MMOCR

Release 0.3.0

OpenMMLab

Dec 24, 2021

GETTING STARTED

You can switch between English and Chinese in the lower-left corner of the layout.

CHAPTER

ONE

INSTALLATION

1.1 Prerequisites

- Linux (Windows is not officially supported)
- Python 3.7
- PyTorch 1.6 or higher
- torchvision 0.7.0
- CUDA 10.1
- NCCL 2
- GCC 5.4.0 or higher
- MMCV >= 1.3.8
- MMDetection >= 2.14.0

We have tested the following versions of OS and softwares:

- OS: Ubuntu 16.04
- CUDA: 10.1
- GCC(G++): 5.4.0
- MMCV 1.3.8
- MMDetection 2.14.0
- PyTorch 1.6.0
- torchvision 0.7.0

MMOCR depends on PyTorch and mmdetection.

1.2 Step-by-Step Installation Instructions

a. Create a conda virtual environment and activate it.

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

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

```
conda install pytorch==1.6.0 torchvision==0.7.0 cudatoolkit=10.1 -c pytorch
```

Note: Make sure that your compilation CUDA version and runtime CUDA version match. You can check the supported CUDA version for precompiled packages on the PyTorch website.

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

```
pip install mmcv-full -f https://download.openmmlab.com/mmcv/dist/{cu_version}/{torch_
↪version}/index.html
```

Please replace `{cu_version}` and `{torch_version}` in the url to your desired one. For example, to install the latest `mmcv-full` with `CUDA 11` and `PyTorch 1.7.0`, use the following command:

```
pip install mmcv-full -f https://download.openmmlab.com/mmcv/dist/cu110/torch1.7.0/
↪index.html
```

Note that mmocr 0.2.1 or later requires mmcv 1.3.8 or later.

If it compiles during installation, then please check that the cuda version and pytorch version **exactly** matches the version in the mmcv-full installation command. For example, pytorch 1.7.0 and 1.7.1 are treated differently.

See official installation for different versions of MMCV compatible to different PyTorch and CUDA versions.

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

d. Install mmdet, we recommend you to install the latest `mmdet` with pip. See here for different versions of `mmdet`.

```
pip install mmdet
```

Optionally you can choose to install `mmdet` following the official installation.

e. Clone the mmocr repository.

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

f. Install build requirements and then install MMOCR.

```
pip install -r requirements.txt
pip install -v -e . # or "python setup.py develop"
export PYTHONPATH=$(pwd):$PYTHONPATH
```

1.3 Full Set-up Script

Here is the full script for setting up mmocr with conda.

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

# install latest pytorch prebuilt with the default prebuilt CUDA version (usually the␣
↪latest)
conda install pytorch==1.6.0 torchvision==0.7.0 cudatoolkit=10.1 -c pytorch

# install the latest mmcv-full
pip install mmcv-full -f https://download.openmmlab.com/mmcv/dist/cu101/torch1.6.0/
↪index.html

# install mmdetection
pip install mmdet

# install mmocr
git clone https://github.com/open-mmlab/mmocr.git
cd mmocr

pip install -r requirements.txt
pip install -v -e .  # or "python setup.py develop"
export PYTHONPATH=$(pwd):$PYTHONPATH
```

1.4 Another option: Docker Image

We provide a Dockerfile to build an image.

```
# build an image with PyTorch 1.6, CUDA 10.1
docker build -t mmocr docker/
```

Run it with

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

1.5 Prepare Datasets

It is recommended to symlink the dataset root to `mmocr/data`. Please refer to datasets.md to prepare your datasets. If your folder structure is different, you may need to change the corresponding paths in config files.

The `mmocr` folder is organized as follows:

```
├── configs/
├── demo/
├── docker/
├── docs/
├── LICENSE
├── mmocr/
├── README.md
├── requirements/
```

(continues on next page)

(continued from previous page)

```
├── requirements.txt
├── resources/
├── setup.cfg
├── setup.py
├── tests/
├── tools/
```

CHAPTER

TWO

GETTING STARTED

In this guide we will show you some useful commands and familiarize you with MMOCR. We also provide a notebook that can help you get the most out of MMOCR.

2.1 Installation

Check out our *installation guide* for full steps.

2.2 Dataset Preparation

MMOCR supports numerous datasets which are classified by the type of their corresponding tasks. You may find their preparation steps in these sections: *Detection Datasets*, *Recognition Datasets*, *KIE Datasets* and *NER Datasets*.

2.3 Inference with Pretrained Models

You can perform end-to-end OCR on our demo image with one simple line of command:

```
python mmocr/utils/ocr.py demo/demo_text_ocr.jpg --print-result --imshow
```

Its detection result will be printed out and a new window will pop up with result visualization. More demo and full instructions can be found in Inference.

2.4 Training

2.4.1 Training with Toy Dataset

We provide a toy dataset under `tests/data` on which you can get a sense of training before the academic dataset is prepared.

For example, to train a text recognition task with `seg` method and toy dataset,

```
python tools/train.py configs/textrecog/seg/seg_r31_1by16_fpnocr_toy_dataset.py --
↪work-dir seg
```

To train a text recognition task with `sar` method and toy dataset,

```
python tools/train.py configs/textrecog/sar/sar_r31_parallel_decoder_toy_dataset.py --
↪work-dir sar
```

2.4.2 Training with Academic Dataset

Once you have prepared required academic dataset following our instruction, the only last thing to check is if the model's config points MMOCR to the correct dataset path. Suppose we want to train DBNet on ICDAR 2015, and part of `configs/textdet/dbnet/dbnet_r18_fpnc_1200e_icdar2015.py` looks like the following:

```
dataset_type = 'IcdarDataset'
data_root = 'data/icdar2015'
data = dict(
    train=dict(
        type=dataset_type,
        ann_file=data_root + '/instances_training.json',
        img_prefix=data_root + '/imgs',
        pipeline=train_pipeline)
    val=dict(
        type=dataset_type,
        ann_file=data_root + '/instances_test.json',
        img_prefix=data_root + '/imgs',
        pipeline=test_pipeline),
    test=dict(
        type=dataset_type,
        ann_file=data_root + '/instances_test.json',
        img_prefix=data_root + '/imgs',
        pipeline=test_pipeline))
```

You would need to check if `data/icdar2015` is right. Then you can start training with the command:

```
python tools/train.py configs/textdet/dbnet/dbnet_r18_fpnc_1200e_icdar2015.py --work-
↪dir dbnet
```

You can find full training instructions, explanations and useful training configs in *Training*.

2.5 Testing

Suppose now you have finished the training of DBNet and the latest model has been saved in `dbnet/latest.pth`. You can evaluate its performance on the test set using the `hmean-iou` metric with the following command:

```
python tools/test.py configs/textdet/dbnet/dbnet_r18_fpnc_1200e_icdar2015.py dbnet/
↪latest.pth --eval hmean-iou
```

Evaluating any pretrained model accessible online is also allowed:

```
python tools/test.py configs/textdet/dbnet/dbnet_r18_fpnc_1200e_icdar2015.py https://
↪download.openmmlab.com/mmocr/textdet/dbnet/dbnet_r18_fpnc_sbn_1200e_icdar2015_
↪20210329-ba3ab597.pth --eval hmean-iou
```

More instructions on testing are available in *Testing*.

CHAPTER

THREE

DEMO

We provide an easy-to-use API for the demo and application purpose in ocr.py script.

The API can be called through command line (CL) or by calling it from another python script.

3.1 Example 1: Text Detection

Instruction: Perform detection inference on an image with the TextSnake recognition model, export the result in a json file (default) and save the visualization file.

- CL interface:

```
python mmocr/utils/ocr.py demo/demo_text_det.jpg --output demo/det_out.jpg --det␣
↪TextSnake --recog None --export demo/
```

- Python interface:

```
from mmocr.utils.ocr import MMOCR

# Load models into memory
ocr = MMOCR(det='TextSnake', recog=None)

# Inference
results = ocr.readtext('demo/demo_text_det.jpg', output='demo/det_out.jpg', export=
↪'demo/')
```

3.2 Example 2: Text Recognition

Instruction: Perform batched recognition inference on a folder with hundreds of image with the CRNN_TPS recognition model and save the visualization results in another folder. *Batch size is set to 10 to prevent out of memory CUDA runtime errors.*

- CL interface:

```
python mmocr/utils/ocr.py %INPUT_FOLDER_PATH% --det None --recog CRNN_TPS --batch-
↪mode --single-batch-size 10 --output %OUPUT_FOLDER_PATH%
```

- Python interface:

```
from mmocr.utils.ocr import MMOCR

# Load models into memory
ocr = MMOCR(det=None, recog='CRNN_TPS')

# Inference
results = ocr.readtext(%INPUT_FOLDER_PATH%, output = %OUTPUT_FOLDER_PATH%, batch_
↪mode=True, single_batch_size = 10)
```

3.3 Example 3: Text Detection + Recognition

Instruction: Perform ocr (det + recog) inference on the demo/demo_text_det.jpg image with the PANet_IC15 (default) detection model and SAR (default) recognition model, print the result in the terminal and show the visualization.

- CL interface:

```
python mmocr/utils/ocr.py demo/demo_text_ocr.jpg --print-result --imshow
```

*Note: When calling the script from the command line, the script assumes configs are saved in the `configs/` folder. User can customize the directory by specifying the value of `config_dir`. *

- Python interface:

```
from mmocr.utils.ocr import MMOCR

# Load models into memory
ocr = MMOCR()

# Inference
results = ocr.readtext('demo/demo_text_ocr.jpg', print_result=True, imshow=True)
```

3.4 Example 4: Text Detection + Recognition + Key Information Extraction

Instruction: Perform end-to-end ocr (det + recog) inference first with PS_CTW detection model and SAR recognition model, then run KIE inference with SDMGR model on the ocr result and show the visualization.

- CL interface:

```
python mmocr/utils/ocr.py demo/demo_kie.jpeg  --det PS_CTW --recog SAR --kie SDMGR --
↪print-result --imshow
```

*Note: When calling the script from the command line, the script assumes configs are saved in the `configs/` folder. User can customize the directory by specifying the value of `config_dir`. *

- Python interface:

```
from mmocr.utils.ocr import MMOCR

# Load models into memory
ocr = MMOCR(det='PS_CTW', recog='SAR', kie='SDMGR')
```

(continues on next page)

(continued from previous page)

```
# Inference
results = ocr.readtext('demo/demo_kie.jpeg', print_result=True, imshow=True)
```

3.5 API Arguments

The API has an extensive list of arguments that you can use. The following tables are for the python interface.

MMOCR():

[1]: `kie` is only effective when both text detection and recognition models are specified.

Note: User can use default pretrained models by specifying `det` and/or `recog`, which is equivalent to specifying their corresponding `*_config` and `*_ckpt`. However, manually specifying `*_config` and `*_ckpt` will always override values set by `det` and/or `recog`. Similar rules also apply to `kie`, `kie_config` and `kie_ckpt`.

3.5.1 readtext():

[1]: Make sure that the model is compatible with batch mode.

[2]: Only effective when the script is running in det + recog mode.

All arguments are the same for the cli, all you need to do is add 2 hyphens at the beginning of the argument and replace underscores by hyphens. (*Example:* `det_batch_size` becomes `--det-batch-size`)

For bool type arguments, putting the argument in the command stores it as true. (*Example:* `python mmocr/utils/ocr.py demo/demo_text_det.jpg --batch_mode --print_result` means that `batch_mode` and `print_result` are set to `True`)

3.6 Models

Text detection:

Text recognition:

Key information extraction:

3.7 Additional info

- To perform det + recog inference (end2end ocr), both the `det` and `recog` arguments must be defined.
- To perform only detection set the `recog` argument to `None`.
- To perform only recognition set the `det` argument to `None`.
- `details` argument only works with end2end ocr.
- `det_batch_size` and `recog_batch_size` arguments define the number of images you want to forward to the model at the same time. For maximum speed, set this to the highest number you can. The max batch size is limited by the model complexity and the GPU VRAM size.

If you have any suggestions for new features, feel free to open a thread or even PR :)

CHAPTER

FOUR

TRAINING

4.1 Training on a Single Machine

You can use `tools/train.py` to train a model in a single machine with one or more GPUs.

Here is the full usage of the script:

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

4.2 Training on Multiple Machines

MMOCR implements **distributed** training with `MMDistributedDataParallel`. (Please refer to datasets.md to prepare your datasets)

```
[PORT={PORT}] ./tools/dist_train.sh ${CONFIG_FILE} ${WORK_DIR} ${GPU_NUM} [PY_ARGS]
```

4.3 Training with Slurm

If you run MMOCR on a cluster managed with Slurm, you can use the script `slurm_train.sh`.

```
[GPUS=${GPUS}] [GPUS_PER_NODE=${GPUS_PER_NODE}] [CPUS_PER_TASK=${CPUS_PER_TASK}]␣
↪[SRUN_ARGS=${SRUN_ARGS}] ./tools/slurm_train.sh ${PARTITION} ${JOB_NAME} ${CONFIG_
↪FILE} ${WORK_DIR} [PY_ARGS]
```

Here is an example of using 8 GPUs to train a text detection model on the dev partition.

```
./tools/slurm_train.sh dev psenet-ic15 configs/textdet/psenet/psenet_r50_fpnf_sbn_1x_
↪icdar2015.py /nfs/xxxx/psenet-ic15
```

4.3.1 Running Multiple Training Jobs on a Single Machine

If you are launching multiple training jobs on a single machine with Slurm, you may need to modify the port in configs to avoid communication conflicts.

For example, in `config1.py`,

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

In `config2.py`,

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

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

```
CUDA_VISIBLE_DEVICES=0,1,2,3 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}
```

4.4 Commonly Used Training Configs

Here we list some configs that are frequently used during training for quick reference.

```
total_epochs = 1200
data = dict(
    # Note: User can configure general settings of train, val and test dataloader by␣
↪specifying them here. However, their values can be overrided in dataloader's config.
    samples_per_gpu=8, # Batch size per GPU
    workers_per_gpu=4, # Number of workers to process data for each GPU
    train_dataloader=dict(samples_per_gpu=10, drop_last=True),    # Batch size = 10,␣
↪workers_per_gpu = 4
    val_dataloader=dict(samples_per_gpu=6, workers_per_gpu=1),  # Batch size = 6,␣
↪workers_per_gpu = 1
    test_dataloader=dict(workers_per_gpu=16),  # Batch size = 8, workers_per_gpu = 16
    ...
)
# Evaluation
evaluation = dict(interval=1, by_epoch=True)  # Evaluate the model every epoch
# Saving and Logging
checkpoint_config = dict(interval=1)  # Save a checkpoint every epoch
log_config = dict(
    interval=5,  # Print out the model's performance every 5 iterations
    hooks=[
        dict(type='TextLoggerHook')
    ])
# Optimizer
optimizer = dict(type='SGD', lr=0.02, momentum=0.9, weight_decay=0.0001)  # Supports␣
↪all optimizers in PyTorch and shares the same parameters
optimizer_config = dict(grad_clip=None)  # Parameters for the optimizer hook. See␣
↪https://github.com/open-mmlab/mmcv/blob/master/mmcv/runner/hooks/optimizer.py for␣
↪implementation details
# Learning policy
lr_config = dict(policy='poly', power=0.9, min_lr=1e-7, by_epoch=True)
```

CHAPTER

FIVE

TESTING

We introduce the way to test pretrained models on datasets here.

5.1 Testing with Single GPU

You can use `tools/test.py` to perform single GPU inference. For example, to evaluate DBNet on IC15: (You can download pretrained models from *Model Zoo*):

```
./tools/dist_test.sh configs/textdet/dbnet/dbnet_r18_fpnc_1200e_icdar2015.py dbnet_
↪r18_fpnc_sbn_1200e_icdar2015_20210329-ba3ab597.pth --eval hmean-iou
```

And here is the full usage of the script:

```
python tools/test.py ${CONFIG_FILE} ${CHECKPOINT_FILE} [ARGS]
```

5.2 Testing with Multiple GPUs

MMOCR implements **distributed** testing with `MMDistributedDataParallel`.

You can use the following command to test a dataset with multiple GPUs.

```
[PORT={PORT}] ./tools/dist_test.sh ${CONFIG_FILE} ${CHECKPOINT_FILE} ${GPU_NUM} [PY_
↪ARGS]
```

For example,

```
./tools/dist_test.sh configs/example_config.py work_dirs/example_exp/example_model_
↪20200202.pth 1 --eval hmean-iou
```

5.3 Testing with Slurm

If you run MMOCR on a cluster managed with Slurm, you can use the script `tools/slurm_test.sh`.

```
[GPUS=${GPUS}] [GPUS_PER_NODE=${GPUS_PER_NODE}] [SRUN_ARGS=${SRUN_ARGS}] ./tools/
↪slurm_test.sh ${PARTITION} ${JOB_NAME} ${CONFIG_FILE} ${CHECKPOINT_FILE} [PY_ARGS]
```

Here is an example of using 8 GPUs to test an example model on the 'dev' partition with job name 'test_job'.

```
GPUS=8 ./tools/slurm_test.sh dev test_job configs/example_config.py work_dirs/example_
↪exp/example_model_20200202.pth --eval hmean-iou
```

5.4 Batch Testing

By default, MMOCR tests the model image by image. For faster inference, you may change `data.val_dataloader.samples_per_gpu` and `data.test_dataloader.samples_per_gpu` in the config. For example,

```
data = dict(
    ...
    val_dataloader=dict(samples_per_gpu=16),
    test_dataloader=dict(samples_per_gpu=16),
    ...
)
```

will test the model with 16 images in a batch.

Warning: Batch testing may incur performance decrease of the model due to the different behavior of the data preprocessing pipeline.

CHAPTER

SIX

DEPLOYMENT

We provide deployment tools under `tools/deployment` directory.

6.1 Convert to ONNX (experimental)

We provide a script to convert the model to ONNX format. The converted model could be visualized by tools like Netron. Besides, we also support comparing the output results between Pytorch and ONNX model.

```
python tools/deployment/pytorch2onnx.py
    ${MODEL_CONFIG_PATH} \
    ${MODEL_CKPT_PATH} \
    ${MODEL_TYPE} \
    ${IMAGE_PATH} \
    --output-file ${OUTPUT_FILE} \
    --device-id ${DEVICE_ID} \
    --opset-version ${OPSET_VERSION} \
    --verify \
    --verbose \
    --show \
    --dynamic-export
```

Description of arguments:

Note: This tool is still experimental. For now, some customized operators are not supported, and we only support a subset of detection and recognition algorithms.

6.1.1 List of supported models exportable to ONNX

The table below lists the models that are guaranteed to be exportable to ONNX and runnable in ONNX Runtime.

Notes:

- *All models above are tested with Pytorch==1.8.1 and onnxruntime==1.7.0*
- If you meet any problem with the listed models above, please create an issue and it would be taken care of soon.
- Because this feature is experimental and may change fast, please always try with the latest `mmcv` and `mmocr`.

6.2 Convert ONNX to TensorRT (experimental)

We also provide a script to convert ONNX model to TensorRT format. Besides, we support comparing the output results between ONNX and TensorRT model.

```
python tools/deployment/onnx2tensorrt.py
    ${MODEL_CONFIG_PATH} \
    ${MODEL_TYPE} \
    ${IMAGE_PATH} \
    ${ONNX_FILE} \
    --trt-file ${OUT_TENSORRT} \
    --max-shape INT INT INT INT \
    --min-shape INT INT INT INT \
    --workspace-size INT \
    --fp16 \
    --verify \
    --show \
    --verbose
```

Description of arguments:

Note: This tool is still experimental. For now, some customized operators are not supported, and we only support a subset of detection and recognition algorithms.

6.2.1 List of supported models exportable to TensorRT

The table below lists the models that are guaranteed to be exportable to TensorRT engine and runnable in TensorRT.

Notes:

- *All models above are tested with Pytorch==1.8.1, onnxruntime==1.7.0 and tensorrt==7.2.1.6*
- If you meet any problem with the listed models above, please create an issue and it would be taken care of soon.
- Because this feature is experimental and may change fast, please always try with the latest `mmcv` and `mmocr`.

6.3 Evaluate ONNX and TensorRT Models (experimental)

We provide methods to evaluate TensorRT and ONNX models in `tools/deployment/deploy_test.py`.

6.3.1 Prerequisite

To evaluate ONNX and TensorRT models, ONNX, ONNXRuntime and TensorRT should be installed first. Install `mmcv-full` with ONNXRuntime custom ops and TensorRT plugins follow ONNXRuntime in mmcv and TensorRT plugin in mmcv.

6.3.2 Usage

```
python tools/deploy_test.py \
    ${CONFIG_FILE} \
    ${MODEL_PATH} \
    ${MODEL_TYPE} \
    ${BACKEND} \
    --eval ${METRICS} \
    --device ${DEVICE}
```

6.3.3 Description of all arguments

6.4 Results and Models

Notes:

- TensorRT upsampling operation is a little different from PyTorch. For DBNet and PANet, we suggest replacing upsampling operations with the nearest mode to operations with bilinear mode. Here for PANet, here and here for DBNet. As is shown in the above table, networks with tag * mean the upsampling mode is changed.
- Note that changing upsampling mode reduces less performance compared with using the nearest mode. However, the weights of networks are trained through the nearest mode. To pursue the best performance, using bilinear mode for both training and TensorRT deployment is recommended.
- All ONNX and TensorRT models are evaluated with dynamic shapes on the datasets, and images are preprocessed according to the original config file.
- This tool is still experimental, and we only support a subset of detection and recognition algorithms for now.

CHAPTER

SEVEN

OVERVIEW

- Number of checkpoints: 27
- Number of configs: 24
- Number of papers: 17
  - ALGORITHM: 15
  - BACKBONE: 1
  - PREPROCESSOR: 1

For supported datasets, see datasets overview.

7.1 Key Information Extraction Models

- Number of checkpoints: 2
- Number of configs: 2
- Number of papers: 1
  - [ALGORITHM] Spatial Dual-Modality Graph Reasoning for Key Information Extraction ()

7.2 Named Entity Recognition Models

- Number of checkpoints: 1
- Number of configs: 1
- Number of papers: 1
  - [ALGORITHM] Bert: Pre-Training of Deep Bidirectional Transformers for Language Understanding ()

7.3 Text Detection Models

- Number of checkpoints: 14
- Number of configs: 13
- Number of papers: 7
  - [ALGORITHM] Deep Relational Reasoning Graph Network for Arbitrary Shape Text Detection ()
  - [ALGORITHM] Efficient and Accurate Arbitrary-Shaped Text Detection With Pixel Aggregation Network ()
  - [ALGORITHM] Fourier Contour Embedding for Arbitrary-Shaped Text Detection ()
  - [ALGORITHM] Mask R-CNN ()
  - [ALGORITHM] Real-Time Scene Text Detection With Differentiable Binarization ()
  - [ALGORITHM] Shape Robust Text Detection With Progressive Scale Expansion Network ()
  - [ALGORITHM] Textsnake: A Flexible Representation for Detecting Text of Arbitrary Shapes ()

7.4 Text Recognition Models

- Number of checkpoints: 10
- Number of configs: 8
- Number of papers: 8
  - [ALGORITHM] An End-to-End Trainable Neural Network for Image-Based Sequence Recognition and Its Application to Scene Text Recognition ( )
  - [ALGORITHM] Nrtr: A No-Recurrence Sequence-to-Sequence Model for Scene Text Recognition ()
  - [ALGORITHM] On Recognizing Texts of Arbitrary Shapes With 2d Self-Attention ()
  - [ALGORITHM] Robustscanner: Dynamically Enhancing Positional Clues for Robust Text Recognition ()
  - [ALGORITHM] Segocr Simple Baseline. ()
  - [ALGORITHM] Show, Attend and Read: A Simple and Strong Baseline for Irregular Text Recognition ( )
  - [BACKBONE] Show, Attend and Read: A Simple and Strong Baseline for Irregular Text Recognition ( )
  - [PREPROCESSOR] Robust Scene Text Recognition With Automatic Rectification ()

CHAPTER

EIGHT

TEXT DETECTION MODELS

8.1 Real-time Scene Text Detection with Differentiable Binarization

8.1.1 Introduction

[ALGORITHM]

```
@article{Liao_Wan_Yao_Chen_Bai_2020,
    title={Real-Time Scene Text Detection with Differentiable Binarization},
    journal={Proceedings of the AAAI Conference on Artificial Intelligence},
    author={Liao, Minghui and Wan, Zhaoyi and Yao, Cong and Chen, Kai and Bai, Xiang},
    year={2020},
    pages={11474-11481}}
```

8.1.2 Results and models

ICDAR2015

8.2 DRRG

8.2.1 Introduction

[ALGORITHM]

```
@article{zhang2020drrg,
  title={Deep relational reasoning graph network for arbitrary shape text detection},
  author={Zhang, Shi-Xue and Zhu, Xiaobin and Hou, Jie-Bo and Liu, Chang and Yang,␣
↪Chun and Wang, Hongfa and Yin, Xu-Cheng},
  booktitle={CVPR},
  pages={9699-9708},
  year={2020}
}
```

8.2.2 Results and models

CTW1500

8.3 Fourier Contour Embedding for Arbitrary-Shaped Text Detection

8.3.1 Introduction

[ALGORITHM]

```
@InProceedings{zhu2021fourier,
      title={Fourier Contour Embedding for Arbitrary-Shaped Text Detection},
      author={Yiqin Zhu and Jianyong Chen and Lingyu Liang and Zhanghui Kuang and␣
↪Lianwen Jin and Wayne Zhang},
      year={2021},
      booktitle = {CVPR}
      }
```

8.3.2 Results and models

CTW1500

ICDAR2015

8.4 Mask R-CNN

8.4.1 Introduction

[ALGORITHM]

```
@INPROCEEDINGS{8237584,
  author={K. {He} and G. {Gkioxari} and P. {Dollár} and R. {Girshick}},
  booktitle={2017 IEEE International Conference on Computer Vision (ICCV)},
  title={Mask R-CNN},
  year={2017},
  pages={2980-2988},
  doi={10.1109/ICCV.2017.322}}
```

In tuning parameters, we refer to the baseline method in the following article:

```
@article{pmtd,
  author={Jingchao Liu and Xuebo Liu and Jie Sheng and Ding Liang and Xin Li and␣
↪Qingjie Liu},
  title={Pyramid Mask Text Detector},
  journal={CoRR},
  volume={abs/1903.11800},
  year={2019}
}
```

8.4.2 Results and models

CTW1500

ICDAR2015

ICDAR2017

8.5 Efficient and Accurate Arbitrary-Shaped Text Detection with Pixel Aggregation Network

8.5.1 Introduction

[ALGORITHM]

```
@inproceedings{WangXSZWLYS19,
  author={Wenhai Wang and Enze Xie and Xiaoge Song and Yuhang Zang and Wenjia Wang␣
↪and Tong Lu and Gang Yu and Chunhua Shen},
  title={Efficient and Accurate Arbitrary-Shaped Text Detection With Pixel␣
↪Aggregation Network},
  booktitle={ICCV},
  pages={8439--8448},
  year={2019}
  }
```

8.5.2 Results and models

CTW1500

ICDAR2015

8.6 PSENet

8.6.1 Introduction

[ALGORITHM]

```
@inproceedings{wang2019shape,
  title={Shape robust text detection with progressive scale expansion network},
  author={Wang, Wenhai and Xie, Enze and Li, Xiang and Hou, Wenbo and Lu, Tong and Yu,
↪ Gang and Shao, Shuai},
  booktitle={Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern␣
↪Recognition},
  pages={9336--9345},
  year={2019}
}
```

8.6.2 Results and models

CTW1500

ICDAR2015

8.7 Textsnake

8.7.1 Introduction

[ALGORITHM]

```
@article{long2018textsnake,
  title={TextSnake: A Flexible Representation for Detecting Text of Arbitrary Shapes},
  author={Long, Shangbang and Ruan, Jiaqiang and Zhang, Wenjie and He, Xin and Wu,␣
↪Wenhao and Yao, Cong},
  booktitle={ECCV},
  pages={20-36},
  year={2018}
}
```

8.7.2 Results and models

CTW1500

CHAPTER

NINE

TEXT RECOGNITION MODELS

9.1 An end-to-end trainable neural network for image-based sequence recognition and its application to scene text recognition

9.1.1 Introduction

[ALGORITHM]

```
@article{shi2016end,
  title={An end-to-end trainable neural network for image-based sequence recognition
↪and its application to scene text recognition},
  author={Shi, Baoguang and Bai, Xiang and Yao, Cong},
  journal={IEEE transactions on pattern analysis and machine intelligence},
  year={2016}
}
```

9.1.2 Results and Models

Train Dataset

Test Dataset

9.1.3 Results and models

9.2 NRTR

9.2.1 Introduction

[ALGORITHM]

```
@inproceedings{sheng2019nrtr,
  title={NRTR: A no-recurrence sequence-to-sequence model for scene text recognition},
  author={Sheng, Fenfen and Chen, Zhineng and Xu, Bo},
  booktitle={2019 International Conference on Document Analysis and Recognition
↪(ICDAR)},
  pages={781--786},
  year={2019},
  organization={IEEE}
}
```

[BACKBONE]

```
@inproceedings{li2019show,
  title={Show, attend and read: A simple and strong baseline for irregular text␣
↪recognition},
  author={Li, Hui and Wang, Peng and Shen, Chunhua and Zhang, Guyu},
  booktitle={Proceedings of the AAAI Conference on Artificial Intelligence},
  volume={33},
  number={01},
  pages={8610--8617},
  year={2019}
}
```

9.2.2 Dataset

Train Dataset

Test Dataset

9.2.3 Results and Models

Notes:

- `R31-1/16-1/8` means the height of feature from backbone is 1/16 of input image, where 1/8 for width.
- `R31-1/8-1/4` means the height of feature from backbone is 1/8 of input image, where 1/4 for width.

9.3 RobustScanner: Dynamically Enhancing Positional Clues for Robust Text Recognition

9.3.1 Introduction

[ALGORITHM]

```
@inproceedings{yue2020robustscanner,
  title={RobustScanner: Dynamically Enhancing Positional Clues for Robust Text␣
↪Recognition},
  author={Yue, Xiaoyu and Kuang, Zhanghui and Lin, Chenhao and Sun, Hongbin and Zhang,
↪ Wayne},
  booktitle={European Conference on Computer Vision},
  year={2020}
}
```

9.3.2 Dataset

Train Dataset

Test Dataset

9.3.3 Results and Models

9.3.4 References

[1] Li, Hui and Wang, Peng and Shen, Chunhua and Zhang, Guyu. Show, attend and read: A simple and strong baseline for irregular text recognition. In AAAI 2019.

9.4 Show, Attend and Read: A Simple and Strong Baseline for Irregular Text Recognition

9.4.1 Introduction

[ALGORITHM]

```
@inproceedings{li2019show,
  title={Show, attend and read: A simple and strong baseline for irregular text␣
↪recognition},
  author={Li, Hui and Wang, Peng and Shen, Chunhua and Zhang, Guyu},
  booktitle={Proceedings of the AAAI Conference on Artificial Intelligence},
  volume={33},
  number={01},
  pages={8610--8617},
  year={2019}
}
```

9.4.2 Dataset

Train Dataset

Test Dataset

9.4.3 Results and Models

9.4.4 Chinese Dataset

9.4.5 Results and Models

Notes:

- `R31-1/8-1/4` means the height of feature from backbone is 1/8 of input image, where 1/4 for width.
- We did not use beam search during decoding.
- We implemented two kinds of decoder. Namely, `ParallelSARDecoder` and `SequentialSARDecoder`.
  - `ParallelSARDecoder`: Parallel decoding during training with `LSTM` layer. It would be faster.

  - `SequentialSARDecoder`: Sequential Decoding during training with `LSTMCell`. It would be easier to understand.

- For train dataset.
  - We did not construct distinct data groups (20 groups in *[1]*) to train the model group-by-group since it would render model training too complicated.
  - Instead, we randomly selected `2.4m` patches from `Syn90k`, `2.4m` from `SynthText` and `1.2m` from `SynthAdd`, and grouped all data together. See config for details.
- We used 48 GPUs with `total_batch_size = 64 * 48` in the experiment above to speedup training, while keeping the `initial lr = 1e-3` unchanged.

9.4.6 References

[1] Li, Hui and Wang, Peng and Shen, Chunhua and Zhang, Guyu. Show, attend and read: A simple and strong baseline for irregular text recognition. In AAAI 2019.

9.5 SATRN

9.5.1 Introduction

[ALGORITHM]

```
@article{junyeop2019recognizing,
  title={On Recognizing Texts of Arbitrary Shapes with 2D Self-Attention},
  author={Junyeop Lee, Sungrae Park, Jeonghun Baek, Seong Joon Oh, Seonghyeon Kim,␣
↪Hwalsuk Lee},
  year={2019}
}
```

9.5.2 Dataset

Train Dataset

Test Dataset

9.5.3 Results and Models

9.6 SegOCR Simple Baseline.

9.6.1 Introduction

[ALGORITHM]

```
@unpublished{key,
  title={SegOCR Simple Baseline.},
  author={},
  note={Unpublished Manuscript},
```

(continues on next page)

(continued from previous page)

```
  year={2021}
}
```

9.6.2 Dataset

Train Dataset

Test Dataset

9.6.3 Results and Models

Notes:

- `R31-1/16` means the size (both height and width) of feature from backbone is 1/16 of input image.
- `1x` means the size (both height and width) of feature from head is the same with input image.

9.7 CRNN with TPS based STN

9.7.1 Introduction

[ALGORITHM]

```
@article{shi2016end,
  title={An end-to-end trainable neural network for image-based sequence recognition␣
↪and its application to scene text recognition},
  author={Shi, Baoguang and Bai, Xiang and Yao, Cong},
  journal={IEEE transactions on pattern analysis and machine intelligence},
  year={2016}
}
```

[PREPROCESSOR]

```
@article{shi2016robust,
  title={Robust Scene Text Recognition with Automatic Rectification},
  author={Shi, Baoguang and Wang, Xinggang and Lyu, Pengyuan and Yao,
  Cong and Bai, Xiang},
  year={2016}
}
```

9.7.2 Results and Models

Train Dataset

Test Dataset

9.7.3 Results and models

CHAPTER

TEN

KEY INFORMATION EXTRACTION MODELS

10.1 Spatial Dual-Modality Graph Reasoning for Key Information Extraction

10.1.1 Introduction

[ALGORITHM]

```
@misc{sun2021spatial,
      title={Spatial Dual-Modality Graph Reasoning for Key Information Extraction},
      author={Hongbin Sun and Zhanghui Kuang and Xiaoyu Yue and Chenhao Lin and Wayne␣
↪Zhang},
      year={2021},
      eprint={2103.14470},
      archivePrefix={arXiv},
      primaryClass={cs.CV}
}
```

10.1.2 Results and models

WildReceipt

CHAPTER

ELEVEN

NAMED ENTITY RECOGNITION MODELS

11.1 Chinese Named Entity Recognition using BERT + Softmax

11.1.1 Introduction

[ALGORITHM]

```
@article{devlin2018bert,
  title={Bert: Pre-training of deep bidirectional transformers for language
↪understanding},
  author={Devlin, Jacob and Chang, Ming-Wei and Lee, Kenton and Toutanova, Kristina},
  journal={arXiv preprint arXiv:1810.04805},
  year={2018}
}
```

11.1.2 Dataset

Train Dataset

Test Dataset

11.1.3 Results and models

CHAPTER

TWELVE

TEXT DETECTION

12.1 Overview

The structure of the text detection dataset directory is organized as follows.

```
├── ctw1500
│   ├── annotations
│   ├── imgs
│   ├── instances_test.json
│   └── instances_training.json
├── icdar2015
│   ├── imgs
│   ├── instances_test.json
│   └── instances_training.json
├── icdar2017
│   ├── imgs
│   ├── instances_training.json
│   └── instances_val.json
├── synthtext
│   ├── imgs
│   └── instances_training.lmdb
│       ├── data.mdb
│       └── lock.mdb
├── textocr
│   ├── train
│   ├── instances_training.json
│   └── instances_val.json
├── totaltext
│   ├── imgs
│   ├── instances_test.json
│   └── instances_training.json
```

12.2 Important Note

Note: For users who want to train models on CTW1500, ICDAR 2015/2017, and Totaltext dataset, there might be some images containing orientation info in EXIF data. The default OpenCV backend used in MMCV would read them and apply the rotation on the images. However, their gold annotations are made on the raw pixels, and such inconsistency results in false examples in the training set. Therefore, users should use `dict(type='LoadImageFromFile', color_type='color_ignore_orientation')` in pipelines to change MMCV's default loading behaviour. (see DBNet's config for example)

12.3 Preparation Steps

12.3.1 ICDAR 2015

- Step0: Read *Important Note*
- Step1: Download `ch4_training_images.zip`, `ch4_test_images.zip`, `ch4_training_localization_transcription_gt.zip`, `Challenge4_Test_Task1_GT.zip` from homepage
- Step2:

```
mkdir icdar2015 && cd icdar2015
mkdir imgs && mkdir annotations
# For images,
mv ch4_training_images imgs/training
mv ch4_test_images imgs/test
# For annotations,
mv ch4_training_localization_transcription_gt annotations/training
mv Challenge4_Test_Task1_GT annotations/test
```

- Step3: Download instances_training.json and instances_test.json and move them to `icdar2015`
- Or, generate `instances_training.json` and `instances_test.json` with following command:

```
python tools/data/textdet/icdar_converter.py /path/to/icdar2015 -o /path/to/icdar2015␣
↪-d icdar2015 --split-list training test
```

12.3.2 ICDAR 2017

- Follow similar steps as *ICDAR 2015*.

12.3.3 CTW1500

- Step0: Read *Important Note*
- Step1: Download `train_images.zip`, `test_images.zip`, `train_labels.zip`, `test_labels.zip` from github

```
mkdir ctw1500 && cd ctw1500
mkdir imgs && mkdir annotations

# For annotations
cd annotations
wget -O train_labels.zip https://universityofadelaide.box.com/shared/static/
↪jikuazluzyj4lq6umzei7m2ppmt3afyw.zip
wget -O test_labels.zip https://cloudstor.aarnet.edu.au/plus/s/uoeFl0pCN9BOCN5/
↪download
unzip train_labels.zip && mv ctw1500_train_labels training
unzip test_labels.zip -d test
cd ..
# For images
cd imgs
wget -O train_images.zip https://universityofadelaide.box.com/shared/static/
↪py5uwlfyyytbb2pxzq9czvu6fuqbjdh8.zip
```

(continues on next page)

(continued from previous page)

```
wget -O test_images.zip https://universityofadelaide.box.com/shared/static/
↪t4w48ofnqkdw7jyc4t11nsukoeqk9c3d.zip
unzip train_images.zip && mv train_images training
unzip test_images.zip && mv test_images test
```

- Step2: Generate `instances_training.json` and `instances_test.json` with following command:

```
python tools/data/textdet/ctw1500_converter.py /path/to/ctw1500 -o /path/to/ctw1500 --
↪split-list training test
```

12.3.4 SynthText

- Download data.mdb and lock.mdb to `synthtext/instances_training.lmdb/`.

12.3.5 TextOCR

- Step1: Download train_val_images.zip, TextOCR_0.1_train.json and TextOCR_0.1_val.json to `textocr/`.

```
mkdir textocr && cd textocr

# Download TextOCR dataset
wget https://dl.fbaipublicfiles.com/textvqa/images/train_val_images.zip
wget https://dl.fbaipublicfiles.com/textvqa/data/textocr/TextOCR_0.1_train.json
wget https://dl.fbaipublicfiles.com/textvqa/data/textocr/TextOCR_0.1_val.json

# For images
unzip -q train_val_images.zip
mv train_images train
```

- Step2: Generate `instances_training.json` and `instances_val.json` with the following command:

```
python tools/data/textdet/textocr_converter.py /path/to/textocr
```

12.3.6 Totaltext

- Step0: Read *Important Note*
- Step1: Download `totaltext.zip` from github dataset and `groundtruth_text.zip` from github Groundtruth (Our totaltext_converter.py supports groundtruth with both .mat and .txt format).

```
mkdir totaltext && cd totaltext
mkdir imgs && mkdir annotations

# For images
# in ./totaltext
unzip totaltext.zip
mv Images/Train imgs/training
mv Images/Test imgs/test

# For annotations
unzip groundtruth_text.zip
```

(continues on next page)

(continued from previous page)

```
cd Groundtruth
mv Polygon/Train ../annotations/training
mv Polygon/Test ../annotations/test
```

- Step2: Generate `instances_training.json` and `instances_test.json` with the following command:

```
python tools/data/textdet/totaltext_converter.py /path/to/totaltext -o /path/to/
↪totaltext --split-list training test
```

CHAPTER

THIRTEEN

TEXT RECOGNITION

13.1 Overview

The structure of the text recognition dataset directory is organized as follows.

```
├── mixture
│   ├── coco_text
│   │   ├── train_label.txt
│   │   ├── train_words
│   ├── icdar_2011
│   │   ├── training_label.txt
│   │   ├── Challenge1_Training_Task3_Images_GT
│   ├── icdar_2013
│   │   ├── train_label.txt
│   │   ├── test_label_1015.txt
│   │   ├── test_label_1095.txt
│   │   ├── Challenge2_Training_Task3_Images_GT
│   │   ├── Challenge2_Test_Task3_Images
│   ├── icdar_2015
│   │   ├── train_label.txt
│   │   ├── test_label.txt
│   │   ├── ch4_training_word_images_gt
│   │   ├── ch4_test_word_images_gt
│   ├── III5K
│   │   ├── train_label.txt
│   │   ├── test_label.txt
│   │   ├── train
│   │   ├── test
│   ├── ct80
│   │   ├── test_label.txt
│   │   ├── image
│   ├── svt
│   │   ├── test_label.txt
│   │   ├── image
│   ├── svtp
│   │   ├── test_label.txt
│   │   ├── image
│   ├── Syn90k
│   │   ├── shuffle_labels.txt
│   │   ├── label.txt
│   │   ├── label.lmdb
│   │   ├── mnt
│   ├── SynthText
│   │   ├── shuffle_labels.txt
```

(continues on next page)

(continued from previous page)

```
|   |   ├── instances_train.txt
|   |   ├── label.txt
|   |   ├── label.lmdb
|   |   └── synthtext
|   ├── SynthAdd
|   |   ├── label.txt
|   |   ├── label.lmdb
|   |   └── SynthText_Add
|   ├── TextOCR
|   |   ├── image
|   |   ├── train_label.txt
|   |   └── val_label.txt
|   └── Totaltext
|   |   ├── imgs
|   |   ├── annotations
|   |   ├── train_label.txt
|   |   └── test_label.txt
```

(*) Since the official homepage is unavailable now, we provide an alternative for quick reference. However, we do not guarantee the correctness of the dataset.

13.2 Preparation Steps

13.2.1 ICDAR 2013

- Step1: Download `Challenge2_Test_Task3_Images.zip` and `Challenge2_Training_Task3_Images_GT.zip` from homepage
- Step2: Download test_label_1015.txt and train_label.txt
- For `icdar_2015`:
- Step1: Download `ch4_training_word_images_gt.zip` and `ch4_test_word_images_gt.zip` from homepage
- Step2: Download train_label.txt and test_label.txt

13.2.2 IIIT5K

- Step1: Download `IIIT5K-Word_V3.0.tar.gz` from homepage
- Step2: Download train_label.txt and test_label.txt

13.2.3 svt

- Step1: Download `svt.zip` form homepage
- Step2: Download test_label.txt
- Step3:

```
python tools/data/textrecog/svt_converter.py <download_svt_dir_path>
```

13.2.4 ct80

- Step1: Download test_label.txt

13.2.5 svtp

- Step1: Download test_label.txt

13.2.6 coco_text

- Step1: Download from homepage
- Step2: Download train_label.txt

13.2.7 MJSynth (Syn90k)

- Step1: Download `mjsynth.tar.gz` from homepage
- Step2: Download label.txt (8,919,273 annotations) and shuffle_labels.txt (2,400,000 randomly sampled annotations). **Please make sure you're using the right annotation to train the model by checking its dataset specs in Model Zoo.**
- Step3:

```
mkdir Syn90k && cd Syn90k

mv /path/to/mjsynth.tar.gz .

tar -xzf mjsynth.tar.gz

mv /path/to/shuffle_labels.txt .
mv /path/to/label.txt .

# create soft link
cd /path/to/mmocr/data/mixture

ln -s /path/to/Syn90k Syn90k
```

13.2.8 SynthText (Synth800k)

- Step1: Download `SynthText.zip` from homepage
- Step2: Download label.txt (7,266,686 annotations) and shuffle_labels.txt (2,400,000 randomly sampled annotations). **Please make sure you're using the right annotation to train the model by checking its dataset specs in Model Zoo.**
- Step3:

```
mkdir SynthText && cd SynthText
mv /path/to/SynthText.zip .
unzip SynthText.zip
mv SynthText synthtext

mv /path/to/shuffle_labels.txt .
```

(continues on next page)

(continued from previous page)

```
mv /path/to/label.txt .

# create soft link
cd /path/to/mmocr/data/mixture
ln -s /path/to/SynthText SynthText
```

- Step4: Generate cropped images and labels:

```
cd /path/to/mmocr

python tools/data/textrecog/synthtext_converter.py data/mixture/SynthText/gt.mat data/
↪mixture/SynthText/ data/mixture/SynthText/synthtext/SynthText_patch_horizontal --n_
↪proc 8
```

13.2.9 SynthAdd

- Step1: Download `SynthText_Add.zip` from SynthAdd (code:627x))
- Step2: Download label.txt
- Step3:

```
mkdir SynthAdd && cd SynthAdd

mv /path/to/SynthText_Add.zip .

unzip SynthText_Add.zip

mv /path/to/label.txt .

# create soft link
cd /path/to/mmocr/data/mixture

ln -s /path/to/SynthAdd SynthAdd
```

Note: To convert label file with `txt` format to `lmdb` format,

```
python tools/data/utils/txt2lmdb.py -i <txt_label_path> -o <lmdb_label_path>
```

For example,

```
python tools/data/utils/txt2lmdb.py -i data/mixture/Syn90k/label.txt -o data/mixture/
↪Syn90k/label.lmdb
```

13.2.10 TextOCR

- Step1: Download train_val_images.zip, TextOCR_0.1_train.json and TextOCR_0.1_val.json to `textocr/`.

```
mkdir textocr && cd textocr

# Download TextOCR dataset
wget https://dl.fbaipublicfiles.com/textvqa/images/train_val_images.zip
wget https://dl.fbaipublicfiles.com/textvqa/data/textocr/TextOCR_0.1_train.json
wget https://dl.fbaipublicfiles.com/textvqa/data/textocr/TextOCR_0.1_val.json
```

(continues on next page)

(continued from previous page)

```
# For images
unzip -q train_val_images.zip
mv train_images train
```

- Step2: Generate `train_label.txt`, `val_label.txt` and crop images using 4 processes with the following command:

```
python tools/data/textrecog/textocr_converter.py /path/to/textocr 4
```

13.2.11 Totaltext

- Step1: Download `totaltext.zip` from github dataset and `groundtruth_text.zip` from github Groundtruth (Our totaltext_converter.py supports groundtruth with both .mat and .txt format).

```
mkdir totaltext && cd totaltext
mkdir imgs && mkdir annotations

# For images
# in ./totaltext
unzip totaltext.zip
mv Images/Train imgs/training
mv Images/Test imgs/test

# For annotations
unzip groundtruth_text.zip
cd Groundtruth
mv Polygon/Train ../annotations/training
mv Polygon/Test ../annotations/test
```

- Step2: Generate cropped images, `train_label.txt` and `test_label.txt` with the following command (the cropped images will be saved to `data/totaltext/dst_imgs/`):

```
python tools/data/textrecog/totaltext_converter.py /path/to/totaltext -o /path/to/
↪totaltext --split-list training test
```

CHAPTER

FOURTEEN

KEY INFORMATION EXTRACTION

14.1 Overview

The structure of the key information extraction dataset directory is organized as follows.

```
└── wildreceipt
    ├── class_list.txt
    ├── dict.txt
    ├── image_files
    ├── test.txt
    └── train.txt
```

14.2 Preparation Steps

14.2.1 WildReceipt

- Just download and extract wildreceipt.tar.

CHAPTER

FIFTEEN

NAMED ENTITY RECOGNITION

15.1 Overview

The structure of the named entity recognition dataset directory is organized as follows.

```
└── cluener2020
    ├── cluener_predict.json
    ├── dev.json
    ├── README.md
    ├── test.json
    ├── train.json
    └── vocab.txt
```

15.2 Preparation Steps

15.2.1 CLUENER2020

- Download and extract cluener_public.zip to `cluener2020/`
- Download vocab.txt and move `vocab.txt` to `cluener2020/`

CHAPTER

SIXTEEN

DATASET TYPES

16.1 General Introduction

To support the tasks of text detection, text recognition and key information extraction, we have designed some new types of dataset which consist of **loader** and **parser** to load and parse different types of annotation files.

- **loader**: Load the annotation file. There are two types of loader, `HardDiskLoader` and `LmdbLoader`
  - `HardDiskLoader`: Load `txt` format annotation file from hard disk to memory.
  - `LmdbLoader`: Load `lmdb` format annotation file with lmdb backend, which is very useful for **extremely large** annotation files to avoid out-of-memory problem when ten or more GPUs are used, since each GPU will start multiple processes to load annotation file to memory.
- **parser**: Parse the annotation file line-by-line and return with `dict` format. There are two types of parser, `LineStrParser` and `LineJsonParser`.
  - `LineStrParser`: Parse one line in ann file while treating it as a string and separating it to several parts by a `separator`. It can be used on tasks with simple annotation files such as text recognition where each line of the annotation files contains the `filename` and `label` attribute only.
  - `LineJsonParser`: Parse one line in ann file while treating it as a json-string and using `json.loads` to convert it to `dict`. It can be used on tasks with complex annotation files such as text detection where each line of the annotation files contains multiple attributes (e.g. `filename`, `height`, `width`, `box`, `segmentation`, `iscrowd`, `category_id`, etc.).

Here we show some examples of using different combination of `loader` and `parser`.

16.2 General Task

16.2.1 UniformConcatDataset

`UniformConcatDataset` is a dataset wrapper which allows users to apply a universal pipeline on multiple datasets without specifying the pipeline for each of them.

For example, to apply `train_pipeline` on both `train1` and `train2`,

```
data = dict(
    ...
    train=dict(
        type='UniformConcatDataset',
        datasets=[train1, train2],
        pipeline=train_pipeline))
```

16.3 Text Detection Task

16.3.1 TextDetDataset

Dataset with annotation file in line-json txt format

```
dataset_type = 'TextDetDataset'
img_prefix = 'tests/data/toy_dataset/imgs'
test_anno_file = 'tests/data/toy_dataset/instances_test.txt'
test = dict(
    type=dataset_type,
    img_prefix=img_prefix,
    ann_file=test_anno_file,
    loader=dict(
        type='HardDiskLoader',
        repeat=4,
        parser=dict(
            type='LineJsonParser',
            keys=['file_name', 'height', 'width', 'annotations'])),
    pipeline=test_pipeline,
    test_mode=True)
```

The results are generated in the same way as the segmentation-based text recognition task above. You can check the content of the annotation file in `tests/data/toy_dataset/instances_test.txt`. The combination of `HardDiskLoader` and `LineJsonParser` will return a dict for each file by calling `__getitem__`:

```
{"file_name": "test/img_10.jpg", "height": 720, "width": 1280, "annotations": [{
↪"iscrowd": 1, "category_id": 1, "bbox": [260.0, 138.0, 24.0, 20.0], "segmentation":␣
↪[[261, 138, 284, 140, 279, 158, 260, 158]]}, {"iscrowd": 0, "category_id": 1, "bbox
↪": [288.0, 138.0, 129.0, 23.0], "segmentation": [[288, 138, 417, 140, 416, 161, 290,
↪ 157]]}, {"iscrowd": 0, "category_id": 1, "bbox": [743.0, 145.0, 37.0, 18.0],
↪"segmentation": [[743, 145, 779, 146, 780, 163, 746, 163]]}, {"iscrowd": 0,
↪"category_id": 1, "bbox": [783.0, 129.0, 50.0, 26.0], "segmentation": [[783, 129,␣
↪831, 132, 833, 155, 785, 153]]}, {"iscrowd": 1, "category_id": 1, "bbox": [831.0,␣
↪133.0, 43.0, 23.0], "segmentation": [[831, 133, 870, 135, 874, 156, 835, 155]]}, {
↪"iscrowd": 1, "category_id": 1, "bbox": [159.0, 204.0, 72.0, 15.0], "segmentation":␣
↪[[159, 205, 230, 204, 231, 218, 159, 219]]}, {"iscrowd": 1, "category_id": 1, "bbox
↪": [785.0, 158.0, 75.0, 21.0], "segmentation": [[785, 158, 856, 158, 860, 178, 787,␣
↪179]]}, {"iscrowd": 1, "category_id": 1, "bbox": [1011.0, 157.0, 68.0, 16.0],
↪"segmentation": [[1011, 157, 1079, 160, 1076, 173, 1011, 170]]}]}
```

16.3.2 IcdarDataset

Dataset with annotation file in coco-like json format

For text detection, you can also use an annotation file in a COCO format that is defined in MMDetection:

```
dataset_type = 'IcdarDataset'
prefix = 'tests/data/toy_dataset/'
test=dict(
        type=dataset_type,
        ann_file=prefix + 'instances_test.json',
        img_prefix=prefix + 'imgs',
        pipeline=test_pipeline)
```

You can check the content of the annotation file in `tests/data/toy_dataset/instances_test.json`.

Note: Icdar 2015/2017 and ctw1500 annotations need to be converted into the COCO format following the steps in datasets.md.

16.4 Text Recognition Task

16.4.1 OCRDataset

Dataset for encoder-decoder based recognizer

```
dataset_type = 'OCRDataset'
img_prefix = 'tests/data/ocr_toy_dataset/imgs'
train_anno_file = 'tests/data/ocr_toy_dataset/label.txt'
train = dict(
    type=dataset_type,
    img_prefix=img_prefix,
    ann_file=train_anno_file,
    loader=dict(
        type='HardDiskLoader',
        repeat=10,
        parser=dict(
            type='LineStrParser',
            keys=['filename', 'text'],
            keys_idx=[0, 1],
            separator=' ')),
    pipeline=train_pipeline,
    test_mode=False)
```

You can check the content of the annotation file in `tests/data/ocr_toy_dataset/label.txt`. The combination of `HardDiskLoader` and `LineStrParser` will return a dict for each file by calling `__getitem__`: `{'filename': '1223731.jpg', 'text': 'GRAND'}`.

Optional Arguments:

- `repeat`: The number of repeated lines in the annotation files. For example, if there are `10` lines in the annotation file, setting `repeat=10` will generate a corresponding annotation file with size `100`.

If the annotation file is extremely large, you can convert it from txt format to lmdb format with the following command:

```
python tools/data_converter/txt2lmdb.py -i ann_file.txt -o ann_file.lmdb
```

After that, you can use `LmdbLoader` in dataset like below.

```
img_prefix = 'tests/data/ocr_toy_dataset/imgs'
train_anno_file = 'tests/data/ocr_toy_dataset/label.lmdb'
train = dict(
    type=dataset_type,
    img_prefix=img_prefix,
    ann_file=train_anno_file,
    loader=dict(
        type='LmdbLoader',
        repeat=10,
        parser=dict(
            type='LineStrParser',
            keys=['filename', 'text'],
            keys_idx=[0, 1],
            separator=' ')),
```

(continues on next page)

(continued from previous page)

```
    pipeline=train_pipeline,
    test_mode=False)
```

16.4.2 OCRSegDataset

Dataset for segmentation-based recognizer

```
prefix = 'tests/data/ocr_char_ann_toy_dataset/'
train = dict(
    type='OCRSegDataset',
    img_prefix=prefix + 'imgs',
    ann_file=prefix + 'instances_train.txt',
    loader=dict(
        type='HardDiskLoader',
        repeat=10,
        parser=dict(
            type='LineJsonParser',
            keys=['file_name', 'annotations', 'text'])),
    pipeline=train_pipeline,
    test_mode=True)
```

You can check the content of the annotation file in `tests/data/ocr_char_ann_toy_dataset/instances_train.txt`. The combination of `HardDiskLoader` and `LineJsonParser` will return a dict for each file by calling `__getitem__` each time:

```
{"file_name": "resort_88_101_1.png", "annotations": [{"char_text": "F", "char_box":␣
↪[11.0, 0.0, 22.0, 0.0, 12.0, 12.0, 0.0, 12.0]}, {"char_text": "r", "char_box": [23.
↪0, 2.0, 31.0, 1.0, 24.0, 11.0, 16.0, 11.0]}, {"char_text": "o", "char_box": [33.0,␣
↪2.0, 43.0, 2.0, 36.0, 12.0, 25.0, 12.0]}, {"char_text": "m", "char_box": [46.0, 2.0,
↪ 61.0, 2.0, 53.0, 12.0, 39.0, 12.0]}, {"char_text": ":", "char_box": [61.0, 2.0, 69.
↪0, 2.0, 63.0, 12.0, 55.0, 12.0]}], "text": "From:"}
```

CHAPTER

SEVENTEEN

USEFUL TOOLS

We provide some useful tools under `mmocr/tools` directory.

17.1 Publish a Model

Before you upload a model to AWS, you may want to (1) convert the 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. These functionalities could be achieved by `tools/publish_model.py`.

```
python tools/publish_model.py ${INPUT_FILENAME} ${OUTPUT_FILENAME}
```

For example,

```
python tools/publish_model.py work_dirs/psenet/latest.pth psenet_r50_fpnf_sbn_1x_
↪20190801.pth
```

The final output filename will be `psenet_r50_fpnf_sbn_1x_20190801-{hash id}.pth`.

17.2 Convert txt annotation to lmdb format

Sometimes, loading a large txt annotation file with multiple workers can cause OOM (out of memory) error. You can convert the file into lmdb format using `tools/data/utils/txt2lmdb.py` and use LmdbLoader in your config to avoid this issue.

```
python tools/data/utils/txt2lmdb.py -i <txt_label_path> -o <lmdb_label_path>
```

For example,

```
python tools/data/utils/txt2lmdb.py -i data/mixture/Syn90k/label.txt -o data/mixture/
↪Syn90k/label.lmdb
```

CHAPTER

EIGHTEEN

CHANGELOG

18.1 v0.3.0 (25/8/2021)

18.1.1 Highlights

1. We add a new text recognition model – SATRN! Its pretrained checkpoint achieves the best performance over other provided text recognition models. A lighter version of SATRN is also released which can obtain ~98% of the performance of the original model with only 45 MB in size. (@2793145003) #405
2. Improve the demo script, `ocr.py`, which supports applying end-to-end text detection, text recognition and key information extraction models on images with easy-to-use commands. Users can find its full documentation in the demo section. (@samayala22, @manjrekarom) #371, #386, #400, #374, #428
3. Our documentation is reorganized into a clearer structure. More useful contents are on the way! #409, #454
4. The requirement of `Polygon3` is removed since this project is no longer maintained or distributed. We unified all its references to equivalent substitutions in `shapely` instead. #448

18.1.2 Breaking Changes & Migration Guide

1. Upgrade version requirement of MMDetection to 2.14.0 to avoid bugs #382
2. MMOCR now has its own model and layer registries inherited from MMDetection's or MMCV's counterparts. (#436) The modified hierarchical structure of the model registries are now organized as follows.

```
mmcv.MODELS -> mmdet.BACKBONES -> BACKBONES
mmcv.MODELS -> mmdet.NECKS -> NECKS
mmcv.MODELS -> mmdet.ROI_EXTRACTORS -> ROI_EXTRACTORS
mmcv.MODELS -> mmdet.HEADS -> HEADS
mmcv.MODELS -> mmdet.LOSSES -> LOSSES
mmcv.MODELS -> mmdet.DETECTORS -> DETECTORS
mmcv.ACTIVATION_LAYERS -> ACTIVATION_LAYERS
mmcv.UPSAMPLE_LAYERS -> UPSAMPLE_LAYERS
```

To migrate your old implementation to our new backend, you need to change the import path of any registries and their corresponding builder functions (including `build_detectors`) from `mmdet.models.builder` to `mmocr.models.builder`. If you have referred to any model or layer of MMDetection or MMCV in your model config, you need to add `mmdet.` or `mmcv.` prefix to its name to inform the model builder of the right namespace to work on.

Interested users may check out MMCV's tutorial on Registry for in-depth explanations on its mechanism.

18.1.3 New Features

- Automatically replace SyncBN with BN for inference #420, #453
- Support batch inference for CRNN and SegOCR #407
- Support exporting documentation in pdf or epub format #406
- Support `persistent_workers` option in data loader #459

18.1.4 Bug Fixes

- Remove depreciated key in kie_test_imgs.py #381
- Fix dimension mismatch in batch testing/inference of DBNet #383
- Fix the problem of dice loss which stays at 1 with an empty target given #408
- Fix a wrong link in ocr.py (@naarkhoo) #417
- Fix undesired assignment to "pretrained" in test.py #418
- Fix a problem in polygon generation of DBNet #421, #443
- Skip invalid annotations in totaltext_converter #438
- Add zero division handler in poly utils, remove Polygon3 #448

18.1.5 Improvements

- Replace lanms-proper with lanms-neo to support installation on Windows (with special thanks to @gen-ko who has re-distributed this package!)
- Support MIM #394
- Add tests for PyTorch 1.9 in CI #401
- Enables fullscreen layout in readthedocs #413
- General documentation enhancement #395
- Update version checker #427
- Add copyright info #439
- Update citation information #440

18.1.6 Contributors

We thank @2793145003, @samayala22, @manjrekarom, @naarkhoo, @gen-ko, @duanjiaqi, @gaotongxiao, @cuhk-hbsun, @innerlee, @wdsd641417025 for their contribution to this release!

18.2 v0.2.1 (20/7/2021)

18.2.1 Highlights

1. Upgrade to use MMCV-full **>= 1.3.8** and MMDetection **>= 2.13.0** for latest features
2. Add ONNX and TensorRT export tool, supporting the deployment of DBNet, PSENet, PANet and CRNN (experimental) #278, #291, #300, #328
3. Unified parameter initialization method which uses init_cfg in config files #365

18.2.2 New Features

- Support TextOCR dataset #293
- Support Total-Text dataset #266, #273, #357
- Support grouping text detection box into lines #290, #304
- Add benchmark_processing script that benchmarks data loading process #261
- Add SynthText preprocessor for text recognition models #351, #361
- Support batch inference during testing #310
- Add user-friendly OCR inference script #366

18.2.3 Bug Fixes

- Fix improper class ignorance in SDMGR Loss #221
- Fix potential numerical zero division error in DRRG #224
- Fix installing requirements with pip and mim #242
- Fix dynamic input error of DBNet #269
- Fix space parsing error in LineStrParser #285
- Fix textsnake decode error #264
- Correct isort setup #288
- Fix a bug in SDMGR config #316
- Fix kie_test_img for KIE nonvisual #319
- Fix metafiles #342
- Fix different device problem in FCENet #334
- Ignore improper tailing empty characters in annotation files #358
- Docs fixes #247, #255, #265, #267, #268, #270, #276, #287, #330, #355, #367
- Fix NRTR config #356, #370

18.2.4 Improvements

- Add backend for resizeocr #244
- Skip image processing pipelines in SDMGR novisual #260
- Speedup DBNet #263
- Update mmcv installation method in workflow #323
- Add part of Chinese documentations #353, #362
- Add support for ConcatDataset with two workflows #348
- Add list_from_file and list_to_file utils #226
- Speed up sort_vertex #239
- Support distributed evaluation of KIE #234
- Add pretrained FCENet on IC15 #258
- Support CPU for OCR demo #227
- Avoid extra image pre-processing steps #375

18.3 v0.2.0 (18/5/2021)

18.3.1 Highlights

1. Add the NER approach Bert-softmax (NAACL'2019)
2. Add the text detection method DRRG (CVPR'2020)
3. Add the text detection method FCENet (CVPR'2021)
4. Increase the ease of use via adding text detection and recognition end-to-end demo, and colab online demo.
5. Simplify the installation.

18.3.2 New Features

- Add Bert-softmax for Ner task #148
- Add DRRG #189
- Add FCENet #133
- Add end-to-end demo #105
- Support batch inference #86 #87 #178
- Add TPS preprocessor for text recognition #117 #135
- Add demo documentation #151 #166 #168 #170 #171
- Add checkpoint for Chinese recognition #156
- Add metafile #175 #176 #177 #182 #183
- Add support for numpy array inference #74

18.3.3 Bug Fixes

- Fix the duplicated point bug due to transform for textsnake #130
- Fix CTC loss NaN #159
- Fix error raised if result is empty in demo #144
- Fix results missing if one image has a large number of boxes #98
- Fix package missing in dockerfile #109

18.3.4 Improvements

- Simplify installation procedure via removing compiling #188
- Speed up panet post processing so that it can detect dense texts #188
- Add zh-CN README #70 #95
- Support windows #89
- Add Colab #147 #199
- Add 1-step installation using conda environment #193 #194 #195

18.4 v0.1.0 (7/4/2021)

18.4.1 Highlights

- MMOCR is released.

18.4.2 Main Features

- Support text detection, text recognition and the corresponding downstream tasks such as key information extraction.
- For text detection, support both single-step (`PSENet`, `PANet`, `DBNet`, `TextSnake`) and two-step (`MaskRCNN`) methods.
- For text recognition, support CTC-loss based method `CRNN`; Encoder-decoder (with attention) based methods `SAR`, `Robustscanner`; Segmentation based method `SegOCR`; Transformer based method `NRTR`.
- For key information extraction, support GCN based method `SDMG-R`.
- Provide checkpoints and log files for all of the methods above.

CHAPTER

NINETEEN

API REFERENCE

19.1 mmocr.apis

19.2 mmocr.core

19.2.1 evaluation

19.3 mmocr.utils

19.4 mmocr.models

19.4.1 common_backbones

19.4.2 textdet_dense_heads

19.4.3 textdet_necks

19.4.4 textdet_detectors

19.4.5 textdet_losses

19.4.6 textdet_postprocess

19.4.7 textrecog_recognizer

19.4.8 textrecog_backbones

19.4.9 textrecog_necks

19.4.10 textrecog_heads

19.4.11 textrecog_convertors

19.4.12 textrecog_encoders

19.4.13 textrecog_decoders

19.4.14 textrecog_losses

19.4.15 textrecog_backbones

CHAPTER

TWENTY

INDICES AND TABLES

- genindex
- search