MMTracking

Release 1.0.0rc1

MMTracking Authors

Apr 27, 2023

GET STARTED

CHAPTER ONE

INTRODUCTION

MMTracking is an open source video perception toolbox by PyTorch. It is a part of OpenMMLab project.

It supports 4 video tasks:

- Video object detection (VID)
- Single object tracking (SOT)
- Multiple object tracking (MOT)
- Video instance segmentation (VIS)

CHAPTER TWO

MAJOR FEATURES

- **The First Unified Video Perception Platform**

  We are the first open source toolbox that unifies versatile video perception tasks including video object detection, multiple object tracking, single object tracking and video instance segmentation.

- **Modular Design**

  We decompose the video perception framework into different components and one can easily construct a customized method by combining different modules.

- **Simple, Fast and Strong**

  Simple: MMTracking interacts with other OpenMMLab projects. It is built upon MMDetection that we can capitalize any detector only through modifying the configs.

  Fast: All operations run on GPUs. The training and inference speeds are faster than or comparable to other implementations.

  Strong: We reproduce state-of-the-art models and some of them even outperform the official implementations.

CHAPTER THREE

GETTING STARTED

Please refer to *get_started.md* for the basic usage of MMTracking.

A Colab tutorial is provided. You may preview the notebook here or directly run it on Colab.

CHAPTER

FOUR

USER GUIDES

There are some basic usage guides, including:

- *configs*
- *dataset preparation*
- *inference*
- *train and test*
- *visualization*
- *analysis tools*

If you want to learn more advanced guides, you can refer to:

- *data flow*
- *structures*
- *models*
- *datasets*
- *transforms*
- *evaluation*
- *engine*
- convention
- *add modules*
- *add datasets*
- *add transforms*
- *add metrics*
- customized runtime

CHAPTER

FIVE

BENCHMARK AND MODEL ZOO

Results and models are available in the *model zoo*.

CHAPTER

SIX

CONTRIBUTING

We appreciate all contributions to improve MMTracking. Please refer to CONTRIBUTING.md for the contributing guideline and this discussion for development roadmap.

CHAPTER

SEVEN

FAQ

If you encounter any problems in the process of using MMTracking, you can firstly refer to *FAQ*. If not solved, you can post an issue and we will give a response as soon as possible.

CHAPTER

EIGHT

PREREQUISITES

- Linux | macOS | Windows
- Python 3.6+
- PyTorch 1.6+
- CUDA 9.2+ (If you build PyTorch from source, CUDA 9.0 is also compatible)
- GCC 5+
- MMCV
- MMEngine
- MMDetection

The compatible MMTracking, MMEngine, MMCV, and MMDetection versions are as below. Please install the correct version to avoid installation issues.

CHAPTER

NINE

INSTALLATION

9.1 Detailed Instructions

1. Create a conda virtual environment and activate it.

```
conda create -n open-mmlab python=3.9 -y
conda activate open-mmlab
```

2. Install PyTorch and torchvision following the official instructions. Here we use PyTorch 1.10.0 and CUDA 11.1. You may also switch to other version by specifying the version number.

Install with conda

```
conda install pytorch=1.11.0 torchvision cudatoolkit=11.3 -c pytorch
```

Install with pip

```
pip install torch==1.11.0+cu113 torchvision==0.12.0+cu113 --extra-index-url https://
↪download.pytorch.org/whl/cu113
```

3. Install MMEngine

```
pip install mmengine
```

4. Install mmcv, we recommend you to install the pre-build package as below.

```
pip install 'mmcv>=2.0.0rc1' -f https://download.openmmlab.com/mmcv/dist/{cu_
↪version}/{torch_version}/index.html
```

mmcv is only compiled on PyTorch 1.x.0 because the compatibility usually holds between 1.x.0 and 1.x.1. If your PyTorch version is 1.x.1, you can install mmcv compiled with PyTorch 1.x.0 and it usually works well.

```
# We can ignore the micro version of PyTorch
pip install 'mmcv>=2.0.0rc1' -f https://download.openmmlab.com/mmcv/dist/cu113/
↪torch1.11.0/index.html
```

See here for different versions of MMCV compatible to different PyTorch and CUDA versions. Optionally you can choose to compile mmcv from source by the following command

```
git clone -b 2.x https://github.com/open-mmlab/mmcv.git
cd mmcv
MMCV_WITH_OPS=1 pip install -e .  # package mmcv, which contains cuda ops, will be␣
↪installed after this step
```

(continues on next page)

(continued from previous page)

```
# pip install -e .  # package mmcv, which contains no cuda ops, will be installed␣
↪after this step
cd ..
```

Important: You need to run pip uninstall mmcv-lite first if you have mmcv installed. Because if mmcv-lite and mmcv are both installed, there will be ModuleNotFoundError.

5. Install MMDetection

```
pip install 'mmdet>=3.0.0rc0'
```

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

```
git clone -b 3.x https://github.com/open-mmlab/mmdetection.git
cd mmdetection
pip install -r requirements/build.txt
pip install -v -e .  # or "python setup.py develop"
```

6. Clone the MMTracking repository.

```
git clone -b 1.x https://github.com/open-mmlab/mmtracking.git
cd mmtracking
```

7. Install build requirements and then install MMTracking.

```
pip install -r requirements/build.txt
pip install -v -e .  # or "python setup.py develop"
```

8. Install extra dependencies

- For MOT evaluation (required):

```
pip install git+https://github.com/JonathonLuiten/TrackEval.git
```

- For VOT evaluation (optional)

```
pip install git+https://github.com/votchallenge/toolkit.git
```

- For LVIS evaluation (optional):

```
pip install git+https://github.com/lvis-dataset/lvis-api.git
```

- For TAO evaluation (optional):

```
pip install git+https://github.com/TAO-Dataset/tao.git
```

Note:

a. Following the above instructions, MMTracking is installed on `dev` mode , any local modifications made to the code will take effect without the need to reinstall it.

b. If you would like to use `opencv-python-headless` instead of `opencv-python`, you can install it before installing MMCV.

9.2 A from-scratch setup script

Assuming that you already have CUDA 10.1 installed, here is a full script for setting up MMTracking with conda.

```
conda create -n open-mmlab python=3.9 -y
conda activate open-mmlab

conda install pytorch=1.11.0 torchvision cudatoolkit=11.3 -c pytorch

pip install mmengine

# install the latest mmcv
pip install 'mmcv>=2.0.0rc1' -f https://download.openmmlab.com/mmcv/dist/cu113/torch1.11.
↪0/index.html

# install mmdetection
pip install 'mmdet>=3.0.0rc0'

# install mmtracking
git clone -b 1.x https://github.com/open-mmlab/mmtracking.git
cd mmtracking
pip install -r requirements/build.txt
pip install -v -e .
pip install git+https://github.com/JonathonLuiten/TrackEval.git
pip install git+https://github.com/votchallenge/toolkit.git (optional)
pip install git+https://github.com/lvis-dataset/lvis-api.git (optional)
pip install git+https://github.com/TAO-Dataset/tao.git (optional)
```

9.3 Developing with multiple MMTracking versions

The train and test scripts already modify the `PYTHONPATH` to ensure the script use the MMTracking in the current directory.

To use the default MMTracking installed in the environment rather than that you are working with, you can remove the following line in those scripts

```
PYTHONPATH="$(dirname $0)/..":$PYTHONPATH
```

CHAPTER TEN

VERIFICATION

To verify whether MMTracking and the required environment are installed correctly, we can run **one of** MOT, VIS, VID and SOT demo scripts:

Here is an example for MOT demo:

```
python demo/demo_mot_vis.py \
    configs/mot/deepsort/deepsort_faster-rcnn-r50-fpn_8xb2-4e_mot17halftrain_test-
↪mot17halfval.py \
    --input demo/demo.mp4 \
    --output mot.mp4
```

If you want to run more other demos, you can refer to *inference guides*

CHAPTER

ELEVEN

TRAIN & TEST

11.1 Learn about Configs

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

We incorporate modular and inheritance design into our config system, which is convenient to conduct various experiments. If you wish to inspect the config file, you may run `python tools/misc/print_config.py /PATH/TO/CONFIG` to see the complete config.

11.1.1 A brief description of a complete config

A complete config usually contains the following primary fields:

- `model`: the basic config of model, which may contain `data_preprocessor`, modules (e.g., `detector`, `motion`),`train_cfg`, `test_cfg`, etc.
- `train_dataloader`: the config of training dataloader, which usually contains `batch_size`, `num_workers`, `sampler`, `dataset`, etc.
- `val_dataloader`: the config of validation dataloader, which is similar with `train_dataloader`.
- `test_dataloader`: the config of testing dataloader, which is similar with `train_dataloader`.
- `val_evaluator`: the config of validation evaluator. For example, `type='CocoVideoMetric'` for VID task on the ILSVRC benchmark, `type='MOTChallengeMetrics'` for MOT task on the MOTChallenge benchmarks.
- `test_evaluator`: the config of testing evaluator, which is similar with `val_evaluator`.
- `train_cfg`: the config of training loop. For example, `type='EpochBasedTrainLoop'`.
- `val_cfg`: the config of validation loop. For example, `type='ValLoop'`.
- `test_cfg`: the config of testing loop. For example, `type='TestLoop'`.
- `default_hooks`: the config of default hooks, which may include hooks for timer, logger, param_scheduler, checkpoint, sampler_seed, visualization, etc.
- `vis_backends`: the config of visualization backends, which uses `type='LocalVisBackend'` as default.
- `visualizer`: the config of visualizer. For example, `type='DetLocalVisualizer'` for VID task, and `type='TrackLocalVisualizer'` for MOT, VIS, SOT, VOS tasks.
- `param_scheduler`: the config of parameter scheduler, which usually sets the learning rate scheduler.
- `optim_wrapper`: the config of optimizer wrapper, which contains optimization-related information, for example optimizer, gradient clipping, etc.
- `load_from`: load models as a pre-trained model from a given path.

- `resume`: If `True`, resume checkpoints from `load_from`, and the training will be resumed from the epoch when the checkpoint is saved.

11.1.2 Modify 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. We present several examples as follows. For more details, please refer to MMEngine.

- **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.detector.backbone.norm_eval=False` changes the 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 testing pipeline `test_dataloader.dataset.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 test_dataloader.dataset.pipeline.0.type=LoadImageFromWebcam`.

- **Update values of list/tuples.**

  Maybe the value to be updated is a list or a tuple. For example, you can change the key `mean` of `data_preprocessor` by specifying `--cfg-options model.data_preprocessor.mean=[0,0,0]`. Note that **NO** white space is allowed inside the specified value.

11.1.3 Config File Structure

There are 3 basic component types under `config/_base_`, i.e., dataset, model and default_runtime. Many methods could be easily constructed with one of each like DFF, FGFA, SELSA, SORT, DeepSORT. The configs that are composed by components from `_base_` are called *primitive*.

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

For easy understanding, we recommend contributors to inherit from exiting methods. For example, if some modification is made base on Faster R-CNN, user may first inherit the basic Faster R-CNN structure by specifying `_base_ = ../../_base_/models/faster-rcnn_r50-dc5.py`, then modify the necessary fields in the config files.

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

Please refer to MMEngine for detailed documentation.

11.1.4 Config Name Style

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

```
{method}_{module}_{train_cfg}_{train_data}_{test_data}
```

- `{method}`: method name, like `dff`, `deepsort`, `siamese_rpn`.
- `{module}`: basic modules of the method, like `faster-rcnn_r50_fpn`.
- `{train_cfg}`: training config which usually contains batch size, epochs, etc, like `8xb4-80e`.
- `{train_data}`: training data, like `mot17halftrain`.

- {test_data}: testing data, like test-mot17halfval.

11.1.5 FAQ

Ignore some fields in the base configs

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

Use intermediate variables in configs

Some intermediate variables are used in the configs files, like train_pipeline/test_pipeline in datasets. It's worth noting that when modifying intermediate variables in the children configs, user need to pass the intermediate variables into corresponding fields again. For example, we would like to use testing strategy of adaptive stride to test SELSA. ref_img_sampler is intermediate variable we would like to modify.

```
_base_ = ['./selsa_faster-rcnn_r50-dc5_8xb1-7e_imagenetvid.py']
# dataset settings
ref_img_sampler=dict(
    _delete_=True,
    num_ref_imgs=14,
    frame_range=[-7, 7],
    method='test_with_adaptive_stride')
val_dataloader = dict(
    dataset=dict(ref_img_sampler=ref_img_sampler))
test_dataloader = dict(
    dataset=dict(ref_img_sampler=ref_img_sampler))
```

We first define the new ref_img_sampler and pass them into val_dataloader and test_dataloader.

11.2 Dataset Preparation

This page provides the instructions for dataset preparation on existing benchmarks, include

- Video Object Detection
  - ILSVRC
- Multiple Object Tracking
  - MOT Challenge
  - CrowdHuman
  - LVIS
  - TAO
  - DanceTrack
- Single Object Tracking
  - LaSOT
  - UAV123
  - TrackingNet
  - OTB100

    - GOT10k
    - VOT2018
- Video Instance Segmentation
    - YouTube-VIS

11.2.1 1. Download Datasets

Please download the datasets from the official websites. It is recommended to symlink the root of the datasets to `$MMTRACKING/data`.

1.1 Video Object Detection

- For the training and testing of video object detection task, only ILSVRC dataset is needed.
- The `Lists` under `ILSVRC` contains the txt files from here.

1.2 Multiple Object Tracking

- For the training and testing of multi object tracking task, one of the MOT Challenge datasets (e.g. MOT17, TAO and DanceTrack) are needed, CrowdHuman and LVIS can be served as comlementary dataset.
- The `annotations` under `tao` contains the official annotations from here.
- The `annotations` under `lvis` contains the official annotations of lvis-v0.5 which can be downloaded according to here. The synset mapping file `coco_to_lvis_synset.json` used in `./tools/dataset_converters/tao/merge_coco_with_lvis.py` script can be found here.
- For users in China, the following datasets can be downloaded from OpenDataLab with high speed:
    - MOT17
    - CrowdHuman
    - LVIS
    - TAO

1.3 Single Object Tracking

- For the training and testing of single object tracking task, the MSCOCO, ILSVRC, LaSOT, UAV123, TrackingNet, OTB100, GOT10k and VOT2018 datasets are needed.
- For OTB100 dataset, you don't need to download the dataset from the official website manually, since we provide a script to download it.

```
# download OTB100 dataset by web crawling
python ./tools/dataset_converters/otb100/download_otb100.py -o ./data/OTB100/zips -p 8
```

- For VOT2018, we use the official downloading script.

```
# download VOT2018 dataset by web crawling
python ./tools/dataset_converters/vot/download_vot.py --dataset vot2018 --save_path ./
↪data/VOT2018/data
```

- For users in China, the following datasets can be downloaded from OpenDataLab with high speed:
  - LaSOT
  - UAV123
  - TrackingNet
  - OTB100
  - GOT-10k
  - VOT2018

1.4 Video Instance Segmentation

- For the training and testing of video instance segmetatioon task, only one of YouTube-VIS datasets (e.g. YouTube-VIS 2019) is needed.
- YouTube-VIS 2019 dataset can be download from OpenDataLab (recommended for users in China): https://opendatalab.com/YouTubeVIS2019/download

1.5 Data Structure

If your folder structure is different from the following, you may need to change the corresponding paths in config files.

```
mmtracking
├── mmtrack
├── tools
├── configs
├── data
│   ├── coco
│   │   ├── train2017
│   │   ├── val2017
│   │   ├── test2017
│   │   ├── annotations
│   │
│   ├── ILSVRC
│   │   ├── Data
│   │   │   ├── DET
|   │   │   │   ├── train
|   │   │   │   ├── val
|   │   │   │   ├── test
│   │   │   ├── VID
|   │   │   │   ├── train
|   │   │   │   ├── val
|   │   │   │   ├── test
│   │   ├── Annotations
│   │   │   ├── DET
|   │   │   │   ├── train
|   │   │   │   ├── val
│   │   │   ├── VID
|   │   │   │   ├── train
|   │   │   │   ├── val
|   │   ├── Lists
```

(continues on next page)

(continued from previous page)

```
│   │
│   ├── MOT15/MOT16/MOT17/MOT20
│   │   ├── train
│   │   ├── test
│   │
│   ├── DanceTrack
│   │   ├── train
│   │   ├── val
│   │   ├── test
│   │
│   ├── crowdhuman
│   │   ├── annotation_train.odgt
│   │   ├── annotation_val.odgt
│   │   ├── train
│   │   │   ├── Images
│   │   │   ├── CrowdHuman_train01.zip
│   │   │   ├── CrowdHuman_train02.zip
│   │   │   ├── CrowdHuman_train03.zip
│   │   ├── val
│   │   │   ├── Images
│   │   │   ├── CrowdHuman_val.zip
│   │
│   ├── lvis
│   │   ├── train (the same as coco/train2017)
│   │   ├── val (the same as coco/val2017)
│   │   ├── test (the same as coco/test2017)
│   │   ├── annotations
│   │   │   ├── coco_to_lvis_synset.json
│   │   │   ├── lvis_v0.5_train.json
│   │   │   ├── lvis_v0.5_val.json
│   │   │   ├── lvis_v1_train.json
│   │   │   ├── lvis_v1_val.json
│   │   │   ├── lvis_v1_image_info_test_challenge.json
│   │   │   ├── lvis_v1_image_info_test_dev.json
│   │
│   ├── tao
│   │   ├── annotations
│   │   │   ├── test_without_annotations.json
│   │   │   ├── train.json
│   │   │   ├── validation.json
│   │   │   ├── ......
│   │   ├── test
│   │   │   ├── ArgoVerse
│   │   │   ├── AVA
│   │   │   ├── BDD
│   │   │   ├── Charades
│   │   │   ├── HACS
│   │   │   ├── LaSOT
│   │   │   ├── YFCC100M
│   │   ├── train
│   │   ├── val
│   │
```

(continues on next page)

(continued from previous page)

```
│     ├── LaSOT_full
│     │   ├── LaSOTBenchmark
│     │   │   ├── airplane
|     │   │   │   ├── airplane-1
|     │   │   │   ├── airplane-2
|     │   │   │   ├── ......
│     │   │   ├── ......
│     │
│     ├── UAV123
│     │   ├── data_seq
│     │   │   ├── UAV123
│     │   │   │   ├── bike1
│     │   │   │   ├── boat1
│     │   │   │   ├── ......
│     │   ├── anno
│     │   │   ├── UAV123
│     │
│     ├── TrackingNet
│     │   ├── TEST.zip
│     │   ├── TRAIN_0.zip
│     │   ├── ......
│     │   ├── TRAIN_11.zip
│     │
│     ├── OTB100
│     │   |── zips
│     │   │   |── Basketball.zip
│     │   │   |── Biker.zip
│     │   │   |──
│     │
│     ├── GOT10k
│     │   |── full_data
│     │   │   |── train_data
│     │   │   │   ├── GOT-10k_Train_split_01.zip
│     │   │   │   ├── ......
│     │   │   │   ├── GOT-10k_Train_split_19.zip
│     │   │   │   ├── list.txt
│     │   │   |── test_data.zip
│     │   │   |── val_data.zip
│     │
|     ├── VOT2018
|     |   ├── data
|     |   |   ├── ants1
|     │   |   │   ├──color
│     │
│     ├── youtube_vis_2019
│     │   │── train
│     │   │   │── JPEGImages
│     │   │   │── ......
│     │   │── valid
│     │   │   │── JPEGImages
│     │   │   │── ......
│     │   │── test
```

(continues on next page)

(continued from previous page)

```
│   │   │   │── JPEGImages
│   │   │   │── ......
│   │   │── train.json (the official annotation files)
│   │   │── valid.json (the official annotation files)
│   │   │── test.json (the official annotation files)
│   │
│   │── youtube_vis_2021
│   │   │── train
│   │   │   │── JPEGImages
│   │   │   │── instances.json (the official annotation files)
│   │   │   │── ......
│   │   │── valid
│   │   │   │── JPEGImages
│   │   │   │── instances.json (the official annotation files)
│   │   │   │── ......
│   │   │── test
│   │   │   │── JPEGImages
│   │   │   │── instances.json (the official annotation files)
│   │   │   │── ......
```

11.2.2 2. Convert Annotations

We use CocoVID to maintain all datasets in this codebase. In this case, you need to convert the official annotations to this style. We provide scripts and the usages are as following:

```
# ImageNet DET
python ./tools/dataset_converters/ilsvrc/imagenet2coco_det.py -i ./data/ILSVRC -o ./data/
↪ILSVRC/annotations

# ImageNet VID
python ./tools/dataset_converters/ilsvrc/imagenet2coco_vid.py -i ./data/ILSVRC -o ./data/
↪ILSVRC/annotations

# MOT17
# The processing of other MOT Challenge dataset is the same as MOT17
python ./tools/dataset_converters/mot/mot2coco.py -i ./data/MOT17/ -o ./data/MOT17/
↪annotations --split-train --convert-det
python ./tools/dataset_converters/mot/mot2reid.py -i ./data/MOT17/ -o ./data/MOT17/reid -
↪-val-split 0.2 --vis-threshold 0.3

# DanceTrack
python ./tools/dataset_converters/dancetrack/dancetrack2coco.py -i ./data/DanceTrack ./
↪data/DanceTrack/annotations

# CrowdHuman
python ./tools/dataset_converters/mot/crowdhuman2coco.py -i ./data/crowdhuman -o ./data/
↪crowdhuman/annotations

# LVIS
# Merge annotations from LVIS and COCO for training QDTrack
python ./tools/dataset_converters/tao/merge_coco_with_lvis.py --lvis ./data/lvis/
↪annotations/lvis_v0.5_train.json --coco ./data/coco/annotations/instances_train2017.
↪json --mapping ./data/lvis/annotations/coco_to_lvis_synset.json --output-json ./data/
↪lvis/annotations/lvisv0.5+coco_train.json
```

(continues on next page)

(continued from previous page)

```
# TAO
# Generate filtered json file for QDTrack
python ./tools/dataset_converters/tao/tao2coco.py -i ./data/tao/annotations --filter-
↪classes

# LaSOT
python ./tools/dataset_converters/lasot/gen_lasot_infos.py -i ./data/LaSOT_full/
↪LaSOTBenchmark -o ./data/LaSOT_full/annotations

# UAV123
# download annotations
# due to the annotations of all videos in UAV123 are inconsistent, we just download the␣
↪information file generated in advance.
wget https://download.openmmlab.com/mmtracking/data/uav123_infos.txt -P data/UAV123/
↪annotations

# TrackingNet
# unzip files in 'data/trackingnet/*.zip'
bash ./tools/dataset_converters/trackingnet/unzip_trackingnet.sh ./data/TrackingNet
# generate annotations
python ./tools/dataset_converters/trackingnet/gen_trackingnet_infos.py -i ./data/
↪TrackingNet -o ./data/TrackingNet/annotations

# OTB100
# unzip files in 'data/otb100/zips/*.zip'
bash ./tools/dataset_converters/otb100/unzip_otb100.sh ./data/OTB100
# download annotations
# due to the annotations of all videos in OTB100 are inconsistent, we just need to␣
↪download the information file generated in advance.
wget https://download.openmmlab.com/mmtracking/data/otb100_infos.txt -P data/OTB100/
↪annotations

# GOT10k
# unzip 'data/GOT10k/full_data/test_data.zip', 'data/GOT10k/full_data/val_data.zip' and␣
↪files in 'data/GOT10k/full_data/train_data/*.zip'
bash ./tools/dataset_converters/got10k/unzip_got10k.sh ./data/GOT10k
# generate annotations
python ./tools/dataset_converters/got10k/gen_got10k_infos.py -i ./data/GOT10k -o ./data/
↪GOT10k/annotations

# VOT2018
python ./tools/dataset_converters/vot/gen_vot_infos.py -i ./data/VOT2018 -o ./data/
↪VOT2018/annotations --dataset_type vot2018

# YouTube-VIS 2019
python ./tools/dataset_converters/youtubevis/youtubevis2coco.py -i ./data/youtube_vis_
↪2019 -o ./data/youtube_vis_2019/annotations --version 2019

# YouTube-VIS 2021
python ./tools/dataset_converters/youtubevis/youtubevis2coco.py -i ./data/youtube_vis_
↪2021 -o ./data/youtube_vis_2021/annotations --version 2021
```

The folder structure will be as following after your run these scripts:

```
mmtracking
├── mmtrack
├── tools
├── configs
├── data
│   ├── coco
│   │   ├── train2017
│   │   ├── val2017
│   │   ├── test2017
│   │   ├── annotations
│   │
│   ├── ILSVRC
│   │   ├── Data
│   │   │   ├── DET
|   │   │   │   ├── train
|   │   │   │   ├── val
|   │   │   │   ├── test
│   │   │   ├── VID
|   │   │   │   ├── train
|   │   │   │   ├── val
|   │   │   │   ├── test
│   │   ├── Annotations (the official annotation files)
│   │   │   ├── DET
|   │   │   │   ├── train
|   │   │   │   ├── val
│   │   │   ├── VID
|   │   │   │   ├── train
|   │   │   │   ├── val
│   │   ├── Lists
│   │   ├── annotations (the converted annotation files)
│   │
|   ├── MOT15/MOT16/MOT17/MOT20
|   |   ├── train
|   |   ├── test
|   |   ├── annotations
|   |   ├── reid
│   │   │   ├── imgs
│   │   │   ├── meta
│   │
│   ├── DanceTrack
│   │   ├── train
│   │   ├── val
│   │   ├── test
│   │   ├── annotations
│   │
│   ├── crowdhuman
│   │   ├── annotation_train.odgt
│   │   ├── annotation_val.odgt
│   │   ├── train
│   │   │   ├── Images
│   │   │   ├── CrowdHuman_train01.zip
│   │   │   │
```

(continues on next page)

(continued from previous page)

```
│   │   │   ├── CrowdHuman_train02.zip
│   │   │   ├── CrowdHuman_train03.zip
│   │   ├── val
│   │   │   ├── Images
│   │   │   ├── CrowdHuman_val.zip
│   │   ├── annotations
│   │   │   ├── crowdhuman_train.json
│   │   │   ├── crowdhuman_val.json
│   │
│   ├── lvis
│   │   ├── train (the same as coco/train2017)
│   │   ├── val (the same as coco/val2017)
│   │   ├── test (the same as coco/test2017)
│   │   ├── annotations
│   │   │   ├── coco_to_lvis_synset.json
│   │   │   ├── lvisv0.5+coco_train.json
│   │   │   ├── lvis_v0.5_train.json
│   │   │   ├── lvis_v0.5_val.json
│   │   │   ├── lvis_v1_train.json
│   │   │   ├── lvis_v1_val.json
│   │   │   ├── lvis_v1_image_info_test_challenge.json
│   │   │   ├── lvis_v1_image_info_test_dev.json
│   │
│   ├── tao
│   │   ├── annotations
│   │   │   ├── test_482_classes.json
│   │   │   ├── test_without_annotations.json
│   │   │   ├── train.json
│   │   │   ├── train_482_classes.json
│   │   │   ├── validation.json
│   │   │   ├── validation_482_classes.json
│   │   │   ├── ......
│   │   ├── test
│   │   │   ├── ArgoVerse
│   │   │   ├── AVA
│   │   │   ├── BDD
│   │   │   ├── Charades
│   │   │   ├── HACS
│   │   │   ├── LaSOT
│   │   │   ├── YFCC100M
│   │   ├── train
│   │   ├── val
│   │
│   ├── LaSOT_full
│   │   ├── LaSOTBenchmark
│   │   │   ├── airplane
|   │   │   │   ├── airplane-1
|   │   │   │   ├── airplane-2
|   │   │   │   ├── ......
│   │   │   ├── ......
│   │   ├── annotations
│   │
```

(continues on next page)

(continued from previous page)

```
│   ├── UAV123
│   │   ├── data_seq
│   │   │   ├── UAV123
│   │   │   │   ├── bike1
│   │   │   │   ├── boat1
│   │   │   │   ├── ......
│   │   ├── anno (the official annotation files)
│   │   │   ├── UAV123
│   │   ├── annotations (the converted annotation file)
│   │
│   ├── TrackingNet
│   │   ├── TEST
│   │   │   ├── anno (the official annotation files)
│   │   │   ├── zips
│   │   │   ├── frames (the unzipped folders)
│   │   │   │   ├── 0-6LB4FqxoE_0
│   │   │   │   ├── 07Ysk1C0ZX0_0
│   │   │   │   ├── ......
│   │   ├── TRAIN_0
│   │   │   ├── anno (the official annotation files)
│   │   │   ├── zips
│   │   │   ├── frames (the unzipped folders)
│   │   │   │   ├── -3TIfnTSM6c_2
│   │   │   │   ├── a1qoB1eERn0_0
│   │   │   │   ├── ......
│   │   ├── ......
│   │   ├── TRAIN_11
│   │   ├── annotations (the converted annotation file)
│   │
│   ├── OTB100
│   │   ├── zips
│   │   │   ├── Basketball.zip
│   │   │   ├── Biker.zip
│   │   │   │── ......
│   │   ├── annotations
│   │   ├── data
│   │   │   ├── Basketball
│   │   │   │   ├── img
│   │   │   ├── ......
│   │
│   ├── GOT10k
│   │   │── full_data
│   │   │   │── train_data
│   │   │   │   ├── GOT-10k_Train_split_01.zip
│   │   │   │   ├── ......
│   │   │   │   ├── GOT-10k_Train_split_19.zip
│   │   │   │   ├── list.txt
│   │   │   │── test_data.zip
│   │   │   │── val_data.zip
│   │   │── train
│   │   │   ├── GOT-10k_Train_000001
│   │   │   │   ├── ......
```

(continues on next page)

(continued from previous page)

```
│   │   │   ├── GOT-10k_Train_009335
│   │   │   ├── list.txt
│   │   │── test
│   │   │   ├── GOT-10k_Test_000001
│   │   │   │   ├── ......
│   │   │   ├── GOT-10k_Test_000180
│   │   │   ├── list.txt
│   │   │── val
│   │   │   ├── GOT-10k_Val_000001
│   │   │   │   ├── ......
│   │   │   ├── GOT-10k_Val_000180
│   │   │   ├── list.txt
│   │   │── annotations
│   │
|   ├── VOT2018
|   |   ├── data
|   |   |   ├── ants1
|   |   │   │   ├──color
|   |   ├── annotations
│   │   │   ├── ......
│   │
│   ├── youtube_vis_2019
│   │   │── train
│   │   │   │── JPEGImages
│   │   │   │── ......
│   │   │── valid
│   │   │   │── JPEGImages
│   │   │   │── ......
│   │   │── test
│   │   │   │── JPEGImages
│   │   │   │── ......
│   │   │── train.json (the official annotation files)
│   │   │── valid.json (the official annotation files)
│   │   │── test.json (the official annotation files)
│   │   │── annotations (the converted annotation file)
│   │
│   ├── youtube_vis_2021
│   │   │── train
│   │   │   │── JPEGImages
│   │   │   │── instances.json (the official annotation files)
│   │   │   │── ......
│   │   │── valid
│   │   │   │── JPEGImages
│   │   │   │── instances.json (the official annotation files)
│   │   │   │── ......
│   │   │── test
│   │   │   │── JPEGImages
│   │   │   │── instances.json (the official annotation files)
│   │   │   │── ......
│   │   │── annotations (the converted annotation file)
```

The folder of annotations in ILSVRC

There are 3 JSON files in `data/ILSVRC/annotations`:

`imagenet_det_30plus1cls.json`: JSON file containing the annotations information of the training set in ImageNet DET dataset. The `30` in `30plus1cls` denotes the overlapped 30 categories in ImageNet VID dataset, and the `1cls` means we take the other 170 categories in ImageNet DET dataset as a category, named as `other_categeries`.

`imagenet_vid_train.json`: JSON file containing the annotations information of the training set in ImageNet VID dataset.

`imagenet_vid_val.json`: JSON file containing the annotations information of the validation set in ImageNet VID dataset.

The folder of annotations and reid in MOT15/MOT16/MOT17/MOT20

We take MOT17 dataset as examples, the other datasets share similar structure.

There are 8 JSON files in `data/MOT17/annotations`:

`train_cocoformat.json`: JSON file containing the annotations information of the training set in MOT17 dataset.

`train_detections.pkl`: Pickle file containing the public detections of the training set in MOT17 dataset.

`test_cocoformat.json`: JSON file containing the annotations information of the testing set in MOT17 dataset.

`test_detections.pkl`: Pickle file containing the public detections of the testing set in MOT17 dataset.

`half-train_cocoformat.json`, `half-train_detections.pkl`, `half-val_cocoformat.json`and `half-val_detections.pkl` share similar meaning with `train_cocoformat.json` and `train_detections.pkl`. The `half` means we split each video in the training set into half. The first half videos are denoted as `half-train` set, and the second half videos are denoted as`half-val` set.

The structure of `data/MOT17/reid` is as follows:

```
reid
├── imgs
│   ├── MOT17-02-FRCNN_000002
│   │   ├── 000000.jpg
│   │   ├── 000001.jpg
│   │   ├── ...
│   ├── MOT17-02-FRCNN_000003
│   │   ├── 000000.jpg
│   │   ├── 000001.jpg
│   │   ├── ...
├── meta
│   ├── train_80.txt
│   ├── val_20.txt
```

The `80` in `train_80.txt` means the proportion of the training dataset to the whole ReID dataset is 80%. While the proportion of the validation dataset is 20%.

For training, we provide a annotation list `train_80.txt`. Each line of the list contains a filename and its corresponding ground-truth labels. The format is as follows:

```
MOT17-05-FRCNN_000110/000018.jpg 0
MOT17-13-FRCNN_000146/000014.jpg 1
MOT17-05-FRCNN_000088/000004.jpg 2
MOT17-02-FRCNN_000009/000081.jpg 3
```

`MOT17-05-FRCNN_000110` denotes the 110-th person in `MOT17-05-FRCNN` video.

For validation, The annotation list `val_20.txt` remains the same as format above.

Images in `reid/imgs` are cropped from raw images in `MOT17/train` by the corresponding `gt.txt`. The value of ground-truth labels should fall in range `[0, num_classes - 1]`.

The folder of annotations in crowdhuman

There are 2 JSON files in `data/crowdhuman/annotations`:

`crowdhuman_train.json`: JSON file containing the annotations information of the training set in CrowdHuman dataset. `crowdhuman_val.json`: JSON file containing the annotations information of the validation set in CrowdHuman dataset.

The folder of annotations in lvis

There are 8 JSON files in `data/lvis/annotations`

`coco_to_lvis_synset.json`: JSON file containing the mapping relationship between COCO and LVIS categories.

`lvisv0.5+coco_train.json`: JSON file containing the merged annotations.

`lvis_v0.5_train.json`: JSON file containing the annotations information of the training set in lvisv0.5.

`lvis_v0.5_val.json`: JSON file containing the annotations information of the validation set in lvisv0.5.

`lvis_v1_train.json`: JSON file containing the annotations information of the training set in lvisv1.

`lvis_v1_val.json`: JSON file containing the annotations information of the validation set in lvisv1.

`lvis_v1_image_info_test_challenge.json`: JSON file containing the annotations information of the testing set in lvisv1 available for year-round evaluation.

`lvis_v1_image_info_test_dev.json`: JSON file containing the annotations information of the testing set in lvisv1 available only once a year for LVIS Challenge.

The folder of annotations in tao

There are 9 JSON files in `data/tao/annotations`:

`test_categories.json`: JSON file containing a list of categories which will be evaluated on the TAO test set.

`test_without_annotations.json`: JSON for test videos. The 'images' and 'videos' fields contain the images and videos that will be evaluated on the test set.

`test_482_classes.json`: JSON file containing the converted results for test set.

`train.json`: JSON file containing annotations for LVIS categories in TAO train.

`train_482_classes.json`: JSON file containing the converted results for train set.

`train_with_freeform.json`: JSON file containing annotations for all categories in TAO train.

`validation.json`: JSON file containing annotations for LVIS categories in TAO train.

`validation_482_classes.json`: JSON file containing the converted results for validation set.

`validation_with_freeform.json`: JSON file containing annotations for all categories in TAO validation.

The folder of annotations in LaSOT

There are 2 JSON files in `data/LaSOT_full/annotations`:

`lasot_train.json`: JSON file containing the annotations information of the training set in LaSOT dataset. `lasot_test.json`: JSON file containing the annotations information of the testing set in LaSOT dataset.

There are 2 TEXT files in `data/LaSOT_full/annotations`:

`lasot_train_infos.txt`: TEXT file containing the annotations information of the training set in LaSOT dataset. `lasot_test_infos.txt`: TEXT file containing the annotations information of the testing set in LaSOT dataset.

The folder of annotations in UAV123

There are only 1 JSON files in `data/UAV123/annotations`:

`uav123.json`: JSON file containing the annotations information of the UAV123 dataset.

There are only 1 TEXT files in `data/UAV123/annotations`:

`uav123_infos.txt`: TEXT file containing the information of the UAV123 dataset.

The folder of frames and annotations in TrackingNet

There are 511 video directories of TrackingNet testset in `data/TrackingNet/TEST/frames`, and each video directory contains all images of the video. Similar file structures can be seen in `data/TrackingNet/TRAIN_{*}/frames`.

There are 2 JSON files in `data/TrackingNet/annotations`:

`trackingnet_test.json`: JSON file containing the annotations information of the testing set in TrackingNet dataset. `trackingnet_train.json`: JSON file containing the annotations information of the training set in TrackingNet dataset.

There are 2 TEXT files in `data/TrackingNet/annotations`:

`trackingnet_test_infos.txt`: TEXT file containing the information of the testing set in TrackingNet dataset. `trackingnet_train_infos.txt`: TEXT file containing the information of the training set in TrackingNet dataset.

The folder of data and annotations in OTB100

There are 98 video directories of OTB100 dataset in `data/OTB100/data`, and the `img` folder under each video directory contains all images of the video.

There are only 1 JSON files in `data/OTB100/annotations`:

`otb100.json`: JSON file containing the annotations information of the OTB100 dataset.

There are only 1 TEXT files in `data/OTB100/annotations`:

`otb100_infos.txt`: TEXT file containing the information of the OTB100 dataset.

The folder of frames and annotations in GOT10k

There are training video directories in `data/GOT10k/train`, and each video directory contains all images of the video. Similar file structures can be seen in `data/GOT10k/test` and `data/GOT10k/val`.

There are 3 JSON files in `data/GOT10k/annotations`:

`got10k_train.json`: JSON file containing the annotations information of the training set in GOT10k dataset.

`got10k_test.json`: JSON file containing the annotations information of the testing set in GOT10k dataset.

`got10k_val.json`: JSON file containing the annotations information of the valuation set in GOT10k dataset.

There are 5 TEXT files in `data/GOT10k/annotations`:

`got10k_train_infos.txt`: TEXT file containing the information of the training set in GOT10k dataset.

`got10k_test_infos.txt`: TEXT file containing the information of the testing set in GOT10k dataset.

`got10k_val_infos.txt`: TEXT file containing the information of the valuation set in GOT10k dataset.

`got10k_train_vot_infos.txt`: TEXT file containing the information of the `train_vot` split in GOT10k dataset.

`got10k_val_vot_infos.txt`: TEXT file containing the information of the `val_vot` split in GOT10k dataset.

The folder of data and annotations in VOT2018

There are 60 video directories of VOT2018 dataset in `data/VOT2018/data`, and the `color` folder under each video directory contains all images of the video.

There are only 1 JSON files in `data/VOT2018/annotations`:

`vot2018.json`: JSON file containing the annotations information of the VOT2018 dataset.

There are only 1 TEXT files in `data/VOT2018/annotations`:

`vot2018_infos.txt`: TEXT file containing the information of the VOT2018 dataset.

The folder of annotations in youtube_vis_2019/youtube_vis2021

There are 3 JSON files in `data/youtube_vis_2019/annotations` or `data/youtube_vis_2021/annotations`:

`youtube_vis_2019_train.json`/`youtube_vis_2021_train.json`: JSON file containing the annotations information of the training set in youtube_vis_2019/youtube_vis2021 dataset.

`youtube_vis_2019_valid.json`/`youtube_vis_2021_valid.json`: JSON file containing the annotations information of the validation set in youtube_vis_2019/youtube_vis2021 dataset.

`youtube_vis_2019_test.json`/`youtube_vis_2021_test.json`: JSON file containing the annotations information of the testing set in youtube_vis_2019/youtube_vis2021 dataset.

11.3 Inference

We provide demo scripts to inference a given video or a folder that contains continuous images. The source codes are available here.

Note that if you use a folder as the input, the image names there must be **sortable** , which means we can re-order the images according to the numbers contained in the filenames. We now only support reading the images whose filenames end with `.jpg`, `.jpeg` and `.png`.

11.3.1 Inference VID models

This script can inference an input video with a video object detection model.

```
python demo/demo_vid.py \
    ${CONFIG_FILE}\
    --input ${INPUT} \
    --checkpoint ${CHECKPOINT_FILE} \
    [--output ${OUTPUT}] \
    [--device ${DEVICE}] \
    [--show]
```

The `INPUT` and `OUTPUT` support both *mp4 video* format and the *folder* format.

Optional arguments:

- `OUTPUT`: Output of the visualized demo. If not specified, the `--show` is obligate to show the video on the fly.
- `DEVICE`: The device for inference. Options are `cpu` or `cuda:0`, etc.
- `--show`: Whether show the video on the fly.

Examples:

Assume that you have already downloaded the checkpoints to the directory `checkpoints/`, your video filename is `demo.mp4`, and your output path is the `./outputs/`

```
python ./demo/demo_vid.py \
    configs/vid/selsa/selsa_faster-rcnn_r50-dc5_8xb1-7e_imagenetvid.py \
    --input ./demo.mp4 \
    --checkpoint checkpoints/selsa_faster_rcnn_r101_dc5_1x_imagenetvid_20201218_172724-
↪aa961bcc.pth \
    --output ./outputs/ \
    --show
```

11.3.2 Inference MOT/VIS models

This script can inference an input video / images with a multiple object tracking or video instance segmentation model.

```
python demo/demo_mot_vis.py \
    ${CONFIG_FILE} \
    --input ${INPUT} \
    [--output ${OUTPUT}] \
    [--checkpoint ${CHECKPOINT_FILE}] \
    [--score-thr ${SCORE_THR} \
```

(continues on next page)

(continued from previous page)

```
    [--device ${DEVICE}] \
    [--show]
```

The INPUT and OUTPUT support both *mp4 video* format and the *folder* format.

Important: For DeepSORT, SORT, Tracktor, StrongSORT, they need both the weight of the reid and the weight of the detector. Therefore, we can't use --checkpoint to specify it. We need to use init_cfg in the configuration file to set the weight path. Other algorithms such as ByteTrack, OCSORT and QDTrack need not pay attention to this.

Optional arguments:

- OUTPUT: Output of the visualized demo. If not specified, the --show is obligate to show the video on the fly.
- CHECKPOINT_FILE: The checkpoint is optional in case that you already set up the pretrained models in the config by the key init_cfg.
- SCORE_THR: The threshold of score to filter bboxes.
- DEVICE: The device for inference. Options are cpu or cuda:0, etc.
- --show: Whether show the video on the fly.

Examples of running mot model:

```
# Example 1: do not specify --checkpoint to use the default init_cfg
python demo/demo_mot_vis.py \
    configs/mot/sort/sort_faster-rcnn_r50_fpn_8xb2-4e_mot17halftrain_test-mot17halfval.
↪py \
    --input demo/demo.mp4 \
    --output mot.mp4

# Example 2: use --checkpoint
python demo/demo_mot_vis.py \
    configs/mot/bytetrack/bytetrack_yolox_x_8xb4-80e_crowdhuman-mot17halftrain_test-
↪mot17halfval.py \
    --input demo/demo.mp4 \
    --checkpoint checkpoints/bytetrack_yolox_x_crowdhuman_mot17-private-half_20211218_
↪205500-1985c9f0.pth \
    --output mot.mp4
```

Examples of running vis model:

Assume that you have already downloaded the checkpoints to the directory checkpoints/, your video filename is demo.mp4, and your output path is the ./outputs/

```
python demo/demo_mot_vis.py \
    configs/vis/masktrack_rcnn/masktrack-rcnn_mask-rcnn_r50_fpn_8xb1-12e_youtubevis2019.
↪py \
    --input demo.mp4 \
    --checkpoint checkpoints/masktrack_rcnn_r50_fpn_12e_youtubevis2019_20211022_194830-
↪6ca6b91e.pth \
    --output ./outputs/ \
    --show
```

11.3.3 Inference SOT models

This script can inference an input video with a single object tracking model.

```
python demo/demo_sot.py \
    ${CONFIG_FILE}\
    --input ${INPUT} \
    --checkpoint ${CHECKPOINT_FILE} \
    [--output ${OUTPUT}] \
    [--device ${DEVICE}] \
    [--show] \
    [--gt_bbox_file ${GT_BBOX_FILE}]
```

The `INPUT` and `OUTPUT` support both *mp4 video* format and the *folder* format.

Optional arguments:

- `OUTPUT`: Output of the visualized demo. If not specified, the `--show` is obligate to show the video on the fly.
- `DEVICE`: The device for inference. Options are `cpu` or `cuda:0`, etc.
- `--show`: Whether show the video on the fly.
- `GT_BBOX_FILE`: The gt_bbox file path of the video. We only use the gt_bbox of the first frame. If not specified, you would draw init bbox of the video manually.

Examples:

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

```
python ./demo/demo_sot.py \
    configs/sot/siamese_rpn/siamese-rpn_r50_8xb28-20e_imagenetvid-imagenetdet-coco_test-
↪lasot.py \
    --input ${VIDEO_FILE} \
    --checkpoint checkpoints/siamese_rpn_r50_1x_lasot_20211203_151612-da4b3c66.pth \
    --output ${OUTPUT} \
    --show
```

11.4 Learn to train and test

11.4.1 Train

This section will show how to train existing models on supported datasets. The following training environments are supported:

- CPU
- single GPU
- single node multiple GPUs
- multiple nodes

You can also manage jobs with Slurm.

Important:

- You can change the evaluation interval during training by modifying the `train_cfg` as `train_cfg = dict(val_interval=10)`. That means evaluating the model every 10 epochs.

- The default learning rate in all config files is for 8 GPUs. According to the Linear Scaling Rule, you need to set the learning rate proportional to the batch size if you use different GPUs or images per GPU, e.g., `lr=0.01` for 8 GPUs * 1 img/gpu and lr=0.04 for 16 GPUs * 2 imgs/gpu.
- During training, log files and checkpoints will be saved to the working directory, which is specified by CLI argument `--work-dir`. It uses `./work_dirs/CONFIG_NAME` as default.
- If you want the mixed precision training, simply specify CLI argument `--amp`.

1. Train on CPU

The model is default put on cuda device. Only if there are no cuda devices, the model will be put on cpu. So if you want to train the model on CPU, you need to `export CUDA_VISIBLE_DEVICES=-1` to disable GPU visibility first. More details in MMEngine.

```
CUDA_VISIBLE_DEVICES=-1 python tools/train.py ${CONFIG_FILE} [optional arguments]
```

An example of training the VID model DFF on CPU:

```
CUDA_VISIBLE_DEVICES=-1 python tools/train.py configs/vid/dff/dff_faster-rcnn_r50-dc5_
↪8xb1-7e_imagenetvid.py
```

2. Train on single GPU

If you want to train the model on single GPU, you can directly use the `tools/train.py` as follows.

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

You can use `export CUDA_VISIBLE_DEVICES=$GPU_ID` to select the GPU.

An example of training the MOT model ByteTrack on single GPU:

```
CUDA_VISIBLE_DEVICES=2 python tools/train.py configs/mot/bytetrack/bytetrack_yolox_x_
↪8xb4-80e_crowdhuman-mot17halftrain_test-mot17halfval.py
```

3. Train on single node multiple GPUs

We provide `tools/dist_train.sh` to launch training on multiple GPUs. The basic usage is as follows.

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

If you would like to 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.

For example, you can set the port in commands as follows.

```
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
```

An example of training the SOT model SiameseRPN++ on single node multiple GPUs:

```
bash ./tools/dist_train.sh ./configs/sot/siamese_rpn/siamese-rpn_r50_8xb16-20e_
↪imagenetvid-imagenetdet-coco_test-otb100.py 8
```

4. Train on multiple nodes

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 bash tools/dist_train.sh
↪$CONFIG $GPUS
```

On the second machine:

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

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

5. Train with Slurm

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

The basic usage is as follows.

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

An example of training the VIS model MaskTrack R-CNN with Slurm:

```
PORT=29501 \
GPUS_PER_NODE=8 \
SRUN_ARGS="--quotatype=reserved" \
bash ./tools/slurm_train.sh \
mypartition \
masktrack \
configs/vis/masktrack_rcnn/masktrack-rcnn_mask-rcnn_r50_fpn_8xb1-12e_youtubevis2019.py \
./work_dirs/MaskTrack_RCNN \
8
```

11.4.2 Test

This section will show how to test existing models on supported datasets. The following testing environments are supported:

- CPU
- single GPU
- single node multiple GPUs
- multiple nodes

You can also manage jobs with Slurm.

Important:

- You can set the results saving path by modifying the key `outfile_prefix` in evaluator. For example, `val_evaluator = dict(outfile_prefix='results/stark_st1_trackingnet')`. Otherwise, a temporal file will be created and will be removed after evaluation.

- If you just want the formatted results without evaluation, you can set `format_only=True`. For example, `test_evaluator = dict(type='YouTubeVISMetric', metric='youtube_vis_ap', outfile_prefix='./youtube_vis_results', format_only=True)`

1. Test on CPU

The model is default put on cuda device. Only if there are no cuda devices, the model will be put on cpu. So if you want to test the model on CPU, you need to `export CUDA_VISIBLE_DEVICES=-1` to disable GPU visibility first. More details in MMEngine.

```
CUDA_VISIBLE_DEVICES=-1 python tools/test.py ${CONFIG_FILE} [optional arguments]
```

An example of testing the VID model DFF on CPU:

```
CUDA_VISIBLE_DEVICES=-1 python tools/test.py configs/vid/dff/dff_faster-rcnn_r50-dc5_
↪8xb1-7e_imagenetvid.py --checkpoint https://download.openmmlab.com/mmtracking/vid/dff/
↪dff_faster_rcnn_r50_dc5_1x_imagenetvid/dff_faster_rcnn_r50_dc5_1x_imagenetvid_20201227_
↪213250-548911a4.pth
```

2. Test on single GPU

If you want to test the model on single GPU, you can directly use the `tools/test.py` as follows.

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

You can use `export CUDA_VISIBLE_DEVICES=$GPU_ID` to select the GPU.

An example of testing the MOT model ByteTrack on single GPU:

```
CUDA_VISIBLE_DEVICES=2 python tools/test.py configs/mot/bytetrack/bytetrack_yolox_x_8xb4-
↪80e_crowdhuman-mot17halftrain_test-mot17halfval.py --checkpoint https://download.
↪openmmlab.com/mmtracking/mot/bytetrack/bytetrack_yolox_x/bytetrack_yolox_x_crowdhuman_
↪mot17-private-half_20211218_205500-1985c9f0.pth
```

3. Test on single node multiple GPUs

We provide `tools/dist_test.sh` to launch testing on multiple GPUs. The basic usage is as follows.

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

An example of testing the SOT model SiameseRPN++ on single node multiple GPUs:

```
bash ./tools/dist_test.sh ./configs/sot/siamese_rpn/siamese-rpn_r50_8xb16-20e_
↪imagenetvid-imagenetdet-coco_test-otb100.py 8 --checkpoint https://download.openmmlab.
↪com/mmtracking/sot/siamese_rpn/siamese_rpn_r50_1x_otb100/siamese_rpn_r50_20e_otb100_
↪20220421_144232-6b8f1730.pth
```

4. Test on multiple nodes

You can test on multiple nodes, which is similar with "Train on multiple nodes".

5. Test with Slurm

On a cluster managed by Slurm, you can use `slurm_test.sh` to spawn testing jobs. It supports both single-node and multi-node testing.

The basic usage is as follows.

```
bash ./tools/slurm_test.sh ${PARTITION} ${JOB_NAME} ${CONFIG_FILE} ${GPUS}
```

An example of testing the VIS model MaskTrack R-CNN with Slurm:

```
PORT=29501 \
GPUS_PER_NODE=8 \
SRUN_ARGS="--quotatype=reserved" \
bash ./tools/slurm_test.sh \
mypartition \
masktrack \
configs/vis/masktrack_rcnn/masktrack-rcnn_mask-rcnn_r50_fpn_8xb1-12e_youtubevis2019.py \
8 \
--checkpoint https://download.openmmlab.com/mmtracking/vis/masktrack_rcnn/masktrack_rcnn_
↪r50_fpn_12e_youtubevis2019/masktrack_rcnn_r50_fpn_12e_youtubevis2019_20211022_194830-
↪6ca6b91e.pth
```

CHAPTER

TWELVE

USEFUL TOOLS

12.1 Learn about Visualization

12.1.1 Local Visualization

This section will present how to visualize the detection/tracking results with local visualizer.

If you want to draw prediction results, you can turn this feature on by setting `draw=True` in `TrackVisualizationHook` as follows.

```
default_hooks = dict(visualization=dict(type='TrackVisualizationHook', draw=True))
```

Specifically, the `TrackVisualizationHook` has the following arguments:

- `draw`: whether to draw prediction results. If it is False, it means that no drawing will be done. Defaults to False.
- `interval`: The interval of visualization. Defaults to 30.
- `score_thr`: The threshold to visualize the bboxes and masks. Defaults to 0.3.
- `show`: Whether to display the drawn image. Default to False.
- `wait_time`: The interval of show (s). Defaults to 0.
- `test_out_dir`: directory where painted images will be saved in testing process.
- `file_client_args`: Arguments to instantiate a FileClient. Defaults to `dict(backend='disk')`.

In the `TrackVisualizationHook`, a visualizer will be called to implement visualization, i.e., `DetLocalVisualizer` for VID task and `TrackLocalVisualizer` for MOT, VIS, SOT, VOS tasks. We will present the details below. You can refer to MMEngine for more details about Visualization and Hook.

Detection Visualization

We realize the detection visualization with class `DetLocalVisualizer`. You can call it as follows.

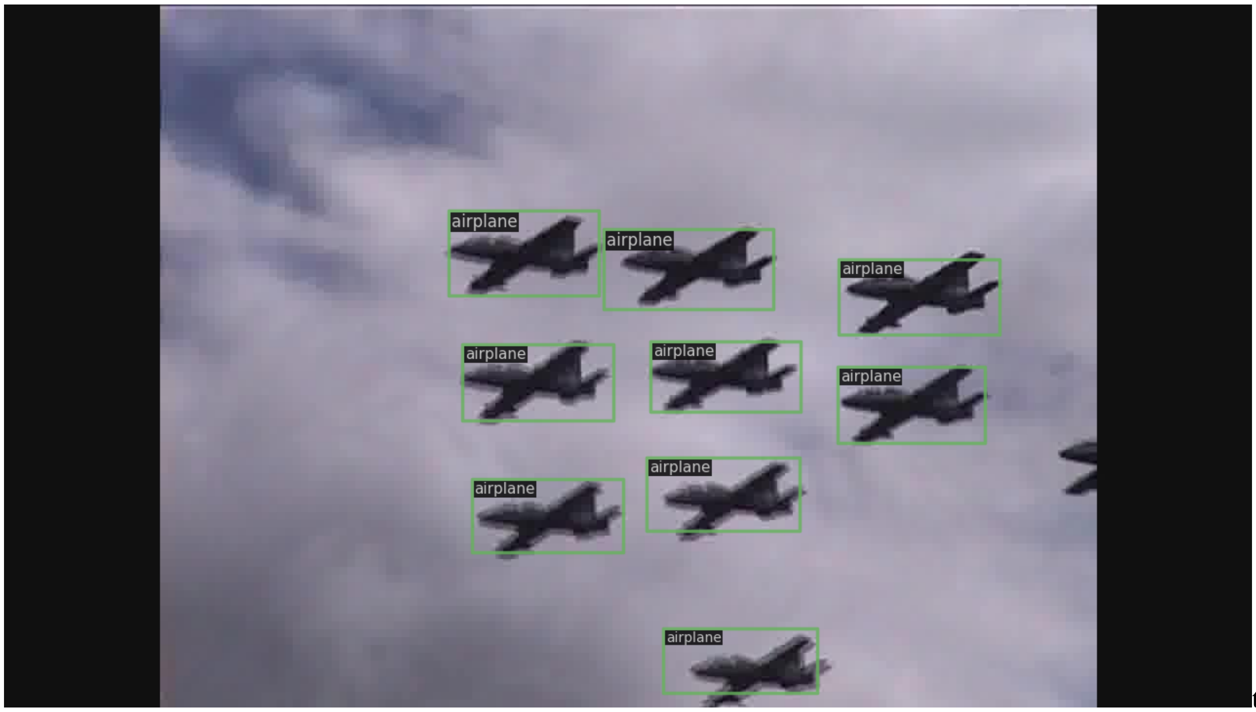
```
visualizer = dict(type='DetLocalVisualizer')
```

It has the following arguments:

- `name`: Name of the instance. Defaults to 'visualizer'.
- `image`: The origin image to draw. The format should be RGB. Defaults to None.
- `vis_backends`: Visual backend config list. Defaults to None.
- `save_dir`: Save file dir for all storage backends. If it is None, the backend storage will not save any data.

- `bbox_color`: Color of bbox lines. The tuple of color should be in BGR order. Defaults to None.
- `text_color`: Color of texts. The tuple of color should be in BGR order. Defaults to (200, 200, 200).
- `line_width`: The linewidth of lines. Defaults to 3.
- `alpha`: The transparency of bboxes or mask. Defaults to 0.8.

Here is a visualization example of DFF:



test_img_29

Tracking Visualization

We realize the tracking visualization with class `TrackLocalVisualizer`. You can call it as follows.

```
visualizer = dict(type='TrackLocalVisualizer')
```

It has the following arguments, which has the same meaning of that in `DetLocalVisualizer`.

`name`, `image`, `vis_backends`, `save_dir`, `line_width`, `alpha`.

Here is a visualization example of DeepSORT:

test_img_89

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

12.2 MOT Test-time Parameter Search

`tools/analysis_tools/mot/mot_param_search.py` can search the parameters of the `tracker` in MOT models. It is used as the same manner with `tools/test.py` but **different** in the configs.

Here is an example that shows how to modify the configs:

1. Define the desirable evaluation metrics to record.

   For example, you can define the `evaluator` as

   ```
   test_evaluator=dict(type='MOTChallengeMetrics', metric=['HOTA', 'CLEAR', 'Identity
   ↪'])
   ```

   Of course, you can also customize the content of `metric` in `test_evaluator`. You are free to choose one or more of `['HOTA', 'CLEAR', 'Identity']`.

2. Define the parameters and the values to search.

   Assume you have a tracker like

   ```
   model=dict(
       tracker=dict(
           type='BaseTracker',
           obj_score_thr=0.5,
           match_iou_thr=0.5
       )
   )
   ```

   If you want to search the parameters of the tracker, just change the value to a list as follow

```
model=dict(
    tracker=dict(
        type='BaseTracker',
        obj_score_thr=[0.4, 0.5, 0.6],
        match_iou_thr=[0.4, 0.5, 0.6, 0.7]
    )
)
```

Then the script will test the totally 12 cases and log the results.

12.3 MOT Error Visualize

`tools/analysis_tools/mot/mot_error_visualize.py` can visualize errors for multiple object tracking. This script needs the result of inference. By Default, the **red** bounding box denotes false positive, the **yellow** bounding box denotes the false negative and the **blue** bounding box denotes ID switch.

```
python tools/analysis_tools/mot/mot_error_visualize.py \
    ${CONFIG_FILE}\
    --input ${INPUT} \
    --result-dir ${RESULT_DIR} \
    [--out-dir ${OUTPUT}] \
    [--fps ${FPS}] \
    [--show] \
    [--backend ${BACKEND}]
```

The `RESULT_DIR` contains the inference results of all videos and the inference result is a `txt` file.

Optional arguments:

- `OUTPUT`: Output of the visualized demo. If not specified, the `--show` is obligate to show the video on the fly.
- `FPS`: FPS of the output video.
- `--show`: Whether show the video on the fly.
- `BACKEND`: The backend to visualize the boxes. Options are `cv2` and `plt`.

12.4 SiameseRPN++ Test-time Parameter Search

`tools/analysis_tools/sot/sot_siamrpn_param_search.py` can search the test-time tracking parameters in SiameseRPN++: `penalty_k`, `lr` and `window_influence`. You need to pass the searching range of each parameter into the argparser.

Example on UAV123 dataset:

```
./tools/analysis_tools/sot/dist_sot_siamrpn_param_search.sh [${CONFIG_FILE}] [$GPUS] \
[--checkpoint ${CHECKPOINT}] [--log ${LOG_FILENAME}] [--eval ${EVAL}] \
[--penalty-k-range 0.01,0.22,0.05] [--lr-range 0.4,0.61,0.05] [--win-infu-range 0.01,0.
↪22,0.05]
```

Example on OTB100 dataset:

```
./tools/analysis_tools/sot/dist_sot_siamrpn_param_search.sh [${CONFIG_FILE}] [$GPUS] \
[--checkpoint ${CHECKPOINT}] [--log ${LOG_FILENAME}] [--eval ${EVAL}] \
[--penalty-k-range 0.3,0.45,0.02] [--lr-range 0.35,0.5,0.02] [--win-infu-range 0.46,0.55,
↪0.02]
```

Example on VOT2018 dataset:

```
./tools/analysis_tools/sot/dist_sot_siamrpn_param_search.sh [${CONFIG_FILE}] [$GPUS] \
[--checkpoint ${CHECKPOINT}] [--log ${LOG_FILENAME}] [--eval ${EVAL}] \
[--penalty-k-range 0.01,0.31,0.05] [--lr-range 0.2,0.51,0.05] [--win-infu-range 0.3,0.56,
↪0.05]
```

12.5 Log Analysis

`tools/analysis_tools/analyze_logs.py` plots loss/mAP curves given a training log file.

```
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
  ```

12.6 Browse dataset

`tools/analysis_tools/browse_dataset.py` can visualize the training dataset to check whether the dataset configuration is correct.

Examples:

```
python tools/analysis_tools/browse_dataset.py ${CONFIG_FILE} [--show-interval ${SHOW_
↪INTERVAL}]
```

Optional arguments:

- `SHOW_INTERVAL`: The interval of show (s).
- `--not-show`: Whether do not show the images on the fly.

12.7 Show SOT evaluation results in video level

The SOT evaluation results are sorted in video level from largest to smallest by the Success metric. You can selectively show the performance results of some good cases or bad cases by setting `eval_show_video_indices`.

```
test_evaluator=dict(
    type='SOTMetric',
    options_after_eval=dict(eval_show_video_indices=10))
```

Here, `eval_show_video_indices` is used to index a `numpy.ndarray`. It can be `int` (positive or negative) or `list`. The positive number `k` means all the top-k reuslts while the negative number means the bottom-k results.

12.8 Save SOT evaluation results and plot them

Save the SOT evaluation result by setting the `SOTMetric` in the config.

```
test_evaluator = dict(
    type='SOTMetric',
    options_after_eval = dict(tracker_name = 'SiamRPN++', saved_eval_res_file = './
↪results/sot_results.json'))
```

The saved result is a dict in the format:

```
dict{tracker_name=dict(
      success = np.ndarray,
      norm_precision = np.ndarray,
      precision = np.ndarray)}
```

The metrics have shape (M,), where M is the number of values corresponding to different thresholds.

Given the saved results, you can plot them using the following command:

```
python ./tools/analysis_tools/sot/sot_plot_curve.py ./results --plot_save_path ./results
```

12.9 Save tracked results and playback them

Save the tracked result by setting the `SOTMetric` in the config.

```
test_evaluator = dict(
    type='SOTMetric',
    options_after_eval = dict(saved_track_res_path='./tracked_results'))
```

Playback the tracked results using the following command:

```
python ./tools/analysis_tools/sot/sot_playback.py  data/OTB100/data/Basketball/img/␣
↪tracked_results/basketball.txt --show --output results/basketball.mp4 --fps 20 --gt_
↪bboxes data/OTB100/data/Basketball/groundtruth_rect.txt
```

12.9.1 Visualization of feature map

Here is an example of calling the Visualizer in MMEngine:

```
# call visualizer at any position
visualizer = Visualizer.get_current_instance()
# set the image as background
visualizer.set_image(image=image)
# draw feature map on the image
drawn_img = visualizer.draw_featmap(feature_map, image, channel_reduction='squeeze_mean')
# show
visualizer.show(drawn_img)
# saved as ${saved_dir}/vis_data/vis_image/feat_0.png
visualizer.add_image('feature_map', drawn_img)
```

More details about visualization of feature map can be seen in visualizer docs and draw_featmap function

CHAPTER

THIRTEEN

BASIC CONCEPTS

CHAPTER

FOURTEEN

COMPONENT CUSTOMIZATION

CHAPTER FIFTEEN

MIGRATION FROM MMTRACKING 0.XX

Compared with the 0.xx versions of MMTracking, the latest 1.xx version of MMTracking has the following important modifications.

15.1 Overall Structures

The `core` in the old versions of MMTracking is splited into `engine`, `evaluation`, `structures`, `visualization` and `model/task_moduls` in the 1.xx version of MMTracking. Details can be seen in the user guides.

15.2 Configs

15.2.1 file names

old: `deepsort_faster-rcnn_fpn_4e_mot17-private-half.py`

new: `deepsort_faster-rcnn-resnet50-fpn_8x2bs-4e_mot17halftrain_test-mot17halfval.py`

15.2.2 keys of dataset loader

old

```
data = dict(
    samples_per_gpu=16,
    workers_per_gpu=4,
    persistent_workers=True,
    samples_per_epoch=60000,
    train=dict(
        type='GOT10kDataset',
        ann_file=data_root +
        'got10k/annotations/got10k_train_infos.txt',
        img_prefix=data_root + 'got10k',
        pipeline=train_pipeline,
        split='train',
        test_mode=False),
    val=dict(
        type='GOT10kDataset',
        ann_file=data_root + 'got10k/annotations/got10k_test_infos.txt',
```

(continues on next page)

(continued from previous page)

```
        img_prefix=data_root + 'got10k',
        pipeline=test_pipeline,
        split='test',
        test_mode=True),
    test=dict(
        type='GOT10kDataset',
        ann_file=data_root + 'got10k/annotations/got10k_test_infos.txt',
        img_prefix=data_root + 'got10k',
        pipeline=test_pipeline,
        split='test',
        test_mode=True))
```

new

```
train_dataloader = dict(
    batch_size=16,
    num_workers=4,
    persistent_workers=True,
    sampler=dict(type='QuotaSampler', samples_per_epoch=60000),
    dataset=dict(
        type='GOT10kDataset',
        data_root=data_root,
        ann_file='GOT10k/annotations/got10k_train_infos.txt',
        data_prefix=dict(img_path='GOT10k'),
        pipeline=train_pipeline,
        test_mode=False))
val_dataloader = dict(
    batch_size=1,
    num_workers=4,
    persistent_workers=True,
    drop_last=False,
    sampler=dict(type='VideoSampler'),
    dataset=dict(
        type='GOT10kDataset',
        data_root='data/',
        ann_file='GOT10k/annotations/got10k_test_infos.txt',
        data_prefix=dict(img_path='GOT10k'),
        pipeline=test_pipeline,
        test_mode=True))
test_dataloader = val_dataloader
```

15.2.3 keys of optimizer

old

```
optimizer = dict(
    type='AdamW',
    lr=0.0001,
    weight_decay=0.0001,
    paramwise_cfg=dict(
        custom_keys=dict(backbone=dict(lr_mult=0.1, decay_mult=1.0))))
```

(continues on next page)

(continued from previous page)

```
optimizer_config = dict(grad_clip=dict(max_norm=0.1, norm_type=2))
```

new

```
optim_wrapper = dict(
    type='OptimWrapper',
    optimizer=dict(type='AdamW', lr=0.0001, weight_decay=0.0001),
    clip_grad=dict(max_norm=0.1, norm_type=2),
    paramwise_cfg=dict(
        custom_keys=dict(backbone=dict(lr_mult=0.1, decay_mult=1.0))))
```

15.2.4 keys of learning scheduler

old

```
lr_config = dict(policy='step', step=[400])
```

new

```
param_scheduler = dict(type='MultiStepLR', milestones=[400], gamma=0.1)
```

15.3 Model

15.3.1 Data preprocessor

The 1.xx versions of MMtracking add TrackDataPreprocessor. The data out from the data pipeline is transformed by this module and then fed into the model.

15.3.2 Train

The training forward of models and heads is performed by calling `loss` function in their respective classes. The arguments of `loss` function in models contain a dict of `Tensor` and a list of `TrackDataSample`.

15.3.3 Test

The test forward of models and heads is performed by calling `predict` function in their respective classes. The arguments of `predict` function in models contain a dict of `Tensor` and a list of `TrackDataSample`.

15.4 Data

15.4.1 data structure

The 1.xx versions of MMtracking add two new data structure: TrackDataSample and ReIDDataSample. These data structures wrap the annotations and predictions from one image (sequence) and are used as interfaces between different components.

15.4.2 dataset class

The 1.xx versions of MMTracking add two base dataset classes which inherient from the `BaseDataset` in MMEngine: `BaseSOTDataset` and `BaseVideoDataset`. The former is only used in SOT and the latter is used for all other tasks.

15.4.3 data pipeline

1. Most of the transforms on image sequences in the old MMTracking are refactored in the latest MMTracking. Specifically, we use `TransformBroadcaster` to wrap the transforms of single image.

Some transforms on image sequences, such as `SeqCropLikeStark`, are reserved since `TransformBroadcaster` doesn't support setting different arguments respectively for each image in the sequence.

1. We pack the `VideoCollect`, `ConcatSameTypeFrames` and `SeqDefaultFormatBundle` in the old MMTracking into `PackTrackInputs` in the latest MMTracking.
2. The normalizaion in the pipeline in the old MMTracking is removed and this operation is implemented in the model forward.

15.4.4 data sampler

The 1.xx versions of MMtracking add `DATA_SAMPLERS` registry. You can customize different dataset samplers in the configs. Details about the samplers can be seen here.

15.5 Evaluation

The old versions of MMTarcking implement evaluation in the dataset class. In the 1.xx versions of MMTracking, we add `METRICS` registry. All evaluation are implemented in the metric classes registered in `METRICS`. Details can be seen here.

15.6 Visualization

The 1.xx versions of MMTracking add `TrackLocalVisualizer` and `DetLocalVisualizer` which are registered in `VISUALIZER`. Compared with the 0.xx versions of MMTracking, we support the visualization of images and feature maps. Details can be seen here.

15.7 Engine

The runner, hook, logging and optimizer in the training, evaluation and test are refactored in the 1.xx versions of MMTracking. Details can be seen in MMEngine.

CHAPTER

SIXTEEN

MMTRACK.APIS

CHAPTER

SEVENTEEN

MMTRACK.DATASETS

17.1 datasets

17.2 api_wrappers

17.3 samplers

17.4 transforms

CHAPTER EIGHTEEN

MMTRACK.ENGINE

18.1 hooks

18.2 schedulers

CHAPTER

NINETEEN

MMTRACK.EVALUATION

19.1 functional

19.2 metrics

CHAPTER

TWENTY

MMTRACK.MODELS

20.1 aggregators

20.2 backbones

20.3 data_preprocessors

20.4 filter

20.5 layers

20.6 losses

20.7 mot

20.8 motion

20.9 reid

20.10 roi_heads

20.11 sot

20.12 task_modules

20.13 track_heads

20.14 trackers

20.15 vid

20.16 vis

CHAPTER

TWENTYONE

MMTRACK.STRUCTURES

21.1 structures

21.2 bbox

CHAPTER

TWENTYTWO

MMTRACK.UTILS

22.1 utils

CHAPTER

TWENTYTHREE

MMTRACK.VISUALIZTION

23.1 visualiztion

CHAPTER

TWENTYFOUR

BENCHMARK AND MODEL ZOO

24.1 Common settings

- We use distributed training.
- All pytorch-style pretrained backbones on ImageNet are from PyTorch model zoo.
- For fair comparison with other codebases, we report the GPU memory as the maximum value of `torch.cuda.max_memory_allocated()` for all 8 GPUs. Note that this value is usually less than what `nvidia-smi` shows.
- We report the inference time as the total time of network forwarding and post-processing, excluding the data loading time. Results are obtained with the script `tools/analysis_tools/benchmark.py` which computes the average time on 2000 images.
- Speed benchmark environments

  HardWare

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

  Software environment

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

24.2 Baselines of video object detection

24.2.1 DFF (CVPR 2017)

Please refer to DFF for details.

24.2.2 FGFA (ICCV 2017)

Please refer to FGFA for details.

24.2.3 SELSA (ICCV 2019)

Please refer to SELSA for details.

24.2.4 Temporal RoI Align (AAAI 2021)

Please refer to Temporal RoI Align for details.

24.3 Baselines of multiple object tracking

24.3.1 SORT (ICIP 2016)

Please refer to SORT for details.

24.3.2 DeepSORT (ICIP 2017)

Please refer to DeepSORT for details.

24.3.3 Tracktor (ICCV 2019)

Please refer to Tracktor for details.

24.3.4 QDTrack (CVPR 2021)

Please refer to QDTrack for details.

24.3.5 ByteTrack (ECCV 2022)

Please refer to ByteTrack for details.

24.3.6 StrongSORT (arvix 2022)

Please refer to StrongSORT for details

24.4 Baselines of single object tracking

24.4.1 SiameseRPN++ (CVPR 2019)

Please refer to SiameseRPN++ for details.

24.4.2 PrDiMP (CVPR 2020)

Please refer to PrDiMP for details.

24.4.3 STARK (ICCV 2021)

Please refer to STARK for details.

24.5 Baselines of video instance segmentation

24.5.1 MaskTrack R-CNN (ICCV 2019)

Please refer to MaskTrack R-CNN for details

24.5.2 Mask2Former (CVPR 2022)

Please refer to Mask2Former for details.

CHAPTER

TWENTYFIVE

CHANGELOG

25.1 v1.0.0rc1 (10/10/2022)

MMTracking 1.0.0rc1 is the 2-nd version of MMTracking 1.x, a part of the OpenMMLab 2.0 projects.

Built upon the new training engine, MMTracking 1.x unifies the interfaces of datasets, models, evaluation, and visualization.

And there are some BC-breaking changes. Please check the migration tutorial for more details.

We also support more methods in MMTracking 1.x, such as StrongSORT for MOT, Mask2Former for VIS, PrDiMP for SOT.

25.2 v0.13.0 (29/04/2022)

25.2.1 Highlights

- Support tracking colab tutorial (#511)

25.2.2 New Features

- Refactor the training datasets of SiamRPN++ (#496), (#518)
- Support loading data from ceph for SOT datasets (#494)
- Support loading data from ceph for MOT challenge dataset (#517)
- Support evaluation metric for VIS task (#501)

25.2.3 Bug Fixes

- Fix a bug in the LaSOT datasets and update the pretrained models of STARK (#483), (#503)
- Fix a bug in the format_results function of VIS task (#504)

25.3 v0.12.0 (01/04/2022)

25.3.1 Highlights

- Support QDTrack algorithm in MOT (#433), (#451), (#461), (#469)

25.3.2 Bug Fixes

- Support empty tensor for selsa aggregator (#463)

25.4 v0.11.0 (04/03/2022)

25.4.1 Highlights

- Support STARK algorithm in SOT (#443), (#440), (#434), (#438), (#435), (#426)
- Support HOTA evaluation metrics for MOT (#417)

25.4.2 New Features

- Support TAO dataset in MOT (#415)

25.5 v0.10.0 (10/02/2022)

25.5.1 New Features

- Support CPU training (#404)

25.5.2 Improvements

- Refactor SOT datasets (#401), (#402), (#393)

25.6 v0.9.0 (05/01/2022)

25.6.1 Highlights

- Support arXiv 2021 manuscript 'ByteTrack: Multi-Object Tracking by Associating Every Detection Box' (#385), (#383), (#372)
- Support ICCV 2019 paper 'Video Instance Segmentation' (#304), (#303), (#298), (#292)

25.6.2 New Features

- Support CrowdHuman dataset for MOT (#366)
- Support VOT2018 dataset for SOT (#305)
- Support YouTube-VIS dataset for VIS (#290)

25.6.3 Bug Fixes

- Fix two significant bugs in SOT and provide new SOT pretrained models (#349)

25.6.4 Improvements

- Refactor LaSOT, TrackingNet dataset and support GOT-10K datasets (#296)
- Support persisitent workers (#348)

25.7 v0.8.0 (03/10/2021)

25.7.1 New Features

- Support OTB100 dataset in SOT (#271)
- Support TrackingNet dataset in SOT (#268)
- Support UAV123 dataset in SOT (#260)

25.7.2 Bug Fixes

- Fix a bug in mot_param_search.py (#270)

25.7.3 Improvements

- Use PyTorch sphinx theme (#274)
- Use pycocotools instead of mmpycocotools (#263)

25.8 v0.7.0 (03/09/2021)

25.8.1 Highlights

- Release code of AAAI 2021 paper 'Temporal ROI Align for Video Object Recognition' (#247)
- Refactor English documentations (#243)
- Add Chinese documentations (#248), (#250)

25.8.2 New Features

- Support fp16 training and testing (#230)
- Release model using ResNeXt-101 as backbone for all VID methods (#254)
- Support the results of Tracktor on MOT15, MOT16 and MOT20 datasets (#217)
- Support visualization for single gpu test (#216)

25.8.3 Bug Fixes

- Fix a bug in MOTP evaluation (#235)
- Fix two bugs in reid training and testing (#249)

25.8.4 Improvements

- Refactor anchor in SiameseRPN++ (#229)
- Unify model initialization (#235)
- Refactor unittest (#231)

25.9 v0.6.0 (30/07/2021)

25.9.1 Highlights

- Fix training bugs of all three tasks (#219), (#221)

25.9.2 New Features

- Support error visualization for mot task (#212)

25.9.3 Bug Fixes

- Fix a bug in SOT demo (#213)

25.9.4 Improvements

- Use MMCV registry (#220)
- Add README.md for reid training (#210)
- Modify dict keys of the outputs of SOT (#223)
- Add Chinese docs including install.md, quick_run.md, model_zoo.md, dataset.md (#205), (#214)

25.10 v0.5.3 (01/07/2021)

25.10.1 New Features

- Support ReID training (#177), (#179), (#180), (#181),
- Support MIM (#158)

25.10.2 Bug Fixes

- Fix evaluation hook (#176)
- Fix a typo in vid config (#171)

25.10.3 Improvements

- Refactor nms config (#167)

25.11 v0.5.2 (03/06/2021)

25.11.1 Improvements

- Fixed typos (#104, #121, #145)
- Added conference reference (#111)
- Updated the link of CONTRIBUTING to mmcv (#112)
- Adapt updates in mmcv (FP16Hook) (#114, #119)
- Added bibtex and links to other codebases (#122)
- Added docker files (#124)
- Used `collect_env` in mmcv (#129)
- Added and updated Chinese README (#135, #147, #148)

25.12 v0.5.1 (01/02/2021)

25.12.1 Bug Fixes

- Fixed ReID checkpoint loading (#80)
- Fixed empty tensor in `track_result` (#86)
- Fixed `wait_time` in MOT demo script (#92)

25.12.2 Improvements

- Support single-stage detector for DeepSORT (#100)

25.13 v0.5.0 (04/01/2021)

25.13.1 Highlights

- MMTracking is released!

25.13.2 New Features

- Support video object detection methods: DFF, FGFA, SELSA
- Support multi object tracking methods: SORT/DeepSORT, Tracktor
- Support single object tracking methods: SiameseRPN++

CHAPTER TWENTYSIX

INDICES AND TABLES

- genindex
- search