# MMEditing

MMEditing Authors

Nov 01, 2022

# GET STARTED

Languages: English |

MMEditing is an open-source toolbox for low-level vision. It supports various tasks, including:

- Image super-resolution
- Video super-resolution
- Video frame interpolation
- Image inpainting
- Image matting
- Image generation
- and possibly more in the future.

It is based on PyTorch and is a part of the OpenMMLab project. Codes are available on GitHub.

CHAPTER

ONE

# DOCUMENTATION

## 1.1 Installation

We highly recommend developers follow our best practices to install MMEditing. However, the whole process is highly customizable. See *Customize Installation* section for more information.

### 1.1.1 Best Practices

The following steps work on Linux, Windows, and macOS. If you have already set up a PyTorch environment, no matter using conda or pip, you can start from **step 3**.

**Step 0.** Download and install Miniconda from official website.

**Step 1.** Create a conda environment and activate it

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

**Step 2.** Install PyTorch following official instructions, e.g.

- On GPU platforms:

  ```
  conda install pytorch=1.10 torchvision cudatoolkit=11.3 -c pytorch
  ```

- On CPU platforms:

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

**Step 3.** Install pre-built MMCV using MIM.

```
pip3 install openmim
mim install mmcv-full==1.5.0
```

**Step 4.** Install MMEditing from the source code.

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

**Step 5.** Verification.

```
cd ~
python -c "import mmedit; print(mmedit.__version__)"
# Example output: 0.14.0
```

The installation is successful if the version number is output correctly.

### 1.1.2 Customize Installation

#### Version of Dependencies

You may change the version of Python, PyTorch, and MMCV by changing the version numbers in **Step 1, 2, and 3**, respectively.

Currently, MMEditing works with

- Python >= 3.6
- PyTorch >= 1.5
- MMCV >= 1.3.13

#### Version of CUDA

When installing PyTorch in **Step 2**, you need to specify the version of CUDA. If you are not clear on which to choose, follow our recommendations:

1. For Ampere-based NVIDIA GPUs, such as GeForce 30 series and NVIDIA A100, **CUDA 11 is a must**.
2. For older NVIDIA GPUs, CUDA 11 is backward compatible, but CUDA 10.2 offers better compatibility and is more lightweight.

Please also make sure the GPU driver satisfies the minimum version requirements. See this table for more information.

Please **Note** that there is no need to install the complete CUDA toolkit if you follow our *best practices* because no CUDA code will be compiled. However, if you hope to compile MMCV or other C++/CUDA operators, you need to install the complete CUDA toolkit from NVIDIA's website, and **its version should match the CUDA version of PyTorch**, which is the version of `cudatoolkit` in `conda install`.

#### Install without Conda

Though we highly recommend using conda to create environments and install PyTorch, it is viable to install PyTorch only with pip, for example, with the following command,

```
pip3 install torch torchvision
```

However, an `--extra-index-url` or `-f` option is usually required to specify the CPU / CUDA version. See PyTorch website for more details.

### Install without MIM

MMCV contains C++ and CUDA extensions, thus depending on PyTorch in a complex way. MIM solves such dependency automatically and makes installation easier. However, it is not a must.

To install MMCV with `pip` instead of MIM, please follow MMCV installation guides. This requires manually specifying a *find-url* based on PyTorch version and its CUDA version.

For example, the following command install `mmcv-full` built for PyTorch 1.10.x and CUDA 11.3.

```
pip install mmcv-full -f https://download.openmmlab.com/mmcv/dist/cu113/torch1.10.0/
↪index.html
```

### On macOS

Pre-built MMCV package is not available for macOS, so you have to build MMCV from the source. Pip will build it automatically during installation, but it requires a C++ compiler. A simple solution is to install Clang with `xcode-select --install`.

Under such circumstances, MIM is not required and `pip install mmcv-full -v` can do the job. See MMCV installation guides for more details.

### On Google Colab

Online machine learning platform such as Google Colab usually has PyTorch installed. Thus we only need to install MMCV and MMEditing with the following commands.

**Step 1.** Install pre-built MMCV using MIM.

```
!pip3 install openmim
!mim install mmcv-full
```

**Step 2.** Install MMEditing from the source.

```
!git clone https://github.com/open-mmlab/mmediting.git
%cd mmediting
!pip3 install -e .
```

**Step 3.** Verification.

```
import mmedit
print(mmedit.__version__)
# Example output: 0.13.0
```

**Note**: within Jupyter, the exclamation mark `!` is used to call external executables and `%cd` is a IPython magic command to change the current working directory of Python.

### 1.1.3 Additional Notes

#### Speed up Installation with Mirror

One can configure conda and pip mirrors to speed up the installation. This step is very practical for users geologically in China.

See the below links (in Chinese) for details

- https://mirrors.tuna.tsinghua.edu.cn/help/pypi/
- https://mirror.tuna.tsinghua.edu.cn/help/anaconda/
- https://developer.aliyun.com/mirror/pypi

#### On `-e .`

You may be curious about what `-e .` means when supplied with `pip install`. Here is the description:

- `-e` means editable mode. When `import mmedit`, modules under the cloned directory are imported. If `pip install` without `-e`, pip will copy cloned codes to somewhere like `lib/python/site-package`. Consequently, modified code under the cloned directory takes no effect unless `pip install` again. This is particularly convenient for developers. If some codes are modified, new codes will be imported next time without reinstallation.
- `.` means code in this directory

You can also use `pip -e .[all]`, which will install more dependencies, especially for pre-commit hooks and unittests.

## 1.2 Demo

We provide some task-specific demo scripts to test a single image.

### 1.2.1 Inpainting

You can use the following commands to test a pair of images for inpainting.

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

If `--imshow` is specified, the demo will also show image with opencv. Examples:

```
python demo/inpainting_demo.py \
    configs/inpainting/global_local/gl_256x256_8x12_celeba.py \
    https://download.openmmlab.com/mmediting/inpainting/global_local/gl_256x256_8x12_
↪celeba_20200619-5af0493f.pth \
    tests/data/image/celeba_test.png \
```

(continues on next page)

(continued from previous page)

```
    tests/data/image/bbox_mask.png \
    tests/data/pred/inpainting_celeba.png
```

The predicted inpainting result will be save in `tests/data/pred/inpainting_celeba.png`.

### 1.2.2 Matting

You can use the following commands to test a pair of images and trimap.

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

If `--imshow` is specified, the demo will also show image with opencv. Examples:

```
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
```

The predicted alpha matte will be save in `tests/data/pred/GT05.png`.

### 1.2.3 Restoration (Image)

You can use the following commands to test an image for restoration.

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

If `--imshow` is specified, the demo will also show image with opencv. Examples:

```
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
```

You can test Ref-SR by providing `--ref-path`. Examples:

```
python demo/restoration_demo.py \
    configs/restorers/ttsr/ttsr-gan_x4_c64b16_g1_500k_CUFED.py \
    https://download.openmmlab.com/mmediting/restorers/ttsr/ttsr-gan_x4_c64b16_g1_500k_
↪CUFED_20210626-2ab28ca0.pth \
    tests/data/test_multiple_gt/sequence_1/00000000.png \
    work_dirs/demo_out.png \
    --ref-path tests/data/test_multiple_gt/sequence_1/00000001.png
```

### 1.2.4 Restoration (Face Image)

You can use the following commands to test an face image for restoration.

```
python demo/restoration_face_demo.py \
    ${CONFIG_FILE} \
    ${CHECKPOINT_FILE} \
    ${IMAGE_FILE} \
    ${SAVE_FILE} \
    [--upscale-factor] \
    [--face-size] \
    [--imshow] \
    [--device ${GPU_ID}]
```

If `--imshow` is specified, the demo will also show image with opencv. Examples:

```
python demo/restoration_face_demo.py \
    configs/restorers/glean/glean_in128out1024_4x2_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
```

### 1.2.5 Restoration (Video)

You can use the following commands to test a video for restoration.

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

It supports both the sliding-window framework and the recurrent framework. Examples:

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 \
```

(continues on next page)

(continued from previous page)

```
    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
```

The restored video will be save in `output/`.

### 1.2.6 video frame interpolation

You can use the following commands to test a video for frame interpolation.

```
python demo/video_interpolation_demo.py \
    ${CONFIG_FILE} \
    ${CHECKPOINT_FILE} \
    ${INPUT_DIR} \
    ${OUTPUT_DIR} \
    [--fps-multiplier ${FPS_MULTIPLIER}] \
    [--fps ${FPS}]
```

`${INPUT_DIR}` / `${OUTPUT_DIR}` can be a path of video file or the folder of a sequence of ordered images. If `${OUTPUT_DIR}` is a path of video file, its frame rate can be determined by the frame rate of input video and `fps_multiplier`, or be determined by `fps` directly (the former has higher priority). Examples:

The frame rate of output video is determined by the frame rate of input video and `fps_multiplier`

```
python demo/video_interpolation_demo.py \
    configs/video_interpolators/cain/cain_b5_320k_vimeo-triplet.py \
    https://download.openmmlab.com/mmediting/video_interpolators/cain/cain_b5_320k_vimeo-
↪triple_20220117-647f3de2.pth \
    tests/data/test_inference.mp4 \
    tests/data/test_inference_vfi_out.mp4 \
    --fps-multiplier 2.0
```

The frame rate of output video is determined by `fps`:

```
python demo/video_interpolation_demo.py \
    configs/video_interpolators/cain/cain_b5_320k_vimeo-triplet.py \
    https://download.openmmlab.com/mmediting/video_interpolators/cain/cain_b5_320k_vimeo-
↪triple_20220117-647f3de2.pth \
    tests/data/test_inference.mp4 \
    tests/data/test_inference_vfi_out.mp4 \
    --fps 60.0
```

### 1.2.7 Generation

```
python demo/generation_demo.py \
    ${CONFIG_FILE} \
    ${CHECKPOINT_FILE} \
    ${IMAGE_FILE} \
    ${SAVE_FILE} \
    [--unpaired-path ${UNPAIRED_IMAGE_FILE}] \
    [--imshow] \
    [--device ${GPU_ID}]
```

If `--unpaired-path` is specified (used for CycleGAN), the model will perform unpaired image-to-image translation. If `--imshow` is specified, the demo will also show image with opencv. Examples:

Paired:

```
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 (also show image with opencv):

```
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
```

## 1.3 Inference with pre-trained models

We provide testing scripts to evaluate pre-trained models on a whole dataset, as well as some task-specific image demos.

### 1.3.1 Test a pre-trained model

MMEditing implements **distributed** testing with `MMDistributedDataParallel`.

#### Test with single/multiple GPUs

You can use the following commands to test a pre-trained model with single/multiple GPUs.

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

# multi-gpu testing
./tools/dist_test.sh ${CONFIG_FILE} ${CHECKPOINT_FILE} ${GPU_NUM} [--out ${RESULT_FILE}]␣
↪[--save-path ${IMAGE_SAVE_PATH}]
```

For example,

```
# single-gpu testing
python tools/test.py configs/example_config.py work_dirs/example_exp/example_model_
↪20200202.pth --out work_dirs/example_exp/results.pkl

# multi-gpu testing
./tools/dist_test.sh configs/example_config.py work_dirs/example_exp/example_model_
↪20200202.pth --save-path work_dirs/example_exp/results/
```

### Test with Slurm

If you run MMEditing on a cluster managed with slurm, you can use the script `slurm_test.sh`. (This script also supports single machine testing.)

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

Here is an example of using 8 GPUs to test an example model on the 'dev' partition with job name 'test'.

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

You can check slurm_test.sh for full arguments and environment variables.

### Optional arguments

- `--out`: Specify the filename of the output results in pickle format. If not given, the results will not be saved to a file.
- `--save-path`: Specify the path to store edited images. If not given, the images will not be saved.
- `--seed`: Random seed during testing. This argument is used for fixed results in some tasks such as inpainting.
- `--deterministic`: Related to `--seed`, this argument decides whether to set deterministic options for CUDNN backend. If specified, it will set `torch.backends.cudnn.deterministic` to True and `torch.backends.cudnn.benchmark` to False.
- `--cfg-options`: If specified, the key-value pair optional cfg will be merged into config file.

Note: Currently, we do NOT use `--eval` argument like MMDetection to specify evaluation metrics. The evaluation metrics are given in the config files (see config.md).

## 1.4 Train a model

MMEditing implements **distributed** training with `MMDistributedDataParallel`.

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

By default we evaluate the model on the validation set after several iterations, you can change the evaluation interval by adding the interval argument in the training config.

```
evaluation = dict(interval=1e4, by_epoch=False)  # This evaluates the model per 1e4␣
↪iterations.
```

(continues on next page)

(continued from previous page)

### 1.4.1 Train with single/multiple GPUs

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

Optional arguments are:

- `--no-validate` (**not suggested**): By default, the codebase will perform evaluation every k iterations during the training. To disable this behavior, use `--no-validate`.
- `--work-dir ${WORK_DIR}`: Override the working directory specified in the config file.
- `--resume-from ${CHECKPOINT_FILE}`: Resume from a previous checkpoint file.
- `--cfg-options`: If specified, the key-value pair optional cfg will be merged into config file.

Difference between `resume-from` and `load-from`: `resume-from` loads both the model weights and optimizer status, and the iteration is also inherited from the specified checkpoint. It is usually used for resuming the training process that is interrupted accidentally. `load-from` only loads the model weights and the training iteration starts from 0. It is usually used for fine-tuning.

#### Train with multiple nodes

To launch distributed training on multiple machines, which can be accessed via IPs, run following commands:

On the first machine:

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

On the second machine:

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

To speed up network communication, high speed network hardware, such as Infiniband, is recommended. Please refer to PyTorch docs for more information.

### 1.4.2 Train with Slurm

If you run MMEditing on a cluster managed with slurm, you can use the script `slurm_train.sh`. (This script also supports single machine training.)

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

Here is an example of using 8 GPUs to train an inpainting model on the dev partition.

```
GPUS=8 ./tools/slurm_train.sh dev configs/inpainting/gl_places.py /nfs/xxxx/gl_places_256
```

You can check slurm_train.sh for full arguments and environment variables.

### 1.4.3 Launch multiple jobs on a single machine

If you launch multiple jobs on a single machine, e.g., 2 jobs of 4-GPU training on a machine with 8 GPUs, you need to specify different ports (29500 by default) for each job to avoid communication conflict.

If you use `dist_train.sh` to launch training jobs, you can set the port in commands.

```
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
```

If you launch training jobs with Slurm, you need to modify the config files (usually the 6th line from the bottom in config files) to set different communication ports.

In `config1.py`,

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

In `config2.py`,

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

Then you can launch two jobs with `config1.py` ang `config2.py`.

```
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}
```

## 1.5 Learn about Configs

We use python files as our config system. You can find all the provided configs under `$MMEditing/configs`.

### 1.5.1 Config Name Style

We follow the below style to name config files. Contributors are advised to follow the same style.

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

`{xxx}` is required field and `[yyy]` is optional.

- `{model}`: model type like `srcnn`, `dim`, etc.
- `[model setting]`: specific setting for some model, like `resolution` for input images, `stage name` for training, etc.
- `{backbone}`: backbone type like `r50` (ResNet-50), `x101` (ResNeXt-101).
- `[refiner]`: refiner type like `pln` (Plain Refiner).
- `[norm_setting]`: `bn` (Batch Normalization) is used unless specified, other norm layer type could be `gn` (Group Normalization), `syncbn` (Synchronized Batch Normalization).
- `[misc]`: miscellaneous setting/plugins of model, e.g. `dconv`, `gcb`, `attention`, `albu`, `mstrain`.
- `[gpu x batch_per_gpu]`: GPUs and samples per GPU, `8x2` is used by default.

- `{schedule}`: training schedule, `20k`, `100k`, etc. `20k` means 20,000 iterations. `100k` means 100,000 iterations.
- `{dataset}`: dataset like `places` (for inpainting), `comp1k` (for matting), `div2k` (for restoration) and `paired` (for generation).

### 1.5.2 Config System for Generation

Same as MMDetection, we incorporate modular and inheritance design into our config system, which is convenient to conduct various experiments.

#### An Example - pix2pix

To help the users have a basic idea of a complete config and the modules in a generation system, we make brief comments on the config of pix2pix as the following. For more detailed usage and the corresponding alternative for each modules, please refer to the API documentation.

```
## model settings
model = dict(
    type='Pix2Pix',  ## The name of synthesizer
    generator=dict(
        type='UnetGenerator',  ## The name of generator
        in_channels=3,  ## The input channels of generator
        out_channels=3,  ## The output channels of generator
        num_down=8,  ## The umber of downsamplings in the generator
        base_channels=64,  ## The number of channels at the last conv layer of generator
        norm_cfg=dict(type='BN'),  ## The config of norm layer
        use_dropout=True,  ## Whether to use dropout layers in the generator
        init_cfg=dict(type='normal', gain=0.02)),  ## The config of initialization
    discriminator=dict(
        type='PatchDiscriminator',  ## The name of discriminator
        in_channels=6,  ## The input channels of discriminator
        base_channels=64,  ## The number of channels at the first conv layer of␣
↪discriminator
        num_conv=3,  ## The number of stacked intermediate conv layers (excluding input␣
↪and output conv layer) in the discriminator
        norm_cfg=dict(type='BN'),  ## The config of norm layer
        init_cfg=dict(type='normal', gain=0.02)),  ## The config of initialization
    gan_loss=dict(
        type='GANLoss',  ## The name of GAN loss
        gan_type='vanilla',  ## The type of GAN loss
        real_label_val=1.0,  ## The value for real label of GAN loss
        fake_label_val=0.0,  ## The value for fake label of GAN loss
        loss_weight=1.0),  ## The weight of GAN loss
    pixel_loss=dict(type='L1Loss', loss_weight=100.0, reduction='mean'))
## model training and testing settings
train_cfg = dict(
    direction='b2a')  ## Image-to-image translation direction (the model training␣
↪direction, same as testing direction) for pix2pix. Model default: a2b
test_cfg = dict(
    direction='b2a',   ## Image-to-image translation direction (the model training␣
↪direction, same as testing direction) for pix2pix. Model default: a2b
    show_input=True)  ## Whether to show input real images when saving testing images␣
↪for pix2pix
```

(continues on next page)

(continued from previous page)

```
## dataset settings
train_dataset_type = 'GenerationPairedDataset'  ## The type of dataset for training
val_dataset_type = 'GenerationPairedDataset'  ## The type of dataset for validation/
↪testing
img_norm_cfg = dict(mean=[0.5, 0.5, 0.5], std=[0.5, 0.5, 0.5])  ## Image normalization␣
↪config to normalize the input images
train_pipeline = [
    dict(
        type='LoadPairedImageFromFile',  ## Load a pair of images from file path pipeline
        io_backend='disk',  ## IO backend where images are store
        key='pair',  ## Keys to find corresponding path
        flag='color'),  ## Loading flag for images
    dict(
        type='Resize',  ## Resize pipeline
        keys=['img_a', 'img_b'],  ## The keys of images to be resized
        scale=(286, 286),  ## The scale to resize images
        interpolation='bicubic'),  ## Algorithm used for interpolation when resizing␣
↪images
    dict(
        type='FixedCrop',  ## Fixed crop pipeline, cropping paired images to a specific␣
↪size at a specific position for pix2pix training
        keys=['img_a', 'img_b'],  ## The keys of images to be cropped
        crop_size=(256, 256)),  ## The size to crop images
    dict(
        type='Flip',  ## Flip pipeline
        keys=['img_a', 'img_b'],  ## The keys of images to be flipped
        direction='horizontal'),  ## Flip images horizontally or vertically
    dict(
        type='RescaleToZeroOne',  ## Rescale images from [0, 255] to [0, 1]
        keys=['img_a', 'img_b']),  ## The keys of images to be rescaled
    dict(
        type='Normalize',  ## Image normalization pipeline
        keys=['img_a', 'img_b'],  ## The keys of images to be normalized
        to_rgb=True,  ## Whether to convert image channels from BGR to RGB
        **img_norm_cfg),  ## Image normalization config (see above for the definition of␣
↪`img_norm_cfg`)
    dict(
       type='ImageToTensor',  ## Image to tensor pipeline
       keys=['img_a', 'img_b']),  ## The keys of images to be converted from image to␣
↪tensor
    dict(
        type='Collect',  ## Pipeline that decides which keys in the data should be␣
↪passed to the synthesizer
        keys=['img_a', 'img_b'],  ## The keys of images
        meta_keys=['img_a_path', 'img_b_path'])  ## The meta keys of images
]
test_pipeline = [
    dict(
        type='LoadPairedImageFromFile',  ## Load a pair of images from file path pipeline
        io_backend='disk',  ## IO backend where images are store
        key='pair',  ## Keys to find corresponding path
```

(continues on next page)

(continued from previous page)

```
        flag='color'),  ## Loading flag for images
    dict(
        type='Resize',  ## Resize pipeline
        keys=['img_a', 'img_b'],  ## The keys of images to be resized
        scale=(256, 256),  ## The scale to resize images
        interpolation='bicubic'),  ## Algorithm used for interpolation when resizing␣
↪images
    dict(
        type='RescaleToZeroOne',  ## Rescale images from [0, 255] to [0, 1]
        keys=['img_a', 'img_b']),  ## The keys of images to be rescaled
    dict(
        type='Normalize',  ## Image normalization pipeline
        keys=['img_a', 'img_b'],  ## The keys of images to be normalized
        to_rgb=True,  ## Whether to convert image channels from BGR to RGB
        **img_norm_cfg),  ## Image normalization config (see above for the definition of␣
↪`img_norm_cfg`)
    dict(
       type='ImageToTensor',  ## Image to tensor pipeline
       keys=['img_a', 'img_b']),  ## The keys of images to be converted from image to␣
↪tensor
    dict(
        type='Collect',  ## Pipeline that decides which keys in the data should be␣
↪passed to the synthesizer
        keys=['img_a', 'img_b'],  ## The keys of images
        meta_keys=['img_a_path', 'img_b_path'])  ## The meta keys of images
]
data_root = 'data/pix2pix/facades'  ## The root path of data
data = dict(
    samples_per_gpu=1,  ## Batch size of a single GPU
    workers_per_gpu=4,  ## Worker to pre-fetch data for each single GPU
    drop_last=True,  ## Whether to drop out the last batch of data in training
    val_samples_per_gpu=1,  ## Batch size of a single GPU in validation
    val_workers_per_gpu=0,  ## Worker to pre-fetch data for each single GPU in validation
    train=dict(  ## Training dataset config
        type=train_dataset_type,
        dataroot=data_root,
        pipeline=train_pipeline,
        test_mode=False),
    val=dict(  ## Validation dataset config
        type=val_dataset_type,
        dataroot=data_root,
        pipeline=test_pipeline,
        test_mode=True),
    test=dict(  ## Testing dataset config
        type=val_dataset_type,
        dataroot=data_root,
        pipeline=test_pipeline,
        test_mode=True))

## optimizer
optimizers = dict(  ## Config used to build optimizer, support all the optimizers in␣
↪PyTorch whose arguments are also the same as those in PyTorch
```

(continues on next page)

(continued from previous page)

```
    generator=dict(type='Adam', lr=2e-4, betas=(0.5, 0.999)),
    discriminator=dict(type='Adam', lr=2e-4, betas=(0.5, 0.999)))

## learning policy
lr_config = dict(policy='Fixed', by_epoch=False)  ## Learning rate scheduler config used␣
↪to register LrUpdater hook

## checkpoint saving
checkpoint_config = dict(interval=4000, save_optimizer=True, by_epoch=False)  ## Config␣
↪to set the checkpoint hook, Refer to https://github.com/open-mmlab/mmcv/blob/master/
↪mmcv/runner/hooks/checkpoint.py for implementation.
evaluation = dict(  ## The config to build the evaluation hook
    interval=4000,  ## Evaluation interval
    save_image=True)  ## Whether to save images
log_config = dict(  ## config to register logger hook
    interval=100,  ## Interval to print the log
    hooks=[
        dict(type='TextLoggerHook', by_epoch=False),   ## The logger used to record the␣
↪training process
        ## dict(type='TensorboardLoggerHook')  # The Tensorboard logger is also supported
    ])
visual_config = None  ## The config to build the visualization hook

## runtime settings
total_iters = 80000  ## Total iterations to train the model
cudnn_benchmark = True  ## Set cudnn_benchmark
dist_params = dict(backend='nccl')  ## Parameters to setup distributed training, the␣
↪port can also be set
log_level = 'INFO'  ## The level of logging
load_from = None  ## Load models as a pre-trained model from a given path. This will not␣
↪resume training
resume_from = None  ## Resume checkpoints from a given path, the training will be␣
↪resumed from the epoch when the checkpoint's is saved
workflow = [('train', 1)]  ## Workflow for runner. [('train', 1)] means there is only one␣
↪workflow and the workflow named 'train' is executed once. Keep this unchanged when␣
↪training current generation models
exp_name = 'pix2pix_facades'  ## The experiment name
work_dir = f'./work_dirs/{exp_name}'  ## Directory to save the model checkpoints and␣
↪logs for the current experiments.
```

### 1.5.3 Config System for Inpainting

#### Config Name Style

Same as MMDetection, we incorporate modular and inheritance design into our config system, which is convenient to conduct various experiments.

### Config Field Description

To help the users have a basic idea of a complete config and the modules in a inpainting system, we make brief comments on the config of Global&Local as the following. For more detailed usage and the corresponding alternative for each modules, please refer to the API documentation.

```
model = dict(
    type='GLInpaintor', ## The name of inpaintor
    encdec=dict(
        type='GLEncoderDecoder', ## The name of encoder-decoder
        encoder=dict(type='GLEncoder', norm_cfg=dict(type='SyncBN')), ## The config of␣
↪encoder
        decoder=dict(type='GLDecoder', norm_cfg=dict(type='SyncBN')), ## The config of␣
↪decoder
        dilation_neck=dict(
            type='GLDilationNeck', norm_cfg=dict(type='SyncBN'))), ## The config of␣
↪dilation neck
    disc=dict(
        type='GLDiscs', ## The name of discriminator
        global_disc_cfg=dict(
            in_channels=3, ## The input channel of discriminator
            max_channels=512, ## The maximum middle channel in discriminator
            fc_in_channels=512 * 4 * 4, ## The input channel of last fc layer
            fc_out_channels=1024, ## The output channel of last fc channel
            num_convs=6, ## The number of convs used in discriminator
            norm_cfg=dict(type='SyncBN') ## The config of norm layer
        ),
        local_disc_cfg=dict(
            in_channels=3, ## The input channel of discriminator
            max_channels=512, ## The maximum middle channel in discriminator
            fc_in_channels=512 * 4 * 4, ## The input channel of last fc layer
            fc_out_channels=1024, ## The output channel of last fc channel
            num_convs=5, ## The number of convs used in discriminator
            norm_cfg=dict(type='SyncBN') ## The config of norm layer
        ),
    ),
    loss_gan=dict(
        type='GANLoss', ## The name of GAN loss
        gan_type='vanilla', ## The type of GAN loss
        loss_weight=0.001 ## The weight of GAN loss
    ),
    loss_l1_hole=dict(
        type='L1Loss', ## The type of l1 loss
        loss_weight=1.0 ## The weight of l1 loss
    ),
    pretrained=None) ## The path of pretrained weight

train_cfg = dict(
    disc_step=1, ## The steps of training discriminator before training generator
    iter_tc=90000, ## Iterations of warming up generator
    iter_td=100000, ## Iterations of warming up discriminator
    start_iter=0, ## Starting iteration
    local_size=(128, 128)) ## The size of local patches
```

(continues on next page)

(continued from previous page)

```
test_cfg = dict(metrics=['l1']) ## The config of testing scheme

dataset_type = 'ImgInpaintingDataset' ## The type of dataset
input_shape = (256, 256) ## The shape of input image

train_pipeline = [
    dict(type='LoadImageFromFile', key='gt_img'), ## The config of loading image
    dict(
        type='LoadMask', ## The type of loading mask pipeline
        mask_mode='bbox', ## The type of mask
        mask_config=dict(
            max_bbox_shape=(128, 128), ## The shape of bbox
            max_bbox_delta=40, ## The changing delta of bbox height and width
            min_margin=20,  ## The minimum margin from bbox to the image border
            img_shape=input_shape)),  ## The input image shape
    dict(
        type='Crop', ## The type of crop pipeline
        keys=['gt_img'],  ## The keys of images to be cropped
        crop_size=(384, 384),  ## The size of cropped patch
        random_crop=True,  ## Whether to use random crop
    ),
    dict(
        type='Resize',  ## The type of resizing pipeline
        keys=['gt_img'],  ## They keys of images to be resized
        scale=input_shape,  ## The scale of resizing function
        keep_ratio=False,  ## Whether to keep ratio during resizing
    ),
    dict(
        type='Normalize',  ## The type of normalizing pipeline
        keys=['gt_img'],  ## The keys of images to be normed
        mean=[127.5] * 3,  ## Mean value used in normalization
        std=[127.5] * 3,  ## Std value used in normalization
        to_rgb=False),  ## Whether to transfer image channels to rgb
    dict(type='GetMaskedImage'),  ## The config of getting masked image pipeline
    dict(
        type='Collect',  ## The type of collecting data from current pipeline
        keys=['gt_img', 'masked_img', 'mask', 'mask_bbox'],  ## The keys of data to be␣
↪collected
        meta_keys=['gt_img_path']),  ## The meta keys of data to be collected
    dict(type='ImageToTensor', keys=['gt_img', 'masked_img', 'mask']),  ## The config␣
↪dict of image to tensor pipeline
    dict(type='ToTensor', keys=['mask_bbox'])  ## The config dict of ToTensor pipeline
]

test_pipeline = train_pipeline  ## Constructing testing/validation pipeline

data_root = 'data/places365'  ## Set data root

data = dict(
    samples_per_gpu=12,  ## Batch size of a single GPU
    workers_per_gpu=8,  ## Worker to pre-fetch data for each single GPU
    val_samples_per_gpu=1,  ## Batch size of a single GPU in validation
```

(continues on next page)

(continued from previous page)

```
    val_workers_per_gpu=8,  ## Worker to pre-fetch data for each single GPU in validation
    drop_last=True,  ## Whether to drop out the last batch of data
    train=dict(  ## Train dataset config
        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(  ## Validation dataset config
        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(  ## Config used to build optimizer, support all the optimizers in␣
↪PyTorch whose arguments are also the same as those in PyTorch
    generator=dict(type='Adam', lr=0.0004), disc=dict(type='Adam', lr=0.0004))

lr_config = dict(policy='Fixed', by_epoch=False)  ## Learning rate scheduler config used␣
↪to register LrUpdater hook

checkpoint_config = dict(by_epoch=False, interval=50000)  ## Config to set the␣
↪checkpoint hook, Refer to https://github.com/open-mmlab/mmcv/blob/master/mmcv/runner/
↪hooks/checkpoint.py for implementation.
log_config = dict(  ## config to register logger hook
    interval=100,  ## Interval to print the log
    hooks=[
        dict(type='TextLoggerHook', by_epoch=False),
        ## dict(type='TensorboardLoggerHook'),  # The Tensorboard logger is also supported
        ## dict(type='PaviLoggerHook', init_kwargs=dict(project='mmedit'))
    ])  ## The logger used to record the training process.

visual_config = dict(  ## config to register visualization hook
    type='MMEditVisualizationHook',
    output_dir='visual',
    interval=1000,
    res_name_list=[
        'gt_img', 'masked_img', 'fake_res', 'fake_img', 'fake_gt_local'
    ],
)  ## The logger used to visualize the training process.

evaluation = dict(interval=50000)  ## The config to build the evaluation hook

total_iters = 500002
dist_params = dict(backend='nccl')  ## Parameters to setup distributed training, the␣
↪port can also be set.
log_level = 'INFO'  ## The level of logging.
work_dir = None  ## Directory to save the model checkpoints and logs for the current␣
↪experiments.
load_from = None  ## load models as a pre-trained model from a given path. This will not␣
↪resume training.
```

(continues on next page)

(continued from previous page)

```
resume_from = None  ## Resume checkpoints from a given path, the training will be␣
↪resumed from the epoch when the checkpoint's is saved.
workflow = [('train', 10000)]  ## Workflow for runner. [('train', 1)] means there is only␣
↪one workflow and the workflow named 'train' is executed once. The workflow trains the␣
↪model by 12 epochs according to the total_epochs.
exp_name = 'gl_places'  ## The experiment name
find_unused_parameters = False  ## Whether to set find unused parameters in ddp
```

### 1.5.4 Config System for Matting

Same as MMDetection, we incorporate modular and inheritance design into our config system, which is convenient to conduct various experiments.

#### An Example - Deep Image Matting Model

To help the users have a basic idea of a complete config, we make a brief comments on the config of the original DIM model we implemented as the following. For more detailed usage and the corresponding alternative for each modules, please refer to the API documentation.

```
## model settings
model = dict(
    type='DIM',  ## The name of model (we call mattor).
    backbone=dict(  ## The config of the backbone.
        type='SimpleEncoderDecoder',  ## The type of the backbone.
        encoder=dict(  ## The config of the encoder.
            type='VGG16'),  ## The type of the encoder.
        decoder=dict(  ## The config of the decoder.
            type='PlainDecoder')),  ## The type of the decoder.
    pretrained='./weights/vgg_state_dict.pth',  ## The pretrained weight of the encoder␣
↪to be loaded.
    loss_alpha=dict(  ## The config of the alpha loss.
        type='CharbonnierLoss',  ## The type of the loss for predicted alpha matte.
        loss_weight=0.5),  ## The weight of the alpha loss.
    loss_comp=dict(  ## The config of the composition loss.
        type='CharbonnierCompLoss',  ## The type of the composition loss.
        loss_weight=0.5))  ## The weight of the composition loss.
train_cfg = dict(  ## Config of training DIM model.
    train_backbone=True,  ## In DIM stage1, backbone is trained.
    train_refiner=False)  ## In DIM stage1, refiner is not trained.
test_cfg = dict(  ## Config of testing DIM model.
    refine=False,  ## Whether use refiner output as output, in stage1, we don't use it.
    metrics=['SAD', 'MSE', 'GRAD', 'CONN'])  ## The metrics used when testing.

## data settings
dataset_type = 'AdobeComp1kDataset'  ## Dataset type, this will be used to define the␣
↪dataset.
data_root = 'data/adobe_composition-1k'  ## Root path of data.
img_norm_cfg = dict(  ## Image normalization config to normalize the input images.
    mean=[0.485, 0.456, 0.406],  ## Mean values used to pre-training the pre-trained␣
↪backbone models.
```

(continues on next page)

(continued from previous page)

```
    std=[0.229, 0.224, 0.225],  ## Standard variance used to pre-training the pre-
↪trained backbone models.
    to_rgb=True)  ## The channel orders of image used to pre-training the pre-trained␣
↪backbone models.
train_pipeline = [  ## Training data processing pipeline.
    dict(
        type='LoadImageFromFile',  ## Load alpha matte from file.
        key='alpha',  ## Key of alpha matte in annotation file. The pipeline will read␣
↪alpha matte from path `alpha_path`.
        flag='grayscale'),  ## Load as grayscale image which has shape (height, width).
    dict(
        type='LoadImageFromFile',  ## Load image from file.
        key='fg'),  ## Key of image to load. The pipeline will read fg from path `fg_
↪path`.
    dict(
        type='LoadImageFromFile',  ## Load image from file.
        key='bg'),  ## Key of image to load. The pipeline will read bg from path `bg_
↪path`.
    dict(
        type='LoadImageFromFile',  ## Load image from file.
        key='merged'),  ## Key of image to load. The pipeline will read merged from path␣
↪`merged_path`.
    dict(
        type='CropAroundUnknown',  ## Crop images around unknown area (semi-transparent␣
↪area).
        keys=['alpha', 'merged', 'ori_merged', 'fg', 'bg'],  ## Images to crop.
        crop_sizes=[320, 480, 640]),  ## Candidate crop size.
    dict(
        type='Flip',  ## Augmentation pipeline that flips the images.
        keys=['alpha', 'merged', 'ori_merged', 'fg', 'bg']),  ## Images to be flipped.
    dict(
        type='Resize',  ## Augmentation pipeline that resizes the images.
        keys=['alpha', 'merged', 'ori_merged', 'fg', 'bg'],  ## Images to be resized.
        scale=(320, 320),  ## Target size.
        keep_ratio=False),  ## Whether to keep the ratio between height and width.
    dict(
        type='GenerateTrimap',  ## Generate trimap from alpha matte.
        kernel_size=(1, 30)),  ## Kernel size range of the erode/dilate kernel.
    dict(
        type='RescaleToZeroOne',  ## Rescale images from [0, 255] to [0, 1].
        keys=['merged', 'alpha', 'ori_merged', 'fg', 'bg']),  ## Images to be rescaled.
    dict(
        type='Normalize',  ## Augmentation pipeline that normalize the input images.
        keys=['merged'],  ## Images to be normalized.
        **img_norm_cfg),  ## Normalization config. See above for definition of `img_norm_
↪cfg`
    dict(
        type='Collect',  ## Pipeline that decides which keys in the data should be␣
↪passed to the model
        keys=['merged', 'alpha', 'trimap', 'ori_merged', 'fg', 'bg'],  ## Keys to pass␣
↪to the model
        meta_keys=[]),  ## Meta information keys. In training, meta information is not␣
↪needed.
```

(continues on next page)

(continued from previous page)

```
    dict(
        type='ImageToTensor',  ## Convert images to tensor.
        keys=['merged', 'alpha', 'trimap', 'ori_merged', 'fg', 'bg']),  ## Images to be␣
↪converted to Tensor.
]
test_pipeline = [
    dict(
        type='LoadImageFromFile',  ## Load alpha matte.
        key='alpha',  ## Key of alpha matte in annotation file. The pipeline will read␣
↪alpha matte from path `alpha_path`.
        flag='grayscale',
        save_original_img=True),
    dict(
        type='LoadImageFromFile',  ## Load image from file
        key='trimap',  ## Key of image to load. The pipeline will read trimap from path␣
↪`trimap_path`.
        flag='grayscale',  ## Load as grayscale image which has shape (height, width).
        save_original_img=True),  ## Save a copy of trimap for calculating metrics. It␣
↪will be saved with key `ori_trimap`
    dict(
        type='LoadImageFromFile',  ## Load image from file
        key='merged'),  ## Key of image to load. The pipeline will read merged from path␣
↪`merged_path`.
    dict(
        type='Pad',  ## Pipeline to pad images to align with the downsample factor of␣
↪the model.
        keys=['trimap', 'merged'],  ## Images to be padded.
        mode='reflect'),  ## Mode of the padding.
    dict(
        type='RescaleToZeroOne',  ## Same as it in train_pipeline.
        keys=['merged', 'ori_alpha']),  ## Images to be rescaled.
    dict(
        type='Normalize',  ## Same as it in train_pipeline.
        keys=['merged'],
        **img_norm_cfg),
    dict(
        type='Collect',  ## Same as it in train_pipeline.
        keys=['merged', 'trimap'],
        meta_keys=[
            'merged_path', 'pad', 'merged_ori_shape', 'ori_alpha',
            'ori_trimap'
        ]),
    dict(
        type='ImageToTensor',  ## Same as it in train_pipeline.
        keys=['merged', 'trimap']),
]
data = dict(
    samples_per_gpu=1,  ## Batch size of a single GPU.
    workers_per_gpu=4,  ## Worker to pre-fetch data for each single GPU.
    drop_last=True,  ## Use drop_last in data_loader.
    train=dict(  ## Train dataset config.
        type=dataset_type,  ## Type of dataset.
```

(continues on next page)

(continued from previous page)

```
        ann_file=f'{data_root}/training_list.json',  ## Path of annotation file
        data_prefix=data_root,  ## Prefix of image path.
        pipeline=train_pipeline),  ## See above for train_pipeline
    val=dict(  ## Validation dataset config.
        type=dataset_type,
        ann_file=f'{data_root}/test_list.json',
        data_prefix=data_root,
        pipeline=test_pipeline),  ## See above for test_pipeline
    test=dict(  ## Test dataset config.
        type=dataset_type,
        ann_file=f'{data_root}/test_list.json',
        data_prefix=data_root,
        pipeline=test_pipeline))  ## See above for test_pipeline

## optimizer
optimizers = dict(type='Adam', lr=0.00001)  ## Config used to build optimizer, support␣
↪all the optimizers in PyTorch whose arguments are also the same as those in PyTorch.
## learning policy
lr_config = dict(  ## Learning rate scheduler config used to register LrUpdater hook
    policy='Fixed')  ## The policy of scheduler, also support CosineAnnealing, Cyclic,␣
↪etc. Refer to details of supported LrUpdater from https://github.com/open-mmlab/mmcv/
↪blob/master/mmcv/runner/hooks/lr_updater.py#L9.

## checkpoint saving
checkpoint_config = dict(  ## Config to set the checkpoint hook, Refer to https://github.
↪com/open-mmlab/mmcv/blob/master/mmcv/runner/hooks/checkpoint.py for implementation.
    interval=40000,  ## The save interval is 40000 iterations.
    by_epoch=False)  ## Count by iterations.
evaluation = dict(  ## The config to build the evaluation hook.
    interval=40000)  ## Evaluation interval.
log_config = dict(  ## Config to register logger hook.
    interval=10,  ## Interval to print the log.
    hooks=[
        dict(type='TextLoggerHook', by_epoch=False),  ## The logger used to record the␣
↪training process.
        ## dict(type='TensorboardLoggerHook')  # The Tensorboard logger is also supported.
    ])

## runtime settings
total_iters = 1000000  ## Total iterations to train the model.
dist_params = dict(backend='nccl')  ## Parameters to setup distributed training, the␣
↪port can also be set.
log_level = 'INFO'  ## The level of logging.
work_dir = './work_dirs/dim_stage1'  ## Directory to save the model checkpoints and logs␣
↪for the current experiments.
load_from = None  ## load models as a pre-trained model from a given path. This will not␣
↪resume training.
resume_from = None  ## Resume checkpoints from a given path, the training will be␣
↪resumed from the epoch when the checkpoint's is saved.
workflow = [('train', 1)]  ## Workflow for runner. [('train', 1)] means there is only one␣
↪workflow and the workflow named 'train' is executed once. Keep this unchanged when␣
↪training current matting models.
```

### 1.5.5 Config System for Restoration

#### An Example - EDSR

To help the users have a basic idea of a complete config, we make a brief comments on the config of the EDSR model we implemented as the following. For more detailed usage and the corresponding alternative for each modules, please refer to the API documentation.

```
exp_name = 'edsr_x2c64b16_1x16_300k_div2k'  ## The experiment name

scale = 2  ## Scale factor for upsampling
## model settings
model = dict(
    type='BasicRestorer',  ## Name of the model
    generator=dict(  ## Config of the generator
        type='EDSR',  ## Type of the generator
        in_channels=3,  ## Channel number of inputs
        out_channels=3,  ## Channel number of outputs
        mid_channels=64,  ## Channel number of intermediate features
        num_blocks=16,  ## Block number in the trunk network
        upscale_factor=scale, ## Upsampling factor
        res_scale=1,  ## Used to scale the residual in residual block
        rgb_mean=(0.4488, 0.4371, 0.4040),  ## Image mean in RGB orders
        rgb_std=(1.0, 1.0, 1.0)),  ## Image std in RGB orders
    pixel_loss=dict(type='L1Loss', loss_weight=1.0, reduction='mean'))  ## Config for␣
↪pixel loss
## model training and testing settings
train_cfg = None  ## Training config
test_cfg = dict(  ## Test config
    metrics=['PSNR'],  ## Metrics used during testing
    crop_border=scale)  ## Crop border during evaluation

## dataset settings
train_dataset_type = 'SRAnnotationDataset'  ## Dataset type for training
val_dataset_type = 'SRFolderDataset'  ##  Dataset type for validation
train_pipeline = [  ## Training data processing pipeline
    dict(type='LoadImageFromFile',  ## Load images from files
        io_backend='disk',  ## io backend
        key='lq',  ## Keys in results to find corresponding path
        flag='unchanged'),  ## flag for reading images
    dict(type='LoadImageFromFile',  ## Load images from files
        io_backend='disk',  ## io backend
        key='gt',  ## Keys in results to find corresponding path
        flag='unchanged'),  ## flag for reading images
    dict(type='RescaleToZeroOne', keys=['lq', 'gt']),  ## Rescale images from [0, 255]␣
↪to [0, 1]
    dict(type='Normalize',  ## Augmentation pipeline that normalize the input images
        keys=['lq', 'gt'],  ## Images to be normalized
        mean=[0, 0, 0],  ## Mean values
        std=[1, 1, 1],  ## Standard variance
        to_rgb=True),  ## Change to RGB channel
    dict(type='PairedRandomCrop', gt_patch_size=96),  ## Paired random crop
    dict(type='Flip',  ## Flip images
```

(continues on next page)

(continued from previous page)

```
        keys=['lq', 'gt'],  ## Images to be flipped
        flip_ratio=0.5,  ## Flip ratio
        direction='horizontal'),  ## Flip direction
    dict(type='Flip',  ## Flip images
        keys=['lq', 'gt'],  ## Images to be flipped
        flip_ratio=0.5,  ## Flip ratio
        direction='vertical'),  ## Flip direction
    dict(type='RandomTransposeHW',  ## Random transpose h and w for images
        keys=['lq', 'gt'],  ## Images to be transposed
        transpose_ratio=0.5  ## Transpose ratio
        ),
    dict(type='Collect',  ## Pipeline that decides which keys in the data should be␣
↪passed to the model
        keys=['lq', 'gt'],  ## Keys to pass to the model
        meta_keys=['lq_path', 'gt_path']), ## Meta information keys. In training, meta␣
↪information is not needed
    dict(type='ImageToTensor',  ## Convert images to tensor
        keys=['lq', 'gt'])  ## Images to be converted to Tensor
]
test_pipeline = [  ## Test pipeline
    dict(
        type='LoadImageFromFile',  ## Load images from files
        io_backend='disk',  ## io backend
        key='lq',  ## Keys in results to find corresponding path
        flag='unchanged'),  ## flag for reading images
    dict(
        type='LoadImageFromFile',  ## Load images from files
        io_backend='disk',  ## io backend
        key='gt',  ## Keys in results to find corresponding path
        flag='unchanged'),  ## flag for reading images
    dict(type='RescaleToZeroOne', keys=['lq', 'gt']),  ## Rescale images from [0, 255]␣
↪to [0, 1]
    dict(
        type='Normalize',  ## Augmentation pipeline that normalize the input images
        keys=['lq', 'gt'],  ## Images to be normalized
        mean=[0, 0, 0],  ## Mean values
        std=[1, 1, 1],  ## Standard variance
        to_rgb=True),  ## Change to RGB channel
    dict(type='Collect',  ## Pipeline that decides which keys in the data should be␣
↪passed to the model
        keys=['lq', 'gt'],  ## Keys to pass to the model
        meta_keys=['lq_path', 'gt_path']),  ## Meta information keys
    dict(type='ImageToTensor',  ## Convert images to tensor
        keys=['lq', 'gt'])  ## Images to be converted to Tensor
]

data = dict(
    ## train
    samples_per_gpu=16,  ## Batch size of a single GPU
    workers_per_gpu=6,  ## Worker to pre-fetch data for each single GPU
    drop_last=True,  ## Use drop_last in data_loader
    train=dict(  ## Train dataset config
```

(continues on next page)

(continued from previous page)

```
        type='RepeatDataset',  ## Repeated dataset for iter-based model
        times=1000,  ## Repeated times for RepeatDataset
        dataset=dict(
            type=train_dataset_type,  ## Type of dataset
            lq_folder='data/DIV2K/DIV2K_train_LR_bicubic/X2_sub',  ## Path for lq folder
            gt_folder='data/DIV2K/DIV2K_train_HR_sub',  ## Path for gt folder
            ann_file='data/DIV2K/meta_info_DIV2K800sub_GT.txt',  ## Path for annotation␣
↪file
            pipeline=train_pipeline,  ## See above for train_pipeline
            scale=scale)),  ## Scale factor for upsampling
    ## val
    val_samples_per_gpu=1,  ## Batch size of a single GPU for validation
    val_workers_per_gpu=1,  ## Worker to pre-fetch data for each single GPU for␣
↪validation
    val=dict(
        type=val_dataset_type,  ## Type of dataset
        lq_folder='data/val_set5/Set5_bicLRx2',  ## Path for lq folder
        gt_folder='data/val_set5/Set5_mod12',  ## Path for gt folder
        pipeline=test_pipeline,  ## See above for test_pipeline
        scale=scale,  ## Scale factor for upsampling
        filename_tmpl='{}'),  ## filename template
    ## test
    test=dict(
        type=val_dataset_type,  ## Type of dataset
        lq_folder='data/val_set5/Set5_bicLRx2',  ## Path for lq folder
        gt_folder='data/val_set5/Set5_mod12',  ## Path for gt folder
        pipeline=test_pipeline,  ## See above for test_pipeline
        scale=scale,  ## Scale factor for upsampling
        filename_tmpl='{}'))  ## filename template

## optimizer
optimizers = dict(generator=dict(type='Adam', lr=1e-4, betas=(0.9, 0.999)))  ## Config␣
↪used to build optimizer, support all the optimizers in PyTorch whose arguments are␣
↪also the same as those in PyTorch

## learning policy
total_iters = 300000  ## Total training iters
lr_config = dict( ## Learning rate scheduler config used to register LrUpdater hook
    policy='Step', by_epoch=False, step=[200000], gamma=0.5)  ## The policy of scheduler,
↪ also support CosineAnnealing, Cyclic, etc. Refer to details of supported LrUpdater␣
↪from https://github.com/open-mmlab/mmcv/blob/master/mmcv/runner/hooks/lr_updater.py#L9.

checkpoint_config = dict(  ## Config to set the checkpoint hook, Refer to https://github.
↪com/open-mmlab/mmcv/blob/master/mmcv/runner/hooks/checkpoint.py for implementation.
    interval=5000,  ## The save interval is 5000 iterations
    save_optimizer=True,  ## Also save optimizers
    by_epoch=False)  ## Count by iterations
evaluation = dict(  ## The config to build the evaluation hook
    interval=5000,  ## Evaluation interval
    save_image=True,  ## Save images during evaluation
    gpu_collect=True)  ## Use gpu collect
log_config = dict(  ## Config to register logger hook
```

(continues on next page)

(continued from previous page)

```
    interval=100,  ## Interval to print the log
    hooks=[
        dict(type='TextLoggerHook', by_epoch=False),  ## The logger used to record the␣
↪training process
        dict(type='TensorboardLoggerHook'),  ## The Tensorboard logger is also supported
    ])
visual_config = None  ## Visual config, we do not use it.

## runtime settings
dist_params = dict(backend='nccl')  ## Parameters to setup distributed training, the␣
↪port can also be set
log_level = 'INFO'  ## The level of logging
work_dir = f'./work_dirs/{exp_name}'  ## Directory to save the model checkpoints and␣
↪logs for the current experiments
load_from = None ## load models as a pre-trained model from a given path. This will not␣
↪resume training
resume_from = None ## Resume checkpoints from a given path, the training will be resumed␣
↪from the iteration when the checkpoint's is saved
workflow = [('train', 1)]  ## Workflow for runner. [('train', 1)] means there is only one␣
↪workflow and the workflow named 'train' is executed once. Keep this unchanged when␣
↪training current matting models
```

## 1.6 Use intermediate variables in configs

Some intermediate variables are used in the configs files, like `train_pipeline/test_pipeline` in datasets.

For example, we would like to first define the `train_pipeline/test_pipeline` and pass them into `data`. Thus, `train_pipeline/test_pipeline` are intermediate variable.

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

(continues on next page)

(continued from previous page)

```
    dict(type='ImageToTensor', keys=['lq', 'gt'])
]

data = dict(
    # train
    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)),
    # val
    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  # empty cache in every iteration.
```

## 1.7 Tutorial 1: Customize Datasets

### 1.7.1 Supported Data Format

#### Image Super-Resolution

- SRAnnotationDataset General paired image dataset with an annotation file for image restoration.
- SRFolderDataset General paired image folder dataset for image restoration.
- SRFolderGTDataset General ground-truth image folder dataset for image restoration, where low-quality image should be generated in pipeline.
- SRFolderRefDataset General paired image folder dataset for reference-based image restoration.
- SRLmdbDataset General paired image lmdb dataset for image restoration.
- SRFacialLandmarkDataset Facial image and landmark dataset with an annotation file.

#### Video Super-Resolution

- SRFolderMultipleGTDataset General dataset for video super resolution, used for recurrent networks.
- SRREDSDataset REDS dataset for video super resolution.
- SRREDSMultipleGTDataset REDS dataset for video super resolution for recurrent networks.
- SRTestMultipleGTDataset Test dataset for video super resolution for recurrent networks.
- SRVid4Dataset Vid4 dataset for video super resolution.
- SRVimeo90KDataset Vimeo90K dataset for video super resolution.
- SRVimeo90KMultipleGTDataset Vimeo90K dataset for video super resolution for recurrent networks.

#### Video Frame Interpolation

- VFIVimeo90KDataset Vimeo90K dataset for video frame interpolation.

#### Matting

- AdobeComp1kDataset Adobe composition-1k dataset.

#### Inpainting

- ImgInpaintingDataset Only use the image name information from annotation file.

#### Generation

- GenerationPairedDataset General paired image folder dataset for image generation.
- GenerationUnpairedDataset General unpaired image folder dataset for image generation.

### 1.7.2 Support new data format

You can reorganize new data formats to existing format.

Or create a new dataset in mmedit/datasets to load the data.

Inheriting from the base class of datasets will make it easier to create a new dataset

- BaseSRDataset
- BaseVFIDataset
- BaseMattingDataset
- BaseGenerationDataset

Here is an example of create a dataset for video frame interpolation:

```
import os
import os.path as osp

from .base_vfi_dataset import BaseVFIDataset
from .registry import DATASETS
```

(continues on next page)

(continued from previous page)

```
@DATASETS.register_module()
class NewVFIDataset(BaseVFIDataset):
    """Introduce the dataset

    Examples of file structure.

    Args:
        pipeline (list[dict | callable]): A sequence of data transformations.
        folder (str | :obj:`Path`): Path to the folder.
        ann_file (str | :obj:`Path`): Path to the annotation file.
        test_mode (bool): Store `True` when building test dataset.
            Default: `False`.
    """

    def __init__(self, pipeline, folder, ann_file, test_mode=False):
        super().__init__(pipeline, folder, ann_file, test_mode)
        self.data_infos = self.load_annotations()

    def load_annotations(self):
        """Load annoations for the dataset.

        Returns:
            list[dict]: A list of dicts for paired paths and other information.
        """
        data_infos = []
        ...
        return data_infos
```

If you want create a dataset for a new low level CV task (e.g. denoise, derain, defog, and de-reflection), you can inheriting from BaseDataset.

Here is an example of create a base dataset for denoising:

```
import copy
from abc import ABCMeta, abstractmethod

from torch.utils.data import Dataset

from .pipelines import Compose

IMG_EXTENSIONS = ('.jpg', '.JPG', '.jpeg', '.JPEG', '.png', '.PNG', '.ppm',
                  '.PPM', '.bmp', '.BMP', '.tif', '.TIF', '.tiff', '.TIFF')


class BaseDnDataset(BaseDataset):
    """Base class for denoising datasets.
    """

    # If any extra parameter is required, please rewrite the `__init__`
    # def __init__(self, pipeline, new_para, test_mode=False):
```

(continues on next page)

(continued from previous page)

```
    #     super().__init__(pipeline, test_mode)
    #     self.new_para = new_para

    @staticmethod
    def scan_folder(path):
        """Obtain image path list (including sub-folders) from a given folder.

        Args:
            path (str | :obj:`Path`): Folder path.

        Returns:
            list[str]: image list obtained form given folder.
        """

        if isinstance(path, (str, Path)):
            path = str(path)
        else:
            raise TypeError("'path' must be a str or a Path object, "
                            f'but received {type(path)}.')

        images = list(scandir(path, suffix=IMG_EXTENSIONS, recursive=True))
        images = [osp.join(path, v) for v in images]
        assert images, f'{path} has no valid image file.'
        return images

    def __getitem__(self, idx):
        """Get item at each call.

        Args:
            idx (int): Index for getting each item.

        Returns:
            dict: The output dict of pipeline.
        """
        results = copy.deepcopy(self.data_infos[idx])
        return self.pipeline(results)

    def evaluate(self, results, logger=None):
        """Evaluate with different metrics.

        Args:
            results (list[tuple]): The output of forward_test() of the model.

        Return:
            dict: Evaluation results dict.
        """
        if not isinstance(results, list):
            raise TypeError(f'results must be a list, but got {type(results)}')
        assert len(results) == len(self), (
            'The length of results is not equal to the dataset len: '
            f'{len(results)} != {len(self)}')
```

(continues on next page)

(continued from previous page)

```
        results = [res['eval_result'] for res in results]  # a list of dict
        eval_result = defaultdict(list)  # a dict of list

        for res in results:
            for metric, val in res.items():
                eval_result[metric].append(val)
        for metric, val_list in eval_result.items():
            assert len(val_list) == len(self), (
                f'Length of evaluation result of {metric} is {len(val_list)}, '
                f'should be {len(self)}')

        # average the results
        eval_result = {
            metric: sum(values) / len(self)
            for metric, values in eval_result.items()
        }

        return eval_result
```

Welcome to submit new dataset classes to MMEditing.

### 1.7.3 Customize datasets by dataset wrappers

#### Repeat dataset

We use RepeatDataset as wrapper to repeat the dataset. For example, suppose the original dataset is Dataset_A, to repeat it, the config looks like the following

```
dataset_A_train = dict(
        type='RepeatDataset',
        times=N,
        dataset=dict(  # This is the original config of Dataset_A
            type='Dataset_A',
            ...
            pipeline=train_pipeline
        )
    )
```

## 1.8 Tutorial 2: Customize Data Pipelines

### 1.8.1 Design of Data pipelines

Following typical conventions, we use `Dataset` and `DataLoader` for data loading with multiple workers. `Dataset` returns a dict of data items corresponding the arguments of models' forward method.

The data preparation pipeline and the dataset is decomposed. Usually a dataset defines how to process the annotations and a data pipeline defines all the steps to prepare a data dict.

A pipeline consists of a sequence of operations. Each operation takes a dict as input and also output a dict for the next transform.

The operations are categorized into data loading, pre-processing, and formatting

Here is a pipeline example for BasicVSR++.

```
train_pipeline = [
    dict(
        type='LoadImageFromFileList',
        io_backend='disk',
        key='lq',
        channel_order='rgb'),
    dict(
        type='LoadImageFromFileList',
        io_backend='disk',
        key='gt',
        channel_order='rgb'),
    dict(type='RescaleToZeroOne', keys=['lq', 'gt']),
    dict(type='PairedRandomCrop', gt_patch_size=256),
    dict(
        type='Flip', keys=['lq', 'gt'], flip_ratio=0.5,
        direction='horizontal'),
    dict(type='Flip', keys=['lq', 'gt'], flip_ratio=0.5, direction='vertical'),
    dict(type='RandomTransposeHW', keys=['lq', 'gt'], transpose_ratio=0.5),
    dict(type='MirrorSequence', keys=['lq', 'gt']),
    dict(type='FramesToTensor', keys=['lq', 'gt']),
    dict(type='Collect', keys=['lq', 'gt'], meta_keys=['lq_path', 'gt_path'])
]

val_pipeline = [
    dict(type='GenerateSegmentIndices', interval_list=[1]),
    dict(
        type='LoadImageFromFileList',
        io_backend='disk',
        key='lq',
        channel_order='rgb'),
    dict(
        type='LoadImageFromFileList',
        io_backend='disk',
        key='gt',
        channel_order='rgb'),
    dict(type='RescaleToZeroOne', keys=['lq', 'gt']),
    dict(type='FramesToTensor', keys=['lq', 'gt']),
    dict(
        type='Collect',
        keys=['lq', 'gt'],
        meta_keys=['lq_path', 'gt_path', 'key'])
]

test_pipeline = [
    dict(
        type='LoadImageFromFileList',
        io_backend='disk',
        key='lq',
        channel_order='rgb'),
    dict(
```

(continues on next page)

(continued from previous page)

```
        type='LoadImageFromFileList',
        io_backend='disk',
        key='gt',
        channel_order='rgb'),
    dict(type='RescaleToZeroOne', keys=['lq', 'gt']),
    dict(type='MirrorSequence', keys=['lq']),
    dict(type='FramesToTensor', keys=['lq', 'gt']),
    dict(
        type='Collect',
        keys=['lq', 'gt'],
        meta_keys=['lq_path', 'gt_path', 'key'])
]
```

For each operation, we list the related dict fields that are added/updated/removed, the dict fields marked by '*' are optional.

### Data loading

`LoadImageFromFile`

- add: img, img_path, img_ori_shape, *ori_img

`LoadImageFromFileList`

- add: imgs, img_paths, img_ori_shapes, *ori_imgs

`RandomLoadResizeBg`

- add: bg

`LoadMask`

- add: mask

`GetSpatialDiscountMask`

- add: discount_mask

`LoadPairedImageFromFile`

- add: img, img_a, img_b, img_path, img_a_path, img_b_path, img_ori_shape, img_a_ori_shape, img_b_ori_shape, *ori_img, *ori_img_a, *ori_img_b

### Pre-processing

`Resize`

- add: scale_factor, keep_ratio, interpolation, backend
- update: specified by `keys`

`MATLABLikeResize`

- add: scale, output_shape
- update: specified by `keys`

`RandomRotation`

- add: degrees

- update: specified by `keys`

`Flip`

- add: flip, flip_direction
- update: specified by `keys`

`Pad`

- add: pad
- update: specified by `keys`

`RandomAffine`

- update: specified by `keys`

`RandomJitter`

- update: fg (img)

`ColorJitter`

- update: specified by `keys`

`BinarizeImage`

- update: specified by `keys`

`RandomMaskDilation`

- add: img_dilate_kernel_size

`RandomTransposeHW`

- add: transpose

`RandomDownSampling`

- update: scale, gt (img), lq (img)

`RandomBlur`

- update: specified by `keys`

`RandomResize`

- update: specified by `keys`

`RandomNoise`

- update: specified by `keys`

`RandomJPEGCompression`

- update: specified by `keys`

`RandomVideoCompression`

- update: specified by `keys`

`DegradationsWithShuffle`

- update: specified by `keys`

`GenerateFrameIndices`

- update: img_path (gt_path, lq_path)

`GenerateFrameIndiceswithPadding`

- update: img_path (gt_path, lq_path)

`TemporalReverse`

- add: reverse
- update: specified by `keys`

`GenerateSegmentIndices`

- add: interval
- update: img_path (gt_path, lq_path)

`MirrorSequence`

- update: specified by `keys`

`CopyValues`

- add: specified by `dst_key`

`Quantize`

- update: specified by `keys`

`UnsharpMasking`

- add: img_unsharp

`Crop`

- add: img_crop_bbox, crop_size
- update: specified by `keys`

`RandomResizedCrop`

- add: img_crop_bbox
- update: specified by `keys`

`FixedCrop`

- add: crop_size, crop_pos
- update: specified by `keys`

`PairedRandomCrop`

- update: gt (img), lq (img)

`CropAroundCenter`

- add: crop_bbox
- update: fg (img), alpha (img), trimap (img), bg (img)

`CropAroundUnknown`

- add: crop_bbox
- update: specified by `keys`

`CropAroundFg`

- add: crop_bbox
- update: specified by `keys`

`ModCrop`

- update: gt (img)

`CropLike`

- update: specified by `target_key`

`GetMaskedImage`

- add: masked_img

`GenerateHeatmap`

- add: heatmap

`GenerateCoordinateAndCell`

- add: coord, cell
- update: gt (img)

`Normalize`

- add: img_norm_cfg
- update: specified by `keys`

`RescaleToZeroOne`

- update: specified by `keys`

...

### Formatting

`ToTensor`

- update: specified by `keys`.

`ImageToTensor`

- update: specified by `keys`.

`FramesToTensor`

- update: specified by `keys`.

`FormatTrimap`

- update: trimap

`Collect`

- add: img_meta (the keys of img_meta is specified by `meta_keys`)
- remove: all other keys except for those specified by `keys`

### 1.8.2 Extend and use custom pipelines

1. Write a new pipeline in a file, e.g., in `my_pipeline.py`. It takes a dict as input and returns a dict.

```
import random
from mmdet.datasets import PIPELINES


@PIPELINES.register_module()
class MyTransform:
    """Add your transform

    Args:
        p (float): Probability of shifts. Default 0.5.
    """

    def __init__(self, p=0.5):
        self.p = p

    def __call__(self, results):
        if random.random() > self.p:
            results['dummy'] = True
        return results
```

2. Import and use the pipeline in your config file.

Make sure the import is relative to where your train script is located.

```
train_pipeline = [
    ...
    dict(type='MyTransform', p=0.2),
    ...
]
```

## 1.9 Tutorial 3: Customize Models

MMEditing supports multiple tasks, each of which has different settings. Fortunately, their customization is similar. Here, we use a super-resolution model, BasicVSR, as an example in this tutorial. You will be able to define your model based on your own needs after this tutorial.

We first need to create BasicVSR in `mmedit/models/restorers/basicvsr.py` .

```
from ..registry import MODELS
from .basic_restorer import BasicRestorer

@MODELS.register_module()
class BasicVSR(BasicRestorer):

    def __init__(self,
                 generator,
                 pixel_loss,
                 train_cfg=None,
                 test_cfg=None,
```

(continues on next page)

(continued from previous page)

```
                 pretrained=None):
        super().__init__(generator, pixel_loss, train_cfg, test_cfg,
                         pretrained)

        # fix pre-trained networks
        self.fix_iter = train_cfg.get('fix_iter', 0) if train_cfg else 0
        self.is_weight_fixed = False

        # count training steps
        self.register_buffer('step_counter', torch.zeros(1))
```

### 1.9.1 Model Argument

The values of these arguments are taken from the configuration file. Let's have a glance at the model part in the configuration file, you can find the complete config at `configs/restorers/basicvsr/basicvsr_reds4.py` .

```
model = dict(
    type='BasicVSR',
    generator=dict(
        type='BasicVSRNet',
        mid_channels=64,
        num_blocks=30,
        spynet_pretrained='https://download.openmmlab.com/mmediting/restorers/'
        'basicvsr/spynet_20210409-c6c1bd09.pth'),
    pixel_loss=dict(type='CharbonnierLoss', loss_weight=1.0, reduction='mean'))
train_cfg = dict(fix_iter=5000)
test_cfg = dict(metrics=['PSNR', 'SSIM'], crop_border=0)
```

We will now go through them one by one.

### 1.9.2 generator

`generator` specifies the network architecture, which is called **backbone** in MMEditing. The definition of the backbone is straightforward, but there is one thing that needs our attention.

#### Defining Backbone

Create a new file `mmedit/models/backbones/basicvsr_net.py` . The definition is standard. Please do make sure the line `@BACKBONES.register_module()` is added for all modules you would like to use.

```
import torch.nn as nn

from ..builder import BACKBONES

@BACKBONES.register_module()
class BasicVSRNet(nn.Module):

    def __init__(self, mid_channels, num_blocks, spynet_pretrained):
        pass
```

(continues on next page)

(continued from previous page)

```
    def forward(self, x):
        pass
```

#### Importing Module

This is the part we need to be careful. We need to add the following line to `mmedit/models/backbones/__init__.py` to use the defined backbone.

```
from .basicvsr_net import BasicVSRNet
```

### 1.9.3 Specification in Configuration File

Given the above model, the specification in the configuration file is straightforward. We see that the argument `type` is just the name of the backbone, and other arguments correspond to that in the backbone.

```
generator=dict(
    type='BasicVSRNet',
    mid_channels=64,
    num_blocks=30,
    spynet_pretrained='https://download.openmmlab.com/mmediting/restorers/'
    'basicvsr/spynet_20210409-c6c1bd09.pth')
```

### 1.9.4 pixel_loss

`pixel_loss` refers to the loss used in BasicVSR. The specification of the loss is similar to that of the backbone.

#### Defining Loss

Let's use Charbonnier loss as an example. We first define the loss in `mmedit/models/losses/pixelwise_loss.py`. The decorator `masked_loss` enables the loss to be weighted and masked for each element. Again, do make sure that the line `@LOSSES.register_module()` is included.

```
from ..registry import LOSSES
from .utils import masked_loss

@masked_loss
def charbonnier_loss(pred, target, eps=1e-12):
    return torch.sqrt((pred - target)**2 + eps)

@LOSSES.register_module()
class CharbonnierLoss(nn.Module):
    def __init__(self,
                 loss_weight=1.0,
                 reduction='mean',
                 sample_wise=False,
                 eps=1e-12):
        super().__init__()
```

(continues on next page)

(continued from previous page)

```
        if reduction not in ['none', 'mean', 'sum']:
            raise ValueError(f'Unsupported reduction mode: {reduction}. '
                             f'Supported ones are: {_reduction_modes}')

        self.loss_weight = loss_weight
        self.reduction = reduction
        self.sample_wise = sample_wise
        self.eps = eps

    def forward(self, pred, target, weight=None, **kwargs):
        return self.loss_weight * charbonnier_loss(
            pred,
            target,
            weight,
            eps=self.eps,
            reduction=self.reduction,
            sample_wise=self.sample_wise)
```

Similarly, we need to add the following line to `mmedit/models/losses/__init__.py` .

```
from .pixelwise_loss import CharbonnierLoss
```

Then, the specification in the config follows naturally.

```
pixel_loss=dict(type='CharbonnierLoss', loss_weight=1.0, reduction='mean')
```

### 1.9.5 train_cfg and test_cfg

`train_cfg` and `test_cfg` are just additional parameters you want to pass to the model. For example, in BasicVSR, a constant is passed to the model to fix a part of the network for a certain number of iterations.

```
self.fix_iter = train_cfg.get('fix_iter', 0) if train_cfg else 0
```

### 1.9.6 Model Functions

The model functions are used to control the training and test. In this tutorial, we will highlight a few important ones. For more details of the functions, you may refer to here.

#### train_step

This corresponds to the pipeline of each iteration, including forward and backward. In this example, the output and losses are computed. They are then used for backpropagation. More details of the forward process is discussed below.

```
def train_step(self, data_batch, optimizer):
    outputs = self(**data_batch, test_mode=False)
    loss, log_vars = self.parse_losses(outputs.pop('losses'))

    # optimize
    optimizer['generator'].zero_grad()
```

(continues on next page)

(continued from previous page)

```
    loss.backward()
    optimizer['generator'].step()

    outputs.update({'log_vars': log_vars})
    return outputs
```

### forward_train

This corresponds to the forward process. In this example, we will compute `output` given `lq` . Then `pixel_loss` is computed between `output` and `gt` . The computed loss will then be passed to a dictionary for further computations, including backpropagation. If you have any other losses, you should also include them here.

```
def forward_train(self, lq, gt):
    losses = dict()
    output = self.generator(lq)
    loss_pix = self.pixel_loss(output, gt)
    losses['loss_pix'] = loss_pix
    outputs = dict(
        losses=losses,
        num_samples=len(gt.data),
        results=dict(lq=lq.cpu(), gt=gt.cpu(), output=output.cpu()))
    return outputs
```

### forward_test

This corresponds to the validation and test. For example, you need to specify how you perform evaluation (i.e. calculation of metrics) and how you save the outputs.

```
def forward_test(self,
                 lq,
                 gt=None,
                 meta=None,
                 save_image=False,
                 save_path=None,
                 iteration=None):

        output = self.generator(lq)
        if self.test_cfg is not None and self.test_cfg.get('metrics', None):
            assert gt is not None, (
                'evaluation with metrics must have gt images.')
            results = dict(eval_result=self.evaluate(output, gt))
        else:
            results = dict(lq=lq.cpu(), output=output.cpu())
            if gt is not None:
                results['gt'] = gt.cpu()

        # save image
        if save_image:
            lq_path = meta[0]['lq_path']
            folder_name = osp.splitext(osp.basename(lq_path))[0]
```

(continues on next page)

(continued from previous page)

```
            if isinstance(iteration, numbers.Number):
                save_path = osp.join(save_path, folder_name,
                                     f'{folder_name}-{iteration + 1:06d}.png')
            elif iteration is None:
                save_path = osp.join(save_path, f'{folder_name}.png')
            else:
                raise ValueError('iteration should be number or None, '
                                 f'but got {type(iteration)}')
            mmcv.imwrite(tensor2img(output), save_path)

        return results
```

## 1.10 Tutorial 4: Customize Losses

`losses` are registered as `LOSSES` in `MMEditing`. Customizing losses is similar to customizing any other model. This section is mainly for clarifying the design of loss modules in our repo. Importantly, when writing your own loss modules, you should follow the same design, so that the new loss module can be adopted in our framework without extra effort.

### 1.10.1 Design of loss modules

In general, to implement a loss module, we will write a function implementation and then wrap it with a class implementation. Take the MSELoss as an example:

```
@masked_loss
def mse_loss(pred, target):
    return F.mse_loss(pred, target, reduction='none')

@LOSSES.register_module()
class MSELoss(nn.Module):

    def __init__(self, loss_weight=1.0, reduction='mean', sample_wise=False):
        # codes can be found in ``mmedit/models/losses/pixelwise_loss.py``

    def forward(self, pred, target, weight=None, **kwargs):
        # codes can be found in ``mmedit/models/losses/pixelwise_loss.py``
```

Given the definition of the loss, we can now use the loss by simply defining it in the configuration file:

```
pixel_loss=dict(type='MSELoss', loss_weight=1.0, reduction='mean')
```

Note that `pixel_loss` above must be defined in the model. Please refer to `customize_models` for more details. Similar to model customization, in order to use your customized loss, you need to import the loss in `mmedit/models/losses/__init__.py` after writing it.

## 1.11 Overview

- Number of checkpoints: 80
- Number of configs: 71
- Number of papers: 28
  - ALGORITHM: 28

For supported datasets, see *datasets overview*.

### 1.11.1 Inpainting Models

- Number of checkpoints: 8
- Number of configs: 9
- Number of papers: 5
  - [ALGORITHM] Aggregated Contextual Transformations for High-Resolution Image Inpainting ()
  - [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 ()

### 1.11.2 Matting Models

- Number of checkpoints: 9
- Number of configs: 9
- Number of papers: 3
  - [ALGORITHM] Deep Image Matting ()
  - [ALGORITHM] Indices Matter: Learning to Index for Deep Image Matting ()
  - [ALGORITHM] Natural Image Matting via Guided Contextual Attention ()

### 1.11.3 Super-Resolution Models

- Number of checkpoints: 46
- Number of configs: 40
- Number of papers: 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] 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] Realbasicvsr: Investigating Tradeoffs in Real-World Video Super-Resolution ()
  - [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 ()

### 1.11.4 Generation Models

- Number of checkpoints: 10
- Number of configs: 10
- Number of papers: 2
  - [ALGORITHM] Image-to-Image Translation With Conditional Adversarial Networks ()
  - [ALGORITHM] Unpaired Image-to-Image Translation Using Cycle-Consistent Adversarial Networks ()

### 1.11.5 Frame-Interpolation Models

- Number of checkpoints: 7
- Number of configs: 3
- Number of papers: 3
  - [ALGORITHM] Channel Attention Is All You Need for Video Frame Interpolation ()
  - [ALGORITHM] Flavr: Flow-Agnostic Video Representations for Fast Frame Interpolation ()
  - [ALGORITHM] Video Enhancement With Task-Oriented Flow ()

## 1.12 Inpainting Models

### 1.12.1 AOT-GAN (TVCG'2021)

AOT-GAN: Aggregated Contextual Transformations for High-Resolution Image Inpainting

### Abstract

State-of-the-art image inpainting approaches can suffer from generating distorted structures and blurry textures in high-resolution images (e.g., 512x512). The challenges mainly drive from (1) image content reasoning from distant contexts, and (2) fine-grained texture synthesis for a large missing region. To overcome these two challenges, we propose an enhanced GAN-based model, named Aggregated COntextual-Transformation GAN (AOT-GAN), for high-resolution image inpainting. Specifically, to enhance context reasoning, we construct the generator of AOT-GAN by stacking multiple layers of a proposed AOT block. The AOT blocks aggregate contextual transformations from various receptive fields, allowing to capture both informative distant image contexts and rich patterns of interest for context reasoning. For improving texture synthesis, we enhance the discriminator of AOT-GAN by training it with a tailored mask-prediction task. Such a training objective forces the discriminator to distinguish the detailed appearances of real and synthesized patches, and in turn, facilitates the generator to synthesize clear textures. Extensive comparisons on Places2, the most challenging benchmark with 1.8 million high-resolution images of 365 complex scenes, show that our model outperforms the state-of-the-art by a significant margin in terms of FID with 38.60% relative improvement. A user study including more than 30 subjects further validates the superiority of AOT-GAN. We further evaluate the proposed AOT-GAN in practical applications, e.g., logo removal, face editing, and object removal. Results show that our model achieves promising completions in the real world. We release code and models in this https URL.

### Results and models

#### Places365-Challenge

More results for different mask area:

### Citation

```
@inproceedings{yan2021agg,
  author = {Zeng, Yanhong and Fu, Jianlong and Chao, Hongyang and Guo, Baining},
  title = {Aggregated Contextual Transformations for High-Resolution Image Inpainting},
  booktitle = {Arxiv},
  pages={-},
  year = {2020}
}
```

## 1.12.2 DeepFillv1 (CVPR'2018)

> Generative Image Inpainting with Contextual Attention

### Abstract

Recent deep learning based approaches have shown promising results for the challenging task of inpainting large missing regions in an image. These methods can generate visually plausible image structures and textures, but often create distorted structures or blurry textures inconsistent with surrounding areas. This is mainly due to ineffectiveness of convolutional neural networks in explicitly borrowing or copying information from distant spatial locations. On the other hand, traditional texture and patch synthesis approaches are particularly suitable when it needs to borrow textures from the surrounding regions. Motivated by these observations, we propose a new deep generative model-based approach which can not only synthesize novel image structures but also explicitly utilize surrounding image features as references during network training to make better predictions. The model is a feed-forward, fully convolutional neural network which can process images with multiple holes at arbitrary locations and with variable sizes during the test time. Experiments on multiple datasets including faces (CelebA, CelebA-HQ), textures (DTD) and natural images

(ImageNet, Places2) demonstrate that our proposed approach generates higher-quality inpainting results than existing ones.

#### Results and models

**Places365-Challenge**

**CelebA-HQ**

#### Citation

```
@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}
}
```

### 1.12.3 DeepFillv2 (CVPR'2019)

> Free-Form Image Inpainting with Gated Convolution

#### Abstract

We present a generative image inpainting system to complete images with free-form mask and guidance. The system is based on gated convolutions learned from millions of images without additional labelling efforts. The proposed gated convolution solves the issue of vanilla convolution that treats all input pixels as valid ones, generalizes partial convolution by providing a learnable dynamic feature selection mechanism for each channel at each spatial location across all layers. Moreover, as free-form masks may appear anywhere in images with any shape, global and local GANs designed for a single rectangular mask are not applicable. Thus, we also present a patch-based GAN loss, named SN-PatchGAN, by applying spectral-normalized discriminator on dense image patches. SN-PatchGAN is simple in formulation, fast and stable in training. Results on automatic image inpainting and user-guided extension demonstrate that our system generates higher-quality and more flexible results than previous methods. Our system helps user quickly remove distracting objects, modify image layouts, clear watermarks and edit faces.

#### Results and models

**Places365-Challenge**

**CelebA-HQ**

#### Citation

```
@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}
}
```

### 1.12.4 Global&Local (ToG'2017)

> Globally and Locally Consistent Image Completion

#### Abstract

We present a novel approach for image completion that results in images that are both locally and globally consistent. With a fully-convolutional neural network, we can complete images of arbitrary resolutions by filling in missing regions of any shape. To train this image completion network to be consistent, we use global and local context discriminators that are trained to distinguish real images from completed ones. The global discriminator looks at the entire image to assess if it is coherent as a whole, while the local discriminator looks only at a small area centered at the completed region to ensure the local consistency of the generated patches. The image completion network is then trained to fool the both context discriminator networks, which requires it to generate images that are indistinguishable from real ones with regard to overall consistency as well as in details. We show that our approach can be used to complete a wide variety of scenes. Furthermore, in contrast with the patch-based approaches such as PatchMatch, our approach can generate fragments that do not appear elsewhere in the image, which allows us to naturally complete the image.

#### Results and models

*Note that we do not apply the post-processing module in Global&Local for a fair comparison with current deep inpainting methods.*

**Places365-Challenge**

**CelebA-HQ**

#### Citation

```
@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}
}
```

### 1.12.5 PConv (ECCV'2018)

Image Inpainting for Irregular Holes Using Partial Convolutions

#### Abstract

Existing deep learning based image inpainting methods use a standard convolutional network over the corrupted image, using convolutional filter responses conditioned on both valid pixels as well as the substitute values in the masked holes (typically the mean value). This often leads to artifacts such as color discrepancy and blurriness. Post-processing is usually used to reduce such artifacts, but are expensive and may fail. We propose the use of partial convolutions, where the convolution is masked and renormalized to be conditioned on only valid pixels. We further include a mechanism to automatically generate an updated mask for the next layer as part of the forward pass. Our model outperforms other methods for irregular masks. We show qualitative and quantitative comparisons with other methods to validate our approach.

#### Results and models

**Places365-Challenge**

**CelebA-HQ**

#### Citation

```
@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}
}
```

## 1.13 Matting Models

### 1.13.1 DIM (CVPR'2017)

Deep Image Matting

#### Abstract

Image matting is a fundamental computer vision problem and has many applications. Previous algorithms have poor performance when an image has similar foreground and background colors or complicated textures. The main reasons are prior methods 1) only use low-level features and 2) lack high-level context. In this paper, we propose a novel deep learning based algorithm that can tackle both these problems. Our deep model has two parts. The first part is a deep convolutional encoder-decoder network that takes an image and the corresponding trimap as inputs and predict the alpha matte of the image. The second part is a small convolutional network that refines the alpha matte predictions of the first network to have more accurate alpha values and sharper edges. In addition, we also create a large-scale image matting dataset including 49300 training images and 1000 testing images. We evaluate our algorithm on the image

matting benchmark, our testing set, and a wide variety of real images. Experimental results clearly demonstrate the superiority of our algorithm over previous methods.

### Results and models

**NOTE**

- stage1: train the encoder-decoder part without the refinement part.
- stage2: fix the encoder-decoder part and train the refinement part.
- stage3: fine-tune the whole network.

  The performance of the model is not stable during the training. Thus, the reported performance is not from the last checkpoint. Instead, it is the best performance of all validations during training.

  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.

### Citation

```
@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}
}
```

## 1.13.2 GCA (AAAI'2020)

> Natural Image Matting via Guided Contextual Attention

### Abstract

Over the last few years, deep learning based approaches have achieved outstanding improvements in natural image matting. Many of these methods can generate visually plausible alpha estimations, but typically yield blurry structures or textures in the semitransparent area. This is due to the local ambiguity of transparent objects. One possible solution is to leverage the far-surrounding information to estimate the local opacity. Traditional affinity-based methods often suffer from the high computational complexity, which are not suitable for high resolution alpha estimation. Inspired by affinity-based method and the successes of contextual attention in inpainting, we develop a novel end-to-end approach for natural image matting with a guided contextual attention module, which is specifically designed for image matting. Guided contextual attention module directly propagates high-level opacity information globally based on the learned low-level affinity. The proposed method can mimic information flow of affinity-based methods and utilize rich features learned by deep neural networks simultaneously. Experiment results on Composition-1k testing set and this http URL benchmark dataset demonstrate that our method outperforms state-of-the-art approaches in natural image matting.

#### Results and models

**More results**

#### Citation

```
@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}
}
```

### 1.13.3 IndexNet (ICCV'2019)

> Indices Matter: Learning to Index for Deep Image Matting

#### Abstract

We show that existing upsampling operators can be unified with the notion of the index function. This notion is inspired by an observation in the decoding process of deep image matting where indices-guided unpooling can recover boundary details much better than other upsampling operators such as bilinear interpolation. By looking at the indices as a function of the feature map, we introduce the concept of learning to index, and present a novel index-guided encoder-decoder framework where indices are self-learned adaptively from data and are used to guide the pooling and upsampling operators, without the need of supervision. At the core of this framework is a flexible network module, termed IndexNet, which dynamically predicts indices given an input. Due to its flexibility, IndexNet can be used as a plug-in applying to any off-the-shelf convolutional networks that have coupled downsampling and upsampling stages.

#### Results and models

> 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.

**More result**

#### Citation

```
@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}
}
```

## 1.14 Super-Resolution Models

### 1.14.1 BasicVSR (CVPR'2021)

> BasicVSR: The Search for Essential Components in Video Super-Resolution and Beyond

#### Abstract

Video super-resolution (VSR) approaches tend to have more components than the image counterparts as they need to exploit the additional temporal dimension. Complex designs are not uncommon. In this study, we wish to untangle the knots and reconsider some most essential components for VSR guided by four basic functionalities, i.e., Propagation, Alignment, Aggregation, and Upsampling. By reusing some existing components added with minimal redesigns, we show a succinct pipeline, BasicVSR, that achieves appealing improvements in terms of speed and restoration quality in comparison to many state-of-the-art algorithms. We conduct systematic analysis to explain how such gain can be obtained and discuss the pitfalls. We further show the extensibility of BasicVSR by presenting an information-refill mechanism and a coupled propagation scheme to facilitate information aggregation. The BasicVSR and its extension, IconVSR, can serve as strong baselines for future VSR approaches.

#### Results and models

Evaluated on RGB channels for REDS4 and Y channel for others. The metrics are `PSNR` / `SSIM` . The pretrained weights of SPyNet can be found here.

#### Citation

```
@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}
}
```

### 1.14.2 BasicVSR++ (CVPR'2022)

> BasicVSR++: Improving Video Super-Resolution with Enhanced Propagation and Alignment

#### Abstract

A recurrent structure is a popular framework choice for the task of video super-resolution. The state-of-the-art method BasicVSR adopts bidirectional propagation with feature alignment to effectively exploit information from the entire input video. In this study, we redesign BasicVSR by proposing second-order grid propagation and flow-guided deformable alignment. We show that by empowering the recurrent framework with the enhanced propagation and alignment, one can exploit spatiotemporal information across misaligned video frames more effectively. The new components lead to an improved performance under a similar computational constraint. In particular, our model BasicVSR++ surpasses BasicVSR by 0.82 dB in PSNR with similar number of parameters. In addition to video super-resolution, BasicVSR++ generalizes well to other video restoration tasks such as compressed video enhancement. In NTIRE 2021, BasicVSR++ obtains three champions and one runner-up in the Video Super-Resolution and Compressed Video Enhancement Challenges. Codes and models will be released to MMEditing.

#### Results and models

The pretrained weights of SPyNet can be found here.

Note that the following models are finetuned from smaller models. The training schemes of these models will be released when MMEditing reaches 5k stars. We provide the pre-trained models here.

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

#### Citation

```
@InProceedings{chan2022basicvsrplusplus,
  author = {Chan, Kelvin C.K. and Zhou, Shangchen and Xu, Xiangyu and Loy, Chen Change},
  title = {BasicVSR++: Improving Video Super-Resolution with Enhanced Propagation and Alignment},
  booktitle = {Proceedings of the IEEE conference on computer vision and pattern recognition},
  year = {2022}
}
```

### 1.14.3 DIC (CVPR'2020)

> Deep Face Super-Resolution with Iterative Collaboration between Attentive Recovery and Landmark Estimation

### Abstract

Recent works based on deep learning and facial priors have succeeded in super-resolving severely degraded facial images. However, the prior knowledge is not fully exploited in existing methods, since facial priors such as landmark and component maps are always estimated by low-resolution or coarsely super-resolved images, which may be inaccurate and thus affect the recovery performance. In this paper, we propose a deep face super-resolution (FSR) method with iterative collaboration between two recurrent networks which focus on facial image recovery and landmark estimation respectively. In each recurrent step, the recovery branch utilizes the prior knowledge of landmarks to yield higher-quality images which facilitate more accurate landmark estimation in turn. Therefore, the iterative information interaction between two processes boosts the performance of each other progressively. Moreover, a new attentive fusion module is designed to strengthen the guidance of landmark maps, where facial components are generated individually and aggregated attentively for better restoration. Quantitative and qualitative experimental results show the proposed method significantly outperforms state-of-the-art FSR methods in recovering high-quality face images.

### Results and models

Evaluated on RGB channels, `scale` pixels in each border are cropped before evaluation. The metrics are `PSNR / SSIM` .

In the log data of `dic_gan_x8c48b6_g4_150k_CelebAHQ`, DICGAN is verified on the first 9 pictures of the test set of CelebA-HQ, so `PSNR/SSIM` shown in the follow table is different from the log data.

### Citation

```
@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}
}
```

## 1.14.4 EDSR (CVPR'2017)

> Enhanced Deep Residual Networks for Single Image Super-Resolution

### Abstract

Recent research on super-resolution has progressed with the development of deep convolutional neural networks (DCNN). In particular, residual learning techniques exhibit improved performance. In this paper, we develop an enhanced deep super-resolution network (EDSR) with performance exceeding those of current state-of-the-art SR methods. The significant performance improvement of our model is due to optimization by removing unnecessary modules in conventional residual networks. The performance is further improved by expanding the model size while we stabilize the training procedure. We also propose a new multi-scale deep super-resolution system (MDSR) and training method, which can reconstruct high-resolution images of different upscaling factors in a single model. The proposed methods show superior performance over the state-of-the-art methods on benchmark datasets and prove its excellence by winning the NTIRE2017 Super-Resolution Challenge.

### Results and models

Evaluated on RGB channels, `scale` pixels in each border are cropped before evaluation. The metrics are `PSNR / SSIM` .

### Citation

```
@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}
}
```

## 1.14.5 EDVR (CVPRW'2019)

EDVR: Video Restoration with Enhanced Deformable Convolutional Networks

### Abstract

Video restoration tasks, including super-resolution, deblurring, etc, are drawing increasing attention in the computer vision community. A challenging benchmark named REDS is released in the NTIRE19 Challenge. This new benchmark challenges existing methods from two aspects: (1) how to align multiple frames given large motions, and (2) how to effectively fuse different frames with diverse motion and blur. In this work, we propose a novel Video Restoration framework with Enhanced Deformable networks, termed EDVR, to address these challenges. First, to handle large motions, we devise a Pyramid, Cascading and Deformable (PCD) alignment module, in which frame alignment is done at the feature level using deformable convolutions in a coarse-to-fine manner. Second, we propose a Temporal and Spatial Attention (TSA) fusion module, in which attention is applied both temporally and spatially, so as to emphasize important features for subsequent restoration. Thanks to these modules, our EDVR wins the champions and outperforms the second place by a large margin in all four tracks in the NTIRE19 video restoration and enhancement challenges. EDVR also demonstrates superior performance to state-of-the-art published methods on video super-resolution and deblurring.

### Results and models

Evaluated on RGB channels. The metrics are `PSNR / SSIM` .

#### Citation

```
@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},
}
```

### 1.14.6 ESRGAN (ECCVW'2018)

ESRGAN: Enhanced Super-Resolution Generative Adversarial Networks

#### Abstract

The Super-Resolution Generative Adversarial Network (SRGAN) is a seminal work that is capable of generating realistic textures during single image super-resolution. However, the hallucinated details are often accompanied with unpleasant artifacts. To further enhance the visual quality, we thoroughly study three key components of SRGAN - network architecture, adversarial loss and perceptual loss, and improve each of them to derive an Enhanced SRGAN (ESRGAN). In particular, we introduce the Residual-in-Residual Dense Block (RRDB) without batch normalization as the basic network building unit. Moreover, we borrow the idea from relativistic GAN to let the discriminator predict relative realness instead of the absolute value. Finally, we improve the perceptual loss by using the features before activation, which could provide stronger supervision for brightness consistency and texture recovery. Benefiting from these improvements, the proposed ESRGAN achieves consistently better visual quality with more realistic and natural textures than SRGAN and won the first place in the PIRM2018-SR Challenge.

#### Results and models

Evaluated on RGB channels, `scale` pixels in each border are cropped before evaluation. The metrics are `PSNR / SSIM` .

#### Citation

```
@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}
}
```

### 1.14.7 GLEAN (CVPR'2021)

GLEAN: Generative Latent Bank for Large-Factor Image Super-Resolution

#### Abstract

We show that pre-trained Generative Adversarial Networks (GANs), e.g., StyleGAN, can be used as a latent bank to improve the restoration quality of large-factor image super-resolution (SR). While most existing SR approaches attempt to generate realistic textures through learning with adversarial loss, our method, Generative LatEnt bANk (GLEAN), goes beyond existing practices by directly leveraging rich and diverse priors encapsulated in a pre-trained GAN. But unlike prevalent GAN inversion methods that require expensive image-specific optimization at runtime, our approach only needs a single forward pass to generate the upscaled image. GLEAN can be easily incorporated in a simple encoder-bank-decoder architecture with multi-resolution skip connections. Switching the bank allows the method to deal with images from diverse categories, e.g., cat, building, human face, and car. Images upscaled by GLEAN show clear improvements in terms of fidelity and texture faithfulness in comparison to existing methods.

#### Results and models

For the meta info used in training and test, please refer to here. The results are evaluated on RGB channels.

#### Citation

```
@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}
}
```

### 1.14.8 IconVSR (CVPR'2021)

BasicVSR: The Search for Essential Components in Video Super-Resolution and Beyond

#### Abstract

Video super-resolution (VSR) approaches tend to have more components than the image counterparts as they need to exploit the additional temporal dimension. Complex designs are not uncommon. In this study, we wish to untangle the knots and reconsider some most essential components for VSR guided by four basic functionalities, i.e., Propagation, Alignment, Aggregation, and Upsampling. By reusing some existing components added with minimal redesigns, we show a succinct pipeline, BasicVSR, that achieves appealing improvements in terms of speed and restoration quality in comparison to many state-of-the-art algorithms. We conduct systematic analysis to explain how such gain can be obtained and discuss the pitfalls. We further show the extensibility of BasicVSR by presenting an information-refill mechanism and a coupled propagation scheme to facilitate information aggregation. The BasicVSR and its extension, IconVSR, can serve as strong baselines for future VSR approaches.

### Results and models

Evaluated on RGB channels for REDS4 and Y channel for others. The metrics are `PSNR` / `SSIM` . The pretrained weights of the IconVSR components can be found here: SPyNet, EDVR-M for REDS, and EDVR-M for Vimeo-90K.

### Citation

```
@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}
}
```

## 1.14.9 LIIF (CVPR'2021)

> Learning Continuous Image Representation with Local Implicit Image Function

### Abstract

How to represent an image? While the visual world is presented in a continuous manner, machines store and see the images in a discrete way with 2D arrays of pixels. In this paper, we seek to learn a continuous representation for images. Inspired by the recent progress in 3D reconstruction with implicit neural representation, we propose Local Implicit Image Function (LIIF), which takes an image coordinate and the 2D deep features around the coordinate as inputs, predicts the RGB value at a given coordinate as an output. Since the coordinates are continuous, LIIF can be presented in arbitrary resolution. To generate the continuous representation for images, we train an encoder with LIIF representation via a self-supervised task with super-resolution. The learned continuous representation can be presented in arbitrary resolution even extrapolate to x30 higher resolution, where the training tasks are not provided. We further show that LIIF representation builds a bridge between discrete and continuous representation in 2D, it naturally supports the learning tasks with size-varied image ground-truths and significantly outperforms the method with resizing the ground-truths.

### Results and models

Note:

- refers to ditto.
- Evaluated on RGB channels, `scale` pixels in each border are cropped before evaluation.

### Citation

```
@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}
}
```

## 1.14.10 RDN (CVPR'2018)

> Residual Dense Network for Image Super-Resolution

### Abstract

A very deep convolutional neural network (CNN) has recently achieved great success for image super-resolution (SR) and offered hierarchical features as well. However, most deep CNN based SR models do not make full use of the hierarchical features from the original low-resolution (LR) images, thereby achieving relatively-low performance. In this paper, we propose a novel residual dense network (RDN) to address this problem in image SR. We fully exploit the hierarchical features from all the convolutional layers. Specifically, we propose residual dense block (RDB) to extract abundant local features via dense connected convolutional layers. RDB further allows direct connections from the state of preceding RDB to all the layers of current RDB, leading to a contiguous memory (CM) mechanism. Local feature fusion in RDB is then used to adaptively learn more effective features from preceding and current local features and stabilizes the training of wider network. After fully obtaining dense local features, we use global feature fusion to jointly and adaptively learn global hierarchical features in a holistic way. Extensive experiments on benchmark datasets with different degradation models show that our RDN achieves favorable performance against state-of-the-art methods.

### Results and models

Evaluated on RGB channels, `scale` pixels in each border are cropped before evaluation. The metrics are `PSNR / SSIM` .

### Citation

```
@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}
}
```

### 1.14.11 RealBasicVSR (CVPR'2022)

RealBasicVSR: Investigating Tradeoffs in Real-World Video Super-Resolution

#### Abstract

The diversity and complexity of degradations in real-world video super-resolution (VSR) pose non-trivial challenges in inference and training. First, while long-term propagation leads to improved performance in cases of mild degradations, severe in-the-wild degradations could be exaggerated through propagation, impairing output quality. To balance the tradeoff between detail synthesis and artifact suppression, we found an image pre-cleaning stage indispensable to reduce noises and artifacts prior to propagation. Equipped with a carefully designed cleaning module, our RealBasicVSR outperforms existing methods in both quality and efficiency. Second, real-world VSR models are often trained with diverse degradations to improve generalizability, requiring increased batch size to produce a stable gradient. Inevitably, the increased computational burden results in various problems, including 1) speed-performance tradeoff and 2) batch-length tradeoff. To alleviate the first tradeoff, we propose a stochastic degradation scheme that reduces up to 40% of training time without sacrificing performance. We then analyze different training settings and suggest that employing longer sequences rather than larger batches during training allows more effective uses of temporal information, leading to more stable performance during inference. To facilitate fair comparisons, we propose the new VideoLQ dataset, which contains a large variety of real-world low-quality video sequences containing rich textures and patterns. Our dataset can serve as a common ground for benchmarking. Code, models, and the dataset will be made publicly available.

#### Results and models

Evaluated on Y channel. The code for computing NRQM, NIQE, and PI can be found here. MATLAB official code is used to compute BRISQUE.

#### Citation

```
@InProceedings{chan2022investigating,
  author = {Chan, Kelvin C.K. and Zhou, Shangchen and Xu, Xiangyu and Loy, Chen Change},
  title = {RealBasicVSR: Investigating Tradeoffs in Real-World Video Super-Resolution},
  booktitle = {Proceedings of the IEEE conference on computer vision and pattern␣
↪recognition},
  year = {2022}
}
```

### 1.14.12 Real-ESRGAN (ICCVW'2021)

Real-ESRGAN: Training Real-World Blind Super-Resolution with Pure Synthetic Data

### Abstract

Though many attempts have been made in blind super-resolution to restore low-resolution images with unknown and complex degradations, they are still far from addressing general real-world degraded images. In this work, we extend the powerful ESRGAN to a practical restoration application (namely, Real-ESRGAN), which is trained with pure synthetic data. Specifically, a high-order degradation modeling process is introduced to better simulate complex real-world degradations. We also consider the common ringing and overshoot artifacts in the synthesis process. In addition, we employ a U-Net discriminator with spectral normalization to increase discriminator capability and stabilize the training dynamics. Extensive comparisons have shown its superior visual performance than prior works on various real datasets. We also provide efficient implementations to synthesize training pairs on the fly.

### Results and models

Evaluated on RGB channels. The metrics are `PSNR/SSIM`.

### Citation

```
@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}
}
```

## 1.14.13 SRCNN (TPAMI'2015)

> Image Super-Resolution Using Deep Convolutional Networks

### Abstract

We propose a deep learning method for single image super-resolution (SR). Our method directly learns an end-to-end mapping between the low/high-resolution images. The mapping is represented as a deep convolutional neural network (CNN) that takes the low-resolution image as the input and outputs the high-resolution one. We further show that traditional sparse-coding-based SR methods can also be viewed as a deep convolutional network. But unlike traditional methods that handle each component separately, our method jointly optimizes all layers. Our deep CNN has a lightweight structure, yet demonstrates state-of-the-art restoration quality, and achieves fast speed for practical on-line usage. We explore different network structures and parameter settings to achieve trade-offs between performance and speed. Moreover, we extend our network to cope with three color channels simultaneously, and show better overall reconstruction quality.

### Results and models

Evaluated on RGB channels, `scale` pixels in each border are cropped before evaluation. The metrics are `PSNR / SSIM` .

### Citation

```
@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}
}
```

## 1.14.14 SRGAN (CVPR'2016)

Photo-Realistic Single Image Super-Resolution Using a Generative Adversarial Network

### Abstract

Despite the breakthroughs in accuracy and speed of single image super-resolution using faster and deeper convolutional neural networks, one central problem remains largely unsolved: how do we recover the finer texture details when we super-resolve at large upscaling factors? The behavior of optimization-based super-resolution methods is principally driven by the choice of the objective function. Recent work has largely focused on minimizing the mean squared reconstruction error. The resulting estimates have high peak signal-to-noise ratios, but they are often lacking high-frequency details and are perceptually unsatisfying in the sense that they fail to match the fidelity expected at the higher resolution. In this paper, we present SRGAN, a generative adversarial network (GAN) for image super-resolution (SR). To our knowledge, it is the first framework capable of inferring photo-realistic natural images for 4x upscaling factors. To achieve this, we propose a perceptual loss function which consists of an adversarial loss and a content loss. The adversarial loss pushes our solution to the natural image manifold using a discriminator network that is trained to differentiate between the super-resolved images and original photo-realistic images. In addition, we use a content loss motivated by perceptual similarity instead of similarity in pixel space. Our deep residual network is able to recover photo-realistic textures from heavily downsampled images on public benchmarks. An extensive mean-opinion-score (MOS) test shows hugely significant gains in perceptual quality using SRGAN. The MOS scores obtained with SRGAN are closer to those of the original high-resolution images than to those obtained with any state-of-the-art method.

### Results and models

Evaluated on RGB channels, `scale` pixels in each border are cropped before evaluation.

The metrics are `PSNR / SSIM` .

#### Citation

```
@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}
}
```

### 1.14.15 TDAN (CVPR'2020)

> TDAN: Temporally Deformable Alignment Network for Video Super-Resolution

#### Abstract

Video super-resolution (VSR) aims to restore a photo-realistic high-resolution (HR) video frame from both its corresponding low-resolution (LR) frame (reference frame) and multiple neighboring frames (supporting frames). Due to varying motion of cameras or objects, the reference frame and each support frame are not aligned. Therefore, temporal alignment is a challenging yet important problem for VSR. Previous VSR methods usually utilize optical flow between the reference frame and each supporting frame to wrap the supporting frame for temporal alignment. Therefore, the performance of these image-level wrapping-based models will highly depend on the prediction accuracy of optical flow, and inaccurate optical flow will lead to artifacts in the wrapped supporting frames, which also will be propagated into the reconstructed HR video frame. To overcome the limitation, in this paper, we propose a temporal deformable alignment network (TDAN) to adaptively align the reference frame and each supporting frame at the feature level without computing optical flow. The TDAN uses features from both the reference frame and each supporting frame to dynamically predict offsets of sampling convolution kernels. By using the corresponding kernels, TDAN transforms supporting frames to align with the reference frame. To predict the HR video frame, a reconstruction network taking aligned frames and the reference frame is utilized. Experimental results demonstrate the effectiveness of the proposed TDAN-based VSR model.

#### Results and models

Evaluated on Y-channel. 8 pixels in each border are cropped before evaluation. The metrics are `PSNR / SSIM` .

**Train**

You can use the following command to train a model.

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

TDAN is trained with two stages.

**Stage 1**: Train with a larger learning rate (1e-4)

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

**Stage 2**: Fine-tune with a smaller learning rate (5e-5)

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

For more details, you can refer to **Train a model** part in getting_started.

**Test**

You can use the following command to test a model.

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

Example: Test TDAN on SPMCS-30 using Bicubic downsampling.

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

For more details, you can refer to **Inference with pretrained models** part in getting_started.

### Citation

```
@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}
}
```

## 1.14.16 TOFlow (IJCV'2019)

> Video Enhancement with Task-Oriented Flow

### Abstract

Many video enhancement algorithms rely on optical flow to register frames in a video sequence. Precise flow estimation is however intractable; and optical flow itself is often a sub-optimal representation for particular video processing tasks. In this paper, we propose task-oriented flow (TOFlow), a motion representation learned in a self-supervised, task-specific manner. We design a neural network with a trainable motion estimation component and a video processing component, and train them jointly to learn the task-oriented flow. For evaluation, we build Vimeo-90K, a large-scale, high-quality video dataset for low-level video processing. TOFlow outperforms traditional optical flow on standard benchmarks as well as our Vimeo-90K dataset in three video processing tasks: frame interpolation, video denoising/deblocking, and video super-resolution.

### Results and models

Evaluated on RGB channels. The metrics are `PSNR / SSIM` .

### Citation

```
@article{xue2019video,
  title={Video enhancement with task-oriented flow},
  author={Xue, Tianfan and Chen, Baian and Wu, Jiajun and Wei, Donglai and Freeman,␣
↪William T},
  journal={International Journal of Computer Vision},
  volume={127},
  number={8},
  pages={1106--1125},
  year={2019},
  publisher={Springer}
}
```

## 1.14.17 TTSR (CVPR'2020)

> Learning Texture Transformer Network for Image Super-Resolution

### Abstract

We study on image super-resolution (SR), which aims to recover realistic textures from a low-resolution (LR) image. Recent progress has been made by taking high-resolution images as references (Ref), so that relevant textures can be transferred to LR images. However, existing SR approaches neglect to use attention mechanisms to transfer high-resolution (HR) textures from Ref images, which limits these approaches in challenging cases. In this paper, we propose a novel Texture Transformer Network for Image Super-Resolution (TTSR), in which the LR and Ref images are formulated as queries and keys in a transformer, respectively. TTSR consists of four closely-related modules optimized for image generation tasks, including a learnable texture extractor by DNN, a relevance embedding module, a hard-attention module for texture transfer, and a soft-attention module for texture synthesis. Such a design encourages joint feature learning across LR and Ref images, in which deep feature correspondences can be discovered by attention, and thus accurate texture features can be transferred. The proposed texture transformer can be further stacked in a cross-scale way, which enables texture recovery from different levels (e.g., from 1x to 4x magnification). Extensive experiments show that TTSR achieves significant improvements over state-of-the-art approaches on both quantitative and qualitative evaluations.

### Results and models

Evaluated on RGB channels, `scale` pixels in each border are cropped before evaluation. The metrics are `PSNR / SSIM` .

#### Citation

```
@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}
}
```

## 1.15 Generation Models

### 1.15.1 CycleGAN (ICCV'2017)

> Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks

#### Abstract

Image-to-image translation is a class of vision and graphics problems where the goal is to learn the mapping between an input image and an output image using a training set of aligned image pairs. However, for many tasks, paired training data will not be available. We present an approach for learning to translate an image from a source domain X to a target domain Y in the absence of paired examples. Our goal is to learn a mapping G:X→Y such that the distribution of images from G(X) is indistinguishable from the distribution Y using an adversarial loss. Because this mapping is highly under-constrained, we couple it with an inverse mapping F:Y→X and introduce a cycle consistency loss to push F(G(X))X (and vice versa). Qualitative results are presented on several tasks where paired training data does not exist, including collection style transfer, object transfiguration, season transfer, photo enhancement, etc. Quantitative comparisons against several prior methods demonstrate the superiority of our approach.

#### Results and models

We use `FID` and `IS` metrics to evaluate the generation performance of CycleGAN.

Note: With a larger identity loss, the image-to-image translation becomes more conservative, which makes less changes. The original authors did not say what is the best weight for identity loss. Thus, in addition to the default setting, we also set the weight of identity loss to 0 (denoting `id0` ) to make a more comprehensive comparison.

#### Citation

```
@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}
}
```

### 1.15.2 Pix2Pix (CVPR'2017)

> Image-to-Image Translation with Conditional Adversarial Networks

#### Abstract

We investigate conditional adversarial networks as a general-purpose solution to image-to-image translation problems. These networks not only learn the mapping from input image to output image, but also learn a loss function to train this mapping. This makes it possible to apply the same generic approach to problems that traditionally would require very different loss formulations. We demonstrate that this approach is effective at synthesizing photos from label maps, reconstructing objects from edge maps, and colorizing images, among other tasks. As a community, we no longer hand-engineer our mapping functions, and this work suggests we can achieve reasonable results without hand-engineering our loss functions either.

#### Results and models

We use `FID` and `IS` metrics to evaluate the generation performance of pix2pix.

Note: we strictly follow the paper setting in Section 3.3:

"*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.*"

i.e., `use model.train()` mode, thus may lead to slightly different inference results every time.

#### Citation

```
@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}
}
```

## 1.16 Frame-Interpolation Models

### 1.16.1 CAIN (AAAI'2020)

> Channel Attention Is All You Need for Video Frame Interpolation

### Abstract

Prevailing video frame interpolation techniques rely heavily on optical flow estimation and require additional model complexity and computational cost; it is also susceptible to error propagation in challenging scenarios with large motion and heavy occlusion. To alleviate the limitation, we propose a simple but effective deep neural network for video frame interpolation, which is end-to-end trainable and is free from a motion estimation network component. Our algorithm employs a special feature reshaping operation, referred to as PixelShuffle, with a channel attention, which replaces the optical flow computation module. The main idea behind the design is to distribute the information in a feature map into multiple channels and extract motion information by attending the channels for pixel-level frame synthesis. The model given by this principle turns out to be effective in the presence of challenging motion and occlusion. We construct a comprehensive evaluation benchmark and demonstrate that the proposed approach achieves outstanding performance compared to the existing models with a component for optical flow computation.

### Results and models

Evaluated on RGB channels. The metrics are `PSNR / SSIM` . The learning rate adjustment strategy is `Step LR scheduler with min_lr clipping`.

### Citation

```
@inproceedings{choi2020channel,
  title={Channel attention is all you need for video frame interpolation},
  author={Choi, Myungsub and Kim, Heewon and Han, Bohyung and Xu, Ning and Lee, Kyoung␣
↪Mu},
  booktitle={Proceedings of the AAAI Conference on Artificial Intelligence},
  volume={34},
  number={07},
  pages={10663--10671},
  year={2020}
}
```

## 1.16.2 FLAVR (arXiv'2020)

FLAVR: Flow-Agnostic Video Representations for Fast Frame Interpolation

### Abstract

Most modern frame interpolation approaches rely on explicit bidirectional optical flows between adjacent frames, thus are sensitive to the accuracy of underlying flow estimation in handling occlusions while additionally introducing computational bottlenecks unsuitable for efficient deployment. In this work, we propose a flow-free approach that is completely end-to-end trainable for multi-frame video interpolation. Our method, FLAVR, is designed to reason about non-linear motion trajectories and complex occlusions implicitly from unlabeled videos and greatly simplifies the process of training, testing and deploying frame interpolation models. Furthermore, FLAVR delivers up to 6× speed up compared to the current state-of-the-art methods for multi-frame interpolation while consistently demonstrating superior qualitative and quantitative results compared with prior methods on popular benchmarks including Vimeo-90K, Adobe-240FPS, and GoPro. Finally, we show that frame interpolation is a competitive self-supervised pre-training task for videos via demonstrating various novel applications of FLAVR including action recognition, optical flow estimation, motion magnification, and video object tracking. Code and trained models are provided in the supplementary material.

#### Results and models

Evaluated on RGB channels. The metrics are `PSNR / SSIM`.

Note: FLAVR for x8 VFI task will supported in the future.

#### Citation

```
@article{kalluri2020flavr,
  title={Flavr: Flow-agnostic video representations for fast frame interpolation},
  author={Kalluri, Tarun and Pathak, Deepak and Chandraker, Manmohan and Tran, Du},
  journal={arXiv preprint arXiv:2012.08512},
  year={2020}
}
```

### 1.16.3 TOFlow (IJCV'2019)

> Video Enhancement with Task-Oriented Flow

#### Abstract

Many video enhancement algorithms rely on optical flow to register frames in a video sequence. Precise flow estimation is however intractable; and optical flow itself is often a sub-optimal representation for particular video processing tasks. In this paper, we propose task-oriented flow (TOFlow), a motion representation learned in a self-supervised, task-specific manner. We design a neural network with a trainable motion estimation component and a video processing component, and train them jointly to learn the task-oriented flow. For evaluation, we build Vimeo-90K, a large-scale, high-quality video dataset for low-level video processing. TOFlow outperforms traditional optical flow on standard benchmarks as well as our Vimeo-90K dataset in three video processing tasks: frame interpolation, video denoising/deblocking, and video super-resolution.

#### Results and models

Evaluated on RGB channels. The metrics are `PSNR / SSIM`.

Note: These pretrained SPyNets don't contain BN layer since `batch_size=1`, which is consistent with `https://github.com/Coldog2333/pytoflow`.

#### Citation

```
@article{xue2019video,
  title={Video enhancement with task-oriented flow},
  author={Xue, Tianfan and Chen, Baian and Wu, Jiajun and Wei, Donglai and Freeman,␣
↪William T},
  journal={International Journal of Computer Vision},
  volume={127},
  number={8},
  pages={1106--1125},
  year={2019},
  publisher={Springer}
}
```

## 1.17 Overview

- Number of papers: 11
  - DATASET: 11

For supported editing algorithms, see *modelzoo overview*.

### 1.17.1 Generation Datasets

- Number of papers: 2
  - [DATASET] Image-to-Image Translation With Conditional Adversarial Networks ( Paired Dataset for Pix2pix )
  - [DATASET] Unpaired Image-to-Image Translation Using Cycle-Consistent Adversarial Networks ( Unpaired Dataset for CycleGAN )

### 1.17.2 Inpainting Datasets

- Number of papers: 3
  - [DATASET] Context Encoders: Feature Learning by Inpainting ( Paris Street View Dataset )
  - [DATASET] Places: A 10 Million Image Database for Scene Recognition ( Places365 Dataset )
  - [DATASET] Progressive Growing of Gans for Improved Quality, Stability, and Variation ( CelebA-HQ Dataset )

### 1.17.3 Matting Datasets

- Number of papers: 1
  - [DATASET] Deep Image Matting ( Composition-1k Dataset )

### 1.17.4 Super-Resolution Datasets

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

### 1.17.5 Video Frame Interpolation Datasets

- Number of papers: 0

## 1.18 Inpainting Datasets

It is recommended to symlink the dataset root to `$MMEDITING/data`. If your folder structure is different, you may need to change the corresponding paths in config files.

MMEditing supported inpainting datasets:

- *Paris Street View* [ Homepage ]
- *CelebA-HQ* [ Homepage ]
- *Places365* [ Homepage ]

As we only need images for inpainting task, further preparation is not necessary and the folder structure can be different from the example. You can utilize the information provided by the original dataset like `Place365` (e.g. `meta`). Also, you can easily scan the data set and list all of the images to a specific `txt` file. Here is an example for the `Places365_val.txt` from Places365 and we will only use the image name information in inpainting.

```
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
```

### 1.18.1 CelebA-HQ Dataset

```
@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}
}
```

Follow the instructions here to prepare the dataset.

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

### 1.18.2 Paris Street View Dataset

```
@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}
}
```

Obtain the dataset here.

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

### 1.18.3 Places365 Dataset

```
 @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}
 }
```

Prepare the data from Places365.

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

# 1.19 Matting Datasets

It is recommended to symlink the dataset root to `$MMEDITING/data`. If your folder structure is different, you may need to change the corresponding paths in config files.

MMEditing supported matting datasets:

- *Composition-1k* [ Homepage ]

## 1.19.1 Composition-1k Dataset

### Introduction

```
@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}
}
```

The Adobe Composition-1k dataset consists of foreground images and their corresponding alpha images. To get the full dataset, one need to composite the foregrounds with selected backgrounds from the COCO dataset and the Pascal VOC dataset.

### Obtain and Extract

Please follow the instructions of paper authors to obtain the Composition-1k (comp1k) dataset.

### Composite the full dataset

The Adobe composition-1k dataset contains only `alpha` and `fg` (and `trimap` in test set). It is needed to merge `fg` with COCO data (training) or VOC data (test) before training or evaluation. Use the following script to perform image composition and generate annotation files for training or testing:

```
## The script is run under the root folder of MMEditing
python tools/data/matting/comp1k/preprocess_comp1k_dataset.py data/adobe_composition-1k␣
↪data/coco data/VOCdevkit --composite
```

The generated data is stored under `adobe_composition-1k/Training_set` and `adobe_composition-1k/Test_set` respectively. If you only want to composite test data (since compositing training data is time-consuming), you can skip compositing the training set by removing the `--composite` option:

```
## skip compositing training set
python tools/data/matting/comp1k/preprocess_comp1k_dataset.py data/adobe_composition-1k␣
↪data/coco data/VOCdevkit
```

If you only want to preprocess test data, i.e. for FBA, you can skip the train set by adding the `--skip-train` option:

```
## skip preprocessing training set
python tools/data/matting/comp1k/preprocess_comp1k_dataset.py data/adobe_composition-1k␣
↪data/coco data/VOCdevkit --skip-train
```

(continues on next page)

(continued from previous page)

Currently, GCA and FBA support online composition of training data. But you can modify the data pipeline of other models to perform online composition instead of loading composited images (we called it `merged` in our data pipeline).

### Check Directory Structure for DIM

The result folder structure should look like:

```
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
│   │   │   ├── 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
```

### Prepare the dataset for FBA

FBA adopts dynamic dataset augmentation proposed in Learning-base Sampling for Natural Image Matting. In addition, to reduce artifacts during augmentation, it uses the extended version of foreground as foreground. We provide scripts to estimate foregrounds.

Prepare the test set as follows:

```
## skip preprocessing training set, as it composites online during training
python tools/data/matting/comp1k/preprocess_comp1k_dataset.py data/adobe_composition-1k␣
↪data/coco data/VOCdevkit --skip-train
```

Extend the foreground of training set as follows:

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

### Check Directory Structure for DIM

The final folder structure should look like:

```
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
```

## 1.20 Super-Resolution Datasets

It is recommended to symlink the dataset root to `$MMEDITING/data`. If your folder structure is different, you may need to change the corresponding paths in config files.

MMEditing supported super-resolution datasets:

- Image Super-Resolution
  - *DIV2K* [ Homepage ]
- Video Super-Resolution
  - *REDS* [ Homepage ]
  - *Vimeo90K* [ Homepage ]

### 1.20.1 DF2K_OST Dataset

```
@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}
}
```

- The DIV2K dataset can be downloaded from here (We use the training set only).
- The Flickr2K dataset can be downloaded here (We use the training set only).
- The OST dataset can be downloaded here (We use the training set OutdoorSceneTrain_v2 only).

Please first put all the images into the `GT` folder (naming does not need to be in order):

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

#### Crop sub-images

For faster IO, we recommend to crop the images to sub-images. We provide such a script:

```
python tools/data/super-resolution/df2k_ost/preprocess_df2k_ost_dataset.py --data-root ./
↪data/df2k_ost
```

The generated data is stored under `df2k_ost` and the data structure is as follows, where `_sub` indicates the sub-images.

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

#### Prepare LMDB dataset for DF2K_OST

If you want to use LMDB datasets for faster IO speed, you can make LMDB files by:

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

### 1.20.2 DIV2K Dataset

```
@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}
}
```

- Training dataset: DIV2K dataset.
- Validation dataset: Set5 and Set14.

```
mmediting
├── mmedit
├── tools
├── configs
├── data
    ├── DIV2K
        ├── DIV2K_train_HR
        ├── DIV2K_train_LR_bicubic
            ├── X2
            ├── X3
            ├── X4
```

(continues on next page)

(continued from previous page)

```
│   │   ├── DIV2K_valid_HR
│   │   ├── DIV2K_valid_LR_bicubic
│   │   │   ├── X2
│   │   │   ├── X3
│   │   │   └── X4
│   ├── val_set5
│   │   ├── Set5_bicLRx2
│   │   ├── Set5_bicLRx3
│   │   └── Set5_bicLRx4
│   ├── val_set14
│   │   ├── Set14_bicLRx2
│   │   ├── Set14_bicLRx3
│   │   └── Set14_bicLRx4
```

### Crop sub-images

For faster IO, we recommend to crop the DIV2K images to sub-images. We provide such a script:

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

The generated data is stored under `DIV2K` and the data structure is as follows, where `_sub` indicates the sub-images.

```
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
│   │   ├── ...
...
```

#### Prepare annotation list

If you use the annotation mode for the dataset, you first need to prepare a specific `txt` file.

Each line in the annotation file contains the image names and image shape (usually for the ground-truth images), separated by a white space.

Example of an annotation file:

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

#### Prepare LMDB dataset for DIV2K

If you want to use LMDB datasets for faster IO speed, you can make LMDB files by:

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

### 1.20.3 REDS Dataset

```
@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}
}
```

- Training dataset: REDS dataset.
- Validation dataset: REDS dataset and Vid4.

Note that we merge train and val datasets in REDS for easy switching between REDS4 partition (used in EDVR) and the official validation partition. The original val dataset (clip names from 000 to 029) are modified to avoid conflicts with training dataset (total 240 clips). Specifically, the clip names are changed to 240, 241, … 269.

You can prepare the REDS dataset by running:

```
python tools/data/super-resolution/reds/preprocess_reds_dataset.py --root-path ./data/
↪REDS
```

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

(continues on next page)

(continued from previous page)

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

### Prepare LMDB dataset for REDS

If you want to use LMDB datasets for faster IO speed, you can make LMDB files by:

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

### Crop to sub-images

MMEditing also support cropping REDS images to sub-images for faster IO. We provide such a script:

```
python tools/data/super-resolution/reds/crop_sub_images.py --data-root ./data/REDS  -
↪scales 4
```

The generated data is stored under `REDS` and the data structure is as follows, where `_sub` indicates the sub-images.

```
mmediting
├── mmedit
├── tools
├── configs
├── data
│   ├── REDS
│   │   ├── train_sharp
│   │   │   ├── 000
│   │   │   ├── 001
│   │   │   ├── ...
│   │   ├── train_sharp_sub
│   │   │   ├── 000_s001
│   │   │   ├── 000_s002
│   │   │   ├── ...
│   │   │   ├── 001_s001
│   │   │   ├── ...
│   │   ├── train_sharp_bicubic
│   │   │   ├── X4
│   │   │   │   ├── 000
│   │   │   │   ├── 001
│   │   │   │   ├── ...
│   │   │   ├── X4_sub
│   │   │   ├── 000_s001
│   │   │   ├── 000_s002
```

(continues on next page)

(continued from previous page)

```
│   │   │   ├── ...
│   │   │   ├── 001_s001
│   │   │   ├── ...
```

Note that by default `preprocess_reds_dataset.py` does not make lmdb and annotation file for the cropped dataset. You may need to modify the scripts a little bit for such operations.

### 1.20.4 Vid4 Dataset

```
@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}
}
```

The Vid4 dataset can be downloaded from here. There are two degradations in the dataset.

1. BIx4 contains images downsampled by bicubic interpolation
2. BDx4 contains images blurred by Gaussian kernel with =1.6, followed by a subsampling every four pixels.

### 1.20.5 Vimeo90K Dataset

```
@article{xue2019video,
  title={Video Enhancement with Task-Oriented Flow},
  author={Xue, Tianfan and Chen, Baian and Wu, Jiajun and Wei, Donglai and Freeman,␣
↪William T},
  journal={International Journal of Computer Vision (IJCV)},
  volume={127},
  number={8},
  pages={1106--1125},
  year={2019},
  publisher={Springer}
}
```

The training and test datasets can be download from here.

The Vimeo90K dataset has a `clip/sequence/img` folder structure:

```
├── GT/LQ
│   ├── 00001
│   │   ├── 0001
│   │   │   ├── im1.png
│   │   │   ├── im2.png
│   │   │   ├── ...
│   │   ├── 0002
```

(continues on next page)

(continued from previous page)

```
│   │   ├── 0003
│   │   ├── ...
│   ├── 00002
│   ├── ...
```

### Prepare the annotation files for Vimeo90K dataset

To prepare the annotation file for training, you need to download the official training list path for Vimeo90K from the official website, and run the following command:

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

The annotation file for test is generated similarly.

### Prepare LMDB dataset for Vimeo90K

If you want to use LMDB datasets for faster IO speed, you can make LMDB files by:

```
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
```

## 1.21 Generation Datasets

It is recommended to symlink the dataset root to `$MMEDITING/data`. If your folder structure is different, you may need to change the corresponding paths in config files.

MMEditing supported generation datasets:

- *Paired Dataset for Pix2pix* [ Homepage ]
- *Unpaired Dataset for CycleGAN* [ Homepage ]

### 1.21.1 Paired Dataset for Pix2pix

```
@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}
}
```

You can download paired datasets from here. Then, you need to unzip and move corresponding datasets to follow the folder structure shown above. The datasets have been well-prepared by the original authors.

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

### 1.21.2 Unpaired Dataset for CycleGAN

```
@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}
}
```

You can download unpaired datasets from here. Then, you need to unzip and move corresponding datasets to follow the folder structure shown above. The datasets have been well-prepared by the original authors.

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

## 1.22 Useful tools

We provide lots of useful tools under `tools/` directory.

### 1.22.1 Get the FLOPs and params (experimental)

We provide a script adapted from flops-counter.pytorch to compute the FLOPs and params of a given model.

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

For example,

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

You will get the result like this.

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

**Note**: This tool is still experimental and we do not guarantee that the number is correct. You may well use the result for simple comparisons, but double check it before you adopt it in technical reports or papers.

(1) FLOPs are related to the input shape while parameters are not. The default input shape is (1, 3, 250, 250). (2) Some operators are not counted into FLOPs like GN and custom operators. You can add support for new operators by modifying `mmcv/cnn/utils/flops_counter.py`.

### 1.22.2 Publish a model

Before you upload a model to AWS, you may want to (1) convert model weights to CPU tensors, (2) delete the optimizer states and (3) compute the hash of the checkpoint file and append the hash id to the filename.

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

E.g.,

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

The final output filename will be `example_model_20200202-{hash id}.pth`.

### 1.22.3 Convert to ONNX (experimental)

We provide a script to convert model to ONNX format. The converted model could be visualized by tools like Netron. Besides, we also support comparing the output results between Pytorch and ONNX model.

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

Description of arguments:

- `config` : The path of a model config file.
- `checkpoint` : The path of a model checkpoint file.
- `model_type` :The model type of the config file, options: `inpainting`, `mattor`, `restorer`, `synthesizer`.
- `image_path` : path to input image file.
- `--trimap-path` : path to input trimap file, used in mattor model.
- `--output-file`: The path of output ONNX model. If not specified, it will be set to `tmp.onnx`.
- `--opset-version` : ONNX opset version, default to 11.
- `--show`: Determines whether to print the architecture of the exported model. If not specified, it will be set to `False`.
- `--verify`: Determines whether to verify the correctness of an exported model. If not specified, it will be set to `False`.
- `--dynamic-export`: Determines whether to export ONNX model with dynamic input and output shapes. If not specified, it will be set to `False`.

**Note**: This tool is still experimental. Some customized operators are not supported for now. And we only support `mattor` and `restorer` for now.

#### List of supported models exportable to ONNX

The table below lists the models that are guaranteed to be exportable to ONNX and runnable in ONNX Runtime.

**Notes**:

- *All models above are tested with Pytorch==1.6.0 and onnxruntime==1.5.1*
- If you meet any problem with the listed models above, please create an issue and it would be taken care of soon. For models not included in the list, please try to solve them by yourself.
- Because this feature is experimental and may change fast, please always try with the latest `mmcv` and `mmedit`.

### 1.22.4 Convert ONNX to TensorRT (experimental)

We also provide a script to convert ONNX model to TensorRT format. Besides, we support comparing the output results between ONNX and TensorRT model.

```
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
```

Description of arguments:

- `config` : The path of a model config file.
- `model_type` :The model type of the config file, options: `inpainting`, `mattor`, `restorer`, `synthesizer`.
- `img_path` : The path to input image file.
- `onnx_file` : The path to input ONNX file.
- `--trt-file` : The path of output TensorRT model. If not specified, it will be set to `tmp.trt`.
- `--max-shape` : Maximum shape of model input.
- `--min-shape` : Minimum shape of model input.
- `--workspace-size`: Max workspace size in GiB. If not specified, it will be set to 1 GiB.
- `--fp16`: Determines whether to export TensorRT with fp16 mode. If not specified, it will be set to `False`.
- `--show`: Determines whether to show the output of ONNX and TensorRT. If not specified, it will be set to `False`.
- `--verify`: Determines whether to verify the correctness of an exported model. If not specified, it will be set to `False`.
- `--verbose`: Determines whether to verbose logging messages while creating TensorRT engine. If not specified, it will be set to `False`.

**Note**: This tool is still experimental. Some customized operators are not supported for now. We only support `restorer` for now. While generating ONNX file of SRCNN, replace 'bicubic' with 'bilinear' in SCRNN model here. For TensorRT does not support bicubic interpolation by now and final performance will be weaken by about 4%.

#### List of supported models exportable to TensorRT

The table below lists the models that are guaranteed to be exportable to TensorRT engine and runnable in TensorRT.

**Notes**:

- *All models above are tested with Pytorch==1.8.1, onnxruntime==1.7.0 and tensorrt==7.2.3.4*
- If you meet any problem with the listed models above, please create an issue and it would be taken care of soon. For models not included in the list, please try to solve them by yourself.
- Because this feature is experimental and may change fast, please always try with the latest `mmcv` and `mmedit`.

### 1.22.5 Evaluate ONNX and TensorRT Models (experimental)

We provide methods to evaluate TensorRT and ONNX models in `tools/deploy_test.py`.

#### Prerequisite

To evaluate ONNX and TensorRT models, onnx, onnxruntime and TensorRT should be installed first. Install `mmcv-full` with ONNXRuntime custom ops and TensorRT plugins follow ONNXRuntime in mmcv and TensorRT plugin in mmcv.

### Usage

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

### Description of all arguments

- `config`: The path of a model config file.
- `model`: The path of a TensorRT or an ONNX model file.
- `backend`: The backend for testing, choose tensorrt or onnxruntime.
- `--out`: The path of output result file in pickle format.
- `--save-path`: The path to store images and if not given, it will not save image.
- `--cfg-options`: Override some settings in the used config file, the key-value pair in `xxx=yyy` format will be merged into config file.

### Results and Models

**Notes**:

- All ONNX and TensorRT models are evaluated with dynamic shape on the datasets and images are preprocessed according to the original config file.
- This tool is still experimental, and we only support `restorer` for now.

## 1.23 mmedit.core

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

Distributed evaluation hook.

**Parameters**

- **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.

**Parameters** **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.

**Parameters**

- **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.

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

**evaluate**(*runner*, *results*)
Evaluation function.

**Parameters**

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

class mmedit.core.**L1Evaluation**
L1 evaluation metric.

**Parameters** **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'.

**Parameters**

- **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.

**Parameters**

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

**Returns** Current learning rate.

**Return type** float

class mmedit.core.**MMEditVisualizationHook**(*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.

**Parameters**

- **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.

**Parameters** **runner** (*object*) – The runner.

**class** mmedit.core.**ReduceLrUpdaterHook**(*val_metric: Optional[str] = None*, *mode: str = 'min'*, *factor: float = 0.1*, *patience: int = 10*, *threshold: float = 0.0001*, *threshold_mode: str = 'rel'*, *cooldown: int = 0*, *min_lr: float = 0.0*, *eps: float = 1e-08*, *verbose: bool = False*, *epoch_base_valid=None*, ***kwargs*)

ReduceLROnPlateau Scheduler.

Reduce learning rate when a metric has stopped improving. This scheduler reads a metrics quantity and if no improvement is seen for a 'patience' number of epochs, the learning rate is reduced.

**Parameters**

- **val_metric**(*str, optional*) – Metrics to be evaluated. If val_metric is None, the metrics will be loss value. Default: None.
- **mode** (*str, optional*) – One of *min*, *max*. In *min* mode, lr will be reduced when the quantity monitored has stopped decreasing; in *max* mode it will be reduced when the quantity monitored has stopped increasing. Default: 'min'.
- **factor** (*float, optional*) – Factor by which the learning rate will be reduced. new_lr = lr * factor. Default: 0.1.
- **patience** (*int, optional*) – Number of epochs with no improvement after which learning rate will be reduced. For example, if *patience = 2*, then we will ignore the first 2 epochs with no improvement, and will only decrease the LR after the 3rd epoch if the loss still hasn't improved then. Default: 10.
- **threshold**(*float, optional*) – Threshold for measuring the new optimum, to only focus on significant changes. Default: 1e-4.
- **threshold_mode** (*str, optional*) – One of *rel*, *abs*. In *rel* mode, dynamic_threshold = best * ( 1 + threshold ) in 'max' mode or best * ( 1 - threshold ) in *min* mode. In *abs* mode, dynamic_threshold = best + threshold in *max* mode or best - threshold in *min* mode. Default: 'rel'.

- **cooldown** (*int, optional*) – Number of epochs to wait before resuming normal operation after lr has been reduced. Default: 0.
- **min_lr** (*float, optional*) – Minimum LR value to keep. If LR after decay is lower than *min_lr*, it will be clipped to this value. Default: 0.
- **eps** (*float, optional*) – Minimal decay applied to lr. If the difference between new and old lr is smaller than eps, the update is ignored. Default: 1e-8.
- **verbose** (*bool*) – If `True`, prints a message to stdout for each update. Default: `False`.
- **epoch_base_valid** (*None | Bool*) – Whether use epoch base valid. If *None*, follow *by_epoch* (inherited from *LrUpdaterHook*). Default: None.

class mmedit.core.**VisualizationHook**(**args*, ***kwargs*)

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`.

**Parameters**

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

**Returns** The initialized optimizers.

**Return type** dict[`torch.optim.Optimizer`] | `torch.optim.Optimizer`

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

Calculate mean average error for evaluation.

**Parameters**

- **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. Options are 'RGB2Y', 'BGR2Y' and None. Default: None.

**Returns** mae result.

**Return type** float

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

**Parameters**

- **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.

**Returns** psnr result.

**Return type** 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.

**Parameters**

- **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'.

**Returns** reordered image.

**Return type** 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.

**Parameters**

- **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.

**Returns** ssim result.

**Return type** 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.

**Parameters**

- **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.

**Returns** 3D ndarray of shape (H x W x C) or 2D ndarray of shape (H x W).

**Return type** (Tensor | list[Tensor])

## 1.24 mmedit.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 annotations for Adobe Composition-1k dataset.

It loads image paths from json file.

**Returns** Loaded dict.

**Return type** 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.

**Parameters**

- **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.

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

**Returns** Returned testing batch.

**Return type** Tensor

**prepare_train_data**(*idx*)
Prepare training data.

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

**Returns** Returned training batch.

**Return type** 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)

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

**Returns** Evaluation results dict.

**Return type** dict

**static scan_folder**(*path*)
Obtain image path list (including sub-folders) from a given folder.

**Parameters** **path** (str | Path) – Folder path.

**Returns** Image list obtained from the given folder.

**Return type** 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.

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

**Returns** Evaluation results dict.

**Return type** dict

**class** mmedit.datasets.**BaseSRDataset**(*pipeline*, *scale*, *test_mode=False*)
Base class for super resolution datasets.

**evaluate**(*results*, *logger=None*)
Evaluate with different metrics.

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

**Returns** Evaluation results dict.

**Return type** dict

**static scan_folder**(*path*)
Obtain image path list (including sub-folders) from a given folder.

**Parameters** **path** (str | Path) – Folder path.

**Returns** image list obtained form given folder.

**Return type** list[str]

**class** mmedit.datasets.**BaseVFIDataset**(*pipeline*, *folder*, *ann_file*, *test_mode=False*)
Base class for video frame interpolation datasets.

**evaluate**(*results*, *logger=None*)
Evaluate with different metrics.

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

**Returns** Evaluation results dict.

**Return type** dict

**load_annotations**()
Abstract function for loading annotation.

All subclasses should overwrite this function

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).

**Parameters**

- **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.

**Returns** List that contains paired image paths.

**Return type** 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.

**Parameters**

- **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.

**Parameters** **dataroot** (*str*) – Path to the folder root for unpaired images of one domain.

**Returns** List that contains unpaired image paths of one domain.

**Return type** list[dict]

prepare_test_data(*idx*)
Prepare unpaired test data.

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

**Returns** Prepared test data batch.

**Return type** list[dict]

prepare_train_data(*idx*)
Prepare unpaired training data.

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

**Returns** Prepared training data batch.

**Return type** dict

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

load_annotations()
Load annotations for dataset.

**Returns** Contain dataset annotations.

**Return type** 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.

**Parameters**

- **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)
```

**Parameters**

- **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 annotations 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.

**Returns** A list of dicts for paired paths of LQ and GT.

**Return type** list[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=*)
```

**Parameters**

- **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 annotations 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.

**Returns**

**A list of dicts for GT path and landmark.** Contains: gt_path, bbox, shape, landmark.

**Return type** list[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'
```

**Parameters**

- **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 annotations for SR dataset.

It loads the LQ and GT image path from folders.

**Returns** A list of dicts for paired paths of LQ and GT.

**Return type** list[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
```

**Parameters**

- **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 annotations for SR dataset.

It loads the GT image path from folder.

**Returns** A list of dicts for path of GT.

**Return type** list[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
```

**Parameters**

- **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 annotations for the dataset.

**Returns** Returned list of dicts for paired paths of LQ and GT.

**Return type** list[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'
```

**Parameters**

- **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 annotations for SR dataset.

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

**Returns** A list of dicts for paired paths of ref, LQ and GT.

**Return type** list[dict]

**class** mmedit.datasets.**SRFolderVideoDataset**(*lq_folder*, *gt_folder*, *num_input_frames*, *pipeline*, *scale*, *ann_file=None*, *filename_tmpl='{:08d}'*, *start_idx=0*, *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
...
```

**Parameters**

- **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}'.
- **start_idx** (*int*) – The index corresponds to the first frame in the sequence. Default: 0.
- **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.

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

**Returns** Evaluation results dict.

**Return type** dict

**load_annotations**()

Load annotations for the dataset.

**Returns** A list of dicts for paired paths and other information.

**Return type** list[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.

**Parameters**

- **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 annotations for SR dataset.

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

**Returns** A list of dicts for paired paths of LQ and GT.

**Return type** list[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)
```

**Parameters**

- **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 annotations for REDS dataset.

**Returns** A list of dicts for paired paths and other information.

**Return type** list[dict]

class mmedit.datasets.**SRREDSMultipleGTDataset**(*lq_folder*, *gt_folder*, *num_input_frames*, *pipeline*, *scale*, *val_partition='official'*, *repeat=1*, *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.

**Parameters**

- **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’.
- **repeat** (*int*) – Number of replication of the validation set. This is used to allow training REDS4 with more than 4 GPUs. For example, if 8 GPUs are used, this number can be set to 2. Default: 1.
- **test_mode** (*bool*) – Store *True* when building test dataset. Default: *False*.

**load_annotations**()

Load annotations for REDS dataset.

**Returns** A list of dicts for paired paths and other information.

**Return type** list[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.

**Parameters**

- **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.

**Returns** A list of dicts for paired paths and other information.

**Return type** list[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)
```

**Parameters**

- **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.

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

**Returns** Evaluation results dict.

**Return type** dict

**load_annotations**()
Load annotations for Vid4 dataset.

**Returns** A list of dicts for paired paths and other information.

**Return type** list[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)
```

**Parameters**

- **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 annotations for VimeoK dataset.

**Returns** A list of dicts for paired paths and other information.

**Return type** list[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)
```

**Parameters**

- **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 annotations for Vimeo-90K dataset.

**Returns** A list of dicts for paired paths and other information.

**Return type** list[dict]

class mmedit.datasets.**VFIVimeo90K7FramesDataset**(*folder*, *ann_file*, *pipeline*, *input_frames*, *target_frames*, *test_mode=False*)

Utilize Vimeo90K dataset (7 frames) for video frame interpolation.

Load 7 GT (Ground-Truth) frames from the dataset, predict several frame(s) from other 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. number of frames
3. image shape

Examples:

```
00001/0266 7 (256,448,3)
00001/0268 7 (256,448,3)
```

Note: Only *video frame folder* is required information.

**Parameters**

- **folder** (str | Path) – Path to image folder.
- **ann_file** (str | Path) – Path to the annotation file.
- **pipeline** (*list[dict | callable]*) – A sequence of data transformations.
- **input_frames** (*list[int]*) – Index of input frames.
- **target_frames** (*list[int]*) – Index of target frames.
- **test_mode** (*bool*) – Store *True* when building test dataset. Default: *False*.

load_annotations()
Load annoations for Vimeo-90K dataset.

**Returns** A list of dicts for paired paths and other information.

**Return type** list[dict]

class mmedit.datasets.**VFIVimeo90KDataset**(*pipeline*, *folder*, *ann_file*, *test_mode=False*)
Vimeo90K dataset for video frame interpolation.

The dataset loads two input 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:

Examples:

```
00001/0389
00001/0402
```

**Parameters**

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

load_annotations()
Load annotations for VimeoK dataset.

**Returns** A list of dicts for paired paths and other information.

**Return type** list[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.

**Parameters**

- **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.

**Returns** A PyTorch dataloader.

**Return type** 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`.

**Parameters**

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

**Returns** The constructed dataset.

**Return type** Dataset

## 1.25 mmedit.datasets.pipelines

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

**Parameters**

- **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".

**Parameters**

- **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.**ColorJitter**(*keys*, *channel_order='rgb'*, ***kwargs*)
An interface for torch color jitter so that it can be invoked in mmediting pipeline.

Randomly change the brightness, contrast and saturation of an image. Modified keys are the attributes specified in "keys".

**Parameters**

- **keys** (*list[str]*) – The images to be resized.
- **channel_order** (*str*) – Order of channel, candidates are 'bgr' and 'rgb'. Default: 'rgb'.

**Notes: **kwards follows the args list of** `torchvision.transforms.ColorJitter`.

**brightness (float or tuple of float (min, max)): How much to jitter** brightness. brightness_factor is chosen uniformly from [max(0, 1 - brightness), 1 + brightness] or the given [min, max]. Should be non negative numbers.

**contrast (float or tuple of float (min, max)): How much to jitter** contrast. contrast_factor is chosen uniformly from [max(0, 1 - contrast), 1 + contrast] or the given [min, max]. Should be non negative numbers.

**saturation (float or tuple of float (min, max)): How much to jitter** saturation. saturation_factor is chosen uniformly from [max(0, 1 - saturation), 1 + saturation] or the given [min, max]. Should be non negative numbers.

**hue (float or tuple of float (min, max)): How much to jitter hue.** hue_factor is chosen uniformly from [-hue, hue] or the given [min, max]. Should have 0<= hue <= 0.5 or -0.5 <= min <= max <= 0.5.

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

Compose a data pipeline with a sequence of transforms.

**Parameters** **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 $(fg_1, \alpha_1)$ is from the current sample and $(fg_2, \alpha_2)$ is the randomly loaded sample. With the above composition, $\alpha_{new}$ is still in *[0, 1]*.

Required keys are "alpha" and "fg". Modified keys are "alpha" and "fg".

**Parameters**

- **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".

**Parameters**

- **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*, *is_pad_zeros=False*)

Crop data to specific size for training.

**Parameters**

- **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.
- **is_pad_zeros** (*bool, optional*) – Whether to pad the image with 0 if crop_size is greater than image size. Default: False.

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".

**Parameters** **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*.

**Parameters**

- **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.

**Parameters**

- **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 .

**Parameters**

- **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.**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".

**Parameters**

- **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.

**Parameters**

- **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.

**Parameters**

- **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".

**Parameters** **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.

**Parameters**

- **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

**Parameters**

- **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

**Parameters** **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.

**Parameters**

- **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.

**Parameters**

- **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

**Parameters**

- **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".

**Parameters**

- **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".

**Parameters**

- **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.

**Parameters**

- **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.

**Parameters**

- **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.

**Parameters**

- **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.

**Parameters**

- **mask_width** (*int*) – The width of bbox hole.

- **mask_height** (*int*) – The height of bbox height.

**Returns** Spatial discounting mask.

**Return type** np.ndarray

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

Convert image type to *torch.Tensor* type.

**Parameters**

- **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.

**Parameters**

- **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.

**Parameters**

- **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.**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.
```

**Parameters**

- **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".

**Parameters**

- **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".

**Parameters**

- **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).

**Parameters** **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.

**Parameters**

- **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.

**Parameters**

- **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".

**Parameters** **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".

**Parameters** **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".

**Parameters** **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".

**Parameters**

- **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".

**Parameters**

- **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.

**Parameters**

- **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".

**Parameters**

- **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".

**Parameters** **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".

**Parameters**

- **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.

**Parameters**

- **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".

**Parameters**

- **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".

**Parameters**

- **params** (*dict*) – A dictionary specifying the degradation settings.
- **keys** (*list[str]*) – A list specifying the keys whose values are modified.

class mmedit.datasets.pipelines.**RandomResizedCrop**(*keys*, *crop_size*, *scale=(0.08, 1.0)*, *ratio=(0.75, 1.3333333333333333)*, *interpolation='bilinear'*)
Crop data to random size and aspect ratio.

A crop of a random proportion of the original image and a random aspect ratio of the original aspect ratio is made. The cropped image is finally resized to a given size specified by 'crop_size'. Modified keys are the attributes specified in "keys".

This code is partially adopted from torchvision.transforms.RandomResizedCrop: [https://pytorch.org/vision/stable/_modules/torchvision/transforms/ transforms.html#RandomResizedCrop].

**Parameters**

- **keys** (*list[str]*) – The images to be resized and random-cropped.
- **crop_size** (*int | tuple[int]*) – Target spatial size (h, w).
- **scale** (*tuple[float], optional*) – Range of the proportion of the original image to be cropped. Default: (0.08, 1.0).
- **ratio** (*tuple[float], optional*) – Range of aspect ratio of the crop. Default: (3. / 4., 4. / 3.).
- **interpolation** (*str, optional*) – Algorithm used for interpolation. It can be only either one of the following: "nearest" | "bilinear" | "bicubic" | "area" | "lanczos". Default: "bilinear".

**get_params**(*data*)
Get parameters for a random sized crop.

**Parameters** **data** (*np.ndarray*) – Image of type numpy array to be cropped.

**Returns** A tuple containing the coordinates of the top left corner and the chosen crop size.

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".

**Parameters**

- **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".

**Parameters**

- **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.

**Parameters** **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.

**Parameters**

- **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".

**Parameters**

- **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.

**Parameters** **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.

**Parameters**

- **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".

## 1.26 mmedit.models

### 1.26.1 models

class mmedit.models.**AOTInpaintor**(*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 AOT-GAN method.

This inpaintor is implemented according to the paper: Aggregated Contextual Transformations for High-Resolution Image Inpainting

**forward_test**(*masked_img*, *mask*, *save_image=False*, *save_path=None*, *iteration=None*, ***kwargs*)
Forward function for testing.

**Parameters**

- **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. Default: False.
- **save_path** (*str, optional*) – If given a valid str, the reuslts will be saved in this path. Default: None.
- **iteration** (*int, optional*) – Iteration number. Default: None.

**Returns** Contain output results and eval metrics (if exist).

**Return type** dict

**forward_train_d**(*data_batch*, *is_real*, *is_disc*, *mask*)
Forward function in discriminator training step.

In this function, we compute the prediction for each data batch (real or fake). Meanwhile, the standard gan loss will be computed with several proposed losses for stable training.

**Parameters**

- **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.
- **mask** (*torch.Tensor*) – Mask of data.

**Returns** Contains the loss items computed in this function.

**Return type** 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.

**Parameters**

- **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.

**Returns**

**Dict contains the results computed within this** function for visualization and dict contains the loss items computed in this function.

**Return type** 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. compute reconstruction losses for generator 3. compute adversarial loss for discriminator 4. optimize generator 5. optimize discriminator

**Parameters**

- **data_batch** (*torch.Tensor*) – Batch of data as input.
- **optimizer** (*dict[torch.optim.Optimizer]*) – Dict with optimizers for generator and discriminator (if exist).

**Returns**

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

**Return type** dict

**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.

**Parameters**

- **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`.

**Parameters**

- **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`.

**Returns** The evaluation result.

**Return type** dict

**forward**(*merged*, *trimap*, *meta*, *alpha=None*, *test_mode=False*, ***kwargs*)
Defines the computation performed at every call.

**Parameters**

- **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.

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

**Return type** 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.

**Parameters**

- **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.

**Parameters** **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.

**Parameters**

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

**Returns** The reshaped predicted alpha.

**Return type** np.ndarray

**save_image**(*pred_alpha*, *meta*, *save_path*, *iteration*)
Save predicted alpha to file.

**Parameters**

- **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.

**Parameters**

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

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

**Return type** 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.

**Parameters**

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

**Returns** Forward results.

**Return type** 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.

**Parameters** **losses** (*dict*) – Loss dict.

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

**Return type** 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.**BasicInterpolator**(*generator*, *pixel_loss*, *train_cfg=None*, *test_cfg=None*, *required_frames=2*, *step_frames=1*, *pretrained=None*)
Basic model for video interpolation.

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

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

**Parameters**

- **generator** (*dict*) – Config for the generator structure.
- **pixel_loss** (*dict*) – Config for pixel-wise loss.
- **train_cfg** (*dict*) – Config for training. Default: None.
- **test_cfg** (*dict*) – Config for testing. Default: None.
- **required_frames** (*int*) – Required frames in each process. Default: 2
- **step_frames** (*int*) – Step size of video frame interpolation. Default: 1
- **pretrained** (*str*) – Path for pretrained model. Default: None.

**evaluate**(*output*, *target*)
Evaluation function.

**Parameters**

- **output** (*Tensor*) – Model output.
- **target** (*Tensor*) – GT Tensor.

**Returns** Evaluation results.

**Return type** dict

**forward**(*inputs*, *target=None*, *test_mode=False*, ***kwargs*)
Forward function.

**Parameters**

- **inputs** (*Tensor*) – Tensor of input frames.
- **target** (*Tensor*) – Tensor of target frame. 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.

**Parameters** **img** (*Tensor*) – Input frames.

**Returns** Output frame(s).

**Return type** Tensor

**forward_test**(*inputs*, *target=None*, *meta=None*, *save_image=False*, *save_path=None*, *iteration=None*)
Testing forward function.

This is a basic function, interpolate a frame between the given two frames.

**Parameters**

- **inputs** (*Tensor*) – Tensor of input frames.
- **target** (*Tensor*) – Tensor of target frame(s). Default: None.
- **meta** (*list[dict]*) – Meta data, such as path of target file. 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.

**Returns** Output results.

**Return type** dict

**forward_train**(*inputs*, *target*)
Training forward function.

This is a basic function, interpolate a frame between the given two frames.

**Parameters**

- **inputs** (*Tensor*) – Tensor of input frame(s).
- **target** (*Tensor*) – Tensor of target frame(s).

**Returns** Output tensor.

**Return type** Tensor

**init_weights**(*pretrained=None*)
Init weights for models.

**Parameters** **pretrained** (*str, optional*) – Path for pretrained weights. If given None, pretrained weights will not be loaded. Defaults to None.

**static merge_frames**(*input_tensors*, *output_tensors*)
merge input frames and output frames.

This is a basic function, interpolate a frame between the given two frames.

**Parameters**

- **input_tensors** (*Tensor*) – The input frames with shape [n, 2, c, h, w]
- **output_tensors** (*Tensor*) – The output frames with shape [n, 1, c, h, w].

**Returns**

**The final frames.** in_frame, out_frame, in_frame, out_frame, in_frame ...

**Return type** list[np.array]

**split_frames**(*input_tensors*)
split input tensors for inference.

**Parameters** **input_tensors** (*Tensor*) – Tensor of input frames with shape [1, t, c, h, w]

**Returns** Split tensor with shape [t-1, 2, c, h, w]

**Return type** Tensor

**train_step**(*data_batch*, *optimizer*)
Train step.

**Parameters**

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

**Returns** Returned output.

**Return type** dict

**val_step**(*data_batch*, ***kwargs*)
Validation step.

**Parameters**

- **data_batch** (*dict*) – A batch of data.
- **kwargs** (*dict*) – Other arguments for `val_step`.

**Returns** Returned output.

**Return type** dict

**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*.

**Parameters**

- **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.

**Parameters**

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

**Returns** Evaluation results.

**Return type** dict

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

**Parameters**

- **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.

**Parameters** **img** (*Tensor*) – Input image.

**Returns** Output image.

**Return type** Tensor

**forward_test**(*lq*, *gt=None*, *meta=None*, *save_image=False*, *save_path=None*, *iteration=None*)
Testing forward function.

**Parameters**

- **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.

**Returns** Output results.

**Return type** dict

**forward_train**(*lq*, *gt*)
Training forward function.

**Parameters**

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

**Returns** Output tensor.

**Return type** Tensor

**init_weights**(*pretrained=None*)
Init weights for models.

**Parameters** **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.

**Parameters**

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

**Returns** Returned output.

**Return type** dict

**val_step**(*data_batch*, ***kwargs*)
Validation step.

**Parameters**

- **data_batch** (*dict*) – A batch of data.
- **kwargs** (*dict*) – Other arguments for `val_step`.

**Returns** Returned output.

**Return type** dict

**class** mmedit.models.**CAIN**(*generator*, *pixel_loss*, *train_cfg=None*, *test_cfg=None*, *required_frames=2*, *step_frames=1*, *pretrained=None*)
CAIN model for Video Interpolation.

Paper: Channel Attention Is All You Need for Video Frame Interpolation Ref repo: https://github.com/myungsub/CAIN

**Parameters**

- **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.

**forward_test**(*inputs*, *target=None*, *meta=None*, *save_image=False*, *save_path=None*, *iteration=None*)
Testing forward function.

**Parameters**

- **inputs** (*Tensor*) – The input Tensor with shape (n, 2, c, h, w).
- **target** (*Tensor*) – The target Tensor with shape (n, c, h, w).
- **meta** (*list[dict]*) – Meta data, such as path of target file. 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.

**Returns**

**Output results, which contain either key(s)**

1. 'eval_result'.
2. 'inputs', 'pred'.
3. 'inputs', 'pred', and 'target'.

**Return type** dict

**forward_train**(*inputs*, *target*)
Training forward function.

**Parameters**

- **inputs** (*Tensor*) – Tensor of inputs frames with shape (n, 2, c, h, w).
- **target** (*Tensor*) – Tensor of target frame with shape (n, c, h, w).

**Returns** Output tensor.

**Return type** Tensor

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

**Parameters**

- **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.

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

**Returns** Loss dict.

**Return type** dict

**backward_generators**(*outputs*)

Backward function for the generators.

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

**Returns** Loss dict.

**Return type** dict

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

Forward function.

**Parameters**

- **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.

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

**Returns** Dummy output produced by forwarding the dummy input.

**Return type** Tensor

**forward_test**(*img_a*, *img_b*, *meta*, *save_image=False*, *save_path=None*, *iteration=None*)
Forward function for testing.

**Parameters**

- **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.

**Returns** Dict of forward and evaluation results for testing.

**Return type** dict

**forward_train**(*img_a*, *img_b*, *meta*)
Forward function for training.

**Parameters**

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

**Returns** Dict of forward results for training.

**Return type** dict

**get_module**(*module*)
Get *nn.ModuleDict* to fit the *MMDistributedDataParallel* interface.

**Parameters** **module** (*MMDistributedDataParallel | nn.ModuleDict*) – The input module that needs processing.

**Returns** The ModuleDict of multiple networks.

**Return type** nn.ModuleDict

**init_weights**(*pretrained=None*)
Initialize weights for the model.

**Parameters** **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.

**Parameters**

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

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

**Return type** Tensor, Tensor, list[str]

`train_step`(*data_batch*, *optimizer*)
Training step function.

**Parameters**

- **data_batch** (*dict*) – Dict of the input data batch.
- **optimizer** (*dict[torch.optim.Optimizer]*) – Dict of optimizers for the generators and discriminators.

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

**Return type** dict

`val_step`(*data_batch*, ***kwargs*)
Validation step function.

**Parameters**

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

**Returns** Dict of evaluation results for validation.

**Return type** 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

---

**Note:** 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.

---

**Parameters**

- **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.

**Parameters**

- **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.

**Returns** Contains the predicted alpha and evaluation result.

**Return type** dict

**forward_train**(*merged*, *trimap*, *meta*, *alpha*, *ori_merged*, *fg*, *bg*)
Defines the computation performed at every training call.

**Parameters**

- **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.

**Returns** Contains the loss items and batch information.

**Return type** 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.

**Parameters**

- **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.

**Returns** Contain loss value with its name.

**Return type** 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.

**Parameters**

- **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.

**Returns** Contains the loss items computed in this function.

**Return type** 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.

**Parameters**

- **model** (*nn.Module*) – This model may wrapped with DDP or not.
- **module_name** (*str*) – The name of specific module.

**Returns** Returned sub module.

**Return type** 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.

**Parameters**

- **data_batch** (*torch.Tensor*) – Batch of data as input.
- **optimizer** (*dict[torch.optim.Optimizer]*) – Dict with optimizers for generator and discriminator (if have).

**Returns** Dict with loss, information for logger, the number of samples and results for visualization.

**Return type** dict

**two_stage_loss**(*stage1_data*, *stage2_data*, *data_batch*)
Calculate two-stage loss.

**Parameters**

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

**Returns** Contain losses with name.

**Return type** 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.

**Parameters**

- **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.

**Parameters**

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

**Returns** Returned output.

**Return type** dict

class mmedit.models.**FLAVR**(*generator*, *pixel_loss*, *train_cfg=None*, *test_cfg=None*, *pretrained=None*)
Basic model for video interpolation.

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

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

**Parameters**

- **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.

**static merge_frames**(*input_tensors*, *output_tensors*)
merge input frames and output frames.

Interpolate a frame between the given two frames.

**Merged from** [[in1, in2, in3, in4], [in2, in3, in4, in5], ...] [[out1], [out2], [out3], ...]

**to** [in1, in2, out1, in3, out2, ..., in(-3), out(-1), in(-2), in(-1)]

**Parameters**

- **input_tensors** (*Tensor*) – The input frames with shape [n, 4, c, h, w]
- **output_tensors** (*Tensor*) – The output frames with shape [n, 1, c, h, w].

**Returns** The final frames.

**Return type** list[np.array]

class mmedit.models.**FeedbackHourglass**(*mid_channels*, *num_keypoints*)
Feedback Hourglass model for face landmark.

It has a style of:

```
-- preprocessing ----- Hourglass ----->
                   ^                 |
                   |_________________|
```

**Parameters**

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

**forward**(*x*, *last_hidden=None*)
Forward function.

**Parameters**

- **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.

**Returns** Heatmap of facial landmark. feedback (Tensor): Feedback Tensor.

**Return type** 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

**Parameters**

- **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.

**Parameters**

- **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.

**Returns** Contains the predicted alpha and evaluation result.

**Return type** dict

**forward_train**(*merged*, *trimap*, *meta*, *alpha*)
Forward function for training GCA model.

**Parameters**

- **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.

**Returns** Contains the loss items and batch information.

**Return type** 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.

**Parameters**

- **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.

**Parameters**

- **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.

**Returns** 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.

**Return type** 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.

**Parameters**

- **data_batch** (*torch.Tensor*) – Batch of data as input.
- **optimizer** (*dict[torch.optim.Optimizer]*) – Dict with optimizers for generator and discriminator (if have).

**Returns** Dict with loss, information for logger, the number of samples and results for visualization.

**Return type** 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

**Parameters**

- **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.

**Parameters**

- **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.

**Returns** Contains the predicted alpha and evaluation result.

**Return type** dict

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

Forward function for training IndexNet model.

**Parameters**

- **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).

**Returns** Contains the loss items and batch information.

**Return type** 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.

**Parameters**

- **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.

**Parameters** **x** (*Tensor*) – Input tensor with shape (n, 3, h, w).

**Returns**

**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).

**Return type** Tuple[Tensor]

**init_weights**(*pretrained=None*, *strict=True*)
Init weights for models.

**Parameters**

- **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.

**Parameters**

- **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.

**Parameters**

- **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.

**Returns** Dict contains output results.

**Return type** dict

**forward_dummy**(*x*)
Forward dummy function for getting flops.

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

**Returns** Results tensor with shape of (n, 3, h, w).

**Return type** torch.Tensor

**forward_test**(*masked_img*, *mask*, *save_image=False*, *save_path=None*, *iteration=None*, ***kwargs*)
Forward function for testing.

**Parameters**

- **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.

**Returns** Contain output results and eval metrics (if have).

**Return type** 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.

**Parameters**

- **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.

**Returns** Contains the loss items computed in this function.

**Return type** 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.

**Parameters**

- **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.

**Returns** Dict contains the results computed within this function for visualization and dict contains the loss items computed in this function.

**Return type** tuple(dict)

**init_weights**(*pretrained=None*)
Init weights for models.

**Parameters** **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.

**Parameters**

- **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.

**Parameters**

- **data_batch** (*torch.Tensor*) – Batch of data as input.
- **optimizer** (*dict[torch.optim.Optimizer]*) – Dict with optimizers for generator and discriminator (if have).

**Returns** Dict with loss, information for logger, the number of samples and results for visualization.

**Return type** dict

**val_step**(*data_batch*, ***kwargs*)
Forward function for evaluation.

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

**Returns** Contain the results from model.

**Return type** 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.

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

**Returns** Results tensor with shape of (n, 3, h, w).

**Return type** torch.Tensor

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

Forward function for testing.

**Parameters**

- **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.

**Returns** Contain output results and eval metrics (if have).

**Return type** 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.

**Parameters**

- **data_batch** (*torch.Tensor*) – Batch of data as input.
- **optimizer** (*dict[torch.optim.Optimizer]*) – Dict with optimizers for generator and discriminator (if have).

**Returns** Dict with loss, information for logger, the number of samples and results for visualization.

**Return type** 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

**Parameters**

- **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.

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

**Returns** Loss dict.

**Return type** dict

**backward_generator**(*outputs*)

Backward function for the generator.

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

**Returns** Loss dict.

**Return type** dict

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

Forward function.

**Parameters**

- **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.

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

**Returns** Dummy output produced by forwarding the dummy input.

**Return type** Tensor

**forward_test**(*img_a*, *img_b*, *meta*, *save_image=False*, *save_path=None*, *iteration=None*)
Forward function for testing.

**Parameters**

- **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.

**Returns** Dict of forward and evaluation results for testing.

**Return type** dict

**forward_train**(*img_a*, *img_b*, *meta*)
Forward function for training.

**Parameters**

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

**Returns** Dict of forward results for training.

**Return type** dict

**init_weights**(*pretrained=None*)
Initialize weights for the model.

**Parameters** **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.

**Parameters**

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

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

**Return type** Tensor, Tensor, list[str]

**train_step**(*data_batch*, *optimizer*)
Training step function.

**Parameters**

- **data_batch** (*dict*) – Dict of the input data batch.
- **optimizer** (*dict[torch.optim.Optimizer]*) – Dict of optimizers for the generator and discriminator.

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

**Return type** dict

`val_step`(*data_batch*, ***kwargs*)
Validation step function.

**Parameters**

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

**Returns** Dict of evaluation results for validation.

**Return type** 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.

**Parameters**

- **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.

**Parameters**

- **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.

**Parameters** **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.

**Parameters**

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

**Returns** Returned output.

**Return type** 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.

**Parameters**

- **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), . . . ]

**Returns**

Soft-Attention tensor. (N, 1, H, W) textures (Tuple[Tensor]): Transferred GT textures.

[(N, C, H, W), (N, C/2, 2H, 2W), . . . ]

**Return type** soft_attention (Tensor)

**gather**(*inputs*, *dim*, *index*)

Hard Attention. Gathers values along an axis specified by dim.

**Parameters**

- **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.

**Parameters**

- **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.

**Parameters**

- **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_**'.

**Returns** Contain loss value with its name.

**Return type** dict

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

Forward function for testing.

**Parameters**

- **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.

**Returns** Contain output results and eval metrics (if have).

**Return type** dict

**save_visualization**(*img*, *filename*)

Save visualization results.

**Parameters**

- **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.

**Parameters**

- **data_batch** (*torch.Tensor*) – Batch of data as input.
- **optimizer** (*dict[torch.optim.Optimizer]*) – Dict with optimizers for generator and discriminator (if have).

**Returns** Dict with loss, information for logger, the number of samples and results for visualization.

**Return type** dict

**two_stage_loss**(*stage1_data*, *stage2_data*, *data_batch*)
Calculate two-stage loss.

**Parameters**

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

**Returns** Contain losses with name.

**Return type** dict

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

**Parameters**

- **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.

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

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

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

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

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

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

**Parameters**

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

### 1.26.2 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".

**Parameters**

- **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.

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

**Returns** Output tensor.

**Return type** 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

**Parameters**

- **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.

**Parameters** **attention_score** (*torch.Tensor*) – The attention score with shape of (n, c, h, w).

**Returns** The overlap factor will be returned.

**Return type** 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.

**Parameters**

- **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.

**Returns** Features after contextural attention.

**Return type** 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

**Parameters**

- **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.

**Returns** 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)*.

**Return type** 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.

**Parameters**

- **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.

**Returns** Updated correlation map with mask.

**Return type** torch.Tensor

**patch_copy_deconv**(*attention_score*, *context_filter*)
Copy patches using deconv.

**Parameters**

- **attention_score** (*torch.Tensor*) – Tensor with shape of (n, l , h, w).
- **context_filter** (*torch.Tensor*) – Filter kernel.

**Returns** Tensor with shape of (n, c, h, w).

**Return type** torch.Tensor

**patch_correlation**(*x*, *kernel*)
Calculate patch correlation.

**Parameters**

- **x** (*torch.Tensor*) – Input tensor.
- **kernel** (*torch.Tensor*) – Kernel tensor.

**Returns** Tensor with shape of (n, l, h, w).

**Return type** 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.

**Parameters**

- **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.

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

**Returns** Output tensor.

**Return type** 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.

**Parameters**

- **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.

**Parameters** **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.

**Parameters**

- **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.

**Parameters**

- **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.

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

**Return type** Tensor

**compute_similarity_map**(*img_feat*, *img_ps*)

Compute similarity between image feature patches.

**Parameters**

- **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).

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

**Return type** Tensor

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

Extract image feature, alpha feature unknown patches.

**Parameters**

- **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).

**Returns**

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).

**Return type** 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)*.

**Parameters**

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

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

**Return type** Tensor

**forward**(*img_feat*, *alpha_feat*, *unknown=None*, *softmax_scale=1.0*)
Forward function of GCAModule.

**Parameters**

- **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.

**Returns** The augmented alpha feature.

**Return type** Tensor

**process_unknown_mask**(*unknown*, *img_feat*, *softmax_scale*)
Process unknown mask.

**Parameters**

- **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.

**Returns**

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).

**Return type** tuple

**propagate_alpha_feature**(*gca_score*, *alpha_ps*)
Propagate alpha feature based on guided attention score.

**Parameters**

- **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).

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

**Return type** 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.

**Parameters**

- **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.

**Parameters**

- **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.

**Parameters**

- **x** (*torch.Tensor*) – Input tensor with shape of $(n, *, c)$. Same as `torch.nn.Linear`.
- **activate** (*bool, optional*) – Whether to use activation layer. Defaults to True.

**Returns** Same as `torch.nn.Linear`.

**Return type** 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.

**Parameters**

- **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.

**Parameters**

- **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.

**Returns**

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.

**Return type** 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]

**Parameters**

- **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.

**Parameters**

- **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.

**Returns** Results after partial conv. torch.Tensor : Updated mask will be returned if mask is given and `return_mask` is True.

**Return type** torch.Tensor

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

Pixel Shuffle upsample layer.

**Parameters**

- **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.

**Returns** Upsampled feature map.

**forward**(*x*)

Forward function for PixelShufflePack.

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

**Returns** Forward results.

**Return type** 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-+-
 |________________|
```

**Parameters**

- **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.

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

**Returns** Forward results.

**Return type** 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.

**Parameters**

- **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.

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

**Returns** Forward results.

**Return type** 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.

**Parameters**

- **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.

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

**Returns** Output tensor with shape of (n, c, h', w').

**Return type** torch.Tensor

class mmedit.models.common.**SpatialTemporalEnsemble**(*is_temporal_ensemble=False*)

Apply spatial and temporal ensemble and compute outputs.

**Parameters** **is_temporal_ensemble** (*bool, optional*) – Whether to apply ensemble temporally. If True, the sequence will also be flipped temporally. If the input is an image, this argument must be set to False. Default: False.

**forward**(*imgs*, *model*)

Apply spatial and temporal ensemble.

**Parameters**

- **imgs** (*torch.Tensor*) – The images to be processed by the model. Its size should be either (n, t, c, h, w) or (n, c, h, w).
- **model** (*nn.Module*) – The model to process the images.

**Returns** Output of the model with spatial ensemble applied.

**Return type** torch.Tensor

**spatial_ensemble**(*imgs*, *model*)

Apply spatial ensemble.

**Parameters**

- **imgs** (*torch.Tensor*) – The images to be processed by the model. Its size should be either (n, t, c, h, w) or (n, c, h, w).
- **model** (*nn.Module*) – The model to process the images.

**Returns** Output of the model with spatial ensemble applied.

**Return type** 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.

**Parameters**

- **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.

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

**Returns** Forward results.

**Return type** Tensor

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

Initialize network weights.

**Parameters**

- **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.

**Parameters**

- **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.

**Returns** Extracted patches. The dimension of the output should be the same as *img*.

**Return type** (torch.Tensor | numpy.array)

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

Extract patch from a given bbox.

**Parameters**

- **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).

**Returns** Extracted patches. The dimension of the output should be the same as *img*.

**Return type** (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.

**Parameters**

- **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.

**Returns** Warped image or feature map.

**Return type** 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.

**Parameters**

- **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.

**Parameters**

- **block** (*nn.module*) – nn.module class for basic block.
- **num_blocks** (*int*) – number of blocks.

**Returns** Stacked blocks in nn.Sequential.

**Return type** nn.Sequential

mmedit.models.common.**pixel_unshuffle**(*x*, *scale*)

Down-sample by pixel unshuffle.

**Parameters**

- **x** (*Tensor*) – Input tensor.
- **scale** (*int*) – Scale factor.

**Returns** Output tensor.

**Return type** 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.

**Parameters**

- **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.

**Returns** Modified bboxes.

**Return type** (np.ndarray | torch.Tensor)

mmedit.models.common.**set_requires_grad**(*nets*, *requires_grad=False*)

Set requires_grad for all the networks.

**Parameters**

- **nets** (*nn.Module | list[nn.Module]*) – A list of networks or a single network.
- **requires_grad** (*bool*) – Whether the networks require gradients or not

### 1.26.3 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

**Parameters**

- **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.

**Parameters** **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)’.

**Parameters** **lrs** (*tensor*) – Input LR images with shape (n, t, c, h, w)

**Returns**

**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).

**Return type** tuple(Tensor)

**forward**(*lrs*)

Forward function for BasicVSR.

**Parameters** **lrs** (*Tensor*) – Input LR sequence with shape (n, t, c, h, w).

**Returns** Output HR sequence with shape (n, t, c, 4h, 4w).

**Return type** Tensor

**init_weights**(*pretrained=None*, *strict=True*)
Init weights for models.

**Parameters**

- **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.

**Paper:** BasicVSR++: Improving Video Super-Resolution with Enhanced Propagation and Alignment

**Parameters**

- **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.

**Parameters** **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)’.

**Parameters** **lqs** (*tensor*) – Input low quality (LQ) sequence with shape (n, t, c, h, w).

**Returns**

**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).

**Return type** tuple(Tensor)

**forward**(*lqs*)
Forward function for BasicVSR++.

**Parameters** **lqs** (*tensor*) – Input low quality (LQ) sequence with shape (n, t, c, h, w).

**Returns** Output HR sequence with shape (n, t, c, 4h, 4w).

**Return type** Tensor

**init_weights**(*pretrained=None*, *strict=True*)
Init weights for models.

**Parameters**

- **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.

**Parameters**

- **dict** (*feats*) – Features from previous branches. Each component is a list of tensors with shape (n, c, h, w).
- **flows** (*tensor*) – Optical flows with shape (n, t - 1, 2, h, w).
- **module_name** (*str*) – The name of the propagation branches. Can either be 'backward_1', 'forward_1', 'backward_2', 'forward_2'.

**Returns**

**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.

**Return type** dict(list[tensor])

**upsample**(*lqs*, *feats*)
Compute the output image given the features.

**Parameters**

- **lqs** (*tensor*) – Input low quality (LQ) sequence with shape (n, t, c, h, w).
- **feats** (*dict*) – The features from the propagation branches.

**Returns** Output HR sequence with shape (n, t, c, 4h, 4w).

**Return type** Tensor

**class** mmedit.models.backbones.**CAINNet**(*in_channels=3*, *kernel_size=3*, *num_block_groups=5*, *num_block_layers=12*, *depth=3*, *reduction=16*, *norm=None*, *padding=7*, *act=LeakyReLU(negative_slope=0.2, inplace=True)*)
CAIN network structure.

Paper: Channel Attention Is All You Need for Video Frame Interpolation. Ref repo: https://github.com/myungsub/CAIN

**Parameters**

- **in_channels** (*int*) – Channel number of inputs. Default: 3.
- **kernel_size** (*int*) – Kernel size of CAINNet. Default: 3.
- **num_block_groups** (*int*) – Number of block groups. Default: 5.
- **num_block_layers** (*int*) – Number of blocks in a group. Default: 12.
- **depth** (*int*) – Down scale depth, scale = 2**depth. Default: 3.
- **reduction** (*int*) – Channel reduction of CA. Default: 16.
- **norm** (*str | None*) – Normalization layer. If it is None, no normalization is performed. Default: None.
- **padding** (*int*) – Padding of CAINNet. Default: 7.
- **act** (*function*) – activate function. Default: nn.LeakyReLU(0.2, True).

**forward**(*imgs*, *padding_flag=False*)
Forward function.

**Parameters**

- **imgs** (*Tensor*) – Input tensor with shape (n, 2, c, h, w).
- **padding_flag** (*bool*) – Padding or not. Default: False.

**Returns** Forward results.

**Return type** Tensor

**init_weights**(*pretrained=None*, *strict=True*)
Init weights for models.

**Parameters**

- **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.**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.

**Parameters**

- **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.

**Parameters**

- **x** (*torch.Tensor*) – Input tensor with shape of (n, c, h, w).
- **mask** (*torch.Tensor*) – Input tensor with shape of (n, 1, h, w).

**Returns** Output tensor with shape of (n, c, h', w').

**Return type** 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

**Parameters**

- **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.

**Parameters** **x** (*Tensor*) – Input tensor.

**Returns** Forward results. sr_outputs (list[Tensor]): forward sr results. heatmap_outputs (list[Tensor]): forward heatmap results.

**Return type** Tensor

**init_weights**(*pretrained=None*, *strict=True*)
Init weights for models.

**Parameters**

- **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

**Parameters**

- **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.

**Parameters** **input_dict** (*dict | torch.Tensor*) – Input dict with middle features or torch.Tensor.

**Returns** Output tensor with shape of (n, c, h, w).

**Return type** 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

**Parameters**

- **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.

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

**Returns** Output tensor with shape of (n, c, h', w').

**Return type** 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

**Parameters**

- **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.

**Parameters** **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.

**Returns** 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.

**Return type** tuple[torch.Tensor]

**init_weights**(*pretrained=None*)

Init weights for models.

**Parameters** **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.

**Parameters**

- **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.

**Parameters** **x** (*Tensor*) – Input feature map with shape (N, C, H, W).

**Returns** Encoder index feature and decoder index feature.

**Return type** 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

**Parameters**

- **in_channels** (*int*) – Channel number of inputs.
- **out_channels** (*int*) – Channel number of outputs.
- **mid_channels** (*int*) – Channel number of intermediate features. Default: 64.
- **num_blocks** (*int*) – Block number in the trunk network. Default: 16.
- **upscale_factor** (*int*) – Upsampling factor. Support 2^n and 3. Default: 4.
- **res_scale** (*float*) – Used to scale the residual in residual block. Default: 1.
- **rgb_mean** (*list[float]*) – Image mean in RGB orders. Default: [0.4488, 0.4371, 0.4040], calculated from DIV2K dataset.
- **rgb_std** (*list[float]*) – Image std in RGB orders. In EDSR, it uses [1.0, 1.0, 1.0]. Default: [1.0, 1.0, 1.0].

**forward**(*x*)
Forward function.

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

**Returns** Forward results.

**Return type** Tensor

**init_weights**(*pretrained=None*, *strict=True*)
Init weights for models.

**Parameters**

- **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.

**Parameters**

- **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.

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

**Returns** SR center frame with shape (n, c, h, w).

**Return type** Tensor

**init_weights**(*pretrained=None*, *strict=True*)

Init weights for models.

**Parameters**

- **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.

**Parameters**

- **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.

**Parameters** **inputs** (*dict*) – Output dict of FbaEncoder.

**Returns** Predicted alpha, fg and bg of the current batch.

**Return type** Tensor

**init_weights**(*pretrained=None*)
Init weights for the model.

**Parameters** **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.

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

**Returns** Output tensor.

**Return type** Tensor

**class** mmedit.models.backbones.**FLAVRNet**(*num_input_frames*, *num_output_frames*, *mid_channels_list=[512, 256, 128, 64]*, *encoder_layers_list=[2, 2, 2, 2]*, *bias=False*, *norm_cfg=None*, *join_type='concat'*, *up_mode='transpose'*)
PyTorch implementation of FLAVR for video frame interpolation.

**Paper:** FLAVR: Flow-Agnostic Video Representations for Fast Frame Interpolation

Ref repo: https://github.com/tarun005/FLAVR

**Parameters**

- **num_input_frames** (*int*) – Number of input frames.
- **num_output_frames** (*int*) – Number of output frames.
- **mid_channels_list** (*list[int]*) – List of number of mid channels. Default: [512, 256, 128, 64]
- **encoder_layers_list** (*list[int]*) – List of number of layers in encoder. Default: [2, 2, 2, 2]
- **bias** (*bool*) – If `True`, adds a learnable bias to the conv layers. Default: `True`
- **norm_cfg** (*dict | None*) – Config dict for normalization layer. Default: None
- **join_type** (*str*) – Join type of tensors from decoder and encoder. Candidates are `concat` and `add`. Default: `concat`
- **up_mode** (*str*) – Up-mode UpConv3d, candidates are `transpose` and `trilinear`. Default: `transpose`

**forward**(*images: torch.Tensor*)
Defines the computation performed at every call.

Should be overridden by all subclasses.

---

**Note:** 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.

**Parameters**

- **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.**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

**Parameters**

- **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.

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

**Returns** Output tensor with shape of (n, c, h', w').

**Return type** 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

**Parameters**

- **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.

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

**Returns** Output tensor with shape of (n, c, h', w').

**Return type** 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.

**Parameters**

- **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.

**Parameters** **lq** (*Tensor*) – Input LR image with shape (n, c, h, w).

**Returns** Output HR image.

**Return type** Tensor

**init_weights**(*pretrained=None*, *strict=True*)

Init weights for models.

**Parameters**

- **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

**Parameters**

- **norm_cfg** (*dict*) – Config dict to build norm layer.
- **act_cfg** (*dict*) – Config dict for activation layer, "relu" by default.

**forward**(*x*)

Forward Function.

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

**Returns** Output tensor with shape of (n, c, h', w').

**Return type** 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)

**Parameters**

- **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.

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

**Returns** Output tensor with shape of (n, c, h', w').

**Return type** torch.Tensor

**init_weights**(*pretrained=None*)

Init weights for models.

**Parameters** **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.

**Parameters**

- **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.

**Parameters** **x** (*Tensor*) – Input feature map with shape (N, C, H, W).

**Returns** Encoder index feature and decoder index feature.

**Return type** 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

**Parameters**

- **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.

**Parameters** **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)'.

**Parameters** **lrs** (*tensor*) – Input LR images with shape (n, t, c, h, w)

**Returns**

**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).

**Return type** 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)

**Returns**

**The keyframe features. Each key corresponds to the** indices in keyframe_idx.

**Return type** dict(Tensor)

**forward**(*lrs*)

Forward function for IconVSR.

**Parameters** **lrs** (*Tensor*) – Input LR tensor with shape (n, t, c, h, w).

**Returns** Output HR tensor with shape (n, t, c, 4h, 4w).

**Return type** Tensor

**init_weights**(*pretrained=None*, *strict=True*)

Init weights for models.

**Parameters**

- **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.

**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.

**Parameters** **lrs** (*Tensor*) – Input LR sequence with shape (n, t, c, h, w).

**Returns** Padded LR sequence with shape (n, t, c, h_pad, w_pad).

**Return type** Tensor

class mmedit.models.backbones.**IndexNetDecoder**(*in_channels*, *kernel_size=5*, *norm_cfg={'type': 'BN'}*, *separable_conv=False*)

**forward**(*inputs*)

Forward function.

**Parameters** **inputs** (*dict*) – Output dict of IndexNetEncoder.

**Returns** Predicted alpha matte of the current batch.

**Return type** 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.

**Parameters**

- **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.

**Raises**

- **ValueError** – out_stride must 16 or 32.
- **NameError** – Supported index_mode are {'holistic', 'o2o', 'm2o'}.

**forward**(*x*)

Forward function.

**Parameters** **x** (*Tensor*) – Input feature map with shape (N, C, H, W).

**Returns** Output tensor, shortcut feature and decoder index feature.

**Return type** dict

**freeze_bn**()

Set BatchNorm modules in the model to evaluation mode.

**init_weights**(*pretrained=None*)

Init weights for the model.

**Parameters** **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.

**Parameters**

- **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.

**Parameters**

- **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.

**Returns** Output tensor with shape (N, C, H', W').

**Return type** 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

**Parameters**

- **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.

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

**Returns** Forward results.

**Return type** 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

**Parameters**

- **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.

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

**Returns** Forward results.

**Return type** 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.

**Parameters**

- **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.

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

**Returns** Forward results.

**Return type** Tensor

**init_weights**(*pretrained=None*, *strict=True*)

Init weights for models.

**Parameters**

- **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

**Parameters**

- **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.

**Parameters** **input_dict** (*dict | torch.Tensor*) – Input dict with middle features or torch.Tensor.

**Returns** Output tensor with shape of (n, c, h, w).

**Return type** 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

**Parameters**

- **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.

**Parameters**

- **x** (*torch.Tensor*) – Masked image with shape (n, c, h, w).
- **mask** (*torch.Tensor*) – Mask tensor with shape (n, c, h, w).

**Returns** Contains the results and middle level features in this module. *hidden_feats* contain the middle feature maps and *hidden_masks* store updated masks.

**Return type** 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.

**Parameters** **mode** (*bool*) – whether to set training mode (`True`) or evaluation mode (`False`). Default: `True`.

**Returns** self

**Return type** Module

class mmedit.models.backbones.**PConvEncoderDecoder**(*encoder*, *decoder*)

Encoder-Decoder with partial conv module.

**Parameters**

- **encoder** (*dict*) – Config of the encoder.
- **decoder** (*dict*) – Config of the decoder.

**forward**(*x*, *mask_in*)

Forward Function.

**Parameters**

- **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).

**Returns** Output tensor with shape of (n, c, h', w').

**Return type** torch.Tensor

**init_weights**(*pretrained=None*)
Init weights for models.

**Parameters** **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.

**Parameters** **in_channels** (*int*) – Channel num of input features.

**forward**(*inputs*)
Forward function of PlainDecoder.

**Parameters** **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.

**Returns** Output tensor.

**Return type** 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.

Most of the implementation follows the implementation in: 'https://github.com/sanghyun-son/EDSR-PyTorch.git' 'EDSR-PyTorch/blob/master/src/model/rdn.py' Copyright (c) 2017, sanghyun-son, under MIT license.

**Parameters**

- **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.

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

**Returns** Forward results.

**Return type** Tensor

**init_weights**(*pretrained=None*, *strict=True*)
Init weights for models.

**Parameters**

- **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*, *upscale_factor=4*)
Networks consisting of Residual in Residual Dense Block, which is used in ESRGAN and Real-ESRGAN.

ESRGAN: Enhanced Super-Resolution Generative Adversarial Networks. Real-ESRGAN: Training Real-World Blind Super-Resolution with Pure Synthetic Data. # noqa: E501 Currently, it supports [x1/x2/x4] upsampling scale factor.

**Parameters**

- **in_channels** (*int*) – Channel number of inputs.
- **out_channels** (*int*) – Channel number of outputs.
- **mid_channels** (*int*) – Channel number of intermediate features. Default: 64
- **num_blocks** (*int*) – Block number in the trunk network. Defaults: 23
- **growth_channels** (*int*) – Channels for each growth. Default: 32.
- **upscale_factor** (*int*) – Upsampling factor. Support x1, x2 and x4. Default: 4.

**forward**(*x*)
Forward function.

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

**Returns** Forward results.

**Return type** Tensor

**init_weights**(*pretrained=None*, *strict=True*)
Init weights for models.

**Parameters**

- **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

**Parameters**

- **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.

---

**Note:** 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.

**Parameters**

- **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
                |
feat5 -------- layer2
             |
out ---  layer1
```

- gca module also requires unknown tensor generated by trimap which is ignored in the above graph.

**Parameters**

- **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.

**Parameters** **inputs** (*dict*) – Output dictionary of the ResGCAEncoder containing:

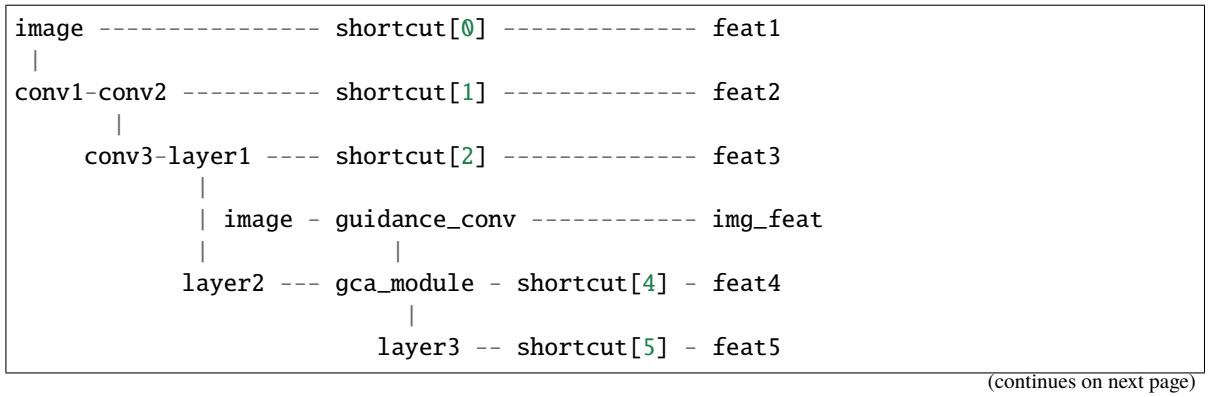
- out (Tensor): Output of the ResGCAEncoder.
- feat1 (Tensor): Shortcut connection from input image.
- feat2 (Tensor): Shortcut connection from conv2 of ResGCAEncoder.
- feat3 (Tensor): Shortcut connection from layer1 of ResGCAEncoder.
- feat4 (Tensor): Shortcut connection from layer2 of ResGCAEncoder.
- feat5 (Tensor): Shortcut connection from layer3 of ResGCAEncoder.
- img_feat (Tensor): Image feature extracted by guidance head.
- unknown (Tensor): Unknown tensor generated by trimap.

**Returns** Output tensor.

**Return type** 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.




(continues on next page)

(continued from previous page)

```
                                |
                        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.

**Parameters**

- **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.

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

**Returns** Contains the output tensor, shortcut feature and intermediate feature.

**Return type** 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.

**Parameters**

- **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.

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

**Returns** Output tensor.

**Return type** 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.

**Parameters**

- **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.

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

**Returns** Output tensor.

**Return type** 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
                   |
```

(continues on next page)

(continued from previous page)

```
feat5 ------- layer2
           |
out ---  layer1
```

**Parameters**

- **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.

**Parameters** **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.

**Returns** Output tensor.

**Return type** 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.

**Parameters**

- **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.

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

**Returns** Contains the output tensor and shortcut feature.

**Return type** 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.

**Parameters**

- **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.

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

**Returns** Forward results.

**Return type** Tensor

**init_weights**(*pretrained=None*, *strict=True*)
Initialize weights for the model.

**Parameters**

- **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.

**Parameters**

- **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.

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

**Returns** Forward results.

**Return type** Tensor

**init_weights**(*pretrained=None*, *strict=True*)
Init weights for models.

**Parameters**

- **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.

**Parameters**

- **encoder** (*dict*) – Config of the encoder.
- **decoder** (*dict*) – Config of the decoder.

**forward**(**args*, ***kwargs*)
Forward function.

**Returns** The output tensor of the decoder.

**Return type** 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

**Parameters**

- **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.

**Parameters** **lrs** (*Tensor*) – Input LR sequence with shape (n, t, c, h, w).

**Returns** Output HR image with shape (n, c, 4h, 4w) and aligned LR images with shape (n, t, c, h, w).

**Return type** tuple[Tensor]

**init_weights**(*pretrained=None*, *strict=True*)

Init weights for models.

**Parameters**

- **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

**Parameters** **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.

**Parameters** **img** (*Tensor*) – Output image.

**Returns** Denormalized image.

**Return type** Tensor

**forward**(*lrs*)

**Parameters** **lrs** – Input lr frames: (b, 7, 3, h, w).

**Returns** SR frame: (b, 3, h, w).

**Return type** Tensor

**init_weights**(*pretrained=None*, *strict=True*)
Init weights for models.

**Parameters**

- **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.

**Parameters** **img** (*Tensor*) – Input image.

**Returns** Normalized image.

**Return type** Tensor

**class** mmedit.models.backbones.**TOFlowVFINet**(*rgb_mean=[0.485, 0.456, 0.406]*, *rgb_std=[0.229, 0.224, 0.225]*, *flow_cfg={'norm_cfg': None, 'pretrained': None}*)
PyTorch implementation of TOFlow for video frame interpolation.

Paper: Xue et al., Video Enhancement with Task-Oriented Flow, IJCV 2018 Code reference:

1. https://github.com/anchen1011/toflow
2. https://github.com/Coldog2333/pytoflow

**Parameters**

- **rgb_mean** (*list[float]*) – Image mean in RGB orders. Default: [0.485, 0.456, 0.406]
- **rgb_std** (*list[float]*) – Image std in RGB orders. Default: [0.229, 0.224, 0.225]
- **flow_cfg** (*dict*) – Config of SPyNet. Default: dict(norm_cfg=None, pretrained=None)

**denormalize**(*img*)
Denormalize the output image.

**Parameters** **img** (*Tensor*) – Output image.

**Returns** Denormalized image.

**Return type** Tensor

**forward**(*inputs*)

**Parameters** **inputs** – Input frames with shape of (b, 2, 3, h, w).

**Returns** Interpolated frame with shape of (b, 3, h, w).

**Return type** Tensor

**init_weights**(*pretrained=None*, *strict=True*)
Init weights for models.

**Parameters**

- **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.

**Parameters** **img** (*Tensor*) – Input image.

**Returns** Normalized image.

**Return type** Tensor

**spatial_padding**(*inputs*)
Apply pdding spatially.

Since the SPyNet module in TOFlow requires that the resolution is a multiple of 16, we apply padding to the input LR images if their resolution is not divisible by 16.

**Parameters** **inputs** (*Tensor*) – Input sequence with shape (n, 2, c, h, w).

**Returns** Padded sequence with shape (n, 2, c, h_pad, w_pad).

**Return type** 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'.

**Parameters**

- **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.

**Parameters**

- **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), ...]

**Returns** Forward results.

**Return type** Tensor

**init_weights**(*pretrained=None*, *strict=True*)
Init weights for models.

**Parameters**

- **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.

**Parameters**

- **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.

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

**Returns** Forward results.

**Return type** Tensor

**init_weights**(*pretrained=None*, *strict=True*)

Initialize weights for the model.

**Parameters**

- **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.

**Parameters**

- **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.

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

**Returns** Dict containing output tensor and maxpooling indices.

**Return type** dict

### 1.26.4 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.

**Parameters**

- **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.

**Parameters**

- **x** (*torch.Tensor*) – Input tensor with shape of (n, c, h, w).
- **mask** (*torch.Tensor*) – Input tensor with shape of (n, 1, h, w).

**Returns** Output tensor with shape of (n, c, h', w').

**Return type** 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.

**Parameters**

- **global_disc_cfg** (*dict*) – Config dict for global discriminator.
- **local_disc_cfg** (*dict*) – Config dict for local discriminator.

**forward**(*x*)
Forward function.

**Parameters** **x** (*tuple[torch.Tensor]*) – Contains global image and the local image patch.

**Returns** Contains the prediction from discriminators in global image and local image patch.

**Return type** tuple[torch.Tensor]

**init_weights**(*pretrained=None*)
Init weights for models.

**Parameters** **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

**Parameters**

- **global_disc_cfg** (*dict*) – Config dict to build global discriminator.
- **local_disc_cfg** (*dict*) – Config dict to build local discriminator.

**forward**(*x*)
Forward function.

**Parameters** **x** (*tuple[torch.Tensor]*) – Contains global image and the local image patch.

**Returns** Contains the prediction from discriminators in global image and local image patch.

**Return type** tuple[torch.Tensor]

**init_weights**(*pretrained=None*)
Init weights for models.

**Parameters** **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.

**Parameters**

- **in_channels** (*int*) – Channel number of inputs. Default: 3.
- **mid_channels** (*int*) – Channel number of base intermediate features. Default: 64.

**forward**(*x*)
Forward function.

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

**Returns** Forward results.

**Return type** Tensor

**init_weights**(*pretrained=None*, *strict=True*)
Init weights for models.

**Parameters**

- **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.

**Parameters**

- **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.

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

**Returns** Output tensor with shape of (n, c, h', w') or (n, c).

**Return type** torch.Tensor

**init_weights**(*pretrained=None*)
Init weights for models.

**Parameters** **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.

**Parameters**

- **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.

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

**Returns** Forward results.

**Return type** Tensor

**init_weights**(*pretrained=None*)
Initialize weights for the model.

**Parameters** **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.

**Parameters**

- **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.

**Parameters**

- **x** (*Tensor*) – The input feature map of refiner.
- **raw_alpha** (*Tensor*) – The raw predicted alpha matte.

**Returns** The refined alpha matte.

**Return type** 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.

**Parameters**

- **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.

**Parameters** **x** (*torch.Tensor*) – Input image tensor.

**Returns** Predict score for the input image.

**Return type** 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.

**Parameters**

- **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*.

**Parameters**

- **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.

**Returns**

**Generated image tensor or dictionary** containing more data.

**Return type** 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.

**Parameters** **mode** (*bool*) – whether to set training mode (`True`) or evaluation mode (`False`). Default: `True`.

**Returns** self

**Return type** Module

`class mmedit.models.components.UNetDiscriminatorWithSpectralNorm`(*in_channels*, *mid_channels=64*, *skip_connection=True*)
A U-Net discriminator with spectral normalization.

**Parameters**

- **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.

**Parameters** **img** (*Tensor*) – Input tensor with shape (n, c, h, w).

**Returns** Forward results.

**Return type** Tensor

`init_weights`(*pretrained=None*, *strict=True*)
Init weights for models.

**Parameters**

- **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.

### 1.26.5 losses

`class mmedit.models.losses.CharbonnierCompLoss`(*loss_weight=1.0*, *reduction='mean'*, *sample_wise=False*, *eps=1e-12*)
Charbonnier composition loss.

**Parameters**

- **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*)

**Parameters**

- **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".

**Parameters**

- **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.

**Parameters**

- **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.

**Parameters** **loss_weight** (*float, optional*) – Loss weight. Defaults to 1.0.

**forward**(*x*)

Forward function.

**Parameters** **x** (*Tensor*) – Tensor with shape (n, c, h, w)

**Returns** Loss.

**Return type** 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.

**Parameters**

- **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*, *mask=None*)

**Parameters**

- **input** (*Tensor*) – The input for the loss module, i.e., the network prediction.
- **target_is_real** (*bool*) – Whether the target is real or fake.
- **is_disc** (*bool*) – Whether the loss for discriminators or not. Default: False.

**Returns** GAN loss value.

**Return type** Tensor

**get_target_label**(*input*, *target_is_real*)
Get target label.

**Parameters**

- **input** (*Tensor*) – Input tensor.
- **target_is_real** (*bool*) – Whether the target is real or fake.

**Returns**

**Target tensor. Return bool for wgan, otherwise,** return Tensor.

**Return type** (bool | Tensor)

class mmedit.models.losses.**GaussianBlur**(*kernel_size=(71, 71)*, *sigma=(10.0, 10.0)*)
A Gaussian filter which blurs a given tensor with a two-dimensional gaussian kernel by convolving it along each channel. Batch operation is supported.

This function is modified from kornia.filters.gaussian: *<https://kornia.readthedocs.io/en/latest/_modules/kornia/filters/gaussian.ht*

**Parameters**

- **kernel_size** (*tuple[int]*) – The size of the kernel. Default: (71, 71).
- **sigma** (*tuple[float]*) – The standard deviation of the kernel.
- **Default** (*10.0, 10.0*) –

**Returns** The Gaussian-blurred tensor.

**Return type** Tensor

**Shape:**

- input: Tensor with shape of (n, c, h, w)
- output: Tensor with shape of (n, c, h, w)

static compute_zero_padding(*kernel_size*)
Compute zero padding tuple.

forward(*x*)
Defines the computation performed at every call.

Should be overridden by all subclasses.

---

**Note:** 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.

---

get_1d_gaussian_kernel(*kernel_size*, *sigma*)
Get the Gaussian filter coefficients in one dimension (x or y direction).

**Parameters**

- **kernel_size** (*int*) – Kernel filter size in x or y direction. Should be odd and positive.
- **sigma** (*float*) – Gaussian standard deviation in x or y direction.

**Returns**

**A 1D torch tensor with gaussian filter** coefficients in x or y direction.

**Return type** kernel_1d (Tensor)

get_2d_gaussian_kernel(*kernel_size*, *sigma*)
Get the two-dimensional Gaussian filter matrix coefficients.

**Parameters**

- **kernel_size** (*tuple[int]*) – Kernel filter size in the x and y direction. The kernel sizes should be odd and positive.
- **sigma** (*tuple[int]*) – Gaussian standard deviation in the x and y direction.

**Returns**

**A 2D torch tensor with gaussian filter** matrix coefficients.

**Return type** kernel_2d (Tensor)

class mmedit.models.losses.**GradientLoss**(*loss_weight=1.0*, *reduction='mean'*)
Gradient loss.

**Parameters**

- **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*)

**Parameters**

- **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.

**Parameters** **loss_weight** (*float*) – Loss weight. Default: 1.0.

**forward**(*discriminator*, *real_data*, *fake_data*, *mask=None*)
Forward function.

**Parameters**

- **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.

**Returns** Loss.

**Return type** Tensor

class mmedit.models.losses.**L1CompositionLoss**(*loss_weight=1.0*, *reduction='mean'*, *sample_wise=False*)
L1 composition loss.

**Parameters**

- **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*)

**Parameters**

- **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.

**Parameters**

- **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.

**Parameters**

- **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.

**Parameters**

- **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.

**Parameters**

- **pred** (*Tensor*) – Predicted tensor.
- **gt** (*Tensor*) – GT tensor.

**Returns** Forward results.

**Return type** Tensor

class mmedit.models.losses.**MSECompositionLoss**(*loss_weight=1.0*, *reduction='mean'*, *sample_wise=False*)
MSE (L2) composition loss.

**Parameters**

- **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*)

**Parameters**

- **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.

**Parameters**

- **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.

**Parameters**

- **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.

**Parameters** **loss_weight** (*float, optional*) – Loss weight. Defaults to 1.0.

**forward**(*pred*, *mask=None*)
Forward function.

**Parameters**

- **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.

**Returns** [description]

**Return type** [type]

class mmedit.models.losses.**PerceptualLoss**(*layer_weights*, *layer_weights_style=None*, *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.

**Parameters**

- **layers_weights** (*dict*) – The weight for each layer of vgg feature for perceptual loss. Here is an example: {'4': 1., '9': 1., '18': 1.}, which means the 5th, 10th and 18th feature layer will be extracted with weight 1.0 in calculating losses.
- **layers_weights_style** (*dict*) – The weight for each layer of vgg feature for style loss. If set to 'None', the weights are set equal to the weights for perceptual loss. Default: None.
- **vgg_type** (*str*) – The type of vgg network used as feature extractor. Default: 'vgg19'.
- **use_input_norm** (*bool*) – If True, normalize the input image in vgg. Default: True.
- **perceptual_weight** (*float*) – If *perceptual_weight > 0*, the perceptual loss will be calculated and the loss will multiplied by the weight. Default: 1.0.

- **style_weight** (*float*) – If *style_weight > 0*, the style loss will be calculated and the loss will multiplied by the weight. Default: 1.0.
- **norm_img** (*bool*) – If True, the image will be normed to [0, 1]. Note that this is different from the *use_input_norm* which norm the input in in forward function of vgg according to the statistics of dataset. Importantly, the input image must be in range [-1, 1].
- **pretrained** (*str*) – Path for pretrained weights. Default: 'torchvision://vgg19'.
- **criterion** (*str*) – Criterion type. Options are 'l1' and 'mse'. Default: 'l1'.

**forward**(*x*, *gt*)
Forward function.

**Parameters**

- **x** (*Tensor*) – Input tensor with shape (n, c, h, w).
- **gt** (*Tensor*) – Ground-truth tensor with shape (n, c, h, w).

**Returns** Forward results.

**Return type** 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.

**Parameters**

- **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.

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

**Returns** Forward results.

**Return type** Tensor

**init_weights**(*model*, *pretrained*)
Init weights.

**Parameters**

- **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.

**Parameters**

- **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.

**Parameters**

- **maps** (*Tuple[Tensor]*) – Input tensors.
- **soft_attention** (*Tensor*) – Soft-attention tensor.
- **textures** (*Tuple[Tensor]*) – Ground-truth tensors.

**Returns** Forward results.

**Return type** Tensor

mmedit.models.losses.**mask_reduce_loss**(*loss*, *weight=None*, *reduction='mean'*, *sample_wise=False*)
Apply element-wise weight and reduce loss.

**Parameters**

- **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.

**Returns** Processed loss values.

**Return type** Tensor

mmedit.models.losses.**reduce_loss**(*loss*, *reduction*)
Reduce loss as specified.

**Parameters**

- **loss** (*Tensor*) – Elementwise loss tensor.
- **reduction** (*str*) – Options are "none", "mean" and "sum".

**Returns** Reduced loss tensor.

**Return type** Tensor

## 1.27 mmedit.utils

mmedit.utils.**deprecated_function**(*since: str*, *removed_in: str*, *instructions: str*) → Callable
Marks functions as deprecated.

Throw a warning when a deprecated function is called, and add a note in the docstring. Modified from https://github.com/pytorch/pytorch/blob/master/torch/onnx/_deprecation.py :param since: The version when the function was first deprecated. :type since: str :param removed_in: The version when the function will be removed. :type removed_in: str :param instructions: The action users should take. :type instructions: str

**Returns** A new function, which will be deprecated soon.

**Return type** Callable

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".

**Parameters**

- **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.

**Returns** The root logger.

**Return type** logging.Logger

mmedit.utils.**setup_multi_processes**(*cfg*)
Setup multi-processing environment variables.

# 1.28 Changelog

## 1.28.1 v0.16.0 (31/10/2022)

**Deprecations**

`VisualizationHook` is deprecated. Users should use `MMEditVisualizationHook` instead. (#1375)

```
visual_config = dict(  # config to register visualization hook
  type='VisualizationHook',
  output_dir='visual',
  interval=1000,
  res_name_list=[
      'gt_img', 'masked_img', 'fake_res', 'fake_img', 'fake_gt_local'
  ],
)
```

```
visual_config = dict(  # config to register visualization hook
  type='MMEditVisualizationHook',
  output_dir='visual',
  interval=1000,
  res_name_list=[
      'gt_img', 'masked_img', 'fake_res', 'fake_img', 'fake_gt_local'
  ],
)
```

**New Features & Improvements**

- Improve arguments type in `preprocess_div2k_dataset.py`. (#1381)
- Update docstring of RDN. (#1326)
- Update the introduction in readme. (#)

**Bug Fixes**

- Fix FLAVR register in `mmedit/models/video_interpolators` when importing `FLAVR`. (#1186)
- Fix data path processing in `restoration_video_inference.py`. (#1262)
- Fix the number of channels in RDB. (#1292, #1311)

**Contributors**

A total of 5 developers contributed to this release. Thanks @LeoXing1996, @Z-Fran, @zengyh1900, @ryanxingql, @ruoningYu.

### 1.28.2 v0.15.2 (09/09/2022)

**Improvements**

- [Docs] Fix typos in docs. by @Yulv-git in https://github.com/open-mmlab/mmediting/pull/1079
- [Docs] fix model_zoo and datasets docs link by @Z-Fran in https://github.com/open-mmlab/mmediting/pull/1043
- [Docs] fix typos in readme. by @arch-user-france1 in https://github.com/open-mmlab/mmediting/pull/1078
- [Improve] FLAVR demo by @Yshuo-Li in https://github.com/open-mmlab/mmediting/pull/954
- [Fix] Update MMCV_MAX to 1.7 by @wangruohui in https://github.com/open-mmlab/mmediting/pull/1001
- [Improve] Fix niqe_pris_params.npz path when installed as package by @ychfan in https://github.com/open-mmlab/mmediting/pull/995
- [CI] update github workflow, circleci and github templates by @zengyh1900 in https://github.com/open-mmlab/mmediting/pull/1087

**Contributors**

@wangruohui @Yshuo-Li @zengyh1900 @Z-Fran @ychfan @arch-user-france1 @Yulv-git

### 1.28.3 v0.15.1 (04/07/2022)

**Bug Fixes**

- [Fix] Update cain_b5_g1b32_vimeo90k_triplet.py (#929)
- [Docs] Fix link to OST dataset (#933)

**Improvements**

- [Docs] Update instruction to OST dataset (#937)
- [CI] No actual execution in CUDA envs (#921)
- [Docs] Add watermark to demo video (#935)
- [Tests] Add mim ci (#928)
- [Docs] Update README.md of FLAVR (#919)
- [Improve] Update md-format in .pre-commit-config.yaml (#917)
- [Improve] Add miminstall.txt in setup.py (#916)
- [Fix] Fix clutter in dim/README.md (#913)
- [Improve] Skip problematic opencv-python versions (#833)

**Contributors**

@wangruohui @Yshuo-Li

### 1.28.4 v0.15.0 (01/06/2022)

**Highlights**

1. Support FLAVR
2. Support AOT-GAN
3. Support CAIN with ReduceLROnPlateau Scheduler

**New Features**

- Add configs for AOT-GAN (#681)
- Support Vimeo90k-triplet dataset (#810)
- Add default config for mm-assistant (#827)
- Support CPU demo (#848)
- Support `use_cache` and `backend` in LoadImageFromFileList (#857)
- Support VFIVimeo90K7FramesDataset (#858)
- Support ColorJitter for VFI (#859)
- Support ReduceLrUpdaterHook (#860)
- Support `after_val_epoch` in IterBaseRunner (#861)
- Support FLAVR Net (#866, #867, #897)
- Support MAE metric (#871)
- Use mdformat (#888)
- Support CAIN with ReduceLROnPlateau Scheduler (#906)

**Bug Fixes**

- Change - to _ for restoration_demo.py (#834)
- Remove recommonmark in requirements/docs.txt (#844)
- Move EDVR to VSR category in README.md (#849)
- Remove , in multi-line F-string in crop.py (#855)
- Modify double `lq_path` to `gt_path` in test_pipeline (#862)
- Fix unittest of TOF-VFI (#873)
- Fix wrong frames in VFI demo (#891)
- Fix logo & contrib guideline on README (#898)
- Normalizing trimap in indexnet_dimaug_mobv2_1x16_78k_comp1k.py (#901)

**Improvements**

- Add `--cfg-options` in train/test scripts (#826)
- Update MMCV_MAX to 1.6 (#829)
- Update TOFlow in README (#835)

- Recover beirf installation steps & merge optional requirements (#836)
- Use {MMEditing Contributors} in citation (#838)
- Add tutorial for customizing losses (#839)
- Add installation guide (wiki ver) in README (#845)
- Add a 'need help to traslate' note on Chinese documentation (#850)
- Add wechat QR code in README_zh-CN.md (#851)
- Support non-zero frame index for SRFolderVideoDataset & Fix Typos (#853)
- Create README.md for docker (#856)
- Optimize IO for flow_warp (#881)
- Move wiki/installation to docs (#883)
- Add `myst_heading_anchors` (#887)
- Use checkpoint link in inpainting demo (#892)

**Contributors**

@wangruohui @quincylin1 @nijkah @jayagami @ckkelvinchan @ryanxingql @NK-CS-ZZL @Yshuo-Li

### 1.28.5 v0.14.0 (01/04/2022)

**Highlights**

1. Support TOFlow in video frame interpolation

**New Features**

- Support AOT-GAN (#677)
- Use `--diff-seed` to set different torch seed on different rank (#781)
- Support streaming reading of frames in video interpolation demo (#790)
- Support dist_train without slurm (#791)
- Put LQ into CPU for restoration_video_demo (#792)
- Support gray normalization constant in EDSR (#793)
- Support TOFlow in video frame interpolation (#806, #811)
- Support seed in DistributedSampler and sync seed across ranks (#815)

**Bug Fixes**

- Update link in README files (#782, #786, #819, #820)
- Fix matting tutorial, and fix links to colab (#795)
- Invert `flip_ratio` in `RandomAffine` pipeline (#799)
- Update preprocess_div2k_dataset.py (#801)
- Update SR Colab Demo Installation Method and Set5 link (#807)
- Fix Y/GRB mistake in EDSR README (#812)
- Replace pytorch install command to conda in README(_zh-CN).md (#816)

**Improvements**

- Update CI (#650)
- Update requirements.txt (#725, #817)
- Add Tutorial of dataset (#758), pipeline (#779), model (#766)
- Update index and TOC tree (#767)
- Make update_model_index.py compatible on windows (#768)
- Update doc build system (#769)
- Update keyword and classifier for setuptools (#773)
- Renovate installation (#776, #800)
- Update BasicVSR++ and RealBasicVSR docs (#778)
- Update citation (#785, #787)
- Regroup docs (#788)
- Use full name of config as 'Name' in metafile (#798)
- Update figure and video demo in README (#802)
- Add `clamp(0, 1)` in test of video frame interpolation (#805)
- Use hyphen for command line args in demo & tools (#808), and keep underline for required arguments in python files (#822)
- Make dataset.pipeline a dedicated section in doc (#813)
- Update mmcv-full>=1.3.13 to support DCN on CPU (#823)

**Contributors**

@wangruohui @ckkelvinchan @Yshuo-Li @nijkah @wdmwhh @freepoet @quincylin1

### 1.28.6 v0.13.0 (01/03/2022)

**Highlights**

1. Support CAIN
2. Support EDVR-L
3. Support running in Windows

**New Features**

- Add test-time ensemble for images and videos and support ensemble in BasicVSR series (#585)
- Support AOT-GAN (work in progress) (#674, #675, #676)
- Support CAIN (#683, #691, #709, #713)
- Add basic interpolater (#687)
- Add BaseVFIDataset and VFIVimeo90KDataset (#695, #697)
- Add video interpolation demo (#688, #717)
- Support various scales in RRDBNet (#699)
- Support Ref-SR inference (#716)
- Support EDVR-L on REDS (#719)

- Support CPU training (#720)
- Support running in Windows (#732, #738)
- Support DCN on CPU (#735)

**Bug Fixes**

- Fix link address in docs (#703, #704)
- Fix ARG `MMCV` in Dockerfile (#708)
- Fix file permission of non-executable files (#718)
- Fix some deprecation warning related to numpy (#728)
- Delete `__init__` in `TestVFIDataset` (#731)
- Fix data type in docstring of several Datasets (#739)
- Fix math notation in docstring (#741)
- Fix missing folders in copyright commit hook (#754)
- Delete duplicate test in loading (#756)

**Improvements**

- Update Pillow from 6.2.2 to 8.4 in CI (#693)
- Add argument 'repeat' to SRREDSMultipleGTDataset (#672)
- Deprecate the support for "python setup.py test" (#701)
- Add setup multi-processing both in train and test (#707)
- Add OpenMMLab website and platform links (#710)
- Refact README files of all methods (#712)
- Replace string version comparison with `package.version.parse` (#723)
- Add docs of Ref-SR demo and video frame interpolation demo (#724)
- Add interpolation and refact README.md (#726)
- Update isort version in pre-commit hook (#727)
- Redesign CI for Linux (#734)
- Update install.md (#763)
- Reorganizing OpenMMLab projects in readme (#764)
- Add deprecation message for deploy tools (#765)

**Contributors**

@wangruohui @ckkelvinchan @Yshuo-Li @quincylin1 @Juggernaut93 @anse3832 @nijkah

### 1.28.7 v0.12.0 (31/12/2021)

**Highlights**

1. Support RealBasicVSR
2. Support Real-ESRGAN checkpoint

**New Features**

- Support video input and output in restoration demo (#622)
- Support RealBasicVSR (#632, #633, #647, #680)
- Support Real-ESRGAN checkpoint (#635)
- Support conversion to y-channel when loading images (643)
- Support random video compression during training (#646)
- Support crop sequence (#648)
- Support pixel_unshuffle (#684)

**Bug Fixes**

- Change 'target_size' for RandomResize from list to tuple (#617)
- Fix folder creation in preprocess_df2k_ost_dataset.py (#623)
- Change TDAN config path in README (#625)
- Change 'radius' to 'kernel_size' for UnsharpMasking in Real-ESRNet config (#626)
- Fix bug in MATLABLikeResize (#630)
- Fix 'flow_warp' comment (#655)
- Fix the error of Model Zoo and Datasets in docs (#664)
- Fix bug in 'random_degradations' (#673)
- Limit opencv-python version (#689)

**Improvements**

- Translate docs to Chinese (#576, #577, #578, #579, #581, #582, #584, #587, #588, #589, #590, #591, #592, #593, #594, #595, #596, #641, #647, #656, #665, #666)
- Add UNetDiscriminatorWithSpectralNorm (#605)
- Use PyTorch sphinx theme (#607, #608)
- Update mmcv (#609), mmflow (#621), mmfewshot (#634) and mmhuman3d (#649) in docs
- Convert minimum GCC version to 5.4 (#612)
- Add tiff in SRDataset IMG_EXTENSIONS (#614)
- Update metafile and update_model_index.py (#615)
- Update preprocess_df2k_ost_dataset.py (#624)
- Add Abstract to README (#628, #636)
- Align NIQE to MATLAB results (#631)
- Add official markdown lint hook (#639)
- Skip CI when some specific files were changed (#640)

- Update docs/conf.py (#644, #651)
- Try to create a symbolic link on windows (#645)
- Cancel previous runs that are not completed (#650)
- Update path of configs in demo.md and getting_started.md (#658, #659)
- Use mmcv root model registry (#660)
- Update README.md (#654, #663)
- Refactor the structure of documentation (#668)
- Add script to crop REDS images into sub-images for faster IO (#669)
- Capitalize the first letter of the task name in the metafile (#678)
- Update FixedCrop for cropping image sequence (#682)

### 1.28.8 v0.11.0 (03/11/2021)

**Highlights**

- GLEAN for blind face image restoration #530
- Real-ESRGAN model #546

**New Features**

- Exponential Moving Average Hook #542
- Support DF2K_OST dataset #566

**Improvements**

- Add MATLAB-like bicubic interpolation #507
- Support random degradations during training #504
- Support torchserve #568

### 1.28.9 v0.10.0 (12/08/2021).

**Highlights**

1. Support LIIF-RDN (CVPR'2021)
2. Support BasicVSR++ (NTIRE'2021)

**New Features**

- Support loading annotation from file for video SR datasets (#423)
- Support persistent worker (#426)
- Support LIIF-RDN (#428, #440)
- Support BasicVSR++ (#451, #467)
- Support mim (#455)

**Bug Fixes**

- Fix bug in stat.py (#420)
- Fix astype error in function tensor2img (#429)

- Fix device error caused by torch.new_tensor when pytorch >= 1.7 (#465)
- Fix _non_dist_train in .mmedit/apis/train.py (#473)
- Fix multi-node distributed test (#478)

**Breaking Changes**

- Refactor LIIF for pytorch2onnx (#425)

**Improvements**

- Update Chinese docs (#415, #416, #418, #421, #424, #431, #442)
- Add CI of pytorch 1.9.0 (#444)
- Refactor README.md of configs (#452)
- Avoid loading pretrained VGG in unittest (#466)
- Support specifying scales in preprocessing div2k dataset (#472)
- Support all formats in readthedocs (#479)
- Use version_info of mmcv (#480)
- Remove unnecessary codes in restoration_video_demo.py (#484)
- Change priority of DistEvalIterHook to 'LOW' (#489)
- Reset resource limit (#491)
- Update QQ QR code in README_CN.md (#494)
- Add `myst_parser` (#495)
- Add license header (#496)
- Fix typo of StyleGAN modules (#427)
- Fix typo in docs/demo.md (#453, #454)
- Fix typo in tools/data/super-resolution/reds/README.md (#469)

## 1.28.10 v0.9.0 (30/06/2021).

**Highlights**

1. Support DIC and DIC-GAN (CVPR'2020)
2. Support GLEAN Cat 8x (CVPR'2021)
3. Support TTSR-GAN (CVPR'2020)
4. Add colab tutorial for super-resolution

**New Features**

- Add DIC (#342, #345, #348, #350, #351, #357, #363, #365, #366)
- Add SRFolderMultipleGTDataset (#355)
- Add GLEAN Cat 8x (#367)
- Add SRFolderVideoDataset (#370)
- Add colab tutorial for super-resolution (#380)
- Add TTSR-GAN (#372, #381, #383, #398)

- Add DIC-GAN (#392, #393, #394)

**Bug Fixes**

- Fix bug in restoration_video_inference.py (#379)
- Fix Config of LIIF (#368)
- Change the path to pre-trained EDVR-M (#396)
- Fix normalization in restoration_video_inference (#406)
- Fix [brush_stroke_mask] error in unittest (#409)

**Breaking Changes**

- Change mmcv minimum version to v1.3 (#378)

**Improvements**

- Correct Typos in code (#371)
- Add Custom_hooks (#362)
- Refactor unittest folder structure (#386)
- Add documents and download link for Vid4 (#399)
- Update model zoo for documents (#400)
- Update metafile (407)

### 1.28.11 v0.8.0 (31/05/2021).

**Highlights**

1. Support GLEAN (CVPR'2021)
2. Support TTSR (CVPR'2020)
3. Support TDAN (CVPR'2020)

**New Features**

- Add GLEAN (#296, #332)
- Support PWD metafile (#298)
- Support CropLike in pipeline (#299)
- Add TTSR (#301, #304, #307, #311, #311, #312, #313, #314, #321, #326, #327)
- Add TDAN (#316, #334, #347)
- Add onnx2tensorrt (#317)
- Add tensorrt evaluation (#328)
- Add SRFacicalLandmarkDataset (#329)
- Add key point auxiliary model for DIC (#336, #341)
- Add demo for video super-resolution methods (#275)
- Add SR Folder Ref Dataset (#292)
- Support FLOPs calculation of video SR models (#309)

**Bug Fixes**

- Fix find_unused_parameters in PyTorch 1.8 for BasicVSR (#290)
- Fix error in publish_model.py for pt>=1.6 (#291)
- Fix PSNR when input is uint8 (#294)

**Improvements**

- Support backend in LoadImageFromFile (#293, #303)
- Update `metric_average_mode` of video SR dataset (#319)
- Add error message in restoration_demo.py (324)
- Minor correction in getting_started.md (#339)
- Update description for Vimeo90K (#349)
- Support start_index in GenerateSegmentIndices (#338)
- Support different filename templates in GenerateSegmentIndices (#325)
- Support resize by scale-factor (#295, #310)

### 1.28.12 v0.7.0 (30/04/2021).

**Highlights**

1. Support BasicVSR (CVPR'2021)
2. Support IconVSR (CVPR'2021)
3. Support RDN (CVPR'2018)
4. Add onnx evaluation tool

**New Features**

- Add RDN (#233, #260, #280)
- Add MultipleGT datasets (#238)
- Add BasicVSR and IconVSR (#245, #252, #253, #254, #264, #274, #258, #252, #264)
- Add onnx evaluation tool (#279)

**Bug Fixes**

- Fix onnx conversion of maxunpool2d (#243)
- Fix inpainting in `demo.md` (#248)
- Tiny fix of config file of EDSR (#251)
- Fix link in README (#256)
- Fix restoration_inference key missing bug (#270)
- Fix the usage of channel_order in `loading.py` (#271)
- Fix the command of inpainting (#278)
- Fix `preprocess_vimeo90k_dataset.py` args name (#281)

**Improvements**

- Support `empty_cache` option in `test.py` (#261)
- Update projects in README (#249, #276)

- Support Y-channel PSNR and SSIM (#250)
- Add zh-CN README (#262)
- Update pytorch2onnx doc (#265)
- Remove extra quotation in English readme (#268)
- Change tags to comment (#269)
- List `model zoo` in README (#284, #285, #286)

### 1.28.13 v0.6.0 (08/04/2021).

**Highlights**

1. Support Local Implicit Image Function (LIIF)
2. Support exporting DIM and GCA from Pytorch to ONNX

**New Features**

- Add readthedocs config files and fix docstring (#92)
- Add github action file (#94)
- Support exporting DIM and GCA from Pytorch to ONNX (#105)
- Support concatenating datasets (#106)
- Support `non_dist_train` validation (#110)
- Add matting colab tutorial (#111)
- Support niqe metric (#114)
- Support PoolDataLoader for parrots (#134)
- Support collect-env (#137, #143)
- Support pt1.6 cpu/gpu in CI (#138)
- Support fp16 (139, #144)
- Support publishing to pypi (#149)
- Add modelzoo statistics (#171, #182, #186)
- Add doc of datasets (194)
- Support extended foreground option. (#195, #199, #200, #210)
- Support nn.MaxUnpool2d (#196)
- Add some FBA components (#203, #209, #215, #220)
- Support random down sampling in pipeline (#222)
- Support SR folder GT Dataset (#223)
- Support Local Implicit Image Function (LIIF) (#224, #226, #227, #234, #239)

**Bug Fixes**

- Fix `_non_dist_train` in train api (#104)
- Fix setup and CI (#109)
- Fix redundant loop bug in Normalize (#121)

- Fix `get_hash` in `setup.py` (#124)
- Fix `tool/preprocess_reds_dataset.py` (#148)
- Fix slurm train tutorial in `getting_started.md` (#162)
- Fix pip install bug (#173)
- Fix bug in config file (#185)
- Fix broken links of datasets (#236)
- Fix broken links of model zoo (#242)

**Breaking Changes**

- Refactor data loader configs (#201)

**Improvements**

- Updata requirements.txt (#95, #100)
- Update teaser (#96)
- Updata README (#93, #97, #98, #152)
- Updata model_zoo (#101)
- Fix typos (#102, #188, #191, #197, #208)
- Adopt adjust_gamma from skimage and reduce dependencies (#112)
- remove `.gitlab-ci.yml` (#113)
- Update import of first party (#115)
- Remove citation and contact (#122)
- Update version file (#136)
- Update download url (#141)
- Update `setup.py` (#150)
- Update the highest version of supported mmcv (#153, #154)
- modify `Crop` to handle a sequence of video frames (#164)
- Add links to other mm projects (#179, #180)
- Add config type (#181)
- Refactor docs (#184)
- Add config link (#187)
- Update file structure (#192)
- Update config doc (#202)
- Update `slurm_train.md` script (#204)
- Improve code style (#206, #207)
- Use `file_client` in CompositeFg (#212)
- Replace `random` with `numpy.random` (#213)
- Refactor `loader_cfg` (#214)

#### 1.28.14 v0.5.0 (09/07/2020).

Note that **MMSR** has been merged into this repo, as a part of MMEditing. With elaborate designs of the new framework and careful implementations, hope MMEditing could provide better experience.

## 1.29 English

## 1.30

CHAPTER

TWO

# INDICES AND TABLES

- genindex
- modindex
- search

# PYTHON MODULE INDEX

## m

# INDEX

## D



## E



## F

## G



## H



## I

## L

## M

## N



## O



## P



## Q



## R

## S



## T

## U



## V



## W