# MMDetection

*Release 1.0.0*

**May 08, 2020**

# Contents

CHAPTER 1

# Installation

## 1.1 Requirements

- Linux (Windows is not officially supported)
- Python 3.5+
- PyTorch 1.1 or higher
- CUDA 9.0 or higher
- NCCL 2
- GCC 4.9 or higher
- mmcv

We have tested the following versions of OS and softwares:

- OS: Ubuntu 16.04/18.04 and CentOS 7.2
- CUDA: 9.0/9.2/10.0/10.1
- NCCL: 2.1.15/2.2.13/2.3.7/2.4.2
- GCC(G++): 4.9/5.3/5.4/7.3

## 1.2 Install mmdetection

a. Create a conda virtual environment and activate it.

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

b. Install PyTorch and torchvision following the official instructions, e.g.,

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

c. Clone the mmdetection repository.

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

d. Install build requirements and then install mmdetection. (We install pycocotools via the github repo instead of pypi because the pypi version is old and not compatible with the latest numpy.)

```
pip install -r requirements/build.txt
pip install "git+https://github.com/cocodataset/cocoapi.git#subdirectory=PythonAPI"
pip install -v -e .  # or "python setup.py develop"
```

Note:

1. The git commit id will be written to the version number with step d, e.g. 0.6.0+2e7045c. The version will also be saved in trained models. It is recommended that you run step d each time you pull some updates from github. If C++/CUDA codes are modified, then this step is compulsory.
2. Following the above instructions, mmdetection is installed on `dev` mode, any local modifications made to the code will take effect without the need to reinstall it (unless you submit some commits and want to update the version number).
3. If you would like to use `opencv-python-headless` instead of `opencv-python`, you can install it before installing MMCV.
4. Some dependencies are optional. Simply running `pip install -v -e .` will only install the minimum runtime requirements. To use optional dependencies like `albumentations` and `imagecorruptions` either install them manually with `pip install -r requirements/optional.txt` or specify desired extras when calling `pip` (e.g. `pip install -v -e .[optional]`). Valid keys for the extras field are: `all`, `tests`, `build`, and `optional`.

## 1.3 Another option: Docker Image

We provide a Dockerfile to build an image.

```
# build an image with PyTorch 1.1, CUDA 10.0 and CUDNN 7.5
docker build -t mmdetection docker/
```

## 1.4 Prepare datasets

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

```
mmdetection
├── mmdet
├── tools
├── configs
├── data
│   ├── coco
│   │   ├── annotations
│   │   ├── train2017
```

(continues on next page)

(continued from previous page)

```
│   │   ├── val2017
│   │   └── test2017
│   ├── cityscapes
│   │   ├── annotations
│   │   ├── leftImg8bit
│   │   │   ├── train
│   │   │   └── val
│   │   └── gtFine
│   │   │   ├── train
│   │   │   └── val
│   ├── VOCdevkit
│   │   ├── VOC2007
│   │   └── VOC2012
```

The cityscapes annotations have to be converted into the coco format using `tools/convert_datasets/cityscapes.py`:

```
pip install cityscapesscripts
python tools/convert_datasets/cityscapes.py ./data/cityscapes --nproc 8 --out_dir ./
↪data/cityscapes/annotations
```

Current the config files in `cityscapes` use COCO pre-trained weights to initialize. You could download the pre-trained models in advance if network is unavailable or slow, otherwise it would cause errors at the beginning of training.

## 1.5 A from-scratch setup script

Here is a full script for setting up mmdetection with conda and link the dataset path (supposing that your COCO dataset path is $COCO_ROOT).

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

conda install -c pytorch pytorch torchvision -y
git clone https://github.com/open-mmlab/mmdetection.git
cd mmdetection
pip install -r requirements/build.txt
pip install "git+https://github.com/cocodataset/cocoapi.git#subdirectory=PythonAPI"
pip install -v -e .

mkdir data
ln -s $COCO_ROOT data
```

## 1.6 Using multiple MMDetection versions

If there are more than one mmdetection on your machine, and you want to use them alternatively, the recommended way is to create multiple conda environments and use different environments for different versions.

Another way is to insert the following code to the main scripts (`train.py`, `test.py` or any other scripts you run)

```
import os.path as osp
import sys
sys.path.insert(0, osp.join(osp.dirname(osp.abspath(__file__)), '../'))
```

Or run the following command in the terminal of corresponding folder to temporally use the current one.

```
export PYTHONPATH=`pwd`:$PYTHONPATH
```

CHAPTER 2

# Getting Started

This page provides basic tutorials about the usage of MMDetection. For installation instructions, please see *INSTALL.md*.

## 2.1 Inference with pretrained models

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

### 2.1.1 Test a dataset

- [x] single GPU testing
- [x] multiple GPU testing
- [x] visualize detection results

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]

# 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.

- `EVAL_METRICS`: Items to be evaluated on the results. Allowed values depend on the dataset, e.g., `proposal_fast`, `proposal`, `bbox`, `segm` are available for COCO, `mAP`, `recall` for PASCAL VOC. Cityscapes could be evaluated by `cityscapes` as well as all COCO metrics.
- `--show`: If specified, detection 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`.

If you would like to evaluate the dataset, do not specify `--show` at the same time.

Examples:

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

1. Test Faster R-CNN and visualize the results. Press any key for the next image.

```
python tools/test.py configs/faster_rcnn_r50_fpn_1x.py \
    checkpoints/faster_rcnn_r50_fpn_1x_20181010-3d1b3351.pth \
    --show
```

1. Test Faster R-CNN on PASCAL VOC (without saving the test results) and evaluate the mAP.

```
python tools/test.py configs/pascal_voc/faster_rcnn_r50_fpn_1x_voc.py \
    checkpoints/SOME_CHECKPOINT.pth \
    --eval mAP
```

1. Test Mask R-CNN with 8 GPUs, and evaluate the bbox and mask AP.

```
./tools/dist_test.sh configs/mask_rcnn_r50_fpn_1x.py \
    checkpoints/mask_rcnn_r50_fpn_1x_20181010-069fa190.pth \
    8 --out results.pkl --eval bbox segm
```

1. Test Mask R-CNN on COCO test-dev with 8 GPUs, and generate the json file to be submit to the official evaluation server.

```
./tools/dist_test.sh configs/mask_rcnn_r50_fpn_1x.py \
    checkpoints/mask_rcnn_r50_fpn_1x_20181010-069fa190.pth \
    8 --format_only --options "jsonfile_prefix=./mask_rcnn_test-dev_results"
```

You will get two json files `mask_rcnn_test-dev_results.bbox.json` and `mask_rcnn_test-dev_results.segm.json`.

1. Test Mask R-CNN on Cityscapes test with 8 GPUs, and generate the txt and png files to be submit to the official evaluation server.

```
./tools/dist_test.sh configs/cityscapes/mask_rcnn_r50_fpn_1x_cityscapes.py \
    checkpoints/mask_rcnn_r50_fpn_1x_cityscapes_20200227-afe51d5a.pth \
    8  --format_only --options "txtfile_prefix=./mask_rcnn_cityscapes_test_results"
```

The generated png and txt would be under `./mask_rcnn_cityscapes_test_results` directory.

### 2.1.2 Webcam demo

We provide a webcam demo to illustrate the results.

```
python demo/webcam_demo.py ${CONFIG_FILE} ${CHECKPOINT_FILE} [--device ${GPU_ID}] [--
↪camera-id ${CAMERA-ID}] [--score-thr ${SCORE_THR}]
```

Examples:

```
python demo/webcam_demo.py configs/faster_rcnn_r50_fpn_1x.py \
    checkpoints/faster_rcnn_r50_fpn_1x_20181010-3d1b3351.pth
```

### 2.1.3 High-level APIs for testing images

#### Synchronous interface

Here is an example of building the model and test given images.

```
from mmdet.apis import init_detector, inference_detector, show_result
import mmcv

config_file = 'configs/faster_rcnn_r50_fpn_1x.py'
checkpoint_file = 'checkpoints/faster_rcnn_r50_fpn_1x_20181010-3d1b3351.pth'

# build the model from a config file and a checkpoint file
model = init_detector(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_detector(model, img)
# visualize the results in a new window
show_result(img, result, model.CLASSES)
# or save the visualization results to image files
show_result(img, result, model.CLASSES, out_file='result.jpg')

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

A notebook demo can be found in demo/inference_demo.ipynb.

#### Asynchronous interface - supported for Python 3.7+

Async interface allows not to block CPU on GPU bound inference code and enables better CPU/GPU utilization for single threaded application. Inference can be done concurrently either between different input data samples or between different models of some inference pipeline.

See `tests/async_benchmark.py` to compare the speed of synchronous and asynchronous interfaces.

```
import asyncio
import torch
from mmdet.apis import init_detector, async_inference_detector, show_result
from mmdet.utils.contextmanagers import concurrent

async def main():
    config_file = 'configs/faster_rcnn_r50_fpn_1x.py'
    checkpoint_file = 'checkpoints/faster_rcnn_r50_fpn_1x_20181010-3d1b3351.pth'
    device = 'cuda:0'
    model = init_detector(config_file, checkpoint=checkpoint_file, device=device)

    # queue is used for concurrent inference of multiple images
```

(continues on next page)

(continued from previous page)

```
    streamqueue = asyncio.Queue()
    # queue size defines concurrency level
    streamqueue_size = 3

    for _ in range(streamqueue_size):
        streamqueue.put_nowait(torch.cuda.Stream(device=device))

    # test a single image and show the results
    img = 'test.jpg'  # or img = mmcv.imread(img), which will only load it once

    async with concurrent(streamqueue):
        result = await async_inference_detector(model, img)

    # visualize the results in a new window
    show_result(img, result, model.CLASSES)
    # or save the visualization results to image files
    show_result(img, result, model.CLASSES, out_file='result.jpg')

asyncio.run(main())
```

## 2.2 Train a model

MMDetection 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 each epoch, you can change the evaluation interval by adding the interval argument in the training config.

```
evaluation = dict(interval=12)  # This evaluate the model per 12 epoch.
```

***Important***: The default learning rate in config files is for 8 GPUs and 2 img/gpu (batch size = 8*2 = 16). According to the Linear Scaling Rule, you need to set the learning rate proportional to the batch size if you use different GPUs or images per GPU, e.g., lr=0.01 for 4 GPUs * 2 img/gpu and lr=0.08 for 16 GPUs * 4 img/gpu.

### 2.2.1 Train with a single GPU

```
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}`.

### 2.2.2 Train with multiple GPUs

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

Optional arguments are:

- `--validate` (**strongly recommended**): Perform evaluation at every k (default value is 1, which can be modified like this) epochs during the training.

- `--work_dir ${WORK_DIR}`: Override the working directory specified in the config file.
- `--resume_from ${CHECKPOINT_FILE}`: Resume from a previous checkpoint file.

Difference between `resume_from` and `load_from`: `resume_from` loads both the model weights and optimizer status, and the epoch is also inherited from the specified checkpoint. It is usually used for resuming the training process that is interrupted accidentally. `load_from` only loads the model weights and the training epoch starts from 0. It is usually used for finetuning.

### 2.2.3 Train with multiple machines

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

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

Here is an example of using 16 GPUs to train Mask R-CNN on the dev partition.

```
./tools/slurm_train.sh dev mask_r50_1x configs/mask_rcnn_r50_fpn_1x.py /nfs/xxxx/mask_
↪rcnn_r50_fpn_1x 16
```

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

If you have just multiple machines connected with ethernet, you can refer to pytorch launch utility. Usually it is slow if you do not have high speed networking like infiniband.

### 2.2.4 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.

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

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

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

In `config1.py`,

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

In `config2.py`,

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

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

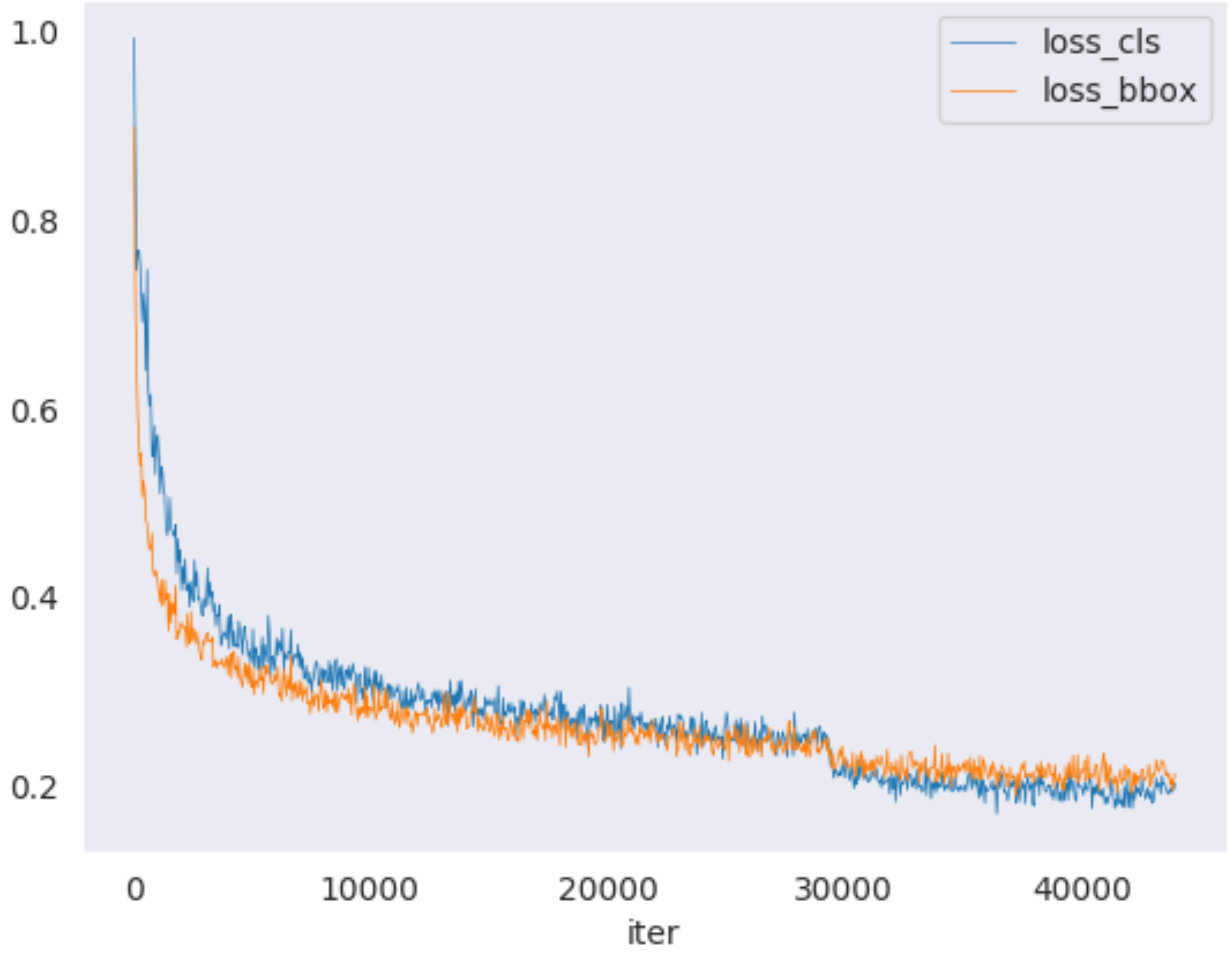
```
CUDA_VISIBLE_DEVICES=0,1,2,3 ./tools/slurm_train.sh ${PARTITION} ${JOB_NAME} config1.
↪py ${WORK_DIR} 4
CUDA_VISIBLE_DEVICES=4,5,6,7 ./tools/slurm_train.sh ${PARTITION} ${JOB_NAME} config2.
↪py ${WORK_DIR} 4
```

## 2.3 Useful tools

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

### 2.3.1 Analyze logs

You can plot loss/mAP curves given a training log file. Run `pip install seaborn` first to install the dependency.




loss curve image

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

Examples:

- Plot the classification loss of some run.

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

- Plot the classification and regression loss of some run, and save the figure to a pdf.

```
python tools/analyze_logs.py plot_curve log.json --keys loss_cls loss_reg --out␣
↪losses.pdf
```

- Compare the bbox mAP of two runs in the same figure.

```
python tools/analyze_logs.py plot_curve log1.json log2.json --keys bbox_mAP --legend␣
↪run1 run2
```

You can also compute the average training speed.

```
python tools/analyze_logs.py cal_train_time ${CONFIG_FILE} [--include-outliers]
```

The output is expected to be like the following.

```
-----Analyze train time of work_dirs/some_exp/20190611_192040.log.json-----
slowest epoch 11, average time is 1.2024
fastest epoch 1, average time is 1.1909
time std over epochs is 0.0028
average iter time: 1.1959 s/iter
```

### 2.3.2 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, 1280, 800)
Flops: 239.32 GMac
Params: 37.74 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. You can add support for new operators by modifying `mmdet/utils/flops_counter.py`. (3) The FLOPs of two-stage detectors is dependent on the number of proposals.

### 2.3.3 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/faster_rcnn/latest.pth faster_rcnn_r50_fpn_1x_
↪20190801.pth
```

The final output filename will be `faster_rcnn_r50_fpn_1x_20190801-{hash id}.pth`.

### 2.3.4 Test the robustness of detectors

Please refer to ROBUSTNESS_BENCHMARKING.md.

### 2.3.5 Convert to ONNX (experimental)

We provide a script to convert model to ONNX format. The converted model could be visualized by tools like Netron.

```
python tools/pytorch2onnx.py ${CONFIG_FILE} ${CHECKPOINT_FILE} --out ${ONNX_FILE} [--
↪shape ${INPUT_SHAPE}]
```

**Note**: This tool is still experimental. Customized operators are not supported for now. We set `use_torchvision=True` on-the-fly for `RoIPool` and `RoIAlign`.

## 2.4 How-to

### 2.4.1 Use my own datasets

The simplest way is to convert your dataset to existing dataset formats (COCO or PASCAL VOC).

Here we show an example of adding a custom dataset of 5 classes, assuming it is also in COCO format.

In `mmdet/datasets/my_dataset.py`:

```
from .coco import CocoDataset
from .registry import DATASETS

@DATASETS.register_module
class MyDataset(CocoDataset):

    CLASSES = ('a', 'b', 'c', 'd', 'e')
```

In `mmdet/datasets/__init__.py`:

```
from .my_dataset import MyDataset
```

Then you can use `MyDataset` in config files, with the same API as CocoDataset.

It is also fine if you do not want to convert the annotation format to COCO or PASCAL format. Actually, we define a simple annotation format and all existing datasets are processed to be compatible with it, either online or offline.

The annotation of a dataset is a list of dict, each dict corresponds to an image. There are 3 field `filename` (relative path), `width`, `height` for testing, and an additional field `ann` for training. `ann` is also a dict containing at least 2 fields: `bboxes` and `labels`, both of which are numpy arrays. Some datasets may provide annotations like crowd/difficult/ignored bboxes, we use `bboxes_ignore` and `labels_ignore` to cover them.

Here is an example.

```
[
    {
        'filename': 'a.jpg',
        'width': 1280,
        'height': 720,
        'ann': {
            'bboxes': <np.ndarray, float32> (n, 4),
```

(continues on next page)

(continued from previous page)

```
            'labels': <np.ndarray, int64> (n, ),
            'bboxes_ignore': <np.ndarray, float32> (k, 4),
            'labels_ignore': <np.ndarray, int64> (k, ) (optional field)
        }
    },
    ...
]
```

There are two ways to work with custom datasets.

- online conversion

  You can write a new Dataset class inherited from `CustomDataset`, and overwrite two methods `load_annotations(self, ann_file)` and `get_ann_info(self, idx)`, like CocoDataset and VOCDataset.

- offline conversion

  You can convert the annotation format to the expected format above and save it to a pickle or json file, like pascal_voc.py. Then you can simply use `CustomDataset`.

### 2.4.2 Customize optimizer

An example of customized optimizer `CopyOfSGD` is defined in `mmdet/core/optimizer/copy_of_sgd.py`. More generally, a customized optimizer could be defined as following.

In `mmdet/core/optimizer/my_optimizer.py`:

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

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

In `mmdet/core/optimizer/__init__.py`:

```
from .my_optimizer import MyOptimizer
```

Then you can use `MyOptimizer` in `optimizer` field of config files.

### 2.4.3 Develop new components

We basically categorize model components into 4 types.

- backbone: usually an FCN network to extract feature maps, e.g., ResNet, MobileNet.
- neck: the component between backbones and heads, e.g., FPN, PAFPN.
- head: the component for specific tasks, e.g., bbox prediction and mask prediction.
- roi extractor: the part for extracting RoI features from feature maps, e.g., RoI Align.

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

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

```
import torch.nn as nn

from ..registry 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
```

1. Import the module in `mmdet/models/backbones/__init__.py`.

```
from .mobilenet import MobileNet
```

1. Use it in your config file.

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

For more information on how it works, you can refer to *TECHNICAL_DETAILS.md* (TODO).

CHAPTER 3

# Benchmark and Model Zoo

## 3.1 Environment

### 3.1.1 Hardware

- 8 NVIDIA Tesla V100 GPUs
- Intel Xeon 4114 CPU @ 2.20GHz

### 3.1.2 Software environment

- Python 3.6 / 3.7
- PyTorch 1.1
- CUDA 9.0.176
- CUDNN 7.0.4
- NCCL 2.1.15

## 3.2 Mirror sites

We use AWS as the main site to host our model zoo, and maintain a mirror on aliyun. You can replace `https://s3.ap-northeast-2.amazonaws.com/open-mmlab` with `https://open-mmlab.oss-cn-beijing.aliyuncs.com` in model urls.

## 3.3 Common settings

- All FPN baselines and RPN-C4 baselines were trained using 8 GPU with a batch size of 16 (2 images per GPU). Other C4 baselines were trained using 8 GPU with a batch size of 8 (1 image per GPU).
- All models were trained on `coco_2017_train`, and tested on the `coco_2017_val`.
- We use distributed training and BN layer stats are fixed.
- We adopt the same training schedules as Detectron. 1x indicates 12 epochs and 2x indicates 24 epochs, which corresponds to slightly less iterations than Detectron and the difference can be ignored.
- All pytorch-style pretrained backbones on ImageNet are from PyTorch model zoo.
- For fair comparison with other codebases, we report the GPU memory as the maximum value of `torch.cuda.max_memory_allocated()` for all 8 GPUs. Note that this value is usually less than what `nvidia-smi` shows.
- We report the inference time as the overall time including data loading, network forwarding and post processing.

## 3.4 Baselines

More models with different backbones will be added to the model zoo.

### 3.4.1 RPN

### 3.4.2 Faster R-CNN

### 3.4.3 Mask R-CNN

### 3.4.4 Fast R-CNN (with pre-computed proposals)

### 3.4.5 RetinaNet

### 3.4.6 Cascade R-CNN

### 3.4.7 Cascade Mask R-CNN

**Notes:**

- The `20e` schedule in Cascade (Mask) R-CNN indicates decreasing the lr at 16 and 19 epochs, with a total of 20 epochs.

### 3.4.8 Hybrid Task Cascade (HTC)

**Notes:**

- Please refer to Hybrid Task Cascade for details and more a powerful model (50.7/43.9).

### 3.4.9 SSD

**Notes:**

- `cudnn.benchmark` is set as `True` for SSD training and testing.
- Inference time is reported for batch size = 1 and batch size = 8.
- The speed on COCO and VOC are different due to model parameters and nms.

### 3.4.10 Group Normalization (GN)

Please refer to Group Normalization for details.

### 3.4.11 Weight Standardization

Please refer to Weight Standardization for details.

### 3.4.12 Deformable Convolution v2

Please refer to Deformable Convolutional Networks for details.

### 3.4.13 CARAFE: Content-Aware ReAssembly of FEatures

Please refer to CARAFE for details.

### 3.4.14 Instaboost

Please refer to Instaboost for details.

### 3.4.15 Libra R-CNN

Please refer to Libra R-CNN for details.

### 3.4.16 Guided Anchoring

Please refer to Guided Anchoring for details.

### 3.4.17 FCOS

Please refer to FCOS for details.

### 3.4.18 FoveaBox

Please refer to FoveaBox for details.

### 3.4.19 RepPoints

Please refer to RepPoints for details.

### 3.4.20 FreeAnchor

Please refer to FreeAnchor for details.

### 3.4.21 Grid R-CNN (plus)

Please refer to Grid R-CNN for details.

### 3.4.22 GHM

Please refer to GHM for details.

### 3.4.23 GCNet

Please refer to GCNet for details.

### 3.4.24 HRNet

Please refer to HRNet for details.

### 3.4.25 Mask Scoring R-CNN

Please refer to Mask Scoring R-CNN for details.

### 3.4.26 Train from Scratch

Please refer to Rethinking ImageNet Pre-training for details.

### 3.4.27 NAS-FPN

Please refer to NAS-FPN for details.

### 3.4.28 ATSS

Please refer to ATSS for details.

### 3.4.29 Other datasets

We also benchmark some methods on PASCAL VOC, Cityscapes and WIDER FACE.

## 3.5 Comparison with Detectron and maskrcnn-benchmark

We compare mmdetection with Detectron and maskrcnn-benchmark. The backbone used is R-50-FPN.

In general, mmdetection has 3 advantages over Detectron.

- **Higher performance** (especially in terms of mask AP)
- **Faster training speed**
- **Memory efficient**

### 3.5.1 Performance

Detectron and maskrcnn-benchmark use caffe-style ResNet as the backbone. We report results using both caffe-style (weights converted from here) and pytorch-style (weights from the official model zoo) ResNet backbone, indicated as *pytorch-style results* / *caffe-style results*.

We find that pytorch-style ResNet usually converges slower than caffe-style ResNet, thus leading to slightly lower results in 1x schedule, but the final results of 2x schedule is higher.

### 3.5.2 Training Speed

The training speed is measure with s/iter. The lower, the better.

*1. Facebook's Big Basin servers (P100/V100) is slightly faster than the servers we use. mmdetection can also run slightly faster on FB's servers.

*2. For fair comparison, we list the caffe-style results here.

### 3.5.3 Inference Speed

The inference speed is measured with fps (img/s) on a single GPU. The higher, the better.

### 3.5.4 Training memory

There is no doubt that maskrcnn-benchmark and mmdetection is more memory efficient than Detectron, and the main advantage is PyTorch itself. We also perform some memory optimizations to push it forward.

Note that Caffe2 and PyTorch have different apis to obtain memory usage with different implementations. For all codebases, `nvidia-smi` shows a larger memory usage than the reported number in the above table.

CHAPTER 4

# Technical Details

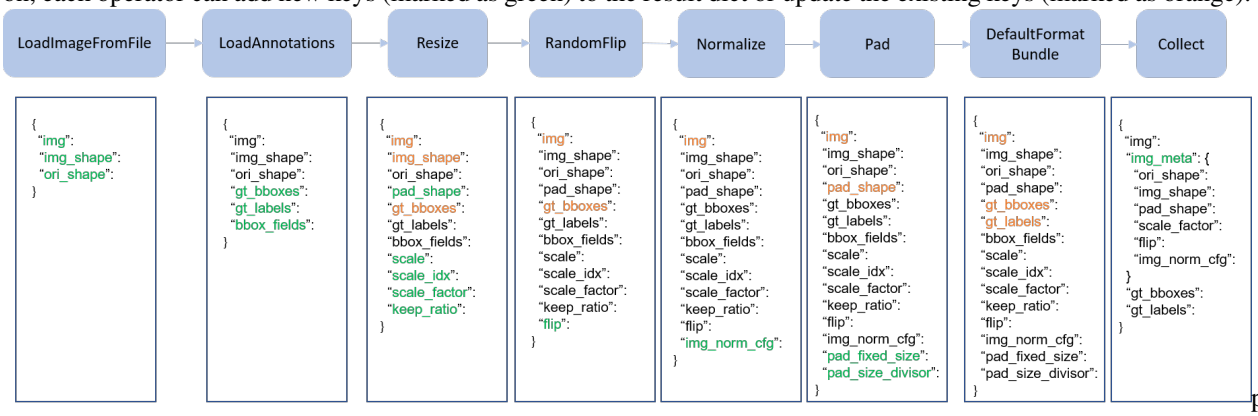
In this section, we will introduce the main units of training a detector: data pipeline, model and iteration pipeline.

## 4.1 Data pipeline

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 object detection may not be the same size (image size, gt bbox size, etc.), 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.

We present a classical pipeline in the following figure. The blue blocks are pipeline operations. With the pipeline going on, each operator can add new keys (marked as green) to the result dict or update the existing keys (marked as orange).



pipeline figure

The operations are categorized into data loading, pre-processing, formatting and test-time augmentation.

Here is an pipeline example for Faster R-CNN.

```
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', with_bbox=True),
    dict(type='Resize', img_scale=(1333, 800), keep_ratio=True),
    dict(type='RandomFlip', flip_ratio=0.5),
    dict(type='Normalize', **img_norm_cfg),
    dict(type='Pad', size_divisor=32),
    dict(type='DefaultFormatBundle'),
    dict(type='Collect', keys=['img', 'gt_bboxes', 'gt_labels']),
]
test_pipeline = [
    dict(type='LoadImageFromFile'),
    dict(
        type='MultiScaleFlipAug',
        img_scale=(1333, 800),
        flip=False,
        transforms=[
            dict(type='Resize', keep_ratio=True),
            dict(type='RandomFlip'),
            dict(type='Normalize', **img_norm_cfg),
            dict(type='Pad', size_divisor=32),
            dict(type='ImageToTensor', keys=['img']),
            dict(type='Collect', keys=['img']),
        ])
]
```

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

### 4.1.1 Data loading

`LoadImageFromFile`

- add: img, img_shape, ori_shape

`LoadAnnotations`

- add: gt_bboxes, gt_bboxes_ignore, gt_labels, gt_masks, gt_semantic_seg, bbox_fields, mask_fields

`LoadProposals`

- add: proposals

### 4.1.2 Pre-processing

`Resize`

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

`RandomFlip`

- add: flip
- update: img, *bbox_fields, *mask_fields, *seg_fields

`Pad`

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

`RandomCrop`

- update: img, pad_shape, gt_bboxes, gt_labels, gt_masks, *bbox_fields

`Normalize`

- add: img_norm_cfg
- update: img

`SegRescale`

- update: gt_semantic_seg

`PhotoMetricDistortion`

- update: img

`Expand`

- update: img, gt_bboxes

`MinIoURandomCrop`

- update: img, gt_bboxes, gt_labels

`Corrupt`

- update: img

### 4.1.3 Formatting

`ToTensor`

- update: specified by `keys`.

`ImageToTensor`

- update: specified by `keys`.

`Transpose`

- update: specified by `keys`.

`ToDataContainer`

- update: specified by `fields`.

`DefaultFormatBundle`

- update: img, proposals, gt_bboxes, gt_bboxes_ignore, gt_labels, gt_masks, 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`

### 4.1.4 Test time augmentation

`MultiScaleFlipAug`

## 4.2 Model

In MMDetection, model components are basically categorized as 4 types.

- backbone: usually a FCN network to extract feature maps, e.g., ResNet.
- neck: the part between backbones and heads, e.g., FPN, ASPP.
- head: the part for specific tasks, e.g., bbox prediction and mask prediction.
- roi extractor: the part for extracting features from feature maps, e.g., RoI Align.

We also write implement some general detection pipelines with the above components, such as `SingleStageDetector` and `TwoStageDetector`.

### 4.2.1 Build a model with basic components

Following some basic pipelines (e.g., two-stage detectors), the model structure can be customized through config files with no pains.

If we want to implement some new components, e.g, the path aggregation FPN structure in Path Aggregation Network for Instance Segmentation, there are two things to do.

1. create a new file in `mmdet/models/necks/pafpn.py`.

```
from ..registry import NECKS

@NECKS.register
class PAFPN(nn.Module):

    def __init__(self,
                 in_channels,
                 out_channels,
                 num_outs,
                 start_level=0,
                 end_level=-1,
                 add_extra_convs=False):
        pass

    def forward(self, inputs):
        # implementation is ignored
        pass
```

2. Import the module in `mmdet/models/necks/__init__.py`.

```
from .pafpn import PAFPN
```

3. modify the config file from

```
neck=dict(
    type='FPN',
    in_channels=[256, 512, 1024, 2048],
    out_channels=256,
    num_outs=5)
```

   to

```
neck=dict(
    type='PAFPN',
    in_channels=[256, 512, 1024, 2048],
    out_channels=256,
    num_outs=5)
```

We will release more components (backbones, necks, heads) for research purpose.

### 4.2.2 Write a new model

To write a new detection pipeline, you need to inherit from `BaseDetector`, which defines the following abstract methods.

- `extract_feat()`: given an image batch of shape (n, c, h, w), extract the feature map(s).
- `forward_train()`: forward method of the training mode
- `simple_test()`: single scale testing without augmentation
- `aug_test()`: testing with augmentation (multi-scale, flip, etc.)

TwoStageDetector is a good example which shows how to do that.

## 4.3 Iteration pipeline

We adopt distributed training for both single machine and multiple machines. Supposing that the server has 8 GPUs, 8 processes will be started and each process runs on a single GPU.

Each process keeps an isolated model, data loader, and optimizer. Model parameters are only synchronized once at the beginning. After a forward and backward pass, gradients will be allreduced among all GPUs, and the optimizer will update model parameters. Since the gradients are allreduced, the model parameter stays the same for all processes after the iteration.

## 4.4 Other information

For more information, please refer to our technical report.

CHAPTER 5

# Changelog

## 5.1 v1.1.0 (24/2/2020)

**Highlights**

- Dataset evaluation is rewritten with a unified api, which is used by both evaluation hooks and test scripts.
- Support new methods: CARAFE.

**Breaking Changes**

- The new MMDDP inherits from the official DDP, thus the `__init__` api is changed to be the same as official DDP.
- The `mask_head` field in HTC config files is modified.
- The evaluation and testing script is updated.
- In all transforms, instance masks are stored as a numpy array shaped (n, h, w) instead of a list of (h, w) arrays, where n is the number of instances.

**Bug Fixes**

- Fix IOU assigners when ignore_iof_thr > 0 and there is no pred boxes. (#2135)
- Fix mAP evaluation when there are no ignored boxes. (#2116)
- Fix the empty RoI input for Deformable RoI Pooling. (#2099)
- Fix the dataset settings for multiple workflows. (#2103)
- Fix the warning related to `torch.uint8` in PyTorch 1.4. (#2105)
- Fix the inference demo on devices other than gpu:0. (#2098)
- Fix Dockerfile. (#2097)
- Fix the bug that `pad_val` is unused in Pad transform. (#2093)
- Fix the albumentation transform when there is no ground truth bbox. (#2032)

**Improvements**

- Use torch instead of numpy for random sampling. (#2094)
- Migrate to the new MMDDP implementation in MMCV v0.3. (#2090)
- Add meta information in logs. (#2086)
- Rewrite Soft NMS with pytorch extension and remove cython as a dependency. (#2056)
- Rewrite dataset evaluation. (#2042, #2087, #2114, #2128)
- Use numpy array for masks in transforms. (#2030)

**New Features**

- Implement "CARAFE: Content-Aware ReAssembly of FEatures". (#1583)
- Add `worker_init_fn()` in data_loader when seed is set. (#2066, #2111)
- Add logging utils. (#2035)

## 5.2 v1.0.0 (30/1/2020)

This release mainly improves the code quality and add more docstrings.

**Highlights**

- Documentation is online now: https://mmdetection.readthedocs.io.
- Support new models: ATSS.
- DCN is now available with the api `build_conv_layer` and `ConvModule` like the normal conv layer.
- A tool to collect environment information is available for trouble shooting.

**Bug Fixes**

- Fix the incompatibility of the latest numpy and pycocotools. (#2024)
- Fix the case when distributed package is unavailable, e.g., on Windows. (#1985)
- Fix the dimension issue for `refine_bboxes()`. (#1962)
- Fix the typo when `seg_prefix` is a list. (#1906)
- Add segmentation map cropping to RandomCrop. (#1880)
- Fix the return value of `ga_shape_target_single()`. (#1853)
- Fix the loaded shape of empty proposals. (#1819)
- Fix the mask data type when using albumentation. (#1818)

**Improvements**

- Enhance AssignResult and SamplingResult. (#1995)
- Add ability to overwrite existing module in Registry. (#1982)
- Reorganize requirements and make albumentations and imagecorruptions optional. (#1969)
- Check NaN in `SSDHead`. (#1935)
- Encapsulate the DCN in ResNe(X)t into a ConvModule & Conv_layers. (#1894)
- Refactoring for mAP evaluation and support multiprocessing and logging. (#1889)

- Init the root logger before constructing Runner to log more information. (#1865)
- Split `SegResizeFlipPadRescale` into different existing transforms. (#1852)
- Move `init_dist()` to MMCV. (#1851)
- Documentation and docstring improvements. (#1971, #1938, #1869, #1838)
- Fix the color of the same class for mask visualization. (#1834)
- Remove the option `keep_all_stages` in HTC and Cascade R-CNN. (#1806)

**New Features**

- Add two test-time options `crop_mask` and `rle_mask_encode` for mask heads. (#2013)
- Support loading grayscale images as single channel. (#1975)
- Implement "Bridging the Gap Between Anchor-based and Anchor-free Detection via Adaptive Training Sample Selection". (#1872)
- Add sphinx generated docs. (#1859, #1864)
- Add GN support for flops computation. (#1850)
- Collect env info for trouble shooting. (#1812)

## 5.3 v1.0rc1 (13/12/2019)

The RC1 release mainly focuses on improving the user experience, and fixing bugs.

**Highlights**

- Support new models: FoveaBox, RepPoints and FreeAnchor.
- Add a Dockerfile.
- Add a jupyter notebook demo and a webcam demo.
- Setup the code style and CI.
- Add lots of docstrings and unit tests.
- Fix lots of bugs.

**Breaking Changes**

- There was a bug for computing COCO-style mAP w.r.t different scales (AP_s, AP_m, AP_l), introduced by #621. (#1679)

**Bug Fixes**

- Fix a sampling interval bug in Libra R-CNN. (#1800)
- Fix the learning rate in SSD300 WIDER FACE. (#1781)
- Fix the scaling issue when `keep_ratio=False`. (#1730)
- Fix typos. (#1721, #1492, #1242, #1108, #1107)
- Fix the shuffle argument in `build_dataloader`. (#1693)
- Clip the proposal when computing mask targets. (#1688)
- Fix the "index out of range" bug for samplers in some corner cases. (#1610, #1404)
- Fix the NMS issue on devices other than GPU:0. (#1603)

- Fix SSD Head and GHM Loss on CPU. (#1578)
- Fix the OOM error when there are too many gt bboxes. (#1575)
- Fix the wrong keyword argument `nms_cfg` in HTC. (#1573)
- Process masks and semantic segmentation in Expand and MinIoUCrop transforms. (#1550, #1361)
- Fix a scale bug in the Non Local op. (#1528)
- Fix a bug in transforms when `gt_bboxes_ignore` is None. (#1498)
- Fix a bug when `img_prefix` is None. (#1497)
- Pass the device argument to `grid_anchors` and `valid_flags`. (#1478)
- Fix the data pipeline for test_robustness. (#1476)
- Fix the argument type of deformable pooling. (#1390)
- Fix the coco_eval when there are only two classes. (#1376)
- Fix a bug in Modulated DeformableConv when deformable_group>1. (#1359)
- Fix the mask cropping in RandomCrop. (#1333)
- Fix zero outputs in DeformConv when not running on cuda:0. (#1326)
- Fix the type issue in Expand. (#1288)
- Fix the inference API. (#1255)
- Fix the inplace operation in Expand. (#1249)
- Fix the from-scratch training config. (#1196)
- Fix inplace add in RoIExtractor which cause an error in PyTorch 1.2. (#1160)
- Fix FCOS when input images has no positive sample. (#1136)
- Fix recursive imports. (#1099)

**Improvements**

- Print the config file and mmdet version in the log. (#1721)
- Lint the code before compiling in travis CI. (#1715)
- Add a probability argument for the `Expand` transform. (#1651)
- Update the PyTorch and CUDA version in the docker file. (#1615)
- Raise a warning when specifying `--validate` in non-distributed training. (#1624, #1651)
- Beautify the mAP printing. (#1614)
- Add pre-commit hook. (#1536)
- Add the argument `in_channels` to backbones. (#1475)
- Add lots of docstrings and unit tests, thanks to @Erotemic. (#1603, #1517, #1506, #1505, #1491, #1479, #1477, #1475, #1474)
- Add support for multi-node distributed test when there is no shared storage. (#1399)
- Optimize Dockerfile to reduce the image size. (#1306)
- Update new results of HRNet. (#1284, #1182)
- Add an argument `no_norm_on_lateral` in FPN. (#1240)

- Test the compiling in CI. (#1235)
- Move docs to a separate folder. (#1233)
- Add a jupyter notebook demo. (#1158)
- Support different type of dataset for training. (#1133)
- Use int64_t instead of long in cuda kernels. (#1131)
- Support unsquare RoIs for bbox and mask heads. (#1128)
- Manually add type promotion to make compatible to PyTorch 1.2. (#1114)
- Allowing validation dataset for computing validation loss. (#1093)
- Use `.scalar_type()` instead of `.type()` to suppress some warnings. (#1070)

**New Features**

- Add an option `--with_ap` to compute the AP for each class. (#1549)
- Implement "FreeAnchor: Learning to Match Anchors for Visual Object Detection". (#1391)
- Support Albumentations for augmentations in the data pipeline. (#1354)
- Implement "FoveaBox: Beyond Anchor-based Object Detector". (#1339)
- Support horizontal and vertical flipping. (#1273, #1115)
- Implement "RepPoints: Point Set Representation for Object Detection". (#1265)
- Add test-time augmentation to HTC and Cascade R-CNN. (#1251)
- Add a COCO result analysis tool. (#1228)
- Add Dockerfile. (#1168)
- Add a webcam demo. (#1155, #1150)
- Add FLOPs counter. (#1127)
- Allow arbitrary layer order for ConvModule. (#1078)

## 5.4 v1.0rc0 (27/07/2019)

- Implement lots of new methods and components (Mixed Precision Training, HTC, Libra R-CNN, Guided Anchoring, Empirical Attention, Mask Scoring R-CNN, Grid R-CNN (Plus), GHM, GCNet, FCOS, HRNet, Weight Standardization, etc.). Thank all collaborators!
- Support two additional datasets: WIDER FACE and Cityscapes.
- Refactoring for loss APIs and make it more flexible to adopt different losses and related hyper-parameters.
- Speed up multi-gpu testing.
- Integrate all compiling and installing in a single script.

## 5.5 v0.6.0 (14/04/2019)

- Up to 30% speedup compared to the model zoo.
- Support both PyTorch stable and nightly version.

- Replace NMS and SigmoidFocalLoss with Pytorch CUDA extensions.

## 5.6 v0.6rc0(06/02/2019)

- Migrate to PyTorch 1.0.

## 5.7 v0.5.7 (06/02/2019)

- Add support for Deformable ConvNet v2. (Many thanks to the authors and @chengdazhi)
- This is the last release based on PyTorch 0.4.1.

## 5.8 v0.5.6 (17/01/2019)

- Add support for Group Normalization.
- Unify RPNHead and single stage heads (RetinaHead, SSDHead) with AnchorHead.

## 5.9 v0.5.5 (22/12/2018)

- Add SSD for COCO and PASCAL VOC.
- Add ResNeXt backbones and detection models.
- Refactoring for Samplers/Assigners and add OHEM.
- Add VOC dataset and evaluation scripts.

## 5.10 v0.5.4 (27/11/2018)

- Add SingleStageDetector and RetinaNet.

## 5.11 v0.5.3 (26/11/2018)

- Add Cascade R-CNN and Cascade Mask R-CNN.
- Add support for Soft-NMS in config files.

## 5.12 v0.5.2 (21/10/2018)

- Add support for custom datasets.
- Add a script to convert PASCAL VOC annotations to the expected format.

## 5.13 v0.5.1 (20/10/2018)

- Add BBoxAssigner and BBoxSampler, the `train_cfg` field in config files are restructured.
- `ConvFCRoIHead` / `SharedFCRoIHead` are renamed to `ConvFCBBoxHead` / `SharedFCBBoxHead` for consistency.

CHAPTER 6

# Indices and tables

- genindex
- search