MMOCR

Release 1.0.1

OpenMMLab

Sep 12, 2023

GET STARTED

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

CHAPTER

ONE

OVERVIEW

MMOCR is an open source toolkit based on PyTorch and MMDetection, supporting numerous OCR-related models, including text detection, text recognition, and key information extraction. In addition, it supports widely-used academic datasets and provides many useful tools, assisting users in exploring various aspects of models and datasets and implementing high-quality algorithms. Generally, it has the following features.

- **One-stop, Multi-model**: MMOCR supports various OCR-related tasks and implements the latest models for text detection, recognition, and key information extraction.
- **Modular Design**: MMOCR's modular design allows users to define and reuse modules in the model on demand.
- **Various Useful Tools**: MMOCR provides a number of analysis tools, including visualizers, validation scripts, evaluators, etc., to help users troubleshoot, finetune or compare models.
- **Powered by OpenMMLab**: Like other algorithm libraries in OpenMMLab family, MMOCR follows OpenMMLab's rigorous development guidelines and interface conventions, significantly reducing the learning cost of users familiar with other projects in OpenMMLab family. In addition, benefiting from the unified interfaces among OpenMMLab, you can easily call the models implemented in other OpenMMLab projects (e.g. MMDetection) in MMOCR, facilitating cross-domain research and real-world applications.

Together with the release of OpenMMLab 2.0, MMOCR now also comes to its 1.0.0 version, which has made significant BC-breaking changes, resulting in less code redundancy, higher code efficiency and an overall more systematic and consistent design.

Considering that there are some backward incompatible changes in this version compared to 0.x, we have prepared a detailed *migration guide*. It lists all the changes made in the new version and the steps required to migrate. We hope this guide can help users familiar with the old framework to complete the upgrade as quickly as possible. Though this may take some time, we believe that the new features brought by MMOCR and the OpenMMLab ecosystem will make it all worthwhile.

Next, please read the section according to your actual needs.

- We recommend that beginners go through *Quick Run* to get familiar with MMOCR and master the usage of MMOCR by reading the examples in **User Guides**.
- Intermediate and advanced developers are suggested to learn the background, conventions, and recommended implementations of each component from **Basic Concepts**.
- Read our *FAQ* to find answers to frequently asked questions.
- If you can't find the answers you need in the documentation, feel free to raise an issue.
- Everyone is welcome to be a contributor! Read the *contribution guide* to learn how to contribute to MMOCR!

CHAPTER

TWO

INSTALLATION

2.1 Prerequisites

- Linux | Windows | macOS
- Python 3.7
- PyTorch 1.6 or higher
- torchvision 0.7.0
- CUDA 10.1
- NCCL 2
- GCC 5.4.0 or higher

2.2 Environment Setup

Note: If you are experienced with PyTorch and have already installed it, just skip this part and jump to the *next section*. Otherwise, you can follow these steps for the preparation.

Step 0. Download and install Miniconda from the official website.

Step 1. Create a conda environment and activate it.

```
conda create --name openmmlab python=3.8 -y
conda activate openmmlab
```

Step 2. Install PyTorch following official instructions, e.g.

GPU Platform

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

CPU Platform

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

2.3 Installation Steps

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

2.3.1 Best Practices

Step 0. Install MMEngine, MMCV and MMDetection using MIM.

```
pip install -U openmim
mim install mmengine
mim install mmcv
mim install mmdet
```

Step 1. Install MMOCR.

If you wish to run and develop MMOCR directly, install it from **source** (recommended).

If you use MMOCR as a dependency or third-party package, install it via **MIM**.

Install from Source

```
git clone https://github.com/open-mmlab/mmocr.git
cd mmocr
pip install -v -e .
# "-v" increases pip's verbosity.
# "-e" means installing the project in editable mode,
# That is, any local modifications on the code will take effect immediately.
```

Install via MIM

```
mim install mmocr
```

Step 2. **(Optional)** If you wish to use any transform involving `albumentations` (For example, `Albu` in ABINet's pipeline), or any dependency for building documentation or running unit tests, please install the dependency using the following command:

Install from Source

```
# install albu
pip install -r requirements/albu.txt
# install the dependencies for building documentation and running unit tests
pip install -r requirements.txt
```

Install via MIM

```
pip install albumentations>=1.1.0 --no-binary qudida,albumentations
```

Note: We recommend checking the environment after installing `albumentations` to ensure that `opencv-python` and `opencv-python-headless` are not installed together, otherwise it might cause unexpected issues. If that's unfortunately the case, please uninstall `opencv-python-headless` to make sure MMOCR's visualization utilities can work.

Refer to albumentations's official documentation for more details.

2.3.2 Verify the installation

You may verify the installation via this inference demo.

Python

Run the following code in a Python interpreter:

```
>>> from mmocr.apis import MMOCRInferencer
>>> ocr = MMOCRInferencer(det='DBNet', rec='CRNN')
>>> ocr('demo/demo_text_ocr.jpg', show=True, print_result=True)
```

Shell

If you installed MMOCR from source, you can run the following in MMOCR's root directory:

```
python tools/infer.py demo/demo_text_ocr.jpg --det DBNet --rec CRNN --show --print-result
```

You should be able to see a pop-up image and the inference result printed out in the console upon successful verification.

```
# Inference result
{'predictions': [{'rec_texts': ['cbanks', 'docecea', 'grouf', 'pwate', 'chobnsonsg',
↪'soxee', 'oeioh', 'c', 'sones', 'lbrandec', 'sretalg', '11', 'to8', 'round', 'sale',
↪'year',
'ally', 'sie', 'sall'], 'rec_scores': [...], 'det_polygons': [...], 'det_scores':
[...]}]}
```

Note: If you are running MMOCR on a server without GUI or via SSH tunnel with X11 forwarding disabled, you may not see the pop-up window.

2.4 Customize Installation

2.4.1 CUDA versions

When installing PyTorch, you need to specify the version of CUDA. If you are not clear on which to choose, follow our recommendations:

- For Ampere-based NVIDIA GPUs, such as GeForce 30 series and NVIDIA A100, CUDA 11 is a must.
- For older NVIDIA GPUs, CUDA 11 is backward compatible, but CUDA 10.2 offers better compatibility and is more lightweight.

Please make sure the GPU driver satisfies the minimum version requirements. See this table for more information.

Note: Installing CUDA runtime libraries is enough if you follow our best practices, because no CUDA code will be compiled locally. However if you hope to compile MMCV from source or develop other CUDA operators, you need to install the complete CUDA toolkit from NVIDIA's website, and its version should match the CUDA version of PyTorch. i.e., the specified version of cudatoolkit in `conda install` command.

2.4.2 Install MMCV without MIM

MMCV contains C++ and CUDA extensions, thus depending on PyTorch in a complex way. MIM solves such dependencies automatically and makes the installation easier. However, it is not a must.

To install MMCV with pip instead of MIM, please follow MMCV installation guides. This requires manually specifying a find-url based on PyTorch version and its CUDA version.

For example, the following command install mmcv-full built for PyTorch 1.10.x and CUDA 11.3.

```
pip install `mmcv>=2.0.0rc1`  -f https://download.openmmlab.com/mmcv/dist/cu113/torch1.
↪10/index.html
```

2.4.3 Install on CPU-only platforms

MMOCR can be built for CPU-only environment. In CPU mode you can train (requires MMCV version >= 1.4.4), test or inference a model.

However, some functionalities are gone in this mode:

- Deformable Convolution
- Modulated Deformable Convolution
- ROI pooling
- SyncBatchNorm

If you try to train/test/inference a model containing above ops, an error will be raised. The following table lists affected algorithms.

2.4.4 Using MMOCR with Docker

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
```

2.5 Dependency on MMEngine, MMCV & MMDetection

MMOCR has different version requirements on MMEngine, MMCV and MMDetection at each release to guarantee the implementation correctness. Please refer to the table below and ensure the package versions fit the requirement.

CHAPTER

THREE

QUICK RUN

This chapter will take you through the basic functions of MMOCR. And we assume you *installed MMOCR from source*. You may check out the tutorial notebook for how to perform inference, training and testing interactively.

3.1 Inference

Run the following in MMOCR's root directory:

```
python tools/infer.py demo/demo_text_ocr.jpg --det DBNet --rec CRNN --show --print-result
```

You should be able to see a pop-up image and the inference result printed out in the console.

```
# Inference result
{'predictions': [{'rec_texts': ['cbanks', 'docecea', 'grouf', 'pwate', 'chobnsonsg',
↪'soxee', 'oeioh', 'c', 'sones', 'lbrandec', 'sretalg', '11', 'to8', 'round', 'sale',
↪'year',
'ally', 'sie', 'sall'], 'rec_scores': [...], 'det_polygons': [...], 'det_scores':
[...]}]}
```

Note: If you are running MMOCR on a server without GUI or via SSH tunnel with X11 forwarding disabled, you may not see the pop-up window.

A detailed description of MMOCR's inference interface can be found *here*

In addition to using our well-provided pre-trained models, you can also train models on your own datasets. In the next section, we will take you through the basic functions of MMOCR by training DBNet on the mini ICDAR 2015 dataset as an example.

3.2 Prepare a Dataset

Since the variety of OCR dataset formats are not conducive to either switching or joint training of multiple datasets, MMOCR proposes a uniform *data format*, and provides *dataset preparer* for commonly used OCR datasets. Usually, to use those datasets in MMOCR, you just need to follow the steps to get them ready for use.

Note: But here, efficiency means everything.

Here, we have prepared a lite version of ICDAR 2015 dataset for demonstration purposes. Download our pre-prepared zip and extract it to the `data/` directory under mmocr to get our prepared image and annotation file.

```
wget https://download.openmmlab.com/mmocr/data/icdar2015/mini_icdar2015.tar.gz
mkdir -p data/
tar xzvf mini_icdar2015.tar.gz -C data/
```

3.3 Modify the Config

Once the dataset is prepared, we will then specify the location of the training set and the training parameters by modifying the config file.

In this example, we will train a DBNet using resnet18 as its backbone. Since MMOCR already has a config file for the full ICDAR 2015 dataset (`configs/textdet/dbnet/dbnet_resnet18_fpnc_1200e_icdar2015.py`), we just need to make some modifications on top of it.

We first need to modify the path to the dataset. In this config, most of the key config files are imported in `_base_`, such as the database configuration from `configs/textdet/_base_/datasets/icdar2015.py`. Open that file and replace the path pointed to by `icdar2015_textdet_data_root` in the first line with:

```
icdar2015_textdet_data_root = 'data/mini_icdar2015'
```

Also, because of the reduced dataset size, we have to reduce the number of training epochs to 400 accordingly, shorten the validation interval as well as the weight storage interval to 10 rounds, and drop the learning rate decay strategy. The following lines of configuration can be directly put into `configs/textdet/dbnet/dbnet_resnet18_fpnc_1200e_icdar2015.py` to take effect.

```
# Save checkpoints every 10 epochs, and only keep the latest checkpoint
default_hooks = dict(
    checkpoint=dict(
        type='CheckpointHook',
        interval=10,
        max_keep_ckpts=1,
    ))
# Set the maximum number of epochs to 400, and validate the model every 10 epochs
train_cfg = dict(type='EpochBasedTrainLoop', max_epochs=400, val_interval=10)
# Fix learning rate as a constant
param_scheduler = [
    dict(type='ConstantLR', factor=1.0),
]
```

Here, we have rewritten the corresponding parameters in the base configuration directly through the inheritance (MMEngine: Config) mechanism of the config. The original fields are distributed in `configs/textdet/_base_/schedules/schedule_sgd_1200e.py` and `configs/textdet/_base_/default_runtime.py`.

Note: For a more detailed description of config, please refer to *here*.

3.4 Browse the Dataset

Before we start the training, we can also visualize the image processed by training-time *data transforms*. It's quite simple: pass the config file we need to visualize into the browse_dataset.py script.

```
python tools/analysis_tools/browse_dataset.py configs/textdet/dbnet/dbnet_resnet18_fpnc_
↪1200e_icdar2015.py
```

The transformed images and annotations will be displayed one by one in a pop-up window.

Note: For details on the parameters and usage of this script, please refer to *here*.

Tip: In addition to satisfying our curiosity, visualization can also help us check the parts that may affect the model's performance before training, such as problems in configs, datasets and data transforms.

3.5 Training

Start the training by running the following command:

```
python tools/train.py configs/textdet/dbnet/dbnet_resnet18_fpnc_1200e_icdar2015.py
```

Depending on the system environment, MMOCR will automatically use the best device for training. If a GPU is available, a single GPU training will be started by default. When you start to see the output of the losses, you have successfully started the training.

```
2022/08/22 18:42:22 - mmengine - INFO - Epoch(train) [1][5/7]  lr: 7.0000e-03  memory:␣
↪7730  data_time: 0.4496  loss_prob: 14.6061  loss_thr: 2.2904  loss_db: 0.9879  loss:␣
↪17.8843  time: 1.8666
2022/08/22 18:42:24 - mmengine - INFO - Exp name: dbnet_resnet18_fpnc_1200e_icdar2015
2022/08/22 18:42:28 - mmengine - INFO - Epoch(train) [2][5/7]  lr: 7.0000e-03  memory:␣
↪6695  data_time: 0.2052  loss_prob: 6.7840  loss_thr: 1.4114  loss_db: 0.9855  loss: 9.
↪1809  time: 0.7506
2022/08/22 18:42:29 - mmengine - INFO - Exp name: dbnet_resnet18_fpnc_1200e_icdar2015
2022/08/22 18:42:33 - mmengine - INFO - Epoch(train) [3][5/7]  lr: 7.0000e-03  memory:␣
↪6690  data_time: 0.2101  loss_prob: 3.0700  loss_thr: 1.1800  loss_db: 0.9967  loss: 5.
↪2468  time: 0.6244
2022/08/22 18:42:33 - mmengine - INFO - Exp name: dbnet_resnet18_fpnc_1200e_icdar2015
```

Without extra configurations, model weights will be saved to `work_dirs/dbnet_resnet18_fpnc_1200e_icdar2015/`, while the logs will be stored in `work_dirs/dbnet_resnet18_fpnc_1200e_icdar2015/TIMESTAMP/`. Next, we just need to wait with some patience for training to finish.

Note: For advanced usage of training, such as CPU training, multi-GPU training, and cluster training, please refer to *Training and Testing*.

3.6 Testing

After 400 epochs, we observe that DBNet performs best in the last epoch, with hmean reaching 60.86 (You may see a different result):

```
08/22 19:24:52 - mmengine - INFO - Epoch(val) [400][100/100]  icdar/precision: 0.7285 ␣
↪icdar/recall: 0.5226  icdar/hmean: 0.6086
```

Note: It may not have been trained to be optimal, but it is sufficient for a demo.

However, this value only reflects the performance of DBNet on the mini ICDAR 2015 dataset. For a comprehensive evaluation, we also need to see how it performs on out-of-distribution datasets. For example, `tests/data/det_toy_dataset` is a very small real dataset that we can use to verify the actual performance of DBNet.

Before testing, we also need to make some changes to the location of the dataset. Open `configs/textdet/_base_/datasets/icdar2015.py` and change `data_root` of `icdar2015_textdet_test` to `tests/data/det_toy_dataset`:

```
# ...
icdar2015_textdet_test = dict(
    type='OCRDataset',
    data_root='tests/data/det_toy_dataset',
    #  ...
    )
```

Start testing:

```
python tools/test.py configs/textdet/dbnet/dbnet_resnet18_fpnc_1200e_icdar2015.py work_
↪dirs/dbnet_resnet18_fpnc_1200e_icdar2015/epoch_400.pth
```

And get the outputs like:

```
08/21 21:45:59 - mmengine - INFO - Epoch(test) [5/10]    memory: 8562
08/21 21:45:59 - mmengine - INFO - Epoch(test) [10/10]    eta: 0:00:00  time: 0.4893 ␣
↪data_time: 0.0191  memory: 283
08/21 21:45:59 - mmengine - INFO - Evaluating hmean-iou...
08/21 21:45:59 - mmengine - INFO - prediction score threshold: 0.30, recall: 0.6190,␣
↪precision: 0.4815, hmean: 0.5417
08/21 21:45:59 - mmengine - INFO - prediction score threshold: 0.40, recall: 0.6190,␣
↪precision: 0.5909, hmean: 0.6047
08/21 21:45:59 - mmengine - INFO - prediction score threshold: 0.50, recall: 0.6190,␣
↪precision: 0.6842, hmean: 0.6500
08/21 21:45:59 - mmengine - INFO - prediction score threshold: 0.60, recall: 0.6190,␣
↪precision: 0.7222, hmean: 0.6667
08/21 21:45:59 - mmengine - INFO - prediction score threshold: 0.70, recall: 0.3810,␣
↪precision: 0.8889, hmean: 0.5333
08/21 21:45:59 - mmengine - INFO - prediction score threshold: 0.80, recall: 0.0000,␣
↪precision: 0.0000, hmean: 0.0000
08/21 21:45:59 - mmengine - INFO - prediction score threshold: 0.90, recall: 0.0000,␣
↪precision: 0.0000, hmean: 0.0000
08/21 21:45:59 - mmengine - INFO - Epoch(test) [10/10]  icdar/precision: 0.7222  icdar/
↪recall: 0.6190  icdar/hmean: 0.6667
```

The model achieves an hmean of 0.6667 on this dataset.

Note: For advanced usage of testing, such as CPU testing, multi-GPU testing, and cluster testing, please refer to *Training and Testing*.

3.7 Visualize the Outputs

We can also visualize its prediction output in `test.py`. You can open a pop-up visualization window with the `show` parameter; and can also specify the directory where the prediction result images are exported with the `show-dir` parameter.

```
python tools/test.py configs/textdet/dbnet/dbnet_resnet18_fpnc_1200e_icdar2015.py work_
↪dirs/dbnet_resnet18_fpnc_1200e_icdar2015/epoch_400.pth --show-dir imgs/
```

The true labels and predicted values are displayed in a tiled fashion in the visualization results. The green boxes in the left panel indicate the true labels and the red boxes in the right panel indicate the predicted values.

Note: For a description of more visualization features, see *here*.

CHAPTER

FOUR

FAQ

4.1 General

Q1 I'm getting the warning like `unexpected key in source state_dict: fc.weight, fc.bias`, is there something wrong?

A It's not an error. It occurs because the backbone network is pretrained on image classification tasks, where the last fc layer is required to generate the classification output. However, the fc layer is no longer needed when the backbone network is used to extract features in downstream tasks, and therefore these weights can be safely skipped when loading the checkpoint.

Q2 MMOCR terminates with an error: `shapely.errors.TopologicalError: The operation 'GEOSIntersection_r' could not be performed. Likely cause is invalidity of the geometry`. How could I fix it?

A This error occurs because of some invalid polygons (e.g., polygons with self-intersections) existing in the dataset or generated by some non-rigorous data transforms. These polygons can be fixed by adding `FixInvalidPolygon` transform after the transform likely to introduce invalid polygons. For example, a common practice is to append it after `LoadOCRAnnotations` in both train and test pipeline. The resulting pipeline should look like:

```
train_pipeline = [
    ...
    dict(
        type='LoadOCRAnnotations',
        with_polygon=True,
        with_bbox=True,
        with_label=True,
    ),
    dict(type='FixInvalidPolygon', min_poly_points=4),
    ...
]
```

In practice, we find that Totaltext contains some invalid polygons and using `FixInvalidPolygon` is a must. Here is an example config.

Q3 Getting `libpng warning: iCCP: known incorrect sRGB profile` when loading images with `cv2` backend.

A This is a warning from `libpng` and it is safe to ignore. It is caused by the `icc` profile in the image. You can use `pillow` backend to avoid this warning:

```
train_pipeline = [
    dict(
```

(continues on next page)

(continued from previous page)

```
        type='LoadImageFromFile',
        imdecode_backend='pillow'),
    ...
]
```

4.2 Text Recognition

Q1 What are the steps to train text recognition models with my own dictionary?

A In MMOCR 1.0, you only need to modify the config and point `Dictionary` to your custom dict file. For example, if you want to train SAR model (https://github.com/open-mmlab/mmocr/blob/75c06d34bbc01d3d11dfd7afc098b6cdeee82579/configs/textrecog/sar/sar_resnet31_parallel-decoder_5e_st-sub_mj-sub_sa_real.py) with your own dictionary placed at `/my/dict.txt`, you can modify `dictionary.dict_file` term in base config to:

```
dictionary = dict(
    type='Dictionary',
    dict_file='/my/dict.txt',
    with_start=True,
    with_end=True,
    same_start_end=True,
    with_padding=True,
    with_unknown=True)
```

Now you are good to go. You can also find more information in Dictionary API.

Q2 How to properly visualize non-English characters?

A You can customize `font_families` or `font_properties` in visualizer. For example, to visualize Korean:

`configs/textrecog/_base_/default_runtime.py`:

```
visualizer = dict(
    type='TextRecogLocalVisualizer',
    name='visualizer',
    font_families='NanumGothic', # new feature
    vis_backends=vis_backends)
```

It's also fine to pass the font path to visualizer:

```
visualizer = dict(
    type='TextRecogLocalVisualizer',
    name='visualizer',
    font_properties='path/to/font_file',
    vis_backends=vis_backends)
```

CHAPTER

FIVE

INFERENCE

In OpenMMLab, all the inference operations are unified into a new interface - `Inferencer`. `Inferencer` is designed to expose a neat and simple API to users, and shares very similar interface across different OpenMMLab libraries.

In MMOCR, Inferencers are constructed in different levels of task abstraction.

- Standard Inferencer: Following OpenMMLab's convention, each fundamental task in MMOCR has a standard Inferencer, namely `TextDetInferencer` (text detection), `TextRecInferencer` (text recognition), `TextSpottingInferencer` (end-to-end OCR), and `KIEInferencer` (key information extraction). They are designed to perform inference on a single task, and can be chained together to perform inference on a series of tasks. They also share very similar interface, have standard input/output protocol, and overall follow the OpenMMLab design.
- **MMOCRInferencer**: We also provide `MMOCRInferencer`, a convenient inference interface only designed for MMOCR. It encapsulates and chains all the Inferencers in MMOCR, so users can use this Inferencer to perform a series of tasks on an image and directly get the final result in an end-to-end manner. *However, it has a relatively different interface from other standard Inferencers, and some of standard Inferencer functionalities might be sacrificed for the sake of simplicity.*

For new users, we recommend using **MMOCRInferencer** to test out different combinations of models.

If you are a developer and wish to integrate the models into your own project, we recommend using **standard Inferencers**, as they are more flexible and standardized, equipped with full functionalities.

5.1 Basic Usage

MMOCRInferencer

As of now, `MMOCRInferencer` can perform inference on the following tasks:

- Text detection
- Text recognition
- OCR (text detection + text recognition)
- Key information extraction (text detection + text recognition + key information extraction)
- *OCR (text spotting)* (coming soon)

For convenience, `MMOCRInferencer` provides both Python and command line interfaces. For example, if you want to perform OCR inference on `demo/demo_text_ocr.jpg` with `DBNet` as the text detection model and `CRNN` as the text recognition model, you can simply run the following command:

Python

```
>>> from mmocr.apis import MMOCRInferencer
>>> # Load models into memory
>>> ocr = MMOCRInferencer(det='DBNet', rec='SAR')
>>> # Perform inference
>>> ocr('demo/demo_text_ocr.jpg', show=True)
```

Bash

```
python tools/infer.py demo/demo_text_ocr.jpg --det DBNet --rec SAR --show
```

The resulting OCR output will be displayed in a new window:

Note: If you are running MMOCR on a server without GUI or via SSH tunnel with X11 forwarding disabled, the `show` option will not work. However, you can still save visualizations to files by setting `out_dir` and `save_vis=True` arguments. Read *Dumping Results* for details.

Depending on the initialization arguments, `MMOCRInferencer` can run in different modes. For example, it can run in KIE mode if it is initialized with `det`, `rec` and `kie` specified.

Python

```
>>> kie = MMOCRInferencer(det='DBNet', rec='SAR', kie='SDMGR')
>>> kie('demo/demo_kie.jpeg', show=True)
```

Bash

```
python tools/infer.py demo/demo_kie.jpeg --det DBNet --rec SAR --kie SDMGR --show
```

The output image should look like this:

You may have found that the Python interface and the command line interface of `MMOCRInferencer` are very similar. The following sections will use the Python interface as an example to introduce the usage of `MMOCRInferencer`. For more information about the command line interface, please refer to *Command Line Interface*.

Standard Inferencer

In general, all the standard Inferencers across OpenMMLab share a very similar interface. The following example shows how to use `TextDetInferencer` to perform inference on a single image.

```
>>> from mmocr.apis import TextDetInferencer
>>> # Load models into memory
>>> inferencer = TextDetInferencer(model='DBNet')
>>> # Inference
>>> inferencer('demo/demo_text_ocr.jpg', show=True)
```

The visualization result should look like:

5.2 Initialization

Each Inferencer must be initialized with a model. You can also choose the inference device during initialization.

5.2.1 Model Initialization

MMOCRInferencer

For each task, `MMOCRInferencer` takes two arguments in the form of `xxx` and `xxx_weights` (e.g. `det` and `det_weights`) for initialization, and there are many ways to initialize a model for inference. We will take `det` and `det_weights` as an example to illustrate some typical ways to initialize a model.

- To infer with MMOCR's pre-trained model, passing its name to the argument `det` can work. The weights will be automatically downloaded and loaded from OpenMMLab's model zoo. Check *Weights* for available model names.

```
>>> MMOCRInferencer(det='DBNet')
```

- To load custom config and weight, you can pass the path to the config file to `det` and the path to the weight to `det_weights`.

```
>>> MMOCRInferencer(det='path/to/dbnet_config.py', det_weights='path/to/dbnet.pth')
```

You may click on the "Standard Inferencer" tab to find more initialization methods.

Standard Inferencer

Every standard `Inferencer` accepts two parameters, `model` and `weights`. (In `MMOCRInferencer`, they are referred to as `xxx` and `xxx_weights`)

- `model` takes either the name of a model, or the path to a config file as input. The name of a model is obtained from the model's metafile (Example) indexed from model-index.yml. You can find the list of available weights *here*.
- `weights` accepts the path to a weight file.

There are various ways to initialize a model.

- To infer with MMOCR's pre-trained model, you can pass its name to `model`. The weights will be automatically downloaded and loaded from OpenMMLab's model zoo.

```
>>> from mmocr.apis import TextDetInferencer
>>> inferencer = TextDetInferencer(model='DBNet')
```

Note: The model type must match the Inferencer type.

You can load another weight by passing its path/url to `weights`.

```
>>> inferencer = TextDetInferencer(model='DBNet', weights='path/to/dbnet.pth')
```

- To load custom config and weight, you can pass the path to the config file to `model` and the path to the weight to `weights`.

```
>>> inferencer = TextDetInferencer(model='path/to/dbnet_config.py', weights='path/
↪to/dbnet.pth')
```

- By default, MMEngine dumps config to the weight. If you have a weight trained on MMEngine, you can also pass the path to the weight file to `weights` without specifying `model`:

```
>>> # It will raise an error if the config file cannot be found in the weight
>>> inferencer = TextDetInferencer(weights='path/to/dbnet.pth')
```

- Passing config file to `model` without specifying `weight` will result in a randomly initialized model.

5.2.2 Device

Each Inferencer instance is bound to a device. By default, the best device is automatically decided by MMEngine. You can also alter the device by specifying the `device` argument. For example, you can use the following code to create an Inferencer on GPU 1.

MMOCRInferencer

```
>>> inferencer = MMOCRInferencer(det='DBNet', device='cuda:1')
```

Standard Inferencer

```
>>> inferencer = TextDetInferencer(model='DBNet', device='cuda:1')
```

To create an Inferencer on CPU:

MMOCRInferencer

```
>>> inferencer = MMOCRInferencer(det='DBNet', device='cpu')
```

Standard Inferencer

```
>>> inferencer = TextDetInferencer(model='DBNet', device='cpu')
```

Refer to torch.device for all the supported forms.

5.3 Inference

Once the Inferencer is initialized, you can directly pass in the raw data to be inferred and get the inference results from return values.

5.3.1 Input

MMOCRInferencer / TextDetInferencer / TextRecInferencer / TextSpottingInferencer

Input can be either of these types:

- str: Path/URL to the image.

```
>>> inferencer('demo/demo_text_ocr.jpg')
```

- array: Image in numpy array. It should be in BGR order.

```
>>> import mmcv
>>> array = mmcv.imread('demo/demo_text_ocr.jpg')
>>> inferencer(array)
```

- list: A list of basic types above. Each element in the list will be processed separately.

```
>>> inferencer(['img_1.jpg', 'img_2.jpg])
>>> # You can even mix the types
>>> inferencer(['img_1.jpg', array])
```

- str: Path to the directory. All images in the directory will be processed.

```
>>> inferencer('tests/data/det_toy_dataset/imgs/test/')
```

KIEInferencer

Input can be a dict or list[dict], where each dictionary contains following keys:

- `img` (str or ndarray): Path to the image or the image itself. If KIE Inferencer is used in no-visual mode, this key is not required. If it's an numpy array, it should be in BGR order.
- `img_shape` (tuple(int, int)): Image shape in (H, W). Only required when KIE Inferencer is used in no-visual mode and no `img` is provided.
- `instances` (list[dict]): A list of instances.

Each `instance` looks like the following:

```
{
    # A nested list of 4 numbers representing the bounding box of
    # the instance, in (x1, y1, x2, y2) order.
    "bbox": np.array([[x1, y1, x2, y2], [x1, y1, x2, y2], ...],
                     dtype=np.int32),

    # List of texts.
    "texts": ['text1', 'text2', ...],
}
```

5.3.2 Output

By default, each `Inferencer` returns the prediction results in a dictionary format.

- `visualization` contains the visualized predictions. But it's an empty list by default unless `return_vis=True`.
- `predictions` contains the predictions results in a json-serializable format. As presented below, the contents are slightly different depending on the task type.

  MMOCRInferencer

```
{
    'predictions' : [
      # Each instance corresponds to an input image
      {
        'det_polygons': [...],  # 2d list of length (N,), format: [x1, y1, x2, y2, .
↪..]
        'det_scores': [...],  # float list of length (N,)
        'det_bboxes': [...],   # 2d list of shape (N, 4), format: [min_x, min_y,␣
↪max_x, max_y]
        'rec_texts': [...],  # str list of length (N,)
        'rec_scores': [...],  # float list of length (N,)
        'kie_labels': [...],  # node labels, length (N, )
```

(continues on next page)

(continued from previous page)

```
        'kie_scores': [...],  # node scores, length (N, )
        'kie_edge_scores': [...],  # edge scores, shape (N, N)
        'kie_edge_labels': [...]  # edge labels, shape (N, N)
      },
      ...
    ],
    'visualization' : [
      array(..., dtype=uint8),
    ]
}
```

Standard Inferencer

TextDetInferencer

```
{
    'predictions' : [
      # Each instance corresponds to an input image
      {
        'polygons': [...],  # 2d list of len (N,) in the format of [x1, y1, x2, y2,␣
↪...]
        'bboxes': [...],  # 2d list of shape (N, 4), in the format of [min_x, min_y,
↪ max_x, max_y]
        'scores': [...]  # list of float, len (N, )
      },
    ]
    'visualization' : [
      array(..., dtype=uint8),
    ]
}
```

TextRecInferencer

```
{
    'predictions' : [
      # Each instance corresponds to an input image
      {
        'text': '...',  # a string
        'scores': 0.1,  # a float
      },
      ...
    ]
    'visualization' : [
      array(..., dtype=uint8),
    ]
}
```

TextSpottingInferencer

```
{
    'predictions' : [
      # Each instance corresponds to an input image
      {
```

(continues on next page)

(continued from previous page)

```
        'polygons': [...],  # 2d list of len (N,) in the format of [x1, y1, x2, y2,␣
↪...]
        'bboxes': [...],  # 2d list of shape (N, 4), in the format of [min_x, min_y,
↪ max_x, max_y]
        'scores': [...]  # list of float, len (N, )
        'texts': ['...',]  # list of texts, len (N, )
      },
    ]
    'visualization' : [
      array(..., dtype=uint8),
    ]
}
```

KIEInferencer

```
{
    'predictions' : [
      # Each instance corresponds to an input image
      {
        'labels': [...],  # node label, len (N,)
        'scores': [...],  # node scores, len (N, )
        'edge_scores': [...],  # edge scores, shape (N, N)
        'edge_labels': [...],  # edge labels, shape (N, N)
      },
    ]
    'visualization' : [
      array(..., dtype=uint8),
    ]
}
```

If you wish to get the raw outputs from the model, you can set `return_datasamples` to `True` to get the original DataSample, which will be stored in `predictions`.

5.3.3 Dumping Results

Apart from obtaining predictions from the return value, you can also export the predictions/visualizations to files by setting `out_dir` and `save_pred`/`save_vis` arguments.

```
>>> inferencer('img_1.jpg', out_dir='outputs/', save_pred=True, save_vis=True)
```

Results in the directory structure like:

```
outputs
├── preds
│   └── img_1.json
└── vis
    └── img_1.jpg
```

The filename of each file is the same as the corresponding input image filename. If the input image is an array, the filename will be a number starting from 0.

5.3.4 Batch Inference

You can customize the batch size by setting `batch_size`. The default batch size is 1.

5.4 API

Here are extensive lists of parameters that you can use.

MMOCRInferencer

MMOCRInferencer.__init__():

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

MMOCRInferencer.__call__()

Standard Inferencer

Inferencer.__init__():

Inferencer.__call__()

5.5 Command Line Interface

Note: This section is only applicable to `MMOCRInferencer`.

You can use `tools/infer.py` to perform inference through `MMOCRInferencer`. Its general usage is as follows:

```
python tools/infer.py INPUT_PATH [--det DET] [--det-weights ...] ...
```

where `INPUT_PATH` is a required field, which should be a path to an image or a folder. Command-line parameters follow the mapping relationship with the Python interface parameters as follows:

- To convert the Python interface parameters to the command line ones, you need to add two `--` in front of the Python interface parameters, and replace the underscore `_` with the hyphen `-`. For example, `out_dir` becomes `--out-dir`.
- For boolean type parameters, putting the parameter in the command is equivalent to specifying it as True. For example, `--show` will specify the `show` parameter as True.

In addition, the command line will not display the inference result by default. You can use the `--print-result` parameter to view the inference result.

Here is an example:

```
python tools/infer.py demo/demo_text_ocr.jpg --det DBNet --rec SAR --show --print-result
```

Running this command will give the following result:

```
{'predictions': [{'rec_texts': ['CBank', 'Docbcba', 'GROUP', 'MAUN', 'CROBINSONS', 'AOCOC
↪', '916M3', 'BOO9', 'Oven', 'BRANDS', 'ARETAIL', '14', '70<UKN>S', 'ROUND', 'SALE',
↪'YEAR', 'ALLY', 'SALE', 'SALE'],
'rec_scores': [0.9753464579582214, ...], 'det_polygons': [[551.9930285844646, 411.
↪9138765335083, 553.6153911653112,
383.53195309638977, 620.2410061195247, 387.33785033226013, 618.6186435386782, 415.
↪71977376937866], ...], 'det_scores': [0.8230461478233337, ...]}]}
```

(continues on next page)

(continued from previous page)

CHAPTER

SIX

CONFIG

MMOCR mainly uses Python files as configuration files. The design of its configuration file system integrates the ideas of modularity and inheritance to facilitate various experiments.

6.1 Common Usage

Note: This section is recommended to be read together with the primary usage in MMEngine: Config.

There are three most common operations in MMOCR: inheritance of configuration files, reference to `_base_` variables, and modification of `_base_` variables. Config provides two syntaxes for inheriting and modifying `_base_`, one for Python, Json, and Yaml, and one for Python configuration files only. In MMOCR, we **prefer the Python-only syntax**, so this will be the basis for further description.

The `configs/textdet/dbnet/dbnet_resnet18_fpnc_1200e_icdar2015.py` is used as an example to illustrate the three common uses.

```
_base_ = [
    '_base_dbnet_resnet18_fpnc.py',
    '../_base_/datasets/icdar2015.py',
    '../_base_/default_runtime.py',
    '../_base_/schedules/schedule_sgd_1200e.py',
]

# dataset settings
icdar2015_textdet_train = _base_.icdar2015_textdet_train
icdar2015_textdet_train.pipeline = _base_.train_pipeline
icdar2015_textdet_test = _base_.icdar2015_textdet_test
icdar2015_textdet_test.pipeline = _base_.test_pipeline

train_dataloader = dict(
    batch_size=16,
    num_workers=8,
    persistent_workers=True,
    sampler=dict(type='DefaultSampler', shuffle=True),
    dataset=icdar2015_textdet_train)

val_dataloader = dict(
    batch_size=1,
    num_workers=4,
```

(continues on next page)

(continued from previous page)

```
    persistent_workers=True,
    sampler=dict(type='DefaultSampler', shuffle=False),
    dataset=icdar2015_textdet_test)
```

6.1.1 Configuration Inheritance

There is an inheritance mechanism for configuration files, i.e. one configuration file A can use another configuration file B as its base and inherit all the fields directly from it, thus avoiding a lot of copy-pasting.

In `dbnet_resnet18_fpnc_1200e_icdar2015.py` you can see that

```
_base_ = [
    '_base_dbnet_resnet18_fpnc.py',
    '../_base_/datasets/icdar2015.py',
    '../_base_/default_runtime.py',
    '../_base_/schedules/schedule_sgd_1200e.py',
]
```

The above statement reads all the base configuration files in the list, and all the fields in them are loaded into `dbnet_resnet18_fpnc_1200e_icdar2015.py`. We can see the structure of the configuration file after it has been parsed by running the following statement in a Python interpretation.

```
from mmengine import Config
db_config = Config.fromfile('configs/textdet/dbnet/dbnet_resnet18_fpnc_1200e_icdar2015.py
↪')
print(db_config)
```

It can be found that the parsed configuration contains all the fields and information in the base configuration.

Note: Variables with the same name cannot exist in each `_base_` profile.

6.1.2 _base_ Variable References

Sometimes we may need to reference some fields in the `_base_` configuration directly in order to avoid duplicate definitions. Suppose we want to get the variable `pseudo` in the `_base_` configuration, we can get the variable in the `_base_` configuration directly via `_base_.pseudo`.

This syntax has been used extensively in the configuration of MMOCR, and the dataset and pipeline configurations for each model in MMOCR are referenced in the base configuration. For example,

```
icdar2015_textdet_train = _base_.icdar2015_textdet_train
# ...
train_dataloader = dict(
    # ...
    dataset=icdar2015_textdet_train)
```

6.1.3 _base_ Variable Modification

In MMOCR, different algorithms usually have different pipelines in different datasets, so there are often scenarios to modify the `pipeline` in the dataset. There are also many scenarios where you need to modify variables in the `_base_` configuration, for example, modifying the training strategy of an algorithm, replacing some modules of an algorithm(backbone, etc.). Users can directly modify the referenced `_base_` variables using Python syntax. For dict, we also provide a method similar to class attribute modification to modify the contents of the dictionary directly.

1. Dictionary

   Here is an example of modifying `pipeline` in a dataset.

   The dictionary can be modified using Python syntax:

   ```
   # Get the dataset in _base_
   icdar2015_textdet_train = _base_.icdar2015_textdet_train
   # You can modify the variables directly with Python's update
   icdar2015_textdet_train.update(pipeline=_base_.train_pipeline)
   ```

   It can also be modified in the same way as changing Python class attributes.

   ```
   # Get the dataset in _base_
   icdar2015_textdet_train = _base_.icdar2015_textdet_train
   # The class property method is modified
   icdar2015_textdet_train.pipeline = _base_.train_pipeline
   ```

2. List

   Suppose the variable `pseudo = [1, 2, 3]` in the `_base_` configuration needs to be modified to `[1, 2, 4]`:

   ```
   # pseudo.py
   pseudo = [1, 2, 3]
   ```

   Can be rewritten directly as.

   ```
   _base_ = ['pseudo.py']
   pseudo = [1, 2, 4]
   ```

   Or modify the list using Python syntax:

   ```
   _base_ = ['pseudo.py']
   pseudo = _base_.pseudo
   pseudo[2] = 4
   ```

6.1.4 Command Line Modification

Sometimes we only want to fix part of the configuration and do not want to modify the configuration file itself. For example, if you want to change the learning rate during an experiment but do not want to write a new configuration file, you can pass in parameters on the command line to override the relevant configuration.

We can pass `--cfg-options` on the command line and modify the corresponding fields directly with the arguments after it. For example, we can run the following command to modify the learning rate temporarily for this training session.

```
python tools/train.py example.py --cfg-options optim_wrapper.optimizer.lr=1
```

For more detailed usage, refer to MMEngine: Command Line Modification.

6.2 Configuration Content

With config files and Registry, MMOCR can modify the training parameters as well as the model configuration without invading the code. Specifically, users can customize the following modules in the configuration file: environment configuration, hook configuration, log configuration, training strategy configuration, data-related configuration, model-related configuration, evaluation configuration, and visualization configuration.

This document will take the text detection algorithm `DBNet` and the text recognition algorithm `CRNN` as examples to introduce the contents of Config in detail.

6.2.1 Environment Configuration

```
default_scope = 'mmocr'
env_cfg = dict(
    cudnn_benchmark=True,
    mp_cfg=dict(mp_start_method='fork', opencv_num_threads=0),
    dist_cfg=dict(backend='nccl'))
randomness = dict(seed=None)
```

There are three main components:

- Set the default `scope` of all registries to `mmocr`, ensuring that all modules are searched first from the `MMOCR` codebase. If the module does not exist, the search will continue from the upstream algorithm libraries `MMEngine` and `MMCV`, see MMEngine: Registry for more details.
- `env_cfg` configures the distributed environment, see MMEngine: Runner for more details.
- `randomness`: Some settings to make the experiment as reproducible as possible like seed and deterministic. See MMEngine: Runner for more details.

6.2.2 Hook Configuration

Hooks are divided into two main parts, default hooks, which are required for all tasks to run, and custom hooks, which generally serve specific algorithms or specific tasks (there are no custom hooks in MMOCR so far).

```
default_hooks = dict(
    timer=dict(type='IterTimerHook'), # Time recording, including data time as well as␣
↪model inference time
    logger=dict(type='LoggerHook', interval=1), # Collect logs from different components
    param_scheduler=dict(type='ParamSchedulerHook'), # Update some hyper-parameters in␣
↪optimizer
    checkpoint=dict(type='CheckpointHook', interval=1),# Save checkpoint. `interval`␣
↪control save interval
    sampler_seed=dict(type='DistSamplerSeedHook'), # Data-loading sampler for␣
↪distributed training.
    sync_buffer=dict(type='SyncBuffersHook'), # Synchronize buffer in case of␣
↪distributed training
    visualization=dict( # Visualize the results of val and test
        type='VisualizationHook',
        interval=1,
        enable=False,
        show=False,
        draw_gt=False,
```

(continues on next page)

(continued from previous page)

```
        draw_pred=False))
custom_hooks = []
```

Here is a brief description of a few hooks whose parameters may be changed frequently. For a general modification method, refer to Modify configuration.

- `LoggerHook`: Used to configure the behavior of the logger. For example, by modifying `interval` you can control the interval of log printing, so that the log is printed once per `interval` iteration, for more settings refer to LoggerHook API.
- `CheckpointHook`: Used to configure checkpoint-related behavior, such as saving optimal and/or latest weights. You can also modify `interval` to control the checkpoint saving interval. More settings can be found in CheckpointHook API
- `VisualizationHook`: Used to configure visualization-related behavior, such as visualizing predicted results during validation or testing. **Default is off**. This Hook also depends on Visualization Configuration. You can refer to *Visualizer* for more details. For more configuration, you can refer to *VisualizationHook API*.

If you want to learn more about the configuration of the default hooks and their functions, you can refer to MMEngine: Hooks.

6.2.3 Log Configuration

This section is mainly used to configure the log level and the log processor.

```
log_level = 'INFO' # Logging Level
log_processor = dict(type='LogProcessor',
                     window_size=10,
                     by_epoch=True)
```

- The logging severity level is the same as that of Python: logging
- The log processor is mainly used to control the format of the output, detailed functions can be found in MMEngine: logging.
  - `by_epoch=True` indicates that the logs are output in accordance to "epoch", and the log format needs to be consistent with the `type='EpochBasedTrainLoop'` parameter in `train_cfg`. For example, if you want to output logs by iteration number, you need to set `by_epoch=False` in `log_processor` and `type='IterBasedTrainLoop'` in `train_cfg`.
  - `window_size` indicates the smoothing window of the loss, i.e. the average value of the various losses for the last `window_size` iterations. the final loss value printed in logger is the average of all the losses.

6.2.4 Training Strategy Configuration

This section mainly contains optimizer settings, learning rate schedules and `Loop` settings.

Training strategies usually vary for different tasks (text detection, text recognition, key information extraction). Here we explain the example configuration in `CRNN`, which is a text recognition model.

```
# optimizer
optim_wrapper = dict(
    type='OptimWrapper', optimizer=dict(type='Adadelta', lr=1.0))
param_scheduler = [dict(type='ConstantLR', factor=1.0)]
train_cfg = dict(type='EpochBasedTrainLoop',
```

(continues on next page)

(continued from previous page)

```
                    max_epochs=5, # train epochs
                    val_interval=1) # val interval
val_cfg = dict(type='ValLoop')
test_cfg = dict(type='TestLoop')
```

- `optim_wrapper` : It contains two main parts, OptimWrapper and Optimizer. Detailed usage information can be found in MMEngine: Optimizer Wrapper.
  - The Optimizer wrapper supports different training strategies, including mixed-accuracy training (AMP), gradient accumulation, and gradient truncation.
  - All PyTorch optimizers are supported in the optimizer settings. All supported optimizers are available in PyTorch Optimizer List.
- `param_scheduler` : learning rate tuning strategy, supports most of the learning rate schedulers in PyTorch, such as `ExponentialLR`, `LinearLR`, `StepLR`, `MultiStepLR`, etc., and is used in much the same way, see scheduler interface, and more features can be found in the MMEngine: Optimizer Parameter Tuning Strategy.
- `train/test/val_cfg` : the execution flow of the task, MMEngine provides four kinds of flow: `EpochBasedTrainLoop`, `IterBasedTrainLoop`, `ValLoop`, `TestLoop` More can be found in MMEngine: loop controller.

6.2.5 Data-related Configuration

Dataset Configuration

It is mainly about two parts.

- The location of the dataset(s), including images and annotation files.
- Data augmentation related configurations. In the OCR domain, data augmentation is usually strongly associated with the model.

More parameter configurations can be found in Data Base Class.

The naming convention for dataset fields in MMOCR is

```
{dataset}_{task}_{train/val/test} = dict(...)
```

- dataset: See dataset abbreviations
- task: `det`(text detection), `rec`(text recognition), `kie`(key information extraction)
- train/val/test: Dataset split.

For example, for text recognition tasks, Syn90k is used as the training set, while icdar2013 and icdar2015 serve as the test sets. These are configured as follows.

```
# text recognition dataset configuration
mjsynth_textrecog_train = dict(
    type='OCRDataset',
    data_root='data/rec/Syn90k/',
    data_prefix=dict(img_path='mnt/ramdisk/max/90kDICT32px'),
    ann_file='train_labels.json',
    test_mode=False,
    pipeline=None)
```

(continues on next page)

(continued from previous page)

```
icdar2013_textrecog_test = dict(
    type='OCRDataset',
    data_root='data/rec/icdar_2013/',
    data_prefix=dict(img_path='Challenge2_Test_Task3_Images/'),
    ann_file='test_labels.json',
    test_mode=True,
    pipeline=None)

icdar2015_textrecog_test = dict(
    type='OCRDataset',
    data_root='data/rec/icdar_2015/',
    data_prefix=dict(img_path='ch4_test_word_images_gt/'),
    ann_file='test_labels.json',
    test_mode=True,
    pipeline=None)
```

Data Pipeline Configuration

In MMOCR, dataset construction and data preparation are decoupled from each other. In other words, dataset classes such as `OCRDataset` are responsible for reading and parsing annotation files, while Data Transforms further implement data loading, data augmentation, data formatting and other related functions.

In general, there are different augmentation strategies for training and testing, so there are usually `training_pipeline` and `testing_pipeline`. More information can be found in *Data Transforms*

- The data augmentation process of the training pipeline is usually: data loading (LoadImageFromFile) -> annotation information loading (LoadXXXAnntation) -> data augmentation -> data formatting (PackXXXInputs).
- The data augmentation flow of the test pipeline is usually: Data Loading (LoadImageFromFile) -> Data Augmentation -> Annotation Loading (LoadXXXAnntation) -> Data Formatting (PackXXXInputs).

Due to the specificity of the OCR task, different models have different data augmentation techniques, and even the same model can have different data augmentation strategies for different datasets. Take `CRNN` as an example.

```
# Data Augmentation
train_pipeline = [
    dict(
        type='LoadImageFromFile',
        color_type='grayscale',
        ignore_empty=True,
        min_size=5),
    dict(type='LoadOCRAnnotations', with_text=True),
    dict(type='Resize', scale=(100, 32), keep_ratio=False),
    dict(
        type='PackTextRecogInputs',
        meta_keys=('img_path', 'ori_shape', 'img_shape', 'valid_ratio'))
]
test_pipeline = [
    dict(
        type='LoadImageFromFile',
        color_type='grayscale'),
    dict(
        type='RescaleToHeight',
```

(continues on next page)

(continued from previous page)

```
        height=32,
        min_width=32,
        max_width=None,
        width_divisor=16),
    dict(type='LoadOCRAnnotations', with_text=True),
    dict(
        type='PackTextRecogInputs',
        meta_keys=('img_path', 'ori_shape', 'img_shape', 'valid_ratio'))
]
```

Dataloader Configuration

The main configuration information needed to construct the dataset loader (dataloader), see PyTorch DataLoader for more tutorials.

```
# Dataloader
train_dataloader = dict(
    batch_size=64,
    num_workers=8,
    persistent_workers=True,
    sampler=dict(type='DefaultSampler', shuffle=True),
    dataset=dict(
        type='ConcatDataset',
        datasets=[mjsynth_textrecog_train],
        pipeline=train_pipeline))
val_dataloader = dict(
    batch_size=1,
    num_workers=4,
    persistent_workers=True,
    drop_last=False,
    sampler=dict(type='DefaultSampler', shuffle=False),
    dataset=dict(
        type='ConcatDataset',
        datasets=[icdar2013_textrecog_test, icdar2015_textrecog_test],
        pipeline=test_pipeline))
test_dataloader = val_dataloader
```

6.2.6 Model-related Configuration

Network Configuration

This section configures the network architecture. Different algorithmic tasks use different network architectures. Find more info about network architecture in *structures*

Text Detection

Text detection consists of several parts:

- `data_preprocessor`: data_preprocessor
- `backbone`: backbone network configuration
- `neck`: neck network configuration
- `det_head`: detection head network configuration
  - `module_loss`: module loss configuration
  - `postprocessor`: postprocessor configuration

We present the model configuration in text detection using DBNet as an example.

```
model = dict(
    type='DBNet',
    data_preprocessor=dict(
        type='TextDetDataPreprocessor',
        mean=[123.675, 116.28, 103.53],
        std=[58.395, 57.12, 57.375],
        bgr_to_rgb=True,
        pad_size_divisor=32)
    backbone=dict(
        type='mmdet.ResNet',
        depth=18,
        num_stages=4,
        out_indices=(0, 1, 2, 3),
        frozen_stages=-1,
        norm_cfg=dict(type='BN', requires_grad=True),
        init_cfg=dict(type='Pretrained', checkpoint='torchvision://resnet18'),
        norm_eval=False,
        style='caffe'),
    neck=dict(
        type='FPNC', in_channels=[64, 128, 256, 512], lateral_channels=256),
    det_head=dict(
        type='DBHead',
        in_channels=256,
        module_loss=dict(type='DBModuleLoss'),
        postprocessor=dict(type='DBPostprocessor', text_repr_type='quad')))
```

Text Recognition

Text recognition mainly contains:

- `data_processor`: data preprocessor configuration
- `preprocessor`: network preprocessor configuration, e.g. TPS
- `backbone`: backbone configuration
- `encoder`: encoder configuration
- `decoder`: decoder configuration
  - `module_loss`: decoder module loss configuration

  - `postprocessor`: decoder postprocessor configuration
  - `dictionary`: dictionary configuration

Using CRNN as an example.

```
# model
model = dict(
   type='CRNN',
   data_preprocessor=dict(
        type='TextRecogDataPreprocessor', mean=[127], std=[127])
    preprocessor=None,
    backbone=dict(type='VeryDeepVgg', leaky_relu=False, input_channels=1),
    encoder=None,
    decoder=dict(
        type='CRNNDecoder',
        in_channels=512,
        rnn_flag=True,
        module_loss=dict(type='CTCModuleLoss', letter_case='lower'),
        postprocessor=dict(type='CTCPostProcessor'),
        dictionary=dict(
            type='Dictionary',
            dict_file='dicts/lower_english_digits.txt',
            with_padding=True)))
```

Checkpoint Loading Configuration

The model weights in the checkpoint file can be loaded via the `load_from` parameter, simply by setting the `load_from` parameter to the path of the checkpoint file.

You can also resume training by setting `resume=True` to load the training status information in the checkpoint. When both `load_from` and `resume=True` are set, MMEngine will load the training state from the checkpoint file at the `load_from` path.

If only `resume=True` is set, the executor will try to find and read the latest checkpoint file from the `work_dir` folder

```
load_from = None # Path to load checkpoint
resume = False # whether resume
```

More can be found in MMEngine: Load Weights or Recover Training and *OCR Advanced Tips - Resume Training from Checkpoints*.

6.2.7 Evaluation Configuration

In model validation and model testing, quantitative measurement of model accuracy is often required. MMOCR performs this function by means of `Metric` and `Evaluator`. For more information, please refer to MMEngine: Evaluation and *Evaluation*

Evaluator

Evaluator is mainly used to manage multiple datasets and multiple `Metrics`. For single and multiple dataset cases, there are single and multiple dataset evaluators, both of which can manage multiple `Metrics`.

The single-dataset evaluator is configured as follows.

```
# Single Dataset Single Metric
val_evaluator = dict(
    type='Evaluator',
    metrics=dict())

# Single Dataset Multiple Metric
val_evaluator = dict(
    type='Evaluator',
    metrics=[...])
```

`MultiDatasetsEvaluator` differs from single-dataset evaluation in two aspects: `type` and `dataset_prefixes`. The evaluator type must be `MultiDatasetsEvaluator` and cannot be omitted. The `dataset_prefixes` is mainly used to distinguish the results of different datasets with the same evaluation metrics, see *MultiDatasetsEvaluation*.

Assuming that we need to test accuracy on IC13 and IC15 datasets, the configuration is as follows.

```
#  Multiple datasets, single Metric
val_evaluator = dict(
    type='MultiDatasetsEvaluator',
    metrics=dict(),
    dataset_prefixes=['IC13', 'IC15'])

# Multiple datasets, multiple Metrics
val_evaluator = dict(
    type='MultiDatasetsEvaluator',
    metrics=[...],
    dataset_prefixes=['IC13', 'IC15'])
```

Metric

A metric evaluates a model's performance from a specific perspective. While there is no such common metric that fits all the tasks, MMOCR provides enough flexibility such that multiple metrics serving the same task can be used simultaneously. Here we list task-specific metrics for reference.

Text detection: *HmeanIOUMetric*

Text recognition: *WordMetric*, *CharMetric*, *OneMinusNEDMetric*

Key information extraction: *F1Metric*

Text detection as an example, using a single `Metric` in the case of single dataset evaluation.

```
val_evaluator = dict(type='HmeanIOUMetric')
```

Take text recognition as an example, multiple datasets (`IC13` and `IC15`) are evaluated using multiple `Metrics` (`WordMetric` and `CharMetric`).

```
val_evaluator = dict(
    type='MultiDatasetsEvaluator',
    metrics=[
        dict(
            type='WordMetric',
            mode=['exact', 'ignore_case', 'ignore_case_symbol']),
        dict(type='CharMetric')
    ],
    dataset_prefixes=['IC13', 'IC15'])
test_evaluator = val_evaluator
```

6.2.8 Visualization Configuration

Each task is bound to a task-specific visualizer. The visualizer is mainly used for visualizing or storing intermediate results of user models and visualizing val and test prediction results. The visualization results can also be stored in different backends such as WandB, TensorBoard, etc. through the corresponding visualization backend. Commonly used modification operations can be found in *visualization*.

The default configuration of visualization for text detection is as follows.

```
vis_backends = [dict(type='LocalVisBackend')]
visualizer = dict(
    type='TextDetLocalVisualizer',  # Different visualizers for different tasks
    vis_backends=vis_backends,
    name='visualizer')
```

6.3 Directory Structure

All configuration files of `MMOCR` are placed under the `configs` folder. To avoid config files from being too long and improve their reusability and clarity, MMOCR takes advantage of the inheritance mechanism and split config files into eight sections. Since each section is closely related to the task type, MMOCR provides a task folder for each task in `configs/`, namely `textdet` (text detection task), `textrecog` (text recognition task), and `kie` (key information extraction). Each folder is further divided into two parts: `_base_` folder and algorithm configuration folders.

1. the `_base_` folder stores some general config files unrelated to specific algorithms, and each section is divided into datasets, training strategies and runtime configurations by directory.
2. The algorithm configuration folder stores config files that are strongly related to the algorithm. The algorithm configuration folder has two kinds of config files.
    1. Config files starting with `_base_`: Configures the model and data pipeline of an algorithm. In OCR domain, data augmentation strategies are generally strongly related to the algorithm, so the model and data pipeline are usually placed in the same config file.
    2. Other config files, i.e. the algorithm-specific configurations on the specific dataset(s): These are the full config files that further configure training and testing settings, aggregating `_base_` configurations that are scattered in different locations. Inside some modifications to the fields in `_base_` configs may be performed, such as data pipeline, training strategy, etc.

All these config files are distributed in different folders according to their contents as follows:

The final directory structure is as follows.

```
configs
├── textdet
│   ├── _base_
│   │   ├── datasets
│   │   │   ├── icdar2015.py
│   │   │   ├── icdar2017.py
│   │   │   └── totaltext.py
│   │   ├── schedules
│   │   │   └── schedule_adam_600e.py
│   │   └── default_runtime.py
│   └── dbnet
│       ├── _base_dbnet_resnet18_fpnc.py
│       └── dbnet_resnet18_fpnc_1200e_icdar2015.py
├── textrecog
│   ├── _base_
│   │   ├── datasets
│   │   │   ├── icdar2015.py
│   │   │   ├── icdar2017.py
│   │   │   └── totaltext.py
│   │   ├── schedules
│   │   │   └── schedule_adam_base.py
│   │   └── default_runtime.py
│   └── crnn
│       ├── _base_crnn_mini-vgg.py
│       └── crnn_mini-vgg_5e_mj.py
└── kie
    ├── _base_
    │   ├──datasets
    │   └── default_runtime.py
    └── sgdmr
        └── sdmgr_novisual_60e_wildreceipt_openset.py
```

6.4 Naming Conventions

MMOCR has a convention to name config files, and contributors to the code base need to follow the same naming rules. The file names are divided into four sections: algorithm information, module information, training information, and data information. Words that logically belong to different sections are connected by an underscore `'_'`, and multiple words in the same section are connected by a hyphen `'-'`.

```
{{algorithm info}}_{{module info}}_{{training info}}_{{data info}}.py
```

- algorithm info: the name of the algorithm, such as dbnet, crnn, etc.
- module info: list some intermediate modules in the order of data flow. Its content depends on the algorithm, and some modules strongly related to the model will be omitted to avoid an overly long name. For example:
  - For the text detection task and the key information extraction task :

    ```
    {{algorithm info}}_{{backbone}}_{{neck}}_{{head}}_{{training info}}_{{data info}
    ↪}.py
    ```

    `{head}` is usually omitted since it's algorithm-specific.

  - For text recognition tasks.

```
{{algorithm info}}_{{backbone}}_{{encoder}}_{{decoder}}_{{training info}}_{
↪{data info}}.py
```

    Since encoder and decoder are generally bound to the algorithm, they are usually omitted.

- training info: some settings of the training strategy, including batch size, schedule, etc.
- data info: dataset name, modality, input size, etc., such as icdar2015 and synthtext.

CHAPTER

SEVEN

DATASET PREPARATION

7.1 Introduction

After decades of development, the OCR community has produced a series of related datasets that often provide annotations of text in a variety of styles, making it necessary for users to convert these datasets to the required format when using them. MMOCR supports dozens of commonly used text-related datasets and provides a *data preparation script* to help users prepare the datasets with only one command.

In this section, we will introduce a typical process of preparing a dataset for MMOCR:

1. *Download datasets and convert its format to the suggested one*
2. *Modify the config file*

However, the first step is not necessary if you already have a dataset in the format that MMOCR supports. You can read *Dataset Classes* for more details.

7.2 Downloading Datasets and Converting Format

As an example of the data preparation steps, you can use the following command to prepare the ICDAR 2015 dataset for text detection task.

```
python tools/dataset_converters/prepare_dataset.py icdar2015 --task textdet
```

Then, the dataset has been downloaded and converted to MMOCR format, and the file directory structure is as follows:

```
data/icdar2015
├── textdet_imgs
│   ├── test
│   └── train
├── textdet_test.json
└── textdet_train.json
```

Once your dataset has been prepared, you can use the *browse_dataset.py* to visualize the dataset and check if the annotations are correct.

```
python tools/analysis_tools/browse_dataset.py configs/textdet/_base_/datasets/icdar2015.
↪py
```

7.3 Dataset Configuration

7.3.1 Single Dataset Training

When training or evaluating a model on new datasets, we need to write the dataset config where the image path, annotation path, and image prefix are set. The path `configs/xxx/_base_/datasets/` is pre-configured with the commonly used datasets in MMOCR (if you use `prepare_dataset.py` to prepare dataset, this config will be generated automatically), here we take the ICDAR 2015 dataset as an example (see `configs/textdet/_base_/datasets/icdar2015.py`).

```
icdar2015_textdet_data_root = 'data/icdar2015' # dataset root path

# Train set config
icdar2015_textdet_train = dict(
    type='OCRDataset',
    data_root=icdar2015_textdet_data_root,                # dataset root path
    ann_file='textdet_train.json',                        # name of annotation
    filter_cfg=dict(filter_empty_gt=True, min_size=32),  # filtering empty images
    pipeline=None)
# Test set config
icdar2015_textdet_test = dict(
    type='OCRDataset',
    data_root=icdar2015_textdet_data_root,
    ann_file='textdet_test.json',
    test_mode=True,
    pipeline=None)
```

After configuring the dataset, we can import it in the corresponding model configs. For example, to train the "DBNet_R18" model on the ICDAR 2015 dataset.

```
_base_ = [
    '_base_dbnet_r18_fpnc.py',
    '../_base_/datasets/icdar2015.py',  # import the dataset config
    '../_base_/default_runtime.py',
    '../_base_/schedules/schedule_sgd_1200e.py',
]

icdar2015_textdet_train = _base_.icdar2015_textdet_train            # specify the␣
↪training set
icdar2015_textdet_train.pipeline = _base_.train_pipeline   # specify the training␣
↪pipeline
icdar2015_textdet_test = _base_.icdar2015_textdet_test              # specify the␣
↪testing set
icdar2015_textdet_test.pipeline = _base_.test_pipeline     # specify the testing pipeline

train_dataloader = dict(
    batch_size=16,
    num_workers=8,
    persistent_workers=True,
    sampler=dict(type='DefaultSampler', shuffle=True),
    dataset=icdar2015_textdet_train)    # specify the dataset in train_dataloader

val_dataloader = dict(
```

(continues on next page)

(continued from previous page)

```
    batch_size=1,
    num_workers=4,
    persistent_workers=True,
    sampler=dict(type='DefaultSampler', shuffle=False),
    dataset=icdar2015_textdet_test)    # specify the dataset in val_dataloader

test_dataloader = val_dataloader
```

7.3.2 Multi-dataset Training

In addition, *ConcatDataset* enables users to train or test the model on a combination of multiple datasets. You just need to set the dataset type in the dataloader to `ConcatDataset` in the configuration file and specify the corresponding list of datasets.

```
train_list = [ic11, ic13, ic15]
train_dataloader = dict(
    dataset=dict(
        type='ConcatDataset', datasets=train_list, pipeline=train_pipeline))
```

For example, the following configuration uses the MJSynth dataset for training and 6 academic datasets (CUTE80, IIIT5K, SVT, SVTP, ICDAR2013, ICDAR2015) for testing.

```
_base_ = [ # Import all dataset configurations you want to use
    '../_base_/datasets/mjsynth.py',
    '../_base_/datasets/cute80.py',
    '../_base_/datasets/iiit5k.py',
    '../_base_/datasets/svt.py',
    '../_base_/datasets/svtp.py',
    '../_base_/datasets/icdar2013.py',
    '../_base_/datasets/icdar2015.py',
    '../_base_/default_runtime.py',
    '../_base_/schedules/schedule_adadelta_5e.py',
    '_base_crnn_mini-vgg.py',
]

# List of training datasets
train_list = [_base_.mjsynth_textrecog_train]
# List of testing datasets
test_list = [
    _base_.cute80_textrecog_test, _base_.iiit5k_textrecog_test, _base_.svt_textrecog_
↪test,
    _base_.svtp_textrecog_test, _base_.icdar2013_textrecog_test, _base_.icdar2015_
↪textrecog_test
]

# Use ConcatDataset to combine the datasets in the list
train_dataset = dict(
       type='ConcatDataset', datasets=train_list, pipeline=_base_.train_pipeline)
test_dataset = dict(
       type='ConcatDataset', datasets=test_list, pipeline=_base_.test_pipeline)
```

(continues on next page)

(continued from previous page)

```
train_dataloader = dict(
    batch_size=192 * 4,
    num_workers=32,
    persistent_workers=True,
    sampler=dict(type='DefaultSampler', shuffle=True),
    dataset=train_dataset)

test_dataloader = dict(
    batch_size=1,
    num_workers=4,
    persistent_workers=True,
    drop_last=False,
    sampler=dict(type='DefaultSampler', shuffle=False),
    dataset=test_dataset)

val_dataloader = test_dataloader
```

CHAPTER

EIGHT

TRAINING AND TESTING

To meet diverse requirements, MMOCR supports training and testing models on various devices, including PCs, work stations, computation clusters, etc.

8.1 Single GPU Training and Testing

8.1.1 Training

`tools/train.py` provides the basic training service. MMOCR recommends using GPUs for model training and testing, but it still enables CPU-Only training and testing. For example, the following commands demonstrate how to train a DBNet model using a single GPU or CPU.

```
# Train the specified MMOCR model by calling tools/train.py
CUDA_VISIBLE_DEVICES= python tools/train.py ${CONFIG_FILE} [PY_ARGS]

# Training
# Example 1: Training DBNet with CPU
CUDA_VISIBLE_DEVICES=-1 python tools/train.py configs/textdet/dbnet/dbnet_resnet50-dcnv2_
↪fpnc_1200e_icdar2015.py

# Example 2: Specify to train DBNet with gpu:0, specify the working directory as dbnet/,␣
↪and turn on mixed precision (amp) training
CUDA_VISIBLE_DEVICES=0 python tools/train.py configs/textdet/dbnet/dbnet_resnet50-dcnv2_
↪fpnc_1200e_icdar2015.py --work-dir dbnet/ --amp
```

Note: If multiple GPUs are available, you can specify a certain GPU, e.g. the third one, by setting CUDA_VISIBLE_DEVICES=3.

The following table lists all the arguments supported by `train.py`. Args without the `--` prefix are mandatory, while others are optional.

8.1.2 Test

`tools/test.py` provides the basic testing service, which is used in a similar way to the training script. For example, the following command demonstrates test a DBNet model on a single GPU or CPU.

```
# Test a pretrained MMOCR model by calling tools/test.py
CUDA_VISIBLE_DEVICES= python tools/test.py ${CONFIG_FILE} ${CHECKPOINT_FILE} [PY_ARGS]

# Test
# Example 1: Testing DBNet with CPU
CUDA_VISIBLE_DEVICES=-1 python tools/test.py configs/textdet/dbnet/dbnet_resnet50-dcnv2_
↪fpnc_1200e_icdar2015.py dbnet_r50.pth

# Example 2: Testing DBNet on gpu:0
CUDA_VISIBLE_DEVICES=0 python tools/test.py configs/textdet/dbnet/dbnet_resnet50-dcnv2_
↪fpnc_1200e_icdar2015.py dbnet_r50.pth
```

The following table lists all the arguments supported by `test.py`. Args without the `--` prefix are mandatory, while others are optional.

8.2 Training and Testing with Multiple GPUs

For large models, distributed training or testing significantly improves the efficiency. For this purpose, MMOCR provides distributed scripts `tools/dist_train.sh` and `tools/dist_test.sh` implemented based on MMDistributedDataParallel.

```
# Training
NNODES=${NNODES} NODE_RANK=${NODE_RANK} PORT=${MASTER_PORT} MASTER_ADDR=${MASTER_ADDR} ./
↪tools/dist_train.sh ${CONFIG_FILE} ${GPU_NUM} [PY_ARGS]

# Testing
NNODES=${NNODES} NODE_RANK=${NODE_RANK} PORT=${MASTER_PORT} MASTER_ADDR=${MASTER_ADDR} ./
↪tools/dist_test.sh ${CONFIG_FILE} ${CHECKPOINT_FILE} ${GPU_NUM} [PY_ARGS]
```

The following table lists the arguments supported by `dist_*.sh`.

These two scripts enable training and testing on **single-machine multi-GPU** or **multi-machine multi-GPU**. See the following example for usage.

8.2.1 Single-machine Multi-GPU

The following commands demonstrate how to train and test with a specified number of GPUs on a **single machine** with multiple GPUs.

1. **Training**

   Training DBNet using 4 GPUs on a single machine.

   ```
   tools/dist_train.sh configs/textdet/dbnet/dbnet_r50dcnv2_fpnc_1200e_icdar2015.py 4
   ```

2. **Testing**

   Testing DBNet using 4 GPUs on a single machine.

```
tools/dist_test.sh configs/textdet/dbnet/dbnet_r50dcnv2_fpnc_1200e_icdar2015.py␣
↪dbnet_r50.pth 4
```

8.2.2 Launching Multiple Tasks on Single Machine

For a workstation equipped with multiple GPUs, the user can launch multiple tasks simultaneously by specifying the GPU IDs. For example, the following command demonstrates how to test DBNet with GPU `[0, 1, 2, 3]` and train CRNN on GPU `[4, 5, 6, 7]`.

```
# Specify gpu:0,1,2,3 for testing and assign port number 29500
CUDA_VISIBLE_DEVICES=0,1,2,3 PORT=29500 ./tools/dist_test.sh configs/textdet/dbnet/dbnet_
↪r50dcnv2_fpnc_1200e_icdar2015.py dbnet_r50.pth 4

# Specify gpu:4,5,6,7 for training and assign port number 29501
CUDA_VISIBLE_DEVICES=4,5,6,7 PORT=29501 ./tools/dist_train.sh configs/textrecog/crnn/
↪crnn_academic_dataset.py 4
```

Note: `dist_train.sh` sets `MASTER_PORT` to `29500` by default. When other processes already occupy this port, the program will get a runtime error `RuntimeError: Address already in use`. In this case, you need to set `MASTER_PORT` to another free port number in the range of `(0~65535)`.

8.2.3 Multi-machine Multi-GPU Training and Testing

You can launch a task on multiple machines connected to the same network. MMOCR relies on `torch.distributed` package for distributed training. Find more information at PyTorch's launch utility.

1. **Training**

   The following command demonstrates how to train DBNet on two machines with a total of 4 GPUs.

   ```
   # Say that you want to launch the training job on two machines
   # On the first machine:
   NNODES=2 NODE_RANK=0 PORT=29500 MASTER_ADDR=10.140.0.169 tools/dist_train.sh␣
   ↪configs/textdet/dbnet/dbnet_r50dcnv2_fpnc_1200e_icdar2015.py 2
   # On the second machine:
   NNODES=2 NODE_RANK=1 PORT=29501 MASTER_ADDR=10.140.0.169 tools/dist_train.sh␣
   ↪configs/textdet/dbnet/dbnet_r50dcnv2_fpnc_1200e_icdar2015.py 2
   ```

2. **Testing**

   The following command demonstrates how to test DBNet on two machines with a total of 4 GPUs.

   ```
   # Say that you want to launch the testing job on two machines
   # On the first machine:
   NNODES=2 NODE_RANK=0 PORT=29500 MASTER_ADDR=10.140.0.169 tools/dist_test.sh configs/
   ↪textdet/dbnet/dbnet_r50dcnv2_fpnc_1200e_icdar2015.py dbnet_r50.pth 2
   # On the second machine:
   NNODES=2 NODE_RANK=1 PORT=29501 MASTER_ADDR=10.140.0.169 tools/dist_test.sh configs/
   ↪textdet/dbnet/dbnet_r50dcnv2_fpnc_1200e_icdar2015.py dbnet_r50.pth 2
   ```

Note: The speed of the network could be the bottleneck of training.

8.3 Training and Testing with Slurm Cluster

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

```
# tools/slurm_train.sh provides scripts for submitting training tasks on clusters␣
↪managed by the slurm
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]

# tools/slurm_test.sh provides scripts for submitting testing tasks on clusters managed␣
↪by the slurm
GPUS=${GPUS} GPUS_PER_NODE=${GPUS_PER_NODE} CPUS_PER_TASK=${CPUS_PER_TASK} SRUN_ARGS=$
↪{SRUN_ARGS} ./tools/slurm_test.sh ${PARTITION} ${JOB_NAME} ${CONFIG_FILE} ${CHECKPOINT_
↪FILE} ${WORK_DIR} [PY_ARGS]
```

These scripts enable training and testing on slurm clusters, see the following examples.

1. Training

   Here is an example of using 1 GPU to train a DBNet model on the dev partition.

   ```
   # Example: Request 1 GPU resource on dev partition for DBNet training task
   GPUS=1 GPUS_PER_NODE=1 CPUS_PER_TASK=5 tools/slurm_train.sh dev db_r50 configs/
   ↪textdet/dbnet/dbnet_r50dcnv2_fpnc_1200e_icdar2015.py work_dir
   ```

2. Testing

   Similarly, the following example requests 1 GPU for testing.

   ```
   # Example: Request 1 GPU resource on dev partition for DBNet testing task
   GPUS=1 GPUS_PER_NODE=1 CPUS_PER_TASK=5 tools/slurm_test.sh dev db_r50 configs/
   ↪textdet/dbnet/dbnet_r50dcnv2_fpnc_1200e_icdar2015.py dbnet_r50.pth work_dir
   ```

8.4 Advanced Tips

8.4.1 Resume Training from a Checkpoint

`tools/train.py` allows users to resume training from a checkpoint by specifying the `--resume` parameter, where it will automatically resume training from the latest saved checkpoint.

```
# Example: Resuming training from the latest checkpoint
python tools/train.py configs/textdet/dbnet/dbnet_r50dcnv2_fpnc_1200e_icdar2015.py 4 --
↪resume
```

By default, the program will automatically resume training from the last successfully saved checkpoint in the last training session, i.e. `latest.pth`. However,

```
# Example: Set the path of the checkpoint you want to load in the configuration file
load_from = 'work_dir/dbnet/models/epoch_10000.pth'
```

8.4.2 Mixed Precision Training

Mixed precision training offers significant computational speedup by performing operations in half-precision format, while storing minimal information in single-precision to retain as much information as possible in critical parts of the network. In MMOCR, the users can enable the automatic mixed precision training by simply add `--amp`.

```
# Example: Using automatic mixed precision training
python tools/train.py configs/textdet/dbnet/dbnet_r50dcnv2_fpnc_1200e_icdar2015.py 4 --
↪amp
```

The following table shows the support of each algorithm in MMOCR for automatic mixed precision training.

8.4.3 Automatic Learning Rate Scaling

MMOCR sets default initial learning rates for each model in the configuration file. However, these initial learning rates may not be applicable when the user uses a different `batch_size` than our preset `base_batch_size`. Therefore, we provide a tool to automatically scale the learning rate, which can be called by adding the `--auto-scale-lr`.

```
# Example: Using automatic learning rate scaling
python tools/train.py configs/textdet/dbnet/dbnet_r50dcnv2_fpnc_1200e_icdar2015.py 4 --
↪auto-scale-lr
```

8.4.4 Visualize the Predictions

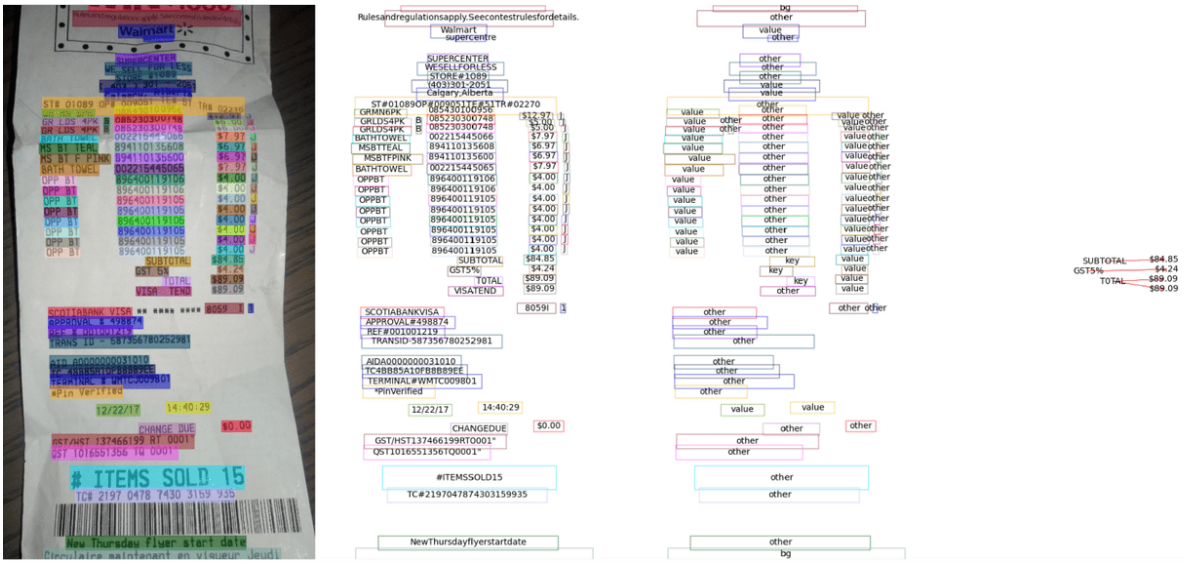
`tools/test.py` provides the visualization interface to facilitate the qualitative analysis of the OCR models.

(Green boxes are GTs, while red boxes are predictions)

GABBANA

GABBANA

(Green font is the GT, red font is the prediction)



(From left to right: original image, text detection and recognition result, text classification result, relationship)

```
# Example 1: Show the visualization results per 2 seconds
python tools/test.py configs/textdet/dbnet/dbnet_r50dcnv2_fpnc_1200e_icdar2015.py dbnet_
↪r50.pth --show --wait-time 2

# Example 2: For systems that do not support graphical interfaces (such as computing␣
↪clusters, etc.), the visualization results can be dumped in the specified path
python tools/test.py configs/textdet/dbnet/dbnet_r50dcnv2_fpnc_1200e_icdar2015.py dbnet_
↪r50.pth --show-dir ./vis_results
```

The visualization-related parameters in `tools/test.py` are described as follows.

8.4.5 Test Time Augmentation

Test time augmentation (TTA) is a technique that is used to improve the performance of a model by performing data augmentation on the input image at test time. It is a simple yet effective method to improve the performance of a model. In MMOCR, we support TTA in the following ways:

Note: TTA is only supported for text recognition models.

```
python tools/test.py configs/textrecog/crnn/crnn_mini-vgg_5e_mj.py checkpoints/crnn_mini-
↪vgg_5e_mj.pth --tta
```

CHAPTER

NINE

VISUALIZATION

Before reading this tutorial, it is recommended to read MMEngine's MMEngine: Visualization documentation to get a first glimpse of the `Visualizer` definition and usage.

In brief, the `Visualizer` is implemented in MMEngine to meet the daily visualization needs, and contains three main functions:

- Implement common drawing APIs, such as `draw_bboxes` which implements bounding box drawing functions, `draw_lines` implements the line drawing function.
- Support writing visualization results, learning rate curves, loss function curves, and verification accuracy curves to various backends, including local disks and common deep learning training logging tools such as TensorBoard and Wandb.
- Support calling anywhere in the code to visualize or record intermediate states of the model during training or testing, such as feature maps and validation results.

Based on MMEngine's Visualizer, MMOCR comes with a variety of pre-built visualization tools that can be used by the user by simply modifying the following configuration files.

- The `tools/analysis_tools/browse_dataset.py` script provides a dataset visualization function that draws images and corresponding annotations after Data Transforms, as described in *`browse_dataset.py`*.
- MMEngine implements `LoggerHook`, which uses `Visualizer` to write the learning rate, loss and evaluation results to the backend set by `Visualizer`. Therefore, by modifying the `Visualizer` backend in the configuration file, for example to `TensorBoardVISBackend` or `WandbVISBackend`, you can implement logging to common training logging tools such as `TensorBoard` or `WandB`, thus making it easy for users to use these visualization tools to analyze and monitor the training process.
- The `VisualizerHook` is implemented in MMOCR, which uses the `Visualizer` to visualize or store the prediction results of the validation or prediction phase into the backend set by the `Visualizer`, so by modifying the `Visualizer` backend in the configuration file, for example, to `TensorBoardVISBackend` or `WandbVISBackend`, you can implement storing the predicted images to `TensorBoard` or `Wandb`.

9.1 Configuration

Thanks to the use of the registration mechanism, in MMOCR we can set the behavior of the `Visualizer` by modifying the configuration file. Usually, we define the default configuration for the visualizer in `task/_base_/default_runtime.py`, see *configuration tutorial* for details.

```
vis_backends = [dict(type='LocalVisBackend')]
visualizer = dict(
    type='TextxxxLocalVisualizer', # use different visualizers for different tasks
```

(continues on next page)

(continued from previous page)

```
    vis_backends=vis_backends,
    name='visualizer')
```

Based on the above example, we can see that the configuration of `Visualizer` consists of two main parts, namely, the type of `Visualizer` and the visualization backend `vis_backends` it uses.

- For different OCR tasks, various visualizers are pre-configured in MMOCR, including *TextDetLocalVisualizer*, *TextRecogLocalVisualizer*, *TextSpottingLocalVisualizer* and *KIELocalVisualizer*. These visualizers extend the basic Visulizer API according to the characteristics of their tasks and implement the corresponding tag information interface `add_datasamples`. For example, users can directly use `TextDetLocalVisualizer` to visualize labels or predictions for text detection tasks.
- MMOCR sets the visualization backend `vis_backend` to the local visualization backend `LocalVisBackend` by default, saving all visualization results and other training information in a local folder.

9.2 Storage

MMOCR uses the local visualization backend `LocalVisBackend` by default, and the model loss, learning rate, model evaluation accuracy and visualization The information stored in `VisualizerHook` and `LoggerHook`, including loss, learning rate, evaluation accuracy will be saved to the `{work_dir}/{config_name}/{time}/{vis_data}` folder by default. In addition, MMOCR also supports other common visualization backends, such as `TensorboardVisBackend` and `WandbVisBackend`, and you only need to change the `vis_backends` type in the configuration file to the corresponding visualization backend. For example, you can store data to `TensorBoard` and `Wandb` by simply inserting the following code block into the configuration file.

```
_base_.visualizer.vis_backends = [
    dict(type='LocalVisBackend'),
    dict(type='TensorboardVisBackend'),
    dict(type='WandbVisBackend'),]
```

9.3 Plot

9.3.1 Plot the prediction results

MMOCR mainly uses *VisualizationHook* to plot the prediction results of validation and test, by default `VisualizationHook` is off, and the default configuration is as follows.

```
visualization=dict( # user visualization of validation and test results
    type='VisualizationHook',
    enable=False,
    interval=1,
    show=False,
    draw_gt=False,
    draw_pred=False)
```

The following table shows the parameters supported by `VisualizationHook`.

If you want to enable `VisualizationHook` related functions and configurations during training or testing, you only need to modify the configuration, take `dbnet_resnet18_fpnc_1200e_icdar2015.py` as an example, draw annotations and predictions at the same time, and display the images, the configuration can be modified as follows

```
visualization = _base_.default_hooks.visualization
visualization.update(
    dict(enable=True, show=True, draw_gt=True, draw_pred=True))
```

If you only want to see the predicted result information you can just let `draw_pred=True`

```
visualization = _base_.default_hooks.visualization
visualization.update(
    dict(enable=True, show=True, draw_gt=False, draw_pred=True))
```

The `test.py` procedure is further simplified by providing the `--show` and `--show-dir` parameters to visualize the annotation and prediction results during the test without modifying the configuration.

```
# Show test results
python tools/test.py configs/textdet/dbnet/dbnet_resnet18_fpnc_1200e_icdar2015.py dbnet_
↪r18_fpnc_1200e_icdar2015/epoch_400.pth --show

# Specify where to store the prediction results
python tools/test.py configs/textdet/dbnet/dbnet_resnet18_fpnc_1200e_icdar2015.py dbnet_
↪r18_fpnc_1200e_icdar2015/epoch_400.pth --show-dir imgs/
```

CHAPTER

TEN

USEFUL TOOLS

10.1 Visualization Tools

10.1.1 Dataset Visualization Tool

MMOCR provides a dataset visualization tool `tools/visualizations/browse_datasets.py` to help users troubleshoot possible dataset-related problems. You just need to specify the path to the training config (usually stored in `configs/textdet/dbnet/xxx.py`) or the dataset config (usually stored in `configs/textdet/_base_/datasets/xxx.py`), and the tool will automatically plots the transformed (or original) images and labels.

Usage

```
python tools/visualizations/browse_dataset.py \
    ${CONFIG_FILE} \
    [-o, --output-dir ${OUTPUT_DIR}] \
    [-p, --phase ${DATASET_PHASE}] \
    [-m, --mode ${DISPLAY_MODE}] \
    [-t, --task ${DATASET_TASK}] \
    [-n, --show-number ${NUMBER_IMAGES_DISPLAY}] \
    [-i, --show-interval ${SHOW_INTERRVAL}] \
    [--cfg-options ${CFG_OPTIONS}]
```

Examples

The following example demonstrates how to use the tool to visualize the training data used by the "DBNet_R50_icdar2015" model.

```
# Example: Visualizing the training data used by dbnet_r50dcn_v2_fpnc_1200e_icadr2015␣
↪model
python tools/visualizations/browse_dataset.py configs/textdet/dbnet/dbnet_resnet50-dcnv2_
↪fpnc_1200e_icdar2015.py
```

By default, the visualization mode is "transformed", and you will see the images & annotations being transformed by the pipeline:

If you just want to visualize the original dataset, simply set the mode to "original":

```
python tools/visualizations/browse_dataset.py configs/textdet/dbnet/dbnet_resnet50-dcnv2_
↪fpnc_1200e_icdar2015.py -m original
```

Or, to visualize the entire pipeline:

```
python tools/visualizations/browse_dataset.py configs/textdet/dbnet/dbnet_resnet50-dcnv2_
↪fpnc_1200e_icdar2015.py -m pipeline
```

In addition, users can also visualize the original images and their corresponding labels of the dataset by specifying the path to the dataset config file, for example:

```
python tools/visualizations/browse_dataset.py configs/textrecog/_base_/datasets/
↪icdar2015.py
```

Some datasets might have multiple variants. For example, the test split of `icdar2015` textrecog dataset has two variants, which the base dataset config defines as follows:

```
icdar2015_textrecog_test = dict(
    ann_file='textrecog_test.json',
    # ...
    )

icdar2015_1811_textrecog_test = dict(
    ann_file='textrecog_test_1811.json',
    # ...
)
```

In this case, you can specify the variant name to visualize the corresponding dataset:

```
python tools/visualizations/browse_dataset.py configs/textrecog/_base_/datasets/
↪icdar2015.py -p icdar2015_1811_textrecog_test
```

Based on this tool, users can easily verify if the annotation of a custom dataset is correct.

10.1.2 Hyper-parameter Scheduler Visualization

This tool aims to help the user to check the hyper-parameter scheduler of the optimizer (without training), which support the "learning rate" or "momentum"

Introduce the scheduler visualization tool

```
python tools/visualizations/vis_scheduler.py \
    ${CONFIG_FILE} \
    [-p, --parameter ${PARAMETER_NAME}] \
    [-d, --dataset-size ${DATASET_SIZE}] \
    [-n, --ngpus ${NUM_GPUs}] \
    [-s, --save-path ${SAVE_PATH}] \
    [--title ${TITLE}] \
    [--style ${STYLE}] \
    [--window-size ${WINDOW_SIZE}] \
    [--cfg-options]
```

Description of all arguments

- `config`: The path of a model config file.

- **-p, --parameter**: The param to visualize its change curve, choose from "lr" and "momentum". Default to use "lr".
- **-d, --dataset-size**: The size of the datasets. If set`build_dataset` will be skipped and `${DATASET_SIZE}` will be used as the size. Default to use the function `build_dataset`.
- **-n, --ngpus**: The number of GPUs used in training, default to be 1.
- **-s, --save-path**: The learning rate curve plot save path, default not to save.
- `--title`: Title of figure. If not set, default to be config file name.
- `--style`: Style of plt. If not set, default to be `whitegrid`.
- `--window-size`: The shape of the display window. If not specified, it will be set to `12*7`. If used, it must be in the format `'W*H'`.
- `--cfg-options`: Modifications to the configuration file, refer to *Learn about Configs*.

Note: Loading annotations maybe consume much time, you can directly specify the size of the dataset with `-d, dataset-size` to save time.

How to plot the learning rate curve without training

You can use the following command to plot the step learning rate schedule used in the config `configs/textdet/dbnet/dbnet_resnet50-dcnv2_fpnc_1200e_icdar2015.py`:

```
python tools/visualizations/vis_scheduler.py configs/textdet/dbnet/dbnet_resnet50-dcnv2_
↪fpnc_1200e_icdar2015.py -d 100
```

10.2 Analysis Tools

10.2.1 Offline Evaluation Tool

For saved prediction results, we provide an offline evaluation script `tools/analysis_tools/offline_eval.py`. The following example demonstrates how to use this tool to evaluate the output of the "PSENet" model offline.

```
# When running the test script for the first time, you can save the output of the model␣
↪by specifying the --save-preds parameter
python tools/test.py ${CONFIG_FILE} ${CHECKPOINT_FILE} --save-preds
# Example: Testing on PSENet
python tools/test.py configs/textdet/psenet/psenet_r50_fpnf_600e_icdar2015.py epoch_600.
↪pth --save-preds

# Then, using the saved outputs for offline evaluation
python tools/analysis_tool/offline_eval.py ${CONFIG_FILE} ${PRED_FILE}
# Example: Offline evaluation of saved PSENet results
python tools/analysis_tools/offline_eval.py configs/textdet/psenet/psenet_r50_fpnf_600e_
↪icdar2015.py work_dirs/psenet_r50_fpnf_600e_icdar2015/epoch_600.pth_predictions.pkl
```

`-save-preds` saves the output to `work_dir/CONFIG_NAME/MODEL_NAME_predictions.pkl` by default

In addition, based on this tool, users can also convert predictions obtained from other libraries into MMOCR-supported formats, then use MMOCR's built-in metrics to evaluate them.

10.2.2 Calculate FLOPs and the Number of Parameters

We provide a method to calculate the FLOPs and the number of parameters, first we install the dependencies using the following command.

```
pip install fvcore
```

The usage of the script to calculate FLOPs and the number of parameters is as follows.

```
python tools/analysis_tools/get_flops.py ${config} --shape ${IMAGE_SHAPE}
```

For example, you can run the following command to get FLOPs and the number of parameters of `dbnet_resnet18_fpnc_100k_synthtext.py`:

```
python tools/analysis_tools/get_flops.py configs/textdet/dbnet/dbnet_resnet18_fpnc_100k_
↪synthtext.py --shape 1024 1024
```

The output is as follows:

```
input shape is  (1, 3, 1024, 1024)
| module                     | #parameters or shape | #flops  |
| :------------------------- | :------------------- | :------ |
| model                      | 12.341M              | 63.955G |
| backbone                   | 11.177M              | 38.159G |
| backbone.conv1             | 9.408K               | 2.466G  |
| backbone.conv1.weight      | (64, 3, 7, 7)        |         |
| backbone.bn1               | 0.128K               | 83.886M |
| backbone.bn1.weight        | (64,)                |         |
| backbone.bn1.bias          | (64,)                |         |
| backbone.layer1            | 0.148M               | 9.748G  |
| backbone.layer1.0          | 73.984K              | 4.874G  |
| backbone.layer1.1          | 73.984K              | 4.874G  |
| backbone.layer2            | 0.526M               | 8.642G  |
| backbone.layer2.0          | 0.23M                | 3.79G   |
| backbone.layer2.1          | 0.295M               | 4.853G  |
| backbone.layer3            | 2.1M                 | 8.616G  |
| backbone.layer3.0          | 0.919M               | 3.774G  |
| backbone.layer3.1          | 1.181M               | 4.842G  |
| backbone.layer4            | 8.394M               | 8.603G  |
| backbone.layer4.0          | 3.673M               | 3.766G  |
| backbone.layer4.1          | 4.721M               | 4.837G  |
| neck                       | 0.836M               | 14.887G |
| neck.lateral_convs         | 0.246M               | 2.013G  |
| neck.lateral_convs.0.conv  | 16.384K              | 1.074G  |
| neck.lateral_convs.1.conv  | 32.768K              | 0.537G  |
| neck.lateral_convs.2.conv  | 65.536K              | 0.268G  |
| neck.lateral_convs.3.conv  | 0.131M               | 0.134G  |
| neck.smooth_convs          | 0.59M                | 12.835G |
| neck.smooth_convs.0.conv   | 0.147M               | 9.664G  |
| neck.smooth_convs.1.conv   | 0.147M               | 2.416G  |
| neck.smooth_convs.2.conv   | 0.147M               | 0.604G  |
| neck.smooth_convs.3.conv   | 0.147M               | 0.151G  |
| det_head                   | 0.329M               | 10.909G |
| det_head.binarize          | 0.164M               | 10.909G |
```

(continues on next page)

(continued from previous page)

```
| det_head.binarize.0       | 0.147M               | 9.664G  |
| det_head.binarize.1       | 0.128K               | 20.972M |
| det_head.binarize.3       | 16.448K              | 1.074G  |
| det_head.binarize.4       | 0.128K               | 83.886M |
| det_head.binarize.6       | 0.257K               | 67.109M |
| det_head.threshold        | 0.164M               |         |
| det_head.threshold.0      | 0.147M               |         |
| det_head.threshold.1      | 0.128K               |         |
| det_head.threshold.3      | 16.448K              |         |
| det_head.threshold.4      | 0.128K               |         |
| det_head.threshold.6      | 0.257K               |         |
!!!Please be cautious if you use the results in papers. You may need to check if all ops␣
↪are supported and verify that the flops computation is correct.
```

CHAPTER

ELEVEN

DATA STRUCTURES AND ELEMENTS

MMOCR uses MMEngine: Abstract Data Element to encapsulate the data required for each task into `data_sample`. The base class has implemented basic add/delete/update/check functions and supports data migration between different devices, as well as dictionary-like and tensor-like operations, which also allows the interfaces of different algorithms to be unified.

Thanks to the unified data structures, the data flow between each module in the algorithm libraries, such as *`visualizer`*, *`evaluator`*, *`dataset`*, is greatly simplified. In MMOCR, we have the following conventions for different data types.

- **xxxData**: Single granularity data annotation or model output. Currently MMEngine has three built-in granularities of data elements, including instance-level data (`InstanceData`), pixel-level data (`PixelData`) and image-level label data (`LabelData`). Among the tasks currently supported by MMOCR, text detection and key information extraction tasks use `InstanceData` to encapsulate the bounding boxes and the corresponding box label, while the text recognition task uses `LabelData` to encapsulate the text content.
- **xxxDataSample**: inherited from MMEngine: Base Data Element, used to hold **all** annotation and prediction information that required by a single task. For example, *`TextDetDataSample`* for the text detection, *`TextRecogDataSample`* for text recognition, and *`KIEDataSample`* for the key information extraction task.

In the following, we will introduce the practical application of data elements **xxxData** and data samples **xxxDataSample** in MMOCR, respectively.

11.1 Data Elements - xxxData

`InstanceData` and `LabelData` are the `BaseDataElement` defined in `MMEngine` to encapsulate different granularity of annotation data or model output. In MMOCR, we have used `InstanceData` and `LabelData` for encapsulating the data types actually used in OCR-related tasks.

11.1.1 InstanceData

In the **text detection** task, the detector concentrate on instance-level text samples, so we use `InstanceData` to encapsulate the data needed for this task. Typically, its required training annotation and prediction output contain rectangular or polygonal bounding boxes, as well as bounding box labels. Since the text detection task has only one positive sample class, "text", in MMOCR we use `0` to number this class by default. The following code example shows how to use the `InstanceData` to encapsulate the data used in the text detection task.

```
import torch
from mmengine.structures import InstanceData

# defining gt_instance for encapsulating the ground truth data
gt_instance = InstanceData()
```

(continues on next page)

(continued from previous page)

```
gt_instance.bbox = torch.Tensor([[0, 0, 10, 10], [10, 10, 20, 20]])
gt_instance.polygons = torch.Tensor([[[0, 0], [10, 0], [10, 10], [0, 10]],
                                     [[10, 10], [20, 10], [20, 20], [10, 20]]])
gt_instance.label = torch.Tensor([0, 0])

# defining pred_instance for encapsulating the prediction data
pred_instances = InstanceData()
pred_polygons, scores = model(input)
pred_instances.polygons = pred_polygons
pred_instances.scores = scores
```

The conventions for the fields in `InstanceData` in MMOCR are shown in the table below. It is important to note that the length of each field in `InstanceData` must be equal to the number of instances `N` in the sample.

11.1.2 LabelData

For **text recognition** tasks, both labeled content and predicted content are wrapped using `LabelData`.

```
import torch
from mmengine.data import LabelData

# defining gt_text for encapsulating the ground truth data
gt_text = LabelData()
gt_text.item = 'MMOCR'

# defining pred_text for encapsulating the prediction data
pred_text = LabelData()
index, score = model(input)
text = dictionary.idx2str(index)
pred_text.score = score
pred_text.item = text
```

The conventions for the `LabelData` fields in MMOCR are shown in the following table.

11.2 DataSample xxxDataSample

By defining a uniform data structure, we can easily encapsulate the annotation data and prediction results in a unified way, making data transfer between different modules of the code base easier. In MMOCR, we have designed three data structures based on the data needed in three tasks: *`TextDetDataSample`*, *`TextRecogDataSample`*, and *`KIEDataSample`*. These data structures all inherit from MMEngine: Base Data Element, which is used to hold all annotation and prediction information required by each task.

11.2.1 Text Detection - TextDetDataSample

TextDetDataSample is used to encapsulate the data needed for the text detection task. It contains two main fields `gt_instances` and `pred_instances`, which are used to store the annotation information and prediction results respectively.

The fields of *InstanceData* that will be used are:

Since text detection models usually only output one of the bboxes/polygons, we only need to make sure that one of these two is assigned a value.

The following sample code demonstrates the use of `TextDetDataSample`.

```
import torch
from mmengine.data import TextDetDataSample

data_sample = TextDetDataSample()
# Define the ground truth data
img_meta = dict(img_shape=(800, 1196, 3), pad_shape=(800, 1216, 3))
gt_instances = InstanceData(metainfo=img_meta)
gt_instances.bboxes = torch.rand((5, 4))
gt_instances.labels = torch.zeros((5,), dtype=torch.long)
data_sample.gt_instances = gt_instances

# Define the prediction data
pred_instances = InstanceData()
pred_instances.bboxes = torch.rand((5, 4))
pred_instances.labels = torch.zeros((5,), dtype=torch.long)
data_sample.pred_instances = pred_instances
```

11.2.2 Text Recognition - TextRecogDataSample

TextRecogDataSample is used to encapsulate the data for the text recognition task. It has two fields, `gt_text` and `pred_text` , which are used to store annotation information and prediction results, respectively.

The following sample code demonstrates the use of *TextRecogDataSample*.

```
import torch
from mmengine.data import TextRecogDataSample

data_sample = TextRecogDataSample()
# Define the ground truth data
img_meta = dict(img_shape=(800, 1196, 3), pad_shape=(800, 1216, 3))
gt_text = LabelData(metainfo=img_meta)
gt_text.item = 'mmocr'
data_sample.gt_text = gt_text

# Define the prediction data
pred_text = LabelData(metainfo=img_meta)
pred_text.item = 'mmocr'
data_sample.pred_text = pred_text
```

The fields of `LabelData` that will be used are:

11.2.3 Key Information Extraction - KIEDataSample

KIEDataSample is used to encapsulate the data needed for the KIE task. It also contains two fields, `gt_instances` and `pred_instances`, which are used to store annotation information and prediction results respectively.

The `InstanceData` fields that will be used by this task are shown in the following table.

Warning: Since there is no unified standard for model implementation of KIE tasks, the design currently considers only SDMGR model usage scenarios. Therefore, the design is subject to change as we support more KIE models.

The following sample code shows the use of *KIEDataSample*.

```
import torch
from mmengine.data import KIEDataSample

data_sample = KIEDataSample()
# Define the ground truth data
img_meta = dict(img_shape=(800, 1196, 3),pad_shape=(800, 1216, 3))
gt_instances = InstanceData(metainfo=img_meta)
gt_instances.bboxes = torch.rand((5, 4))
gt_instances.labels = torch.zeros((5,), dtype=torch.long)
gt_instances.texts = ['text1', 'text2', 'text3', 'text4', 'text5']
gt_instances.edge_lebels = torch.randint(-1, 2, (5, 5))
data_sample.gt_instances = gt_instances

# Define the prediction data
pred_instances = InstanceData()
pred_instances.bboxes = torch.rand((5, 4))
pred_instances.labels = torch.rand((5,))
pred_instances.edge_labels = torch.randint(-1, 2, (10, 10))
pred_instances.edge_scores = torch.rand((10, 10))
data_sample.pred_instances = pred_instances
```

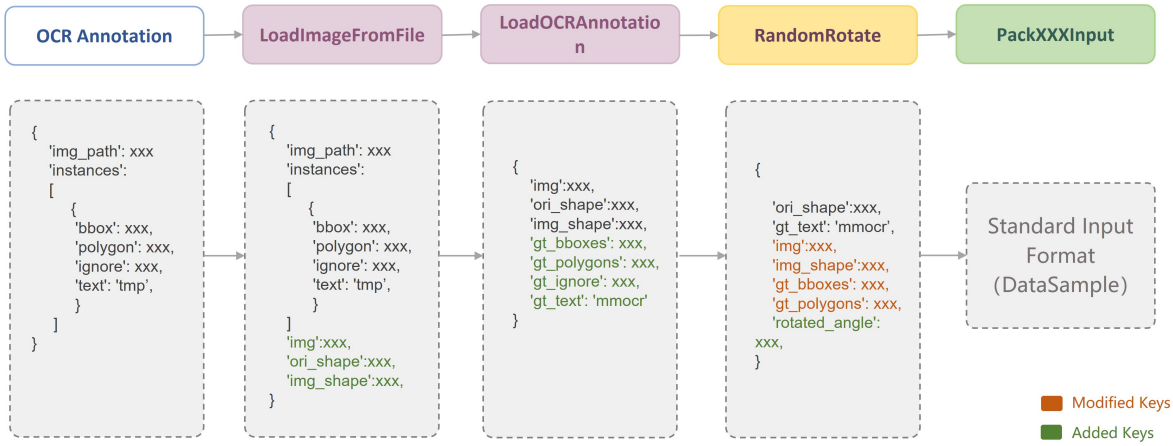
CHAPTER

TWELVE

DATA TRANSFORMS AND PIPELINE

In the design of MMOCR, dataset construction and preparation are decoupled. That is, dataset construction classes such as *OCRDataset* are responsible for loading and parsing annotation files; while data transforms further apply data preprocessing, augmentation, formatting, and other related functions. Currently, there are five types of data transforms implemented in MMOCR, as shown in the following table.

Since each data transform class is independent of each other, we can easily combine any data transforms to build a data pipeline after we have defined the data fields. As shown in the following figure, in MMOCR, a typical training data pipeline consists of three stages: **data loading**, **data augmentation**, and **data formatting**. Users only need to define the data pipeline list in the configuration file and specify the specific data transform class and its parameters:




```
train_pipeline_r18 = [
    # Loading images
    dict(
        type='LoadImageFromFile',
        color_type='color_ignore_orientation'),
    # Loading annotations
    dict(
        type='LoadOCRAnnotations',
        with_polygon=True,
        with_bbox=True,
        with_label=True,
    ),
    # Data augmentation
    dict(
        type='ImgAugWrapper',
```

(continues on next page)

(continued from previous page)

```
        args=[['Fliplr', 0.5],
              dict(cls='Affine', rotate=[-10, 10]), ['Resize', [0.5, 3.0]]]),
    dict(type='RandomCrop', min_side_ratio=0.1),
    dict(type='Resize', scale=(640, 640), keep_ratio=True),
    dict(type='Pad', size=(640, 640)),
    # Data formatting
    dict(
        type='PackTextDetInputs',
        meta_keys=('img_path', 'ori_shape', 'img_shape'))
]
```

Tip: More tutorials about data pipeline configuration can be found in the *Config Doc*. Next, we will briefly introduce the data transforms supported in MMOCR according to their categories.

For each data transform, MMOCR provides a detailed docstring. For example, in the header of each data transform class, we annotate `Required Keys`, `Modified Keys` and `Added Keys`. The `Required Keys` represent the mandatory fields that should be included in the input required by the data transform, while the `Modified Keys` and `Added Keys` indicate that the transform may modify or add the fields into the original data. For example, `LoadImageFromFile` implements the image loading function, whose `Required Keys` is the image path `img_path`, and the `Modified Keys` includes the loaded image `img`, the current size of the image `img_shape`, the original size of the image `ori_shape`, and other image attributes.

```
@TRANSFORMS.register_module()
class LoadImageFromFile(MMCV_LoadImageFromFile):
    # We provide detailed docstring for each data transform.
    """Load an image from file.

    Required Keys:

    - img_path

    Modified Keys:

    - img
    - img_shape
    - ori_shape
    """
```

Note: In the data pipeline of MMOCR, the image and label information are saved in a dictionary. By using the unified fields, the data can be freely transferred between different data transforms. Therefore, it is very important to understand the conventional fields used in MMOCR.

For your convenience, the following table lists the conventional keys used in MMOCR data transforms.

12.1 Data Loading

Data loading transforms mainly implement the functions of loading data from different formats and backends. Currently, the following data loading transforms are implemented in MMOCR:

12.2 Data Augmentation

Data augmentation is an indispensable process in text detection and recognition tasks. Currently, MMOCR has implemented dozens of data augmentation modules commonly used in OCR fields, which are classified into ocr_transforms.py, textdet_transforms.py, and textrecog_transforms.py.

Specifically, `ocr_transforms.py` implements generic OCR data augmentation modules such as `RandomCrop` and `RandomRotate`:

`textdet_transforms.py` implements text detection related data augmentation modules:

`textrecog_transforms.py` implements text recognition related data augmentation modules:

Warning: The above table only briefly introduces some selected data augmentation methods, for more information please refer to the API documentation or the code docstrings.

12.3 Data Formatting

Data formatting transforms are responsible for packaging images, ground truth labels, and other information into a dictionary. Different tasks usually rely on different formatting transforms. For example:

12.4 Cross Project Data Adapters

The cross-project data adapters bridge the data formats between MMOCR and other OpenMMLab libraries such as MMDetection, making it possible to call models implemented in other OpenMMLab projects. Currently, MMOCR has implemented *`MMDet2MMOCR`* and *`MMOCR2MMDet`*, allowing data to be converted between MMDetection and MMOCR formats; with these adapters, users can easily train any detectors supported by MMDetection in MMOCR. For example, we provide a tutorial to show how to train Mask R-CNN as a text detector in MMOCR.

12.5 Wrappers

To facilitate the use of popular third-party CV libraries in MMOCR, we provide wrappers in `wrappers.py` to unify the data format between MMOCR and other third-party libraries. Users can directly configure the data transforms provided by these libraries in the configuration file of MMOCR. The supported wrappers are as follows:

12.5.1 ImgAugWrapper Example

For example, in the original ImgAug, we can define a `Sequential` type data augmentation pipeline as follows to perform random flipping, random rotation and random scaling on the image:

```
import imgaug.augmenters as iaa

aug = iaa.Sequential(
  iaa.Fliplr(0.5),                # horizontally flip 50% of all images
  iaa.Affine(rotate=(-10, 10)),   # rotate by -10 to +10 degrees
  iaa.Resize((0.5, 3.0))          # scale images to 50-300% of their size
)
```

In MMOCR, we can directly configure the above data augmentation pipeline in `train_pipeline` as follows:

```
dict(
  type='ImgAugWrapper',
  args=[
    ['Fliplr', 0.5],
    dict(cls='Affine', rotate=[-10, 10]),
    ['Resize', [0.5, 3.0]],
  ]
)
```

Specifically, the `args` parameter accepts a list, and each element in the list can be a list or a dictionary. If it is a list, the first element of the list is the class name in `imgaug.augmenters`, and the following elements are the initialization parameters of the class; if it is a dictionary, the `cls` key corresponds to the class name in `imgaug.augmenters`, and the other key-value pairs correspond to the initialization parameters of the class.

12.5.2 TorchVisionWrapper Example

For example, in the original TorchVision, we can define a `Compose` type data transformation pipeline as follows to perform color jittering on the image:

```
import torchvision.transforms as transforms

aug = transforms.Compose([
  transforms.ColorJitter(
    brightness=32.0 / 255,  # brightness jittering range
    saturation=0.5)         # saturation jittering range
])
```

In MMOCR, we can directly configure the above data transformation pipeline in `train_pipeline` as follows:

```
dict(
  type='TorchVisionWrapper',
  op='ColorJitter',
  brightness=32.0 / 255,
  saturation=0.5
)
```

Specifically, the `op` parameter is the class name in `torchvision.transforms`, and the following parameters correspond to the initialization parameters of the class.

CHAPTER

THIRTEEN

EVALUATION

Note: Before reading this document, we recommend that you first read MMEngine: Model Accuracy Evaluation Basics.

13.1 Metrics

MMOCR implements widely-used evaluation metrics for text detection, text recognition and key information extraction tasks based on the MMEngine: BaseMetric base class. Users can specify the metric used in the validation and test phases by modifying the `val_evaluator` and `test_evaluator` fields in the configuration file. For example, the following config shows how to use `HmeanIOUMetric` to evaluate the model performance in text detection task.

```
val_evaluator = dict(type='HmeanIOUMetric')
test_evaluator = val_evaluator

# In addition, MMOCR also supports the combined evaluation of multiple metrics for the␣
↪same task, such as using WordMetric and CharMetric at the same time
val_evaluator = [
    dict(type='WordMetric', mode=['exact', 'ignore_case', 'ignore_case_symbol']),
    dict(type='CharMetric')
]
```

Tip: More evaluation related configurations can be found in the *evaluation configuration tutorial*.

As shown in the following table, MMOCR currently supports 5 evaluation metrics for text detection, text recognition, and key information extraction tasks, including `HmeanIOUMetric`, `WordMetric`, `CharMetric`, `OneMinusNEDMetric`, and `F1Metric`.

In general, the evaluation metric used in each task is conventionally determined. Users usually do not need to understand or manually modify the internal implementation of the evaluation metric. However, to facilitate more customized requirements, this document will further introduce the specific implementation details and configurable parameters of the built-in metrics in MMOCR.

13.1.1 HmeanIOUMetric

HmeanIOUMetric is one of the most widely used evaluation metrics in text detection tasks, because it calculates the harmonic mean (H-mean) between the detection precision (P) and recall rate (R). The `HmeanIOUMetric` can be calculated by the following equation:

$$H = \frac{2}{\frac{1}{P} + \frac{1}{R}} = \frac{2PR}{P + R}$$

In addition, since it is equivalent to the F-score (also known as F-measure or F-metric) when $\beta = 1$, `HmeanIOUMetric` is sometimes written as `F1Metric` or `f1-score`:

$$F_1 = (1 + \beta^2) \cdot \frac{PR}{\beta^2 \cdot P + R} = \frac{2PR}{P + R}$$

In MMOCR, the calculation of `HmeanIOUMetric` can be summarized as the following steps:

1. Filter out invalid predictions

   - Filter out predictions with a score is lower than `pred_score_thrs`
   - Filter out predictions overlapping with `ignored` ground truth boxes with an overlap ratio higher than `ignore_precision_thr`

   It is worth noting that `pred_score_thrs` will **automatically search** for the **best threshold** within a certain range by default, and users can also customize the search range by manually modifying the configuration file:

   ```
   # By default, HmeanIOUMetric searches the best threshold within the range [0.3, 0.
   ↪9] with a step size of 0.1
   val_evaluator = dict(type='HmeanIOUMetric', pred_score_thrs=dict(start=0.3, stop=0.
   ↪9, step=0.1))
   ```

2. Calculate the IoU matrix

   - At the data processing stage, `HmeanIOUMetric` will calculate and maintain an $M \times N$ IoU matrix `iou_metric` for the convenience of the subsequent bounding box pairing step. Here, M and N represent the number of label bounding boxes and filtered prediction bounding boxes, respectively. Therefore, each element of this matrix stores the IoU between the m-th label bounding box and the n-th prediction bounding box.

3. Compute the number of GT samples that can be accurately matched based on the corresponding pairing strategy

   Although `HmeanIOUMetric` can be calculated by a fixed formula, there may still be some subtle differences in the specific implementations. These differences mainly reflect the use of different strategies to match gt and predicted bounding boxes, which leads to the difference in final scores. Currently, MMOCR supports two matching strategies, namely `vanilla` and `max_matching`, for the `HmeanIOUMetric`. As shown below, users can specify the matching strategies in the config.

   - `vanilla` matching strategy

     By default, `HmeanIOUMetric` adopts the `vanilla` matching strategy, which is consistent with the `hmean-iou` implementation in MMOCR 0.x and the **official** text detection competition evaluation standard of ICDAR series. The matching strategy adopts the first-come-first-served matching method to pair the labels and predictions.

     ```
     # By default, HmeanIOUMetric adopts 'vanilla' matching strategy
     val_evaluator = dict(type='HmeanIOUMetric')
     ```

   - `max_matching` matching strategy

     To address the shortcomings of the existing matching mechanism, MMOCR has implemented a more efficient matching strategy to maximize the number of matches.

```
# Specify to use 'max_matching' matching strategy
val_evaluator = dict(type='HmeanIOUMetric', strategy='max_matching')
```

Note: We recommend that research-oriented developers use the default `vanilla` matching strategy to ensure consistency with other papers. For industry-oriented developers, you can use the `max_matching` matching strategy to achieve optimized performance.

4. Compute the final evaluation score according to the aforementioned matching strategy

13.1.2 WordMetric

WordMetric implements **word-level** text recognition evaluation metrics and includes three text matching modes, namely `exact`, `ignore_case`, and `ignore_case_symbol`. Users can freely combine the output of one or more text matching modes in the configuration file by modifying the `mode` field.

```
# Use WordMetric for text recognition task
val_evaluator = [
    dict(type='WordMetric', mode=['exact', 'ignore_case', 'ignore_case_symbol'])
]
```

- `exact`Full matching mode, i.e., only when the predicted text and the ground truth text are exactly the same, the predicted text is considered to be correct.
- `ignore_case`The mode ignores the case of the predicted text and the ground truth text.
- `ignore_case_symbol`The mode ignores the case and symbols of the predicted text and the ground truth text. This is also the text recognition accuracy reported by most academic papers. The performance reported by MMOCR uses the `ignore_case_symbol` mode by default.

Assume that the real label is `MMOCR!` and the model output is `mmocr`. The `WordMetric` scores under the three matching modes are: `{'exact': 0, 'ignore_case': 0, 'ignore_case_symbol': 1}`.

13.1.3 CharMetric

CharMetric implements **character-level** text recognition evaluation metrics that are **case-insensitive**.

```
# Use CharMetric for text recognition task
val_evaluator = [dict(type='CharMetric')]
```

Specifically, `CharMetric` will output two evaluation metrics, namely `char_precision` and `char_recall`. Let the number of correctly predicted characters (True Positive) be σ_{tp}, then the precision P and recall R can be calculated by the following equation:

$$P = \frac{\sigma_{tp}}{\sigma_{pred}}, R = \frac{\sigma_{tp}}{\sigma_{gt}}$$

where σ_{gt} and σ_{pred} represent the total number of characters in the label text and the predicted text, respectively.

For example, assume that the label text is "MM**O**CR" and the predicted text is "mm**0**c**R1**". The score of the `CharMetric` is:

$$P = \frac{4}{6}, R = \frac{4}{5}$$

13.1.4 OneMinusNEDMetric

OneMinusNEDMetric(1-N.E.D) is commonly used for text recognition evaluation of Chinese or English **text line-level** annotations. Unlike the full matching metric that requires the prediction and the gt text to be exactly the same, `1-N.E.D` uses the normalized edit distance (also known as Levenshtein Distance) to measure the difference between the predicted and the gt text, so that the performance difference of the model can be better distinguished when evaluating long texts. Assume that the real and predicted texts are s_i and $\hat{s}_i$, respectively, and their lengths are l_i and $\hat{l}_i$, respectively. The `OneMinusNEDMetric` score can be calculated by the following formula:

$$score = 1 - \frac{1}{N}\sum_{i=1}^{N}\frac{D(s_i, \hat{s}_i)}{max(l_i, \hat{l}_i)}$$

where N is the total number of samples, and $D(s_1, s_2)$ is the edit distance between two strings.

For example, assume that the real label is "OpenMMLabMMOCR", the prediction of model A is "0penMMLab-MMOCR", and the prediction of model B is "uvwxyz". The results of the full matching and `OneMinusNEDMetric` evaluation metrics are as follows:

As shown in the table above, although the model A only predicted one letter incorrectly, both models got 0 in when using full-match strategy. However, the `OneMinusNEDMetric` evaluation metric can better distinguish the performance of the two models on **long texts**.

13.1.5 F1Metric

F1Metric implements the F1-Metric evaluation metric for KIE tasks and provides two modes, namely `micro` and `macro`.

```
val_evaluator = [
    dict(type='F1Metric', mode=['micro', 'macro'],
]
```

- `micro` mode: Calculate the global F1-Metric score based on the total number of True Positive, False Negative, and False Positive.
- `macro` modeCalculate the F1-Metric score for each class and then take the average.

13.1.6 Customized Metric

MMOCR supports the implementation of customized evaluation metrics for users who pursue higher customization. In general, users only need to create a customized evaluation metric class `CustomizedMetric` and inherit MMEngine: BaseMetric. Then, the data format processing method `process` and the metric calculation method `compute_metrics` need to be overwritten respectively. Finally, add it to the `METRICS` registry to implement any customized evaluation metric.

```
from mmengine.evaluator import BaseMetric
from mmocr.registry import METRICS

@METRICS.register_module()
class CustomizedMetric(BaseMetric):

    def process(self, data_batch: Sequence[Dict], predictions: Sequence[Dict]):
    """ process receives two parameters, data_batch stores the gt label information, and␣
↪predictions stores the predicted results.
```

(continues on next page)

(continued from previous page)

```
    """
        pass

    def compute_metrics(self, results: List):
    """ compute_metric receives the results of the process method as input and returns␣
↪the evaluation results.
    """
        pass
```

Note: More details can be found in MMEngine Documentation: BaseMetric.

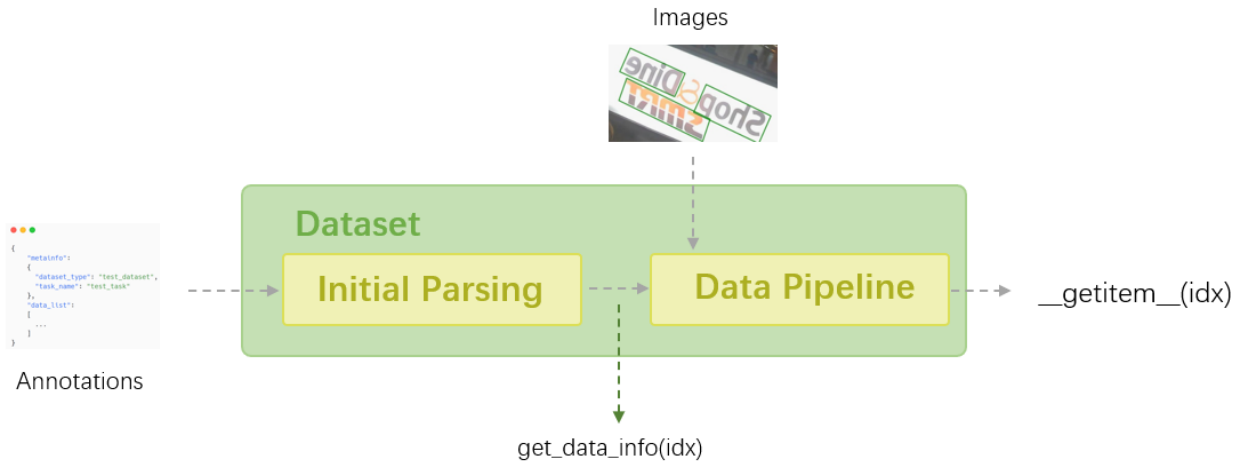
CHAPTER

FOURTEEN

DATASET

14.1 Overview

In MMOCR, all the datasets are processed via different Dataset classes based on mmengine.BaseDataset. Dataset classes are responsible for loading the data and performing initial parsing, then fed to *data pipeline* for data preprocessing, augmentation, formatting, etc.



In this tutorial, we will introduce some common interfaces of the Dataset class, and the usage of Dataset implementations in MMOCR as well as the annotation types they support.

Tip: Dataset class supports some advanced features, such as lazy initialization and data serialization, and takes advantage of various dataset wrappers to perform data concatenation, repeating, and category balancing. These content will not be covered in this tutorial, but you can read MMEngine: BaseDataset for more details.

14.2 Common Interfaces

Now, let's look at a concrete example and learn some typical interfaces of a Dataset class. `OCRDataset` is a widely used Dataset implementation in MMOCR, and is suggested as a default Dataset type in MMOCR as its associated annotation format is flexible enough to support *all* the OCR tasks (*more info*). Now we will instantiate an `OCRDataset` object wherein the toy dataset in `tests/data/det_toy_dataset` will be loaded.

```
from mmocr.datasets import OCRDataset
from mmengine.registry import init_default_scope
init_default_scope('mmocr')

train_pipeline = [
    dict(
        type='LoadImageFromFile'),
    dict(
        type='LoadOCRAnnotations',
        with_polygon=True,
        with_bbox=True,
        with_label=True,
    ),
    dict(type='RandomCrop', min_side_ratio=0.1),
    dict(type='Resize', scale=(640, 640), keep_ratio=True),
    dict(type='Pad', size=(640, 640)),
    dict(
        type='PackTextDetInputs',
        meta_keys=('img_path', 'ori_shape', 'img_shape'))
]
dataset = OCRDataset(
    data_root='tests/data/det_toy_dataset',
    ann_file='textdet_test.json',
    test_mode=False,
    pipeline=train_pipeline)
```

Let's peek the size of this dataset:

```
>>> print(len(dataset))

10
```

Typically, a Dataset class loads and stores two types of information: (1) **meta information**: Some meta descriptors of the dataset's property, such as available object categories in this dataset. (2) **annotation**: The path to images, and their labels. We can access the meta information in `dataset.metainfo`:

```
>>> from pprint import pprint
>>> pprint(dataset.metainfo)

{'category': [{'id': 0, 'name': 'text'}],
 'dataset_type': 'TextDetDataset',
 'task_name': 'textdet'}
```

As for the annotations, we can access them via `dataset.get_data_info(idx)`, which returns a dictionary containing the information of the `idx`-th sample in the dataset that is initially parsed, but not yet processed by *data pipeline*.

```
>>> from pprint import pprint
>>> pprint(dataset.get_data_info(0))

{'height': 720,
 'img_path': 'tests/data/det_toy_dataset/test/img_10.jpg',
 'instances': [{'bbox': [260.0, 138.0, 284.0, 158.0],
```

(continues on next page)

(continued from previous page)

```
                'bbox_label': 0,
                'ignore': True,
                'polygon': [261, 138, 284, 140, 279, 158, 260, 158]},
                ...,
               {'bbox': [1011.0, 157.0, 1079.0, 173.0],
                'bbox_label': 0,
                'ignore': True,
                'polygon': [1011, 157, 1079, 160, 1076, 173, 1011, 170]}],
 'sample_idx': 0,
 'seg_map': 'test/gt_img_10.txt',
 'width': 1280}
```

On the other hand, we can get the sample fully processed by data pipeline via `dataset[idx]` or `dataset.__getitem__(idx)`, which is directly feedable to models and perform a full train/test cycle. It has two fields:

- `inputs`: The image after data augmentation;
- `data_samples`: The *DataSample* that contains the augmented annotations, and meta information appended by some data transforms to keep track of some key properties of this sample.

```
>>> pprint(dataset[0])

{'data_samples': <TextDetDataSample(

    META INFORMATION
    ori_shape: (720, 1280)
    img_path: 'tests/data/det_toy_dataset/imgs/test/img_10.jpg'
    img_shape: (640, 640)

    DATA FIELDS
    gt_instances: <InstanceData(

            META INFORMATION

            DATA FIELDS
            labels: tensor([0, 0, 0])
            polygons: [array([207.33984 , 104.65409 , 208.34634 ,  84.528305, 231.49594 ,
                        86.54088 , 226.46341 , 104.65409 , 207.33984 , 104.65409 ],
                      dtype=float32), array([237.53496 , 103.6478  , 235.52196 ,  84.
↪528305, 365.36096 ,
                        86.54088 , 364.35446 , 107.67296 , 237.53496 , 103.6478  ],
                      dtype=float32), array([105.68293, 166.03773, 105.68293, 151.94969,␣
↪177.14471, 150.94339,
                       178.15121, 165.03145, 105.68293, 166.03773], dtype=float32)]
            ignored: tensor([ True, False,  True])
            bboxes: tensor([[207.3398,  84.5283, 231.4959, 104.6541],
                        [235.5220,  84.5283, 365.3610, 107.6730],
                        [105.6829, 150.9434, 178.1512, 166.0377]])
        ) at 0x7f7359f04fa0>
) at 0x7f735a0508e0>,
 'inputs': tensor([[[129, 111, 131,  ...,   0,   0,   0], ...
                   [ 19,  18,  15,  ...,   0,   0,   0]]], dtype=torch.uint8)}
```

14.3 Dataset Classes and Annotation Formats

Each Dataset implementation can only load datasets in a specific annotation format. Here lists all supported Dataset classes and their compatible annotation formats, as well as an example config that showcases how to use them in practice.

Note: If you are not familiar with the config system, you may find *Dataset Configuration* helpful.

14.3.1 OCRDataset

Usually, there are many different types of annotations in OCR datasets, and the formats often vary between different subtasks, such as text detection and text recognition. These differences can result in the need for different data loading code when using different datasets, increasing the learning and maintenance costs for users.

In MMOCR, we propose a unified dataset format that can adapt to all three subtasks of OCR: text detection, text recognition, and text spotting. This design maximizes the uniformity of the dataset, allows for the reuse of data annotations across different tasks, and makes dataset management more convenient. Considering that popular dataset formats are still inconsistent, MMOCR provides *Dataset Preparer* to help users convert their datasets to MMOCR format. We also strongly encourage researchers to develop their own datasets based on this data format.

Annotation Format

This annotation file is a `.json` file that stores a `dict`, containing both `metainfo` and `data_list`, where the former includes basic information about the dataset and the latter consists of the label item of each target instance. Here presents an extensive list of all the fields in the annotation file, but some fields are used in a subset of tasks and can be ignored in other tasks.

```
{
    "metainfo":
    {
      "dataset_type": "TextDetDataset",  # Options: TextDetDataset/TextRecogDataset/
↪TextSpotterDataset
      "task_name": "textdet",  #  Options: textdet/textspotter/textrecog
      "category": [{"id": 0, "name": "text"}]  # Used in textdet/textspotter
    },
    "data_list":
    [
      {
        "img_path": "test_img.jpg",
        "height": 604,
        "width": 640,
        "instances":  # multiple instances in one image
        [
          {
            "bbox": [0, 0, 10, 20],  # in textdet/textspotter, [x1, y1, x2, y2].
            "bbox_label": 0,  # The object category, always 0 (text) in MMOCR
            "polygon": [0, 0, 0, 10, 10, 20, 20, 0], # in textdet/textspotter. [x1, y1,␣
↪x2, y2, ....]
            "text": "mmocr",  # in textspotter/textrecog
            "ignore": False # in textspotter/textdet. Whether to ignore this sample␣
↪during training
```

(continues on next page)

(continued from previous page)

```
          },
          #...
        ],
      }
      #... multiple images
    ]
}
```

Example Config

Here is a part of config example where we make `train_dataloader` use `OCRDataset` to load the ICDAR2015 dataset for a text detection model. Keep in mind that `OCRDataset` can load any OCR datasets prepared by Dataset Preparer regardless of its task. That is, you can use it for text recognition and text spotting, but you still have to modify the transform types in `pipeline` according to the needs of different tasks.

```
pipeline = [
    dict(
        type='LoadImageFromFile'),
    dict(
        type='LoadOCRAnnotations',
        with_polygon=True,
        with_bbox=True,
        with_label=True,
    ),
    dict(
        type='PackTextDetInputs',
        meta_keys=('img_path', 'ori_shape', 'img_shape'))
]

icdar2015_textdet_train = dict(
    type='OCRDataset',
    data_root='data/icdar2015',
    ann_file='textdet_train.json',
    filter_cfg=dict(filter_empty_gt=True, min_size=32),
    pipeline=pipeline)

train_dataloader = dict(
    batch_size=16,
    num_workers=8,
    persistent_workers=True,
    sampler=dict(type='DefaultSampler', shuffle=True),
    dataset=icdar2015_textdet_train)
```

14.3.2 RecogLMDBDataset

Reading images or labels from files can be slow when data are excessive, e.g. on a scale of millions. Besides, in academia, most of the scene text recognition datasets are stored in lmdb format, including images and labels. (Example)

To get closer to the mainstream practice and enhance the data storage efficiency, MMOCR supports loading images and labels from lmdb datasets via `RecogLMDBDataset`.

Annotation Format

MMOCR requires the following keys for LMDB datasets:

- `num_samples`: The parameter describing the data volume of the dataset.
- The keys of images and labels are in the format of `image-000000001` and `label-000000001`, respectively. The index starts from 1.

MMOCR has a toy LMDB dataset in `tests/data/rec_toy_dataset/imgs.lmdb`. You can get a sense of the format with the following code snippet.

```
>>> import lmdb
>>>
>>> env = lmdb.open('tests/data/rec_toy_dataset/imgs.lmdb')
>>> txn = env.begin()
>>> for k, v in txn.cursor():
>>>     print(k, v)

b'image-000000001' b'\xff...'
b'image-000000002' b'\xff...'
b'image-000000003' b'\xff...'
b'image-000000004' b'\xff...'
b'image-000000005' b'\xff...'
b'image-000000006' b'\xff...'
b'image-000000007' b'\xff...'
b'image-000000008' b'\xff...'
b'image-000000009' b'\xff...'
b'image-000000010' b'\xff...'
b'label-000000001' b'GRAND'
b'label-000000002' b'HOTEL'
b'label-000000003' b'HOTEL'
b'label-000000004' b'PACIFIC'
b'label-000000005' b'03/09/2009'
b'label-000000006' b'ANING'
b'label-000000007' b'Virgin'
b'label-000000008' b'america'
b'label-000000009' b'ATTACK'
b'label-000000010' b'DAVIDSON'
b'num-samples' b'10'
```

Example Config

Here is a part of config example where we make `train_dataloader` use `RecogLMDBDataset` to load the toy dataset. Since `RecogLMDBDataset` loads images as numpy arrays, don't forget to use `LoadImageFromNDArray` instead of `LoadImageFromFile` in the pipeline for successful loading.

```
pipeline = [
    dict(
        type='LoadImageFromNDArray'),
    dict(
        type='LoadOCRAnnotations',
        with_text=True,
    ),
    dict(
        type='PackTextRecogInputs',
        meta_keys=('img_path', 'ori_shape', 'img_shape'))
]

toy_textrecog_train = dict(
    type='RecogLMDBDataset',
    data_root='tests/data/rec_toy_dataset/',
    ann_file='imgs.lmdb',
    pipeline=pipeline)

train_dataloader = dict(
    batch_size=16,
    num_workers=8,
    persistent_workers=True,
    sampler=dict(type='DefaultSampler', shuffle=True),
    dataset=toy_textrecog_train)
```

14.3.3 RecogTextDataset

Prior to MMOCR 1.0, MMOCR 0.x takes text files as input for text recognition. These formats has been deprecated in MMOCR 1.0, and this class could be removed anytime in the future. *More info*

Annotation Format

Text files can either be in `txt` format or `jsonl` format. The simple `.txt` annotations separate image name and word annotation by a blank space, which cannot handle the case when spaces are included in a text instance.

```
img1.jpg OpenMMLab
img2.jpg MMOCR
```

The JSON Line format uses a dictionary-like structure to represent the annotations, where the keys `filename` and `text` store the image name and word label, respectively.

```
{"filename": "img1.jpg", "text": "OpenMMLab"}
{"filename": "img2.jpg", "text": "MMOCR"}
```

Example Config

Here is a part of config example where we use `RecogTextDataset` to load the old txt labels in training, and the old jsonl labels in testing.

```
pipeline = [
    dict(
        type='LoadImageFromFile'),
    dict(
        type='LoadOCRAnnotations',
        with_polygon=True,
        with_bbox=True,
        with_label=True,
    ),
    dict(
        type='PackTextDetInputs',
        meta_keys=('img_path', 'ori_shape', 'img_shape'))
]

 # loading 0.x txt format annos
 txt_dataset = dict(
     type='RecogTextDataset',
     data_root=data_root,
     ann_file='old_label.txt',
     data_prefix=dict(img_path='imgs'),
     parser_cfg=dict(
         type='LineStrParser',
         keys=['filename', 'text'],
         keys_idx=[0, 1]),
     pipeline=pipeline)


train_dataloader = dict(
    batch_size=16,
    num_workers=8,
    persistent_workers=True,
    sampler=dict(type='DefaultSampler', shuffle=True),
    dataset=txt_dataset)

 # loading 0.x json line format annos
 jsonl_dataset = dict(
     type='RecogTextDataset',
     data_root=data_root,
     ann_file='old_label.jsonl',
     data_prefix=dict(img_path='imgs'),
     parser_cfg=dict(
         type='LineJsonParser',
         keys=['filename', 'text'],
     pipeline=pipeline))

test_dataloader = dict(
    batch_size=16,
    num_workers=8,
```

(continues on next page)

(continued from previous page)

```
    persistent_workers=True,
    sampler=dict(type='DefaultSampler', shuffle=False),
    dataset=jsonl_dataset)
```

14.3.4 IcdarDataset

Prior to MMOCR 1.0, MMOCR 0.x takes COCO-like format annotations as input for text detection. These formats has been deprecated in MMOCR 1.0, and this class could be removed anytime in the future. *More info*

Annotation Format

```
{
  "images": [
    {
      "id": 1,
      "width": 800,
      "height": 600,
      "file_name": "test.jpg"
    }
  ],
  "annotations": [
    {
      "id": 1,
      "image_id": 1,
      "category_id": 1,
      "bbox": [0,0,10,10],
      "segmentation": [
          [0,0,10,0,10,10,0,10]
      ],
      "area": 100,
      "iscrowd": 0
    }
  ]
}
```

Example Config

Here is a part of config example where we make `train_dataloader` use `IcdarDataset` to load the old labels.

```
pipeline = [
    dict(
        type='LoadImageFromFile'),
    dict(
        type='LoadOCRAnnotations',
        with_polygon=True,
        with_bbox=True,
        with_label=True,
    ),
    dict(
```

(continues on next page)

(continued from previous page)

```
        type='PackTextDetInputs',
        meta_keys=('img_path', 'ori_shape', 'img_shape'))
]

icdar2015_textdet_train = dict(
    type='IcdarDatasetDataset',
    data_root='data/det/icdar2015',
    ann_file='instances_training.json',
    filter_cfg=dict(filter_empty_gt=True, min_size=32),
    pipeline=pipeline)

train_dataloader = dict(
    batch_size=16,
    num_workers=8,
    persistent_workers=True,
    sampler=dict(type='DefaultSampler', shuffle=True),
    dataset=icdar2015_textdet_train)
```

14.3.5 WildReceiptDataset

It's customized for WildReceipt dataset only.

Annotation Format

```
// Close Set
{
  "file_name": "image_files/Image_16/11/d5de7f2a20751e50b84c747c17a24cd98bed3554.jpeg",
  "height": 1200,
  "width": 1600,
  "annotations":
    [
      {
        "box": [550.0, 190.0, 937.0, 190.0, 937.0, 104.0, 550.0, 104.0],
        "text": "SAFEWAY",
        "label": 1
      },
      {
        "box": [1048.0, 211.0, 1074.0, 211.0, 1074.0, 196.0, 1048.0, 196.0],
        "text": "TM",
        "label": 25
      }
    ], //...
}

// Open Set
{
  "file_name": "image_files/Image_12/10/845be0dd6f5b04866a2042abd28d558032ef2576.jpeg",
  "height": 348,
  "width": 348,
  "annotations":
```

(continues on next page)

(continued from previous page)


```
    [
      {
        "box": [114.0, 19.0, 230.0, 19.0, 230.0, 1.0, 114.0, 1.0],
        "text": "CHOEUN",
        "label": 2,
        "edge": 1
      },
      {
        "box": [97.0, 35.0, 236.0, 35.0, 236.0, 19.0, 97.0, 19.0],
        "text": "KOREANRESTAURANT",
        "label": 2,
        "edge": 1
      }
    ]
}
```


Example Config

Please refer to SDMGR's config for more details.

CHAPTER

FIFTEEN

OVERVIEW & FEATURES[COMING SOON]

Coming Soon!

CHAPTER

SIXTEEN

DATA FLOW[COMING SOON]

Coming Soon!

CHAPTER

SEVENTEEN

MODELS[COMING SOON]

Coming Soon!

CHAPTER

EIGHTEEN

VISUALIZERS[COMING SOON]

Coming Soon!

CHAPTER

NINETEEN

CONVENTION[COMING SOON]

Coming Soon!

CHAPTER

TWENTY

ENGINE[COMING SOON]

Coming Soon!

CHAPTER

TWENTYONE

OVERVIEW

21.1 Supported Datasets

Dataset Name	Text Detection	Text Recognition	Text Spotting	KIE
cocotextv2	✓	✓	✓	
ctw1500	✓	✓	✓	
cute80		✓		
funsd	✓	✓	✓	
icdar2013	✓	✓	✓	
icdar2015	✓	✓	✓	
iiit5k		✓		
mjsynth		✓		
naf	✓	✓	✓	
sroie	✓	✓	✓	
svt	✓	✓	✓	
svtp		✓		
synthtext	✓	✓	✓	
textocr	✓	✓	✓	
totaltext	✓	✓	✓	
wildreceipt	✓	✓	✓	✓

21.2 Dataset Details

21.2.1 COCO Text v2

> "COCO-Text: Dataset and Benchmark for Text Detection and Recognition in Natural Images", *arXiv*, 2016. PDF

A. Basic Info

- Official Website: cocotextv2
- Year: 2016
- Language: ['English']
- Scene: ['Natural Scene']
- Annotation Granularity: ['Word']
- Supported Tasks: ['textdet', 'textrecog', 'textspotting']
- License: CC BY 4.0

Text Detection/Spotting

```
{
  "cats": {},
  "anns": {
      "45346": {
          "mask":[468.9,286.7,468.9,295.2,493.0,295.8,493.0,287.2],
          "class":"machine printed",
          "bbox":[468.9,286.7,24.1,9.1],
          "image_id":522579,
          "id":167312,
          "language":"english",
          "area":55.5,
          "utf8_string":"the",
          "legibility":"legible"
      },
      // ...
  },
  "imgs": {
      "522579": {
          "file_name":"COCO_train2014_000000522579.jpg",
          "height":476,
          "width":640,
          "id":522579,
          "set":"train",
      },
      // ...
  },
  "imgToAnns": {
      "522579": [167294, 167295, 167296, 167297, 167298, 167299, 167300, 167301, 167302,␣
↪167303, 167304, 167305, 167306, 167307, 167308, 167309, 167310, 167311, 167312, 167313,
↪ 167314, 167315, 167316, 167317],
      // ...
  },
```

(continues on next page)

(continued from previous page)

```
  "info": {}
}
```

C. Reference

```
@article{veit2016coco, title={Coco-text: Dataset and benchmark for text detection and␣
↪recognition in natural images}, author={Veit, Andreas and Matera, Tomas and Neumann,␣
↪Lukas and Matas, Jiri and Belongie, Serge}, journal={arXiv preprint arXiv:1601.07140},␣
↪year={2016}}
```

21.2.2 CTW1500

"Curved scene text detection via transverse and longitudinal sequence connection", *PR*, 2019. PDF

A. Basic Info

- Official Website: ctw1500
- Year: 2019
- Language: ['English']
- Scene: ['Scene']
- Annotation Granularity: ['Word', 'Line']
- Supported Tasks: ['textrecog', 'textdet', 'textspotting']
- License: N/A

C. Reference

```
@article{liu2019curved, title={Curved scene text detection via transverse and␣
↪longitudinal sequence connection}, author={Liu, Yuliang and Jin, Lianwen and Zhang,␣
↪Shuaitao and Luo, Canjie and Zhang, Sheng}, journal={Pattern Recognition}, volume={90},
↪ pages={337--345}, year={2019}, publisher={Elsevier} }
```

21.2.3 CUTE80

"A Robust Arbitrary Text Detection System for Natural Scene Images", *ESWA*, 2014. PDF

A. Basic Info

- Official Website: cute80
- Year: 2014
- Language: ['English']
- Scene: ['Natural Scene']
- Annotation Granularity: ['Word']
- Supported Tasks: ['textrecog']
- License: N/A

Text Recognition

```
# timage/img_name text 1 text

timage/001.jpg RONALDO 1 RONALDO
timage/002.jpg 7 1 7
timage/003.jpg SEACREST 1 SEACREST
timage/004.jpg BEACH 1 BEACH
```

C. Reference

```
@article{risnumawan2014robust, title={A robust arbitrary text detection system for␣
↪natural scene images}, author={Risnumawan, Anhar and Shivakumara, Palaiahankote and␣
↪Chan, Chee Seng and Tan, Chew Lim}, journal={Expert Systems with Applications}, volume=
↪{41}, number={18}, pages={8027--8048}, year={2014}, publisher={Elsevier}}
```

21.2.4 FUNSD

"FUNSD: A Dataset for Form Understanding in Noisy Scanned Documents", *ICDAR*, 2019. PDF

A. Basic Info

- Official Website: funsd
- Year: 2019
- Language: ['English']
- Scene: ['Document']
- Annotation Granularity: ['Word']
- Supported Tasks: ['textdet', 'textrecog', 'textspotting']
- License: FUNSD License

Text Detection/Recognition/Spotting

```
{
  "form": [
    {
      "id": 0,
      "text": "Registration No.",
      "box": [
          94,
          169,
          191,
          186
      ],
      "linking": [
          [
              0,
              1
          ]
      ],
      "label": "question",
      "words": [
          {
```

(continues on next page)

(continued from previous page)


```
                "text": "Registration",
                "box": [
                    94,
                    169,
                    168,
                    186
                ]
            },
            {
                "text": "No.",
                "box": [
                    170,
                    169,
                    191,
                    183
                ]
            }
        ]
    },
    {
        "id": 1,
        "text": "533",
        "box": [
            209,
            169,
            236,
            182
        ],
        "label": "answer",
        "words": [
            {
                "box": [
                    209,
                    169,
                    236,
                    182
                ],
                "text": "533"
            }
        ],
        "linking": [
            [
                0,
                1
            ]
        ]
    }
  ]
}
```


C. Reference

```
@inproceedings{jaume2019, title = {FUNSD: A Dataset for Form Understanding in Noisy␣
↪Scanned Documents}, author = {Guillaume Jaume, Hazim Kemal Ekenel, Jean-Philippe␣
↪Thiran}, booktitle = {Accepted to ICDAR-OST}, year = {2019}}
```

21.2.5 Incidental Scene Text IC13

"ICDAR 2013 Robust Reading Competition", *ICDAR*, 2013. PDF

A. Basic Info

- Official Website: icdar2013
- Year: 2013
- Language: ['English']
- Scene: ['Natural Scene']
- Annotation Granularity: ['Word']
- Supported Tasks: ['textdet', 'textrecog', 'textspotting']
- License: N/A

Text Detection

```
# train split
# x1 y1 x2 y2 "transcript"

158 128 411 181 "Footpath"
443 128 501 169 "To"
64 200 363 243 "Colchester"

# test split
# x1, y1, x2, y2, "transcript"

38, 43, 920, 215, "Tiredness"
275, 264, 665, 450, "kills"
0, 699, 77, 830, "A"
```

Text Recognition

```
# img_name, "text"

word_1.png, "PROPER"
word_2.png, "FOOD"
word_3.png, "PRONTO"
```

C. Reference

```
@inproceedings{karatzas2013icdar, title={ICDAR 2013 robust reading competition}, author=
↪{Karatzas, Dimosthenis and Shafait, Faisal and Uchida, Seiichi and Iwamura, Masakazu␣
↪and i Bigorda, Lluis Gomez and Mestre, Sergi Robles and Mas, Joan and Mota, David␣
↪Fernandez and Almazan, Jon Almazan and De Las Heras, Lluis Pere}, booktitle={2013 12th␣
↪international conference on document analysis and recognition}, pages={1484--1493},␣
↪year={2013}, organization={IEEE}}
```

21.2.6 Incidental Scene Text IC15

"ICDAR 2015 Competition on Robust Reading", *ICDAR*, 2015. PDF

A. Basic Info

- Official Website: icdar2015
- Year: 2015
- Language: ['English']
- Scene: ['Natural Scene']
- Annotation Granularity: ['Word']
- Supported Tasks: ['textdet', 'textrecog', 'textspotting']
- License: CC BY 4.0

Text Detection

```
# x1,y1,x2,y2,x3,y3,x4,y4,trans

377,117,463,117,465,130,378,130,Genaxis Theatre
493,115,519,115,519,131,493,131,[06]
374,155,409,155,409,170,374,170,###
```

Text Recognition

```
# img_name, "text"

word_1.png, "Genaxis Theatre"
word_2.png, "[06]"
word_3.png, "62-03"
```

C. Reference

```
@inproceedings{karatzas2015icdar, title={ICDAR 2015 competition on robust reading},␣
↪author={Karatzas, Dimosthenis and Gomez-Bigorda, Lluis and Nicolaou, Anguelos and␣
↪Ghosh, Suman and Bagdanov, Andrew and Iwamura, Masakazu and Matas, Jiri and Neumann,␣
↪Lukas and Chandrasekhar, Vijay Ramaseshan and Lu, Shijian and others}, booktitle={2015␣
↪13th international conference on document analysis and recognition (ICDAR)}, pages=
↪{1156--1160}, year={2015}, organization={IEEE}}
```

21.2.7 IIIT5K

"Scene Text Recognition using Higher Order Language Priors", *BMVC*, 2012. PDF

A. Basic Info

- Official Website: iiit5k
- Year: 2012
- Language: ['English']
- Scene: ['Natural Scene']
- Annotation Granularity: ['Word']

- Supported Tasks: ['textrecog']
- License: N/A

Text Recognition

```
# img_name, "text"

train/1009_2.png You
train/1017_1.png Rescue
train/1017_2.png mission
```

C. Reference

```
@InProceedings{MishraBMVC12, author    = "Mishra, A. and Alahari, K. and Jawahar, C.~V.",
↪ title     = "Scene Text Recognition using Higher Order Language Priors", booktitle =
↪"BMVC", year      = "2012"}
```

21.2.8 Synthetic Word Dataset (MJSynth/Syn90k)

> "Reading Text in the Wild with Convolutional Neural Networks", *International Journal of Computer Vision*, 2016. PDF

A. Basic Info

- Official Website: mjsynth
- Year: 2016
- Language: ['English']
- Scene: ['Synthesis']
- Annotation Granularity: ['Word']
- Supported Tasks: ['textrecog']
- License: N/A

Text Recognition

```
./3000/7/182_slinking_71711.jpg 71711
./3000/7/182_REMODELERS_64541.jpg 64541
```

C. Reference

```
@InProceedings{Jaderberg14c, author       = "Max Jaderberg and Karen Simonyan and Andrea␣
↪Vedaldi and Andrew Zisserman", title        = "Synthetic Data and Artificial Neural␣
↪Networks for Natural Scene Text Recognition", booktitle    = "Workshop on Deep␣
↪Learning, NIPS", year         = "2014", }
@Article{Jaderberg16, author       = "Max Jaderberg and Karen Simonyan and Andrea␣
↪Vedaldi and Andrew Zisserman", title        = "Reading Text in the Wild with␣
↪Convolutional Neural Networks", journal      = "International Journal of Computer␣
↪Vision", number       = "1", volume       = "116", pages        = "1--20", month      ␣
↪ = "jan", year         = "2016", }
```

21.2.9 NAF

"Deep Visual Template-Free Form Parsing", *ICDAR*, 2019. PDF

A. Basic Info

- Official Website: naf
- Year: 2019
- Language: ['English']
- Scene: ['Document', 'Handwritten']
- Annotation Granularity: ['Word', 'Line']
- Supported Tasks: ['textrecog', 'textdet', 'textspotting']
- License: CDLA

Text Detection/Recognition/Spotting


```
{"fieldBBs": [{"poly_points": [[435, 1406], [466, 1406], [466, 1439], [435, 1439]], "type
↪": "fieldCheckBox", "id": "f0", "isBlank": 1}, {"poly_points": [[435, 1444], [469,␣
↪1444], [469, 1478], [435, 1478]], "type": "fieldCheckBox", "id": "f1", "isBlank": 1}],
 "textBBs": [{"poly_points": [[1183, 1337], [2028, 1345], [2032, 1395], [1186, 1398]],
↪"type": "text", "id": "t0"}, {"poly_points": [[492, 1336], [809, 1338], [809, 1379],␣
↪[492, 1378]], "type": "text", "id": "t1"}, {"poly_points": [[512, 1375], [798, 1376],␣
↪[798, 1405], [512, 1404]], "type": "textInst", "id": "t2"}], "imageFilename":
↪"007182398_00026.jpg", "transcriptions": {"f0": "\u00bf\u00bf\u00bf \u00bf\u00bf\u00bf␣
↪18/1/49 \u00bf\u00bf\u00bf\u00bf\u00bf", "f1": "U.S. Navy 53rd. Naval Const. Batt.",
↪"t0": "APPLICATION FOR HEADSTONE OR MARKER", "t1": "ORIGINAL"}}
```


C. Reference


```
@inproceedings{davis2019deep, title={Deep visual template-free form parsing}, author=
↪{Davis, Brian and Morse, Bryan and Cohen, Scott and Price, Brian and Tensmeyer, Chris},
↪ booktitle={2019 International Conference on Document Analysis and Recognition (ICDAR)}
↪, pages={134--141}, year={2019}, organization={IEEE}}
```


21.2.10 Scanned Receipts OCR and Information Extraction

"ICDAR2019 Competition on Scanned Receipt OCR and Information Extraction", *ICDAR*, 2019. PDF

A. Basic Info

- Official Website: sroie
- Year: 2019
- Language: ['English']
- Scene: ['Document']
- Annotation Granularity: ['Word']
- Supported Tasks: ['textdet', 'textrecog', 'textspotting']
- License: CC BY 4.0

Text Detection, Text Recognition and Text Spotting

```
# x1,y1,x2,y2,x3,y3,x4,y4,trans

72,25,326,25,326,64,72,64,TAN WOON YANN
50,82,440,82,440,121,50,121,BOOK TA .K(TAMAN DAYA) SDN BND
205,121,285,121,285,139,205,139,789417-W
```

C. Reference

```
@INPROCEEDINGS{8977955, author={Huang, Zheng and Chen, Kai and He, Jianhua and Bai,␣
↪Xiang and Karatzas, Dimosthenis and Lu, Shijian and Jawahar, C. V.}, booktitle={2019␣
↪International Conference on Document Analysis and Recognition (ICDAR)}, title=
↪{ICDAR2019 Competition on Scanned Receipt OCR and Information Extraction}, year={2019},
↪ volume={}, number={}, pages={1516-1520}, doi={10.1109/ICDAR.2019.00244}}
```

21.2.11 Street View Text Dataset (SVT)

> "Word Spotting in the Wild", *ECCV*, 2010. PDF

A. Basic Info

- Official Website: svt
- Year: 2010
- Language: ['English']
- Scene: ['Natural Scene']
- Annotation Granularity: ['Word']
- Supported Tasks: ['textdet', 'textrecog', 'textspotting']
- License: N/A

Text Detection/Recognition/Spotting

```
<image>
  <imageName>img/14_03.jpg</imageName>
  <address>341 Southwest 10th Avenue Portland OR</address>
  <lex>
  LIVING,ROOM,THEATERS,KENNY,ZUKE,DELICATESSEN,CLYDE,COMMON,ACE,HOTEL,PORTLAND,ROSE,CITY,
↪BOOKS,STUMPTOWN,COFFEE,ROASTERS,RED,CAP,GARAGE,FISH,GROTTO,SEAFOOD,RESTAURANT,AURA,
↪RESTAURANT,LOUNGE,ROCCO,PIZZA,PASTA,BUFFALO,EXCHANGE,MARK,SPENCER,LIGHT,FEZ,BALLROOM,
↪READING,FRENZY,ROXY,SCANDALS,MARTINOTTI,CAFE,DELI,CROWSENBERG,HALF
  </lex>
  <Resolution x="1280" y="880"/>
  <taggedRectangles>
    <taggedRectangle height="75" width="236" x="375" y="253">
      <tag>LIVING</tag>
    </taggedRectangle>
    <taggedRectangle height="76" width="175" x="639" y="272">
      <tag>ROOM</tag>
    </taggedRectangle>
    <taggedRectangle height="87" width="281" x="839" y="283">
      <tag>THEATERS</tag>
    </taggedRectangle>
```

(continues on next page)

(continued from previous page)

```
  </taggedRectangles>
</image>
```

C. Reference

```
@inproceedings{wang2010word, title={Word spotting in the wild}, author={Wang, Kai and␣
↪Belongie, Serge}, booktitle={European conference on computer vision}, pages={591--604},
↪ year={2010}, organization={Springer}}
```

21.2.12 Street View Text Perspective (SVT-P)

"Recognizing Text with Perspective Distortion in Natural Scenes", *ICCV*, 2013. PDF

A. Basic Info

- Official Website: svtp
- Year: 2013
- Language: ['English']
- Scene: ['Natural Scene']
- Annotation Granularity: ['Word']
- Supported Tasks: ['textrecog']
- License: N/A

Text Recognition

```
13_15_0_par.jpg WYNDHAM
13_15_1_par.jpg HOTEL
12_16_0_par.jpg UNITED
```

C. Reference

```
@inproceedings{phan2013recognizing, title={Recognizing text with perspective distortion␣
↪in natural scenes}, author={Phan, Trung Quy and Shivakumara, Palaiahnakote and Tian,␣
↪Shangxuan and Tan, Chew Lim}, booktitle={Proceedings of the IEEE International␣
↪Conference on Computer Vision}, pages={569--576}, year={2013}}
```

21.2.13 SynthText in the Wild Dataset

"Synthetic Data for Text Localisation in Natural Images", *CVPR*, 2016. PDF

A. Basic Info

- Official Website: synthtext
- Year: 2016
- Language: ['English']
- Scene: ['Synthesis']
- Annotation Granularity: ['Word', 'Character']
- Supported Tasks: ['textdet', 'textrecog', 'textspotting']

- License: Synthext Custom

Text Detection/Recognition/Spotting

```
{
    "imnames": [['8/ballet_106_0.jpg', ...]],
    "wordBB": [[[420.58957   418.85016   448.08478   410.3094    117.745026
                 322.30963   322.6857    159.09138   154.27284   260.14597
                 431.9315    427.52274   296.86508    99.56819   108.96211  ]
                [512.3321    431.88342   519.4515    499.81183   179.0544
                 377.97382   376.4993    203.64464   193.77492   313.61514
                 487.58023   484.64633   365.83176   142.49403   144.90457  ]
                [511.92203   428.7077    518.7375    499.0373    172.1684
                 378.35858   377.2078    203.3191    193.0739    319.69186
                 485.6758    482.571     365.76303   142.31898   144.43858  ]
                [420.1795    415.67444   447.3708    409.53485   110.859024
                 322.6944    323.3942    158.76585   153.57182   266.2227
                 430.02707   425.44742   296.79636    99.39314   108.49613  ]]

               [[ 21.06382    46.19922    47.570374   73.95366   197.17792
                   9.993624   48.437763    9.064571   49.659035  208.57095
                 118.41646   162.82489    29.548729    5.800581   28.812992 ]
                [ 23.069519   48.254295   50.130234   77.18146   208.71487
                   8.999153   46.69632     9.698633   50.869553  203.25742
                 122.64043   168.38647    29.660484    6.2558594  29.602367 ]
                [ 41.827087   68.39458    70.03627    98.65903   245.30832
                  30.534437   68.589294   32.57161    73.74529   264.40634
                 147.7303    189.70224    72.08       22.759935   50.81941  ]
                [ 39.82139    66.3395     67.47641    95.43123   233.77136
                  31.528908   70.33074    31.937548   72.534775  269.71988
                 143.50633   184.14066    71.96825    22.304657   50.030033 ]], ...],
    "charBB": [[[423.16126397 439.60847343 450.66887979 466.31976402 479.76190495
                 504.59927448 418.80489444 450.13965942 464.16775197 480.46891089
                 502.46437709 413.02373632 433.01396211 446.7222192  470.28467827
                 482.51674486 116.52285438 139.51408587 150.7448586  162.03366629
                 322.84717946 333.54881536 343.28386485 363.07416389 323.48968759
                 337.98503283 356.66355903 160.48517048 174.1707753  189.64454066
                 155.7637383  167.45490471 179.63644201 262.2183876  271.75848874
                 284.05396524 298.26103738 432.8464733  449.15387392 468.07231897
                 428.11482147 445.61538159 469.24565878 296.86441324 323.6603118
                 344.09880401 101.14677814 110.45423597 120.54555495 131.18342618
                 132.20545124 110.01673682 120.83144568 131.35885673]
                [438.2997574  452.61288403 466.31976402 482.22585715 498.3934528
                 512.20555863 431.88338084 466.11639619 481.73414937 499.62012025
                 519.36789779 432.51717267 449.23571387 465.73425964 484.45139112
                 499.59056304 140.27413679 149.59811175 160.13352083 169.59504507
                 333.55849014 344.33923741 361.08275796 378.09844418 339.92898685
                 355.57692063 376.51230484 174.1707753  189.07871028 203.64462646
                 165.22739457 181.27572412 193.60260894 270.99557614 283.13281739
                 298.75499435 313.61511672 447.1421735  470.27065563 487.02126631
                 446.97485257 468.98979567 484.64633864 317.88691577 341.16094163
                 365.8300006  111.15280603 120.54555495 130.72086821 135.27663717
                 142.4726875  120.1331955  133.07976304 144.75919258]
                [435.54895424 449.95797159 464.5848793  480.68235876 497.04793842
```

(continues on next page)

(continued from previous page)

```
  511.1101386  428.95660757 463.61882066 480.14247127 498.2535215
  518.03243928 429.36600266 447.19056345 463.89483785 482.21016814
  498.18529977 142.63162835 152.55587851 162.80539142 172.21885945
  333.35620309 344.09880401 360.86201193 377.82379299 339.7646859
  355.37508239 376.1110999  172.46032372 187.37816388 201.39094518
  163.04321987 178.99078221 191.89681939 275.3073355  286.08373072
  301.85539131 318.57227103 444.54207279 467.53925436 485.27070558
  444.57367155 466.90671029 482.56302723 317.62908407 340.9131681
  365.44465854 109.40501176 119.4999228  129.67892444 134.35253232
  140.97421069 118.61779828 131.34019115 143.25688164]
 [420.17946701 436.74150236 448.74896556 464.5848793  478.18853922
  503.4152019  415.67442461 447.3707845  462.35927516 478.8614766
  500.86810735 409.54560397 430.77026495 444.64606264 467.79077782
  480.89051912 119.14629674 142.63162835 153.56593297 164.78799774
  322.69436747 333.35620309 343.11884239 362.84714115 323.37931952
  337.83763574 356.35573621 158.76583616 172.46032372 187.37816388
  153.57183805 165.15781218 177.92125239 266.22269514 274.45156305
  286.82608962 302.69695881 430.02705241 446.01814255 466.05208347
  425.44741792 443.19481667 466.90671029 296.79634428 323.49707084
  343.82488703  99.39315359 109.40501176 119.4999228  130.25798537
  130.70149005 108.49612777 119.08444238 129.84935461]]

[[ 22.26958901  21.60559248  27.0241972   27.25747678  27.45783459
   28.73896576  47.91255579  47.80732383  53.77711568  54.24219042
   52.00169325  74.79043429  80.45929285  81.04748707  76.11658669
   82.58335942 203.67278213 201.2743445  205.59358622 205.51198143
   10.06536976  10.82312635  16.77203865  16.31842372  54.80444433
   54.66492     47.33822371  15.08534083  15.18716407   9.62607092
   51.06813224  50.18928243  56.16019366 220.78902143 236.08062638
  231.69267533 209.73652786 124.25352842 119.99631725 128.73732717
  165.78411123 167.31764153 167.05531699  29.97351822  31.5116502
   31.14650552   5.88513488  12.51324147  12.57920537   8.21515307
    8.21998849  35.66412031  29.17945741  36.00660903]
 [ 22.46075572  21.76391911  27.25747678  27.49456029  27.73554156
   28.85582217  48.25428361  48.21714995  54.27828788  54.78857757
   52.4595556   75.57743634  81.15533616  81.86325615  76.681392
   83.31596322 210.04771309 203.83983042 208.00417391 207.41791524
    9.79265706  10.55231862  16.36406888  15.97405105  54.64620856
   54.49559004  47.09756263  15.18716407  15.29808166   9.69862498
   51.27597632  50.48652154  56.49239954 216.92183074 232.02141018
  226.44624213 203.25738931 125.19349641 121.32658508 130.00428964
  167.43676857 169.36588297 168.38645076  29.58279603  31.19899202
   30.75826599   5.92344996  12.57920537  12.64571832   8.23451892
    8.26856497  35.82646468  29.342662    36.22165159]
 [ 40.15739982  40.47241401  40.79219178  41.14411963  41.50190876
   41.80934074  66.81590976  68.05921213  68.6519006   69.30152766
   70.01097963  96.14641662  96.04484417  96.89110144  97.81897661
   98.62829468 237.26055111 240.35280825 243.54641271 245.04022528
   31.33842788  31.14650552  30.84702178  30.54399042  69.80098672
   68.7212013   68.62479627  32.13243303  32.34474067  32.54416771
   72.82501686  73.31372392  73.70922459 267.74318222 265.39839711
  259.52741156 253.14023308 144.60810334 145.23371653 147.69958337
```

(continues on next page)

(continued from previous page)

```
                186.00278322 188.17713786 189.70144388  71.89351759  53.62266986
                 54.40060855  22.41084398  22.51791234  22.62587258  17.11356079
                 22.74567232  50.25232032  46.05692507  50.79345235]
               [ 39.82138755  40.18347166  40.44598236  40.79219178  41.08959901
                 41.64111176  66.33948982  67.47640971  68.01403337  68.60595247
                 69.3953105   95.13188979  95.21297344  95.91593691  97.08847413
                 97.75212171 229.94285119 237.26055111 240.66752705 242.74145162
                 31.52890731  31.33842788  31.16401306  30.81155638  69.87135926
                 68.80273568  68.71664209  31.93753588  32.13243303  32.34474067
                 72.53476992  72.88981775  73.28094858 269.71986636 267.92938572
                262.93698624 256.88902439 143.50635029 143.61251781 146.24080653
                184.14064261 185.86853729 188.17713786  71.96823746  53.79651809
                 54.60870874  22.30465649  22.41084398  22.51791234  17.07939535
                 22.63671808  50.03002471  45.81009198  50.49899163]], ...],
    "txt": [['Lines:\nI lost\nKevin ' 'will                ' 'line\nand           '
             'and\nthe             ' '(and                ' 'the\nout            '
             'you                ' "don't\n pkg          "], ...]
}
```

C. Reference

```
@InProceedings{Gupta16, author       = "Ankush Gupta and Andrea Vedaldi and Andrew␣
↪Zisserman", title        = "Synthetic Data for Text Localisation in Natural Images",␣
↪booktitle    = "IEEE Conference on Computer Vision and Pattern Recognition", year     ␣
↪    = "2016", }
```

21.2.14 Text OCR

"TextOCR: Towards large-scale end-to-end reasoning for arbitrary-shaped scene text", *CVPR*, 2021. PDF

A. Basic Info

- Official Website: textocr
- Year: 2021
- Language: ['English']
- Scene: ['Natural Scene']
- Annotation Granularity: ['Word']
- Supported Tasks: ['textdet', 'textrecog', 'textspotting']
- License: CC BY 4.0

Text Detection/Recognition/Spotting

```
{
  "imgs": {
    "OpenImages_ImageID_1": {
      "id": "OpenImages_ImageID_1",
      "width": "INT, Width of the image",
      "height": "INT, Height of the image",
      "set": "Split train|val|test",
```

(continues on next page)

(continued from previous page)

```
      "filename": "train|test/OpenImages_ImageID_1.jpg"
    },
    "OpenImages_ImageID_2": {
      "...": "..."
    }
  },
  "anns": {
    "OpenImages_ImageID_1_1": {
      "id": "STR, OpenImages_ImageID_1_1, Specifies the nth annotation for an image",
      "image_id": "OpenImages_ImageID_1",
      "bbox": [
        "FLOAT x1",
        "FLOAT y1",
        "FLOAT x2",
        "FLOAT y2"
      ],
      "points": [
        "FLOAT x1",
        "FLOAT y1",
        "FLOAT x2",
        "FLOAT y2",
        "...",
        "FLOAT xN",
        "FLOAT yN"
      ],
      "utf8_string": "text for this annotation",
      "area": "FLOAT, area of this box"
    },
    "OpenImages_ImageID_1_2": {
      "...": "..."
    },
    "OpenImages_ImageID_2_1": {
      "...": "..."
    }
  },
  "img2Anns": {
    "OpenImages_ImageID_1": [
      "OpenImages_ImageID_1_1",
      "OpenImages_ImageID_1_2",
      "OpenImages_ImageID_1_2"
    ],
    "OpenImages_ImageID_N": [
      "..."
    ]
  }
}
```

C. Reference

```
@inproceedings{singh2021textocr, title={{TextOCR}: Towards large-scale end-to-end␣
↪reasoning for arbitrary-shaped scene text}, author={Singh, Amanpreet and Pang, Guan␣
↪and Toh, Mandy and Huang, Jing and Galuba, Wojciech and Hassner, Tal}, journal={The␣
↪Conference on Computer Vision and Pattern Recognition}, year={2021}}
```

(continues on next page)

(continued from previous page)

21.2.15 Total Text

"Total-Text: Towards Orientation Robustness in Scene Text Detection", *IJDAR*, 2020. PDF

A. Basic Info

- Official Website: totaltext
- Year: 2020
- Language: ['English']
- Scene: ['Natural Scene']
- Annotation Granularity: ['Word']
- Supported Tasks: ['textdet', 'textrecog', 'textspotting']
- License: BSD-3

Text Detection/Spotting

```
x: [[259 313 389 427 354 302]], y: [[542 462 417 459 507 582]], ornt: [u'c'],␣
↪transcriptions: [u'PAUL']
x: [[400 478 494 436]], y: [[398 380 448 465]], ornt: [u'#'], transcriptions: [u'#']
```

C. Reference

```
@article{CK2019, author = {Chee Kheng Chng and Chee Seng Chan and Chenglin Liu}, title =
↪{Total-Text: Towards Orientation Robustness in Scene Text Detection}, journal =
↪{International Journal on Document Analysis and Recognition (IJDAR)}, volume = {23},␣
↪pages = {31-52}, year = {2020}, doi = {10.1007/s10032-019-00334-z}}
```

21.2.16 WildReceipt

"Spatial Dual-Modality Graph Reasoning for Key Information Extraction", *arXiv*, 2021. PDF

A. Basic Info

- Official Website: wildreceipt
- Year: 2021
- Language: ['English']
- Scene: ['Receipt']
- Annotation Granularity: ['Word']
- Supported Tasks: ['kie', 'textdet', 'textrecog', 'textspotting']
- License: N/A

KIE

```
// Close Set
{
  "file_name": "image_files/Image_16/11/d5de7f2a20751e50b84c747c17a24cd98bed3554.jpeg",
  "height": 1200,
  "width": 1600,
  "annotations":
    [
      {
        "box": [550.0, 190.0, 937.0, 190.0, 937.0, 104.0, 550.0, 104.0],
        "text": "SAFEWAY",
        "label": 1
      },
      {
        "box": [1048.0, 211.0, 1074.0, 211.0, 1074.0, 196.0, 1048.0, 196.0],
        "text": "TM",
        "label": 25
      }
    ], //...
}

// Open Set
{
  "file_name": "image_files/Image_12/10/845be0dd6f5b04866a2042abd28d558032ef2576.jpeg",
  "height": 348,
  "width": 348,
  "annotations":
    [
      {
        "box": [114.0, 19.0, 230.0, 19.0, 230.0, 1.0, 114.0, 1.0],
        "text": "CHOEUN",
        "label": 2,
        "edge": 1
      },
      {
        "box": [97.0, 35.0, 236.0, 35.0, 236.0, 19.0, 97.0, 19.0],
        "text": "KOREANRESTAURANT",
        "label": 2,
        "edge": 1
      }
    ]
}
```

C. Reference

```
@article{sun2021spatial, title={Spatial Dual-Modality Graph Reasoning for Key␣
↪Information Extraction}, author={Sun, Hongbin and Kuang, Zhanghui and Yue, Xiaoyu and␣
↪Lin, Chenhao and Zhang, Wayne}, journal={arXiv preprint arXiv:2103.14470}, year={2021}␣
↪}
```

CHAPTER

TWENTYTWO

DATASET PREPARER (BETA)

Note: Dataset Preparer is still in beta version and might not be stable enough. You are welcome to try it out and report any issues to us.

22.1 One-click data preparation script

MMOCR provides a unified one-stop data preparation script `prepare_dataset.py`.

Only one line of command is needed to complete the data download, decompression, format conversion, and basic configure generation.

```
python tools/dataset_converters/prepare_dataset.py [-h] [--nproc NPROC] [--task {textdet,
↪textrecog,textspotting,kie}] [--splits SPLITS [SPLITS ...]] [--lmdb] [--overwrite-cfg]␣
↪[--dataset-zoo-path DATASET_ZOO_PATH] datasets [datasets ...]
```

For example, the following command shows how to use the script to prepare the ICDAR2015 dataset for text detection task.

```
python tools/dataset_converters/prepare_dataset.py icdar2015 --task textdet --overwrite-
↪cfg
```

Also, the script supports preparing multiple datasets at the same time. For example, the following command shows how to prepare the ICDAR2015 and TotalText datasets for text recognition task.

```
python tools/dataset_converters/prepare_dataset.py icdar2015 totaltext --task textrecog -
↪-overwrite-cfg
```

To check the supported datasets of Dataset Preparer, please refer to *Dataset Zoo*. Some of other datasets that need to be prepared manually are listed in *Text Detection* and *Text Recognition*.

For users in China, more datasets can be downloaded from the opensource dataset platform: OpenDataLab. After downloading the data, you can place the files listed in `data_obtainer.save_name` in `data/cache` and rerun the script.

22.2 Advanced Usage

22.2.1 LMDB Format

In text recognition tasks, we usually use LMDB format to store data to speed up data loading. When using the `prepare_dataset.py` script to prepare data, you can store data to the LMDB format by the `--lmdb` parameter. For example:

```
python tools/dataset_converters/prepare_dataset.py icdar2015 --task textrecog --lmdb
```

As soon as the dataset is prepared, Dataset Preparer will generate `icdar2015_lmdb.py` in the `configs/textrecog/_base_/datasets/` directory. You can inherit this file and point the `dataloader` to the LMDB dataset. Moreover, the LMDB dataset needs to be loaded by `LoadImageFromNDArray`, thus you also need to modify `pipeline`.

For example, if we want to change the training set of `configs/textrecog/crnn/crnn_mini-vgg_5e_mj.py` to icdar2015 generated before, we need to perform the following modifications:

1. Modify `configs/textrecog/crnn/crnn_mini-vgg_5e_mj.py`:

```
_base_ = [
     '../_base_/datasets/icdar2015_lmdb.py',  # point to icdar2015 lmdb dataset
      ...
 ]

 train_list = [_base_.icdar2015_lmdb_textrecog_train]
 ...
```

2. Modify `train_pipeline` in `configs/textrecog/crnn/_base_crnn_mini-vgg.py`, change `LoadImageFromFile` to `LoadImageFromNDArray`:

```
train_pipeline = [
 dict(
     type='LoadImageFromNDArray',
     color_type='grayscale',
     file_client_args=file_client_args,
     ignore_empty=True,
     min_size=2),
 ...
]
```

22.3 Design

There are many OCR datasets with different languages, annotation formats, and scenarios. There are generally two ways to use these datasets: to quickly understand the relevant information about the dataset, or to use it to train models. To meet these two usage scenarios, MMOCR provides dataset automatic preparation scripts. The dataset automatic preparation script uses modular design, which greatly enhances scalability, and allows users to easily configure other public or private datasets. The configuration files for the dataset automatic preparation script are uniformly stored in the `dataset_zoo/` directory. Users can find all the configuration files for the dataset preparation scripts officially supported by MMOCR in this directory. The directory structure of this folder is as follows:

```
dataset_zoo/
├── icdar2015
```

(continues on next page)

(continued from previous page)

```
    ├── metafile.yml
    ├── sample_anno.md
    ├── textdet.py
    ├── textrecog.py
    └── textspotting.py
└── wildreceipt
    ├── metafile.yml
    ├── sample_anno.md
    ├── kie.py
    ├── textdet.py
    ├── textrecog.py
    └── textspotting.py
```

22.3.1 Dataset-related Information

The relevant information of a dataset includes the annotation format, annotation examples, and basic statistical information of the dataset. Although this information can be found on the official website of each dataset, it is scattered across various websites, and users need to spend a lot of time to discover the basic information of the dataset. Therefore, MMOCR has designed some paradigms to help users quickly understand the basic information of the dataset. MMOCR divides the relevant information of the dataset into two parts. One part is the basic information of the dataset, including the year of publication, the authors of the paper, and copyright information, etc. The other part is the annotation information of the dataset, including the annotation format and annotation examples. MMOCR provides a paradigm for each part, and contributors can fill in the basic information of the dataset according to the paradigm. This way, users can quickly understand the basic information of the dataset. Based on the basic information of the dataset, MMOCR provides a `metafile.yml` file, which contains the basic information of the corresponding dataset, including the year of publication, the authors of the paper, and copyright information, etc. In this way, users can quickly understand the basic information of the dataset. This file is not mandatory during the dataset preparation process (so users can ignore it when adding their own private datasets), but to better understand the information of various public datasets, MMOCR recommends that users read the corresponding metafile information before using the dataset preparation script to understand whether the characteristics of the dataset meet the user's needs. MMOCR uses ICDAR2015 as an example, and its sample content is shown below:

```
Name: 'Incidental Scene Text IC15'
Paper:
  Title: ICDAR 2015 Competition on Robust Reading
  URL: https://rrc.cvc.uab.es/files/short_rrc_2015.pdf
  Venue: ICDAR
  Year: '2015'
  BibTeX: '@inproceedings{karatzas2015icdar,
  title={ICDAR 2015 competition on robust reading},
  author={Karatzas, Dimosthenis and Gomez-Bigorda, Lluis and Nicolaou, Anguelos and␣
↪Ghosh, Suman and Bagdanov, Andrew and Iwamura, Masakazu and Matas, Jiri and Neumann,␣
↪Lukas and Chandrasekhar, Vijay Ramaseshan and Lu, Shijian and others},
  booktitle={2015 13th international conference on document analysis and recognition␣
↪(ICDAR)},
  pages={1156--1160},
  year={2015},
  organization={IEEE}}'
Data:
  Website: https://rrc.cvc.uab.es/?ch=4
  Language:
```

(continues on next page)

(continued from previous page)

```
  - English
Scene:
  - Natural Scene
Granularity:
  - Word
Tasks:
  - textdet
  - textrecog
  - textspotting
License:
  Type: CC BY 4.0
  Link: https://creativecommons.org/licenses/by/4.0/
```

Specifically, MMOCR lists the meaning of each field in the following table:

For the annotation information of the dataset, MMOCR provides a `sample_anno.md` file, which users can use as a template to fill in the annotation information of the dataset, so that users can quickly understand the annotation information of the dataset. MMOCR uses ICDAR2015 as an example, and the sample content is as follows:

````
  **Text Detection**

  ```text
  # x1,y1,x2,y2,x3,y3,x4,y4,trans

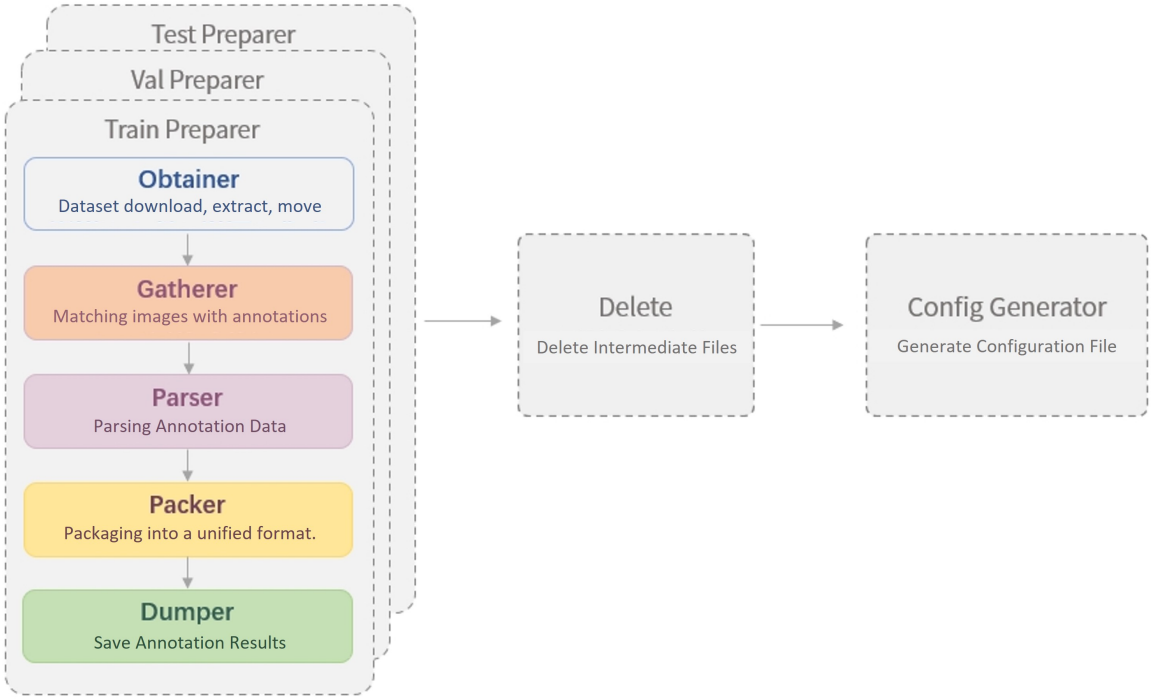
  377,117,463,117,465,130,378,130,Genaxis Theatre
  493,115,519,115,519,131,493,131,[06]
  374,155,409,155,409,170,374,170,###
````

`sample_anno.md` provides annotation information for different tasks of the dataset, including the format of the annotation files (text corresponds to `txt` files, and the format of the annotation files can also be found in `meta.yml`), and examples of the annotations.

With the information in these two files, users can quickly understand the basic information of the dataset. Additionally, MMOCR has summarized the basic information of all datasets, and users can view the basic information of all datasets in the Overview.

22.3.2 Dataset Usage

After decades of development, the OCR field has seen a series of related datasets emerge, often providing text annotation files in various styles, making it necessary for users to perform format conversion when using these datasets. Therefore, to facilitate dataset preparation for users, we have designed the Dataset Preparer to help users quickly prepare datasets in the format supported by MMOCR. For details, please refer to the *Dataset Format* document. The following figure shows a typical workflow for running the Dataset Preparer.
````



The figure shows that when running the Dataset Preparer, the following operations will be performed in sequence:

1. For the training set, validation set, and test set, the preparers will perform:
    1. Dataset download, extraction, and movement (Obtainer)
    2. *Matching annotations with images (Gatherer)*
    3. *Parsing original annotations (Parser)*
    4. *Packing annotations into a unified format (Packer)*
    5. *Saving annotations (Dumper)*
2. Delete files (Delete)
3. Generate the configuration file for the data set (Config Generator).

To handle various types of datasets, MMOCR has designed each component as a plug-and-play module, and allows users to configure the dataset preparation process through configuration files located in `dataset_zoo/`. These configuration files are in Python format and can be used in the same way as other configuration files in MMOCR, as described in the *Configuration File documentation*.

In `dataset_zoo/`, each dataset has its own folder, and the configuration files are named after the task to distinguish different configurations under different tasks. Taking the text detection part of ICDAR2015 as an example, the sample configuration file `dataset_zoo/icdar2015/textdet.py` is shown below:

```
data_root = 'data/icdar2015'
cache_path = 'data/cache'
train_preparer = dict(
    obtainer=dict(
        type='NaiveDataObtainer',
        cache_path=cache_path,
        files=[
            dict(
```

(continues on next page)

(continued from previous page)

```
                url='https://rrc.cvc.uab.es/downloads/ch4_training_images.zip',
                save_name='ic15_textdet_train_img.zip',
                md5='c51cbace155dcc4d98c8dd19d378f30d',
                content=['image'],
                mapping=[['ic15_textdet_train_img', 'textdet_imgs/train']]),
            dict(
                url='https://rrc.cvc.uab.es/downloads/'
                'ch4_training_localization_transcription_gt.zip',
                save_name='ic15_textdet_train_gt.zip',
                md5='3bfaf1988960909014f7987d2343060b',
                content=['annotation'],
                mapping=[['ic15_textdet_train_gt', 'annotations/train']]),
        ]),
    gatherer=dict(
        type='PairGatherer',
        img_suffixes=['.jpg', '.JPG'],
        rule=[r'img_(\d+)\.([jJ][pP][gG])', r'gt_img_\1.txt']),
    parser=dict(type='ICDARTxtTextDetAnnParser', encoding='utf-8-sig'),
    packer=dict(type='TextDetPacker'),
    dumper=dict(type='JsonDumper'),
)

test_preparer = dict(
    obtainer=dict(
        type='NaiveDataObtainer',
        cache_path=cache_path,
        files=[
            dict(
                url='https://rrc.cvc.uab.es/downloads/ch4_test_images.zip',
                save_name='ic15_textdet_test_img.zip',
                md5='97e4c1ddcf074ffcc75feff2b63c35dd',
                content=['image'],
                mapping=[['ic15_textdet_test_img', 'textdet_imgs/test']]),
            dict(
                url='https://rrc.cvc.uab.es/downloads/'
                'Challenge4_Test_Task4_GT.zip',
                save_name='ic15_textdet_test_gt.zip',
                md5='8bce173b06d164b98c357b0eb96ef430',
                content=['annotation'],
                mapping=[['ic15_textdet_test_gt', 'annotations/test']]),
        ]),
    gatherer=dict(
        type='PairGatherer',
        img_suffixes=['.jpg', '.JPG'],
        rule=[r'img_(\d+)\.([jJ][pP][gG])', r'gt_img_\1.txt']),
    parser=dict(type='ICDARTxtTextDetAnnParser', encoding='utf-8-sig'),
    packer=dict(type='TextDetPacker'),
    dumper=dict(type='JsonDumper'),
)

delete = ['annotations', 'ic15_textdet_test_img', 'ic15_textdet_train_img']
config_generator = dict(type='TextDetConfigGenerator')
```

Dataset download extraction and movement (Obtainer)

The `obtainer` module in Dataset Preparer is responsible for downloading, extracting, and moving the dataset. Currently, MMOCR only provides the `NaiveDataObtainer`. Generally speaking, the built-in `NaiveDataObtainer` is sufficient for downloading most datasets that can be accessed through direct links, and supports operations such as extraction, moving files, and renaming. However, MMOCR currently does not support automatically downloading datasets stored in resources that require login, such as Baidu or Google Drive. Here is a brief introduction to the `NaiveDataObtainer`.

The `files` field is a list, and each element in the list is a dictionary used to describe the download information of a dataset file. The table below shows the meaning of each field:

The Dataset Preparer follows the following conventions:

- Images of different types of datasets are moved to the corresponding category `{taskname}_imgs/{split}/` folder, such as `textdet_imgs/train/`.
- For a annotation file containing annotation information for all images, the annotations are moved to `annotations/{split}.*` file, such as `annotations/train.json`.
- For a annotation file containing annotation information for one image, all annotation files are moved to `annotations/{split}/` folder, such as `annotations/train/`.
- For some other special cases, such as all training, testing, and validation images are in one folder, the images can be moved to a self-set folder, such as `{taskname}_imgs/imgs/`, and the image storage location should be specified in the subsequent `gatherer` module.

An example configuration is as follows:

```
    obtainer=dict(
        type='NaiveDataObtainer',
        cache_path=cache_path,
        files=[
            dict(
                url='https://rrc.cvc.uab.es/downloads/ch4_training_images.zip',
                save_name='ic15_textdet_train_img.zip',
                md5='c51cbace155dcc4d98c8dd19d378f30d',
                content=['image'],
                mapping=[['ic15_textdet_train_img', 'textdet_imgs/train']]),
            dict(
                url='https://rrc.cvc.uab.es/downloads/'
                'ch4_training_localization_transcription_gt.zip',
                save_name='ic15_textdet_train_gt.zip',
                md5='3bfaf1988960909014f7987d2343060b',
                content=['annotation'],
                mapping=[['ic15_textdet_train_gt', 'annotations/train']]),
        ]),
```

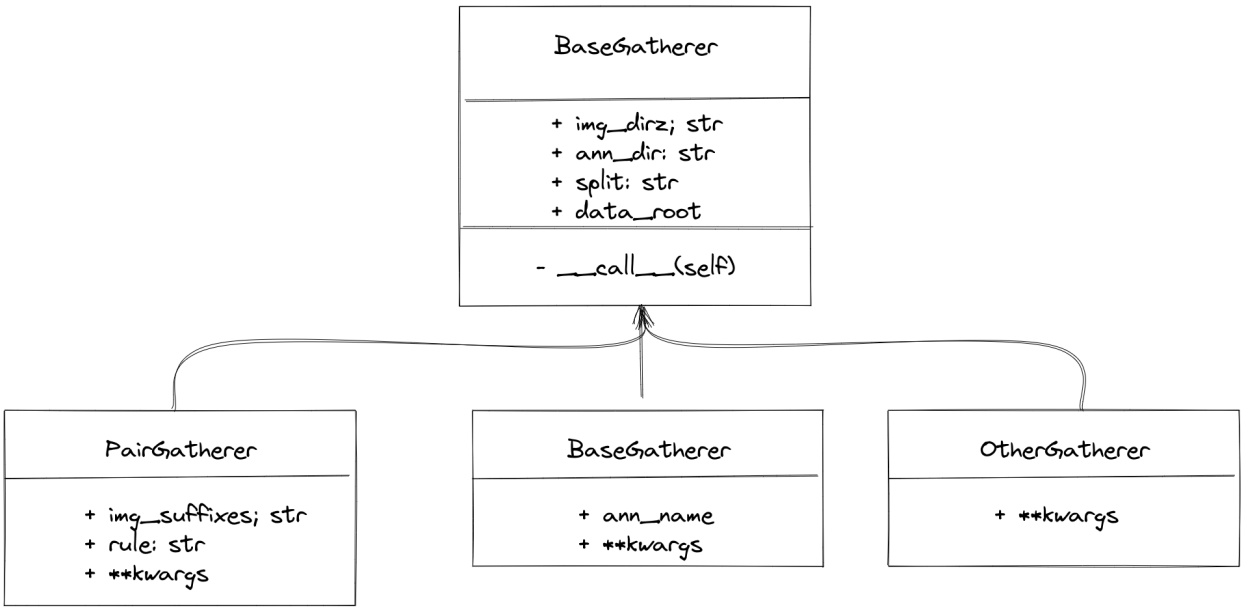
Dataset collection (Gatherer)

The `gatherer` module traverses the files in the dataset directory, matches image files with their corresponding annotation files, and organizes a file list for the `parser` module to read. Therefore, it is necessary to know the matching rules between image files and annotation files in the current dataset. There are two commonly used annotation storage formats for OCR datasets: one is multiple annotation files corresponding to multiple images, and the other is a single annotation file corresponding to multiple images, for example:

```
Many-to-Many
├── {taskname}_imgs/{split}/img_img_1.jpg
├── annotations/{split}/gt_img_1.txt
├── {taskname}_imgs/{split}/img_2.jpg
├── annotations/{split}/gt_img_2.txt
├── {taskname}_imgs/{split}/img_3.JPG
├── annotations/{split}/gt_img_3.txt

One-to-Many
├── {taskname}/{split}/img_1.jpg
├── {taskname}/{split}/img_2.jpg
├── {taskname}/{split}/img_3.JPG
├── annotations/gt.txt
```

Specific design is as follows:



MMOCR has built-in `PairGatherer` and `MonoGatherer` to handle the two common cases mentioned above. `PairGatherer` is used for many-to-many situations, while `MonoGatherer` is used for one-to-many situations.

Note: To simplify processing, the gatherer assumes that the dataset's images and annotations are stored separately in `{taskname}_imgs/{split}/` and `annotations/`, respectively. In particular, for many-to-many situations, the annotation file needs to be placed in `annotations/{split}`.

- In the many-to-many case, `PairGatherer` needs to find the image files and corresponding annotation files according to a certain naming convention. First, the suffix of the image needs to be specified by the `img_suffixes`

parameter, as in the example above `img_suffixes=[.jpg,.JPG]`. In addition, a pair of regular expressions `rule` is used to specify the correspondence between the image and annotation files. For example, `rule=[r'img_(\d+)\.([jJ][pP][gG])'r'gt_img_\1.txt']`. The first regular expression is used to match the image file name, `\d+` is used to match the image sequence number, and `([jJ][pP][gG])` is used to match the image suffix. The second regular expression is used to match the annotation file name, where `\1` associates the matched image sequence number with the annotation file sequence number. An example configuration is:

```
    gatherer=dict(
        type='PairGatherer',
        img_suffixes=['.jpg', '.JPG'],
        rule=[r'img_(\d+)\.([jJ][pP][gG])', r'gt_img_\1.txt']),
```

For the case of one-to-many, it is usually simple, and the user only needs to specify the annotation file name. For example, for the training set configuration:

```
    gatherer=dict(type='MonoGatherer', ann_name='train.txt'),
```

MMOCR has also made conventions on the return value of `Gatherer`. `Gatherer` returns a tuple with two elements. The first element is a list of image paths (including all image paths) or the folder containing all images. The second element is a list of annotation file paths (including all annotation file paths) or the path of the annotation file (the annotation file contains all image annotation information). Specifically, the return value of `PairGatherer` is (list of image paths, list of annotation file paths), as shown below:

```
    (['{taskname}_imgs/{split}/img_1.jpg', '{taskname}_imgs/{split}/img_2.jpg', '
↪{taskname}_imgs/{split}/img_3.JPG'],
    ['annotations/{split}/gt_img_1.txt', 'annotations/{split}/gt_img_2.txt',
↪'annotations/{split}/gt_img_3.txt'])
```

`MonoGatherer` returns a tuple containing the path to the image directory and the path to the annotation file, as follows:

```
    ('{taskname}/{split}', 'annotations/gt.txt')
```

Dataset parsing (Parser)

`Parser` is mainly used to parse the original annotation files. Since the original annotation formats vary greatly, MMOCR provides `BaseParser` as a base class, which users can inherit to implement their own `Parser`. In `BaseParser`, MMOCR has designed two interfaces: `parse_files` and `parse_file`, where the annotation parsing is conventionally carried out. For the two different input situations of `Gatherer` (many-to-many, one-to-many), the implementations of these two interfaces should be different.

- `BaseParser` by default handles the many-to-many situation. Among them, `parse_files` distributes the data in parallel to multiple `parse_file` processes, and each `parse_file` parses the annotation of a single image separately.
- For the one-to-many situation, the user needs to override `parse_files` to implement loading the annotation and returning standardized results.

The interface of `BaseParser` is defined as follows:

```
class BaseParser:
    def __call__(self, img_paths, ann_paths):
            return self.parse_files(img_paths, ann_paths)

    def parse_files(self, img_paths: Union[List[str], str],
```

(continues on next page)

(continued from previous page)

```
                    ann_paths: Union[List[str], str]) -> List[Tuple]:
        samples = track_parallel_progress_multi_args(
            self.parse_file, (img_paths, ann_paths), nproc=self.nproc)
        return samples

    @abstractmethod
    def parse_file(self, img_path: str, ann_path: str) -> Tuple:

        raise NotImplementedError
```

In order to ensure the uniformity of subsequent modules, MMOCR has made conventions for the return values of `parse_files` and `parse_file`. The return value of `parse_file` is a tuple, the first element of which is the image path, and the second element is the annotation information. The annotation information is a list, each element of which is a dictionary with the fields `poly`, `text`, and `ignore`, as shown below:

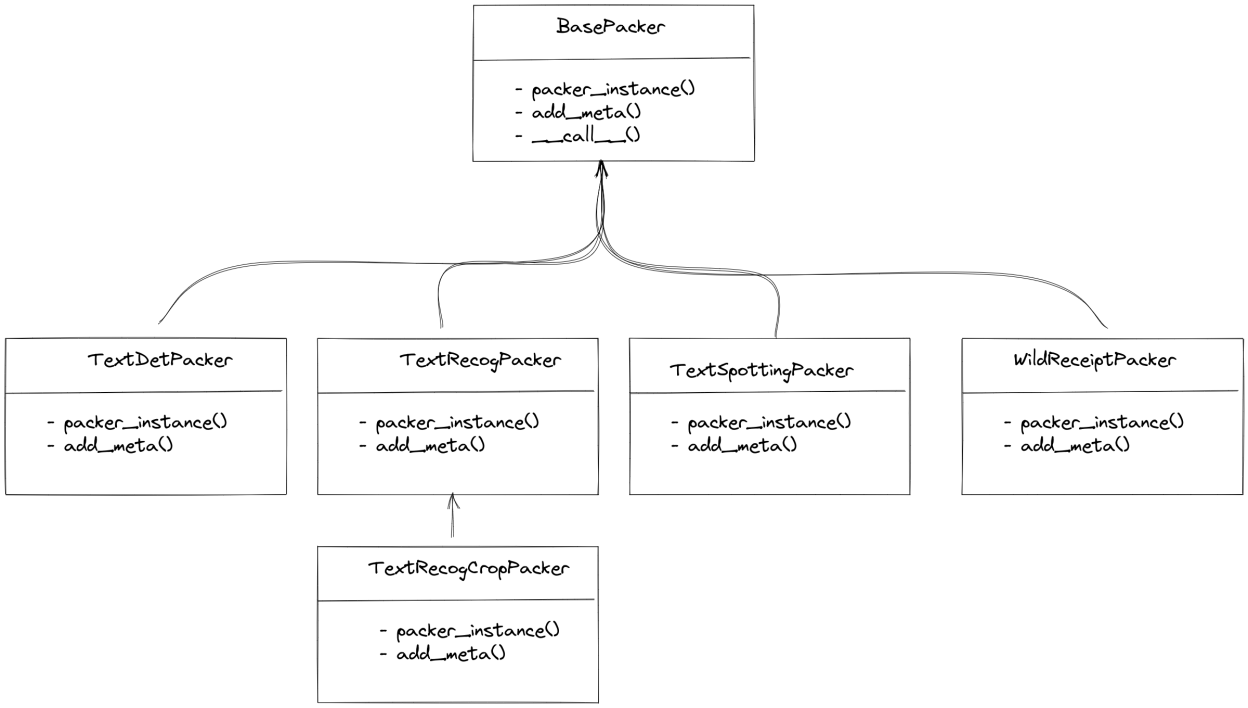
```
# An example of returned values:
(
    'imgs/train/xxx.jpg',
    [
        dict(
            poly=[0, 1, 1, 1, 1, 0, 0, 0],
            text='hello',
            ignore=False),
        ...
    ]
)
```

The output of `parse_files` is a list, and each element in the list is the return value of `parse_file`. An example is:

```
[
    (
        'imgs/train/xxx.jpg',
        [
            dict(
                poly=[0, 1, 1, 1, 1, 0, 0, 0],
                text='hello',
                ignore=False),
            ...
        ]
    ),
    ...
]
```

Dataset Conversion (Packer)

`Packer` is mainly used to convert data into a unified annotation format, because the input data is the output of parsers and the format has been fixed. Therefore, the packer only needs to convert the input format into a unified annotation format for each task. Currently, MMOCR supports tasks such as text detection, text recognition, end-to-end OCR, and key information extraction, and MMOCR has a corresponding packer for each task, as shown below:



For text detection, end-to-end OCR, and key information extraction, MMOCR has a unique corresponding `Packer`. However, for text recognition, MMOCR provides two `Packer` options: `TextRecogPacker` and `TextRecogCropPacker`, due to the existence of two types of datasets:

- Each image is a recognition sample, and the annotation information returned by the `parser` is only a `dict(text='xxx')`. In this case, `TextRecogPacker` can be used.
- The dataset does not crop text from the image, and it essentially contains end-to-end OCR annotations that include the position information of the text and the corresponding text information. `TextRecogCropPacker` will crop the text from the image and then convert it into the unified format for text recognition.

Annotation Saving (Dumper)

The `dumper` module is used to determine what format the data should be saved in. Currently, MMOCR supports `JsonDumper`, `WildreceiptOpensetDumper`, and `TextRecogLMDBDumper`. They are used to save data in the standard MMOCR JSON format, the Wildreceipt format, and the LMDB format commonly used in the academic community for text recognition, respectively.

Delete files (Delete)

When processing a dataset, temporary files that are not needed may be generated. Here, a list of such files or folders can be passed in, which will be deleted when the conversion is finished.

Generate the configuration file for the dataset (ConfigGenerator)

In order to automatically generate basic configuration files after preparing the dataset, MMOCR has implemented `TextDetConfigGenerator`, `TextRecogConfigGenerator`, and `TextSpottingConfigGenerator` for each task. The main parameters supported by these generators are as follows:

After preparing all the files for the dataset, the configuration generator will automatically generate the basic configuration files required to call the dataset. Below is a minimal example of a `TextDetConfigGenerator` configuration:

```
config_generator = dict(type='TextDetConfigGenerator')
```

The generated file will be placed by default under `configs/{task}/_base_/datasets/`. In this example, the basic configuration file for the ICDAR 2015 dataset will be generated at `configs/textdet/_base_/datasets/icdar2015.py`.

```
icdar2015_textdet_data_root = 'data/icdar2015'

icdar2015_textdet_train = dict(
    type='OCRDataset',
    data_root=icdar2015_textdet_data_root,
    ann_file='textdet_train.json',
    filter_cfg=dict(filter_empty_gt=True, min_size=32),
    pipeline=None)

icdar2015_textdet_test = dict(
    type='OCRDataset',
    data_root=icdar2015_textdet_data_root,
    ann_file='textdet_test.json',
    test_mode=True,
    pipeline=None)
```

If the dataset is special and there are several variants of the annotations, the configuration generator also supports generating variables pointing to each variant in the base configuration. However, this requires users to differentiate them by using different `dataset_postfix` when setting up. For example, the ICDAR 2015 text recognition dataset has two annotation versions for the test set, the original version and the 1811 version, which can be specified in `test_anns` as follows:

```
config_generator = dict(
    type='TextRecogConfigGenerator',
    test_anns=[
        dict(ann_file='textrecog_test.json'),
```

(continues on next page)

(continued from previous page)

```
        dict(dataset_postfix='857', ann_file='textrecog_test_857.json')
    ])
```

The configuration generator will generate the following configurations:

```
icdar2015_textrecog_data_root = 'data/icdar2015'

icdar2015_textrecog_train = dict(
    type='OCRDataset',
    data_root=icdar2015_textrecog_data_root,
    ann_file='textrecog_train.json',
    pipeline=None)

icdar2015_textrecog_test = dict(
    type='OCRDataset',
    data_root=icdar2015_textrecog_data_root,
    ann_file='textrecog_test.json',
    test_mode=True,
    pipeline=None)

icdar2015_1811_textrecog_test = dict(
    type='OCRDataset',
    data_root=icdar2015_textrecog_data_root,
    ann_file='textrecog_test_1811.json',
    test_mode=True,
    pipeline=None)
```

With this file, MMOCR can directly import this dataset into the `dataloader` from the model configuration file (the following sample is excerpted from `configs/textdet/dbnet/dbnet_resnet18_fpnc_1200e_icdar2015.py`):

```
_base_ = [
    '../_base_/datasets/icdar2015.py',
    # ...
]

# dataset settings
icdar2015_textdet_train = _base_.icdar2015_textdet_train
icdar2015_textdet_test = _base_.icdar2015_textdet_test
# ...

train_dataloader = dict(
    dataset=icdar2015_textdet_train)

val_dataloader = dict(
    dataset=icdar2015_textdet_test)

test_dataloader = val_dataloader
```

Note: By default, the configuration generator does not overwrite existing base configuration files unless the user manually specifies `overwrite-cfg` when running the script.

22.4 Adding a new dataset to Dataset Preparer

22.4.1 Adding Public Datasets

MMOCR has already supported many *commonly used public datasets*. If the dataset you want to use has not been supported yet and you are willing to *contribute to the MMOCR* open-source community, you can follow the steps below to add a new dataset.

In the following example, we will show you how to add the **ICDAR2013** dataset step by step.

Adding `metafile.yml`

First, make sure that the dataset you want to add does not already exist in `dataset_zoo/`. Then, create a new folder named after the dataset you want to add, such as `icdar2013/` (usually, use lowercase alphanumeric characters without symbols to name the dataset). In the `icdar2013/` folder, create a `metafile.yml` file and fill in the basic information of the dataset according to the following template:

```
Name: 'Incidental Scene Text IC13'
Paper:
  Title: ICDAR 2013 Robust Reading Competition
  URL: https://www.imlab.jp/publication_data/1352/icdar_competition_report.pdf
  Venue: ICDAR
  Year: '2013'
  BibTeX: '@inproceedings{karatzas2013icdar,
  title={ICDAR 2013 robust reading competition},
  author={Karatzas, Dimosthenis and Shafait, Faisal and Uchida, Seiichi and Iwamura,␣
↪Masakazu and i Bigorda, Lluis Gomez and Mestre, Sergi Robles and Mas, Joan and Mota,␣
↪David Fernandez and Almazan, Jon Almazan and De Las Heras, Lluis Pere},
  booktitle={2013 12th international conference on document analysis and recognition},
  pages={1484--1493},
  year={2013},
  organization={IEEE}}'
Data:
  Website: https://rrc.cvc.uab.es/?ch=2
  Language:
    - English
  Scene:
    - Natural Scene
  Granularity:
    - Word
  Tasks:
    - textdet
    - textrecog
    - textspotting
  License:
    Type: N/A
    Link: N/A
  Format: .txt
  Keywords:
    - Horizontal
```

Add Annotation Examples

Finally, you can add an annotation example file `sample_anno.md` under the `dataset_zoo/icdar2013/` directory to help the documentation script add annotation examples when generating documentation. The annotation example file is a Markdown file that typically contains the raw data format of a single sample. For example, the following code block shows a sample data file for the ICDAR2013 dataset:

````
  **Text Detection**

  ```text
  # train split
  # x1 y1 x2 y2 "transcript"

  158 128 411 181 "Footpath"
  443 128 501 169 "To"
  64 200 363 243 "Colchester"

  # test split
  # x1, y1, x2, y2, "transcript"

  38, 43, 920, 215, "Tiredness"
  275, 264, 665, 450, "kills"
  0, 699, 77, 830, "A"
````

Add configuration files for corresponding tasks

In the `dataset_zoo/icdar2013` directory, add a `.py` configuration file named after the task. For example, `textdet.py`, `textrecog.py`, `textspotting.py`, `kie.py`, etc. The configuration template is shown below:

```
data_root = ''
data_cache = 'data/cache'
train_prepare = dict(
    obtainer=dict(
        type='NaiveObtainer',
        data_cache=data_cache,
        files=[
            dict(
                url='xx',
                md5='',
                save_name='xxx',
                mapping=list())
            ]),
    gatherer=dict(type='xxxGatherer', **kwargs),
    parser=dict(type='xxxParser', **kwargs),
    packer=dict(type='TextxxxPacker'), # Packer for the task
    dumper=dict(type='JsonDumper'),
)
test_prepare = dict(
    obtainer=dict(
        type='NaiveObtainer',
        data_cache=data_cache,
        files=[
```

(continues on next page)
```

(continued from previous page)

```
            dict(
                url='xx',
                md5='',
                save_name='xxx',
                mapping=list())
              ]),
    gatherer=dict(type='xxxGatherer', **kwargs),
    parser=dict(type='xxxParser', **kwargs),
    packer=dict(type='TextxxxPacker'), # Packer for the task
    dumper=dict(type='JsonDumper'),
)
```

Taking the file detection task as an example, let's introduce the specific content of the configuration file. In general, users do not need to implement new `obtainer`, `gatherer`, `packer`, or `dumper`, but usually need to implement a new `parser` according to the annotation format of the dataset.

Regarding the configuration of `obtainer`, we will not go into detail here, and you can refer to Data set download, extraction, and movement (Obtainer).

For the `gatherer`, by observing the obtained ICDAR2013 dataset files, we found that each image has a corresponding `.txt` format annotation file:

```
data_root
├── textdet_imgs/train/
│   ├── img_1.jpg
│   ├── img_2.jpg
│   └── ...
├── annotations/train/
│   ├── gt_img_1.txt
│   ├── gt_img_2.txt
│   └── ...
```

Moreover, the name of each annotation file corresponds to the image: `gt_img_1.txt` corresponds to `img_1.jpg`, and so on. Therefore, `PairGatherer` can be used to match them.

```
gatherer=dict(
      type='PairGatherer',
      img_suffixes=['.jpg'],
      rule=[r'(\w+)\.jpg', r'gt_\1.txt'])
```

The first regular expression in the rule is used to match the image file name, and the second regular expression is used to match the annotation file name. Here, `(\w+)` is used to match the image file name, and `gt_\1.txt` is used to match the annotation file name, where `\1` represents the content matched by the first regular expression. That is, it replaces `img_xx.jpg` with `gt_img_xx.txt`.

Next, you need to implement a `parser` to parse the original annotation files into a standard format. Usually, before adding a new dataset, users can browse the *details page* of the supported datasets and check if there is a dataset with the same format. If there is, you can use the parser of that dataset directly. Otherwise, you need to implement a new format parser.

Data format parsers are stored in the `mmocr/datasets/preparers/parsers` directory. All parsers need to inherit from `BaseParser` and implement the `parse_file` or `parse_files` method. For more information, please refer to *Parsing original annotations (Parser)*.

By observing the annotation files of the ICDAR2013 dataset:

```
158 128 411 181 "Footpath"
443 128 501 169 "To"
64 200 363 243 "Colchester"
542, 710, 938, 841, "break"
87, 884, 457, 1021, "could"
517, 919, 831, 1024, "save"
```

We found that the built-in `ICDARTxtTextDetAnnParser` already meets the requirements, so we can directly use this parser and configure it in the `preparer`.

```
parser=dict(
     type='ICDARTxtTextDetAnnParser',
     remove_strs=[',', '"'],
     encoding='utf-8',
     format='x1 y1 x2 y2 trans',
     separator=' ',
     mode='xyxy')
```

In the configuration for the `ICDARTxtTextDetAnnParser`, `remove_strs=[',', '"']` is specified to remove extra quotes and commas in the annotation files. In the `format` section, `x1 y1 x2 y2 trans` indicates that each line in the annotation file contains four coordinates and a text content separated by spaces (`separator=' '`). Also, `mode` is set to `xyxy`, which means that the coordinates in the annotation file are the coordinates of the top-left and bottom-right corners, so that `ICDARTxtTextDetAnnParser` can parse the annotations into a unified format.

For the `packer`, taking the file detection task as an example, its `packer` is `TextDetPacker`, and its configuration is as follows:

```
packer=dict(type='TextDetPacker')
```

Finally, specify the `dumper`, which is generally saved in json format. Its configuration is as follows:

```
dumper=dict(type='JsonDumper')
```

After the above configuration, the configuration file for the ICDAR2013 training set is as follows:

```
train_preparer = dict(
    obtainer=dict(
        type='NaiveDataObtainer',
        cache_path=cache_path,
        files=[
            dict(
                url='https://rrc.cvc.uab.es/downloads/'
                'Challenge2_Training_Task12_Images.zip',
                save_name='ic13_textdet_train_img.zip',
                md5='a443b9649fda4229c9bc52751bad08fb',
                content=['image'],
                mapping=[['ic13_textdet_train_img', 'textdet_imgs/train']]),
            dict(
                url='https://rrc.cvc.uab.es/downloads/'
                'Challenge2_Training_Task1_GT.zip',
                save_name='ic13_textdet_train_gt.zip',
                md5='f3a425284a66cd67f455d389c972cce4',
                content=['annotation'],
                mapping=[['ic13_textdet_train_gt', 'annotations/train']]),
```

(continues on next page)

(continued from previous page)

```
        ]),
    gatherer=dict(
        type='PairGatherer',
        img_suffixes=['.jpg'],
        rule=[r'(\w+)\.jpg', r'gt_\1.txt']),
    parser=dict(
        type='ICDARTxtTextDetAnnParser',
        remove_strs=[',', '"'],
        format='x1 y1 x2 y2 trans',
        separator=' ',
        mode='xyxy'),
    packer=dict(type='TextDetPacker'),
    dumper=dict(type='JsonDumper'),
)
```

To automatically generate the basic configuration after the dataset is prepared, you also need to configure the corresponding task's `config_generator`.

In this example, since it is a text detection task, you only need to set the generator to `TextDetConfigGenerator`.

```
config_generator = dict(type='TextDetConfigGenerator')
```

22.4.2 Use DataPreparer to prepare customized dataset

[Coming Soon]

CHAPTER

TWENTYTHREE

TEXT DETECTION

Note: This page is a manual preparation guide for datasets not yet supported by *Dataset Preparer*, which all these scripts will be eventually migrated into.

23.1 Overview

23.1.1 Install AWS CLI (optional)

- Since there are some datasets that require the AWS CLI to be installed in advance, we provide a quick installation guide here:

```
curl "https://awscli.amazonaws.com/awscli-exe-linux-x86_64.zip" -o "awscliv2.zip"
unzip awscliv2.zip
sudo ./aws/install
./aws/install -i /usr/local/aws-cli -b /usr/local/bin
!aws configure
# this command will require you to input keys, you can skip them except
# for the Default region name
# AWS Access Key ID [None]:
# AWS Secret Access Key [None]:
# Default region name [None]: us-east-1
# Default output format [None]
```

For users in China, these datasets can also be downloaded from OpenDataLab with high speed:

- CTW1500
- ICDAR2013
- ICDAR2015
- Totaltext
- MSRA-TD500

23.2 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 pipeline config for example)

23.3 ICDAR 2011 (Born-Digital Images)

- Step1: Download `Challenge1_Training_Task12_Images.zip`, `Challenge1_Training_Task1_GT.zip`, `Challenge1_Test_Task12_Images.zip`, and `Challenge1_Test_Task1_GT.zip` from homepage `Task 1. 1: Text Localization (2013 edition)`.

```
mkdir icdar2011 && cd icdar2011
mkdir imgs && mkdir annotations

# Download ICDAR 2011
wget https://rrc.cvc.uab.es/downloads/Challenge1_Training_Task12_Images.zip --no-
↪check-certificate
wget https://rrc.cvc.uab.es/downloads/Challenge1_Training_Task1_GT.zip --no-check-
↪certificate
wget https://rrc.cvc.uab.es/downloads/Challenge1_Test_Task12_Images.zip --no-check-
↪certificate
wget https://rrc.cvc.uab.es/downloads/Challenge1_Test_Task1_GT.zip --no-check-
↪certificate

# For images
unzip -q Challenge1_Training_Task12_Images.zip -d imgs/training
unzip -q Challenge1_Test_Task12_Images.zip -d imgs/test
# For annotations
unzip -q Challenge1_Training_Task1_GT.zip -d annotations/training
unzip -q Challenge1_Test_Task1_GT.zip -d annotations/test

rm Challenge1_Training_Task12_Images.zip && rm Challenge1_Test_Task12_Images.zip &&␣
↪rm Challenge1_Training_Task1_GT.zip && rm Challenge1_Test_Task1_GT.zip
```

- Step 2: Generate `instances_training.json` and `instances_test.json` with the following command:

```
python tools/dataset_converters/textdet/ic11_converter.py PATH/TO/icdar2011 --nproc␣
↪4
```

- After running the above codes, the directory structure should be as follows:

```
├── icdar2011
│   ├── imgs
│   ├── instances_test.json
│   └── instances_training.json
```

23.4 ICDAR 2017

- Follow similar steps as ICDAR 2015.
- The resulting directory structure looks like the following:

```
├── icdar2017
│   ├── imgs
│   ├── annotations
│   ├── instances_training.json
│   └── instances_val.json
```

23.5 CurvedSynText150k

- Step1: Download syntext1.zip and syntext2.zip to `CurvedSynText150k/`.
- Step2:

```
unzip -q syntext1.zip
mv train.json train1.json
unzip images.zip
rm images.zip

unzip -q syntext2.zip
mv train.json train2.json
unzip images.zip
rm images.zip
```

- Step3: Download instances_training.json to `CurvedSynText150k/`
- Or, generate `instances_training.json` with following command:

```
python tools/dataset_converters/common/curvedsyntext_converter.py PATH/TO/
↪CurvedSynText150k --nproc 4
```

- The resulting directory structure looks like the following:

```
├── CurvedSynText150k
│   ├── syntext_word_eng
│   ├── emcs_imgs
│   └── instances_training.json
```

23.6 DeText

- Step1: Download `ch9_training_images.zip`, `ch9_training_localization_transcription_gt.zip`, `ch9_validation_images.zip`, and `ch9_validation_localization_transcription_gt.zip` from **Task 3: End to End** on the homepage.

```
mkdir detext && cd detext
mkdir imgs && mkdir annotations && mkdir imgs/training && mkdir imgs/val && mkdir␣
↪annotations/training && mkdir annotations/val
```

(continues on next page)

(continued from previous page)

```
# Download DeText
wget https://rrc.cvc.uab.es/downloads/ch9_training_images.zip --no-check-certificate
wget https://rrc.cvc.uab.es/downloads/ch9_training_localization_transcription_gt.
↪zip --no-check-certificate
wget https://rrc.cvc.uab.es/downloads/ch9_validation_images.zip --no-check-
↪certificate
wget https://rrc.cvc.uab.es/downloads/ch9_validation_localization_transcription_gt.
↪zip --no-check-certificate

# Extract images and annotations
unzip -q ch9_training_images.zip -d imgs/training && unzip -q ch9_training_
↪localization_transcription_gt.zip -d annotations/training && unzip -q ch9_
↪validation_images.zip -d imgs/val && unzip -q ch9_validation_localization_
↪transcription_gt.zip -d annotations/val

# Remove zips
rm ch9_training_images.zip && rm ch9_training_localization_transcription_gt.zip &&␣
↪rm ch9_validation_images.zip && rm ch9_validation_localization_transcription_gt.
↪zip
```

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

```
python tools/dataset_converters/textdet/detext_converter.py PATH/TO/detext --nproc 4
```

- After running the above codes, the directory structure should be as follows:

```
├── detext
│   ├── annotations
│   ├── imgs
│   ├── instances_test.json
│   └── instances_training.json
```

23.7 Lecture Video DB

- Step1: Download IIIT-CVid.zip to `lv/`.

```
mkdir lv && cd lv

# Download LV dataset
wget http://cdn.iiit.ac.in/cdn/preon.iiit.ac.in/~kartik/IIIT-CVid.zip
unzip -q IIIT-CVid.zip

mv IIIT-CVid/Frames imgs

rm IIIT-CVid.zip
```

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

```
python tools/dataset_converters/textdet/lv_converter.py PATH/TO/lv --nproc 4
```

- The resulting directory structure looks like the following:

```
├── lv
│   ├── imgs
│   ├── instances_test.json
│   ├── instances_training.json
│   └── instances_val.json
```

23.8 LSVT

- Step1: Download train_full_images_0.tar.gz, train_full_images_1.tar.gz, and train_full_labels.json to `lsvt/`.

```
mkdir lsvt && cd lsvt

# Download LSVT dataset
wget https://dataset-bj.cdn.bcebos.com/lsvt/train_full_images_0.tar.gz
wget https://dataset-bj.cdn.bcebos.com/lsvt/train_full_images_1.tar.gz
wget https://dataset-bj.cdn.bcebos.com/lsvt/train_full_labels.json

mkdir annotations
tar -xf train_full_images_0.tar.gz && tar -xf train_full_images_1.tar.gz
mv train_full_labels.json annotations/ && mv train_full_images_1/*.jpg train_full_
↪images_0/
mv train_full_images_0 imgs

rm train_full_images_0.tar.gz && rm train_full_images_1.tar.gz && rm -rf train_full_
↪images_1
```

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

```
# Annotations of LSVT test split is not publicly available, split a validation
# set by adding --val-ratio 0.2
python tools/dataset_converters/textdet/lsvt_converter.py PATH/TO/lsvt
```

- After running the above codes, the directory structure should be as follows:

```
├── lsvt
│   ├── imgs
│   ├── instances_training.json
│   └── instances_val.json (optional)
```

23.9 IMGUR

- Step1: Run `download_imgur5k.py` to download images. You can merge PR#5 in your local repository to enable a **much faster** parallel execution of image download.

```
mkdir imgur && cd imgur

git clone https://github.com/facebookresearch/IMGUR5K-Handwriting-Dataset.git

# Download images from imgur.com. This may take SEVERAL HOURS!
python ./IMGUR5K-Handwriting-Dataset/download_imgur5k.py --dataset_info_dir ./
↪IMGUR5K-Handwriting-Dataset/dataset_info/ --output_dir ./imgs

# For annotations
mkdir annotations
mv ./IMGUR5K-Handwriting-Dataset/dataset_info/*.json annotations

rm -rf IMGUR5K-Handwriting-Dataset
```

- Step2: Generate `instances_train.json`, `instance_val.json` and `instances_test.json` with the following command:

```
python tools/dataset_converters/textdet/imgur_converter.py PATH/TO/imgur
```

- After running the above codes, the directory structure should be as follows:

```
├── imgur
│   ├── annotations
│   ├── imgs
│   ├── instances_test.json
│   ├── instances_training.json
│   └── instances_val.json
```

23.10 KAIST

- Step1: Complete download KAIST_all.zip to `kaist/`.

```
mkdir kaist && cd kaist
mkdir imgs && mkdir annotations

# Download KAIST dataset
wget http://www.iapr-tc11.org/dataset/KAIST_SceneText/KAIST_all.zip
unzip -q KAIST_all.zip

rm KAIST_all.zip
```

- Step2: Extract zips:

```
python tools/dataset_converters/common/extract_kaist.py PATH/TO/kaist
```

- Step3: Generate `instances_training.json` and `instances_val.json` (optional) with following command:

```
# Since KAIST does not provide an official split, you can split the dataset by␣
↪adding --val-ratio 0.2
python tools/dataset_converters/textdet/kaist_converter.py PATH/TO/kaist --nproc 4
```

- After running the above codes, the directory structure should be as follows:

```
├── kaist
│   ├── annotations
│   ├── imgs
│   ├── instances_training.json
│   └── instances_val.json (optional)
```

23.11 MTWI

- Step1: Download `mtwi_2018_train.zip` from homepage.

```
mkdir mtwi && cd mtwi

unzip -q mtwi_2018_train.zip
mv image_train imgs && mv txt_train annotations

rm mtwi_2018_train.zip
```

- Step2: Generate `instances_training.json` and `instance_val.json` (optional) with the following command:

```
# Annotations of MTWI test split is not publicly available, split a validation
# set by adding --val-ratio 0.2
python tools/dataset_converters/textdet/mtwi_converter.py PATH/TO/mtwi --nproc 4
```

- After running the above codes, the directory structure should be as follows:

```
├── mtwi
│   ├── annotations
│   ├── imgs
│   ├── instances_training.json
│   └── instances_val.json (optional)
```

23.12 ReCTS

- Step1: Download ReCTS.zip to `rects/` from the homepage.

```
mkdir rects && cd rects

# Download ReCTS dataset
# You can also find Google Drive link on the dataset homepage
wget https://datasets.cvc.uab.es/rrc/ReCTS.zip --no-check-certificate
unzip -q ReCTS.zip
```

(continues on next page)

(continued from previous page)

```
mv img imgs && mv gt_unicode annotations

rm ReCTS.zip && rm -rf gt
```

- Step2: Generate `instances_training.json` and `instances_val.json` (optional) with following command:

```
# Annotations of ReCTS test split is not publicly available, split a validation
# set by adding --val-ratio 0.2
python tools/dataset_converters/textdet/rects_converter.py PATH/TO/rects --nproc 4 -
↪-val-ratio 0.2
```

- After running the above codes, the directory structure should be as follows:

```
├── rects
│   ├── annotations
│   ├── imgs
│   ├── instances_val.json (optional)
│   └── instances_training.json
```

23.13 ILST

- Step1: Download `IIIT-ILST` from onedrive
- Step2: Run the following commands

```
unzip -q IIIT-ILST.zip && rm IIIT-ILST.zip
cd IIIT-ILST

# rename files
cd Devanagari && for i in `ls`; do mv -f $i `echo "devanagari_"$i`; done && cd ..
cd Malayalam && for i in `ls`; do mv -f $i `echo "malayalam_"$i`; done && cd ..
cd Telugu && for i in `ls`; do mv -f $i `echo "telugu_"$i`; done && cd ..

# transfer image path
mkdir imgs && mkdir annotations
mv Malayalam/{*jpg,*jpeg} imgs/ && mv Malayalam/*xml annotations/
mv Devanagari/*jpg imgs/ && mv Devanagari/*xml annotations/
mv Telugu/*jpeg imgs/ && mv Telugu/*xml annotations/

# remove unnecessary files
rm -rf Devanagari && rm -rf Malayalam && rm -rf Telugu && rm -rf README.txt
```

- Step3: Generate `instances_training.json` and `instances_val.json` (optional). Since the original dataset doesn't have a validation set, you may specify `--val-ratio` to split the dataset. E.g., if val-ratio is 0.2, then 20% of the data are left out as the validation set in this example.

```
python tools/dataset_converters/textdet/ilst_converter.py    PATH/TO/IIIT-ILST --
↪nproc 4
```

- After running the above codes, the directory structure should be as follows:

```
├── IIIT-ILST
│   ├── annotations
│   ├── imgs
│   ├── instances_val.json (optional)
│   └── instances_training.json
```

23.14 VinText

- Step1: Download vintext.zip to `vintext`

```
mkdir vintext && cd vintext

# Download dataset from google drive
wget --load-cookies /tmp/cookies.txt "https://docs.google.com/uc?export=download&
↪confirm=$(wget --quiet --save-cookies /tmp/cookies.txt --keep-session-cookies --
↪no-check-certificate 'https://docs.google.com/uc?export=download&
↪id=1UUQhNvzgpZy7zXBFQp0Qox-BBjunZ0ml' -O- | sed -rn 's/.*confirm=([0-9A-Za-z_]+).
↪*/\1\n/p')&id=1UUQhNvzgpZy7zXBFQp0Qox-BBjunZ0ml" -O vintext.zip && rm -rf /tmp/
↪cookies.txt

# Extract images and annotations
unzip -q vintext.zip && rm vintext.zip
mv vietnamese/labels ./ && mv vietnamese/test_image ./ && mv vietnamese/train_
↪images ./ && mv vietnamese/unseen_test_images ./
rm -rf vietnamese

# Rename files
mv labels annotations && mv test_image test && mv train_images  training && mv␣
↪unseen_test_images  unseen_test
mkdir imgs
mv training imgs/ && mv test imgs/ && mv unseen_test imgs/
```

- Step2: Generate `instances_training.json`, `instances_test.json` and `instances_unseen_test.json`

```
python tools/dataset_converters/textdet/vintext_converter.py PATH/TO/vintext --
↪nproc 4
```

- After running the above codes, the directory structure should be as follows:

```
├── vintext
│   ├── annotations
│   ├── imgs
│   ├── instances_test.json
│   ├── instances_unseen_test.json
│   └── instances_training.json
```

23.15 BID

- Step1: Download BID Dataset.zip
- Step2: Run the following commands to preprocess the dataset

```
# Rename
mv BID\ Dataset.zip BID_Dataset.zip

# Unzip and Rename
unzip -q BID_Dataset.zip && rm BID_Dataset.zip
mv BID\ Dataset BID

# The BID dataset has a problem of permission, and you may
# add permission for this file
chmod -R 777 BID
cd BID
mkdir imgs && mkdir annotations

# For images and annotations
mv CNH_Aberta/*in.jpg imgs && mv CNH_Aberta/*txt annotations && rm -rf CNH_Aberta
mv CNH_Frente/*in.jpg imgs && mv CNH_Frente/*txt annotations && rm -rf CNH_Frente
mv CNH_Verso/*in.jpg imgs && mv CNH_Verso/*txt annotations && rm -rf CNH_Verso
mv CPF_Frente/*in.jpg imgs && mv CPF_Frente/*txt annotations && rm -rf CPF_Frente
mv CPF_Verso/*in.jpg imgs && mv CPF_Verso/*txt annotations && rm -rf CPF_Verso
mv RG_Aberto/*in.jpg imgs && mv RG_Aberto/*txt annotations && rm -rf RG_Aberto
mv RG_Frente/*in.jpg imgs && mv RG_Frente/*txt annotations && rm -rf RG_Frente
mv RG_Verso/*in.jpg imgs && mv RG_Verso/*txt annotations && rm -rf RG_Verso

# Remove unnecessary files
rm -rf desktop.ini
```

- Step3: - Step3: Generate `instances_training.json` and `instances_val.json` (optional). Since the original dataset doesn't have a validation set, you may specify `--val-ratio` to split the dataset. E.g., if val-ratio is 0.2, then 20% of the data are left out as the validation set in this example.

```
python tools/dataset_converters/textdet/bid_converter.py PATH/TO/BID --nproc 4
```

- After running the above codes, the directory structure should be as follows:

```
├── BID
│   ├── annotations
│   ├── imgs
│   ├── instances_training.json
│   └── instances_val.json (optional)
```

23.16 RCTW

- Step1: Download `train_images.zip.001`, `train_images.zip.002`, and `train_gts.zip` from the homepage, extract the zips to `rctw/imgs` and `rctw/annotations`, respectively.
- Step2: Generate `instances_training.json` and `instances_val.json` (optional). Since the test annotations are not publicly available, you may specify `--val-ratio` to split the dataset. E.g., if val-ratio is 0.2, then 20% of the data are left out as the validation set in this example.

```
# Annotations of RCTW test split is not publicly available, split a validation set␣
↪by adding --val-ratio 0.2
python tools/dataset_converters/textdet/rctw_converter.py PATH/TO/rctw --nproc 4
```

- After running the above codes, the directory structure should be as follows:

```
├── rctw
│   ├── annotations
│   ├── imgs
│   ├── instances_training.json
│   └── instances_val.json (optional)
```

23.17 HierText

- Step1 (optional): Install AWS CLI.
- Step2: Clone HierText repo to get annotations

```
mkdir HierText
git clone https://github.com/google-research-datasets/hiertext.git
```

- Step3: Download `train.tgz`, `validation.tgz` from aws

```
aws s3 --no-sign-request cp s3://open-images-dataset/ocr/train.tgz .
aws s3 --no-sign-request cp s3://open-images-dataset/ocr/validation.tgz .
```

- Step4: Process raw data

```
# process annotations
mv hiertext/gt ./
rm -rf hiertext
mv gt annotations
gzip -d annotations/train.jsonl.gz
gzip -d annotations/validation.jsonl.gz
# process images
mkdir imgs
mv train.tgz imgs/
mv validation.tgz imgs/
tar -xzvf imgs/train.tgz
tar -xzvf imgs/validation.tgz
```

- Step5: Generate `instances_training.json` and `instance_val.json`. HierText includes different levels of annotation, from paragraph, line, to word. Check the original paper for details. E.g. set `--level paragraph`

to get paragraph-level annotation. Set `--level line` to get line-level annotation. set `--level word` to get word-level annotation.

```
# Collect word annotation from HierText  --level word
python tools/dataset_converters/textdet/hiertext_converter.py PATH/TO/HierText --
↪level word --nproc 4
```

- After running the above codes, the directory structure should be as follows:

```
├── HierText
│   ├── annotations
│   ├── imgs
│   ├── instances_training.json
│   └── instances_val.json
```

23.18 ArT

- Step1: Download `train_images.tar.gz`, and `train_labels.json` from the homepage to `art/`

```
mkdir art && cd art
mkdir annotations

# Download ArT dataset
wget https://dataset-bj.cdn.bcebos.com/art/train_images.tar.gz --no-check-
↪certificate
wget https://dataset-bj.cdn.bcebos.com/art/train_labels.json --no-check-certificate

# Extract
tar -xf train_images.tar.gz
mv train_images imgs
mv train_labels.json annotations/

# Remove unnecessary files
rm train_images.tar.gz
```

- Step2: Generate `instances_training.json` and `instances_val.json` (optional). Since the test annotations are not publicly available, you may specify `--val-ratio` to split the dataset. E.g., if val-ratio is 0.2, then 20% of the data are left out as the validation set in this example.

```
# Annotations of ArT test split is not publicly available, split a validation set␣
↪by adding --val-ratio 0.2
python tools/data/textdet/art_converter.py PATH/TO/art --nproc 4
```

- After running the above codes, the directory structure should be as follows:

```
├── art
│   ├── annotations
│   ├── imgs
│   ├── instances_training.json
│   └── instances_val.json (optional)
```

CHAPTER

TWENTYFOUR

TEXT RECOGNITION

Note: This page is a manual preparation guide for datasets not yet supported by *Dataset Preparer*, which all these scripts will be eventually migrated into.

24.1 Overview

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

24.1.1 Install AWS CLI (optional)

- Since there are some datasets that require the AWS CLI to be installed in advance, we provide a quick installation guide here:

```
curl "https://awscli.amazonaws.com/awscli-exe-linux-x86_64.zip" -o "awscliv2.zip"
unzip awscliv2.zip
sudo ./aws/install
./aws/install -i /usr/local/aws-cli -b /usr/local/bin
!aws configure
# this command will require you to input keys, you can skip them except
# for the Default region name
# AWS Access Key ID [None]:
# AWS Secret Access Key [None]:
# Default region name [None]: us-east-1
# Default output format [None]
```

For users in China, these datasets can also be downloaded from OpenDataLab with high speed:

- icdar_2013
- icdar_2015
- IIIT5K
- ct80
- svt
- Totaltext
- IAM

24.2 ICDAR 2011 (Born-Digital Images)

- Step1: Download `Challenge1_Training_Task3_Images_GT.zip`, `Challenge1_Test_Task3_Images.zip`, and `Challenge1_Test_Task3_GT.txt` from homepage `Task 1.3: Word Recognition (2013 edition)`.

```
mkdir icdar2011 && cd icdar2011
mkdir annotations

# Download ICDAR 2011
wget https://rrc.cvc.uab.es/downloads/Challenge1_Training_Task3_Images_GT.zip --no-
↪check-certificate
wget https://rrc.cvc.uab.es/downloads/Challenge1_Test_Task3_Images.zip --no-check-
↪certificate
wget https://rrc.cvc.uab.es/downloads/Challenge1_Test_Task3_GT.txt --no-check-
↪certificate

# For images
mkdir crops
unzip -q Challenge1_Training_Task3_Images_GT.zip -d crops/train
unzip -q Challenge1_Test_Task3_Images.zip -d crops/test

# For annotations
mv Challenge1_Test_Task3_GT.txt annotations && mv crops/train/gt.txt annotations/
↪Challenge1_Train_Task3_GT.txt
```

- Step2: Convert original annotations to `train_labels.json` and `test_labels.json` with the following command:

```
python tools/dataset_converters/textrecog/ic11_converter.py PATH/TO/icdar2011
```

- After running the above codes, the directory structure should be as follows:

```
├── icdar2011
│   ├── crops
│   ├── train_labels.json
│   └── test_labels.json
```

24.3 coco_text

- Step1: Download from homepage
- Step2: Download train_labels.json
- After running the above codes, the directory structure should be as follows:

```
├── coco_text
│   ├── train_labels.json
│   └── train_words
```

24.4 SynthAdd

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

```
mkdir SynthAdd && cd SynthAdd

mv /path/to/SynthText_Add.zip .

unzip SynthText_Add.zip

mv /path/to/train_labels.json .

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

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

- After running the above codes, the directory structure should be as follows:

```
├── SynthAdd
│   ├── train_labels.json
│   └── SynthText_Add
```

24.5 OpenVINO

- Step1 (optional): Install AWS CLI.
- Step2: Download Open Images subsets `train_1`, `train_2`, `train_5`, `train_f`, and `validation` to `openvino/`.

```
mkdir openvino && cd openvino

# Download Open Images subsets
for s in 1 2 5 f; do
  aws s3 --no-sign-request cp s3://open-images-dataset/tar/train_${s}.tar.gz .
done
aws s3 --no-sign-request cp s3://open-images-dataset/tar/validation.tar.gz .

# Download annotations
for s in 1 2 5 f; do
  wget https://storage.openvinotoolkit.org/repositories/openvino_training_
↪extensions/datasets/open_images_v5_text/text_spotting_openimages_v5_train_${s}.
↪json
done
wget https://storage.openvinotoolkit.org/repositories/openvino_training_extensions/
↪datasets/open_images_v5_text/text_spotting_openimages_v5_validation.json

# Extract images
```

(continues on next page)

(continued from previous page)

```
mkdir -p openimages_v5/val
for s in 1 2 5 f; do
  tar zxf train_${s}.tar.gz -C openimages_v5
done
tar zxf validation.tar.gz -C openimages_v5/val
```

- Step3: Generate `train_{1,2,5,f}_labels.json`, `val_labels.json` and crop images using 4 processes with the following command:

```
python tools/dataset_converters/textrecog/openvino_converter.py /path/to/openvino 4
```

- After running the above codes, the directory structure should be as follows:

```
├── OpenVINO
│   ├── image_1
│   ├── image_2
│   ├── image_5
│   ├── image_f
│   ├── image_val
│   ├── train_1_labels.json
│   ├── train_2_labels.json
│   ├── train_5_labels.json
│   ├── train_f_labels.json
│   └── val_labels.json
```

24.6 DeText

- Step1: Download `ch9_training_images.zip`, `ch9_training_localization_transcription_gt.zip`, `ch9_validation_images.zip`, and `ch9_validation_localization_transcription_gt.zip` from **Task 3: End to End** on the homepage.

```
mkdir detext && cd detext
mkdir imgs && mkdir annotations && mkdir imgs/training && mkdir imgs/val && mkdir␣
↪annotations/training && mkdir annotations/val

# Download DeText
wget https://rrc.cvc.uab.es/downloads/ch9_training_images.zip --no-check-certificate
wget https://rrc.cvc.uab.es/downloads/ch9_training_localization_transcription_gt.
↪zip --no-check-certificate
wget https://rrc.cvc.uab.es/downloads/ch9_validation_images.zip --no-check-
↪certificate
wget https://rrc.cvc.uab.es/downloads/ch9_validation_localization_transcription_gt.
↪zip --no-check-certificate

# Extract images and annotations
unzip -q ch9_training_images.zip -d imgs/training && unzip -q ch9_training_
↪localization_transcription_gt.zip -d annotations/training && unzip -q ch9_
↪validation_images.zip -d imgs/val && unzip -q ch9_validation_localization_
↪transcription_gt.zip -d annotations/val
```

(continues on next page)

(continued from previous page)

```
# Remove zips
rm ch9_training_images.zip && rm ch9_training_localization_transcription_gt.zip &&␣
↪rm ch9_validation_images.zip && rm ch9_validation_localization_transcription_gt.
↪zip
```

- Step2: Generate `train_labels.json` and `test_labels.json` with following command:

```
# Add --preserve-vertical to preserve vertical texts for training, otherwise
# vertical images will be filtered and stored in PATH/TO/detext/ignores
python tools/dataset_converters/textrecog/detext_converter.py PATH/TO/detext --
↪nproc 4
```

- After running the above codes, the directory structure should be as follows:

```
├── detext
│   ├── crops
│   ├── ignores
│   ├── train_labels.json
│   └── test_labels.json
```

24.7 NAF

- Step1: Download labeled_images.tar.gz to `naf/`.

```
mkdir naf && cd naf

# Download NAF dataset
wget https://github.com/herobd/NAF_dataset/releases/download/v1.0/labeled_images.
↪tar.gz
tar -zxf labeled_images.tar.gz

# For images
mkdir annotations && mv labeled_images imgs

# For annotations
git clone https://github.com/herobd/NAF_dataset.git
mv NAF_dataset/train_valid_test_split.json annotations/ && mv NAF_dataset/groups␣
↪annotations/

rm -rf NAF_dataset && rm labeled_images.tar.gz
```

- Step2: Generate `train_labels.json`, `val_labels.json`, and `test_labels.json` with following command:

```
# Add --preserve-vertical to preserve vertical texts for training, otherwise
# vertical images will be filtered and stored in PATH/TO/naf/ignores
python tools/dataset_converters/textrecog/naf_converter.py PATH/TO/naf --nproc 4
```

- After running the above codes, the directory structure should be as follows:

```
├── naf
│   ├── crops
│   ├── train_labels.json
│   ├── val_labels.json
│   └── test_labels.json
```

24.8 Lecture Video DB

Warning: This section is not fully tested yet.

Note: The LV dataset has already provided cropped images and the corresponding annotations

- Step1: Download IIIT-CVid.zip to `lv/`.

```
mkdir lv && cd lv

# Download LV dataset
wget http://cdn.iiit.ac.in/cdn/preon.iiit.ac.in/~kartik/IIIT-CVid.zip
unzip -q IIIT-CVid.zip

# For image
mv IIIT-CVid/Crops ./

# For annotation
mv IIIT-CVid/train.txt train_labels.json && mv IIIT-CVid/val.txt val_label.txt &&␣
↪mv IIIT-CVid/test.txt test_labels.json

rm IIIT-CVid.zip
```

- Step2: Generate `train_labels.json`, `val.json`, and `test.json` with following command:

```
python tools/dataset_converters/textdreog/lv_converter.py PATH/TO/lv
```

- After running the above codes, the directory structure should be as follows:

```
├── lv
│   ├── Crops
│   ├── train_labels.json
│   └── test_labels.json
```

24.9 LSVT

Warning: This section is not fully tested yet.

- Step1: Download train_full_images_0.tar.gz, train_full_images_1.tar.gz, and train_full_labels.json to `lsvt/`.

```
mkdir lsvt && cd lsvt

# Download LSVT dataset
wget https://dataset-bj.cdn.bcebos.com/lsvt/train_full_images_0.tar.gz
wget https://dataset-bj.cdn.bcebos.com/lsvt/train_full_images_1.tar.gz
wget https://dataset-bj.cdn.bcebos.com/lsvt/train_full_labels.json

mkdir annotations
tar -xf train_full_images_0.tar.gz && tar -xf train_full_images_1.tar.gz
mv train_full_labels.json annotations/ && mv train_full_images_1/*.jpg train_full_
↪images_0/
mv train_full_images_0 imgs

rm train_full_images_0.tar.gz && rm train_full_images_1.tar.gz && rm -rf train_full_
↪images_1
```

- Step2: Generate `train_labels.json` and `val_label.json` (optional) with the following command:

```
# Annotations of LSVT test split is not publicly available, split a validation
# set by adding --val-ratio 0.2
# Add --preserve-vertical to preserve vertical texts for training, otherwise
# vertical images will be filtered and stored in PATH/TO/lsvt/ignores
python tools/dataset_converters/textdrecog/lsvt_converter.py PATH/TO/lsvt --nproc 4
```

- After running the above codes, the directory structure should be as follows:

```
├── lsvt
│   ├── crops
│   ├── ignores
│   ├── train_labels.json
│   └── val_label.json (optional)
```

24.10 IMGUR

Warning: This section is not fully tested yet.

- Step1: Run `download_imgur5k.py` to download images. You can merge PR#5 in your local repository to enable a **much faster** parallel execution of image download.

```
mkdir imgur && cd imgur

git clone https://github.com/facebookresearch/IMGUR5K-Handwriting-Dataset.git
```

(continues on next page)

(continued from previous page)

```
# Download images from imgur.com. This may take SEVERAL HOURS!
python ./IMGUR5K-Handwriting-Dataset/download_imgur5k.py --dataset_info_dir ./
↪IMGUR5K-Handwriting-Dataset/dataset_info/ --output_dir ./imgs

# For annotations
mkdir annotations
mv ./IMGUR5K-Handwriting-Dataset/dataset_info/*.json annotations

rm -rf IMGUR5K-Handwriting-Dataset
```

- Step2: Generate `train_labels.json`, `val_label.txt` and `test_labels.json` and crop images with the following command:

```
python tools/dataset_converters/textrecog/imgur_converter.py PATH/TO/imgur
```

- After running the above codes, the directory structure should be as follows:

```
├── imgur
│   ├── crops
│   ├── train_labels.json
│   ├── test_labels.json
│   └── val_label.json
```

24.11 KAIST

Warning: This section is not fully tested yet.

- Step1: Download KAIST_all.zip to `kaist/`.

```
mkdir kaist && cd kaist
mkdir imgs && mkdir annotations

# Download KAIST dataset
wget http://www.iapr-tc11.org/dataset/KAIST_SceneText/KAIST_all.zip
unzip -q KAIST_all.zip && rm KAIST_all.zip
```

- Step2: Extract zips:

```
python tools/dataset_converters/common/extract_kaist.py PATH/TO/kaist
```

- Step3: Generate `train_labels.json` and `val_label.json` (optional) with following command:

```
# Since KAIST does not provide an official split, you can split the dataset by␣
↪adding --val-ratio 0.2
# Add --preserve-vertical to preserve vertical texts for training, otherwise
# vertical images will be filtered and stored in PATH/TO/kaist/ignores
python tools/dataset_converters/textrecog/kaist_converter.py PATH/TO/kaist --nproc 4
```

- After running the above codes, the directory structure should be as follows:

```
├── kaist
│   ├── crops
│   ├── ignores
│   ├── train_labels.json
│   └── val_label.json (optional)
```

24.12 MTWI

Warning: This section is not fully tested yet.

- Step1: Download `mtwi_2018_train.zip` from homepage.

```
mkdir mtwi && cd mtwi

unzip -q mtwi_2018_train.zip
mv image_train imgs && mv txt_train annotations

rm mtwi_2018_train.zip
```

- Step2: Generate `train_labels.json` and `val_label.json` (optional) with the following command:

```
# Annotations of MTWI test split is not publicly available, split a validation
# set by adding --val-ratio 0.2
# Add --preserve-vertical to preserve vertical texts for training, otherwise
# vertical images will be filtered and stored in PATH/TO/mtwi/ignores
python tools/dataset_converters/textrecog/mtwi_converter.py PATH/TO/mtwi --nproc 4
```

- After running the above codes, the directory structure should be as follows:

```
├── mtwi
│   ├── crops
│   ├── train_labels.json
│   └── val_label.json (optional)
```

24.13 ReCTS

Warning: This section is not fully tested yet.

- Step1: Download ReCTS.zip to `rects/` from the homepage.

```
mkdir rects && cd rects

# Download ReCTS dataset
# You can also find Google Drive link on the dataset homepage
wget https://datasets.cvc.uab.es/rrc/ReCTS.zip --no-check-certificate
unzip -q ReCTS.zip
```

(continues on next page)

(continued from previous page)

```
mv img imgs && mv gt_unicode annotations

rm ReCTS.zip -f && rm -rf gt
```

- Step2: Generate `train_labels.json` and `val_label.json` (optional) with the following command:

```
# Annotations of ReCTS test split is not publicly available, split a validation
# set by adding --val-ratio 0.2
# Add --preserve-vertical to preserve vertical texts for training, otherwise
# vertical images will be filtered and stored in PATH/TO/rects/ignores
python tools/dataset_converters/textrecog/rects_converter.py PATH/TO/rects --nproc 4
```

- After running the above codes, the directory structure should be as follows:

```
├── rects
│   ├── crops
│   ├── ignores
│   ├── train_labels.json
│   └── val_label.json (optional)
```

24.14 ILST

Warning: This section is not fully tested yet.

- Step1: Download `IIIT-ILST.zip` from onedrive link
- Step2: Run the following commands

```
unzip -q IIIT-ILST.zip && rm IIIT-ILST.zip
cd IIIT-ILST

# rename files
cd Devanagari && for i in `ls`; do mv -f $i `echo "devanagari_"$i`; done && cd ..
cd Malayalam && for i in `ls`; do mv -f $i `echo "malayalam_"$i`; done && cd ..
cd Telugu && for i in `ls`; do mv -f $i `echo "telugu_"$i`; done && cd ..

# transfer image path
mkdir imgs && mkdir annotations
mv Malayalam/{*jpg,*jpeg} imgs/ && mv Malayalam/*xml annotations/
mv Devanagari/*jpg imgs/ && mv Devanagari/*xml annotations/
mv Telugu/*jpeg imgs/ && mv Telugu/*xml annotations/

# remove unnecessary files
rm -rf Devanagari && rm -rf Malayalam && rm -rf Telugu && rm -rf README.txt
```

- Step3: Generate `train_labels.json` and `val_label.json` (optional) and crop images using 4 processes with the following command (add `--preserve-vertical` if you wish to preserve the images containing vertical texts). Since the original dataset doesn't have a validation set, you may specify `--val-ratio` to split the dataset. E.g., if val-ratio is 0.2, then 20% of the data are left out as the validation set in this example.

```
python tools/dataset_converters/textrecog/ilst_converter.py PATH/TO/IIIT-ILST --
↪nproc 4
```

- After running the above codes, the directory structure should be as follows:

```
├── IIIT-ILST
│   ├── crops
│   ├── ignores
│   ├── train_labels.json
│   └── val_label.json (optional)
```

24.15 VinText

Warning: This section is not fully tested yet.

- Step1: Download vintext.zip to `vintext`

```
mkdir vintext && cd vintext

# Download dataset from google drive
wget --load-cookies /tmp/cookies.txt "https://docs.google.com/uc?export=download&
↪confirm=$(wget --quiet --save-cookies /tmp/cookies.txt --keep-session-cookies --
↪no-check-certificate 'https://docs.google.com/uc?export=download&
↪id=1UUQhNvzgpZy7zXBFQp0Qox-BBjunZ0ml' -O- | sed -rn 's/.*confirm=([0-9A-Za-z_]+).
↪*/\1\n/p')&id=1UUQhNvzgpZy7zXBFQp0Qox-BBjunZ0ml" -O vintext.zip && rm -rf /tmp/
↪cookies.txt

# Extract images and annotations
unzip -q vintext.zip && rm vintext.zip
mv vietnamese/labels ./ && mv vietnamese/test_image ./ && mv vietnamese/train_
↪images ./ && mv vietnamese/unseen_test_images ./
rm -rf vietnamese

# Rename files
mv labels annotations && mv test_image test && mv train_images  training && mv␣
↪unseen_test_images  unseen_test
mkdir imgs
mv training imgs/ && mv test imgs/ && mv unseen_test imgs/
```

- Step2: Generate `train_labels.json`, `test_labels.json`, `unseen_test_labels.json`, and crop images using 4 processes with the following command (add `--preserve-vertical` if you wish to preserve the images containing vertical texts).

```
python tools/dataset_converters/textrecog/vintext_converter.py PATH/TO/vietnamese --
↪nproc 4
```

- After running the above codes, the directory structure should be as follows:

```
├── vintext
│   ├── crops
```

(continues on next page)

(continued from previous page)

```
|   ├── ignores
|   ├── train_labels.json
|   ├── test_labels.json
|   └── unseen_test_labels.json
```

24.16 BID

Warning: This section is not fully tested yet.

- Step1: Download BID Dataset.zip
- Step2: Run the following commands to preprocess the dataset

```
# Rename
mv BID\ Dataset.zip BID_Dataset.zip

# Unzip and Rename
unzip -q BID_Dataset.zip && rm BID_Dataset.zip
mv BID\ Dataset BID

# The BID dataset has a problem of permission, and you may
# add permission for this file
chmod -R 777 BID
cd BID
mkdir imgs && mkdir annotations

# For images and annotations
mv CNH_Aberta/*in.jpg imgs && mv CNH_Aberta/*txt annotations && rm -rf CNH_Aberta
mv CNH_Frente/*in.jpg imgs && mv CNH_Frente/*txt annotations && rm -rf CNH_Frente
mv CNH_Verso/*in.jpg imgs && mv CNH_Verso/*txt annotations && rm -rf CNH_Verso
mv CPF_Frente/*in.jpg imgs && mv CPF_Frente/*txt annotations && rm -rf CPF_Frente
mv CPF_Verso/*in.jpg imgs && mv CPF_Verso/*txt annotations && rm -rf CPF_Verso
mv RG_Aberto/*in.jpg imgs && mv RG_Aberto/*txt annotations && rm -rf RG_Aberto
mv RG_Frente/*in.jpg imgs && mv RG_Frente/*txt annotations && rm -rf RG_Frente
mv RG_Verso/*in.jpg imgs && mv RG_Verso/*txt annotations && rm -rf RG_Verso

# Remove unnecessary files
rm -rf desktop.ini
```

- Step3: Generate `train_labels.json` and `val_label.json` (optional) and crop images using 4 processes with the following command (add `--preserve-vertical` if you wish to preserve the images containing vertical texts). Since the original dataset doesn't have a validation set, you may specify `--val-ratio` to split the dataset. E.g., if test-ratio is 0.2, then 20% of the data are left out as the validation set in this example.

```
python tools/dataset_converters/textrecog/bid_converter.py PATH/TO/BID --nproc 4
```

- After running the above codes, the directory structure should be as follows:

```
├── BID
│   ├── crops
│   ├── ignores
│   ├── train_labels.json
│   └── val_label.json (optional)
```

24.17 RCTW

Warning: This section is not fully tested yet.

- Step1: Download `train_images.zip.001`, `train_images.zip.002`, and `train_gts.zip` from the homepage, extract the zips to `rctw/imgs` and `rctw/annotations`, respectively.
- Step2: Generate `train_labels.json` and `val_label.json` (optional). Since the original dataset doesn't have a validation set, you may specify `--val-ratio` to split the dataset. E.g., if val-ratio is 0.2, then 20% of the data are left out as the validation set in this example.

```
# Annotations of RCTW test split is not publicly available, split a validation set␣
↪by adding --val-ratio 0.2
# Add --preserve-vertical to preserve vertical texts for training, otherwise␣
↪vertical images will be filtered and stored in PATH/TO/rctw/ignores
python tools/dataset_converters/textrecog/rctw_converter.py PATH/TO/rctw --nproc 4
```

- After running the above codes, the directory structure should be as follows:

```
├── rctw
│   ├── crops
│   ├── ignores
│   ├── train_labels.json
│   └── val_label.json (optional)
```

24.18 HierText

Warning: This section is not fully tested yet.

- Step1 (optional): Install AWS CLI.
- Step2: Clone HierText repo to get annotations

```
mkdir HierText
git clone https://github.com/google-research-datasets/hiertext.git
```

- Step3: Download `train.tgz`, `validation.tgz` from aws

```
aws s3 --no-sign-request cp s3://open-images-dataset/ocr/train.tgz .
aws s3 --no-sign-request cp s3://open-images-dataset/ocr/validation.tgz .
```

- Step4: Process raw data

```
# process annotations
mv hiertext/gt ./
rm -rf hiertext
mv gt annotations
gzip -d annotations/train.json.gz
gzip -d annotations/validation.json.gz
# process images
mkdir imgs
mv train.tgz imgs/
mv validation.tgz imgs/
tar -xzvf imgs/train.tgz
tar -xzvf imgs/validation.tgz
```

- Step5: Generate `train_labels.json` and `val_label.json`. HierText includes different levels of annotation, including `paragraph`, `line`, and `word`. Check the original paper for details. E.g. set `--level paragraph` to get paragraph-level annotation. Set `--level line` to get line-level annotation. set `--level word` to get word-level annotation.

```
# Collect word annotation from HierText  --level word
# Add --preserve-vertical to preserve vertical texts for training, otherwise␣
↪vertical images will be filtered and stored in PATH/TO/HierText/ignores
python tools/dataset_converters/textrecog/hiertext_converter.py PATH/TO/HierText --
↪level word --nproc 4
```

- After running the above codes, the directory structure should be as follows:

```
├── HierText
│   ├── crops
│   ├── ignores
│   ├── train_labels.json
│   └── val_label.json
```

24.19 ArT

Warning: This section is not fully tested yet.

- Step1: Download `train_images.tar.gz`, and `train_labels.json` from the homepage to `art/`

```
mkdir art && cd art
mkdir annotations

# Download ArT dataset
wget https://dataset-bj.cdn.bcebos.com/art/train_task2_images.tar.gz
wget https://dataset-bj.cdn.bcebos.com/art/train_task2_labels.json

# Extract
tar -xf train_task2_images.tar.gz
mv train_task2_images crops
```

(continues on next page)

(continued from previous page)

```
mv train_task2_labels.json annotations/

# Remove unnecessary files
rm train_images.tar.gz
```

- Step2: Generate `train_labels.json` and `val_label.json` (optional). Since the test annotations are not publicly available, you may specify `--val-ratio` to split the dataset. E.g., if val-ratio is 0.2, then 20% of the data are left out as the validation set in this example.

```
# Annotations of ArT test split is not publicly available, split a validation set␣
↪by adding --val-ratio 0.2
python tools/dataset_converters/textrecog/art_converter.py PATH/TO/art
```

- After running the above codes, the directory structure should be as follows:

```
├── art
│   ├── crops
│   ├── train_labels.json
│   └── val_label.json (optional)
```

CHAPTER

TWENTYFIVE

KEY INFORMATION EXTRACTION

Note: This page is a manual preparation guide for datasets not yet supported by *Dataset Preparer*, which all these scripts will be eventually migrated into.

25.1 Overview

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

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

25.2 Preparation Steps

25.2.1 WildReceipt

- Just download and extract wildreceipt.tar.

25.2.2 WildReceiptOpenset

- Step0: have WildReceipt prepared.
- Step1: Convert annotation files to OpenSet format:

```
# You may find more available arguments by running
# python tools/data/kie/closeset_to_openset.py -h
python tools/data/kie/closeset_to_openset.py data/wildreceipt/train.txt data/wildreceipt/
↪openset_train.txt
python tools/data/kie/closeset_to_openset.py data/wildreceipt/test.txt data/wildreceipt/
↪openset_test.txt
```

Note: You can learn more about the key differences between CloseSet and OpenSet annotations in our tutorial.

CHAPTER

TWENTYSIX

OVERVIEW

26.1 Weights

Here are the list of weights available for *Inference*.

For the ease of reference, some weights may have shorter aliases, which will be separated by / in the table. For example, "DB_r18 / dbnet_resnet18_fpnc_1200e_icdar2015" means that you can use either DB_r18 or dbnet_resnet18_fpnc_1200e_icdar2015 to initialize the Inferencer:

```
>>> from mmocr.apis import TextDetInferencer
>>> inferencer = TextDetInferencer(model='DB_r18')
>>> # equivalent to
>>> inferencer = TextDetInferencer(model='dbnet_resnet18_fpnc_1200e_icdar2015')
```

26.1.1 Text Detection

Model	README	ICDAR2015 (hmean-iou)	CTW1500 (hmean-iou)	Totaltext (hmean-iou)
DB_r18/dbnet_resnet18_fpnc_1200e_icdar2015	link	0.8169	-	-
dbnet_resnet50_fpnc_1200e_icdar2015	link	0.8504	-	-
dbnet_resnet50-dcnv2_fpnc_1200e_icdar2015	link	0.8543	-	-
DB_r50 / DBNet / dbnet_resnet50-oclip_fpnc_1200e_icdar2015	link	0.8644	-	-
dbnet_resnet18_fpnc_1200e_totaltext	link	-	-	0.8182
DBPP_r50/dbnetpp_resnet50_fpnc_1200e_icdar2015	link	0.8622	-	-
dbnetpp_resnet50-dcnv2_fpnc_1200e_icdar2015	link	0.8684	-	-
DBNetpp/dbnetpp_resnet50-oclip_fpnc_1200e_icdar2015	link	0.8837	-	-
MaskRCNN_CTW / mask-rcnn_resnet50_fpn_160e_ctw1500	link	-	0.7458	-
mask-rcnn_resnet50-oclip_fpn_160e_ctw1500	link	-	0.7562	-
MaskRCNN_IC15 / mask-rcnn_resnet50_fpn_160e_icdar2015	link	0.8182	-	-
MaskRCNN/mask-rcnn_resnet50-oclip_fpn_160e_icdar2015	link	0.8703	-	-
DRRG/drrg_resnet50_fpn-unet_1200e_ctw1500	link	-	0.8467	-
FCE_CTW_DCNv2 / fcenet_resnet50-dcnv2_fpn_1500e_ctw1500	link	-	0.8488	-
fcenet_resnet50-oclip_fpn_1500e_ctw1500	link	-	0.8192	-
FCE_IC15/fcenet_resnet50_fpn_1500e_icdar2015	link	0.8528	-	-
FCENet/fcenet_resnet50-oclip_fpn_1500e_icdar2015	link	0.8604	-	-
fcenet_resnet50_fpn_1500e_totaltext	link	-	-	0.8134
PANet_CTW/panet_resnet18_fpem-ffm_600e_ctw1500	link	-	0.777	-
PANet_IC15 / panet_resnet18_fpem-ffm_600e_icdar2015	link	0.7848	-	-
PS_CTW/psenet_resnet50_fpnf_600e_ctw1500	link	-	0.7793	-
psenet_resnet50-oclip_fpnf_600e_ctw1500	link	-	0.8037	-
PS_IC15/psenet_resnet50_fpnf_600e_icdar2015	link	0.7998	-	-
PSENet/psenet_resnet50-oclip_fpnf_600e_icdar2015	link	0.8478	-	-
textsnake_resnet50_fpn-unet_1200e_ctw1500	link	-	0.8286	-
TextSnake/textsnake_resnet50-oclip_fpn-unet_1200e_ctw1500	link	0.8529	-	-

26.1.2 Text Recognition

Note: Avg is the average on IIIT5K, SVT, ICDAR2013, ICDAR2015, SVTP, CT80.

Model	README	Avg (word_acc)	IIIT5K (word_acc)	SVT (word_acc)	IC-DAR2013 (word_acc)	IC-DAR2015 (word_acc)	SVTP (word_acc)	CT80 (word_acc)
ABINet_Vision / abinet-vision_20e_st-an_mj	link	0.88	0.95	0.91	0.94	0.79	0.84	0.84
ABINet / abinet_20e_st-an_mj	link	0.91	0.96	0.94	0.95	0.81	0.89	0.88
ASTER / aster_resnet45_6e_st_mj	link	0.86	0.94	0.89	0.93	0.77	0.81	0.85
CRNN / crnn_mini-vgg_5e_mj	link	0.70	0.81	0.81	0.87	0.56	0.61	0.57
MAERec / maerec_b_union14m	link	0.96	0.98	0.98	0.98	0.90	0.94	0.99
MASTER / master_resnet31_12e_st_mj_sa	link	0.88	0.95	0.90	0.95	0.76	0.85	0.89
nrtr_modality-transform_6e_st_mj	link	0.83	0.92	0.88	0.94	0.72	0.78	0.75
NRTR / NRTR_1/8-1/4 / nrtr_resnet31-1by8-1by4_6e_st_mj	link	0.87	0.95	0.88	0.95	0.76	0.80	0.89
NRTR_1/16-1/8 / nrtr_resnet31-1by16-1by8_6e_st_mj	link	0.87	0.95	0.90	0.94	0.74	0.80	0.89
svtr-small / svtr-small_20e_st_mj	link	0.86	0.86	0.90	0.94	0.75	0.85	0.89
svtr-base / svtr-base_20e_st_mj	link	0.87	0.86	0.92	0.94	0.74	0.84	0.90
RobustScanner / robustscanner_resnet31_5e_st-sub_mj-sub_sa_real	link	0.87	0.95	0.89	0.93	0.76	0.81	0.87
SAR / sar_resnet31_parallel-decoder_5e_st-sub_mj-sub_sa_real	link	0.88	0.95	0.88	0.94	0.76	0.83	0.90
sar_resnet31_sequential-decoder_5e_st-sub_mj-sub_sa_real	link	0.87	0.96	0.87	0.94	0.77	0.81	0.89
SATRN / satrn_shallow_5e_st_mj	link	0.90	0.96	0.92	0.96	0.80	0.88	0.90
SATRN_sm / satrn_shallow-small_5e_st_mj	link	0.88	0.94	0.90	0.96	0.79	0.86	0.85

26.1.3 Key Information Extraction

Model	README	wildreceipt (macro_f1)
SDMGR / sdmgr_unet16_60e_wildreceipt	link	0.89
sdmgr_novisual_60e_wildreceipt	link	0.87
sdmgr_novisual_60e_wildreceipt_openset	link	0.93

26.2 Statistics

- Number of checkpoints: 55
- Number of configs: 49
- Number of papers: 20
  - ALGORITHM: 20

26.2.1 BackBones

- Number of checkpoints: 1
- Number of configs: 0
- Number of papers: 1
  - [ALGORITHM] *Language Matters: A Weakly Supervised Vision-Language Pre-Training Approach for Scene Text Detection and Spotting*

26.2.2 Text Detection Models

- Number of checkpoints: 29
- Number of configs: 29
- Number of papers: 8
  - [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 and Adaptive Scale Fusion*
  - [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*

26.2.3 Text Recognition Models

- Number of checkpoints: 23
- Number of configs: 17
- Number of papers: 10
  - [ALGORITHM] *An End-to-End Trainable Neural Network for Image-Based Sequence Recognition and Its Application to Scene Text Recognition*
  - [ALGORITHM] *Aster: An Attentional Scene Text Recognizer With Flexible Rectification*
  - [ALGORITHM] *Master: Multi-Aspect Non-Local Network for 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] *Read Like Humans: Autonomous, Bidirectional and Iterative Language Modeling for Scene Text Recognition*
  - [ALGORITHM] *Revisiting Scene Text Recognition: A Data Perspective*
  - [ALGORITHM] *Robustscanner: Dynamically Enhancing Positional Clues for Robust Text Recognition*
  - [ALGORITHM] *Show, Attend and Read: A Simple and Strong Baseline for Irregular Text Recognition*
  - [ALGORITHM] *Svtr: Scene Text Recognition With a Single Visual Model*

26.2.4 Key Information Extraction Models

- Number of checkpoints: 3
- Number of configs: 3
- Number of papers: 1
  - [ALGORITHM] *Spatial Dual-Modality Graph Reasoning for Key Information Extraction*

CHAPTER

TWENTYSEVEN

SOTA MODELS

Here are some selected project implementations that are not yet included in MMOCR package, but are ready to use.

27.1 ABCNet: Real-time Scene Text Spotting with Adaptive Bezier-Curve Network

This is an implementation of ABCNet based on MMOCR, MMCV, and MMEngine.

ABCNet is a conceptually novel, efficient, and fully convolutional framework for text spotting, which address the problem by proposing the Adaptive Bezier-Curve Network (ABCNet). Our contributions are three-fold: 1) For the first time, we adaptively fit arbitrarily-shaped text by a parameterized Bezier curve. 2) We design a novel BezierAlign layer for extracting accurate convolution features of a text instance with arbitrary shapes, significantly improving the precision compared with previous methods. 3) Compared with standard bounding box detection, our Bezier curve detection introduces negligible computation overhead, resulting in superiority of our method in both efficiency and accuracy. Experiments on arbitrarily-shaped benchmark datasets, namely Total-Text and CTW1500, demonstrate that ABCNet achieves state-of-the-art accuracy, meanwhile significantly improving the speed. In particular, on Total-Text, our realtime version is over 10 times faster than recent state-of-the-art methods with a competitive recognition accuracy.

27.1.1 Status

Inference	Train	README
✓	✓	link

27.2 ABCNet v2: Adaptive Bezier-Curve Network for Real-time End-to-end Text Spotting

This is an implementation of ABCNetV2 based on MMOCR, MMCV, and MMEngine.

ABCNetV2 contributions are four-fold: 1) For the first time, we adaptively fit arbitrarily-shaped text by a parameterized Bezier curve, which, compared with segmentation-based methods, can not only provide structured output but also controllable representation. 2) We design a novel BezierAlign layer for extracting accurate convolution features of a text instance of arbitrary shapes, significantly improving the precision of recognition over previous methods. 3) Different from previous methods, which often suffer from complex post-processing and sensitive hyper-parameters, our ABCNet v2 maintains a simple pipeline with the only post-processing non-maximum suppression (NMS). 4) As the performance of text recognition closely depends on feature alignment, ABCNet v2 further adopts a simple yet effective coordinate convolution to encode the position of the convolutional filters, which leads to a considerable improvement

with negligible computation overhead. Comprehensive experiments conducted on various bilingual (English and Chinese) benchmark datasets demonstrate that ABCNet v2 can achieve state-of-the-art performance while maintaining very high efficiency.

27.2.1 Status

Inference	Train	README
✓		link

27.3 SPTS: Single-Point Text Spotting

This is an implementation of SPTS based on MMOCR, MMCV, and MMEngine.

Existing scene text spotting (i.e., end-to-end text detection and recognition) methods rely on costly bounding box annotations (e.g., text-line, word-level, or character-level bounding boxes). For the first time, we demonstrate that training scene text spotting models can be achieved with an extremely low-cost annotation of a single-point for each instance. We propose an end-to-end scene text spotting method that tackles scene text spotting as a sequence prediction task. Given an image as input, we formulate the desired detection and recognition results as a sequence of discrete tokens and use an auto-regressive Transformer to predict the sequence. The proposed method is simple yet effective, which can achieve state-of-the-art results on widely used benchmarks. Most significantly, we show that the performance is not very sensitive to the positions of the point annotation, meaning that it can be much easier to be annotated or even be automatically generated than the bounding box that requires precise positions. We believe that such a pioneer attempt indicates a significant opportunity for scene text spotting applications of a much larger scale than previously possible.

27.3.1 Status

Inference	Train	README
✓	✓	link

CHAPTER

TWENTYEIGHT

BACKBONES

28.1 oCLIP

Language Matters: A Weakly Supervised Vision-Language Pre-training Approach for Scene Text Detection and Spotting

28.1.1 Abstract

Recently, Vision-Language Pre-training (VLP) techniques have greatly benefited various vision-language tasks by jointly learning visual and textual representations, which intuitively helps in Optical Character Recognition (OCR) tasks due to the rich visual and textual information in scene text images. However, these methods cannot well cope with OCR tasks because of the difficulty in both instance-level text encoding and image-text pair acquisition (i.e. images and captured texts in them). This paper presents a weakly supervised pre-training method, oCLIP, which can acquire effective scene text representations by jointly learning and aligning visual and textual information. Our network consists of an image encoder and a character-aware text encoder that extract visual and textual features, respectively, as well as a visual-textual decoder that models the interaction among textual and visual features for learning effective scene text representations. With the learning of textual features, the pre-trained model can attend texts in images well with character awareness. Besides, these designs enable the learning from weakly annotated texts (i.e. partial texts in images without text bounding boxes) which mitigates the data annotation constraint greatly. Experiments over the weakly annotated images in ICDAR2019-LSVT show that our pre-trained model improves F-score by +2.5% and +4.8% while transferring its weights to other text detection and spotting networks, respectively. In addition, the proposed method outperforms existing pre-training techniques consistently across multiple public datasets (e.g., +3.2% and +1.3% for Total-Text and CTW1500).

28.1.2 Models

Note: The model is converted from the official oCLIP.

28.1.3 Supported Text Detection Models

28.1.4 Citation

```
@article{xue2022language,
  title={Language Matters: A Weakly Supervised Vision-Language Pre-training Approach for␣
↪Scene Text Detection and Spotting},
  author={Xue, Chuhui and Zhang, Wenqing and Hao, Yu and Lu, Shijian and Torr, Philip␣
↪and Bai, Song},
  journal={Proceedings of the European Conference on Computer Vision (ECCV)},
  year={2022}
}
```

CHAPTER

TWENTYNINE

TEXT DETECTION MODELS

29.1 DBNet

Real-time Scene Text Detection with Differentiable Binarization

29.1.1 Abstract

Recently, segmentation-based methods are quite popular in scene text detection, as the segmentation results can more accurately describe scene text of various shapes such as curve text. However, the post-processing of binarization is essential for segmentation-based detection, which converts probability maps produced by a segmentation method into bounding boxes/regions of text. In this paper, we propose a module named Differentiable Binarization (DB), which can perform the binarization process in a segmentation network. Optimized along with a DB module, a segmentation network can adaptively set the thresholds for binarization, which not only simplifies the post-processing but also enhances the performance of text detection. Based on a simple segmentation network, we validate the performance improvements of DB on five benchmark datasets, which consistently achieves state-of-the-art results, in terms of both detection accuracy and speed. In particular, with a light-weight backbone, the performance improvements by DB are significant so that we can look for an ideal tradeoff between detection accuracy and efficiency. Specifically, with a backbone of ResNet-18, our detector achieves an F-measure of 82.8, running at 62 FPS, on the MSRA-TD500 dataset.

29.1.2 Results and models

SynthText

ICDAR2015

Total Text

29.1.3 Citation

```
@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}}
```

29.2 DBNetpp

Real-Time Scene Text Detection with Differentiable Binarization and Adaptive Scale Fusion

29.2.1 Abstract

Recently, segmentation-based scene text detection methods have drawn extensive attention in the scene text detection field, because of their superiority in detecting the text instances of arbitrary shapes and extreme aspect ratios, profiting from the pixel-level descriptions. However, the vast majority of the existing segmentation-based approaches are limited to their complex post-processing algorithms and the scale robustness of their segmentation models, where the post-processing algorithms are not only isolated to the model optimization but also time-consuming and the scale robustness is usually strengthened by fusing multi-scale feature maps directly. In this paper, we propose a Differentiable Binarization (DB) module that integrates the binarization process, one of the most important steps in the post-processing procedure, into a segmentation network. Optimized along with the proposed DB module, the segmentation network can produce more accurate results, which enhances the accuracy of text detection with a simple pipeline. Furthermore, an efficient Adaptive Scale Fusion (ASF) module is proposed to improve the scale robustness by fusing features of different scales adaptively. By incorporating the proposed DB and ASF with the segmentation network, our proposed scene text detector consistently achieves state-of-the-art results, in terms of both detection accuracy and speed, on five standard benchmarks.

29.2.2 Results and models

SynthText

ICDAR2015

29.2.3 Citation

```
@article{liao2022real,
    title={Real-Time Scene Text Detection with Differentiable Binarization and Adaptive␣
↪Scale Fusion},
    author={Liao, Minghui and Zou, Zhisheng and Wan, Zhaoyi and Yao, Cong and Bai, Xiang}
↪,
    journal={IEEE Transactions on Pattern Analysis and Machine Intelligence},
    year={2022},
    publisher={IEEE}
}
```

29.3 DRRG

Deep relational reasoning graph network for arbitrary shape text detection

29.3.1 Abstract

Arbitrary shape text detection is a challenging task due to the high variety and complexity of scenes texts. In this paper, we propose a novel unified relational reasoning graph network for arbitrary shape text detection. In our method, an innovative local graph bridges a text proposal model via Convolutional Neural Network (CNN) and a deep relational reasoning network via Graph Convolutional Network (GCN), making our network end-to-end trainable. To be concrete, every text instance will be divided into a series of small rectangular components, and the geometry attributes (e.g., height, width, and orientation) of the small components will be estimated by our text proposal model. Given the geometry attributes, the local graph construction model can roughly establish linkages between different text components. For further reasoning and deducing the likelihood of linkages between the component and its neighbors, we adopt a graph-based network to perform deep relational reasoning on local graphs. Experiments on public available datasets demonstrate the state-of-the-art performance of our method.

29.3.2 Results and models

CTW1500

29.3.3 Citation

```
@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}
}
```

29.4 FCENet

Fourier Contour Embedding for Arbitrary-Shaped Text Detection

29.4.1 Abstract

One of the main challenges for arbitrary-shaped text detection is to design a good text instance representation that allows networks to learn diverse text geometry variances. Most of existing methods model text instances in image spatial domain via masks or contour point sequences in the Cartesian or the polar coordinate system. However, the mask representation might lead to expensive post-processing, while the point sequence one may have limited capability to model texts with highly-curved shapes. To tackle these problems, we model text instances in the Fourier domain and propose one novel Fourier Contour Embedding (FCE) method to represent arbitrary shaped text contours as compact signatures. We further construct FCENet with a backbone, feature pyramid networks (FPN) and a simple post-processing with the Inverse Fourier Transformation (IFT) and Non-Maximum Suppression (NMS). Different from previous methods, FCENet first predicts compact Fourier signatures of text instances, and then reconstructs text contours via IFT and NMS during test. Extensive experiments demonstrate that FCE is accurate and robust to fit contours of scene texts even with highly-curved shapes, and also validate the effectiveness and the good generalization of FCENet for arbitrary-shaped text detection. Furthermore, experimental results show that our FCENet is superior to the state-of-the-art (SOTA) methods on CTW1500 and Total-Text, especially on challenging highly-curved text subset.

29.4.2 Results and models

CTW1500

ICDAR2015

Total Text

29.4.3 Citation

```
@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}
      }
```

29.5 Mask R-CNN

Mask R-CNN

29.5.1 Abstract

We present a conceptually simple, flexible, and general framework for object instance segmentation. Our approach efficiently detects objects in an image while simultaneously generating a high-quality segmentation mask for each instance. The method, called Mask R-CNN, extends Faster R-CNN by adding a branch for predicting an object mask in parallel with the existing branch for bounding box recognition. Mask R-CNN is simple to train and adds only a small overhead to Faster R-CNN, running at 5 fps. Moreover, Mask R-CNN is easy to generalize to other tasks, e.g., allowing us to estimate human poses in the same framework. We show top results in all three tracks of the COCO suite of challenges, including instance segmentation, bounding-box object detection, and person keypoint detection. Without bells and whistles, Mask R-CNN outperforms all existing, single-model entries on every task, including the COCO 2016 challenge winners. We hope our simple and effective approach will serve as a solid baseline and help ease future research in instance-level recognition.

29.5.2 Results and models

CTW1500

ICDAR2015

29.5.3 Citation

```
@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},
```

(continues on next page)

(continued from previous page)

```
  pages={2980-2988},
  doi={10.1109/ICCV.2017.322}}
```

29.6 PANet

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

29.6.1 Abstract

Scene text detection, an important step of scene text reading systems, has witnessed rapid development with convolutional neural networks. Nonetheless, two main challenges still exist and hamper its deployment to real-world applications. The first problem is the trade-off between speed and accuracy. The second one is to model the arbitrary-shaped text instance. Recently, some methods have been proposed to tackle arbitrary-shaped text detection, but they rarely take the speed of the entire pipeline into consideration, which may fall short in practical this http URL this paper, we propose an efficient and accurate arbitrary-shaped text detector, termed Pixel Aggregation Network (PAN), which is equipped with a low computational-cost segmentation head and a learnable post-processing. More specifically, the segmentation head is made up of Feature Pyramid Enhancement Module (FPEM) and Feature Fusion Module (FFM). FPEM is a cascadable U-shaped module, which can introduce multi-level information to guide the better segmentation. FFM can gather the features given by the FPEMs of different depths into a final feature for segmentation. The learnable post-processing is implemented by Pixel Aggregation (PA), which can precisely aggregate text pixels by predicted similarity vectors. Experiments on several standard benchmarks validate the superiority of the proposed PAN. It is worth noting that our method can achieve a competitive F-measure of 79.9% at 84.2 FPS on CTW1500.

29.6.2 Results and models

CTW1500

ICDAR2015

29.6.3 Citation

```
@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}
  }
```

29.7 PSENet

Shape robust text detection with progressive scale expansion network

29.7.1 Abstract

Scene text detection has witnessed rapid progress especially with the recent development of convolutional neural networks. However, there still exists two challenges which prevent the algorithm into industry applications. On the one hand, most of the state-of-art algorithms require quadrangle bounding box which is in-accurate to locate the texts with arbitrary shape. On the other hand, two text instances which are close to each other may lead to a false detection which covers both instances. Traditionally, the segmentation-based approach can relieve the first problem but usually fail to solve the second challenge. To address these two challenges, in this paper, we propose a novel Progressive Scale Expansion Network (PSENet), which can precisely detect text instances with arbitrary shapes. More specifically, PSENet generates the different scale of kernels for each text instance, and gradually expands the minimal scale kernel to the text instance with the complete shape. Due to the fact that there are large geometrical margins among the minimal scale kernels, our method is effective to split the close text instances, making it easier to use segmentation-based methods to detect arbitrary-shaped text instances. Extensive experiments on CTW1500, Total-Text, ICDAR 2015 and ICDAR 2017 MLT validate the effectiveness of PSENet. Notably, on CTW1500, a dataset full of long curve texts, PSENet achieves a F-measure of 74.3% at 27 FPS, and our best F-measure (82.2%) outperforms state-of-art algorithms by 6.6%. The code will be released in the future.

29.7.2 Results and models

CTW1500

ICDAR2015

29.7.3 Citation

```
@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}
}
```

29.8 Textsnake

TextSnake: A Flexible Representation for Detecting Text of Arbitrary Shapes

29.8.1 Abstract

Driven by deep neural networks and large scale datasets, scene text detection methods have progressed substantially over the past years, continuously refreshing the performance records on various standard benchmarks. However, limited by the representations (axis-aligned rectangles, rotated rectangles or quadrangles) adopted to describe text, existing methods may fall short when dealing with much more free-form text instances, such as curved text, which are actually very common in real-world scenarios. To tackle this problem, we propose a more flexible representation for scene text, termed as TextSnake, which is able to effectively represent text instances in horizontal, oriented and curved forms. In TextSnake, a text instance is described as a sequence of ordered, overlapping disks centered at symmetric axes, each of which is associated with potentially variable radius and orientation. Such geometry attributes are estimated via a Fully Convolutional Network (FCN) model. In experiments, the text detector based on TextSnake achieves state-of-the-art or comparable performance on Total-Text and SCUT-CTW1500, the two newly published benchmarks with special emphasis on curved text in natural images, as well as the widely-used datasets ICDAR 2015 and MSRA-TD500. Specifically, TextSnake outperforms the baseline on Total-Text by more than 40% in F-measure.

29.8.2 Results and models

CTW1500

29.8.3 Citation

```
@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}
}
```

CHAPTER

THIRTY

TEXT RECOGNITION MODELS

30.1 ABINet

Read Like Humans: Autonomous, Bidirectional and Iterative Language Modeling for Scene Text Recognition

30.1.1 Abstract

Linguistic knowledge is of great benefit to scene text recognition. However, how to effectively model linguistic rules in end-to-end deep networks remains a research challenge. In this paper, we argue that the limited capacity of language models comes from: 1) implicitly language modeling; 2) unidirectional feature representation; and 3) language model with noise input. Correspondingly, we propose an autonomous, bidirectional and iterative ABINet for scene text recognition. Firstly, the autonomous suggests to block gradient flow between vision and language models to enforce explicitly language modeling. Secondly, a novel bidirectional cloze network (BCN) as the language model is proposed based on bidirectional feature representation. Thirdly, we propose an execution manner of iterative correction for language model which can effectively alleviate the impact of noise input. Additionally, based on the ensemble of iterative predictions, we propose a self-training method which can learn from unlabeled images effectively. Extensive experiments indicate that ABINet has superiority on low-quality images and achieves state-of-the-art results on several mainstream benchmarks. Besides, the ABINet trained with ensemble self-training shows promising improvement in realizing human-level recognition.

30.1.2 Dataset

Train Dataset

Test Dataset

30.1.3 Results and models

Note:

1. ABINet allows its encoder to run and be trained without decoder and fuser. Its encoder is designed to recognize texts as a stand-alone model and therefore can work as an independent text recognizer. We release it as ABINet-Vision.
2. Facts about the pretrained model: MMOCR does not have a systematic pipeline to pretrain the language model (LM) yet, thus the weights of LM are converted from the official pretrained model. The weights of ABINet-Vision are directly used as the vision model of ABINet.

We also provide ABINet trained on Union14M

- Evaluated on six common benchmarks
- Evaluated on Union14M-Benchmark

30.1.4 Citation

```
@article{fang2021read,
  title={Read Like Humans: Autonomous, Bidirectional and Iterative Language Modeling for Scene Text Recognition},
  author={Fang, Shancheng and Xie, Hongtao and Wang, Yuxin and Mao, Zhendong and Zhang, Yongdong},
    booktitle={Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition},
  year={2021}
}
```

30.2 ASTER

ASTER: An Attentional Scene Text Recognizer with Flexible Rectification

30.2.1 Abstract

A challenging aspect of scene text recognition is to handle text with distortions or irregular layout. In particular, perspective text and curved text are common in natural scenes and are difficult to recognize. In this work, we introduce ASTER, an end-to-end neural network model that comprises a rectification network and a recognition network. The rectification network adaptively transforms an input image into a new one, rectifying the text in it. It is powered by a flexible Thin-Plate Spline transformation which handles a variety of text irregularities and is trained without human annotations. The recognition network is an attentional sequence-to-sequence model that predicts a character sequence directly from the rectified image. The whole model is trained end to end, requiring only images and their groundtruth text. Through extensive experiments, we verify the effectiveness of the rectification and demonstrate the state-of-the-art recognition performance of ASTER. Furthermore, we demonstrate that ASTER is a powerful component in end-to-end recognition systems, for its ability to enhance the detector.

30.2.2 Dataset

Train Dataset

Test Dataset

30.2.3 Results and models

We also provide ASTER trained on Union14M

- Evaluated on six common benchmarks
- Evaluated on Union14M-Benchmark

30.2.4 Citation

```
@article{shi2018aster,
  title={Aster: An attentional scene text recognizer with flexible rectification},
  author={Shi, Baoguang and Yang, Mingkun and Wang, Xinggang and Lyu, Pengyuan and Yao,␣
↪Cong and Bai, Xiang},
  journal={IEEE transactions on pattern analysis and machine intelligence},
  volume={41},
  number={9},
  pages={2035--2048},
  year={2018},
  publisher={IEEE}
}
```

30.3 CRNN

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

30.3.1 Abstract

Image-based sequence recognition has been a long-standing research topic in computer vision. In this paper, we investigate the problem of scene text recognition, which is among the most important and challenging tasks in image-based sequence recognition. A novel neural network architecture, which integrates feature extraction, sequence modeling and transcription into a unified framework, is proposed. Compared with previous systems for scene text recognition, the proposed architecture possesses four distinctive properties: (1) It is end-to-end trainable, in contrast to most of the existing algorithms whose components are separately trained and tuned. (2) It naturally handles sequences in arbitrary lengths, involving no character segmentation or horizontal scale normalization. (3) It is not confined to any predefined lexicon and achieves remarkable performances in both lexicon-free and lexicon-based scene text recognition tasks. (4) It generates an effective yet much smaller model, which is more practical for real-world application scenarios. The experiments on standard benchmarks, including the IIIT-5K, Street View Text and ICDAR datasets, demonstrate the superiority of the proposed algorithm over the prior arts. Moreover, the proposed algorithm performs well in the task of image-based music score recognition, which evidently verifies the generality of it.

30.3.2 Dataset

Train Dataset

Test Dataset

30.3.3 Results and models

30.3.4 Citation

```
@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},
```

(continues on next page)

(continued from previous page)

```
  journal={IEEE transactions on pattern analysis and machine intelligence},
  year={2016}
}
```

30.4 MAERec

Revisiting Scene Text Recognition: A Data Perspective

30.4.1 Abstract

This paper aims to re-assess scene text recognition (STR) from a data-oriented perspective. We begin by revisiting the six commonly used benchmarks in STR and observe a trend of performance saturation, whereby only 2.91% of the benchmark images cannot be accurately recognized by an ensemble of 13 representative models. While these results are impressive and suggest that STR could be considered solved, however, we argue that this is primarily due to the less challenging nature of the common benchmarks, thus concealing the underlying issues that STR faces. To this end, we consolidate a large-scale real STR dataset, namely Union14M, which comprises 4 million labeled images and 10 million unlabeled images, to assess the performance of STR models in more complex real-world scenarios. Our experiments demonstrate that the 13 models can only achieve an average accuracy of 66.53% on the 4 million labeled images, indicating that STR still faces numerous challenges in the real world. By analyzing the error patterns of the 13 models, we identify seven open challenges in STR and develop a challenge-driven benchmark consisting of eight distinct subsets to facilitate further progress in the field. Our exploration demonstrates that STR is far from being solved and leveraging data may be a promising solution. In this regard, we find that utilizing the 10 million unlabeled images through self-supervised pre-training can significantly improve the robustness of STR model in real-world scenarios and leads to state-of-the-art performance.

30.4.2 Dataset

Train Dataset

Test Dataset

- On six common benchmarks
- On Union14M-Benchmark

30.4.3 Results and Models

- Evaluated on six common benchmarks
- Evaluated on Union14M-Benchmark
- **To train with MAERec, you need to download pretrained ViT weight and load it in the config file. Check here for instructions**

30.4.4 Citation

```
@misc{jiang2023revisiting,
      title={Revisiting Scene Text Recognition: A Data Perspective},
      author={Qing Jiang and Jiapeng Wang and Dezhi Peng and Chongyu Liu and Lianwen Jin}
↪,
      year={2023},
      eprint={2307.08723},
      archivePrefix={arXiv},
      primaryClass={cs.CV}
}
```

30.5 MASTER

MASTER: Multi-aspect non-local network for scene text recognition

30.5.1 Abstract

Attention-based scene text recognizers have gained huge success, which leverages a more compact intermediate representation to learn 1d- or 2d- attention by a RNN-based encoder-decoder architecture. However, such methods suffer from attention-drift problem because high similarity among encoded features leads to attention confusion under the RNN-based local attention mechanism. Moreover, RNN-based methods have low efficiency due to poor parallelization. To overcome these problems, we propose the MASTER, a self-attention based scene text recognizer that (1) not only encodes the input-output attention but also learns self-attention which encodes feature-feature and target-target relationships inside the encoder and decoder and (2) learns a more powerful and robust intermediate representation to spatial distortion, and (3) owns a great training efficiency because of high training parallelization and a high-speed inference because of an efficient memory-cache mechanism. Extensive experiments on various benchmarks demonstrate the superior performance of our MASTER on both regular and irregular scene text.

30.5.2 Dataset

Train Dataset

Test Dataset

30.5.3 Results and Models

30.5.4 Citation

```
@article{Lu2021MASTER,
  title={MASTER: Multi-Aspect Non-local Network for Scene Text Recognition},
  author={Ning Lu and Wenwen Yu and Xianbiao Qi and Yihao Chen and Ping Gong and Rong␣
↪Xiao and Xiang Bai},
  journal={Pattern Recognition},
  year={2021}
}
```

30.6 NRTR

NRTR: A No-Recurrence Sequence-to-Sequence Model For Scene Text Recognition

30.6.1 Abstract

Scene text recognition has attracted a great many researches due to its importance to various applications. Existing methods mainly adopt recurrence or convolution based networks. Though have obtained good performance, these methods still suffer from two limitations: slow training speed due to the internal recurrence of RNNs, and high complexity due to stacked convolutional layers for long-term feature extraction. This paper, for the first time, proposes a no-recurrence sequence-to-sequence text recognizer, named NRTR, that dispenses with recurrences and convolutions entirely. NRTR follows the encoder-decoder paradigm, where the encoder uses stacked self-attention to extract image features, and the decoder applies stacked self-attention to recognize texts based on encoder output. NRTR relies solely on self-attention mechanism thus could be trained with more parallelization and less complexity. Considering scene image has large variation in text and background, we further design a modality-transform block to effectively transform 2D input images to 1D sequences, combined with the encoder to extract more discriminative features. NRTR achieves state-of-the-art or highly competitive performance on both regular and irregular benchmarks, while requires only a small fraction of training time compared to the best model from the literature (at least 8 times faster).

30.6.2 Dataset

Train Dataset

Test Dataset

30.6.3 Results and Models

We also provide NRTR trained on Union14M

- Evaluated on six common benchmarks
- Evaluated on Union14M-Benchmark

30.6.4 Citation

```
@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}
}
```

30.7 RobustScanner

RobustScanner: Dynamically Enhancing Positional Clues for Robust Text Recognition

30.7.1 Abstract

The attention-based encoder-decoder framework has recently achieved impressive results for scene text recognition, and many variants have emerged with improvements in recognition quality. However, it performs poorly on contextless texts (e.g., random character sequences) which is unacceptable in most of real application scenarios. In this paper, we first deeply investigate the decoding process of the decoder. We empirically find that a representative character-level sequence decoder utilizes not only context information but also positional information. Contextual information, which the existing approaches heavily rely on, causes the problem of attention drift. To suppress such side-effect, we propose a novel position enhancement branch, and dynamically fuse its outputs with those of the decoder attention module for scene text recognition. Specifically, it contains a position aware module to enable the encoder to output feature vectors encoding their own spatial positions, and an attention module to estimate glimpses using the positional clue (i.e., the current decoding time step) only. The dynamic fusion is conducted for more robust feature via an element-wise gate mechanism. Theoretically, our proposed method, dubbed \emph{RobustScanner}, decodes individual characters with dynamic ratio between context and positional clues, and utilizes more positional ones when the decoding sequences with scarce context, and thus is robust and practical. Empirically, it has achieved new state-of-the-art results on popular regular and irregular text recognition benchmarks while without much performance drop on contextless benchmarks, validating its robustness in both contextual and contextless application scenarios.

30.7.2 Dataset

Train Dataset

Test Dataset

30.7.3 Results and Models

30.7.4 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.

30.7.5 Citation

```
@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}
}
```

30.8 SAR

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

30.8.1 Abstract

Recognizing irregular text in natural scene images is challenging due to the large variance in text appearance, such as curvature, orientation and distortion. Most existing approaches rely heavily on sophisticated model designs and/or extra fine-grained annotations, which, to some extent, increase the difficulty in algorithm implementation and data collection. In this work, we propose an easy-to-implement strong baseline for irregular scene text recognition, using off-the-shelf neural network components and only word-level annotations. It is composed of a 31-layer ResNet, an LSTM-based encoder-decoder framework and a 2-dimensional attention module. Despite its simplicity, the proposed method is robust and achieves state-of-the-art performance on both regular and irregular scene text recognition benchmarks.

30.8.2 Dataset

Train Dataset

Test Dataset

30.8.3 Results and Models

We also provide ASTER trained on Union14M

- Evaluated on six common benchmarks
- Evaluated on Union14M-Benchmark

30.8.4 Citation

```
@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}
}
```

30.9 SATRN

On Recognizing Texts of Arbitrary Shapes with 2D Self-Attention

30.9.1 Abstract

Scene text recognition (STR) is the task of recognizing character sequences in natural scenes. While there have been great advances in STR methods, current methods still fail to recognize texts in arbitrary shapes, such as heavily curved or rotated texts, which are abundant in daily life (e.g. restaurant signs, product labels, company logos, etc). This paper introduces a novel architecture to recognizing texts of arbitrary shapes, named Self-Attention Text Recognition Network (SATRN), which is inspired by the Transformer. SATRN utilizes the self-attention mechanism to describe two-dimensional (2D) spatial dependencies of characters in a scene text image. Exploiting the full-graph propagation of self-attention, SATRN can recognize texts with arbitrary arrangements and large inter-character spacing. As a result, SATRN outperforms existing STR models by a large margin of 5.7 pp on average in "irregular text" benchmarks. We provide empirical analyses that illustrate the inner mechanisms and the extent to which the model is applicable (e.g. rotated and multi-line text). We will open-source the code.

30.9.2 Dataset

Train Dataset

Test Dataset

30.9.3 Results and Models

We also provide SATRN trained on Union14M

- Evaluated on six common benchmarks
- Evaluated on Union14M-Benchmark

30.9.4 Citation

```
@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}
}
```

30.10 SVTR

SVTR: Scene Text Recognition with a Single Visual Model

30.10.1 Abstract

Dominant scene text recognition models commonly contain two building blocks, a visual model for feature extraction and a sequence model for text transcription. This hybrid architecture, although accurate, is complex and less efficient. In this study, we propose a Single Visual model for Scene Text recognition within the patch-wise image tokenization framework, which dispenses with the sequential modeling entirely. The method, termed SVTR, firstly decomposes an image text into small patches named character components. Afterward, hierarchical stages are recurrently carried out by component-level mixing, merging and/or combining. Global and local mixing blocks are devised to perceive the inter-character and intra-character patterns, leading to a multi-grained character component perception. Thus, characters are recognized by a simple linear prediction. Experimental results on both English and Chinese scene text recognition tasks demonstrate the effectiveness of SVTR. SVTR-L (Large) achieves highly competitive accuracy in English and outperforms existing methods by a large margin in Chinese, while running faster. In addition, SVTR-T (Tiny) is an effective and much smaller model, which shows appealing speed at inference.

30.10.2 Dataset

Train Dataset

Test Dataset

30.10.3 Results and Models

Note: The implementation and configuration follow the original code and paper, but there is still a gap between the reproduced results and the official ones. We appreciate any suggestions to improve its performance.

30.10.4 Citation

```
@inproceedings{ijcai2022p124,
  title     = {SVTR: Scene Text Recognition with a Single Visual Model},
  author    = {Du, Yongkun and Chen, Zhineng and Jia, Caiyan and Yin, Xiaoting and Zheng,
↪ Tianlun and Li, Chenxia and Du, Yuning and Jiang, Yu-Gang},
  booktitle = {Proceedings of the Thirty-First International Joint Conference on
               Artificial Intelligence, {IJCAI-22}},
  publisher = {International Joint Conferences on Artificial Intelligence Organization},
  editor    = {Lud De Raedt},
  pages     = {884--890},
  year      = {2022},
  month     = {7},
  note      = {Main Track},
  doi       = {10.24963/ijcai.2022/124},
  url       = {https://doi.org/10.24963/ijcai.2022/124},
}
```

CHAPTER

THIRTYONE

KEY INFORMATION EXTRACTION MODELS

31.1 SDMGR

Spatial Dual-Modality Graph Reasoning for Key Information Extraction

31.1.1 Abstract

Key information extraction from document images is of paramount importance in office automation. Conventional template matching based approaches fail to generalize well to document images of unseen templates, and are not robust against text recognition errors. In this paper, we propose an end-to-end Spatial Dual-Modality Graph Reasoning method (SDMG-R) to extract key information from unstructured document images. We model document images as dual-modality graphs, nodes of which encode both the visual and textual features of detected text regions, and edges of which represent the spatial relations between neighboring text regions. The key information extraction is solved by iteratively propagating messages along graph edges and reasoning the categories of graph nodes. In order to roundly evaluate our proposed method as well as boost the future research, we release a new dataset named WildReceipt, which is collected and annotated tailored for the evaluation of key information extraction from document images of unseen templates in the wild. It contains 25 key information categories, a total of about 69000 text boxes, and is about 2 times larger than the existing public datasets. Extensive experiments validate that all information including visual features, textual features and spatial relations can benefit key information extraction. It has been shown that SDMG-R can effectively extract key information from document images of unseen templates, and obtain new state-of-the-art results on the recent popular benchmark SROIE and our WildReceipt. Our code and dataset will be publicly released.

31.1.2 Results and models

WildReceipt

WildReceiptOpenset

31.1.3 Citation

```
@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},
```

(continues on next page)

(continued from previous page)

```
      primaryClass={cs.CV}
}
```

CHAPTER

THIRTYTWO

BRANCHES

This documentation aims to provide a comprehensive understanding of the purpose and features of each branch in MMOCR.

32.1 Branch Overview

32.1.1 1. `main`

The `main` branch serves as the default branch for the MMOCR project. It contains the latest stable version of MMOCR, currently housing the code for MMOCR 1.x (e.g. v1.0.0). The `main` branch ensures users have access to the most recent and reliable version of the software.

32.1.2 2. `dev-1.x`

The `dev-1.x` branch is dedicated to the development of the next major version of MMOCR. This branch will routinely undergo reliance tests, and the passing commits will be squashed in a release and published to the `main` branch. By having a separate development branch, the project can continue to evolve without impacting the stability of the `main` branch. **All the PRs should be merged into the `dev-1.x` branch.**

32.1.3 3. `0.x`

The `0.x` branch serves as an archive for MMOCR 0.x (e.g. v0.6.3). This branch will no longer actively receive updates or improvements, but it remains accessible for historical reference or for users who have not yet upgraded to MMOCR 1.x.

32.1.4 3. `1.x`

It's an alias of `main` branch, which is intended for a smooth transition from the compatibility period. It will be removed in mid 2023.

Note: The branches mapping has been changed in 2023.04.06. For the legacy branches mapping and the guide for migration, please refer to the *branch migration guide*.

CHAPTER

THIRTYTHREE

CONTRIBUTION GUIDE

OpenMMLab welcomes everyone who is interested in contributing to our projects and accepts contribution in the form of PR.

33.1 What is PR

PR is the abbreviation of `Pull Request`. Here's the definition of PR in the official document of Github.

```
Pull requests let you tell others about changes you have pushed to a branch in a␣
↪repository on GitHub. Once a pull request is opened, you can discuss and review the␣
↪potential changes with collaborators and add follow-up commits before your changes are␣
↪merged into the base branch.
```

33.2 Basic Workflow

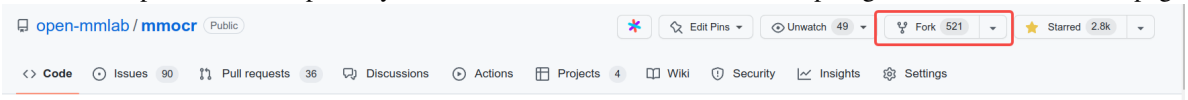
1. Get the most recent codebase
2. Checkout a new branch from `dev-1.x` branch, depending on the version of the codebase you want to contribute to.
3. Commit your changes (*Don't forget to use pre-commit hooks!*)
4. Push your changes and create a PR
5. Discuss and review your code
6. Merge your branch to `dev-1.x` branch

33.3 Procedures in detail

33.3.1 1. Get the most recent codebase

- When you work on your first PR

  Fork the OpenMMLab repository: click the **fork** button at the top right corner of Github page



  Clone forked repository to local

```
git clone git@github.com:XXX/mmocr.git
```

Add source repository to upstream

```
git remote add upstream git@github.com:open-mmlab/mmocr
```

- After your first PR

  Checkout the latest branch of the local repository and pull the latest branch of the source repository. Here we assume that you are working on the `dev-1.x` branch.

```
git checkout dev-1.x
git pull upstream dev-1.x
```

33.3.2 2. Checkout a new branch from `dev-1.x` branch

```
git checkout -b branchname
```

Tip: To make commit history clear, we strongly recommend you checkout the `dev-1.x` branch before creating a new branch.

33.3.3 3. Commit your changes

- If you are a first-time contributor, please install and initialize pre-commit hooks from the repository root directory first.

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

- Commit your changes as usual. Pre-commit hooks will be triggered to stylize your code before each commit.

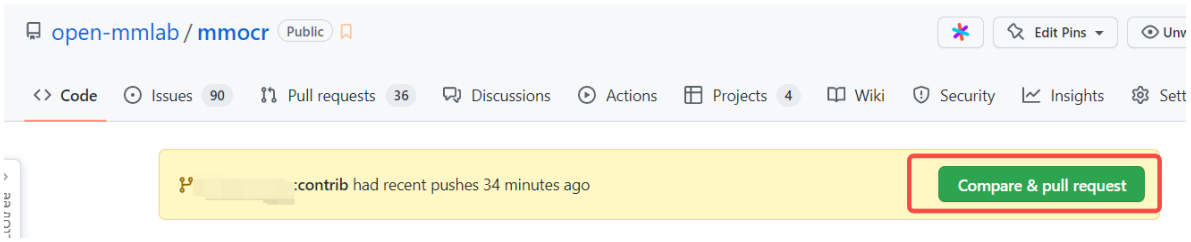
```
# coding
git add [files]
git commit -m 'messages'
```

Note: Sometimes your code may be changed by pre-commit hooks. In this case, please remember to re-stage the modified files and commit again.

33.3.4 4. Push your changes to the forked repository and create a PR

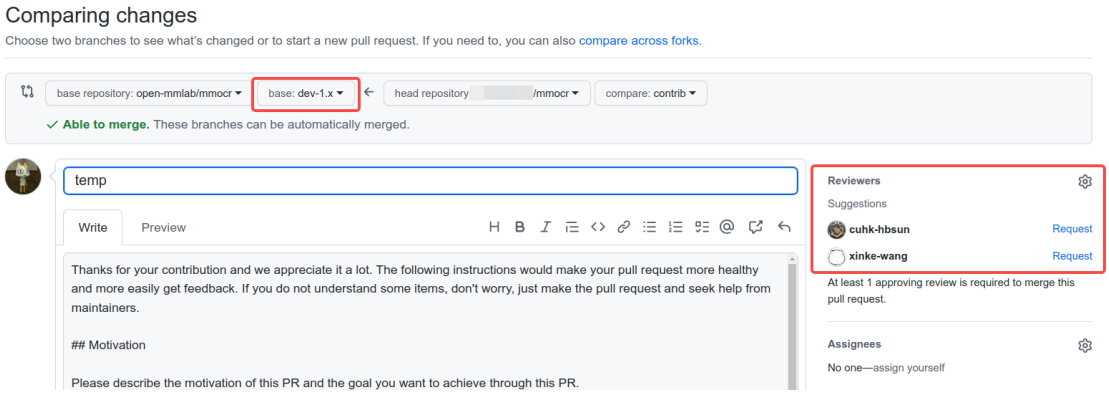
- Push the branch to your forked remote repository

```
git push origin branchname
```



- Create a PR

- Revise PR message template to describe your motivation and modifications made in this PR. You can also link the related issue to the PR manually in the PR message (For more information, checkout the official guidance).
- Specifically, if you are contributing to `dev-1.x`, you will have to change the base branch of the PR to `dev-1.x` in the PR page, since the default base branch is `main`.



- You can also ask a specific person to review the changes you've proposed.

33.3.5 5. Discuss and review your code

- Modify your codes according to reviewers' suggestions and then push your changes.

33.3.6 6. Merge your branch to `dev-1.x` branch and delete the branch

- After the PR is merged by the maintainer, you can delete the branch you created in your forked repository.

```
git branch -d branchname # delete local branch
git push origin --delete branchname # delete remote branch
```

33.4 PR Specs

1. Use pre-commit hook to avoid issues of code style
2. One short-time branch should be matched with only one PR
3. Accomplish a detailed change in one PR. Avoid large PR
   - Bad: Support Faster R-CNN
   - Acceptable: Add a box head to Faster R-CNN
   - Good: Add a parameter to box head to support custom conv-layer number

4. Provide clear and significant commit message
5. Provide clear and meaningful PR description
    - Task name should be clarified in title. The general format is: [Prefix] Short description of the PR (Suffix)
    - Prefix: add new feature [Feature], fix bug [Fix], related to documents [Docs], in developing [WIP] (which will not be reviewed temporarily)
    - Introduce main changes, results and influences on other modules in short description
    - Associate related issues and pull requests with a milestone

CHAPTER

THIRTYFOUR

CHANGELOG OF V1.X

34.1 v1.0.0 (04/06/2023)

We are excited to announce the first official release of MMOCR 1.0, with numerous enhancements, bug fixes, and the introduction of new dataset support!

34.1.1 Highlights

- Support for SCUT-CTW1500, SynthText, and MJSynth datasets
- Updated FAQ and documentation
- Deprecation of file_client_args in favor of backend_args
- Added a new MMOCR tutorial notebook

34.1.2 New Features & Enhancement

- Add SCUT-CTW1500 by @Mountchicken in https://github.com/open-mmlab/mmocr/pull/1677
- Cherry Pick #1205 by @gaotongxiao in https://github.com/open-mmlab/mmocr/pull/1774
- Make lanms-neo optional by @gaotongxiao in https://github.com/open-mmlab/mmocr/pull/1772
- SynthText by @gaotongxiao in https://github.com/open-mmlab/mmocr/pull/1779
- Deprecate file_client_args and use backend_args instead by @gaotongxiao in https://github.com/open-mmlab/mmocr/pull/1765
- MJSynth by @gaotongxiao in https://github.com/open-mmlab/mmocr/pull/1791
- Add MMOCR tutorial notebook by @gaotongxiao in https://github.com/open-mmlab/mmocr/pull/1771
- decouple batch_size to det_batch_size, rec_batch_size and kie_batch_size in MMOCRInferencer by @hugotong6425 in https://github.com/open-mmlab/mmocr/pull/1801
- Accepts local-rank in train.py and test.py by @gaotongxiao in https://github.com/open-mmlab/mmocr/pull/1806
- update stitch_boxes_into_lines by @cherryjm in https://github.com/open-mmlab/mmocr/pull/1824
- Add tests for pytorch 2.0 by @gaotongxiao in https://github.com/open-mmlab/mmocr/pull/1836

34.1.3 Docs

- FAQ by @gaotongxiao in https://github.com/open-mmlab/mmocr/pull/1773
- Remove LoadImageFromLMDB from docs by @gaotongxiao in https://github.com/open-mmlab/mmocr/pull/1767
- Mark projects in docs by @gaotongxiao in https://github.com/open-mmlab/mmocr/pull/1766
- add opendatalab download link by @jorie-peng in https://github.com/open-mmlab/mmocr/pull/1753
- Fix some deadlinks in the docs by @gaotongxiao in https://github.com/open-mmlab/mmocr/pull/1469
- Fix quick run by @gaotongxiao in https://github.com/open-mmlab/mmocr/pull/1775
- Dataset by @gaotongxiao in https://github.com/open-mmlab/mmocr/pull/1782
- Update faq by @gaotongxiao in https://github.com/open-mmlab/mmocr/pull/1817
- more social network links by @fengshiwest in https://github.com/open-mmlab/mmocr/pull/1818
- Update docs after branch switching by @gaotongxiao in https://github.com/open-mmlab/mmocr/pull/1834

34.1.4 Bug Fixes:

- Place dicts to .mim by @gaotongxiao in https://github.com/open-mmlab/mmocr/pull/1781
- Test svtr_small instead of svtr_tiny by @gaotongxiao in https://github.com/open-mmlab/mmocr/pull/1786
- Add pse weight to metafile by @gaotongxiao in https://github.com/open-mmlab/mmocr/pull/1787
- Synthtext metafile by @gaotongxiao in https://github.com/open-mmlab/mmocr/pull/1788
- Clear up some unused scripts by @gaotongxiao in https://github.com/open-mmlab/mmocr/pull/1798
- if dst not exists, when move a single file may raise a file not exists error. by @KevinNuNu in https://github.com/open-mmlab/mmocr/pull/1803
- CTW1500 by @gaotongxiao in https://github.com/open-mmlab/mmocr/pull/1814
- MJSynth & SynthText Dataset Preparer config by @gaotongxiao in https://github.com/open-mmlab/mmocr/pull/1805
- Use poly_intersection instead of poly.intersection to avoid sup… by @gaotongxiao in https://github.com/open-mmlab/mmocr/pull/1811
- Abinet: fix ValueError: Blur limit must be odd when centered=True. Got: (3, 6) by @hugotong6425 in https://github.com/open-mmlab/mmocr/pull/1821
- Bug generated during kie inference visualization by @Yangget in https://github.com/open-mmlab/mmocr/pull/1830
- Revert sync bn in inferencer by @gaotongxiao in https://github.com/open-mmlab/mmocr/pull/1832
- Fix mmdet digit version by @gaotongxiao in https://github.com/open-mmlab/mmocr/pull/1840

34.1.5 New Contributors

- @jorie-peng made their first contribution in https://github.com/open-mmlab/mmocr/pull/1753
- @hugotong6425 made their first contribution in https://github.com/open-mmlab/mmocr/pull/1801
- @fengshiwest made their first contribution in https://github.com/open-mmlab/mmocr/pull/1818
- @cherryjm made their first contribution in https://github.com/open-mmlab/mmocr/pull/1824
- @Yangget made their first contribution in https://github.com/open-mmlab/mmocr/pull/1830

Thank you to all the contributors for making this release possible! We're excited about the new features and enhancements in this version, and we're looking forward to your feedback and continued support. Happy coding!

Full Changelog: https://github.com/open-mmlab/mmocr/compare/v1.0.0rc6...v1.0.0

34.1.6 Highlights

34.2 v1.0.0rc6 (03/07/2023)

34.2.1 Highlights

1. Two new models, ABCNet v2 (inference only) and SPTS are added to `projects/` folder.
2. Announcing `Inferencer`, a unified inference interface in OpenMMLab for everyone's easy access and quick inference with all the pre-trained weights. Docs
3. Users can use test-time augmentation for text recognition tasks. Docs
4. Support batch augmentation through `BatchAugSampler`, which is a technique used in SPTS.
5. Dataset Preparer has been refactored to allow more flexible configurations. Besides, users are now able to prepare text recognition datasets in LMDB formats. Docs
6. Some textspotting datasets have been revised to enhance the correctness and consistency with the common practice.
7. Potential spurious warnings from `shapely` have been eliminated.

34.2.2 Dependency

This version requires MMEngine >= 0.6.0, MMCV >= 2.0.0rc4 and MMDet >= 3.0.0rc5.

34.2.3 New Features & Enhancements

- Discard deprecated lmdb dataset format and only support img+label now by @gaotongxiao in https://github.com/open-mmlab/mmocr/pull/1681
- abcnetv2 inference by @Harold-lkk in https://github.com/open-mmlab/mmocr/pull/1657
- Add RepeatAugSampler by @gaotongxiao in https://github.com/open-mmlab/mmocr/pull/1678
- SPTS by @gaotongxiao in https://github.com/open-mmlab/mmocr/pull/1696
- Refactor Inferencers by @gaotongxiao in https://github.com/open-mmlab/mmocr/pull/1608
- Dynamic return type for rescale_polygons by @gaotongxiao in https://github.com/open-mmlab/mmocr/pull/1702

- Revise upstream version limit by @gaotongxiao in https://github.com/open-mmlab/mmocr/pull/1703
- TextRecogCropConverter add crop with opencv warpPersepective function by @KevinNuNu in https://github.com/open-mmlab/mmocr/pull/1667
- change cudnn benchmark to false by @Harold-lkk in https://github.com/open-mmlab/mmocr/pull/1705
- Add ST-pretrained DB-series models and logs by @gaotongxiao in https://github.com/open-mmlab/mmocr/pull/1635
- Only keep meta and state_dict when publish model by @Harold-lkk in https://github.com/open-mmlab/mmocr/pull/1729
- Rec TTA by @Harold-lkk in https://github.com/open-mmlab/mmocr/pull/1401
- Speedup formatting by replacing np.transpose with torch… by @gaotongxiao in https://github.com/open-mmlab/mmocr/pull/1719
- Support auto import modules from registry. by @Harold-lkk in https://github.com/open-mmlab/mmocr/pull/1731
- Support batch visualization & dumping in Inferencer by @gaotongxiao in https://github.com/open-mmlab/mmocr/pull/1722
- add a new argument font_properties to set a specific font file in order to draw Chinese characters properly by @KevinNuNu in https://github.com/open-mmlab/mmocr/pull/1709
- Refactor data converter and gather by @Harold-lkk in https://github.com/open-mmlab/mmocr/pull/1707
- Support batch augmentation through BatchAugSampler by @gaotongxiao in https://github.com/open-mmlab/mmocr/pull/1757
- Put all registry into registry.py by @Harold-lkk in https://github.com/open-mmlab/mmocr/pull/1760
- train by @gaotongxiao in https://github.com/open-mmlab/mmocr/pull/1756
- configs for regression benchmark by @gaotongxiao in https://github.com/open-mmlab/mmocr/pull/1755
- Support lmdb format in Dataset Preparer by @gaotongxiao in https://github.com/open-mmlab/mmocr/pull/1762

34.2.4 Docs

- update the link of DBNet by @AllentDan in https://github.com/open-mmlab/mmocr/pull/1672
- Add notice for default branch switching by @gaotongxiao in https://github.com/open-mmlab/mmocr/pull/1693
- docs: Add twitter discord medium youtube link by @vansin in https://github.com/open-mmlab/mmocr/pull/1724
- Remove unsupported datasets in docs by @gaotongxiao in https://github.com/open-mmlab/mmocr/pull/1670

34.2.5 Bug Fixes

- Update dockerfile by @gaotongxiao in https://github.com/open-mmlab/mmocr/pull/1671
- Explicitly create np object array for compatibility by @gaotongxiao in https://github.com/open-mmlab/mmocr/pull/1691
- Fix a minor error in docstring by @Mountchicken in https://github.com/open-mmlab/mmocr/pull/1685
- Fix lint by @triple-Mu in https://github.com/open-mmlab/mmocr/pull/1694
- Fix LoadOCRAnnotation ut by @Harold-lkk in https://github.com/open-mmlab/mmocr/pull/1695
- Fix isort pre-commit error by @KevinNuNu in https://github.com/open-mmlab/mmocr/pull/1697

- Update owners by @xinke-wang in https://github.com/open-mmlab/mmocr/pull/1699
- Detect intersection before using shapley.intersection to eliminate spurious warnings by @gaotongxiao in https://github.com/open-mmlab/mmocr/pull/1710
- Fix some inferencer bugs by @gaotongxiao in https://github.com/open-mmlab/mmocr/pull/1706
- Fix textocr ignore flag by @xinke-wang in https://github.com/open-mmlab/mmocr/pull/1712
- Add missing softmax in ASTER forward_test by @Mountchicken in https://github.com/open-mmlab/mmocr/pull/1718
- Fix head in readme by @vansin in https://github.com/open-mmlab/mmocr/pull/1727
- Fix some browse dataset script bugs and draw textdet gt instance with ignore flags by @KevinNuNu in https://github.com/open-mmlab/mmocr/pull/1701
- icdar textrecog ann parser skip data with ignore flag by @KevinNuNu in https://github.com/open-mmlab/mmocr/pull/1708
- bezier_to_polygon -> bezier2polygon by @double22a in https://github.com/open-mmlab/mmocr/pull/1739
- Fix docs recog CharMetric P/R error definition by @KevinNuNu in https://github.com/open-mmlab/mmocr/pull/1740
- Remove outdated resources in demo/ by @gaotongxiao in https://github.com/open-mmlab/mmocr/pull/1747
- Fix wrong ic13 textspotting split data; add lexicons to ic13, ic15 and totaltext by @gaotongxiao in https://github.com/open-mmlab/mmocr/pull/1758
- SPTS readme by @gaotongxiao in https://github.com/open-mmlab/mmocr/pull/1761

34.2.6 New Contributors

- @triple-Mu made their first contribution in https://github.com/open-mmlab/mmocr/pull/1694
- @double22a made their first contribution in https://github.com/open-mmlab/mmocr/pull/1739

Full Changelog: https://github.com/open-mmlab/mmocr/compare/v1.0.0rc5...v1.0.0rc6

34.3 v1.0.0rc5 (01/06/2023)

34.3.1 Highlights

1. Two models, Aster and SVTR, are added to our model zoo. The full implementation of ABCNet is also available now.
2. Dataset Preparer supports 5 more datasets: CocoTextV2, FUNSD, TextOCR, NAF, SROIE.
3. We have 4 more text recognition transforms, and two helper transforms. See https://github.com/open-mmlab/mmocr/pull/1646 https://github.com/open-mmlab/mmocr/pull/1632 https://github.com/open-mmlab/mmocr/pull/1645 for details.
4. The transform, `FixInvalidPolygon`, is getting smarter at dealing with invalid polygons, and now capable of handling more weird annotations. As a result, a complete training cycle on TotalText dataset can be performed bug-free. The weights of DBNet and FCENet pretrained on TotalText are also released.

34.3.2 New Features & Enhancements

- Update ic15 det config according to DataPrepare by @Harold-lkk in https://github.com/open-mmlab/mmocr/pull/1617
- Refactor icdardataset metainfo to lowercase. by @Harold-lkk in https://github.com/open-mmlab/mmocr/pull/1620
- Add ASTER Encoder by @Mountchicken in https://github.com/open-mmlab/mmocr/pull/1239
- Add ASTER decoder by @Mountchicken in https://github.com/open-mmlab/mmocr/pull/1625
- Add ASTER config by @Mountchicken in https://github.com/open-mmlab/mmocr/pull/1238
- Update ASTER config by @Mountchicken in https://github.com/open-mmlab/mmocr/pull/1629
- Support browse_dataset.py to visualize original dataset by @xinke-wang in https://github.com/open-mmlab/mmocr/pull/1503
- Add CocoTextv2 to dataset preparer by @xinke-wang in https://github.com/open-mmlab/mmocr/pull/1514
- Add Funsd to dataset preparer by @xinke-wang in https://github.com/open-mmlab/mmocr/pull/1550
- Add TextOCR to Dataset Preparer by @xinke-wang in https://github.com/open-mmlab/mmocr/pull/1543
- Refine example projects and readme by @gaotongxiao in https://github.com/open-mmlab/mmocr/pull/1628
- Enhance FixInvalidPolygon, add RemoveIgnored transform by @gaotongxiao in https://github.com/open-mmlab/mmocr/pull/1632
- ConditionApply by @Harold-lkk in https://github.com/open-mmlab/mmocr/pull/1646
- Add NAF to dataset preparer by @Mountchicken in https://github.com/open-mmlab/mmocr/pull/1609
- Add SROIE to dataset preparer by @FerryHuang in https://github.com/open-mmlab/mmocr/pull/1639
- Add svtr decoder by @willpat1213 in https://github.com/open-mmlab/mmocr/pull/1448
- Add missing unit tests by @Mountchicken in https://github.com/open-mmlab/mmocr/pull/1651
- Add svtr encoder by @willpat1213 in https://github.com/open-mmlab/mmocr/pull/1483
- ABCNet train by @Harold-lkk in https://github.com/open-mmlab/mmocr/pull/1610
- Totaltext cfgs for DB and FCE by @gaotongxiao in https://github.com/open-mmlab/mmocr/pull/1633
- Add Aliases to models by @gaotongxiao in https://github.com/open-mmlab/mmocr/pull/1611
- SVTR transforms by @gaotongxiao in https://github.com/open-mmlab/mmocr/pull/1645
- Add SVTR framework and configs by @gaotongxiao in https://github.com/open-mmlab/mmocr/pull/1621
- Issue Template by @Harold-lkk in https://github.com/open-mmlab/mmocr/pull/1663

34.3.3 Docs

- Add Chinese translation for browse_dataset.py by @xinke-wang in https://github.com/open-mmlab/mmocr/pull/1647
- updata abcnet doc by @Harold-lkk in https://github.com/open-mmlab/mmocr/pull/1658
- update the dbnetpp`s readme file by @zhuyue66 in https://github.com/open-mmlab/mmocr/pull/1626
- Inferencer docs by @gaotongxiao in https://github.com/open-mmlab/mmocr/pull/1744

34.3.4 Bug Fixes

- nn.SmoothL1Loss beta can not be zero in PyTorch 1.13 version by @Harold-lkk in https://github.com/open-mmlab/mmocr/pull/1616
- ctc loss bug if target is empty by @Harold-lkk in https://github.com/open-mmlab/mmocr/pull/1618
- Add torch 1.13 by @gaotongxiao in https://github.com/open-mmlab/mmocr/pull/1619
- Remove outdated tutorial link by @gaotongxiao in https://github.com/open-mmlab/mmocr/pull/1627
- Dev 1.x some doc mistakes by @KevinNuNu in https://github.com/open-mmlab/mmocr/pull/1630
- Support custom font to visualize some languages (e.g. Korean) by @ProtossDragoon in https://github.com/open-mmlab/mmocr/pull/1567
- db_module_lossnegative number encountered in sqrt by @KevinNuNu in https://github.com/open-mmlab/mmocr/pull/1640
- Use int instead of np.int by @gaotongxiao in https://github.com/open-mmlab/mmocr/pull/1636
- Remove support for py3.6 by @gaotongxiao in https://github.com/open-mmlab/mmocr/pull/1660

34.3.5 New Contributors

- @zhuyue66 made their first contribution in https://github.com/open-mmlab/mmocr/pull/1626
- @KevinNuNu made their first contribution in https://github.com/open-mmlab/mmocr/pull/1630
- @FerryHuang made their first contribution in https://github.com/open-mmlab/mmocr/pull/1639
- @willpat1213 made their first contribution in https://github.com/open-mmlab/mmocr/pull/1448

Full Changelog: https://github.com/open-mmlab/mmocr/compare/v1.0.0rc4...v1.0.0rc5

34.4 v1.0.0rc4 (12/06/2022)

34.4.1 Highlights

1. Dataset Preparer can automatically generate base dataset configs at the end of the preparation process, and supports 6 more datasets: IIIT5k, CUTE80, ICDAR2013, ICDAR2015, SVT, SVTP.
2. Introducing our `projects/` folder - implementing new models and features into OpenMMLab's algorithm libraries has long been complained to be troublesome due to the rigorous requirements on code quality, which could hinder the fast iteration of SOTA models and might discourage community members from sharing their latest outcome here. We now introduce `projects/` folder, where some experimental features, frameworks and models can be placed, only needed to satisfy the minimum requirement on the code quality. Everyone is welcome to post their implementation of any great ideas in this folder! We also add the first example project to illustrate what we expect a good project to have (check out the raw content of README.md for more info!).
3. Inside the `projects/` folder, we are releasing the preview version of ABCNet, which is the first implementation of text spotting models in MMOCR. It's inference-only now, but the full implementation will be available very soon.

34.4.2 New Features & Enhancements

- Add SVT to dataset preparer by @xinke-wang in https://github.com/open-mmlab/mmocr/pull/1521
- Polish bbox2poly by @gaotongxiao in https://github.com/open-mmlab/mmocr/pull/1532
- Add SVTP to dataset preparer by @xinke-wang in https://github.com/open-mmlab/mmocr/pull/1523
- Iiit5k converter by @Harold-lkk in https://github.com/open-mmlab/mmocr/pull/1530
- Add cute80 to dataset preparer by @xinke-wang in https://github.com/open-mmlab/mmocr/pull/1522
- Add IC13 preparer by @xinke-wang in https://github.com/open-mmlab/mmocr/pull/1531
- Add 'Projects/' folder, and the first example project by @gaotongxiao in https://github.com/open-mmlab/mmocr/pull/1524
- Rename to {dataset-name}_task_train/test by @Harold-lkk in https://github.com/open-mmlab/mmocr/pull/1541
- Add print_config.py to the tools by @IncludeMathH in https://github.com/open-mmlab/mmocr/pull/1547
- Add get_md5 by @gaotongxiao in https://github.com/open-mmlab/mmocr/pull/1553
- Add config generator by @gaotongxiao in https://github.com/open-mmlab/mmocr/pull/1552
- Support IC15_1811 by @gaotongxiao in https://github.com/open-mmlab/mmocr/pull/1556
- Update CT80 config by @gaotongxiao in https://github.com/open-mmlab/mmocr/pull/1555
- Add config generators to all textdet and textrecog configs by @gaotongxiao in https://github.com/open-mmlab/mmocr/pull/1560
- Refactor TPS by @Mountchicken in https://github.com/open-mmlab/mmocr/pull/1240
- Add TextSpottingConfigGenerator by @gaotongxiao in https://github.com/open-mmlab/mmocr/pull/1561
- Add common typing by @Harold-lkk in https://github.com/open-mmlab/mmocr/pull/1596
- Update textrecog config and readme by @gaotongxiao in https://github.com/open-mmlab/mmocr/pull/1597
- Support head loss or postprocessor is None for only infer by @Harold-lkk in https://github.com/open-mmlab/mmocr/pull/1594
- Textspotting datasample by @Harold-lkk in https://github.com/open-mmlab/mmocr/pull/1593
- Simplify mono_gather by @gaotongxiao in https://github.com/open-mmlab/mmocr/pull/1588
- ABCNet v1 infer by @Harold-lkk in https://github.com/open-mmlab/mmocr/pull/1598

34.4.3 Docs

- Add Chinese Guidance on How to Add New Datasets to Dataset Preparer by @xinke-wang in https://github.com/open-mmlab/mmocr/pull/1506
- Update the qq group link by @vansin in https://github.com/open-mmlab/mmocr/pull/1569
- Collapse some sections; update logo url by @gaotongxiao in https://github.com/open-mmlab/mmocr/pull/1571
- Update dataset preparer (CN) by @gaotongxiao in https://github.com/open-mmlab/mmocr/pull/1591

34.4.4 Bug Fixes

- Fix two bugs in dataset preparer by @xinke-wang in https://github.com/open-mmlab/mmocr/pull/1513
- Register bug of CLIPResNet by @jyshee in https://github.com/open-mmlab/mmocr/pull/1517
- Being more conservative on Dataset Preparer by @gaotongxiao in https://github.com/open-mmlab/mmocr/pull/1520
- python -m pip upgrade in windows by @Harold-lkk in https://github.com/open-mmlab/mmocr/pull/1525
- Fix wildreceipt metafile by @xinke-wang in https://github.com/open-mmlab/mmocr/pull/1528
- Fix Dataset Preparer Extract by @xinke-wang in https://github.com/open-mmlab/mmocr/pull/1527
- Fix ICDARTxtParser by @xinke-wang in https://github.com/open-mmlab/mmocr/pull/1529
- Fix Dataset Zoo Script by @xinke-wang in https://github.com/open-mmlab/mmocr/pull/1533
- Fix crop without padding and recog metainfo delete unuse info by @Harold-lkk in https://github.com/open-mmlab/mmocr/pull/1526
- Automatically create nonexistent directory for base configs by @gaotongxiao in https://github.com/open-mmlab/mmocr/pull/1535
- Change mmcv.dump to mmengine.dump by @ProtossDragoon in https://github.com/open-mmlab/mmocr/pull/1540
- mmocr.utils.typing -> mmocr.utils.typing_utils by @gaotongxiao in https://github.com/open-mmlab/mmocr/pull/1538
- Wildreceipt tests by @gaotongxiao in https://github.com/open-mmlab/mmocr/pull/1546
- Fix judge exist dir by @Harold-lkk in https://github.com/open-mmlab/mmocr/pull/1542
- Fix IC13 textdet config by @xinke-wang in https://github.com/open-mmlab/mmocr/pull/1563
- Fix IC13 textrecog annotations by @gaotongxiao in https://github.com/open-mmlab/mmocr/pull/1568
- Auto scale lr by @gaotongxiao in https://github.com/open-mmlab/mmocr/pull/1584
- Fix icdar data parse for text containing separator by @Harold-lkk in https://github.com/open-mmlab/mmocr/pull/1587
- Fix textspotting ut by @Harold-lkk in https://github.com/open-mmlab/mmocr/pull/1599
- Fix TextSpottingConfigGenerator and TextSpottingDataConverter by @gaotongxiao in https://github.com/open-mmlab/mmocr/pull/1604
- Keep E2E Inferencer output simple by @gaotongxiao in https://github.com/open-mmlab/mmocr/pull/1559

34.4.5 New Contributors

- @jyshee made their first contribution in https://github.com/open-mmlab/mmocr/pull/1517
- @ProtossDragoon made their first contribution in https://github.com/open-mmlab/mmocr/pull/1540
- @IncludeMathH made their first contribution in https://github.com/open-mmlab/mmocr/pull/1547

Full Changelog: https://github.com/open-mmlab/mmocr/compare/v1.0.0rc3...v1.0.0rc4

34.5 v1.0.0rc3 (11/03/2022)

34.5.1 Highlights

1. We release several pretrained models using oCLIP-ResNet as the backbone, which is a ResNet variant trained with oCLIP and can significantly boost the performance of text detection models.
2. Preparing datasets is troublesome and tedious, especially in OCR domain where multiple datasets are usually required. In order to free our users from laborious work, we designed a Dataset Preparer to help you get a bunch of datasets ready for use, with only **one line of command**! Dataset Preparer is also crafted to consist of a series of reusable modules, each responsible for handling one of the standardized phases throughout the preparation process, shortening the development cycle on supporting new datasets.

34.5.2 New Features & Enhancements

- Add Dataset Preparer by @xinke-wang in https://github.com/open-mmlab/mmocr/pull/1484
- support modified resnet structure used in oCLIP by @HannibalAPE in https://github.com/open-mmlab/mmocr/pull/1458
- Add oCLIP configs by @gaotongxiao in https://github.com/open-mmlab/mmocr/pull/1509

34.5.3 Docs

- Update install.md by @rogachevai in https://github.com/open-mmlab/mmocr/pull/1494
- Refine some docs by @gaotongxiao in https://github.com/open-mmlab/mmocr/pull/1455
- Update some dataset preparer related docs by @xinke-wang in https://github.com/open-mmlab/mmocr/pull/1502
- oclip readme by @Harold-lkk in https://github.com/open-mmlab/mmocr/pull/1505

34.5.4 Bug Fixes

- Fix offline_eval error caused by new data flow by @gaotongxiao in https://github.com/open-mmlab/mmocr/pull/1500

34.5.5 New Contributors

- @rogachevai made their first contribution in https://github.com/open-mmlab/mmocr/pull/1494
- @HannibalAPE made their first contribution in https://github.com/open-mmlab/mmocr/pull/1458

Full Changelog: https://github.com/open-mmlab/mmocr/compare/v1.0.0rc2...v1.0.0rc3

34.6 v1.0.0rc2 (10/14/2022)

This release relaxes the version requirement of `MMEngine` to `>=0.1.0, < 1.0.0`.

34.7 v1.0.0rc1 (10/09/2022)

34.7.1 Highlights

This release fixes a severe bug leading to inaccurate metric report in multi-GPU training. We release the weights for all the text recognition models in MMOCR 1.0 architecture. The inference shorthand for them are also added back to `ocr.py`. Besides, more documentation chapters are available now.

34.7.2 New Features & Enhancements

- Simplify the Mask R-CNN config by @xinke-wang in https://github.com/open-mmlab/mmocr/pull/1391
- auto scale lr by @Harold-lkk in https://github.com/open-mmlab/mmocr/pull/1326
- Update paths to pretrain weights by @gaotongxiao in https://github.com/open-mmlab/mmocr/pull/1416
- Streamline duplicated split_result in pan_postprocessor by @gaotongxiao in https://github.com/open-mmlab/mmocr/pull/1418
- Update model links in ocr.py and inference.md by @gaotongxiao in https://github.com/open-mmlab/mmocr/pull/1431
- Update rec configs by @gaotongxiao in https://github.com/open-mmlab/mmocr/pull/1417
- Visualizer refine by @Harold-lkk in https://github.com/open-mmlab/mmocr/pull/1411
- Support get flops and parameters in dev-1.x by @vansin in https://github.com/open-mmlab/mmocr/pull/1414

34.7.3 Docs

- intersphinx and api by @Harold-lkk in https://github.com/open-mmlab/mmocr/pull/1367
- Fix quickrun by @gaotongxiao in https://github.com/open-mmlab/mmocr/pull/1374
- Fix some docs issues by @gaotongxiao in https://github.com/open-mmlab/mmocr/pull/1385
- Add Documents for DataElements by @xinke-wang in https://github.com/open-mmlab/mmocr/pull/1381
- config english by @Harold-lkk in https://github.com/open-mmlab/mmocr/pull/1372
- Metrics by @xinke-wang in https://github.com/open-mmlab/mmocr/pull/1399
- Add version switcher to menu by @gaotongxiao in https://github.com/open-mmlab/mmocr/pull/1407
- Data Transforms by @xinke-wang in https://github.com/open-mmlab/mmocr/pull/1392
- Fix inference docs by @gaotongxiao in https://github.com/open-mmlab/mmocr/pull/1415
- Fix some docs by @xinke-wang in https://github.com/open-mmlab/mmocr/pull/1410
- Add maintenance plan to migration guide by @xinke-wang in https://github.com/open-mmlab/mmocr/pull/1413
- Update Recog Models by @xinke-wang in https://github.com/open-mmlab/mmocr/pull/1402

34.7.4 Bug Fixes

- clear metric.results only done in main process by @Harold-lkk in https://github.com/open-mmlab/mmocr/pull/1379
- Fix a bug in MMDetWrapper by @xinke-wang in https://github.com/open-mmlab/mmocr/pull/1393
- Fix browse_dataset.py by @Mountchicken in https://github.com/open-mmlab/mmocr/pull/1398
- ImgAugWrapper: Do not cilp polygons if not applicable by @gaotongxiao in https://github.com/open-mmlab/mmocr/pull/1231
- Fix CI by @gaotongxiao in https://github.com/open-mmlab/mmocr/pull/1365
- Fix merge stage test by @gaotongxiao in https://github.com/open-mmlab/mmocr/pull/1370
- Del CI support for torch 1.5.1 by @gaotongxiao in https://github.com/open-mmlab/mmocr/pull/1371
- Test windows cu111 by @gaotongxiao in https://github.com/open-mmlab/mmocr/pull/1373
- Fix windows CI by @gaotongxiao in https://github.com/open-mmlab/mmocr/pull/1387
- Upgrade pre commit hooks by @Harold-lkk in https://github.com/open-mmlab/mmocr/pull/1429
- Skip invalid augmented polygons in ImgAugWrapper by @gaotongxiao in https://github.com/open-mmlab/mmocr/pull/1434

34.7.5 New Contributors

- @vansin made their first contribution in https://github.com/open-mmlab/mmocr/pull/1414

Full Changelog: https://github.com/open-mmlab/mmocr/compare/v1.0.0rc0...v1.0.0rc1

34.8 v1.0.0rc0 (09/01/2022)

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

34.8.1 Highlights

1. **New engines**. MMOCR 1.x is based on MMEngine, which provides a general and powerful runner that allows more flexible customizations and significantly simplifies the entrypoints of high-level interfaces.
2. **Unified interfaces**. As a part of the OpenMMLab 2.0 projects, MMOCR 1.x unifies and refactors the interfaces and internal logics of train, testing, datasets, models, evaluation, and visualization. All the OpenMMLab 2.0 projects share the same design in those interfaces and logics to allow the emergence of multi-task/modality algorithms.
3. **Cross project calling**. Benefiting from the unified design, you can use the models implemented in other OpenMMLab projects, such as MMDet. We provide an example of how to use MMDetection's Mask R-CNN through `MMDetWrapper`. Check our documents for more details. More wrappers will be released in the future.
4. **Stronger visualization**. We provide a series of useful tools which are mostly based on brand-new visualizers. As a result, it is more convenient for the users to explore the models and datasets now.
5. **More documentation and tutorials**. We add a bunch of documentation and tutorials to help users get started more smoothly. Read it here.

34.8.2 Breaking Changes

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

Dependencies

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

Training and testing

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

Configs

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

Dataset

The Dataset classes implemented in MMOCR 1.x all inherits from the `BaseDetDataset`, which inherits from the BaseDataset in MMEngine. There are several changes of Dataset in MMOCR 1.x.

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

Data Transforms

The data transforms in MMOCR 1.x all inherits from those in MMCV>=2.0.0rc0, which follows a new convention in OpenMMLab 2.0 projects. The changes are listed as below:

- The interfaces are also changed. Please refer to the API Reference
- The functionality of some data transforms (e.g., `Resize`) are decomposed into several transforms.
- The same data transforms in different OpenMMLab 2.0 libraries have the same augmentation implementation and the logic of the same arguments, i.e., `Resize` in MMDet 3.x and MMOCR 1.x will resize the image in the exact same manner given the same arguments.

Model

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

- The model interfaces, including the input and output formats, are significantly simplified and unified following the new convention in MMOCR 1.x. Specifically, all the input data in training and testing are packed into `inputs` and `data_samples`, where `inputs` contains model inputs like a list of image tensors, and `data_samples` contains other information of the current data sample such as ground truths and model predictions. In this way, different tasks in MMOCR 1.x can share the same input arguments, which makes the models more general and suitable for multi-task learning.
- The model has a data preprocessor module, which is used to pre-process the input data of model. In MMOCR 1.x, the data preprocessor usually does necessary steps to form the input images into a batch, such as padding. It can also serve as a place for some special data augmentations or more efficient data transformations like normalization.
- The internal logic of model have been changed. In MMOCR 0.x, model used `forward_train` and `simple_test` to deal with different model forward logics. In MMOCR 1.x and OpenMMLab 2.0, the forward function has three modes: `loss`, `predict`, and `tensor` for training, inference, and tracing or other purposes, respectively. The forward function calls `self.loss()`, `self.predict()`, and `self._forward()` given the modes `loss`, `predict`, and `tensor`, respectively.

Evaluation

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

Visualization

The functions of visualization in MMOCR 1.x are removed. Instead, in OpenMMLab 2.0 projects, we use Visualizer to visualize data. MMOCR 1.x implements `TextDetLocalVisualizer`, `TextRecogLocalVisualizer`, and `KIELocalVisualizer` to allow visualization of ground truths, model predictions, and feature maps, etc., at any place, for the three tasks supported in MMOCR. It also supports to dump the visualization data to any external visualization backends such as Tensorboard and Wandb. Check our Visualization Document for more details.

34.8.3 Improvements

- Most models enjoy a performance improvement from the new framework and refactor of data transforms. For example, in MMOCR 1.x, DBNet-R50 achieves **0.854** hmean score on ICDAR 2015, while the counterpart can only get **0.840** hmean score in MMOCR 0.x.
- Support mixed precision training of most of the models. However, the rest models are not supported yet because the operators they used might not be representable in fp16. We will update the documentation and list the results of mixed precision training.

34.8.4 Ongoing changes

1. Test-time augmentation: which was supported in MMOCR 0.x, is not implemented yet in this version due to limited time slot. We will support it in the following releases with a new and simplified design.
2. Inference interfaces: a unified inference interfaces will be supported in the future to ease the use of released models.
3. Interfaces of useful tools that can be used in notebook: more useful tools that implemented in the `tools/` directory will have their python interfaces so that they can be used through notebook and in downstream libraries.
4. Documentation: we will add more design docs, tutorials, and migration guidance so that the community can deep dive into our new design, participate the future development, and smoothly migrate downstream libraries to MMOCR 1.x.

CHAPTER

THIRTYFIVE

OVERVIEW

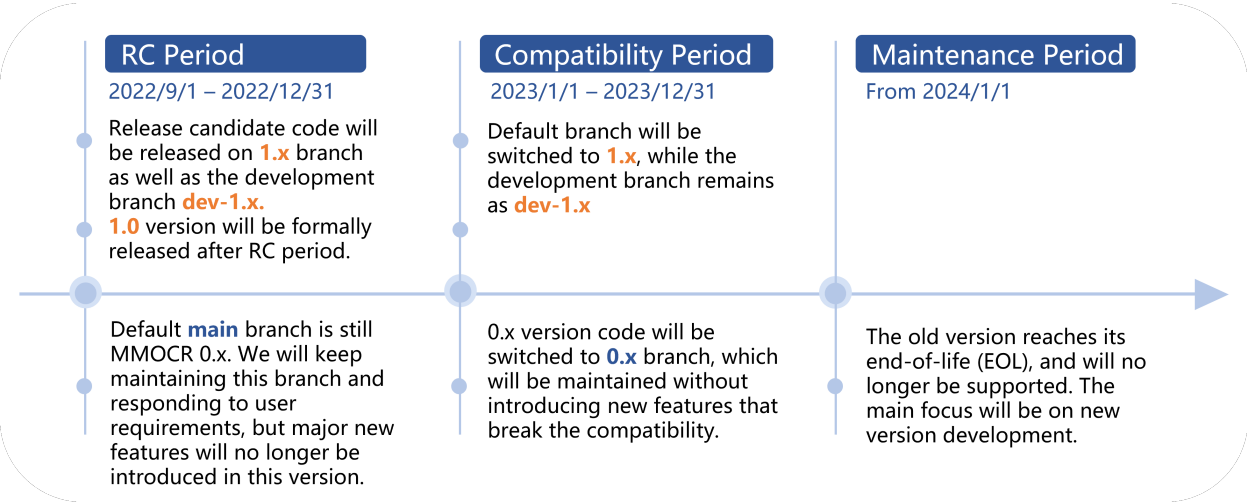
Along with the release of OpenMMLab 2.0, MMOCR 1.0 made many significant changes, resulting in less redundant, more efficient code and a more consistent overall design. However, these changes break backward compatibility. We understand that with such huge changes, it is not easy for users familiar with the old version to adapt to the new version. Therefore, we prepared a detailed migration guide to make the transition as smooth as possible so that all users can enjoy the productivity benefits of the new MMOCR and the entire OpenMMLab 2.0 ecosystem.

Warning: MMOCR 1.0 depends on the new foundational library for training deep learning models MMEngine, and therefore has an entirely different dependency chain compared with MMOCR 0.x. Even if you have a well-rounded MMOCR 0.x environment before, you still need to create a new python environment for MMOCR 1.0. We provide a detailed *installation guide* for reference.

Next, please read the sections according to your requirements.

- Read *What's new in MMOCR 1.x* to learn about the new features and changes in MMOCR 1.x.
- If you want to migrate a model trained in version 0.x to use it directly in version 1.0, please read *Pretrained Model Migration*.
- If you want to train the model, please read *Dataset Migration* and *Data Transform Migration*.
- If you want to develop on MMOCR, please read *Code Migration*, *Branch Migration* and Upstream Library Changes.

As shown in the following figure, the maintenance plan of MMOCR 1.x version is mainly divided into three stages, namely "RC Period", "Compatibility Period" and "Maintenance Period". For old versions, we will no longer add major new features. Therefore, we strongly recommend users to migrate to MMOCR 1.x version as soon as possible.



CHAPTER

THIRTYSIX

WHAT'S NEW IN MMOCR 1.X

Here are some highlights of MMOCR 1.x compared to 0.x.

1. **New engines**. MMOCR 1.x is based on MMEngine, which provides a general and powerful runner that allows more flexible customizations and significantly simplifies the entrypoints of high-level interfaces.
2. **Unified interfaces**. As a part of the OpenMMLab 2.0 projects, MMOCR 1.x unifies and refactors the interfaces and internal logics of train, testing, datasets, models, evaluation, and visualization. All the OpenMMLab 2.0 projects share the same design in those interfaces and logics to allow the emergence of multi-task/modality algorithms.
3. **Cross project calling**. Benefiting from the unified design, you can use the models implemented in other OpenMMLab projects, such as MMDet. We provide an example of how to use MMDetection's Mask R-CNN through `MMDetWrapper`. Check our documents for more details. More wrappers will be released in the future.
4. **Stronger visualization**. We provide a series of useful tools which are mostly based on brand-new visualizers. As a result, it is more convenient for the users to explore the models and datasets now.
5. **More documentation and tutorials**. We add a bunch of documentation and tutorials to help users get started more smoothly.
6. **One-stop Dataset Preparaion**. Multiple datasets are instantly ready with only one line of command, via our Dataset Preparer.
7. **Embracing more `projects/`**: We now introduce `projects/` folder, where some experimental features, frameworks and models can be placed, only needed to satisfy the minimum requirement on the code quality. Everyone is welcome to post their implementation of any great ideas in this folder! Learn more from our example project.
8. **More models**. MMOCR 1.0 supports more tasks and more state-of-the-art models!

CHAPTER

THIRTYSEVEN

BRANCH MIGRATION

At an earlier stage, MMOCR had three branches: `main`, `1.x`, and `dev-1.x`. Some of these branches have been renamed together with the official MMOCR 1.0.0 release, and here is the changelog.

- `main` branch housed the code for MMOCR 0.x (e.g., v0.6.3). Now it has been renamed to `0.x`.
- `1.x` contained the code for MMOCR 1.x (e.g., 1.0.0rc6). Now it is an alias of `main`, and will be removed in mid 2023.
- `dev-1.x` was the development branch for MMOCR 1.x. Now it remains unchanged.

For more information about the branches, check out *branches*.

37.1 Resolving Conflicts When Upgrading the `main` branch

For users who wish to upgrade from the old `main` branch that has the code for MMOCR 0.x, the non-fast-forwarded-able nature of the upgrade may cause conflicts. To resolve these conflicts, follow the steps below:

1. Commit all the changes you have on `main` if you have any. Backup your current `main` branch by creating a copy.

```
git checkout main
git add --all
git commit -m 'backup'
git checkout -b main_backup
```

2. Fetch the latest changes from the remote repository.

```
git remote add openmmlab git@github.com:open-mmlab/mmocr.git
git fetch openmmlab
```

3. Reset the `main` branch to the latest `main` branch on the remote repository by running `git reset --hard openmmlab/main`.

```
git checkout main
git reset --hard openmmlab/main
```

By following these steps, you can successfully upgrade your `main` branch.

CHAPTER

THIRTYEIGHT

CODE MIGRATION

MMOCR has been designed in a way that there are a lot of shortcomings in the initial version in order to balance the tasks of text detection, recognition and key information extraction. In this 1.0 release, MMOCR synchronizes its new model architecture to align as much as possible with the overall OpenMMLab design and to achieve structural uniformity within the algorithm library. Although this upgrade is not fully backward compatible, we summarize the changes that may be of interest to developers for those who need them.

38.1 Fundamental Changes

Functional boundaries of modules has not been clearly defined in MMOCR 0.x. In MMOCR 1.0, we address this issue by refactoring the design of model modules. Here are some major changes in 1.0:

- MMOCR 1.0 no longer supports named entity recognition tasks since it's not in the scope of OCR.
- The module that computes the loss in a model is named as *Module Loss*, which is also responsible for the conversion of gold annotations into loss targets. Another module, *Postprocessor*, is responsible for decoding the model raw output into `DataSample` for the corresponding task at prediction time.
- The inputs of all models are now organized as a dictionary that consists of two keys: `inputs`, containing the original features of the images, and `List[DataSample]`, containing the meta-information of the images. At training time, the output format of a model is standardized to a dictionary containing the loss tensors. Similarly, a model generates a sequence of `DataSample`s containing the prediction outputs in testing.
- In MMOCR 0.x, the majority of classes named `XXLoss` have the implementations closely bound to the corresponding model, while their names made users hard to tell them apart from other generic losses like `DiceLoss`. In 1.0, they are renamed to the form `XXModuleLoss`. (e.g. `DBLoss` was renamed to `DBModuleLoss`). The key to their configurations in config files is also changed from `loss` to `module_loss`.
- The names of generic loss classes that are not related to the model implementation are kept as `XXLoss`. (e.g. `MaskedBCELoss`) They are all placed under `mmocr/models/common/losses`.
- Changes under `mmocr/models/common/losses`: `DiceLoss` is renamed to `MaskedDiceLoss`. `FocalLoss` has been removed.
- MMOCR 1.0 adds a *Dictionary* module which originates from *label converter*. It is used in text recognition and key information extraction tasks.

38.2 Text Detection Models

38.2.1 Key Changes (TL;DR)

- The model weights from MMOCR 0.x still works in the 1.0, but the fields starting with `bbox_head` in the state dict `state_dict` need to be renamed to `det_head`.
- `XXTargets` transforms, which were responsible for genearting detection targets, have been merged into `XXModuleLoss`.

38.2.2 SingleStageTextDetector

- The original inheritance chain was `mmdet.BaseDetector->SingleStageDetector->SingleStageTextDetector`. Now `SingleStageTextDetector` is directly inherited from `BaseDetector` without extra dependency on MMDetection, and `SingleStageDetector` is deleted.
- `bbox_head` is renamed to `det_head`.
- `train_cfg`, `test_cfg` and `pretrained` fields are removed.
- `forward_train()` and `simple_test()` are refactored to `loss()` and `predict()`. The part of `simple_test()` that was responsible for splitting the raw output of the model and feeding it into `head.get_bounary()` is integrated into `BaseTextDetPostProcessor`.
- `TextDetectorMixin` has been removed since its implementation overlaps with `TextDetLocalVisualizer`.

38.2.3 Head

- `HeadMixin`, the base class that `XXXHead` had to inherit from in version 0.x, has been replaced by `BaseTextDetHead`. `get_boundary()` and `resize_boundary()` are now rewritten as `__call__()` and `rescale()` in `BaseTextDetPostProcessor`.

38.2.4 ModuleLoss

- Data transforms `XXXTargets` in text detection tasks are all moved to `XXXModuleLoss._get_target_single()`. Target-related configurations are no longer specified in the data pipeline but in `XXXLoss` instead.

38.2.5 Postprocessor

- The logic in the original `XXXPostprocessor.__call__()` are transferred to the refactored `XXXPostprocessor.get_text_instances()`.
- `BasePostprocessor` is refactored to `BaseTextDetPostProcessor`. This base class splits and processes the model output predictions one by one and supports automatic scaling of the output polygon or bounding box based on `scale_factor`.

38.3 Text Recognition

38.3.1 Key Changes (TL;DR)

- Due to the change of the character order and some bugs in the model architecture being fixed, the recognition model weights in 0.x can no longer be directly used in 1.0. We will provide a migration script and tutorial for those who need it.
- The support of SegOCR has been removed. TPS-CRNN will still be supported in a later version.
- Test time augmentation will be supported in the upcoming release.
- *Label converter* module has been removed and its functions have been split into *Dictionary*, *ModuleLoss* and *Postprocessor*.
- The definition of `max_seq_len` has been unified and now it represents the original output length of the model.

38.3.2 Label Converter

- The original label converters had spelling errors (written as label convertors). We fixed them by removing label converters from this project.
- The part responsible for converting characters/strings to and from numeric indexes was extracted to *Dictionary*.
- In older versions, different label converters would have different special character sets and character order. In version 0.x, the character order was as follows.

In 1.0, instead of designing different dictionaries and character orders for different tasks, we have a unified *Dictionary* implementation with the character order always as characters, <BOS/EOS>, <PAD>, <UKN>. <BLK> in `CTCConvertor` has been equivalently replaced by <PAD>.

- *Label convertor* originally supported three ways to initialize dictionaries: `dict_type`, `dict_file` and `dict_list`, which are now reduced to `dict_file` only in `Dictionary`. Also, we have put those pre-defined character sets originally supported in `dict_type` into `dicts/` directory now. The corresponding mapping is as follows:
- The implementation of `str2tensor()` in *label converter* has been moved to `ModuleLoss.get_targets()`. The following table shows the correspondence between the old and new method implementations. Note that the old and new implementations are not identical.
- The implementation of `tensor2idx()` in *label converter* has been moved to `Postprocessor.get_single_prediction()`. The following table shows the correspondence between the old and new method implementations. Note that the old and new implementations are not identical.

38.4 Key Information Extraction

38.4.1 Key Changes (TL;DR)

- Due to changes in the inputs to the model, the model weights obtained in 0.x can no longer be directly used in 1.0.

38.4.2 KIEDataset & OpensetKIEDataset

- The part that reads data is kept in `WildReceiptDataset`.
- The part that additionally processes the nodes and edges is moved to `LoadKIEAnnotation`.
- The part that uses dictionaries to transform text is moved to `SDMGRHead.convert_text()`, with the help of *Dictionary*.
- The part of `compute_relation()` that computes the relationships between text boxes is moved to `SDMGRHead.compute_relations()`. It's now done inside the model.
- The part that evaluates the model performance is done in `F1Metric`.
- The part of `OpensetKIEDataset` that processes model's edge outputs is moved to `SDMGRPostProcessor`.

38.4.3 SDMGR

- `show_result()` is integrated into `KIEVisualizer`.
- The part of `forward_test()` that post-processes the output is organized in `SDMGRPostProcessor`.

38.5 Utils Migration

Utility functions are now grouped together under `mmocr/utils/`. Here are the scopes of the files in this directory:

- bbox_utils.py: bounding box related functions.
- check_argument.py: used to check argument type.
- collect_env.py: used to collect running environment.
- data_converter_utils.py: used for data format conversion.
- fileio.py: file input and output related functions.
- img_utils.py: image processing related functions.
- mask_utils.py: mask related functions.
- ocr.py: used for MMOCR inference.
- parsers.py: used for parsing datasets.
- polygon_utils.py: polygon related functions.
- setup_env.py: used for initialize MMOCR.
- string_utils.py: string related functions.
- typing.py: defines the abbreviation of types used in MMOCR.

CHAPTER

THIRTYNINE

DATASET MIGRATION

Based on the new design of BaseDataset in MMEngine, we have refactored the base OCR dataset class *OCRDataset* in MMOCR 1.0. The following document describes the differences between the old and new dataset formats in MMOCR, and how to migrate from the deprecated version to the latest. For users who do not want to migrate datasets at this time, we also provide a temporary solution in *Section Compatibility*.

Note: The Key Information Extraction task still uses the original WildReceipt dataset annotation format.

39.1 Review of Old Dataset Formats

MMOCR version 0.x implements a number of dataset classes, such as `IcdarDataset`, `TextDetDataset` for text detection tasks, and `OCRDataset`, `OCRSegDataset` for text recognition tasks. At the same time, the annotations may vary in different formats, such as `.txt`, `.json`, `.jsonl`. Users have to manually configure the `Loader` and the `Parser` while customizing the datasets.

39.1.1 Text Detection

For the text detection task, `IcdarDataset` uses a COCO-like annotation format.

```
{
  "images": [
    {
      "id": 1,
      "width": 800,
      "height": 600,
      "file_name": "test.jpg"
    }
  ],
  "annotations": [
    {
      "id": 1,
      "image_id": 1,
      "category_id": 1,
      "bbox": [0,0,10,10],
      "segmentation": [
          [0,0,10,0,10,10,0,10]
      ],
```

(continues on next page)

(continued from previous page)

```
      "area": 100,
      "iscrowd": 0
    }
  ]
}
```

The `TextDetDataset` uses the JSON Line storage format, converting COCO-like labels to strings and saves them in `.txt` or `.jsonl` format files.

```
{"file_name": "test/img_2.jpg", "height": 720, "width": 1280,  "annotations": [{"iscrowd
↪": 0, "category_id": 1, "bbox": [602.0, 173.0,  33.0, 24.0], "segmentation": [[602,␣
↪173, 635, 175, 634, 197, 602,  196]]}, {"iscrowd": 0, "category_id": 1, "bbox": [734.0,
↪ 310.0, 58.0,  54.0], "segmentation": [[734, 310, 792, 320, 792, 364, 738, 361]]}]}
{"file_name": "test/img_5.jpg", "height": 720, "width": 1280,  "annotations": [{"iscrowd
↪": 1, "category_id": 1, "bbox": [405.0, 409.0,  32.0, 52.0], "segmentation": [[408,␣
↪409, 437, 436, 434, 461, 405,  433]]}, {"iscrowd": 1, "category_id": 1, "bbox": [435.0,
↪ 434.0, 8.0,  33.0], "segmentation": [[437, 434, 443, 440, 441, 467, 435, 462]]}]}
```

39.1.2 Text Recognition

For text recognition tasks, there are two annotation formats in MMOCR version 0.x. The simple `.txt` annotations separate image name and word annotation by a blank space, which cannot handle the case when spaces are included in a text instance.

```
img1.jpg OpenMMLab
img2.jpg MMOCR
```

The JSON Line format uses a dictionary-like structure to represent the annotations, where the keys `filename` and `text` store the image name and word label, respectively.

```
{"filename": "img1.jpg", "text": "OpenMMLab"}
{"filename": "img2.jpg", "text": "MMOCR"}
```

39.2 New Dataset Format

To solve the dataset issues, MMOCR 1.x adopts a unified dataset design introduced in MMEngine. Each annotation file is a `.json` file that stores a `dict`, containing both `metainfo` and `data_list`, where the former includes basic information about the dataset and the latter consists of the label item of each target instance.

```
{
  "metainfo":
    {
      "classes": ("cat", "dog"),
      // ...
    },
  "data_list":
    [
      {
        "img_path": "xxx/xxx_0.jpg",
```

(continues on next page)

(continued from previous page)

```
        "img_label": 0,
        // ...
      },
      // ...
    ]
}
```

Based on the above structure, we introduced `TextDetDataset`, `TextRecogDataset` for MMOCR-specific tasks.

39.2.1 Text Detection

Introduction of the New Format

The `TextDetDataset` holds the information required by the text detection task, such as bounding boxes and labels. We refer users to `tests/data/det_toy_dataset/instances_test.json` which is an example annotation for `TextDetDataset`.

```
{
  "metainfo":
    {
      "dataset_type": "TextDetDataset",
      "task_name": "textdet",
      "category": [{"id": 0, "name": "text"}]
    },
  "data_list":
    [
      {
        "img_path": "test_img.jpg",
        "height": 640,
        "width": 640,
        "instances":
          [
            {
              "polygon": [0, 0, 0, 10, 10, 20, 20, 0],
              "bbox": [0, 0, 10, 20],
              "bbox_label": 0,
              "ignore": False
            }
            // ...
          ]
      }
    ]
}
```

The bounding box format is as follows: `[min_x, min_y, max_x, max_y]`

Migration Script

We provide a migration script to help users migrate old annotation files to the new format.

```
python tools/dataset_converters/textdet/data_migrator.py ${IN_PATH} ${OUT_PATH}
```

39.2.2 Text Recognition

Introduction of the New Format

The `TextRecogDataset` holds the information required by the text detection task, such as text and image path. We refer users to `tests/data/rec_toy_dataset/labels.json` which is an example annotation for `TextRecogDataset`.

```
{
  "metainfo":
    {
      "dataset_type": "TextRecogDataset",
      "task_name": "textrecog",
    },
    "data_list":
    [
      {
        "img_path": "test_img.jpg",
        "instances":
            [
              {
                "text": "GRAND"
              }
            ]
      }
    ]
}
```

Migration Script

We provide a migration script to help users migrate old annotation files to the new format.

```
python tools/dataset_converters/textrecog/data_migrator.py ${IN_PATH} ${OUT_PATH} --
↪format ${txt, jsonl, lmdb}
```

39.3 Compatibility

In consideration of the cost to users for data migration, we have temporarily made MMOCR version 1.x compatible with the old MMOCR 0.x format.

Note: The code and components used for compatibility with the old data format may be completely removed in a future release. Therefore, we strongly recommend that users migrate their datasets to the new data format.

Specifically, we provide three dataset classes *IcdarDataset*, *RecogTextDataset*, *RecogLMDBDataset* to support the old formats.

1. *IcdarDataset* supports COCO-like format annotations for text detection. You just need to add a new dataset config to `configs/textdet/_base_/datasets` and specify its dataset type as `IcdarDataset`.

```
data_root = 'data/det/icdar2015'
train_anno_path = 'instances_training.json'

train_dataset = dict(
    type='IcdarDataset',
    data_root=data_root,
    ann_file=train_anno_path,
    data_prefix=dict(img_path='imgs/'),
    filter_cfg=dict(filter_empty_gt=True, min_size=32),
    pipeline=None)
```

2. *RecogTextDataset* supports `.txt` and `.jsonl` format annotations for text recognition. You just need to add a new dataset config to `configs/textrecog/_base_/datasets` and specify its dataset type as `RecogTextDataset`. For example, the following example shows how to configure and load the 0.x format labels `old_label.txt` and `old_label.jsonl` from the toy dataset.

```
data_root = 'tests/data/rec_toy_dataset/'

# loading 0.x txt format annos
txt_dataset = dict(
    type='RecogTextDataset',
    data_root=data_root,
    ann_file='old_label.txt',
    data_prefix=dict(img_path='imgs'),
    parser_cfg=dict(
        type='LineStrParser',
        keys=['filename', 'text'],
        keys_idx=[0, 1]),
    pipeline=[])

# loading 0.x json line format annos
jsonl_dataset = dict(
    type='RecogTextDataset',
    data_root=data_root,
    ann_file='old_label.jsonl',
    data_prefix=dict(img_path='imgs'),
    parser_cfg=dict(
        type='LineJsonParser',
        keys=['filename', 'text'],
    pipeline=[]))
```

3. *RecogLMDBDataset* supports LMDB format dataset (img+labels) for text recognition. You just need to add a new dataset config to `configs/textrecog/_base_/datasets` and specify its dataset type as `RecogLMDBDataset`. For example, the following example shows how to configure and load the **both labels and images** `imgs.lmdb` from the toy dataset.

- set the dataset type to `RecogLMDBDataset`

```
# Specify the dataset type as RecogLMDBDataset
data_root = 'tests/data/rec_toy_dataset/'

lmdb_dataset = dict(
    type='RecogLMDBDataset',
    data_root=data_root,
    ann_file='imgs.lmdb',
    pipeline=None)
```

- replace the *LoadImageFromFile* with `LoadImageFromNDArray` in the data pipelines in `train_pipeline` and `test_pipeline`., for example

```
train_pipeline = [dict(type='LoadImageFromNDArray')]
```

CHAPTER

FORTY

PRETRAINED MODEL MIGRATION

Due to the extensive refactoring and fixing of the model structure in the new version, MMOCR 1.x does not support load weights trained by the old version. We have updated the pre-training weights and logs of all models on our website.

In addition, we are working on the development of a weight migration tool for text detection tasks and plan to release it in the near future. Since the text recognition and key information extraction models are too much modified and the migration is lossy, we do not plan to support them accordingly for the time being. If you have specific requirements, please feel free to raise an Issue.

CHAPTER

FORTYONE

DATA TRANSFORM MIGRATION

41.1 Introduction

In MMOCR version 0.x, we implemented a series of **Data Transform** methods in `mmocr/datasets/pipelines/xxx_transforms.py`. However, these modules are scattered all over the place and lack a standardized design. Therefore, we refactored all the data transform modules in MMOCR version 1.x. According to the task type, they are now defined in `ocr_transforms.py`, `textdet_transforms.py`, and `textrecog_transforms.py`, respectively, under `mmocr/datasets/transforms`. Specifically, `ocr_transforms.py` implements the data augmentation methods for OCR-related tasks in general, while `textdet_transforms.py` and `textrecog_transforms.py` implement data augmentation transforms related to text detection and text recognition tasks, respectively.

Since some of the modules were renamed, merged or separated during the refactoring process, the new interface and default parameters may be inconsistent with the old version. Therefore, this migration guide will introduce how to configure the new data transforms to achieve the identical behavior as the old version.

41.2 Configuration Migration Guide

41.2.1 Data Formatting Related Data Transforms

1. `Collect` + `CustomFormatBundle` -> *PackTextDetInputs*/*PackTextRecogInputs*

`PackxxxInputs` implements both `Collect` and `CustomFormatBundle` functions, and no longer has `key` parameters, the generation of training targets is moved to be done in `loss` modules.

```
dict(
    type='CustomFormatBundle',
    keys=['gt_shrink', 'gt_shrink_mask', 'gt_thr', 'gt_thr_mask'],
    meta_keys=['img_path', 'ori_shape', 'img_shape'],
    visualize=dict(flag=False, boundary_key='gt_shrink')),
dict(
    type='Collect',
    keys=['img', 'gt_shrink', 'gt_shrink_mask', 'gt_thr', 'gt_thr_mask'])
```

```
dict(
  type='PackTextDetInputs',
  meta_keys=('img_path', 'ori_shape', 'img_shape'))
```

41.2.2 Data Augmentation Related Data Transforms

1. `ResizeOCR` -> *`Resize`*, *`RescaleToHeight`*, *`PadToWidth`*

   The original `ResizeOCR` is now split into three data augmentation modules.

   When `keep_aspect_ratio=False`, it is equivalent to `Resize` in version 1.x. Its configuration can be modified as follows.

```
dict(
  type='ResizeOCR',
  height=32,
  min_width=100,
  max_width=100,
  keep_aspect_ratio=False)
```

```
dict(
  type='Resize',
  scale=(100, 32),
  keep_ratio=False)
```

When `keep_aspect_ratio=True` and `max_width=None`. The image will be rescaled to a fixed size alongside the height while keeping the aspect ratio the same as the origin.

```
dict(
  type='ResizeOCR',
  height=32,
  min_width=32,
  max_width=None,
  width_downsample_ratio = 1.0 / 16
  keep_aspect_ratio=True)
```

```
dict(
  type='RescaleToHeight',
  height=32,
  min_width=32,
  max_width=None,
  width_divisor=16),
```

When `keep_aspect_ratio=True` and `max_width` is a fixed value. The image will be rescaled to a fixed size alongside the height while keeping the aspect ratio the same as the origin. Then, the width will be padded or cropped to `max_width`. That is to say, the shape of the output image is always `(height, max_width)`.

```
dict(
  type='ResizeOCR',
  height=32,
  min_width=32,
  max_width=100,
  width_downsample_ratio = 1.0 / 16,
  keep_aspect_ratio=True)
```

```
dict(
  type='RescaleToHeight',
  height=32,
```

(continues on next page)

(continued from previous page)

```
  min_width=32,
  max_width=100,
  width_divisor=16),
dict(
  type='PadToWidth',
  width=100)
```

2. `RandomRotateTextDet` & `RandomRotatePolyInstances` -> *RandomRotate*

   We implemented all random rotation-related data augmentation in `RandomRotate` in version 1.x. Its default behavior is identical to the `RandomRotateTextDet` in version 0.x.

Note: The default value of "max_angle" might be different from the old version, so the users are suggested to manually set the number.

```
dict(type='RandomRotateTextDet')
```

```
dict(type='RandomRotate', max_angle=10)
```

For `RandomRotatePolyInstances`it is supposed to set `use_canvas=True`

```
dict(
  type='RandomRotatePolyInstances',
  rotate_ratio=0.5, # Specify the execution probability
  max_angle=60,
  pad_with_fixed_color=False)
```

```
# Wrap the data transforms with RandomApply and specify the execution probability
dict(
  type='RandomApply',
  transforms=[
    dict(type='RandomRotate',
    max_angle=60,
    pad_with_fixed_color=False,
    use_canvas=True)],
  prob=0.5) # Specify the execution probability
```

Note: In version 0.x, some data augmentation methods specified execution probability by defining an internal variable "xxx_ratio", such as "rotate_ratio", "crop_ratio", etc. In version 1.x, these parameters have been removed. Now we can use "RandomApply" to wrap different data transforms and specify their execution probabilities.

---

3. `RandomCropFlip` -> *TextDetRandomCropFlip*

   Currently, only the method name has been changed, and other parameters remain the same.

4. `RandomCropPolyInstances` -> *RandomCrop*

   In MMOCR version 1.x, `crop_ratio` and `instance_key` are removed. The `gt_polygons` is now used as the target for cropping.

```
dict(
  type='RandomCropPolyInstances',
  instance_key='gt_masks',
  crop_ratio=0.8, # Specify the execution probability
  min_side_ratio=0.3)
```

```
# Wrap the data transforms with RandomApply and specify the execution probability
dict(
  type='RandomApply',
  transforms=[dict(type='RandomCrop', min_side_ratio=0.3)],
  prob=0.8) # Specify the execution probability
```

5. RandomCropInstances -> *TextDetRandomCrop*

   In MMOCR version 1.x, `crop_ratio` and `instance_key` are removed. The `gt_polygons` is now used as the target for cropping.

```
dict(
  type='RandomCropInstances',
  target_size=(800800),
  instance_key='gt_kernels')
```

```
dict(
  type='TextDetRandomCrop',
  target_size=(800800))
```

6. EastRandomCrop -> *RandomCrop* + *Resize* + `mmengine.Pad`

   `EastRandomCrop` was implemented by applying cropping, scaling and padding to the input image. Now, the same effect can be achieved by combining three data transforms.

```
dict(
  type='EastRandomCrop',
  max_tries=10,
  min_crop_side_ratio=0.1,
  target_size=(640, 640))
```

```
dict(type='RandomCrop', min_side_ratio=0.1),
dict(type='Resize', scale=(640,640), keep_ratio=True),
dict(type='Pad', size=(640,640))
```

7. RandomScaling -> `mmengine.RandomResize`

   The `RandomScaling` is now replaced with `mmengine.RandomResize`.

```
 dict(
  type='RandomScaling',
  size=800,
  scale=(0.75, 2.5))
```

```
dict(
  type='RandomResize',
  scale=(800, 800),
```

(continues on next page)

(continued from previous page)

```
  ratio_range=(0.75, 2.5),
  keep_ratio=True)
```

Note: By default, the data pipeline will search for the corresponding data transforms from the register of the current *scope*, and if that data transform does not exist, it will continue to search in the upstream library, such as MMCV and MMEngine. For example, the `RandomResize` transform is not implemented in MMOCR, but it can be directly called in the configuration, as the program will automatically search for it from MMCV. In addition, you can also specify *scope* by adding a prefix. For example, `mmengine.RandomResize` will force it to use `RandomResize` implemented in MMEngine, which is useful when a method of the same name exists in both upstream and downstream libraries. It is noteworthy that all of the data transforms implemented in MMCV are registered to MMEngine, that is why we use `mmengine.RandomResize` but not `mmcv.RandomResize`.

8. `SquareResizePad` -> *`Resize` + `SourceImagePad`*

   `SquareResizePad` implements two branches and uses one of them randomly based on the `pad_ratio`. Specifically, one branch first resizes the image and then pads it to a certain size; while the other branch only resizes the image. To enhance the reusability of the different modules, we split this data transform into a combination of `Resize` + `SourceImagePad` in version 1.x, and control the branches via `RandomChoice`.

```
dict(
  type='SquareResizePad',
  target_size=800,
  pad_ratio=0.6)
```

```
dict(
  type='RandomChoice',
  transforms=[
    [
      dict(
        type='Resize',
        scale=800,
        keep_ratio=True),
      dict(
        type='SourceImagePad',
        target_scale=800)
    ],
    [
      dict(
        type='Resize',
        scale=800,
        keep_ratio=False)
    ]
  ],
  prob=[0.4, 0.6]), # Probability of selection of two combinations
```

Note: In version 1.x, the random choice wrapper "RandomChoice" replaces "OneOfWrapper", allowing random selection of data transform combinations.

9. `RandomWrapper` -> `mmengine.RandomApply`

   In version 1.x, the `RandomWrapper` wrapper has been replaced with `RandomApply` in MMEngine, which is used

to specify the probability of performing a data transform. And the probability `p` is now named `prob`.

```
dict(
 type='RandomWrapper',
 p=0.25,
 transforms=[
     dict(type='PyramidRescale'),
 ])
```

```
dict(
  type='RandomApply',
  prob=0.25,
  transforms=[
    dict(type='PyramidRescale'),
  ])
```

10. `OneOfWrapper` -> `mmengine.RandomChoice`

    The random choice wrapper is now renamed to `RandomChoice` and is used in exactly the same way as before.

11. `ScaleAspectJitter` -> *`ShortScaleAspectJitter`*, *`BoundedScaleAspectJitter`*

    The `ScaleAspectJitter` implemented several different image size jittering strategies, which has now been split into several independent data transforms.

    When `resize_type='indep_sample_in_range'`, it is equivalent to `RandomResize`.

```
dict(
  type='ScaleAspectJitter',
  img_scale=None,
  keep_ratio=False,
  resize_type='indep_sample_in_range',
  scale_range=(640, 2560))
```

```
 dict(
  type='RandomResize',
  scale=(640, 640),
  ratio_range=(1.0, 4.125),
  resize_type='Resize',
  keep_ratio=True))
```

When `resize_type='long_short_bound'`, we implemented `BoundedScaleAspectJitter`, which randomly rescales the image so that the long and short sides of the image are around the bound; then jitters the aspect ratio.

```
dict(
  type='ScaleAspectJitter',
  img_scale=[(3000, 736)],  # Unused
  ratio_range=(0.7, 1.3),
  aspect_ratio_range=(0.9, 1.1),
  multiscale_mode='value',
  long_size_bound=800,
  short_size_bound=480,
  resize_type='long_short_bound',
  keep_ratio=False)
```

```
dict(
  type='BoundedScaleAspectJitter',
  long_size_bound=800,
  short_size_bound=480,
  ratio_range=(0.7, 1.3),
  aspect_ratio_range=(0.9, 1.1))
```

When `resize_type='round_min_img_scale'`, we implemented `ShortScaleAspectJitter`, which rescales the image for its shorter side to reach the `short_size` and then jitters its aspect ratio, finally rescales the shape guaranteed to be divided by scale_divisor.

```
dict(
  type='ScaleAspectJitter',
  img_scale=[(3000, 640)],
  ratio_range=(0.7, 1.3),
  aspect_ratio_range=(0.9, 1.1),
  multiscale_mode='value',
  keep_ratio=False)
```

```
dict(
  type='ShortScaleAspectJitter',
  short_size=640,
  ratio_range=(0.7, 1.3),
  aspect_ratio_range=(0.9, 1.1),
  scale_divisor=32),
```

CHAPTER

FORTYTWO

MMOCR.APIS

mmocr.apis

- *Inferencers*

42.1 Inferencers

MMOCRInferencer	MMOCR Inferencer.
TextDetInferencer	Text Detection inferencer.
TextRecInferencer	Text Recognition inferencer.
TextSpotInferencer	Text Spotting inferencer.
KIEInferencer	Key Information Extraction Inferencer.

42.1.1 MMOCRInferencer

class mmocr.apis.inferencers.**MMOCRInferencer**(*det=None*, *det_weights=None*, *rec=None*, *rec_weights=None*, *kie=None*, *kie_weights=None*, *device=None*)

MMOCR Inferencer. It's a wrapper around three base task inferenecers: TextDetInferencer, TextRecInferencer and KIEInferencer, and it can be used to perform end-to-end OCR or KIE inference.

Parameters

- **det** (*Optional[Union[ConfigType, str]]*) – Pretrained text detection algorithm. It's the path to the config file or the model name defined in metafile. Defaults to None.
- **det_weights** (*Optional[str]*) – Path to the custom checkpoint file of the selected det model. If it is not specified and "det" is a model name of metafile, the weights will be loaded from metafile. Defaults to None.
- **rec** (*Optional[Union[ConfigType, str]]*) – Pretrained text recognition algorithm. It's the path to the config file or the model name defined in metafile. Defaults to None.
- **rec_weights** (*Optional[str]*) – Path to the custom checkpoint file of the selected rec model. If it is not specified and "rec" is a model name of metafile, the weights will be loaded from metafile. Defaults to None.
- **kie** (*Optional[Union[ConfigType, str]]*) – Pretrained key information extraction algorithm. It's the path to the config file or the model name defined in metafile. Defaults to

None.

- **kie_weights** (*Optional[str]*) – Path to the custom checkpoint file of the selected kie model. If it is not specified and "kie" is a model name of metafile, the weights will be loaded from metafile. Defaults to None.
- **device** (*Optional[str]*) – Device to run inference. If None, the available device will be automatically used. Defaults to None.

Return type None

forward(*inputs*, *batch_size=1*, *det_batch_size=None*, *rec_batch_size=None*, *kie_batch_size=None*, ***forward_kwargs*)

Forward the inputs to the model.

Parameters

- **inputs** (*InputsType*) – The inputs to be forwarded.
- **batch_size** (*int*) – Batch size. Defaults to 1.
- **det_batch_size** (*Optional[int]*) – Batch size for text detection model. Overwrite batch_size if it is not None. Defaults to None.
- **rec_batch_size** (*Optional[int]*) – Batch size for text recognition model. Overwrite batch_size if it is not None. Defaults to None.
- **kie_batch_size** (*Optional[int]*) – Batch size for KIE model. Overwrite batch_size if it is not None. Defaults to None.

Returns The prediction results. Possibly with keys "det", "rec", and "kie"..

Return type Dict

postprocess(*preds*, *visualization=None*, *print_result=False*, *save_pred=False*, *pred_out_dir=''*)

Process the predictions and visualization results from `forward` and `visualize`.

This method should be responsible for the following tasks:

1. Convert datasamples into a json-serializable dict if needed.
2. Pack the predictions and visualization results and return them.
3. Dump or log the predictions.

Parameters

- **preds** (*PredType*) – Predictions of the model.
- **visualization** (*Optional[np.ndarray]*) – Visualized predictions.
- **print_result** (*bool*) – Whether to print the result. Defaults to False.
- **save_pred** (*bool*) – Whether to save the inference result. Defaults to False.
- **pred_out_dir** (*str*) – File to save the inference results w/o visualization. If left as empty, no file will be saved. Defaults to ''.

Returns

Inference and visualization results, mapped from "predictions" and "visualization".

Return type Dict

visualize(*inputs*, *preds*, ***kwargs*)

Visualize predictions.

Parameters

- **inputs** (*List[Union[str, np.ndarray]]*) – Inputs for the inferencer.
- **preds** (*List[Dict]*) – Predictions of the model.
- **show** (*bool*) – Whether to display the image in a popup window. Defaults to False.
- **wait_time** (*float*) – The interval of show (s). Defaults to 0.
- **draw_pred** (*bool*) – Whether to draw predicted bounding boxes. Defaults to True.
- **pred_score_thr** (*float*) – Minimum score of bboxes to draw. Defaults to 0.3.
- **save_vis** (*bool*) – Whether to save the visualization result. Defaults to False.
- **img_out_dir** (*str*) – Output directory of visualization results. If left as empty, no file will be saved. Defaults to ‘’.

Returns Returns visualization results only if applicable.

Return type List[np.ndarray] or None

42.1.2 TextDetInferencer

class mmocr.apis.inferencers.**TextDetInferencer**(*model=None*, *weights=None*, *device=None*, *scope='mmocr'*)

Text Detection inferencer.

Parameters

- **model** (*str, optional*) – Path to the config file or the model name defined in metafile. For example, it could be “dbnet_resnet18_fpnc_1200e_icdar2015” or “configs/textdet/dbnet/dbnet_resnet18_fpnc_1200e_icdar2015.py”. If model is not specified, user must provide the *weights* saved by MMEngine which contains the config string. Defaults to None.
- **weights** (*str, optional*) – Path to the checkpoint. If it is not specified and model is a model name of metafile, the weights will be loaded from metafile. Defaults to None.
- **device** (*str, optional*) – Device to run inference. If None, the available device will be automatically used. Defaults to None.
- **scope** (*str, optional*) – The scope of the model. Defaults to “mmocr”.

Return type None

pred2dict(*data_sample*)

Extract elements necessary to represent a prediction into a dictionary. It’s better to contain only basic data elements such as strings and numbers in order to guarantee it’s json-serializable.

Parameters **data_sample** (TextDetDataSample) – The data sample to be converted.

Returns The output dictionary.

Return type dict

42.1.3 TextRecInferencer

class mmocr.apis.inferencers.**TextRecInferencer**(*model=None*, *weights=None*, *device=None*, *scope='mmocr'*)

Text Recognition inferencer.

Parameters

- **model** (*str, optional*) – Path to the config file or the model name defined in metafile. For example, it could be "crnn_mini-vgg_5e_mj" or "configs/textrecog/crnn/crnn_mini-vgg_5e_mj.py". If model is not specified, user must provide the *weights* saved by MMEngine which contains the config string. Defaults to None.
- **weights** (*str, optional*) – Path to the checkpoint. If it is not specified and model is a model name of metafile, the weights will be loaded from metafile. Defaults to None.
- **device** (*str, optional*) – Device to run inference. If None, the available device will be automatically used. Defaults to None.
- **scope** (*str, optional*) – The scope of the model. Defaults to "mmocr".

Return type None

pred2dict(*data_sample*)

Extract elements necessary to represent a prediction into a dictionary. It's better to contain only basic data elements such as strings and numbers in order to guarantee it's json-serializable.

Parameters **data_sample** (TextRecogDataSample) – The data sample to be converted.

Returns The output dictionary.

Return type dict

42.1.4 TextSpotInferencer

class mmocr.apis.inferencers.**TextSpotInferencer**(*model=None*, *weights=None*, *device=None*, *scope='mmocr'*)

Text Spotting inferencer.

Parameters

- **model** (*str, optional*) – Path to the config file or the model name defined in metafile. For example, it could be "dbnet_resnet18_fpnc_1200e_icdar2015" or "configs/textdet/dbnet/dbnet_resnet18_fpnc_1200e_icdar2015.py". If model is not specified, user must provide the *weights* saved by MMEngine which contains the config string. Defaults to None.
- **weights** (*str, optional*) – Path to the checkpoint. If it is not specified and model is a model name of metafile, the weights will be loaded from metafile. Defaults to None.
- **device** (*str, optional*) – Device to run inference. If None, the available device will be automatically used. Defaults to None.
- **scope** (*str, optional*) – The scope of the model. Defaults to "mmocr".

Return type None

pred2dict(*data_sample*)

Extract elements necessary to represent a prediction into a dictionary. It's better to contain only basic data elements such as strings and numbers in order to guarantee it's json-serializable.

Parameters **data_sample** (*TextSpottingDataSample*) – The data sample to be converted.

Returns The output dictionary.

Return type dict

42.1.5 KIEInferencer

class mmocr.apis.inferencers.**KIEInferencer**(*model=None*, *weights=None*, *device=None*, *scope='mmocr'*)

Key Information Extraction Inferencer.

Parameters

- **model** (*str, optional*) – Path to the config file or the model name defined in metafile. For example, it could be "sdmgr_unet16_60e_wildreceipt" or "configs/kie/sdmgr/sdmgr_unet16_60e_wildreceipt.py". If model is not specified, user must provide the *weights* saved by MMEngine which contains the config string. Defaults to None.
- **weights** (*str, optional*) – Path to the checkpoint. If it is not specified and model is a model name of metafile, the weights will be loaded from metafile. Defaults to None.
- **device** (*str, optional*) – Device to run inference. If None, the available device will be automatically used. Defaults to None.
- **scope** (*str, optional*) – The scope of the model. Defaults to "mmocr".

Return type None

static kie_collate(*data_batch*)

A collate function designed for KIE, where the first element (input) is a dict and we only want to keep it as-is instead of batching elements inside.

Returns Transversed Data in the same format as the data_itement of `data_batch`.

Return type Any

Parameters **data_batch** (*Sequence*) –

pred2dict(*data_sample*)

Extract elements necessary to represent a prediction into a dictionary. It's better to contain only basic data elements such as strings and numbers in order to guarantee it's json-serializable.

Parameters **data_sample** (TextRecogDataSample) – The data sample to be converted.

Returns The output dictionary.

Return type dict

visualize(*inputs*, *preds*, *return_vis=False*, *show=False*, *wait_time=0*, *draw_pred=True*, *pred_score_thr=0.3*, *save_vis=False*, *img_out_dir=''*)

Visualize predictions.

Parameters

- **inputs** (*List[Union[str, np.ndarray]]*) – Inputs for the inferencer.
- **preds** (*List[Dict]*) – Predictions of the model.
- **return_vis** (*bool*) – Whether to return the visualization result. Defaults to False.
- **show** (*bool*) – Whether to display the image in a popup window. Defaults to False.
- **wait_time** (*float*) – The interval of show (s). Defaults to 0.
- **draw_pred** (*bool*) – Whether to draw predicted bounding boxes. Defaults to True.

- **pred_score_thr** (*float*) – Minimum score of bboxes to draw. Defaults to 0.3.
- **save_vis** (*bool*) – Whether to save the visualization result. Defaults to False.
- **img_out_dir** (*str*) – Output directory of visualization results. If left as empty, no file will be saved. Defaults to ‘’.

Returns Returns visualization results only if applicable.

Return type List[np.ndarray] or None

CHAPTER

FORTYTHREE

MMOCR.STRUCTURES

TextDetDataSample	A data structure interface of MMOCR.
TextRecogDataSample	A data structure interface of MMOCR for text recognition.
KIEDataSample	A data structure interface of MMOCR.

43.1 TextDetDataSample

class mmocr.structures.**TextDetDataSample**(*, *metainfo=None*, ***kwargs*)

A data structure interface of MMOCR. They are used as interfaces between different components.

The attributes in `TextDetDataSample` are divided into two parts:

- ``gt_instances``(InstanceData): Ground truth of instance annotations.
- ``pred_instances``(InstanceData): Instances of model predictions.

Examples

```
>>> import torch
>>> import numpy as np
>>> from mmengine.structures import InstanceData
>>> from mmocr.data import TextDetDataSample
>>> # gt_instances
>>> data_sample = TextDetDataSample()
>>> img_meta = dict(img_shape=(800, 1196, 3),
...                 pad_shape=(800, 1216, 3))
>>> gt_instances = InstanceData(metainfo=img_meta)
>>> gt_instances.bboxes = torch.rand((5, 4))
>>> gt_instances.labels = torch.rand((5,))
>>> data_sample.gt_instances = gt_instances
>>> assert 'img_shape' in data_sample.gt_instances.metainfo_keys()
>>> len(data_sample.gt_instances)
5
>>> print(data_sample)
<TextDetDataSample(
    META INFORMATION
    DATA FIELDS
    gt_instances: <InstanceData(
```

(continues on next page)

(continued from previous page)

```
            META INFORMATION
            pad_shape: (800, 1216, 3)
            img_shape: (800, 1196, 3)
            DATA FIELDS
            labels: tensor([0.8533, 0.1550, 0.5433, 0.7294, 0.5098])
            bboxes:
            tensor([[9.7725e-01, 5.8417e-01, 1.7269e-01, 6.5694e-01],
                    [1.7894e-01, 5.1780e-01, 7.0590e-01, 4.8589e-01],
                    [7.0392e-01, 6.6770e-01, 1.7520e-01, 1.4267e-01],
                    [2.2411e-01, 5.1962e-01, 9.6953e-01, 6.6994e-01],
                    [4.1338e-01, 2.1165e-01, 2.7239e-04, 6.8477e-01]])
        ) at 0x7f21fb1b9190>
) at 0x7f21fb1b9880>
>>> # pred_instances
>>> pred_instances = InstanceData(metainfo=img_meta)
>>> pred_instances.bboxes = torch.rand((5, 4))
>>> pred_instances.scores = torch.rand((5,))
>>> data_sample = TextDetDataSample(pred_instances=pred_instances)
>>> assert 'pred_instances' in data_sample
>>> data_sample = TextDetDataSample()
>>> gt_instances_data = dict(
...                         bboxes=torch.rand(2, 4),
...                         labels=torch.rand(2),
...                         masks=np.random.rand(2, 2, 2))
>>> gt_instances = InstanceData(**gt_instances_data)
>>> data_sample.gt_instances = gt_instances
>>> assert 'gt_instances' in data_sample
>>> assert 'masks' in data_sample.gt_instances
```

Parameters **metainfo** (*Optional[dict]*) –

Return type None

property gt_instances: mmengine.structures.instance_data.InstanceData
groundtruth instances.

Type InstanceData

property pred_instances: mmengine.structures.instance_data.InstanceData
prediction instances.

Type InstanceData

43.2 TextRecogDataSample

class mmocr.structures.**TextRecogDataSample**(*, *metainfo=None*, ***kwargs*)
A data structure interface of MMOCR for text recognition. They are used as interfaces between different components.

The attributes in `TextRecogDataSample` are divided into two parts:

- ``gt_text``(LabelData): Ground truth text.
- ``pred_text``(LabelData): predictions text.

Examples

```
>>> import torch
>>> import numpy as np
>>> from mmengine.structures import LabelData
>>> from mmocr.data import TextRecogDataSample
>>> # gt_text
>>> data_sample = TextRecogDataSample()
>>> img_meta = dict(img_shape=(800, 1196, 3),
...                 pad_shape=(800, 1216, 3))
>>> gt_text = LabelData(metainfo=img_meta)
>>> gt_text.item = 'mmocr'
>>> data_sample.gt_text = gt_text
>>> assert 'img_shape' in data_sample.gt_text.metainfo_keys()
>>> print(data_sample)
<TextRecogDataSample(
    META INFORMATION
    DATA FIELDS
    gt_text: <LabelData(
            META INFORMATION
            pad_shape: (800, 1216, 3)
            img_shape: (800, 1196, 3)
            DATA FIELDS
            item: 'mmocr'
        ) at 0x7f21fb1b9190>
) at 0x7f21fb1b9880>
>>> # pred_text
>>> pred_text = LabelData(metainfo=img_meta)
>>> pred_text.item = 'mmocr'
>>> data_sample = TextRecogDataSample(pred_text=pred_text)
>>> assert 'pred_text' in data_sample
>>> data_sample = TextRecogDataSample()
>>> gt_text_data = dict(item='mmocr')
>>> gt_text = LabelData(**gt_text_data)
>>> data_sample.gt_text = gt_text
>>> assert 'gt_text' in data_sample
>>> assert 'item' in data_sample.gt_text
```

Parameters **metainfo** (*Optional[dict]*) –

Return type None

property gt_text: mmengine.structures.label_data.LabelData

ground truth text.

Type LabelData

property pred_text: mmengine.structures.label_data.LabelData

prediction text.

Type LabelData

43.3 KIEDataSample

class mmocr.structures.**KIEDataSample**(*, *metainfo=None*, ***kwargs*)

A data structure interface of MMOCR. They are used as interfaces between different components.

The attributes in KIEDataSample are divided into two parts:

- ``gt_instances``(InstanceData): Ground truth of instance annotations.
- ``pred_instances``(InstanceData): Instances of model predictions.

Examples

```
>>> import torch
>>> import numpy as np
>>> from mmengine.structures import InstanceData
>>> from mmocr.data import KIEDataSample
>>> # gt_instances
>>> data_sample = KIEDataSample()
>>> img_meta = dict(img_shape=(800, 1196, 3),
...                 pad_shape=(800, 1216, 3))
>>> gt_instances = InstanceData(metainfo=img_meta)
>>> gt_instances.bboxes = torch.rand((5, 4))
>>> gt_instances.labels = torch.rand((5,))
>>> data_sample.gt_instances = gt_instances
>>> assert 'img_shape' in data_sample.gt_instances.metainfo_keys()
>>> len(data_sample.gt_instances)
5
>>> print(data_sample)
<KIEDataSample(
    META INFORMATION
    DATA FIELDS
    gt_instances: <InstanceData(
            META INFORMATION
            pad_shape: (800, 1216, 3)
            img_shape: (800, 1196, 3)
            DATA FIELDS
            labels: tensor([0.8533, 0.1550, 0.5433, 0.7294, 0.5098])
            bboxes:
            tensor([[9.7725e-01, 5.8417e-01, 1.7269e-01, 6.5694e-01],
                    [1.7894e-01, 5.1780e-01, 7.0590e-01, 4.8589e-01],
                    [7.0392e-01, 6.6770e-01, 1.7520e-01, 1.4267e-01],
                    [2.2411e-01, 5.1962e-01, 9.6953e-01, 6.6994e-01],
                    [4.1338e-01, 2.1165e-01, 2.7239e-04, 6.8477e-01]])
        ) at 0x7f21fb1b9190>
) at 0x7f21fb1b9880>
>>> # pred_instances
>>> pred_instances = InstanceData(metainfo=img_meta)
>>> pred_instances.bboxes = torch.rand((5, 4))
>>> pred_instances.scores = torch.rand((5,))
>>> data_sample = KIEDataSample(pred_instances=pred_instances)
>>> assert 'pred_instances' in data_sample
>>> data_sample = KIEDataSample()
```

(continues on next page)

(continued from previous page)

```
>>> gt_instances_data = dict(
...                         bboxes=torch.rand(2, 4),
...                         labels=torch.rand(2))
>>> gt_instances = InstanceData(**gt_instances_data)
>>> data_sample.gt_instances = gt_instances
>>> assert 'gt_instances' in data_sample
```

Parameters **metainfo** (*Optional[dict]*) –

Return type None

property gt_instances: mmengine.structures.instance_data.InstanceData

groundtruth instances.

Type InstanceData

property pred_instances: mmengine.structures.instance_data.InstanceData

prediction instances.

Type InstanceData

CHAPTER

FORTYFOUR

MMOCR.DATASETS

mmocr.datasets

44.1 Samplers

BatchAugSampler	Sampler that repeats the same data elements for num_repeats times.

44.1.1 BatchAugSampler

class mmocr.datasets.samplers.**BatchAugSampler**(*dataset*, *shuffle=True*, *num_repeats=3*, *seed=None*)

Sampler that repeats the same data elements for num_repeats times. The batch size should be divisible by num_repeats.

It ensures that different each augmented version of a sample will be visible to a different process (GPU). Heavily based on torch.utils.data.DistributedSampler.

This sampler was modified from https://github.com/facebookresearch/deit/blob/0c4b8f60/samplers.py Used in Copyright (c) 2015-present, Facebook, Inc.

Parameters

- **dataset** (*Sized*) – The dataset.
- **shuffle** (*bool*) – Whether shuffle the dataset or not. Defaults to True.
- **num_repeats** (*int*) – The repeat times of every sample. Defaults to 3.
- **seed** (*int, optional*) – Random seed used to shuffle the sampler if `shuffle=True`. This number should be identical across all processes in the distributed group. Defaults to None.

set_epoch(*epoch*)

Sets the epoch for this sampler.

When `shuffle=True`, this ensures all replicas use a different random ordering for each epoch. Otherwise, the next iteration of this sampler will yield the same ordering.

Parameters **epoch** (*int*) – Epoch number.

Return type None

44.2 Datasets

OCRDataset	OCRDataset for text detection and text recognition.
WildReceiptDataset	WildReceipt Dataset for key information extraction.

44.2.1 OCRDataset

class mmocr.datasets.**OCRDataset**(*ann_file=''*, *metainfo=None*, *data_root=''*, *data_prefix={'img_path': ''}*, *filter_cfg=None*, *indices=None*, *serialize_data=True*, *pipeline=[]*, *test_mode=False*, *lazy_init=False*, *max_refetch=1000*)

OCRDataset for text detection and text recognition.

The annotation format is shown as follows.

```
{
    "metainfo":
    {
      "dataset_type": "test_dataset",
      "task_name": "test_task"
    },
    "data_list":
    [
      {
        "img_path": "test_img.jpg",
        "height": 604,
        "width": 640,
        "instances":
        [
          {
            "bbox": [0, 0, 10, 20],
            "bbox_label": 1,
            "mask": [0,0,0,10,10,20,20,0],
            "text": '123'
          },
          {
            "bbox": [10, 10, 110, 120],
            "bbox_label": 2,
            "mask": [10,10],10,110,110,120,120,10]],
            "extra_anns": '456'
          }
        ]
      },
    ]
}
```

Parameters

- **ann_file** (*str*) – Annotation file path. Defaults to ‘’.
- **metainfo** (*dict, optional*) – Meta information for dataset, such as class information. Defaults to None.
- **data_root** (*str, optional*) – The root directory for `data_prefix` and `ann_file`. Defaults to None.
- **data_prefix** (*dict*) – Prefix for training data. Defaults to dict(img_path=”).
- **filter_cfg** (*dict, optional*) – Config for filter data. Defaults to None.
- **indices** (*int or Sequence[int], optional*) – Support using first few data in annotation file to facilitate training/testing on a smaller dataset. Defaults to None which means using all `data_infos`.
- **serialize_data** (*bool, optional*) – Whether to hold memory using serialized objects, when enabled, data loader workers can use shared RAM from master process instead of making a copy. Defaults to True.
- **pipeline** (*list, optional*) – Processing pipeline. Defaults to [].
- **test_mode** (*bool, optional*) – `test_mode=True` means in test phase. Defaults to False.
- **lazy_init** (*bool, optional*) – Whether to load annotation during instantiation. In some cases, such as visualization, only the meta information of the dataset is needed, which is not necessary to load annotation file. `OCRdataset` can skip load annotations to save time by set `lazy_init=False`. Defaults to False.
- **max_refetch** (*int, optional*) – If `OCRdataset.prepare_data` get a None img. The maximum extra number of cycles to get a valid image. Defaults to 1000.

Note: OCRDataset collects meta information from *annotation file* (the lowest priority), ``OCRDataset.METAINFO``(medium) and *metainfo parameter* (highest) passed to constructors. The lower priority meta information will be overwritten by higher one.

Examples

Assume the annotation file is given above. >>> class CustomDataset(OCRDataset): >>> METAINFO: dict = dict(task_name=’custom_task’, >>> dataset_type=’custom_type’) >>> metainfo=dict(task_name=’custom_task_name’) >>> custom_dataset = CustomDataset(>>> ‘path/to/ann_file’, >>> metainfo=metainfo) >>> # meta information of annotation file will be overwritten by >>> # *CustomDataset.METAINFO*. The merged meta information will >>> # further be overwritten by argument *metainfo*. >>> custom_dataset.metainfo {‘task_name’: custom_task_name, dataset_type: custom_type}

44.2.2 WildReceiptDataset

class mmocr.datasets.**WildReceiptDataset**(*directed=False*, *ann_file=''*, *metainfo=None*, *data_root=''*, *data_prefix={'img_path': ''}*, *filter_cfg=None*, *indices=None*, *serialize_data=True*, *pipeline=Ellipsis*, *test_mode=False*, *lazy_init=False*, *max_refetch=1000*)

WildReceipt Dataset for key information extraction. There are two files to be loaded: metainfo and annotation. The metainfo file contains the mapping between classes and labels. The annotation file contains the all necessary information about the image, such as bounding boxes, texts, and labels etc.

The metainfo file is a text file with the following format:

```
0 Ignore
1 Store_name_value
2 Store_name_key
```

The annotation format is shown as follows.

```
{
    "file_name": "a.jpeg",
    "height": 348,
    "width": 348,
    "annotations": [
        {
            "box": [
                114.0,
                19.0,
                230.0,
                19.0,
                230.0,
                1.0,
                114.0,
                1.0
            ],
            "text": "CHOEUN",
            "label": 1
        },
        {
            "box": [
                97.0,
                35.0,
                236.0,
                35.0,
                236.0,
                19.0,
                97.0,
                19.0
            ],
            "text": "KOREANRESTAURANT",
            "label": 2
        }
    ]
}
```

Parameters

- **directed** (*bool*) – Whether to use directed graph. Defaults to False.
- **ann_file** (*str*) – Annotation file path. Defaults to ‘’.
- **metainfo** (*str or dict, optional*) – Meta information for dataset, such as class information. If it’s a string, it will be treated as a path to the class file from which the class information will be loaded. Defaults to None.
- **data_root** (*str, optional*) – The root directory for `data_prefix` and `ann_file`. Defaults to ‘’.
- **data_prefix** (*dict, optional*) – Prefix for training data. Defaults to dict(img_path=”).
- **filter_cfg** (*dict, optional*) – Config for filter data. Defaults to None.
- **indices** (*int or Sequence[int], optional*) – Support using first few data in annotation file to facilitate training/testing on a smaller dataset. Defaults to None which means using all `data_infos`.
- **serialize_data** (*bool, optional*) – Whether to hold memory using serialized objects, when enabled, data loader workers can use shared RAM from master process instead of making a copy. Defaults to True.
- **pipeline** (*list, optional*) – Processing pipeline. Defaults to [].
- **test_mode** (*bool, optional*) – `test_mode=True` means in test phase. Defaults to False.
- **lazy_init** (*bool, optional*) – Whether to load annotation during instantiation. In some cases, such as visualization, only the meta information of the dataset is needed, which is not necessary to load annotation file. `Basedataset` can skip load annotations to save time by set `lazy_init=False`. Defaults to False.
- **max_refetch** (*int, optional*) – If `Basedataset.prepare_data` get a None img. The maximum extra number of cycles to get a valid image. Defaults to 1000.

`load_data_list()`
Load data list from annotation file.

Returns A list of annotation dict.

Return type List[dict]

`parse_data_info(raw_data_info)`
Parse data info from raw data info.

Parameters **raw_data_info** (*dict*) – Raw data info.

Returns

Parsed data info.

- img_path (str): Path to the image.
- img_shape (tuple(int, int)): Image shape in (H, W).
- instances (list[dict]): A list of instances. - bbox (ndarray(dtype=np.float32)): Shape (4,). Bounding box. - text (str): Annotation text. - edge_label (int): Edge label. - bbox_label (int): Bounding box label.

Return type dict

44.3 Compatible Datasets

IcdarDataset	Dataset for text detection while ann_file in coco format.
RecogLMDBDataset	RecogLMDBDataset for text recognition.
RecogTextDataset	RecogTextDataset for text recognition.

44.3.1 IcdarDataset

class mmocr.datasets.**IcdarDataset**(**args*, *seg_map_suffix='.png'*, *proposal_file=None*, *file_client_args=None*, *backend_args=None*, *return_classes=False*, ***kwargs*)

Dataset for text detection while ann_file in coco format.

Parameters

- **ann_file** (*str*) – Annotation file path. Defaults to ‘’.
- **metainfo** (*dict, optional*) – Meta information for dataset, such as class information. Defaults to None.
- **data_root** (*str*) – The root directory for `data_prefix` and `ann_file`. Defaults to ‘’.
- **data_prefix** (*dict*) – Prefix for training data. Defaults to dict(img_path=”).
- **filter_cfg** (*dict, optional*) – Config for filter data. Defaults to None.
- **indices** (*int or Sequence[int], optional*) – Support using first few data in annotation file to facilitate training/testing on a smaller dataset. Defaults to None which means using all `data_infos`.
- **serialize_data** (*bool, optional*) – Whether to hold memory using serialized objects, when enabled, data loader workers can use shared RAM from master process instead of making a copy. Defaults to True.
- **pipeline** (*list, optional*) – Processing pipeline. Defaults to [].
- **test_mode** (*bool, optional*) – `test_mode=True` means in test phase. Defaults to False.
- **lazy_init** (*bool, optional*) – Whether to load annotation during instantiation. In some cases, such as visualization, only the meta information of the dataset is needed, which is not necessary to load annotation file. `Basedataset` can skip load annotations to save time by set `lazy_init=False`. Defaults to False.
- **max_refetch** (*int, optional*) – If `Basedataset.prepare_data` get a None img. The maximum extra number of cycles to get a valid image. Defaults to 1000.

parse_data_info(*raw_data_info*)

Parse raw annotation to target format.

Parameters **raw_data_info** (*dict*) – Raw data information loaded from `ann_file`

Returns Parsed annotation.

Return type Union[dict, List[dict]]

44.3.2 RecogLMDBDataset

class mmocr.datasets.**RecogLMDBDataset**(*ann_file=''*, *img_color_type='color'*, *metainfo=None*, *data_root=''*, *data_prefix={'img_path': ''}*, *filter_cfg=None*, *indices=None*, *serialize_data=True*, *pipeline=[]*, *test_mode=False*, *lazy_init=False*, *max_refetch=1000*)

RecogLMDBDataset for text recognition.

The annotation format should be in lmdb format. The lmdb file should contain three keys: 'num-samples', 'label-xxxxxxxxx' and 'image-xxxxxxxxx', where 'xxxxxxxxx' is the index of the image. The value of 'num-samples' is the total number of images. The value of 'label-xxxxxxx' is the text label of the image, and the value of 'image-xxxxxxx' is the image data.

following keys: Each item fetched from this dataset will be a dict containing the following keys:

- img (ndarray): The loaded image.
- img_path (str): The image key.
- instances (list[dict]): The list of annotations for the image.

Parameters

- **ann_file** (*str*) – Annotation file path. Defaults to ''.
- **img_color_type** (*str*) – The flag argument for :func:`mmcv.imfrombytes`, which determines how the image bytes will be parsed. Defaults to 'color'.
- **metainfo** (*dict, optional*) – Meta information for dataset, such as class information. Defaults to None.
- **data_root** (*str*) – The root directory for `data_prefix` and `ann_file`. Defaults to ''.
- **data_prefix** (*dict*) – Prefix for training data. Defaults to `dict(img_path='')`.
- **filter_cfg** (*dict, optional*) – Config for filter data. Defaults to None.
- **indices** (*int or Sequence[int], optional*) – Support using first few data in annotation file to facilitate training/testing on a smaller dataset. Defaults to None which means using all `data_infos`.
- **serialize_data** (*bool, optional*) – Whether to hold memory using serialized objects, when enabled, data loader workers can use shared RAM from master process instead of making a copy. Defaults to True.
- **pipeline** (*list, optional*) – Processing pipeline. Defaults to [].
- **test_mode** (*bool, optional*) – `test_mode=True` means in test phase. Defaults to False.
- **lazy_init** (*bool, optional*) – Whether to load annotation during instantiation. In some cases, such as visualization, only the meta information of the dataset is needed, which is not necessary to load annotation file. `RecogLMDBDataset` can skip load annotations to save time by set `lazy_init=False`. Defaults to False.
- **max_refetch** (*int, optional*) – If `RecogLMDBdataset.prepare_data` get a None img. The maximum extra number of cycles to get a valid image. Defaults to 1000.

close()
Close lmdb environment.

load_data_list()
Load annotations from an annotation file named as `self.ann_file`

Returns A list of annotation.

Return type List[dict]

`parse_data_info`(*raw_anno_info*)
Parse raw annotation to target format.

Parameters **raw_anno_info** (*str*) – One raw data information loaded from `ann_file`.

Returns Parsed annotation.

Return type (dict)

`prepare_data`(*idx*)
Get data processed by `self.pipeline`.

Parameters **idx** (*int*) – The index of `data_info`.

Returns Depends on `self.pipeline`.

Return type Any

44.3.3 RecogTextDataset

`class mmocr.datasets.RecogTextDataset`(*ann_file=''*, *backend_args=None*, *parser_cfg={'keys': ['filename', 'text'], 'type': 'LineJsonParser'}*, *metainfo=None*, *data_root=''*, *data_prefix={'img_path': ''}*, *filter_cfg=None*, *indices=None*, *serialize_data=True*, *pipeline=[]*, *test_mode=False*, *lazy_init=False*, *max_refetch=1000*)
RecogTextDataset for text recognition.

The annotation format can be both in jsonl and txt. If the annotation file is in jsonl format, it should be a list of dicts. If the annotation file is in txt format, it should be a list of lines.

The annotation formats are shown as follows. - txt format .. code-block:: none

`test_img1.jpg OpenMMLab test_img2.jpg MMOCR`

- jsonl format

```
``{"filename": "test_img1.jpg", "text": "OpenMMLab"}``
``{"filename": "test_img2.jpg", "text": "MMOCR"}``
```

Parameters

- **ann_file** (*str*) – Annotation file path. Defaults to ‘’.
- **backend_args** (*dict, optional*) – Arguments to instantiate the prefix of uri corresponding backend. Defaults to None.
- **parse_cfg** (*dict, optional*) – Config of parser for parsing annotations. Use `LineJsonParser` when the annotation file is in jsonl format with keys of `filename` and `text`. The keys in parse_cfg should be consistent with the keys in jsonl annotations. The first key in parse_cfg should be the key of the path in jsonl annotations. The second key in parse_cfg should be the key of the text in jsonl Use `LineStrParser` when the annotation file is in txt format. Defaults to `dict(type='LineJsonParser', keys=['filename', 'text'])`.
- **metainfo** (*dict, optional*) – Meta information for dataset, such as class information. Defaults to None.

- **data_root** (*str*) – The root directory for `data_prefix` and `ann_file`. Defaults to ''.
- **data_prefix** (*dict*) – Prefix for training data. Defaults to `dict(img_path='')`.
- **filter_cfg** (*dict, optional*) – Config for filter data. Defaults to None.
- **indices** (*int or Sequence[int], optional*) – Support using first few data in annotation file to facilitate training/testing on a smaller dataset. Defaults to None which means using all `data_infos`.
- **serialize_data** (*bool, optional*) – Whether to hold memory using serialized objects, when enabled, data loader workers can use shared RAM from master process instead of making a copy. Defaults to True.
- **pipeline** (*list, optional*) – Processing pipeline. Defaults to [].
- **test_mode** (*bool, optional*) – `test_mode=True` means in test phase. Defaults to False.
- **lazy_init** (*bool, optional*) – Whether to load annotation during instantiation. In some cases, such as visualization, only the meta information of the dataset is needed, which is not necessary to load annotation file. `RecogTextDataset` can skip load annotations to save time by set `lazy_init=False`. Defaults to False.
- **max_refetch** (*int, optional*) – If `RecogTextDataset.prepare_data` get a None img. The maximum extra number of cycles to get a valid image. Defaults to 1000.

load_data_list()
Load annotations from an annotation file named as `self.ann_file`

Returns A list of annotation.

Return type List[dict]

parse_data_info(*raw_anno_info*)
Parse raw annotation to target format.

Parameters **raw_anno_info** (*str*) – One raw data information loaded from `ann_file`.

Returns Parsed annotation.

Return type (dict)

44.4 Dataset Wrapper

ConcatDataset	A wrapper of concatenated dataset.

44.4.1 ConcatDataset

class mmocr.datasets.**ConcatDataset**(*datasets*, *pipeline=[]*, *verify_meta=True*, *force_apply=False*, *lazy_init=False*)
A wrapper of concatenated dataset.

Same as `torch.utils.data.dataset.ConcatDataset` and support lazy_init.

Note: `ConcatDataset` should not inherit from `BaseDataset` since `get_subset` and `get_subset_` could produce ambiguous meaning sub-dataset which conflicts with original dataset. If you want to use a sub-dataset

of `ConcatDataset`, you should set `indices` arguments for wrapped dataset which inherit from `BaseDataset`.

Parameters

- **datasets** (*Sequence[BaseDataset] or Sequence[dict]*) – A list of datasets which will be concatenated.
- **pipeline** (*list, optional*) – Processing pipeline to be applied to all of the concatenated datasets. Defaults to [].
- **verify_meta** (*bool*) – Whether to verify the consistency of meta information of the concatenated datasets. Defaults to True.
- **force_apply** (*bool*) – Whether to force apply pipeline to all datasets if any of them already has the pipeline configured. Defaults to False.
- **lazy_init** (*bool, optional*) – Whether to load annotation during instantiation. Defaults to False.

CHAPTER

FORTYFIVE

MMOCR.DATASETS

mmocr.datasets.transforms

45.1 Loading

LoadImageFromFile	Load an image from file.
LoadOCRAnnotations	Load and process the `instances` annotation provided by dataset.
LoadKIEAnnotations	Load and process the `instances` annotation provided by dataset.
InferencerLoader	Load the image in Inferencer's pipeline.

45.1.1 LoadImageFromFile

class mmocr.datasets.transforms.**LoadImageFromFile**(*to_float32=False*, *color_type='color'*, *imdecode_backend='cv2'*, *file_client_args=None*, *min_size=0*, *ignore_empty=False*, ***, *backend_args=None*)

Load an image from file.

Required Keys:

- img_path

Modified Keys:

- img

- img_shape
- ori_shape

Parameters

- **to_float32** (*bool*) – Whether to convert the loaded image to a float32 numpy array. If set to False, the loaded image is an uint8 array. Defaults to False.
- **color_type** (*str*) – The flag argument for :func:mmcv.imfrombytes. Defaults to 'color'.
- **imdecode_backend** (*str*) – The image decoding backend type. The backend argument for :func:mmcv.imfrombytes. See :func:mmcv.imfrombytes for details. Defaults to 'cv2'.
- **file_client_args** (*dict*) – Arguments to instantiate a FileClient. See mmengine.fileio.FileClient for details. Defaults to None. It will be deprecated in future. Please use backend_args instead. Deprecated in version 1.0.0rc6.
- **backend_args** (*dict, optional*) – Instantiates the corresponding file backend. It may contain *backend* key to specify the file backend. If it contains, the file backend corresponding to this value will be used and initialized with the remaining values, otherwise the corresponding file backend will be selected based on the prefix of the file path. Defaults to None. New in version 1.0.0rc6.
- **ignore_empty** (*bool*) – Whether to allow loading empty image or file path not existent. Defaults to False.
- **min_size** (*int*) – The minimum size of the image to be loaded. If the image is smaller than the minimum size, it will be regarded as a broken image. Defaults to 0.

Return type None

transform(*results*)

Functions to load image.

Parameters **results** (*dict*) – Result dict from :obj:mmcv.BaseDataset.

Returns The dict contains loaded image and meta information.

Return type dict

45.1.2 LoadOCRAnnotations

class mmocr.datasets.transforms.**LoadOCRAnnotations**(*with_bbox=False*, *with_label=False*, *with_polygon=False*, *with_text=False*, ***kwargs*)

Load and process the `instances` annotation provided by dataset.

The annotation format is as the following:

```
{
    'instances':
    [
        {
        # List of 4 numbers representing the bounding box of the
        # instance, in (x1, y1, x2, y2) order.
        # used in text detection or text spotting tasks.
        'bbox': [x1, y1, x2, y2],

        # Label of instance, usually it's 0.
```

(continues on next page)

(continued from previous page)

```
        # used in text detection or text spotting tasks.
        'bbox_label': 0,

        # List of n numbers representing the polygon of the
        # instance, in (xn, yn) order.
        # used in text detection/ textspotter.
        "polygon": [x1, y1, x2, y2, ... xn, yn],

        # The flag indicating whether the instance should be ignored.
        # used in text detection or text spotting tasks.
        "ignore": False,

        # The groundtruth of text.
        # used in text recognition or text spotting tasks.
        "text": 'tmp',
        }
    ]
}
```

After this module, the annotation has been changed to the format below:

```
{
    # In (x1, y1, x2, y2) order, float type. N is the number of bboxes
    # in np.float32
    'gt_bboxes': np.ndarray(N, 4)
     # In np.int64 type.
    'gt_bboxes_labels': np.ndarray(N, )
    # In (x1, y1,..., xk, yk) order, float type.
    # in list[np.float32]
    'gt_polygons': list[np.ndarray(2k, )]
     # In np.bool_ type.
    'gt_ignored': np.ndarray(N, )
     # In list[str]
    'gt_texts': list[str]
}
```

Required Keys:

- instances
  - bbox (optional)
  - bbox_label (optional)
  - polygon (optional)
  - ignore (optional)
  - text (optional)

Added Keys:

- gt_bboxes (np.float32)
- gt_bboxes_labels (np.int64)
- gt_polygons (list[np.float32])

- gt_ignored (**np.bool_**)
- gt_texts (list[str])

Parameters

- **with_bbox** (*bool*) – Whether to parse and load the bbox annotation. Defaults to False.
- **with_label** (*bool*) – Whether to parse and load the label annotation. Defaults to False.
- **with_polygon** (*bool*) – Whether to parse and load the polygon annotation. Defaults to False.
- **with_text** (*bool*) – Whether to parse and load the text annotation. Defaults to False.

Return type None

transform(*results*)

Function to load multiple types annotations.

Parameters **results** (*dict*) – Result dict from :obj:OCRDataset.

Returns The dict contains loaded bounding box, label polygon and text annotations.

Return type dict

45.1.3 LoadKIEAnnotations

class mmocr.datasets.transforms.**LoadKIEAnnotations**(*with_bbox=True*, *with_label=True*, *with_text=True*, *directed=False*, *key_node_idx=None*, *value_node_idx=None*, ***kwargs*)

Load and process the `instances` annotation provided by dataset.

The annotation format is as the following:

```
{
    # A nested list of 4 numbers representing the bounding box of the
    # instance, in (x1, y1, x2, y2) order.
    'bbox': np.array([[x1, y1, x2, y2], [x1, y1, x2, y2], ...],
                     dtype=np.int32),

    # Labels of boxes. Shape is (N,).
    'bbox_labels': np.array([0, 2, ...], dtype=np.int32),

    # Labels of edges. Shape (N, N).
    'edge_labels': np.array([0, 2, ...], dtype=np.int32),

    # List of texts.
    "texts": ['text1', 'text2', ...],
}
```

After this module, the annotation has been changed to the format below:

```
{
    # In (x1, y1, x2, y2) order, float type. N is the number of bboxes
    # in np.float32
    'gt_bboxes': np.ndarray(N, 4),
```

(continues on next page)

(continued from previous page)

```
    # In np.int64 type.
    'gt_bboxes_labels': np.ndarray(N, ),
    # In np.int32 type.
    'gt_edges_labels': np.ndarray(N, N),
    # In list[str]
    'gt_texts': list[str],
    # tuple(int)
    'ori_shape': (H, W)
}
```

Required Keys:

- bboxes
- bbox_labels
- edge_labels
- texts

Added Keys:

- gt_bboxes (np.float32)
- gt_bboxes_labels (np.int64)
- gt_edges_labels (np.int64)
- gt_texts (list[str])
- ori_shape (tuple[int])

Parameters

- **with_bbox** (*bool*) – Whether to parse and load the bbox annotation. Defaults to True.
- **with_label** (*bool*) – Whether to parse and load the label annotation. Defaults to True.
- **with_text** (*bool*) – Whether to parse and load the text annotation. Defaults to True.
- **directed** (*bool*) – Whether build edges as a directed graph. Defaults to False.
- **key_node_idx** (*int, optional*) – Key node label, used to mask out edges that are not connected from key nodes to value nodes. It has to be specified together with `value_node_idx`. Defaults to None.
- **value_node_idx** (*int, optional*) – Value node label, used to mask out edges that are not connected from key nodes to value nodes. It has to be specified together with `key_node_idx`. Defaults to None.

Return type None

transform(*results*)

Function to load multiple types annotations.

Parameters **results** (*dict*) – Result dict from :obj:`OCRDataset`.

Returns The dict contains loaded bounding box, label polygon and text annotations.

Return type dict

45.1.4 InferencerLoader

class mmocr.datasets.transforms.**InferencerLoader**(***kwargs*)

Load the image in Inferencer's pipeline.

Modified Keys:

- img
- img_path
- img_shape
- ori_shape

Parameters **to_float32** (*bool*) – Whether to convert the loaded image to a float32 numpy array. If set to False, the loaded image is an uint8 array. Defaults to False.

Return type None

transform(*single_input*)

Transform function to add image meta information.

Parameters **single_input** (*str or dict or np.ndarray*) – The raw input from inferencer.

Returns The dict contains loaded image and meta information.

Return type dict

45.2 TextDet Transforms

BoundedScaleAspectJitter	First randomly rescale the image so that the longside and shortside of the image are around the bound; then jitter its aspect ratio.
RandomFlip	Flip the image & bbox polygon.
SourceImagePad	Pad Image to target size.
ShortScaleAspectJitter	First rescale the image for its shorter side to reach the short_size and then jitter its aspect ratio, final rescale the shape guaranteed to be divided by scale_divisor.
TextDetRandomCrop	Randomly select a region and crop images to a target size and make sure to contain text region.
TextDetRandomCropFlip	Random crop and flip a patch in the image.

45.2.1 BoundedScaleAspectJitter

class mmocr.datasets.transforms.**BoundedScaleAspectJitter**(*long_size_bound*, *short_size_bound*, *ratio_range=(0.7, 1.3)*, *aspect_ratio_range=(0.9, 1.1)*, *resize_type='Resize'*, ***resize_kwargs*)

First randomly rescale the image so that the longside and shortside of the image are around the bound; then jitter its aspect ratio.

Required Keys:

- img
- img_shape
- gt_bboxes (optional)
- gt_polygons (optional)

Modified Keys:

- img
- img_shape
- gt_bboxes (optional)
- gt_polygons (optional)

Added Keys:

- scale
- scale_factor
- keep_ratio

Parameters

- **long_size_bound** (*int*) – The approximate bound for long size.
- **short_size_bound** (*int*) – The approximate bound for short size.
- **size_jitter_range** (*tuple(float, float)*) – Range of the ratio used to jitter the size. Defaults to (0.7, 1.3).
- **aspect_ratio_jitter_range** (*tuple(float, float)*) – Range of the ratio used to jitter its aspect ratio. Defaults to (0.9, 1.1).
- **resize_type** (*str*) – The type of resize class to use. Defaults to "Resize".
- ****resize_kwargs** – Other keyword arguments for the `resize_type`.
- **ratio_range** (*Tuple[float, float]*) –
- **aspect_ratio_range** (*Tuple[float, float]*) –

Return type None

transform(*results*)

The transform function. All subclass of BaseTransform should override this method.

This function takes the result dict as the input, and can add new items to the dict or modify existing items in the dict. And the result dict will be returned in the end, which allows to concate multiple transforms into a pipeline.

Parameters **results** (*dict*) – The result dict.

Returns The result dict.

Return type dict

45.2.2 RandomFlip

class mmocr.datasets.transforms.**RandomFlip**(*prob=None*, *direction='horizontal'*, *swap_seg_labels=None*)

Flip the image & bbox polygon.

There are 3 flip modes:

- **prob is float, direction is string: the image will be** `direction``ly flipped with probability of ``prob`. E.g., `prob=0.5`, `direction='horizontal'`, then image will be horizontally flipped with probability of 0.5.
- **prob is float, direction is list of string: the image will** be `direction[i]``ly flipped with probability of ``prob/len(direction)`. E.g., `prob=0.5`, `direction=['horizontal', 'vertical']`, then image will be horizontally flipped with probability of 0.25, vertically with probability of 0.25.
- **prob is list of float, direction is list of string:** given `len(prob) == len(direction)`, the image will be `direction[i]``ly flipped with probability of ``prob[i]`. E.g., `prob=[0.3, 0.5]`, `direction=['horizontal', 'vertical']`, then image will be horizontally flipped with probability of 0.3, vertically with probability of 0.5.

Required Keys:

- img
- gt_bboxes (optional)
- gt_polygons (optional)

Modified Keys:

- img
- gt_bboxes (optional)
- gt_polygons (optional)

Added Keys:

- flip
- flip_direction

Parameters

- **prob** (*float | list[float], optional*) – The flipping probability. Defaults to None.
- **direction** (*str | list[str]*) – The flipping direction. Options If input is a list, the length must equal `prob`. Each element in `prob` indicates the flip probability of corresponding direction. Defaults to 'horizontal'.
- **swap_seg_labels** (*Optional[Sequence]*) –

Return type None

flip_polygons(*polygons*, *img_shape*, *direction*)

Flip polygons horizontally, vertically or diagonally.

Parameters

- **polygons** (*list[numpy.ndarray*) – polygons.
- **img_shape** (*tuple[int]*) – Image shape (height, width)

- **direction** (*str*) – Flip direction. Options are 'horizontal', 'vertical' and 'diagonal'.

Returns Flipped polygons.

Return type list[numpy.ndarray]

45.2.3 SourceImagePad

class mmocr.datasets.transforms.**SourceImagePad**(*target_scale*, *crop_ratio=0.1111111111111111*)

Pad Image to target size. It will randomly crop an area from the original image and resize it to the target size, then paste the original image to its top left corner.

Required Keys:

- img

Modified Keys:

- img
- img_shape

Added Keys: - pad_shape - pad_fixed_size

Parameters

- **target_scale** (*int or tuple[int, int]*) – The target size of padded image. If it's an integer, then the padding size would be (target_size, target_size). If it's tuple, then `target_scale[0]` should be the width and `target_scale[1]` should be the height. The size of the padded image will be (target_scale[1], target_scale[0])
- **crop_ratio** (*float or Tuple[float, float]*) – Relative size for the crop region. If `crop_ratio` is a float, then the initial crop size would be `(crop_ratio * img.shape[0], crop_ratio * img.shape[1])` . If `crop_ratio` is a tuple, then `crop_ratio[0]` is for the width and `crop_ratio[1]` is for the height. The initial crop size would be `(crop_ratio[1] * img.shape[0], crop_ratio[0] * img.shape[1])`. Defaults to 1./9.

Return type None

transform(*results*)

Pad Image to target size. It will randomly select a small area from the original image and resize it to the target size, then paste the original image to its top left corner.

Parameters **results** (*Dict*) – Result dict containing the data to transform.

Returns The transformed data.

Return type (Dict)

45.2.4 ShortScaleAspectJitter

class mmocr.datasets.transforms.**ShortScaleAspectJitter**(*short_size=736*, *ratio_range=(0.7, 1.3)*, *aspect_ratio_range=(0.9, 1.1)*, *scale_divisor=1*, *resize_type='Resize'*, ***resize_kwargs*)

First rescale the image for its shorter side to reach the short_size and then jitter its aspect ratio, final rescale the shape guaranteed to be divided by scale_divisor.

Required Keys:

- img
- img_shape
- gt_bboxes (optional)
- gt_polygons (optional)

Modified Keys:

- img
- img_shape
- gt_bboxes (optional)
- gt_polygons (optional)

Added Keys:

- scale
- scale_factor
- keep_ratio

Parameters

- **short_size** (*int*) – Target shorter size before jittering the aspect ratio. Defaults to 736.
- **short_size_jitter_range** (*tuple(float, float)*) – Range of the ratio used to jitter the target shorter size. Defaults to (0.7, 1.3).
- **aspect_ratio_jitter_range** (*tuple(float, float)*) – Range of the ratio used to jitter its aspect ratio. Defaults to (0.9, 1.1).
- **scale_divisor** (*int*) – The scale divisor. Defaults to 1.
- **resize_type** (*str*) – The type of resize class to use. Defaults to "Resize".
- ****resize_kwargs** – Other keyword arguments for the `resize_type`.
- **ratio_range** (*Tuple[float, float]*) –
- **aspect_ratio_range** (*Tuple[float, float]*) –

Return type None

transform(*results*)

Short Scale Aspect Jitter. :param results: Result dict containing the data to transform. :type results: dict

Returns The transformed data.

Return type dict

Parameters **results** (*Dict*) –

45.2.5 TextDetRandomCrop

class mmocr.datasets.transforms.**TextDetRandomCrop**(*target_size*, *positive_sample_ratio=0.625*)

Randomly select a region and crop images to a target size and make sure to contain text region. This transform may break up text instances, and for broken text instances, we will crop it's bbox and polygon coordinates. This transform is recommend to be used in segmentation-based network.

Required Keys:

- img
- gt_polygons
- gt_bboxes
- gt_bboxes_labels
- gt_ignored

Modified Keys:

- img
- img_shape
- gt_polygons
- gt_bboxes
- gt_bboxes_labels
- gt_ignored

Parameters

- **target_size** (*tuple(int, int) or int*) – Target size for the cropped image. If it's a tuple, then target width and target height will be `target_size[0]` and `target_size[1]`, respectively. If it's an integer, them both target width and target height will be `target_size`.
- **positive_sample_ratio** (*float*) – The probability of sampling regions that go through text regions. Defaults to 5. / 8.

Return type None

transform(*results*)

Applying random crop on results. :param results: Result dict contains the data to transform. :type results: dict

Returns The transformed data

Return type dict

Parameters **results** (*Dict*) –

45.2.6 TextDetRandomCropFlip

class mmocr.datasets.transforms.**TextDetRandomCropFlip**(*pad_ratio=0.1*, *crop_ratio=0.5*, *iter_num=1*, *min_area_ratio=0.2*, *epsilon=0.01*)

Random crop and flip a patch in the image. Only used in text detection task.

Required Keys:

- img
- gt_bboxes
- gt_polygons

Modified Keys:

- img
- gt_bboxes
- gt_polygons

Parameters

- **pad_ratio** (*float*) – The ratio of padding. Defaults to 0.1.
- **crop_ratio** (*float*) – The ratio of cropping. Defaults to 0.5.
- **iter_num** (*int*) – Number of operations. Defaults to 1.
- **min_area_ratio** (*float*) – Minimal area ratio between cropped patch and original image. Defaults to 0.2.
- **epsilon** (*float*) – The threshold of polygon IoU between cropped area and polygon, which is used to avoid cropping text instances. Defaults to 0.01.

Return type None

transform(*results*)

Applying random crop flip on results.

Parameters **results** (*dict*) – Result dict containing the data to transform

Returns The transformed data

Return type dict

45.3 TextRecog Transforms

TextRecogGeneralAug	A general geometric augmentation tool for text images in the CVPR 2020 paper "Learn to Augment: Joint Data Augmentation and Network Optimization for Text Recognition".
CropHeight	Randomly crop the image's height, either from top or bottom.
ImageContentJitter	Jitter the image contents.
ReversePixels	Reverse image pixels.
PyramidRescale	Resize the image to the base shape, downsample it with gaussian pyramid, and rescale it back to original size.

continues on next page

Table 3 – continued from previous page

PadToWidth	Only pad the image's width.
RescaleToHeight	Rescale the image to the height according to setting and keep the aspect ratio unchanged if possible.

45.3.1 TextRecogGeneralAug

class mmocr.datasets.transforms.**TextRecogGeneralAug**

A general geometric augmentation tool for text images in the CVPR 2020 paper "Learn to Augment: Joint Data Augmentation and Network Optimization for Text Recognition". It applies distortion, stretching, and perspective transforms to an image.

This implementation is adapted from https://github.com/RubanSeven/Text-Image-Augmentation-python/blob/master/augment.py # noqa

TODO: Split this transform into three transforms.

Required Keys:

- img

Modified Keys:

- img
- img_shape

tia_distort(*img*, *segment=4*)

Image distortion.

Parameters

- **img** (*np.ndarray*) – The image.
- **segment** (*int*) – The number of segments to divide the image along the width. Defaults to 4.

Return type numpy.ndarray

tia_perspective(*img*)

Image perspective transformation.

Parameters

- **img** (*np.ndarray*) – The image.
- **segment** (*int*) – The number of segments to divide the image along the width. Defaults to 4.

Return type numpy.ndarray

tia_stretch(*img*, *segment=4*)

Image stretching.

Parameters

- **img** (*np.ndarray*) – The image.
- **segment** (*int*) – The number of segments to divide the image along the width. Defaults to 4.

Return type numpy.ndarray

transform(*results*)
Call function to pad images.

Parameters **results** (*dict*) – Result dict from loading pipeline.

Returns Updated result dict.

Return type dict

warp_mls(*src*, *src_pts*, *dst_pts*, *dst_w*, *dst_h*, *trans_ratio=1.0*)
Warp the image.

Parameters

- **src** (*numpy.ndarray*) –
- **src_pts** (*List[int]*) –
- **dst_pts** (*List[int]*) –
- **dst_w** (*int*) –
- **dst_h** (*int*) –
- **trans_ratio** (*float*) –

Return type numpy.ndarray

45.3.2 CropHeight

class mmocr.datasets.transforms.**CropHeight**(*min_pixels=1*, *max_pixels=8*)
Randomly crop the image's height, either from top or bottom.

Adapted from https://github.com/PaddlePaddle/PaddleOCR/blob/release%2F2.6/ppocr/data/imaug/rec_img_aug.py # noqa

Required Keys:

- img

Modified Keys:

- img
- img_shape

Parameters

- **crop_min** (*int*) – Minimum pixel(s) to crop. Defaults to 1.
- **crop_max** (*int*) – Maximum pixel(s) to crop. Defaults to 8.
- **min_pixels** (*int*) –
- **max_pixels** (*int*) –

Return type None

transform(*results*)
Transform function to crop images.

Parameters **results** (*dict*) – Result dict from loading pipeline.

Returns Cropped results.

Return type dict

45.3.3 ImageContentJitter

class mmocr.datasets.transforms.**ImageContentJitter**

Jitter the image contents.

Adapted from https://github.com/PaddlePaddle/PaddleOCR/blob/release%2F2.6/ppocr/data/imaug/rec_img_aug.py # noqa

Required Keys:

- img

Modified Keys:

- img

transform(*results*, *jitter_ratio=0.01*)

Transform function to jitter images.

Parameters

- **results** (*dict*) – Result dict from loading pipeline.
- **jitter_ratio** (*float*) – Controls the strength of jittering. Defaults to 0.01.

Returns Jittered results.

Return type dict

45.3.4 ReversePixels

class mmocr.datasets.transforms.**ReversePixels**

Reverse image pixels.

Adapted from https://github.com/PaddlePaddle/PaddleOCR/blob/release%2F2.6/ppocr/data/imaug/rec_img_aug.py # noqa

Required Keys:

- img

Modified Keys:

- img

transform(*results*)

Transform function to reverse image pixels.

Parameters **results** (*dict*) – Result dict from loading pipeline.

Returns Reversed results.

Return type dict

45.3.5 PyramidRescale

class mmocr.datasets.transforms.**PyramidRescale**(*factor=4*, *base_shape=(128, 512)*, *randomize_factor=True*)

Resize the image to the base shape, downsample it with gaussian pyramid, and rescale it back to original size.

Adapted from https://github.com/FangShancheng/ABINet.

Required Keys:

- img (ndarray)

Modified Keys:

- img (ndarray)

Parameters

- **factor** (*int*) – The decay factor from base size, or the number of downsampling operations from the base layer.
- **base_shape** (*tuple[int, int]*) – The shape (width, height) of the base layer of the pyramid.
- **randomize_factor** (*bool*) – If True, the final factor would be a random integer in [0, factor].

Return type None

transform(*results*)

Applying pyramid rescale on results.

Parameters **results** (*dict*) – Result dict containing the data to transform.

Returns The transformed data.

Return type Dict

45.3.6 PadToWidth

class mmocr.datasets.transforms.**PadToWidth**(*width*, *pad_cfg={'type': 'Pad'}*)

Only pad the image's width.

Required Keys:

- img

Modified Keys:

- img
- img_shape

Added Keys:

- pad_shape
- pad_fixed_size
- pad_size_divisor
- valid_ratio

Parameters

- **width** (*int*) – Target width of padded image. Defaults to None.
- **pad_cfg** (*dict*) – Config to construct the Resize transform. Refer to Pad for detail. Defaults to `dict(type='Pad')`.

Return type None

transform(*results*)

Call function to pad images.

Parameters **results** (*dict*) – Result dict from loading pipeline.

Returns Updated result dict.

Return type dict

45.3.7 RescaleToHeight

class mmocr.datasets.transforms.**RescaleToHeight**(*height*, *min_width=None*, *max_width=None*, *width_divisor=1*, *resize_type='Resize'*, ***resize_kwargs*)

Rescale the image to the height according to setting and keep the aspect ratio unchanged if possible. However, if any of `min_width`, `max_width` or `width_divisor` are specified, aspect ratio may still be changed to ensure the width meets these constraints.

Required Keys:

- img

Modified Keys:

- img
- img_shape

Added Keys:

- scale
- scale_factor
- keep_ratio

Parameters

- **height** (*int*) – Height of rescaled image.
- **min_width** (*int, optional*) – Minimum width of rescaled image. Defaults to None.
- **max_width** (*int, optional*) – Maximum width of rescaled image. Defaults to None.
- **width_divisor** (*int*) – The divisor of width size. Defaults to 1.
- **resize_type** (*str*) – The type of resize class to use. Defaults to "Resize".
- ****resize_kwargs** – Other keyword arguments for the `resize_type`.

Return type None

transform(*results*)

Transform function to resize images, bounding boxes and polygons.

Parameters **results** (*dict*) – Result dict from loading pipeline.

Returns Resized results.

Return type dict

45.4 OCR Transforms

RandomCrop	Randomly crop images and make sure to contain at least one intact instance.
RandomRotate	Randomly rotate the image, boxes, and polygons.
Resize	Resize image & bboxes & polygons.
FixInvalidPolygon	Fix invalid polygons in the dataset.
RemoveIgnored	Removed ignored elements from the pipeline.

45.4.1 RandomCrop

class mmocr.datasets.transforms.**RandomCrop**(*min_side_ratio=0.4*)

Randomly crop images and make sure to contain at least one intact instance.

Required Keys:

- img
- gt_polygons
- gt_bboxes
- gt_bboxes_labels
- gt_ignored
- gt_texts (optional)

Modified Keys:

- img
- img_shape
- gt_polygons
- gt_bboxes
- gt_bboxes_labels
- gt_ignored
- gt_texts (optional)

Parameters **min_side_ratio** (*float*) – The ratio of the shortest edge of the cropped image to the original image size.

Return type None

transform(*results*)

Applying random crop on results. :param results: Result dict contains the data to transform. :type results: dict

Returns The transformed data.

Return type dict

Parameters **results** (*Dict*) –

45.4.2 RandomRotate

class mmocr.datasets.transforms.**RandomRotate**(*max_angle=10*, *pad_with_fixed_color=False*, *pad_value=(0, 0, 0)*, *use_canvas=False*)

Randomly rotate the image, boxes, and polygons. For recognition task, only the image will be rotated. If set `use_canvas` as True, the shape of rotated image might be modified based on the rotated angle size, otherwise, the image will keep the shape before rotation.

Required Keys:

- img
- img_shape
- gt_bboxes (optional)
- gt_polygons (optional)

Modified Keys:

- img
- img_shape (optional)
- gt_bboxes (optional)
- gt_polygons (optional)

Added Keys:

- rotated_angle

Parameters

- **max_angle** (*int*) – The maximum rotation angle (can be bigger than 180 or a negative). Defaults to 10.
- **pad_with_fixed_color** (*bool*) – The flag for whether to pad rotated image with fixed value. Defaults to False.
- **pad_value** (*tuple[int, int, int]*) – The color value for padding rotated image. Defaults to (0, 0, 0).
- **use_canvas** (*bool*) – Whether to create a canvas for rotated image. Defaults to False. If set true, the image shape may be modified.

Return type None

transform(*results*)

Applying random rotate on results.

Parameters

- **results** (*Dict*) – Result dict containing the data to transform.
- **center_shift** (*Tuple[int, int]*) – The shifting offset of the center point

Returns The transformed data

Return type dict

45.4.3 Resize

class mmocr.datasets.transforms.**Resize**(*scale=None*, *scale_factor=None*, *keep_ratio=False*, *clip_object_border=True*, *backend='cv2'*, *interpolation='bilinear'*)

Resize image & bboxes & polygons.

This transform resizes the input image according to `scale` or `scale_factor`. Bboxes and polygons are then resized with the same scale factor. if `scale` and `scale_factor` are both set, it will use `scale` to resize.

Required Keys:

- img
- img_shape
- gt_bboxes
- gt_polygons

Modified Keys:

- img
- img_shape
- gt_bboxes
- gt_polygons

Added Keys:

- scale
- scale_factor
- keep_ratio

Parameters

- **scale** (*int or tuple*) – Image scales for resizing. Defaults to None.
- **scale_factor** (*float or tuple[float, float]*) – Scale factors for resizing. It's either a factor applicable to both dimensions or in the form of (scale_w, scale_h). Defaults to None.
- **keep_ratio** (*bool*) – Whether to keep the aspect ratio when resizing the image. Defaults to False.
- **clip_object_border** (*bool*) – Whether to clip the objects outside the border of the image. Defaults to True.
- **backend** (*str*) – Image resize backend, choices are 'cv2' and 'pillow'. These two backends generates slightly different results. Defaults to 'cv2'.
- **interpolation** (*str*) – Interpolation method, accepted values are "nearest", "bilinear", "bicubic", "area", "lanczos" for 'cv2' backend, "nearest", "bilinear" for 'pillow' backend. Defaults to 'bilinear'.

Return type None

transform(*results*)

Transform function to resize images, bounding boxes and polygons.

Parameters **results** (*dict*) – Result dict from loading pipeline.

Returns Resized results, 'img', 'gt_bboxes', 'gt_polygons', 'scale', 'scale_factor', 'height', 'width', and 'keep_ratio' keys are updated in result dict.

Return type dict

45.4.4 FixInvalidPolygon

class mmocr.datasets.transforms.**FixInvalidPolygon**(*mode='fix'*, *min_poly_points=4*, *fix_from_bbox=True*)

Fix invalid polygons in the dataset.

Required Keys:

- gt_polygons
- gt_ignored (optional)
- gt_bboxes (optional)
- gt_bboxes_labels (optional)
- gt_texts (optional)

Modified Keys:

- gt_polygons
- gt_ignored (optional)
- gt_bboxes (optional)
- gt_bboxes_labels (optional)
- gt_texts (optional)

Parameters

- **mode** (*str*) – The mode of fixing invalid polygons. Options are 'fix' and 'ignore'. For the 'fix' mode, the transform will try to fix the invalid polygons to a valid one by eliminating the self-intersection or converting the bboxes to polygons. If it can't be fixed by any means (e.g. the polygon contains less than 3 points or it's actually a line/point), the annotation will be removed. For the 'ignore' mode, the invalid polygons will be set to "ignored" during training. Defaults to 'fix'.
- **min_poly_points** (*int*) – Minimum number of the coordinate points in a polygon. Defaults to 4.
- **fix_from_bbox** (*bool*) – Whether to convert the bboxes to polygons when the polygon is invalid and not directly fixable. Defaults to True.

Return type None

transform(*results*)

Fix invalid polygons.

Parameters **results** (*dict*) – Result dict containing the data to transform.

Returns The transformed data. If all the polygons are unfixable, return None.

Return type Optional[dict]

45.4.5 RemoveIgnored

class mmocr.datasets.transforms.**RemoveIgnored**

Removed ignored elements from the pipeline.

Required Keys:

- gt_ignored
- gt_polygons (optional)
- gt_bboxes (optional)
- gt_bboxes_labels (optional)
- gt_texts (optional)

Modified Keys:

- gt_ignored
- gt_polygons (optional)
- gt_bboxes (optional)
- gt_bboxes_labels (optional)
- gt_texts (optional)

transform(*results*)

The transform function. All subclass of BaseTransform should override this method.

This function takes the result dict as the input, and can add new items to the dict or modify existing items in the dict. And the result dict will be returned in the end, which allows to concate multiple transforms into a pipeline.

Parameters **results** (*dict*) – The result dict.

Returns The result dict.

Return type dict

45.5 Formatting

PackTextDetInputs	Pack the inputs data for text detection.
PackTextRecogInputs	Pack the inputs data for text recognition.
PackKIEInputs	Pack the inputs data for key information extraction.

45.5.1 PackTextDetInputs

class mmocr.datasets.transforms.**PackTextDetInputs**(*meta_keys=('img_path', 'ori_shape', 'img_shape', 'scale_factor', 'flip', 'flip_direction')*)

Pack the inputs data for text detection.

The type of outputs is *dict*:

- inputs: image converted to tensor, whose shape is (C, H, W).
- data_samples: Two components of `TextDetDataSample` will be updated:
  - gt_instances (InstanceData): Depending on annotations, a subset of the following keys will be updated:

* bboxes (torch.Tensor((N, 4), dtype=torch.float32)): The groundtruth of bounding boxes in the form of [x1, y1, x2, y2]. Renamed from 'gt_bboxes'.
* labels (torch.LongTensor(N)): The labels of instances. Renamed from 'gt_bboxes_labels'.
* polygons(list[np.array((2k,), dtype=np.float32)]): The groundtruth of polygons in the form of [x1, y1,..., xk, yk]. Each element in polygons may have different number of points. Renamed from 'gt_polygons'. Using numpy instead of tensor is that polygon usually is not the output of model and operated on cpu.
* ignored (torch.BoolTensor((N,))): The flag indicating whether the corresponding instance should be ignored. Renamed from 'gt_ignored'.
* texts (list[str]): The groundtruth texts. Renamed from 'gt_texts'.

– metainfo (dict): 'metainfo' is always populated. The contents of the 'metainfo' depends on `meta_keys`. By default it includes:

* "img_path": Path to the image file.
* "img_shape": Shape of the image input to the network as a tuple (h, w). Note that the image may be zero-padded afterward on the bottom/right if the batch tensor is larger than this shape.
* "scale_factor": A tuple indicating the ratio of width and height of the preprocessed image to the original one.
* "ori_shape": Shape of the preprocessed image as a tuple (h, w).
* "pad_shape": Image shape after padding (if any Pad-related transform involved) as a tuple (h, w).
* "flip": A boolean indicating if the image has been flipped.
* `flip_direction`: the flipping direction.

Parameters **meta_keys** (*Sequence[str], optional*) – Meta keys to be converted to the metainfo of `TextDetSample`. Defaults to `('img_path', 'ori_shape', 'img_shape', 'scale_factor', 'flip', 'flip_direction')`.

transform(*results*)

Method to pack the input data.

Parameters **results** (*dict*) – Result dict from the data pipeline.

Returns

- 'inputs' (obj:*torch.Tensor*): Data for model forwarding.
- 'data_samples' (obj:*DetDataSample*): The annotation info of the sample.

Return type dict

45.5.2 PackTextRecogInputs

class mmocr.datasets.transforms.**PackTextRecogInputs**(*meta_keys=('img_path', 'ori_shape', 'img_shape', 'pad_shape', 'valid_ratio')*)

Pack the inputs data for text recognition.

The type of outputs is *dict*:

- inputs: Image as a tensor, whose shape is (C, H, W).
- data_samples: Two components of `TextRecogDataSample` will be updated:
  – gt_text (LabelData):

* item(str): The groundtruth of text. Rename from 'gt_texts'.

- metainfo (dict): 'metainfo' is always populated. The contents of the 'metainfo' depends on `meta_keys`. By default it includes:
  * "img_path": Path to the image file.
  * "ori_shape": Shape of the preprocessed image as a tuple (h, w).
  * "img_shape": Shape of the image input to the network as a tuple (h, w). Note that the image may be zero-padded afterward on the bottom/right if the batch tensor is larger than this shape.
  * "valid_ratio": The proportion of valid (unpadded) content of image on the x-axis. It defaults to 1 if not set in pipeline.

Parameters `meta_keys` (*Sequence[str], optional*) – Meta keys to be converted to the metainfo of `TextRecogDataSampel`. Defaults to `('img_path', 'ori_shape', 'img_shape', 'pad_shape', 'valid_ratio')`.

`transform`(*results*)

Method to pack the input data.

Parameters `results` (*dict*) – Result dict from the data pipeline.

Returns

- 'inputs' (obj:*torch.Tensor*): Data for model forwarding.
- **'data_samples' (obj:*TextRecogDataSample*): The annotation info** of the sample.

Return type dict

45.5.3 PackKIEInputs

`class mmocr.datasets.transforms.PackKIEInputs`(*meta_keys=()*)

Pack the inputs data for key information extraction.

The type of outputs is *dict*:

- inputs: image converted to tensor, whose shape is (C, H, W).
- data_samples: Two components of `TextDetDataSample` will be updated:
  - gt_instances (InstanceData): Depending on annotations, a subset of the following keys will be updated:
    * bboxes (torch.Tensor((N, 4), dtype=torch.float32)): The groundtruth of bounding boxes in the form of [x1, y1, x2, y2]. Renamed from 'gt_bboxes'.
    * labels (torch.LongTensor(N)): The labels of instances. Renamed from 'gt_bboxes_labels'.
    * edge_labels (torch.LongTensor(N, N)): The edge labels. Renamed from 'gt_edges_labels'.
    * texts (list[str]): The groundtruth texts. Renamed from 'gt_texts'.
  - metainfo (dict): 'metainfo' is always populated. The contents of the 'metainfo' depends on `meta_keys`. By default it includes:
    * "img_path": Path to the image file.
    * "img_shape": Shape of the image input to the network as a tuple (h, w). Note that the image may be zero-padded afterward on the bottom/right if the batch tensor is larger than this shape.
    * "scale_factor": A tuple indicating the ratio of width and height of the preprocessed image to the original one.

* “ori_shape”: Shape of the preprocessed image as a tuple (h, w).

Parameters **meta_keys** (*Sequence[str], optional*) – Meta keys to be converted to the metainfo of `TextDetSample`. Defaults to `('img_path', 'ori_shape', 'img_shape', 'scale_factor', 'flip', 'flip_direction')`.

transform(*results*)

Method to pack the input data.

Parameters **results** (*dict*) – Result dict from the data pipeline.

Returns

- ‘inputs’ (obj:*torch.Tensor*): Data for model forwarding.
- ’data_samples’ (obj:*DetDataSample*): The annotation info of the sample.

Return type dict

45.6 Transform Wrapper

ImgAugWrapper	A wrapper around imgaug https://github.com/aleju/imgaug.
TorchVisionWrapper	A wrapper around torchvision transforms.

45.6.1 ImgAugWrapper

class mmocr.datasets.transforms.**ImgAugWrapper**(*args=None*, *fix_poly_trans={'type': 'FixInvalidPolygon'}*)

A wrapper around imgaug https://github.com/aleju/imgaug.

Find available augmenters at https://imgaug.readthedocs.io/en/latest/source/overview_of_augmenters.html.

Required Keys:

- img
- gt_polygons (optional for text recognition)
- gt_bboxes (optional for text recognition)
- gt_bboxes_labels (optional for text recognition)
- gt_ignored (optional for text recognition)
- gt_texts (optional)

Modified Keys:

- img
- gt_polygons (optional for text recognition)
- gt_bboxes (optional for text recognition)
- gt_bboxes_labels (optional for text recognition)
- gt_ignored (optional for text recognition)
- img_shape (optional)

- gt_texts (optional)

Parameters

- **args** (*list[list or dict]], optional*) – The argumentation list. For details, please refer to imgaug document. Take args=[['Fliplr', 0.5], dict(cls='Affine', rotate=[-10, 10]), ['Resize', [0.5, 3.0]]] as an example. The args horizontally flip images with probability 0.5, followed by random rotation with angles in range [-10, 10], and resize with an independent scale in range [0.5, 3.0] for each side of images. Defaults to None.
- **fix_poly_trans** (*dict*) – The transform configuration to fix invalid polygons. Set it to None if no fixing is needed. Defaults to dict(type='FixInvalidPolygon').

Return type None

transform(*results*)

Transform the image and annotation data.

Parameters **results** (*dict*) – Result dict containing the data to transform.

Returns The transformed data.

Return type dict

45.6.2 TorchVisionWrapper

class mmocr.datasets.transforms.**TorchVisionWrapper**(*op*, ***kwargs*)

A wrapper around torchvision transforms. It applies specific transform to `img` and updates `height` and `width` accordingly.

Required Keys:

- img (ndarray): The input image.

Modified Keys:

- img (ndarray): The modified image.
- img_shape (tuple(int, int)): The shape of the image in (height, width).

Warning: This transform only affects the image but not its associated annotations, such as word bounding boxes and polygons. Therefore, it may only be applicable to text recognition tasks.

Parameters

- **op** (*str*) – The name of any transform class in `torchvision.transforms()`.
- ****kwargs** – Arguments that will be passed to initializer of torchvision transform.

Return type None

transform(*results*)

Transform the image.

Parameters **results** (*dict*) – Result dict from the data loader.

Returns Transformed results.

Return type dict

45.7 Adapter

MMDet2MMOCR	Convert transforms's data format from MMDet to MMOCR.
MMOCR2MMDet	Convert transforms's data format from MMOCR to MMDet.

45.7.1 MMDet2MMOCR

class mmocr.datasets.transforms.**MMDet2MMOCR**

Convert transforms's data format from MMDet to MMOCR.

Required Keys:

- gt_masks (PolygonMasks | BitmapMasks) (optional)
- gt_ignore_flags (np.bool) (optional)

Added Keys:

- gt_polygons (list[np.ndarray])
- gt_ignored (np.ndarray)

transform(*results*)

Convert MMDet's data format to MMOCR's data format.

Parameters **results** (*Dict*) – Result dict containing the data to transform.

Returns The transformed data.

Return type (Dict)

45.7.2 MMOCR2MMDet

class mmocr.datasets.transforms.**MMOCR2MMDet**(*poly2mask=False*)

Convert transforms's data format from MMOCR to MMDet.

Required Keys:

- img_shape
- gt_polygons (List[ndarray]) (optional)
- gt_ignored (np.bool) (optional)

Added Keys:

- gt_masks (PolygonMasks | BitmapMasks) (optional)
- gt_ignore_flags (np.bool) (optional)

Parameters **poly2mask** (*bool*) – Whether to convert mask to bitmap. Default: True.

Return type None

transform(*results*)

Convert MMOCR's data format to MMDet's data format.

Parameters **results** (*Dict*) – Result dict containing the data to transform.

Returns The transformed data.

Return type (Dict)

CHAPTER

FORTYSIX

MMOCR.MODELS

46.1 models.common

46.1.1 BackBones

UNet	UNet backbone.

UNet

class mmocr.models.common.**UNet**(*in_channels=3*, *base_channels=64*, *num_stages=5*, *strides=(1, 1, 1, 1, 1)*, *enc_num_convs=(2, 2, 2, 2, 2)*, *dec_num_convs=(2, 2, 2, 2)*, *downsamples=(True, True, True, True)*, *enc_dilations=(1, 1, 1, 1, 1)*, *dec_dilations=(1, 1, 1, 1)*, *with_cp=False*, *conv_cfg=None*, *norm_cfg={'type': 'BN'}*, *act_cfg={'type': 'ReLU'}*, *upsample_cfg={'type': 'InterpConv'}*, *norm_eval=False*, *dcn=None*, *plugins=None*, *init_cfg=[{'type': 'Kaiming', 'layer': 'Conv2d'}, {'type': 'Constant', 'layer': ['_BatchNorm', 'GroupNorm'], 'val': 1}]*)

UNet backbone. U-Net: Convolutional Networks for Biomedical Image Segmentation. https://arxiv.org/pdf/1505.04597.pdf

Parameters

- **in_channels** (*int*) – Number of input image channels. Default" 3.
- **base_channels** (*int*) – Number of base channels of each stage. The output channels of the first stage. Default: 64.
- **num_stages** (*int*) – Number of stages in encoder, normally 5. Default: 5.
- **strides** (*Sequence[int 1 | 2]*) – Strides of each stage in encoder. len(strides) is equal to num_stages. Normally the stride of the first stage in encoder is 1. If strides[i]=2, it uses stride convolution to downsample in the correspondence encoder stage. Default: (1, 1, 1, 1, 1).
- **enc_num_convs** (*Sequence[int]*) – Number of convolutional layers in the convolution block of the correspondence encoder stage. Default: (2, 2, 2, 2, 2).
- **dec_num_convs** (*Sequence[int]*) – Number of convolutional layers in the convolution block of the correspondence decoder stage. Default: (2, 2, 2, 2).
- **downsamples** (*Sequence[int]*) – Whether use MaxPool to downsample the feature map after the first stage of encoder (stages: [1, num_stages)). If the correspondence encoder stage use stride convolution (strides[i]=2), it will never use MaxPool to downsample, even downsamples[i-1]=True. Default: (True, True, True, True).
- **enc_dilations** (*Sequence[int]*) – Dilation rate of each stage in encoder. Default: (1, 1, 1, 1, 1).
- **dec_dilations** (*Sequence[int]*) – Dilation rate of each stage in decoder. Default: (1, 1, 1, 1).
- **with_cp** (*bool*) – Use checkpoint or not. Using checkpoint will save some memory while slowing down the training speed. Default: False.
- **conv_cfg** (*dict | None*) – Config dict for convolution layer. Default: None.
- **norm_cfg** (*dict | None*) – Config dict for normalization layer. Default: dict(type='BN').

- **act_cfg** (*dict | None*) – Config dict for activation layer in ConvModule. Default: dict(type='ReLU').
- **upsample_cfg** (*dict*) – The upsample config of the upsample module in decoder. Default: dict(type='InterpConv').
- **norm_eval** (*bool*) – Whether to set norm layers to eval mode, namely, freeze running stats (mean and var). Note: Effect on Batch Norm and its variants only. Default: False.
- **dcn** (*bool*) – Use deformable convolution in convolutional layer or not. Default: None.
- **plugins** (*dict*) – plugins for convolutional layers. Default: None.

Notice: The input image size should be divisible by the whole downsample rate of the encoder. More detail of the whole downsample rate can be found in UNet._check_input_divisible.

forward(*x*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

Note: Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

train(*mode=True*)

Convert the model into training mode while keep normalization layer freezed.

46.1.2 Dictionary

Dictionary	The class generates a dictionary for recognition.

Dictionary

class mmocr.models.common.**Dictionary**(*dict_file*, *with_start=False*, *with_end=False*, *same_start_end=False*, *with_padding=False*, *with_unknown=False*, *start_token='<BOS>'*, *end_token='<EOS>'*, *start_end_token='<BOS/EOS>'*, *padding_token='<PAD>'*, *unknown_token='<UKN>'*)

The class generates a dictionary for recognition. It pre-defines four special tokens: `start_token`, `end_token`, `pad_token`, and `unknown_token`, which will be sequentially placed at the end of the dictionary when their corresponding flags are True.

Parameters

- **dict_file** (*str*) – The path of Character dict file which a single character must occupies a line.
- **with_start** (*bool*) – The flag to control whether to include the start token. Defaults to False.
- **with_end** (*bool*) – The flag to control whether to include the end token. Defaults to False.
- **same_start_end** (*bool*) – The flag to control whether the start token and end token are the same. It only works when both `with_start` and `with_end` are True. Defaults to False.

- **with_padding** (*bool*) – The padding token may represent more than a padding. It can also represent tokens like the blank token in CTC or the background token in SegOCR. Defaults to False.
- **with_unknown** (*bool*) – The flag to control whether to include the unknown token. Defaults to False.
- **start_token** (*str*) – The start token as a string. Defaults to '<BOS>'.
- **end_token** (*str*) – The end token as a string. Defaults to '<EOS>'.
- **start_end_token** (*str*) – The start/end token as a string. if start and end is the same. Defaults to '<BOS/EOS>'.
- **padding_token** (*str*) – The padding token as a string. Defaults to '<PAD>'.
- **unknown_token** (*str, optional*) – The unknown token as a string. If it's set to None and with_unknown is True, the unknown token will be skipped when converting string to index. Defaults to '<UKN>'.

Return type None

char2idx(*char*, *strict=True*)

Convert a character to an index via Dictionary.dict.

Parameters

- **char** (*str*) – The character to convert to index.
- **strict** (*bool*) – The flag to control whether to raise an exception when the character is not in the dictionary. Defaults to True.

Returns The index of the character.

Return type int

property dict: list

Returns a list of characters to recognize, where special tokens are counted.

Type list

idx2str(*index*)

Convert a list of index to string.

Parameters **index** (*list[int]*) – The list of indexes to convert to string.

Returns The converted string.

Return type str

property num_classes: int

Number of output classes. Special tokens are counted.

Type int

str2idx(*string*)

Convert a string to a list of indexes via Dictionary.dict.

Parameters **string** (*str*) – The string to convert to indexes.

Returns The list of indexes of the string.

Return type list

46.1.3 Losses

MaskedBalancedBCEWithLogitsLoss	This loss combines a Sigmoid layers and a masked balanced BCE loss in one single class.
MaskedDiceLoss	Masked dice loss.
MaskedSmoothL1Loss	Masked Smooth L1 loss.
MaskedSquareDiceLoss	Masked square dice loss.
MaskedBCEWithLogitsLoss	This loss combines a Sigmoid layers and a masked BCE loss in one single class.
SmoothL1Loss	Smooth L1 loss.
CrossEntropyLoss	Cross entropy loss.
MaskedBalancedBCELoss	Masked Balanced BCE loss.
MaskedBCELoss	Masked BCE loss.

MaskedBalancedBCEWithLogitsLoss

class mmocr.models.common.**MaskedBalancedBCEWithLogitsLoss**(*reduction='none'*, *negative_ratio=3*, *fallback_negative_num=0*, *eps=1e-06*)

This loss combines a Sigmoid layers and a masked balanced BCE loss in one single class. It's AMP-eligible.

Parameters

- **reduction** (*str, optional*) – The method to reduce the loss. Options are 'none', 'mean' and 'sum'. Defaults to 'none'.
- **negative_ratio** (*float or int, optional*) – Maximum ratio of negative samples to positive ones. Defaults to 3.
- **fallback_negative_num** (*int, optional*) – When the mask contains no positive samples, the number of negative samples to be sampled. Defaults to 0.
- **eps** (*float, optional*) – Eps to avoid zero-division error. Defaults to 1e-6.

Return type None

forward(*pred*, *gt*, *mask=None*)

Forward function.

Parameters

- **pred** (*torch.Tensor*) – The prediction in any shape.
- **gt** (*torch.Tensor*) – The learning target of the prediction in the same shape as pred.
- **mask** (*torch.Tensor, optional*) – Binary mask in the same shape of pred, indicating positive regions to calculate the loss. Whole region will be taken into account if not provided. Defaults to None.

Returns The loss value.

Return type torch.Tensor

MaskedDiceLoss

class mmocr.models.common.**MaskedDiceLoss**(*eps=1e-06*)
Masked dice loss.

Parameters **eps** (*float, optional*) – Eps to avoid zero-divison error. Defaults to 1e-6.

Return type None

forward(*pred*, *gt*, *mask=None*)
Forward function.

Parameters

- **pred** (*torch.Tensor*) – The prediction in any shape.
- **gt** (*torch.Tensor*) – The learning target of the prediction in the same shape as pred.
- **mask** (*torch.Tensor, optional*) – Binary mask in the same shape of pred, indicating positive regions to calculate the loss. Whole region will be taken into account if not provided. Defaults to None.

Returns The loss value.

Return type torch.Tensor

MaskedSmoothL1Loss

class mmocr.models.common.**MaskedSmoothL1Loss**(*beta=1*, *eps=1e-06*)
Masked Smooth L1 loss.

Parameters

- **beta** (*float, optional*) – The threshold in the piecewise function. Defaults to 1.
- **eps** (*float, optional*) – Eps to avoid zero-division error. Defaults to 1e-6.

Return type None

forward(*pred*, *gt*, *mask=None*)
Forward function.

Parameters

- **pred** (*torch.Tensor*) – The prediction in any shape.
- **gt** (*torch.Tensor*) – The learning target of the prediction in the same shape as pred.
- **mask** (*torch.Tensor, optional*) – Binary mask in the same shape of pred, indicating positive regions to calculate the loss. Whole region will be taken into account if not provided. Defaults to None.

Returns The loss value.

Return type torch.Tensor

MaskedSquareDiceLoss

class mmocr.models.common.**MaskedSquareDiceLoss**(*eps=0.001*)
Masked square dice loss.

Parameters **eps** (*float, optional*) – Eps to avoid zero-divison error. Defaults to 1e-3.

Return type None

forward(*pred*, *gt*, *mask=None*)
Forward function.

Parameters

- **pred** (*torch.Tensor*) – The prediction in any shape.
- **gt** (*torch.Tensor*) – The learning target of the prediction in the same shape as pred.
- **mask** (*torch.Tensor, optional*) – Binary mask in the same shape of pred, indicating positive regions to calculate the loss. Whole region will be taken into account if not provided. Defaults to None.

Returns The loss value.

Return type torch.Tensor

MaskedBCEWithLogitsLoss

class mmocr.models.common.**MaskedBCEWithLogitsLoss**(*eps=1e-06*)
This loss combines a Sigmoid layers and a masked BCE loss in one single class. It's AMP-eligible.

Parameters **eps** (*float*) – Eps to avoid zero-division error. Defaults to 1e-6.

Return type None

forward(*pred*, *gt*, *mask=None*)
Forward function.

Parameters

- **pred** (*torch.Tensor*) – The prediction in any shape.
- **gt** (*torch.Tensor*) – The learning target of the prediction in the same shape as pred.
- **mask** (*torch.Tensor, optional*) – Binary mask in the same shape of pred, indicating positive regions to calculate the loss. Whole region will be taken into account if not provided. Defaults to None.

Returns The loss value.

Return type torch.Tensor

SmoothL1Loss

class mmocr.models.common.**SmoothL1Loss**(*size_average=None*, *reduce=None*, *reduction='mean'*, *beta=1.0*)
Smooth L1 loss.

Parameters

- **reduction** (*str*) –
- **beta** (*float*) –

Return type None

CrossEntropyLoss

class mmocr.models.common.**CrossEntropyLoss**(*weight=None*, *size_average=None*, *ignore_index=- 100*, *reduce=None*, *reduction='mean'*, *label_smoothing=0.0*)
Cross entropy loss.

Parameters

- **weight** (*Optional[torch.Tensor]*) –
- **ignore_index** (*int*) –
- **reduction** (*str*) –
- **label_smoothing** (*float*) –

Return type None

MaskedBalancedBCELoss

class mmocr.models.common.**MaskedBalancedBCELoss**(*reduction='none'*, *negative_ratio=3*, *fallback_negative_num=0*, *eps=1e-06*)
Masked Balanced BCE loss.

Parameters

- **reduction** (*str, optional*) – The method to reduce the loss. Options are 'none', 'mean' and 'sum'. Defaults to 'none'.
- **negative_ratio** (*float or int*) – Maximum ratio of negative samples to positive ones. Defaults to 3.
- **fallback_negative_num** (*int*) – When the mask contains no positive samples, the number of negative samples to be sampled. Defaults to 0.
- **eps** (*float*) – Eps to avoid zero-division error. Defaults to 1e-6.

Return type None

forward(*pred*, *gt*, *mask=None*)
Forward function.

Parameters

- **pred** (*torch.Tensor*) – The prediction in any shape.
- **gt** (*torch.Tensor*) – The learning target of the prediction in the same shape as pred.

- **mask** (*torch.Tensor, optional*) – Binary mask in the same shape of pred, indicating positive regions to calculate the loss. Whole region will be taken into account if not provided. Defaults to None.

Returns The loss value.

Return type torch.Tensor

MaskedBCELoss

class mmocr.models.common.**MaskedBCELoss**(*eps=1e-06*)

Masked BCE loss.

Parameters **eps** (*float*) – Eps to avoid zero-division error. Defaults to 1e-6.

Return type None

forward(*pred*, *gt*, *mask=None*)

Forward function.

Parameters

- **pred** (*torch.Tensor*) – The prediction in any shape.
- **gt** (*torch.Tensor*) – The learning target of the prediction in the same shape as pred.
- **mask** (*torch.Tensor, optional*) – Binary mask in the same shape of pred, indicating positive regions to calculate the loss. Whole region will be taken into account if not provided. Defaults to None.

Returns The loss value.

Return type torch.Tensor

46.1.4 Layers

TFEncoderLayer	Transformer Encoder Layer.
TFDecoderLayer	Transformer Decoder Layer.

TFEncoderLayer

class mmocr.models.common.**TFEncoderLayer**(*d_model=512*, *d_inner=256*, *n_head=8*, *d_k=64*, *d_v=64*, *dropout=0.1*, *qkv_bias=False*, *act_cfg={'type': 'mmengine.GELU'}*, *operation_order=None*)

Transformer Encoder Layer.

Parameters

- **d_model** (*int*) – The number of expected features in the decoder inputs (default=512).
- **d_inner** (*int*) – The dimension of the feedforward network model (default=256).
- **n_head** (*int*) – The number of heads in the multiheadattention models (default=8).
- **d_k** (*int*) – Total number of features in key.
- **d_v** (*int*) – Total number of features in value.
- **dropout** (*float*) – Dropout layer on attn_output_weights.

- **qkv_bias** (*bool*) – Add bias in projection layer. Default: False.
- **act_cfg** (*dict*) – Activation cfg for feedforward module.
- **operation_order** (*tuple[str]*) – The execution order of operation in transformer. Such as ('self_attn', 'norm', 'ffn', 'norm') or ('norm', 'self_attn', 'norm', 'ffn'). DefaultNone.

forward(*x*, *mask=None*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

Note: Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

TFDecoderLayer

class mmocr.models.common.**TFDecoderLayer**(*d_model=512*, *d_inner=256*, *n_head=8*, *d_k=64*, *d_v=64*, *dropout=0.1*, *qkv_bias=False*, *act_cfg={'type': 'mmengine.GELU'}*, *operation_order=None*)

Transformer Decoder Layer.

Parameters

- **d_model** (*int*) – The number of expected features in the decoder inputs (default=512).
- **d_inner** (*int*) – The dimension of the feedforward network model (default=256).
- **n_head** (*int*) – The number of heads in the multiheadattention models (default=8).
- **d_k** (*int*) – Total number of features in key.
- **d_v** (*int*) – Total number of features in value.
- **dropout** (*float*) – Dropout layer on attn_output_weights.
- **qkv_bias** (*bool*) – Add bias in projection layer. Default: False.
- **act_cfg** (*dict*) – Activation cfg for feedforward module.
- **operation_order** (*tuple[str]*) – The execution order of operation in transformer. Such as ('self_attn', 'norm', 'enc_dec_attn', 'norm', 'ffn', 'norm') or ('norm', 'self_attn', 'norm', 'enc_dec_attn', 'norm', 'ffn'). DefaultNone.

forward(*dec_input*, *enc_output*, *self_attn_mask=None*, *dec_enc_attn_mask=None*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

Note: Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

46.1.5 Modules

ScaledDotProductAttention	Scaled Dot-Product Attention Module.
MultiHeadAttention	Multi-Head Attention module.
PositionwiseFeedForward	Two-layer feed-forward module.
PositionalEncoding	Fixed positional encoding with sine and cosine functions.

ScaledDotProductAttention

class mmocr.models.common.**ScaledDotProductAttention**(*temperature*, *attn_dropout=0.1*)

Scaled Dot-Product Attention Module. This code is adopted from https://github.com/jadore801120/attention-is-all-you-need-pytorch.

Parameters

- **temperature** (*float*) – The scale factor for softmax input.
- **attn_dropout** (*float*) – Dropout layer on attn_output_weights.

forward(*q*, *k*, *v*, *mask=None*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

Note: Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

MultiHeadAttention

class mmocr.models.common.**MultiHeadAttention**(*n_head=8*, *d_model=512*, *d_k=64*, *d_v=64*, *dropout=0.1*, *qkv_bias=False*)

Multi-Head Attention module.

Parameters

- **n_head** (*int*) – The number of heads in the multiheadattention models (default=8).
- **d_model** (*int*) – The number of expected features in the decoder inputs (default=512).
- **d_k** (*int*) – Total number of features in key.
- **d_v** (*int*) – Total number of features in value.
- **dropout** (*float*) – Dropout layer on attn_output_weights.
- **qkv_bias** (*bool*) – Add bias in projection layer. Default: False.

forward(*q*, *k*, *v*, *mask=None*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

Note: Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

PositionwiseFeedForward

class mmocr.models.common.**PositionwiseFeedForward**(*d_in*, *d_hid*, *dropout=0.1*, *act_cfg={'type': 'Relu'}*)
Two-layer feed-forward module.

Parameters

- **d_in** (*int*) – The dimension of the input for feedforward network model.
- **d_hid** (*int*) – The dimension of the feedforward network model.
- **dropout** (*float*) – Dropout layer on feedforward output.
- **act_cfg** (*dict*) – Activation cfg for feedforward module.

forward(*x*)
Defines the computation performed at every call.

Should be overridden by all subclasses.

Note: Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

PositionalEncoding

class mmocr.models.common.**PositionalEncoding**(*d_hid=512*, *n_position=200*, *dropout=0*)
Fixed positional encoding with sine and cosine functions.

forward(*x*)

Parameters **x** (*Tensor*) – Tensor of shape (batch_size, pos_len, d_hid, ...)

46.2 models.textdet

46.2.1 Detectors

SingleStageTextDetector	The class for implementing single stage text detector.
DBNet	The class for implementing DBNet text detector: Real-time Scene Text Detection with Differentiable Binarization.
PANet	The class for implementing PANet text detector:
PSENet	The class for implementing PSENet text detector: Shape Robust Text Detection with Progressive Scale Expansion Network.

continues on next page

Table 6 – continued from previous page

TextSnake	The class for implementing TextSnake text detector: TextSnake: A Flexible Representation for Detecting Text of Arbitrary Shapes.
FCENet	The class for implementing FCENet text detector FCENet(CVPR2021): Fourier Contour Embedding for Arbitrary-shaped Text Detection
DRRG	The class for implementing DRRG text detector.
MMDetWrapper	A wrapper of MMDet's model.

SingleStageTextDetector

class mmocr.models.textdet.**SingleStageTextDetector**(*backbone*, *det_head*, *neck=None*, *data_preprocessor=None*, *init_cfg=None*)

The class for implementing single stage text detector.

Single-stage text detectors directly and densely predict bounding boxes or polygons on the output features of the backbone + neck (optional).

Parameters

- **backbone** (*dict*) – Backbone config.
- **neck** (*dict, optional*) – Neck config. If None, the output from backbone will be directly fed into `det_head`.
- **det_head** (*dict*) – Head config.
- **data_preprocessor** (*dict, optional*) – Model preprocessing config for processing the input image data. Keys allowed are ``to_rgb``(bool), ``pad_size_divisor``(int), ``pad_value``(int or float), ``mean``(int or float) and ``std``(int or float). Preprcessing order: 1. to rgb; 2. normalization 3. pad. Defaults to None.
- **init_cfg** (*dict or list[dict], optional*) – Initialization configs. Defaults to None.

Return type None

extract_feat(*inputs*)

Extract features.

Parameters **inputs** (*Tensor*) – Image tensor with shape (N, C, H ,W).

Returns Multi-level features that may have different resolutions.

Return type Tensor or tuple[Tensor]

loss(*inputs*, *data_samples*)

Calculate losses from a batch of inputs and data samples.

Parameters

- **inputs** (*torch.Tensor*) – Input images of shape (N, C, H, W). Typically these should be mean centered and std scaled.
- **data_samples** (*list[TextDetDataSample]*) – A list of N datasamples, containing meta information and gold annotations for each of the images.

Returns A dictionary of loss components.

Return type dict[str, Tensor]

predict(*inputs*, *data_samples*)
Predict results from a batch of inputs and data samples with post- processing.

Parameters

- **inputs** (*torch.Tensor*) – Images of shape (N, C, H, W).
- **data_samples** (*list[TextDetDataSample]*) – A list of N datasamples, containing meta information and gold annotations for each of the images.

Returns

A list of N datasamples of prediction results. Each DetDataSample usually contain 'pred_instances'. And the `pred_instances` usually contains following keys.

- **scores (Tensor): Classification scores, has a shape** (num_instance,)
- **labels (Tensor): Labels of bboxes, has a shape** (num_instances,).
- **bboxes (Tensor): Has a shape (num_instances, 4),** the last dimension 4 arrange as (x1, y1, x2, y2).
- **polygons (list[np.ndarray]): The length is num_instances.** Each element represents the polygon of the instance, in (xn, yn) order.

Return type list[*TextDetDataSample*]

DBNet

class mmocr.models.textdet.**DBNet**(*backbone*, *det_head*, *neck=None*, *data_preprocessor=None*, *init_cfg=None*)
The class for implementing DBNet text detector: Real-time Scene Text Detection with Differentiable Binarization.

[https://arxiv.org/abs/1911.08947].

Parameters

- **backbone** (*Dict*) –
- **det_head** (*Dict*) –
- **neck** (*Optional[Dict]*) –
- **data_preprocessor** (*Optional[Dict]*) –
- **init_cfg** (*Optional[Dict]*) –

Return type None

PANet

class mmocr.models.textdet.**PANet**(*backbone*, *det_head*, *neck=None*, *data_preprocessor=None*, *init_cfg=None*)
The class for implementing PANet text detector:

Efficient and Accurate Arbitrary-Shaped Text Detection with Pixel Aggregation Network [https://arxiv.org/abs/1908.05900].

Parameters

- **backbone** (*Dict*) –
- **det_head** (*Dict*) –

- **neck** (*Optional[Dict]*) –
- **data_preprocessor** (*Optional[Dict]*) –
- **init_cfg** (*Optional[Dict]*) –

Return type None

PSENet

class mmocr.models.textdet.**PSENet**(*backbone*, *det_head*, *neck=None*, *data_preprocessor=None*, *init_cfg=None*)

The class for implementing PSENet text detector: Shape Robust Text Detection with Progressive Scale Expansion Network.

[https://arxiv.org/abs/1806.02559].

Parameters

- **backbone** (*Dict*) –
- **det_head** (*Dict*) –
- **neck** (*Optional[Dict]*) –
- **data_preprocessor** (*Optional[Dict]*) –
- **init_cfg** (*Optional[Dict]*) –

Return type None

TextSnake

class mmocr.models.textdet.**TextSnake**(*backbone*, *det_head*, *neck=None*, *data_preprocessor=None*, *init_cfg=None*)

The class for implementing TextSnake text detector: TextSnake: A Flexible Representation for Detecting Text of Arbitrary Shapes.

[https://arxiv.org/abs/1807.01544]

Parameters

- **backbone** (*Dict*) –
- **det_head** (*Dict*) –
- **neck** (*Optional[Dict]*) –
- **data_preprocessor** (*Optional[Dict]*) –
- **init_cfg** (*Optional[Dict]*) –

Return type None

FCENet

class mmocr.models.textdet.**FCENet**(*backbone*, *det_head*, *neck=None*, *data_preprocessor=None*, *init_cfg=None*)

The class for implementing FCENet text detector FCENet(CVPR2021): Fourier Contour Embedding for Arbitrary-shaped Text

Detection

[https://arxiv.org/abs/2104.10442]

Parameters

- **backbone** (*Dict*) –
- **det_head** (*Dict*) –
- **neck** (*Optional[Dict]*) –
- **data_preprocessor** (*Optional[Dict]*) –
- **init_cfg** (*Optional[Dict]*) –

Return type None

DRRG

class mmocr.models.textdet.**DRRG**(*backbone*, *det_head*, *neck=None*, *data_preprocessor=None*, *init_cfg=None*)

The class for implementing DRRG text detector. Deep Relational Reasoning Graph Network for Arbitrary Shape Text Detection.

[https://arxiv.org/abs/2003.07493]

Parameters

- **backbone** (*Dict*) –
- **det_head** (*Dict*) –
- **neck** (*Optional[Dict]*) –
- **data_preprocessor** (*Optional[Dict]*) –
- **init_cfg** (*Optional[Dict]*) –

Return type None

MMDetWrapper

class mmocr.models.textdet.**MMDetWrapper**(*cfg*, *text_repr_type='poly'*)

A wrapper of MMDet's model.

Parameters

- **cfg** (*dict*) – The config of the model.
- **text_repr_type** (*str*) – The boundary encoding type 'poly' or 'quad'. Defaults to 'poly'.

Return type None

adapt_predictions(*data*, *data_samples*)

Convert Instance datas from MMDet into MMOCR's format.

Parameters

- **data** (*List[mmdet.structures.det_data_sample.DetDataSample]*) – (list[DetDataSample]): Detection results of the input images. Each DetDataSample usually contain 'pred_instances'. And the `pred_instances` usually contains following keys. - scores (Tensor): Classification scores, has a shape

  (num_instance,)

  - **labels (Tensor): Labels of bboxes, has a shape** (num_instances,).
  - **bboxes (Tensor): Has a shape (num_instances, 4),** the last dimension 4 arrange as (x1, y1, x2, y2).
  - masks (Tensor, Optional): Has a shape (num_instances, H, W).

- **data_samples** (list[`TextDetDataSample`]) – The annotation data of every samples.

Returns

A list of N datasamples containing ground truth and prediction results. The polygon results are saved in `TextDetDataSample.pred_instances.polygons` The confidence scores are saved in `TextDetDataSample.pred_instances.scores`.

Return type list[*TextDetDataSample*]

forward(*inputs*, *data_samples=None*, *mode='tensor'*, ***kwargs*)

The unified entry for a forward process in both training and test.

The method works in three modes: "tensor", "predict" and "loss":

- "tensor": Forward the whole network and return tensor or tuple of

tensor without any post-processing, same as a common nn.Module. - "predict": Forward and return the predictions, which are fully processed to a list of `DetDataSample`. - "loss": Forward and return a dict of losses according to the given inputs and data samples.

Note that this method doesn't handle either back propagation or parameter update, which are supposed to be done in `train_step()`.

Parameters

- **inputs** (*torch.Tensor*) – The input tensor with shape (N, C, ...) in general.
- **data_samples** (*Optional[Union[List[mmocr.structures.textdet_data_sample.TextDetDataSample], List[mmdet.structures.det_data_sample.DetDataSample]]]*) –
- **mode** (*str*) –

Return type Union[Dict[str, torch.Tensor], List[mmdet.structures.det_data_sample.DetDataSample], Tuple[torch.Tensor], torch.Tensor]

:param data_samples (list[DetDataSample] or: list[TextDetDataSample]): The annotation data of every sample. When in "predict" mode, it should be a list of `TextDetDataSample`. Otherwise they are :obj:`DetDataSample`s. Defaults to None.

Parameters

- **mode** (*str*) – Running mode. Defaults to 'tensor'.
- **inputs** (*torch.Tensor*) –

- **data_samples** (*Optional[Union[List[mmocr.structures.textdet_data_sample.TextDetDataSample], List[mmdet.structures.det_data_sample.DetDataSample]]]*) –

Returns

The return type depends on `mode`.

- If `mode="tensor"`, return a tensor or a tuple of tensor.
- If `mode="predict"`, return a list of `TextDetDataSample`.
- If `mode="loss"`, return a dict of tensor.

Return type Union[Dict[str, torch.Tensor], List[mmdet.structures.det_data_sample.DetDataSample], Tuple[torch.Tensor], torch.Tensor]

46.2.2 Data Preprocessors

TextDetDataPreprocessor	Image pre-processor for detection tasks.

TextDetDataPreprocessor

class mmocr.models.textdet.**TextDetDataPreprocessor**(*mean=None*, *std=None*, *pad_size_divisor=1*, *pad_value=0*, *bgr_to_rgb=False*, *rgb_to_bgr=False*, *batch_augments=None*)

Image pre-processor for detection tasks.

Comparing with the `mmengine.ImgDataPreprocessor`,

1. It supports batch augmentations.

2. It will additionally append batch_input_shape and pad_shape to data_samples considering the object detection task.

It provides the data pre-processing as follows

- Collate and move data to the target device.
- Pad inputs to the maximum size of current batch with defined `pad_value`. The padding size can be divisible by a defined `pad_size_divisor`
- Stack inputs to batch_inputs.
- Convert inputs from bgr to rgb if the shape of input is (3, H, W).
- Normalize image with defined std and mean.
- Do batch augmentations during training.

Parameters

- **mean** (*Sequence[Number], optional*) – The pixel mean of R, G, B channels. Defaults to None.
- **std** (*Sequence[Number], optional*) – The pixel standard deviation of R, G, B channels. Defaults to None.
- **pad_size_divisor** (*int*) – The size of padded image should be divisible by `pad_size_divisor`. Defaults to 1.

- **pad_value** (*Number*) – The padded pixel value. Defaults to 0.
- **pad_mask** (*bool*) – Whether to pad instance masks. Defaults to False.
- **mask_pad_value** (*int*) – The padded pixel value for instance masks. Defaults to 0.
- **pad_seg** (*bool*) – Whether to pad semantic segmentation maps. Defaults to False.
- **seg_pad_value** (*int*) – The padded pixel value for semantic segmentation maps. Defaults to 255.
- **bgr_to_rgb** (*bool*) – whether to convert image from BGR to RGB. Defaults to False.
- **rgb_to_bgr** (*bool*) – whether to convert image from RGB to RGB. Defaults to False.
- **batch_augments** (*list[dict], optional*) – Batch-level augmentations

Return type None

forward(*data*, *training=False*)

Perform normalizationpadding and bgr2rgb conversion based on `BaseDataPreprocessor`.

Parameters

- **data** (*dict*) – data sampled from dataloader.
- **training** (*bool*) – Whether to enable training time augmentation.

Returns Data in the same format as the model input.

Return type dict

46.2.3 Necks

FPEM_FFM	This code is from https://github.com/WenmuZhou/PAN.pytorch.
FPNF	FPN-like fusion module in Shape Robust Text Detection with Progressive Scale Expansion Network.
FPNC	FPN-like fusion module in Real-time Scene Text Detection with Differentiable Binarization.
FPN_UNet	The class for implementing DRRG and TextSnake U-Net-like FPN.

FPEM_FFM

class mmocr.models.textdet.**FPEM_FFM**(*in_channels*, *conv_out=128*, *fpem_repeat=2*, *align_corners=False*, *init_cfg={'distribution': 'uniform', 'layer': 'Conv2d', 'type': 'Xavier'}*)

This code is from https://github.com/WenmuZhou/PAN.pytorch.

Parameters

- **in_channels** (*list[int]*) – A list of 4 numbers of input channels.
- **conv_out** (*int*) – Number of output channels.
- **fpem_repeat** (*int*) – Number of FPEM layers before FFM operations.
- **align_corners** (*bool*) – The interpolation behaviour in FFM operation, used in `torch.nn.functional.interpolate()`.
- **init_cfg** (*dict or list[dict], optional*) – Initialization configs.

Return type None

forward(*x*)

Parameters **x** (*list[Tensor]*) – A list of four tensors of shape (N, C_i, H_i, W_i), representing C2, C3, C4, C5 features respectively. C_i should matches the number in `in_channels`.

Returns Four tensors of shape (N, C_{out}, H_0, W_0) where C_{out} is `conv_out`.

Return type tuple[Tensor]

FPNF

class mmocr.models.textdet.**FPNF**(*in_channels=[256, 512, 1024, 2048]*, *out_channels=256*, *fusion_type='concat'*, *init_cfg={'distribution': 'uniform', 'layer': 'Conv2d', 'type': 'Xavier'}*)

FPN-like fusion module in Shape Robust Text Detection with Progressive Scale Expansion Network.

Parameters

- **in_channels** (*list[int]*) – A list of number of input channels. Defaults to [256, 512, 1024, 2048].
- **out_channels** (*int*) – The number of output channels. Defaults to 256.
- **fusion_type** (*str*) – Type of the final feature fusion layer. Available options are "concat" and "add". Defaults to "concat".
- **init_cfg** (*dict or list[dict], optional*) – Initialization configs. Defaults to dict(type='Xavier', layer='Conv2d', distribution='uniform')

Return type None

forward(*inputs*)

Parameters **inputs** (*list[Tensor]*) – Each tensor has the shape of (N, C_i, H_i, W_i). It usually expects 4 tensors (C2-C5 features) from ResNet.

Returns A tensor of shape (N, C_{out}, H_0, W_0) where C_{out} is `out_channels`.

Return type Tensor

FPNC

class mmocr.models.textdet.**FPNC**(*in_channels*, *lateral_channels=256*, *out_channels=64*, *bias_on_lateral=False*, *bn_re_on_lateral=False*, *bias_on_smooth=False*, *bn_re_on_smooth=False*, *asf_cfg=None*, *conv_after_concat=False*, *init_cfg=[{'type': 'Kaiming', 'layer': 'Conv'}, {'type': 'Constant', 'layer': 'BatchNorm', 'val': 1.0, 'bias': 0.0001}]*)

FPN-like fusion module in Real-time Scene Text Detection with Differentiable Binarization.

This was partially adapted from https://github.com/MhLiao/DB and https://github.com/WenmuZhou/DBNet.pytorch.

Parameters

- **in_channels** (*list[int]*) – A list of numbers of input channels.
- **lateral_channels** (*int*) – Number of channels for lateral layers.

- **out_channels** (*int*) – Number of output channels.
- **bias_on_lateral** (*bool*) – Whether to use bias on lateral convolutional layers.
- **bn_re_on_lateral** (*bool*) – Whether to use BatchNorm and ReLU on lateral convolutional layers.
- **bias_on_smooth** (*bool*) – Whether to use bias on smoothing layer.
- **bn_re_on_smooth** (*bool*) – Whether to use BatchNorm and ReLU on smoothing layer.
- **asf_cfg** (*dict, optional*) – Adaptive Scale Fusion module configs. The attention_type can be 'ScaleChannelSpatial'.
- **conv_after_concat** (*bool*) – Whether to add a convolution layer after the concatenation of predictions.
- **init_cfg** (*dict or list[dict], optional*) – Initialization configs.

Return type None

forward(*inputs*)

Parameters **inputs** (*list[Tensor]*) – Each tensor has the shape of (N, C_i, H_i, W_i). It usually expects 4 tensors (C2-C5 features) from ResNet.

Returns A tensor of shape (N, C_{out}, H_0, W_0) where C_{out} is `out_channels`.

Return type Tensor

FPN_UNet

class mmocr.models.textdet.**FPN_UNet**(*in_channels*, *out_channels*, *init_cfg={'distribution': 'uniform', 'layer': ['Conv2d', 'ConvTranspose2d'], 'type': 'Xavier'}*)

The class for implementing DRRG and TextSnake U-Net-like FPN.

DRRG: Deep Relational Reasoning Graph Network for Arbitrary Shape Text Detection.

TextSnake: A Flexible Representation for Detecting Text of Arbitrary Shapes.

Parameters

- **in_channels** (*list[int]*) – Number of input channels at each scale. The length of the list should be 4.
- **out_channels** (*int*) – The number of output channels.
- **init_cfg** (*dict or list[dict], optional*) – Initialization configs.

Return type None

forward(*x*)

Parameters **x** (*list[Tensor] | tuple[Tensor]*) – A list of four tensors of shape (N, C_i, H_i, W_i), representing C2, C3, C4, C5 features respectively. C_i should matches the number in `in_channels`.

Returns Shape (N, C, H, W) where $H = 4H_0$ and $W = 4W_0$.

Return type Tensor

46.2.4 Heads

BaseTextDetHead	Base head for text detection, build the loss and postprocessor.
PSEHead	The class for PSENet head.
PANHead	The class for PANet head.
DBHead	The class for DBNet head.
FCEHead	The class for implementing FCENet head.
TextSnakeHead	The class for TextSnake head: TextSnake: A Flexible Representation for Detecting Text of Arbitrary Shapes.
DRRGHead	The class for DRRG head: Deep Relational Reasoning Graph Network for Arbitrary Shape Text Detection.

BaseTextDetHead

class mmocr.models.textdet.**BaseTextDetHead**(*module_loss=None*, *postprocessor=None*, *init_cfg=None*)

Base head for text detection, build the loss and postprocessor.

1. The `init_weights` method is used to initialize head's model parameters. After detector initialization, `init_weights` is triggered when `detector.init_weights()` is called externally.

2. The `loss` method is used to calculate the loss of head, which includes two steps: (1) the head model performs forward propagation to obtain the feature maps (2) The `module_loss` method is called based on the feature maps to calculate the loss.

```
loss(): forward() -> module_loss()
```

3. The `predict` method is used to predict detection results, which includes two steps: (1) the head model performs forward propagation to obtain the feature maps (2) The `postprocessor` method is called based on the feature maps to predict detection results including post-processing.

```
predict(): forward() -> postprocessor()
```

4. The `loss_and_predict` method is used to return loss and detection results at the same time. It will call head's `forward`, `module_loss` and `postprocessor` methods in order.

```
loss_and_predict(): forward() -> module_loss() -> postprocessor()
```

Parameters

- **loss** (*dict, optional*) – Config to build loss. Defaults to None.
- **postprocessor** (*dict, optional*) – Config to build postprocessor. Defaults to None.
- **init_cfg** (*dict or list[dict], optional*) – Initialization configs. Defaults to None.
- **module_loss** (*Optional[Dict]*) –

Return type None

loss(*x*, *data_samples*)

Perform forward propagation and loss calculation of the detection head on the features of the upstream network.

Parameters

- **x** (*tuple[Tensor]*) – Features from the upstream network, each is a 4D-tensor.
- **data_samples** (List[DetDataSample]) – The Data Samples. It usually includes information such as *gt_instance*, *gt_panoptic_seg* and *gt_sem_seg*.

Returns A dictionary of loss components.

Return type dict

loss_and_predict(*x*, *data_samples*)

Perform forward propagation of the head, then calculate loss and predictions from the features and data samples.

Parameters

- **x** (*tuple[Tensor]*) – Features from FPN.
- **data_samples** (list[DetDataSample]) – Each item contains the meta information of each image and corresponding annotations.

Returns

the return value is a tuple contains:

- losses: (dict[str, Tensor]): A dictionary of loss components.
- predictions (list[InstanceData]): Detection results of each image after the post process.

Return type tuple

predict(*x*, *data_samples*)

Perform forward propagation of the detection head and predict detection results on the features of the upstream network.

Parameters

- **x** (*tuple[Tensor]*) – Multi-level features from the upstream network, each is a 4D-tensor.
- **data_samples** (List[DetDataSample]) – The Data Samples. It usually includes information such as *gt_instance*, *gt_panoptic_seg* and *gt_sem_seg*.

Returns Detection results of each image after the post process.

Return type SampleList

PSEHead

class mmocr.models.textdet.**PSEHead**(*in_channels*, *hidden_dim*, *out_channel*, *module_loss={'type': 'PSEModuleLoss'}*, *postprocessor={'text_repr_type': 'poly', 'type': 'PSEPostprocessor'}*, *init_cfg=None*)

The class for PSENet head.

Parameters

- **in_channels** (*list[int]*) – A list of numbers of input channels.
- **hidden_dim** (*int*) – The hidden dimension of the first convolutional layer.
- **out_channel** (*int*) – Number of output channels.
- **module_loss** (*dict*) – Configuration dictionary for loss type. Supported loss types are "PANModuleLoss" and "PSEModuleLoss". Defaults to PSEModuleLoss.

- **postprocessor** (*dict*) – Config of postprocessor for PSENet.
- **init_cfg** (*dict or list[dict], optional*) – Initialization configs.

Return type None

PANHead

class mmocr.models.textdet.**PANHead**(*in_channels*, *hidden_dim*, *out_channel*, *module_loss={'type': 'PANModuleLoss'}*, *postprocessor={'text_repr_type': 'poly', 'type': 'PANPostprocessor'}*, *init_cfg=[{'type': 'Normal', 'mean': 0, 'std': 0.01, 'layer': 'Conv2d'}, {'type': 'Constant', 'val': 1, 'bias': 0, 'layer': 'BN'}]*)

The class for PANet head.

Parameters

- **in_channels** (*list[int]*) – A list of 4 numbers of input channels.
- **hidden_dim** (*int*) – The hidden dimension of the first convolutional layer.
- **out_channel** (*int*) – Number of output channels.
- **module_loss** (*dict*) – Configuration dictionary for loss type. Defaults to dict(type='PANModuleLoss')
- **postprocessor** (*dict*) – Config of postprocessor for PANet. Defaults to dict(type='PANPostprocessor', text_repr_type='poly').
- **init_cfg** (*list[dict]*) – Initialization configs. Defaults to [dict(type='Normal', mean=0, std=0.01, layer='Conv2d'),

  dict(type='Constant', val=1, bias=0, layer='BN')]

Return type None

forward(*inputs*, *data_samples=None*)

PAN head forward. :param inputs: Each tensor has the shape of

(N, C_i, W, H), where $\sum_i C_i = C_{in}$ and C_{in} is `input_channels`.

Parameters

- **data_samples** (*list[TextDetDataSample], optional*) – A list of data samples. Defaults to None.
- **inputs** (*list[Tensor] | Tensor*) –

Returns A tensor of shape (N, C_{out}, W, H) where C_{out} is `output_channels`.

Return type Tensor

DBHead

class mmocr.models.textdet.**DBHead**(*in_channels*, *with_bias=False*, *module_loss={'type': 'DBModuleLoss'}*, *postprocessor={'text_repr_type': 'quad', 'type': 'DBPostprocessor'}*, *init_cfg=[{'type': 'Kaiming', 'layer': 'Conv'}, {'type': 'Constant', 'layer': 'BatchNorm', 'val': 1.0, 'bias': 0.0001}]*)

The class for DBNet head.

This was partially adapted from https://github.com/MhLiao/DB

Parameters

- **in_channels** (*int*) – The number of input channels.
- **with_bias** (*bool*) – Whether add bias in Conv2d layer. Defaults to False.
- **module_loss** (*dict*) – Config of loss for dbnet. Defaults to `dict(type='DBModuleLoss')`
- **postprocessor** (*dict*) – Config of postprocessor for dbnet.
- **init_cfg** (*dict or list[dict], optional*) – Initialization configs.

Return type None

forward(*img*, *data_samples=None*, *mode='predict'*)

Parameters

- **img** (*Tensor*) – Shape (N, C, H, W).
- **data_samples** (*list[TextDetDataSample], optional*) – A list of data samples. Defaults to None.
- **mode** (*str*) – Forward mode. It affects the return values. Options are "loss", "predict" and "both". Defaults to "predict".
  - `loss`: Run the full network and return the prob logits, threshold map and binary map.
  - `predict`: Run the binarzation part and return the prob map only.
  - `both`: Run the full network and return prob logits, threshold map, binary map and prob map.

Returns Its type depends on `mode`, read its docstring for details. Each has the shape of $(N, 4H, 4W)$.

Return type Tensor or tuple(Tensor)

loss(*x*, *batch_data_samples*)

Perform forward propagation and loss calculation of the detection head on the features of the upstream network.

Parameters

- **x** (*tuple[Tensor]*) – Features from the upstream network, each is a 4D-tensor.
- **batch_data_samples** (List[`DetDataSample`]) – The Data Samples. It usually includes information such as *gt_instance*, *gt_panoptic_seg* and *gt_sem_seg*.

Returns A dictionary of loss components.

Return type dict

loss_and_predict(*x*, *batch_data_samples*)

Perform forward propagation of the head, then calculate loss and predictions from the features and data samples.

Parameters

- **x** (*tuple[Tensor]*) – Features from FPN.
- **batch_data_samples** (list[`DetDataSample`]) – Each item contains the meta information of each image and corresponding annotations.

Returns

the return value is a tuple contains:

- losses: (dict[str, Tensor]): A dictionary of loss components.
- predictions (list[`InstanceData`]): Detection results of each image after the post process.

Return type tuple

predict(*x*, *batch_data_samples*)

Perform forward propagation of the detection head and predict detection results on the features of the upstream network.

Parameters

- **x** (*tuple[Tensor]*) – Multi-level features from the upstream network, each is a 4D-tensor.
- **batch_data_samples** (List[`DetDataSample`]) – The Data Samples. It usually includes information such as *gt_instance*, *gt_panoptic_seg* and *gt_sem_seg*.

Returns Detection results of each image after the post process.

Return type SampleList

FCEHead

class mmocr.models.textdet.**FCEHead**(*in_channels*, *fourier_degree=5*, *module_loss={'num_sample': 50, 'type': 'FCEModuleLoss'}*, *postprocessor={'alpha': 1.0, 'beta': 2.0, 'num_reconstr_points': 50, 'score_thr': 0.3, 'text_repr_type': 'poly', 'type': 'FCEPostprocessor'}*, *init_cfg={'mean': 0, 'override': [{'name': 'out_conv_cls'}, {'name': 'out_conv_reg'}], 'std': 0.01, 'type': 'Normal'}*)

The class for implementing FCENet head.

FCENet(CVPR2021): Fourier Contour Embedding for Arbitrary-shaped Text Detection

Parameters

- **in_channels** (*int*) – The number of input channels.
- **fourier_degree** (*int*) – The maximum Fourier transform degree k. Defaults to 5.
- **module_loss** (*dict*) – Config of loss for FCENet. Defaults to `dict(type='FCEModuleLoss', num_sample=50)`.
- **postprocessor** (*dict*) – Config of postprocessor for FCENet.
- **init_cfg** (*dict, optional*) – Initialization configs.

Return type None

forward(*inputs*, *data_samples=None*)

Parameters

- **inputs** (*List[Tensor]*) – Each tensor has the shape of (N, C_i, H_i, W_i).
- **data_samples** (*list[TextDetDataSample], optional*) – A list of data samples. Defaults to None.

Returns A list of dict with keys of `cls_res`, `reg_res` corresponds to the classification result and regression result computed from the input tensor with the same index. They have the shapes of $(N, C_{cls,i}, H_i, W_i)$ and $(N, C_{out,i}, H_i, W_i)$.

Return type list[dict]

forward_single(*x*)

Forward function for a single feature level.

Parameters **x** (*Tensor*) – The input tensor with the shape of (N, C_i, H_i, W_i).

Returns The classification and regression result with the shape of $(N, C_{cls,i}, H_i, W_i)$ and $(N, C_{out,i}, H_i, W_i)$.

Return type Tensor

TextSnakeHead

class mmocr.models.textdet.**TextSnakeHead**(*in_channels*, *out_channels=5*, *downsample_ratio=1.0*, *module_loss={'type': 'TextSnakeModuleLoss'}*, *postprocessor={'text_repr_type': 'poly', 'type': 'TextSnakePostprocessor'}*, *init_cfg={'mean': 0, 'override': {'name': 'out_conv'}, 'std': 0.01, 'type': 'Normal'}*)

The class for TextSnake head: TextSnake: A Flexible Representation for Detecting Text of Arbitrary Shapes.

TextSnake: A Flexible Representation for Detecting Text of Arbitrary Shapes.

Parameters

- **in_channels** (*int*) – Number of input channels.
- **out_channels** (*int*) – Number of output channels.
- **downsample_ratio** (*float*) – Downsample ratio.
- **module_loss** (*dict*) – Configuration dictionary for loss type. Defaults to `dict(type='TextSnakeModuleLoss')`.
- **postprocessor** (*dict*) – Config of postprocessor for TextSnake.
- **init_cfg** (*dict or list[dict], optional*) – Initialization configs.

Return type None

forward(*inputs*, *data_samples=None*)

Parameters

- **inputs** (*torch.Tensor*) – Shape (N, C_{in}, H, W), where C_{in} is `in_channels`. H and W should be the same as the input of backbone.
- **data_samples** (*list[TextDetDataSample], optional*) – A list of data samples. Defaults to None.

Returns A tensor of shape $(N, 5, H, W)$, where the five channels represent [0]: text score, [1]: center score, [2]: sin, [3] cos, [4] radius, respectively.

Return type Tensor

DRRGHead

class mmocr.models.textdet.**DRRGHead**(*in_channels*, *k_at_hops=(8, 4)*, *num_adjacent_linkages=3*, *node_geo_feat_len=120*, *pooling_scale=1.0*, *pooling_output_size=(4, 3)*, *nms_thr=0.3*, *min_width=8.0*, *max_width=24.0*, *comp_shrink_ratio=1.03*, *comp_ratio=0.4*, *comp_score_thr=0.3*, *text_region_thr=0.2*, *center_region_thr=0.2*, *center_region_area_thr=50*, *local_graph_thr=0.7*, *module_loss={'type': 'DRRGModuleLoss'}*, *postprocessor={'link_thr': 0.85, 'type': 'DRRGPostprocessor'}*, *init_cfg={'mean': 0, 'override': {'name': 'out_conv'}, 'std': 0.01, 'type': 'Normal'}*)

The class for DRRG head: Deep Relational Reasoning Graph Network for Arbitrary Shape Text Detection.

Parameters

- **in_channels** (*int*) – The number of input channels.
- **k_at_hops** (*tuple(int)*) – The number of i-hop neighbors, i = 1, 2. Defaults to (8, 4).
- **num_adjacent_linkages** (*int*) – The number of linkages when constructing adjacent matrix. Defaults to 3.
- **node_geo_feat_len** (*int*) – The length of embedded geometric feature vector of a component. Defaults to 120.
- **pooling_scale** (*float*) – The spatial scale of rotated RoI-Align. Defaults to 1.0.
- **pooling_output_size** (*tuple(int)*) – The output size of RRoI-Aligning. Defaults to (4, 3).
- **nms_thr** (*float*) – The locality-aware NMS threshold of text components. Defaults to 0.3.
- **min_width** (*float*) – The minimum width of text components. Defaults to 8.0.
- **max_width** (*float*) – The maximum width of text components. Defaults to 24.0.
- **comp_shrink_ratio** (*float*) – The shrink ratio of text components. Defaults to 1.03.
- **comp_ratio** (*float*) – The reciprocal of aspect ratio of text components. Defaults to 0.4.
- **comp_score_thr** (*float*) – The score threshold of text components. Defaults to 0.3.
- **text_region_thr** (*float*) – The threshold for text region probability map. Defaults to 0.2.
- **center_region_thr** (*float*) – The threshold for text center region probability map. Defaults to 0.2.
- **center_region_area_thr** (*int*) – The threshold for filtering small-sized text center region. Defaults to 50.
- **local_graph_thr** (*float*) – The threshold to filter identical local graphs. Defaults to 0.7.
- **module_loss** (*dict*) – The config of loss that DRRGHead uses. Defaults to `dict(type='DRRGModuleLoss')`.
- **postprocessor** (*dict*) – Config of postprocessor for Drrg. Defaults to `dict(type='DrrgPostProcessor', link_thr=0.85)`.

- **init_cfg** (*dict or list[dict], optional*) – Initialization configs. Defaults to dict(type='Normal', override=dict(name='out_conv'), mean=0, std=0.01).

Return type None

forward(*inputs*, *data_samples=None*)

Run DRRG head in prediction mode, and return the raw tensors only. :param inputs: Shape of $(1, C, H, W)$. :type inputs: Tensor :param data_samples: A list of data

samples. Defaults to None.

Returns

Returns (edge, score, text_comps).

- edge (ndarray): The edge array of shape $(N_{edges}, 2)$ where each row is a pair of text component indices that makes up an edge in graph.
- score (ndarray): The score array of shape $(N_{edges},)$, corresponding to the edge above.
- text_comps (ndarray): The text components of shape $(M, 9)$ where each row corresponds to one box and its score: (x1, y1, x2, y2, x3, y3, x4, y4, score).

Return type tuple

Parameters

- **inputs** (*torch.Tensor*) –
- **data_samples** (*list[TextDetDataSample], optional*) –

loss(*inputs*, *data_samples*)

Loss function.

Parameters

- **inputs** (*Tensor*) – Shape of (N, C, H, W).
- **data_samples** (*List[TextDetDataSample]*) – List of data samples.

Returns

- **pred_maps (Tensor): Prediction map with shape** $(N, 6, H, W)$.
- **gcn_pred (Tensor): Prediction from GCN module, with** shape $(N, 2)$.
- **gt_labels (Tensor): Ground-truth label of shape** (m, n) where $m * n = N$.

Return type tuple(pred_maps, gcn_pred, gt_labels)

46.2.5 Module Losses

SegBasedModuleLoss	Base class for the module loss of segmentation-based text detection algorithms with some handy utilities.
PANModuleLoss	The class for implementing PANet loss.
PSEModuleLoss	The class for implementing PSENet loss.
DBModuleLoss	The class for implementing DBNet loss.
TextSnakeModuleLoss	The class for implementing TextSnake loss.
FCEModuleLoss	The class for implementing FCENet loss.
DRRGModuleLoss	The class for implementing DRRG loss.

SegBasedModuleLoss

class mmocr.models.textdet.**SegBasedModuleLoss**

Base class for the module loss of segmentation-based text detection algorithms with some handy utilities.

Return type None

PANModuleLoss

class mmocr.models.textdet.**PANModuleLoss**(*loss_text={'type': 'MaskedSquareDiceLoss'}*, *loss_kernel={'type': 'MaskedSquareDiceLoss'}*, *loss_embedding={'type': 'PANEmbLossV1'}*, *weight_text=1.0*, *weight_kernel=0.5*, *weight_embedding=0.25*, *ohem_ratio=3*, *shrink_ratio=(1.0, 0.5)*, *max_shrink_dist=20*, *reduction='mean'*)

The class for implementing PANet loss. This was partially adapted from https://github.com/whai362/pan_pp.pytorch and https://github.com/WenmuZhou/PAN.pytorch.

PANet: Efficient and Accurate Arbitrary- Shaped Text Detection with Pixel Aggregation Network.

Parameters

- **loss_text** (*dict*) – dict(type='MaskedSquareDiceLoss').
- **loss_kernel** (*dict*) – dict(type='MaskedSquareDiceLoss').
- **loss_embedding** (*dict*) – dict(type='PANEmbLossV1').
- **weight_text** (*float*) – The weight of text loss. Defaults to 1.
- **weight_kernel** (*float*) – The weight of kernel loss. Defaults to 0.5.
- **weight_embedding** (*float*) – The weight of embedding loss. Defaults to 0.25.
- **ohem_ratio** (*float*) – The negative/positive ratio in ohem. Defaults to 3.
- **shrink_ratio** (*tuple[float]*) – The ratio of shrinking kernel. Defaults to (1.0, 0.5).
- **max_shrink_dist** (*int or float*) – The maximum shrinking distance. Defaults to 20.
- **reduction** (*str*) – The way to reduce the loss. Available options are "mean" and "sum". Defaults to 'mean'.

Return type None

forward(*preds*, *data_samples*)

Compute PAN loss.

Parameters

- **preds** (*dict*) – Raw predictions from model with shape (N, C, H, W).
- **data_samples** (*list[TextDetDataSample]*) – The data samples.

Returns The dict for pan losses with loss_text, loss_kernel, loss_aggregation and loss_discrimination.

Return type dict

get_targets(*data_samples*)

Generate the gt targets for PANet.

Parameters

- **results** (*dict*) – The input result dictionary.
- **data_samples** (*Sequence[mmocr.structures.textdet_data_sample.TextDetDataSample]*) –

Returns The output result dictionary.

Return type results (dict)

PSEModuleLoss

class mmocr.models.textdet.**PSEModuleLoss**(*weight_text=0.7*, *weight_kernel=0.3*, *loss_text={'type': 'MaskedSquareDiceLoss'}*, *loss_kernel={'type': 'MaskedSquareDiceLoss'}*, *ohem_ratio=3*, *reduction='mean'*, *kernel_sample_type='adaptive'*, *shrink_ratio=(1.0, 0.9, 0.8, 0.7, 0.6, 0.5, 0.4)*, *max_shrink_dist=20*)

The class for implementing PSENet loss. This is partially adapted from https://github.com/whai362/PSENet.

PSENet: Shape Robust Text Detection with Progressive Scale Expansion Network.

Parameters

- **weight_text** (*float*) – The weight of text loss. Defaults to 0.7.
- **weight_kernel** (*float*) – The weight of text kernel. Defaults to 0.3.
- **loss_text** (*dict*) – Loss type for text. Defaults to dict('MaskedSquareDiceLoss').
- **loss_kernel** (*dict*) – Loss type for kernel. Defaults to dict('MaskedSquareDiceLoss').
- **ohem_ratio** (*int or float*) – The negative/positive ratio in ohem. Defaults to 3.
- **reduction** (*str*) – The way to reduce the loss. Defaults to 'mean'. Options are 'mean' and 'sum'.
- **kernel_sample_type** (*str*) – The way to sample kernel. Defaults to adaptive. Options are 'adaptive' and 'hard'.
- **shrink_ratio** (*tuple*) – The ratio for shirinking text instances. Defaults to (1.0, 0.9, 0.8, 0.7, 0.6, 0.5, 0.4).
- **max_shrink_dist** (*int or float*) – The maximum shrinking distance. Defaults to 20.

Return type None

forward(*preds*, *data_samples*)

Compute PSENet loss.

Parameters

- **preds** (*torch.Tensor*) – Raw predictions from model with shape (N, C, H, W).
- **data_samples** (*list[TextDetDataSample]*) – The data samples.

Returns The dict for pse losses with loss_text, loss_kernel, loss_aggregation and loss_discrimination.

Return type dict

DBModuleLoss

class mmocr.models.textdet.**DBModuleLoss**(*loss_prob={'type': 'MaskedBalancedBCEWithLogitsLoss'}*, *loss_thr={'beta': 0, 'type': 'MaskedSmoothL1Loss'}*, *loss_db={'type': 'MaskedDiceLoss'}*, *weight_prob=5.0*, *weight_thr=10.0*, *shrink_ratio=0.4*, *thr_min=0.3*, *thr_max=0.7*, *min_sidelength=8*)

The class for implementing DBNet loss.

This is partially adapted from https://github.com/MhLiao/DB.

Parameters

- **loss_prob** (*dict*) – The loss config for probability map. Defaults to dict(type='MaskedBalancedBCEWithLogitsLoss').
- **loss_thr** (*dict*) – The loss config for threshold map. Defaults to dict(type='MaskedSmoothL1Loss', beta=0).
- **loss_db** (*dict*) – The loss config for binary map. Defaults to dict(type='MaskedDiceLoss').
- **weight_prob** (*float*) – The weight of probability map loss. Denoted as α in paper. Defaults to 5.
- **weight_thr** (*float*) – The weight of threshold map loss. Denoted as β in paper. Defaults to 10.
- **shrink_ratio** (*float*) – The ratio of shrunk text region. Defaults to 0.4.
- **thr_min** (*float*) – The minimum threshold map value. Defaults to 0.3.
- **thr_max** (*float*) – The maximum threshold map value. Defaults to 0.7.
- **min_sidelength** (*int or float*) – The minimum sidelength of the minimum rotated rectangle around any text region. Defaults to 8.

Return type None

forward(*preds*, *data_samples*)

Compute DBNet loss.

Parameters

- **preds** (*tuple(tensor)*) – Raw predictions from model, containing `prob_logits`, `thr_map` and `binary_map`. Each is a tensor of shape (N, H, W).
- **data_samples** (*list[TextDetDataSample]*) – The data samples.

Returns The dict for dbnet losses with loss_prob, loss_db and loss_thr.

Return type results(dict)

get_targets(*data_samples*)

Generate loss targets from data samples.

Parameters **data_samples** (*list(TextDetDataSample)*) – Ground truth data samples.

Returns A tuple of four tensors as DBNet targets.

Return type tuple

TextSnakeModuleLoss

class mmocr.models.textdet.**TextSnakeModuleLoss**(*ohem_ratio=3.0*, *downsample_ratio=1.0*, *orientation_thr=2.0*, *resample_step=4.0*, *center_region_shrink_ratio=0.3*, *loss_text={'eps': 1e-05, 'fallback_negative_num': 100, 'type': 'MaskedBalancedBCEWithLogitsLoss'}*, *loss_center={'type': 'MaskedBCEWithLogitsLoss'}*, *loss_radius={'type': 'MaskedSmoothL1Loss'}*, *loss_sin={'type': 'MaskedSmoothL1Loss'}*, *loss_cos={'type': 'MaskedSmoothL1Loss'}*)

The class for implementing TextSnake loss. This is partially adapted from https://github.com/princewang1994/TextSnake.pytorch.

TextSnake: A Flexible Representation for Detecting Text of Arbitrary Shapes.

Parameters

- **ohem_ratio** (*float*) – The negative/positive ratio in ohem.
- **downsample_ratio** (*float*) – Downsample ratio. Defaults to 1.0. TODO: remove it.
- **orientation_thr** (*float*) – The threshold for distinguishing between head edge and tail edge among the horizontal and vertical edges of a quadrangle.
- **resample_step** (*float*) – The step of resampling.
- **center_region_shrink_ratio** (*float*) – The shrink ratio of text center.
- **loss_text** (*dict*) – The loss config used to calculate the text loss.
- **loss_center** (*dict*) – The loss config used to calculate the center loss.
- **loss_radius** (*dict*) – The loss config used to calculate the radius loss.
- **loss_sin** (*dict*) – The loss config used to calculate the sin loss.
- **loss_cos** (*dict*) – The loss config used to calculate the cos loss.

Return type None

forward(*preds*, *data_samples*)

Parameters

- **preds** (*Tensor*) – The prediction map of shape $(N, 5, H, W)$, where each dimension is the map of "text_region", "center_region", "sin_map", "cos_map", and "radius_map" respectively.
- **data_samples** (*list[TextDetDataSample]*) – The data samples.

Returns A loss dict with `loss_text`, `loss_center`, `loss_radius`, `loss_sin` and `loss_cos`.

Return type dict

get_targets(*data_samples*)

Generate loss targets from data samples.

Parameters **data_samples** (*list(TextDetDataSample)*) – Ground truth data samples.

Returns tuple(gt_text_masks, gt_masks, gt_center_region_masks, gt_radius_maps, gt_sin_maps, gt_cos_maps): A tuple of six lists of ndarrays as the targets.

Return type Tuple

vector_angle(*vec1*, *vec2*)
Compute the angle between two vectors.

Parameters

- **vec1** (*numpy.ndarray*) –
- **vec2** (*numpy.ndarray*) –

Return type numpy.ndarray

vector_cos(*vec*)
Compute the cos of the angle between vector and x-axis.

Parameters **vec** (*numpy.ndarray*) –

Return type float

vector_sin(*vec*)
Compute the sin of the angle between vector and x-axis.

Parameters **vec** (*numpy.ndarray*) –

Return type float

vector_slope(*vec*)
Compute the slope of a vector.

Parameters **vec** (*numpy.ndarray*) –

Return type float

FCEModuleLoss

class mmocr.models.textdet.**FCEModuleLoss**(*fourier_degree*, *num_sample*, *negative_ratio=3.0*, *resample_step=4.0*, *center_region_shrink_ratio=0.3*, *level_size_divisors=(8, 16, 32)*, *level_proportion_range=((0, 0.4), (0.3, 0.7), (0.6, 1.0))*, *loss_tr={'type': 'MaskedBalancedBCELoss'}*, *loss_tcl={'type': 'MaskedBCELoss'}*, *loss_reg_x={'reduction': 'none', 'type': 'SmoothL1Loss'}*, *loss_reg_y={'reduction': 'none', 'type': 'SmoothL1Loss'}*)
The class for implementing FCENet loss.

FCENet(CVPR2021): Fourier Contour Embedding for Arbitrary-shaped Text Detection

Parameters

- **fourier_degree** (*int*) – The maximum Fourier transform degree k.
- **num_sample** (*int*) – The sampling points number of regression loss. If it is too small, fcenet tends to be overfitting.
- **negative_ratio** (*float or int*) – Maximum ratio of negative samples to positive ones in OHEM. Defaults to 3.
- **resample_step** (*float*) – The step size for resampling the text center line (TCL). It's better not to exceed half of the minimum width.
- **center_region_shrink_ratio** (*float*) – The shrink ratio of text center region.
- **level_size_divisors** (*tuple(int)*) – The downsample ratio on each level.

- **level_proportion_range** (*tuple(tuple(int))*) – The range of text sizes assigned to each level.
- **loss_tr** (*dict*) – The loss config used to calculate the text region loss. Defaults to dict(type='MaskedBalancedBCELoss').
- **loss_tcl** (*dict*) – The loss config used to calculate the text center line loss. Defaults to dict(type='MaskedBCELoss').
- **loss_reg_x** (*dict*) – The loss config used to calculate the regression loss on x axis. Defaults to dict(type='MaskedSmoothL1Loss').
- **loss_reg_y** (*dict*) – The loss config used to calculate the regression loss on y axis. Defaults to dict(type='MaskedSmoothL1Loss').

Return type None

forward(*preds*, *data_samples*)

Compute FCENet loss.

Parameters

- **preds** (*list[dict]*) – A list of dict with keys of `cls_res`, `reg_res` corresponds to the classification result and regression result computed from the input tensor with the same index. They have the shapes of $(N, C_{cls,i}, H_i, W_i)$ and :math: *(N, C_{out,i}, H_i, W_i)*.
- **data_samples** (*list[TextDetDataSample]*) – The data samples.

Returns

The dict for fcenet losses with loss_text, loss_center, loss_reg_x and loss_reg_y.

Return type dict

forward_single(*pred*, *gt*)

Compute loss for one feature level.

Parameters

- **pred** (*dict*) – A dict with keys `cls_res` and `reg_res` corresponds to the classification result and regression result from one feature level.
- **gt** (*Tensor*) – Ground truth for one feature level. Cls and reg targets are concatenated along the channel dimension.

Returns A list of losses for each feature level.

Return type list[Tensor]

get_targets(*data_samples*)

Generate loss targets for fcenet from data samples.

Parameters **data_samples** (*list(TextDetDataSample)*) – Ground truth data samples.

Returns

A tuple of three tensors from three different feature level as FCENet targets.

Return type tuple[Tensor]

DRRGModuleLoss

class mmocr.models.textdet.**DRRGModuleLoss**(*ohem_ratio=3.0*, *downsample_ratio=1.0*, *orientation_thr=2.0*, *resample_step=8.0*, *num_min_comps=9*, *num_max_comps=600*, *min_width=8.0*, *max_width=24.0*, *center_region_shrink_ratio=0.3*, *comp_shrink_ratio=1.0*, *comp_w_h_ratio=0.3*, *text_comp_nms_thr=0.25*, *min_rand_half_height=8.0*, *max_rand_half_height=24.0*, *jitter_level=0.2*, *loss_text={'eps': 1e-05, 'fallback_negative_num': 100, 'type': 'MaskedBalancedBCEWithLogitsLoss'}*, *loss_center={'type': 'MaskedBCEWithLogitsLoss'}*, *loss_top={'reduction': 'none', 'type': 'SmoothL1Loss'}*, *loss_btm={'reduction': 'none', 'type': 'SmoothL1Loss'}*, *loss_sin={'type': 'MaskedSmoothL1Loss'}*, *loss_cos={'type': 'MaskedSmoothL1Loss'}*, *loss_gcn={'type': 'CrossEntropyLoss'}*)

The class for implementing DRRG loss. This is partially adapted from https://github.com/GXYM/DRRG licensed under the MIT license.

DRRG: Deep Relational Reasoning Graph Network for Arbitrary Shape Text Detection.

Parameters

- **ohem_ratio** (*float*) – The negative/positive ratio in ohem. Defaults to 3.0.
- **downsample_ratio** (*float*) – Downsample ratio. Defaults to 1.0. TODO: remove it.
- **orientation_thr** (*float*) – The threshold for distinguishing between head edge and tail edge among the horizontal and vertical edges of a quadrangle. Defaults to 2.0.
- **resample_step** (*float*) – The step size for resampling the text center line. Defaults to 8.0.
- **num_min_comps** (*int*) – The minimum number of text components, which should be larger than k_hop1 mentioned in paper. Defaults to 9.
- **num_max_comps** (*int*) – The maximum number of text components. Defaults to 600.
- **min_width** (*float*) – The minimum width of text components. Defaults to 8.0.
- **max_width** (*float*) – The maximum width of text components. Defaults to 24.0.
- **center_region_shrink_ratio** (*float*) – The shrink ratio of text center regions. Defaults to 0.3.
- **comp_shrink_ratio** (*float*) – The shrink ratio of text components. Defaults to 1.0.
- **comp_w_h_ratio** (*float*) – The width to height ratio of text components. Defaults to 0.3.
- **min_rand_half_height** (*float*) – The minimum half-height of random text components. Defaults to 8.0.
- **max_rand_half_height** (*float*) – The maximum half-height of random text components. Defaults to 24.0.
- **jitter_level** (*float*) – The jitter level of text component geometric features. Defaults to 0.2.
- **loss_text** (*dict*) – The loss config used to calculate the text loss. Defaults to `dict(type='MaskedBalancedBCEWithLogitsLoss', fallback_negative_num=100, eps=1e-5)`.

- **loss_center** (*dict*) – The loss config used to calculate the center loss. Defaults to `dict(type='MaskedBCEWithLogitsLoss')`.
- **loss_top** (*dict*) – The loss config used to calculate the top loss, which is a part of the height loss. Defaults to `dict(type='SmoothL1Loss', reduction='none')`.
- **loss_btm** (*dict*) – The loss config used to calculate the bottom loss, which is a part of the height loss. Defaults to `dict(type='SmoothL1Loss', reduction='none')`.
- **loss_sin** (*dict*) – The loss config used to calculate the sin loss. Defaults to `dict(type='MaskedSmoothL1Loss')`.
- **loss_cos** (*dict*) – The loss config used to calculate the cos loss. Defaults to `dict(type='MaskedSmoothL1Loss')`.
- **loss_gcn** (*dict*) – The loss config used to calculate the GCN loss. Defaults to `dict(type='CrossEntropyLoss')`.
- **text_comp_nms_thr** (*float*) –

Return type None

forward(*preds*, *data_samples*)
Compute Drrg loss.

Parameters

- **preds** (*tuple*) – The prediction tuple(pred_maps, gcn_pred, gt_labels), each of shape $(N, 6, H, W)$, $(N, 2)$ and (m, n), where $m * n = N$.
- **data_samples** (*list[TextDetDataSample]*) – The data samples.

Returns A loss dict with `loss_text`, `loss_center`, `loss_height`, `loss_sin`, `loss_cos`, and `loss_gcn`.

Return type dict

get_targets(*data_samples*)
Generate loss targets from data samples.

Parameters **data_samples** (*list(TextDetDataSample)*) – Ground truth data samples.

Returns A tuple of 8 lists of tensors as DRRG targets. Read docstring of `_get_target_single` for more details.

Return type tuple

46.2.6 Postprocessors

BaseTextDetPostProcessor	Base postprocessor for text detection models.
PSEPostprocessor	Decoding predictions of PSENet to instances.
PANPostprocessor	Convert scores to quadrangles via post processing in PANet.
DBPostprocessor	Decoding predictions of DbNet to instances.
DRRGPostprocessor	Merge text components and construct boundaries of text instances.
FCEPostprocessor	Decoding predictions of FCENet to instances.
TextSnakePostprocessor	Decoding predictions of TextSnake to instances.

BaseTextDetPostProcessor

class mmocr.models.textdet.**BaseTextDetPostProcessor**(*text_repr_type='poly'*, *rescale_fields=None*, *train_cfg=None*, *test_cfg=None*)

Base postprocessor for text detection models.

Parameters

- **text_repr_type** (*str*) – The boundary encoding type, 'poly' or 'quad'. Defaults to 'poly'.
- **rescale_fields** (*list[str], optional*) – The bbox/polygon field names to be rescaled. If None, no rescaling will be performed.
- **train_cfg** (*dict, optional*) – The parameters to be passed to `self.get_text_instances` in training. Defaults to None.
- **test_cfg** (*dict, optional*) – The parameters to be passed to `self.get_text_instances` in testing. Defaults to None.

Return type None

get_text_instances(*pred_results*, *data_sample*, ***kwargs*)

Get text instance predictions of one image.

Parameters

- **pred_result** (*tuple(Tensor)*) – Prediction results of an image.
- **data_sample** (TextDetDataSample) – Datasample of an image.
- ****kwargs** – Other parameters. Configurable via `__init__.train_cfg` and `__init__.test_cfg`.
- **pred_results** (*Union[torch.Tensor, List[torch.Tensor]]*) –

Returns A new DataSample with predictions filled in. The polygon/bbox results are usually saved in `TextDetDataSample.pred_instances.polygons` or `TextDetDataSample.pred_instances.bboxes`. The confidence scores are saved in `TextDetDataSample.pred_instances.scores`.

Return type *TextDetDataSample*

poly_nms(*polygons*, *scores*, *threshold*)

Non-maximum suppression for text detection.

Parameters

- **polygons** (*list[ndarray]*) – List of polygons.
- **scores** (*list[float]*) – List of scores.
- **threshold** (*float*) – Threshold for NMS.

Returns

- keep_polys (list[ndarray]): List of preserved polygons after NMS.
- keep_scores (list[float]): List of preserved scores after NMS.

Return type tuple(keep_polys, keep_scores)

rescale(*results*, *scale_factor*)

Rescale results in `results.pred_instances` according to `scale_factor`, whose keys are defined in `self.rescale_fields`. Usually used to rescale bboxes and/or polygons.

Parameters

- **results** (TextDetDataSample) – The post-processed prediction results.
- **scale_factor** (*tuple(int)*) – (w_scale, h_scale)

Returns Prediction results with rescaled results.

Return type *TextDetDataSample*

split_results(*pred_results*)

Split batched tensor(s) along the first dimension pack split tensors into a list.

Parameters **pred_results** (*tensor or list[tensor]*) – Raw result tensor(s) from detection head. Each tensor usually has the shape of (N, ...)

Returns

N tensors if pred_results is a tensor, or a list of N lists of tensors if `pred_results` is a list of tensors.

Return type list[tensor] or list[list[tensor]]

PSEPostprocessor

class mmocr.models.textdet.**PSEPostprocessor**(*text_repr_type='poly'*, *rescale_fields=['polygons']*, *min_kernel_confidence=0.5*, *score_threshold=0.3*, *min_kernel_area=0*, *min_text_area=16*, *downsample_ratio=0.25*)

Decoding predictions of PSENet to instances. This is partially adapted from https://github.com/whai362/PSENet.

Parameters

- **text_repr_type** (*str*) – The boundary encoding type 'poly' or 'quad'. Defaults to 'poly'.
- **rescale_fields** (*list[str]*) – The bbox/polygon field names to be rescaled. If None, no rescaling will be performed. Defaults to ['polygons'].
- **min_kernel_confidence** (*float*) – The minimal kernel confidence. Defaults to 0.5.
- **score_threshold** (*float*) – The minimal text average confidence. Defaults to 0.3.
- **min_kernel_area** (*int*) – The minimal text kernel area. Defaults to 0.
- **min_text_area** (*int*) – The minimal text instance region area. Defaults to 16.
- **downsample_ratio** (*float*) – Downsample ratio. Defaults to 0.25.

Return type None

get_text_instances(*pred_results*, *data_sample*, ***kwargs*)

Parameters

- **pred_result** (*torch.Tensor*) – Prediction results of an image which is a tensor of shape (N, H, W).
- **data_sample** (TextDetDataSample) – Datasample of an image.
- **pred_results** (*torch.Tensor*) –

Returns A new DataSample with predictions filled in. Polygons and results are saved in `TextDetDataSample.pred_instances.polygons`. The confidence scores are saved in `TextDetDataSample.pred_instances.scores`.

Return type *TextDetDataSample*

PANPostprocessor

class mmocr.models.textdet.**PANPostprocessor**(*text_repr_type='poly'*, *score_threshold=0.3*, *rescale_fields=['polygons']*, *min_text_confidence=0.5*, *min_kernel_confidence=0.5*, *distance_threshold=3.0*, *min_text_area=16*, *downsample_ratio=0.25*)

Convert scores to quadrangles via post processing in PANet. This is partially adapted from https://github.com/WenmuZhou/PAN.pytorch.

Parameters

- **text_repr_type** (*str*) – The boundary encoding type 'poly' or 'quad'. Defaults to 'poly'.
- **score_threshold** (*float*) – The minimal text score. Defaults to 0.3.
- **rescale_fields** (*list[str]*) – The bbox/polygon field names to be rescaled. If None, no rescaling will be performed. Defaults to ['polygons'].
- **min_text_confidence** (*float*) – The minimal text confidence. Defaults to 0.5.
- **min_kernel_confidence** (*float*) – The minimal kernel confidence. Defaults to 0.5.
- **distance_threshold** (*float*) – The minimal distance between the point to mean of text kernel. Defaults to 3.0.
- **min_text_area** (*int*) – The minimal text instance region area. Defaults to 16.
- **downsample_ratio** (*float*) – Downsample ratio. Defaults to 0.25.

Return type None

get_text_instances(*pred_results*, *data_sample*, ***kwargs*)

Get text instance predictions of one image.

Parameters

- **pred_result** (*torch.Tensor*) – Prediction results of an image which is a tensor of shape (N, H, W).
- **data_sample** (TextDetDataSample) – Datasample of an image.
- **pred_results** (*torch.Tensor*) –

Returns A new DataSample with predictions filled in. Polygons and results are saved in `TextDetDataSample.pred_instances.polygons`. The confidence scores are saved in `TextDetDataSample.pred_instances.scores`.

Return type *TextDetDataSample*

DBPostprocessor

class mmocr.models.textdet.**DBPostprocessor**(*text_repr_type='poly'*, *rescale_fields=['polygons']*, *mask_thr=0.3*, *min_text_score=0.3*, *min_text_width=5*, *unclip_ratio=1.5*, *epsilon_ratio=0.01*, *max_candidates=3000*, ***kwargs*)

Decoding predictions of DbNet to instances. This is partially adapted from https://github.com/MhLiao/DB.

Parameters

- **text_repr_type** (*str*) – The boundary encoding type 'poly' or 'quad'. Defaults to 'poly'.
- **rescale_fields** (*list[str]*) – The bbox/polygon field names to be rescaled. If None, no rescaling will be performed. Defaults to ['polygons'].
- **mask_thr** (*float*) – The mask threshold value for binarization. Defaults to 0.3.
- **min_text_score** (*float*) – The threshold value for converting binary map to shrink text regions. Defaults to 0.3.
- **min_text_width** (*int*) – The minimum width of boundary polygon/box predicted. Defaults to 5.
- **unclip_ratio** (*float*) – The unclip ratio for text regions dilation. Defaults to 1.5.
- **epsilon_ratio** (*float*) – The epsilon ratio for approximation accuracy. Defaults to 0.01.
- **max_candidates** (*int*) – The maximum candidate number. Defaults to 3000.

Return type None

get_text_instances(*prob_map*, *data_sample*)

Get text instance predictions of one image.

Parameters

- **pred_result** (*Tensor*) – DBNet's output `prob_map` of shape (H, W).
- **data_sample** (TextDetDataSample) – Datasample of an image.
- **prob_map** (*torch.Tensor*) –

Returns A new DataSample with predictions filled in. Polygons and results are saved in `TextDetDataSample.pred_instances.polygons`. The confidence scores are saved in `TextDetDataSample.pred_instances.scores`.

Return type *TextDetDataSample*

DRRGPostprocessor

class mmocr.models.textdet.**DRRGPostprocessor**(*link_thr=0.8*, *edge_len_thr=50.0*, *rescale_fields=['polygons']*, ***kwargs*)

Merge text components and construct boundaries of text instances.

Parameters

- **link_thr** (*float*) – The edge score threshold. Defaults to 0.8.
- **edge_len_thr** (*int or float*) – The edge length threshold. Defaults to 50.
- **rescale_fields** (*list[str]*) – The bbox/polygon field names to be rescaled. If None, no rescaling will be performed. Defaults to [polygons'].

Return type None

get_text_instances(*pred_results*, *data_sample*)
Get text instance predictions of one image.

Parameters

- **pred_result** (*tuple(ndarray, ndarray, ndarray)*) – Prediction results edge, score and text_comps. Each of shape $(N_{edges}, 2)$, $(N_{edges},)$ and $(M, 9)$, respectively.
- **data_sample** (TextDetDataSample) – Datasample of an image.
- **pred_results** (*Tuple[numpy.ndarray, numpy.ndarray, numpy.ndarray]*) –

Returns The original dataSample with predictions filled in. Polygons and results are saved in `TextDetDataSample.pred_instances.polygons`. The confidence scores are saved in `TextDetDataSample.pred_instances.scores`.

Return type *TextDetDataSample*

split_results(*pred_results*)
Split batched elements in pred_results along the first dimension into `batch_num` sub-elements and regather them into a list of dicts.

However, DRRG only outputs one batch at inference time, so this function is a no-op.

Parameters **pred_results** (*Tuple[numpy.ndarray, numpy.ndarray, numpy.ndarray]*) –

Return type List[Tuple]

FCEPostprocessor

class mmocr.models.textdet.**FCEPostprocessor**(*fourier_degree*, *num_reconstr_points*, *rescale_fields=['polygons']*, *scales=[8, 16, 32]*, *text_repr_type='poly'*, *alpha=1.0*, *beta=2.0*, *score_thr=0.3*, *nms_thr=0.1*, ***kwargs*)
Decoding predictions of FCENet to instances.

Parameters

- **fourier_degree** (*int*) – The maximum Fourier transform degree k.
- **num_reconstr_points** (*int*) – The points number of the polygon reconstructed from predicted Fourier coefficients.
- **rescale_fields** (*list[str]*) – The bbox/polygon field names to be rescaled. If None, no rescaling will be performed. Defaults to ['polygons'].
- **scales** (*list[int]*) – The down-sample scale of each layer. Defaults to [8, 16, 32].
- **text_repr_type** (*str*) –

  Boundary encoding type 'poly' or 'quad'. Defaults to 'poly'.

  alpha (float): The parameter to calculate final scores $Score_{final} = (Score^{a}_{textregion}lpha) * (Score^{b}_{textcenter_{r}egion}eta)$. Defaults to 1.0.
- **beta** (*float*) – The parameter to calculate final score. Defaults to 2.0.
- **score_thr** (*float*) – The threshold used to filter out the final candidates.Defaults to 0.3.
- **nms_thr** (*float*) – The threshold of nms. Defaults to 0.1.

- **alpha** (*float*) –

Return type None

get_text_instances(*pred_results*, *data_sample*)
Get text instance predictions of one image.

Parameters

- **pred_results** (*List[dict]*) – A list of dict with keys of `cls_res`, `reg_res` corresponding to the classification result and regression result computed from the input tensor with the same index. They have the shapes of $(N, C_{cls,i}, H_i, W_i)$ and $(N, C_{out,i}, H_i, W_i)$.
- **data_sample** (`TextDetDataSample`) – Datasample of an image.

Returns A new DataSample with predictions filled in. Polygons and results are saved in `TextDetDataSample.pred_instances.polygons`. The confidence scores are saved in `TextDetDataSample.pred_instances.scores`.

Return type *TextDetDataSample*

split_results(*pred_results*)
Split batched elements in pred_results along the first dimension into `batch_num` sub-elements and regather them into a list of dicts.

Parameters **pred_results** (*list[dict]*) – A list of dict with keys of `cls_res`, `reg_res` corresponding to the classification result and regression result computed from the input tensor with the same index. They have the shapes of $(N, C_{cls,i}, H_i, W_i)$ and $(N, C_{out,i}, H_i, W_i)$.

Returns N lists. Each list contains three dicts from different feature level.

Return type list[list[dict]]

TextSnakePostprocessor

class mmocr.models.textdet.**TextSnakePostprocessor**(*text_repr_type='poly'*, *min_text_region_confidence=0.6*, *min_center_region_confidence=0.2*, *min_center_area=30*, *disk_overlap_thr=0.03*, *radius_shrink_ratio=1.03*, *rescale_fields=['polygons']*, ***kwargs*)
Decoding predictions of TextSnake to instances. This was partially adapted from https://github.com/princewang1994/TextSnake.pytorch.

Parameters

- **text_repr_type** (*str*) – The boundary encoding type 'poly' or 'quad'.
- **min_text_region_confidence** (*float*) – The confidence threshold of text region in TextSnake.
- **min_center_region_confidence** (*float*) – The confidence threshold of text center region in TextSnake.
- **min_center_area** (*int*) – The minimal text center region area.
- **disk_overlap_thr** (*float*) – The radius overlap threshold for merging disks.
- **radius_shrink_ratio** (*float*) – The shrink ratio of ordered disks radii.

- **rescale_fields** (*list[str], optional*) – The bbox/polygon field names to be rescaled. If None, no rescaling will be performed.

Return type None

get_text_instances(*pred_results*, *data_sample*)

Parameters

- **pred_results** (*torch.Tensor*) – Prediction map with shape (C, H, W).
- **data_sample** (TextDetDataSample) – Datasample of an image.

Returns The instance boundary and its confidence.

Return type list[list[float]]

split_results(*pred_results*)

Split the prediction results into text score and kernel score.

Parameters **pred_results** (*torch.Tensor*) – The prediction results.

Returns The text score and kernel score.

Return type List[torch.Tensor]

46.3 models.textrecog

46.3.1 Recognizers

—

46.3.2 Data Preprocessors

—

46.3.3 Preprocessors

—

46.3.4 BackBones

—

46.3.5 Encoders

—

46.3.6 Decoders

—

46.3.7 Module Losses

—

46.3.8 Postprocessors

—

46.3.9 Layers

—

46.4 models.kie

46.4.1 Extractors

SDMGR	The implementation of the paper: Spatial Dual-Modality Graph Reasoning for Key Information Extraction.

SDMGR

class mmocr.models.kie.**SDMGR**(*backbone=None*, *roi_extractor=None*, *neck=None*, *kie_head=None*, *dictionary=None*, *data_preprocessor=None*, *init_cfg=None*)

The implementation of the paper: Spatial Dual-Modality Graph Reasoning for Key Information Extraction. https://arxiv.org/abs/2103.14470.

Parameters

- **backbone** (*dict, optional*) – Config of backbone. If None, None will be passed to kie_head during training and testing. Defaults to None.
- **roi_extractor** (*dict, optional*) – Config of roi extractor. Only applicable when backbone is not None. Defaults to None.
- **neck** (*dict, optional*) – Config of neck. Defaults to None.
- **kie_head** (*dict*) – Config of KIE head. Defaults to None.
- **dictionary** (*dict, optional*) – Config of dictionary. Defaults to None.

- **data_preprocessor** (*dict or ConfigDict, optional*) – The pre-process config of BaseDataPreprocessor. it usually includes, pad_size_divisor, pad_value, mean and std. It has to be None when working in non-visual mode. Defaults to None.
- **init_cfg** (*dict or list[dict], optional*) – Initialization configs. Defaults to None.

Return type None

extract_feat(*img*, *gt_bboxes*)

Extract features from images if self.backbone is not None. It returns None otherwise.

Parameters

- **img** (*torch.Tensor*) – The input image with shape (N, C, H, W).
- **gt_bboxes** (*list[torch.Tensor]*) – A list of ground truth bounding boxes, each of shape $(N_i, 4)$.

Returns The extracted features with shape (N, E).

Return type torch.Tensor

forward(*inputs*, *data_samples=None*, *mode='tensor'*, ***kwargs*)

The unified entry for a forward process in both training and test.

The method should accept three modes: "tensor", "predict" and "loss":

- "tensor": Forward the whole network and return tensor or tuple of

tensor without any post-processing, same as a common nn.Module. - "predict": Forward and return the predictions, which are fully processed to a list of DetDataSample. - "loss": Forward and return a dict of losses according to the given inputs and data samples.

Note that this method doesn't handle neither back propagation nor optimizer updating, which are done in the train_step().

Parameters

- **inputs** (*torch.Tensor*) – The input tensor with shape (N, C, ...) in general.
- **data_samples** (list[DetDataSample], optional) – The annotation data of every samples. Defaults to None.
- **mode** (*str*) – Return what kind of value. Defaults to 'tensor'.

Returns

The return type depends on mode.

- If mode="tensor", return a tensor or a tuple of tensor.
- If mode="predict", return a list of DetDataSample.
- If mode="loss", return a dict of tensor.

Return type torch.Tensor

loss(*inputs*, *data_samples*, ***kwargs*)

Calculate losses from a batch of inputs and data samples.

Parameters

- **inputs** (*torch.Tensor*) – Input images of shape (N, C, H, W). Typically these should be mean centered and std scaled.

- **data_samples** (*list[KIEDataSample]*) – A list of N datasamples, containing meta information and gold annotations for each of the images.

Returns A dictionary of loss components.

Return type dict[str, Tensor]

predict(*inputs*, *data_samples*, ***kwargs*)

Predict results from a batch of inputs and data samples with post- processing. :param inputs: Input images of shape (N, C, H, W).

Typically these should be mean centered and std scaled.

Parameters

- **data_samples** (*list[KIEDataSample]*) – A list of N datasamples, containing meta information and gold annotations for each of the images.
- **inputs** (*torch.Tensor*) –

Returns A list of datasamples of prediction results. Results are stored in `pred_instances.labels` and `pred_instances.edge_labels`.

Return type List[*KIEDataSample*]

46.4.2 Heads

SDMGRHead	SDMGR Head.

SDMGRHead

class mmocr.models.kie.**SDMGRHead**(*dictionary*, *num_classes=26*, *visual_dim=64*, *fusion_dim=1024*, *node_input=32*, *node_embed=256*, *edge_input=5*, *edge_embed=256*, *num_gnn=2*, *bidirectional=False*, *relation_norm=10.0*, *module_loss={'type': 'SDMGRModuleLoss'}*, *postprocessor={'type': 'SDMGRPostProcessor'}*, *init_cfg={'mean': 0, 'override': {'name': 'edge_embed'}, 'std': 0.01, 'type': 'Normal'}*)

SDMGR Head.

Parameters

- **dictionary** (dict or `Dictionary`) – The config for *Dictionary* or the instance of *Dictionary*.
- **num_classes** (*int*) – Number of class labels. Defaults to 26.
- **visual_dim** (*int*) – Dimension of visual features E. Defaults to 64.
- **fusion_dim** (*int*) – Dimension of fusion layer. Defaults to 1024.
- **node_input** (*int*) – Dimension of raw node embedding. Defaults to 32.
- **node_embed** (*int*) – Dimension of node embedding. Defaults to 256.
- **edge_input** (*int*) – Dimension of raw edge embedding. Defaults to 5.
- **edge_embed** (*int*) – Dimension of edge embedding. Defaults to 256.
- **num_gnn** (*int*) – Number of GNN layers. Defaults to 2.

- **bidirectional** (*bool*) – Whether to use bidirectional RNN to embed nodes. Defaults to False.
- **relation_norm** (*float*) – Norm to map value from one range to another.= Defaults to 10.
- **module_loss** (*dict*) – Module Loss config. Defaults to `dict(type='SDMGRModuleLoss')`.
- **postprocessor** (*dict*) – Postprocessor config. Defaults to `dict(type='SDMGRPostProcessor')`.
- **init_cfg** (*dict or list[dict], optional*) – Initialization configs.

Return type None

compute_relations(*data_samples*)

Compute the relations between every two boxes for each datasample, then return the concatenated relations.

Parameters **data_samples** (*List[mmocr.structures.kie_data_sample.KIEDataSample]*) –

Return type torch.Tensor

convert_texts(*data_samples*)

Extract texts in datasamples and pack them into a batch.

Parameters **data_samples** (*List[KIEDataSample]*) – List of data samples.

Returns

- node_nums (List[int]): A list of node numbers for each sample.
- char_nums (List[Tensor]): A list of character numbers for each sample.
- nodes (Tensor): A tensor of shape (N, C) where C is the maximum number of characters in a sample.

Return type tuple(List[int], List[Tensor], Tensor)

forward(*inputs*, *data_samples*)

Parameters

- **inputs** (*torch.Tensor*) – Shape (N, E).
- **data_samples** (*List[KIEDataSample]*) – List of data samples.

Returns

- node_cls (Tensor): Raw logits scores for nodes. Shape (N, C_l) where C_l is number of classes.
- edge_cls (Tensor): Raw logits scores for edges. Shape $(N * N, 2)$.

Return type tuple(Tensor, Tensor)

loss(*inputs*, *data_samples*)

Calculate losses from a batch of inputs and data samples. :param inputs: Shape (N, E). :type inputs: torch.Tensor :param data_samples: List of data samples. :type data_samples: List[KIEDataSample]

Returns A dictionary of loss components.

Return type dict[str, tensor]

Parameters

- **inputs** (*torch.Tensor*) –
- **data_samples** (*List[mmocr.structures.kie_data_sample.KIEDataSample]*) –

predict(*inputs*, *data_samples*)
Predict results from a batch of inputs and data samples with post- processing.

Parameters

- **inputs** (*torch.Tensor*) – Shape (N, E).
- **data_samples** (*List[KIEDataSample]*) – List of data samples.

Returns

A list of datasamples of prediction results. Results are stored in `pred_instances.labels`, `pred_instances.scores`, `pred_instances.edge_labels` and `pred_instances.edge_scores`.

- labels (Tensor): An integer tensor of shape (N,) indicating bbox labels for each image.
- scores (Tensor): A float tensor of shape (N,), indicating the confidence scores for node label predictions.
- edge_labels (Tensor): An integer tensor of shape (N, N) indicating the connection between nodes. Options are 0, 1.
- edge_scores (Tensor): A float tensor of shape (N,), indicating the confidence scores for edge predictions.

Return type List[*KIEDataSample*]

46.4.3 Module Losses

SDMGRModuleLoss	The implementation the loss of key information extraction proposed in the paper: Spatial Dual-Modality Graph Reasoning for Key Information Extraction.

SDMGRModuleLoss

class mmocr.models.kie.**SDMGRModuleLoss**(*weight_node=1.0*, *weight_edge=1.0*, *ignore_idx=- 100*)
The implementation the loss of key information extraction proposed in the paper: Spatial Dual-Modality Graph Reasoning for Key Information Extraction.

Parameters

- **weight_node** (*float*) – Weight of node loss. Defaults to 1.0.
- **weight_edge** (*float*) – Weight of edge loss. Defaults to 1.0.
- **ignore_idx** (*int*) – Node label to ignore. Defaults to -100.

Return type None

forward(*preds*, *data_samples*)
Forward function.

Parameters

- **preds** (*tuple(Tensor, Tensor)*) –

- **data_samples** (*list[KIEDataSample]*) – A list of datasamples containing `gt_instances.labels` and `gt_instances.edge_labels`.

Returns Loss dict, containing `loss_node`, `loss_edge`, `acc_node` and `acc_edge`.

Return type dict(str, Tensor)

46.4.4 Postprocessors

SDMGRPostProcessor	Postprocessor for SDMGR.

SDMGRPostProcessor

class mmocr.models.kie.**SDMGRPostProcessor**(*link_type='none'*, *key_node_idx=None*, *value_node_idx=None*)

Postprocessor for SDMGR. It converts the node and edge scores into labels and edge labels. If the link_type is not "none", it reconstructs the edge labels with different strategies specified by `link_type`, which is generally known as the "openset" mode. In "openset" mode, only the edges connecting from "key" to "value" nodes will be constructed.

Parameters

- **link_type** (*str*) – The type of link to be constructed. Defaults to 'none'. Options are:
  - 'none': The simplest link type involving no edge postprocessing. The edge prediction will be returned as-is.
  - 'one-to-one': One key node can be connected to one value node.
  - 'one-to-many': One key node can be connected to multiple value nodes.
  - 'many-to-one': Multiple key nodes can be connected to one value node.
  - 'many-to-many': No restrictions on the number of edges that a key/value node can have.
- **key_node_idx** (*int, optional*) – The label index of the key node. It must be specified if `link_type` is not "none". Defaults to None.
- **value_node_idx** (*int, optional*) – The index of the value node. It must be specified if `link_type` is not "none". Defaults to None.

decode_edges(*node_labels*, *edge_scores*, *edge_labels*)

Reconstruct the edges and update edge scores according to `link_type`.

Parameters

- **data_sample** (KIEDataSample) – A datasample containing prediction results.
- **node_labels** (*torch.Tensor*) –
- **edge_scores** (*torch.Tensor*) –
- **edge_labels** (*torch.Tensor*) –

Returns

- **edge_scores (Tensor): A float tensor of shape (N, N)** indicating the confidence scores for edge predictions.

- **edge_labels (Tensor): An integer tensor of shape (N, N)** indicating the connection between nodes. Options are 0, 1.

Return type tuple(Tensor, Tensor)

CHAPTER

FORTYSEVEN

MMOCR.EVALUATION

mmocr.evaluation

47.1 Evaluator

MultiDatasetsEvaluator	Wrapper class to compose class: *ConcatDataset* and multiple `BaseMetric` instances.

47.1.1 MultiDatasetsEvaluator

class mmocr.evaluation.evaluator.**MultiDatasetsEvaluator**(*metrics*, *dataset_prefixes*)

Wrapper class to compose class: *ConcatDataset* and multiple `BaseMetric` instances. The metrics will be evaluated on each dataset slice separately. The name of the each metric is the concatenation of the dataset prefix, the metric prefix and the key of metric - e.g. *dataset_prefix/metric_prefix/accuracy*.

Parameters

- **metrics** (*dict or BaseMetric or Sequence*) – The config of metrics.
- **dataset_prefixes** (*Sequence[str]*) – The prefix of each dataset. The length of this sequence should be the same as the length of the datasets.

Return type None

evaluate(*size*)

Invoke `evaluate` method of each metric and collect the metrics dictionary.

Parameters **size** (*int*) – Length of the entire validation dataset. When batch size > 1, the dataloader may pad some data samples to make sure all ranks have the same length of dataset slice. The `collect_results` function will drop the padded data based on this size.

Returns Evaluation results of all metrics. The keys are the names of the metrics, and the values are corresponding results.

Return type dict

47.2 TextDet Metric

HmeanIOUMetric	HmeanIOU metric.

47.2.1 HmeanIOUMetric

class mmocr.evaluation.metrics.**HmeanIOUMetric**(*match_iou_thr=0.5*, *ignore_precision_thr=0.5*, *pred_score_thrs={'start': 0.3, 'step': 0.1, 'stop': 0.9}*, *strategy='vanilla'*, *collect_device='cpu'*, *prefix=None*)

HmeanIOU metric.

This method computes the hmean iou metric, which is done in the following steps:

- Filter the prediction polygon:
  - Scores is smaller than minimum prediction score threshold.
  - The proportion of the area that intersects with gt ignored polygon is greater than ignore_precision_thr.
- Computing an M x N IoU matrix, where each element indexing E_mn represents the IoU between the m-th valid GT and n-th valid prediction.
- Based on different prediction score threshold: - Obtain the ignored predictions according to prediction score.

  The filtered predictions will not be involved in the later metric computations.

  - Based on the IoU matrix, get the match metric according to

  `match_iou_thr`. - Based on different *strategy*, accumulate the match number.
- calculate H-mean under different prediction score threshold.

Parameters

- **match_iou_thr** (*float*) – IoU threshold for a match. Defaults to 0.5.
- **ignore_precision_thr** (*float*) – Precision threshold when prediction and gt ignored polygons are matched. Defaults to 0.5.
- **pred_score_thrs** (*dict*) – Best prediction score threshold searching space. Defaults to dict(start=0.3, stop=0.9, step=0.1).
- **strategy** (*str*) – Polygon matching strategy. Options are 'max_matching' and 'vanilla'. 'max_matching' refers to the optimum strategy that maximizes the number of matches. Vanilla strategy matches gt and pred polygons if both of them are never matched before. It was used in MMOCR 0.x and and academia. Defaults to 'vanilla'.
- **collect_device** (*str*) – Device name used for collecting results from different ranks during distributed training. Must be 'cpu' or 'gpu'. Defaults to 'cpu'.

- **prefix** (*str, optional*) – The prefix that will be added in the metric names to disambiguate homonymous metrics of different evaluators. If prefix is not provided in the argument, self.default_prefix will be used instead. Defaults to None

Return type None

compute_metrics(*results*)

Compute the metrics from processed results.

Parameters **results** (*list[dict]*) – The processed results of each batch.

Returns The computed metrics. The keys are the names of the metrics, and the values are corresponding results.

Return type dict

process(*data_batch*, *data_samples*)

Process one batch of data samples and predictions. The processed results should be stored in `self.results`, which will be used to compute the metrics when all batches have been processed.

Parameters

- **data_batch** (*Sequence[Dict]*) – A batch of data from dataloader.
- **data_samples** (*Sequence[Dict]*) – A batch of outputs from the model.

Return type None

47.3 TextRecog Metric

WordMetric	Word metrics for text recognition task.
CharMetric	Character metrics for text recognition task.
OneMinusNEDMetric	One minus NED metric for text recognition task.

47.3.1 WordMetric

class mmocr.evaluation.metrics.**WordMetric**(*mode='ignore_case_symbol'*, *valid_symbol='[^A-Z^a-z^0-9^-]'*, *collect_device='cpu'*, *prefix=None*)

Word metrics for text recognition task.

Parameters

- **mode** (*str or list[str]*) – Options are: - 'exact': Accuracy at word level. - 'ignore_case': Accuracy at word level, ignoring letter case.
  - 'ignore_case_symbol': Accuracy at word level, ignoring letter case and symbol. (Default metric for academic evaluation)

  If mode is a list, then metrics in mode will be calculated separately. Defaults to 'ignore_case_symbol'
- **valid_symbol** (*str*) – Valid characters. Defaults to '[^A-Z^a-z^0-9^-]'

- **collect_device** (*str*) – Device name used for collecting results from different ranks during distributed training. Must be 'cpu' or 'gpu'. Defaults to 'cpu'.
- **prefix** (*str, optional*) – The prefix that will be added in the metric names to disambiguate homonymous metrics of different evaluators. If prefix is not provided in the argument, self.default_prefix will be used instead. Defaults to None.

Return type None

compute_metrics(*results*)

Compute the metrics from processed results.

Parameters **results** (*list[Dict]*) – The processed results of each batch.

Returns The computed metrics. The keys are the names of the metrics, and the values are corresponding results.

Return type Dict

process(*data_batch*, *data_samples*)

Process one batch of data_samples. The processed results should be stored in `self.results`, which will be used to compute the metrics when all batches have been processed.

Parameters

- **data_batch** (*Sequence[Dict]*) – A batch of gts.
- **data_samples** (*Sequence[Dict]*) – A batch of outputs from the model.

Return type None

47.3.2 CharMetric

class mmocr.evaluation.metrics.**CharMetric**(*valid_symbol='[^A-Z^a-z^0-9^-]'*, *collect_device='cpu'*, *prefix=None*)

Character metrics for text recognition task.

Parameters

- **valid_symbol** (*str*) – Valid characters. Defaults to '[^A-Z^a-z^0-9^-]'
- **collect_device** (*str*) – Device name used for collecting results from different ranks during distributed training. Must be 'cpu' or 'gpu'. Defaults to 'cpu'.
- **prefix** (*str, optional*) – The prefix that will be added in the metric names to disambiguate homonymous metrics of different evaluators. If prefix is not provided in the argument, self.default_prefix will be used instead. Defaults to None.

Return type None

compute_metrics(*results*)

Compute the metrics from processed results.

Parameters **results** (*list[Dict]*) – The processed results of each batch.

Returns The computed metrics. The keys are the names of the metrics, and the values are corresponding results.

Return type Dict

process(*data_batch*, *data_samples*)

Process one batch of data_samples. The processed results should be stored in `self.results`, which will be used to compute the metrics when all batches have been processed.

Parameters

- **data_batch** (*Sequence[Dict]*) – A batch of gts.
- **data_samples** (*Sequence[Dict]*) – A batch of outputs from the model.

Return type None

47.3.3 OneMinusNEDMetric

class mmocr.evaluation.metrics.**OneMinusNEDMetric**(*valid_symbol='[^A-Z^a-z^0-9^-]'*, *collect_device='cpu'*, *prefix=None*)

One minus NED metric for text recognition task.

Parameters

- **valid_symbol** (*str*) – Valid characters. Defaults to '[^A-Z^a-z^0-9^-]'
- **collect_device** (*str*) – Device name used for collecting results from different ranks during distributed training. Must be 'cpu' or 'gpu'. Defaults to 'cpu'.
- **prefix** (*str, optional*) – The prefix that will be added in the metric names to disambiguate homonymous metrics of different evaluators. If prefix is not provided in the argument, self.default_prefix will be used instead. Defaults to None

Return type None

compute_metrics(*results*)

Compute the metrics from processed results.

Parameters **results** (*list[Dict]*) – The processed results of each batch.

Returns The computed metrics. The keys are the names of the metrics, and the values are corresponding results.

Return type Dict

process(*data_batch*, *data_samples*)

Process one batch of data_samples. The processed results should be stored in `self.results`, which will be used to compute the metrics when all batches have been processed.

Parameters

- **data_batch** (*Sequence[Dict]*) – A batch of gts.
- **data_samples** (*Sequence[Dict]*) – A batch of outputs from the model.

Return type None

47.4 KIE Metric

F1Metric	Compute F1 scores.

47.4.1 F1Metric

class mmocr.evaluation.metrics.**F1Metric**(*num_classes*, *key='labels'*, *mode='micro'*, *cared_classes=[]*, *ignored_classes=[]*, *collect_device='cpu'*, *prefix=None*)

Compute F1 scores.

Parameters

- **num_classes** (*int*) – Number of labels.
- **key** (*str*) – The key name of the predicted and ground truth labels. Defaults to 'labels'.
- **mode** (*str or list[str]*) – Options are: - 'micro': Calculate metrics globally by counting the total true

  positives, false negatives and false positives.

  - 'macro': Calculate metrics for each label, and find their unweighted mean.

  If mode is a list, then metrics in mode will be calculated separately. Defaults to 'micro'.
- **cared_classes** (*list[int]*) – The indices of the labels particpated in the metirc computing. If both `cared_classes` and `ignored_classes` are empty, all classes will be taken into account. Defaults to []. Note: `cared_classes` and `ignored_classes` cannot be specified together.
- **ignored_classes** (*list[int]*) – The index set of labels that are ignored when computing metrics. If both `cared_classes` and `ignored_classes` are empty, all classes will be taken into account. Defaults to []. Note: `cared_classes` and `ignored_classes` cannot be specified together.
- **collect_device** (*str*) – Device name used for collecting results from different ranks during distributed training. Must be 'cpu' or 'gpu'. Defaults to 'cpu'.
- **prefix** (*str, optional*) – The prefix that will be added in the metric names to disambiguate homonymous metrics of different evaluators. If prefix is not provided in the argument, self.default_prefix will be used instead. Defaults to None.

Return type None

Warning: Only non-negative integer labels are involved in computing. All negative ground truth labels will be ignored.

compute_metrics(*results*)

Compute the metrics from processed results.

Parameters **results** (*list[Dict]*) – The processed results of each batch.

Returns

The f1 scores. The keys are the names of the metrics, and the values are corresponding results. Possible keys are 'micro_f1' and 'macro_f1'.

Return type dict[str, float]

process(*data_batch*, *data_samples*)

Process one batch of data_samples. The processed results should be stored in `self.results`, which will be used to compute the metrics when all batches have been processed.

Parameters

- **data_batch** (*Sequence[Dict]*) – A batch of gts.
- **data_samples** (*Sequence[Dict]*) – A batch of outputs from the model.

Return type None

CHAPTER

FORTYEIGHT

MMOCR.VISUALIZATION

BaseLocalVisualizer	The MMOCR Text Detection Local Visualizer.
TextDetLocalVisualizer	The MMOCR Text Detection Local Visualizer.
TextRecogLocalVisualizer	MMOCR Text Detection Local Visualizer.
TextSpottingLocalVisualizer	
KIELocalVisualizer	The MMOCR Text Detection Local Visualizer.

48.1 BaseLocalVisualizer

class mmocr.visualization.**BaseLocalVisualizer**(*name='visualizer'*, *font_families='sans-serif'*, *font_properties=None*, ***kwargs*)

The MMOCR Text Detection Local Visualizer.

Parameters

- **name** (*str*) – Name of the instance. Defaults to 'visualizer'.
- **image** (*np.ndarray, optional*) – the origin image to draw. The format should be RGB. Defaults to None.
- **vis_backends** (*list, optional*) – Visual backend config list. Default to None.
- **save_dir** (*str, optional*) – Save file dir for all storage backends. If it is None, the backend storage will not save any data.
- **fig_save_cfg** (*dict*) – Keyword parameters of figure for saving. Defaults to empty dict.
- **fig_show_cfg** (*dict*) – Keyword parameters of figure for showing. Defaults to empty dict.
- **is_openset** (*bool, optional*) – Whether the visualizer is used in OpenSet. Defaults to False.
- **font_families** (*Union[str, List[str]]*) – The font families of labels. Defaults to 'sans-serif'.
- **font_properties** (*Union[str, FontProperties], optional*) – The font properties of texts. The format should be a path str to font file or a *font_manager.FontProperties()* object. If you want to draw Chinese texts, you need to prepare a font file that can show Chinese characters properly. For example: *simhei.ttf*,`simsun.ttc`,`simkai.ttf` and so on. Then set font_properties=matplotlib.font_manager.FontProperties (fname='path/to/font_file')

or font_properties='path/to/font_file' This function need mmengine version >=0.6.0. Defaults to None.

Return type None

get_bboxes_image(*image*, *bboxes*, *colors='g'*, *filling=False*, *line_width=0.5*, *alpha=0.5*)

Draw bboxes on image.

Parameters

- **image** (*np.ndarray*) – The origin image to draw. The format should be RGB.
- **bboxes** (*Union[np.ndarray, torch.Tensor]*) – The bboxes to draw.
- **colors** (*Union[str, Sequence[str]]*) – The colors of bboxes. `colors` can have the same length with bboxes or just single value. If `colors` is single value, all the bboxes will have the same colors. Refer to *matplotlib.colors* for full list of formats that are accepted. Defaults to 'g'.
- **filling** (*bool*) – Whether to fill the bboxes. Defaults to False.
- **line_width** (*Union[int, float]*) – The line width of bboxes. Defaults to 0.5.
- **alpha** (*float*) – The alpha of bboxes. Defaults to 0.5.
- **self** (*mmengine.visualization.visualizer.Visualizer*) –

Returns The image with bboxes drawn.

Return type np.ndarray

get_labels_image(*image*, *labels*, *bboxes*, *colors='k'*, *font_size=10*, *auto_font_size=False*, *font_families='sans-serif'*, *font_properties=None*)

Draw labels on image.

Parameters

- **image** (*np.ndarray*) – The origin image to draw. The format should be RGB.
- **labels** (*Union[np.ndarray, torch.Tensor]*) – The labels to draw.
- **bboxes** (*Union[np.ndarray, torch.Tensor]*) – The bboxes to draw.
- **colors** (*Union[str, Sequence[str]]*) – The colors of labels. `colors` can have the same length with labels or just single value. If `colors` is single value, all the labels will have the same colors. Refer to *matplotlib.colors* for full list of formats that are accepted. Defaults to 'k'.
- **font_size** (*Union[int, float]*) – The font size of labels. Defaults to 10.
- **auto_font_size** (*bool*) – Whether to automatically adjust font size. Defaults to False.
- **font_families** (*Union[str, List[str]]*) – The font families of labels. Defaults to 'sans-serif'.
- **font_properties** (*Union[str, FontProperties], optional*) – The font properties of texts. The format should be a path str to font file or a *font_manager.FontProperties()* object. If you want to draw Chinese texts, you need to prepare a font file that can show Chinese characters properly. For example: *simhei.ttf*,`simsun.ttc`,`simkai.ttf` and so on. Then set font_properties=matplotlib.font_manager.FontProperties (fname='path/to/font_file') or font_properties='path/to/font_file'. This function need mmengine version >=0.6.0. Defaults to None.

Return type numpy.ndarray

get_polygons_image(*image*, *polygons*, *colors='g'*, *filling=False*, *line_width=0.5*, *alpha=0.5*)
Draw polygons on image.

Parameters

- **image** (*np.ndarray*) – The origin image to draw. The format should be RGB.
- **polygons** (*Sequence[np.ndarray]*) – The polygons to draw. The shape should be (N, 2).
- **colors** (*Union[str, Sequence[str]]*) – The colors of polygons. `colors` can have the same length with polygons or just single value. If `colors` is single value, all the polygons will have the same colors. Refer to *matplotlib.colors* for full list of formats that are accepted. Defaults to 'g'.
- **filling** (*bool*) – Whether to fill the polygons. Defaults to False.
- **line_width** (*Union[int, float]*) – The line width of polygons. Defaults to 0.5.
- **alpha** (*float*) – The alpha of polygons. Defaults to 0.5.

Returns The image with polygons drawn.

Return type np.ndarray

48.2 TextDetLocalVisualizer

class mmocr.visualization.**TextDetLocalVisualizer**(*name='visualizer'*, *image=None*, *with_poly=True*, *with_bbox=False*, *vis_backends=None*, *save_dir=None*, *gt_color='g'*, *gt_ignored_color='b'*, *pred_color='r'*, *line_width=2*, *alpha=0.8*)

The MMOCR Text Detection Local Visualizer.

Parameters

- **name** (*str*) – Name of the instance. Defaults to 'visualizer'.
- **image** (*np.ndarray, optional*) – The origin image to draw. The format should be RGB. Defaults to None.
- **with_poly** (*bool*) – Whether to draw polygons. Defaults to True.
- **with_bbox** (*bool*) – Whether to draw bboxes. Defaults to False.
- **vis_backends** (*list, optional*) – Visual backend config list. Defaults to None.
- **save_dir** (*str, optional*) – Save file dir for all storage backends. If it is None, the backend storage will not save any data.
- **gt_color** (*Union[str, tuple, list[str], list[tuple]]*) – The colors of GT polygons and bboxes. `colors` can have the same length with lines or just single value. If `colors` is single value, all the lines will have the same colors. Refer to *matplotlib.colors* for full list of formats that are accepted. Defaults to 'g'.
- **gt_ignored_color** (*Union[str, tuple, list[str], list[tuple]]*) – The colors of ignored GT polygons and bboxes. `colors` can have the same length with lines or just single value. If `colors` is single value, all the lines will have the same colors. Refer to *matplotlib.colors* for full list of formats that are accepted. Defaults to 'b'.

- **pred_color** (*Union[str, tuple, list[str], list[tuple]]*) – The colors of pred polygons and bboxes. `colors` can have the same length with lines or just single value. If `colors` is single value, all the lines will have the same colors. Refer to *matplotlib.colors* for full list of formats that are accepted. Defaults to 'r'.
- **line_width** (*int, float*) – The linewidth of lines. Defaults to 2.
- **alpha** (*float*) – The transparency of bboxes or polygons. Defaults to 0.8.

Return type None

add_datasample(*name*, *image*, *data_sample=None*, *draw_gt=True*, *draw_pred=True*, *show=False*, *wait_time=0*, *out_file=None*, *pred_score_thr=0.3*, *step=0*)

Draw datasample and save to all backends.

- If GT and prediction are plotted at the same time, they are

displayed in a stitched image where the left image is the ground truth and the right image is the prediction. - If `show` is True, all storage backends are ignored, and the images will be displayed in a local window. - If `out_file` is specified, the drawn image will be saved to `out_file`. This is usually used when the display is not available.

Parameters

- **name** (*str*) – The image identifier.
- **image** (*np.ndarray*) – The image to draw.
- **data_sample** (`TextDetDataSample`, optional) –

  TextDetDataSample which contains gt and prediction. Defaults to None.
- **draw_gt** (*bool*) – Whether to draw GT TextDetDataSample. Defaults to True.
- **draw_pred** (*bool*) – Whether to draw Predicted TextDetDataSample. Defaults to True.
- **show** (*bool*) – Whether to display the drawn image. Default to False.
- **wait_time** (*float*) – The interval of show (s). Defaults to 0.
- **out_file** (*str*) – Path to output file. Defaults to None.
- **pred_score_thr** (*float*) – The threshold to visualize the bboxes and masks. Defaults to 0.3.
- **step** (*int*) – Global step value to record. Defaults to 0.

Return type None

48.3 TextRecogLocalVisualizer

class mmocr.visualization.**TextRecogLocalVisualizer**(*name='visualizer'*, *image=None*, *vis_backends=None*, *save_dir=None*, *gt_color='g'*, *pred_color='r'*, ***kwargs*)

MMOCR Text Detection Local Visualizer.

Parameters

- **name** (*str*) – Name of the instance. Defaults to 'visualizer'.
- **image** (*np.ndarray, optional*) – The origin image to draw. The format should be RGB. Defaults to None.

- **vis_backends** (*list, optional*) – Visual backend config list. Defaults to None.
- **save_dir** (*str, optional*) – Save file dir for all storage backends. If it is None, the backend storage will not save any data.
- **gt_color** (*str or tuple[int, int, int]*) – Colors of GT text. The tuple of color should be in RGB order. Or using an abbreviation of color, such as *'g'* for *'green'*. Defaults to 'g'.
- **pred_color** (*str or tuple[int, int, int]*) – Colors of Predicted text. The tuple of color should be in RGB order. Or using an abbreviation of color, such as *'r'* for *'red'*. Defaults to 'r'.

Return type None

add_datasample(*name*, *image*, *data_sample=None*, *draw_gt=True*, *draw_pred=True*, *show=False*, *wait_time=0*, *pred_score_thr=None*, *out_file=None*, *step=0*)

Visualize datasample and save to all backends.

- If GT and prediction are plotted at the same time, they are

displayed in a stitched image where the left image is the ground truth and the right image is the prediction. - If `show` is True, all storage backends are ignored, and the images will be displayed in a local window. - If `out_file` is specified, the drawn image will be saved to `out_file`. This is usually used when the display is not available.

Parameters

- **name** (*str*) – The image title. Defaults to 'image'.
- **image** (*np.ndarray*) – The image to draw.
- **data_sample** (`TextRecogDataSample`, optional) – TextRecogDataSample which contains gt and prediction. Defaults to None.
- **draw_gt** (*bool*) – Whether to draw GT TextRecogDataSample. Defaults to True.
- **draw_pred** (*bool*) – Whether to draw Predicted TextRecogDataSample. Defaults to True.
- **show** (*bool*) – Whether to display the drawn image. Defaults to False.
- **wait_time** (*float*) – The interval of show (s). Defaults to 0.
- **out_file** (*str*) – Path to output file. Defaults to None.
- **step** (*int*) – Global step value to record. Defaults to 0.
- **pred_score_thr** (*float*) – Threshold of prediction score. It's not used in this function. Defaults to None.

Return type None

48.4 TextSpottingLocalVisualizer

class mmocr.visualization.**TextSpottingLocalVisualizer**(*name='visualizer'*, *font_families='sans-serif'*, *font_properties=None*, ***kwargs*)

Parameters

- **name** (*str*) –
- **font_families** (*Union[str, List[str]]*) –
- **font_properties** (*Optional[Union[str, matplotlib.font_manager.FontProperties]]*) –

Return type None

add_datasample(*name*, *image*, *data_sample=None*, *draw_gt=True*, *draw_pred=True*, *show=False*, *wait_time=0*, *pred_score_thr=0.5*, *out_file=None*, *step=0*)

Draw datasample and save to all backends.

- If GT and prediction are plotted at the same time, they are

displayed in a stitched image where the left image is the ground truth and the right image is the prediction. - If `show` is True, all storage backends are ignored, and the images will be displayed in a local window. - If `out_file` is specified, the drawn image will be saved to `out_file`. This is usually used when the display is not available.

Parameters

- **name** (*str*) – The image identifier.
- **image** (*np.ndarray*) – The image to draw.
- **data_sample** (`TextSpottingDataSample`, optional) –

  TextDetDataSample which contains gt and prediction. Defaults to None.
- **draw_gt** (*bool*) – Whether to draw GT TextDetDataSample. Defaults to True.
- **draw_pred** (*bool*) – Whether to draw Predicted TextDetDataSample. Defaults to True.
- **show** (*bool*) – Whether to display the drawn image. Default to False.
- **wait_time** (*float*) – The interval of show (s). Defaults to 0.
- **out_file** (*str*) – Path to output file. Defaults to None.
- **pred_score_thr** (*float*) – The threshold to visualize the bboxes and masks. Defaults to 0.3.
- **step** (*int*) – Global step value to record. Defaults to 0.

Return type None

48.5 KIELocalVisualizer

class mmocr.visualization.**KIELocalVisualizer**(*name='kie_visualizer'*, *is_openset=False*, ***kwargs*)

The MMOCR Text Detection Local Visualizer.

Parameters

- **name** (*str*) – Name of the instance. Defaults to 'visualizer'.
- **image** (*np.ndarray, optional*) – the origin image to draw. The format should be RGB. Defaults to None.
- **vis_backends** (*list, optional*) – Visual backend config list. Default to None.
- **save_dir** (*str, optional*) – Save file dir for all storage backends. If it is None, the backend storage will not save any data.
- **fig_save_cfg** (*dict*) – Keyword parameters of figure for saving. Defaults to empty dict.
- **fig_show_cfg** (*dict*) – Keyword parameters of figure for showing. Defaults to empty dict.
- **is_openset** (*bool, optional*) – Whether the visualizer is used in OpenSet. Defaults to False.

Return type None

add_datasample(*name*, *image*, *data_sample=None*, *draw_gt=True*, *draw_pred=True*, *show=False*, *wait_time=0*, *pred_score_thr=None*, *out_file=None*, *step=0*)

Draw datasample and save to all backends.

- If GT and prediction are plotted at the same time, they are

displayed in a stitched image where the left image is the ground truth and the right image is the prediction. - If `show` is True, all storage backends are ignored, and the images will be displayed in a local window. - If `out_file` is specified, the drawn image will be saved to `out_file`. This is usually used when the display is not available.

Parameters

- **name** (*str*) – The image identifier.
- **image** (*np.ndarray*) – The image to draw.
- **data_sample** (`KIEDataSample`, optional) –

  KIEDataSample which contains gt and prediction. Defaults to None.
- **draw_gt** (*bool*) – Whether to draw GT KIEDataSample. Defaults to True.
- **draw_pred** (*bool*) – Whether to draw Predicted KIEDataSample. Defaults to True.
- **show** (*bool*) – Whether to display the drawn image. Default to False.
- **wait_time** (*float*) – The interval of show (s). Defaults to 0.
- **pred_score_thr** (*float*) – The threshold to visualize the bboxes and masks. Defaults to 0.3.
- **out_file** (*str*) – Path to output file. Defaults to None.
- **step** (*int*) – Global step value to record. Defaults to 0.

Return type None

draw_arrows(*x_data*, *y_data*, *colors='C1'*, *line_widths=1*, *line_styles='-'*, *arrow_tail_widths=0.001*, *arrow_head_widths=None*, *arrow_head_lengths=None*, *arrow_shapes='full'*, *overhangs=0*)

Draw single or multiple arrows.

Parameters

- **x_data** (*np.ndarray or torch.Tensor*) – The x coordinate of each line' start and end points.
- **y_data** (*np.ndarray, torch.Tensor*) – The y coordinate of each line' start and end points.
- **colors** (*str or tuple or list[str or tuple]*) – The colors of lines. `colors` can have the same length with lines or just single value. If `colors` is single value, all the lines will have the same colors. Reference to https://matplotlib.org/stable/gallery/color/named_colors.html for more details. Defaults to 'g'.
- **line_widths** (*int or float or list[int or float]*) – The linewidth of lines. `line_widths` can have the same length with lines or just single value. If `line_widths` is single value, all the lines will have the same linewidth. Defaults to 2.
- **line_styles** (*str or list[str]*) – The linestyle of lines. `line_styles` can have the same length with lines or just single value. If `line_styles` is single value, all the lines will have the same linestyle. Defaults to '-'.
- **arrow_tail_widths** (*int or float or list[int, float]*) – The width of arrow tails. `arrow_tail_widths` can have the same length with lines or just single value. If `arrow_tail_widths` is single value, all the lines will have the same width. Defaults to 0.001.
- **arrow_head_widths** (*int or float or list[int, float]*) – The width of arrow heads. `arrow_head_widths` can have the same length with lines or just single value. If `arrow_head_widths` is single value, all the lines will have the same width. Defaults to None.
- **arrow_head_lengths** (*int or float or list[int, float]*) – The length of arrow heads. `arrow_head_lengths` can have the same length with lines or just single value. If `arrow_head_lengths` is single value, all the lines will have the same length. Defaults to None.
- **arrow_shapes** (*str or list[str]*) – The shapes of arrow heads. `arrow_shapes` can have the same length with lines or just single value. If `arrow_shapes` is single value, all the lines will have the same shape. Defaults to 'full'.
- **overhangs** (*int or list[int]*) – The overhangs of arrow heads. `overhangs` can have the same length with lines or just single value. If `overhangs` is single value, all the lines will have the same overhangs. Defaults to 0.

Return type mmengine.visualization.visualizer.Visualizer

CHAPTER

FORTYNINE

MMOCR.ENGINE

mmocr.engine

- *Hooks*

49.1 Hooks

VisualizationHook	Detection Visualization Hook.

49.1.1 VisualizationHook

class mmocr.engine.hooks.**VisualizationHook**(*enable=False*, *interval=50*, *score_thr=0.3*, *show=False*, *draw_pred=False*, *draw_gt=False*, *wait_time=0.0*, *backend_args=None*)

Detection Visualization Hook. Used to visualize validation and testing process prediction results.

Parameters

- **enable** (*bool*) – Whether to enable this hook. Defaults to False.
- **interval** (*int*) – The interval of visualization. Defaults to 50.
- **score_thr** (*float*) – The threshold to visualize the bboxes and masks. It's only useful for text detection. Defaults to 0.3.
- **show** (*bool*) – Whether to display the drawn image. Defaults to False.
- **wait_time** (*float*) – The interval of show in seconds. Defaults to 0.
- **backend_args** (*dict, optional*) – Instantiates the corresponding file backend. It may contain *backend* key to specify the file backend. If it contains, the file backend corresponding to this value will be used and initialized with the remaining values, otherwise the corresponding file backend will be selected based on the prefix of the file path. Defaults to None.
- **draw_pred** (*bool*) –
- **draw_gt** (*bool*) –

Return type None

after_test_iter(*runner*, *batch_idx*, *data_batch*, *outputs*)
Run after every testing iterations.

Parameters

- **runner** (Runner) – The runner of the testing process.
- **batch_idx** (*int*) – The index of the current batch in the val loop.
- **data_batch** (*Sequence[dict]*) – Data from dataloader.
- **outputs** (*Sequence[Union[mmocr.structures.textdet_data_sample.TextDetDataSample, mmocr.structures.textrecog_data_sample.TextRecogDataSample]]*) –

Return type None

:param outputs (Sequence[TextDetDataSample or: TextRecogDataSample]): Outputs from model.

after_val_iter(*runner*, *batch_idx*, *data_batch*, *outputs*)
Run after every `self.interval` validation iterations.

Parameters

- **runner** (Runner) – The runner of the validation process.
- **batch_idx** (*int*) – The index of the current batch in the val loop.
- **data_batch** (*Sequence[dict]*) – Data from dataloader.
- **outputs** (*Sequence[Union[mmocr.structures.textdet_data_sample.TextDetDataSample, mmocr.structures.textrecog_data_sample.TextRecogDataSample]]*) –

Return type None

:param outputs (Sequence[TextDetDataSample or: TextRecogDataSample]): Outputs from model.

CHAPTER

FIFTY

MMOCR.UTILS

mmocr.utils

50.1 Image Utils

50.2 Box Utils

bbox2poly	Converting a bounding box to a polygon.
bbox_center_distance	Calculate the distance between the center points of two bounding boxes.
bbox_diag_distance	Calculate the diagonal length of a bounding box (distance between the top-left and bottom-right).
bezier2polygon	Sample points from the boundary of a polygon enclosed by two Bezier curves, which are controlled by `bezier_points`.
is_on_same_line	Check if two boxes are on the same line by their y-axis coordinates.
rescale_bboxes	Rescale bboxes according to scale_factor.
stitch_boxes_into_lines	Stitch fragmented boxes of words into lines.

50.2.1 mmocr.utils.bbox2poly

mmocr.utils.**bbox2poly**(*bbox*, *mode='xyxy'*)

Converting a bounding box to a polygon.

Parameters

- **bbox** (*ArrayLike*) – A bbox. In any form can be accessed by 1-D indices. E.g. list[float], np.ndarray, or torch.Tensor. bbox is written in

  [x1, y1, x2, y2].

- **mode** (*str*) – Specify the format of bbox. Can be 'xyxy' or 'xywh'. Defaults to 'xyxy'.

Returns The converted polygon [x1, y1, x2, y1, x2, y2, x1, y2].

Return type np.array

50.2.2 mmocr.utils.bbox_center_distance

mmocr.utils.**bbox_center_distance**(*box1*, *box2*)

Calculate the distance between the center points of two bounding boxes.

Parameters

- **box1** (*ArrayLike*) – The first bounding box represented in [x1, y1, x2, y2].
- **box2** (*ArrayLike*) – The second bounding box represented in [x1, y1, x2, y2].

Returns The distance between the center points of two bounding boxes.

Return type float

50.2.3 mmocr.utils.bbox_diag_distance

mmocr.utils.**bbox_diag_distance**(*box*)

Calculate the diagonal length of a bounding box (distance between the top-left and bottom-right).

Parameters

- **box** (*ArrayLike*) – The bounding box represented in
- **[x1** –
- **y1** –
- **x2** –
- **y2** –
- **x3** –
- **y3** –
- **x4** –
- **or [x1** (*y4]*) –
- **y1** –
- **x2** –
- **y2]** –

Returns The diagonal length of the bounding box.

Return type float

50.2.4 mmocr.utils.bezier2polygon

mmocr.utils.**bezier2polygon**(*bezier_points*, *num_sample=20*)

Sample points from the boundary of a polygon enclosed by two Bezier curves, which are controlled by `bezier_points`.

Parameters

- **bezier_points** (*ndarray*) – A $(2, 4, 2)$ array of 8 Bezeir points or its equalivance. The first 4 points control the curve at one side and the last four control the other side.
- **num_sample** (*int*) – The number of sample points at each Bezeir curve. Defaults to 20.

Returns A list of 2*num_sample points representing the polygon extracted from Bezier curves.

Return type list[ndarray]

Warning: The points are not guaranteed to be ordered. Please use *mmocr.utils.sort_points()* to sort points if necessary.

50.2.5 mmocr.utils.is_on_same_line

mmocr.utils.**is_on_same_line**(*box_a*, *box_b*, *min_y_overlap_ratio=0.8*)

Check if two boxes are on the same line by their y-axis coordinates.

Two boxes are on the same line if they overlap vertically, and the length of the overlapping line segment is greater than min_y_overlap_ratio * the height of either of the boxes.

Parameters

- **box_a** (*list), box_b (list*) – Two bounding boxes to be checked
- **min_y_overlap_ratio** (*float*) – The minimum vertical overlapping ratio allowed for boxes in the same line

Returns The bool flag indicating if they are on the same line

50.2.6 mmocr.utils.rescale_bboxes

mmocr.utils.**rescale_bboxes**(*bboxes*, *scale_factor*, *mode='mul'*)

Rescale bboxes according to scale_factor.

The behavior is different depending on the mode. When mode is 'mul', the coordinates will be multiplied by scale_factor, which is usually used in preprocessing transforms such as `Resize()`. The coordinates will be divided by scale_factor if mode is 'div'. It can be used in postprocessors to recover the bboxes in the original image size.

Parameters

- **bboxes** (*np.ndarray]*) – Bounding bboxes in shape (N, 4)
- **scale_factor** (*tuple(int, int)*) – (w_scale, h_scale).

- **model** (*str*) – Rescale mode. Can be 'mul' or 'div'. Defaults to 'mul'.
- **mode** (*str*) –

Returns Rescaled bboxes.

Return type list[np.ndarray]

50.2.7 mmocr.utils.stitch_boxes_into_lines

mmocr.utils.**stitch_boxes_into_lines**(*boxes*, *max_x_dist=10*, *min_y_overlap_ratio=0.8*)

Stitch fragmented boxes of words into lines.

Note: part of its logic is inspired by @Johndirr (https://github.com/faustomorales/keras-ocr/issues/22)

Parameters

- **boxes** (*list*) – List of ocr results to be stitched
- **max_x_dist** (*int*) – The maximum horizontal distance between the closest edges of neighboring boxes in the same line
- **min_y_overlap_ratio** (*float*) – The minimum vertical overlapping ratio allowed for any pairs of neighboring boxes in the same line

Returns List of merged boxes and texts

Return type merged_boxes(list[dict])

50.3 Point Utils

point_distance	Calculate the distance between two points.
points_center	Calculate the center of a set of points.

50.3.1 mmocr.utils.point_distance

mmocr.utils.**point_distance**(*pt1*, *pt2*)

Calculate the distance between two points.

Parameters

- **pt1** (*ArrayLike*) – The first point.
- **pt2** (*ArrayLike*) – The second point.

Returns The distance between two points.

Return type float

50.3.2 mmocr.utils.points_center

mmocr.utils.**points_center**(*points*)

Calculate the center of a set of points.

Parameters **points** (*ArrayLike*) – A set of points.

Returns The coordinate of center point.

Return type np.ndarray

50.4 Polygon Utils

boundary_iou	Calculate the IOU between two boundaries.
crop_polygon	Crop polygon to be within a box region.
is_poly_inside_rect	Check if the polygon is inside the target region.
offset_polygon	Offset (expand/shrink) the polygon by the target distance.
poly2bbox	Converting a polygon to a bounding box.
poly2shapely	Convert a polygon to shapely.geometry.Polygon.
poly_intersection	Calculate the intersection area between two polygons.
poly_iou	Calculate the IOU between two polygons.
poly_make_valid	Convert a potentially invalid polygon to a valid one by eliminating self-crossing or self-touching parts.
poly_union	Calculate the union area between two polygons.
polys2shapely	Convert a nested list of boundaries to a list of Polygons.
rescale_polygon	Rescale a polygon according to scale_factor.
rescale_polygons	Rescale polygons according to scale_factor.
shapely2poly	Convert a nested list of boundaries to a list of Polygons.
sort_points	Sort arbitrary points in clockwise order in Cartesian coordinate, you may need to reverse the output sequence if you are using OpenCV's image coordinate.
sort_vertex	Sort box vertices in clockwise order from left-top first.
sort_vertex8	Sort vertex with 8 points [x1 y1 x2 y2 x3 y3 x4 y4]

50.4.1 mmocr.utils.boundary_iou

mmocr.utils.**boundary_iou**(*src*, *target*, *zero_division=0*)

Calculate the IOU between two boundaries.

Parameters

- **src** (*list*) – Source boundary.
- **target** (*list*) – Target boundary.
- **zero_division** (*int or float*) – The return value when invalid boundary exists.

Returns The iou between two boundaries.

Return type float

50.4.2 mmocr.utils.crop_polygon

mmocr.utils.**crop_polygon**(*polygon*, *crop_box*)

Crop polygon to be within a box region.

Parameters

- **polygon** (*ndarray*) – polygon in shape (N,).
- **crop_box** (*ndarray*) – target box region in shape (4,).

Returns

Cropped polygon. If the polygon is not within the crop box, return None.

Return type np.array or None

50.4.3 mmocr.utils.is_poly_inside_rect

mmocr.utils.**is_poly_inside_rect**(*poly*, *rect*)

Check if the polygon is inside the target region. :param poly: Polygon in shape (N,). :type poly: ArrayLike :param rect: Target region [x1, y1, x2, y2]. :type rect: ndarray

Returns Whether the polygon is inside the cropping region.

Return type bool

Parameters

- **poly** (*Union[Sequence[Sequence[Sequence[Sequence[Sequence[Any]]]]], numpy.typing._array_like._SupportsArray[numpy.dtype], Sequence[numpy.typing._array_like._SupportsArray[numpy.dtype]], Sequence[Sequence[numpy.typing._array_like._SupportsArray[numpy.dtype]]], Sequence[Sequence[Sequence[numpy.typing._array_like._SupportsArray[numpy.dtype]]]], Sequence[Sequence[Sequence[Sequence[numpy.typing._array_like._SupportsArray[numpy.dtype]]]]], bool, int, float, complex, str, bytes, Sequence[Union[bool, int, float, complex, str, bytes]], Sequence[Sequence[Union[bool, int, float, complex, str, bytes]]], Sequence[Sequence[Sequence[Union[bool, int, float, complex, str, bytes]]]], Sequence[Sequence[Sequence[Sequence[Union[bool, int, float, complex, str, bytes]]]]]]*) –
- **rect** (*numpy.ndarray*) –

50.4.4 mmocr.utils.offset_polygon

mmocr.utils.**offset_polygon**(*poly*, *distance*)

Offset (expand/shrink) the polygon by the target distance. It's a wrapper around pyclipper based on Vatti clipping algorithm.

> **Warning:** Polygon coordinates will be casted to int type in PyClipper. Mind the potential precision loss caused by the casting.

Parameters

- **poly** (*ArrayLike*) – A polygon. In any form can be converted to an 1-D numpy array. E.g. list[float], np.ndarray, or torch.Tensor. Polygon is written in [x1, y1, x2, y2, ...].
- **distance** (*float*) – The offset distance. Positive value means expanding, negative value means shrinking.

Returns 1-D Offsetted polygon ndarray in float32 type. If the result polygon is invalid or has been split into several parts, return an empty array.

Return type np.array

50.4.5 mmocr.utils.poly2bbox

mmocr.utils.**poly2bbox**(*polygon*)

Converting a polygon to a bounding box.

Parameters **polygon** – A polygon. In any form can be converted to an 1-D numpy array. E.g. list[float], np.ndarray, or torch.Tensor. Polygon is written in [x1, y1, x2, y2, ...].

Return type numpy.array

50.4.6 mmocr.utils.poly2shapely

mmocr.utils.**poly2shapely**(*polygon*)

Convert a polygon to shapely.geometry.Polygon.

Parameters **polygon** (*ArrayLike*) – A set of points of 2k shape.

Returns A polygon object.

Return type polygon (Polygon)

50.4.7 mmocr.utils.poly_intersection

mmocr.utils.**poly_intersection**(*poly_a*, *poly_b*, *invalid_ret=None*, *return_poly=False*)

Calculate the intersection area between two polygons.

Parameters

- **poly_a** (*Polygon*) – Polygon a.
- **poly_b** (*Polygon*) – Polygon b.
- **invalid_ret** (*float or int, optional*) – The return value when the invalid polygon exists. If it is not specified, the function allows the computation to proceed with invalid polygons by cleaning the their self-touching or self-crossing parts. Defaults to None.
- **return_poly** (*bool*) – Whether to return the polygon of the intersection Defaults to False.

Returns Returns the intersection area or a tuple `(area, Optional[poly_obj])`, where the *area* is the intersection area between two polygons and *poly_obj* is The Polygon object of the intersection area, which will be *None* if the input is invalid. *poly_obj* will be returned only if *return_poly* is *True*.

Return type float or tuple(float, Polygon)

50.4.8 mmocr.utils.poly_iou

mmocr.utils.**poly_iou**(*poly_a*, *poly_b*, *zero_division=0.0*)

Calculate the IOU between two polygons.

Parameters

- **poly_a** (*Polygon*) – Polygon a.
- **poly_b** (*Polygon*) – Polygon b.
- **zero_division** (*float*) – The return value when invalid polygon exists.

Returns The IoU between two polygons.

Return type float

50.4.9 mmocr.utils.poly_make_valid

mmocr.utils.**poly_make_valid**(*poly*)

Convert a potentially invalid polygon to a valid one by eliminating self-crossing or self-touching parts. Note that if the input is a line, the returned polygon could be an empty one.

Parameters **poly** (*Polygon*) – A polygon needed to be converted.

Returns A valid polygon, which might be empty.

Return type Polygon

50.4.10 mmocr.utils.poly_union

mmocr.utils.**poly_union**(*poly_a*, *poly_b*, *invalid_ret=None*, *return_poly=False*)

Calculate the union area between two polygons.

Parameters

- **poly_a** (*Polygon*) – Polygon a.
- **poly_b** (*Polygon*) – Polygon b.
- **invalid_ret** (*float or int, optional*) – The return value when the invalid polygon exists. If it is not specified, the function allows the computation to proceed with invalid polygons by cleaning the their self-touching or self-crossing parts. Defaults to False.
- **return_poly** (*bool*) – Whether to return the polygon of the union. Defaults to False.

Returns Returns a tuple `(area, Optional[poly_obj])`, where the *area* is the union between two polygons and *poly_obj* is the Polygon or MultiPolygon object of the union of the inputs. The type of object depends on whether they intersect or not. Set as *None* if the input is invalid. *poly_obj* will be returned only if *return_poly* is *True*.

Return type tuple

50.4.11 mmocr.utils.polys2shapely

mmocr.utils.**polys2shapely**(*polygons*)

Convert a nested list of boundaries to a list of Polygons.

Parameters **polygons** (*list*) – The point coordinates of the instance boundary.

Returns Converted shapely.Polygon.

Return type list

50.4.12 mmocr.utils.rescale_polygon

mmocr.utils.**rescale_polygon**(*polygon*, *scale_factor*, *mode='mul'*)

Rescale a polygon according to scale_factor.

The behavior is different depending on the mode. When mode is 'mul', the coordinates will be multiplied by scale_factor, which is usually used in preprocessing transforms such as `Resize()`. The coordinates will be divided by scale_factor if mode is 'div'. It can be used in postprocessors to recover the polygon in the original image size.

Parameters

- **polygon** (*ArrayLike*) – A polygon. In any form can be converted to an 1-D numpy array. E.g. list[float], np.ndarray, or torch.Tensor. Polygon is written in [x1, y1, x2, y2, ...].
- **scale_factor** (*tuple(int, int)*) – (w_scale, h_scale).
- **model** (*str*) – Rescale mode. Can be 'mul' or 'div'. Defaults to 'mul'.
- **mode** (*str*) –

Returns Rescaled polygon.

Return type np.ndarray

50.4.13 mmocr.utils.rescale_polygons

mmocr.utils.**rescale_polygons**(*polygons*, *scale_factor*, *mode='mul'*)

Rescale polygons according to scale_factor.

The behavior is different depending on the mode. When mode is 'mul', the coordinates will be multiplied by scale_factor, which is usually used in preprocessing transforms such as `Resize()`. The coordinates will be divided by scale_factor if mode is 'div'. It can be used in postprocessors to recover the polygon in the original image size.

Parameters

- **polygons** (*list[ArrayLike] or ArrayLike*) – A list of polygons, each written in [x1, y1, x2, y2, ...] and in any form can be converted to an 1-D numpy array. E.g. list[list[float]], list[np.ndarray], or list[torch.Tensor].
- **scale_factor** (*tuple(int, int)*) – (w_scale, h_scale).
- **model** (*str*) – Rescale mode. Can be 'mul' or 'div'. Defaults to 'mul'.
- **mode** (*str*) –

Returns Rescaled polygons. The type of the return value depends on the type of the input polygons.

Return type list[np.ndarray] or np.ndarray

50.4.14 mmocr.utils.shapely2poly

mmocr.utils.**shapely2poly**(*polygon*)
Convert a nested list of boundaries to a list of Polygons.

Parameters **polygon** (*Polygon*) – A polygon represented by shapely.Polygon.

Returns Converted numpy array

Return type np.array

50.4.15 mmocr.utils.sort_points

mmocr.utils.**sort_points**(*points*)
Sort arbitrary points in clockwise order in Cartesian coordinate, you may need to reverse the output sequence if you are using OpenCV's image coordinate.

Reference: https://github.com/novioleo/Savior/blob/master/Utils/GeometryUtils.py.

Warning: This function can only sort convex polygons.

Parameters **points** (*list[ndarray] or ndarray or list[list]*) – A list of unsorted boundary points.

Returns A list of points sorted in clockwise order.

Return type list[ndarray]

50.4.16 mmocr.utils.sort_vertex

mmocr.utils.**sort_vertex**(*points_x*, *points_y*)
Sort box vertices in clockwise order from left-top first.

Parameters

- **points_x** (*list[float]*) – x of four vertices.
- **points_y** (*list[float]*) – y of four vertices.

Returns

Sorted x and y of four vertices.

- sorted_points_x (list[float]): x of sorted four vertices.
- sorted_points_y (list[float]): y of sorted four vertices.

Return type tuple[list[float], list[float]]

50.4.17 mmocr.utils.sort_vertex8

mmocr.utils.**sort_vertex8**(*points*)
Sort vertex with 8 points [x1 y1 x2 y2 x3 y3 x4 y4]

50.5 Mask Utils

fill_hole	Fill holes in matrix.

50.5.1 mmocr.utils.fill_hole

mmocr.utils.**fill_hole**(*input_mask*)

Fill holes in matrix.

Input:

[[0, 0, 0, 0, 0, 0, 0], [0, 1, 1, 1, 1, 1, 0], [0, 1, 0, 0, 0, 1, 0], [0, 1, 1, 1, 1, 1, 0], [0, 0, 0, 0, 0, 0, 0]]

Output:

[[0, 0, 0, 0, 0, 0, 0], [0, 1, 1, 1, 1, 1, 0], [0, 1, 1, 1, 1, 1, 0], [0, 1, 1, 1, 1, 1, 0], [0, 0, 0, 0, 0, 0, 0]]

Parameters **input_mask** (*ArrayLike*) – The input mask.

Returns The output mask that has been filled.

Return type np.array

50.6 Misc Utils

equal_len	
is_2dlist	check x is 2d-list([[1], []]) or 1d empty list([]).
is_3dlist	check x is 3d-list([[[1], []]]) or 2d empty list([[], []]) or 1d empty list([]).
is_none_or_type	
is_type_list	

50.6.1 mmocr.utils.equal_len

mmocr.utils.**equal_len**(**argv*)

50.6.2 mmocr.utils.is_2dlist

mmocr.utils.**is_2dlist**(*x*)
check x is 2d-list([[1], []]) or 1d empty list([]).

Notice: The reason that it contains 1d empty list is because some arguments from gt annotation file or model prediction may be empty, but usually, it should be 2d-list.

50.6.3 mmocr.utils.is_3dlist

mmocr.utils.**is_3dlist**(*x*)
check x is 3d-list([[[1], []]]) or 2d empty list([[], []]) or 1d empty list([]).

Notice: The reason that it contains 1d or 2d empty list is because some arguments from gt annotation file or model prediction may be empty, but usually, it should be 3d-list.

50.6.4 mmocr.utils.is_none_or_type

mmocr.utils.**is_none_or_type**(*x*, *type*)

50.6.5 mmocr.utils.is_type_list

mmocr.utils.**is_type_list**(*x*, *type*)

50.7 Setup Env

register_all_modules	Register all modules in mmocr into the registries.

50.7.1 mmocr.utils.register_all_modules

mmocr.utils.**register_all_modules**(*init_default_scope=True*)
Register all modules in mmocr into the registries.

Parameters **init_default_scope** (*bool*) – Whether initialize the mmocr default scope. When *init_default_scope=True*, the global default scope will be set to *mmocr*, and all registries will build modules from mmocr's registry node. To understand more about the registry, please refer to https://github.com/open-mmlab/mmengine/blob/main/docs/en/tutorials/registry.md Defaults to True.

Return type None

CHAPTER

FIFTYONE

WELCOME TO THE OPENMMLAB COMMUNITY

Scan the QR code below to follow the OpenMMLab team's **Zhihu Official Account** and join the OpenMMLab team's **QQ Group**, or join the official communication WeChat group by adding the WeChat, or join our **Slack**

We will provide you with the OpenMMLab community

- share the latest core technologies of AI frameworks
- Explaining PyTorch common module source Code
- News related to the release of OpenMMLab
- Introduction of cutting-edge algorithms developed by OpenMMLab Get the more efficient answer and feedback
- Provide a platform for communication with developers from all walks of life

The OpenMMLab community looks forward to your participation!

CHAPTER

FIFTYTWO

ENGLISH

CHAPTER

FIFTYTHREE

CHAPTER

FIFTYFOUR

INDICES AND TABLES

- genindex
- search

PYTHON MODULE INDEX

m

INDEX

H

I

K

L

M

N

O

P

R

S

T

U

V

W