –Contain stage2 results.
- **data_batch** (*dict*) –Contain data needed to calculate loss.

返回 Contain losses with name.

返回类型 dict

class mmedit.models.**ESRGAN**(*generator*, *discriminator=None*, *gan_loss=None*, *pixel_loss=None*, *perceptual_loss=None*, *train_cfg=None*, *test_cfg=None*, *pretrained=None*)

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.
- **pretrained** (*str*) –Path for pretrained model. Default: None.

train_step(*data_batch*, *optimizer*)

Train step.

参数

- **data_batch** (*dict*) –A batch of data.
- **optimizer** (*obj*) –Optimizer.

返回 Returned output.

返回类型 dict

class mmedit.models.**FeedbackHourglass**(*mid_channels*, *num_keypoints*)

Feedback Hourglass model for face landmark.

It has a style of:

```
-- preprocessing ----- Hourglass ----->
                    ^                |
                    |________________|
```

参数

- **mid_channels** (*int*) –Number of channels in the intermediate features.
- **num_keypoints** (*int*) –Number of keypoints.

forward(*x*, *last_hidden=None*)

Forward function.

参数

- **x** (*Tensor*) –Input tensor with shape (n, c, h, w).
- **last_hidden** (*Tensor | None*) –The feedback of FeedbackHourglass. In first step, last_hidden=None. Otherwise, last_hidden is the past output of FeedbackHourglass. Default: None.

返回 Heatmap of facial landmark. feedback (Tensor): Feedback Tensor.

返回类型 heatmap (Tensor)

class mmedit.models.**GCA**(*backbone*, *train_cfg=None*, *test_cfg=None*, *pretrained=None*, *loss_alpha=None*)

Guided Contextual Attention image matting model.

https://arxiv.org/abs/2001.04069

参数

- **backbone** (*dict*) –Config of backbone.
- **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.
- **pretrained** (*str*) –Path of the pretrained model.
- **loss_alpha** (*dict*) –Config of the alpha prediction loss. Default: None.

forward_test(*merged*, *trimap*, *meta*, *save_image=False*, *save_path=None*, *iteration=None*)

Defines the computation performed at every test call.

参数

- **merged** (*Tensor*) –Image to predict alpha matte.
- **trimap** (*Tensor*) –Trimap of the input image.
- **meta** (*list[dict]*) –Meta data about the current data batch. Currently only batch_size 1 is supported. It may contain information needed to calculate metrics (ori_alpha and ori_trimap) or save predicted alpha matte (merged_path).
- **save_image** (*bool, optional*) –Whether save predicted alpha matte. Defaults to False.
- **save_path** (*str, optional*) –The directory to save predicted alpha matte. Defaults to None.
- **iteration** (*int, optional*) –If given as None, the saved alpha matte will have the same file name with merged_path in meta dict. If given as an int, the saved alpha matte would named with postfix _{iteration}.png. Defaults to None.

返回 Contains the predicted alpha and evaluation result.

返回类型 dict

forward_train(*merged*, *trimap*, *meta*, *alpha*)

Forward function for training GCA model.

参数

- **merged** (*Tensor*) –with shape (N, C, H, W) encoding input images. Typically these should be mean centered and std scaled.
- **trimap** (*Tensor*) –with shape (N, C', H, W). Tensor of trimap. C' might be 1 or 3.
- **meta** (*list[dict]*) –Meta data about the current data batch.
- **alpha** (*Tensor*) –with shape (N, 1, H, W). Tensor of alpha.

返回 Contains the loss items and batch information.

返回类型 dict

class mmedit.models.**GLInpaintor**(*encdec*, *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*, *train_cfg=None*, *test_cfg=None*, *pretrained=None*)

Inpaintor for global&local method.

This inpaintor is implemented according to the paper: Globally and Locally Consistent Image Completion

Importantly, this inpaintor is an example for using custom training schedule based on *OneStageInpaintor*.

The training pipeline of global&local 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 theoriginal 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 perceptural and style loss with composed image as input.
- **loss_out_percep** (*dict*) –Config for perceptural 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.
- **pretrained** (*str*) –Path for pretrained model. Default None.

generator_loss (*fake_res*, *fake_img*, *fake_local*, *data_batch*)

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.
- **data_batch** (*dict*) –Contain other elements for computing losses.

返回 A tuple containing two dictionaries. The first one is the result dict, which contains the results computed within this function for visualization. The second one is the loss dict, containing loss items computed in this function.

返回类型 tuple[dict]

train_step (*data_batch*, *optimizer*)

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 in current schedule)
3. optimize generator (if in current schedule)

If `self.train_cfg.disc_step > 1`, the train step will contain multiple iterations for optimizing discriminator with different input data and sonly one iteration for optimizing generator after *disc_step* iterations for discriminator.

参数

- **data_batch** (*torch.Tensor*) –Batch of data as input.
- **optimizer** (*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 mmedit.models.**IndexNet**(*backbone*, *train_cfg=None*, *test_cfg=None*, *pretrained=None*, *loss_alpha=None*, *loss_comp=None*)

IndexNet matting model.

This implementation follows: Indices Matter: Learning to Index for Deep Image Matting

参数

- **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.
- **pretrained** (*str*) –path of pretrained model.
- **loss_alpha** (*dict*) –Config of the alpha prediction loss. Default: None.
- **loss_comp** (*dict*) –Config of the composition loss. Default: None.

forward_test(*merged*, *trimap*, *meta*, *save_image=False*, *save_path=None*, *iteration=None*)

Defines the computation performed at every test call.

参数

- **merged** (*Tensor*) –Image to predict alpha matte.
- **trimap** (*Tensor*) –Trimap of the input image.
- **meta** (*list[dict]*) –Meta data about the current data batch. Currently only batch_size 1 is supported. It may contain information needed to calculate metrics (`ori_alpha` and `ori_trimap`) or save predicted alpha matte (`merged_path`).
- **save_image** (*bool, optional*) –Whether save predicted alpha matte. Defaults to False.
- **save_path** (*str, optional*) –The directory to save predicted alpha matte. Defaults to None.

- **iteration** (*int, optional*) –If given as None, the saved alpha matte will have the same file name with `merged_path` in meta dict. If given as an int, the saved alpha matte would named with postfix `_{iteration}.png`. Defaults to None.

返回 Contains the predicted alpha and evaluation result.

返回类型 dict

forward_train(*merged*, *trimap*, *meta*, *alpha*, *ori_merged*, *fg*, *bg*)

Forward function for training IndexNet model.

参数

- **merged** (*Tensor*) –Input images tensor with shape (N, C, H, W). Typically these should be mean centered and std scaled.
- **trimap** (*Tensor*) –Tensor of trimap with shape (N, 1, H, W).
- **meta** (*list[dict]*) –Meta data about the current data batch.
- **alpha** (*Tensor*) –Tensor of alpha with shape (N, 1, H, W).
- **ori_merged** (*Tensor*) –Tensor of origin merged images (not normalized) with shape (N, C, H, W).
- **fg** (*Tensor*) –Tensor of foreground with shape (N, C, H, W).
- **bg** (*Tensor*) –Tensor of background with shape (N, C, H, W).

返回 Contains the loss items and batch information.

返回类型 dict

class mmedit.models.**LTE**(*requires_grad=True*, *pixel_range=1.0*, *pretrained=None*, *load_pretrained_vgg=True*)

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 geature. Default: 1.
- **pretrained** (*str*) –Path for pretrained model. Default: None.
- **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.

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]

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 mmedit.models.**OneStageInpaintor** (*encdec*, *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*, *train_cfg=None*, *test_cfg=None*, *pretrained=None*)

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

- **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 perceptural and style loss with composed image as input.
- **loss_out_percep** (*dict*) –Config for perceptural 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.
- **pretrained** (*str*) –Path for pretrained model. Default None.

forward (*masked_img*, *mask*, *test_mode=True*, ***kwargs*)

Forward function.

参数

- **masked_img** (*torch.Tensor*) –Image with hole as input.
- **mask** (*torch.Tensor*) –Mask as input.
- **test_mode** (*bool, optional*) –Whether use testing mode. Defaults to True.

返回 Dict contains output results.

返回类型 dict

forward_dummy (*x*)

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

forward_test (*masked_img*, *mask*, *save_image=False*, *save_path=None*, *iteration=None*, ***kwargs*)

Forward function for testing.

参数

- **masked_img** (*torch.Tensor*) –Tensor with shape of (n, 3, h, w).
- **mask** (*torch.Tensor*) –Tensor with shape of (n, 1, h, w).
- **save_image** (*bool, optional*) –If True, results will be saved as image. Defaults to False.
- **save_path** (*str, optional*) –If given a valid str, the reuslts will be saved in this path. Defaults to None.
- **iteration** (*int, optional*) –Iteration number. Defaults to None.

返回 Contain output results and eval metrics (if have).

返回类型 dict

forward_train(**args*, ***kwargs*)

Forward function for training.

In this version, we do not use this interface.

forward_train_d(*data_batch*, *is_real*, *is_disc*)

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** (*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*, *fake_img*, *data_batch*)

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.
- **data_batch** (*dict*) –Contain other elements for computing losses.

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)

init_weights(*pretrained=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.

save_visualization(*img*, *filename*)

Save visualization results.

参数

- **img** (*torch.Tensor*) –Tensor with shape of (n, 3, h, w).
- **filename** (*str*) –Path to save visualization.

train_step(*data_batch*, *optimizer*)

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 gerator after *disc_step* iterations for discriminator.

参数

- **data_batch** (*torch.Tensor*) –Batch of data as input.
- **optimizer** (*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

val_step(*data_batch*, ***kwargs*)

Forward function for evaluation.

参数 **data_batch** (*dict*) –Contain data for forward.

返回 Contain the results from model.

返回类型 dict

class mmedit.models.**PConvInpaintor**(*encdec*, *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*, *train_cfg=None*, *test_cfg=None*, *pretrained=None*)

forward_dummy(*x*)

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

forward_test(*masked_img*, *mask*, *save_image=False*, *save_path=None*, *iteration=None*, ***kwargs*)

Forward function for testing.

参数

- **masked_img** (*torch.Tensor*) –Tensor with shape of (n, 3, h, w).
- **mask** (*torch.Tensor*) –Tensor with shape of (n, 1, h, w).
- **save_image** (*bool, optional*) –If True, results will be saved as image. Defaults to False.
- **save_path** (*str, optional*) –If given a valid str, the results will be saved in this path. Defaults to None.
- **iteration** (*int, optional*) –Iteration number. Defaults to None.

返回 Contain output results and eval metrics (if have).

返回类型 dict

train_step(*data_batch*, *optimizer*)

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 gerator after *disc_step* iterations for discriminator.

参数

- **data_batch** (*torch.Tensor*) –Batch of data as input.
- **optimizer** (*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 mmedit.models.**Pix2Pix**(*generator*, *discriminator*, *gan_loss*, *pixel_loss=None*, *train_cfg=None*, *test_cfg=None*, *pretrained=None*)

Pix2Pix model for paired image-to-image translation.

Ref: Image-to-Image Translation with Conditional Adversarial Networks

参数

- **generator** (*dict*) –Config for the generator.

- **discriminator** (*dict*) –Config for the discriminator.
- **gan_loss** (*dict*) –Config for the gan loss.
- **pixel_loss** (*dict*) –Config for the pixel 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 generator update. *disc_init_steps*: how many discriminator updates at the start of the training. These two keys are useful when training with WGAN. *direction*: image-to-image translation direction (the model training direction): a2b | b2a.
- **test_cfg** (*dict*) –Config for testing. Default: None. You may change the testing of gan by setting: *direction*: image-to-image translation direction (the model training direction, same as testing direction): a2b | b2a. *show_input*: whether to show input real images.
- **pretrained** (*str*) –Path for pretrained model. Default: None.

backward_discriminator(*outputs*)

Backward function for the discriminator.

参数 **outputs** (*dict*) –Dict of forward results.

返回 Loss dict.

返回类型 dict

backward_generator(*outputs*)

Backward function for the generator.

参数 **outputs** (*dict*) –Dict of forward results.

返回 Loss dict.

返回类型 dict

forward(*img_a*, *img_b*, *meta*, *test_mode=False*, ***kwargs*)

Forward function.

参数

- **img_a** (*Tensor*) –Input image from domain A.
- **img_b** (*Tensor*) –Input image from domain B.
- **meta** (*list[dict]*) –Input meta data.
- **test_mode** (*bool*) –Whether in test mode or not. Default: False.
- **kwargs** (*dict*) –Other arguments.

forward_dummy(*img*)

Used for computing network FLOPs.

参数 **img** (*Tensor*) –Dummy input used to compute FLOPs.

返回 Dummy output produced by forwarding the dummy input.

返回类型 Tensor

forward_test(*img_a*, *img_b*, *meta*, *save_image=False*, *save_path=None*, *iteration=None*)

Forward function for testing.

参数

- **img_a** (*Tensor*) –Input image from domain A.
- **img_b** (*Tensor*) –Input image from domain B.
- **meta** (*list[dict]*) –Input meta data.
- **save_image** (*bool, optional*) –If True, results will be saved as images. Default: False.
- **save_path** (*str, optional*) –If given a valid str path, the results will be saved in this path. Default: None.
- **iteration** (*int, optional*) –Iteration number. Default: None.

返回 Dict of forward and evaluation results for testing.

返回类型 dict

forward_train(*img_a*, *img_b*, *meta*)

Forward function for training.

参数

- **img_a** (*Tensor*) –Input image from domain A.
- **img_b** (*Tensor*) –Input image from domain B.
- **meta** (*list[dict]*) –Input meta data.

返回 Dict of forward results for training.

返回类型 dict

init_weights(*pretrained=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.

setup(*img_a*, *img_b*, *meta*)

Perform necessary pre-processing steps.

参数

- **img_a** (*Tensor*) –Input image from domain A.
- **img_b** (*Tensor*) –Input image from domain B.

- **meta** (*list[dict]*) –Input meta data.

返回 The real images from domain A/B, and the image path as the metadata.

返回类型 Tensor, Tensor, list[str]

train_step(*data_batch*, *optimizer*)

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.

返回 Dict of loss, information for logger, the number of samples and results for visualization.

返回类型 dict

val_step(*data_batch*, ***kwargs*)

Validation step function.

参数

- **data_batch** (*dict*) –Dict of the input data batch.
- **kwargs** (*dict*) –Other arguments.

返回 Dict of evaluation results for validation.

返回类型 dict

class mmedit.models.**SRGAN**(*generator*, *discriminator=None*, *gan_loss=None*, *pixel_loss=None*, *perceptual_loss=None*, *train_cfg=None*, *test_cfg=None*, *pretrained=None*)

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. 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.
- **pretrained** (*str*) –Path for pretrained model. Default: None.

forward (*lq*, *gt=None*, *test_mode=False*, ***kwargs*)

Forward function.

参数

- **lq** (*Tensor*) –Input lq images.
- **gt** (*Tensor*) –Ground-truth image. Default: None.
- **test_mode** (*bool*) –Whether in test mode or not. Default: False.
- **kwargs** (*dict*) –Other arguments.

init_weights (*pretrained=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.

train_step (*data_batch*, *optimizer*)

Train step.

参数

- **data_batch** (*dict*) –A batch of data.
- **optimizer** (*obj*) –Optimizer.

返回 Returned output.

返回类型 dict

class mmedit.models.**SearchTransformer**

Search texture reference by transformer.

Include relevance embedding, hard-attention and soft-attention.

forward (*lq_up*, *ref_downup*, *refs*)

Texture transformer

Q = LTE(lq_up) K = LTE(ref_downup) 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

- **args are features come from extractor** (*All*) –These features contain 3 levels. When upscale_factor=4, the size ratio of these features is level3:level2:level1 = 1:2:4.
- **lq_up** (*Tensor*) –Tensor of 4x bicubic-upsampled lq image. (N, C, H, W)
- **ref_downup** (*Tensor*) –Tensor of ref_downup. ref_downup 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

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 soft_attention (Tensor)

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)

class mmedit.models.**TwoStageInpaintor**(**args*, *stage1_loss_type=('loss_l1_hole')*, *stage2_loss_type=('loss_l1_hole', 'loss_gan')*, *input_with_ones=True*, *disc_input_with_mask=False*, ***kwargs*)

Two-Stage Inpaintor.

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

- **stage1_loss_type** (*tuple[str]*) –Contains the loss names used in the first stage model.
- **stage2_loss_type** (*tuple[str]*) –Contains the loss names used in the second stage model.
- **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.

calculate_loss_with_type (*loss_type*, *fake_res*, *fake_img*, *gt*, *mask*, *prefix='stage1_'*)

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_'.

返回 Contain loss value with its name.

返回类型 dict

forward_test (*masked_img*, *mask*, *save_image=False*, *save_path=None*, *iteration=None*, ***kwargs*)

Forward function for testing.

参数

- **masked_img** (*torch.Tensor*) –Tensor with shape of (n, 3, h, w).
- **mask** (*torch.Tensor*) –Tensor with shape of (n, 1, h, w).
- **save_image** (*bool, optional*) –If True, results will be saved as image. Defaults to False.
- **save_path** (*str, optional*) –If given a valid str, the results will be saved in this path. Defaults to None.
- **iteration** (*int, optional*) –Iteration number. Defaults to None.

返回 Contain output results and eval metrics (if have).

返回类型 dict

save_visualization (*img*, *filename*)

Save visualization results.

参数

- **img** (*torch.Tensor*) –Tensor with shape of (n, 3, h, w).
- **filename** (*str*) –Path to save visualization.

train_step (*data_batch*, *optimizer*)

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 gerator after *disc_step* iterations for discriminator.

参数

- **data_batch** (*torch.Tensor*) –Batch of data as input.
- **optimizer** (*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

two_stage_loss(*stage1_data*, *stage2_data*, *data_batch*)

Calculate two-stage loss.

参数

- **stage1_data** (*dict*) –Contain stage1 results.
- **stage2_data** (*dict*) –Contain stage2 results.
- **data_batch** (*dict*) –Contain data needed to calculate loss.

返回 Contain losses with name.

返回类型 dict

mmedit.models.**build**(*cfg*, *registry*, *default_args=None*)

Build module function.

参数

- **cfg** (*dict*) –Configuration for building modules.
- **registry** (*obj*) –`registry` object.
- **default_args** (*dict, optional*) –Default arguments. Defaults to None.

mmedit.models.**build_backbone**(*cfg*)

Build backbone.

参数 **cfg** (*dict*) –Configuration for building backbone.

mmedit.models.**build_component**(*cfg*)

Build component.

参数 **cfg** (*dict*) –Configuration for building component.

mmedit.models.**build_loss**(*cfg*)

Build loss.

参数 **cfg** (*dict*) –Configuration for building loss.

mmedit.models.**build_model**(*cfg*, *train_cfg=None*, *test_cfg=None*)

Build model.

参数

- **cfg** (*dict*) –Configuration for building model.
- **train_cfg** (*dict*) –Training configuration. Default: None.
- **test_cfg** (*dict*) –Testing configuration. Default: None.

22.2 common

class mmedit.models.common.**ASPP**(*in_channels*, *out_channels=256*, *mid_channels=256*, *dilations=(12, 24, 36)*, *conv_cfg=None*, *norm_cfg={'type': 'BN'}*, *act_cfg={'type': 'ReLU'}*, *separable_conv=False*)

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.
- **mid_channels** (*int*) –Output channels of the intermediate ASPP conv modules.
- **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*)

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 mmedit.models.common.**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*)

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.

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

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

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)

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
...
```

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

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

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

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

class mmedit.models.common.**DepthwiseSeparableConvModule**(*in_channels*, *out_channels*, *kernel_size*, *stride=1*, *padding=0*, *dilation=1*, *norm_cfg=None*, *act_cfg={'type': 'ReLU'}*, *dw_norm_cfg='default'*, *dw_act_cfg='default'*, *pw_norm_cfg='default'*, *pw_act_cfg='default'*, ***kwargs*)

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*)

Forward function.

参数 **x** (*Tensor*) –Input tensor with shape (N, C, H, W).

返回 Output tensor.

返回类型 Tensor

class mmedit.models.common.**GANImageBuffer**(*buffer_size*, *buffer_ratio=0.5*)

This class implements an image buffer that stores previously generated images.

This buffer allows us to update the discriminator using a history of generated images rather than the ones produced by the latest generator to reduce model oscillation.

参数

- **buffer_size** (*int*) –The size of image buffer. If buffer_size = 0, no buffer will be created.
- **buffer_ratio** (*float*) –The chance / possibility to use the images previously stored in the buffer.

query(*images*)

Query current image batch using a history of generated images.

参数 **images** (*Tensor*) –Current image batch without history information.

class mmedit.models.common.**GCAModule**(*in_channels*, *out_channels*, *kernel_size=3*, *stride=1*, *rate=2*, *pad_args={'mode': 'reflect'}*, *interpolation='nearest'*, *penalty=-10000.0*, *eps=0.0001*)

Guided Contextual Attention Module.

From https://arxiv.org/pdf/2001.04069.pdf. Based on https://github.com/nbei/Deep-Flow-Guided-Video-Inpainting. This module use image feature map to augment the alpha feature map with guided contextual attention score.

Image feature and alpha feature are unfolded to small patches and later used as conv kernel. Thus, we refer the unfolding size as kernel size. Image feature patches have a default kernel size 3 while the kernel size of alpha feature

patches could be specified by *rate* (see *rate* below). The image feature patches are used to convolve with the image feature itself to calculate the contextual attention. Then the attention feature map is convolved by alpha feature patches to obtain the attention alpha feature. At last, the attention alpha feature is added to the input alpha feature.

参数

- **in_channels** (*int*) –Input channels of the guided contextual attention module.
- **out_channels** (*int*) –Output channels of the guided contextual attention module.
- **kernel_size** (*int*) –Kernel size of image feature patches. Default 3.
- **stride** (*int*) –Stride when unfolding the image feature. Default 1.
- **rate** (*int*) –The downsample rate of image feature map. The corresponding kernel size and stride of alpha feature patches will be *rate x 2* and *rate*. It could be regarded as the granularity of the gca module. Default: 2.
- **pad_args** (*dict*) –Parameters of padding when convolve image feature with image feature patches or alpha feature patches. Allowed keys are *mode* and *value*. See torch.nn.functional.pad() for more information. Default: dict(mode=’reflect’).
- **interpolation** (*str*) –Interpolation method in upsampling and downsampling.
- **penalty** (*float*) –Punishment hyperparameter to avoid a large correlation between each unknown patch and itself.
- **eps** (*float*) –A small number to avoid dividing by 0 when calculating the normed image feature patch. Default: 1e-4.

compute_guided_attention_score(*similarity_map*, *unknown_ps*, *scale*, *self_mask*)

Compute guided attention score.

参数

- **similarity_map** (*Tensor*) –Similarity map of image feature with shape (1, img_h*img_w, img_h, img_w).
- **unknown_ps** (*Tensor*) –Unknown area patches tensor of shape (1, img_h*img_w, 1, 1).
- **scale** (*Tensor*) –Softmax scale of known and unknown area: [unknown_scale, known_scale].
- **self_mask** (*Tensor*) –Self correlation mask of shape (1, img_h*img_w, img_h, img_w). At (1, i*i, i, i) mask value equals -1e4 for i in [1, img_h*img_w] and other area is all zero.

返回 Similarity map between image feature patches with shape (1, img_h*img_w, img_h, img_w).

返回类型 Tensor

compute_similarity_map(*img_feat*, *img_ps*)

Compute similarity between image feature patches.

参数

- **img_feat** (*Tensor*) –Image feature map of shape (1, img_c, img_h, img_w).
- **img_ps** (*Tensor*) –Image feature patches tensor of shape (1, img_h*img_w, img_c, img_ks, img_ks).

返回 Similarity map between image feature patches with shape (1, img_h*img_w, img_h, img_w).

返回类型 Tensor

extract_feature_maps_patches (*img_feat*, *alpha_feat*, *unknown*)

Extract image feature, alpha feature unknown patches.

参数

- **img_feat** (*Tensor*) –Image feature map of shape (N, img_c, img_h, img_w).
- **alpha_feat** (*Tensor*) –Alpha feature map of shape (N, alpha_c, ori_h, ori_w).
- **unknown** (*Tensor, optional*) –Unknown area map generated by trimap of shape (N, 1, img_h, img_w).

返回

3-tuple of

`Tensor`: Image feature patches of shape (N, img_h*img_w, img_c, img_ks, img_ks).

`Tensor`: Guided contextual attention alpha feature map. (N, img_h*img_w, alpha_c, alpha_ks, alpha_ks).

`Tensor`: Unknown mask of shape (N, img_h*img_w, 1, 1).

返回类型 tuple

extract_patches (*x*, *kernel_size*, *stride*)

Extract feature patches.

The feature map will be padded automatically to make sure the number of patches is equal to *(H / stride) * (W / stride)*.

参数

- **x** (*Tensor*) –Feature map of shape (N, C, H, W).
- **kernel_size** (*int*) –Size of each patches.
- **stride** (*int*) –Stride between patches.

返回 Extracted patches of shape (N, (H / stride) * (W / stride) , C, kernel_size, kernel_size).

返回类型 Tensor

forward (*img_feat*, *alpha_feat*, *unknown=None*, *softmax_scale=1.0*)

Forward function of GCAModule.

参数

- **img_feat** (*Tensor*) –Image feature map of shape (N, ori_c, ori_h, ori_w).
- **alpha_feat** (*Tensor*) –Alpha feature map of shape (N, alpha_c, ori_h, ori_w).
- **unknown** (*Tensor, optional*) –Unknown area map generated by trimap. If specified, this tensor should have shape (N, 1, ori_h, ori_w).
- **softmax_scale** (*float, optional*) –The softmax scale of the attention if unknown area is not provided in forward. Default: 1.

返回 The augmented alpha feature.

返回类型 Tensor

process_unknown_mask (*unknown*, *img_feat*, *softmax_scale*)

Process unknown mask.

参数

- **unknown** (*Tensor, optional*) –Unknown area map generated by trimap of shape (N, 1, ori_h, ori_w)
- **img_feat** (*Tensor*) –The interpolated image feature map of shape (N, img_c, img_h, img_w).
- **softmax_scale** (*float, optional*) –The softmax scale of the attention if unknown area is not provided in forward. Default: 1.

返回

2-tuple of

`Tensor`: Interpolated unknown area map of shape (N, img_h*img_w, img_h, img_w).

`Tensor`: Softmax scale tensor of known and unknown area of shape (N, 2).

返回类型 tuple

propagate_alpha_feature (*gca_score*, *alpha_ps*)

Propagate alpha feature based on guided attention score.

参数

- **gca_score** (*Tensor*) –Guided attention score map of shape (1, img_h*img_w, img_h, img_w).
- **alpha_ps** (*Tensor*) –Alpha feature patches tensor of shape (1, img_h*img_w, alpha_c, alpha_ks, alpha_ks).

返回 Propagated alpha feature map of shape (1, alpha_c, alpha_h, alpha_w).

返回类型 Tensor

class mmedit.models.common.**ImgNormalize**(*pixel_range*, *img_mean*, *img_std*, *sign=- 1*)

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 mmedit.models.common.**LinearModule**(*in_features*, *out_features*, *bias=True*, *act_cfg={'type': 'ReLU'}*, *inplace=True*, *with_spectral_norm=False*, *order=('linear', 'act')*)

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.
- **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 linear module.
- **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").

forward(*x*, *activate=True*)

Forward Function.

参数

- **x** (*torch.Tensor*) –Input tensor with shape of (n, *, # noqa: W605 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 mmedit.models.common.**MaskConvModule**(**args*, ***kwargs*)

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").

forward (*x*, *mask=None*, *activate=True*, *norm=True*, *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.
- **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 mmedit.models.common.**PartialConv2d**(**args*, *multi_channel=False*, *eps=1e-08*, ***kwargs*)

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 mmedit.models.common.**PixelShufflePack**(*in_channels*, *out_channels*, *scale_factor*, *upsample_kernel*)

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.

forward(*x*)

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

init_weights()

Initialize weights for PixelShufflePack.

class mmedit.models.common.**ResidualBlockNoBN**(*mid_channels=64*, *res_scale=1.0*)

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.

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()

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"

class mmedit.models.common.**ResidualBlockWithDropout**(*channels*, *padding_mode*, *norm_cfg={'type': 'BN'}*, *use_dropout=True*)

Define a Residual Block with dropout layers.

Ref: Deep Residual Learning for Image Recognition

A residual block is a conv block with skip connections. A dropout layer is added between two common conv modules.

参数

- **channels** (*int*) –Number of channels in the conv layer.
- **padding_mode** (*str*) –The name of padding layer: 'reflect' | 'replicate' | 'zeros'.
- **norm_cfg** (*dict*) –Config dict to build norm layer. Default: *dict(type='IN')*.
- **use_dropout** (*bool*) –Whether to use dropout layers. Default: True.

forward(*x*)

Forward function. Add skip connections without final ReLU.

参数 **x** (*Tensor*) –Input tensor with shape (n, c, h, w).

返回 Forward results.

返回类型 Tensor

class mmedit.models.common.**SimpleGatedConvModule**(*in_channels*, *out_channels*, *kernel_size*, *feat_act_cfg={'type': 'ELU'}*, *gate_act_cfg={'type': 'Sigmoid'}*, ***kwargs*)

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.
- **gate_act_cfg** (*dict*) –Config dict for gate activation layer.
- **kwargs** (*keyword arguments*) –Same as *ConvModule*.

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 mmedit.models.common.**UnetSkipConnectionBlock**(*outer_channels*, *inner_channels*, *in_channels=None*, *submodule=None*, *is_outermost=False*, *is_innermost=False*, *norm_cfg={'type': 'BN'}*, *use_dropout=False*)

Construct a Unet submodule with skip connections, with the following structure: downsampling - *submodule* - upsampling.

参数

- **outer_channels** (*int*) –Number of channels at the outer conv layer.
- **inner_channels** (*int*) –Number of channels at the inner conv layer.
- **in_channels** (*int*) –Number of channels in input images/features. If is None, equals to *outer_channels*. Default: None.
- **submodule** (UnetSkipConnectionBlock) –Previously constructed submodule. Default: None.
- **is_outermost** (*bool*) –Whether this module is the outermost module. Default: False.
- **is_innermost** (*bool*) –Whether this module is the innermost module. Default: False.
- **norm_cfg** (*dict*) –Config dict to build norm layer. Default: *dict(type='BN')*.
- **use_dropout** (*bool*) –Whether to use dropout layers. Default: False.

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

mmedit.models.common.**default_init_weights**(*module*, *scale=1*)

Initialize network weights.

参数

- **modules** (*nn.Module*) –Modules to be initialized.
- **scale** (*float*) –Scale initialized weights, especially for residual blocks.

mmedit.models.common.**extract_around_bbox**(*img*, *bbox*, *target_size*, *channel_first=True*)

Extract patches around the given bbox.

参数

- **bbox** (*np.ndarray | torch.Tensor*) –Bboxes to be modified. Bbox can be in batch or not.
- **target_size** (*List(int)*) –Target size of final bbox.

返回 Extracted patches. The dimension of the output should be the same as *img*.

返回类型 (torch.Tensor | numpy.array)

mmedit.models.common.**extract_bbox_patch** (*bbox*, *img*, *channel_first=True*)

Extract patch from a given bbox

参数

- **bbox** (*torch.Tensor | numpy.array*) –Bbox with (top, left, h, w). If *img* has batch dimension, the *bbox* must be stacked at first dimension. The shape should be (4,) or (n, 4).
- **img** (*torch.Tensor | numpy.array*) –Image data to be extracted. If organized in batch dimension, the batch dimension must be the first order like (n, h, w, c) or (n, c, h, w).
- **channel_first** (*bool*) –If True, the channel dimension of img is before height and width, e.g. (c, h, w). Otherwise, the img shape (samples in the batch) is like (h, w, c).

返回 Extracted patches. The dimension of the output should be the same as *img*.

返回类型 (torch.Tensor | numpy.array)

mmedit.models.common.**flow_warp** (*x*, *flow*, *interpolation='bilinear'*, *padding_mode='zeros'*, *align_corners=True*)

Warp an image or a feature map with optical flow.

参数

- **x** (*Tensor*) –Tensor with size (n, c, h, w).
- **flow** (*Tensor*) –Tensor with size (n, h, w, 2). The last dimension is a two-channel, denoting the width and height relative offsets. Note that the values are not normalized to [-1, 1].
- **interpolation** (*str*) –Interpolation mode: ‘nearest’ or ‘bilinear’. Default: ‘bilinear’.
- **padding_mode** (*str*) –Padding mode: ‘zeros’ or ‘border’ or ‘reflection’. Default: ‘zeros’.
- **align_corners** (*bool*) –Whether align corners. Default: True.

返回 Warped image or feature map.

返回类型 Tensor

mmedit.models.common.**generation_init_weights** (*module*, *init_type='normal'*, *init_gain=0.02*)

Default initialization of network weights for image generation.

By default, we use normal init, but xavier and kaiming might work better for some applications.

参数

- **module** (*nn.Module*) –Module to be initialized.
- **init_type** (*str*) –The name of an initialization method: normal | xavier | kaiming | orthogonal.
- **init_gain** (*float*) –Scaling factor for normal, xavier and orthogonal.

mmedit.models.common.**make_layer**(*block*, *num_blocks*, ***kwarg*)

Make layers by stacking the same blocks.

参数

- **block** (*nn.module*) –nn.module class for basic block.
- **num_blocks** (*int*) –number of blocks.

返回 Stacked blocks in nn.Sequential.

返回类型 nn.Sequential

mmedit.models.common.**pixel_unshuffle**(*x*, *scale*)

Down-sample by pixel unshuffle.

参数

- **x** (*Tensor*) –Input tensor.
- **scale** (*int*) –Scale factor.

返回 Output tensor.

返回类型 Tensor

mmedit.models.common.**scale_bbox**(*bbox*, *target_size*)

Modify bbox to target size.

The original bbox will be enlarged to the target size with the original bbox in the center of the new bbox.

参数

- **bbox** (*np.ndarray | torch.Tensor*) –Bboxes to be modified. Bbox can be in batch or not. The shape should be (4,) or (n, 4).
- **target_size** (*tuple[int]*) –Target size of final bbox.

返回 Modified bboxes.

返回类型 (np.ndarray | torch.Tensor)

mmedit.models.common.**set_requires_grad**(*nets*, *requires_grad=False*)

Set requires_grad for all the networks.

参数

- **nets** (*nn.Module | list[nn.Module]*) –A list of networks or a single network.

- **requires_grad** (*bool*) –Whether the networks require gradients or not

22.3 backbones

class mmedit.models.backbones.**BasicVSRNet**(*mid_channels=64*, *num_blocks=30*, *spynet_pretrained=None*)

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

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: None.
- **strict** (*boo, optional*) –Whether strictly load the pretrained model. Defaults to True.

class mmedit.models.backbones.**BasicVSRPlusPlus** (*mid_channels=64*, *num_blocks=7*, *max_residue_magnitude=10*, *is_low_res_input=True*, *spynet_pretrained=None*, *cpu_cache_length=100*)

BasicVSR++ network structure.

Support either x4 upsampling or same size output. Since DCN is used in this model, it can only be used with CUDA enabled. If CUDA is not enabled, feature alignment will be skipped.

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)

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

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. Default: None.
- **strict** (*bool, optional*) –Whether strictly load the pretrained model. Default: True.

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 propgation 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 propgation branches.

返回 Output HR sequence with shape (n, t, c, 4h, 4w).

返回类型 Tensor

class mmedit.models.backbones.**ContextualAttentionNeck**(*in_channels*, *conv_type='conv'*, *conv_cfg=None*, *norm_cfg=None*, *act_cfg={'type': 'ELU'}*, *contextual_attention_args={'softmax_scale': 10.0}*, ***kwargs*)

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*) –

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 mmedit.models.backbones.**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*)

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

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

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 mmedit.models.backbones.**DeepFillDecoder**(*in_channels*, *conv_type='conv'*, *norm_cfg=None*, *act_cfg={'type': 'ELU'}*, *out_act_cfg={'max': 1.0, 'min': - 1.0, 'type': 'clip'}*, *channel_factor=1.0*, ***kwargs*)

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*) –

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 mmedit.models.backbones.**DeepFillEncoder**(*in_channels=5*, *conv_type='conv'*, *norm_cfg=None*, *act_cfg={'type': 'ELU'}*, *encoder_type='stage1'*, *channel_factor=1.0*, ***kwargs*)

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*) –

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 mmedit.models.backbones.**DeepFillEncoderDecoder**(*stage1={'decoder': {'in_channels': 128, 'type': 'DeepFillDecoder'}, 'dilation_neck': {'act_cfg': {'type': 'ELU'}, 'in_channels': 128, 'type': 'GLDilationNeck'}, 'encoder': {'type': 'DeepFillEncoder'}, 'type': 'GLEncoderDecoder'}*, *stage2={'type': 'DeepFillRefiner'}*, *return_offset=False*)

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(*pretrained=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 mmedit.models.backbones.**DepthwiseIndexBlock**(*in_channels*, *norm_cfg={'type': 'BN'}*, *use_context=False*, *use_nonlinear=False*, *mode='o2o'*)

Depthwise index block.

From https://arxiv.org/abs/1908.00672.

参数

- **in_channels** (*int*) –Input channels of the holistic index block.
- **kernel_size** (*int*) –Kernel size of the conv layers. Default: 2.
- **padding** (*int*) –Padding number of the conv layers. Default: 0.
- **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*.
- **norm_cfg** (*dict*) –Config dict for normalization layer. Default: dict(type='BN').
- **use_nonlinear** (*bool*) –Whether add a non-linear conv layer in the index blocks. 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 mmedit.models.backbones.**EDSR**(*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)*)

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** (*tuple[float]*) –Image mean in RGB orders. Default: (0.4488, 0.4371, 0.4040), calculated from DIV2K dataset.
- **rgb_std** (*tuple[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

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 mmedit.models.backbones.**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*)

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.

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(*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 mmedit.models.backbones.**FBADecoder**(*pool_scales*, *in_channels*, *channels*, *conv_cfg=None*, *norm_cfg={'type': 'BN'}*, *act_cfg={'type': 'ReLU'}*, *align_corners=False*)

Decoder for FBA matting.

pool_scales (tuple[int]): Pooling scales used in Pooling Pyramid Module.

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.

forward(*inputs*)

Forward function. :param inputs: Output dict of FbaEncoder. :type inputs: dict

返回 Predicted alpha, fg and bg of the current batch.

返回类型 Tensor

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.

class mmedit.models.backbones.**FBAResnetDilated**(*depth*, *in_channels*, *stem_channels*, *base_channels*, *num_stages=4*, *strides=(1, 2, 2, 2)*, *dilations=(1, 1, 2, 4)*, *deep_stem=False*, *avg_down=False*, *frozen_stages=- 1*, *act_cfg={'type': 'ReLU'}*, *conv_cfg=None*, *norm_cfg={'type': 'BN'}*, *with_cp=False*, *multi_grid=None*, *contract_dilation=False*, *zero_init_residual=True*)

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 mmedit.models.backbones.**GLDecoder**(*in_channels=256*, *norm_cfg=None*, *act_cfg={'type': 'ReLU'}*, *out_act='clip'*)

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 mmedit.models.backbones.**GLDilationNeck**(*in_channels=256*, *conv_type='conv'*, *norm_cfg=None*, *act_cfg={'type': 'ReLU'}*, ***kwargs*)

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*) –

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 mmedit.models.backbones.**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*, *pretrained=None*, *bgr2rgb=False*)

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 covolutional 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:

- styelgan2-ffhq-config-f: http://download.openmmlab.com/mmgen/stylegan2/official_weights/stylegan2-ffhq-config-f-official_20210327_171224-bce9310c.pth # noqa
- stylegan2-horse-config-f: http://download.openmmlab.com/mmgen/stylegan2/official_weights/stylegan2-horse-config-f-official_20210327_173203-ef3e69ca.pth # noqa
- stylegan2-car-config-f: http://download.openmmlab.com/mmgen/stylegan2/official_weights/stylegan2-car-config-f-official_20210327_172340-8cfe053c.pth # noqa
- styelgan2-cat-config-f: http://download.openmmlab.com/mmgen/stylegan2/official_weights/stylegan2-cat-config-f-official_20210327_172444-15bc485b.pth # noqa

- stylegan2-church-config-f: http://download.openmmlab.com/mmgen/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 mulitiplier 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 defaultly adopt mixing style mode. 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.

- **pretrained** (*dict | None, optional*) –Information for pretained models. The necessary key is ‘ckpt_path’. Besides, you can also provide ‘prefix’ to load the generator part from the whole state dict. Defaults to None.
- **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

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 mmedit.models.backbones.**GLEncoder**(*norm_cfg=None*, *act_cfg={'type': 'ReLU'}*)

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 mmedit.models.backbones.**GLEncoderDecoder**(*encoder={'type': 'GLEncoder'}*, *decoder={'type': 'GLDecoder'}*, *dilation_neck={'type': 'GLDilationNeck'}*)

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

init_weights(*pretrained=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 mmedit.models.backbones.**HolisticIndexBlock**(*in_channels*, *norm_cfg={'type': 'BN'}*, *use_context=False*, *use_nonlinear=False*)

Holistic Index Block.

From https://arxiv.org/abs/1908.00672.

参数

- **in_channels** (*int*) –Input channels of the holistic index block.
- **kernel_size** (*int*) –Kernel size of the conv layers. Default: 2.
- **padding** (*int*) –Padding number of the conv layers. Default: 0.
- **norm_cfg** (*dict*) –Config dict for normalization layer. Default: dict(type=’BN’).
- **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 mmedit.models.backbones.**IconVSR**(*mid_channels=64*, *num_blocks=30*, *keyframe_stride=5*, *padding=2*, *spynet_pretrained=None*, *edvr_pretrained=None*)

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.

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)

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)

forward(*lrs*)

Forward function for IconVSR. :param lrs: Input LR tensor with shape (n, t, c, h, w). :type lrs: Tensor

返回 Output HR tensor with shape (n, t, c, 4h, 4w).

返回类型 Tensor

init_weights(*pretrained=None*, *strict=True*)

Init weights for models. :param pretrained: Path for pretrained weights. If given

None, pretrained weights will not be loaded. Defaults to None.

参数 **strict** (*boo, optional*) –Whether strictly load the pretrained model. Defaults to True.

spatial_padding(*lrs*)

Apply pdding 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

class mmedit.models.backbones.**IndexNetDecoder**(*in_channels*, *kernel_size=5*, *norm_cfg={'type': 'BN'}*, *separable_conv=False*)

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

init_weights()

Init weights for the module.

class mmedit.models.backbones.**IndexNetEncoder**(*in_channels*, *out_stride=32*, *width_mult=1*, *index_mode='m2o'*, *aspp=True*, *norm_cfg={'type': 'BN'}*, *freeze_bn=False*, *use_nonlinear=True*, *use_context=True*)

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 nonlinearty 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.

引发

- **ValueError** –out_stride must 16 or 32.
- **NameError** –Supported index_mode are {'holistic', 'o2o', 'm2o'}.

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

freeze_bn()

Set BatchNorm modules in the model to evaluation mode.

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.

class mmedit.models.backbones.**IndexedUpsample**(*in_channels*, *out_channels*, *kernel_size=5*, *norm_cfg={'type': 'BN'}*, *conv_module=<class 'mmcv.cnn.bricks.conv_module.ConvModule'>*)

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.

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

init_weights()

Init weights for the module.

class mmedit.models.backbones.**LIIFEDSR**(*encoder*, *imnet*, *local_ensemble=True*, *feat_unfold=True*, *cell_decode=True*, *eval_bsize=None*)

LIIF net based on EDSR.

Paper: Learning Continuous Image Representation with Local Implicit Image Function

参数

- **encoder** (*dict*) –Config for the generator.
- **imnet** (*dict*) –Config for the imnet.
- **local_ensemble** (*bool*) –Whether to use local ensemble. Default: True.
- **feat_unfold** (*bool*) –Whether to use feature unfold. Default: True.
- **cell_decode** (*bool*) –Whether to use cell decode. Default: True.
- **eval_bsize** (*int*) –Size of batched predict. Default: None.

gen_feature(*x*)

Generate feature.

参数 **x** (*Tensor*) –Input tensor with shape (n, c, h, w).

返回 Forward results.

返回类型 Tensor

class mmedit.models.backbones.**LIIFRDN**(*encoder*, *imnet*, *local_ensemble=True*, *feat_unfold=True*, *cell_decode=True*, *eval_bsize=None*)

LIIF net based on RDN.

Paper: Learning Continuous Image Representation with Local Implicit Image Function

参数

- **encoder** (*dict*) –Config for the generator.
- **imnet** (*dict*) –Config for the imnet.
- **local_ensemble** (*bool*) –Whether to use local ensemble. Default: True.
- **feat_unfold** (*bool*) –Whether to use feat unfold. Default: True.
- **cell_decode** (*bool*) –Whether to use cell decode. Default: True.
- **eval_bsize** (*int*) –Size of batched predict. Default: None.

gen_feature(*x*)

Generate feature.

参数 **x** (*Tensor*) –Input tensor with shape (n, c, h, w).

返回 Forward results.

返回类型 Tensor

class mmedit.models.backbones.**MSRResNet**(*in_channels*, *out_channels*, *mid_channels=64*, *num_blocks=16*, *upscale_factor=4*)

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.

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(*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 mmedit.models.backbones.**PConvDecoder**(*num_layers=7*, *interpolation='nearest'*, *conv_cfg={'multi_channel': True, 'type': 'PConv'}*, *norm_cfg={'type': 'BN'}*)

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 mmedit.models.backbones.**PConvEncoder**(*in_channels=3*, *num_layers=7*, *conv_cfg={'multi_channel': True, 'type': 'PConv'}*, *norm_cfg={'requires_grad': True, 'type': 'BN'}*, *norm_eval=False*)

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.

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

train(*mode=True*)

Sets the module in training mode.

This has any effect only on certain modules. See documentations of particular modules for details of their behaviors in training/evaluation mode, if they are affected, e.g. `Dropout`, `BatchNorm`, etc.

参数 **mode** (*bool*) –whether to set training mode (`True`) or evaluation mode (`False`). Default: `True`.

返回 self

返回类型 Module

class mmedit.models.backbones.**PConvEncoderDecoder**(*encoder*, *decoder*)

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

init_weights(*pretrained=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 mmedit.models.backbones.**PlainDecoder**(*in_channels*)

Simple decoder from Deep Image Matting.

参数 **in_channels** (*int*) –Channel num of input features.

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

init_weights()

Init weights for the module.

class mmedit.models.backbones.**RDN**(*in_channels*, *out_channels*, *mid_channels=64*, *num_blocks=16*, *upscale_factor=4*, *num_layers=8*, *channel_growth=64*)

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.

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

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 mmedit.models.backbones.**RRDBNet**(*in_channels*, *out_channels*, *mid_channels=64*, *num_blocks=23*, *growth_channels=32*)

Networks consisting of Residual in Residual Dense Block, which is used in ESRGAN.

ESRGAN: Enhanced Super-Resolution Generative Adversarial Networks. Currently, it supports 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.

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(*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 mmedit.models.backbones.**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*)

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*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

注解: Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

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: None.
- **strict** (*boo, optional*) –Whether strictly load the pretrained model. Defaults to True.

class mmedit.models.backbones.**ResGCADecoder** (*block*, *layers*, *in_channels*, *kernel_size=3*, *conv_cfg=None*, *norm_cfg={'type': 'BN'}*, *act_cfg={'inplace': True, 'negative_slope': 0.2, 'type': 'LeakyReLU'}*, *with_spectral_norm=False*, *late_downsample=False*)

ResNet decoder with shortcut connection and gca module.

```
feat1 ---------------------------------------- conv2 --- out
                                                |
feat2 ------------------------------------ conv1
                                          |
feat3 ------------------------------- layer4
                                    |
feat4, img_feat -- gca_module - layer3
```

(下页继续)

(续上页)

```
                |
feat5 ------- layer2
           |
out ---  layer1
```

- gca module also requires unknown tensor generated by trimap which is ignored in the above graph.

参数

- **block** (*str*) –Type of residual block. Currently only *BasicBlockDec* is implemented.
- **layers** (*list[int]*) –Number of layers in each block.
- **in_channels** (*int*) –Channel number of input features.
- **kernel_size** (*int*) –Kernel size of the conv layers in the decoder.
- **conv_cfg** (*dict*) –Dictionary to construct convolution layer. If it is None, 2d convolution will be applied. Default: None.
- **norm_cfg** (*dict*) –Config dict for normalization layer. "BN" by default.
- **act_cfg** (*dict*) –Config dict for activation layer, "ReLU" by default.
- **late_downsample** (*bool*) –Whether to adopt late downsample strategy, Default: False.

forward(*inputs*)

Forward function of resnet shortcut decoder.

参数 **inputs** (*dict*) –Output dictionary of the ResGCAEncoder containing:

- out (Tensor): Output of the ResGCAEncoder.
- feat1 (Tensor): Shortcut connection from input image.
- feat2 (Tensor): Shortcut connection from conv2 of ResGCAEncoder.
- feat3 (Tensor): Shortcut connection from layer1 of ResGCAEncoder.
- feat4 (Tensor): Shortcut connection from layer2 of ResGCAEncoder.
- feat5 (Tensor): Shortcut connection from layer3 of ResGCAEncoder.
- img_feat (Tensor): Image feature extracted by guidance head.
- unknown (Tensor): Unknown tensor generated by trimap.

返回 Output tensor.

返回类型 Tensor

class mmedit.models.backbones.**ResGCAEncoder**(*block*, *layers*, *in_channels*, *conv_cfg=None*, *norm_cfg={'type': 'BN'}*, *act_cfg={'type': 'ReLU'}*, *with_spectral_norm=False*, *late_downsample=False*, *order=('conv', 'act', 'norm')*)

ResNet backbone with shortcut connection and gca module.

```
image ---------------- shortcut[0] -------------- feat1
 |
conv1-conv2 ---------- shortcut[1] -------------- feat2
       |
     conv3-layer1 ---- shortcut[2] -------------- feat3
             |
             | image - guidance_conv ------------ img_feat
             |              |
            layer2 --- gca_module - shortcut[4] - feat4
                           |
                        layer3 -- shortcut[5] - feat5
                           |
                         layer4 --------------- out
```

- gca module also requires unknown tensor generated by trimap which is ignored in the above graph.

Implementation of Natural Image Matting via Guided Contextual Attention https://arxiv.org/pdf/2001.04069.pdf.

参数

- **block** (*str*) –Type of residual block. Currently only *BasicBlock* is implemented.
- **layers** (*list[int]*) –Number of layers in each block.
- **in_channels** (*int*) –Number of input channels.
- **conv_cfg** (*dict*) –Dictionary to construct convolution layer. If it is None, 2d convolution will be applied. Default: None.
- **norm_cfg** (*dict*) –Config dict for normalization layer. "BN" by default.
- **act_cfg** (*dict*) –Config dict for activation layer, "ReLU" by default.
- **late_downsample** (*bool*) –Whether to adopt late downsample strategy. Default: False.
- **order** (*tuple[str]*) –Order of *conv*, *norm* and *act* layer in shortcut convolution module. Default: ('conv', 'act', 'norm').

forward(*x*)

Forward function.

参数 **x** (*Tensor*) –Input tensor with shape (N, C, H, W).

返回 Contains the output tensor, shortcut feature and intermediate feature.

返回类型 dict

class mmedit.models.backbones.**ResNetDec**(*block*, *layers*, *in_channels*, *kernel_size=3*, *conv_cfg=None*, *norm_cfg={'type': 'BN'}*, *act_cfg={'inplace': True, 'negative_slope': 0.2, 'type': 'LeakyReLU'}*, *with_spectral_norm=False*, *late_downsample=False*)

ResNet decoder for image matting.

This class is adopted from https://github.com/Yaoyi-Li/GCA-Matting.

参数

- **block** (*str*) –Type of residual block. Currently only *BasicBlockDec* is implemented.
- **layers** (*list[int]*) –Number of layers in each block.
- **in_channels** (*int*) –Channel num of input features.
- **kernel_size** (*int*) –Kernel size of the conv layers in the decoder.
- **conv_cfg** (*dict*) –dictionary to construct convolution layer. If it is None, 2d convolution will be applied. Default: None.
- **norm_cfg** (*dict*) –Config dict for normalization layer. "BN" by default.
- **act_cfg** (*dict*) –Config dict for activation layer, "ReLU" by default.
- **with_spectral_norm** (*bool*) –Whether use spectral norm after conv. Default: False.
- **late_downsample** (*bool*) –Whether to adopt late downsample strategy, Default: False.

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

init_weights()

Init weights for the module.

class mmedit.models.backbones.**ResNetEnc**(*block*, *layers*, *in_channels*, *conv_cfg=None*, *norm_cfg={'type': 'BN'}*, *act_cfg={'type': 'ReLU'}*, *with_spectral_norm=False*, *late_downsample=False*)

ResNet encoder for image matting.

This class is adopted from https://github.com/Yaoyi-Li/GCA-Matting. Implement and pre-train on ImageNet with the tricks from https://arxiv.org/abs/1812.01187 without the mix-up part.

参数

- **block** (*str*) –Type of residual block. Currently only *BasicBlock* is implemented.

- **layers** (*list[int]*) –Number of layers in each block.
- **in_channels** (*int*) –Number of input channels.
- **conv_cfg** (*dict*) –dictionary to construct convolution layer. If it is None, 2d convolution will be applied. Default: None.
- **norm_cfg** (*dict*) –Config dict for normalization layer. "BN" by default.
- **act_cfg** (*dict*) –Config dict for activation layer, "ReLU" by default.
- **with_spectral_norm** (*bool*) –Whether use spectral norm after conv. Default: False.
- **late_downsample** (*bool*) –Whether to adopt late downsample strategy, Default: False.

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 mmedit.models.backbones.**ResShortcutDec**(*block*, *layers*, *in_channels*, *kernel_size=3*, *conv_cfg=None*, *norm_cfg={'type': 'BN'}*, *act_cfg={'inplace': True, 'negative_slope': 0.2, 'type': 'LeakyReLU'}*, *with_spectral_norm=False*, *late_downsample=False*)

ResNet decoder for image matting with shortcut connection.

```
feat1 --------------------------- conv2 --- out
                                |
feat2 ---------------------- conv1
                           |
feat3 ----------------- layer4
                     |
feat4 ------------ layer3
                |
feat5 ------- layer2
           |
out ---  layer1
```

参数

- **block** (*str*) –Type of residual block. Currently only *BasicBlockDec* is implemented.
- **layers** (*list[int]*) –Number of layers in each block.
- **in_channels** (*int*) –Channel number of input features.
- **kernel_size** (*int*) –Kernel size of the conv layers in the decoder.

- **conv_cfg** (*dict*) –Dictionary to construct convolution layer. If it is None, 2d convolution will be applied. Default: None.
- **norm_cfg** (*dict*) –Config dict for normalization layer. "BN" by default.
- **act_cfg** (*dict*) –Config dict for activation layer, "ReLU" by default.
- **late_downsample** (*bool*) –Whether to adopt late downsample strategy, Default: False.

forward(*inputs*)

Forward function of resnet shortcut decoder.

参数 **inputs** (*dict*) –Output dictionary of the ResNetEnc containing:

- out (Tensor): Output of the ResNetEnc.
- feat1 (Tensor): Shortcut connection from input image.
- feat2 (Tensor): Shortcut connection from conv2 of ResNetEnc.
- feat3 (Tensor): Shortcut connection from layer1 of ResNetEnc.
- feat4 (Tensor): Shortcut connection from layer2 of ResNetEnc.
- feat5 (Tensor): Shortcut connection from layer3 of ResNetEnc.

返回 Output tensor.

返回类型 Tensor

class mmedit.models.backbones.**ResShortcutEnc**(*block*, *layers*, *in_channels*, *conv_cfg=None*, *norm_cfg={'type': 'BN'}*, *act_cfg={'type': 'ReLU'}*, *with_spectral_norm=False*, *late_downsample=False*, *order=('conv', 'act', 'norm')*)

ResNet backbone for image matting with shortcut connection.

```
image ---------------- shortcut[0] --- feat1
  |
conv1-conv2 ---------- shortcut[1] --- feat2
       |
      conv3-layer1 --- shortcut[2] --- feat3
              |
             layer2 -- shortcut[4] --- feat4
               |
              layer3 - shortcut[5] --- feat5
                |
               layer4 ---------------- out
```

Baseline model of Natural Image Matting via Guided Contextual Attention https://arxiv.org/pdf/2001.04069.pdf.

参数

- **block** (*str*) –Type of residual block. Currently only *BasicBlock* is implemented.
- **layers** (*list[int]*) –Number of layers in each block.
- **in_channels** (*int*) –Number of input channels.
- **conv_cfg** (*dict*) –Dictionary to construct convolution layer. If it is None, 2d convolution will be applied. Default: None.
- **norm_cfg** (*dict*) –Config dict for normalization layer. "BN" by default.
- **act_cfg** (*dict*) –Config dict for activation layer, "ReLU" by default.
- **with_spectral_norm** (*bool*) –Whether use spectral norm after conv. Default: False.
- **late_downsample** (*bool*) –Whether to adopt late downsample strategy. Default: False.
- **order** (*tuple[str]*) –Order of *conv*, *norm* and *act* layer in shortcut convolution module. Default: ('conv', 'act', 'norm').

forward(*x*)

Forward function.

参数 **x** (*Tensor*) –Input tensor with shape (N, C, H, W).

返回 Contains the output tensor and shortcut feature.

返回类型 dict

class mmedit.models.backbones.**ResnetGenerator**(*in_channels*, *out_channels*, *base_channels=64*, *norm_cfg={'type': 'IN'}*, *use_dropout=False*, *num_blocks=9*, *padding_mode='reflect'*, *init_cfg={'gain': 0.02, 'type': 'normal'}*)

Construct a Resnet-based generator that consists of residual blocks between a few downsampling/upsampling operations.

参数

- **in_channels** (*int*) –Number of channels in input images.
- **out_channels** (*int*) –Number of channels in output images.
- **base_channels** (*int*) –Number of filters at the last conv layer. Default: 64.
- **norm_cfg** (*dict*) –Config dict to build norm layer. Default: *dict(type='IN')*.
- **use_dropout** (*bool*) –Whether to use dropout layers. Default: False.
- **num_blocks** (*int*) –Number of residual blocks. Default: 9.
- **padding_mode** (*str*) –The name of padding layer in conv layers: 'reflect' | 'replicate' | 'zeros'. Default: 'reflect'.

- **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*)

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(*pretrained=None*, *strict=True*)

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.
- **strict** (*bool, optional*) –Whether to allow different params for the model and checkpoint. Default: True.

class mmedit.models.backbones.**SRCNN**(*channels=(3, 64, 32, 3)*, *kernel_sizes=(9, 1, 5)*, *upscale_factor=4*)

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

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 mmedit.models.backbones.**SimpleEncoderDecoder**(*encoder*, *decoder*)

Simple encoder-decoder model from matting.

参数

- **encoder** (*dict*) –Config of the encoder.
- **decoder** (*dict*) –Config of the decoder.

forward(**args*, ***kwargs*)

Forward function.

返回 The output tensor of the decoder.

返回类型 Tensor

class mmedit.models.backbones.**TDANNet**(*in_channels=3*, *mid_channels=64*, *out_channels=3*, *num_blocks_before_align=5*, *num_blocks_after_align=10*)

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_before_align** –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]

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: None.
- **strict** (*boo, optional*) –Whether strictly load the pretrained model. Defaults to True.

class mmedit.models.backbones.**TOFlow** (*adapt_official_weights=False*)

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

denormalize (*img*)

Denormalize the output image.

参数 **img** (*Tensor*) –Output image.

返回 Denormalized image.

返回类型 Tensor

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

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.

normalize(*img*)

Normalize the input image.

参数 **img**(*Tensor*) –Input image.

返回 Normalized image.

返回类型 Tensor

class mmedit.models.backbones.**TTSRNet**(*in_channels*, *out_channels*, *mid_channels=64*, *texture_channels=64*, *num_blocks=(16, 16, 8, 4)*, *res_scale=1.0*)

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
- **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.

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

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 mmedit.models.backbones.**UnetGenerator**(*in_channels*, *out_channels*, *num_down=8*, *base_channels=64*, *norm_cfg={'type': 'BN'}*, *use_dropout=False*, *init_cfg={'gain': 0.02, 'type': 'normal'}*)

Construct the Unet-based generator from the innermost layer to the outermost layer, which is a recursive process.

参数

- **in_channels** (*int*) –Number of channels in input images.
- **out_channels** (*int*) –Number of channels in output images.
- **num_down** (*int*) –Number of downsamplings in Unet. If *num_down* is 8, the image with size 256x256 will become 1x1 at the bottleneck. Default: 8.
- **base_channels** (*int*) –Number of channels at the last conv layer. Default: 64.
- **norm_cfg** (*dict*) –Config dict to build norm layer. Default: *dict(type='BN')*.
- **use_dropout** (*bool*) –Whether to use dropout layers. Default: False.
- **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*)

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(*pretrained=None*, *strict=True*)

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.
- **strict** (*bool, optional*) –Whether to allow different params for the model and checkpoint. Default: True.

class mmedit.models.backbones.**VGG16**(*in_channels*, *batch_norm=False*, *aspp=False*, *dilations=None*)

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.

forward(*x*)

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

22.4 components

class mmedit.models.components.**DeepFillRefiner**(*encoder_attention={'encoder_type': 'stage2_attention', 'type': 'DeepFillEncoder'}, encoder_conv={'encoder_type': 'stage2_conv', 'type': 'DeepFillEncoder'}, dilation_neck={'act_cfg': {'type': 'ELU'}, 'in_channels': 128, 'type': 'GLDilationNeck'}, contextual_attention={'in_channels': 128, 'type': 'ContextualAttentionNeck'}, decoder={'in_channels': 256, 'type': 'DeepFillDecoder'}*)

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 mmedit.models.components.**DeepFillv1Discriminators** (*global_disc_cfg*, *local_disc_cfg*)

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 (*pretrained=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 mmedit.models.components.**GLDiscs** (*global_disc_cfg*, *local_disc_cfg*)

Discriminators in Global&Local

This discriminator contains a local discriminator and a global discriminator as described in the original paper: Globally and locally Consistent Image Completion

参数

- **global_disc_cfg** (*dict*) –Config dict to build global discriminator.
- **local_disc_cfg** (*dict*) –Config dict to build 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(*pretrained=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 mmedit.models.components.**ModifiedVGG**(*in_channels*, *mid_channels*)

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

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 mmedit.models.components.**MultiLayerDiscriminator**(*in_channels*, *max_channels*, *num_convs=5*, *fc_in_channels=None*, *fc_out_channels=1024*, *kernel_size=5*, *conv_cfg=None*, *norm_cfg=None*, *act_cfg={'type': 'ReLU'}*, *out_act_cfg={'type': 'ReLU'}*, *with_input_norm=True*, *with_out_convs=False*, *with_spectral_norm=False*, ***kwargs*)

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).
- **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.
- **fc_out_channels** (*int*) –Output dimension of the fully connected layer.
- **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*)

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=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 mmedit.models.components.**PatchDiscriminator**(*in_channels*, *base_channels=64*, *num_conv=3*, *norm_cfg={'type': 'BN'}*, *init_cfg={'gain': 0.02, 'type': 'normal'}*)

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*)

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(*pretrained=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 mmedit.models.components.**PlainRefiner**(*conv_channels=64*, *pretrained=None*)

Simple refiner from Deep Image Matting.

参数

- **conv_channels** (*int*) –Number of channels produced by the three main convolutional layer.
- **loss_refine** (*dict*) –Config of the loss of the refiner. Default: None.
- **pretrained** (*str*) –Name of pretrained model. Default: None.

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 mmedit.models.components.**StyleGAN2Discriminator**(*in_size*, *channel_multiplier=2*, *blur_kernel=[1, 3, 3, 1]*, *mbstd_cfg={'channel_groups': 1, 'group_size': 4}*, *pretrained=None*, *bgr2rgb=False*)

StyleGAN2 Discriminator.

This module comes from MMGeneration. In the future, this code will be removed and StyleGANv2 will be directly imported from mmgeneration.

The architecture of this discriminator is proposed in StyleGAN2. 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:

- styelgan2-ffhq-config-f: http://download.openmmlab.com/mmgen/stylegan2/official_weights/stylegan2-ffhq-config-f-official_20210327_171224-bce9310c.pth # noqa
- stylegan2-horse-config-f: http://download.openmmlab.com/mmgen/stylegan2/official_weights/stylegan2-horse-config-f-official_20210327_173203-ef3e69ca.pth # noqa
- stylegan2-car-config-f: http://download.openmmlab.com/mmgen/stylegan2/official_weights/stylegan2-car-config-f-official_20210327_172340-8cfe053c.pth # noqa
- styelgan2-cat-config-f: http://download.openmmlab.com/mmgen/stylegan2/official_weights/stylegan2-cat-config-f-official_20210327_172444-15bc485b.pth # noqa
- stylegan2-church-config-f: http://download.openmmlab.com/mmgen/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
discriminator = StyleGAN2Discriminator(1024, 512,
                                       pretrained=dict(
                                           ckpt_path=ckpt_http,
                                           prefix='discriminator'))
```

Of course, you can also download the checkpoint in advance and set `ckpt_path` with local path.

Note that our implementation adopts BGR image as input, while the original StyleGAN2 provides RGB images to the discriminator. Thus, we provide `bgr2rgb` argument to convert the image space. If your images follow the RGB order, please set it to `True` accordingly.

参数

- **in_size** (*int*) –The input size of images.
- **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].
- **mbstd_cfg** (*dict, optional*) –Configs for minibatch-stddev layer. Defaults to dict(group_size=4, channel_groups=1).
- **pretrained** (*dict | None, optional*) –Information for pretained models. The necessary key is 'ckpt_path'. Besides, you can also provide 'prefix' to load the generator part from the whole state dict. Defaults to None.
- **bgr2rgb** (*bool, optional*) –Whether to flip the image channel dimension. Defaults to False.

forward(*x*)

Forward function.

参数 **x** (*torch.Tensor*) –Input image tensor.

返回 Predict score for the input image.

返回类型 torch.Tensor

class mmedit.models.components.**StyleGANv2Generator**(*out_size*, *style_channels*, *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*, *pretrained=None*, *bgr2rgb=False*)

StyleGAN2 Generator.

This module comes from MMGeneration. In the future, this code will be removed and StyleGANv2 will be directly imported from mmgeneration.

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:

- styelgan2-ffhq-config-f: http://download.openmmlab.com/mmgen/stylegan2/official_weights/stylegan2-ffhq-config-f-official_20210327_171224-bce9310c.pth # noqa
- stylegan2-horse-config-f: http://download.openmmlab.com/mmgen/stylegan2/official_weights/stylegan2-horse-config-f-official_20210327_173203-ef3e69ca.pth # noqa
- stylegan2-car-config-f: http://download.openmmlab.com/mmgen/stylegan2/official_weights/stylegan2-car-config-f-official_20210327_172340-8cfe053c.pth # noqa
- styelgan2-cat-config-f: http://download.openmmlab.com/mmgen/stylegan2/official_weights/stylegan2-cat-config-f-official_20210327_172444-15bc485b.pth # noqa
- stylegan2-church-config-f: http://download.openmmlab.com/mmgen/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

- **out_size** (*int*) –The output size of the StyleGAN2 generator.
- **style_channels** (*int*) –The number of channels for style code.
- **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 defaultly adopt mixing style mode. 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.
- **pretrained** (*dict | None, optional*) –Information for pretained models. The necessary key is ‘ckpt_path’. Besides, you can also provide ‘prefix’ to load the generator part from the whole state dict. Defaults to None.
- **bgr2rgb** (*bool, optional*) –Whether to flip the image channel dimension. Defaults to False.

forward (*styles*, *num_batches=- 1*, *return_noise=False*, *return_latents=False*, *inject_index=None*, *truncation=1*, *truncation_latent=None*, *input_is_latent=False*, *injected_noise=None*, *randomize_noise=True*)

Forward function.

This function has been integrated with the truncation trick. Please refer to the usage of *truncation* and *truncation_latent*.

参数

- **styles** (*torch.Tensor | list[torch.Tensor] | callable | None*) –In StyleGAN2, you can provide noise tensor or latent tensor. Given a list containing more than one noise or latent tensors, style mixing trick will be used in training. Of course, 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.
- **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.
- **inject_index** (*int | None, optional*) –The index number for mixing style codes. Defaults to None.
- **truncation** (*float, optional*) –Truncation factor. Give value less than 1., the truncation trick will be adopted. Defaults to 1.
- **truncation_latent** (*torch.Tensor, optional*) –Mean truncation latent. Defaults to None.
- **input_is_latent** (*bool, optional*) –If *True*, the input tensor is the latent tensor. Defaults to False.

- **injected_noise** (*torch.Tensor | None, optional*) –Given a tensor, the random noise will be fixed as this input injected noise. Defaults to None.
- **randomize_noise** (*bool, optional*) –If *False*, images are sampled with the buffered noise tensor injected to the style conv block. Defaults to True.

返回

Generated image tensor or dictionary containing more data.

返回类型 torch.Tensor | dict

train(*mode=True*)

Sets the module in training mode.

This has any effect only on certain modules. See documentations of particular modules for details of their behaviors in training/evaluation mode, if they are affected, e.g. `Dropout`, `BatchNorm`, etc.

参数 **mode** (*bool*) –whether to set training mode (`True`) or evaluation mode (`False`). Default: `True`.

返回 self

返回类型 Module

class mmedit.models.components.**UNetDiscriminatorWithSpectralNorm**(*in_channels, mid_channels=64, skip_connection=True*)

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

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.

22.5 losses

class mmedit.models.losses.**CharbonnierCompLoss**(*loss_weight=1.0*, *reduction='mean'*, *sample_wise=False*, *eps=1e-12*)

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*, *fg*, *bg*, *ori_merged*, *weight=None*, ***kwargs*)

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 mmedit.models.losses.**CharbonnierLoss**(*loss_weight=1.0*, *reduction='mean'*, *sample_wise=False*, *eps=1e-12*)

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*, *target*, *weight=None*, ***kwargs*)

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 mmedit.models.losses.**DiscShiftLoss**(*loss_weight=0.1*)

Disc shift loss.

参数 **loss_weight** (*float, optional*) –Loss weight. Defaults to 1.0.

forward(*x*)

Forward function.

参数 **x** (*Tensor*) –Tensor with shape (n, c, h, w)

返回 Loss.

返回类型 Tensor

class mmedit.models.losses.**GANLoss**(*gan_type*, *real_label_val=1.0*, *fake_label_val=0.0*, *loss_weight=1.0*)

Define GAN loss.

参数

- **gan_type** (*str*) –Support 'vanilla', 'lsgan', 'wgan', 'hinge'.
- **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.

forward(*input*, *target_is_real*, *is_disc=False*)

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

get_target_label(*input*, *target_is_real*)

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)

class mmedit.models.losses.**GradientLoss**(*loss_weight=1.0*, *reduction='mean'*)

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*, *target*, *weight=None*)

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 mmedit.models.losses.**GradientPenaltyLoss**(*loss_weight=1.0*)

Gradient penalty loss for wgan-gp.

参数 **loss_weight** (*float*) –Loss weight. Default: 1.0.

forward(*discriminator*, *real_data*, *fake_data*, *mask=None*)

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

class mmedit.models.losses.**L1CompositionLoss**(*loss_weight=1.0*, *reduction='mean'*, *sample_wise=False*)

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*, *fg*, *bg*, *ori_merged*, *weight=None*, ***kwargs*)

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 mmedit.models.losses.**L1Loss**(*loss_weight=1.0*, *reduction='mean'*, *sample_wise=False*)

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*, *target*, *weight=None*, ***kwargs*)

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 mmedit.models.losses.**LightCNNFeatureLoss**(*pretrained*, *loss_weight=1.0*, *criterion='l1'*)

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*, *gt*)

Forward function.

参数

- **pred** (*Tensor*) –Predicted tensor.
- **gt** (*Tensor*) –GT tensor.

返回 Forward results.

返回类型 Tensor

class mmedit.models.losses.**MSECompositionLoss**(*loss_weight=1.0*, *reduction='mean'*, *sample_wise=False*)

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*, *fg*, *bg*, *ori_merged*, *weight=None*, ***kwargs*)

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 mmedit.models.losses.**MSELoss**(*loss_weight=1.0*, *reduction='mean'*, *sample_wise=False*)

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*, *target*, *weight=None*, ***kwargs*)

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 mmedit.models.losses.**MaskedTVLoss**(*loss_weight=1.0*)

Masked TV loss.

参数 **loss_weight** (*float, optional*) –Loss weight. Defaults to 1.0.

forward(*pred*, *mask=None*)

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 mmedit.models.losses.**PerceptualLoss**(*layer_weights*, *vgg_type='vgg19'*, *use_input_norm=True*, *perceptual_weight=1.0*, *style_weight=1.0*, *norm_img=True*, *pretrained='torchvision://vgg19'*, *criterion='l1'*)

Perceptual loss with commonly used style loss.

参数

- **layers_weights** (*dict*) –The weight for each layer of vgg feature. 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.
- **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*, *gt*)

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

class mmedit.models.losses.**PerceptualVGG**(*layer_name_list*, *vgg_type='vgg19'*, *use_input_norm=True*, *pretrained='torchvision://vgg19'*)

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*)

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*, *pretrained*)

Init weights.

参数

- **model** (*nn.Module*) –Models to be inited.
- **pretrained** (*str*) –Path for pretrained weights.

class mmedit.models.losses.**TransferalPerceptualLoss**(*loss_weight=1.0*, *use_attention=True*, *criterion='mse'*)

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: ‘l1’.

forward(*maps*, *soft_attention*, *textures*)

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

`mmedit.models.losses.`**`mask_reduce_loss`**(*loss*, *weight=None*, *reduction='mean'*, *sample_wise=False*)

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

`mmedit.models.losses.`**`reduce_loss`**(*loss*, *reduction*)

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

CHAPTER 23

mmedit.utils

mmedit.utils.**get_root_logger**(*log_file=None*, *log_level=20*)

Get the root logger.

The logger will be initialized if it has not been initialized. By default a StreamHandler will be added. If *log_file* is specified, a FileHandler will also be added. The name of the root logger is the top-level package name, e.g., "mmedit".

参数

- **log_file** (*str | None*) –The log filename. If specified, a FileHandler will be added to the root logger.
- **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.

返回 The root logger.

返回类型 logging.Logger

CHAPTER 24

English

CHAPTER 25

简体中文

CHAPTER 26

Indices and tables

- genindex
- modindex
- search

Python 模块索引

m

索引

A

B

C

D

E

F

G

H

I

L

M

N

O

P

Q

R

S

T

U

V

W

⍰

模块