MMSegmentation

Release 0.28.0

MMSegmentation Authors

Sep 21, 2022

GET STARTED

CHAPTER

ONE

PREREQUISITES

In this section we demonstrate how to prepare an environment with PyTorch.

MMSegmentation works on Linux, Windows and macOS. It requires Python 3.6+, CUDA 9.2+ and PyTorch 1.3+.

Note: If you are experienced with PyTorch and have already installed it, just skip this part and jump to the *next section*. Otherwise, you can follow these steps for the preparation.

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

Step 1. Create a conda environment and activate it.

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

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

On GPU platforms:

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

On CPU platforms:

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

CHAPTER

TWO

INSTALLATION

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

2.1 Best Practices

Step 0. Install MMCV using MIM.

```
pip install -U openmim
mim install mmcv-full
```

Step 1. Install MMSegmentation.

Case a: If you develop and run mmseg directly, install it from source:

```
git clone https://github.com/open-mmlab/mmsegmentation.git
cd mmsegmentation
pip install -v -e .
# "-v" means verbose, or more output
# "-e" means installing a project in editable mode,
# thus any local modifications made to the code will take effect without reinstallation.
```

Case b: If you use mmsegmentation as a dependency or third-party package, install it with pip:

```
pip install mmsegmentation
```

2.2 Verify the installation

To verify whether MMSegmentation is installed correctly, we provide some sample codes to run an inference demo.

Step 1. We need to download config and checkpoint files.

```
mim download mmsegmentation --config pspnet_r50-d8_512x1024_40k_cityscapes --dest .
```

The downloading will take several seconds or more, depending on your network environment. When it is done, you will find two files `pspnet_r50-d8_512x1024_40k_cityscapes.py` and `pspnet_r50-d8_512x1024_40k_cityscapes_20200605_003338-2966598c.pth` in your current folder.

Step 2. Verify the inference demo.

Option (a). If you install mmsegmentation from source, just run the following command.

```
python demo/image_demo.py demo/demo.png configs/pspnet/pspnet_r50-d8_512x1024_40k_
↪cityscapes.py pspnet_r50-d8_512x1024_40k_cityscapes_20200605_003338-2966598c.pth --
↪device cuda:0 --out-file result.jpg
```

You will see a new image `result.jpg` on your current folder, where segmentation masks are covered on all objects.

Option (b). If you install mmsegmentation with pip, open you python interpreter and copy&paste the following codes.

```python
from mmseg.apis import inference_segmentor, init_segmentor
import mmcv

config_file = 'pspnet_r50-d8_512x1024_40k_cityscapes.py'
checkpoint_file = 'pspnet_r50-d8_512x1024_40k_cityscapes_20200605_003338-2966598c.pth'

# build the model from a config file and a checkpoint file
model = init_segmentor(config_file, checkpoint_file, device='cuda:0')

# test a single image and show the results
img = 'test.jpg'  # or img = mmcv.imread(img), which will only load it once
result = inference_segmentor(model, img)
# visualize the results in a new window
model.show_result(img, result, show=True)
# or save the visualization results to image files
# you can change the opacity of the painted segmentation map in (0, 1].
model.show_result(img, result, out_file='result.jpg', opacity=0.5)

# test a video and show the results
video = mmcv.VideoReader('video.mp4')
for frame in video:
   result = inference_segmentor(model, frame)
   model.show_result(frame, result, wait_time=1)
```

You can modify the code above to test a single image or a video, both of these options can verify that the installation was successful.

2.3 Customize Installation

2.3.1 CUDA versions

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

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

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

Note: Installing CUDA runtime libraries is enough if you follow our best practices, because no CUDA code will be compiled locally. However if you hope to compile MMCV from source or develop other CUDA operators, you need to

install the complete CUDA toolkit from NVIDIA's website, and its version should match the CUDA version of PyTorch. i.e., the specified version of cudatoolkit in `conda install` command.

2.3.2 Install MMCV without MIM

MMCV contains C++ and CUDA extensions, thus depending on PyTorch in a complex way. MIM solves such dependencies automatically and makes the 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/index.
↪html
```

2.3.3 Install on CPU-only platforms

MMSegmentation can be built for CPU only environment. In CPU mode you can train (requires MMCV version >= 1.4.4), test or inference a model.

2.3.4 Install on Google Colab

Google Colab usually has PyTorch installed, thus we only need to install MMCV and MMSegmentation with the following commands.

Step 1. Install MMCV using MIM.

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

Step 2. Install MMSegmentation from the source.

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

Step 3. Verification.

```
import mmseg
print(mmseg.__version__)
# Example output: 0.24.1
```

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

2.3.5 Using MMSegmentation with Docker

We provide a Dockerfile to build an image. Ensure that your docker version >=19.03.

```
# build an image with PyTorch 1.11, CUDA 11.3
# If you prefer other versions, just modified the Dockerfile
docker build -t mmsegmentation docker/
```

Run it with

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

2.4 Trouble shooting

If you have some issues during the installation, please first view the *FAQ* page. You may open an issue on GitHub if no solution is found.

CHAPTER

THREE

PREPARE DATASETS

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

```
mmsegmentation
├── mmseg
├── tools
├── configs
├── data
│   ├── cityscapes
│   │   ├── leftImg8bit
│   │   │   ├── train
│   │   │   ├── val
│   │   ├── gtFine
│   │   │   ├── train
│   │   │   ├── val
│   ├── VOCdevkit
│   │   ├── VOC2012
│   │   │   ├── JPEGImages
│   │   │   ├── SegmentationClass
│   │   │   ├── ImageSets
│   │   │   │   ├── Segmentation
│   │   ├── VOC2010
│   │   │   ├── JPEGImages
│   │   │   ├── SegmentationClassContext
│   │   │   ├── ImageSets
│   │   │   │   ├── SegmentationContext
│   │   │   │   │   ├── train.txt
│   │   │   │   │   ├── val.txt
│   │   │   ├── trainval_merged.json
│   │   ├── VOCaug
│   │   │   ├── dataset
│   │   │   │   ├── cls
│   ├── ade
│   │   ├── ADEChallengeData2016
│   │   │   ├── annotations
│   │   │   │   ├── training
│   │   │   │   ├── validation
│   │   │   ├── images
│   │   │   │   ├── training
│   │   │   │   ├── validation
```

(continues on next page)

(continued from previous page)

```
|   ├── coco_stuff10k
|   |   ├── images
|   |   |   ├── train2014
|   |   |   ├── test2014
|   |   ├── annotations
|   |   |   ├── train2014
|   |   |   ├── test2014
|   |   ├── imagesLists
|   |   |   ├── train.txt
|   |   |   ├── test.txt
|   |   |   ├── all.txt
|   ├── coco_stuff164k
|   |   ├── images
|   |   |   ├── train2017
|   |   |   ├── val2017
|   |   ├── annotations
|   |   |   ├── train2017
|   |   |   ├── val2017
|   ├── CHASE_DB1
|   |   ├── images
|   |   |   ├── training
|   |   |   ├── validation
|   |   ├── annotations
|   |   |   ├── training
|   |   |   ├── validation
|   ├── DRIVE
|   |   ├── images
|   |   |   ├── training
|   |   |   ├── validation
|   |   ├── annotations
|   |   |   ├── training
|   |   |   ├── validation
|   ├── HRF
|   |   ├── images
|   |   |   ├── training
|   |   |   ├── validation
|   |   ├── annotations
|   |   |   ├── training
|   |   |   ├── validation
|   ├── STARE
|   |   ├── images
|   |   |   ├── training
|   |   |   ├── validation
|   |   ├── annotations
|   |   |   ├── training
|   |   |   ├── validation
|   ├── dark_zurich
|   |   ├── gps
|   |   |   ├── val
|   |   |   └── val_ref
|   |   ├── gt
|   |   |   └── val
```

(continues on next page)

(continued from previous page)

```
|   |   ├── LICENSE.txt
|   |   ├── lists_file_names
|   |   |   ├── val_filenames.txt
|   |   |   └── val_ref_filenames.txt
|   |   ├── README.md
|   |   └── rgb_anon
|   |   |   ├── val
|   |   |   └── val_ref
|   ├── NighttimeDrivingTest
|   |   ├── gtCoarse_daytime_trainvaltest
|   |   |   └── test
|   |   |       └── night
|   |   └── leftImg8bit
|   |   |   └── test
|   |   |       └── night
|   ├── loveDA
|   |   ├── img_dir
|   |   |   ├── train
|   |   |   ├── val
|   |   |   ├── test
|   |   ├── ann_dir
|   |   |   ├── train
|   |   |   ├── val
|   ├── potsdam
|   |   ├── img_dir
|   |   |   ├── train
|   |   |   ├── val
|   |   ├── ann_dir
|   |   |   ├── train
|   |   |   ├── val
|   ├── vaihingen
|   |   ├── img_dir
|   |   |   ├── train
|   |   |   ├── val
|   |   ├── ann_dir
|   |   |   ├── train
|   |   |   ├── val
|   ├── iSAID
|   |   ├── img_dir
|   |   |   ├── train
|   |   |   ├── val
|   |   |   ├── test
|   |   ├── ann_dir
|   |   |   ├── train
|   |   |   ├── val
|   |   |   ├──
```

3.1 Cityscapes

The data could be found here after registration.

By convention, `**labelTrainIds.png` are used for cityscapes training. We provided a scripts based on cityscapess-cripts to generate `**labelTrainIds.png`.

```
# --nproc means 8 process for conversion, which could be omitted as well.
python tools/convert_datasets/cityscapes.py data/cityscapes --nproc 8
```

3.2 Pascal VOC

Pascal VOC 2012 could be downloaded from here. Beside, most recent works on Pascal VOC dataset usually exploit extra augmentation data, which could be found here.

If you would like to use augmented VOC dataset, please run following command to convert augmentation annotations into proper format.

```
# --nproc means 8 process for conversion, which could be omitted as well.
python tools/convert_datasets/voc_aug.py data/VOCdevkit data/VOCdevkit/VOCaug --nproc 8
```

Please refer to concat dataset for details about how to concatenate them and train them together.

3.3 ADE20K

The training and validation set of ADE20K could be download from this link. We may also download test set from here.

3.4 Pascal Context

The training and validation set of Pascal Context could be download from here. You may also download test set from here after registration.

To split the training and validation set from original dataset, you may download trainval_merged.json from here.

If you would like to use Pascal Context dataset, please install Detail and then run the following command to convert annotations into proper format.

```
python tools/convert_datasets/pascal_context.py data/VOCdevkit data/VOCdevkit/VOC2010/
↪trainval_merged.json
```

3.5 COCO Stuff 10k

The data could be downloaded here by wget.

For COCO Stuff 10k dataset, please run the following commands to download and convert the dataset.

```
# download
mkdir coco_stuff10k && cd coco_stuff10k
wget http://calvin.inf.ed.ac.uk/wp-content/uploads/data/cocostuffdataset/cocostuff-10k-
↪v1.1.zip

# unzip
unzip cocostuff-10k-v1.1.zip

# --nproc means 8 process for conversion, which could be omitted as well.
python tools/convert_datasets/coco_stuff10k.py /path/to/coco_stuff10k --nproc 8
```

By convention, mask labels in `/path/to/coco_stuff164k/annotations/*2014/*_labelTrainIds.png` are used for COCO Stuff 10k training and testing.

3.6 COCO Stuff 164k

For COCO Stuff 164k dataset, please run the following commands to download and convert the augmented dataset.

```
# download
mkdir coco_stuff164k && cd coco_stuff164k
wget http://images.cocodataset.org/zips/train2017.zip
wget http://images.cocodataset.org/zips/val2017.zip
wget http://calvin.inf.ed.ac.uk/wp-content/uploads/data/cocostuffdataset/stuffthingmaps_
↪trainval2017.zip

# unzip
unzip train2017.zip -d images/
unzip val2017.zip -d images/
unzip stuffthingmaps_trainval2017.zip -d annotations/

# --nproc means 8 process for conversion, which could be omitted as well.
python tools/convert_datasets/coco_stuff164k.py /path/to/coco_stuff164k --nproc 8
```

By convention, mask labels in `/path/to/coco_stuff164k/annotations/*2017/*_labelTrainIds.png` are used for COCO Stuff 164k training and testing.

The details of this dataset could be found at here.

3.7 CHASE DB1

The training and validation set of CHASE DB1 could be download from here.

To convert CHASE DB1 dataset to MMSegmentation format, you should run the following command:

```
python tools/convert_datasets/chase_db1.py /path/to/CHASEDB1.zip
```

The script will make directory structure automatically.

3.8 DRIVE

The training and validation set of DRIVE could be download from here. Before that, you should register an account. Currently '1st_manual' is not provided officially.

To convert DRIVE dataset to MMSegmentation format, you should run the following command:

```
python tools/convert_datasets/drive.py /path/to/training.zip /path/to/test.zip
```

The script will make directory structure automatically.

3.9 HRF

First, download healthy.zip, glaucoma.zip, diabetic_retinopathy.zip, healthy_manualsegm.zip, glaucoma_manualsegm.zip and diabetic_retinopathy_manualsegm.zip.

To convert HRF dataset to MMSegmentation format, you should run the following command:

```
python tools/convert_datasets/hrf.py /path/to/healthy.zip /path/to/healthy_manualsegm.
↪zip /path/to/glaucoma.zip /path/to/glaucoma_manualsegm.zip /path/to/diabetic_
↪retinopathy.zip /path/to/diabetic_retinopathy_manualsegm.zip
```

The script will make directory structure automatically.

3.10 STARE

First, download stare-images.tar, labels-ah.tar and labels-vk.tar.

To convert STARE dataset to MMSegmentation format, you should run the following command:

```
python tools/convert_datasets/stare.py /path/to/stare-images.tar /path/to/labels-ah.tar /
↪path/to/labels-vk.tar
```

The script will make directory structure automatically.

3.11 Dark Zurich

Since we only support test models on this dataset, you may only download the validation set.

3.12 Nighttime Driving

Since we only support test models on this dataset, you may only download the test set.

3.13 LoveDA

The data could be downloaded from Google Drive here.

Or it can be downloaded from zenodo, you should run the following command:

```
# Download Train.zip
wget https://zenodo.org/record/5706578/files/Train.zip
# Download Val.zip
wget https://zenodo.org/record/5706578/files/Val.zip
# Download Test.zip
wget https://zenodo.org/record/5706578/files/Test.zip
```

For LoveDA dataset, please run the following command to download and re-organize the dataset.

```
python tools/convert_datasets/loveda.py /path/to/loveDA
```

Using trained model to predict test set of LoveDA and submit it to server can be found here.

More details about LoveDA can be found here.

3.14 ISPRS Potsdam

The Potsdam dataset is for urban semantic segmentation used in the 2D Semantic Labeling Contest - Potsdam.

The dataset can be requested at the challenge homepage. The '2_Ortho_RGB.zip' and '5_Labels_all_noBoundary.zip' are required.

For Potsdam dataset, please run the following command to download and re-organize the dataset.

```
python tools/convert_datasets/potsdam.py /path/to/potsdam
```

In our default setting, it will generate 3456 images for training and 2016 images for validation.

3.15 ISPRS Vaihingen

The Vaihingen dataset is for urban semantic segmentation used in the 2D Semantic Labeling Contest - Vaihingen.

The dataset can be requested at the challenge homepage. The 'ISPRS_semantic_labeling_Vaihingen.zip' and 'ISPRS_semantic_labeling_Vaihingen_ground_truth_eroded_COMPLETE.zip' are required.

For Vaihingen dataset, please run the following command to download and re-organize the dataset.

```
python tools/convert_datasets/vaihingen.py /path/to/vaihingen
```

In our default setting (`clip_size` =512, `stride_size`=256), it will generate 344 images for training and 398 images for validation.

3.16 iSAID

The data images could be download from DOTA-v1.0 (train/val/test)

The data annotations could be download from iSAID (train/val)

The dataset is a Large-scale Dataset for Instance Segmentation (also have segmantic segmentation) in Aerial Images.

You may need to follow the following structure for dataset preparation after downloading iSAID dataset.

```
│   ├── iSAID
│   │   ├── train
│   │   │   ├── images
│   │   │   │   ├── part1.zip
│   │   │   │   ├── part2.zip
│   │   │   │   ├── part3.zip
│   │   │   ├── Semantic_masks
│   │   │   │   ├── images.zip
│   │   ├── val
│   │   │   ├── images
│   │   │   │   ├── part1.zip
│   │   │   ├── Semantic_masks
│   │   │   │   ├── images.zip
│   │   ├── test
│   │   │   ├── images
│   │   │   │   ├── part1.zip
│   │   │   │   ├── part2.zip
```

```
python tools/convert_datasets/isaid.py /path/to/iSAID
```

In our default setting (`patch_width`=896, `patch_height`=896,`overlap_area`=384), it will generate 33978 images for training and 11644 images for validation.

CHAPTER

FOUR

BENCHMARK AND MODEL ZOO

4.1 Common settings

- We use distributed training with 4 GPUs by default.
- All pytorch-style pretrained backbones on ImageNet are train by ourselves, with the same procedure in the paper. Our ResNet style backbone are based on ResNetV1c variant, where the 7x7 conv in the input stem is replaced with three 3x3 convs.
- For the consistency across different hardwares, we report the GPU memory as the maximum value of `torch.cuda.max_memory_allocated()` for all 4 GPUs with `torch.backends.cudnn.benchmark=False`. Note that this value is usually less than what `nvidia-smi` shows.
- We report the inference time as the total time of network forwarding and post-processing, excluding the data loading time. Results are obtained with the script `tools/benchmark.py` which computes the average time on 200 images with `torch.backends.cudnn.benchmark=False`.
- There are two inference modes in this framework.
  - `slide` mode: The `test_cfg` will be like `dict(mode='slide', crop_size=(769, 769), stride=(513, 513))`.

    In this mode, multiple patches will be cropped from input image, passed into network individually. The crop size and stride between patches are specified by `crop_size` and `stride`. The overlapping area will be merged by average
  - `whole` mode: The `test_cfg` will be like `dict(mode='whole')`.

    In this mode, the whole imaged will be passed into network directly.

    By default, we use `slide` inference for 769x769 trained model, `whole` inference for the rest.
- For input size of 8x+1 (e.g. 769), `align_corner=True` is adopted as a traditional practice. Otherwise, for input size of 8x (e.g. 512, 1024), `align_corner=False` is adopted.

4.2 Baselines

4.2.1 FCN

Please refer to FCN for details.

4.2.2 PSPNet

Please refer to PSPNet for details.

4.2.3 DeepLabV3

Please refer to DeepLabV3 for details.

4.2.4 PSANet

Please refer to PSANet for details.

4.2.5 DeepLabV3+

Please refer to DeepLabV3+ for details.

4.2.6 UPerNet

Please refer to UPerNet for details.

4.2.7 NonLocal Net

Please refer to NonLocal Net for details.

4.2.8 EncNet

Please refer to EncNet for details.

4.2.9 CCNet

Please refer to CCNet for details.

4.2.10 DANet

Please refer to DANet for details.

4.2.11 APCNet

Please refer to APCNet for details.

4.2.12 HRNet

Please refer to HRNet for details.

4.2.13 GCNet

Please refer to GCNet for details.

4.2.14 DMNet

Please refer to DMNet for details.

4.2.15 ANN

Please refer to ANN for details.

4.2.16 OCRNet

Please refer to OCRNet for details.

4.2.17 Fast-SCNN

Please refer to Fast-SCNN for details.

4.2.18 ResNeSt

Please refer to ResNeSt for details.

4.2.19 Semantic FPN

Please refer to Semantic FPN for details.

4.2.20 PointRend

Please refer to PointRend for details.

4.2.21 MobileNetV2

Please refer to MobileNetV2 for details.

4.2.22 MobileNetV3

Please refer to MobileNetV3 for details.

4.2.23 EMANet

Please refer to EMANet for details.

4.2.24 DNLNet

Please refer to DNLNet for details.

4.2.25 CGNet

Please refer to CGNet for details.

4.2.26 Mixed Precision (FP16) Training

Please refer Mixed Precision (FP16) Training on BiSeNetV2 for details.

4.2.27 U-Net

Please refer to U-Net for details.

4.2.28 ViT

Please refer to ViT for details.

4.2.29 Swin

Please refer to Swin for details.

4.2.30 SETR

Please refer to SETR for details.

4.3 Speed benchmark

4.3.1 Hardware

- 8 NVIDIA Tesla V100 (32G) GPUs
- Intel(R) Xeon(R) Gold 6148 CPU @ 2.40GHz

4.3.2 Software environment

- Python 3.7
- PyTorch 1.5
- CUDA 10.1
- CUDNN 7.6.03
- NCCL 2.4.08

4.3.3 Training speed

For fair comparison, we benchmark all implementations with ResNet-101V1c. The input size is fixed to 1024x512 with batch size 2.

The training speed is reported as followed, in terms of second per iter (s/iter). The lower, the better.

Note: The output stride of DeepLabV3+ is 8.

CHAPTER

FIVE

MODEL ZOO STATISTICS

- Number of papers: 44
  - ALGORITHM: 34
  - BACKBONE: 10
- Number of checkpoints: 598
  - [ALGORITHM] ANN (16 ckpts)
  - [ALGORITHM] APCNet (12 ckpts)
  - [BACKBONE] BEiT (2 ckpts)
  - [ALGORITHM] BiSeNetV1 (11 ckpts)
  - [ALGORITHM] BiSeNetV2 (4 ckpts)
  - [ALGORITHM] CCNet (16 ckpts)
  - [ALGORITHM] CGNet (2 ckpts)
  - [BACKBONE] ConvNeXt (6 ckpts)
  - [ALGORITHM] DANet (16 ckpts)
  - [ALGORITHM] DeepLabV3 (41 ckpts)
  - [ALGORITHM] DeepLabV3+ (42 ckpts)
  - [ALGORITHM] DMNet (12 ckpts)
  - [ALGORITHM] DNLNet (12 ckpts)
  - [ALGORITHM] DPT (1 ckpts)
  - [ALGORITHM] EMANet (4 ckpts)
  - [ALGORITHM] EncNet (12 ckpts)
  - [ALGORITHM] ERFNet (1 ckpts)
  - [ALGORITHM] FastFCN (12 ckpts)
  - [ALGORITHM] Fast-SCNN (1 ckpts)
  - [ALGORITHM] FCN (41 ckpts)
  - [ALGORITHM] GCNet (16 ckpts)
  - [BACKBONE] HRNet (37 ckpts)
  - [ALGORITHM] ICNet (12 ckpts)

- [ALGORITHM] ISANet (16 ckpts)
- [ALGORITHM] K-Net (7 ckpts)
- [BACKBONE] MAE (1 ckpts)
- [BACKBONE] MobileNetV2 (8 ckpts)
- [BACKBONE] MobileNetV3 (4 ckpts)
- [ALGORITHM] NonLocal Net (16 ckpts)
- [ALGORITHM] OCRNet (24 ckpts)
- [ALGORITHM] PointRend (4 ckpts)
- [ALGORITHM] PSANet (16 ckpts)
- [ALGORITHM] PSPNet (54 ckpts)
- [BACKBONE] ResNeSt (8 ckpts)
- [ALGORITHM] SegFormer (13 ckpts)
- [ALGORITHM] Segmenter (5 ckpts)
- [ALGORITHM] Semantic FPN (4 ckpts)
- [ALGORITHM] SETR (7 ckpts)
- [ALGORITHM] STDC (4 ckpts)
- [BACKBONE] Swin Transformer (8 ckpts)
- [BACKBONE] Twins (12 ckpts)
- [ALGORITHM] UNet (25 ckpts)
- [ALGORITHM] UPerNet (22 ckpts)
- [BACKBONE] Vision Transformer (11 ckpts)

CHAPTER

SIX

TRAIN A MODEL

MMSegmentation implements distributed training and non-distributed training, which uses `MMDistributedDataParallel` and `MMDataParallel` respectively.

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 some iterations, you can change the evaluation interval by adding the interval argument in the training config.

```
evaluation = dict(interval=4000)  # This evaluate the model per 4000 iterations.
```

Important: The default learning rate in config files is for 4 GPUs and 2 img/gpu (batch size = 4x2 = 8). Equivalently, you may also use 8 GPUs and 1 imgs/gpu since all models using cross-GPU SyncBN.

To trade speed with GPU memory, you may pass in `--cfg-options model.backbone.with_cp=True` to enable checkpoint in backbone.

6.1 Train on a single machine

6.1.1 Train with a single GPU

official support:

```
sh tools/dist_train.sh ${CONFIG_FILE} 1 [optional arguments]
```

experimental support (Convert SyncBN to BN):

```
python tools/train.py ${CONFIG_FILE} [optional arguments]
```

If you want to specify the working directory in the command, you can add an argument `--work-dir ${YOUR_WORK_DIR}`.

6.1.2 Train with CPU

The process of training on the CPU is consistent with single GPU training if machine does not have GPU. If it has GPUs but not wanting to use it, we just need to disable GPUs before the training process.

```
export CUDA_VISIBLE_DEVICES=-1
```

And then run the script above.

> **Warning:** The process of training on the CPU is consistent with single GPU training. We just need to disable GPUs before the training process.

6.1.3 Train with multiple GPUs

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

Optional arguments are:

- `--no-validate` (**not suggested**): By default, the codebase will perform evaluation at 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 (to continue the training process).
- `--load-from ${CHECKPOINT_FILE}`: Load weights from a checkpoint file (to start finetuning for another task).
- `--deterministic`: Switch on "deterministic" mode which slows down training but the results are reproducible.

Difference between `resume-from` and `load-from`:

- `resume-from` loads both the model weights and optimizer state including the iteration number.
- `load-from` loads only the model weights, starts the training from iteration 0.

An example:

```
# checkpoints and logs saved in WORK_DIR=work_dirs/pspnet_r50-d8_512x512_80k_ade20k/
# If work_dir is not set, it will be generated automatically.
sh tools/dist_train.sh configs/pspnet/pspnet_r50-d8_512x512_80k_ade20k.py 8 --work_dir␣
↪work_dirs/pspnet_r50-d8_512x512_80k_ade20k/ --deterministic
```

Note: During training, checkpoints and logs are saved in the same folder structure as the config file under `work_dirs/`. Custom work directory is not recommended since evaluation scripts infer work directories from the config file name. If you want to save your weights somewhere else, please use symlink, for example:

```
ln -s ${YOUR_WORK_DIRS} ${MMSEG}/work_dirs
```

6.1.4 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. Otherwise, there will be error message saying `RuntimeError: Address already in use`.

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

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

6.2 Train with multiple machines

If you launch with multiple machines simply connected with ethernet, you can simply run following commands:

On the first machine:

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

On the second machine:

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

Usually it is slow if you do not have high speed networking like InfiniBand.

6.3 Manage jobs with Slurm

Slurm is a good job scheduling system for computing clusters. On a cluster managed by Slurm, you can use slurm_train.sh to spawn training jobs. It supports both single-node and multi-node training.

Train with multiple machines:

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

Here is an example of using 16 GPUs to train PSPNet on the dev partition.

```
GPUS=16 sh tools/slurm_train.sh dev pspr50 configs/pspnet/pspnet_r50-d8_512x1024_40k_
↪cityscapes.py work_dirs/pspnet_r50-d8_512x1024_40k_cityscapes/
```

When using 'slurm_train.sh' to start multiple tasks on a node, different ports need to be specified. Three settings are provided:

Option 1:

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 and config2.py.

```
CUDA_VISIBLE_DEVICES=0,1,2,3 GPUS=4 sh tools/slurm_train.sh ${PARTITION} ${JOB_NAME}␣
↪config1.py tmp_work_dir_1
CUDA_VISIBLE_DEVICES=4,5,6,7 GPUS=4 sh tools/slurm_train.sh ${PARTITION} ${JOB_NAME}␣
↪config2.py tmp_work_dir_2
```

Option 2:

You can set different communication ports without the need to modify the configuration file, but have to set the `cfg-options` to overwrite the default port in configuration file.

```
CUDA_VISIBLE_DEVICES=0,1,2,3 GPUS=4 sh tools/slurm_train.sh ${PARTITION} ${JOB_NAME}␣
↪config1.py tmp_work_dir_1 --cfg-options dist_params.port=29500
CUDA_VISIBLE_DEVICES=4,5,6,7 GPUS=4 sh tools/slurm_train.sh ${PARTITION} ${JOB_NAME}␣
↪config2.py tmp_work_dir_2 --cfg-options dist_params.port=29501
```

Option 3:

You can set the port in the command using the environment variable 'MASTER_PORT':

```
CUDA_VISIBLE_DEVICES=0,1,2,3 GPUS=4 MASTER_PORT=29500 sh tools/slurm_train.sh $
↪{PARTITION} ${JOB_NAME} config1.py tmp_work_dir_1
CUDA_VISIBLE_DEVICES=4,5,6,7 GPUS=4 MASTER_PORT=29501 sh tools/slurm_train.sh $
↪{PARTITION} ${JOB_NAME} config2.py tmp_work_dir_2
```

CHAPTER

SEVEN

INFERENCE WITH PRETRAINED MODELS

We provide testing scripts to evaluate a whole dataset (Cityscapes, PASCAL VOC, ADE20k, etc.), and also some high-level apis for easier integration to other projects.

7.1 Test a dataset

- single GPU
- CPU
- single node multiple GPU
- multiple node

You can use the following commands to test a dataset.

```
# single-gpu testing
python tools/test.py ${CONFIG_FILE} ${CHECKPOINT_FILE} [--out ${RESULT_FILE}] [--eval $
↪{EVAL_METRICS}] [--show]

# CPU: If GPU unavailable, directly running single-gpu testing command above
# CPU: If GPU available, disable GPUs and run single-gpu testing script
export CUDA_VISIBLE_DEVICES=-1
python tools/test.py ${CONFIG_FILE} ${CHECKPOINT_FILE} [--out ${RESULT_FILE}] [--eval $
↪{EVAL_METRICS}] [--show]

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

Optional arguments:

- `RESULT_FILE`: Filename of the output results in pickle format. If not specified, the results will not be saved to a file. (After mmseg v0.17, the output results become pre-evaluation results or format result paths)
- `EVAL_METRICS`: Items to be evaluated on the results. Allowed values depend on the dataset, e.g., `mIoU` is available for all dataset. Cityscapes could be evaluated by `cityscapes` as well as standard `mIoU` metrics.
- `--show`: If specified, segmentation results will be plotted on the images and shown in a new window. It is only applicable to single GPU testing and used for debugging and visualization. Please make sure that GUI is available in your environment, otherwise you may encounter the error like `cannot connect to X server`.
- `--show-dir`: If specified, segmentation results will be plotted on the images and saved to the specified directory. It is only applicable to single GPU testing and used for debugging and visualization. You do NOT need a GUI available in your environment for using this option.

- `--eval-options`: Optional parameters for `dataset.format_results` and `dataset.evaluate` during evaluation. When `efficient_test=True`, it will save intermediate results to local files to save CPU memory. Make sure that you have enough local storage space (more than 20GB). (`efficient_test` argument does not have effect after mmseg v0.17, we use a progressive mode to evaluation and format results which can largely save memory cost and evaluation time.)

Examples:

Assume that you have already downloaded the checkpoints to the directory `checkpoints/`.

1. Test PSPNet and visualize the results. Press any key for the next image.

```
python tools/test.py configs/pspnet/pspnet_r50-d8_512x1024_40k_cityscapes.py \
    checkpoints/pspnet_r50-d8_512x1024_40k_cityscapes_20200605_003338-2966598c.pth \
    --show
```

2. Test PSPNet and save the painted images for latter visualization.

```
python tools/test.py configs/pspnet/pspnet_r50-d8_512x1024_40k_cityscapes.py \
    checkpoints/pspnet_r50-d8_512x1024_40k_cityscapes_20200605_003338-2966598c.pth \
    --show-dir psp_r50_512x1024_40ki_cityscapes_results
```

3. Test PSPNet on PASCAL VOC (without saving the test results) and evaluate the mIoU.

```
python tools/test.py configs/pspnet/pspnet_r50-d8_512x1024_20k_voc12aug.py \
    checkpoints/pspnet_r50-d8_512x1024_20k_voc12aug_20200605_003338-c57ef100.pth \
    --eval mAP
```

4. Test PSPNet with 4 GPUs, and evaluate the standard mIoU and cityscapes metric.

```
./tools/dist_test.sh configs/pspnet/pspnet_r50-d8_512x1024_40k_cityscapes.py \
    checkpoints/pspnet_r50-d8_512x1024_40k_cityscapes_20200605_003338-2966598c.pth \
    4 --out results.pkl --eval mIoU cityscapes
```

Note: There is some gap (~0.1%) between cityscapes mIoU and our mIoU. The reason is that cityscapes average each class with class size by default. We use the simple version without average for all datasets.

---

5. Test PSPNet on cityscapes test split with 4 GPUs, and generate the png files to be submit to the official evaluation server.

   First, add following to config file `configs/pspnet/pspnet_r50-d8_512x1024_40k_cityscapes.py`,

```
data = dict(
    test=dict(
        img_dir='leftImg8bit/test',
        ann_dir='gtFine/test'))
```

   Then run test.

```
./tools/dist_test.sh configs/pspnet/pspnet_r50-d8_512x1024_40k_cityscapes.py \
    checkpoints/pspnet_r50-d8_512x1024_40k_cityscapes_20200605_003338-2966598c.pth \
    4 --format-only --eval-options "imgfile_prefix=./pspnet_test_results"
```

   You will get png files under `./pspnet_test_results` directory. You may run `zip -r results.zip pspnet_test_results/` and submit the zip file to evaluation server.

6. CPU memory efficient test DeeplabV3+ on Cityscapes (without saving the test results) and evaluate the mIoU.

```
python tools/test.py \
configs/deeplabv3plus/deeplabv3plus_r18-d8_512x1024_80k_cityscapes.py \
deeplabv3plus_r18-d8_512x1024_80k_cityscapes_20201226_080942-cff257fe.pth \
--eval-options efficient_test=True \
--eval mIoU
```

   Using `pmap` to view CPU memory footprint, it used 2.25GB CPU memory with `efficient_test=True` and 11.06GB CPU memory with `efficient_test=False` . This optional parameter can save a lot of memory. (After mmseg v0.17, efficient_test has not effect and we use a progressive mode to evaluation and format results efficiently by default.)

7. Test PSPNet on LoveDA test split with 1 GPU, and generate the png files to be submit to the official evaluation server.

   First, add following to config file `configs/pspnet/pspnet_r50-d8_512x512_80k_loveda.py`,

```
data = dict(
    test=dict(
        img_dir='img_dir/test',
        ann_dir='ann_dir/test'))
```

   Then run test.

```
python ./tools/test.py configs/pspnet/pspnet_r50-d8_512x512_80k_loveda.py \
    checkpoints/pspnet_r50-d8_512x512_80k_loveda_20211104_155728-88610f9f.pth \
    --format-only --eval-options "imgfile_prefix=./pspnet_test_results"
```

   You will get png files under `./pspnet_test_results` directory. You may run `zip -r -j Results.zip pspnet_test_results/` and submit the zip file to evaluation server.

CHAPTER

EIGHT

TUTORIAL 1: LEARN ABOUT CONFIGS

We incorporate modular and inheritance design into our config system, which is convenient to conduct various experiments. If you wish to inspect the config file, you may run `python tools/print_config.py /PATH/TO/CONFIG` to see the complete config. You may also pass `--cfg-options xxx.yyy=zzz` to see updated config.

8.1 Config File Structure

There are 4 basic component types under `config/_base_`, dataset, model, schedule, default_runtime. Many methods could be easily constructed with one of each like DeepLabV3, PSPNet. The configs that are composed by components from `_base_` are called *primitive*.

For all configs under the same folder, it is recommended to have only **one** *primitive* config. All other configs should inherit from the *primitive* config. In this way, the maximum of inheritance level is 3.

For easy understanding, we recommend contributors to inherit from existing methods. For example, if some modification is made base on DeepLabV3, user may first inherit the basic DeepLabV3 structure by specifying `_base_ = ../deeplabv3/deeplabv3_r50_512x1024_40ki_cityscapes.py`, then modify the necessary fields in the config files.

If you are building an entirely new method that does not share the structure with any of the existing methods, you may create a folder `xxxnet` under `configs`,

Please refer to mmcv for detailed documentation.

8.2 Config Name Style

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

```
{model}_{backbone}_[misc]_[gpu x batch_per_gpu]_{resolution}_{iterations}_{dataset}
```

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

- `{model}`: model type like `psp`, `deeplabv3`, etc.
- `{backbone}`: backbone type like `r50` (ResNet-50), `x101` (ResNeXt-101).
- `[misc]`: miscellaneous setting/plugins of model, e.g. `dconv`, `gcb`, `attention`, `mstrain`.
- `[gpu x batch_per_gpu]`: GPUs and samples per GPU, `8x2` is used by default.
- `{iterations}`: number of training iterations like `160k`.
- `{dataset}`: dataset like `cityscapes`, `voc12aug`, `ade`.

8.3 An Example of PSPNet

To help the users have a basic idea of a complete config and the modules in a modern semantic segmentation system, we make brief comments on the config of PSPNet using ResNet50V1c as the following. For more detailed usage and the corresponding alternative for each module, please refer to the API documentation.

```
norm_cfg = dict(type='SyncBN', requires_grad=True)  # Segmentation usually uses SyncBN
model = dict(
    type='EncoderDecoder',  # Name of segmentor
    pretrained='open-mmlab://resnet50_v1c',  # The ImageNet pretrained backbone to be␣
↪loaded
    backbone=dict(
        type='ResNetV1c',  # The type of backbone. Please refer to mmseg/models/
↪backbones/resnet.py for details.
        depth=50,  # Depth of backbone. Normally 50, 101 are used.
        num_stages=4,  # Number of stages of backbone.
        out_indices=(0, 1, 2, 3),  # The index of output feature maps produced in each␣
↪stages.
        dilations=(1, 1, 2, 4),  # The dilation rate of each layer.
        strides=(1, 2, 1, 1),  # The stride of each layer.
        norm_cfg=dict(  # The configuration of norm layer.
            type='SyncBN',  # Type of norm layer. Usually it is SyncBN.
            requires_grad=True),   # Whether to train the gamma and beta in norm
        norm_eval=False,  # Whether to freeze the statistics in BN
        style='pytorch',  # The style of backbone, 'pytorch' means that stride 2 layers␣
↪are in 3x3 conv, 'caffe' means stride 2 layers are in 1x1 convs.
        contract_dilation=True),  # When dilation > 1, whether contract first layer of␣
↪dilation.
    decode_head=dict(
        type='PSPHead',  # Type of decode head. Please refer to mmseg/models/decode_
↪heads for available options.
        in_channels=2048,  # Input channel of decode head.
        in_index=3,  # The index of feature map to select.
        channels=512,  # The intermediate channels of decode head.
        pool_scales=(1, 2, 3, 6),  # The avg pooling scales of PSPHead. Please refer to␣
↪paper for details.
        dropout_ratio=0.1,  # The dropout ratio before final classification layer.
        num_classes=19,  # Number of segmentation class. Usually 19 for cityscapes, 21␣
↪for VOC, 150 for ADE20k.
        norm_cfg=dict(type='SyncBN', requires_grad=True),  # The configuration of norm␣
↪layer.
        align_corners=False,  # The align_corners argument for resize in decoding.
        loss_decode=dict(  # Config of loss function for the decode_head.
            type='CrossEntropyLoss',  # Type of loss used for segmentation.
            use_sigmoid=False,  # Whether use sigmoid activation for segmentation.
            loss_weight=1.0)),  # Loss weight of decode head.
    auxiliary_head=dict(
        type='FCNHead',  # Type of auxiliary head. Please refer to mmseg/models/decode_
↪heads for available options.
        in_channels=1024,  # Input channel of auxiliary head.
        in_index=2,  # The index of feature map to select.
        channels=256,  # The intermediate channels of decode head.
        num_convs=1,  # Number of convs in FCNHead. It is usually 1 in auxiliary head.
```

(continues on next page)

(continued from previous page)

```
        concat_input=False,  # Whether concat output of convs with input before␣
↪classification layer.
        dropout_ratio=0.1,  # The dropout ratio before final classification layer.
        num_classes=19,  # Number of segmentation class. Usually 19 for cityscapes, 21␣
↪for VOC, 150 for ADE20k.
        norm_cfg=dict(type='SyncBN', requires_grad=True),  # The configuration of norm␣
↪layer.
        align_corners=False,  # The align_corners argument for resize in decoding.
        loss_decode=dict(  # Config of loss function for the decode_head.
            type='CrossEntropyLoss',  # Type of loss used for segmentation.
            use_sigmoid=False,  # Whether use sigmoid activation for segmentation.
            loss_weight=0.4)))  # Loss weight of auxiliary head, which is usually 0.4 of␣
↪decode head.
train_cfg = dict()  # train_cfg is just a place holder for now.
test_cfg = dict(mode='whole')  # The test mode, options are 'whole' and 'sliding'. 'whole':␣
↪whole image fully-convolutional test. 'sliding': sliding crop window on the image.
dataset_type = 'CityscapesDataset'  # Dataset type, this will be used to define the␣
↪dataset.
data_root = 'data/cityscapes/'  # Root path of data.
img_norm_cfg = dict(  # Image normalization config to normalize the input images.
    mean=[123.675, 116.28, 103.53],  # Mean values used to pre-training the pre-trained␣
↪backbone models.
    std=[58.395, 57.12, 57.375],  # 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.
crop_size = (512, 1024)  # The crop size during training.
train_pipeline = [  # Training pipeline.
    dict(type='LoadImageFromFile'),  # First pipeline to load images from file path.
    dict(type='LoadAnnotations'),  # Second pipeline to load annotations for current␣
↪image.
    dict(type='Resize',  # Augmentation pipeline that resize the images and their␣
↪annotations.
        img_scale=(2048, 1024),  # The largest scale of image.
        ratio_range=(0.5, 2.0)), # The augmented scale range as ratio.
    dict(type='RandomCrop',  # Augmentation pipeline that randomly crop a patch from␣
↪current image.
        crop_size=(512, 1024),  # The crop size of patch.
        cat_max_ratio=0.75),  # The max area ratio that could be occupied by single␣
↪category.
    dict(
        type='RandomFlip',  # Augmentation pipeline that flip the images and their␣
↪annotations
        flip_ratio=0.5),  # The ratio or probability to flip
    dict(type='PhotoMetricDistortion'),  # Augmentation pipeline that distort current␣
↪image with several photo metric methods.
    dict(
        type='Normalize',  # Augmentation pipeline that normalize the input images
        mean=[123.675, 116.28, 103.53],  # These keys are the same of img_norm_cfg since␣
↪the
        std=[58.395, 57.12, 57.375],  # keys of img_norm_cfg are used here as arguments
        to_rgb=True),
```

(continues on next page)

(continued from previous page)

```
    dict(type='Pad',  # Augmentation pipeline that pad the image to specified size.
        size=(512, 1024),  # The output size of padding.
        pad_val=0,  # The padding value for image.
        seg_pad_val=255),  # The padding value of 'gt_semantic_seg'.
    dict(type='DefaultFormatBundle'),  # Default format bundle to gather data in the␣
↪pipeline
    dict(type='Collect',  # Pipeline that decides which keys in the data should be␣
↪passed to the segmentor
        keys=['img', 'gt_semantic_seg'])
]
test_pipeline = [
    dict(type='LoadImageFromFile'),  # First pipeline to load images from file path
    dict(
        type='MultiScaleFlipAug',  # An encapsulation that encapsulates the test time␣
↪augmentations
        img_scale=(2048, 1024),  # Decides the largest scale for testing, used for the␣
↪Resize pipeline
        flip=False,  # Whether to flip images during testing
        transforms=[
            dict(type='Resize',  # Use resize augmentation
                 keep_ratio=True),  # Whether to keep the ratio between height and width,
↪ the img_scale set here will be suppressed by the img_scale set above.
            dict(type='RandomFlip'),  # Thought RandomFlip is added in pipeline, it is␣
↪not used when flip=False
            dict(
                type='Normalize',  # Normalization config, the values are from img_norm_
↪cfg
                mean=[123.675, 116.28, 103.53],
                std=[58.395, 57.12, 57.375],
                to_rgb=True),
            dict(type='ImageToTensor', # Convert image to tensor
                keys=['img']),
            dict(type='Collect', # Collect pipeline that collect necessary keys for␣
↪testing.
                keys=['img'])
        ])
]
data = dict(
    samples_per_gpu=2,  # Batch size of a single GPU
    workers_per_gpu=2,  # Worker to pre-fetch data for each single GPU
    train=dict(  # Train dataset config
        type='CityscapesDataset',  # Type of dataset, refer to mmseg/datasets/ for␣
↪details.
        data_root='data/cityscapes/',  # The root of dataset.
        img_dir='leftImg8bit/train',  # The image directory of dataset.
        ann_dir='gtFine/train',  # The annotation directory of dataset.
        pipeline=[  # pipeline, this is passed by the train_pipeline created before.
            dict(type='LoadImageFromFile'),
            dict(type='LoadAnnotations'),
            dict(
                type='Resize', img_scale=(2048, 1024), ratio_range=(0.5, 2.0)),
            dict(type='RandomCrop', crop_size=(512, 1024), cat_max_ratio=0.75),
```

(continues on next page)

(continued from previous page)

```
            dict(type='RandomFlip', flip_ratio=0.5),
            dict(type='PhotoMetricDistortion'),
            dict(
                type='Normalize',
                mean=[123.675, 116.28, 103.53],
                std=[58.395, 57.12, 57.375],
                to_rgb=True),
            dict(type='Pad', size=(512, 1024), pad_val=0, seg_pad_val=255),
            dict(type='DefaultFormatBundle'),
            dict(type='Collect', keys=['img', 'gt_semantic_seg'])
        ]),
    val=dict(  # Validation dataset config
        type='CityscapesDataset',
        data_root='data/cityscapes/',
        img_dir='leftImg8bit/val',
        ann_dir='gtFine/val',
        pipeline=[  # Pipeline is passed by test_pipeline created before
            dict(type='LoadImageFromFile'),
            dict(
                type='MultiScaleFlipAug',
                img_scale=(2048, 1024),
                flip=False,
                transforms=[
                    dict(type='Resize', keep_ratio=True),
                    dict(type='RandomFlip'),
                    dict(
                        type='Normalize',
                        mean=[123.675, 116.28, 103.53],
                        std=[58.395, 57.12, 57.375],
                        to_rgb=True),
                    dict(type='ImageToTensor', keys=['img']),
                    dict(type='Collect', keys=['img'])
                ])
        ]),
    test=dict(
        type='CityscapesDataset',
        data_root='data/cityscapes/',
        img_dir='leftImg8bit/val',
        ann_dir='gtFine/val',
        pipeline=[
            dict(type='LoadImageFromFile'),
            dict(
                type='MultiScaleFlipAug',
                img_scale=(2048, 1024),
                flip=False,
                transforms=[
                    dict(type='Resize', keep_ratio=True),
                    dict(type='RandomFlip'),
                    dict(
                        type='Normalize',
                        mean=[123.675, 116.28, 103.53],
                        std=[58.395, 57.12, 57.375],
```

(continues on next page)

(continued from previous page)

```
                    to_rgb=True),
                dict(type='ImageToTensor', keys=['img']),
                dict(type='Collect', keys=['img'])
            ])
        ]))
log_config = dict(  # config to register logger hook
    interval=50,  # Interval to print the log
    hooks=[
        dict(type='TextLoggerHook', by_epoch=False),
        dict(type='TensorboardLoggerHook', by_epoch=False),
        dict(type='MMSegWandbHook', by_epoch=False, # The Wandb logger is also supported,
↪ It requires `wandb` to be installed.
             init_kwargs={'entity': "OpenMMLab", # The entity used to log on Wandb
                          'project': "MMSeg", # Project name in WandB
                          'config': cfg_dict}), # Check https://docs.wandb.ai/ref/python/
↪init for more init arguments.
        # MMSegWandbHook is mmseg implementation of WandbLoggerHook. ClearMLLoggerHook,␣
↪DvcliveLoggerHook, MlflowLoggerHook, NeptuneLoggerHook, PaviLoggerHook,␣
↪SegmindLoggerHook are also supported based on MMCV implementation.
    ])

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 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. The workflow trains the model␣
↪by 40000 iterations according to the `runner.max_iters`.
cudnn_benchmark = True  # Whether use cudnn_benchmark to speed up, which is fast for␣
↪fixed input size.
optimizer = dict(  # Config used to build optimizer, support all the optimizers in␣
↪PyTorch whose arguments are also the same as those in PyTorch
    type='SGD',  # Type of optimizers, refer to https://github.com/open-mmlab/mmcv/blob/
↪master/mmcv/runner/optimizer/default_constructor.py#L13 for more details
    lr=0.01,  # Learning rate of optimizers, see detail usages of the parameters in the␣
↪documentation of PyTorch
    momentum=0.9,  # Momentum
    weight_decay=0.0005)  # Weight decay of SGD
optimizer_config = dict()  # Config used to build the optimizer hook, refer to https://
↪github.com/open-mmlab/mmcv/blob/master/mmcv/runner/hooks/optimizer.py#L8 for␣
↪implementation details.
lr_config = dict(
    policy='poly',  # The policy of scheduler, also support Step, 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.
    power=0.9,  # The power of polynomial decay.
    min_lr=0.0001,  # The minimum learning rate to stable the training.
    by_epoch=False)  # Whether count by epoch or not.
runner = dict(
```

(continues on next page)

(continued from previous page)

```
    type='IterBasedRunner', # Type of runner to use (i.e. IterBasedRunner or␣
↪EpochBasedRunner)
    max_iters=40000) # Total number of iterations. For EpochBasedRunner use `max_epochs`
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.
    by_epoch=False,  # Whether count by epoch or not.
    interval=4000)  # The save interval.
evaluation = dict(  # The config to build the evaluation hook. Please refer to mmseg/
↪core/evaluation/eval_hook.py for details.
    interval=4000,  # The interval of evaluation.
    metric='mIoU')  # The evaluation metric.
```

8.4 FAQ

8.4.1 Ignore some fields in the base configs

Sometimes, you may set `_delete_=True` to ignore some of the fields in base configs. You may refer to mmcv for simple illustration.

In MMSegmentation, for example, to change the backbone of PSPNet with the following config.

```
norm_cfg = dict(type='SyncBN', requires_grad=True)
model = dict(
    type='MaskRCNN',
    pretrained='torchvision://resnet50',
    backbone=dict(
        type='ResNetV1c',
        depth=50,
        num_stages=4,
        out_indices=(0, 1, 2, 3),
        dilations=(1, 1, 2, 4),
        strides=(1, 2, 1, 1),
        norm_cfg=norm_cfg,
        norm_eval=False,
        style='pytorch',
        contract_dilation=True),
    decode_head=dict(...),
    auxiliary_head=dict(...))
```

`ResNet` and `HRNet` use different keywords to construct.

```
_base_ = '../pspnet/psp_r50_512x1024_40ki_cityscpaes.py'
norm_cfg = dict(type='SyncBN', requires_grad=True)
model = dict(
    pretrained='open-mmlab://msra/hrnetv2_w32',
    backbone=dict(
        _delete_=True,
        type='HRNet',
```

(continues on next page)

(continued from previous page)

```
        norm_cfg=norm_cfg,
        extra=dict(
            stage1=dict(
                num_modules=1,
                num_branches=1,
                block='BOTTLENECK',
                num_blocks=(4, ),
                num_channels=(64, )),
            stage2=dict(
                num_modules=1,
                num_branches=2,
                block='BASIC',
                num_blocks=(4, 4),
                num_channels=(32, 64)),
            stage3=dict(
                num_modules=4,
                num_branches=3,
                block='BASIC',
                num_blocks=(4, 4, 4),
                num_channels=(32, 64, 128)),
            stage4=dict(
                num_modules=3,
                num_branches=4,
                block='BASIC',
                num_blocks=(4, 4, 4, 4),
                num_channels=(32, 64, 128, 256)))),
    decode_head=dict(...),
    auxiliary_head=dict(...))
```

The `_delete_=True` would replace all old keys in `backbone` field with new keys.

8.4.2 Use intermediate variables in configs

Some intermediate variables are used in the configs files, like `train_pipeline/test_pipeline` in datasets. It's worth noting that when modifying intermediate variables in the children configs, user need to pass the intermediate variables into corresponding fields again. For example, we would like to change multi scale strategy to train/test a PSPNet. `train_pipeline/test_pipeline` are intermediate variable we would like to modify.

```
_base_ = '../pspnet/psp_r50_512x1024_40ki_cityscapes.py'
crop_size = (512, 1024)
img_norm_cfg = dict(
    mean=[123.675, 116.28, 103.53], std=[58.395, 57.12, 57.375], to_rgb=True)
train_pipeline = [
    dict(type='LoadImageFromFile'),
    dict(type='LoadAnnotations'),
    dict(type='Resize', img_scale=(2048, 1024), ratio_range=(1.0, 2.0)),  # change to [1.
↪, 2.]
    dict(type='RandomCrop', crop_size=crop_size, cat_max_ratio=0.75),
    dict(type='RandomFlip', flip_ratio=0.5),
    dict(type='PhotoMetricDistortion'),
    dict(type='Normalize', **img_norm_cfg),
```

(continues on next page)

(continued from previous page)

```
    dict(type='Pad', size=crop_size, pad_val=0, seg_pad_val=255),
    dict(type='DefaultFormatBundle'),
    dict(type='Collect', keys=['img', 'gt_semantic_seg']),
]
test_pipeline = [
    dict(type='LoadImageFromFile'),
    dict(
        type='MultiScaleFlipAug',
        img_scale=(2048, 1024),
        img_ratios=[0.5, 0.75, 1.0, 1.25, 1.5, 1.75],  # change to multi scale testing
        flip=False,
        transforms=[
            dict(type='Resize', keep_ratio=True),
            dict(type='RandomFlip'),
            dict(type='Normalize', **img_norm_cfg),
            dict(type='ImageToTensor', keys=['img']),
            dict(type='Collect', keys=['img']),
        ])
]
data = dict(
    train=dict(pipeline=train_pipeline),
    val=dict(pipeline=test_pipeline),
    test=dict(pipeline=test_pipeline))
```

We first define the new `train_pipeline/test_pipeline` and pass them into `data`.

Similarly, if we would like to switch from `SyncBN` to `BN` or `MMSyncBN`, we need to substitute every `norm_cfg` in the config.

```
_base_ = '../pspnet/psp_r50_512x1024_40ki_cityscpaes.py'
norm_cfg = dict(type='BN', requires_grad=True)
model = dict(
    backbone=dict(norm_cfg=norm_cfg),
    decode_head=dict(norm_cfg=norm_cfg),
    auxiliary_head=dict(norm_cfg=norm_cfg))
```

CHAPTER

NINE

TUTORIAL 2: CUSTOMIZE DATASETS

9.1 Data configuration

`data` in config file is the variable for data configuration, to define the arguments that are used in datasets and dataloaders.

Here is an example of data configuration:

```
data = dict(
    samples_per_gpu=4,
    workers_per_gpu=4,
    train=dict(
        type='ADE20KDataset',
        data_root='data/ade/ADEChallengeData2016',
        img_dir='images/training',
        ann_dir='annotations/training',
        pipeline=train_pipeline),
    val=dict(
        type='ADE20KDataset',
        data_root='data/ade/ADEChallengeData2016',
        img_dir='images/validation',
        ann_dir='annotations/validation',
        pipeline=test_pipeline),
    test=dict(
        type='ADE20KDataset',
        data_root='data/ade/ADEChallengeData2016',
        img_dir='images/validation',
        ann_dir='annotations/validation',
        pipeline=test_pipeline))
```

- `train`, `val` and `test`: The `configs` to build dataset instances for model training, validation and testing by using `build and registry` mechanism.
- `samples_per_gpu`: How many samples per batch and per gpu to load during model training, and the `batch_size` of training is equal to `samples_per_gpu` times gpu number, e.g. when using 8 gpus for distributed data parallel trainig and `samples_per_gpu=4`, the `batch_size` is `8*4=32`. If you would like to define `batch_size` for testing and validation, please use `test_dataloaser` and `val_dataloader` with mmseg >=0.24.1.
- `workers_per_gpu`: How many subprocesses per gpu to use for data loading. `0` means that the data will be loaded in the main process.

Note: `samples_per_gpu` only works for model training, and the default setting of `samples_per_gpu` is 1 in mmseg when model testing and validation (DO NOT support batch inference yet).

Note: before v0.24.1, except `train`, `val` `test`, `samples_per_gpu` and `workers_per_gpu`, the other keys in `data` must be the input keyword arguments for `dataloader` in pytorch, and the dataloaders used for model training, validation and testing have the same input arguments. In v0.24.1, mmseg supports to use `train_dataloader`, `test_dataloaser` and `val_dataloader` to specify different keyword arguments, and still supports the overall arguments definition but the specific dataloader setting has a higher priority.

Here is an example for specific dataloader:

```
data = dict(
    samples_per_gpu=4,
    workers_per_gpu=4,
    shuffle=True,
    train=dict(type='xxx', ...),
    val=dict(type='xxx', ...),
    test=dict(type='xxx', ...),
    # Use different batch size during validation and testing.
    val_dataloader=dict(samples_per_gpu=1, workers_per_gpu=4, shuffle=False),
    test_dataloader=dict(samples_per_gpu=1, workers_per_gpu=4, shuffle=False))
```

Assume only one gpu used for model training and testing, as the priority of the overall arguments definition is low, the batch_size for training is 4 and dataset will be shuffled, and batch_size for testing and validation is 1, and dataset will not be shuffled.

To make data configuration much clearer, we recommend use specific dataloader setting instead of overall dataloader setting after v0.24.1, just like:

```
data = dict(
    train=dict(type='xxx', ...),
    val=dict(type='xxx', ...),
    test=dict(type='xxx', ...),
    # Use specific dataloader setting
    train_dataloader=dict(samples_per_gpu=4, workers_per_gpu=4, shuffle=True),
    val_dataloader=dict(samples_per_gpu=1, workers_per_gpu=4, shuffle=False),
    test_dataloader=dict(samples_per_gpu=1, workers_per_gpu=4, shuffle=False))
```

Note: in model training, default values in the script of mmseg for dataloader are `shuffle=True, and drop_last=True`, in model validation and testing, default values are `shuffle=False, and drop_last=False`

9.2 Customize datasets by reorganizing data

The simplest way is to convert your dataset to organize your data into folders.

An example of file structure is as followed.

```
├── data
│   ├── my_dataset
│   │   ├── img_dir
│   │   │   ├── train
│   │   │   │   ├── xxx{img_suffix}
│   │   │   │   ├── yyy{img_suffix}
│   │   │   │   ├── zzz{img_suffix}
│   │   │   ├── val
│   │   ├── ann_dir
│   │   │   ├── train
```

(continues on next page)

(continued from previous page)

```
│   │   │   │   ├── xxx{seg_map_suffix}
│   │   │   │   ├── yyy{seg_map_suffix}
│   │   │   │   ├── zzz{seg_map_suffix}
│   │   │   ├── val
```

A training pair will consist of the files with same suffix in img_dir/ann_dir.

If `split` argument is given, only part of the files in img_dir/ann_dir will be loaded. We may specify the prefix of files we would like to be included in the split txt.

More specifically, for a split txt like following,

```
xxx
zzz
```

Only `data/my_dataset/img_dir/train/xxx{img_suffix}`, `data/my_dataset/img_dir/train/zzz{img_suffix}`, `data/my_dataset/ann_dir/train/xxx{seg_map_suffix}`, `data/my_dataset/ann_dir/train/zzz{seg_map_suffix}` will be loaded.

Note: The annotations are images of shape (H, W), the value pixel should fall in range `[0, num_classes - 1]`. You may use `'P'` mode of pillow to create your annotation image with color.

9.3 Customize datasets by mixing dataset

MMSegmentation also supports to mix dataset for training. Currently it supports to concat, repeat and multi-image mix datasets.

9.3.1 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
        )
    )
```

9.3.2 Concatenate dataset

There 2 ways to concatenate the dataset.

1. If the datasets you want to concatenate are in the same type with different annotation files, you can concatenate the dataset configs like the following.

    1. You may concatenate two `ann_dir`.

    ```
    dataset_A_train = dict(
        type='Dataset_A',
        img_dir = 'img_dir',
        ann_dir = ['anno_dir_1', 'anno_dir_2'],
        pipeline=train_pipeline
    )
    ```

    2. You may concatenate two `split`.

    ```
    dataset_A_train = dict(
        type='Dataset_A',
        img_dir = 'img_dir',
        ann_dir = 'anno_dir',
        split = ['split_1.txt', 'split_2.txt'],
        pipeline=train_pipeline
    )
    ```

    3. You may concatenate two `ann_dir` and `split` simultaneously.

    ```
    dataset_A_train = dict(
        type='Dataset_A',
        img_dir = 'img_dir',
        ann_dir = ['anno_dir_1', 'anno_dir_2'],
        split = ['split_1.txt', 'split_2.txt'],
        pipeline=train_pipeline
    )
    ```

    In this case, `ann_dir_1` and `ann_dir_2` are corresponding to `split_1.txt` and `split_2.txt`.

2. In case the dataset you want to concatenate is different, you can concatenate the dataset configs like the following.

```
dataset_A_train = dict()
dataset_B_train = dict()

data = dict(
    imgs_per_gpu=2,
    workers_per_gpu=2,
    train = [
        dataset_A_train,
        dataset_B_train
    ],
    val = dataset_A_val,
    test = dataset_A_test
    )
```

A more complex example that repeats `Dataset_A` and `Dataset_B` by N and M times, respectively, and then concatenates the repeated datasets is as the following.

```
dataset_A_train = dict(
    type='RepeatDataset',
    times=N,
    dataset=dict(
        type='Dataset_A',
        ...
        pipeline=train_pipeline
    )
)
dataset_A_val = dict(
    ...
    pipeline=test_pipeline
)
dataset_A_test = dict(
    ...
    pipeline=test_pipeline
)
dataset_B_train = dict(
    type='RepeatDataset',
    times=M,
    dataset=dict(
        type='Dataset_B',
        ...
        pipeline=train_pipeline
    )
)
data = dict(
    imgs_per_gpu=2,
    workers_per_gpu=2,
    train = [
        dataset_A_train,
        dataset_B_train
    ],
    val = dataset_A_val,
    test = dataset_A_test
)
```

9.3.3 Multi-image Mix Dataset

We use `MultiImageMixDataset` as a wrapper to mix images from multiple datasets. `MultiImageMixDataset` can be used by multiple images mixed data augmentation like mosaic and mixup.

An example of using `MultiImageMixDataset` with `Mosaic` data augmentation:

```
train_pipeline = [
    dict(type='LoadImageFromFile'),
    dict(type='LoadAnnotations'),
    dict(type='RandomMosaic', prob=1),
    dict(type='Resize', img_scale=(1024, 512), keep_ratio=True),
    dict(type='RandomFlip', prob=0.5),
    dict(type='Normalize', **img_norm_cfg),
```

(continues on next page)

(continued from previous page)

```
    dict(type='DefaultFormatBundle'),
    dict(type='Collect', keys=['img', 'gt_semantic_seg']),
]

train_dataset = dict(
    type='MultiImageMixDataset',
    dataset=dict(
        classes=classes,
        palette=palette,
        type=dataset_type,
        reduce_zero_label=False,
        img_dir=data_root + "images/train",
        ann_dir=data_root + "annotations/train",
        pipeline=[
            dict(type='LoadImageFromFile'),
            dict(type='LoadAnnotations'),
        ]
    ),
    pipeline=train_pipeline
)
```

CHAPTER

TEN

TUTORIAL 3: CUSTOMIZE DATA PIPELINES

10.1 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. Since the data in semantic segmentation may not be the same size, we introduce a new `DataContainer` type in MMCV to help collect and distribute data of different size. See here for more details.

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, formatting and test-time augmentation.

Here is an pipeline example for PSPNet.

```
img_norm_cfg = dict(
    mean=[123.675, 116.28, 103.53], std=[58.395, 57.12, 57.375], to_rgb=True)
crop_size = (512, 1024)
train_pipeline = [
    dict(type='LoadImageFromFile'),
    dict(type='LoadAnnotations'),
    dict(type='Resize', img_scale=(2048, 1024), ratio_range=(0.5, 2.0)),
    dict(type='RandomCrop', crop_size=crop_size, cat_max_ratio=0.75),
    dict(type='RandomFlip', flip_ratio=0.5),
    dict(type='PhotoMetricDistortion'),
    dict(type='Normalize', **img_norm_cfg),
    dict(type='Pad', size=crop_size, pad_val=0, seg_pad_val=255),
    dict(type='DefaultFormatBundle'),
    dict(type='Collect', keys=['img', 'gt_semantic_seg']),
]
test_pipeline = [
    dict(type='LoadImageFromFile'),
    dict(
        type='MultiScaleFlipAug',
        img_scale=(2048, 1024),
        # img_ratios=[0.5, 0.75, 1.0, 1.25, 1.5, 1.75],
        flip=False,
        transforms=[
            dict(type='Resize', keep_ratio=True),
            dict(type='RandomFlip'),
            dict(type='Normalize', **img_norm_cfg),
```

(continues on next page)

(continued from previous page)

```
            dict(type='ImageToTensor', keys=['img']),
            dict(type='Collect', keys=['img']),
        ])
]
```

For each operation, we list the related dict fields that are added/updated/removed.

10.1.1 Data loading

`LoadImageFromFile`

- add: img, img_shape, ori_shape

`LoadAnnotations`

- add: gt_semantic_seg, seg_fields

10.1.2 Pre-processing

`Resize`

- add: scale, scale_idx, pad_shape, scale_factor, keep_ratio
- update: img, img_shape, *seg_fields

`RandomFlip`

- add: flip
- update: img, *seg_fields

`Pad`

- add: pad_fixed_size, pad_size_divisor
- update: img, pad_shape, *seg_fields

`RandomCrop`

- update: img, pad_shape, *seg_fields

`Normalize`

- add: img_norm_cfg
- update: img

`SegRescale`

- update: gt_semantic_seg

`PhotoMetricDistortion`

- update: img

10.1.3 Formatting

`ToTensor`

- update: specified by `keys`.

`ImageToTensor`

- update: specified by `keys`.

`Transpose`

- update: specified by `keys`.

`ToDataContainer`

- update: specified by `fields`.

`DefaultFormatBundle`

- update: img, gt_semantic_seg

`Collect`

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

10.1.4 Test time augmentation

`MultiScaleFlipAug`

10.2 Extend and use custom pipelines

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

```
from mmseg.datasets import PIPELINES

@PIPELINES.register_module()
class MyTransform:

    def __call__(self, results):
        results['dummy'] = True
        return results
```

2. Import the new class.

```
from .my_pipeline import MyTransform
```

3. Use it in config files.

```
img_norm_cfg = dict(
    mean=[123.675, 116.28, 103.53], std=[58.395, 57.12, 57.375], to_rgb=True)
crop_size = (512, 1024)
train_pipeline = [
    dict(type='LoadImageFromFile'),
    dict(type='LoadAnnotations'),
```

(continues on next page)

(continued from previous page)

```
    dict(type='Resize', img_scale=(2048, 1024), ratio_range=(0.5, 2.0)),
    dict(type='RandomCrop', crop_size=crop_size, cat_max_ratio=0.75),
    dict(type='RandomFlip', flip_ratio=0.5),
    dict(type='PhotoMetricDistortion'),
    dict(type='Normalize', **img_norm_cfg),
    dict(type='Pad', size=crop_size, pad_val=0, seg_pad_val=255),
    dict(type='MyTransform'),
    dict(type='DefaultFormatBundle'),
    dict(type='Collect', keys=['img', 'gt_semantic_seg']),
]
```

CHAPTER

ELEVEN

TUTORIAL 4: CUSTOMIZE MODELS

11.1 Customize optimizer

Assume you want to add a optimizer named as `MyOptimizer`, which has arguments `a`, `b`, and `c`. You need to first implement the new optimizer in a file, e.g., in `mmseg/core/optimizer/my_optimizer.py`:

```
from mmcv.runner import OPTIMIZERS
from torch.optim import Optimizer


@OPTIMIZERS.register_module
class MyOptimizer(Optimizer):

    def __init__(self, a, b, c)
```

Then add this module in `mmseg/core/optimizer/__init__.py` thus the registry will find the new module and add it:

```
from .my_optimizer import MyOptimizer
```

Then you can use `MyOptimizer` in `optimizer` field of config files. In the configs, the optimizers are defined by the field `optimizer` like the following:

```
optimizer = dict(type='SGD', lr=0.02, momentum=0.9, weight_decay=0.0001)
```

To use your own optimizer, the field can be changed as

```
optimizer = dict(type='MyOptimizer', a=a_value, b=b_value, c=c_value)
```

We already support to use all the optimizers implemented by PyTorch, and the only modification is to change the `optimizer` field of config files. For example, if you want to use `ADAM`, though the performance will drop a lot, the modification could be as the following.

```
optimizer = dict(type='Adam', lr=0.0003, weight_decay=0.0001)
```

The users can directly set arguments following the API doc of PyTorch.

11.2 Customize optimizer constructor

Some models may have some parameter-specific settings for optimization, e.g. weight decay for BatchNoarm layers. The users can do those fine-grained parameter tuning through customizing optimizer constructor.

```
from mmcv.utils import build_from_cfg

from mmcv.runner import OPTIMIZER_BUILDERS
from .cocktail_optimizer import CocktailOptimizer


@OPTIMIZER_BUILDERS.register_module
class CocktailOptimizerConstructor(object):

    def __init__(self, optimizer_cfg, paramwise_cfg=None):

    def __call__(self, model):

        return my_optimizer
```

11.3 Develop new components

There are mainly 2 types of components in MMSegmentation.

- backbone: usually stacks of convolutional network to extract feature maps, e.g., ResNet, HRNet.
- head: the component for semantic segmentation map decoding.

11.3.1 Add new backbones

Here we show how to develop new components with an example of MobileNet.

1. Create a new file `mmseg/models/backbones/mobilenet.py`.

```
import torch.nn as nn

from ..builder import BACKBONES


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

    def __init__(self, arg1, arg2):
        pass

    def forward(self, x):  # should return a tuple
        pass

    def init_weights(self, pretrained=None):
        pass
```

2. Import the module in mmseg/models/backbones/__init__.py.

```
from .mobilenet import MobileNet
```

3. Use it in your config file.

```
model = dict(
    ...
    backbone=dict(
        type='MobileNet',
        arg1=xxx,
        arg2=xxx),
    ...
```

11.3.2 Add new heads

In MMSegmentation, we provide a base BaseDecodeHead for all segmentation head. All newly implemented decode heads should be derived from it. Here we show how to develop a new head with the example of PSPNet as the following.

First, add a new decode head in mmseg/models/decode_heads/psp_head.py. PSPNet implements a decode head for segmentation decode. To implement a decode head, basically we need to implement three functions of the new module as the following.

```
@HEADS.register_module()
class PSPHead(BaseDecodeHead):

    def __init__(self, pool_scales=(1, 2, 3, 6), **kwargs):
        super(PSPHead, self).__init__(**kwargs)

    def init_weights(self):

    def forward(self, inputs):
```

Next, the users need to add the module in the mmseg/models/decode_heads/__init__.py thus the corresponding registry could find and load them.

To config file of PSPNet is as the following

```
norm_cfg = dict(type='SyncBN', requires_grad=True)
model = dict(
    type='EncoderDecoder',
    pretrained='pretrain_model/resnet50_v1c_trick-2cccc1ad.pth',
    backbone=dict(
        type='ResNetV1c',
        depth=50,
        num_stages=4,
        out_indices=(0, 1, 2, 3),
        dilations=(1, 1, 2, 4),
        strides=(1, 2, 1, 1),
        norm_cfg=norm_cfg,
        norm_eval=False,
        style='pytorch',
        contract_dilation=True),
```

(continues on next page)

(continued from previous page)

```
    decode_head=dict(
        type='PSPHead',
        in_channels=2048,
        in_index=3,
        channels=512,
        pool_scales=(1, 2, 3, 6),
        dropout_ratio=0.1,
        num_classes=19,
        norm_cfg=norm_cfg,
        align_corners=False,
        loss_decode=dict(
            type='CrossEntropyLoss', use_sigmoid=False, loss_weight=1.0)))
```

11.3.3 Add new loss

Assume you want to add a new loss as `MyLoss` for segmentation decode. To add a new loss function, the users need implement it in `mmseg/models/losses/my_loss.py`. The decorator `weighted_loss` enable the loss to be weighted for each element.

```
import torch
import torch.nn as nn

from ..builder import LOSSES
from .utils import weighted_loss

@weighted_loss
def my_loss(pred, target):
    assert pred.size() == target.size() and target.numel() > 0
    loss = torch.abs(pred - target)
    return loss

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

    def __init__(self, reduction='mean', loss_weight=1.0):
        super(MyLoss, self).__init__()
        self.reduction = reduction
        self.loss_weight = loss_weight

    def forward(self,
                pred,
                target,
                weight=None,
                avg_factor=None,
                reduction_override=None):
        assert reduction_override in (None, 'none', 'mean', 'sum')
        reduction = (
            reduction_override if reduction_override else self.reduction)
        loss = self.loss_weight * my_loss(
            pred, target, weight, reduction=reduction, avg_factor=avg_factor)
```

(continues on next page)

(continued from previous page)

```
        return loss
```

Then the users need to add it in the `mmseg/models/losses/__init__.py`.

```
from .my_loss import MyLoss, my_loss
```

To use it, modify the `loss_xxx` field. Then you need to modify the `loss_decode` field in the head. `loss_weight` could be used to balance multiple losses.

```
loss_decode=dict(type='MyLoss', loss_weight=1.0))
```

CHAPTER

TWELVE

TUTORIAL 5: TRAINING TRICKS

MMSegmentation support following training tricks out of box.

12.1 Different Learning Rate(LR) for Backbone and Heads

In semantic segmentation, some methods make the LR of heads larger than backbone to achieve better performance or faster convergence.

In MMSegmentation, you may add following lines to config to make the LR of heads 10 times of backbone.

```
optimizer=dict(
    paramwise_cfg = dict(
        custom_keys={
            'head': dict(lr_mult=10.)}))
```

With this modification, the LR of any parameter group with `'head'` in name will be multiplied by 10. You may refer to MMCV doc for further details.

12.2 Online Hard Example Mining (OHEM)

We implement pixel sampler here for training sampling. Here is an example config of training PSPNet with OHEM enabled.

```
_base_ = './pspnet_r50-d8_512x1024_40k_cityscapes.py'
model=dict(
    decode_head=dict(
        sampler=dict(type='OHEMPixelSampler', thresh=0.7, min_kept=100000)) )
```

In this way, only pixels with confidence score under 0.7 are used to train. And we keep at least 100000 pixels during training. If `thresh` is not specified, pixels of top `min_kept` loss will be selected.

12.3 Class Balanced Loss

For dataset that is not balanced in classes distribution, you may change the loss weight of each class. Here is an example for cityscapes dataset.

```
_base_ = './pspnet_r50-d8_512x1024_40k_cityscapes.py'
model=dict(
    decode_head=dict(
        loss_decode=dict(
            type='CrossEntropyLoss', use_sigmoid=False, loss_weight=1.0,
            # DeepLab used this class weight for cityscapes
            class_weight=[0.8373, 0.9180, 0.8660, 1.0345, 1.0166, 0.9969, 0.9754,
                          1.0489, 0.8786, 1.0023, 0.9539, 0.9843, 1.1116, 0.9037,
                          1.0865, 1.0955, 1.0865, 1.1529, 1.0507])))
```

`class_weight` will be passed into `CrossEntropyLoss` as `weight` argument. Please refer to PyTorch Doc for details.

12.4 Multiple Losses

For loss calculation, we support multiple losses training concurrently. Here is an example config of training `unet` on `DRIVE` dataset, whose loss function is `1:3` weighted sum of `CrossEntropyLoss` and `DiceLoss`:

```
_base_ = './fcn_unet_s5-d16_64x64_40k_drive.py'
model = dict(
    decode_head=dict(loss_decode=[dict(type='CrossEntropyLoss', loss_name='loss_ce',␣
↪loss_weight=1.0),
            dict(type='DiceLoss', loss_name='loss_dice', loss_weight=3.0)]),
    auxiliary_head=dict(loss_decode=[dict(type='CrossEntropyLoss', loss_name='loss_ce',
↪loss_weight=1.0),
            dict(type='DiceLoss', loss_name='loss_dice', loss_weight=3.0)]),
    )
```

In this way, `loss_weight` and `loss_name` will be weight and name in training log of corresponding loss, respectively.

Note: If you want this loss item to be included into the backward graph, `loss_` must be the prefix of the name.

12.5 Ignore specified label index in loss calculation

In default setting, `avg_non_ignore=False` which means each pixel counts for loss calculation although some of them belong to ignore-index labels.

For loss calculation, we support ignore index of certain label by `avg_non_ignore` and `ignore_index`. In this way, the average loss would only be calculated in non-ignored labels which may achieve better performance, and here is the reference. Here is an example config of training `unet` on `Cityscapes` dataset: in loss calculation it would ignore label 0 which is background and loss average is only calculated on non-ignore labels:

```
_base_ = './fcn_unet_s5-d16_4x4_512x1024_160k_cityscapes.py'
model = dict(
    decode_head=dict(
        ignore_index=0,
        loss_decode=dict(
```

(continues on next page)

(continued from previous page)

```
            type='CrossEntropyLoss', use_sigmoid=False, loss_weight=1.0, avg_non_
↪ignore=True),
    auxiliary_head=dict(
        ignore_index=0,
        loss_decode=dict(
            type='CrossEntropyLoss', use_sigmoid=False, loss_weight=1.0, avg_non_
↪ignore=True)),
    ))
```

CHAPTER

THIRTEEN

TUTORIAL 6: CUSTOMIZE RUNTIME SETTINGS

13.1 Customize optimization settings

13.1.1 Customize optimizer supported by Pytorch

We already support to use all the optimizers implemented by PyTorch, and the only modification is to change the `optimizer` field of config files. For example, if you want to use `ADAM` (note that the performance could drop a lot), the modification could be as the following.

```
optimizer = dict(type='Adam', lr=0.0003, weight_decay=0.0001)
```

To modify the learning rate of the model, the users only need to modify the `lr` in the config of optimizer. The users can directly set arguments following the API doc of PyTorch.

13.1.2 Customize self-implemented optimizer

1. Define a new optimizer

A customized optimizer could be defined as following.

Assume you want to add a optimizer named `MyOptimizer`, which has arguments `a`, `b`, and `c`. You need to create a new directory named `mmseg/core/optimizer`. And then implement the new optimizer in a file, e.g., in `mmseg/core/optimizer/my_optimizer.py`:

```
from .registry import OPTIMIZERS
from torch.optim import Optimizer


@OPTIMIZERS.register_module()
class MyOptimizer(Optimizer):

    def __init__(self, a, b, c)
```

2. Add the optimizer to registry

To find the above module defined above, this module should be imported into the main namespace at first. There are two options to achieve it.

- Modify `mmseg/core/optimizer/__init__.py` to import it.

  The newly defined module should be imported in `mmseg/core/optimizer/__init__.py` so that the registry will find the new module and add it:

```
from .my_optimizer import MyOptimizer
```

- Use `custom_imports` in the config to manually import it

```
custom_imports = dict(imports=['mmseg.core.optimizer.my_optimizer'], allow_failed_
↪imports=False)
```

The module `mmseg.core.optimizer.my_optimizer` will be imported at the beginning of the program and the class `MyOptimizer` is then automatically registered. Note that only the package containing the class `MyOptimizer` should be imported. `mmseg.core.optimizer.my_optimizer.MyOptimizer` **cannot** be imported directly.

Actually users can use a totally different file directory structure using this importing method, as long as the module root can be located in `PYTHONPATH`.

3. Specify the optimizer in the config file

Then you can use `MyOptimizer` in `optimizer` field of config files. In the configs, the optimizers are defined by the field `optimizer` like the following:

```
optimizer = dict(type='SGD', lr=0.02, momentum=0.9, weight_decay=0.0001)
```

To use your own optimizer, the field can be changed to

```
optimizer = dict(type='MyOptimizer', a=a_value, b=b_value, c=c_value)
```

13.1.3 Customize optimizer constructor

Some models may have some parameter-specific settings for optimization, e.g. weight decay for BatchNorm layers. The users can do those fine-grained parameter tuning through customizing optimizer constructor.

```
from mmcv.utils import build_from_cfg

from mmcv.runner.optimizer import OPTIMIZER_BUILDERS, OPTIMIZERS
from mmseg.utils import get_root_logger
from .my_optimizer import MyOptimizer


@OPTIMIZER_BUILDERS.register_module()
class MyOptimizerConstructor(object):

    def __init__(self, optimizer_cfg, paramwise_cfg=None):

    def __call__(self, model):

```

(continues on next page)

(continued from previous page)

```
        return my_optimizer
```

The default optimizer constructor is implemented here, which could also serve as a template for new optimizer constructor.

13.1.4 Additional settings

Tricks not implemented by the optimizer should be implemented through optimizer constructor (e.g., set parameter-wise learning rates) or hooks. We list some common settings that could stabilize the training or accelerate the training. Feel free to create PR, issue for more settings.

- **Use gradient clip to stabilize training**: Some models need gradient clip to clip the gradients to stabilize the training process. An example is as below:

```
optimizer_config = dict(
    _delete_=True, grad_clip=dict(max_norm=35, norm_type=2))
```

  If your config inherits the base config which already sets the `optimizer_config`, you might need `_delete_=True` to override the unnecessary settings. See the config documentation for more details.

- **Use momentum schedule to accelerate model convergence**: We support momentum scheduler to modify model's momentum according to learning rate, which could make the model converge in a faster way. Momentum scheduler is usually used with LR scheduler, for example, the following config is used in 3D detection to accelerate convergence. For more details, please refer to the implementation of CyclicLrUpdater and CyclicMomentumUpdater.

```
lr_config = dict(
    policy='cyclic',
    target_ratio=(10, 1e-4),
    cyclic_times=1,
    step_ratio_up=0.4,
)
momentum_config = dict(
    policy='cyclic',
    target_ratio=(0.85 / 0.95, 1),
    cyclic_times=1,
    step_ratio_up=0.4,
)
```

13.2 Customize training schedules

By default we use step learning rate with 40k/80k schedule, this calls `PolyLrUpdaterHook` in MMCV. We support many other learning rate schedule here, such as `CosineAnnealing` and `Poly` schedule. Here are some examples

- Step schedule:

```
lr_config = dict(policy='step', step=[9, 10])
```

- ConsineAnnealing schedule:

```
lr_config = dict(
    policy='CosineAnnealing',
    warmup='linear',
    warmup_iters=1000,
    warmup_ratio=1.0 / 10,
    min_lr_ratio=1e-5)
```

13.3 Customize workflow

Workflow is a list of (phase, epochs) to specify the running order and epochs. By default it is set to be

```
workflow = [('train', 1)]
```

which means running 1 epoch for training. Sometimes user may want to check some metrics (e.g. loss, accuracy) about the model on the validate set. In such case, we can set the workflow as

```
[('train', 1), ('val', 1)]
```

so that 1 epoch for training and 1 epoch for validation will be run iteratively.

Note:

1. The parameters of model will not be updated during val epoch.
2. Keyword `total_epochs` in the config only controls the number of training epochs and will not affect the validation workflow.
3. Workflows `[('train', 1), ('val', 1)]` and `[('train', 1)]` will not change the behavior of `EvalHook` because `EvalHook` is called by `after_train_epoch` and validation workflow only affect hooks that are called through `after_val_epoch`. Therefore, the only difference between `[('train', 1), ('val', 1)]` and `[('train', 1)]` is that the runner will calculate losses on validation set after each training epoch.

13.4 Customize hooks

13.4.1 Use hooks implemented in MMCV

If the hook is already implemented in MMCV, you can directly modify the config to use the hook as below

```
custom_hooks = [
    dict(type='MyHook', a=a_value, b=b_value, priority='NORMAL')
]
```

13.4.2 Modify default runtime hooks

There are some common hooks that are not registered through `custom_hooks`, they are

- log_config
- checkpoint_config
- evaluation
- lr_config
- optimizer_config
- momentum_config

In those hooks, only the logger hook has the `VERY_LOW` priority, others' priority are `NORMAL`. The above-mentioned tutorials already covers how to modify `optimizer_config`, `momentum_config`, and `lr_config`. Here we reveals how what we can do with `log_config`, `checkpoint_config`, and `evaluation`.

Checkpoint config

The MMCV runner will use `checkpoint_config` to initialize `CheckpointHook`.

```
checkpoint_config = dict(interval=1)
```

The users could set `max_keep_ckpts` to only save only small number of checkpoints or decide whether to store state dict of optimizer by `save_optimizer`. More details of the arguments are here

Log config

The `log_config` wraps multiple logger hooks and enables to set intervals. Now MMCV supports `WandbLoggerHook`, `MlflowLoggerHook`, and `TensorboardLoggerHook`. The detail usages can be found in the doc.

```
log_config = dict(
    interval=50,
    hooks=[
        dict(type='TextLoggerHook'),
        dict(type='TensorboardLoggerHook')
    ])
```

Evaluation config

The config of `evaluation` will be used to initialize the `EvalHook`. Except the key `interval`, other arguments such as `metric` will be passed to the `dataset.evaluate()`

```
evaluation = dict(interval=1, metric='mIoU')
```

CHAPTER

FOURTEEN

USEFUL TOOLS

Apart from training/testing scripts, We provide lots of useful tools under the `tools/` directory.

14.1 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}]
```

You will get the result like this.

```
==============================
Input shape: (3, 2048, 1024)
Flops: 1429.68 GMac
Params: 48.98 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, 1280, 800). (2) Some operators are not counted into FLOPs like GN and custom operators.

14.2 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/pspnet/latest.pth psp_r50_hszhao_200ep.pth
```

The final output filename will be `psp_r50_512x1024_40ki_cityscapes-{hash id}.pth`.

14.3 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 \
    ${CONFIG_FILE} \
    --checkpoint ${CHECKPOINT_FILE} \
    --output-file ${ONNX_FILE} \
    --input-img ${INPUT_IMG} \
    --shape ${INPUT_SHAPE} \
    --rescale-shape ${RESCALE_SHAPE} \
    --show \
    --verify \
    --dynamic-export \
    --cfg-options \
      model.test_cfg.mode="whole"
```

Description of arguments:

- `config` : The path of a model config file.
- `--checkpoint` : The path of a model checkpoint file.
- `--output-file`: The path of output ONNX model. If not specified, it will be set to `tmp.onnx`.
- `--input-img` : The path of an input image for conversion and visualize.
- `--shape`: The height and width of input tensor to the model. If not specified, it will be set to img_scale of test_pipeline.
- `--rescale-shape`: rescale shape of output, set this value to avoid OOM, only work on `slide` mode.
- `--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`.
- `--cfg-options`:Update config options.

Note: This tool is still experimental. Some customized operators are not supported for now.

14.4 Evaluate ONNX model

We provide `tools/deploy_test.py` to evaluate ONNX model with different backend.

14.4.1 Prerequisite

- Install onnx and onnxruntime-gpu

```
pip install onnx onnxruntime-gpu
```

- Install TensorRT following how-to-build-tensorrt-plugins-in-mmcv(optional)

14.4.2 Usage

```
python tools/deploy_test.py \
    ${CONFIG_FILE} \
    ${MODEL_FILE} \
    ${BACKEND} \
    --out ${OUTPUT_FILE} \
    --eval ${EVALUATION_METRICS} \
    --show \
    --show-dir ${SHOW_DIRECTORY} \
    --cfg-options ${CFG_OPTIONS} \
    --eval-options ${EVALUATION_OPTIONS} \
    --opacity ${OPACITY} \
```

Description of all arguments

- `config`: The path of a model config file.
- `model`: The path of a converted model file.
- `backend`: Backend of the inference, options: `onnxruntime`, `tensorrt`.
- `--out`: The path of output result file in pickle format.
- `--format-only` : Format the output results without perform evaluation. It is useful when you want to format the result to a specific format and submit it to the test server. If not specified, it will be set to `False`. Note that this argument is **mutually exclusive** with `--eval`.
- `--eval`: Evaluation metrics, which depends on the dataset, e.g., "mIoU" for generic datasets, and "cityscapes" for Cityscapes. Note that this argument is **mutually exclusive** with `--format-only`.
- `--show`: Show results flag.
- `--show-dir`: Directory where painted images will be saved
- `--cfg-options`: Override some settings in the used config file, the key-value pair in `xxx=yyy` format will be merged into config file.
- `--eval-options`: Custom options for evaluation, the key-value pair in `xxx=yyy` format will be kwargs for `dataset.evaluate()` function
- `--opacity`: Opacity of painted segmentation map. In (0, 1] range.

14.4.3 Results and Models

Note: TensorRT is only available on configs with `whole mode`.

14.5 Convert to TorchScript (experimental)

We also provide a script to convert model to TorchScript format. You can use the pytorch C++ API LibTorch inference the trained model. The converted model could be visualized by tools like Netron. Besides, we also support comparing the output results between PyTorch and TorchScript model.

```
python tools/pytorch2torchscript.py \
    ${CONFIG_FILE} \
    --checkpoint ${CHECKPOINT_FILE} \
    --output-file ${ONNX_FILE}
    --shape ${INPUT_SHAPE}
    --verify \
    --show
```

Description of arguments:

- `config` : The path of a pytorch model config file.
- `--checkpoint` : The path of a pytorch model checkpoint file.
- `--output-file`: The path of output TorchScript model. If not specified, it will be set to `tmp.pt`.
- `--input-img` : The path of an input image for conversion and visualize.
- `--shape`: The height and width of input tensor to the model. If not specified, it will be set to `512 512`.
- `--show`: Determines whether to print the traced graph 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`.

Note: It's only support PyTorch>=1.8.0 for now.

Note: This tool is still experimental. Some customized operators are not supported for now.

Examples:

- Convert the cityscapes PSPNet pytorch model.

```
python tools/pytorch2torchscript.py configs/pspnet/pspnet_r50-d8_512x1024_40k_
↪cityscapes.py \
--checkpoint checkpoints/pspnet_r50-d8_512x1024_40k_cityscapes_20200605_003338-
↪2966598c.pth \
--output-file checkpoints/pspnet_r50-d8_512x1024_40k_cityscapes_20200605_003338-
↪2966598c.pt \
--shape 512 1024
```

14.6 Convert to TensorRT (experimental)

A script to convert ONNX model to TensorRT format.

Prerequisite

- install `mmcv-full` with ONNXRuntime custom ops and TensorRT plugins follow ONNXRuntime in mmcv and TensorRT plugin in mmcv.
- Use *pytorch2onnx* to convert the model from PyTorch to ONNX.

Usage

```
python ${MMSEG_PATH}/tools/onnx2tensorrt.py \
    ${CFG_PATH} \
    ${ONNX_PATH} \
    --trt-file ${OUTPUT_TRT_PATH} \
    --min-shape ${MIN_SHAPE} \
    --max-shape ${MAX_SHAPE} \
    --input-img ${INPUT_IMG} \
    --show \
    --verify
```

Description of all arguments

- `config` : Config file of the model.
- `model` : Path to the input ONNX model.
- `--trt-file` : Path to the output TensorRT engine.
- `--max-shape` : Maximum shape of model input.
- `--min-shape` : Minimum shape of model input.
- `--fp16` : Enable fp16 model conversion.
- `--workspace-size` : Max workspace size in GiB.
- `--input-img` : Image for visualize.
- `--show` : Enable result visualize.
- `--dataset` : Palette provider, `CityscapesDataset` as default.
- `--verify` : Verify the outputs of ONNXRuntime and TensorRT.
- `--verbose` : Whether to verbose logging messages while creating TensorRT engine. Defaults to False.

Note: Only tested on whole mode.

CHAPTER

FIFTEEN

MISCELLANEOUS

15.1 Print the entire config

`tools/print_config.py` prints the whole config verbatim, expanding all its imports.

```
python tools/print_config.py \
  ${CONFIG} \
  --graph \
  --cfg-options ${OPTIONS [OPTIONS...]} \
```

Description of arguments:

- `config` : The path of a pytorch model config file.
- `--graph` : Determines whether to print the models graph.
- `--cfg-options`: Custom options to replace the config file.

15.2 Plot training logs

`tools/analyze_logs.py` plots loss/mIoU curves given a training log file. `pip install seaborn` first to install the dependency.

```
python tools/analyze_logs.py xxx.log.json [--keys ${KEYS}] [--legend ${LEGEND}] [--
↪backend ${BACKEND}] [--style ${STYLE}] [--out ${OUT_FILE}]
```

Examples:

- Plot the mIoU, mAcc, aAcc metrics.

  ```
  python tools/analyze_logs.py log.json --keys mIoU mAcc aAcc --legend mIoU mAcc aAcc
  ```

- Plot loss metric.

  ```
  python tools/analyze_logs.py log.json --keys loss --legend loss
  ```

15.3 Model conversion

`tools/model_converters/` provide several scripts to convert pretrain models released by other repos to MMSegmentation style.

15.3.1 ViT Swin MiT Transformer Models

- ViT

  `tools/model_converters/vit2mmseg.py` convert keys in timm pretrained vit models to MMSegmentation style.

```
python tools/model_converters/vit2mmseg.py ${SRC} ${DST}
```

- Swin

  `tools/model_converters/swin2mmseg.py` convert keys in official pretrained swin models to MMSegmentation style.

```
python tools/model_converters/swin2mmseg.py ${SRC} ${DST}
```

- SegFormer

  `tools/model_converters/mit2mmseg.py` convert keys in official pretrained mit models to MMSegmentation style.

```
python tools/model_converters/mit2mmseg.py ${SRC} ${DST}
```

CHAPTER

SIXTEEN

MODEL SERVING

In order to serve an `MMSegmentation` model with TorchServe, you can follow the steps:

16.1 1. Convert model from MMSegmentation to TorchServe

```
python tools/torchserve/mmseg2torchserve.py ${CONFIG_FILE} ${CHECKPOINT_FILE} \
--output-folder ${MODEL_STORE} \
--model-name ${MODEL_NAME}
```

Note: ${MODEL_STORE} needs to be an absolute path to a folder.

16.2 2. Build `mmseg-serve` docker image

```
docker build -t mmseg-serve:latest docker/serve/
```

16.3 3. Run `mmseg-serve`

Check the official docs for running TorchServe with docker.

In order to run in GPU, you need to install nvidia-docker. You can omit the `--gpus` argument in order to run in CPU.

Example:

```
docker run --rm \
--cpus 8 \
--gpus device=0 \
-p8080:8080 -p8081:8081 -p8082:8082 \
--mount type=bind,source=$MODEL_STORE,target=/home/model-server/model-store \
mmseg-serve:latest
```

Read the docs about the Inference (8080), Management (8081) and Metrics (8082) APIs

16.4 4. Test deployment

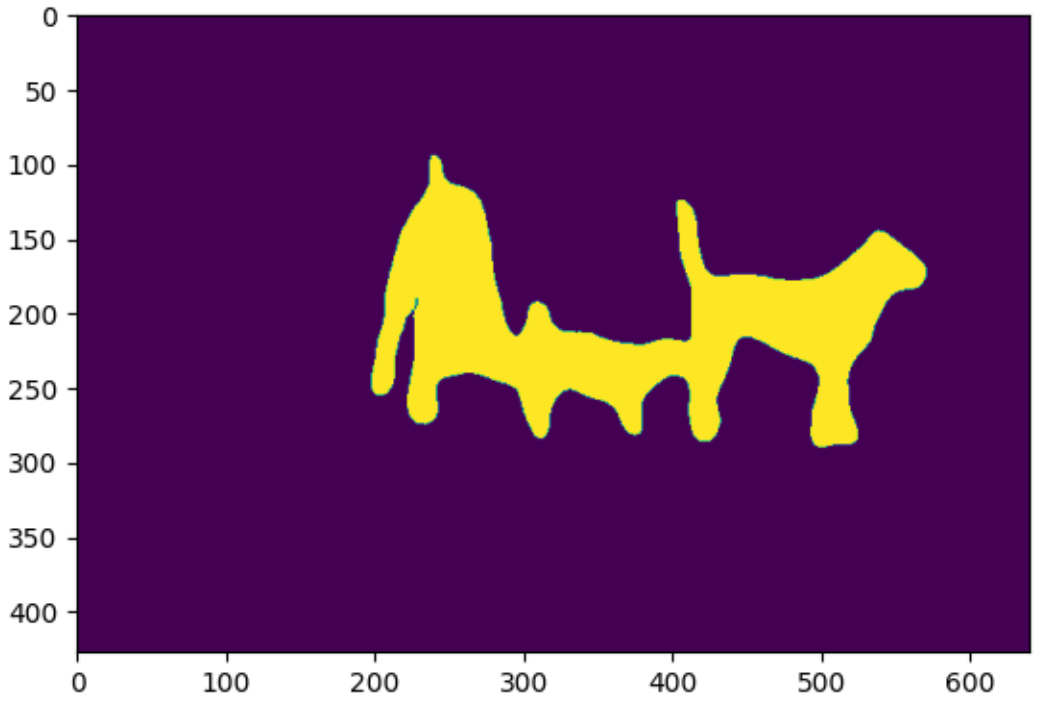
```
curl -O https://raw.githubusercontent.com/open-mmlab/mmsegmentation/master/resources/
↪3dogs.jpg
curl http://127.0.0.1:8080/predictions/${MODEL_NAME} -T 3dogs.jpg -o 3dogs_mask.png
```

The response will be a ".png" mask.

You can visualize the output as follows:

```
import matplotlib.pyplot as plt
import mmcv
plt.imshow(mmcv.imread("3dogs_mask.png", "grayscale"))
plt.show()
```

You should see something similar to:



And you can use `test_torchserve.py` to compare result of torchserve and pytorch, and visualize them.

```
python tools/torchserve/test_torchserve.py ${IMAGE_FILE} ${CONFIG_FILE} ${CHECKPOINT_
↪FILE} ${MODEL_NAME}
[--inference-addr ${INFERENCE_ADDR}] [--result-image ${RESULT_IMAGE}] [--device ${DEVICE}
↪]
```

Example:

```
python tools/torchserve/test_torchserve.py \
demo/demo.png \
configs/fcn/fcn_r50-d8_512x1024_40k_cityscapes.py \
checkpoint/fcn_r50-d8_512x1024_40k_cityscapes_20200604_192608-efe53f0d.pth \
fcn
```

CHAPTER

SEVENTEEN

CONFUSION MATRIX

In order to generate and plot a `nxn` confusion matrix where `n` is the number of classes, you can follow the steps:

17.1 1.Generate a prediction result in pkl format using `test.py`

```
python tools/test.py ${CONFIG_FILE} ${CHECKPOINT_FILE} [--out ${PATH_TO_RESULT_FILE}]
```

Note that the argument for `--eval` should be `None` so that the result file contains numpy type of prediction results. The usage for distribution test is just the same.

Example:

```
python tools/test.py \
configs/fcn/fcn_r50-d8_512x1024_40k_cityscapes.py \
checkpoint/fcn_r50-d8_512x1024_40k_cityscapes_20200604_192608-efe53f0d.pth \
--out result/pred_result.pkl
```

17.2 2. Use `confusion_matrix.py` to generate and plot a confusion matrix

```
python tools/confusion_matrix.py ${CONFIG_FILE} ${PATH_TO_RESULT_FILE} ${SAVE_DIR} --show
```

Description of arguments:

- `config`: Path to the test config file.
- `prediction_path`: Path to the prediction .pkl result.
- `save_dir`: Directory where confusion matrix will be saved.
- `--show`: Enable result visualize.
- `--color-theme`: Theme of the matrix color map.
- `--cfg_options`: Custom options to replace the config file.

Example:

```
python tools/confusion_matrix.py \
configs/fcn/fcn_r50-d8_512x1024_40k_cityscapes.py \
result/pred_result.pkl \
```

(continues on next page)

(continued from previous page)

```
result/confusion_matrix \
--show
```

CHAPTER

EIGHTEEN

CHANGELOG

18.1 V0.28.0 (9/8/2022)

New Features

- Support Tversky Loss (#1896)

Bug Fixes

- Fix binary segmentation (#2016)
- Fix config files (#1901, #1893, #1871)
- Revise documentation (#1844, #1980, #2025, #1982)
- Fix confusion matrix calculation (#1992)
- Fix decode head forward_train error (#1997)

Contributors

- @suchot made their first contribution in https://github.com/open-mmlab/mmsegmention/pull/1844
- @TimoK93 made their first contribution in https://github.com/open-mmlab/mmsegmentation/pull/1992

18.2 V0.27.0 (7/28/2022)

Enhancement

- Add Swin-L Transformer models (#1471)
- Update ERFNet results (#1744)

Bug Fixes

- Revise documentation (#1761, #1755, #1802)
- Fix colab tutorial (#1779)
- Fix segformer checkpoint url (#1785)

Contributors

- @DataSttructure made their first contribution in https://github.com/open-mmlab/mmsegmentation/pull/1802
- @AkideLiu made their first contribution in https://github.com/open-mmlab/mmsegmentation/pull/1785
- @mawanda-jun made their first contribution in https://github.com/open-mmlab/mmsegmentation/pull/1761
- @Yan-Daojiang made their first contribution in https://github.com/open-mmlab/mmsegmentation/pull/1755

18.3 V0.26.0 (7/1/2022)

Highlights

- Update New SegFormer models on ADE20K (1705)
- Dedicated MMSegWandbHook for MMSegmentation (1603)

New Features

- Update New SegFormer models on ADE20K (1705)
- Dedicated MMSegWandbHook for MMSegmentation (1603)
- Add UPerNet r18 results (1669)

Enhancement

- Keep dimension of `cls_token_weight` for easier ONNX deployment (1642)
- Support infererence with padding (1607)

Bug Fixes

- Fix typos (#1640, #1667, #1656, #1699, #1702, #1695, #1707, #1708, #1721)

Documentation

- Fix `mdformat` version to support python3.6 and remove ruby installation (1672)

Contributors

- @RunningLeon made their first contribution in https://github.com/open-mmlab/mmsegmentation/pull/1642
- @zhouzaida made their first contribution in https://github.com/open-mmlab/mmsegmentation/pull/1655
- @tkhe made their first contribution in https://github.com/open-mmlab/mmsegmentation/pull/1667
- @rotorliu made their first contribution in https://github.com/open-mmlab/mmsegmentation/pull/1656
- @EvelynWang-0423 made their first contribution in https://github.com/open-mmlab/mmsegmentation/pull/1679
- @ZhaoYi1222 made their first contribution in https://github.com/open-mmlab/mmsegmentation/pull/1616
- @Sanster made their first contribution in https://github.com/open-mmlab/mmsegmentation/pull/1704
- @ayulockin made their first contribution in https://github.com/open-mmlab/mmsegmentation/pull/1603

18.4 V0.25.0 (6/2/2022)

Highlights

- Support PyTorch backend on MLU (1515)

Bug Fixes

- Fix the error of BCE loss when batch size is 1 (1629)
- Fix bug of `resize` function when align_corners is True (1592)
- Fix Dockerfile to run demo script in docker container (1568)
- Correct inference_demo.ipynb path (1576)
- Fix the `build_segmentor` in colab demo (1551)

- Fix md2yml script (1633, 1555)
- Fix main line link in MAE README.md (1556)
- Fix fastfcn `crop_size` in README.md by (1597)
- Pip upgrade when testing windows platform (1610)

Improvements

- Delete DS_Store file (1549)
- Revise owners.yml (1621, 1534)

Documentation

- Rewrite the installation guidance (1630)
- Format readme (1635)
- Replace markdownlint with mdformat to avoid ruby installation (1591)
- Add explanation and usage instructions for data configuration (1548)
- Configure Myst-parser to parse anchor tag (1589)
- Update QR code and link for QQ group (1598, 1574)

Contributors

- @atinfinity made their first contribution in https://github.com/open-mmlab/mmsegmentation/pull/1568
- @DoubleChuang made their first contribution in https://github.com/open-mmlab/mmsegmentation/pull/1576
- @alpha-baymax made their first contribution in https://github.com/open-mmlab/mmsegmentation/pull/1515
- @274869388 made their first contribution in https://github.com/open-mmlab/mmsegmentation/pull/1629

18.5 V0.24.1 (5/1/2022)

Bug Fixes

- Fix `LayerDecayOptimizerConstructor` for MAE training (#1539, #1540)

18.6 V0.24.0 (4/29/2022)

Highlights

- Support MAE: Masked Autoencoders Are Scalable Vision Learners
- Support Resnet strikes back

New Features

- Support MAE: Masked Autoencoders Are Scalable Vision Learners (1307, 1523)
- Support Resnet strikes back (1390)
- Support extra dataloader settings in configs (1435)

Bug Fixes

- Fix input previous results for the last cascade_decode_head (#1450)
- Fix validation loss logging (#1494)

- Fix the bug in binary_cross_entropy (1527)
- Support single channel prediction for Binary Cross Entropy Loss (#1454)
- Fix potential bugs in accuracy.py (1496)
- Avoid converting label ids twice by label map during evaluation (1417)
- Fix bug about label_map (1445)
- Fix image save path bug in Windows (1423)
- Fix MMSegmentation Colab demo (1501, 1452)
- Migrate azure blob for beit checkpoints (1503)
- Fix bug in `tools/analyse_logs.py` caused by wrong plot_iter in some cases (1428)

Improvements

- Merge BEiT and ConvNext's LR decay optimizer constructors (#1438)
- Register optimizer constructor with mmseg (#1456)
- Refactor transformer encode layer in ViT and BEiT backbone (#1481)
- Add `build_pos_embed` and `build_layers` for BEiT (1517)
- Add `with_cp` to mit and vit (1431)
- Fix inconsistent dtype of `seg_label` in stdc decode (1463)
- Delete random seed for training in `dist_train.sh` (1519)
- Revise high `workers_per_gpus` in config file (#1506)
- Add GPG keys and del mmcv version in Dockerfile (1534)
- Update checkpoint for model in deeplabv3plus (#1487)
- Add `DistSamplerSeedHook` to set epoch number to dataloader when runner is `EpochBasedRunner` (1449)
- Provide URLs of Swin Transformer pretrained models (1389)
- Updating Dockerfiles From Docker Directory and `get_started.md` to reach latest stable version of Python, PyTorch and MMCV (1446)

Documentation

- Add more clearly statement of CPU training/inference (1518)

Contributors

- @jiangyitong made their first contribution in https://github.com/open-mmlab/mmsegmentation/pull/1431
- @kahkeng made their first contribution in https://github.com/open-mmlab/mmsegmentation/pull/1447
- @Nourollah made their first contribution in https://github.com/open-mmlab/mmsegmentation/pull/1446
- @androbaza made their first contribution in https://github.com/open-mmlab/mmsegmentation/pull/1452
- @Yzichen made their first contribution in https://github.com/open-mmlab/mmsegmentation/pull/1445
- @whu-pzhang made their first contribution in https://github.com/open-mmlab/mmsegmentation/pull/1423
- @panfeng-hover made their first contribution in https://github.com/open-mmlab/mmsegmentation/pull/1417
- @Johnson-Wang made their first contribution in https://github.com/open-mmlab/mmsegmentation/pull/1496
- @jere357 made their first contribution in https://github.com/open-mmlab/mmsegmentation/pull/1460

- @mfernezir made their first contribution in https://github.com/open-mmlab/mmsegmentation/pull/1494
- @donglixp made their first contribution in https://github.com/open-mmlab/mmsegmentation/pull/1503
- @YuanLiuuuuuu made their first contribution in https://github.com/open-mmlab/mmsegmentation/pull/1307
- @Dawn-bin made their first contribution in https://github.com/open-mmlab/mmsegmentation/pull/1527

18.7 V0.23.0 (4/1/2022)

Highlights

- Support BEiT: BERT Pre-Training of Image Transformers
- Support K-Net: Towards Unified Image Segmentation
- Add `avg_non_ignore` of CELoss to support average loss over non-ignored elements
- Support dataset initialization with file client

New Features

- Support BEiT: BERT Pre-Training of Image Transformers (#1404)
- Support K-Net: Towards Unified Image Segmentation (#1289)
- Support dataset initialization with file client (#1402)
- Add class name function for STARE datasets (#1376)
- Support different seeds on different ranks when distributed training (#1362)
- Add `nlc2nchw2nlc` and `nchw2nlc2nchw` to simplify tensor with different dimension operation (#1249)

Improvements

- Synchronize random seed for distributed sampler (#1411)
- Add script and documentation for multi-machine distributed training (#1383)

Bug Fixes

- Add `avg_non_ignore` of CELoss to support average loss over non-ignored elements (#1409)
- Fix some wrong URLs of models or logs in `./configs` (#1336)
- Add title and color theme arguments to plot function in `tools/confusion_matrix.py` (#1401)
- Fix outdated link in Colab demo (#1392)
- Fix typos (#1424, #1405, #1371, #1366, #1363)

Documentation

- Add FAQ document (#1420)
- Fix the config name style description in official docs(#1414)

Contributors

- @kinglintianxia made their first contribution in https://github.com/open-mmlab/mmsegmentation/pull/1371
- @CCODING04 made their first contribution in https://github.com/open-mmlab/mmsegmentation/pull/1376
- @mob5566 made their first contribution in https://github.com/open-mmlab/mmsegmentation/pull/1401
- @xiongnemo made their first contribution in https://github.com/open-mmlab/mmsegmentation/pull/1392

- @Xiangxu-0103 made their first contribution in https://github.com/open-mmlab/mmsegmentation/pull/1405

18.8 V0.22.1 (3/9/2022)

Bug Fixes

- Fix the ZeroDivisionError that all pixels in one image is ignored. (#1336)

Improvements

- Provide URLs of STDC, Segmenter and Twins pretrained models (#1272)

18.9 V0.22 (3/04/2022)

Highlights

- Support ConvNeXt: A ConvNet for the 2020s. Please use the latest MMClassification (0.21.0) to try it out.
- Support iSAID aerial Dataset.
- Officially Support inference on Windows OS.

New Features

- Support ConvNeXt: A ConvNet for the 2020s. (#1216)
- Support iSAID aerial Dataset. (#1115
- Generating and plotting confusion matrix. (#1301)

Improvements

- Refactor 4 decoder heads (ASPP, FCN, PSP, UPer): Split forward function into `_forward_feature` and `cls_seg`. (#1299)
- Add `min_size` arg in `Resize` to keep the shape after resize bigger than slide window. (#1318)
- Revise pre-commit-hooks. (#1315)
- Add win-ci. (#1296)

Bug Fixes

- Fix `mlp_ratio` type in Swin Transformer. (#1274)
- Fix path errors in `./demo` . (#1269)
- Fix bug in conversion of potsdam. (#1279)
- Make accuracy take into account `ignore_index`. (#1259)
- Add Pytorch HardSwish assertion in unit test. (#1294)
- Fix wrong palette value in vaihingen. (#1292)
- Fix the bug that SETR cannot load pretrain. (#1293)
- Update correct `In Collection` in metafile of each configs. (#1239)
- Upload completed STDC models. (#1332)
- Fix `DNLHead` exports onnx inference difference type Cast error. (#1161)

Contributors

- @JiaYanhao made their first contribution in https://github.com/open-mmlab/mmsegmentation/pull/1269
- @andife made their first contribution in https://github.com/open-mmlab/mmsegmentation/pull/1281
- @SBCV made their first contribution in https://github.com/open-mmlab/mmsegmentation/pull/1279
- @HJoonKwon made their first contribution in https://github.com/open-mmlab/mmsegmentation/pull/1259
- @Tsingularity made their first contribution in https://github.com/open-mmlab/mmsegmentation/pull/1290
- @Waterman0524 made their first contribution in https://github.com/open-mmlab/mmsegmentation/pull/1115
- @MeowZheng made their first contribution in https://github.com/open-mmlab/mmsegmentation/pull/1315
- @linfangjian01 made their first contribution in https://github.com/open-mmlab/mmsegmentation/pull/1318

18.10 V0.21.1 (2/9/2022)

Bug Fixes

- Fix typos in docs. (#1263)
- Fix repeating log by `setup_multi_processes`. (#1267)
- Upgrade isort in pre-commit hook. (#1270)

Improvements

- Use MMCV load_state_dict func in ViT/Swin. (#1272)
- Add exception for PointRend for support CPU-only. (#1271)

18.11 V0.21 (1/29/2022)

Highlights

- Officially Support CPUs training and inference, please use the latest MMCV (1.4.4) to try it out.
- Support Segmenter: Transformer for Semantic Segmentation (ICCV'2021).
- Support ISPRS Potsdam and Vaihingen Dataset.
- Add Mosaic transform and `MultiImageMixDataset` class in `dataset_wrappers`.

New Features

- Support Segmenter: Transformer for Semantic Segmentation (ICCV'2021) (#955)
- Support ISPRS Potsdam and Vaihingen Dataset (#1097, #1171)
- Add segformer's benchmark on cityscapes (#1155)
- Add auto resume (#1172)
- Add Mosaic transform and `MultiImageMixDataset` class in `dataset_wrappers` (#1093, #1105)
- Add log collector (#1175)

Improvements

- New-style CPU training and inference (#1251)
- Add UNet benchmark with multiple losses supervision (#1143)

Bug Fixes

- Fix the model statistics in doc for readthedoc (#1153)
- Set random seed for `palette` if not given (#1152)
- Add `COCOStuffDataset` in `class_names.py` (#1222)
- Fix bug in non-distributed multi-gpu training/testing (#1247)
- Delete unnecessary lines of STDCHead (#1231)

Contributors

- @jbwang1997 made their first contribution in https://github.com/open-mmlab/mmsegmentation/pull/1152
- @BeaverCC made their first contribution in https://github.com/open-mmlab/mmsegmentation/pull/1206
- @Echo-minn made their first contribution in https://github.com/open-mmlab/mmsegmentation/pull/1214
- @rstrudel made their first contribution in https://github.com/open-mmlab/mmsegmentation/pull/955

18.12 V0.20.2 (12/15/2021)

Bug Fixes

- Revise –option to –options to avoid BC-breaking. (#1140)

18.13 V0.20.1 (12/14/2021)

Improvements

- Change options to cfg-options (#1129)

Bug Fixes

- Fix `<!-- [ABSTRACT] -->` in metafile. (#1127)
- Fix correct `num_classes` of HRNet in `LoveDA` dataset (#1136)

18.14 V0.20 (12/10/2021)

Highlights

- Support Twins (#989)
- Support a real-time segmentation model STDC (#995)
- Support a widely-used segmentation model in lane detection ERFNet (#960)
- Support A Remote Sensing Land-Cover Dataset LoveDA (#1028)
- Support focal loss (#1024)

New Features

- Support Twins (#989)
- Support a real-time segmentation model STDC (#995)
- Support a widely-used segmentation model in lane detection ERFNet (#960)
- Add SETR cityscapes benchmark (#1087)

- Add BiSeNetV1 COCO-Stuff 164k benchmark (#1019)
- Support focal loss (#1024)
- Add Cutout transform (#1022)

Improvements

- Set a random seed when the user does not set a seed (#1039)
- Add CircleCI setup (#1086)
- Skip CI on ignoring given paths (#1078)
- Add abstract and image for every paper (#1060)
- Create a symbolic link on windows (#1090)
- Support video demo using trained model (#1014)

Bug Fixes

- Fix incorrectly loading init_cfg or pretrained models of several transformer models (#999, #1069, #1102)
- Fix EfficientMultiheadAttention in SegFormer (#1037)
- Remove `fp16` folder in `configs` (#1031)
- Fix several typos in .yml file (Dice Metric #1041, ADE20K dataset #1120, Training Memory (GB) #1083)
- Fix test error when using `--show-dir` (#1091)
- Fix dist training infinite waiting issue (#1035)
- Change the upper version of mmcv to 1.5.0 (#1096)
- Fix symlink failure on Windows (#1038)
- Cancel previous runs that are not completed (#1118)
- Unified links of readthedocs in docs (#1119)

Contributors

- @Junjue-Wang made their first contribution in https://github.com/open-mmlab/mmsegmentation/pull/1028
- @ddebby made their first contribution in https://github.com/open-mmlab/mmsegmentation/pull/1066
- @del-zhenwu made their first contribution in https://github.com/open-mmlab/mmsegmentation/pull/1078
- @KangBK0120 made their first contribution in https://github.com/open-mmlab/mmsegmentation/pull/1106
- @zergzzlun made their first contribution in https://github.com/open-mmlab/mmsegmentation/pull/1091
- @fingertap made their first contribution in https://github.com/open-mmlab/mmsegmentation/pull/1035
- @irvingzhang0512 made their first contribution in https://github.com/open-mmlab/mmsegmentation/pull/1014
- @littleSunlxy made their first contribution in https://github.com/open-mmlab/mmsegmentation/pull/989
- @lkm2835
- @RockeyCoss
- @MengzhangLI
- @Junjun2016
- @xiexinch
- @xvjiarui

18.15 V0.19 (11/02/2021)

Highlights

- Support TIMMBackbone wrapper (#998)
- Support custom hook (#428)
- Add codespell pre-commit hook (#920)
- Add FastFCN benchmark on ADE20K (#972)

New Features

- Support TIMMBackbone wrapper (#998)
- Support custom hook (#428)
- Add FastFCN benchmark on ADE20K (#972)
- Add codespell pre-commit hook and fix typos (#920)

Improvements

- Make inputs & channels smaller in unittests (#1004)
- Change `self.loss_decode` back to `dict` in Single Loss situation (#1002)

Bug Fixes

- Fix typo in usage example (#1003)
- Add contiguous after permutation in ViT (#992)
- Fix the invalid link (#985)
- Fix bug in CI with python 3.9 (#994)
- Fix bug when loading class name form file in custom dataset (#923)

Contributors

- @ShoupingShan made their first contribution in https://github.com/open-mmlab/mmsegmentation/pull/923
- @RockeyCoss made their first contribution in https://github.com/open-mmlab/mmsegmentation/pull/954
- @HarborYuan made their first contribution in https://github.com/open-mmlab/mmsegmentation/pull/992
- @lkm2835 made their first contribution in https://github.com/open-mmlab/mmsegmentation/pull/1003
- @gszh made their first contribution in https://github.com/open-mmlab/mmsegmentation/pull/428
- @VVssssk
- @MengzhangLI
- @Junjun2016

18.16 V0.18 (10/07/2021)

Highlights

- Support three real-time segmentation models (ICNet #884, BiSeNetV1 #851, and BiSeNetV2 #804)
- Support one efficient segmentation model (FastFCN #885)
- Support one efficient non-local/self-attention based segmentation model (ISANet #70)
- Support COCO-Stuff 10k and 164k datasets (#625)
- Support evaluate concated dataset separately (#833)
- Support loading GT for evaluation from multi-file backend (#867)

New Features

- Support three real-time segmentation models (ICNet #884, BiSeNetV1 #851, and BiSeNetV2 #804)
- Support one efficient segmentation model (FastFCN #885)
- Support one efficient non-local/self-attention based segmentation model (ISANet #70)
- Support COCO-Stuff 10k and 164k datasets (#625)
- Support evaluate concated dataset separately (#833)

Improvements

- Support loading GT for evaluation from multi-file backend (#867)
- Auto-convert SyncBN to BN when training on DP automatly(#772)
- Refactor Swin-Transformer (#800)

Bug Fixes

- Update mmcv installation in dockerfile (#860)
- Fix number of iteration bug when resuming checkpoint in distributed train (#866)
- Fix parsing parse in val_step (#906)

18.17 V0.17 (09/01/2021)

Highlights

- Support SegFormer
- Support DPT
- Support Dark Zurich and Nighttime Driving datasets
- Support progressive evaluation

New Features

- Support SegFormer (#599)
- Support DPT (#605)
- Support Dark Zurich and Nighttime Driving datasets (#815)
- Support progressive evaluation (#709)

Improvements

- Add multiscale_output interface and unittests for HRNet (#830)
- Support inherit cityscapes dataset (#750)
- Fix some typos in README.md (#824)
- Delete convert function and add instruction to ViT/Swin README.md (#791)
- Add vit/swin/mit convert weight scripts (#783)
- Add copyright files (#796)

Bug Fixes

- Fix invalid checkpoint link in inference_demo.ipynb (#814)
- Ensure that items in dataset have the same order across multi machine (#780)
- Fix the log error (#766)

18.18 V0.16 (08/04/2021)

Highlights

- Support PyTorch 1.9
- Support SegFormer backbone MiT
- Support md2yml pre-commit hook
- Support frozen stage for HRNet

New Features

- Support SegFormer backbone MiT (#594)
- Support md2yml pre-commit hook (#732)
- Support mim (#717)
- Add mmseg2torchserve tool (#552)

Improvements

- Support hrnet frozen stage (#743)
- Add template of reimplementation questions (#741)
- Output pdf and epub formats for readthedocs (#742)
- Refine the docstring of ResNet (#723)
- Replace interpolate with resize (#731)
- Update resource limit (#700)
- Update config.md (#678)

Bug Fixes

- Fix ATTENTION registry (#729)
- Fix analyze log script (#716)
- Fix doc api display (#725)
- Fix patch_embed and pos_embed mismatch error (#685)

- Fix efficient test for multi-node (#707)
- Fix init_cfg in resnet backbone (#697)
- Fix efficient test bug (#702)
- Fix url error in config docs (#680)
- Fix mmcv installation (#676)
- Fix torch version (#670)

Contributors

@sshuair @xiexinch @Junjun2016 @mmeendez8 @xvjiarui @sennnnn @puhsu @BIGWangYuDong @keke1u @daavoo

18.19 V0.15 (07/04/2021)

Highlights

- Support ViT, SETR, and Swin-Transformer
- Add Chinese documentation
- Unified parameter initialization

Bug Fixes

- Fix typo and links (#608)
- Fix Dockerfile (#607)
- Fix ViT init (#609)
- Fix mmcv version compatible table (#658)
- Fix model links of DMNEt (#660)

New Features

- Support loading DeiT weights (#538)
- Support SETR (#531, #635)
- Add config and models for ViT backbone with UperHead (#520, #635)
- Support Swin-Transformer (#511)
- Add higher accuracy FastSCNN (#606)
- Add Chinese documentation (#666)

Improvements

- Unified parameter initialization (#567)
- Separate CUDA and CPU in github action CI (#602)
- Support persistent dataloader worker (#646)
- Update meta file fields (#661, #664)

18.20 V0.14 (06/02/2021)

Highlights

- Support ONNX to TensorRT
- Support MIM

Bug Fixes

- Fix ONNX to TensorRT verify (#547)
- Fix save best for EvalHook (#575)

New Features

- Support loading DeiT weights (#538)
- Support ONNX to TensorRT (#542)
- Support output results for ADE20k (#544)
- Support MIM (#549)

Improvements

- Add option for ViT output shape (#530)
- Infer batch size using len(result) (#532)
- Add compatible table between MMSeg and MMCV (#558)

18.21 V0.13 (05/05/2021)

Highlights

- Support Pascal Context Class-59 dataset.
- Support Visual Transformer Backbone.
- Support mFscore metric.

Bug Fixes

- Fixed Colaboratory tutorial (#451)
- Fixed mIoU calculation range (#471)
- Fixed sem_fpn, unet README.md (#492)
- Fixed `num_classes` in FCN for Pascal Context 60-class dataset (#488)
- Fixed FP16 inference (#497)

New Features

- Support dynamic export and visualize to pytorch2onnx (#463)
- Support export to torchscript (#469, #499)
- Support Pascal Context Class-59 dataset (#459)
- Support Visual Transformer backbone (#465)
- Support UpSample Neck (#512)
- Support mFscore metric (#509)

Improvements

- Add more CI for PyTorch (#460)
- Add print model graph args for tools/print_config.py (#451)
- Add cfg links in modelzoo README.md (#468)
- Add BaseSegmentor import to segmentors/**init**.py (#495)
- Add MMOCR, MMGeneration links (#501, #506)
- Add Chinese QR code (#506)
- Use MMCV MODEL_REGISTRY (#515)
- Add ONNX testing tools (#498)
- Replace data_dict calling 'img' key to support MMDet3D (#514)
- Support reading class_weight from file in loss function (#513)
- Make tags as comment (#505)
- Use MMCV EvalHook (#438)

18.22 V0.12 (04/03/2021)

Highlights

- Support FCN-Dilate 6 model.
- Support Dice Loss.

Bug Fixes

- Fixed PhotoMetricDistortion Doc (#388)
- Fixed install scripts (#399)
- Fixed Dice Loss multi-class (#417)

New Features

- Support Dice Loss (#396)
- Add plot logs tool (#426)
- Add opacity option to show_result (#425)
- Speed up mIoU metric (#430)

Improvements

- Refactor unittest file structure (#440)
- Fix typos in the repo (#449)
- Include class-level metrics in the log (#445)

18.23 V0.11 (02/02/2021)

Highlights

- Support memory efficient test, add more UNet models.

Bug Fixes

- Fixed TTA resize scale (#334)
- Fixed CI for pip 20.3 (#307)
- Fixed ADE20k test (#359)

New Features

- Support memory efficient test (#330)
- Add more UNet benchmarks (#324)
- Support Lovasz Loss (#351)

Improvements

- Move train_cfg/test_cfg inside model (#341)

18.24 V0.10 (01/01/2021)

Highlights

- Support MobileNetV3, DMNet, APCNet. Add models of ResNet18V1b, ResNet18V1c, ResNet50V1b.

Bug Fixes

- Fixed CPU TTA (#276)
- Fixed CI for pip 20.3 (#307)

New Features

- Add ResNet18V1b, ResNet18V1c, ResNet50V1b, ResNet101V1b models (#316)
- Support MobileNetV3 (#268)
- Add 4 retinal vessel segmentation benchmark (#315)
- Support DMNet (#313)
- Support APCNet (#299)

Improvements

- Refactor Documentation page (#311)
- Support resize data augmentation according to original image size (#291)

18.25 V0.9 (30/11/2020)

Highlights

- Support 4 medical dataset, UNet and CGNet.

New Features

- Support RandomRotate transform (#215, #260)
- Support RGB2Gray transform (#227)
- Support Rerange transform (#228)
- Support ignore_index for BCE loss (#210)
- Add modelzoo statistics (#263)
- Support Dice evaluation metric (#225)
- Support Adjust Gamma transform (#232)
- Support CLAHE transform (#229)

Bug Fixes

- Fixed detail API link (#267)

18.26 V0.8 (03/11/2020)

Highlights

- Support 4 medical dataset, UNet and CGNet.

New Features

- Support customize runner (#118)
- Support UNet (#161)
- Support CHASE_DB1, DRIVE, STARE, HRD (#203)
- Support CGNet (#223)

18.27 V0.7 (07/10/2020)

Highlights

- Support Pascal Context dataset and customizing class dataset.

Bug Fixes

- Fixed CPU inference (#153)

New Features

- Add DeepLab OS16 models (#154)
- Support Pascal Context dataset (#133)
- Support customizing dataset classes (#71)
- Support customizing dataset palette (#157)

Improvements

- Support 4D tensor output in ONNX (#150)
- Remove redundancies in ONNX export (#160)
- Migrate to MMCV DepthwiseSeparableConv (#158)
- Migrate to MMCV collect_env (#137)
- Use img_prefix and seg_prefix for loading (#153)

18.28 V0.6 (10/09/2020)

Highlights

- Support new methods i.e. MobileNetV2, EMANet, DNL, PointRend, Semantic FPN, Fast-SCNN, ResNeSt.

Bug Fixes

- Fixed sliding inference ONNX export (#90)

New Features

- Support MobileNet v2 (#86)
- Support EMANet (#34)
- Support DNL (#37)
- Support PointRend (#109)
- Support Semantic FPN (#94)
- Support Fast-SCNN (#58)
- Support ResNeSt backbone (#47)
- Support ONNX export (experimental) (#12)

Improvements

- Support Upsample in ONNX (#100)
- Support Windows install (experimental) (#75)
- Add more OCRNet results (#20)
- Add PyTorch 1.6 CI (#64)
- Get version and githash automatically (#55)

18.29 v0.5.1 (11/08/2020)

Highlights

- Support FP16 and more generalized OHEM

Bug Fixes

- Fixed Pascal VOC conversion script (#19)
- Fixed OHEM weight assign bug (#54)
- Fixed palette type when palette is not given (#27)

New Features

- Support FP16 (#21)
- Generalized OHEM (#54)

Improvements

- Add load-from flag (#33)
- Fixed training tricks doc about different learning rates of model (#26)

CHAPTER

NINETEEN

FREQUENTLY ASKED QUESTIONS (FAQ)

We list some common troubles faced by many users and their corresponding solutions here. Feel free to enrich the list if you find any frequent issues and have ways to help others to solve them. If the contents here do not cover your issue, please create an issue using the provided templates and make sure you fill in all required information in the template.

19.1 Installation

The compatible MMSegmentation and MMCV versions are as below. Please install the correct version of MMCV to avoid installation issues.

You need to run `pip uninstall mmcv` first if you have mmcv installed. If mmcv and mmcv-full are both installed, there will be `ModuleNotFoundError`.

- "No module named 'mmcv.ops'"; "No module named 'mmcv._ext'".
  1. Uninstall existing mmcv in the environment using `pip uninstall mmcv`.
  2. Install mmcv-full following the installation instruction.

19.2 How to know the number of GPUs needed to train the model

- Infer from the name of the config file of the model. You can refer to the `Config Name Style` part of Learn about Configs. For example, for config file with name `segformer_mit-b0_8x1_1024x1024_160k_cityscapes.py`, `8x1` means training the model corresponding to it needs 8 GPUs, and the batch size of each GPU is 1.
- Infer from the log file. Open the log file of the model and search `nGPU` in the file. The number of figures following `nGPU` is the number of GPUs needed to train the model. For instance, searching for `nGPU` in the log file yields the record `nGPU 0,1,2,3,4,5,6,7`, which indicates that eight GPUs are needed to train the model.

19.3 What does the auxiliary head mean

Briefly, it is a deep supervision trick to improve the accuracy. In the training phase, `decode_head` is for decoding semantic segmentation output, `auxiliary_head` is just adding an auxiliary loss, the segmentation result produced by it has no impact to your model's result, it just works in training. You may read this paper for more information.

19.4 Why is the log file not created

In the train script, we call `get_root_logger`at Line 167, and `get_root_logger` in mmseg calls `get_logger` in mmcv, mmcv will return the same logger which has beed initialized in 'mmsegmentation/tools/train.py' with the parameter `log_file`. There is only one logger (initialized with `log_file`) during training. Ref: https://github.com/open-mmlab/mmcv/blob/21bada32560c7ed7b15b017dc763d862789e29a8/mmcv/utils/logging.py#L9-L16

If you find the log file not been created, you might check if `mmcv.utils.get_logger` is called elsewhere.

19.5 How to output the image for painting the segmentation mask when running the test script

In the test script, we provide `show-dir` argument to control whether output the painted images. Users might run the following command:

```
python tools/test.py {config} {checkpoint} --show-dir {/path/to/save/image} --opacity 1
```

CHAPTER

TWENTY

ENGLISH

CHAPTER

TWENTYONE

CHAPTER

TWENTYTWO

MMSEG.APIS

mmseg.apis.**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., "mmseg".

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

mmseg.apis.**inference_segmentor**(*model*, *imgs*)
Inference image(s) with the segmentor.

Parameters

- **model** (*nn.Module*) – The loaded segmentor.
- **imgs** (*str/ndarray or list[str/ndarray]*) – Either image files or loaded images.

Returns The segmentation result.

Return type (list[Tensor])

mmseg.apis.**init_random_seed**(*seed=None*, *device='cuda'*)
Initialize random seed.

If the seed is not set, the seed will be automatically randomized, and then broadcast to all processes to prevent some potential bugs. :param seed: The seed. Default to None. :type seed: int, Optional :param device: The device where the seed will be put on.

Default to 'cuda'.

Returns Seed to be used.

Return type int

mmseg.apis.**init_segmentor**(*config*, *checkpoint=None*, *device='cuda:0'*)
Initialize a segmentor from config file.

Parameters

- **config** (str or mmcv.Config) – Config file path or the config object.
- **checkpoint** (*str, optional*) – Checkpoint path. If left as None, the model will not load any weights.
- **device** (*str, optional*) – 0'. Use 'cpu' for loading model on CPU.

Returns The constructed segmentor.

Return type nn.Module

mmseg.apis.**multi_gpu_test**(*model*, *data_loader*, *tmpdir=None*, *gpu_collect=False*, *efficient_test=False*, *pre_eval=False*, *format_only=False*, *format_args={}*)

Test model with multiple gpus by progressive mode.

This method tests model with multiple gpus and collects the results under two different modes: gpu and cpu modes. By setting 'gpu_collect=True' it encodes results to gpu tensors and use gpu communication for results collection. On cpu mode it saves the results on different gpus to 'tmpdir' and collects them by the rank 0 worker.

Parameters

- **model** (*nn.Module*) – Model to be tested.
- **data_loader** (*utils.data.Dataloader*) – Pytorch data loader.
- **tmpdir** (*str*) – Path of directory to save the temporary results from different gpus under cpu mode. The same path is used for efficient test. Default: None.
- **gpu_collect** (*bool*) – Option to use either gpu or cpu to collect results. Default: False.
- **efficient_test** (*bool*) – Whether save the results as local numpy files to save CPU memory during evaluation. Mutually exclusive with pre_eval and format_results. Default: False.
- **pre_eval** (*bool*) – Use dataset.pre_eval() function to generate pre_results for metric evaluation. Mutually exclusive with efficient_test and format_results. Default: False.
- **format_only** (*bool*) – Only format result for results commit. Mutually exclusive with pre_eval and efficient_test. Default: False.
- **format_args** (*dict*) – The args for format_results. Default: {}.

Returns list of evaluation pre-results or list of save file names.

Return type list

mmseg.apis.**set_random_seed**(*seed*, *deterministic=False*)

Set random seed.

Parameters

- **seed** (*int*) – Seed to be used.
- **deterministic** (*bool*) – Whether to set the deterministic option for CUDNN backend, i.e., set *torch.backends.cudnn.deterministic* to True and *torch.backends.cudnn.benchmark* to False. Default: False.

mmseg.apis.**show_result_pyplot**(*model*, *img*, *result*, *palette=None*, *fig_size=(15, 10)*, *opacity=0.5*, *title=''*, *block=True*, *out_file=None*)

Visualize the segmentation results on the image.

Parameters

- **model** (*nn.Module*) – The loaded segmentor.
- **img** (*str or np.ndarray*) – Image filename or loaded image.
- **result** (*list*) – The segmentation result.

- **palette** (*list[list[int]] | None*) – The palette of segmentation map. If None is given, random palette will be generated. Default: None
- **fig_size** (*tuple*) – Figure size of the pyplot figure.
- **opacity** (*float*) – Opacity of painted segmentation map. Default 0.5. Must be in (0, 1] range.
- **title** (*str*) – The title of pyplot figure. Default is ''.
- **block** (*bool*) – Whether to block the pyplot figure. Default is True.
- **out_file** (*str or None*) – The path to write the image. Default: None.

mmseg.apis.**single_gpu_test**(*model*, *data_loader*, *show=False*, *out_dir=None*, *efficient_test=False*, *opacity=0.5*, *pre_eval=False*, *format_only=False*, *format_args={}*)

Test with single GPU by progressive mode.

Parameters

- **model** (*nn.Module*) – Model to be tested.
- **data_loader** (*utils.data.Dataloader*) – Pytorch data loader.
- **show** (*bool*) – Whether show results during inference. Default: False.
- **out_dir** (*str, optional*) – If specified, the results will be dumped into the directory to save output results.
- **efficient_test** (*bool*) – Whether save the results as local numpy files to save CPU memory during evaluation. Mutually exclusive with pre_eval and format_results. Default: False.
- **opacity** (*float*) – Opacity of painted segmentation map. Default 0.5. Must be in (0, 1] range.
- **pre_eval** (*bool*) – Use dataset.pre_eval() function to generate pre_results for metric evaluation. Mutually exclusive with efficient_test and format_results. Default: False.
- **format_only** (*bool*) – Only format result for results commit. Mutually exclusive with pre_eval and efficient_test. Default: False.
- **format_args** (*dict*) – The args for format_results. Default: {}.

Returns list of evaluation pre-results or list of save file names.

Return type list

mmseg.apis.**train_segmentor**(*model*, *dataset*, *cfg*, *distributed=False*, *validate=False*, *timestamp=None*, *meta=None*)

Launch segmentor training.

CHAPTER

TWENTYTHREE

MMSEG.CORE

23.1 seg

class mmseg.core.seg.**BasePixelSampler**(***kwargs*)
Base class of pixel sampler.

abstract sample(*seg_logit*, *seg_label*)
Placeholder for sample function.

class mmseg.core.seg.**OHEMPixelSampler**(*context*, *thresh=None*, *min_kept=100000*)
Online Hard Example Mining Sampler for segmentation.

Parameters

- **context** (*nn.Module*) – The context of sampler, subclass of BaseDecodeHead.
- **thresh** (*float, optional*) – The threshold for hard example selection. Below which, are prediction with low confidence. If not specified, the hard examples will be pixels of top min_kept loss. Default: None.
- **min_kept** (*int, optional*) – The minimum number of predictions to keep. Default: 100000.

sample(*seg_logit*, *seg_label*)
Sample pixels that have high loss or with low prediction confidence.

Parameters

- **seg_logit** (*torch.Tensor*) – segmentation logits, shape (N, C, H, W)
- **seg_label** (*torch.Tensor*) – segmentation label, shape (N, 1, H, W)

Returns segmentation weight, shape (N, H, W)

Return type torch.Tensor

mmseg.core.seg.**build_pixel_sampler**(*cfg*, ***default_args*)
Build pixel sampler for segmentation map.

23.2 evaluation

class mmseg.core.evaluation.**DistEvalHook**(**args*, *by_epoch=False*, *efficient_test=False*, *pre_eval=False*, ***kwargs*)

Distributed EvalHook, with efficient test support.

Parameters

- **by_epoch** (*bool*) – Determine perform evaluation by epoch or by iteration. If set to True, it will perform by epoch. Otherwise, by iteration. Default: False.
- **efficient_test** (*bool*) – Whether save the results as local numpy files to save CPU memory during evaluation. Default: False.
- **pre_eval** (*bool*) – Whether to use progressive mode to evaluate model. Default: False.

Returns The prediction results.

Return type list

class mmseg.core.evaluation.**EvalHook**(**args*, *by_epoch=False*, *efficient_test=False*, *pre_eval=False*, ***kwargs*)

Single GPU EvalHook, with efficient test support.

Parameters

- **by_epoch** (*bool*) – Determine perform evaluation by epoch or by iteration. If set to True, it will perform by epoch. Otherwise, by iteration. Default: False.
- **efficient_test** (*bool*) – Whether save the results as local numpy files to save CPU memory during evaluation. Default: False.
- **pre_eval** (*bool*) – Whether to use progressive mode to evaluate model. Default: False.

Returns The prediction results.

Return type list

mmseg.core.evaluation.**eval_metrics**(*results*, *gt_seg_maps*, *num_classes*, *ignore_index*, *metrics=['mIoU']*, *nan_to_num=None*, *label_map={}*, *reduce_zero_label=False*, *beta=1*)

Calculate evaluation metrics :param results: List of prediction segmentation

maps or list of prediction result filenames.

Parameters

- **gt_seg_maps** (*list[ndarray] | list[str] | Iterables*) – list of ground truth segmentation maps or list of label filenames.
- **num_classes** (*int*) – Number of categories.
- **ignore_index** (*int*) – Index that will be ignored in evaluation.
- **metrics** (*list[str] | str*) – Metrics to be evaluated, 'mIoU' and 'mDice'.
- **nan_to_num** (*int, optional*) – If specified, NaN values will be replaced by the numbers defined by the user. Default: None.
- **label_map** (*dict*) – Mapping old labels to new labels. Default: dict().
- **reduce_zero_label** – Whether ignore zero label. Default: False.

mmseg.core.evaluation.**get_classes**(*dataset*)

Get class names of a dataset.

mmseg.core.evaluation.**get_palette**(*dataset*)
Get class palette (RGB) of a dataset.

mmseg.core.evaluation.**intersect_and_union**(*pred_label*, *label*, *num_classes*, *ignore_index*, *label_map={}*, *reduce_zero_label=False*)
Calculate intersection and Union.

Parameters

- **pred_label** (*ndarray | str*) – Prediction segmentation map or predict result filename.
- **label** (*ndarray | str*) – Ground truth segmentation map or label filename.
- **num_classes** (*int*) – Number of categories.
- **ignore_index** (*int*) – Index that will be ignored in evaluation.
- **label_map** (*dict*) – Mapping old labels to new labels. The parameter will work only when label is str. Default: dict().
- **reduce_zero_label** – Whether ignore zero label. The parameter will work only when label is str. Default: False.

mmseg.core.evaluation.**mean_dice**(*results*, *gt_seg_maps*, *num_classes*, *ignore_index*, *nan_to_num=None*, *label_map={}*, *reduce_zero_label=False*)
Calculate Mean Dice (mDice)

Parameters

- **results** (*list[ndarray] | list[str]*) – List of prediction segmentation maps or list of prediction result filenames.
- **gt_seg_maps** (*list[ndarray] | list[str]*) – list of ground truth segmentation maps or list of label filenames.
- **num_classes** (*int*) – Number of categories.
- **ignore_index** (*int*) – Index that will be ignored in evaluation.
- **nan_to_num** (*int, optional*) – If specified, NaN values will be replaced by the numbers defined by the user. Default: None.
- **label_map** (*dict*) – Mapping old labels to new labels. Default: dict().
- **reduce_zero_label** – Whether ignore zero label. Default: False.

mmseg.core.evaluation.**mean_fscore**(*results*, *gt_seg_maps*, *num_classes*, *ignore_index*, *nan_to_num=None*, *label_map={}*, *reduce_zero_label=False*, *beta=1*)
Calculate Mean F-Score (mFscore)

Parameters

- **results** (*list[ndarray] | list[str]*) – List of prediction segmentation maps or list of prediction result filenames.
- **gt_seg_maps** (*list[ndarray] | list[str]*) – list of ground truth segmentation maps or list of label filenames.
- **num_classes** (*int*) – Number of categories.
- **ignore_index** (*int*) – Index that will be ignored in evaluation.
- **nan_to_num** (*int, optional*) – If specified, NaN values will be replaced by the numbers defined by the user. Default: None.
- **label_map** (*dict*) – Mapping old labels to new labels. Default: dict().

- **reduce_zero_label** (*bool*) – Whether ignore zero label. Default: False.
- **beta** – Determines the weight of recall in the combined score. Default: False.

mmseg.core.evaluation.**mean_iou**(*results*, *gt_seg_maps*, *num_classes*, *ignore_index*, *nan_to_num=None*, *label_map={}*, *reduce_zero_label=False*)

Calculate Mean Intersection and Union (mIoU)

Parameters

- **results** (*list[ndarray] | list[str]*) – List of prediction segmentation maps or list of prediction result filenames.
- **gt_seg_maps** (*list[ndarray] | list[str]*) – list of ground truth segmentation maps or list of label filenames.
- **num_classes** (*int*) – Number of categories.
- **ignore_index** (*int*) – Index that will be ignored in evaluation.
- **nan_to_num** (*int, optional*) – If specified, NaN values will be replaced by the numbers defined by the user. Default: None.
- **label_map** (*dict*) – Mapping old labels to new labels. Default: dict().
- **reduce_zero_label** – Whether ignore zero label. Default: False.

mmseg.core.evaluation.**pre_eval_to_metrics**(*pre_eval_results*, *metrics=['mIoU']*, *nan_to_num=None*, *beta=1*)

Convert pre-eval results to metrics.

Parameters

- **pre_eval_results** (*list[tuple[torch.Tensor]]*) – per image eval results for computing evaluation metric
- **metrics** (*list[str] | str*) – Metrics to be evaluated, 'mIoU' and 'mDice'.
- **nan_to_num** – If specified, NaN values will be replaced by the numbers defined by the user. Default: None.

23.3 utils

mmseg.core.utils.**add_prefix**(*inputs*, *prefix*)

Add prefix for dict.

Parameters

- **inputs** (*dict*) – The input dict with str keys.
- **prefix** (*str*) – The prefix to add.

Returns The dict with keys updated with `prefix`.

Return type dict

mmseg.core.utils.**sync_random_seed**(*seed=None*, *device='cuda'*)

Make sure different ranks share the same seed. All workers must call this function, otherwise it will deadlock. This method is generally used in *DistributedSampler*, because the seed should be identical across all processes in the distributed group.

In distributed sampling, different ranks should sample non-overlapped data in the dataset. Therefore, this function is used to make sure that each rank shuffles the data indices in the same order based on the same seed. Then different ranks could use different indices to select non-overlapped data from the same data list.

Parameters

- **seed** (*int, Optional*) – The seed. Default to None.
- **device** (*str*) – The device where the seed will be put on. Default to 'cuda'.

Returns Seed to be used.

Return type int

CHAPTER

TWENTYFOUR

MMSEG.DATASETS

24.1 datasets

class mmseg.datasets.**ADE20KDataset**(***kwargs*)

ADE20K dataset.

In segmentation map annotation for ADE20K, 0 stands for background, which is not included in 150 categories. `reduce_zero_label` is fixed to True. The `img_suffix` is fixed to '.jpg' and `seg_map_suffix` is fixed to '.png'.

format_results(*results*, *imgfile_prefix*, *to_label_id=True*, *indices=None*)

Format the results into dir (standard format for ade20k evaluation).

Parameters

- **results** (*list*) – Testing results of the dataset.
- **imgfile_prefix** (*str | None*) – The prefix of images files. It includes the file path and the prefix of filename, e.g., "a/b/prefix".
- **to_label_id** (*bool*) – whether convert output to label_id for submission. Default: False
- **indices** (*list[int], optional*) – Indices of input results, if not set, all the indices of the dataset will be used. Default: None.

Returns

(result_files, tmp_dir), result_files is a list containing

the image paths, tmp_dir is the temporal directory created for saving json/png files when img_prefix is not specified.

Return type tuple

results2img(*results*, *imgfile_prefix*, *to_label_id*, *indices=None*)

Write the segmentation results to images.

Parameters

- **results** (*list[ndarray]*) – Testing results of the dataset.
- **imgfile_prefix** (*str*) – The filename prefix of the png files. If the prefix is "somepath/xxx", the png files will be named "somepath/xxx.png".
- **to_label_id** (*bool*) – whether convert output to label_id for submission.
- **indices** (*list[int], optional*) – Indices of input results, if not set, all the indices of the dataset will be used. Default: None.

Returns str]: result txt files which contains corresponding semantic segmentation images.

Return type list[str

class mmseg.datasets.**COCOStuffDataset**(***kwargs*)

COCO-Stuff dataset.

In segmentation map annotation for COCO-Stuff, Train-IDs of the 10k version are from 1 to 171, where 0 is the ignore index, and Train-ID of COCO Stuff 164k is from 0 to 170, where 255 is the ignore index. So, they are all 171 semantic categories. `reduce_zero_label` is set to True and False for the 10k and 164k versions, respectively. The `img_suffix` is fixed to '.jpg', and `seg_map_suffix` is fixed to '.png'.

class mmseg.datasets.**ChaseDB1Dataset**(***kwargs*)

Chase_db1 dataset.

In segmentation map annotation for Chase_db1, 0 stands for background, which is included in 2 categories. `reduce_zero_label` is fixed to False. The `img_suffix` is fixed to '.png' and `seg_map_suffix` is fixed to '_1stHO.png'.

class mmseg.datasets.**CityscapesDataset**(*img_suffix='_leftImg8bit.png'*, *seg_map_suffix='_gtFine_labelTrainIds.png'*, ***kwargs*)

Cityscapes dataset.

The `img_suffix` is fixed to '_leftImg8bit.png' and `seg_map_suffix` is fixed to '_gtFine_labelTrainIds.png' for Cityscapes dataset.

evaluate(*results*, *metric='mIoU'*, *logger=None*, *imgfile_prefix=None*)

Evaluation in Cityscapes/default protocol.

Parameters

- **results** (*list*) – Testing results of the dataset.
- **metric** (*str | list[str]*) – Metrics to be evaluated.
- **logger** (*logging.Logger | None | str*) – Logger used for printing related information during evaluation. Default: None.
- **imgfile_prefix** (*str | None*) – The prefix of output image file, for cityscapes evaluation only. It includes the file path and the prefix of filename, e.g., "a/b/prefix". If results are evaluated with cityscapes protocol, it would be the prefix of output png files. The output files would be png images under folder "a/b/prefix/xxx.png", where "xxx" is the image name of cityscapes. If not specified, a temp file will be created for evaluation. Default: None.

Returns Cityscapes/default metrics.

Return type dict[str, float]

format_results(*results*, *imgfile_prefix*, *to_label_id=True*, *indices=None*)

Format the results into dir (standard format for Cityscapes evaluation).

Parameters

- **results** (*list*) – Testing results of the dataset.
- **imgfile_prefix** (*str*) – The prefix of images files. It includes the file path and the prefix of filename, e.g., "a/b/prefix".
- **to_label_id** (*bool*) – whether convert output to label_id for submission. Default: False
- **indices** (*list[int], optional*) – Indices of input results, if not set, all the indices of the dataset will be used. Default: None.

Returns

(result_files, tmp_dir), result_files is a list containing the image paths, tmp_dir is the temporal directory created for saving json/png files when img_prefix is not specified.

Return type tuple

results2img(*results*, *imgfile_prefix*, *to_label_id*, *indices=None*)
Write the segmentation results to images.

Parameters

- **results** (*list[ndarray]*) – Testing results of the dataset.
- **imgfile_prefix** (*str*) – The filename prefix of the png files. If the prefix is "somepath/xxx", the png files will be named "somepath/xxx.png".
- **to_label_id** (*bool*) – whether convert output to label_id for submission.
- **indices** (*list[int], optional*) – Indices of input results, if not set, all the indices of the dataset will be used. Default: None.

Returns str]: result txt files which contains corresponding semantic segmentation images.

Return type list[str

class mmseg.datasets.**ConcatDataset**(*datasets*, *separate_eval=True*)
A wrapper of concatenated dataset.

Same as `torch.utils.data.dataset.ConcatDataset`, but support evaluation and formatting results

Parameters

- **datasets** (list[Dataset]) – A list of datasets.
- **separate_eval** (*bool*) – Whether to evaluate the concatenated dataset results separately, Defaults to True.

evaluate(*results*, *logger=None*, ***kwargs*)
Evaluate the results.

Parameters

- **results** (*list[tuple[torch.Tensor]] | list[str]*) – per image pre_eval results or predict segmentation map for computing evaluation metric.
- **logger** (*logging.Logger | str | None*) – Logger used for printing related information during evaluation. Default: None.

Returns

float]: evaluate results of the total dataset or each separate

dataset if *self.separate_eval=True*.

Return type dict[str

format_results(*results*, *imgfile_prefix*, *indices=None*, ***kwargs*)
format result for every sample of ConcatDataset.

get_dataset_idx_and_sample_idx(*indice*)
Return dataset and sample index when given an indice of ConcatDataset.

Parameters **indice** (*int*) – indice of sample in ConcatDataset

Returns the index of sub dataset the sample belong to int: the index of sample in its corresponding subset

Return type int

pre_eval(*preds*, *indices*)
do pre eval for every sample of ConcatDataset.

class mmseg.datasets.**CustomDataset**(*pipeline*, *img_dir*, *img_suffix='.jpg'*, *ann_dir=None*, *seg_map_suffix='.png'*, *split=None*, *data_root=None*, *test_mode=False*, *ignore_index=255*, *reduce_zero_label=False*, *classes=None*, *palette=None*, *gt_seg_map_loader_cfg=None*, *file_client_args={'backend': 'disk'}*)

Custom dataset for semantic segmentation. An example of file structure is as followed.

```
├── data
│   ├── my_dataset
│   │   ├── img_dir
│   │   │   ├── train
│   │   │   │   ├── xxx{img_suffix}
│   │   │   │   ├── yyy{img_suffix}
│   │   │   │   ├── zzz{img_suffix}
│   │   │   ├── val
│   │   ├── ann_dir
│   │   │   ├── train
│   │   │   │   ├── xxx{seg_map_suffix}
│   │   │   │   ├── yyy{seg_map_suffix}
│   │   │   │   ├── zzz{seg_map_suffix}
│   │   │   ├── val
```

The img/gt_semantic_seg pair of CustomDataset should be of the same except suffix. A valid img/gt_semantic_seg filename pair should be like `xxx{img_suffix}` and `xxx{seg_map_suffix}` (extension is also included in the suffix). If split is given, then `xxx` is specified in txt file. Otherwise, all files in `img_dir/` ``and ``ann_dir` will be loaded. Please refer to `docs/en/tutorials/new_dataset.md` for more details.

Parameters

- **pipeline** (*list[dict]*) – Processing pipeline
- **img_dir** (*str*) – Path to image directory
- **img_suffix** (*str*) – Suffix of images. Default: '.jpg'
- **ann_dir** (*str, optional*) – Path to annotation directory. Default: None
- **seg_map_suffix** (*str*) – Suffix of segmentation maps. Default: '.png'
- **split** (*str, optional*) – Split txt file. If split is specified, only file with suffix in the splits will be loaded. Otherwise, all images in img_dir/ann_dir will be loaded. Default: None
- **data_root** (*str, optional*) – Data root for img_dir/ann_dir. Default: None.
- **test_mode** (*bool*) – If test_mode=True, gt wouldn't be loaded.
- **ignore_index** (*int*) – The label index to be ignored. Default: 255
- **reduce_zero_label** (*bool*) – Whether to mark label zero as ignored. Default: False
- **classes** (*str | Sequence[str], optional*) – Specify classes to load. If is None, `cls.CLASSES` will be used. Default: None.
- **palette** (*Sequence[Sequence[int]]] | np.ndarray | None*) – The palette of segmentation map. If None is given, and self.PALETTE is None, random palette will be generated. Default: None
- **gt_seg_map_loader_cfg** (*dict, optional*) – build LoadAnnotations to load gt for evaluation, load from disk by default. Default: None.

- **file_client_args** (*dict*) – Arguments to instantiate a FileClient. See `mmcv.fileio.FileClient` for details. Defaults to `dict(backend='disk')`.

evaluate(*results*, *metric='mIoU'*, *logger=None*, *gt_seg_maps=None*, ***kwargs*)
Evaluate the dataset.

Parameters

- **results** (*list[tuple[torch.Tensor]] | list[str]*) – per image pre_eval results or predict segmentation map for computing evaluation metric.
- **metric** (*str | list[str]*) – Metrics to be evaluated. 'mIoU', 'mDice' and 'mFscore' are supported.
- **logger** (*logging.Logger | None | str*) – Logger used for printing related information during evaluation. Default: None.
- **gt_seg_maps** (*generator[ndarray]*) – Custom gt seg maps as input, used in ConcatDataset

Returns Default metrics.

Return type dict[str, float]

format_results(*results*, *imgfile_prefix*, *indices=None*, ***kwargs*)
Place holder to format result to dataset specific output.

get_ann_info(*idx*)
Get annotation by index.

Parameters **idx** (*int*) – Index of data.

Returns Annotation info of specified index.

Return type dict

get_classes_and_palette(*classes=None*, *palette=None*)
Get class names of current dataset.

Parameters

- **classes** (*Sequence[str] | str | None*) – If classes is None, use default CLASSES defined by builtin dataset. If classes is a string, take it as a file name. The file contains the name of classes where each line contains one class name. If classes is a tuple or list, override the CLASSES defined by the dataset.
- **palette** (*Sequence[Sequence[int]] | np.ndarray | None*) – The palette of segmentation map. If None is given, random palette will be generated. Default: None

get_gt_seg_map_by_idx(*index*)
Get one ground truth segmentation map for evaluation.

get_gt_seg_maps(*efficient_test=None*)
Get ground truth segmentation maps for evaluation.

load_annotations(*img_dir*, *img_suffix*, *ann_dir*, *seg_map_suffix*, *split*)
Load annotation from directory.

Parameters

- **img_dir** (*str*) – Path to image directory
- **img_suffix** (*str*) – Suffix of images.
- **ann_dir** (*str|None*) – Path to annotation directory.

- **seg_map_suffix** (*str|None*) – Suffix of segmentation maps.
- **split** (*str|None*) – Split txt file. If split is specified, only file with suffix in the splits will be loaded. Otherwise, all images in img_dir/ann_dir will be loaded. Default: None

Returns All image info of dataset.

Return type list[dict]

pre_eval(*preds*, *indices*)
Collect eval result from each iteration.

Parameters

- **preds** (*list[torch.Tensor] | torch.Tensor*) – the segmentation logit after argmax, shape (N, H, W).
- **indices** (*list[int] | int*) – the prediction related ground truth indices.

Returns

(area_intersect, area_union, area_prediction, area_ground_truth).

Return type list[torch.Tensor]

pre_pipeline(*results*)
Prepare results dict for pipeline.

prepare_test_img(*idx*)
Get testing data after pipeline.

Parameters **idx** (*int*) – Index of data.

Returns

Testing data after pipeline with new keys introduced by pipeline.

Return type dict

prepare_train_img(*idx*)
Get training data and annotations after pipeline.

Parameters **idx** (*int*) – Index of data.

Returns

Training data and annotation after pipeline with new keys introduced by pipeline.

Return type dict

class mmseg.datasets.**DRIVEDataset**(***kwargs*)
DRIVE dataset.

In segmentation map annotation for DRIVE, 0 stands for background, which is included in 2 categories. `reduce_zero_label` is fixed to False. The `img_suffix` is fixed to '.png' and `seg_map_suffix` is fixed to '_manual1.png'.

class mmseg.datasets.**DarkZurichDataset**(***kwargs*)
DarkZurichDataset dataset.

class mmseg.datasets.**HRFDataset**(***kwargs*)
HRF dataset.

In segmentation map annotation for HRF, 0 stands for background, which is included in 2 categories. `reduce_zero_label` is fixed to False. The `img_suffix` is fixed to '.png' and `seg_map_suffix` is fixed to '.png'.

class mmseg.datasets.**ISPRSDataset**(***kwargs*)
ISPRS dataset.

In segmentation map annotation for LoveDA, 0 is the ignore index. `reduce_zero_label` should be set to True. The `img_suffix` and `seg_map_suffix` are both fixed to '.png'.

class mmseg.datasets.**LoveDADataset**(***kwargs*)
LoveDA dataset.

In segmentation map annotation for LoveDA, 0 is the ignore index. `reduce_zero_label` should be set to True. The `img_suffix` and `seg_map_suffix` are both fixed to '.png'.

format_results(*results*, *imgfile_prefix*, *indices=None*)
Format the results into dir (standard format for LoveDA evaluation).

Parameters

- **results** (*list*) – Testing results of the dataset.
- **imgfile_prefix** (*str*) – The prefix of images files. It includes the file path and the prefix of filename, e.g., "a/b/prefix".
- **indices** (*list[int], optional*) – Indices of input results, if not set, all the indices of the dataset will be used. Default: None.

Returns

(result_files, tmp_dir), result_files is a list containing the image paths, tmp_dir is the temporal directory created for saving json/png files when img_prefix is not specified.

Return type tuple

results2img(*results*, *imgfile_prefix*, *indices=None*)
Write the segmentation results to images.

Parameters

- **results** (*list[ndarray]*) – Testing results of the dataset.
- **imgfile_prefix** (*str*) – The filename prefix of the png files. If the prefix is "somepath/xxx", the png files will be named "somepath/xxx.png".
- **indices** (*list[int], optional*) – Indices of input results, if not set, all the indices of the dataset will be used. Default: None.

Returns str]: result txt files which contains corresponding semantic segmentation images.

Return type list[str

class mmseg.datasets.**MultiImageMixDataset**(*dataset*, *pipeline*, *skip_type_keys=None*)
A wrapper of multiple images mixed dataset.

Suitable for training on multiple images mixed data augmentation like mosaic and mixup. For the augmentation pipeline of mixed image data, the *get_indexes* method needs to be provided to obtain the image indexes, and you can set *skip_flags* to change the pipeline running process.

Parameters

- **dataset** (*CustomDataset*) – The dataset to be mixed.
- **pipeline** (*Sequence[dict]*) – Sequence of transform object or config dict to be composed.
- **skip_type_keys** (*list[str], optional*) – Sequence of type string to be skip pipeline. Default to None.

update_skip_type_keys(*skip_type_keys*)
Update skip_type_keys.

It is called by an external hook.

Parameters **skip_type_keys** (*list[str], optional*) – Sequence of type string to be skip pipeline.

class mmseg.datasets.**NightDrivingDataset**(**kwargs*)
NightDrivingDataset dataset.

class mmseg.datasets.**PascalContextDataset**(*split*, **kwargs*)
PascalContext dataset.

In segmentation map annotation for PascalContext, 0 stands for background, which is included in 60 categories. `reduce_zero_label` is fixed to False. The `img_suffix` is fixed to '.jpg' and `seg_map_suffix` is fixed to '.png'.

Parameters **split** (*str*) – Split txt file for PascalContext.

class mmseg.datasets.**PascalContextDataset59**(*split*, **kwargs*)
PascalContext dataset.

In segmentation map annotation for PascalContext, 0 stands for background, which is included in 60 categories. `reduce_zero_label` is fixed to False. The `img_suffix` is fixed to '.jpg' and `seg_map_suffix` is fixed to '.png'.

Parameters **split** (*str*) – Split txt file for PascalContext.

class mmseg.datasets.**PascalVOCDataset**(*split*, **kwargs*)
Pascal VOC dataset.

Parameters **split** (*str*) – Split txt file for Pascal VOC.

class mmseg.datasets.**PotsdamDataset**(**kwargs*)
ISPRS Potsdam dataset.

In segmentation map annotation for Potsdam dataset, 0 is the ignore index. `reduce_zero_label` should be set to True. The `img_suffix` and `seg_map_suffix` are both fixed to '.png'.

class mmseg.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 mmseg.datasets.**STAREDataset**(**kwargs*)
STARE dataset.

In segmentation map annotation for STARE, 0 stands for background, which is included in 2 categories. `reduce_zero_label` is fixed to False. The `img_suffix` is fixed to '.png' and `seg_map_suffix` is fixed to '.ah.png'.

mmseg.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 training 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.
- **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** – any keyword argument to be used to initialize DataLoader

Returns A PyTorch dataloader.

Return type DataLoader

mmseg.datasets.**build_dataset**(*cfg*, *default_args=None*)
Build datasets.

class mmseg.datasets.**iSAIDDataset**(***kwargs*)
iSAID: A Large-scale Dataset for Instance Segmentation in Aerial Images In segmentation map annotation for iSAID dataset, which is included in 16 categories. `reduce_zero_label` is fixed to False. The `img_suffix` is fixed to '.png' and `seg_map_suffix` is fixed to '_manual1.png'.

load_annotations(*img_dir*, *img_suffix*, *ann_dir*, *seg_map_suffix=None*, *split=None*)
Load annotation from directory.

Parameters

- **img_dir** (*str*) – Path to image directory
- **img_suffix** (*str*) – Suffix of images.
- **ann_dir** (*str|None*) – Path to annotation directory.
- **seg_map_suffix** (*str|None*) – Suffix of segmentation maps.
- **split** (*str|None*) – Split txt file. If split is specified, only file with suffix in the splits will be loaded. Otherwise, all images in img_dir/ann_dir will be loaded. Default: None

Returns All image info of dataset.

Return type list[dict]

24.2 pipelines

class mmseg.datasets.pipelines.**AdjustGamma**(*gamma=1.0*)
Using gamma correction to process the image.

Parameters **gamma** (*float or int*) – Gamma value used in gamma correction. Default: 1.0.

class mmseg.datasets.pipelines.**CLAHE**(*clip_limit=40.0*, *tile_grid_size=(8, 8)*)
Use CLAHE method to process the image.

See *ZUIDERVELD,K. Contrast Limited Adaptive Histogram Equalization[J]. Graphics Gems, 1994:474-485.* for more information.

Parameters

- **clip_limit** (*float*) – Threshold for contrast limiting. Default: 40.0.
- **tile_grid_size** (*tuple[int]*) – Size of grid for histogram equalization. Input image will be divided into equally sized rectangular tiles. It defines the number of tiles in row and column. Default: (8, 8).

class mmseg.datasets.pipelines.**Collect**(*keys*, *meta_keys=('filename', 'ori_filename', 'ori_shape', 'img_shape', 'pad_shape', 'scale_factor', 'flip', 'flip_direction', 'img_norm_cfg')*)
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_semantic_seg".

The "img_meta" item is always populated. The contents of the "img_meta" dictionary depends on "meta_keys". By default this includes:

- **"img_shape": shape of the image input to the network as a tuple** (h, w, c). Note that images may be zero padded on the bottom/right if the batch tensor is larger than this shape.
- "scale_factor": a float indicating the preprocessing scale
- "flip": a boolean indicating if image flip transform was used
- "filename": path to the image file
- "ori_shape": original shape of the image as a tuple (h, w, c)
- "pad_shape": image shape after padding
- **"img_norm_cfg": a dict of normalization information:**
  - mean - per channel mean subtraction
  - std - per channel std divisor
  - to_rgb - bool indicating if bgr was converted to rgb

Parameters

- **keys** (*Sequence[str]*) – Keys of results to be collected in `data`.
- **meta_keys** (*Sequence[str], optional*) – Meta keys to be converted to `mmcv.DataContainer` and collected in `data[img_metas]`. Default: (`filename`, `ori_filename`, `ori_shape`, `img_shape`, `pad_shape`, `scale_factor`, `flip`, `flip_direction`, `img_norm_cfg`)

class mmseg.datasets.pipelines.**Compose**(*transforms*)
Compose multiple transforms sequentially.

Parameters **transforms** (*Sequence[dict | callable]*) – Sequence of transform object or config dict to be composed.

class mmseg.datasets.pipelines.**ImageToTensor**(*keys*)

Convert image to `torch.Tensor` by given keys.

The dimension order of input image is (H, W, C). The pipeline will convert it to (C, H, W). If only 2 dimension (H, W) is given, the output would be (1, H, W).

Parameters **keys** (*Sequence[str]*) – Key of images to be converted to Tensor.

class mmseg.datasets.pipelines.**LoadAnnotations**(*reduce_zero_label=False*, *file_client_args={'backend': 'disk'}*, *imdecode_backend='pillow'*)

Load annotations for semantic segmentation.

Parameters

- **reduce_zero_label** (*bool*) – Whether reduce all label value by 1. Usually used for datasets where 0 is background label. Default: False.
- **file_client_args** (*dict*) – Arguments to instantiate a FileClient. See `mmcv.fileio.FileClient` for details. Defaults to `dict(backend='disk')`.
- **imdecode_backend** (*str*) – Backend for `mmcv.imdecode()`. Default: 'pillow'

class mmseg.datasets.pipelines.**LoadImageFromFile**(*to_float32=False*, *color_type='color'*, *file_client_args={'backend': 'disk'}*, *imdecode_backend='cv2'*)

Load an image from file.

Required keys are "img_prefix" and "img_info" (a dict that must contain the key "filename"). Added or updated keys are "filename", "img", "img_shape", "ori_shape" (same as *img_shape*), "pad_shape" (same as *img_shape*), "scale_factor" (1.0) and "img_norm_cfg" (means=0 and stds=1).

Parameters

- **to_float32** (*bool*) – Whether to convert the loaded image to a float32 numpy array. If set to False, the loaded image is an uint8 array. Defaults to False.
- **color_type** (*str*) – The flag argument for `mmcv.imfrombytes()`. Defaults to 'color'.
- **file_client_args** (*dict*) – Arguments to instantiate a FileClient. See `mmcv.fileio.FileClient` for details. Defaults to `dict(backend='disk')`.
- **imdecode_backend** (*str*) – Backend for `mmcv.imdecode()`. Default: 'cv2'

class mmseg.datasets.pipelines.**MultiScaleFlipAug**(*transforms*, *img_scale*, *img_ratios=None*, *flip=False*, *flip_direction='horizontal'*)

Test-time augmentation with multiple scales and flipping.

An example configuration is as followed:

```
img_scale=(2048, 1024),
img_ratios=[0.5, 1.0],
flip=True,
transforms=[
    dict(type='Resize', keep_ratio=True),
    dict(type='RandomFlip'),
    dict(type='Normalize', **img_norm_cfg),
    dict(type='Pad', size_divisor=32),
    dict(type='ImageToTensor', keys=['img']),
```

(continues on next page)

(continued from previous page)

```
    dict(type='Collect', keys=['img']),
]
```

After MultiScaleFLipAug with above configuration, the results are wrapped into lists of the same length as followed:

```
dict(
    img=[...],
    img_shape=[...],
    scale=[(1024, 512), (1024, 512), (2048, 1024), (2048, 1024)]
    flip=[False, True, False, True]
    ...
)
```

Parameters

- **transforms** (*list[dict]*) – Transforms to apply in each augmentation.
- **img_scale** (*None | tuple | list[tuple]*) – Images scales for resizing.
- **img_ratios** (*float | list[float]*) – Image ratios for resizing
- **flip** (*bool*) – Whether apply flip augmentation. Default: False.
- **flip_direction** (*str | list[str]*) – Flip augmentation directions, options are "horizontal" and "vertical". If flip_direction is list, multiple flip augmentations will be applied. It has no effect when flip == False. Default: "horizontal".

class mmseg.datasets.pipelines.**Normalize**(*mean*, *std*, *to_rgb=True*)

Normalize the image.

Added key is "img_norm_cfg".

Parameters

- **mean** (*sequence*) – Mean values of 3 channels.
- **std** (*sequence*) – Std values of 3 channels.
- **to_rgb** (*bool*) – Whether to convert the image from BGR to RGB, default is true.

class mmseg.datasets.pipelines.**Pad**(*size=None*, *size_divisor=None*, *pad_val=0*, *seg_pad_val=255*)

Pad the image & mask.

There are two padding modes: (1) pad to a fixed size and (2) pad to the minimum size that is divisible by some number. Added keys are "pad_shape", "pad_fixed_size", "pad_size_divisor",

Parameters

- **size** (*tuple, optional*) – Fixed padding size.
- **size_divisor** (*int, optional*) – The divisor of padded size.
- **pad_val** (*float, optional*) – Padding value. Default: 0.
- **seg_pad_val** (*float, optional*) – Padding value of segmentation map. Default: 255.

class mmseg.datasets.pipelines.**PhotoMetricDistortion**(*brightness_delta=32*, *contrast_range=(0.5, 1.5)*, *saturation_range=(0.5, 1.5)*, *hue_delta=18*)

Apply photometric distortion to image sequentially, every transformation is applied with a probability of 0.5. The position of random contrast is in second or second to last.

1. random brightness
2. random contrast (mode 0)
3. convert color from BGR to HSV
4. random saturation
5. random hue
6. convert color from HSV to BGR
7. random contrast (mode 1)

Parameters

- **brightness_delta** (*int*) – delta of brightness.
- **contrast_range** (*tuple*) – range of contrast.
- **saturation_range** (*tuple*) – range of saturation.
- **hue_delta** (*int*) – delta of hue.

brightness(*img*)
Brightness distortion.

contrast(*img*)
Contrast distortion.

convert(*img*, *alpha=1*, *beta=0*)
Multiple with alpha and add beat with clip.

hue(*img*)
Hue distortion.

saturation(*img*)
Saturation distortion.

class mmseg.datasets.pipelines.**RGB2Gray**(*out_channels=None*, *weights=(0.299, 0.587, 0.114)*)

Convert RGB image to grayscale image.

This transform calculate the weighted mean of input image channels with `weights` and then expand the channels to `out_channels`. When `out_channels` is None, the number of output channels is the same as input channels.

Parameters

- **out_channels** (*int*) – Expected number of output channels after transforming. Default: None.
- **weights** (*tuple[float]*) – The weights to calculate the weighted mean. Default: (0.299, 0.587, 0.114).

class mmseg.datasets.pipelines.**RandomCrop**(*crop_size*, *cat_max_ratio=1.0*, *ignore_index=255*)

Random crop the image & seg.

Parameters

- **crop_size** (*tuple*) – Expected size after cropping, (h, w).

- **cat_max_ratio** (*float*) – The maximum ratio that single category could occupy.

crop(*img*, *crop_bbox*)
Crop from `img`

get_crop_bbox(*img*)
Randomly get a crop bounding box.

class mmseg.datasets.pipelines.**RandomCutOut**(*prob*, *n_holes*, *cutout_shape=None*, *cutout_ratio=None*, *fill_in=(0, 0, 0)*, *seg_fill_in=None*)

CutOut operation.

Randomly drop some regions of image used in Cutout. :param prob: cutout probability. :type prob: float :param n_holes: Number of regions to be dropped.

If it is given as a list, number of holes will be randomly selected from the closed interval [*n_holes[0]*, *n_holes[1]*].

Parameters

- **cutout_shape** (*tuple[int, int] | list[tuple[int, int]]*) – The candidate shape of dropped regions. It can be *tuple[int, int]* to use a fixed cutout shape, or *list[tuple[int, int]]* to randomly choose shape from the list.
- **cutout_ratio** (*tuple[float, float] | list[tuple[float, float]]*) – The candidate ratio of dropped regions. It can be *tuple[float, float]* to use a fixed ratio or *list[tuple[float, float]]* to randomly choose ratio from the list. Please note that *cutout_shape* and *cutout_ratio* cannot be both given at the same time.
- **fill_in** (*tuple[float, float, float] | tuple[int, int, int]*) – The value of pixel to fill in the dropped regions. Default: (0, 0, 0).
- **seg_fill_in** (*int*) – The labels of pixel to fill in the dropped regions. If seg_fill_in is None, skip. Default: None.

class mmseg.datasets.pipelines.**RandomFlip**(*prob=None*, *direction='horizontal'*)
Flip the image & seg.

If the input dict contains the key "flip", then the flag will be used, otherwise it will be randomly decided by a ratio specified in the init method.

Parameters

- **prob** (*float, optional*) – The flipping probability. Default: None.
- **direction** (*str, optional*) – The flipping direction. Options are 'horizontal' and 'vertical'. Default: 'horizontal'.

class mmseg.datasets.pipelines.**RandomMosaic**(*prob*, *img_scale=(640, 640)*, *center_ratio_range=(0.5, 1.5)*, *pad_val=0*, *seg_pad_val=255*)

Mosaic augmentation. Given 4 images, mosaic transform combines them into one output image. The output image is composed of the parts from each sub- image.

```
                mosaic transform
                   center_x
          +------------------------------+
          |       pad        |  pad      |
          |      +-----------+           |
          |      |           |           |
          |      |  image1   |--------+  |
```

(continues on next page)

(continued from previous page)

```
           |      |           |        |  |
           |      |           | image2 |  |
center_y   |----+-------------+-----------|
           |    |   cropped   |           |
           |pad |   image3    |  image4   |
           |    |             |           |
           +----|-------------+-----------+
                |             |
                +-------------+

The mosaic transform steps are as follows:
    1. Choose the mosaic center as the intersections of 4 images
    2. Get the left top image according to the index, and randomly
       sample another 3 images from the custom dataset.
    3. Sub image will be cropped if image is larger than mosaic patch
```

Parameters

- **prob** (*float*) – mosaic probability.
- **img_scale** (*Sequence[int]*) – Image size after mosaic pipeline of a single image. The size of the output image is four times that of a single image. The output image comprises 4 single images. Default: (640, 640).
- **center_ratio_range** (*Sequence[float]*) – Center ratio range of mosaic output. Default: (0.5, 1.5).
- **pad_val** (*int*) – Pad value. Default: 0.
- **seg_pad_val** (*int*) – Pad value of segmentation map. Default: 255.

get_indexes(*dataset*)

Call function to collect indexes.

Parameters **dataset** (MultiImageMixDataset) – The dataset.

Returns indexes.

Return type list

class mmseg.datasets.pipelines.**RandomRotate**(*prob*, *degree*, *pad_val=0*, *seg_pad_val=255*, *center=None*, *auto_bound=False*)

Rotate the image & seg.

Parameters

- **prob** (*float*) – The rotation probability.
- **degree** (*float, tuple[float]*) – Range of degrees to select from. If degree is a number instead of tuple like (min, max), the range of degree will be (-degree, +degree)
- **pad_val** (*float, optional*) – Padding value of image. Default: 0.
- **seg_pad_val** (*float, optional*) – Padding value of segmentation map. Default: 255.
- **center** (*tuple[float], optional*) – Center point (w, h) of the rotation in the source image. If not specified, the center of the image will be used. Default: None.
- **auto_bound** (*bool*) – Whether to adjust the image size to cover the whole rotated image. Default: False

class mmseg.datasets.pipelines.**Rerange**(*min_value=0*, *max_value=255*)
Rerange the image pixel value.

Parameters

- **min_value** (*float or int*) – Minimum value of the reranged image. Default: 0.
- **max_value** (*float or int*) – Maximum value of the reranged image. Default: 255.

class mmseg.datasets.pipelines.**Resize**(*img_scale=None*, *multiscale_mode='range'*, *ratio_range=None*, *keep_ratio=True*, *min_size=None*)
Resize images & seg.

This transform resizes the input image to some scale. If the input dict contains the key "scale", then the scale in the input dict is used, otherwise the specified scale in the init method is used.

`img_scale` can be None, a tuple (single-scale) or a list of tuple (multi-scale). There are 4 multiscale modes:

- `ratio_range is not None`:

1. **When img_scale is None, img_scale is the shape of image in results** (img_scale = results['img'].shape[:2]) and the image is resized based on the original size. (mode 1)
2. **When img_scale is a tuple (single-scale), randomly sample a ratio from** the ratio range and multiply it with the image scale. (mode 2)

- `ratio_range is None and multiscale_mode == "range"`: randomly sample a

scale from the a range. (mode 3)

- `ratio_range is None and multiscale_mode == "value"`: randomly sample a

scale from multiple scales. (mode 4)

Parameters

- **img_scale** (*tuple or list[tuple]*) – Images scales for resizing. Default:None.
- **multiscale_mode** (*str*) – Either "range" or "value". Default: 'range'
- **ratio_range** (*tuple[float]*) – (min_ratio, max_ratio). Default: None
- **keep_ratio** (*bool*) – Whether to keep the aspect ratio when resizing the image. Default: True
- **min_size** (*int, optional*) – The minimum size for input and the shape of the image and seg map will not be less than `min_size`. As the shape of model input is fixed like 'SETR' and 'BEiT'. Following the setting in these models, resized images must be bigger than the crop size in `slide_inference`. Default: None

static **random_sample**(*img_scales*)
Randomly sample an img_scale when `multiscale_mode=='range'`.

Parameters **img_scales** (*list[tuple]*) – Images scale range for sampling. There must be two tuples in img_scales, which specify the lower and upper bound of image scales.

Returns

Returns a tuple (img_scale, None), where `img_scale` is sampled scale and None is just a placeholder to be consistent with *random_select()*.

Return type (tuple, None)

static random_sample_ratio(*img_scale*, *ratio_range*)
Randomly sample an img_scale when `ratio_range` is specified.

A ratio will be randomly sampled from the range specified by `ratio_range`. Then it would be multiplied with `img_scale` to generate sampled scale.

Parameters

- **img_scale** (*tuple*) – Images scale base to multiply with ratio.
- **ratio_range** (*tuple[float]*) – The minimum and maximum ratio to scale the `img_scale`.

Returns

Returns a tuple (scale, None), where `scale` is sampled ratio multiplied with `img_scale` and None is just a placeholder to be consistent with *random_select()*.

Return type (tuple, None)

static random_select(*img_scales*)
Randomly select an img_scale from given candidates.

Parameters **img_scales** (*list[tuple]*) – Images scales for selection.

Returns

Returns a tuple (img_scale, scale_dix), where `img_scale` is the selected image scale and `scale_idx` is the selected index in the given candidates.

Return type (tuple, int)

class mmseg.datasets.pipelines.**SegRescale**(*scale_factor=1*)
Rescale semantic segmentation maps.

Parameters **scale_factor** (*float*) – The scale factor of the final output.

class mmseg.datasets.pipelines.**ToDataContainer**(*fields=({'key': 'img', 'stack': True}, {'key': 'gt_semantic_seg'})*)
Convert results to `mmcv.DataContainer` by given fields.

Parameters **fields** (*Sequence[dict]*) – Each field is a dict like `dict(key='xxx', **kwargs)`. The `key` in result will be converted to `mmcv.DataContainer` with `**kwargs`. Default: `(dict(key='img', stack=True), dict(key='gt_semantic_seg'))`.

class mmseg.datasets.pipelines.**ToTensor**(*keys*)
Convert some results to `torch.Tensor` by given keys.

Parameters **keys** (*Sequence[str]*) – Keys that need to be converted to Tensor.

class mmseg.datasets.pipelines.**Transpose**(*keys*, *order*)
Transpose some results by given keys.

Parameters

- **keys** (*Sequence[str]*) – Keys of results to be transposed.
- **order** (*Sequence[int]*) – Order of transpose.

mmseg.datasets.pipelines.**to_tensor**(*data*)
Convert objects of various python types to `torch.Tensor`.

Supported types are: `numpy.ndarray`, `torch.Tensor`, `Sequence`, `int` and `float`.

Parameters **data** (*torch.Tensor | numpy.ndarray | Sequence | int | float*) – Data to be converted.

CHAPTER

TWENTYFIVE

MMSEG.MODELS

25.1 segmentors

class mmseg.models.segmentors.**BaseSegmentor**(*init_cfg=None*)

Base class for segmentors.

abstract aug_test(*imgs*, *img_metas*, ***kwargs*)

Placeholder for augmentation test.

abstract encode_decode(*img*, *img_metas*)

Placeholder for encode images with backbone and decode into a semantic segmentation map of the same size as input.

abstract extract_feat(*imgs*)

Placeholder for extract features from images.

forward(*img*, *img_metas*, *return_loss=True*, ***kwargs*)

Calls either *forward_train()* or *forward_test()* depending on whether `return_loss` is `True`.

Note this setting will change the expected inputs. When `return_loss=True`, img and img_meta are single-nested (i.e. Tensor and List[dict]), and when `resturn_loss=False`, img and img_meta should be double nested (i.e. List[Tensor], List[List[dict]]), with the outer list indicating test time augmentations.

forward_test(*imgs*, *img_metas*, ***kwargs*)

Parameters

- **imgs** (*List[Tensor]*) – the outer list indicates test-time augmentations and inner Tensor should have a shape NxCxHxW, which contains all images in the batch.
- **img_metas** (*List[List[dict]]*) – the outer list indicates test-time augs (multiscale, flip, etc.) and the inner list indicates images in a batch.

abstract forward_train(*imgs*, *img_metas*, ***kwargs*)

Placeholder for Forward function for training.

show_result(*img*, *result*, *palette=None*, *win_name=''*, *show=False*, *wait_time=0*, *out_file=None*, *opacity=0.5*)

Draw *result* over *img*.

Parameters

- **img** (*str or Tensor*) – The image to be displayed.
- **result** (*Tensor*) – The semantic segmentation results to draw over *img*.

- **palette** (*list[list[int]] | np.ndarray | None*) – The palette of segmentation map. If None is given, random palette will be generated. Default: None
- **win_name** (*str*) – The window name.
- **wait_time** (*int*) – Value of waitKey param. Default: 0.
- **show** (*bool*) – Whether to show the image. Default: False.
- **out_file** (*str or None*) – The filename to write the image. Default: None.
- **opacity** (*float*) – Opacity of painted segmentation map. Default 0.5. Must be in (0, 1] range.

Returns Only if not *show* or *out_file*

Return type img (Tensor)

abstract simple_test(*img*, *img_meta*, ***kwargs*)
Placeholder for single image test.

train_step(*data_batch*, *optimizer*, ***kwargs*)
The iteration step during training.

This method defines an iteration step during training, except for the back propagation and optimizer updating, which are done in an optimizer hook. Note that in some complicated cases or models, the whole process including back propagation and optimizer updating is also defined in this method, such as GAN.

Parameters

- **data** (*dict*) – The output of dataloader.
- **optimizer** (`torch.optim.Optimizer` | dict) – The optimizer of runner is passed to `train_step()`. This argument is unused and reserved.

Returns

It should contain at least 3 keys: `loss`, `log_vars`, `num_samples`. `loss` is a tensor for back propagation, which can be a weighted sum of multiple losses. `log_vars` contains all the variables to be sent to the logger. `num_samples` indicates the batch size (when the model is DDP, it means the batch size on each GPU), which is used for averaging the logs.

Return type dict

val_step(*data_batch*, *optimizer=None*, ***kwargs*)
The iteration step during validation.

This method shares the same signature as *train_step()*, but used during val epochs. Note that the evaluation after training epochs is not implemented with this method, but an evaluation hook.

property with_auxiliary_head
whether the segmentor has auxiliary head

Type bool

property with_decode_head
whether the segmentor has decode head

Type bool

property with_neck
whether the segmentor has neck

Type bool

class mmseg.models.segmentors.**CascadeEncoderDecoder**(*num_stages*, *backbone*, *decode_head*, *neck=None*, *auxiliary_head=None*, *train_cfg=None*, *test_cfg=None*, *pretrained=None*, *init_cfg=None*)

Cascade Encoder Decoder segmentors.

CascadeEncoderDecoder almost the same as EncoderDecoder, while decoders of CascadeEncoderDecoder are cascaded. The output of previous decoder_head will be the input of next decoder_head.

encode_decode(*img*, *img_metas*)
Encode images with backbone and decode into a semantic segmentation map of the same size as input.

class mmseg.models.segmentors.**EncoderDecoder**(*backbone*, *decode_head*, *neck=None*, *auxiliary_head=None*, *train_cfg=None*, *test_cfg=None*, *pretrained=None*, *init_cfg=None*)

Encoder Decoder segmentors.

EncoderDecoder typically consists of backbone, decode_head, auxiliary_head. Note that auxiliary_head is only used for deep supervision during training, which could be dumped during inference.

aug_test(*imgs*, *img_metas*, *rescale=True*)
Test with augmentations.

Only rescale=True is supported.

encode_decode(*img*, *img_metas*)
Encode images with backbone and decode into a semantic segmentation map of the same size as input.

extract_feat(*img*)
Extract features from images.

forward_dummy(*img*)
Dummy forward function.

forward_train(*img*, *img_metas*, *gt_semantic_seg*)
Forward function for training.

Parameters

- **img** (*Tensor*) – Input images.
- **img_metas** (*list[dict]*) – List of image info dict where each dict has: 'img_shape', 'scale_factor', 'flip', and may also contain 'filename', 'ori_shape', 'pad_shape', and 'img_norm_cfg'. For details on the values of these keys see *mmseg/datasets/pipelines/formatting.py:Collect*.
- **gt_semantic_seg** (*Tensor*) – Semantic segmentation masks used if the architecture supports semantic segmentation task.

Returns a dictionary of loss components

Return type dict[str, Tensor]

inference(*img*, *img_meta*, *rescale*)
Inference with slide/whole style.

Parameters

- **img** (*Tensor*) – The input image of shape (N, 3, H, W).
- **img_meta** (*dict*) – Image info dict where each dict has: 'img_shape', 'scale_factor', 'flip', and may also contain 'filename', 'ori_shape', 'pad_shape', and 'img_norm_cfg'. For details on the values of these keys see *mmseg/datasets/pipelines/formatting.py:Collect*.

- **rescale** (*bool*) – Whether rescale back to original shape.

Returns The output segmentation map.

Return type Tensor

simple_test(*img*, *img_meta*, *rescale=True*)
Simple test with single image.

slide_inference(*img*, *img_meta*, *rescale*)
Inference by sliding-window with overlap.

If h_crop > h_img or w_crop > w_img, the small patch will be used to decode without padding.

whole_inference(*img*, *img_meta*, *rescale*)
Inference with full image.

25.2 backbones

class mmseg.models.backbones.**BEiT**(*img_size=224*, *patch_size=16*, *in_channels=3*, *embed_dims=768*, *num_layers=12*, *num_heads=12*, *mlp_ratio=4*, *out_indices=- 1*, *qv_bias=True*, *attn_drop_rate=0.0*, *drop_path_rate=0.0*, *norm_cfg={'type': 'LN'}*, *act_cfg={'type': 'GELU'}*, *patch_norm=False*, *final_norm=False*, *num_fcs=2*, *norm_eval=False*, *pretrained=None*, *init_values=0.1*, *init_cfg=None*)
BERT Pre-Training of Image Transformers.

Parameters

- **img_size** (*int | tuple*) – Input image size. Default: 224.
- **patch_size** (*int*) – The patch size. Default: 16.
- **in_channels** (*int*) – Number of input channels. Default: 3.
- **embed_dims** (*int*) – Embedding dimension. Default: 768.
- **num_layers** (*int*) – Depth of transformer. Default: 12.
- **num_heads** (*int*) – Number of attention heads. Default: 12.
- **mlp_ratio** (*int*) – Ratio of mlp hidden dim to embedding dim. Default: 4.
- **out_indices** (*list | tuple | int*) – Output from which stages. Default: -1.
- **qv_bias** (*bool*) – Enable bias for qv if True. Default: True.
- **attn_drop_rate** (*float*) – The drop out rate for attention layer. Default 0.0
- **drop_path_rate** (*float*) – Stochastic depth rate. Default 0.0.
- **norm_cfg** (*dict*) – Config dict for normalization layer. Default: dict(type='LN')
- **act_cfg** (*dict*) – The activation config for FFNs. Default: dict(type='GELU').
- **patch_norm** (*bool*) – Whether to add a norm in PatchEmbed Block. Default: False.
- **final_norm** (*bool*) – Whether to add a additional layer to normalize final feature map. Default: False.
- **num_fcs** (*int*) – The number of fully-connected layers for FFNs. Default: 2.
- **norm_eval** (*bool*) – Whether to set norm layers to eval mode, namely, freeze running stats (mean and var). Note: Effect on Batch Norm and its variants only. Default: False.

- **pretrained** (*str, optional*) – Model pretrained path. Default: None.
- **init_values** (*float*) – Initialize the values of BEiTAttention and FFN with learnable scaling.
- **init_cfg** (*dict or list[dict], optional*) – Initialization config dict. Default: None.

forward(*inputs*)

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()

Initialize the weights.

resize_rel_pos_embed(*checkpoint*)

Resize relative pos_embed weights.

This function is modified from https://github.com/microsoft/unilm/blob/master/beit/semantic_segmentation/mmcv_custom/checkpoint.py. # noqa: E501 Copyright (c) Microsoft Corporation Licensed under the MIT License :param checkpoint: Key and value of the pretrain model. :type checkpoint: dict

Returns

Interpolate the relative pos_embed weights in the pre-train model to the current model size.

Return type state_dict (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 mmseg.models.backbones.**BiSeNetV1**(*backbone_cfg*, *in_channels=3*, *spatial_channels=(64, 64, 64, 128)*, *context_channels=(128, 256, 512)*, *out_indices=(0, 1, 2)*, *align_corners=False*, *out_channels=256*, *conv_cfg=None*, *norm_cfg={'requires_grad': True, 'type': 'BN'}*, *act_cfg={'type': 'ReLU'}*, *init_cfg=None*)

BiSeNetV1 backbone.

This backbone is the implementation of BiSeNet: Bilateral Segmentation Network for Real-time Semantic Segmentation.

Parameters

- **backbone_cfg** – (dict): Config of backbone of Context Path.

- **in_channels** (*int*) – The number of channels of input image. Default: 3.
- **spatial_channels** (*Tuple[int]*) – Size of channel numbers of various layers in Spatial Path. Default: (64, 64, 64, 128).
- **context_channels** (*Tuple[int]*) – Size of channel numbers of various modules in Context Path. Default: (128, 256, 512).
- **out_indices** (*Tuple[int] | int, optional*) – Output from which stages. Default: (0, 1, 2).
- **align_corners** (*bool, optional*) – The align_corners argument of resize operation in Bilateral Guided Aggregation Layer. Default: False.
- **out_channels** (*int*) – The number of channels of output. It must be the same with *in_channels* of decode_head. Default: 256.

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.

---

class mmseg.models.backbones.**BiSeNetV2**(*in_channels=3*, *detail_channels=(64, 64, 128)*, *semantic_channels=(16, 32, 64, 128)*, *semantic_expansion_ratio=6*, *bga_channels=128*, *out_indices=(0, 1, 2, 3, 4)*, *align_corners=False*, *conv_cfg=None*, *norm_cfg={'type': 'BN'}*, *act_cfg={'type': 'ReLU'}*, *init_cfg=None*)

BiSeNetV2: Bilateral Network with Guided Aggregation for Real-time Semantic Segmentation.

This backbone is the implementation of BiSeNetV2.

Parameters

- **in_channels** (*int*) – Number of channel of input image. Default: 3.
- **detail_channels** (*Tuple[int], optional*) – Channels of each stage in Detail Branch. Default: (64, 64, 128).
- **semantic_channels** (*Tuple[int], optional*) – Channels of each stage in Semantic Branch. Default: (16, 32, 64, 128). See Table 1 and Figure 3 of paper for more details.
- **semantic_expansion_ratio** (*int, optional*) – The expansion factor expanding channel number of middle channels in Semantic Branch. Default: 6.
- **bga_channels** (*int, optional*) – Number of middle channels in Bilateral Guided Aggregation Layer. Default: 128.
- **out_indices** (*Tuple[int] | int, optional*) – Output from which stages. Default: (0, 1, 2, 3, 4).
- **align_corners** (*bool, optional*) – The align_corners argument of resize operation in Bilateral Guided Aggregation Layer. Default: False.
- **conv_cfg** (*dict | None*) – Config of conv layers. Default: None.
- **norm_cfg** (*dict | None*) – Config of norm layers. Default: dict(type='BN').

- **act_cfg** (*dict*) – Config of activation layers. Default: dict(type='ReLU').
- **init_cfg** (*dict or list[dict], optional*) – Initialization config dict. Default: None.

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.

---

class mmseg.models.backbones.**CGNet**(*in_channels=3*, *num_channels=(32, 64, 128)*, *num_blocks=(3, 21)*, *dilations=(2, 4)*, *reductions=(8, 16)*, *conv_cfg=None*, *norm_cfg={'requires_grad': True, 'type': 'BN'}*, *act_cfg={'type': 'PReLU'}*, *norm_eval=False*, *with_cp=False*, *pretrained=None*, *init_cfg=None*)

CGNet backbone.

This backbone is the implementation of A Light-weight Context Guided Network for Semantic Segmentation.

Parameters

- **in_channels** (*int*) – Number of input image channels. Normally 3.
- **num_channels** (*tuple[int]*) – Numbers of feature channels at each stages. Default: (32, 64, 128).
- **num_blocks** (*tuple[int]*) – Numbers of CG blocks at stage 1 and stage 2. Default: (3, 21).
- **dilations** (*tuple[int]*) – Dilation rate for surrounding context extractors at stage 1 and stage 2. Default: (2, 4).
- **reductions** (*tuple[int]*) – Reductions for global context extractors at stage 1 and stage 2. Default: (8, 16).
- **conv_cfg** (*dict*) – Config dict for convolution layer. Default: None, which means using conv2d.
- **norm_cfg** (*dict*) – Config dict for normalization layer. Default: dict(type='BN', requires_grad=True).
- **act_cfg** (*dict*) – Config dict for activation layer. Default: dict(type='PReLU').
- **norm_eval** (*bool*) – Whether to set norm layers to eval mode, namely, freeze running stats (mean and var). Note: Effect on Batch Norm and its variants only. Default: False.
- **with_cp** (*bool*) – Use checkpoint or not. Using checkpoint will save some memory while slowing down the training speed. Default: False.
- **pretrained** (*str, optional*) – model pretrained path. Default: None
- **init_cfg** (*dict or list[dict], optional*) – Initialization config dict. Default: None

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.

train(*mode=True*)

Convert the model into training mode will keeping the normalization layer freezed.

class mmseg.models.backbones.**ERFNet**(*in_channels=3*, *enc_downsample_channels=(16, 64, 128)*, *enc_stage_non_bottlenecks=(5, 8)*, *enc_non_bottleneck_dilations=(2, 4, 8, 16)*, *enc_non_bottleneck_channels=(64, 128)*, *dec_upsample_channels=(64, 16)*, *dec_stages_non_bottleneck=(2, 2)*, *dec_non_bottleneck_channels=(64, 16)*, *dropout_ratio=0.1*, *conv_cfg=None*, *norm_cfg={'requires_grad': True, 'type': 'BN'}*, *act_cfg={'type': 'ReLU'}*, *init_cfg=None*)

ERFNet backbone.

This backbone is the implementation of ERFNet: Efficient Residual Factorized ConvNet for Real-time SemanticSegmentation.

Parameters

- **in_channels** (*int*) – The number of channels of input image. Default: 3.
- **enc_downsample_channels** (*Tuple[int]*) – Size of channel numbers of various Downsampler block in encoder. Default: (16, 64, 128).
- **enc_stage_non_bottlenecks** (*Tuple[int]*) – Number of stages of Non-bottleneck block in encoder. Default: (5, 8).
- **enc_non_bottleneck_dilations** (*Tuple[int]*) – Dilation rate of each stage of Non-bottleneck block of encoder. Default: (2, 4, 8, 16).
- **enc_non_bottleneck_channels** (*Tuple[int]*) – Size of channel numbers of various Non-bottleneck block in encoder. Default: (64, 128).
- **dec_upsample_channels** (*Tuple[int]*) – Size of channel numbers of various Deconvolution block in decoder. Default: (64, 16).
- **dec_stages_non_bottleneck** (*Tuple[int]*) – Number of stages of Non-bottleneck block in decoder. Default: (2, 2).
- **dec_non_bottleneck_channels** (*Tuple[int]*) – Size of channel numbers of various Non-bottleneck block in decoder. Default: (64, 16).
- **drop_rate** (*float*) – Probability of an element to be zeroed. Default 0.1.

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.

class mmseg.models.backbones.**FastSCNN**(*in_channels=3*, *downsample_dw_channels=(32, 48)*, *global_in_channels=64*, *global_block_channels=(64, 96, 128)*, *global_block_strides=(2, 2, 1)*, *global_out_channels=128*, *higher_in_channels=64*, *lower_in_channels=128*, *fusion_out_channels=128*, *out_indices=(0, 1, 2)*, *conv_cfg=None*, *norm_cfg={'type': 'BN'}*, *act_cfg={'type': 'ReLU'}*, *align_corners=False*, *dw_act_cfg=None*, *init_cfg=None*)

Fast-SCNN Backbone.

This backbone is the implementation of Fast-SCNN: Fast Semantic Segmentation Network.

Parameters

- **in_channels** (*int*) – Number of input image channels. Default: 3.
- **downsample_dw_channels** (*tuple[int]*) – Number of output channels after the first conv layer & the second conv layer in Learning-To-Downsample (LTD) module. Default: (32, 48).
- **global_in_channels** (*int*) – Number of input channels of Global Feature Extractor(GFE). Equal to number of output channels of LTD. Default: 64.
- **global_block_channels** (*tuple[int]*) – Tuple of integers that describe the output channels for each of the MobileNet-v2 bottleneck residual blocks in GFE. Default: (64, 96, 128).
- **global_block_strides** (*tuple[int]*) – Tuple of integers that describe the strides (downsampling factors) for each of the MobileNet-v2 bottleneck residual blocks in GFE. Default: (2, 2, 1).
- **global_out_channels** (*int*) – Number of output channels of GFE. Default: 128.
- **higher_in_channels** (*int*) – Number of input channels of the higher resolution branch in FFM. Equal to global_in_channels. Default: 64.
- **lower_in_channels** (*int*) – Number of input channels of the lower resolution branch in FFM. Equal to global_out_channels. Default: 128.
- **fusion_out_channels** (*int*) – Number of output channels of FFM. Default: 128.
- **out_indices** (*tuple*) – Tuple of indices of list [higher_res_features, lower_res_features, fusion_output]. Often set to (0,1,2) to enable aux. heads. Default: (0, 1, 2).
- **conv_cfg** (*dict | None*) – Config of conv layers. Default: None
- **norm_cfg** (*dict | None*) – Config of norm layers. Default: dict(type='BN')
- **act_cfg** (*dict*) – Config of activation layers. Default: dict(type='ReLU')
- **align_corners** (*bool*) – align_corners argument of F.interpolate. Default: False
- **dw_act_cfg** (*dict*) – In DepthwiseSeparableConvModule, activation config of depthwise ConvModule. If it is 'default', it will be the same as *act_cfg*. Default: None.
- **init_cfg** (*dict or list[dict], optional*) – Initialization config dict. Default: None

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.

---

class mmseg.models.backbones.**HRNet**(*extra*, *in_channels=3*, *conv_cfg=None*, *norm_cfg={'requires_grad': True, 'type': 'BN'}*, *norm_eval=False*, *with_cp=False*, *frozen_stages=-1*, *zero_init_residual=False*, *multiscale_output=True*, *pretrained=None*, *init_cfg=None*)

HRNet backbone.

This backbone is the implementation of High-Resolution Representations for Labeling Pixels and Regions.

Parameters

- **extra** (*dict*) – Detailed configuration for each stage of HRNet. There must be 4 stages, the configuration for each stage must have 5 keys:
  - num_modules (int): The number of HRModule in this stage.
  - num_branches (int): The number of branches in the HRModule.
  - block (str): The type of convolution block.
  - **num_blocks (tuple): The number of blocks in each branch.** The length must be equal to num_branches.
  - **num_channels (tuple): The number of channels in each branch.** The length must be equal to num_branches.
- **in_channels** (*int*) – Number of input image channels. Normally 3.
- **conv_cfg** (*dict*) – Dictionary to construct and config conv layer. Default: None.
- **norm_cfg** (*dict*) – Dictionary to construct and config norm layer. Use *BN* by default.
- **norm_eval** (*bool*) – Whether to set norm layers to eval mode, namely, freeze running stats (mean and var). Note: Effect on Batch Norm and its variants only. Default: False.
- **with_cp** (*bool*) – Use checkpoint or not. Using checkpoint will save some memory while slowing down the training speed. Default: False.
- **frozen_stages** (*int*) – Stages to be frozen (stop grad and set eval mode). -1 means not freezing any parameters. Default: -1.
- **zero_init_residual** (*bool*) – Whether to use zero init for last norm layer in resblocks to let them behave as identity. Default: False.
- **multiscale_output** (*bool*) – Whether to output multi-level features produced by multiple branches. If False, only the first level feature will be output. Default: True.
- **pretrained** (*str, optional*) – Model pretrained path. Default: None.
- **init_cfg** (*dict or list[dict], optional*) – Initialization config dict. Default: None.

Example

```
>>> from mmseg.models import HRNet
>>> import torch
>>> extra = dict(
>>>     stage1=dict(
>>>         num_modules=1,
>>>         num_branches=1,
>>>         block='BOTTLENECK',
>>>         num_blocks=(4, ),
```

(continues on next page)

(continued from previous page)

```
>>>         num_channels=(64, )),
>>>     stage2=dict(
>>>         num_modules=1,
>>>         num_branches=2,
>>>         block='BASIC',
>>>         num_blocks=(4, 4),
>>>         num_channels=(32, 64)),
>>>     stage3=dict(
>>>         num_modules=4,
>>>         num_branches=3,
>>>         block='BASIC',
>>>         num_blocks=(4, 4, 4),
>>>         num_channels=(32, 64, 128)),
>>>     stage4=dict(
>>>         num_modules=3,
>>>         num_branches=4,
>>>         block='BASIC',
>>>         num_blocks=(4, 4, 4, 4),
>>>         num_channels=(32, 64, 128, 256)))
>>> self = HRNet(extra, in_channels=1)
>>> self.eval()
>>> inputs = torch.rand(1, 1, 32, 32)
>>> level_outputs = self.forward(inputs)
>>> for level_out in level_outputs:
...     print(tuple(level_out.shape))
(1, 32, 8, 8)
(1, 64, 4, 4)
(1, 128, 2, 2)
(1, 256, 1, 1)
```

forward(*x*)
Forward function.

property norm1
the normalization layer named “norm1”

Type nn.Module

property norm2
the normalization layer named “norm2”

Type nn.Module

train(*mode=True*)
Convert the model into training mode will keeping the normalization layer freezed.

class mmseg.models.backbones.**ICNet**(*backbone_cfg*, *in_channels=3*, *layer_channels=(512, 2048)*, *light_branch_middle_channels=32*, *psp_out_channels=512*, *out_channels=(64, 256, 256)*, *pool_scales=(1, 2, 3, 6)*, *conv_cfg=None*, *norm_cfg={'requires_grad': True, 'type': 'BN'}*, *act_cfg={'type': 'ReLU'}*, *align_corners=False*, *init_cfg=None*)
ICNet for Real-Time Semantic Segmentation on High-Resolution Images.

This backbone is the implementation of ICNet.

Parameters

- **backbone_cfg** (*dict*) – Config dict to build backbone. Usually it is ResNet but it can also be other backbones.
- **in_channels** (*int*) – The number of input image channels. Default: 3.
- **layer_channels** (*Sequence[int]*) – The numbers of feature channels at layer 2 and layer 4 in ResNet. It can also be other backbones. Default: (512, 2048).
- **light_branch_middle_channels** (*int*) – The number of channels of the middle layer in light branch. Default: 32.
- **psp_out_channels** (*int*) – The number of channels of the output of PSP module. Default: 512.
- **out_channels** (*Sequence[int]*) – The numbers of output feature channels at each branches. Default: (64, 256, 256).
- **pool_scales** (*tuple[int]*) – Pooling scales used in Pooling Pyramid Module. Default: (1, 2, 3, 6).
- **conv_cfg** (*dict*) – Dictionary to construct and config conv layer. Default: None.
- **norm_cfg** (*dict*) – Dictionary to construct and config norm layer. Default: dict(type='BN').
- **act_cfg** (*dict*) – Dictionary to construct and config act layer. Default: dict(type='ReLU').
- **align_corners** (*bool*) – align_corners argument of F.interpolate. Default: False.
- **init_cfg** (*dict or list[dict], optional*) – Initialization config dict. Default: None.

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.

---

class mmseg.models.backbones.**MAE**(*img_size=224*, *patch_size=16*, *in_channels=3*, *embed_dims=768*, *num_layers=12*, *num_heads=12*, *mlp_ratio=4*, *out_indices=- 1*, *attn_drop_rate=0.0*, *drop_path_rate=0.0*, *norm_cfg={'type': 'LN'}*, *act_cfg={'type': 'GELU'}*, *patch_norm=False*, *final_norm=False*, *num_fcs=2*, *norm_eval=False*, *pretrained=None*, *init_values=0.1*, *init_cfg=None*)

VisionTransformer with support for patch.

Parameters

- **img_size** (*int | tuple*) – Input image size. Default: 224.
- **patch_size** (*int*) – The patch size. Default: 16.
- **in_channels** (*int*) – Number of input channels. Default: 3.
- **embed_dims** (*int*) – embedding dimension. Default: 768.
- **num_layers** (*int*) – depth of transformer. Default: 12.
- **num_heads** (*int*) – number of attention heads. Default: 12.

- **mlp_ratio** (*int*) – ratio of mlp hidden dim to embedding dim. Default: 4.
- **out_indices** (*list | tuple | int*) – Output from which stages. Default: -1.
- **attn_drop_rate** (*float*) – The drop out rate for attention layer. Default 0.0
- **drop_path_rate** (*float*) – stochastic depth rate. Default 0.0.
- **norm_cfg** (*dict*) – Config dict for normalization layer. Default: dict(type='LN')
- **act_cfg** (*dict*) – The activation config for FFNs. Default: dict(type='GELU').
- **patch_norm** (*bool*) – Whether to add a norm in PatchEmbed Block. Default: False.
- **final_norm** (*bool*) – Whether to add a additional layer to normalize final feature map. Default: False.
- **num_fcs** (*int*) – The number of fully-connected layers for FFNs. Default: 2.
- **norm_eval** (*bool*) – Whether to set norm layers to eval mode, namely, freeze running stats (mean and var). Note: Effect on Batch Norm and its variants only. Default: False.
- **pretrained** (*str, optional*) – model pretrained path. Default: None.
- **init_values** (*float*) – Initialize the values of Attention and FFN with learnable scaling. Defaults to 0.1.
- **init_cfg** (*dict or list[dict], optional*) – Initialization config dict. Default: None.

fix_init_weight()
Rescale the initialization according to layer id.

This function is copied from https://github.com/microsoft/unilm/blob/master/beit/modeling_pretrain.py. # noqa: E501 Copyright (c) Microsoft Corporation Licensed under the MIT License

forward(*inputs*)
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()
Initialize the weights.

class mmseg.models.backbones.**MixVisionTransformer**(*in_channels=3*, *embed_dims=64*, *num_stages=4*, *num_layers=[3, 4, 6, 3]*, *num_heads=[1, 2, 4, 8]*, *patch_sizes=[7, 3, 3, 3]*, *strides=[4, 2, 2, 2]*, *sr_ratios=[8, 4, 2, 1]*, *out_indices=(0, 1, 2, 3)*, *mlp_ratio=4*, *qkv_bias=True*, *drop_rate=0.0*, *attn_drop_rate=0.0*, *drop_path_rate=0.0*, *act_cfg={'type': 'GELU'}*, *norm_cfg={'eps': 1e-06, 'type': 'LN'}*, *pretrained=None*, *init_cfg=None*, *with_cp=False*)

The backbone of Segformer.

This backbone is the implementation of SegFormer: Simple and Efficient Design for Semantic Segmentation with Transformers. :param in_channels: Number of input channels. Default: 3. :type in_channels: int :param

embed_dims: Embedding dimension. Default: 768. :type embed_dims: int :param num_stags: The num of stages. Default: 4. :type num_stags: int :param num_layers: The layer number of each transformer encode

layer. Default: [3, 4, 6, 3].

Parameters

- **num_heads** (*Sequence[int]*) – The attention heads of each transformer encode layer. Default: [1, 2, 4, 8].
- **patch_sizes** (*Sequence[int]*) – The patch_size of each overlapped patch embedding. Default: [7, 3, 3, 3].
- **strides** (*Sequence[int]*) – The stride of each overlapped patch embedding. Default: [4, 2, 2, 2].
- **sr_ratios** (*Sequence[int]*) – The spatial reduction rate of each transformer encode layer. Default: [8, 4, 2, 1].
- **out_indices** (*Sequence[int] | int*) – Output from which stages. Default: (0, 1, 2, 3).
- **mlp_ratio** (*int*) – ratio of mlp hidden dim to embedding dim. Default: 4.
- **qkv_bias** (*bool*) – Enable bias for qkv if True. Default: True.
- **drop_rate** (*float*) – Probability of an element to be zeroed. Default 0.0
- **attn_drop_rate** (*float*) – The drop out rate for attention layer. Default 0.0
- **drop_path_rate** (*float*) – stochastic depth rate. Default 0.0
- **norm_cfg** (*dict*) – Config dict for normalization layer. Default: dict(type='LN')
- **act_cfg** (*dict*) – The activation config for FFNs. Default: dict(type='GELU').
- **pretrained** (*str, optional*) – model pretrained path. Default: None.
- **init_cfg** (*dict or list[dict], optional*) – Initialization config dict. Default: None.
- **with_cp** (*bool*) – Use checkpoint or not. Using checkpoint will save some memory while slowing down the training speed. Default: False.

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.

---

init_weights()

Initialize the weights.

class mmseg.models.backbones.**MobileNetV2**(*widen_factor=1.0*, *strides=(1, 2, 2, 2, 1, 2, 1)*, *dilations=(1, 1, 1, 1, 1, 1, 1)*, *out_indices=(1, 2, 4, 6)*, *frozen_stages=- 1*, *conv_cfg=None*, *norm_cfg={'type': 'BN'}*, *act_cfg={'type': 'ReLU6'}*, *norm_eval=False*, *with_cp=False*, *pretrained=None*, *init_cfg=None*)

MobileNetV2 backbone.

This backbone is the implementation of MobileNetV2: Inverted Residuals and Linear Bottlenecks.

Parameters

- **widen_factor** (*float*) – Width multiplier, multiply number of channels in each layer by this amount. Default: 1.0.
- **strides** (*Sequence[int], optional*) – Strides of the first block of each layer. If not specified, default config in `arch_setting` will be used.
- **dilations** (*Sequence[int]*) – Dilation of each layer.
- **out_indices** (*None or Sequence[int]*) – Output from which stages. Default: (7,).
- **frozen_stages** (*int*) – Stages to be frozen (all param fixed). Default: -1, which means not freezing any parameters.
- **conv_cfg** (*dict*) – Config dict for convolution layer. Default: None, which means using conv2d.
- **norm_cfg** (*dict*) – Config dict for normalization layer. Default: dict(type='BN').
- **act_cfg** (*dict*) – Config dict for activation layer. Default: dict(type='ReLU6').
- **norm_eval** (*bool*) – Whether to set norm layers to eval mode, namely, freeze running stats (mean and var). Note: Effect on Batch Norm and its variants only. Default: False.
- **with_cp** (*bool*) – Use checkpoint or not. Using checkpoint will save some memory while slowing down the training speed. Default: False.
- **pretrained** (*str, optional*) – model pretrained path. Default: None
- **init_cfg** (*dict or list[dict], optional*) – Initialization config dict. Default: None

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.

---

make_layer(*out_channels*, *num_blocks*, *stride*, *dilation*, *expand_ratio*)

Stack InvertedResidual blocks to build a layer for MobileNetV2.

Parameters

- **out_channels** (*int*) – out_channels of block.
- **num_blocks** (*int*) – Number of blocks.
- **stride** (*int*) – Stride of the first block.
- **dilation** (*int*) – Dilation of the first block.
- **expand_ratio** (*int*) – Expand the number of channels of the hidden layer in InvertedResidual by this ratio.

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 mmseg.models.backbones.**MobileNetV3**(*arch='small'*, *conv_cfg=None*, *norm_cfg={'type': 'BN'}*, *out_indices=(0, 1, 12)*, *frozen_stages=- 1*, *reduction_factor=1*, *norm_eval=False*, *with_cp=False*, *pretrained=None*, *init_cfg=None*)

MobileNetV3 backbone.

This backbone is the improved implementation of Searching for MobileNetV3.

Parameters

- **arch** (*str*) – Architecture of mobilnetv3, from {'small', 'large'}. Default: 'small'.
- **conv_cfg** (*dict*) – Config dict for convolution layer. Default: None, which means using conv2d.
- **norm_cfg** (*dict*) – Config dict for normalization layer. Default: dict(type='BN').
- **out_indices** (*tuple[int]*) – Output from which layer. Default: (0, 1, 12).
- **frozen_stages** (*int*) – Stages to be frozen (all param fixed). Default: -1, which means not freezing any parameters.
- **norm_eval** (*bool*) – Whether to set norm layers to eval mode, namely, freeze running stats (mean and var). Note: Effect on Batch Norm and its variants only. Default: False.
- **with_cp** (*bool*) – Use checkpoint or not. Using checkpoint will save some memory while slowing down the training speed. Default: False.
- **pretrained** (*str, optional*) – model pretrained path. Default: None
- **init_cfg** (*dict or list[dict], optional*) – Initialization config dict. Default: None

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.

---

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 mmseg.models.backbones.**PCPVT**(*in_channels=3*, *embed_dims=[64, 128, 256, 512]*, *patch_sizes=[4, 2, 2, 2]*, *strides=[4, 2, 2, 2]*, *num_heads=[1, 2, 4, 8]*, *mlp_ratios=[4, 4, 4, 4]*, *out_indices=(0, 1, 2, 3)*, *qkv_bias=False*, *drop_rate=0.0*, *attn_drop_rate=0.0*, *drop_path_rate=0.0*, *norm_cfg={'type': 'LN'}*, *depths=[3, 4, 6, 3]*, *sr_ratios=[8, 4, 2, 1]*, *norm_after_stage=False*, *pretrained=None*, *init_cfg=None*)

The backbone of Twins-PCPVT.

This backbone is the implementation of Twins: Revisiting the Design of Spatial Attention in Vision Transformers.

Parameters

- **in_channels** (*int*) – Number of input channels. Default: 3.
- **embed_dims** (*list*) – Embedding dimension. Default: [64, 128, 256, 512].
- **patch_sizes** (*list*) – The patch sizes. Default: [4, 2, 2, 2].
- **strides** (*list*) – The strides. Default: [4, 2, 2, 2].
- **num_heads** (*int*) – Number of attention heads. Default: [1, 2, 4, 8].
- **mlp_ratios** (*int*) – Ratio of mlp hidden dim to embedding dim. Default: [4, 4, 4, 4].
- **out_indices** (*tuple[int]*) – Output from which stages. Default: (0, 1, 2, 3).
- **qkv_bias** (*bool*) – Enable bias for qkv if True. Default: False.
- **drop_rate** (*float*) – Probability of an element to be zeroed. Default 0.
- **attn_drop_rate** (*float*) – The drop out rate for attention layer. Default 0.0
- **drop_path_rate** (*float*) – Stochastic depth rate. Default 0.0
- **norm_cfg** (*dict*) – Config dict for normalization layer. Default: dict(type='LN')
- **depths** (*list*) – Depths of each stage. Default [3, 4, 6, 3]
- **sr_ratios** (*list*) – Kernel_size of conv in each Attn module in Transformer encoder layer. Default: [8, 4, 2, 1].
- **norm_after_stagebool)** – Add extra norm. Default False.
- **init_cfg** (*dict, optional*) – The Config for initialization. Defaults to None.

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.

---

init_weights()

Initialize the weights.

class mmseg.models.backbones.**ResNeSt**(*groups=1*, *base_width=4*, *radix=2*, *reduction_factor=4*, *avg_down_stride=True*, ***kwargs*)

ResNeSt backbone.

This backbone is the implementation of ResNeSt: Split-Attention Networks.

Parameters

- **groups** (*int*) – Number of groups of Bottleneck. Default: 1
- **base_width** (*int*) – Base width of Bottleneck. Default: 4
- **radix** (*int*) – Radix of SpltAtConv2d. Default: 2
- **reduction_factor** (*int*) – Reduction factor of inter_channels in SplitAttentionConv2d. Default: 4.
- **avg_down_stride** (*bool*) – Whether to use average pool for stride in Bottleneck. Default: True.
- **kwargs** (*dict*) – Keyword arguments for ResNet.

make_res_layer(***kwargs*)
Pack all blocks in a stage into a ResLayer.

class mmseg.models.backbones.**ResNeXt**(*groups=1*, *base_width=4*, ***kwargs*)
ResNeXt backbone.

This backbone is the implementation of Aggregated Residual Transformations for Deep Neural Networks.

Parameters

- **depth** (*int*) – Depth of resnet, from {18, 34, 50, 101, 152}.
- **in_channels** (*int*) – Number of input image channels. Normally 3.
- **num_stages** (*int*) – Resnet stages, normally 4.
- **groups** (*int*) – Group of resnext.
- **base_width** (*int*) – Base width of resnext.
- **strides** (*Sequence[int]*) – Strides of the first block of each stage.
- **dilations** (*Sequence[int]*) – Dilation of each stage.
- **out_indices** (*Sequence[int]*) – Output from which stages.
- **style** (*str*) – *pytorch* or *caffe*. If set to "pytorch", the stride-two layer is the 3x3 conv layer, otherwise the stride-two layer is the first 1x1 conv layer.
- **frozen_stages** (*int*) – Stages to be frozen (all param fixed). -1 means not freezing any parameters.
- **norm_cfg** (*dict*) – dictionary to construct and config norm layer.
- **norm_eval** (*bool*) – Whether to set norm layers to eval mode, namely, freeze running stats (mean and var). Note: Effect on Batch Norm and its variants only.
- **with_cp** (*bool*) – Use checkpoint or not. Using checkpoint will save some memory while slowing down the training speed.
- **zero_init_residual** (*bool*) – whether to use zero init for last norm layer in resblocks to let them behave as identity.

Example

```
>>> from mmseg.models import ResNeXt
>>> import torch
>>> self = ResNeXt(depth=50)
>>> self.eval()
>>> inputs = torch.rand(1, 3, 32, 32)
>>> level_outputs = self.forward(inputs)
>>> for level_out in level_outputs:
...     print(tuple(level_out.shape))
(1, 256, 8, 8)
(1, 512, 4, 4)
(1, 1024, 2, 2)
(1, 2048, 1, 1)
```

make_res_layer(***kwargs*)
Pack all blocks in a stage into a ResLayer

class mmseg.models.backbones.**ResNet**(*depth*, *in_channels=3*, *stem_channels=64*, *base_channels=64*, *num_stages=4*, *strides=(1, 2, 2, 2)*, *dilations=(1, 1, 1, 1)*, *out_indices=(0, 1, 2, 3)*, *style='pytorch'*, *deep_stem=False*, *avg_down=False*, *frozen_stages=- 1*, *conv_cfg=None*, *norm_cfg={'requires_grad': True, 'type': 'BN'}*, *norm_eval=False*, *dcn=None*, *stage_with_dcn=(False, False, False, False)*, *plugins=None*, *multi_grid=None*, *contract_dilation=False*, *with_cp=False*, *zero_init_residual=True*, *pretrained=None*, *init_cfg=None*)

ResNet backbone.

This backbone is the improved implementation of Deep Residual Learning for Image Recognition.

Parameters

- **depth** (*int*) – Depth of resnet, from {18, 34, 50, 101, 152}.
- **in_channels** (*int*) – Number of input image channels. Default: 3.
- **stem_channels** (*int*) – Number of stem channels. Default: 64.
- **base_channels** (*int*) – Number of base channels of res layer. Default: 64.
- **num_stages** (*int*) – Resnet stages, normally 4. Default: 4.
- **strides** (*Sequence[int]*) – Strides of the first block of each stage. Default: (1, 2, 2, 2).
- **dilations** (*Sequence[int]*) – Dilation of each stage. Default: (1, 1, 1, 1).
- **out_indices** (*Sequence[int]*) – Output from which stages. Default: (0, 1, 2, 3).
- **style** (*str*) – *pytorch* or *caffe*. If set to "pytorch", the stride-two layer is the 3x3 conv layer, otherwise the stride-two layer is the first 1x1 conv layer. Default: 'pytorch'.
- **deep_stem** (*bool*) – Replace 7x7 conv in input stem with 3 3x3 conv. Default: False.
- **avg_down** (*bool*) – Use AvgPool instead of stride conv when downsampling in the bottleneck. Default: False.
- **frozen_stages** (*int*) – Stages to be frozen (stop grad and set eval mode). -1 means not freezing any parameters. Default: -1.
- **conv_cfg** (*dict | None*) – Dictionary to construct and config conv layer. When conv_cfg is None, cfg will be set to dict(type='Conv2d'). Default: None.

- **norm_cfg** (*dict*) – Dictionary to construct and config norm layer. Default: dict(type='BN', requires_grad=True).
- **norm_eval** (*bool*) – Whether to set norm layers to eval mode, namely, freeze running stats (mean and var). Note: Effect on Batch Norm and its variants only. Default: False.
- **dcn** (*dict | None*) – Dictionary to construct and config DCN conv layer. When dcn is not None, conv_cfg must be None. Default: None.
- **stage_with_dcn** (*Sequence[bool]*) – Whether to set DCN conv for each stage. The length of stage_with_dcn is equal to num_stages. Default: (False, False, False, False).
- **plugins** (*list[dict]*) – List of plugins for stages, each dict contains:
  - cfg (dict, required): Cfg dict to build plugin.
  - position (str, required): Position inside block to insert plugin,

  options: 'after_conv1', 'after_conv2', 'after_conv3'.
  - stages (tuple[bool], optional): Stages to apply plugin, length

  should be same as 'num_stages'. Default: None.
- **multi_grid** (*Sequence[int]|None*) – Multi grid dilation rates of last stage. Default: None.
- **contract_dilation** (*bool*) – Whether contract first dilation of each layer Default: False.
- **with_cp** (*bool*) – Use checkpoint or not. Using checkpoint will save some memory while slowing down the training speed. Default: False.
- **zero_init_residual** (*bool*) – Whether to use zero init for last norm layer in resblocks to let them behave as identity. Default: True.
- **pretrained** (*str, optional*) – model pretrained path. Default: None.
- **init_cfg** (*dict or list[dict], optional*) – Initialization config dict. Default: None.

Example

```
>>> from mmseg.models import ResNet
>>> import torch
>>> self = ResNet(depth=18)
>>> self.eval()
>>> inputs = torch.rand(1, 3, 32, 32)
>>> level_outputs = self.forward(inputs)
>>> for level_out in level_outputs:
...     print(tuple(level_out.shape))
(1, 64, 8, 8)
(1, 128, 4, 4)
(1, 256, 2, 2)
(1, 512, 1, 1)
```

forward(*x*)
Forward function.

make_res_layer(***kwargs*)
Pack all blocks in a stage into a `ResLayer`.

make_stage_plugins(*plugins*, *stage_idx*)
make plugins for ResNet 'stage_idx'th stage .

Currently we support to insert 'context_block', 'empirical_attention_block', 'nonlocal_block' into the backbone like ResNet/ResNeXt. They could be inserted after conv1/conv2/conv3 of Bottleneck.

An example of plugins format could be : >>> plugins=[... dict(cfg=dict(type='xxx', arg1='xxx'), ... stages=(False, True, True, True), ... position='after_conv2'), ... dict(cfg=dict(type='yyy'), ... stages=(True, True, True, True), ... position='after_conv3'), ... dict(cfg=dict(type='zzz', postfix='1'), ... stages=(True, True, True, True), ... position='after_conv3'), ... dict(cfg=dict(type='zzz', postfix='2'), ... stages=(True, True, True, True), ... position='after_conv3') ...] >>> self = ResNet(depth=18) >>> stage_plugins = self.make_stage_plugins(plugins, 0) >>> assert len(stage_plugins) == 3

Suppose 'stage_idx=0', the structure of blocks in the stage would be: conv1-> conv2->conv3->yyy->zzz1->zzz2

Suppose 'stage_idx=1', the structure of blocks in the stage would be: conv1-> conv2->xxx->conv3->yyy->zzz1->zzz2

If stages is missing, the plugin would be applied to all stages.

Parameters

- **plugins** (*list[dict]*) – List of plugins cfg to build. The postfix is required if multiple same type plugins are inserted.
- **stage_idx** (*int*) – Index of stage to build

Returns Plugins for current stage

Return type list[dict]

property norm1
the normalization layer named "norm1"

Type nn.Module

train(*mode=True*)
Convert the model into training mode while keep normalization layer freezed.

class mmseg.models.backbones.**ResNetV1c**(***kwargs*)
ResNetV1c variant described in [1]_.

Compared with default ResNet(ResNetV1b), ResNetV1c replaces the 7x7 conv in the input stem with three 3x3 convs. For more details please refer to Bag of Tricks for Image Classification with Convolutional Neural Networks.

class mmseg.models.backbones.**ResNetV1d**(***kwargs*)
ResNetV1d variant described in [1]_.

Compared with default ResNet(ResNetV1b), ResNetV1d replaces the 7x7 conv in the input stem with three 3x3 convs. And in the downsampling block, a 2x2 avg_pool with stride 2 is added before conv, whose stride is changed to 1.

class mmseg.models.backbones.**STDCContextPathNet**(*backbone_cfg*, *last_in_channels=(1024, 512)*, *out_channels=128*, *ffm_cfg={'in_channels': 512, 'out_channels': 256, 'scale_factor': 4}*, *upsample_mode='nearest'*, *align_corners=None*, *norm_cfg={'type': 'BN'}*, *init_cfg=None*)
STDCNet with Context Path. The *outs* below is a list of three feature maps from deep to shallow, whose height and width is from small to big, respectively. The biggest feature map of *outs* is outputted for *STDCHead*, where Detail Loss would be calculated by Detail Ground-truth. The other two feature maps are used for Attention Refinement Module, respectively. Besides, the biggest feature map of *outs* and the last output of Attention

Refinement Module are concatenated for Feature Fusion Module. Then, this fusion feature map *feat_fuse* would be outputted for *decode_head*. More details please refer to Figure 4 of original paper.

Parameters

- **backbone_cfg** (*dict*) – Config dict for stdc backbone.
- **last_in_channels** (*tuple(int)*) – two feature maps from stdc backbone. Default: (1024, 512).
- **out_channels** (*int*) – The channels of output feature maps. Default: 128.
- **ffm_cfg** (*dict*) – Config dict for Feature Fusion Module. Default: *dict(in_channels=512, out_channels=256, scale_factor=4)*.
- **upsample_mode** (*str*) – Algorithm used for upsampling: `'nearest' | 'linear' | 'bilinear' | 'bicubic' | 'trilinear'`. Default: `'nearest'`.
- **align_corners** (*str*) – align_corners argument of F.interpolate. It must be *None* if upsample_mode is `'nearest'`. Default: None.
- **norm_cfg** (*dict*) – Config dict for normalization layer. Default: dict(type='BN').
- **init_cfg** (*dict or list[dict], optional*) – Initialization config dict. Default: None.

Returns

The tuple of list of output feature map for auxiliary heads and decoder head.

Return type outputs (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.

---

class mmseg.models.backbones.**STDCNet**(*stdc_type*, *in_channels*, *channels*, *bottleneck_type*, *norm_cfg*, *act_cfg*, *num_convs=4*, *with_final_conv=False*, *pretrained=None*, *init_cfg=None*)

This backbone is the implementation of Rethinking BiSeNet For Real-time Semantic Segmentation.

Parameters

- **stdc_type** (*int*) – The type of backbone structure, *STDCNet1* and`STDCNet2` denotes two main backbones in paper, whose FLOPs is 813M and 1446M, respectively.
- **in_channels** (*int*) – The num of input_channels.
- **channels** (*tuple[int]*) – The output channels for each stage.
- **bottleneck_type** (*str*) – The type of STDC Module type, the value must be 'add' or 'cat'.
- **norm_cfg** (*dict*) – Config dict for normalization layer.
- **act_cfg** (*dict*) – The activation config for conv layers.
- **num_convs** (*int*) – Numbers of conv layer at each STDC Module. Default: 4.
- **with_final_conv** (*bool*) – Whether add a conv layer at the Module output. Default: True.

- **pretrained** (*str, optional*) – Model pretrained path. Default: None.
- **init_cfg** (*dict or list[dict], optional*) – Initialization config dict. Default: None.

Example

```
>>> import torch
>>> stdc_type = 'STDCNet1'
>>> in_channels = 3
>>> channels = (32, 64, 256, 512, 1024)
>>> bottleneck_type = 'cat'
>>> inputs = torch.rand(1, 3, 1024, 2048)
>>> self = STDCNet(stdc_type, in_channels,
...                channels, bottleneck_type).eval()
>>> outputs = self.forward(inputs)
>>> for i in range(len(outputs)):
...     print(f'outputs[{i}].shape = {outputs[i].shape}')
outputs[0].shape = torch.Size([1, 256, 128, 256])
outputs[1].shape = torch.Size([1, 512, 64, 128])
outputs[2].shape = torch.Size([1, 1024, 32, 64])
```

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.

---

class mmseg.models.backbones.**SVT**(*in_channels=3*, *embed_dims=[64, 128, 256]*, *patch_sizes=[4, 2, 2, 2]*, *strides=[4, 2, 2, 2]*, *num_heads=[1, 2, 4]*, *mlp_ratios=[4, 4, 4]*, *out_indices=(0, 1, 2, 3)*, *qkv_bias=False*, *drop_rate=0.0*, *attn_drop_rate=0.0*, *drop_path_rate=0.2*, *norm_cfg={'type': 'LN'}*, *depths=[4, 4, 4]*, *sr_ratios=[4, 2, 1]*, *windiow_sizes=[7, 7, 7]*, *norm_after_stage=True*, *pretrained=None*, *init_cfg=None*)

The backbone of Twins-SVT.

This backbone is the implementation of Twins: Revisiting the Design of Spatial Attention in Vision Transformers.

Parameters

- **in_channels** (*int*) – Number of input channels. Default: 3.
- **embed_dims** (*list*) – Embedding dimension. Default: [64, 128, 256, 512].
- **patch_sizes** (*list*) – The patch sizes. Default: [4, 2, 2, 2].
- **strides** (*list*) – The strides. Default: [4, 2, 2, 2].
- **num_heads** (*int*) – Number of attention heads. Default: [1, 2, 4].
- **mlp_ratios** (*int*) – Ratio of mlp hidden dim to embedding dim. Default: [4, 4, 4].
- **out_indices** (*tuple[int]*) – Output from which stages. Default: (0, 1, 2, 3).
- **qkv_bias** (*bool*) – Enable bias for qkv if True. Default: False.

- **drop_rate** (*float*) – Dropout rate. Default 0.
- **attn_drop_rate** (*float*) – Dropout ratio of attention weight. Default 0.0
- **drop_path_rate** (*float*) – Stochastic depth rate. Default 0.2.
- **norm_cfg** (*dict*) – Config dict for normalization layer. Default: dict(type='LN')
- **depths** (*list*) – Depths of each stage. Default [4, 4, 4].
- **sr_ratios** (*list*) – Kernel_size of conv in each Attn module in Transformer encoder layer. Default: [4, 2, 1].
- **windiow_sizes** (*list*) – Window size of LSA. Default: [7, 7, 7],
- **input_features_slicebool)** – Input features need slice. Default: False.
- **norm_after_stagebool)** – Add extra norm. Default False.
- **strides** – Strides in patch-Embedding modules. Default: (2, 2, 2)
- **init_cfg** (*dict, optional*) – The Config for initialization. Defaults to None.

class mmseg.models.backbones.**SwinTransformer**(*pretrain_img_size=224*, *in_channels=3*, *embed_dims=96*, *patch_size=4*, *window_size=7*, *mlp_ratio=4*, *depths=(2, 2, 6, 2)*, *num_heads=(3, 6, 12, 24)*, *strides=(4, 2, 2, 2)*, *out_indices=(0, 1, 2, 3)*, *qkv_bias=True*, *qk_scale=None*, *patch_norm=True*, *drop_rate=0.0*, *attn_drop_rate=0.0*, *drop_path_rate=0.1*, *use_abs_pos_embed=False*, *act_cfg={'type': 'GELU'}*, *norm_cfg={'type': 'LN'}*, *with_cp=False*, *pretrained=None*, *frozen_stages=- 1*, *init_cfg=None*)

Swin Transformer backbone.

This backbone is the implementation of Swin Transformer: Hierarchical Vision Transformer using Shifted Windows. Inspiration from https://github.com/microsoft/Swin-Transformer.

Parameters

- **pretrain_img_size** (*int | tuple[int]*) – The size of input image when pretrain. Defaults: 224.
- **in_channels** (*int*) – The num of input channels. Defaults: 3.
- **embed_dims** (*int*) – The feature dimension. Default: 96.
- **patch_size** (*int | tuple[int]*) – Patch size. Default: 4.
- **window_size** (*int*) – Window size. Default: 7.
- **mlp_ratio** (*int | float*) – Ratio of mlp hidden dim to embedding dim. Default: 4.
- **depths** (*tuple[int]*) – Depths of each Swin Transformer stage. Default: (2, 2, 6, 2).
- **num_heads** (*tuple[int]*) – Parallel attention heads of each Swin Transformer stage. Default: (3, 6, 12, 24).
- **strides** (*tuple[int]*) – The patch merging or patch embedding stride of each Swin Transformer stage. (In swin, we set kernel size equal to stride.) Default: (4, 2, 2, 2).
- **out_indices** (*tuple[int]*) – Output from which stages. Default: (0, 1, 2, 3).
- **qkv_bias** (*bool, optional*) – If True, add a learnable bias to query, key, value. Default: True

- **qk_scale** (*float | None, optional*) – Override default qk scale of head_dim ** -0.5 if set. Default: None.
- **patch_norm** (*bool*) – If add a norm layer for patch embed and patch merging. Default: True.
- **drop_rate** (*float*) – Dropout rate. Defaults: 0.
- **attn_drop_rate** (*float*) – Attention dropout rate. Default: 0.
- **drop_path_rate** (*float*) – Stochastic depth rate. Defaults: 0.1.
- **use_abs_pos_embed** (*bool*) – If True, add absolute position embedding to the patch embedding. Defaults: False.
- **act_cfg** (*dict*) – Config dict for activation layer. Default: dict(type='LN').
- **norm_cfg** (*dict*) – Config dict for normalization layer at output of backone. Defaults: dict(type='LN').
- **with_cp** (*bool, optional*) – Use checkpoint or not. Using checkpoint will save some memory while slowing down the training speed. Default: False.
- **pretrained** (*str, optional*) – model pretrained path. Default: None.
- **frozen_stages** (*int*) – Stages to be frozen (stop grad and set eval mode). -1 means not freezing any parameters.
- **init_cfg** (*dict, optional*) – The Config for initialization. Defaults to None.

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.

---

init_weights()

Initialize the weights.

train(*mode=True*)

Convert the model into training mode while keep layers freezed.

class mmseg.models.backbones.**TIMMBackbone**(*model_name*, *features_only=True*, *pretrained=True*, *checkpoint_path=''*, *in_channels=3*, *init_cfg=None*, ***kwargs*)

Wrapper to use backbones from timm library. More details can be found in timm .

Parameters

- **model_name** (*str*) – Name of timm model to instantiate.
- **pretrained** (*bool*) – Load pretrained weights if True.
- **checkpoint_path** (*str*) – Path of checkpoint to load after model is initialized.
- **in_channels** (*int*) – Number of input image channels. Default: 3.
- **init_cfg** (*dict, optional*) – Initialization config dict
- ****kwargs** – Other timm & model specific arguments.

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.

---

class mmseg.models.backbones.**UNet**(*in_channels=3*, *base_channels=64*, *num_stages=5*, *strides=(1, 1, 1, 1, 1)*, *enc_num_convs=(2, 2, 2, 2, 2)*, *dec_num_convs=(2, 2, 2, 2)*, *downsamples=(True, True, True, True)*, *enc_dilations=(1, 1, 1, 1, 1)*, *dec_dilations=(1, 1, 1, 1)*, *with_cp=False*, *conv_cfg=None*, *norm_cfg={'type': 'BN'}*, *act_cfg={'type': 'ReLU'}*, *upsample_cfg={'type': 'InterpConv'}*, *norm_eval=False*, *dcn=None*, *plugins=None*, *pretrained=None*, *init_cfg=None*)

UNet backbone.

This backbone is the implementation of U-Net: Convolutional Networks for Biomedical Image Segmentation.

Parameters

- **in_channels** (*int*) – Number of input image channels. Default" 3.
- **base_channels** (*int*) – Number of base channels of each stage. The output channels of the first stage. Default: 64.
- **num_stages** (*int*) – Number of stages in encoder, normally 5. Default: 5.
- **strides** (*Sequence[int 1 | 2]*) – Strides of each stage in encoder. len(strides) is equal to num_stages. Normally the stride of the first stage in encoder is 1. If strides[i]=2, it uses stride convolution to downsample in the correspondence encoder stage. Default: (1, 1, 1, 1, 1).
- **enc_num_convs** (*Sequence[int]*) – Number of convolutional layers in the convolution block of the correspondence encoder stage. Default: (2, 2, 2, 2, 2).
- **dec_num_convs** (*Sequence[int]*) – Number of convolutional layers in the convolution block of the correspondence decoder stage. Default: (2, 2, 2, 2).
- **downsamples** (*Sequence[int]*) – Whether use MaxPool to downsample the feature map after the first stage of encoder (stages: [1, num_stages)). If the correspondence encoder stage use stride convolution (strides[i]=2), it will never use MaxPool to downsample, even downsamples[i-1]=True. Default: (True, True, True, True).
- **enc_dilations** (*Sequence[int]*) – Dilation rate of each stage in encoder. Default: (1, 1, 1, 1, 1).
- **dec_dilations** (*Sequence[int]*) – Dilation rate of each stage in decoder. Default: (1, 1, 1, 1).
- **with_cp** (*bool*) – Use checkpoint or not. Using checkpoint will save some memory while slowing down the training speed. Default: False.
- **conv_cfg** (*dict | None*) – Config dict for convolution layer. Default: None.
- **norm_cfg** (*dict | None*) – Config dict for normalization layer. Default: dict(type='BN').
- **act_cfg** (*dict | None*) – Config dict for activation layer in ConvModule. Default: dict(type='ReLU').

- **upsample_cfg** (*dict*) – The upsample config of the upsample module in decoder. Default: dict(type='InterpConv').
- **norm_eval** (*bool*) – Whether to set norm layers to eval mode, namely, freeze running stats (mean and var). Note: Effect on Batch Norm and its variants only. Default: False.
- **dcn** (*bool*) – Use deformable convolution in convolutional layer or not. Default: None.
- **plugins** (*dict*) – plugins for convolutional layers. Default: None.
- **pretrained** (*str, optional*) – model pretrained path. Default: None
- **init_cfg** (*dict or list[dict], optional*) – Initialization config dict. Default: None

Notice: The input image size should be divisible by the whole downsample rate of the encoder. More detail of the whole downsample rate can be found in UNet._check_input_divisible.

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.

---

train(*mode=True*)

Convert the model into training mode while keep normalization layer freezed.

class mmseg.models.backbones.**VisionTransformer**(*img_size=224*, *patch_size=16*, *in_channels=3*, *embed_dims=768*, *num_layers=12*, *num_heads=12*, *mlp_ratio=4*, *out_indices=- 1*, *qkv_bias=True*, *drop_rate=0.0*, *attn_drop_rate=0.0*, *drop_path_rate=0.0*, *with_cls_token=True*, *output_cls_token=False*, *norm_cfg={'type': 'LN'}*, *act_cfg={'type': 'GELU'}*, *patch_norm=False*, *final_norm=False*, *interpolate_mode='bicubic'*, *num_fcs=2*, *norm_eval=False*, *with_cp=False*, *pretrained=None*, *init_cfg=None*)

Vision Transformer.

This backbone is the implementation of An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale.

Parameters

- **img_size** (*int | tuple*) – Input image size. Default: 224.
- **patch_size** (*int*) – The patch size. Default: 16.
- **in_channels** (*int*) – Number of input channels. Default: 3.
- **embed_dims** (*int*) – embedding dimension. Default: 768.
- **num_layers** (*int*) – depth of transformer. Default: 12.
- **num_heads** (*int*) – number of attention heads. Default: 12.
- **mlp_ratio** (*int*) – ratio of mlp hidden dim to embedding dim. Default: 4.
- **out_indices** (*list | tuple | int*) – Output from which stages. Default: -1.

- **qkv_bias** (*bool*) – enable bias for qkv if True. Default: True.
- **drop_rate** (*float*) – Probability of an element to be zeroed. Default 0.0
- **attn_drop_rate** (*float*) – The drop out rate for attention layer. Default 0.0
- **drop_path_rate** (*float*) – stochastic depth rate. Default 0.0
- **with_cls_token** (*bool*) – Whether concatenating class token into image tokens as transformer input. Default: True.
- **output_cls_token** (*bool*) – Whether output the cls_token. If set True, *with_cls_token* must be True. Default: False.
- **norm_cfg** (*dict*) – Config dict for normalization layer. Default: dict(type='LN')
- **act_cfg** (*dict*) – The activation config for FFNs. Default: dict(type='GELU').
- **patch_norm** (*bool*) – Whether to add a norm in PatchEmbed Block. Default: False.
- **final_norm** (*bool*) – Whether to add a additional layer to normalize final feature map. Default: False.
- **interpolate_mode** (*str*) – Select the interpolate mode for position embeding vector resize. Default: bicubic.
- **num_fcs** (*int*) – The number of fully-connected layers for FFNs. Default: 2.
- **norm_eval** (*bool*) – Whether to set norm layers to eval mode, namely, freeze running stats (mean and var). Note: Effect on Batch Norm and its variants only. Default: False.
- **with_cp** (*bool*) – Use checkpoint or not. Using checkpoint will save some memory while slowing down the training speed. Default: False.
- **pretrained** (*str, optional*) – model pretrained path. Default: None.
- **init_cfg** (*dict or list[dict], optional*) – Initialization config dict. Default: None.

forward(*inputs*)

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()

Initialize the weights.

static resize_pos_embed(*pos_embed*, *input_shpae*, *pos_shape*, *mode*)

Resize pos_embed weights.

Resize pos_embed using bicubic interpolate method. :param pos_embed: Position embedding weights. :type pos_embed: torch.Tensor :param input_shpae: Tuple for (downsampled input image height,

downsampled input image width).

Parameters

- **pos_shape** (*tuple*) – The resolution of downsampled origin training image.

- **mode** (*str*) – Algorithm used for upsampling: `'nearest'` | `'linear'` | `'bilinear'` | `'bicubic'` | `'trilinear'`. Default: `'nearest'`

Returns The resized pos_embed of shape [B, L_new, C]

Return type torch.Tensor

`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

25.3 decode_heads

`class mmseg.models.decode_heads.`**ANNHead**(*project_channels*, *query_scales=(1)*, *key_pool_scales=(1, 3, 6, 8)*, ***kwargs*)

Asymmetric Non-local Neural Networks for Semantic Segmentation.

This head is the implementation of ANNNet.

Parameters

- **project_channels** (*int*) – Projection channels for Nonlocal.
- **query_scales** (*tuple[int]*) – The scales of query feature map. Default: (1,)
- **key_pool_scales** (*tuple[int]*) – The pooling scales of key feature map. Default: (1, 3, 6, 8).

`forward`(*inputs*)

Forward function.

`class mmseg.models.decode_heads.`**APCHead**(*pool_scales=(1, 2, 3, 6)*, *fusion=True*, ***kwargs*)

Adaptive Pyramid Context Network for Semantic Segmentation.

This head is the implementation of APCNet.

Parameters

- **pool_scales** (*tuple[int]*) – Pooling scales used in Adaptive Context Module. Default: (1, 2, 3, 6).
- **fusion** (*bool*) – Add one conv to fuse residual feature.

`forward`(*inputs*)

Forward function.

`class mmseg.models.decode_heads.`**ASPPHead**(*dilations=(1, 6, 12, 18)*, ***kwargs*)

Rethinking Atrous Convolution for Semantic Image Segmentation.

This head is the implementation of DeepLabV3.

Parameters **dilations** (*tuple[int]*) – Dilation rates for ASPP module. Default: (1, 6, 12, 18).

forward(*inputs*)
Forward function.

class mmseg.models.decode_heads.**CCHead**(*recurrence=2*, ***kwargs*)
CCNet: Criss-Cross Attention for Semantic Segmentation.

This head is the implementation of CCNet.

Parameters **recurrence** (*int*) – Number of recurrence of Criss Cross Attention module. Default: 2.

forward(*inputs*)
Forward function.

class mmseg.models.decode_heads.**DAHead**(*pam_channels*, ***kwargs*)
Dual Attention Network for Scene Segmentation.

This head is the implementation of DANet.

Parameters **pam_channels** (*int*) – The channels of Position Attention Module(PAM).

cam_cls_seg(*feat*)
CAM feature classification.

forward(*inputs*)
Forward function.

forward_test(*inputs*, *img_metas*, *test_cfg*)
Forward function for testing, only pam_cam is used.

losses(*seg_logit*, *seg_label*)
Compute pam_cam, pam, cam loss.

pam_cls_seg(*feat*)
PAM feature classification.

class mmseg.models.decode_heads.**DMHead**(*filter_sizes=(1, 3, 5, 7)*, *fusion=False*, ***kwargs*)
Dynamic Multi-scale Filters for Semantic Segmentation.

This head is the implementation of DMNet.

Parameters

- **filter_sizes** (*tuple[int]*) – The size of generated convolutional filters used in Dynamic Convolutional Module. Default: (1, 3, 5, 7).
- **fusion** (*bool*) – Add one conv to fuse DCM output feature.

forward(*inputs*)
Forward function.

class mmseg.models.decode_heads.**DNLHead**(*reduction=2*, *use_scale=True*, *mode='embedded_gaussian'*, *temperature=0.05*, ***kwargs*)
Disentangled Non-Local Neural Networks.

This head is the implementation of DNLNet.

Parameters

- **reduction** (*int*) – Reduction factor of projection transform. Default: 2.
- **use_scale** (*bool*) – Whether to scale pairwise_weight by sqrt(1/inter_channels). Default: False.
- **mode** (*str*) – The nonlocal mode. Options are 'embedded_gaussian', 'dot_product'. Default: 'embedded_gaussian.'.

- **temperature** (*float*) – Temperature to adjust attention. Default: 0.05

forward(*inputs*)
Forward function.

class mmseg.models.decode_heads.**DPTHead**(*embed_dims=768*, *post_process_channels=[96, 192, 384, 768]*, *readout_type='ignore'*, *patch_size=16*, *expand_channels=False*, *act_cfg={'type': 'ReLU'}*, *norm_cfg={'type': 'BN'}*, ***kwargs*)
Vision Transformers for Dense Prediction.

This head is implemented of DPT.

Parameters

- **embed_dims** (*int*) – The embed dimension of the ViT backbone. Default: 768.
- **post_process_channels** (*List*) – Out channels of post process conv layers. Default: [96, 192, 384, 768].
- **readout_type** (*str*) – Type of readout operation. Default: 'ignore'.
- **patch_size** (*int*) – The patch size. Default: 16.
- **expand_channels** (*bool*) – Whether expand the channels in post process block. Default: False.
- **act_cfg** (*dict*) – The activation config for residual conv unit. Default dict(type='ReLU').
- **norm_cfg** (*dict*) – Config dict for normalization layer. Default: dict(type='BN').

forward(*inputs*)
Placeholder of forward function.

class mmseg.models.decode_heads.**DepthwiseSeparableASPPHead**(*c1_in_channels*, *c1_channels*, ***kwargs*)
Encoder-Decoder with Atrous Separable Convolution for Semantic Image Segmentation.

This head is the implementation of DeepLabV3+.

Parameters

- **c1_in_channels** (*int*) – The input channels of c1 decoder. If is 0, the no decoder will be used.
- **c1_channels** (*int*) – The intermediate channels of c1 decoder.

forward(*inputs*)
Forward function.

class mmseg.models.decode_heads.**DepthwiseSeparableFCNHead**(*dw_act_cfg=None*, ***kwargs*)
Depthwise-Separable Fully Convolutional Network for Semantic Segmentation.

This head is implemented according to Fast-SCNN: Fast Semantic Segmentation Network.

Parameters

- **in_channels** (*int*) – Number of output channels of FFM.
- **channels** (*int*) – Number of middle-stage channels in the decode head.
- **concat_input** (*bool*) – Whether to concatenate original decode input into the result of several consecutive convolution layers. Default: True.
- **num_classes** (*int*) – Used to determine the dimension of final prediction tensor.
- **in_index** (*int*) – Correspond with 'out_indices' in FastSCNN backbone.

- **norm_cfg** (*dict | None*) – Config of norm layers.
- **align_corners** (*bool*) – align_corners argument of F.interpolate. Default: False.
- **loss_decode** (*dict*) – Config of loss type and some relevant additional options.
- **dw_act_cfg** (*dict*) – Activation config of depthwise ConvModule. If it is 'default', it will be the same as *act_cfg*. Default: None.

class mmseg.models.decode_heads.**EMAHead**(*ema_channels*, *num_bases*, *num_stages*, *concat_input=True*, *momentum=0.1*, ***kwargs*)

Expectation Maximization Attention Networks for Semantic Segmentation.

This head is the implementation of EMANet.

Parameters

- **ema_channels** (*int*) – EMA module channels
- **num_bases** (*int*) – Number of bases.
- **num_stages** (*int*) – Number of the EM iterations.
- **concat_input** (*bool*) – Whether concat the input and output of convs before classification layer. Default: True
- **momentum** (*float*) – Momentum to update the base. Default: 0.1.

forward(*inputs*)

Forward function.

class mmseg.models.decode_heads.**EncHead**(*num_codes=32*, *use_se_loss=True*, *add_lateral=False*, *loss_se_decode={'loss_weight': 0.2, 'type': 'CrossEntropyLoss', 'use_sigmoid': True}*, ***kwargs*)

Context Encoding for Semantic Segmentation.

This head is the implementation of EncNet.

Parameters

- **num_codes** (*int*) – Number of code words. Default: 32.
- **use_se_loss** (*bool*) – Whether use Semantic Encoding Loss (SE-loss) to regularize the training. Default: True.
- **add_lateral** (*bool*) – Whether use lateral connection to fuse features. Default: False.
- **loss_se_decode** (*dict*) – Config of decode loss. Default: dict(type='CrossEntropyLoss', use_sigmoid=True).

forward(*inputs*)

Forward function.

forward_test(*inputs*, *img_metas*, *test_cfg*)

Forward function for testing, ignore se_loss.

losses(*seg_logit*, *seg_label*)

Compute segmentation and semantic encoding loss.

class mmseg.models.decode_heads.**FCNHead**(*num_convs=2*, *kernel_size=3*, *concat_input=True*, *dilation=1*, ***kwargs*)

Fully Convolution Networks for Semantic Segmentation.

This head is implemented of FCNNet.

Parameters

- **num_convs** (*int*) – Number of convs in the head. Default: 2.
- **kernel_size** (*int*) – The kernel size for convs in the head. Default: 3.
- **concat_input** (*bool*) – Whether concat the input and output of convs before classification layer.
- **dilation** (*int*) – The dilation rate for convs in the head. Default: 1.

forward(*inputs*)
Forward function.

class mmseg.models.decode_heads.**FPNHead**(*feature_strides*, ***kwargs*)
Panoptic Feature Pyramid Networks.

This head is the implementation of Semantic FPN.

Parameters **feature_strides** (*tuple[int]*) – The strides for input feature maps. stack_lateral. All strides suppose to be power of 2. The first one is of largest resolution.

forward(*inputs*)
Placeholder of forward function.

class mmseg.models.decode_heads.**GCHead**(*ratio=0.25*, *pooling_type='att'*, *fusion_types=('channel_add')*, ***kwargs*)
GCNet: Non-local Networks Meet Squeeze-Excitation Networks and Beyond.

This head is the implementation of GCNet.

Parameters

- **ratio** (*float*) – Multiplier of channels ratio. Default: 1/4.
- **pooling_type** (*str*) – The pooling type of context aggregation. Options are 'att', 'avg'. Default: 'avg'.
- **fusion_types** (*tuple[str]*) – The fusion type for feature fusion. Options are 'channel_add', 'channel_mul'. Default: ('channel_add',)

forward(*inputs*)
Forward function.

class mmseg.models.decode_heads.**ISAHead**(*isa_channels*, *down_factor=(8, 8)*, ***kwargs*)
Interlaced Sparse Self-Attention for Semantic Segmentation.

This head is the implementation of ISA.

Parameters

- **isa_channels** (*int*) – The channels of ISA Module.
- **down_factor** (*tuple[int]*) – The local group size of ISA.

forward(*inputs*)
Forward function.

class mmseg.models.decode_heads.**IterativeDecodeHead**(*num_stages*, *kernel_generate_head*, *kernel_update_head*, ***kwargs*)
K-Net: Towards Unified Image Segmentation.

This head is the implementation of **`K-Net:<https://arxiv.org/abs/2106.14855>`_**.

Parameters

- **num_stages** (*int*) – The number of stages (kernel update heads) in IterativeDecodeHead. Default: 3.

- **kernel_generate_head** – (dict): Config of kernel generate head which generate mask predictions, dynamic kernels and class predictions for next kernel update heads.
- **kernel_update_head** (*dict*) – Config of kernel update head which refine dynamic kernels and class predictions iteratively.

forward(*inputs*)
Forward function.

losses(*seg_logit*, *seg_label*)
Compute segmentation loss.

class mmseg.models.decode_heads.**KernelUpdateHead**(*num_classes=150*, *num_ffn_fcs=2*, *num_heads=8*, *num_mask_fcs=3*, *feedforward_channels=2048*, *in_channels=256*, *out_channels=256*, *dropout=0.0*, *act_cfg={'inplace': True, 'type': 'ReLU'}*, *ffn_act_cfg={'inplace': True, 'type': 'ReLU'}*, *conv_kernel_size=1*, *feat_transform_cfg=None*, *kernel_init=False*, *with_ffn=True*, *feat_gather_stride=1*, *mask_transform_stride=1*, *kernel_updator_cfg={'act_cfg': {'inplace': True, 'type': 'ReLU'}, 'feat_channels': 64, 'in_channels': 256, 'norm_cfg': {'type': 'LN'}, 'out_channels': 256, 'type': 'DynamicConv'}*)

Kernel Update Head in K-Net.

Parameters

- **num_classes** (*int*) – Number of classes. Default: 150.
- **num_ffn_fcs** (*int*) – The number of fully-connected layers in FFNs. Default: 2.
- **num_heads** (*int*) – The number of parallel attention heads. Default: 8.
- **num_mask_fcs** (*int*) – The number of fully connected layers for mask prediction. Default: 3.
- **feedforward_channels** (*int*) – The hidden dimension of FFNs. Defaults: 2048.
- **in_channels** (*int*) – The number of channels of input feature map. Default: 256.
- **out_channels** (*int*) – The number of output channels. Default: 256.
- **dropout** (*float*) – The Probability of an element to be zeroed in MultiheadAttention and FFN. Default 0.0.
- **act_cfg** (*dict*) – Config of activation layers. Default: dict(type='ReLU').
- **ffn_act_cfg** (*dict*) – Config of activation layers in FFN. Default: dict(type='ReLU').
- **conv_kernel_size** (*int*) – The kernel size of convolution in Kernel Update Head for dynamic kernel updation. Default: 1.
- **feat_transform_cfg** (*dict | None*) – Config of feature transform. Default: None.
- **kernel_init** (*bool*) – Whether initiate mask kernel in mask head. Default: False.
- **with_ffn** (*bool*) – Whether add FFN in kernel update head. Default: True.
- **feat_gather_stride** (*int*) – Stride of convolution in feature transform. Default: 1.
- **mask_transform_stride** (*int*) – Stride of mask transform. Default: 1.
- **kernel_updator_cfg** (*dict*) – Config of kernel updator. Default: dict(

type='DynamicConv', in_channels=256, feat_channels=64, out_channels=256, act_cfg=dict(type='ReLU', inplace=True), norm_cfg=dict(type='LN')).

forward(*x*, *proposal_feat*, *mask_preds*, *mask_shape=None*)
Forward function of Dynamic Instance Interactive Head.

Parameters

- **x** (*Tensor*) – Feature map from FPN with shape (batch_size, feature_dimensions, H , W).
- **proposal_feat** (*Tensor*) – Intermediate feature get from diihead in last stage, has shape (batch_size, num_proposals, feature_dimensions)
- **mask_preds** (*Tensor*) – mask prediction from the former stage in shape (batch_size, num_proposals, H, W).

Returns The first tensor is predicted mask with shape (N, num_classes, H, W), the second tensor is dynamic kernel with shape (N, num_classes, channels, K, K).

Return type Tuple

init_weights()
Use xavier initialization for all weight parameter and set classification head bias as a specific value when use focal loss.

class mmseg.models.decode_heads.**KernelUpdator**(*in_channels=256*, *feat_channels=64*, *out_channels=None*, *gate_sigmoid=True*, *gate_norm_act=False*, *activate_out=False*, *norm_cfg={'type': 'LN'}*, *act_cfg={'inplace': True, 'type': 'ReLU'}*)
Dynamic Kernel Updator in Kernel Update Head.

Parameters

- **in_channels** (*int*) – The number of channels of input feature map. Default: 256.
- **feat_channels** (*int*) – The number of middle-stage channels in the kernel updator. Default: 64.
- **out_channels** (*int*) – The number of output channels.
- **gate_sigmoid** (*bool*) – Whether use sigmoid function in gate mechanism. Default: True.
- **gate_norm_act** (*bool*) – Whether add normalization and activation layer in gate mechanism. Default: False.
- **activate_out** – Whether add activation after gate mechanism. Default: False.
- **norm_cfg** (*dict | None*) – Config of norm layers. Default: dict(type='LN').
- **act_cfg** (*dict*) – Config of activation layers. Default: dict(type='ReLU').

forward(*update_feature*, *input_feature*)
Forward function of KernelUpdator.

Parameters

- **update_feature** (*torch.Tensor*) – Feature map assembled from each group. It would be reshaped with last dimension shape: *self.in_channels*.
- **input_feature** (*torch.Tensor*) – Intermediate feature with shape: (N, num_classes, conv_kernel_size**2, channels).

Returns The output tensor of shape (N*C1/C2, K*K, C2), where N is the number of classes, C1 and C2 are the feature map channels of KernelUpdateHead and KernelUpdator, respectively.

Return type Tensor

class mmseg.models.decode_heads.**LRASPPHead**(*branch_channels=(32, 64)*, ***kwargs*)

Lite R-ASPP (LRASPP) head is proposed in Searching for MobileNetV3.

This head is the improved implementation of Searching for MobileNetV3.

Parameters **branch_channels** (*tuple[int]*) – The number of output channels in every each branch. Default: (32, 64).

forward(*inputs*)

Forward function.

class mmseg.models.decode_heads.**NLHead**(*reduction=2*, *use_scale=True*, *mode='embedded_gaussian'*, ***kwargs*)

Non-local Neural Networks.

This head is the implementation of NLNet.

Parameters

- **reduction** (*int*) – Reduction factor of projection transform. Default: 2.
- **use_scale** (*bool*) – Whether to scale pairwise_weight by sqrt(1/inter_channels). Default: True.
- **mode** (*str*) – The nonlocal mode. Options are 'embedded_gaussian', 'dot_product'. Default: 'embedded_gaussian.'.

forward(*inputs*)

Forward function.

class mmseg.models.decode_heads.**OCRHead**(*ocr_channels*, *scale=1*, ***kwargs*)

Object-Contextual Representations for Semantic Segmentation.

This head is the implementation of OCRNet.

Parameters

- **ocr_channels** (*int*) – The intermediate channels of OCR block.
- **scale** (*int*) – The scale of probability map in SpatialGatherModule in Default: 1.

forward(*inputs*, *prev_output*)

Forward function.

class mmseg.models.decode_heads.**PSAHead**(*mask_size*, *psa_type='bi-direction'*, *compact=False*, *shrink_factor=2*, *normalization_factor=1.0*, *psa_softmax=True*, ***kwargs*)

Point-wise Spatial Attention Network for Scene Parsing.

This head is the implementation of PSANet.

Parameters

- **mask_size** (*tuple[int]*) – The PSA mask size. It usually equals input size.
- **psa_type** (*str*) – The type of psa module. Options are 'collect', 'distribute', 'bi-direction'. Default: 'bi-direction'
- **compact** (*bool*) – Whether use compact map for 'collect' mode. Default: True.
- **shrink_factor** (*int*) – The downsample factors of psa mask. Default: 2.
- **normalization_factor** (*float*) – The normalize factor of attention.
- **psa_softmax** (*bool*) – Whether use softmax for attention.

forward(*inputs*)
Forward function.

class mmseg.models.decode_heads.**PSPHead**(*pool_scales=(1, 2, 3, 6)*, ***kwargs*)
Pyramid Scene Parsing Network.

This head is the implementation of PSPNet.

Parameters **pool_scales** (*tuple[int]*) – Pooling scales used in Pooling Pyramid Module. Default: (1, 2, 3, 6).

forward(*inputs*)
Forward function.

class mmseg.models.decode_heads.**PointHead**(*num_fcs=3*, *coarse_pred_each_layer=True*, *conv_cfg={'type': 'Conv1d'}*, *norm_cfg=None*, *act_cfg={'inplace': False, 'type': 'ReLU'}*, ***kwargs*)
A mask point head use in PointRend.

This head is implemented of PointRend: Image Segmentation as Rendering. PointHead use shared multi-layer perceptron (equivalent to nn.Conv1d) to predict the logit of input points. The fine-grained feature and coarse feature will be concatenate together for predication.

Parameters

- **num_fcs** (*int*) – Number of fc layers in the head. Default: 3.
- **in_channels** (*int*) – Number of input channels. Default: 256.
- **fc_channels** (*int*) – Number of fc channels. Default: 256.
- **num_classes** (*int*) – Number of classes for logits. Default: 80.
- **class_agnostic** (*bool*) – Whether use class agnostic classification. If so, the output channels of logits will be 1. Default: False.
- **coarse_pred_each_layer** (*bool*) – Whether concatenate coarse feature with the output of each fc layer. Default: True.
- **conv_cfg** (*dict|None*) – Dictionary to construct and config conv layer. Default: dict(type='Conv1d'))
- **norm_cfg** (*dict|None*) – Dictionary to construct and config norm layer. Default: None.
- **loss_point** (*dict*) – Dictionary to construct and config loss layer of point head. Default: dict(type='CrossEntropyLoss', use_mask=True, loss_weight=1.0).

cls_seg(*feat*)
Classify each pixel with fc.

forward(*fine_grained_point_feats*, *coarse_point_feats*)
Placeholder of forward function.

forward_test(*inputs*, *prev_output*, *img_metas*, *test_cfg*)
Forward function for testing.

Parameters

- **inputs** (*list[Tensor]*) – List of multi-level img features.
- **prev_output** (*Tensor*) – The output of previous decode head.
- **img_metas** (*list[dict]*) – List of image info dict where each dict has: 'img_shape', 'scale_factor', 'flip', and may also contain 'filename', 'ori_shape',

'pad_shape', and 'img_norm_cfg'. For details on the values of these keys see *mmseg/datasets/pipelines/formatting.py:Collect*.

- **test_cfg** (*dict*) – The testing config.

Returns Output segmentation map.

Return type Tensor

forward_train(*inputs*, *prev_output*, *img_metas*, *gt_semantic_seg*, *train_cfg*)

Forward function for training. :param inputs: List of multi-level img features. :type inputs: list[Tensor] :param prev_output: The output of previous decode head. :type prev_output: Tensor :param img_metas: List of image info dict where each dict

has: 'img_shape', 'scale_factor', 'flip', and may also contain 'filename', 'ori_shape', 'pad_shape', and 'img_norm_cfg'. For details on the values of these keys see *mmseg/datasets/pipelines/formatting.py:Collect*.

Parameters

- **gt_semantic_seg** (*Tensor*) – Semantic segmentation masks used if the architecture supports semantic segmentation task.
- **train_cfg** (*dict*) – The training config.

Returns a dictionary of loss components

Return type dict[str, Tensor]

get_points_test(*seg_logits*, *uncertainty_func*, *cfg*)

Sample points for testing.

Find `num_points` most uncertain points from `uncertainty_map`.

Parameters

- **seg_logits** (*Tensor*) – A tensor of shape (batch_size, num_classes, height, width) for class-specific or class-agnostic prediction.
- **uncertainty_func** (*func*) – uncertainty calculation function.
- **cfg** (*dict*) – Testing config of point head.

Returns

A tensor of shape (batch_size, num_points) that contains indices from [0, height x width) of the most uncertain points.

point_coords (Tensor): A tensor of shape (batch_size, num_points, 2) that contains [0, 1] x [0, 1] normalized coordinates of the most uncertain points from the `height x width` grid .

Return type point_indices (Tensor)

get_points_train(*seg_logits*, *uncertainty_func*, *cfg*)

Sample points for training.

Sample points in [0, 1] x [0, 1] coordinate space based on their uncertainty. The uncertainties are calculated for each point using 'uncertainty_func' function that takes point's logit prediction as input.

Parameters

- **seg_logits** (*Tensor*) – Semantic segmentation logits, shape (batch_size, num_classes, height, width).

- **uncertainty_func** (*func*) – uncertainty calculation function.
- **cfg** (*dict*) – Training config of point head.

Returns

A tensor of shape (batch_size, num_points, 2) that contains the coordinates of `num_points` sampled points.

Return type point_coords (Tensor)

losses(*point_logits*, *point_label*)
Compute segmentation loss.

class mmseg.models.decode_heads.**SETRMLAHead**(*mla_channels=128*, *up_scale=4*, ***kwargs*)
Multi level feature aggretation head of SETR.

MLA head of SETR.

Parameters

- **mlahead_channels** (*int*) – Channels of conv-conv-4x of multi-level feature aggregation. Default: 128.
- **up_scale** (*int*) – The scale factor of interpolate. Default:4.

forward(*inputs*)
Placeholder of forward function.

class mmseg.models.decode_heads.**SETRUPHead**(*norm_layer={'eps': 1e-06, 'requires_grad': True, 'type': 'LN'}*, *num_convs=1*, *up_scale=4*, *kernel_size=3*, *init_cfg=[{'type': 'Constant', 'val': 1.0, 'bias': 0, 'layer': 'LayerNorm'}, {'type': 'Normal', 'std': 0.01, 'override': {'name': 'conv_seg'}}]*, ***kwargs*)
Naive upsampling head and Progressive upsampling head of SETR.

Naive or PUP head of SETR.

Parameters

- **norm_layer** (*dict*) – Config dict for input normalization. Default: norm_layer=dict(type='LN', eps=1e-6, requires_grad=True).
- **num_convs** (*int*) – Number of decoder convolutions. Default: 1.
- **up_scale** (*int*) – The scale factor of interpolate. Default:4.
- **kernel_size** (*int*) – The kernel size of convolution when decoding feature information from backbone. Default: 3.
- **init_cfg** (*dict | list[dict] | None*) – Initialization config dict. Default: dict(

  type='Constant', val=1.0, bias=0, layer='LayerNorm').

forward(*x*)
Placeholder of forward function.

class mmseg.models.decode_heads.**STDCHead**(*boundary_threshold=0.1*, ***kwargs*)
This head is the implementation of Rethinking BiSeNet For Real-time Semantic Segmentation.

Parameters **boundary_threshold** (*float*) – The threshold of calculating boundary. Default: 0.1.

losses(*seg_logit*, *seg_label*)
Compute Detail Aggregation Loss.

class mmseg.models.decode_heads.**SegformerHead**(*interpolate_mode='bilinear'*, ***kwargs*)
The all mlp Head of segformer.

This head is the implementation of *Segformer <https://arxiv.org/abs/2105.15203> _*.

Parameters **interpolate_mode** – The interpolate mode of MLP head upsample operation. Default: 'bilinear'.

forward(*inputs*)
Placeholder of forward function.

class mmseg.models.decode_heads.**SegmenterMaskTransformerHead**(*in_channels*, *num_layers*, *num_heads*, *embed_dims*, *mlp_ratio=4*, *drop_path_rate=0.1*, *drop_rate=0.0*, *attn_drop_rate=0.0*, *num_fcs=2*, *qkv_bias=True*, *act_cfg={'type': 'GELU'}*, *norm_cfg={'type': 'LN'}*, *init_std=0.02*, ***kwargs*)
Segmenter: Transformer for Semantic Segmentation.

This head is the implementation of **`Segmenter:<https://arxiv.org/abs/2105.05633>`_**.

Parameters

- **backbone_cfg** – (dict): Config of backbone of Context Path.
- **in_channels** (*int*) – The number of channels of input image.
- **num_layers** (*int*) – The depth of transformer.
- **num_heads** (*int*) – The number of attention heads.
- **embed_dims** (*int*) – The number of embedding dimension.
- **mlp_ratio** (*int*) – ratio of mlp hidden dim to embedding dim. Default: 4.
- **drop_path_rate** (*float*) – stochastic depth rate. Default 0.1.
- **drop_rate** (*float*) – Probability of an element to be zeroed. Default 0.0
- **attn_drop_rate** (*float*) – The drop out rate for attention layer. Default 0.0
- **num_fcs** (*int*) – The number of fully-connected layers for FFNs. Default: 2.
- **qkv_bias** (*bool*) – Enable bias for qkv if True. Default: True.
- **act_cfg** (*dict*) – The activation config for FFNs. Default: dict(type='GELU').
- **norm_cfg** (*dict*) – Config dict for normalization layer. Default: dict(type='LN')
- **init_std** (*float*) – The value of std in weight initialization. Default: 0.02.

forward(*inputs*)
Placeholder of forward function.

init_weights()
Initialize the weights.

class mmseg.models.decode_heads.**UPerHead**(*pool_scales=(1, 2, 3, 6)*, ***kwargs*)
Unified Perceptual Parsing for Scene Understanding.

This head is the implementation of UPerNet.

Parameters **pool_scales** (*tuple[int]*) – Pooling scales used in Pooling Pyramid Module applied on the last feature. Default: (1, 2, 3, 6).

forward(*inputs*)
Forward function.

psp_forward(*inputs*)
Forward function of PSP module.

25.4 losses

class mmseg.models.losses.**Accuracy**(*topk=(1)*, *thresh=None*, *ignore_index=None*)
Accuracy calculation module.

forward(*pred*, *target*)
Forward function to calculate accuracy.

Parameters

- **pred** (*torch.Tensor*) – Prediction of models.
- **target** (*torch.Tensor*) – Target for each prediction.

Returns The accuracies under different topk criterions.

Return type tuple[float]

class mmseg.models.losses.**CrossEntropyLoss**(*use_sigmoid=False*, *use_mask=False*, *reduction='mean'*, *class_weight=None*, *loss_weight=1.0*, *loss_name='loss_ce'*, *avg_non_ignore=False*)
CrossEntropyLoss.

Parameters

- **use_sigmoid** (*bool, optional*) – Whether the prediction uses sigmoid of softmax. Defaults to False.
- **use_mask** (*bool, optional*) – Whether to use mask cross entropy loss. Defaults to False.
- **reduction** (*str, optional*) – . Defaults to 'mean'. Options are "none", "mean" and "sum".
- **class_weight** (*list[float] | str, optional*) – Weight of each class. If in str format, read them from a file. Defaults to None.
- **loss_weight** (*float, optional*) – Weight of the loss. Defaults to 1.0.
- **loss_name** (*str, optional*) – Name of the loss item. If you want this loss item to be included into the backward graph, *loss_* must be the prefix of the name. Defaults to 'loss_ce'.
- **avg_non_ignore** (*bool*) – The flag decides to whether the loss is only averaged over non-ignored targets. Default: False. *New in version 0.23.0.*

extra_repr()
Extra repr.

forward(*cls_score*, *label*, *weight=None*, *avg_factor=None*, *reduction_override=None*, *ignore_index=- 100*, ***kwargs*)
Forward function.

property loss_name
Loss Name.

This function must be implemented and will return the name of this loss function. This name will be used to combine different loss items by simple sum operation. In addition, if you want this loss item to be included into the backward graph, *loss_* must be the prefix of the name.

Returns The name of this loss item.

Return type str

class mmseg.models.losses.**DiceLoss**(*smooth=1*, *exponent=2*, *reduction='mean'*, *class_weight=None*, *loss_weight=1.0*, *ignore_index=255*, *loss_name='loss_dice'*, ***kwargs*)

DiceLoss.

This loss is proposed in V-Net: Fully Convolutional Neural Networks for Volumetric Medical Image Segmentation.

Parameters

- **smooth** (*float*) – A float number to smooth loss, and avoid NaN error. Default: 1
- **exponent** (*float*) – An float number to calculate denominator value: sum{x^exponent} + sum{y^exponent}. Default: 2.
- **reduction** (*str, optional*) – The method used to reduce the loss. Options are "none", "mean" and "sum". This parameter only works when per_image is True. Default: 'mean'.
- **class_weight** (*list[float] | str, optional*) – Weight of each class. If in str format, read them from a file. Defaults to None.
- **loss_weight** (*float, optional*) – Weight of the loss. Default to 1.0.
- **ignore_index** (*int | None*) – The label index to be ignored. Default: 255.
- **loss_name** (*str, optional*) – Name of the loss item. If you want this loss item to be included into the backward graph, *loss_* must be the prefix of the name. Defaults to 'loss_dice'.

forward(*pred*, *target*, *avg_factor=None*, *reduction_override=None*, ***kwargs*)

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.

---

property loss_name

Loss Name.

This function must be implemented and will return the name of this loss function. This name will be used to combine different loss items by simple sum operation. In addition, if you want this loss item to be included into the backward graph, *loss_* must be the prefix of the name. :returns: The name of this loss item. :rtype: str

class mmseg.models.losses.**FocalLoss**(*use_sigmoid=True*, *gamma=2.0*, *alpha=0.5*, *reduction='mean'*, *class_weight=None*, *loss_weight=1.0*, *loss_name='loss_focal'*)

forward(*pred*, *target*, *weight=None*, *avg_factor=None*, *reduction_override=None*, *ignore_index=255*, ***kwargs*)

Forward function.

Parameters

- **pred** (*torch.Tensor*) – The prediction with shape (N, C) where C = number of classes, or (N, C, d_1, d_2, ..., d_K) with K1 in the case of K-dimensional loss.
- **target** (*torch.Tensor*) – The ground truth. If containing class indices, shape (N) where each value is 0targets[i]C1, or (N, d_1, d_2, ..., d_K) with K1 in the case of K-dimensional loss. If containing class probabilities, same shape as the input.
- **weight** (*torch.Tensor, optional*) – The weight of loss for each prediction. Defaults to None.
- **avg_factor** (*int, optional*) – Average factor that is used to average the loss. Defaults to None.
- **reduction_override** (*str, optional*) – The reduction method used to override the original reduction method of the loss. Options are "none", "mean" and "sum".
- **ignore_index** (*int, optional*) – The label index to be ignored. Default: 255

Returns The calculated loss

Return type torch.Tensor

property loss_name

Loss Name.

This function must be implemented and will return the name of this loss function. This name will be used to combine different loss items by simple sum operation. In addition, if you want this loss item to be included into the backward graph, *loss_* must be the prefix of the name. :returns: The name of this loss item. :rtype: str

class mmseg.models.losses.**LovaszLoss**(*loss_type='multi_class'*, *classes='present'*, *per_image=False*, *reduction='mean'*, *class_weight=None*, *loss_weight=1.0*, *loss_name='loss_lovasz'*)

LovaszLoss.

This loss is proposed in The Lovasz-Softmax loss: A tractable surrogate for the optimization of the intersection-over-union measure in neural networks.

Parameters

- **loss_type** (*str, optional*) – Binary or multi-class loss. Default: 'multi_class'. Options are "binary" and "multi_class".
- **classes** (*str | list[int], optional*) – Classes chosen to calculate loss. 'all' for all classes, 'present' for classes present in labels, or a list of classes to average. Default: 'present'.
- **per_image** (*bool, optional*) – If per_image is True, compute the loss per image instead of per batch. Default: False.
- **reduction** (*str, optional*) – The method used to reduce the loss. Options are "none", "mean" and "sum". This parameter only works when per_image is True. Default: 'mean'.
- **class_weight** (*list[float] | str, optional*) – Weight of each class. If in str format, read them from a file. Defaults to None.
- **loss_weight** (*float, optional*) – Weight of the loss. Defaults to 1.0.
- **loss_name** (*str, optional*) – Name of the loss item. If you want this loss item to be included into the backward graph, *loss_* must be the prefix of the name. Defaults to 'loss_lovasz'.

forward(*cls_score*, *label*, *weight=None*, *avg_factor=None*, *reduction_override=None*, ***kwargs*)

Forward function.

property loss_name
Loss Name.

This function must be implemented and will return the name of this loss function. This name will be used to combine different loss items by simple sum operation. In addition, if you want this loss item to be included into the backward graph, *loss_* must be the prefix of the name. :returns: The name of this loss item. :rtype: str

class mmseg.models.losses.**TverskyLoss**(*smooth=1*, *class_weight=None*, *loss_weight=1.0*, *ignore_index=255*, *alpha=0.3*, *beta=0.7*, *loss_name='loss_tversky'*)
TverskyLoss. This loss is proposed in `Tversky loss function for image segmentation using 3D fully convolutional deep networks.

<https://arxiv.org/abs/1706.05721>`_. :param smooth: A float number to smooth loss, and avoid NaN error.

Default: 1.

Parameters

- **class_weight** (*list[float] | str, optional*) – Weight of each class. If in str format, read them from a file. Defaults to None.
- **loss_weight** (*float, optional*) – Weight of the loss. Default to 1.0.
- **ignore_index** (*int | None*) – The label index to be ignored. Default: 255.
- **alpha** (*float, in [0, 1]*) – The coefficient of false positives. Default: 0.3.
- **beta** (*float, in [0, 1]*) – The coefficient of false negatives. Default: 0.7. Note: alpha + beta = 1.
- **loss_name** (*str, optional*) – Name of the loss item. If you want this loss item to be included into the backward graph, *loss_* must be the prefix of the name. Defaults to 'loss_tversky'.

forward(*pred*, *target*, ***kwargs*)
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.

---

property loss_name
Loss Name.

This function must be implemented and will return the name of this loss function. This name will be used to combine different loss items by simple sum operation. In addition, if you want this loss item to be included into the backward graph, *loss_* must be the prefix of the name. :returns: The name of this loss item. :rtype: str

mmseg.models.losses.**accuracy**(*pred*, *target*, *topk=1*, *thresh=None*, *ignore_index=None*)
Calculate accuracy according to the prediction and target.

Parameters

- **pred** (*torch.Tensor*) – The model prediction, shape (N, num_class, ...)
- **target** (*torch.Tensor*) – The target of each prediction, shape (N, , ...)

- **ignore_index** (*int | None*) – The label index to be ignored. Default: None
- **topk** (*int | tuple[int], optional*) – If the predictions in `topk` matches the target, the predictions will be regarded as correct ones. Defaults to 1.
- **thresh** (*float, optional*) – If not None, predictions with scores under this threshold are considered incorrect. Default to None.

Returns

If the input `topk` is a single integer, the function will return a single float as accuracy. If `topk` is a tuple containing multiple integers, the function will return a tuple containing accuracies of each `topk` number.

Return type float | tuple[float]

`mmseg.models.losses.`**binary_cross_entropy**(*pred*, *label*, *weight=None*, *reduction='mean'*, *avg_factor=None*, *class_weight=None*, *ignore_index=- 100*, *avg_non_ignore=False*, ***kwargs*)

Calculate the binary CrossEntropy loss.

Parameters

- **pred** (*torch.Tensor*) – The prediction with shape (N, 1).
- **label** (*torch.Tensor*) – The learning label of the prediction. Note: In bce loss, label < 0 is invalid.
- **weight** (*torch.Tensor, optional*) – Sample-wise loss weight.
- **reduction** (*str, optional*) – The method used to reduce the loss. Options are "none", "mean" and "sum".
- **avg_factor** (*int, optional*) – Average factor that is used to average the loss. Defaults to None.
- **class_weight** (*list[float], optional*) – The weight for each class.
- **ignore_index** (*int*) – The label index to be ignored. Default: -100.
- **avg_non_ignore** (*bool*) – The flag decides to whether the loss is only averaged over non-ignored targets. Default: False. *New in version 0.23.0.*

Returns The calculated loss

Return type torch.Tensor

`mmseg.models.losses.`**cross_entropy**(*pred*, *label*, *weight=None*, *class_weight=None*, *reduction='mean'*, *avg_factor=None*, *ignore_index=- 100*, *avg_non_ignore=False*)

cross_entropy. The wrapper function for `F.cross_entropy()`

Parameters

- **pred** (*torch.Tensor*) – The prediction with shape (N, 1).
- **label** (*torch.Tensor*) – The learning label of the prediction.
- **weight** (*torch.Tensor, optional*) – Sample-wise loss weight. Default: None.
- **class_weight** (*list[float], optional*) – The weight for each class. Default: None.
- **reduction** (*str, optional*) – The method used to reduce the loss. Options are 'none', 'mean' and 'sum'. Default: 'mean'.
- **avg_factor** (*int, optional*) – Average factor that is used to average the loss. Default: None.

- **ignore_index** (*int*) – Specifies a target value that is ignored and does not contribute to the input gradients. When avg_non_ignore `` is ``True, and the reduction is ''mean'', the loss is averaged over non-ignored targets. Defaults: -100.
- **avg_non_ignore** (*bool*) – The flag decides to whether the loss is only averaged over non-ignored targets. Default: False. *New in version 0.23.0.*

mmseg.models.losses.**mask_cross_entropy**(*pred*, *target*, *label*, *reduction='mean'*, *avg_factor=None*, *class_weight=None*, *ignore_index=None*, ***kwargs*)

Calculate the CrossEntropy loss for masks.

Parameters

- **pred** (*torch.Tensor*) – The prediction with shape (N, C), C is the number of classes.
- **target** (*torch.Tensor*) – The learning label of the prediction.
- **label** (*torch.Tensor*) – label indicates the class label of the mask' corresponding object. This will be used to select the mask in the of the class which the object belongs to when the mask prediction if not class-agnostic.
- **reduction** (*str, optional*) – The method used to reduce the loss. Options are "none", "mean" and "sum".
- **avg_factor** (*int, optional*) – Average factor that is used to average the loss. Defaults to None.
- **class_weight** (*list[float], optional*) – The weight for each class.
- **ignore_index** (*None*) – Placeholder, to be consistent with other loss. Default: None.

Returns The calculated loss

Return type torch.Tensor

mmseg.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

mmseg.models.losses.**weight_reduce_loss**(*loss*, *weight=None*, *reduction='mean'*, *avg_factor=None*)

Apply element-wise weight and reduce loss.

Parameters

- **loss** (*Tensor*) – Element-wise loss.
- **weight** (*Tensor*) – Element-wise weights.
- **reduction** (*str*) – Same as built-in losses of PyTorch.
- **avg_factor** (*float*) – Average factor when computing the mean of losses.

Returns Processed loss values.

Return type Tensor

mmseg.models.losses.**weighted_loss**(*loss_func*)

Create a weighted version of a given loss function.

To use this decorator, the loss function must have the signature like *loss_func(pred, target, **kwargs)*. The function only needs to compute element-wise loss without any reduction. This decorator will add weight and reduction arguments to the function. The decorated function will have the signature like *loss_func(pred, target, weight=None, reduction='mean', avg_factor=None, **kwargs)*.

Example

```
>>> import torch
>>> @weighted_loss
>>> def l1_loss(pred, target):
>>>     return (pred - target).abs()
```

```
>>> pred = torch.Tensor([0, 2, 3])
>>> target = torch.Tensor([1, 1, 1])
>>> weight = torch.Tensor([1, 0, 1])
```

```
>>> l1_loss(pred, target)
tensor(1.3333)
>>> l1_loss(pred, target, weight)
tensor(1.)
>>> l1_loss(pred, target, reduction='none')
tensor([1., 1., 2.])
>>> l1_loss(pred, target, weight, avg_factor=2)
tensor(1.5000)
```

CHAPTER

TWENTYSIX

INDICES AND TABLES

- genindex
- search

PYTHON MODULE INDEX

m

INDEX

F

G

H

I

K

L

M

N

O

P

R

S

T

U

V

W