MMRotate

Release 1.0.0rc0

MMRotate Authors

Nov 07, 2022

GET STARTED

CHAPTER

ONE

OVERVIEW (TO BE UPDATED)

This chapter introduces the basic conception of rotated object detection and the framework of MMRotate, and provides links to detailed tutorials about MMRotate.

1.1 What is rotated object detection

1.1.1 Problem definition

Benefiting from the vigorous development of general object detection, most current rotated object detection models are based on classic general object detector. With the development of detection tasks, horizontal boxes have been unable to meet the needs of researchers in some subdivisions. We call it rotating object detection by redefining the object representation and increasing the number of regression degrees of freedom to achieve rotated rectangle, quadrilateral, and even arbitrary shape detection. Performing high-precision rotated object detection more efficiently has become a current research hotspot. The following areas are where rotated object detection has been applied or has great potential: face recognition, scene text, remote sensing, self-driving, medical, robotic grasping, etc.

1.1.2 What is rotated box

The most notable difference between rotated object detection and generic detection is the replacement of horizontal box annotations with rotated box annotations. They are defined as follows:

- Horizontal box: A rectangle with the `width` along the `x-axis` and `height` along the `y-axis`. Usually, it can be represented by the coordinates of 2 diagonal vertices `(x_i, y_i)` (i = 1, 2), or it can be represented by the coordinates of the center point and the `width` and `height`, `(x_center, y_center, width, height)`.
- Rotated box: It is obtained by rotating the horizontal box around the center point by an `angle`, and the definition method of its rotated box is obtained by adding a radian parameter `(x_center, y_center, width, height, theta)`, where `theta = angle * pi / 180`. The unit of `theta` is `rad`. When the rotation angle is a multiple of 90°, the rotated box degenerates into a horizontal box. The rotated box annotations exported by the annotation software are usually polygons, which need to be converted to the rotated box definition method before training.

Note: In MMRotate, angle parameters are in radians.

1.1.3 Rotation direction

A rotated box can be obtained by rotating a horizontal box clockwise or counterclockwise around its center point. The rotation direction is closely related to the choice of the coordinate system. The image space adopts the right-handed coordinate system `(y, x)`, where y is `up->down` and x is `left->right`. There are two opposite directions of rotation:

- ClockwiseCW

Schematic of `CW`

```
0------------------> x (0 rad)
|  A-------------B
|  |             |
|  |     box     h
|  |   angle=0   |
|  D------w------C
v
y (pi/2 rad)
```

Rotation matrix of `CW`

$$\begin{pmatrix} \cos\alpha & -\sin\alpha \\ \sin\alpha & \cos\alpha \end{pmatrix}$$

Rotation transformation of `CW`

$$\begin{aligned} P_A = \begin{pmatrix} x_A \\ y_A \end{pmatrix} &= \begin{pmatrix} x_{center} \\ y_{center} \end{pmatrix} + \begin{pmatrix} \cos\alpha & -\sin\alpha \\ \sin\alpha & \cos\alpha \end{pmatrix} \begin{pmatrix} -0.5w \\ -0.5h \end{pmatrix} \\ &= \begin{pmatrix} x_{center} - 0.5w\cos\alpha + 0.5h\sin\alpha \\ y_{center} - 0.5w\sin\alpha - 0.5h\cos\alpha \end{pmatrix} \end{aligned}$$

- CounterclockwiseCCW

Schematic of `CCW`

```
0------------------> x (0 rad)
|  A-------------B
|  |             |
|  |     box     h
|  |   angle=0   |
|  D------w------C
v
y (-pi/2 rad)
```

Rotation matrix of `CCW`

$$\begin{pmatrix} \cos\alpha & \sin\alpha \\ -\sin\alpha & \cos\alpha \end{pmatrix}$$

Rotation transformation of `CCW`

$$\begin{aligned} P_A = \begin{pmatrix} x_A \\ y_A \end{pmatrix} &= \begin{pmatrix} x_{center} \\ y_{center} \end{pmatrix} + \begin{pmatrix} \cos\alpha & \sin\alpha \\ -\sin\alpha & \cos\alpha \end{pmatrix} \begin{pmatrix} -0.5w \\ -0.5h \end{pmatrix} \\ &= \begin{pmatrix} x_{center} - 0.5w\cos\alpha - 0.5h\sin\alpha \\ y_{center} + 0.5w\sin\alpha - 0.5h\cos\alpha \end{pmatrix} \end{aligned}$$

The operators that can set the rotation direction in MMCV are:

- box_iou_rotated (Defaults to `CW`)
- nms_rotated (Defaults to `CW`)
- RoIAlignRotated (Defaults to `CCW`)
- RiRoIAlignRotated (Defaults to `CCW`).

Note: In MMRotate, the rotation direction of the rotated boxes is `CW`.

1.1.4 Definition of rotated box

Due to the difference in the definition range of `theta`, the following three definitions of the rotated box gradually emerge in rotated object detection:

- $D_{oc'}$: OpenCV Definition, `angle(0, 90°]`, `theta(0, pi / 2]`, The angle between the `width` of the rectangle and the positive semi-axis of x is a positive acute angle. This definition comes from the `cv2.minAreaRect` function in OpenCV, which returns an angle in the range `(0, 90°]`.
- D_{le135}: Long Edge Definition (135°)`angle[-45°, 135°)`, `theta[-pi / 4, 3 * pi / 4)` and `width > height`.
- D_{le90}: Long Edge Definition (90°)`angle[-90°, 90°)`, `theta[-pi / 2, pi / 2)` and `width > height`.

The conversion relationship between the three definitions is not involved in MMRotate, so we will not introduce it much more. Refer to the below blog to dive deeper.

Note: MMRotate supports the above three definitions of rotated box simultaneously, which can be flexibly switched through the configuration file.

It should be noted that if the OpenCV version is less than 4.5.1, the angle range of `cv2.minAreaRect` is between `[-90°, 0°)`. Reference In order to facilitate the distinction, the old version of the OpenCV definition is denoted as D_{oc}.

- $D_{oc'}$: OpenCV definition, `opencv>=4.5.1`, `angle(0, 90°]`, `theta(0, pi / 2]`.
- D_{oc} : Old OpenCV definition, `opencv<4.5.1`, `angle[-90°, 0°)`, `theta[-pi / 2, 0)`.

The conversion relationship between the two OpenCV definitions is as follows:

$$D_{oc'}\left(w_{oc'}, h_{oc'}, \theta_{oc'}\right) = \begin{cases} D_{oc}\left(h_{oc}, w_{oc}, \theta_{oc} + \pi/2\right), otherwise \\ D_{oc}\left(w_{oc}, h_{oc}, \theta_{oc} + \pi\right), \theta_{oc} = -\pi/2 \end{cases}$$

$$D_{oc}\left(w_{oc}, h_{oc}, \theta_{oc}\right) = \begin{cases} D_{oc'}\left(h_{oc'}, w_{oc'}, \theta_{oc'} - \pi/2\right), otherwise \\ D_{oc'}\left(w_{oc'}, h_{oc'}, \theta_{oc'} - \pi\right), \theta_{oc'} = \pi/2 \end{cases}$$

Note: Regardless of the OpenCV version you are using, MMRotate will convert the theta of the OpenCV definition to (0, pi / 2].

1.1.5 Evaluation

The code for evaluating mAP involves the calculation of IoU. We can directly calculate the IoU of the rotated boxes or convert the rotated boxes to a polygons and then calculate the polygons IoU (DOTA online evaluation uses the calculation of polygons IoU).

1.2 What is MMRotate

MMRotate is a toolbox that provides a framework for unified implementation and evaluation of rotated object detection, and below is its whole framework:

MMRotate consists of 4 main parts, `datasets`, `models`, `core` and `apis`.

- `datasets` is for data loading and data augmentation. In this part, we support various datasets for rotated object detection algorithms, useful data augmentation transforms in `pipelines` for pre-processing image.
- `models` contains models and loss functions.
- `core` provides evaluation tools for model training and evaluation.
- `apis` provides high-level APIs for models training, testing, and inference.

The module design of MMRotate is as follows:

The following points need to be noted due to different definitions of rotated box:

- Loading annotations
- Data augmentation
- Assigning samples
- Evaluation

1.3 How to Use this Guide

Here is a detailed step-by-step guide to learn more about MMRotate:

1. For installation instructions, please see install.
2. *get_started* is for the basic usage of MMRotate.
3. Refer to the below tutorials to dive deeper:

- Config
- Customize Dataset
- Customize Model
- Customize Runtime

CHAPTER

TWO

GET STARTED

2.1 Prerequisites

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

MMRotate works on Linux and Windows. It requires Python 3.7+, CUDA 9.2+ and PyTorch 1.6+.

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.

```
conda install pytorch==1.8.0 torchvision==0.9.0 cudatoolkit=10.2 -c pytorch
```

2.2 Installation

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

2.2.1 Best Practices

Step 0. Install MMEngine and MMCV using MIM.

```
pip install -U openmim
mim install mmengine
mim install "mmcv>=2.0.0rc2"
```

Step 1. Install MMDetection as a dependency.

```
mim install 'mmdet>=3.0.0rc2'
```

Optionally, you could also build MMDetection from source in case you want to modify the code:

```
git clone https://github.com/open-mmlab/mmdetection.git -b dev-3.x
# "-b dev-3.x" means checkout to the `dev-3.x` branch.
cd mmdetection
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.
```

Step 2. Install MMRotate.

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

```
git clone https://github.com/open-mmlab/mmrotate.git -b dev-1.x
# "-b dev-1.x" means checkout to the `dev-1.x` branch.
cd mmrotate
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 mmrotate as a dependency or third-party package, install it with pip:

```
pip install mmrotate
```

2.2.2 Verify the installation

To verify whether MMRotate 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 mmrotate --config oriented-rcnn-le90_r50_fpn_1x_dota --dest .
```

The downloading will take several seconds or more, depending on your network environment. When it is done, you will find two files `oriented-rcnn-le90_r50_fpn_1x_dota.py` and `oriented_rcnn_r50_fpn_1x_dota_le90-6d2b2ce0.pth` in your current folder.

Step 2. Verify the inference demo.

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

```
python demo/image_demo.py demo/demo.jpg oriented-rcnn-le90_r50_fpn_1x_dota.py oriented_
↪rcnn_r50_fpn_1x_dota_le90-6d2b2ce0.pth --out-file result.jpg
```

You will see a new image `result.jpg` on your current folder, where rotated bounding boxes are plotted on cars, buses, etc.

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

```
from mmdet.apis import init_detector, inference_detector
import mmrotate

config_file = 'oriented-rcnn-le90_r50_fpn_1x_dota.py'
checkpoint_file = 'oriented_rcnn_r50_fpn_1x_dota_le90-6d2b2ce0.pth'
model = init_detector(config_file, checkpoint_file, device='cuda:0')
inference_detector(model, 'demo/demo.jpg')
```

You will see a list of arrays printed, indicating the detected rotated bounding boxes.

2.2.3 Customize Installation

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.

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 built for PyTorch 1.9.x and CUDA 10.2.

```
pip install mmcv -f https://download.openmmlab.com/mmcv/dist/cu102/torch1.8/index.html
```

Install on Google Colab

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

Step 1. Install MMCV and MMDetection using MIM.

```
!pip3 install -U openmim
!mim install "mmcv>=2.0.0rc2"
!mim install 'mmdet>=3.0.0rc2'
```

Step 2. Install MMRotate from the source.

```
!git clone https://github.com/open-mmlab/mmrotate.git -b dev-1.x
%cd mmrotate
!pip install -e .
```

Step 3. Verification.

```
import mmrotate
print(mmrotate.__version__)
# Example output: 1.x
```

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.

Using MMRotate with Docker

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

```
# build an image with PyTorch 1.6, CUDA 10.1
# If you prefer other versions, just modified the Dockerfile
docker build -t mmrotate docker/
```

Run it with

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

2.2.4 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

TRAIN & TEST

MMRotate provides dozens of pretrained detection models in Model Zoo, and supports multiple standard datasets, including DOTA, HRSC2016, SSDD, HRSID, etc. This note will show how to perform common tasks on these existing models and standard datasets:

3.1 Learn about Configs (To be updated)

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/misc/print_config.py /PATH/TO/CONFIG` to see the complete config. The mmrotate is built upon the mmdet, thus it is highly recommended to learn the basics of mmdet.

3.1.1 Modify a config through script arguments

When submitting jobs using "tools/train.py" or "tools/test.py", you may specify `--cfg-options` to in-place modify the config.

- Update config keys of dict chains.

  The config options can be specified following the order of the dict keys in the original config. For example, `--cfg-options model.backbone.norm_eval=False` changes all BN modules in model backbones to `train` mode.

- Update keys inside a list of configs.

  Some config dicts are composed as a list in your config. For example, the training pipeline `data.train.pipeline` is normally a list e.g. `[dict(type='LoadImageFromFile'), ...]`. If you want to change `'LoadImageFromFile'` to `'LoadImageFromWebcam'` in the pipeline, you may specify `--cfg-options data.train.pipeline.0.type=LoadImageFromWebcam`.

- Update values of list/tuples.

  If the value to be updated is a list or a tuple. For example, the config file normally sets `workflow=[('train', 1)]`. If you want to change this key, you may specify `--cfg-options workflow="[(train,1),(val,1)]"`. Note that the quotation mark " is necessary to support list/tuple data types, and that **NO** white space is allowed inside the quotation marks in the specified value.

3.1.2 Config file naming convention

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

```
{model}_[model setting]_{backbone}_{neck}_[norm setting]_[misc]_[gpu x batch_per_gpu]_
↪{dataset}_{data setting}_{angle version}
```

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

- `{model}`: model type like `rotated_faster_rcnn`, `rotated_retinanet`, etc.
- `[model setting]`: specific setting for some model, like `hbb` for `rotated_retinanet`, etc.
- `{backbone}`: backbone type like `r50` (ResNet-50), `swin_tiny` (SWIN-tiny).
- `{neck}`: neck type like `fpn`, `refpn`.
- `[norm_setting]`: `bn` (Batch Normalization) is used unless specified, other norm layer types could be `gn` (Group Normalization), `syncbn` (Synchronized Batch Normalization). `gn-head`/`gn-neck` indicates GN is applied in head/neck only, while `gn-all` means GN is applied in the entire model, e.g. backbone, neck, head.
- `[misc]`: miscellaneous setting/plugins of the model, e.g. `dconv`, `gcb`, `attention`, `albu`, `mstrain`.
- `[gpu x batch_per_gpu]`: GPUs and samples per GPU, `1xb2` is used by default.
- `{dataset}`: dataset like `dota`.
- `{angle version}`: like `oc`, `le135`, or `le90`.

3.1.3 An example of RotatedRetinaNet

To help the users have a basic idea of a complete config and the modules in a modern detection system, we make brief comments on the config of RotatedRetinaNet using ResNet50 and FPN as the following. For more detailed usage and the corresponding alternative for each module, please refer to the API documentation.

```
angle_version = 'oc'  # The angle version
model = dict(
    type='RotatedRetinaNet',  # The name of detector
    backbone=dict(  # The config of backbone
        type='ResNet',  # The type of the backbone
        depth=50,  # The depth of backbone
        num_stages=4,  # Number of stages of the backbone.
        out_indices=(0, 1, 2, 3),  # The index of output feature maps produced in each␣
↪stages
        frozen_stages=1,  # The weights in the first 1 stage are fronzen
        zero_init_residual=False,  # Whether to use zero init for last norm layer in␣
↪resblocks to let them behave as identity.
        norm_cfg=dict(  # The config of normalization layers.
            type='BN',  # Type of norm layer, usually it is BN or GN
            requires_grad=True),  # Whether to train the gamma and beta in BN
        norm_eval=True,  # 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.
        init_cfg=dict(type='Pretrained', checkpoint='torchvision://resnet50')),  # The␣
↪ImageNet pretrained backbone to be loaded
    neck=dict(
        type='FPN',  # The neck of detector is FPN. We also support 'ReFPN'
```

(continues on next page)

(continued from previous page)

```
        in_channels=[256, 512, 1024, 2048],  # The input channels, this is consistent␣
↪with the output channels of backbone
        out_channels=256,  # The output channels of each level of the pyramid feature map
        start_level=1,  # Index of the start input backbone level used to build the␣
↪feature pyramid
        add_extra_convs='on_input',  # It specifies the source feature map of the extra␣
↪convs
        num_outs=5),  # The number of output scales
    bbox_head=dict(
        type='RotatedRetinaHead',# The type of bbox head is 'RRetinaHead'
        num_classes=15,  # Number of classes for classification
        in_channels=256,  # Input channels for bbox head
        stacked_convs=4,  # Number of stacking convs of the head
        feat_channels=256,  # Number of hidden channels
        assign_by_circumhbbox='oc',  # The angle version of obb2hbb
        anchor_generator=dict(  # The config of anchor generator
            type='RotatedAnchorGenerator',  # The type of anchor generator
            octave_base_scale=4,  # The base scale of octave.
            scales_per_octave=3,  #  Number of scales for each octave.
            ratios=[1.0, 0.5, 2.0],  # The ratio between height and width.
            strides=[8, 16, 32, 64, 128]),  # The strides of the anchor generator. This␣
↪is consistent with the FPN feature strides.
        bbox_coder=dict(  # Config of box coder to encode and decode the boxes during␣
↪training and testing
            type='DeltaXYWHAOBBoxCoder',  # Type of box coder.
            angle_range='oc',  # The angle version of box coder.
            norm_factor=None,  # The norm factor of box coder.
            edge_swap=False,  # The edge swap flag of box coder.
            proj_xy=False,  # The project flag of box coder.
            target_means=(0.0, 0.0, 0.0, 0.0, 0.0),  # The target means used to encode␣
↪and decode boxes
            target_stds=(1.0, 1.0, 1.0, 1.0, 1.0)),  # The standard variance used to␣
↪encode and decode boxes
        loss_cls=dict(  # Config of loss function for the classification branch
            type='FocalLoss',  # Type of loss for classification branch
            use_sigmoid=True,  #  Whether the prediction is used for sigmoid or softmax
            gamma=2.0,  # The gamma for calculating the modulating factor
            alpha=0.25,  # A balanced form for Focal Loss
            loss_weight=1.0),  # Loss weight of the classification branch
        loss_bbox=dict(  # Config of loss function for the regression branch
            type='L1Loss',  # Type of loss
            loss_weight=1.0)),  # Loss weight of the regression branch
    train_cfg=dict(  # Config of training hyperparameters
        assigner=dict(  # Config of assigner
            type='MaxIoUAssigner',  # Type of assigner
            pos_iou_thr=0.5,  # IoU >= threshold 0.5 will be taken as positive samples
            neg_iou_thr=0.4,  # IoU < threshold 0.4 will be taken as negative samples
            min_pos_iou=0,  # The minimal IoU threshold to take boxes as positive samples
            ignore_iof_thr=-1,  # IoF threshold for ignoring bboxes
            iou_calculator=dict(type='RBboxOverlaps2D')),  # Type of Calculator for IoU
        allowed_border=-1,  # The border allowed after padding for valid anchors.
        pos_weight=-1,  # The weight of positive samples during training.
```

(continues on next page)

(continued from previous page)

```
        debug=False),  # Whether to set the debug mode
    test_cfg=dict(  # Config of testing hyperparameters
        nms_pre=2000,  # The number of boxes before NMS
        min_bbox_size=0,  # The allowed minimal box size
        score_thr=0.05,  # Threshold to filter out boxes
        nms=dict(iou_thr=0.1), # NMS threshold
        max_per_img=2000))  # The number of boxes to be kept after NMS.
dataset_type = 'DOTADataset'  # Dataset type, this will be used to define the dataset
data_root = '../datasets/split_1024_dota1_0/'  # 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
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
         with_bbox=True),  # Whether to use bounding box, True for detection
    dict(type='RResize',  # Augmentation pipeline that resize the images and their␣
↪annotations
         img_scale=(1024, 1024)),  # The largest scale of image
    dict(type='RRandomFlip',  # Augmentation pipeline that flip the images and their␣
↪annotations
         flip_ratio=0.5,  # The ratio or probability to flip
         version='oc'),  # The angle version
    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),
    dict(type='Pad',  # Padding config
         size_divisor=32),  # The number the padded images should be divisible
    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 detector
         keys=['img', 'gt_bboxes', 'gt_labels'])
]
test_pipeline = [
    dict(type='LoadImageFromFile'),  # First pipeline to load images from file path
    dict(
        type='MultiScaleFlipAug',  # An encapsulation that encapsulates the testing␣
↪augmentations
        img_scale=(1024, 1024),  # Decides the largest scale for testing, used for the␣
↪Resize pipeline
        flip=False,  # Whether to flip images during testing
        transforms=[
            dict(type='RResize'),  # Use resize augmentation
            dict(
```

(continues on next page)

(continued from previous page)

```
                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='Pad',  # Padding config to pad images divisible by 32.
                 size_divisor=32),
            dict(type='DefaultFormatBundle'),  # Default format bundle to gather data in␣
↪the pipeline
            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='DOTADataset',  # Type of dataset
        ann_file=
        '../datasets/split_1024_dota1_0/trainval/annfiles/',  # Path of annotation file
        img_prefix=
        '../datasets/split_1024_dota1_0/trainval/images/',  # Prefix of image path
        pipeline=[  # pipeline, this is passed by the train_pipeline created before.
            dict(type='LoadImageFromFile'),
            dict(type='LoadAnnotations', with_bbox=True),
            dict(type='RResize', img_scale=(1024, 1024)),
            dict(type='RRandomFlip', flip_ratio=0.5, version='oc'),
            dict(
                type='Normalize',
                mean=[123.675, 116.28, 103.53],
                std=[58.395, 57.12, 57.375],
                to_rgb=True),
            dict(type='Pad', size_divisor=32),
            dict(type='DefaultFormatBundle'),
            dict(type='Collect', keys=['img', 'gt_bboxes', 'gt_labels'])
        ],
        version='oc'),
    val=dict(  # Validation dataset config
        type='DOTADataset',
        ann_file=
        '../datasets/split_1024_dota1_0/trainval/annfiles/',
        img_prefix=
        '../datasets/split_1024_dota1_0/trainval/images/',
        pipeline=[
            dict(type='LoadImageFromFile'),
            dict(
                type='MultiScaleFlipAug',
                img_scale=(1024, 1024),
                flip=False,
                transforms=[
                    dict(type='RResize'),
```

(continues on next page)

(continued from previous page)

```
                    dict(
                        type='Normalize',
                        mean=[123.675, 116.28, 103.53],
                        std=[58.395, 57.12, 57.375],
                        to_rgb=True),
                    dict(type='Pad', size_divisor=32),
                    dict(type='DefaultFormatBundle'),
                    dict(type='Collect', keys=['img'])
                ])
        ],
        version='oc'),
    test=dict(  # Test dataset config, modify the ann_file for test-dev/test submission
        type='DOTADataset',
        ann_file=
        '../datasets/split_1024_dota1_0/test/images/',
        img_prefix=
        '../datasets/split_1024_dota1_0/test/images/',
        pipeline=[  # Pipeline is passed by test_pipeline created before
            dict(type='LoadImageFromFile'),
            dict(
                type='MultiScaleFlipAug',
                img_scale=(1024, 1024),
                flip=False,
                transforms=[
                    dict(type='RResize'),
                    dict(
                        type='Normalize',
                        mean=[123.675, 116.28, 103.53],
                        std=[58.395, 57.12, 57.375],
                        to_rgb=True),
                    dict(type='Pad', size_divisor=32),
                    dict(type='DefaultFormatBundle'),
                    dict(type='Collect', keys=['img'])
                ])
        ],
        version='oc'))
evaluation = dict(  # The config to build the evaluation hook
    interval=12,  # Evaluation interval
    metric='mAP')  # Metrics used during evaluation
optimizer = dict(  # Config used to build optimizer
    type='SGD',  # Type of optimizers
    lr=0.0025,  # Learning rate of optimizers
    momentum=0.9,  # Momentum
    weight_decay=0.0001)  # Weight decay of SGD
optimizer_config = dict(  # Config used to build the optimizer hook
    grad_clip=dict(
        max_norm=35,
        norm_type=2))
lr_config = dict(  # Learning rate scheduler config used to register LrUpdater hook
    policy='step',  # The policy of scheduler
    warmup='linear',  # The warmup policy, also support `exp` and `constant`.
    warmup_iters=500,  # The number of iterations for warmup
```

(continues on next page)

(continued from previous page)

```
    warmup_ratio=0.3333333333333333,  # The ratio of the starting learning rate used for␣
↪warmup
    step=[8, 11])  # Steps to decay the learning rate
runner = dict(
    type='EpochBasedRunner',  # Type of runner to use (i.e. IterBasedRunner or␣
↪EpochBasedRunner)
    max_epochs=12) # Runner that runs the workflow in total max_epochs. For␣
↪IterBasedRunner use `max_iters`
checkpoint_config = dict(  # Config to set the checkpoint hook
    interval=12)  # The save interval is 12
log_config = dict(  # config to register logger hook
    interval=50,  # Interval to print the log
    hooks=[
        # dict(type='TensorboardLoggerHook')  # The Tensorboard logger is also supported
        dict(type='TextLoggerHook')
    ])  # The logger used to record the training process.
dist_params = dict(backend='nccl')  # Parameters to setup distributed training, the port␣
↪can also be set.
log_level = 'INFO'  # The level of logging.
load_from = None  # load models as a pre-trained model from a given path. This will not␣
↪resume training.
resume_from = None  # Resume checkpoints from a given path, the training will be resumed␣
↪from the epoch when the checkpoint's is saved.
workflow = [('train', 1)]  # Workflow for runner. [('train', 1)] means there is only one␣
↪workflow and the workflow named 'train' is executed once. The workflow trains the model␣
↪by 12 epochs according to the total_epochs.
work_dir = './work_dirs/rotated_retinanet_hbb_r50_fpn_1x_dota_oc'  # Directory to save␣
↪the model checkpoints and logs for the current experiments.
```

3.1.4 FAQ

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, the user needs to pass the intermediate variables into corresponding fields again. For example, we would like to use an offline multi-scale strategy to train an RoI-Trans. `train_pipeline` are intermediate variables we would like to modify.

```
_base_ = ['./roi-trans-le90_r50_fpn_1x_dota.py']

data_root = '../datasets/split_ms_dota1_0/'
angle_version = 'le90'
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='RResize', img_scale=(1024, 1024)),
    dict(
        type='RRandomFlip',
        flip_ratio=[0.25, 0.25, 0.25],
```

(continues on next page)

(continued from previous page)

```
        direction=['horizontal', 'vertical', 'diagonal'],
        version=angle_version),
    dict(
        type='PolyRandomRotate',
        rotate_ratio=0.5,
        angles_range=180,
        auto_bound=False,
        version=angle_version),
    dict(type='Normalize', **img_norm_cfg),
    dict(type='Pad', size_divisor=32),
    dict(type='DefaultFormatBundle'),
    dict(type='Collect', keys=['img', 'gt_bboxes', 'gt_labels'])
]
data = dict(
    train=dict(
        pipeline=train_pipeline,
        ann_file=data_root + 'trainval/annfiles/',
        img_prefix=data_root + 'trainval/images/'),
    val=dict(
        ann_file=data_root + 'trainval/annfiles/',
        img_prefix=data_root + 'trainval/images/'),
    test=dict(
        ann_file=data_root + 'test/images/',
        img_prefix=data_root + 'test/images/'))
```

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_ = './roi-trans-le90_r50_fpn_1x_dota.py'
norm_cfg = dict(type='BN', requires_grad=True)
model = dict(
    backbone=dict(norm_cfg=norm_cfg),
    neck=dict(norm_cfg=norm_cfg),
    ...)
```

3.2 Dataset Preparation (To be updated)

Please refer to data preparation for dataset preparation.

3.3 Test a model (To be updated)

- single GPU
- single node multiple GPU
- multiple node

You can use the following commands to infer a dataset.

```
# single-gpu
python tools/test.py ${CONFIG_FILE} ${CHECKPOINT_FILE} [optional arguments]

# multi-gpu
./tools/dist_test.sh ${CONFIG_FILE} ${CHECKPOINT_FILE} ${GPU_NUM} [optional arguments]

# multi-node in slurm environment
python tools/test.py ${CONFIG_FILE} ${CHECKPOINT_FILE} [optional arguments] --launcher␣
↪slurm
```

Examples:

Inference RotatedRetinaNet on DOTA-1.0 dataset, which can generate compressed files for online submission. (Please change the data_root firstly.)

```
python ./tools/test.py  \
  configs/rotated_retinanet/rotated-retinanet-rbox-le90_r50_fpn_1x_dota.py \
  checkpoints/SOME_CHECKPOINT.pth --format-only \
  --eval-options submission_dir=work_dirs/Task1_results
```

or

```
./tools/dist_test.sh  \
  configs/rotated_retinanet/rotated-retinanet-rbox-le90_r50_fpn_1x_dota.py \
  checkpoints/SOME_CHECKPOINT.pth 1 --format-only \
  --eval-options submission_dir=work_dirs/Task1_results
```

You can change the test set path in the data_root to the val set or trainval set for the offline evaluation.

```
python ./tools/test.py \
  configs/rotated_retinanet/rotated-retinanet-rbox-le90_r50_fpn_1x_dota.py \
  checkpoints/SOME_CHECKPOINT.pth --eval mAP
```

or

```
./tools/dist_test.sh  \
  configs/rotated_retinanet/rotated-retinanet-rbox-le90_r50_fpn_1x_dota.py \
  checkpoints/SOME_CHECKPOINT.pth 1 --eval mAP
```

You can also visualize the results.

```
python ./tools/test.py \
  configs/rotated_retinanet/rotated-retinanet-rbox-le90_r50_fpn_1x_dota.py \
  checkpoints/SOME_CHECKPOINT.pth \
  --show-dir work_dirs/vis
```

3.4 Train a model

3.4.1 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}`.

3.4.2 Train with multiple GPUs

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

Optional arguments are:

- `--no-validate` (**not suggested**): By default, the codebase will perform evaluation 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.

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.

3.4.3 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.

3.4.4 Manage jobs with Slurm

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

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

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.

3.4.5 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` and `config2.py`.

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

CHAPTER

FOUR

USEFUL TOOLS

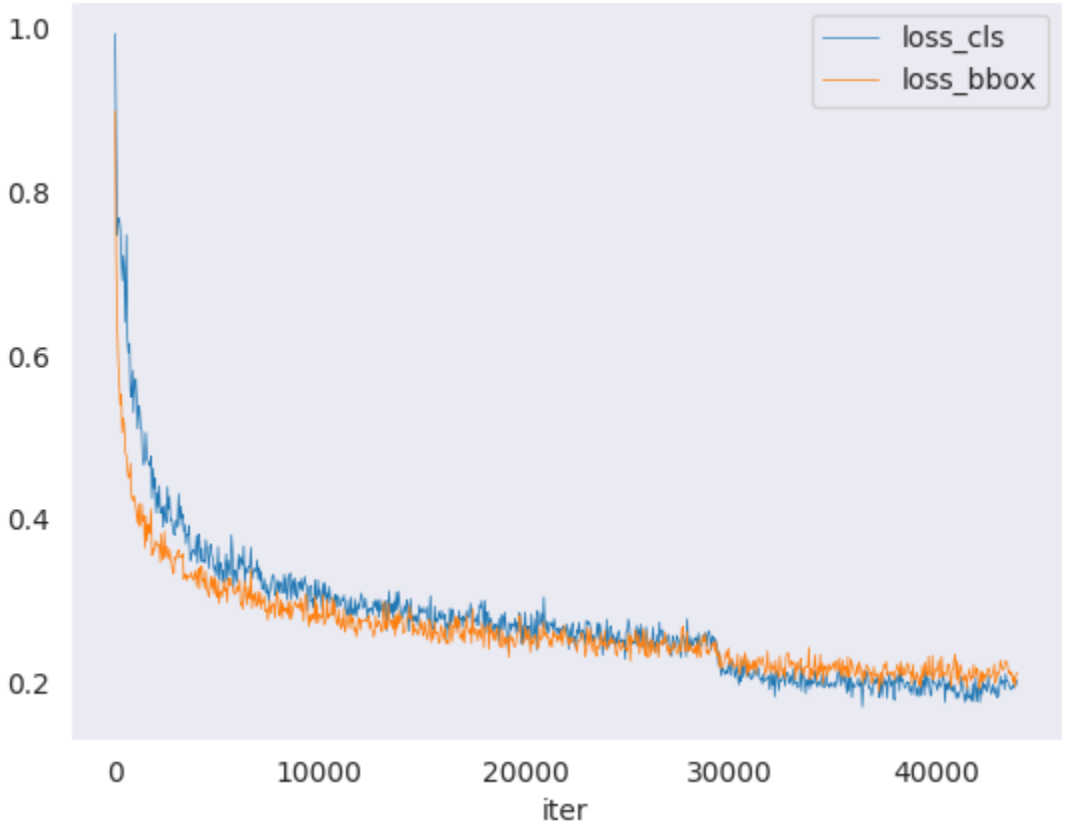
4.1 Useful Tools (To be updated)

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

4.1.1 Log Analysis

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

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

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

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

- Compute the average training speed.

```
python tools/analysis_tools/analyze_logs.py cal_train_time log.json [--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
```

4.1.2 Visualization

Visualize Datasets

`tools/misc/browse_dataset.py` helps the user to browse a detection dataset (both images and bounding box annotations) visually, or save the image to a designated directory.

```
python tools/misc/browse_dataset.py ${CONFIG} [-h] [--skip-type ${SKIP_TYPE[SKIP_TYPE...
↪]}] [--output-dir ${OUTPUT_DIR}] [--not-show] [--show-interval ${SHOW_INTERVAL}]
```

4.1.3 Model Serving

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

1. Convert model from MMRotate to TorchServe

```
python tools/deployment/mmrotate2torchserve.py ${CONFIG_FILE} ${CHECKPOINT_FILE} \
--output-folder ${MODEL_STORE} \
--model-name ${MODEL_NAME}
```

Example:

```
wget -P checkpoint  \
https://download.openmmlab.com/mmrotate/v0.1.0/rotated_faster_rcnn/rotated-faster-rcnn-
↪le90_r50_fpn_1x_dota/rotated_faster_rcnn_r50_fpn_1x_dota_le90-0393aa5c.pth

python tools/deployment/mmrotate2torchserve.py configs/rotated_faster_rcnn/rotated-
↪faster-rcnn-le90_r50_fpn_1x_dota.py checkpoint/rotated_faster_rcnn_r50_fpn_1x_dota_
↪le90-0393aa5c.pth \
--output-folder ${MODEL_STORE} \
--model-name rotated_faster_rcnn
```

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

2. Build `mmrotate-serve` docker image

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

3. Run `mmrotate-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 \
mmrotate-serve:latest
```

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

4. Test deployment

```
curl -O https://raw.githubusercontent.com/open-mmlab/mmrotate/main/demo/demo.jpg
curl http://127.0.0.1:8080/predictions/${MODEL_NAME} -T demo.jpg
```

You should obtain a response similar to:

```
[
  {
    "class_name": "small-vehicle",
```

(continues on next page)

(continued from previous page)


```
    "bbox": [
      584.9473266601562,
      327.2749938964844,
      38.45665740966797,
      16.898427963256836,
      -0.7229751944541931
    ],
    "score": 0.9766026139259338
  },
  {
    "class_name": "small-vehicle",
    "bbox": [
      152.0239715576172,
      305.92572021484375,
      43.144744873046875,
      18.85024642944336,
      0.014928221702575684
    ],
    "score": 0.972826361656189
  },
  {
    "class_name": "large-vehicle",
    "bbox": [
      160.58056640625,
      437.3690185546875,
      55.6795654296875,
      19.31710433959961,
      0.007036328315734863
    ],
    "score": 0.888836681842804
  },
  {
    "class_name": "large-vehicle",
    "bbox": [
      666.2868041992188,
      1011.3961181640625,
      60.396209716796875,
      21.821645736694336,
      0.8549195528030396
    ],
    "score": 0.8240180015563965
  }
]
```


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

```
python tools/deployment/test_torchserver.py ${IMAGE_FILE} ${CONFIG_FILE} ${CHECKPOINT_
↪FILE} ${MODEL_NAME}
[--inference-addr ${INFERENCE_ADDR}] [--device ${DEVICE}] [--score-thr ${SCORE_THR}]
```

Example:

```
python tools/deployment/test_torchserver.py \
demo/demo.jpg \
configs/rotated_faster_rcnn/rotated-faster-rcnn-le90_r50_fpn_1x_dota.py \
rotated_faster_rcnn_r50_fpn_1x_dota_le90-0393aa5c.pth \
rotated_fater_rcnn
```

4.1.4 Model Complexity

`tools/analysis_tools/get_flops.py` is a script adapted from flops-counter.pytorch to compute the FLOPs and params of a given model.

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

You will get the results like this.

```
==============================
Input shape: (3, 1024, 1024)
Flops: 215.92 GFLOPs
Params: 36.42 M
==============================
```

Note: This tool is still experimental and we do not guarantee that the number is absolutely 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, 1024, 1024).
2. Some operators are not counted into FLOPs like DCN and custom operators. So models with dcn such as S2A-Net and RepPoints based model got wrong flops. Refer to `mmcv.cnn.get_model_complexity_info()` for details.
3. The FLOPs of two-stage detectors is dependent on the number of proposals.

Prepare a model for publishing

`tools/model_converters/publish_model.py` helps users to prepare their model for publishing.

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/model_converters/publish_model.py ${INPUT_FILENAME} ${OUTPUT_FILENAME}
```

E.g.,

```
python tools/model_converters/publish_model.py work_dirs/rotated_faster_rcnn/latest.pth␣
↪rotated_faster_rcnn_r50_fpn_1x_dota_le90_20190801.pth
```

The final output filename will be `rotated_faster_rcnn_r50_fpn_1x_dota_le90_20190801-{hash id}.pth`.

4.1.5 Benchmark

FPS Benchmark

`tools/analysis_tools/benchmark.py` helps users to calculate FPS. The FPS value includes model forward and post-processing. In order to get a more accurate value, currently only supports single GPU distributed startup mode.

```
python -m torch.distributed.launch --nproc_per_node=1 --master_port=${PORT} tools/
↪analysis_tools/benchmark.py \
    ${CONFIG} \
    ${CHECKPOINT} \
    [--repeat-num ${REPEAT_NUM}] \
    [--max-iter ${MAX_ITER}] \
    [--log-interval ${LOG_INTERVAL}] \
    --launcher pytorch
```

Examples: Assuming that you have already downloaded the `Rotated Faster R-CNN` model checkpoint to the directory `checkpoints/`.

```
python -m torch.distributed.launch --nproc_per_node=1 --master_port=29500 tools/analysis_
↪tools/benchmark.py \
       configs/rotated_faster_rcnn/rotated-faster-rcnn-le90_r50_fpn_1x_dota.py \
       checkpoints/rotated_faster_rcnn_r50_fpn_1x_dota_le90-0393aa5c.pth \
       --launcher pytorch
```

4.1.6 Miscellaneous

Print the entire config

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

```
python tools/misc/print_config.py ${CONFIG} [-h] [--options ${OPTIONS [OPTIONS...]}]
```

4.1.7 Confusion Matrix

A confusion matrix is a summary of prediction results.

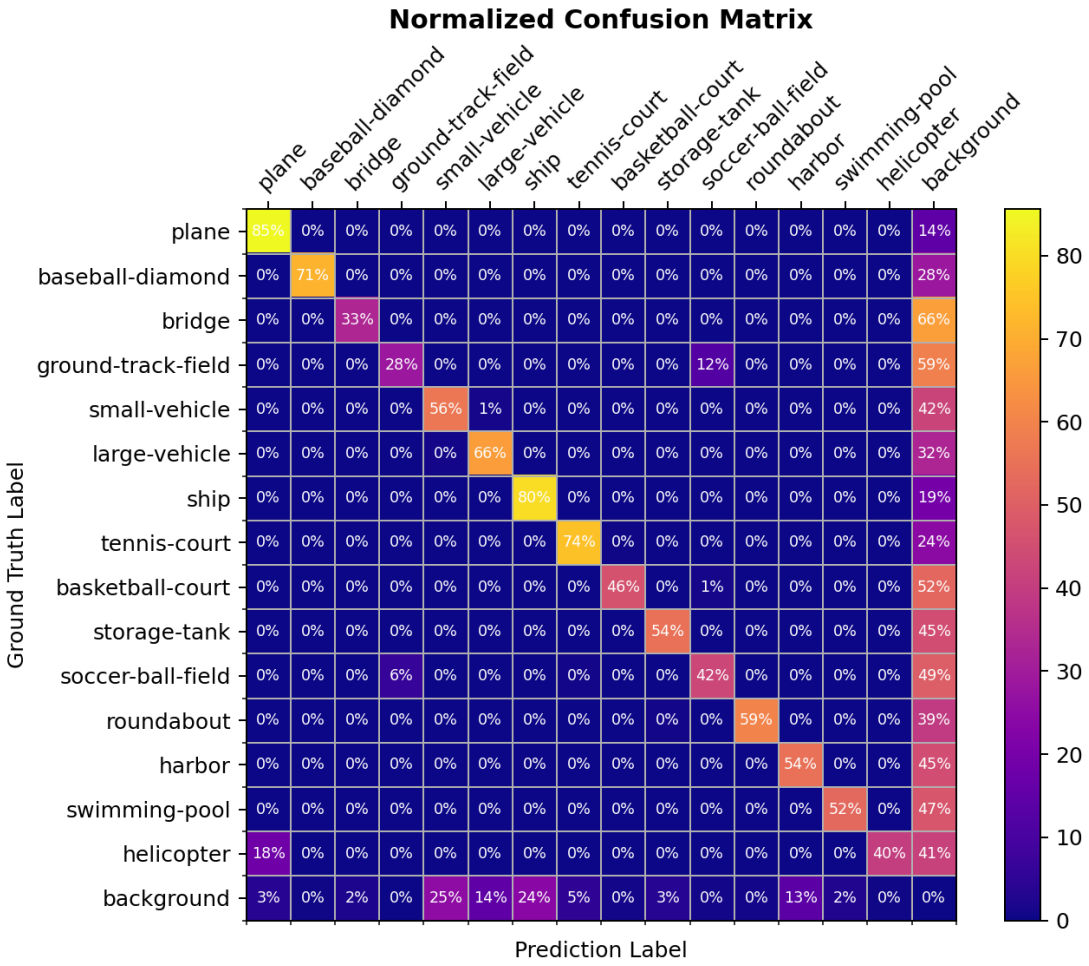
`tools/analysis_tools/confusion_matrix.py` can analyze the prediction results and plot a confusion matrix table.

First, run `tools/test.py` to save the `.pkl` detection results.

Then, run

```
python tools/analysis_tools/confusion_matrix.py ${CONFIG}  ${DETECTION_RESULTS}  ${SAVE_
↪DIR} --show
```

And you will get a confusion matrix like this:



4.2 Model Deployment (To be updated)

MMRotate 1.x fully relies on MMDeploy to deploy models. Please stay tuned and this document will be update soon.

CHAPTER

FIVE

BASIC CONCEPTS

CHAPTER

SIX

COMPONENT CUSTOMIZATION

6.1 Customize Models (To be updated)

We basically categorize model components into 5 types.

- backbone: usually an FCN network to extract feature maps, e.g., ResNet, Swin.
- neck: the component between backbones and heads, e.g., FPN, ReFPN.
- head: the component for specific tasks, e.g., bbox prediction.
- roi extractor: the part for extracting RoI features from feature maps, e.g., RoI Align Rotated.
- loss: the component in head for calculating losses, e.g., FocalLoss, GWDLoss, and KFIoULoss.

6.1.1 Develop new components

Add a new backbone

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

1. Define a new backbone (e.g. MobileNet)

Create a new file `mmrotate/models/backbones/mobilenet.py`.

```python
import torch.nn as nn

from mmrotate.models.builder import ROTATED_BACKBONES


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

    def __init__(self, arg1, arg2):
        pass

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

2. Import the module

You can either add the following line to `mmrotate/models/backbones/__init__.py`

```
from .mobilenet import MobileNet
```

or alternatively add

```
custom_imports = dict(
    imports=['mmrotate.models.backbones.mobilenet'],
    allow_failed_imports=False)
```

to the config file to avoid modifying the original code.

3. Use the backbone in your config file

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

Add new necks

1. Define a neck (e.g. PAFPN)

Create a new file `mmrotate/models/necks/pafpn.py`.

```
from mmrotate.models.builder import ROTATED_NECKS

@ROTATED_NECKS.register_module()
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

You can either add the following line to `mmrotate/models/necks/__init__.py`,

```
from .pafpn import PAFPN
```

or alternatively add

```
custom_imports = dict(
    imports=['mmrotate.models.necks.pafpn.py'],
    allow_failed_imports=False)
```

to the config file and avoid modifying the original code.

3. Modify the config file

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

Add new heads

Here we show how to develop a new head with the example of Double Head R-CNN as the following.

First, add a new bbox head in `mmrotate/models/roi_heads/bbox_heads/double_bbox_head.py`. Double Head R-CNN implements a new bbox head for object detection. To implement a bbox head, basically we need to implement three functions of the new module as the following.

```
from mmrotate.models.builder import ROTATED_HEADS
from mmrotate.models.roi_heads.bbox_heads.bbox_head import BBoxHead

@ROTATED_HEADS.register_module()
class DoubleConvFCBBoxHead(BBoxHead):
    r"""Bbox head used in Double-Head R-CNN

                                    /-> cls
                 /-> shared convs ->
                                    \-> reg
    roi features
                                    /-> cls
                 \-> shared fc    ->
                                    \-> reg
    """  # noqa: W605

    def __init__(self,
                 num_convs=0,
                 num_fcs=0,
                 conv_out_channels=1024,
                 fc_out_channels=1024,
```

(continues on next page)

(continued from previous page)

```
                conv_cfg=None,
                norm_cfg=dict(type='BN'),
                **kwargs):
        kwargs.setdefault('with_avg_pool', True)
        super(DoubleConvFCBBoxHead, self).__init__(**kwargs)


    def forward(self, x_cls, x_reg):
```

Second, implement a new RoI Head if it is necessary. We plan to inherit the new `DoubleHeadRoIHead` from `StandardRoIHead`. We can find that a `StandardRoIHead` already implements the following functions.

```
import torch

from mmdet.core import bbox2result, bbox2roi, build_assigner, build_sampler
from mmrotate.models.builder import ROTATED_HEADS, build_head, build_roi_extractor
from mmrotate.models.roi_heads.base_roi_head import BaseRoIHead
from mmrotate.models.roi_heads.test_mixins import BBoxTestMixin, MaskTestMixin


@ROTATED_HEADS.register_module()
class StandardRoIHead(BaseRoIHead, BBoxTestMixin, MaskTestMixin):
    """Simplest base roi head including one bbox head and one mask head.
    """

    def init_assigner_sampler(self):

    def init_bbox_head(self, bbox_roi_extractor, bbox_head):

    def forward_dummy(self, x, proposals):


    def forward_train(self,
                      x,
                      img_metas,
                      proposal_list,
                      gt_bboxes,
                      gt_labels,
                      gt_bboxes_ignore=None,
                      gt_masks=None):

    def _bbox_forward(self, x, rois):

    def _bbox_forward_train(self, x, sampling_results, gt_bboxes, gt_labels,
                            img_metas):

    def simple_test(self,
                    x,
                    proposal_list,
                    img_metas,
                    proposals=None,
```

(continues on next page)

(continued from previous page)

```
                    rescale=False):
        """Test without augmentation."""
```

Double Head's modification is mainly in the bbox_forward logic, and it inherits other logics from the StandardRoIHead. In the mmrotate/models/roi_heads/double_roi_head.py, we implement the new RoI Head as the following:

```
from mmrotate.models.builder import ROTATED_HEADS
from mmrotate.models.roi_heads.standard_roi_head import StandardRoIHead


@ROTATED_HEADS.register_module()
class DoubleHeadRoIHead(StandardRoIHead):
    """RoI head for Double Head RCNN

    https://arxiv.org/abs/1904.06493
    """

    def __init__(self, reg_roi_scale_factor, **kwargs):
        super(DoubleHeadRoIHead, self).__init__(**kwargs)
        self.reg_roi_scale_factor = reg_roi_scale_factor

    def _bbox_forward(self, x, rois):
        bbox_cls_feats = self.bbox_roi_extractor(
            x[:self.bbox_roi_extractor.num_inputs], rois)
        bbox_reg_feats = self.bbox_roi_extractor(
            x[:self.bbox_roi_extractor.num_inputs],
            rois,
            roi_scale_factor=self.reg_roi_scale_factor)
        if self.with_shared_head:
            bbox_cls_feats = self.shared_head(bbox_cls_feats)
            bbox_reg_feats = self.shared_head(bbox_reg_feats)
        cls_score, bbox_pred = self.bbox_head(bbox_cls_feats, bbox_reg_feats)

        bbox_results = dict(
            cls_score=cls_score,
            bbox_pred=bbox_pred,
            bbox_feats=bbox_cls_feats)
        return bbox_results
```

Last, the users need to add the module in mmrotate/models/bbox_heads/__init__.py and mmrotate/models/roi_heads/__init__.py thus the corresponding registry could find and load them.

Alternatively, the users can add

```
custom_imports=dict(
    imports=['mmrotate.models.roi_heads.double_roi_head', 'mmrotate.models.bbox_heads.
↪double_bbox_head'])
```

to the config file and achieve the same goal.

Add new loss

Assume you want to add a new loss as `MyLoss`, for bounding box regression. To add a new loss function, the users need implement it in `mmrotate/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 mmrotate.models.builder import ROTATED_LOSSES
from mmdet.models.losses.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

@ROTATED_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_bbox = self.loss_weight * my_loss(
            pred, target, weight, reduction=reduction, avg_factor=avg_factor)
        return loss_bbox
```

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

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

Alternatively, you can add

```
custom_imports=dict(
    imports=['mmrotate.models.losses.my_loss'])
```

to the config file and achieve the same goal.

To use it, modify the `loss_xxx` field. Since MyLoss is for regression, you need to modify the `loss_bbox` field in the head.

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

6.2 Customize Datasets (To be updated)

6.2.1 Support new data format

To support a new data format, you can convert them to existing formats (DOTA format). You could choose to convert them offline (before training by a script) or online (implement a new dataset and do the conversion at training). In MMRotate, we recommend to convert the data into DOTA formats and do the conversion offline, thus you only need to modify the config's data annotation paths and classes after the conversion of your data.

Reorganize new data formats to existing format

The simplest way is to convert your dataset to existing dataset formats (DOTA).

The annotation txt files in DOTA format:

```
184 2875 193 2923 146 2932 137 2885 plane 0
66 2095 75 2142 21 2154 11 2107 plane 0
...
```

Each line represents an object and records it as a 10-dimensional array `A`.

- `A[0:8]`: Polygons with format `(x1, y1, x2, y2, x3, y3, x4, y4)`.
- `A[8]`: Category.
- `A[9]`: Difficulty.

After the data pre-processing, there are two steps for users to train the customized new dataset with existing format (e.g. DOTA format):

1. Modify the config file for using the customized dataset.
2. Check the annotations of the customized dataset.

Here we give an example to show the above two steps, which uses a customized dataset of 5 classes with COCO format to train an existing Cascade Mask R-CNN R50-FPN detector.

1. Modify the config file for using the customized dataset

There are two aspects involved in the modification of config file:

1. The `data` field. Specifically, you need to explicitly add the `classes` fields in `data.train`, `data.val` and `data.test`.
2. The `num_classes` field in the `model` part. Explicitly over-write all the `num_classes` from default value (e.g. 80 in COCO) to your classes number.

In `configs/my_custom_config.py`:

```
# the new config inherits the base configs to highlight the necessary modification
_base_ = './rotated_retinanet_hbb_r50_fpn_1x_dota_oc'
```

(continues on next page)

(continued from previous page)

```
# 1. dataset settings
dataset_type = 'DOTADataset'
classes = ('a', 'b', 'c', 'd', 'e')
data = dict(
    samples_per_gpu=2,
    workers_per_gpu=2,
    train=dict(
        type=dataset_type,
        # explicitly add your class names to the field `classes`
        classes=classes,
        ann_file='path/to/your/train/annotation_data',
        img_prefix='path/to/your/train/image_data'),
    val=dict(
        type=dataset_type,
        # explicitly add your class names to the field `classes`
        classes=classes,
        ann_file='path/to/your/val/annotation_data',
        img_prefix='path/to/your/val/image_data'),
    test=dict(
        type=dataset_type,
        # explicitly add your class names to the field `classes`
        classes=classes,
        ann_file='path/to/your/test/annotation_data',
        img_prefix='path/to/your/test/image_data'))

# 2. model settings
model = dict(
    bbox_head=dict(
        type='RotatedRetinaHead',
        # explicitly over-write all the `num_classes` field from default 15 to 5.
        num_classes=15))
```

2. Check the annotations of the customized dataset

Assuming your customized dataset is DOTA format, make sure you have the correct annotations in the customized dataset:

- The `classes` fields in your config file should have exactly the same elements and the same order with the `A[8]` in txt annotations. MMRotate automatically maps the uncontinuous `id` in `categories` to the continuous label indices, so the string order of `name` in `categories` field affects the order of label indices. Meanwhile, the string order of `classes` in config affects the label text during visualization of predicted bounding boxes.

6.2.2 Customize datasets by dataset wrappers

MMRotate also supports many dataset wrappers to mix the dataset or modify the dataset distribution for training. Currently it supports to three dataset wrappers as below:

- `RepeatDataset`: simply repeat the whole dataset.
- `ClassBalancedDataset`: repeat dataset in a class balanced manner.
- `ConcatDataset`: concat datasets.

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
        )
    )
```

Class balanced dataset

We use `ClassBalancedDataset` as wrapper to repeat the dataset based on category frequency. The dataset to repeat needs to instantiate function `self.get_cat_ids(idx)` to support `ClassBalancedDataset`. For example, to repeat `Dataset_A` with `oversample_thr=1e-3`, the config looks like the following

```
dataset_A_train = dict(
        type='ClassBalancedDataset',
        oversample_thr=1e-3,
        dataset=dict(  # This is the original config of Dataset_A
            type='Dataset_A',
            ...
            pipeline=train_pipeline
        )
    )
```

Concatenate dataset

There are three 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.

```
dataset_A_train = dict(
    type='Dataset_A',
    ann_file = ['anno_file_1', 'anno_file_2'],
```

(continues on next page)

(continued from previous page)

```
    pipeline=train_pipeline
)
```

If the concatenated dataset is used for test or evaluation, this manner supports to evaluate each dataset separately. To test the concatenated datasets as a whole, you can set `separate_eval=False` as below.

```
dataset_A_train = dict(
    type='Dataset_A',
    ann_file = ['anno_file_1', 'anno_file_2'],
    separate_eval=False,
    pipeline=train_pipeline
)
```

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
    )
```

If the concatenated dataset is used for test or evaluation, this manner also supports to evaluate each dataset separately.

3. We also support to define `ConcatDataset` explicitly as the following.

```
dataset_A_val = dict()
dataset_B_val = dict()

data = dict(
    imgs_per_gpu=2,
    workers_per_gpu=2,
    train=dataset_A_train,
    val=dict(
        type='ConcatDataset',
        datasets=[dataset_A_val, dataset_B_val],
        separate_eval=False))
```

This manner allows users to evaluate all the datasets as a single one by setting `separate_eval=False`.

Note:

1. The option `separate_eval=False` assumes the datasets use `self.data_infos` during evaluation. Therefore, COCO datasets do not support this behavior since COCO datasets do not fully rely on `self.data_infos` for evaluation. Combining different types of datasets and evaluating them as a whole is not tested thus is not suggested.

2. Evaluating `ClassBalancedDataset` and `RepeatDataset` is not supported thus evaluating concatenated datasets of these types is also not supported.

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
)
```

6.3 Customize Runtime Settings (To be updated)

6.3.1 Customize optimization settings

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.

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 `mmrotate/core/optimizer`. And then implement the new optimizer in a file, e.g., in `mmrotate/core/optimizer/my_optimizer.py`:

```
from mmdet.core.optimizer.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 `mmrotate/core/optimizer/__init__.py` to import it.

  The newly defined module should be imported in `mmrotate/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=['mmrotate.core.optimizer.my_optimizer'], allow_failed_
↪imports=False)
```

The module `mmrotate.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. `mmrotate.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)
```

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 mmrotate.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):

        return my_optimizer
```

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

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,
)
```

6.3.2 Customize training schedules

By default we use step learning rate with 1x schedule, this calls `StepLRHook` in MMCV. We support many other learning rate schedule here, such as `CosineAnnealing` and `Poly` schedule. Here are some examples

- Poly schedule:

```
lr_config = dict(policy='poly', power=0.9, min_lr=1e-4, by_epoch=False)
```

- ConsineAnnealing schedule:

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

6.3.3 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.

6.3.4 Customize hooks

Customize self-implemented hooks

1. Implement a new hook

There are some occasions when the users might need to implement a new hook. MMRotate supports customized hooks in training. Thus the users could implement a hook directly in mmrotate or their mmdet-based codebases and use the hook by only modifying the config in training. Here we give an example of creating a new hook in mmrotate and using it in training.

```
from mmcv.runner import HOOKS, Hook


@HOOKS.register_module()
class MyHook(Hook):

    def __init__(self, a, b):
        pass

    def before_run(self, runner):
        pass

    def after_run(self, runner):
        pass

    def before_epoch(self, runner):
        pass
```

(continues on next page)

(continued from previous page)

```
    def after_epoch(self, runner):
        pass

    def before_iter(self, runner):
        pass

    def after_iter(self, runner):
        pass
```

Depending on the functionality of the hook, the users need to specify what the hook will do at each stage of the training in `before_run`, `after_run`, `before_epoch`, `after_epoch`, `before_iter`, and `after_iter`.

2. Register the new hook

Then we need to make `MyHook` imported. Assuming the file is in `mmrotate/core/utils/my_hook.py` there are two ways to do that:

- Modify `mmrotate/core/utils/__init__.py` to import it.

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

```
from .my_hook import MyHook
```

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

```
custom_imports = dict(imports=['mmrotate.core.utils.my_hook'], allow_failed_
↪imports=False)
```

3. Modify the config

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

You can also set the priority of the hook by adding key `priority` to `'NORMAL'` or `'HIGHEST'` as below

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

By default the hook's priority is set as `NORMAL` during registration.

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

4. Example: `NumClassCheckHook`

We implement a customized hook named NumClassCheckHook to check whether the `num_classes` in head matches the length of `CLASSSES` in `dataset`.

We set it in default_runtime.py.

```
custom_hooks = [dict(type='NumClassCheckHook')]
```

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='bbox')
```

CHAPTER

SEVEN

HOW TO

CHAPTER

EIGHT

MIGRATION

CHAPTER

NINE

MMROTATE.APIS

CHAPTER

TEN

MMROTATE.CORE

10.1 anchor

10.2 bbox

10.3 patch

10.4 evaluation

10.5 post_processing

10.6 visualization

CHAPTER

ELEVEN

MMROTATE.DATASETS

11.1 datasets

11.2 pipelines

CHAPTER

TWELVE

MMROTATE.MODELS

CHAPTER

THIRTEEN

MMROTATE.UTILS

CHAPTER

FOURTEEN

BENCHMARK AND MODEL ZOO (TO BE UPDATED)

- Rotated RetinaNet-OBB/HBB (ICCV'2017)
- Rotated FasterRCNN-OBB (TPAMI'2017)
- Rotated RepPoints-OBB (ICCV'2019)
- Rotated FCOS (ICCV'2019)
- RoI Transformer (CVPR'2019)
- Gliding Vertex (TPAMI'2020)
- Rotated ATSS-OBB (CVPR'2020)
- CSL (ECCV'2020)
- R3Det (AAAI'2021)
- S2A-Net (TGRS'2021)
- ReDet (CVPR'2021)
- Beyond Bounding-Box (CVPR'2021)
- Oriented R-CNN (ICCV'2021)
- GWD (ICML'2021)
- KLD (NeurIPS'2021)
- SASM (AAAI'2022)
- Oriented RepPoints (CVPR'2022)
- KFIoU (arXiv)

14.1 Results on DOTA v1.0

- `MS` means multiple scale image split.
- `RR` means random rotation.

The above models are trained with 1 * 1080Ti/2080Ti and inferred with 1 * 2080Ti.

CHAPTER

FIFTEEN

CONTRIBUTING TO MMROTATE (TO BE UPDATED)

This section introduces following contents:

All kinds of contributions are welcome, including but not limited to the following.

- Fix typo or bugs
- Add documentation or translate the documentation into other languages
- Add new features and components

15.1 Workflow

1. fork and pull the latest MMRotate repository (MMRotate)
2. checkout a new branch (do not use master branch for PRs)
3. commit your changes
4. create a PR

Note: If you plan to add some new features that involve large changes, it is encouraged to open an issue for discussion first.

15.2 Code style

15.2.1 Python

We adopt PEP8 as the preferred code style.

We use the following tools for linting and formatting:

- flake8: A wrapper around some linter tools.
- isort: A Python utility to sort imports.

- yapf: A formatter for Python files.
- codespell: A Python utility to fix common misspellings in text files.
- mdformat: Mdformat is an opinionated Markdown formatter that can be used to enforce a consistent style in Markdown files.
- docformatter: A formatter to format docstring.

Style configurations can be found in setup.cfg.

We use pre-commit hook that checks and formats for `flake8`, `yapf`, `isort`, `trailing whitespaces`, `markdown files`, fixes `end-of-files`, `double-quoted-strings`, `python-encoding-pragma`, `mixed-line-ending`, sorts `requirments.txt` automatically on every commit. The config for a pre-commit hook is stored in .pre-commit-config.

After you clone the repository, you will need to install initialize pre-commit hook.

```
pip install -U pre-commit
```

From the repository folder

```
pre-commit install
```

After this on every commit check code linters and formatter will be enforced.

Important: Before you create a PR, make sure that your code lints and is formatted by yapf.

15.2.2 C++ and CUDA

We follow the Google C++ Style Guide.

CHAPTER

SIXTEEN

CHANGELOG OF V1.X

16.1 v1.0.0rc0 (7/11/2022)

We are excited to announce the release of MMRotate 1.0.0rc0. MMRotate 1.0.0rc0 is the first version of MMRotate 1.x, a part of the OpenMMLab 2.0 projects. Built upon the new training engine, MMRotate 1.x unifies the interfaces of dataset, models, evaluation, and visualization with faster training and testing speed.

16.1.1 Highlights

1. **New engines**. MMRotate 1.x is based on MMEngine, which provides a general and powerful runner that allows more flexible customizations and significantly simplifies the entrypoints of high-level interfaces.
2. **Unified interfaces**. As a part of the OpenMMLab 2.0 projects, MMRotate 1.x unifies and refactors the interfaces and internal logics of train, testing, datasets, models, evaluation, and visualization. All the OpenMMLab 2.0 projects share the same design in those interfaces and logics to allow the emergence of multi-task/modality algorithms.
3. **New BoxType design**. We support data structures RotatedBoxes and QuadriBoxes to encapsulate different kinds of bounding boxes. We are migrating to use data structures of boxes to replace the use of pure tensor boxes. This will unify the usages of different kinds of bounding boxes in MMDetection 3.x and MMRotate 1.x to simplify the implementation and reduce redundant codes.
4. **Stronger visualization**. We provide a series of useful tools which are mostly based on brand-new visualizers. As a result, it is more convenient for the users to explore the models and datasets now.

16.1.2 Breaking Changes

We briefly list the major breaking changes here. We will update the *migration guide* to provide complete details and migration instructions.

Dependencies

- MMRotate 1.x relies on MMEngine to run. MMEngine is a new foundational library for training deep learning models in OpenMMLab 2.0 models. The dependencies of file IO and training are migrated from MMCV 1.x to MMEngine.
- MMRotate 1.x relies on MMCV>=2.0.0rc2. Although MMCV no longer maintains the training functionalities since 2.0.0rc0, MMRotate 1.x relies on the data transforms, CUDA operators, and image processing interfaces in MMCV. Note that the package `mmcv` is the version that provide pre-built CUDA operators and `mmcv-lite` does not since MMCV 2.0.0rc0, while `mmcv-full` has been deprecated.

- MMRotate 1.x relies on MMDetection>=3.0.0rc2.

Training and testing

- MMRotate 1.x uses Runner in MMEngine rather than that in MMCV. The new Runner implements and unifies the building logic of dataset, model, evaluation, and visualizer. Therefore, MMRotate 1.x no longer maintains the building logics of those modules in `mmrotate.train.apis` and `tools/train.py`. Those code have been migrated into MMEngine. Please refer to the migration guide of Runner in MMEngine for more details.
- The Runner in MMEngine also supports testing and validation. The testing scripts are also simplified, which has similar logic as that in training scripts to build the runner.
- The execution points of hooks in the new Runner have been enriched to allow more flexible customization. Please refer to the migration guide of Hook in MMEngine for more details.
- Learning rate and momentum scheduling has been migrated from `Hook` to `Parameter Scheduler` in MMEngine. Please refer to the migration guide of Parameter Scheduler in MMEngine for more details.

Configs

- The Runner in MMEngine uses a different config structures to ease the understanding of the components in runner. Users can refer to the migration guide in MMEngine for migration details.
- The file names of configs and models are also refactored to follow the new rules unified across OpenMMLab 2.0 projects.

Dataset

The Dataset classes implemented in MMRotate 1.x all inherits from the BaseDataset in MMEngine.

- All the datasets support to serialize the data list to reduce the memory when multiple workers are built to accelerate data loading.

Data Transforms

The data transforms in MMRotate 1.x all inherits from those in MMCV>=2.0.0rc2, which follows a new convention in OpenMMLab 2.0 projects. The changes are listed as below:

- The interfaces are also changed. Please refer to the API Reference
- The functionality of some data transforms (e.g., `Rotate`) are decomposed into several transforms.

Model

The models in MMRotate 1.x all inherits from `BaseModel` in MMEngine, which defines a new convention of models in OpenMMLab 2.0 projects. Users can refer to the tutorial of model in MMengine for more details. Accordingly, there are several changes as the following:

- The model interfaces, including the input and output formats, are significantly simplified and unified following the new convention in MMRotate 1.x. Specifically, all the input data in training and testing are packed into `inputs` and `data_samples`, where `inputs` contains model inputs like a list of image tensors, and `data_samples` contains other information of the current data sample such as ground truths and model predictions. In this way, different tasks in MMRotate 1.x can share the same input arguments, which makes the models more general and suitable for multi-task learning.

- The model has a data preprocessor module, which is used to pre-process the input data of model. In MMRotate 1.x, the data preprocessor usually does necessary steps to form the input images into a batch, such as padding. It can also serve as a place for some special data augmentations or more efficient data transformations like normalization.
- The internal logic of model have been changed. In MMRotate 0.x, model used `forward_train` and `simple_test` to deal with different model forward logics. In MMRotate 1.x and OpenMMLab 2.0, the forward function has three modes: `loss`, `predict`, and `tensor` for training, inference, and tracing or other purposes, respectively. The forward function calls `self.loss()`, `self.predict()`, and `self._forward()` given the modes `loss`, `predict`, and `tensor`, respectively.

Evaluation

MMRotate 1.x mainly implements corresponding metrics for each task, which are manipulated by Evaluator to complete the evaluation. In addition, users can build evaluator in MMRotate 1.x to conduct offline evaluation, i.e., evaluate predictions that may not produced by MMRotate, prediction follows our dataset conventions. More details can be find in the Evaluation Tutorial in MMEngine.

Visualization

The functions of visualization in MMRotate 1.x are removed. Instead, in OpenMMLab 2.0 projects, we use Visualizer to visualize data. MMRotate 1.x implements `RotLocalVisualizer` to allow visualization of ground truths, model predictions, and feature maps, etc., at any place. It also supports to dump the visualization data to any external visualization backends such as Tensorboard and Wandb.

16.1.3 Improvements

- Support quadrilateral box detection (#520)
- Support RotatedCocoMetric (#557)
- Support COCO style annotations (#582)
- Support two new SAR datasets: RSDD and SRSDD (#591)

16.1.4 Ongoing changes

1. Test-time augmentation: is not implemented yet in this version due to limited time slot. We will support it in the following releases with a new and simplified design.
2. Inference interfaces: a unified inference interfaces will be supported in the future to ease the use of released models.
3. Interfaces of useful tools that can be used in notebook: more useful tools that implemented in the `tools/` directory will have their python interfaces so that they can be used through notebook and in downstream libraries.
4. Documentation: we will add more design docs, tutorials, and migration guidance so that the community can deep dive into our new design, participate the future development, and smoothly migrate downstream libraries to MMRotate 1.x.

16.1.5 Contributors

A total of 8 developers contributed to this release. Thanks @DonggeunYu @k-papadakis @liuyanyi @yangxue0827 @jbwang1997 @zytx121 @RangiLyu @ZwwWayne

CHAPTER

SEVENTEEN

CHANGELOG V0.X

17.1 v0.3.2 (6/7/2022)

17.1.1 Highlight

- Support Oriented Reppoints (CVPR'22) (#286)
- Support ConvNeXt backbone (CVPR'22) (#343)

17.1.2 New Features

- Support RMosaic. (#344)

17.1.3 Bug Fixes

- Fix max_coordinate in multiclass_nms_rotated. (#346)
- Fix bug in PolyRandomRotate. (#366)
- Fix memory shortage when using huge_image_demo.py. (#368)

17.1.4 Improvements

- Update README.md and INSTALL.md. (#342)
- Fix typo in rotated_fcos_head. (#354)
- Update checkpoint and eval interval of base config. (#347)
- Fix mdformat version to support python3.6 & Add mim to extras_require in setup.py. (#359)
- Add mim test in CI. (#374)

17.1.5 Contributors

A total of 9 developers contributed to this release. Thanks @LiWentomng @heiyuxiaokai @JinYuannn @sltlls @liuyanyi @yangxue0827 @jbwang1997 @zytx121 @ZwwWayne

17.2 v0.3.1 (6/6/2022)

17.2.1 Highlight

- Support Rotated FCOS (#223)

17.2.2 New Features

- Update PolyRandomRotate to support discrete angle value. (#281)
- Support RRandomCrop. (#322)
- Support mask in merge_results and huge_image_demo.py. (#280)
- Support don't filter images without ground truths. (#323)
- Add MultiImageMixDataset in build_dataset. (#331)

17.2.3 Bug Fixes

- Fix error in Windows CI. (#324)
- Fix data path error in config files. (#328)
- Fix bug when visualize the HRSC2016 detect results. (#329)

17.2.4 Improvements

- Add torchserve doc in zh_cn. (#287)
- Fix doc typo in README. (#284)
- Configure Myst-parser to parse anchor tag (#305 #308)
- Replace markdownlint with mdformat for avoiding installing ruby. (#306)
- Fix typo about split gap of multi scale. (#272)

17.2.5 Contributors

A total of 7 developers contributed to this release. Thanks @liuyanyi @nijkah @remi-or @yangxue0827 @jbwang1997 @zytx121 @ZwwWayne

17.3 v0.3.0 (29/4/2022)

17.3.1 Highlight

- Support TorchServe (#160)
- Support Rotated ATSS (CVPR'20) (#179)

17.3.2 New Features

- Update performance of ReDet on HRSC2016. (#203)
- Upgrage visualization to custom colors of different classes. This requires mmdet>=2.22.0. (#187, #267, #270)
- Update Stable KLD, which solve the Nan issue of KLD training. (#183)
- Support setting dataloader arguments in config and add functions to handle config compatibility. (#215) The comparison between the old and new usages is as below.

```
data = dict(
    samples_per_gpu=2, workers_per_gpu=2,
    train=dict(type='xxx', ...),
    val=dict(type='xxx', samples_per_gpu=4, ...),
    test=dict(type='xxx', ...),
)
```

```
# A recommended config that is clear
data = dict(
    train=dict(type='xxx', ...),
    val=dict(type='xxx', ...),
    test=dict(type='xxx', ...),
    # Use different batch size during inference.
    train_dataloader=dict(samples_per_gpu=2, workers_per_gpu=2),
    val_dataloader=dict(samples_per_gpu=4, workers_per_gpu=4),
    test_dataloader=dict(samples_per_gpu=4, workers_per_gpu=4),
)

# Old style still works but allows to set more arguments about data loaders
data = dict(
    samples_per_gpu=2,  # only works for train_dataloader
    workers_per_gpu=2,  # only works for train_dataloader
    train=dict(type='xxx', ...),
    val=dict(type='xxx', ...),
    test=dict(type='xxx', ...),
    # Use different batch size during inference.
    val_dataloader=dict(samples_per_gpu=4, workers_per_gpu=4),
    test_dataloader=dict(samples_per_gpu=4, workers_per_gpu=4),
)
```

- Add get_flops tool (#176)

17.3.3 Bug Fixes

- Fix bug about rotated anchor inside flags. (#197)
- Fix Nan issue of GWD. (#206)
- Fix bug in eval_rbbox_map when labels_ignore is None. (#209)
- Fix bug of 'RoIAlignRotated' object has no attribute 'output_size' (#213)
- Fix bug in unit test for datasets. (#222)
- Fix bug in rotated_reppoints_head. (#246)
- Fix GPG key error in CI and docker. (#269)

17.3.4 Improvements

- Update citation of mmrotate in README.md (#263)
- Update the introduction of SASM (AAAI'22) (#184)
- Fix doc typo in Config File and Model Zoo. (#199)
- Unified RBox definition in doc. (#234)

17.3.5 Contributors

A total of 7 developers contributed to this release. Thanks @nijkah @GamblerZSY @liuyanyi @yangxue0827 @jbwang1997 @zytx121 @ZwwWayne

17.4 v0.2.0 (30/3/2022)

17.4.1 New Features

- Support Circular Smooth Label (CSL, ECCV'20) (#153)
- Support multiple machines dist_train (#143)
- Add browse_dataset tool (#98)
- Add gather_models script (#162)

17.4.2 Bug Fixes

- Remove in-place operations in rbbox_overlaps (#155)
- Fix bug in docstring. (#137)
- Fix bug in HRSCDataset with `clasesswise=ture` (#175)

17.4.3 Improvements

- Add Chinese translation of `docs/zh_cn/tutorials/customize_dataset.md` (#65)
- Add different seeds to different ranks (#102)
- Update from-scratch install script in install.md (#166)
- Improve the arguments of all mmrotate scripts (#168)

17.4.4 Contributors

A total of 6 developers contributed to this release. Thanks @zytx121 @yangxue0827 @ZwwWayne @jbwang1997 @canoe-Z @matrixgame2018

17.5 v0.1.1 (14/3/2022)

17.5.1 New Features

- Support huge image inference (#34)
- Support HRSC Dataset (#96)
- Support mixed precision training (#72)
- Add colab tutorial for beginners (#66)
- Add inference speed statistics tool (#86)
- Add confusion matrix analysis tool (#93)

17.5.2 Bug Fixes

- Fix URL error of Swin pretrained model (#111)
- Fix bug for SASM during training (#105)
- Fix rbbox_overlaps abnormal when the box is too small (#61)
- Fix bug for visualization (#12, #81)
- Fix stuck when compute mAP (#14, #52)
- Fix 'RoIAlignRotated' object has no attribute 'out_size' bug (#51)
- Add missing init_cfg in dense head (#37)
- Fix install an additional mmcv (#17)
- Fix typos in docs (#3, #11, #36)

17.5.3 Improvements

- Move `eval_rbbox_map` from `mmrotate.datasets` to `mmrotate.core.evaluation` (#73)
- Add Windows CI (#31)
- Add copyright commit hook (#30)
- Add Chinese translation of `docs/zh_cn/get_started.md` (#16)
- Add Chinese translation of `docs/zh_cn/tutorials/customize_runtime.md` (#22)
- Add Chinese translation of `docs/zh_cn/tutorials/customize_config.md` (#23)
- Add Chinese translation of `docs/zh_cn/tutorials/customize_models.md` (#27)
- Add Chinese translation of `docs/zh_cn/model_zoo.md` (#28)
- Add Chinese translation of `docs/zh_cn/faq.md` (#33)

17.5.4 Contributors

A total of 13 developers contributed to this release. Thanks @zytx121 @yangxue0827 @jbwang1997 @liuyanyi @DangChuong-DC @RangeKing @liufeinuaa @np-csu @akmalulkhairin @SheffieldCao @BrotherHappy @Abyssaledge @q3394101

CHAPTER

EIGHTEEN

FREQUENTLY ASKED QUESTIONS (TO BE UPDATED)

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.

18.1 MMCV Installation

- Compatibility issue between MMCV and MMDetection; "ConvWS is already registered in conv layer"; "AssertionError: MMCV==xxx is used but incompatible. Please install mmcv>=xxx, <=xxx."

Compatible MMCV, MMDetection and MMRotate versions are shown as below. Please install the correct version of them to avoid installation issues.

Note:

1. If you want to install mmrotate-v0.x, the compatible MMRotate and MMCV versions table can be found at here. Please choose the correct version of MMCV to avoid installation issues.
2. In MMCV-v2.x, `mmcv-full` is rename to `mmcv`, if you want to install `mmcv` without CUDA ops, you can install `mmcv-lite`.

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

18.2 PyTorch/CUDA Environment

- "invalid device function" or "no kernel image is available for execution".
  1. Check if your cuda runtime version (under `/usr/local/`), `nvcc --version` and `conda list cudatoolkit` version match.
  2. Run `python mmdet/utils/collect_env.py` to check whether PyTorch, torchvision, and MMCV are built for the correct GPU architecture. You may need to set `TORCH_CUDA_ARCH_LIST` to reinstall MMCV. The GPU arch table could be found here, i.e. run `TORCH_CUDA_ARCH_LIST=7.0 pip install mmcv-full` to build MMCV for Volta GPUs. The compatibility issue could happen when using old GPUS, e.g., Tesla K80 (3.7) on colab.
  3. Check whether the running environment is the same as that when mmcv/mmdet has compiled. For example, you may compile mmcv using CUDA 10.0 but run it on CUDA 9.0 environments.
- "undefined symbol" or "cannot open xxx.so".

1. If those symbols are CUDA/C++ symbols (e.g., libcudart.so or GLIBCXX), check whether the CUDA/GCC runtimes are the same as those used for compiling mmcv, i.e. run `python mmdet/utils/collect_env.py` to see if `"MMCV Compiler"/"MMCV CUDA Compiler"` is the same as `"GCC"/"CUDA_HOME"`.
2. If those symbols are PyTorch symbols (e.g., symbols containing caffe, aten, and TH), check whether the PyTorch version is the same as that used for compiling mmcv.
3. Run `python mmdet/utils/collect_env.py` to check whether PyTorch, torchvision, and MMCV are built by and running on the same environment.

- "setuptools.sandbox.UnpickleableException: DistutilsSetupError("each element of 'ext_modules' option must be an Extension instance or 2-tuple")"
  1. If you are using miniconda rather than anaconda, check whether Cython is installed as indicated in #3379. You need to manually install Cython first and then run command `pip install -r requirements.txt`.
  2. You may also need to check the compatibility between the `setuptools`, `Cython`, and `PyTorch` in your environment.
- "Segmentation fault".
  1. Check you GCC version and use GCC 5.4. This usually caused by the incompatibility between PyTorch and the environment (e.g., GCC < 4.9 for PyTorch). We also recommend the users to avoid using GCC 5.5 because many feedbacks report that GCC 5.5 will cause "segmentation fault" and simply changing it to GCC 5.4 could solve the problem.
  2. Check whether PyTorch is correctly installed and could use CUDA op, e.g. type the following command in your terminal.

     ```
     python -c 'import torch; print(torch.cuda.is_available())'
     ```

     And see whether they could correctly output results.
  3. If Pytorch is correctly installed, check whether MMCV is correctly installed.

     ```
     python -c 'import mmcv; import mmcv.ops'
     ```

     If MMCV is correctly installed, then there will be no issue of the above two commands.
  4. If MMCV and Pytorch is correctly installed, you man use `ipdb`, `pdb` to set breakpoints or directly add 'print' in mmdetection code and see which part leads the segmentation fault.

18.3 E2CNN

- "ImportError: cannot import name 'container_bacs' from 'torch._six'"
  1. This is because `container_abcs` has been removed since PyTorch 1.9.
  2. Replace

     ```
     from torch.six import container_abcs
     ```

     in `python3.7/site-packages/e2cnn/nn/modules/module_list.py` with

     ```
     TORCH_MAJOR = int(torch.__version__.split('.')[0])
     TORCH_MINOR = int(torch.__version__.split('.')[1])
     if TORCH_MAJOR ==1 and TORCH_MINOR < 8:
         from torch.six import container_abcs
     ```

(continues on next page)

(continued from previous page)

```
else:
    import collections.abs as container_abcs
```

3. Or downgrade the version of Pytorch.

18.4 Training

- "Loss goes Nan"
  1. Check if the dataset annotations are valid: zero-size bounding boxes will cause the regression loss to be Nan due to the commonly used transformation for box regression. Some small size (width or height are smaller than 1) boxes will also cause this problem after data augmentation (e.g., instaboost). So check the data and try to filter out those zero-size boxes and skip some risky augmentations on the small-size boxes when you face the problem.
  2. Reduce the learning rate: the learning rate might be too large due to some reasons, e.g., change of batch size. You can rescale them to the value that could stably train the model.
  3. Extend the warmup iterations: some models are sensitive to the learning rate at the start of the training. You can extend the warmup iterations, e.g., change the `warmup_iters` from 500 to 1000 or 2000.
  4. Add gradient clipping: some models requires gradient clipping to stabilize the training process. The default of `grad_clip` is `None`, you can add gradient clippint to avoid gradients that are too large, i.e., set `optimizer_config=dict(_delete_=True, grad_clip=dict(max_norm=35, norm_type=2))` in your config file. If your config does not inherits from any basic config that contains `optimizer_config=dict(grad_clip=None)`, you can simply add `optimizer_config=dict(grad_clip=dict(max_norm=35, norm_type=2))`.
- "GPU out of memory"
  1. There are some scenarios when there are large amounts of ground truth boxes, which may cause OOM during target assignment. You can set `gpu_assign_thr=N` in the config of assigner thus the assigner will calculate box overlaps through CPU when there are more than N GT boxes.
  2. Set `with_cp=True` in the backbone. This uses the sublinear strategy in PyTorch to reduce GPU memory cost in the backbone.
  3. Try mixed precision training by setting `fp16 = dict(loss_scale='dynamic')` in the config file.
- "RuntimeError: Expected to have finished reduction in the prior iteration before starting a new one"
  1. This error indicates that your module has parameters that were not used in producing loss. This phenomenon may be caused by running different branches in your code in DDP mode.
  2. You can set `find_unused_parameters = True` in the config to solve the above problems or find those unused parameters manually.

18.5 Evaluation

- COCO Dataset, AP or AR = -1
  1. According to the definition of COCO dataset, the small and medium areas in an image are less than 1024 (32*32), 9216 (96*96), respectively.
  2. If the corresponding area has no object, the result of AP and AR will set to -1.

CHAPTER

NINETEEN

ENGLISH

CHAPTER

TWENTY

CHAPTER

TWENTYONE

INDICES AND TABLES

- genindex
- search