MMSegmentation

Release 0.30.0

MMSegmentation Authors

Jul 22, 2023

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
```

Note: If you would like to use albumentations, we suggest using pip install -U albumentations –no-binary qudida,albumentations. If you simply use pip install albumentations>=0.3.2, it will install opencv-python-headless simultaneously (even though you have already installed opencv-python). We recommended checking the environment after installing albumentations to ensure that opencv-python and opencv-python-headless are not installed at the same time, because it might cause unexpected issues if they both installed. Please refer to official documentation for more details.

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.

```
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
|   ├── occlusion-aware-face-dataset
|   |   ├── train.txt
|   |   ├── NatOcc_hand_sot
|   |   |   ├── img
|   |   |   ├── mask
|   |   ├── NatOcc_object
|   |   |   ├── img
```

(continues on next page)

(continued from previous page)

```
│   │   │   ├── mask
│   │   ├── RandOcc
│   │   │   ├── img
│   │   │   ├── mask
│   │   ├── RealOcc
│   │   │   ├── img
│   │   │   ├── mask
│   │   │   ├── split
│   ├── ImageNetS
│   │   ├── ImageNetS919
│   │   │   ├── train-semi
│   │   │   ├── train-semi-segmentation
│   │   │   ├── validation
│   │   │   ├── validation-segmentation
│   │   │   ├── test
│   │   ├── ImageNetS300
│   │   │   ├── train-semi
│   │   │   ├── train-semi-segmentation
│   │   │   ├── validation
│   │   │   ├── validation-segmentation
│   │   │   ├── test
│   │   ├── ImageNetS50
│   │   │   ├── train-semi
│   │   │   ├── train-semi-segmentation
│   │   │   ├── validation
│   │   │   ├── validation-segmentation
│   │   │   ├── test
```

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
```

(continues on next page)

(continued from previous page)

```
# 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.

3.17 Delving into High-Quality Synthetic Face Occlusion Segmentation Datasets

The dataset is generated by two techniques, Naturalistic occlusion generation, Random occlusion generation. you must install face-occlusion-generation and dataset. see more guide in https://github.com/kennyvoo/face-occlusion-generation.git

CHAPTER

FOUR

DATASET PREPARATION

step 1

Create a folder for data generation materials on mmsegmentation folder.

```
mkdir data_materials
```

step 2

Please download the masks (11k-hands_mask.7z,CelebAMask-HQ-masks_corrected.7z) from this drive

Please download the images from CelebAMask-HQ, 11k Hands.zip and dtd-r1.0.1.tar.gz.

step 3

Download a upsampled COCO objects images and masks (coco_object.7z). files can be found in this drive.

Download CelebAMask-HQ and 11k Hands images split txt files. (11k_hands_sample.txt, CelebAMask-HQ-WO-train.txt) found in drive.

download file to ./data_materials

```
CelebAMask-HQ.zip
CelebAMask-HQ-masks_corrected.7z
CelebAMask-HQ-WO-train.txt
RealOcc.7z
RealOcc-Wild.7z
11k-hands_mask.7z
11k Hands.zip
11k_hands_sample.txt
coco_object.7z
dtd-r1.0.1.tar.gz
```

```
apt-get install p7zip-full

cd data_materials

#make occlusion-aware-face-dataset folder
mkdir path-to-mmsegmentaion/data/occlusion-aware-face-dataset

#extract celebAMask-HQ and split by train-set
unzip CelebAMask-HQ.zip
7za x CelebAMask-HQ-masks_corrected.7z -o./CelebAMask-HQ
```

(continues on next page)

(continued from previous page)

```
#copy training data to train-image-folder
rsync -a ./CelebAMask-HQ/CelebA-HQ-img/ --files-from=./CelebAMask-HQ-WO-train.txt ./
↪CelebAMask-HQ-WO-Train_img
#create a file-name txt file for copying mask
basename -s .jpg ./CelebAMask-HQ-WO-Train_img/* > train.txt
#add .png to file-name txt file
xargs -n 1 -i echo {}.png < train.txt > mask_train.txt
#copy training data to train-mask-folder
rsync -a ./CelebAMask-HQ/CelebAMask-HQ-masks_corrected/ --files-from=./mask_train.txt ./
↪CelebAMask-HQ-WO-Train_mask
mv train.txt ../data/occlusion-aware-face-dataset

#extract DTD
tar -zxvf dtd-r1.0.1.tar.gz
mv dtd DTD

#extract hands dataset and split by 200 samples
7za x 11k-hands_masks.7z -o.
unzip Hands.zip
rsync -a ./Hands/ --files-from=./11k_hands_sample.txt ./11k-hands_img

#extract upscaled coco object
7za x coco_object.7z -o.
mv coco_object/* .

#extract validation set
7za x RealOcc.7z -o../data/occlusion-aware-face-dataset
```

Dataset material Organization:

```
├── data_materials
    ├── CelebAMask-HQ-WO-Train_img
        ├── {image}.jpg
    ├── CelebAMask-HQ-WO-Train_mask
        ├── {mask}.png
    ├── DTD
        ├── images
            ├── {classA}
                ├── {image}.jpg
            ├── {classB}
                ├── {image}.jpg
    ├── 11k-hands_img
        ├── {image}.jpg
    ├── 11k-hands_mask
        ├── {mask}.png
    ├── object_image_sr
        ├── {image}.jpg
    ├── object_mask_x4
        ├── {mask}.png
```

CHAPTER

FIVE

DATA GENERATION

```
git clone https://github.com/kennyvoo/face-occlusion-generation.git
cd face_occlusion-generation
```

Example script to generate NatOcc hand dataset

```
CUDA_VISIBLE_DEVICES=0 NUM_WORKERS=4 python main.py \
--config ./configs/natocc_hand.yaml \
--opts OUTPUT_PATH "path/to/mmsegmentation/data/occlusion-aware-face-dataset/NatOcc_hand_
↪sot"\
AUGMENTATION.SOT True \
SOURCE_DATASET.IMG_DIR "path/to/data_materials/CelebAMask-HQ-WO-Train_img" \
SOURCE_DATASET.MASK_DIR "path/to/mmsegmentation/data_materials/CelebAMask-HQ-WO-Train_
↪mask" \
OCCLUDER_DATASET.IMG_DIR "path/to/mmsegmentation/data_materials/11k-hands_img" \
OCCLUDER_DATASET.MASK_DIR "path/to/mmsegmentation/data_materials/11k-hands_masks"
```

Example script to generate NatOcc object dataset

```
CUDA_VISIBLE_DEVICES=0 NUM_WORKERS=4 python main.py \
--config ./configs/natocc_objects.yaml \
--opts OUTPUT_PATH "path/to/mmsegmentation/data/occlusion-aware-face-dataset/NatOcc_
↪object" \
SOURCE_DATASET.IMG_DIR "path/to/mmsegmentation/data_materials/CelebAMask-HQ-WO-Train_img
↪" \
SOURCE_DATASET.MASK_DIR "path/to/mmsegmentation/data_materials/CelebAMask-HQ-WO-Train_
↪mask" \
OCCLUDER_DATASET.IMG_DIR "path/to/mmsegmentation/data_materials/object_image_sr" \
OCCLUDER_DATASET.MASK_DIR "path/to/mmsegmentation/data_materials/object_mask_x4"
```

Example script to generate RandOcc dataset

```
CUDA_VISIBLE_DEVICES=0 NUM_WORKERS=4  python main.py \
--config ./configs/randocc.yaml \
--opts OUTPUT_PATH "path/to/mmsegmentation/data/occlusion-aware-face-dataset/RandOcc" \
SOURCE_DATASET.IMG_DIR "path/to/mmsegmentation/data_materials/CelebAMask-HQ-WO-Train_img/
↪" \
SOURCE_DATASET.MASK_DIR "path/to/mmsegmentation/data_materials/CelebAMask-HQ-WO-Train_
↪mask" \
OCCLUDER_DATASET.IMG_DIR "path/to/jw93/mmsegmentation/data_materials/DTD/images"
```

Dataset Organization:

```
├── data
│   ├── occlusion-aware-face-dataset
│   │   ├── train.txt
│   │   ├── NatOcc_hand_sot
│   │   │   ├── img
│   │   │   │   ├── {image}.jpg
│   │   │   ├── mask
│   │   │   │   ├── {mask}.png
│   │   ├── NatOcc_object
│   │   │   ├── img
│   │   │   │   ├── {image}.jpg
│   │   │   ├── mask
│   │   │   │   ├── {mask}.png
│   │   ├── RandOcc
│   │   │   ├── img
│   │   │   │   ├── {image}.jpg
│   │   │   ├── mask
│   │   │   │   ├── {mask}.png
│   │   ├── RealOcc
│   │   │   ├── img
│   │   │   │   ├── {image}.jpg
│   │   │   ├── mask
│   │   │   │   ├── {mask}.png
│   │   │   ├── split
│   │   │   │   ├── val.txt
```

5.1 ImageNetS

The ImageNet-S dataset is for Large-scale unsupervised/semi-supervised semantic segmentation.

The images and annotations are available on ImageNet-S.

```
│   ├── ImageNetS
│   │   ├── ImageNetS919
│   │   │   ├── train-semi
│   │   │   ├── train-semi-segmentation
│   │   │   ├── validation
│   │   │   ├── validation-segmentation
│   │   │   ├── test
│   │   ├── ImageNetS300
│   │   │   ├── train-semi
│   │   │   ├── train-semi-segmentation
│   │   │   ├── validation
│   │   │   ├── validation-segmentation
│   │   │   ├── test
│   │   ├── ImageNetS50
│   │   │   ├── train-semi
│   │   │   ├── train-semi-segmentation
│   │   │   ├── validation
```

(continues on next page)

(continued from previous page)

```
│   │   │   ├── validation-segmentation
│   │   │   ├── test
```

CHAPTER

SIX

BENCHMARK AND MODEL ZOO

6.1 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.

6.2 Baselines

6.2.1 FCN

Please refer to FCN for details.

6.2.2 PSPNet

Please refer to PSPNet for details.

6.2.3 DeepLabV3

Please refer to DeepLabV3 for details.

6.2.4 PSANet

Please refer to PSANet for details.

6.2.5 DeepLabV3+

Please refer to DeepLabV3+ for details.

6.2.6 UPerNet

Please refer to UPerNet for details.

6.2.7 NonLocal Net

Please refer to NonLocal Net for details.

6.2.8 EncNet

Please refer to EncNet for details.

6.2.9 CCNet

Please refer to CCNet for details.

6.2.10 DANet

Please refer to DANet for details.

6.2.11 APCNet

Please refer to APCNet for details.

6.2.12 HRNet

Please refer to HRNet for details.

6.2.13 GCNet

Please refer to GCNet for details.

6.2.14 DMNet

Please refer to DMNet for details.

6.2.15 ANN

Please refer to ANN for details.

6.2.16 OCRNet

Please refer to OCRNet for details.

6.2.17 Fast-SCNN

Please refer to Fast-SCNN for details.

6.2.18 ResNeSt

Please refer to ResNeSt for details.

6.2.19 Semantic FPN

Please refer to Semantic FPN for details.

6.2.20 PointRend

Please refer to PointRend for details.

6.2.21 MobileNetV2

Please refer to MobileNetV2 for details.

6.2.22 MobileNetV3

Please refer to MobileNetV3 for details.

6.2.23 EMANet

Please refer to EMANet for details.

6.2.24 DNLNet

Please refer to DNLNet for details.

6.2.25 CGNet

Please refer to CGNet for details.

6.2.26 Mixed Precision (FP16) Training

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

6.2.27 U-Net

Please refer to U-Net for details.

6.2.28 ViT

Please refer to ViT for details.

6.2.29 Swin

Please refer to Swin for details.

6.2.30 SETR

Please refer to SETR for details.

6.3 Speed benchmark

6.3.1 Hardware

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

6.3.2 Software environment

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

6.3.3 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

SEVEN

MODEL ZOO STATISTICS

- Number of papers: 47
  - ALGORITHM: 35
  - BACKBONE: 11
  - DATASET: 1
- Number of checkpoints: 610
  - [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)
- [DATASET] ImageNet-S (3 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)
- [BACKBONE] PoolFormer (5 ckpts)
- [ALGORITHM] PSANet (16 ckpts)
- [ALGORITHM] PSPNet (54 ckpts)
- [BACKBONE] ResNeSt (8 ckpts)
- [ALGORITHM] SegFormer (13 ckpts)
- [ALGORITHM] Segmenter (5 ckpts)
- [ALGORITHM] SegNeXt (4 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

EIGHT

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.

8.1 Train on a single machine

8.1.1 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}`.

8.1.2 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.

8.1.3 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
```

8.1.4 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
```

8.2 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.

8.3 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

NINE

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.

9.1 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_512x512_20k_voc12aug.py \
    checkpoints/pspnet_r50-d8_512x512_20k_voc12aug_20200617_101958-ed5dfbd9.pth \
    --eval mIoU
```

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

TEN

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.

10.1 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.

10.2 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`.

10.3 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.
```

10.4 FAQ

10.4.1 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.

10.4.2 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

ELEVEN

TUTORIAL 2: CUSTOMIZE DATASETS

11.1 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 training 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_dataloader` 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_dataloader` 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`

11.2 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.

11.3 Customize datasets by mixing dataset

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

11.3.1 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
        )
    )
```

11.3.2 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
)
```

11.3.3 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

TWELVE

TUTORIAL 3: CUSTOMIZE DATA PIPELINES

12.1 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.

12.1.1 Data loading

`LoadImageFromFile`

- add: img, img_shape, ori_shape

`LoadAnnotations`

- add: gt_semantic_seg, seg_fields

12.1.2 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

12.1.3 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`

12.1.4 Test time augmentation

`MultiScaleFlipAug`

12.2 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

THIRTEEN

TUTORIAL 4: CUSTOMIZE MODELS

13.1 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.

13.2 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
```

13.3 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.

13.3.1 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),
    ...
```

13.3.2 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)))
```

13.3.3 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

FOURTEEN

TUTORIAL 5: TRAINING TRICKS

MMSegmentation support following training tricks out of box.

14.1 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.

14.2 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.

14.3 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.

14.4 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.

14.5 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

FIFTEEN

TUTORIAL 6: CUSTOMIZE RUNTIME SETTINGS

15.1 Customize optimization settings

15.1.1 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.

15.1.2 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)
```

15.1.3 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.

15.1.4 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,
)
```

15.2 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)
```

15.3 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.

15.4 Customize hooks

15.4.1 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')
]
```

15.4.2 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

SIXTEEN

USEFUL TOOLS

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

16.1 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.

16.2 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`.

16.3 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.

16.4 Evaluate ONNX model

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

16.4.1 Prerequisite

- Install onnx and onnxruntime-gpu

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

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

16.4.2 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.

16.4.3 Results and Models

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

16.5 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
```

16.6 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

SEVENTEEN

MISCELLANEOUS

17.1 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.

17.2 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
  ```

17.3 Model conversion

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

17.3.1 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

EIGHTEEN

MODEL SERVING

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

18.1 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.

18.2 2. Build `mmseg-serve` docker image

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

18.3 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

18.4 4. Test deployment

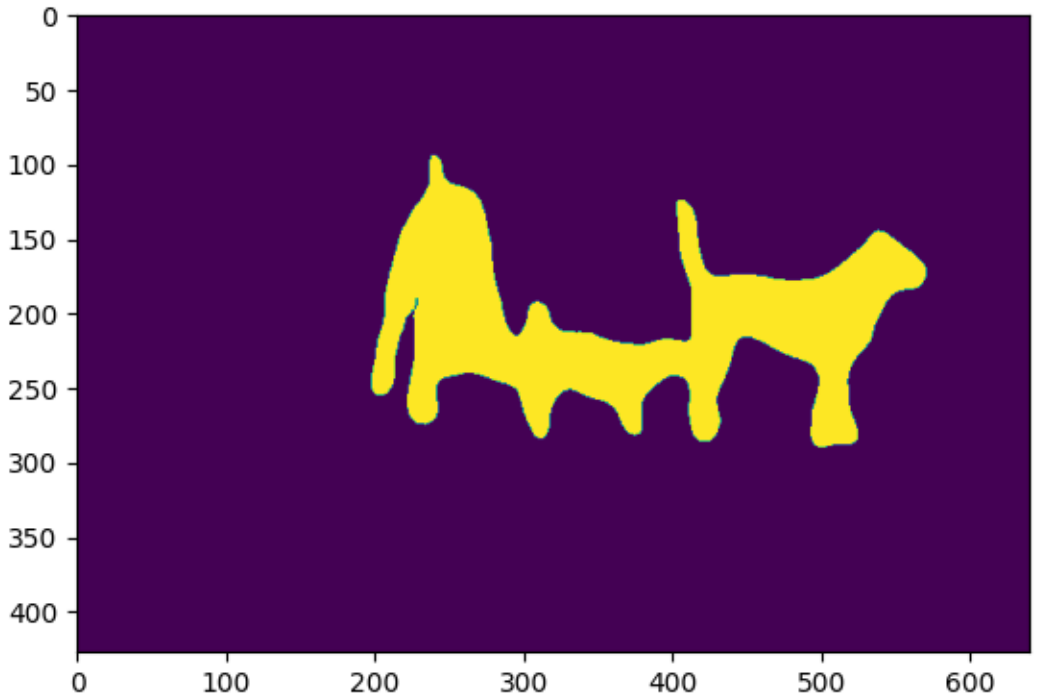
```
curl -O https://raw.githubusercontent.com/open-mmlab/mmsegmentation/master/resources/
↪3dogs.jpg
curl http://127.0.0.1:8080/predictions/${MODEL_NAME} -T 3dogs.jpg -o 3dogs_mask.png
```

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

NINETEEN

CONFUSION MATRIX

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

19.1 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
```

19.2 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

TWENTY

MODEL ENSEMBLE

To complete the integration of prediction probabilities for multiple models, we provide 'tools/model_ensemble.py'

20.1 Usage

```
python tools/model_ensemble.py \
  --config ${CONFIG_FILE1} ${CONFIG_FILE2} ... \
  --checkpoint ${CHECKPOINT_FILE1} ${CHECKPOINT_FILE2} ...\
  --aug-test \
  --out ${OUTPUT_DIR}\
  --gpus ${GPU_USED}\
```

20.2 Description of all arguments

- `--config`: Path to the config file for the ensemble model
- `--checkpoint`: Path to the checkpoint file for the ensemble model
- `--aug-test`: Whether to use flip and multi-scale test
- `--out`: Save folder for model ensemble results
- `--gpus`: Gpu-id used for model ensemble

20.3 Result of model ensemble

- The model ensemble will generate an unrendered segmentation mask for each input, the input shape is `[H, W]`, the segmentation mask shape is `[H, W]`, and each pixel-value in the segmentation mask represents the pixel category after segmentation at that position.
- The filename of the model ensemble result will be named in the same filename as `Ground Truth`. If the filename of `Ground Truth` is called `1.png`, the model ensemble result file will also be named `1.png` and placed in the folder specified by `--out`.

CHAPTER

TWENTYONE

CHANGELOG

21.1 V0.30.0 (01/09/2023)

New Features

- Support Delving into High-Quality Synthetic Face Occlusion Segmentation Datasets (#2194)

Bug Fixes

- Fix incorrect `test_cfg` setting in UNet base configs (#2347)
- Fix KNet `IterativeDecodeHead` bug in master branch (#2333)
- Fix deadlock issue related with MMSegWandbHook (#2398)

Enhancement

- Update CI and pre-commit checking (#2309,#2331)
- Add `Projects/` folder, and the first example project in 0.x (#2457)
- Fix the deprecation of `np.float` and CI configuration problems (#2451)

Documentation

- Add high quality synthetic face occlusion dataset link to readme (#2453)
- Fix the docstring error in the `PascalContextDataset59` class (#2450)

Contributors

- @smttsp made their first contribution in https://github.com/open-mmlab/mmsegmentation/pull/2347
- @MilkClouds made their first contribution in https://github.com/open-mmlab/mmsegmentation/pull/2398
- @Spritea made their first contribution in https://github.com/open-mmlab/mmsegmentation/pull/2450

21.2 V0.29.1 (11/3/2022)

New Features

- Add model ensemble tools (#2218)

Bug Fixes

- Use SyncBN in MobileNetV2 (#2207)

Documentation

- Update FAQ doc about binary segmentation and ReduceZeroLabel (#2206)

- Fix typos (#2249)
- Fix model results (#2190, #2114)

Contributors

- @isLinXu made their first contribution in https://github.com/open-mmlab/mmsegmentation/pull/2219
- @zhijiejia made their first contribution in https://github.com/open-mmlab/mmsegmentation/pull/2218
- @lee-jinhee made their first contribution in https://github.com/open-mmlab/mmsegmentation/pull/2249

21.3 V0.29.0 (10/10/2022)

New Features

- Support PoolFormer (CVPR'2022) (#1537)

Enhancement

- Improve structure and readability for FCNHead (#2142)
- Support IterableDataset in distributed training (#2151)
- Upgrade .dev scripts (#2020)
- Upgrade pre-commit hooks (#2155)

Bug Fixes

- Fix mmseg.api.inference inference_segmentor (#1849)
- fix bug about label_map in evaluation part (#2075)
- Add missing dependencies to torchserve docker file (#2133)
- Fix ddp unittest (#2060)

Contributors

- @jinwonkim93 made their first contribution in https://github.com/open-mmlab/mmsegmentation/pull/1849
- @rlatjcj made their first contribution in https://github.com/open-mmlab/mmsegmentation/pull/2075
- @ShirleyWangCVR made their first contribution in https://github.com/open-mmlab/mmsegmentation/pull/2151
- @mangelroman made their first contribution in https://github.com/open-mmlab/mmsegmentation/pull/2133

21.4 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

21.5 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

21.6 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

21.7 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

21.8 V0.24.1 (5/1/2022)

Bug Fixes

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

21.9 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

21.10 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

21.11 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)

21.12 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

21.13 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)

21.14 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

21.15 V0.20.2 (12/15/2021)

Bug Fixes

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

21.16 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)

21.17 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

21.18 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
- @VVsssssk
- @MengzhangLI
- @Junjun2016

21.19 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)

21.20 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)

21.21 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

21.22 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)

21.23 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)

21.24 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)

21.25 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)

21.26 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)

21.27 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)

21.28 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)

21.29 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)

21.30 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)

21.31 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)

21.32 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

TWENTYTWO

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.

22.1 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.

22.2 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.

22.3 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.

22.4 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 been 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.

22.5 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
```

22.6 How to handle binary segmentation task

MMSegmentation uses `num_classes` and `out_channels` to control output of last layer `self.conv_seg`. More details could be found here.

`num_classes` should be the same as number of types of labels, in binary segmentation task, dataset only has two types of labels: foreground and background, so `num_classes=2`. `out_channels` controls the output channel of last layer of model, it usually equals to `num_classes`. But in binary segmentation task, there are two solutions:

- Set `out_channels=2`, using Cross Entropy Loss in training, using `F.softmax()` and `argmax()` to get prediction of each pixel in inference.
- Set `out_channels=1`, using Binary Cross Entropy Loss in training, using `F.sigmoid()` and `threshold` to get prediction of each pixel in inference. `threshold` is set 0.3 as default.

In summary, to implement binary segmentation methods users should modify below parameters in the `decode_head` and `auxiliary_head` configs. Here is a modification example of pspnet_unet_s5-d16.py:

- (1) `num_classes=2`, `out_channels=2` and `use_sigmoid=False` in `CrossEntropyLoss`.

```
decode_head=dict(
    type='PSPHead',
    in_channels=64,
    in_index=4,
    num_classes=2,
    out_channels=2,
    loss_decode=dict(
        type='CrossEntropyLoss', use_sigmoid=False, loss_weight=1.0)),
auxiliary_head=dict(
    type='FCNHead',
    in_channels=128,
    in_index=3,
    num_classes=2,
    out_channels=2,
```

(continues on next page)

(continued from previous page)

```
    loss_decode=dict(
        type='CrossEntropyLoss', use_sigmoid=False, loss_weight=0.4)),
```

- (2) `num_classes=2`, `out_channels=1` and `use_sigmoid=True` in `CrossEntropyLoss`.

```
decode_head=dict(
    type='PSPHead',
    in_channels=64,
    in_index=4,
    num_classes=2,
    out_channels=1,
    loss_decode=dict(
        type='CrossEntropyLoss', use_sigmoid=True, loss_weight=1.0)),
auxiliary_head=dict(
    type='FCNHead',
    in_channels=128,
    in_index=3,
    num_classes=2,
    out_channels=1,
    loss_decode=dict(
        type='CrossEntropyLoss', use_sigmoid=True, loss_weight=0.4)),
```

22.7 What does `reduce_zero_label` work for?

When loading annotation in MMSegmentation, `reduce_zero_label (bool)` is provided to determine whether reduce all label value by 1:

```
if self.reduce_zero_label:
    # avoid using underflow conversion
    gt_semantic_seg[gt_semantic_seg == 0] = 255
    gt_semantic_seg = gt_semantic_seg - 1
    gt_semantic_seg[gt_semantic_seg == 254] = 255
```

Noted: Please pay attention to label numbers of dataset when using `reduce_zero_label`. If dataset only has two types of labels (i.e., label 0 and 1), it needs to close `reduce_zero_label`, i.e., set `reduce_zero_label=False`.

CHAPTER

TWENTYTHREE

NPU (HUAWEI ASCEND)

23.1 Usage

Please refer to the building documentation of MMCV to install MMCV and MMEngine on NPU devices.

Here we use 4 NPUs on your computer to train the model with the following command:

```
bash tools/dist_train.sh configs/deeplabv3/deeplabv3_r50-d8_512x1024_40k_cityscapes.py 4
```

Also, you can use only one NPU to train the model with the following command:

```
python tools/train.py configs/deeplabv3/deeplabv3_r50-d8_512x1024_40k_cityscapes.py
```

23.2 Models Results

Notes:

- If not specially marked, the results on NPU with amp are the basically same as those on the GPU with FP32.

All above models are provided by Huawei Ascend group.

CHAPTER

TWENTYFOUR

ENGLISH

CHAPTER

TWENTYFIVE

CHAPTER

TWENTYSIX

MMSEG.APIS

CHAPTER

TWENTYSEVEN

MMSEG.CORE

27.1 seg

27.2 evaluation

27.3 utils

CHAPTER

TWENTYEIGHT

MMSEG.DATASETS

28.1 datasets

28.2 pipelines

CHAPTER

TWENTYNINE

MMSEG.MODELS

29.1 segmentors

29.2 backbones

29.3 decode_heads

29.4 losses

CHAPTER

THIRTY

INDICES AND TABLES

- genindex
- search