mmcv Documentation

Release 2.2.0

MMCV Contributors

Apr 25, 2024

GET STARTED

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

CHAPTER

ONE

INTRODUCTION

MMCV is a foundational library for computer vision research and provides the following functionalities.

- *Image/Video processing*
- *Image and annotation visualization*
- *Image transformation*
- *Various CNN architectures*
- *High-quality implementation of common CUDA ops*

It supports the following systems:

- Linux
- Windows
- macOS

It supports many research projects as below:

- MMClassification: OpenMMLab image classification toolbox and benchmark.
- MMDetection: OpenMMLab detection toolbox and benchmark.
- MMDetection3D: OpenMMLab's next-generation platform for general 3D object detection.
- MMRotate: OpenMMLab rotated object detection toolbox and benchmark.
- MMYOLO: OpenMMLab YOLO series toolbox and benchmark.
- MMSegmentation: OpenMMLab semantic segmentation toolbox and benchmark.
- MMOCR: OpenMMLab text detection, recognition, and understanding toolbox.
- MMPose: OpenMMLab pose estimation toolbox and benchmark.
- MMHuman3D: OpenMMLab 3D human parametric model toolbox and benchmark.
- MMSelfSup: OpenMMLab self-supervised learning toolbox and benchmark.
- MMRazor: OpenMMLab model compression toolbox and benchmark.
- MMFewShot: OpenMMLab fewshot learning toolbox and benchmark.
- MMAction2: OpenMMLab's next-generation action understanding toolbox and benchmark.
- MMTracking: OpenMMLab video perception toolbox and benchmark.
- MMFlow: OpenMMLab optical flow toolbox and benchmark.
- MMEditing: OpenMMLab image and video editing toolbox.

- MMGeneration: OpenMMLab image and video generative models toolbox.
- MMDeploy: OpenMMLab model deployment framework.

CHAPTER

TWO

INSTALLATION

There are two versions of MMCV:

- **mmcv**: comprehensive, with full features and various CUDA ops out of box. It takes longer time to build.
- **mmcv-lite**: lite, without CUDA ops but all other features, similar to mmcv<1.0.0. It is useful when you do not need those CUDA ops.

Warning: Do not install both versions in the same environment, otherwise you may encounter errors like `ModuleNotFound`. You need to uninstall one before installing the other. `Installing the full version is highly recommended if CUDA is avaliable`.

2.1 Install mmcv

Before installing mmcv, make sure that PyTorch has been successfully installed following the PyTorch official installation guide. This can be verified using the following command

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

If version information is output, then PyTorch is installed.

2.1.1 Install with mim (recommended)

mim is the package management tool for the OpenMMLab projects, which makes it easy to install mmcv

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

If you find that the above installation command does not use a pre-built package ending with `.whl` but a source package ending with `.tar.gz`, you may not have a pre-build package corresponding to the PyTorch or CUDA or mmcv version, in which case you can *build mmcv from source*.

Looking in links: https://download.openmmlab.com/mmcv/dist/cu102/torch1.8.0/index.html Collecting mmcv Downloading https://download.openmmlab.com/mmcv/dist/cu102/torch1.8.0/mmcv-2.0.0-cp38-cp38-manylinux1_x86_64.whl

Looking in links: https://download.openmmlab.com/mmcv/dist/cu102/torch1.8.0/index.html Collecting mmcv==2.0.0 Downloading mmcv-2.0.0.tar.gz

To install a specific version of mmcv, for example, mmcv version 2.0.0, you can use the following command

```
mim install mmcv==2.0.0
```

Note: If you would like to use `opencv-python-headless` instead of `opencv-python`, e.g., in a minimum container environment or servers without GUI, you can first install it before installing MMCV to skip the installation of `opencv-python`.

Alternatively, if it takes too long to install a dependency library, you can specify the pypi source

```
mim install mmcv -i https://pypi.tuna.tsinghua.edu.cn/simple
```

You can run check_installation.py to check the installation of mmcv-full after running the installation commands.

2.1.2 Install with pip

Use the following command to check the version of CUDA and PyTorch

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

Select the appropriate installation command depending on the type of system, CUDA version, PyTorch version, and MMCV version

If you do not find a corresponding version in the dropdown box above, you probably do not have a pre-built package corresponding to the PyTorch or CUDA or mmcv version, at which point you can *build mmcv from source*.

Note: mmcv is only compiled on PyTorch 1.x.0 because the compatibility usually holds between 1.x.0 and 1.x.1. If your PyTorch version is 1.x.1, you can install mmcv compiled with PyTorch 1.x.0 and it usually works well. For example, if your PyTorch version is 1.8.1, you can feel free to choose 1.8.x.

Note: If you would like to use `opencv-python-headless` instead of `opencv-python`, e.g., in a minimum container environment or servers without GUI, you can first install it before installing MMCV to skip the installation of `opencv-python`.

Alternatively, if it takes too long to install a dependency library, you can specify the pypi source

```
mim install mmcv -i https://pypi.tuna.tsinghua.edu.cn/simple
```

You can run check_installation.py to check the installation of mmcv after running the installation commands.

2.1.3 Using mmcv with Docker

Build with local repository

```
git clone https://github.com/open-mmlab/mmcv.git && cd mmcv
docker build -t mmcv -f docker/release/Dockerfile .
```

Or build with remote repository

```
docker build -t mmcv https://github.com/open-mmlab/mmcv.git#main:docker/release
```

The Dockerfile installs latest released version of mmcv-full by default, but you can specify mmcv versions to install expected versions.

```
docker image build -t mmcv -f docker/release/Dockerfile --build-arg MMCV=2.0.0 .
```

If you also want to use other versions of PyTorch and CUDA, you can also pass them when building docker images.

An example to build an image with PyTorch 1.11 and CUDA 11.3.

```
docker build -t mmcv -f docker/release/Dockerfile \
    --build-arg PYTORCH=1.11.0 \
    --build-arg CUDA=11.3 \
    --build-arg CUDNN=8 \
    --build-arg MMCV=2.0.0 .
```

More available versions of PyTorch and CUDA can be found at dockerhub/pytorch.

2.2 Install mmcv-lite

If you need to use PyTorch-related modules, make sure PyTorch has been successfully installed in your environment by referring to the PyTorch official installation guide.

```
pip install mmcv-lite
```

CHAPTER

THREE

BUILD MMCV FROM SOURCE

3.1 Build mmcv

Before installing mmcv, make sure that PyTorch has been successfully installed following the PyTorch official installation guide. This can be verified using the following command

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

If version information is output, then PyTorch is installed.

Note: If you would like to use `opencv-python-headless` instead of `opencv-python`, e.g., in a minimum container environment or servers without GUI, you can first install it before installing MMCV to skip the installation of `opencv-python`.

3.1.1 Build on Linux

1. Clone the repo

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

2. Install `ninja` and `psutil` to speed up the compilation

   ```
   pip install -r requirements/optional.txt
   ```

3. Check the nvcc version (requires 9.2+. Skip if no GPU available.)

   ```
   nvcc --version
   ```

   If the above command outputs the following message, it means that the nvcc setting is OK, otherwise you need to set CUDA_HOME.

   ```
   nvcc: NVIDIA (R) Cuda compiler driver
   Copyright (c) 2005-2020 NVIDIA Corporation
   Built on Mon_Nov_30_19:08:53_PST_2020
   Cuda compilation tools, release 11.2, V11.2.67
   Build cuda_11.2.r11.2/compiler.29373293_0
   ```

Note: If you want to support ROCm, you can refer to AMD ROCm to install ROCm.

4. Check the gcc version (requires 5.4+)

```
gcc --version
```

5. Start building (takes 10+ min)

```
pip install -e . -v
```

6. Validate the installation

```
python .dev_scripts/check_installation.py
```

If no error is reported by the above command, the installation is successful. If there is an error reported, please check *Frequently Asked Questions* to see if there is already a solution.

If no solution is found, please feel free to open an issue.

3.1.2 Build on macOS

Note: If you are using a mac with apple silicon chip, install the PyTorch 1.13+, otherwise you will encounter the problem in issues#2218.

1. Clone the repo

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

2. Install `ninja` and `psutil` to speed up the compilation

```
pip install -r requirements/optional.txt
```

3. Start building

```
MMCV_WITH_OPS=1 pip install -e .
```

4. Validate the installation

```
python .dev_scripts/check_installation.py
```

If no error is reported by the above command, the installation is successful. If there is an error reported, please check *Frequently Asked Questions* to see if there is already a solution.

If no solution is found, please feel free to open an issue.

3.1.3 Build on Windows

Building MMCV on Windows is a bit more complicated than that on Linux. The following instructions show how to get this accomplished.

Prerequisite

The following software is required for building MMCV on windows. Install them first.

- Git
  - During installation, tick **add git to Path**.
- Visual Studio Community 2019
  - A compiler for C++ and CUDA codes.
- Miniconda
  - Official distributions of Python should work too.
- CUDA 10.2
  - Not required for building CPU version.
  - Customize the installation if necessary. As a recommendation, skip the driver installation if a newer version is already installed.

Note: You should know how to set up environment variables, especially `Path`, on Windows. The following instruction relies heavily on this skill.

Common steps

1. Launch Anaconda prompt from Windows Start menu

   Do not use raw `cmd.exe` s instruction is based on PowerShell syntax.

2. Create a new conda environment

```
(base) PS C:\Users\xxx> conda create --name mmcv python=3.7
(base) PS C:\Users\xxx> conda activate mmcv  # make sure to activate environment␣
↪before any operation
```

3. Install PyTorch. Choose a version based on your need.

```
# CUDA version
(mmcv) PS C:\Users\xxx> conda install pytorch torchvision cudatoolkit=10.2 -c␣
↪pytorch
# CPU version
(mmcv) PS C:\Users\xxx> conda install install pytorch torchvision cpuonly -c pytorch
```

4. Clone the repo

```
(mmcv) PS C:\Users\xxx> git clone https://github.com/open-mmlab/mmcv.git
(mmcv) PS C:\Users\xxx\mmcv> cd mmcv
```

5. Install `ninja` and `psutil` to speed up the compilation

```
(mmcv) PS C:\Users\xxx\mmcv> pip install -r requirements/optional.txt
```

6. Set up MSVC compiler

   Set Environment variable, add `C:\Program Files (x86)\Microsoft Visual Studio\2019\Community\VC\Tools\MSVC\14.27.29110\bin\Hostx86\x64` to `PATH`, so that `cl.exe` will be available in prompt, as shown below.

```
(mmcv) PS C:\Users\xxx\mmcv> cl
Microsoft (R) C/C++ Optimizing  Compiler Version 19.27.29111 for x64
Copyright (C) Microsoft Corporation.   All rights reserved.

usage: cl [ option... ] filename... [ / link linkoption... ]
```

   For compatibility, we use the x86-hosted and x64-targeted compiler. note `Hostx86\x64` in the path.

   You may want to change the system language to English because pytorch will parse text output from `cl.exe` to check its version. However only utf-8 is recognized. Navigate to Control Panel -> Region -> Administrative -> Language for Non-Unicode programs and change it to English.

Build and install MMCV

mmcv can be built in two ways:

1. Full version (CPU ops)

   Module `ops` will be compiled as a pytorch extension, but only x86 code will be compiled. The compiled ops can be executed on CPU only.

2. Full version (CUDA ops)

   Both x86 and CUDA codes of `ops` module will be compiled. The compiled version can be run on both CPU and CUDA-enabled GPU (if implemented).

CPU version

Build and install

```
(mmcv) PS C:\Users\xxx\mmcv> python setup.py build_ext
(mmcv) PS C:\Users\xxx\mmcv> python setup.py develop
```

GPU version

1. Make sure `CUDA_PATH` or `CUDA_HOME` is already set in `envs` via `ls env:`, desired output is shown as below:

```
(mmcv) PS C:\Users\xxx\mmcv> ls env:

Name                           Value
----                           -----
CUDA_PATH                      C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\
↪v10.2
```

(continues on next page)

(continued from previous page)

```
CUDA_PATH_V10_1                 C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\
↪v10.1
CUDA_PATH_V10_2                 C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\
↪v10.2
```

This should already be done by CUDA installer. If not, or you have multiple version of CUDA toolkit installed, set it with

```
(mmcv) PS C:\Users\xxx\mmcv> $env:CUDA_HOME = "C:\Program Files\NVIDIA GPU␣
↪Computing Toolkit\CUDA\v10.2"
# OR
(mmcv) PS C:\Users\xxx\mmcv> $env:CUDA_HOME = $env:CUDA_PATH_V10_2 # if CUDA_PATH_
↪V10_2 is in envs:
```

2. Set CUDA target arch

```
# Here you need to change to the target architecture corresponding to your GPU
(mmcv) PS C:\Users\xxx\mmcv> $env:TORCH_CUDA_ARCH_LIST="7.5"
```

Note: Check your the compute capability of your GPU from here.

```
(mmcv) PS C:\Users\xxx\mmcv> &"C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\
↪v10.2\extras\demo_suite\deviceQuery.exe"
Device 0: "NVIDIA GeForce GTX 1660 SUPER"
CUDA Driver Version / Runtime Version          11.7 / 11.1
CUDA Capability Major/Minor version number:    7.5
```

The 7.5 above indicates the target architecture. Note: You need to replace v10.2 with your CUDA version in the above command.

---

3. Build and install

```
# build
python setup.py build_ext # if success, cl will be launched to compile ops
# install
python setup.py develop
```

Note: If you are compiling against PyTorch 1.6.0, you might meet some errors from PyTorch as described in this issue. Follow this pull request to modify the source code in your local PyTorch installation.

Validate installation

```
(mmcv) PS C:\Users\xxx\mmcv> python .dev_scripts/check_installation.py
```

If no error is reported by the above command, the installation is successful. If there is an error reported, please check *Frequently Asked Questions* to see if there is already a solution. If no solution is found, please feel free to open an issue.

3.2 Build mmcv-lite

If you need to use PyTorch-related modules, make sure PyTorch has been successfully installed in your environment by referring to the PyTorch official installation guide.

1. Clone the repo

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

2. Start building

   ```
   MMCV_WITH_OPS=0 pip install -e . -v
   ```

3. Validate installation

   ```
   python -c 'import mmcv;print(mmcv.__version__)'
   ```

3.3 Build mmcv-full on Cambricon MLU Devices

3.3.1 Install torch_mlu

Option1: Install mmcv-full based on Cambricon docker image

Firstly, install and pull Cambricon docker image (please email service@cambricon.com for the latest release docker):

```
docker pull ${docker image}
```

Run and attach to the docker, *Install mmcv-full on MLU device* and *make sure you've installed mmcv-full on MLU device successfully*

Option2: Install mmcv-full from compiling Cambricon PyTorch source code

Please email service@cambricon.com or contact with Cambricon engineers for a suitable version of CATCH package. After you get the suitable version of CATCH package, please follow the steps in ${CATCH-path}/CONTRIBUTING.md to install Cambricon PyTorch.

3.3.2 Install mmcv-full on Cambricon MLU device

Clone the repo

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

The mlu-ops library will be downloaded to the default directory (mmcv/mlu-ops) while building MMCV. You can also set `MMCV_MLU_OPS_PATH` to an existing mlu-ops library before building as follows:

```
export MMCV_MLU_OPS_PATH=/xxx/xxx/mlu-ops
```

Install mmcv-full

```
cd mmcv
export MMCV_WITH_OPS=1
export FORCE_MLU=1
python setup.py install
```

3.3.3 Test Code

After finishing previous steps, you can run the following python code to make sure that you've installed mmcv-full on MLU device successfully

```
import torch
import torch_mlu
from mmcv.ops import sigmoid_focal_loss
x = torch.randn(3, 10).mlu()
x.requires_grad = True
y = torch.tensor([1, 5, 3]).mlu()
w = torch.ones(10).float().mlu()
output = sigmoid_focal_loss(x, y, 2.0, 0.25, w, 'none')
print(output)
```

CHAPTER

FOUR

API REFERENCE TABLE

Due to the removal of the `mmcv.fileio`, `mmcv.runner`, `mmcv.parallel`, `mmcv.engine`, `mmcv.device` modules, and all classes and most of the functions in the `mmcv.utils` module during the upgrade from MMCV v1.x to MMCV v2.x, which were removed at PR #2179, PR #2216, PR #2217. Therefore, we provide the following API reference table to make it easier to quickly find the migrated interfaces.

4.1 Related issues, PRs and discussions

- Remove runner, parallel, engine and device
- ImportError: cannot import name 'is_list_of' from 'mmcv.utils'
- Could not find the files in MMengine which are removed in MMCV_v2x parallel. example, for DataContainer
- mmcv.cnn.bricks.registry
- Replace mmcv's function and modules imported with mmengine's

4.2 `mmcv.fileio`

4.3 `mmcv.runner`

4.4 `mmcv.parallel`

4.5 `mmcv.engine`

4.6 `mmcv.device`

4.7 `mmcv.utils`

4.8 `mmcv.cnn`

4.9 `mmcv.model_zoo`

CHAPTER

FIVE

DATA PROCESS

5.1 Image

This module provides some image processing methods, which requires `opencv` to be installed first.

5.1.1 Read/Write/Show

To read or write images files, use `imread` or `imwrite`.

```
import mmcv

img = mmcv.imread('test.jpg')
img = mmcv.imread('test.jpg', flag='grayscale')
img_ = mmcv.imread(img)  # nothing will happen, img_ = img
mmcv.imwrite(img, 'out.jpg')
```

To read images from bytes

```
with open('test.jpg', 'rb') as f:
    data = f.read()
img = mmcv.imfrombytes(data)
```

To show an image file or a loaded image

```
mmcv.imshow('tests/data/color.jpg')
# this is equivalent to

for i in range(10):
    img = np.random.randint(256, size=(100, 100, 3), dtype=np.uint8)
    mmcv.imshow(img, win_name='test image', wait_time=200)
```

5.1.2 Color space conversion

Supported conversion methods:

- bgr2gray
- gray2bgr
- bgr2rgb
- rgb2bgr
- bgr2hsv
- hsv2bgr

```
img = mmcv.imread('tests/data/color.jpg')
img1 = mmcv.bgr2rgb(img)
img2 = mmcv.rgb2gray(img1)
img3 = mmcv.bgr2hsv(img)
```

5.1.3 Resize

There are three resize methods. All `imresize_*` methods have an argument `return_scale`, if this argument is `False`, then the return value is merely the resized image, otherwise is a tuple `(resized_img, scale)`.

```
# resize to a given size
mmcv.imresize(img, (1000, 600), return_scale=True)

# resize to the same size of another image
mmcv.imresize_like(img, dst_img, return_scale=False)

# resize by a ratio
mmcv.imrescale(img, 0.5)

# resize so that the max edge no longer than 1000, short edge no longer than 800
# without changing the aspect ratio
mmcv.imrescale(img, (1000, 800))
```

5.1.4 Rotate

To rotate an image by some angle, use `imrotate`. The center can be specified, which is the center of original image by default. There are two modes of rotating, one is to keep the image size unchanged so that some parts of the image will be cropped after rotating, the other is to extend the image size to fit the rotated image.

```
img = mmcv.imread('tests/data/color.jpg')

# rotate the image clockwise by 30 degrees.
img_ = mmcv.imrotate(img, 30)

# rotate the image counterclockwise by 90 degrees.
img_ = mmcv.imrotate(img, -90)

# rotate the image clockwise by 30 degrees, and rescale it by 1.5x at the same time.
```

(continues on next page)

(continued from previous page)

```
img_ = mmcv.imrotate(img, 30, scale=1.5)

# rotate the image clockwise by 30 degrees, with (100, 100) as the center.
img_ = mmcv.imrotate(img, 30, center=(100, 100))

# rotate the image clockwise by 30 degrees, and extend the image size.
img_ = mmcv.imrotate(img, 30, auto_bound=True)
```

5.1.5 Flip

To flip an image, use `imflip`.

```
img = mmcv.imread('tests/data/color.jpg')

# flip the image horizontally
mmcv.imflip(img)

# flip the image vertically
mmcv.imflip(img, direction='vertical')
```

5.1.6 Crop

`imcrop` can crop the image with one or more regions. Each region is represented by the upper left and lower right coordinates as (x1, y1, x2, y2).

```
import mmcv
import numpy as np

img = mmcv.imread('tests/data/color.jpg')

# crop the region (10, 10, 100, 120)
bboxes = np.array([10, 10, 100, 120])
patch = mmcv.imcrop(img, bboxes)

# crop two regions (10, 10, 100, 120) and (0, 0, 50, 50)
bboxes = np.array([[10, 10, 100, 120], [0, 0, 50, 50]])
patches = mmcv.imcrop(img, bboxes)

# crop two regions, and rescale the patches by 1.2x
patches = mmcv.imcrop(img, bboxes, scale=1.2)
```

5.1.7 Padding

There are two methods, `impad` and `impad_to_multiple`, to pad an image to the specific size with given values.

```
img = mmcv.imread('tests/data/color.jpg')

# pad the image to (1000, 1200) with all zeros
img_ = mmcv.impad(img, shape=(1000, 1200), pad_val=0)

# pad the image to (1000, 1200) with different values for three channels.
img_ = mmcv.impad(img, shape=(1000, 1200), pad_val=(100, 50, 200))

# pad the image on left, right, top, bottom borders with all zeros
img_ = mmcv.impad(img, padding=(10, 20, 30, 40), pad_val=0)

# pad the image on left, right, top, bottom borders with different values
# for three channels.
img_ = mmcv.impad(img, padding=(10, 20, 30, 40), pad_val=(100, 50, 200))

# pad an image so that each edge is a multiple of some value.
img_ = mmcv.impad_to_multiple(img, 32)
```

5.2 Video

This module provides the following functionalities:

- A `VideoReader` class with friendly apis to read and convert videos.
- Some methods for editing (cut, concat, resize) videos.
- Optical flow read/write/warp.

5.2.1 VideoReader

The `VideoReader` class provides sequence like apis to access video frames. It will internally cache the frames which have been visited.

```
video = mmcv.VideoReader('test.mp4')

# obtain basic information
print(len(video))
print(video.width, video.height, video.resolution, video.fps)

# iterate over all frames
for frame in video:
    print(frame.shape)

# read the next frame
img = video.read()

# read a frame by index
img = video[100]
```

(continues on next page)

(continued from previous page)

```
# read some frames
img = video[5:10]
```

To convert a video to images or generate a video from a image directory.

```
# split a video into frames and save to a folder
video = mmcv.VideoReader('test.mp4')
video.cvt2frames('out_dir')

# generate video from frames
mmcv.frames2video('out_dir', 'test.avi')
```

5.2.2 Editing utils

There are also some methods for editing videos, which wraps the commands of ffmpeg.

```
# cut a video clip
mmcv.cut_video('test.mp4', 'clip1.mp4', start=3, end=10, vcodec='h264')

# join a list of video clips
mmcv.concat_video(['clip1.mp4', 'clip2.mp4'], 'joined.mp4', log_level='quiet')

# resize a video with the specified size
mmcv.resize_video('test.mp4', 'resized1.mp4', (360, 240))

# resize a video with a scaling ratio of 2
mmcv.resize_video('test.mp4', 'resized2.mp4', ratio=2)
```

5.2.3 Optical flow

mmcv provides the following methods to operate on optical flows.

- IO
- Visualization
- Flow warping

We provide two options to dump optical flow files: uncompressed and compressed. The uncompressed way just dumps the floating numbers to a binary file. It is lossless but the dumped file has a larger size. The compressed way quantizes the optical flow to 0-255 and dumps it as a jpeg image. The flow of x-dim and y-dim will be concatenated into a single image.

1. IO

```
flow = np.random.rand(800, 600, 2).astype(np.float32)
# dump the flow to a flo file (~3.7M)
mmcv.flowwrite(flow, 'uncompressed.flo')
# dump the flow to a jpeg file (~230K)
# the shape of the dumped image is (800, 1200)
mmcv.flowwrite(flow, 'compressed.jpg', quantize=True, concat_axis=1)
```

(continues on next page)

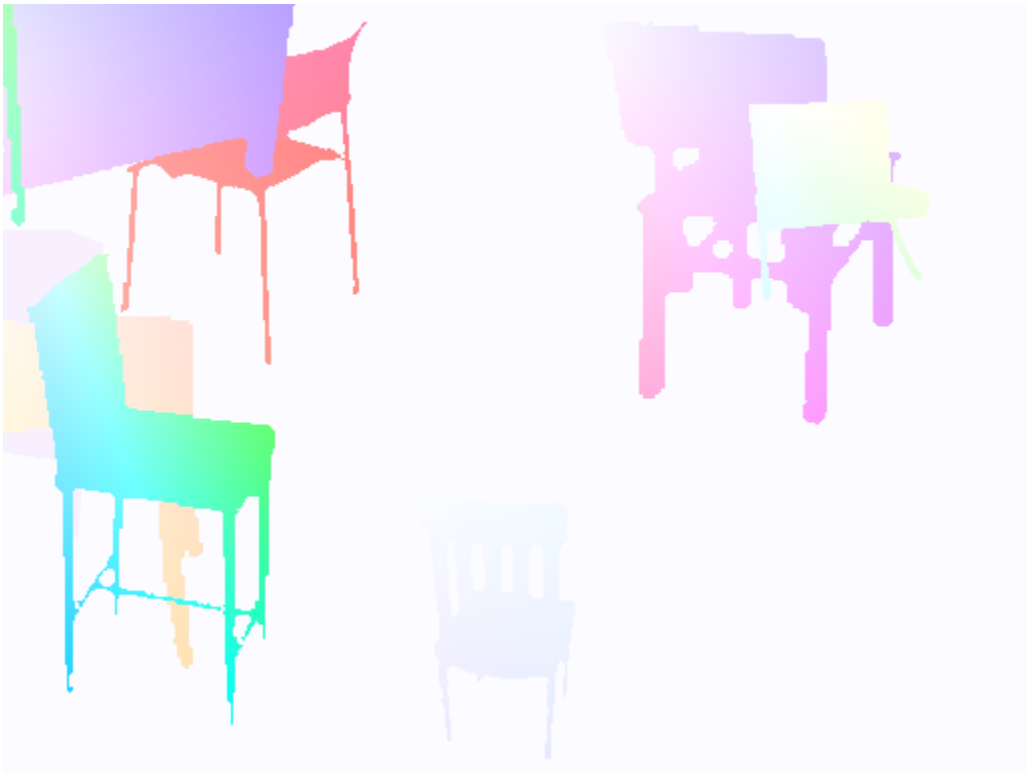
(continued from previous page)

```
# read the flow file, the shape of loaded flow is (800, 600, 2) for both ways
flow = mmcv.flowread('uncompressed.flo')
flow = mmcv.flowread('compressed.jpg', quantize=True, concat_axis=1)
```

2. Visualization

It is possible to visualize optical flows with `mmcv.flowshow()`.

```
mmcv.flowshow(flow)
```



3. Flow warping

```
img1 = mmcv.imread('img1.jpg')
flow = mmcv.flowread('flow.flo')
warped_img2 = mmcv.flow_warp(img1, flow)
```

img1 (left) and img2 (right)

optical flow (img2 -> img1)

warped image and difference with ground truth

CHAPTER

SIX

DATA TRANSFORMATION

In the OpenMMLab algorithm library, dataset construction and data preparation are decoupled. Usually, the construction of the dataset only parses the dataset and records the basic information of each sample, while the data preparation is a series of data transformations including data loading, preprocessing, formatting, and other operations performed according to the basic information of the sample.

6.1 Design of data transformation

In MMCV, we use various callable data transformation classes to manipulate data. These data transformation classes can accept several configuration parameters for the instantiation and then process the input data dictionary by `__call__` method. All data transformation methods accept a dictionary as the input and produce the output as a dictionary as well. A simple example is as follows:

```
>>> import numpy as np
>>> from mmcv.transforms import Resize
>>>
>>> transform = Resize(scale=(224, 224))
>>> data_dict = {'img': np.random.rand(256, 256, 3)}
>>> data_dict = transform(data_dict)
>>> print(data_dict['img'].shape)
(224, 224, 3)
```

The data transformation class reads some fields of the input dictionary and may add or update some fields. The keys of these fields are mostly fixed. For example, `Resize` will always read fields such as `"img"` in the input dictionary. More information about the conventions for input and output fields could be found in the documentation of the corresponding class.

Note: By convention, the order of image shape which is used as **initialization parameters** in data transformation (such as Resize, Pad) is (width, height). In the dictionary returned by the data transformation, the image related shape, such as `img_shape`, `ori_shape`, `pad_shape`, etc., is (height, width).

MMCV provides a unified base class called `BaseTransform` for all data transformation classes:

```
class BaseTransform(metaclass=ABCMeta):

    def __call__(self, results: dict) -> dict:

        return self.transform(results)
```

(continues on next page)

(continued from previous page)

```
    @abstractmethod
    def transform(self, results: dict) -> dict:
        pass
```

All data transformation classes must inherit `BaseTransform` and implement the `transform` method. Both the input and output of the `transform` method are a dictionary. In the **Custom data transformation class** section, we will describe how to implement a data transformation class in more detail.

6.2 Data pipeline

As mentioned above, the inputs and outputs of all data transformations are dictionaries. Moreover, according to the [Convention on Datasets] (TODO) in OpenMMLab, the basic information of each sample in the dataset is also a dictionary. This way, we can connect all data transformation operations end to end and combine them into a data pipeline. This pipeline inputs the information dictionary of the samples in the dataset and outputs the information dictionary after a series of processing.

Taking the classification task as an example, we show a typical data pipeline in the figure below. For each sample, the information stored in the dataset is a dictionary, as shown on the far left in the figure. After each data transformation operation represented by the blue block, a new field (marked in green) will be added to the data dictionary or an existing field (marked in orange) will be updated.

The data pipeline is a list of several data transformation configuration dictionaries in the configuration file. Each dataset needs to set the parameter `pipeline` to define the data preparation operations the dataset needs to perform. The configuration of the above data pipeline in the configuration file is as follows:

```
pipeline = [
    dict(type='LoadImageFromFile'),
    dict(type='Resize', size=256, keep_ratio=True),
    dict(type='CenterCrop', crop_size=224),
    dict(type='Normalize', mean=[123.675, 116.28, 103.53], std=[58.395, 57.12, 57.375]),
    dict(type='ClsFormatBundle')
]

dataset = dict(
    ...
    pipeline=pipeline,
    ...
)
```

6.3 Common data transformation classes

The commonly used data transformation classes can be roughly divided into data loading, data preprocessing and augmentation, and data formatting. In MMCV, we provide some commonly used classes as follows:

6.3.1 Data loading

To support the loading of large-scale datasets, data is usually not loaded when `Dataset` is initialized. Only the corresponding path is loaded. Therefore, it is necessary to load specific data in the data pipeline.

6.3.2 Data preprocessing and enhancement

Data preprocessing and augmentation usually involve transforming the image itself, such as cropping, padding, scaling, etc.

6.3.3 Data formatting

Data formatting operations are type conversions performed on the data.

6.4 Customize data transformation classes

To implement a new data transformation class, you must inherit `BaseTransform` and implement the `transform` method. Here, we use a simple flip transform (`MyFlip`) as an example:

```
import random
import mmcv
from mmcv.transforms import BaseTransform, TRANSFORMS

@TRANSFORMS.register_module()
class MyFlip(BaseTransform):
    def __init__(self, direction: str):
        super().__init__()
        self.direction = direction

    def transform(self, results: dict) -> dict:
        img = results['img']
        results['img'] = mmcv.imflip(img, direction=self.direction)
        return results
```

Now, we can instantiate `MyFlip` as a callable object to handle our data dictionary.

```
import numpy as np

transform = MyFlip(direction='horizontal')
data_dict = {'img': np.random.rand(224, 224, 3)}
data_dict = transform(data_dict)
processed_img = data_dict['img']
```

Alternatively, use `MyFlip` transform in the `pipeline` of the config file.

```
pipeline = [
    ...
    dict(type='MyFlip', direction='horizontal'),
    ...
]
```

It should be noted that if you want to use it in the configuration file, you must ensure that the file where the `MyFlip` class is located can be imported at the runtime.

6.5 Transform wrapper

Transform wrappers are a special class of data transformations. They do not operate on images, labels or other information in the data dictionary by themselves. Instead, they enhance the behavior of data transformations defined in them.

6.5.1 KeyMapper

`KeyMapper` is used to map fields in the data dictionary. For example, image processing transforms usually get their values from the `"img"` field in the data dictionary. But sometimes we want these transforms to handle images in other fields in the data dictionary, such as the `"gt_img"` field.

When used with registry and configuration file, the field map wrapper should be used as follows:

```
pipeline = [
    ...
    dict(type='KeyMapper',
        mapping={
            'img': 'gt_img',  # map "gt_img" to "img"
            'mask': ...,  # The "mask" field in the raw data is not used. That is, for␣
↪wrapped data transformations, the "mask" field is not included in the data
        },
        auto_remap=True,  # remap "img" back to "gt_img" after the transformation
        transforms=[
            # only need to specify "img" in `RandomFlip`
            dict(type='RandomFlip'),
        ])
    ...
]
```

With `KeyMapper`, we don't need to consider various possible input field names in the `transform` method when we implement the data transformation class. We only need to deal with the default fields.

6.5.2 RandomChoice and RandomApply

`RandomChoice` is used to randomly select a data transformation pipeline from the given choices. With this wrapper, we can easily implement some data augmentation functions, such as AutoAugment.

In configuration file, you can use `RandomChoice` as follows:

```
pipeline = [
    ...
    dict(type='RandomChoice',
        transforms=[
            [
                dict(type='Posterize', bits=4),
                dict(type='Rotate', angle=30.)
            ],  # the first combo option
```

(continues on next page)

(continued from previous page)

```
            [
                dict(type='Equalize'),
                dict(type='Rotate', angle=30)
            ],  # the second combo option
        ],
        prob=[0.4, 0.6]  # the prob of each combo
        )
    ...
]
```

`RandomApply` is used to randomly perform a combination of data transformations with a specified probability. For example:

```
pipeline = [
    ...
    dict(type='RandomApply',
        transforms=[dict(type='Rotate', angle=30.)],
        prob=0.3)  # perform the transformation with prob as 0.3
    ...
]
```

6.5.3 TransformBroadcaster

Usually, a data transformation class only reads the target of an operation from one field. While we can also use `KeyMapper` to change the fields read, there is no way to apply transformations to the data of multiple fields at once. To achieve this, we need to use the multi-target extension wrapper `TransformBroadcaster`.

`TransformBroadcaster` has two uses, one is to apply data transformation to multiple specified fields, and the other is to apply data transformation to a group of targets under a field.

1. Apply to multiple fields

   Suppose we need to apply a data transformation to images in two fields `"lq"` (low-quality) and `"gt"` (ground-truth).

   ```
   pipeline = [
       dict(type='TransformBroadcaster',
           # apply to the "lq" and "gt" fields respectively, and set the "img" field␣
   ↪to both
           mapping={'img': ['lq', 'gt']},
           # remap the "img" field back to the original field after the transformation
           auto_remap=True,
           # whether to share random variables in the transformation of each target
           # more introduction will be referred in the following chapters (random␣
   ↪variable sharing)
           share_random_params=True,
           transforms=[
               # only need to manipulate the "img" field in the `RandomFlip` class
               dict(type='RandomFlip'),
           ])
   ]
   ```

   In the `mapping` setting of the multi-target extension, we can also use `...` to ignore the specified original field.

As shown in the following example, the wrapped `RandomCrop` will crop the image in the field `"img"` and update the size of the cropped image if the field `"img_shape"` exists. If we want to do the same random cropping for both image fields `"lq"` and `"gt"` at the same time but update the `"img_shape"` field only once, we can do it as in the example:

```
pipeline = [
    dict(type='TransformBroadcaster',
        mapping={
            'img': ['lq', 'gt'],
            'img_shape': ['img_shape', ...],
         },
        # remap the "img" and "img_shape" fields back to their original fields␣
↪after the transformation
        auto_remap=True,
        # whether to share random variables in the transformation of each target
        # more introduction will be referred in the following chapters (random␣
↪variable sharing)
        share_random_params=True,
        transforms=[
            # "img" and "img_shape" fields are manipulated in the `RandomCrop` class
            # if "img_shape" is missing, only operate on "img"
            dict(type='RandomCrop'),
        ])
]
```

2. A set of targets applied to a field

   Suppose we need to apply a data transformation to the `"images"` field, which is a list of images.

```
pipeline = [
    dict(type='TransformBroadcaster',
        # map each image under the "images" field to the "img" field
        mapping={'img': 'images'},
        # remap the images under the "img" field back to the list in the "images"␣
↪field after the transformation
        auto_remap=True,
        # whether to share random variables in the transformation of each target
        share_random_params=True,
        transforms=[
            # in the `RandomFlip` transformation class, we only need to manipulate␣
↪the "img" field
            dict(type='RandomFlip'),
        ])
]
```

Decorator cache_randomness

In TransformBroadcaster, we provide the share_random_params option to support sharing random states across multiple data transformations. For example, in a super-resolution task, we want to apply **the same** random transformations **simultaneously** to the low-resolution image and the original image. If we use this function in a custom data transformation class, we need to mark which random variables support sharing in the class. This can be achieved with the decorator cache_randomness.

Taking MyFlip from the above example, we want to perform flipping randomly with a certain probability:

```
from mmcv.transforms.utils import cache_randomness

@TRANSFORMS.register_module()
class MyRandomFlip(BaseTransform):
    def __init__(self, prob: float, direction: str):
        super().__init__()
        self.prob = prob
        self.direction = direction

    @cache_randomness  # label the output of the method as a shareable random variable
    def do_flip(self):
        flip = True if random.random() > self.prob else False
        return flip

    def transform(self, results: dict) -> dict:
        img = results['img']
        if self.do_flip():
            results['img'] = mmcv.imflip(img, direction=self.direction)
        return results
```

In the above example, we decorate the do_flip method with cache_randomness, marking the method return value flip as a random variable that supports sharing. Therefore, in the transformation of TransformBroadcaster to multiple targets, the value of this variable will remain the same.

Decorator avoid_cache_randomness

In some cases, we cannot separate the process of generating random variables in data transformation into a class method. For example, modules from third-party libraries used in data transformation encapsulate the relevant parts of random variables inside, making them impossible to be extracted as class methods for data transformation. Such data transformations cannot support shared random variables through the decorator cache_randomness annotation, and thus cannot share random variables during multi-objective expansion.

To avoid misuse of such data transformations in multi-object extensions, we provide another decorator, avoid_cache_randomness, to mark such data transformations:

```
from mmcv.transforms.utils import avoid_cache_randomness

@TRANSFORMS.register_module()
@avoid_cache_randomness
class MyRandomTransform(BaseTransform):

    def transform(self, results: dict) -> dict:
        ...
```

Data transformation classes marked with `avoid_cache_randomness` will throw an exception when their instance is wrapped by `TransformBroadcaster` and the parameter `share_random_params` is set to True. This reminds the user not to use it in this way.

There are a few things to keep in mind when using `avoid_cache_randomness`:

1. `avoid_cache_randomness` is only used to decorate data transformation classes (subclasses of `BaseTransfrom`) and cannot be used to decorate other general classes, class methods, or functions
2. When a data transformation decorated with `avoid_cache_randomness` is used as a base class, its subclasses **will not inherit** its feature. If the subclass is still unable to share random variables, `avoid_cache_randomness` should be used again.
3. A data transformation needs to be modified with `avoid_cache_randomness` only when a data transformation is random and cannot share its random parameters. Data transformations without randomness require no decoration

CHAPTER

SEVEN

VISUALIZATION

mmcv can show images and annotations (currently supported types include bounding boxes).

```
# show an image file
mmcv.imshow('a.jpg')

# show a loaded image
img = np.random.rand(100, 100, 3)
mmcv.imshow(img)

# show image with bounding boxes
img = np.random.rand(100, 100, 3)
bboxes = np.array([[0, 0, 50, 50], [20, 20, 60, 60]])
mmcv.imshow_bboxes(img, bboxes)
```

mmcv can also visualize special images such as optical flows.

```
flow = mmcv.flowread('test.flo')
mmcv.flowshow(flow)
```

CHAPTER

EIGHT

CNN

We provide some building bricks for CNNs, including layer building, module bundles and weight initialization.

8.1 Layer building

We may need to try different layers of the same type when running experiments, but do not want to modify the code from time to time. Here we provide some layer building methods to construct layers from a dict, which can be written in configs or specified via command line arguments.

8.1.1 Usage

A simplest example is

```
from mmcv.cnn import build_conv_layer

cfg = dict(type='Conv3d')
layer = build_conv_layer(cfg, in_channels=3, out_channels=8, kernel_size=3)
```

- `build_conv_layer`: Supported types are Conv1d, Conv2d, Conv3d, Conv (alias for Conv2d).
- `build_norm_layer`: Supported types are BN1d, BN2d, BN3d, BN (alias for BN2d), SyncBN, GN, LN, IN1d, IN2d, IN3d, IN (alias for IN2d).
- `build_activation_layer`: Supported types are ReLU, LeakyReLU, PReLU, RReLU, ReLU6, ELU, Sigmoid, Tanh, GELU.
- `build_upsample_layer`: Supported types are nearest, bilinear, deconv, pixel_shuffle.
- `build_padding_layer`: Supported types are zero, reflect, replicate.

8.1.2 Extension

We also allow extending the building methods with custom layers and operators.

1. Write and register your own module.

```
from mmengine.registry import MODELS

@MODELS.register_module()
class MyUpsample:
```

(continues on next page)

(continued from previous page)

```
    def __init__(self, scale_factor):
        pass

    def forward(self, x):
        pass
```

2. Import `MyUpsample` somewhere (e.g., in `__init__.py`) and then use it.

```
from mmcv.cnn import build_upsample_layer

cfg = dict(type='MyUpsample', scale_factor=2)
layer = build_upsample_layer(cfg)
```

8.2 Module bundles

We also provide common module bundles to facilitate the network construction. `ConvModule` is a bundle of convolution, normalization and activation layers, please refer to the api for details.

```
from mmcv.cnn import ConvModule

# conv + bn + relu
conv = ConvModule(3, 8, 2, norm_cfg=dict(type='BN'))
# conv + gn + relu
conv = ConvModule(3, 8, 2, norm_cfg=dict(type='GN', num_groups=2))
# conv + relu
conv = ConvModule(3, 8, 2)
# conv
conv = ConvModule(3, 8, 2, act_cfg=None)
# conv + leaky relu
conv = ConvModule(3, 8, 3, padding=1, act_cfg=dict(type='LeakyReLU'))
# bn + conv + relu
conv = ConvModule(
    3, 8, 2, norm_cfg=dict(type='BN'), order=('norm', 'conv', 'act'))
```

CHAPTER

NINE

OPS

We implement common ops used in detection, segmentation, etc.

CHAPTER

TEN

MMCV OPERATORS

To make custom operators in MMCV more standard, precise definitions of each operator are listed in this document.

10.1 MMCVBorderAlign

10.1.1 Description

Applies `border_align` over the input feature based on predicted bboxes.

For each border line (e.g. top, left, bottom or right) of each box, border_align does the following:

- uniformly samples `pool_size`+1 positions on this line, involving the start and end points.
- the corresponding features on these points are computed by bilinear interpolation.
- max pooling over all the `pool_size`+1 positions are used for computing pooled feature.

Read BorderDet: Border Feature for Dense Object Detection for more detailed information.

10.1.2 Parameters

10.1.3 Inputs

10.1.4 Outputs

10.1.5 Type Constraints

- T:tensor(float32)

10.2 MMCVCARAFE

10.2.1 Description

CARAFE operator performs feature upsampling.

Read CARAFE: Content-Aware ReAssembly of FEatures for more detailed information.

10.2.2 Parameters

10.2.3 Inputs

10.2.4 Outputs

10.2.5 Type Constraints

- T:tensor(float32)

10.3 MMCVCAWeight

10.3.1 Description

Operator for Criss-Cross Attention Read CCNet: Criss-Cross Attention for SemanticSegmentation for more detailed information.

10.3.2 Parameters

None

10.3.3 Inputs

10.3.4 Outputs

10.3.5 Type Constraints

- T:tensor(float32)

10.4 MMCVCAMap

10.4.1 Description

Operator for Criss-Cross Attention Read CCNet: Criss-Cross Attention for SemanticSegmentation for more detailed information.

10.4.2 Parameters

None

10.4.3 Inputs

10.4.4 Outputs

10.4.5 Type Constraints

- T:tensor(float32)

10.5 MMCVCornerPool

10.5.1 Description

Perform CornerPool on `input` features. Read CornerNet – Detecting Objects as Paired Keypoints for more details.

10.5.2 Parameters

10.5.3 Inputs

10.5.4 Outputs

10.5.5 Type Constraints

- T:tensor(float32)

10.6 MMCVDeformConv2d

10.6.1 Description

Applies a deformable 2D convolution over an input signal composed of several input planes.

Read Deformable Convolutional Networks for detail.

10.6.2 Parameters

10.6.3 Inputs

10.6.4 Outputs

10.6.5 Type Constraints

- T:tensor(float32, Linear)

10.7 MMCVModulatedDeformConv2d

10.7.1 Description

Perform Modulated Deformable Convolution on input feature, read Deformable ConvNets v2: More Deformable, Better Results for detail.

10.7.2 Parameters

10.7.3 Inputs

10.7.4 Outputs

10.7.5 Type Constraints

- T:tensor(float32, Linear)

10.8 MMCVDeformRoIPool

10.8.1 Description

Deformable roi pooling layer

10.8.2 Parameters

10.8.3 Inputs

10.8.4 Outputs

10.8.5 Type Constraints

- T:tensor(float32)

10.9 MMCVMaskedConv2d

10.9.1 Description

Performs a masked 2D convolution from PixelRNN Read Pixel Recurrent Neural Networks for more detailed information.

10.9.2 Parameters

10.9.3 Inputs

10.9.4 Outputs

10.9.5 Type Constraints

- T:tensor(float32)

10.10 MMCVPSAMask

10.10.1 Description

An operator from PSANet.

Read PSANet: Point-wise Spatial Attention Network for Scene Parsing for more detailed information.

10.10.2 Parameters

10.10.3 Inputs

10.10.4 Outputs

10.10.5 Type Constraints

- T:tensor(float32)

10.11 NonMaxSuppression

10.11.1 Description

Filter out boxes has high IoU overlap with previously selected boxes or low score. Output the indices of valid boxes.

Note this definition is slightly different with onnx: NonMaxSuppression

10.11.2 Parameters

10.11.3 Inputs

10.11.4 Outputs

10.11.5 Type Constraints

- T:tensor(float32, Linear)

10.12 MMCVRoIAlign

10.12.1 Description

Perform RoIAlign on output feature, used in bbox_head of most two-stage detectors.

10.12.2 Parameters

10.12.3 Inputs

10.12.4 Outputs

10.12.5 Type Constraints

- T:tensor(float32)

10.13 MMCVRoIAlignRotated

10.13.1 Description

Perform RoI align pooling for rotated proposals

10.13.2 Parameters

10.13.3 Inputs

10.13.4 Outputs

10.13.5 Type Constraints

- T:tensor(float32)

10.14 grid_sampler*

10.14.1 Description

Perform sample from `input` with pixel locations from `grid`.

Check torch.nn.functional.grid_sample for more information.

10.14.2 Parameters

10.14.3 Inputs

10.14.4 Outputs

10.14.5 Type Constraints

- T:tensor(float32, Linear)

10.15 cummax*

10.15.1 Description

Returns a tuple (`values`, `indices`) where `values` is the cumulative maximum elements of `input` in the dimension `dim`. And `indices` is the index location of each maximum value found in the dimension `dim`. Read torch.cummax for more details.

10.15.2 Parameters

10.15.3 Inputs

10.15.4 Outputs

10.15.5 Type Constraints

- T:tensor(float32)

10.16 cummin*

10.16.1 Description

Returns a tuple (`values`, `indices`) where `values` is the cumulative minimum elements of `input` in the dimension `dim`. And `indices` is the index location of each minimum value found in the dimension `dim`. Read torch.cummin for more details.

10.16.2 Parameters

10.16.3 Inputs

10.16.4 Outputs

10.16.5 Type Constraints

- T:tensor(float32)

10.17 Reminders

- Operators endwith * are defined in Torch and are included here for the conversion to ONNX.

CHAPTER

ELEVEN

ENGLISH

CHAPTER

TWELVE

CHAPTER

THIRTEEN

V2.0.0

The OpenMMLab team released a new generation of training engine MMEngine at the World Artificial Intelligence Conference on September 1, 2022. It is a foundational library for training deep learning models. Compared with MMCV, it provides a universal and powerful runner, an open architecture with a more unified interface, and a more customizable training process.

The OpenMMLab team released MMCV v2.0.0 on April 6, 2023. In the 2.x version, it has the following significant changes:

(1) It removed the following components:

- `mmcv.fileio` module, removed in PR #2179. FileIO module from mmengine will be used wherever required.
- `mmcv.runner`, `mmcv.parallel`, `mmcv. engine` and `mmcv.device`, removed in PR #2216.
- All classes in `mmcv.utils` (eg `Config` and `Registry`) and many functions, removed in PR #2217. Only a few functions related to mmcv are reserved.
- `mmcv.onnx`, `mmcv.tensorrt` modules and related functions, removed in PR #2225.
- Removed all root registrars in MMCV and registered classes or functions to the root registrar in MMEngine.

(2) It added the `mmcv.transforms` data transformation module.

(3) It renamed the package name **mmcv** to **mmcv-lite** and **mmcv-full** to **mmcv** in PR #2235. Also, change the default value of the environment variable `MMCV_WITH_OPS` from 0 to 1.

```
# Contains ops, because the highest version of mmcv-full is less than 2.0.0, so there is␣
↪no need to add version restrictions
pip install openmim
mim install mmcv-full

# do not contain ops
pip install openmim
mim install "mmcv < 2.0.0"
```

```
# Contains ops
pip install openmim
mim install mmcv

# Ops are not included, because the starting version of mmcv-lite is 2.0.0rc1, so there␣
↪is no need to add version restrictions
pip install openmim
mim install mmcv-lite
```

CHAPTER

FOURTEEN

V1.3.18

Some ops have different implementations on different devices. Lots of macros and type checks are scattered in several files, which makes the code hard to maintain. For example:

```
  if (input.device().is_cuda()) {
#ifdef MMCV_WITH_CUDA
    CHECK_CUDA_INPUT(input);
    CHECK_CUDA_INPUT(rois);
    CHECK_CUDA_INPUT(output);
    CHECK_CUDA_INPUT(argmax_y);
    CHECK_CUDA_INPUT(argmax_x);

    roi_align_forward_cuda(input, rois, output, argmax_y, argmax_x,
                           aligned_height, aligned_width, spatial_scale,
                           sampling_ratio, pool_mode, aligned);
#else
    AT_ERROR("RoIAlign is not compiled with GPU support");
#endif
  } else {
    CHECK_CPU_INPUT(input);
    CHECK_CPU_INPUT(rois);
    CHECK_CPU_INPUT(output);
    CHECK_CPU_INPUT(argmax_y);
    CHECK_CPU_INPUT(argmax_x);
    roi_align_forward_cpu(input, rois, output, argmax_y, argmax_x,
                          aligned_height, aligned_width, spatial_scale,
                          sampling_ratio, pool_mode, aligned);
  }
```

Registry and dispatcher are added to manage these implementations.

```
void ROIAlignForwardCUDAKernelLauncher(Tensor input, Tensor rois, Tensor output,
                                       Tensor argmax_y, Tensor argmax_x,
                                       int aligned_height, int aligned_width,
                                       float spatial_scale, int sampling_ratio,
                                       int pool_mode, bool aligned);

void roi_align_forward_cuda(Tensor input, Tensor rois, Tensor output,
                            Tensor argmax_y, Tensor argmax_x,
                            int aligned_height, int aligned_width,
                            float spatial_scale, int sampling_ratio,
                            int pool_mode, bool aligned) {
```

(continues on next page)

(continued from previous page)

```
  ROIAlignForwardCUDAKernelLauncher(
      input, rois, output, argmax_y, argmax_x, aligned_height, aligned_width,
      spatial_scale, sampling_ratio, pool_mode, aligned);
}

// register cuda implementation
void roi_align_forward_impl(Tensor input, Tensor rois, Tensor output,
                            Tensor argmax_y, Tensor argmax_x,
                            int aligned_height, int aligned_width,
                            float spatial_scale, int sampling_ratio,
                            int pool_mode, bool aligned);
REGISTER_DEVICE_IMPL(roi_align_forward_impl, CUDA, roi_align_forward_cuda);

// roi_align.cpp
// use the dispatcher to invoke different implementation depending on device type of␣
↪input tensors.
void roi_align_forward_impl(Tensor input, Tensor rois, Tensor output,
                            Tensor argmax_y, Tensor argmax_x,
                            int aligned_height, int aligned_width,
                            float spatial_scale, int sampling_ratio,
                            int pool_mode, bool aligned) {
  DISPATCH_DEVICE_IMPL(roi_align_forward_impl, input, rois, output, argmax_y,
                       argmax_x, aligned_height, aligned_width, spatial_scale,
                       sampling_ratio, pool_mode, aligned);
}
```

CHAPTER

FIFTEEN

V1.3.11

In order to flexibly support more backends and hardwares like `NVIDIA GPUs` and `AMD GPUs`, the directory of `mmcv/ops/csrc` is refactored. Note that this refactoring will not affect the usage in API. For related information, please refer to PR1206.

The original directory was organized as follows.

```
.
├── common_cuda_helper.hpp
├── ops_cuda_kernel.cuh
├── pytorch_cpp_helper.hpp
├── pytorch_cuda_helper.hpp
├── parrots_cpp_helper.hpp
├── parrots_cuda_helper.hpp
├── parrots_cudawarpfunction.cuh
├── onnxruntime
│   ├── onnxruntime_register.h
│   ├── onnxruntime_session_options_config_keys.h
│   ├── ort_mmcv_utils.h
│   ├── ...
│   ├── onnx_ops.h
│   └── cpu
│       ├── onnxruntime_register.cpp
│       ├── ...
│       └── onnx_ops_impl.cpp
├── parrots
│   ├── ...
│   ├── ops.cpp
│   ├── ops_cuda.cu
│   ├── ops_parrots.cpp
│   └── ops_pytorch.h
├── pytorch
│   ├── ...
│   ├── ops.cpp
│   ├── ops_cuda.cu
│   ├── pybind.cpp
└── tensorrt
    ├── trt_cuda_helper.cuh
    ├── trt_plugin_helper.hpp
    ├── trt_plugin.hpp
    ├── trt_serialize.hpp
    ├── ...
```

(continues on next page)

(continued from previous page)

```
    ├── trt_ops.hpp
    └── plugins
        ├── trt_cuda_helper.cu
        ├── trt_plugin.cpp
        ├── ...
        ├── trt_ops.cpp
        └── trt_ops_kernel.cu
```

After refactored, it is organized as follows.

```
.
├── common
│   ├── box_iou_rotated_utils.hpp
│   ├── parrots_cpp_helper.hpp
│   ├── parrots_cuda_helper.hpp
│   ├── pytorch_cpp_helper.hpp
│   ├── pytorch_cuda_helper.hpp
│   └── cuda
│       ├── common_cuda_helper.hpp
│       ├── parrots_cudawarpfunction.cuh
│       ├── ...
│       └── ops_cuda_kernel.cuh
├── onnxruntime
│   ├── onnxruntime_register.h
│   ├── onnxruntime_session_options_config_keys.h
│   ├── ort_mmcv_utils.h
│   ├── ...
│   ├── onnx_ops.h
│   └── cpu
│       ├── onnxruntime_register.cpp
│       ├── ...
│       └── onnx_ops_impl.cpp
├── parrots
│   ├── ...
│   ├── ops.cpp
│   ├── ops_parrots.cpp
│   └── ops_pytorch.h
├── pytorch
│   ├── info.cpp
│   ├── pybind.cpp
│   ├── ...
│   ├── ops.cpp
│   └── cuda
│       ├── ...
│       └── ops_cuda.cu
└── tensorrt
    ├── trt_cuda_helper.cuh
    ├── trt_plugin_helper.hpp
    ├── trt_plugin.hpp
    ├── trt_serialize.hpp
    ├── ...
    ├── trt_ops.hpp
```

(continues on next page)

(continued from previous page)

```
    └── plugins
        ├── trt_cuda_helper.cu
        ├── trt_plugin.cpp
        ├── ...
        ├── trt_ops.cpp
        └── trt_ops_kernel.cu
```

CHAPTER

SIXTEEN

FREQUENTLY ASKED QUESTIONS

We list some common troubles faced by many users and their corresponding solutions here. Feel free to enrich the list if you find any frequent issues and have ways to help others to solve them.

16.1 Installation

- KeyError: "xxx: 'yyy is not in the zzz registry'"

  The registry mechanism will be triggered only when the file of the module is imported. So you need to import that file somewhere. More details can be found at KeyError: "MaskRCNN: 'RefineRoIHead is not in the models registry'".

- "No module named 'mmcv.ops'"; "No module named 'mmcv._ext'"

  1. Uninstall existing mmcv in the environment using `pip uninstall mmcv`
  2. Install mmcv-full following the installation instruction or Build MMCV from source

- "invalid device function" or "no kernel image is available for execution"

  1. Check the CUDA compute capability of you GPU
  2. Run `python mmdet/utils/collect_env.py` to check whether PyTorch, torchvision, and MMCV are built for the correct GPU architecture. You may need to set `TORCH_CUDA_ARCH_LIST` to reinstall MMCV. The compatibility issue could happen when using old GPUS, e.g., Tesla K80 (3.7) on colab.
  3. Check whether the running environment is the same as that when mmcv/mmdet is compiled. For example, you may compile mmcv using CUDA 10.0 bug run it on CUDA9.0 environments

- "undefined symbol" or "cannot open xxx.so"

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

- "RuntimeError: CUDA error: invalid configuration argument"

  This error may be caused by the poor performance of GPU. Try to decrease the value of THREADS_PER_BLOCK and recompile mmcv.

- "RuntimeError: nms is not compiled with GPU support"

This error is because your CUDA environment is not installed correctly. You may try to re-install your CUDA environment and then delete the build/ folder before re-compile mmcv.

- "Segmentation fault"

  1. Check your GCC version and use GCC >= 5.4. This usually caused by the incompatibility between PyTorch and the environment (e.g., GCC < 4.9 for PyTorch). We also recommend the users to avoid using GCC 5.5 because many feedbacks report that GCC 5.5 will cause "segmentation fault" and simply changing it to GCC 5.4 could solve the problem

  2. Check whether PyTorch is correctly installed and could use CUDA op, e.g. type the following command in your terminal and see whether they could correctly output results

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

  3. If PyTorch is correctly installed, check whether MMCV is correctly installed. If MMCV is correctly installed, then there will be no issue of the command

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

  4. If MMCV and PyTorch are correctly installed, you can use `ipdb` to set breakpoints or directly add `print` to debug and see which part leads the `segmentation fault`

- "libtorch_cuda_cu.so: cannot open shared object file"

  `mmcv-full` depends on the share object but it can not be found. We can check whether the object exists in `~/miniconda3/envs/{environment-name}/lib/python3.7/site-packages/torch/lib` or try to re-install the PyTorch.

- "fatal error C1189: #error: – unsupported Microsoft Visual Studio version!"

  If you are building mmcv-full on Windows and the version of CUDA is 9.2, you will probably encounter the error `"C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v9.2\include\crt/host_config.h(133): fatal error C1189: #error: -- unsupported Microsoft Visual Studio version! Only the versions 2012, 2013, 2015 and 2017 are supported!"`, in which case you can use a lower version of Microsoft Visual Studio like vs2017.

- "error: member "torch::jit::detail::ModulePolicy::all_slots" may not be initialized"

  If your version of PyTorch is 1.5.0 and you are building mmcv-full on Windows, you will probably encounter the error `- torch/csrc/jit/api/module.h(474): error: member "torch::jit::detail::ModulePolicy::all_slots" may not be initialized`. The way to solve the error is to replace all the `static constexpr bool all_slots = false;` with `static bool all_slots = false;` at this file `https://github.com/pytorch/pytorch/blob/v1.5.0/torch/csrc/jit/api/module.h`. More details can be found at member "torch::jit::detail::AttributePolicy::all_slots" may not be initialized.

- "error: a member with an in-class initializer must be const"

  If your version of PyTorch is 1.6.0 and you are building mmcv-full on Windows, you will probably encounter the error `"- torch/include\torch/csrc/jit/api/module.h(483): error: a member with an in-class initializer must be const"`. The way to solve the error is to replace all the `CONSTEXPR_EXCEPT_WIN_CUDA` with `const` at `torch/include\torch/csrc/jit/api/module.h`. More details can be found at Ninja: build stopped: subcommand failed.

- "error: member "torch::jit::ProfileOptionalOp::Kind" may not be initialized"

  If your version of PyTorch is 1.7.0 and you are building mmcv-full on Windows, you will probably encounter the error `torch/include\torch/csrc/jit/ir/ir.h(1347): error: member "torch::jit::ProfileOptionalOp::Kind" may not be initialized`. The way to solve the error needs to modify several local files of PyTorch:

  - delete `static constexpr Symbol Kind = ::c10::prim::profile;` and `tatic constexpr Symbol Kind = ::c10::prim::profile_optional;` at `torch/include\torch/csrc/jit/ir/ir.h`
  - replace `explicit operator type&() { return *(this->value); }` with `explicit operator type&() { return *((type*)this->value); }` at `torch\include\pybind11\cast.h`
  - replace all the `CONSTEXPR_EXCEPT_WIN_CUDA` with `const` at `torch/include\torch/csrc/jit/api/module.h`

  More details can be found at Ensure default extra_compile_args.

- Compatibility issue between MMCV and MMDetection; "ConvWS is already registered in conv layer"

  Please install the correct version of MMCV for the version of your MMDetection following the installation instruction.

16.2 Usage

- "RuntimeError: Expected to have finished reduction in the prior iteration before starting a new one"

  1. This error indicates that your module has parameters that were not used in producing loss. This phenomenon may be caused by running different branches in your code in DDP mode. More datails at Expected to have finished reduction in the prior iteration before starting a new one.
  2. You can set `find_unused_parameters = True` in the config to solve the above problems or find those unused parameters manually

- "RuntimeError: Trying to backward through the graph a second time"

  `GradientCumulativeOptimizerHook` and `OptimizerHook` are both set which causes the `loss.backward()` to be called twice so `RuntimeError` was raised. We can only use one of these. More datails at Trying to backward through the graph a second time.

CHAPTER

SEVENTEEN

CONTRIBUTING TO OPENMMLAB

Welcome to the MMCV community, we are committed to building a cutting-edge computer vision foundational library and all kinds of contributions are welcomed, including but not limited to

Fix bug

You can directly post a Pull Request to fix typo in code or documents

The steps to fix the bug of code implementation are as follows.

1. If the modification involve significant changes, you should create an issue first and describe the error information and how to trigger the bug. Other developers will discuss with you and propose an proper solution.
2. Posting a pull request after fixing the bug and adding corresponding unit test.

New Feature or Enhancement

1. If the modification involve significant changes, you should create an issue to discuss with our developers to propose an proper design.
2. Post a Pull Request after implementing the new feature or enhancement and add corresponding unit test.

Document

You can directly post a pull request to fix documents. If you want to add a document, you should first create an issue to check if it is reasonable.

17.1 Pull Request Workflow

If you're not familiar with Pull Request, don't worry! The following guidance will tell you how to create a Pull Request step by step. If you want to dive into the develop mode of Pull Request, you can refer to the official documents

17.1.1 1. Fork and clone

If you are posting a pull request for the first time, you should fork the OpenMMLab repositories by clicking the **Fork** button in the top right corner of the GitHub page, and the forked repositories will appear under your GitHub profile.

Then, you can clone the repositories to local:

```
git clone git@github.com:{username}/mmcv.git
```

After that, you should ddd official repository as the upstream repository

```
git remote add upstream git@github.com:open-mmlab/mmcv
```

Check whether remote repository has been added successfully by `git remote -v`

```
origin        git@github.com:{username}/mmcv.git (fetch)
origin        git@github.com:{username}/mmcv.git (push)
upstream        git@github.com:open-mmlab/mmcv (fetch)
upstream        git@github.com:open-mmlab/mmcv (push)
```

Note: Here's a brief introduction to origin and upstream. When we use "git clone", we create an "origin" remote by default, which points to the repository cloned from. As for "upstream", we add it ourselves to point to the target repository. Of course, if you don't like the name "upstream", you could name it as you wish. Usually, we'll push the code to "origin". If the pushed code conflicts with the latest code in official("upstream"), we should pull the latest code from upstream to resolve the conflicts, and then push to "origin" again. The posted Pull Request will be updated automatically.

17.1.2 2. Configure pre-commit

You should configure pre-commit in the local development environment to make sure the code style matches that of OpenMMLab. **Note**: The following code should be executed under the MMCV directory.

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

Check that pre-commit is configured successfully, and install the hooks defined in `.pre-commit-config.yaml`.

```
pre-commit run --all-files
```

Note: Chinese users may fail to download the pre-commit hooks due to the network issue. In this case, you could download these hooks from gitee by setting the .pre-commit-config-zh-cn.yaml

pre-commit install -c .pre-commit-config-zh-cn.yaml pre-commit run –all-files -c .pre-commit-config-zh-cn.yaml

If the installation process is interrupted, you can repeatedly run `pre-commit run ...` to continue the installation.

If the code does not conform to the code style specification, pre-commit will raise a warning and fixes some of the errors automatically.

If we want to commit our code bypassing the pre-commit hook, we can use the `--no-verify` option(**only for temporarily commit**.

```
git commit -m "xxx" --no-verify
```

17.1.3 3. Create a development branch

After configuring the pre-commit, we should create a branch based on the main branch to develop the new feature or fix the bug. The proposed branch name is `username/pr_name`

```
git checkout -b yhc/refactor_contributing_doc
```

In subsequent development, if the main branch of the local repository is behind the main branch of "upstream", we need to pull the upstream for synchronization, and then execute the above command:

```
git pull upstream main
```

17.1.4 4. Commit the code and pass the unit test

- MMCV introduces mypy to do static type checking to increase the robustness of the code. Therefore, we need to add Type Hints to our code and pass the mypy check. If you are not familiar with Type Hints, you can refer to this tutorial.
- The committed code should pass through the unit test

```
# Pass all unit tests
pytest tests

# Pass the unit test of runner
pytest tests/test_runner/test_runner.py
```

  If the unit test fails for lack of dependencies, you can install the dependencies referring to the *guidance*

- If the documents are modified/added, we should check the rendering result referring to *guidance*

17.1.5 5. Push the code to remote

We could push the local commits to remote after passing through the check of unit test and pre-commit. You can associate the local branch with remote branch by adding `-u` option.

```
git push -u origin {branch_name}
```

This will allow you to use the `git push` command to push code directly next time, without having to specify a branch or the remote repository.

17.1.6 6. Create a Pull Request

(1) Create a pull request in GitHub's Pull request interface

(2) Modify the PR description according to the guidelines so that other developers can better understand your changes

Find more details about Pull Request description in *pull request guidelines*.

note

(a) The Pull Request description should contain the reason for the change, the content of the change, and the impact of the change, and be associated with the relevant Issue (see documentation

(b) If it is your first contribution, please sign the CLA

(c) Check whether the Pull Request pass through the CI

MMCV will run unit test for the posted Pull Request on different platforms (Linux, Window, Mac), based on different versions of Python, PyTorch, CUDA to make sure the code is correct. We can see the specific test information by clicking `Details` in the above image so that we can modify the code.

(3) If the Pull Request passes the CI, then you can wait for the review from other developers. You'll modify the code based on the reviewer's comments, and repeat the steps *4*-*5* until all reviewers approve it. Then, we will merge it ASAP.

17.1.7 7. Resolve conflicts

If your local branch conflicts with the latest main branch of "upstream", you'll need to resolove them. There are two ways to do this:

```
git fetch --all --prune
git rebase upstream/main
```

or

```
git fetch --all --prune
git merge upstream/main
```

If you are very good at handling conflicts, then you can use rebase to resolve conflicts, as this will keep your commit logs tidy. If you are not familiar with `rebase`, then you can use `merge` to resolve conflicts.

17.2 Guidance

17.2.1 Unit test

If you cannot run the unit test of some modules for lacking of some dependencies, such as video module, you can try to install the following dependencies:

```
# Linux
sudo apt-get update -y
sudo apt-get install -y libturbojpeg
sudo apt-get install -y ffmpeg

# Windows
conda install ffmpeg
```

We should also make sure the committed code will not decrease the coverage of unit test, we could run the following command to check the coverage of unit test:

```
python -m coverage run -m pytest /path/to/test_file
python -m coverage html
# check file in htmlcov/index.html
```

17.2.2 Document rendering

If the documents are modified/added, we should check the rendering result. We could install the dependencies and run the following command to render the documents and check the results:

```
pip install -r requirements/docs.txt
cd docs/zh_cn/
# or docs/en
make html
# check file in ./docs/zh_cn/_build/html/index.html
```

17.3 Code style

17.3.1 Python

We adopt PEP8 as the preferred code style.

We use the following tools for linting and formatting:

- flake8: A wrapper around some linter tools.
- isort: A Python utility to sort imports.
- yapf: A formatter for Python files.
- codespell: A Python utility to fix common misspellings in text files.
- mdformat: Mdformat is an opinionated Markdown formatter that can be used to enforce a consistent style in Markdown files.
- docformatter: A formatter to format docstring.

Style configurations of yapf and isort can be found in setup.cfg.

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

17.3.2 C++ and CUDA

We follow the Google C++ Style Guide.

17.4 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

EIGHTEEN

PULL REQUEST (PR)

Content has been migrated to *contributing guidance*.

CHAPTER

NINETEEN

MMCV.IMAGE

mmcv.image

19.1 IO

`imfrombytes`	Read an image from bytes.
`imread`	Read an image.
`imwrite`	Write image to file.
`use_backend`	Select a backend for image decoding.

19.1.1 mmcv.image.imfrombytes

`mmcv.image.`**`imfrombytes`**(*content: bytes*, *flag: str = 'color'*, *channel_order: str = 'bgr'*, *backend: Optional[str] = None*) → numpy.ndarray

Read an image from bytes.

Parameters

- **`content`** (*bytes*) – Image bytes got from files or other streams.
- **`flag`** (*str*) – Same as *imread()*.
- **`channel_order`** (*str*) – The channel order of the output, candidates are 'bgr' and 'rgb'. Default to 'bgr'.
- **`backend`** (*str | None*) – The image decoding backend type. Options are *cv2*, *pillow*, *turbojpeg*, *tifffile*, *None*. If backend is None, the global imread_backend specified by `mmcv.use_backend()` will be used. Default: None.

Returns Loaded image array.

Return type ndarray

Examples

```
>>> img_path = '/path/to/img.jpg'
>>> with open(img_path, 'rb') as f:
>>>     img_buff = f.read()
>>> img = mmcv.imfrombytes(img_buff)
>>> img = mmcv.imfrombytes(img_buff, flag='color', channel_order='rgb')
>>> img = mmcv.imfrombytes(img_buff, backend='pillow')
>>> img = mmcv.imfrombytes(img_buff, backend='cv2')
```

19.1.2 mmcv.image.imread

mmcv.image.**imread**(*img_or_path: Union[numpy.ndarray, str, pathlib.Path]*, *flag: str = 'color'*, *channel_order: str = 'bgr'*, *backend: Optional[str] = None*, *file_client_args: Optional[dict] = None*, ***, *backend_args: Optional[dict] = None*) → numpy.ndarray

Read an image.

Parameters

- **img_or_path** (*ndarray or str or Path*) – Either a numpy array or str or pathlib.Path. If it is a numpy array (loaded image), then it will be returned as is.
- **flag** (*str*) – Flags specifying the color type of a loaded image, candidates are *color*, *grayscale*, *unchanged*, *color_ignore_orientation* and *grayscale_ignore_orientation*. By default, *cv2* and *pillow* backend would rotate the image according to its EXIF info unless called with *unchanged* or **_ignore_orientation* flags. *turbojpeg* and *tifffile* backend always ignore image's EXIF info regardless of the flag. The *turbojpeg* backend only supports *color* and *grayscale*.
- **channel_order** (*str*) – Order of channel, candidates are *bgr* and *rgb*.
- **backend** (*str | None*) – The image decoding backend type. Options are *cv2*, *pillow*, *turbojpeg*, *tifffile*, *None*. If backend is None, the global imread_backend specified by `mmcv.use_backend()` will be used. Default: None.
- **file_client_args** (*dict, optional*) – Arguments to instantiate a FileClient. See `mmengine.fileio.FileClient` for details. Default: None. It will be deprecated in future. Please use `backend_args` instead. Deprecated in version 2.0.0rc4.
- **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 2.0.0rc4.

Returns Loaded image array.

Return type ndarray

Examples

```
>>> import mmcv
>>> img_path = '/path/to/img.jpg'
>>> img = mmcv.imread(img_path)
>>> img = mmcv.imread(img_path, flag='color', channel_order='rgb',
...     backend='cv2')
>>> img = mmcv.imread(img_path, flag='color', channel_order='bgr',
...     backend='pillow')
>>> s3_img_path = 's3://bucket/img.jpg'
>>> # infer the file backend by the prefix s3
>>> img = mmcv.imread(s3_img_path)
>>> # manually set the file backend petrel
>>> img = mmcv.imread(s3_img_path, backend_args={
...     'backend': 'petrel'})
>>> http_img_path = 'http://path/to/img.jpg'
>>> img = mmcv.imread(http_img_path)
>>> img = mmcv.imread(http_img_path, backend_args={
...     'backend': 'http'})
```

19.1.3 mmcv.image.imwrite

mmcv.image.**imwrite**(*img: numpy.ndarray*, *file_path: str*, *params: Optional[list] = None*, *auto_mkdir: Optional[bool] = None*, *file_client_args: Optional[dict] = None*, ***, *backend_args: Optional[dict] = None*) → bool

Write image to file.

Warning: The parameter *auto_mkdir* will be deprecated in the future and every file clients will make directory automatically.

Parameters

- **img** (*ndarray*) – Image array to be written.
- **file_path** (*str*) – Image file path.
- **params** (*None or list*) – Same as opencv *imwrite()* interface.
- **auto_mkdir** (*bool*) – If the parent folder of *file_path* does not exist, whether to create it automatically. It will be deprecated.
- **file_client_args** (*dict, optional*) – Arguments to instantiate a FileClient. See mmengine.fileio.FileClient for details. Default: None. It will be deprecated in future. Please use backend_args instead. Deprecated in version 2.0.0rc4.
- **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 2.0.0rc4.

Returns Successful or not.

Return type bool

Examples

```
>>> # write to hard disk client
>>> ret = mmcv.imwrite(img, '/path/to/img.jpg')
>>> # infer the file backend by the prefix s3
>>> ret = mmcv.imwrite(img, 's3://bucket/img.jpg')
>>> # manually set the file backend petrel
>>> ret = mmcv.imwrite(img, 's3://bucket/img.jpg', backend_args={
...     'backend': 'petrel'})
```

19.1.4 mmcv.image.use_backend

mmcv.image.**use_backend**(*backend: str*) → None

Select a backend for image decoding.

Parameters

- **backend** (*str*) – The image decoding backend type. Options are *cv2*,
- **pillow** – //github.com/lilohuang/PyTurboJPEG)
- **(see https** (*turbojpeg*) – //github.com/lilohuang/PyTurboJPEG)
- **tifffile. turbojpeg is faster but it only supports .jpeg** (*and*) –
- **format.** (*file*) –

19.2 Color Space

bgr2gray	Convert a BGR image to grayscale image.
bgr2hls	Convert a BGR image to HLS
bgr2hsv	Convert a BGR image to HSV
bgr2rgb	Convert a BGR image to RGB
bgr2ycbcr	Convert a BGR image to YCbCr image.
gray2bgr	Convert a grayscale image to BGR image.
gray2rgb	Convert a grayscale image to RGB image.
hls2bgr	Convert a HLS image to BGR
hsv2bgr	Convert a HSV image to BGR
imconvert	Convert an image from the src colorspace to dst colorspace.
rgb2bgr	Convert a RGB image to BGR
rgb2gray	Convert a RGB image to grayscale image.
rgb2ycbcr	Convert a RGB image to YCbCr image.
ycbcr2bgr	Convert a YCbCr image to BGR image.
ycbcr2rgb	Convert a YCbCr image to RGB image.

19.2.1 mmcv.image.bgr2gray

mmcv.image.**bgr2gray**(*img: numpy.ndarray*, *keepdim: bool = False*) → numpy.ndarray

Convert a BGR image to grayscale image.

Parameters

- **img** (*ndarray*) – The input image.
- **keepdim** (*bool*) – If False (by default), then return the grayscale image with 2 dims, otherwise 3 dims.

Returns The converted grayscale image.

Return type ndarray

19.2.2 mmcv.image.bgr2hls

mmcv.image.**bgr2hls**(*img: numpy.ndarray*) → numpy.ndarray

Convert a BGR image to HLS image.

Parameters **img** (*ndarray or str*) – The input image.

Returns The converted HLS image.

Return type ndarray

19.2.3 mmcv.image.bgr2hsv

mmcv.image.**bgr2hsv**(*img: numpy.ndarray*) → numpy.ndarray

Convert a BGR image to HSV image.

Parameters **img** (*ndarray or str*) – The input image.

Returns The converted HSV image.

Return type ndarray

19.2.4 mmcv.image.bgr2rgb

mmcv.image.**bgr2rgb**(*img: numpy.ndarray*) → numpy.ndarray

Convert a BGR image to RGB image.

Parameters **img** (*ndarray or str*) – The input image.

Returns The converted RGB image.

Return type ndarray

19.2.5 mmcv.image.bgr2ycbcr

mmcv.image.**bgr2ycbcr**(*img: numpy.ndarray*, *y_only: bool = False*) → numpy.ndarray
Convert a BGR image to YCbCr image.

The bgr version of rgb2ycbcr. It implements the ITU-R BT.601 conversion for standard-definition television. See more details in https://en.wikipedia.org/wiki/YCbCr#ITU-R_BT.601_conversion.

It differs from a similar function in cv2.cvtColor: *BGR <-> YCrCb*. In OpenCV, it implements a JPEG conversion. See more details in https://en.wikipedia.org/wiki/YCbCr#JPEG_conversion.

Parameters

- **img** (*ndarray*) – The input image. It accepts: 1. np.uint8 type with range [0, 255]; 2. np.float32 type with range [0, 1].
- **y_only** (*bool*) – Whether to only return Y channel. Default: False.

Returns The converted YCbCr image. The output image has the same type and range as input image.

Return type ndarray

19.2.6 mmcv.image.gray2bgr

mmcv.image.**gray2bgr**(*img: numpy.ndarray*) → numpy.ndarray
Convert a grayscale image to BGR image.

Parameters **img** (*ndarray*) – The input image.

Returns The converted BGR image.

Return type ndarray

19.2.7 mmcv.image.gray2rgb

mmcv.image.**gray2rgb**(*img: numpy.ndarray*) → numpy.ndarray
Convert a grayscale image to RGB image.

Parameters **img** (*ndarray*) – The input image.

Returns The converted RGB image.

Return type ndarray

19.2.8 mmcv.image.hls2bgr

mmcv.image.**hls2bgr**(*img: numpy.ndarray*) → numpy.ndarray

Convert a HLS image to BGR image.

Parameters **img** (*ndarray or str*) – The input image.

Returns The converted BGR image.

Return type ndarray

19.2.9 mmcv.image.hsv2bgr

mmcv.image.**hsv2bgr**(*img: numpy.ndarray*) → numpy.ndarray

Convert a HSV image to BGR image.

Parameters **img** (*ndarray or str*) – The input image.

Returns The converted BGR image.

Return type ndarray

19.2.10 mmcv.image.imconvert

mmcv.image.**imconvert**(*img: numpy.ndarray*, *src: str*, *dst: str*) → numpy.ndarray
Convert an image from the src colorspace to dst colorspace.

Parameters

- **img** (*ndarray*) – The input image.
- **src** (*str*) – The source colorspace, e.g., 'rgb', 'hsv'.
- **dst** (*str*) – The destination colorspace, e.g., 'rgb', 'hsv'.

Returns The converted image.

Return type ndarray

19.2.11 mmcv.image.rgb2bgr

mmcv.image.**rgb2bgr**(*img: numpy.ndarray*) → numpy.ndarray

Convert a RGB image to BGR image.

Parameters **img** (*ndarray or str*) – The input image.

Returns The converted BGR image.

Return type ndarray

19.2.12 mmcv.image.rgb2gray

mmcv.image.**rgb2gray**(*img: numpy.ndarray*, *keepdim: bool = False*) → numpy.ndarray
Convert a RGB image to grayscale image.

Parameters

- **img** (*ndarray*) – The input image.
- **keepdim** (*bool*) – If False (by default), then return the grayscale image with 2 dims, otherwise 3 dims.

Returns The converted grayscale image.

Return type ndarray

19.2.13 mmcv.image.rgb2ycbcr

mmcv.image.**rgb2ycbcr**(*img: numpy.ndarray*, *y_only: bool = False*) → numpy.ndarray
Convert a RGB image to YCbCr image.

This function produces the same results as Matlab's *rgb2ycbcr* function. It implements the ITU-R BT.601 conversion for standard-definition television. See more details in https://en.wikipedia.org/wiki/YCbCr#ITU-R_BT.601_conversion.

It differs from a similar function in cv2.cvtColor: *RGB <-> YCrCb*. In OpenCV, it implements a JPEG conversion. See more details in https://en.wikipedia.org/wiki/YCbCr#JPEG_conversion.

Parameters

- **img** (*ndarray*) – The input image. It accepts: 1. np.uint8 type with range [0, 255]; 2. np.float32 type with range [0, 1].
- **y_only** (*bool*) – Whether to only return Y channel. Default: False.

Returns The converted YCbCr image. The output image has the same type and range as input image.

Return type ndarray

19.2.14 mmcv.image.ycbcr2bgr

mmcv.image.**ycbcr2bgr**(*img: numpy.ndarray*) → numpy.ndarray
Convert a YCbCr image to BGR image.

The bgr version of ycbcr2rgb. It implements the ITU-R BT.601 conversion for standard-definition television. See more details in https://en.wikipedia.org/wiki/YCbCr#ITU-R_BT.601_conversion.

It differs from a similar function in cv2.cvtColor: *YCrCb <-> BGR*. In OpenCV, it implements a JPEG conversion. See more details in https://en.wikipedia.org/wiki/YCbCr#JPEG_conversion.

Parameters **img** (*ndarray*) – The input image. It accepts: 1. np.uint8 type with range [0, 255]; 2. np.float32 type with range [0, 1].

Returns The converted BGR image. The output image has the same type and range as input image.

Return type ndarray

19.2.15 mmcv.image.ycbcr2rgb

mmcv.image.**ycbcr2rgb**(*img: numpy.ndarray*) → numpy.ndarray
Convert a YCbCr image to RGB image.

This function produces the same results as Matlab's ycbcr2rgb function. It implements the ITU-R BT.601 conversion for standard-definition television. See more details in https://en.wikipedia.org/wiki/YCbCr#ITU-R_BT.601_conversion.

It differs from a similar function in cv2.cvtColor: *YCrCb <-> RGB*. In OpenCV, it implements a JPEG conversion. See more details in https://en.wikipedia.org/wiki/YCbCr#JPEG_conversion.

Parameters **img** (*ndarray*) – The input image. It accepts: 1. np.uint8 type with range [0, 255]; 2. np.float32 type with range [0, 1].

Returns The converted RGB image. The output image has the same type and range as input image.

Return type ndarray

19.3 Geometric

cutout	Randomly cut out a rectangle from the original img.
imcrop	Crop image patches.
imflip	Flip an image horizontally or vertically.
impad	Pad the given image to a certain shape or pad on all sides with specified padding mode and padding value.
impad_to_multiple	Pad an image to ensure each edge to be multiple to some number.
imrescale	Resize image while keeping the aspect ratio.
imresize	Resize image to a given size.
imresize_like	Resize image to the same size of a given image.
imresize_to_multiple	Resize image according to a given size or scale factor and then rounds up the the resized or rescaled image size to the nearest value that can be divided by the divisor.
imrotate	Rotate an image.
imshear	Shear an image.
imtranslate	Translate an image.
rescale_size	Calculate the new size to be rescaled to.

19.3.1 mmcv.image.cutout

mmcv.image.**cutout**(*img: numpy.ndarray*, *shape: Union[int, Tuple[int, int]]*, *pad_val: Union[int, float, tuple] = 0*) → numpy.ndarray

Randomly cut out a rectangle from the original img.

Parameters

- **img** (*ndarray*) – Image to be cutout.
- **shape** (*int | tuple[int]*) – Expected cutout shape (h, w). If given as a int, the value will be used for both h and w.
- **pad_val** (*int | float | tuple[int | float]*) – Values to be filled in the cut area. Defaults to 0.

Returns The cutout image.

Return type ndarray

19.3.2 mmcv.image.imcrop

mmcv.image.**imcrop**(*img: numpy.ndarray*, *bboxes: numpy.ndarray*, *scale: float = 1.0*, *pad_fill: Optional[Union[float, list]] = None*) → Union[numpy.ndarray, List[numpy.ndarray]]

Crop image patches.

3 steps: scale the bboxes -> clip bboxes -> crop and pad.

Parameters

- **img** (*ndarray*) – Image to be cropped.
- **bboxes** (*ndarray*) – Shape (k, 4) or (4,), location of cropped bboxes.
- **scale** (*float, optional*) – Scale ratio of bboxes, the default value 1.0 means no scaling.

- **pad_fill** (*Number | list[Number]*) – Value to be filled for padding. Default: None, which means no padding.

Returns The cropped image patches.

Return type list[ndarray] | ndarray

19.3.3 mmcv.image.imflip

mmcv.image.**imflip**(*img: numpy.ndarray*, *direction: str = 'horizontal'*) → numpy.ndarray

Flip an image horizontally or vertically.

Parameters

- **img** (*ndarray*) – Image to be flipped.
- **direction** (*str*) – The flip direction, either "horizontal" or "vertical" or "diagonal".

Returns The flipped image.

Return type ndarray

19.3.4 mmcv.image.impad

mmcv.image.**impad**(*img: numpy.ndarray*, ***, *shape: Optional[Tuple[int, int]] = None*, *padding: Optional[Union[int, tuple]] = None*, *pad_val: Union[float, List] = 0*, *padding_mode: str = 'constant'*) → numpy.ndarray

Pad the given image to a certain shape or pad on all sides with specified padding mode and padding value.

Parameters

- **img** (*ndarray*) – Image to be padded.
- **shape** (*tuple[int]*) – Expected padding shape (h, w). Default: None.
- **padding** (*int or tuple[int]*) – Padding on each border. If a single int is provided this is used to pad all borders. If tuple of length 2 is provided this is the padding on left/right and top/bottom respectively. If a tuple of length 4 is provided this is the padding for the left, top, right and bottom borders respectively. Default: None. Note that *shape* and *padding* can not be both set.
- **pad_val** (*Number | Sequence[Number]*) – Values to be filled in padding areas when padding_mode is 'constant'. Default: 0.
- **padding_mode** (*str*) – Type of padding. Should be: constant, edge, reflect or symmetric. Default: constant.
  - constant: pads with a constant value, this value is specified with pad_val.
  - edge: pads with the last value at the edge of the image.
  - reflect: pads with reflection of image without repeating the last value on the edge. For example, padding [1, 2, 3, 4] with 2 elements on both sides in reflect mode will result in [3, 2, 1, 2, 3, 4, 3, 2].
  - symmetric: pads with reflection of image repeating the last value on the edge. For example, padding [1, 2, 3, 4] with 2 elements on both sides in symmetric mode will result in [2, 1, 1, 2, 3, 4, 4, 3]

Returns The padded image.

Return type ndarray

19.3.5 mmcv.image.impad_to_multiple

mmcv.image.**impad_to_multiple**(*img: numpy.ndarray*, *divisor: int*, *pad_val: Union[float, List] = 0*) → numpy.ndarray

Pad an image to ensure each edge to be multiple to some number.

Parameters

- **img** (*ndarray*) – Image to be padded.
- **divisor** (*int*) – Padded image edges will be multiple to divisor.
- **pad_val** (*Number | Sequence[Number]*) – Same as *impad()*.

Returns The padded image.

Return type ndarray

19.3.6 mmcv.image.imrescale

mmcv.image.**imrescale**(*img: numpy.ndarray*, *scale: Union[float, int, Tuple[int, int]]*, *return_scale: bool = False*, *interpolation: str = 'bilinear'*, *backend: Optional[str] = None*) → Union[numpy.ndarray, Tuple[numpy.ndarray, float]]

Resize image while keeping the aspect ratio.

Parameters

- **img** (*ndarray*) – The input image.
- **scale** (*float | int | tuple[int]*) – The scaling factor or maximum size. If it is a float number or an integer, then the image will be rescaled by this factor, else if it is a tuple of 2 integers, then the image will be rescaled as large as possible within the scale.
- **return_scale** (*bool*) – Whether to return the scaling factor besides the rescaled image.
- **interpolation** (*str*) – Same as `resize()`.
- **backend** (*str | None*) – Same as `resize()`.

Returns The rescaled image.

Return type ndarray

19.3.7 mmcv.image.imresize

mmcv.image.**imresize**(*img: numpy.ndarray*, *size: Tuple[int, int]*, *return_scale: bool = False*, *interpolation: str = 'bilinear'*, *out: Optional[numpy.ndarray] = None*, *backend: Optional[str] = None*) → Union[Tuple[numpy.ndarray, float, float], numpy.ndarray]

Resize image to a given size.

Parameters

- **img** (*ndarray*) – The input image.
- **size** (*tuple[int]*) – Target size (w, h).
- **return_scale** (*bool*) – Whether to return *w_scale* and *h_scale*.
- **interpolation** (*str*) – Interpolation method, accepted values are "nearest", "bilinear", "bicubic", "area", "lanczos" for 'cv2' backend, "nearest", "bilinear" for 'pillow' backend.
- **out** (*ndarray*) – The output destination.

- **backend** (*str | None*) – The image resize backend type. Options are *cv2*, *pillow*, *None*. If backend is None, the global imread_backend specified by `mmcv.use_backend()` will be used. Default: None.

Returns (*resized_img*, *w_scale*, *h_scale*) or *resized_img*.

Return type tuple | ndarray

19.3.8 mmcv.image.imresize_like

mmcv.image.**imresize_like**(*img: numpy.ndarray*, *dst_img: numpy.ndarray*, *return_scale: bool = False*, *interpolation: str = 'bilinear'*, *backend: Optional[str] = None*) → Union[Tuple[numpy.ndarray, float, float], numpy.ndarray]

Resize image to the same size of a given image.

Parameters

- **img** (*ndarray*) – The input image.
- **dst_img** (*ndarray*) – The target image.
- **return_scale** (*bool*) – Whether to return *w_scale* and *h_scale*.
- **interpolation** (*str*) – Same as `resize()`.
- **backend** (*str | None*) – Same as `resize()`.

Returns (*resized_img*, *w_scale*, *h_scale*) or *resized_img*.

Return type tuple or ndarray

19.3.9 mmcv.image.imresize_to_multiple

mmcv.image.**imresize_to_multiple**(*img: numpy.ndarray*, *divisor: Union[int, Tuple[int, int]]*, *size: Optional[Union[int, Tuple[int, int]]] = None*, *scale_factor: Optional[Union[float, int, Tuple[float, float], Tuple[int, int]]] = None*, *keep_ratio: bool = False*, *return_scale: bool = False*, *interpolation: str = 'bilinear'*, *out: Optional[numpy.ndarray] = None*, *backend: Optional[str] = None*) → Union[Tuple[numpy.ndarray, float, float], numpy.ndarray]

Resize image according to a given size or scale factor and then rounds up the the resized or rescaled image size to the nearest value that can be divided by the divisor.

Parameters

- **img** (*ndarray*) – The input image.
- **divisor** (*int | tuple*) – Resized image size will be a multiple of divisor. If divisor is a tuple, divisor should be (w_divisor, h_divisor).
- **size** (*None | int | tuple[int]*) – Target size (w, h). Default: None.
- **scale_factor** (*None | float | int | tuple[float] | tuple[int]*) – Multiplier for spatial size. Should match input size if it is a tuple and the 2D style is (w_scale_factor, h_scale_factor). Default: None.
- **keep_ratio** (*bool*) – Whether to keep the aspect ratio when resizing the image. Default: False.
- **return_scale** (*bool*) – Whether to return *w_scale* and *h_scale*.

- **interpolation** (*str*) – Interpolation method, accepted values are "nearest", "bilinear", "bicubic", "area", "lanczos" for 'cv2' backend, "nearest", "bilinear" for 'pillow' backend.
- **out** (*ndarray*) – The output destination.
- **backend** (*str | None*) – The image resize backend type. Options are *cv2*, *pillow*, *None*. If backend is None, the global imread_backend specified by `mmcv.use_backend()` will be used. Default: None.

Returns (*resized_img*, *w_scale*, *h_scale*) or *resized_img*.

Return type tuple | ndarray

19.3.10 mmcv.image.imrotate

mmcv.image.**imrotate**(*img: numpy.ndarray*, *angle: float*, *center: Optional[Tuple[float, float]] = None*, *scale: float = 1.0*, *border_value: int = 0*, *interpolation: str = 'bilinear'*, *auto_bound: bool = False*, *border_mode: str = 'constant'*) → numpy.ndarray

Rotate an image.

Parameters

- **img** (*np.ndarray*) – Image to be rotated.
- **angle** (*float*) – Rotation angle in degrees, positive values mean clockwise rotation.
- **center** (*tuple[float], optional*) – Center point (w, h) of the rotation in the source image. If not specified, the center of the image will be used.
- **scale** (*float*) – Isotropic scale factor.
- **border_value** (*int*) – Border value used in case of a constant border. Defaults to 0.
- **interpolation** (*str*) – Same as `resize()`.
- **auto_bound** (*bool*) – Whether to adjust the image size to cover the whole rotated image.
- **border_mode** (*str*) – Pixel extrapolation method. Defaults to 'constant'.

Returns The rotated image.

Return type np.ndarray

19.3.11 mmcv.image.imshear

mmcv.image.**imshear**(*img: numpy.ndarray*, *magnitude: Union[int, float]*, *direction: str = 'horizontal'*, *border_value: Union[int, Tuple[int, int]] = 0*, *interpolation: str = 'bilinear'*) → numpy.ndarray

Shear an image.

Parameters

- **img** (*ndarray*) – Image to be sheared with format (h, w) or (h, w, c).
- **magnitude** (*int | float*) – The magnitude used for shear.
- **direction** (*str*) – The flip direction, either "horizontal" or "vertical".
- **border_value** (*int | tuple[int]*) – Value used in case of a constant border.
- **interpolation** (*str*) – Same as `resize()`.

Returns The sheared image.

Return type ndarray

19.3.12 mmcv.image.imtranslate

mmcv.image.**imtranslate**(*img: numpy.ndarray*, *offset: Union[int, float]*, *direction: str = 'horizontal'*, *border_value: Union[int, tuple] = 0*, *interpolation: str = 'bilinear'*) → numpy.ndarray

Translate an image.

Parameters

- **img** (*ndarray*) – Image to be translated with format (h, w) or (h, w, c).
- **offset** (*int | float*) – The offset used for translate.
- **direction** (*str*) – The translate direction, either "horizontal" or "vertical".
- **border_value** (*int | tuple[int]*) – Value used in case of a constant border.
- **interpolation** (*str*) – Same as `resize()`.

Returns The translated image.

Return type ndarray

19.3.13 mmcv.image.rescale_size

mmcv.image.**rescale_size**(*old_size: tuple*, *scale: Union[float, int, Tuple[int, int]]*, *return_scale: bool = False*) → tuple

Calculate the new size to be rescaled to.

Parameters

- **old_size** (*tuple[int]*) – The old size (w, h) of image.
- **scale** (*float | int | tuple[int]*) – The scaling factor or maximum size. If it is a float number or an integer, then the image will be rescaled by this factor, else if it is a tuple of 2 integers, then the image will be rescaled as large as possible within the scale.
- **return_scale** (*bool*) – Whether to return the scaling factor besides the rescaled image size.

Returns The new rescaled image size.

Return type tuple[int]

19.4 Photometric

adjust_brightness	Adjust image brightness.
adjust_color	It blends the source image and its gray image:
adjust_contrast	Adjust image contrast.
adjust_hue	Adjust hue of an image.
adjust_lighting	AlexNet-style PCA jitter.
adjust_sharpness	Adjust image sharpness.
auto_contrast	Auto adjust image contrast.
clahe	Use CLAHE method to process the image.

continues on next page

Table 4 – continued from previous page

imdenormalize	
imequalize	Equalize the image histogram.
iminvert	Invert (negate) an image.
imnormalize	Normalize an image with mean and std.
lut_transform	Transform array by look-up table.
posterize	Posterize an image (reduce the number of bits for each color channel)
solarize	Solarize an image (invert all pixel values above a threshold)

19.4.1 mmcv.image.adjust_brightness

mmcv.image.**adjust_brightness**(*img*, *factor=1.0*, *backend=None*)

Adjust image brightness.

This function controls the brightness of an image. An enhancement factor of 0.0 gives a black image. A factor of 1.0 gives the original image. This function blends the source image and the degenerated black image:

$$output = img * factor + degenerated * (1 - factor)$$

Parameters

- **img** (*ndarray*) – Image to be brightened.
- **factor** (*float*) – A value controls the enhancement. Factor 1.0 returns the original image, lower factors mean less color (brightness, contrast, etc), and higher values more. Default 1.
- **backend** (*str | None*) – The image processing backend type. Options are *cv2*, *pillow*, *None*. If backend is None, the global `imread_backend` specified by `mmcv.use_backend()` will be used. Defaults to None.

Returns The brightened image.

Return type ndarray

19.4.2 mmcv.image.adjust_color

mmcv.image.**adjust_color**(*img*, *alpha=1*, *beta=None*, *gamma=0*, *backend=None*)

It blends the source image and its gray image:

$$output = img * alpha + gray\_img * beta + gamma$$

Parameters

- **img** (*ndarray*) – The input source image.
- **alpha** (*int | float*) – Weight for the source image. Default 1.
- **beta** (*int | float*) – Weight for the converted gray image. If None, it's assigned the value (1 - *alpha*).
- **gamma** (*int | float*) – Scalar added to each sum. Same as `cv2.addWeighted()`. Default 0.

- **backend** (*str | None*) – The image processing backend type. Options are *cv2*, *pillow*, *None*. If backend is None, the global `imread_backend` specified by `mmcv.use_backend()` will be used. Defaults to None.

Returns Colored image which has the same size and dtype as input.

Return type ndarray

19.4.3 mmcv.image.adjust_contrast

mmcv.image.**adjust_contrast**(*img*, *factor=1.0*, *backend=None*)

Adjust image contrast.

This function controls the contrast of an image. An enhancement factor of 0.0 gives a solid grey image. A factor of 1.0 gives the original image. It blends the source image and the degenerated mean image:

$$output = img * factor + degenerated * (1 - factor)$$

Parameters

- **img** (*ndarray*) – Image to be contrasted. BGR order.
- **factor** (*float*) – Same as `mmcv.adjust_brightness()`.
- **backend** (*str | None*) – The image processing backend type. Options are *cv2*, *pillow*, *None*. If backend is None, the global `imread_backend` specified by `mmcv.use_backend()` will be used. Defaults to None.

Returns The contrasted image.

Return type ndarray

19.4.4 mmcv.image.adjust_hue

mmcv.image.**adjust_hue**(*img: numpy.ndarray*, *hue_factor: float*, *backend: Optional[str] = None*) → numpy.ndarray

Adjust hue of an image.

The image hue is adjusted by converting the image to HSV and cyclically shifting the intensities in the hue channel (H). The image is then converted back to original image mode.

hue_factor is the amount of shift in H channel and must be in the interval *[-0.5, 0.5]*.

Modified from https://github.com/pytorch/vision/blob/main/torchvision/ transforms/functional.py

Parameters

- **img** (*ndarray*) – Image to be adjusted.
- **hue_factor** (*float*) – How much to shift the hue channel. Should be in [-0.5, 0.5]. 0.5 and -0.5 give complete reversal of hue channel in HSV space in positive and negative direction respectively. 0 means no shift. Therefore, both -0.5 and 0.5 will give an image with complementary colors while 0 gives the original image.
- **backend** (*str | None*) – The image processing backend type. Options are *cv2*, *pillow*, *None*. If backend is None, the global `imread_backend` specified by `mmcv.use_backend()` will be used. Defaults to None.

Returns Hue adjusted image.

Return type ndarray

19.4.5 mmcv.image.adjust_lighting

mmcv.image.**adjust_lighting**(*img*, *eigval*, *eigvec*, *alphastd=0.1*, *to_rgb=True*)
AlexNet-style PCA jitter.

This data augmentation is proposed in ImageNet Classification with Deep Convolutional Neural Networks.

Parameters

- **img** (*ndarray*) – Image to be adjusted lighting. BGR order.
- **eigval** (*ndarray*) – the eigenvalue of the convariance matrix of pixel values, respectively.
- **eigvec** (*ndarray*) – the eigenvector of the convariance matrix of pixel values, respectively.
- **alphastd** (*float*) – The standard deviation for distribution of alpha. Defaults to 0.1
- **to_rgb** (*bool*) – Whether to convert img to rgb.

Returns The adjusted image.

Return type ndarray

19.4.6 mmcv.image.adjust_sharpness

mmcv.image.**adjust_sharpness**(*img*, *factor=1.0*, *kernel=None*)
Adjust image sharpness.

This function controls the sharpness of an image. An enhancement factor of 0.0 gives a blurred image. A factor of 1.0 gives the original image. And a factor of 2.0 gives a sharpened image. It blends the source image and the degenerated mean image:

$$output = img * factor + degenerated * (1 - factor)$$

Parameters

- **img** (*ndarray*) – Image to be sharpened. BGR order.
- **factor** (*float*) – Same as `mmcv.adjust_brightness()`.
- **kernel** (*np.ndarray, optional*) – Filter kernel to be applied on the img to obtain the degenerated img. Defaults to None.

Note: No value sanity check is enforced on the kernel set by users. So with an inappropriate kernel, the `adjust_sharpness` may fail to perform the function its name indicates but end up performing whatever transform determined by the kernel.

Returns The sharpened image.

Return type ndarray

19.4.7 mmcv.image.auto_contrast

mmcv.image.**auto_contrast**(*img*, *cutoff=0*)
Auto adjust image contrast.

This function maximize (normalize) image contrast by first removing cutoff percent of the lightest and darkest pixels from the histogram and remapping the image so that the darkest pixel becomes black (0), and the lightest becomes white (255).

Parameters

- **img** (*ndarray*) – Image to be contrasted. BGR order.
- **cutoff** (*int | float | tuple*) – The cutoff percent of the lightest and darkest pixels to be removed. If given as tuple, it shall be (low, high). Otherwise, the single value will be used for both. Defaults to 0.

Returns The contrasted image.

Return type ndarray

19.4.8 mmcv.image.clahe

mmcv.image.**clahe**(*img*, *clip_limit=40.0*, *tile_grid_size=(8, 8)*)
Use CLAHE method to process the image.

See *ZUIDERVELD,K. Contrast Limited Adaptive Histogram Equalization[J]. Graphics Gems, 1994:474-485.* for more information.

Parameters

- **img** (*ndarray*) – Image to be processed.
- **clip_limit** (*float*) – Threshold for contrast limiting. Default: 40.0.
- **tile_grid_size** (*tuple[int]*) – Size of grid for histogram equalization. Input image will be divided into equally sized rectangular tiles. It defines the number of tiles in row and column. Default: (8, 8).

Returns The processed image.

Return type ndarray

19.4.9 mmcv.image.imdenormalize

mmcv.image.**imdenormalize**(*img*, *mean*, *std*, *to_bgr=True*)

19.4.10 mmcv.image.imequalize

mmcv.image.**imequalize**(*img*)
Equalize the image histogram.

This function applies a non-linear mapping to the input image, in order to create a uniform distribution of grayscale values in the output image.

Parameters **img** (*ndarray*) – Image to be equalized.

Returns The equalized image.

Return type ndarray

19.4.11 mmcv.image.iminvert

mmcv.image.**iminvert**(*img*)
Invert (negate) an image.

Parameters **img** (*ndarray*) – Image to be inverted.

Returns The inverted image.

Return type ndarray

19.4.12 mmcv.image.imnormalize

mmcv.image.**imnormalize**(*img*, *mean*, *std*, *to_rgb=True*)
Normalize an image with mean and std.

Parameters

- **img** (*ndarray*) – Image to be normalized.
- **mean** (*ndarray*) – The mean to be used for normalize.
- **std** (*ndarray*) – The std to be used for normalize.
- **to_rgb** (*bool*) – Whether to convert to rgb.

Returns The normalized image.

Return type ndarray

19.4.13 mmcv.image.lut_transform

mmcv.image.**lut_transform**(*img*, *lut_table*)
Transform array by look-up table.

The function lut_transform fills the output array with values from the look-up table. Indices of the entries are taken from the input array.

Parameters

- **img** (*ndarray*) – Image to be transformed.
- **lut_table** (*ndarray*) – look-up table of 256 elements; in case of multi-channel input array, the table should either have a single channel (in this case the same table is used for all channels) or the same number of channels as in the input array.

Returns The transformed image.

Return type ndarray

19.4.14 mmcv.image.posterize

mmcv.image.**posterize**(*img*, *bits*)
Posterize an image (reduce the number of bits for each color channel)

Parameters

- **img** (*ndarray*) – Image to be posterized.
- **bits** (*int*) – Number of bits (1 to 8) to use for posterizing.

Returns The posterized image.

Return type ndarray

19.4.15 mmcv.image.solarize

mmcv.image.**solarize**(*img*, *thr=128*)
Solarize an image (invert all pixel values above a threshold)

Parameters

- **img** (*ndarray*) – Image to be solarized.
- **thr** (*int*) – Threshold for solarizing (0 - 255).

Returns The solarized image.

Return type ndarray

19.5 Miscellaneous

tensor2imgs	Convert tensor to 3-channel images or 1-channel gray images.

19.5.1 mmcv.image.tensor2imgs

mmcv.image.**tensor2imgs**(*tensor*, *mean: Optional[tuple] = None*, *std: Optional[tuple] = None*, *to_rgb: bool = True*) → list
Convert tensor to 3-channel images or 1-channel gray images.

Parameters

- **tensor** (*torch.Tensor*) – Tensor that contains multiple images, shape (N, C, H, W). C can be either 3 or 1.
- **mean** (*tuple[float], optional*) – Mean of images. If None, (0, 0, 0) will be used for tensor with 3-channel, while (0,) for tensor with 1-channel. Defaults to None.
- **std** (*tuple[float], optional*) – Standard deviation of images. If None, (1, 1, 1) will be used for tensor with 3-channel, while (1,) for tensor with 1-channel. Defaults to None.
- **to_rgb** (*bool, optional*) – Whether the tensor was converted to RGB format in the first place. If so, convert it back to BGR. For the tensor with 1 channel, it must be False. Defaults to True.

Returns A list that contains multiple images.

Return type list[np.ndarray]

CHAPTER

TWENTY

MMCV.VIDEO

mmcv.video

20.1 IO

VideoReader	Video class with similar usage to a list object.
Cache	

20.1.1 VideoReader

class mmcv.video.**VideoReader**(*filename*, *cache_capacity=10*)

Video class with similar usage to a list object.

This video wrapper class provides convenient apis to access frames. There exists an issue of OpenCV's VideoCapture class that jumping to a certain frame may be inaccurate. It is fixed in this class by checking the position after jumping each time. Cache is used when decoding videos. So if the same frame is visited for the second time, there is no need to decode again if it is stored in the cache.

Examples

```
>>> import mmcv
>>> v = mmcv.VideoReader('sample.mp4')
>>> len(v)  # get the total frame number with `len()`
120
>>> for img in v:  # v is iterable
>>>     mmcv.imshow(img)
>>> v[5]  # get the 6th frame
```

current_frame()

Get the current frame (frame that is just visited).

Returns If the video is fresh, return None, otherwise return the frame.

Return type ndarray or None

cvt2frames(*frame_dir*, *file_start=0*, *filename_tmpl='{:06d}.jpg'*, *start=0*, *max_num=0*, *show_progress=True*)

Convert a video to frame images.

Parameters

- **frame_dir** (*str*) – Output directory to store all the frame images.
- **file_start** (*int*) – Filenames will start from the specified number.
- **filename_tmpl** (*str*) – Filename template with the index as the placeholder.
- **start** (*int*) – The starting frame index.
- **max_num** (*int*) – Maximum number of frames to be written.
- **show_progress** (*bool*) – Whether to show a progress bar.

property fourcc

"Four character code" of the video.

Type str

property fps

FPS of the video.

Type float

property frame_cnt

Total frames of the video.

Type int

get_frame(*frame_id*)

Get frame by index.

Parameters **frame_id** (*int*) – Index of the expected frame, 0-based.

Returns Return the frame if successful, otherwise None.

Return type ndarray or None

property height

Height of video frames.

Type int

property opened

Indicate whether the video is opened.

Type bool

property position

Current cursor position, indicating frame decoded.

Type int

read()

Read the next frame.

If the next frame have been decoded before and in the cache, then return it directly, otherwise decode, cache and return it.

Returns Return the frame if successful, otherwise None.

Return type ndarray or None

property resolution
Video resolution (width, height).

Type tuple

property vcap
The raw VideoCapture object.

Type cv2.VideoCapture

property width
Width of video frames.

Type int

20.1.2 Cache

class mmcv.video.**Cache**(*capacity*)

frames2video	Read the frame images from a directory and join them as a video.

20.1.3 mmcv.video.frames2video

mmcv.video.**frames2video**(*frame_dir: str*, *video_file: str*, *fps: float = 30*, *fourcc: str = 'XVID'*, *filename_tmpl: str = '{:06d}.jpg'*, *start: int = 0*, *end: int = 0*, *show_progress: bool = True*) → None
Read the frame images from a directory and join them as a video.

Parameters

- **frame_dir** (*str*) – The directory containing video frames.
- **video_file** (*str*) – Output filename.
- **fps** (*float*) – FPS of the output video.
- **fourcc** (*str*) – Fourcc of the output video, this should be compatible with the output file type.
- **filename_tmpl** (*str*) – Filename template with the index as the variable.
- **start** (*int*) – Starting frame index.
- **end** (*int*) – Ending frame index.
- **show_progress** (*bool*) – Whether to show a progress bar.

20.2 Optical Flow

dequantize_flow	Recover from quantized flow.
flow_from_bytes	Read dense optical flow from bytes.
flow_warp	Use flow to warp img.
flowread	Read an optical flow map.
flowwrite	Write optical flow to file.
quantize_flow	Quantize flow to [0, 255].
sparse_flow_from_bytes	Read the optical flow in KITTI datasets from bytes.

20.2.1 mmcv.video.dequantize_flow

mmcv.video.**dequantize_flow**(*dx: numpy.ndarray*, *dy: numpy.ndarray*, *max_val: float = 0.02*, *denorm: bool = True*) → numpy.ndarray

Recover from quantized flow.

Parameters

- **dx** (*ndarray*) – Quantized dx.
- **dy** (*ndarray*) – Quantized dy.
- **max_val** (*float*) – Maximum value used when quantizing.
- **denorm** (*bool*) – Whether to multiply flow values with width/height.

Returns Dequantized flow.

Return type ndarray

20.2.2 mmcv.video.flow_from_bytes

mmcv.video.**flow_from_bytes**(*content: bytes*) → numpy.ndarray

Read dense optical flow from bytes.

Note: This load optical flow function works for FlyingChairs, FlyingThings3D, Sintel, FlyingChairsOcc datasets, but cannot load the data from ChairsSDHom.

Parameters **content** (*bytes*) – Optical flow bytes got from files or other streams.

Returns Loaded optical flow with the shape (H, W, 2).

Return type ndarray

20.2.3 mmcv.video.flow_warp

mmcv.video.**flow_warp**(*img: numpy.ndarray*, *flow: numpy.ndarray*, *filling_value: int = 0*, *interpolate_mode: str = 'nearest'*) → numpy.ndarray

Use flow to warp img.

Parameters

- **img** (*ndarray*) – Image to be warped.
- **flow** (*ndarray*) – Optical Flow.
- **filling_value** (*int*) – The missing pixels will be set with filling_value.
- **interpolate_mode** (*str*) – bilinear -> Bilinear Interpolation; nearest -> Nearest Neighbor.

Returns Warped image with the same shape of img

Return type ndarray

20.2.4 mmcv.video.flowread

mmcv.video.**flowread**(*flow_or_path: Union[numpy.ndarray, str]*, *quantize: bool = False*, *concat_axis: int = 0*, **args*, ***kwargs*) → numpy.ndarray

Read an optical flow map.

Parameters

- **flow_or_path** (*ndarray or str*) – A flow map or filepath.
- **quantize** (*bool*) – whether to read quantized pair, if set to True, remaining args will be passed to *dequantize_flow()*.
- **concat_axis** (*int*) – The axis that dx and dy are concatenated, can be either 0 or 1. Ignored if quantize is False.

Returns Optical flow represented as a (h, w, 2) numpy array

Return type ndarray

20.2.5 mmcv.video.flowwrite

mmcv.video.**flowwrite**(*flow: numpy.ndarray*, *filename: str*, *quantize: bool = False*, *concat_axis: int = 0*, **args*, ***kwargs*) → None

Write optical flow to file.

If the flow is not quantized, it will be saved as a .flo file losslessly, otherwise a jpeg image which is lossy but of much smaller size. (dx and dy will be concatenated horizontally into a single image if quantize is True.)

Parameters

- **flow** (*ndarray*) – (h, w, 2) array of optical flow.
- **filename** (*str*) – Output filepath.
- **quantize** (*bool*) – Whether to quantize the flow and save it to 2 jpeg images. If set to True, remaining args will be passed to *quantize_flow()*.
- **concat_axis** (*int*) – The axis that dx and dy are concatenated, can be either 0 or 1. Ignored if quantize is False.

20.2.6 mmcv.video.quantize_flow

mmcv.video.**quantize_flow**(*flow: numpy.ndarray*, *max_val: float = 0.02*, *norm: bool = True*) → tuple
Quantize flow to [0, 255].

After this step, the size of flow will be much smaller, and can be dumped as jpeg images.

Parameters

- **flow** (*ndarray*) – (h, w, 2) array of optical flow.
- **max_val** (*float*) – Maximum value of flow, values beyond [-max_val, max_val] will be truncated.
- **norm** (*bool*) – Whether to divide flow values by image width/height.

Returns Quantized dx and dy.

Return type tuple[ndarray]

20.2.7 mmcv.video.sparse_flow_from_bytes

mmcv.video.**sparse_flow_from_bytes**(*content: bytes*) → Tuple[numpy.ndarray, numpy.ndarray]
Read the optical flow in KITTI datasets from bytes.

This function is modified from RAFT load the KITTI datasets.

Parameters **content** (*bytes*) – Optical flow bytes got from files or other streams.

Returns Loaded optical flow with the shape (H, W, 2) and flow valid mask with the shape (H, W).

Return type Tuple(ndarray, ndarray)

20.3 Video Processing

concat_video	Concatenate multiple videos into a single one.
convert_video	Convert a video with ffmpeg.
cut_video	Cut a clip from a video.
resize_video	Resize a video.

20.3.1 mmcv.video.concat_video

mmcv.video.**concat_video**(*video_list: List*, *out_file: str*, *vcodec: Optional[str] = None*, *acodec: Optional[str] = None*, *log_level: str = 'info'*, *print_cmd: bool = False*) → None
Concatenate multiple videos into a single one.

Parameters

- **video_list** (*list*) – A list of video filenames
- **out_file** (*str*) – Output video filename
- **vcodec** (*None or str*) – Output video codec, None for unchanged
- **acodec** (*None or str*) – Output audio codec, None for unchanged
- **log_level** (*str*) – Logging level of ffmpeg.

- **print_cmd** (*bool*) – Whether to print the final ffmpeg command.

20.3.2 mmcv.video.convert_video

mmcv.video.**convert_video**(*in_file: str*, *out_file: str*, *print_cmd: bool = False*, *pre_options: str = ''*, ***kwargs*) → None

Convert a video with ffmpeg.

This provides a general api to ffmpeg, the executed command is:

```
`ffmpeg -y <pre_options> -i <in_file> <options> <out_file>`
```

Options(kwargs) are mapped to ffmpeg commands with the following rules:

- key=val: "-key val"
- key=True: "-key"
- key=False: ""

Parameters

- **in_file** (*str*) – Input video filename.
- **out_file** (*str*) – Output video filename.
- **pre_options** (*str*) – Options appears before "-i <in_file>".
- **print_cmd** (*bool*) – Whether to print the final ffmpeg command.

20.3.3 mmcv.video.cut_video

mmcv.video.**cut_video**(*in_file: str*, *out_file: str*, *start: Optional[float] = None*, *end: Optional[float] = None*, *vcodec: Optional[str] = None*, *acodec: Optional[str] = None*, *log_level: str = 'info'*, *print_cmd: bool = False*) → None

Cut a clip from a video.

Parameters

- **in_file** (*str*) – Input video filename.
- **out_file** (*str*) – Output video filename.
- **start** (*None or float*) – Start time (in seconds).
- **end** (*None or float*) – End time (in seconds).
- **vcodec** (*None or str*) – Output video codec, None for unchanged.
- **acodec** (*None or str*) – Output audio codec, None for unchanged.
- **log_level** (*str*) – Logging level of ffmpeg.
- **print_cmd** (*bool*) – Whether to print the final ffmpeg command.

20.3.4 mmcv.video.resize_video

mmcv.video.**resize_video**(*in_file: str*, *out_file: str*, *size: Optional[tuple] = None*, *ratio: Optional[Union[tuple, float]] = None*, *keep_ar: bool = False*, *log_level: str = 'info'*, *print_cmd: bool = False*) → None

Resize a video.

Parameters

- **in_file** (*str*) – Input video filename.
- **out_file** (*str*) – Output video filename.
- **size** (*tuple*) – Expected size (w, h), eg, (320, 240) or (320, -1).
- **ratio** (*tuple or float*) – Expected resize ratio, (2, 0.5) means (w*2, h*0.5).
- **keep_ar** (*bool*) – Whether to keep original aspect ratio.
- **log_level** (*str*) – Logging level of ffmpeg.
- **print_cmd** (*bool*) – Whether to print the final ffmpeg command.

CHAPTER

TWENTYONE

MMCV.VISUALIZATION

mmcv.visualization

- *Color*
- *Image*
- *Optical Flow*

21.1 Color

Color	An enum that defines common colors.

21.1.1 Color

class mmcv.visualization.**Color**(*value*)

An enum that defines common colors.

Contains red, green, blue, cyan, yellow, magenta, white and black.

color_val	Convert various input to color tuples.

21.1.2 mmcv.visualization.color_val

mmcv.visualization.**color_val**(*color: Union[*mmcv.visualization.color.Color*, str, tuple, int, numpy.ndarray]*) → tuple

Convert various input to color tuples.

Parameters **color** (*Color*/str/tuple/int/ndarray) – Color inputs

Returns A tuple of 3 integers indicating BGR channels.

Return type tuple[int]

21.2 Image

imshow	Show an image.
imshow_bboxes	Draw bboxes on an image.
imshow_det_bboxes	Draw bboxes and class labels (with scores) on an image.

21.2.1 mmcv.visualization.imshow

mmcv.visualization.**imshow**(*img: Union[str, numpy.ndarray]*, *win_name: str = ''*, *wait_time: int = 0*)

Show an image.

Parameters

- **img** (*str or ndarray*) – The image to be displayed.
- **win_name** (*str*) – The window name.
- **wait_time** (*int*) – Value of waitKey param.

21.2.2 mmcv.visualization.imshow_bboxes

mmcv.visualization.**imshow_bboxes**(*img: Union[str, numpy.ndarray]*, *bboxes: Union[list, numpy.ndarray]*, *colors: Union[*mmcv.visualization.color.Color*, str, tuple, int, numpy.ndarray] = 'green'*, *top_k: int = -1*, *thickness: int = 1*, *show: bool = True*, *win_name: str = ''*, *wait_time: int = 0*, *out_file: Optional[str] = None*)

Draw bboxes on an image.

Parameters

- **img** (*str or ndarray*) – The image to be displayed.
- **bboxes** (*list or ndarray*) – A list of ndarray of shape (k, 4).
- **colors** (*Color or str or tuple or int or ndarray*) – A list of colors.
- **top_k** (*int*) – Plot the first k bboxes only if set positive.
- **thickness** (*int*) – Thickness of lines.
- **show** (*bool*) – Whether to show the image.
- **win_name** (*str*) – The window name.
- **wait_time** (*int*) – Value of waitKey param.
- **out_file** (*str, optional*) – The filename to write the image.

Returns The image with bboxes drawn on it.

Return type ndarray

21.2.3 mmcv.visualization.imshow_det_bboxes

mmcv.visualization.**imshow_det_bboxes**(*img: Union[str, numpy.ndarray]*, *bboxes: numpy.ndarray*, *labels: numpy.ndarray*, *class_names: Optional[List[str]] = None*, *score_thr: float = 0*, *bbox_color: Union[*mmcv.visualization.color.Color*, str, tuple, int, numpy.ndarray] = 'green'*, *text_color: Union[*mmcv.visualization.color.Color*, str, tuple, int, numpy.ndarray] = 'green'*, *thickness: int = 1*, *font_scale: float = 0.5*, *show: bool = True*, *win_name: str = ''*, *wait_time: int = 0*, *out_file: Optional[str] = None*)

Draw bboxes and class labels (with scores) on an image.

Parameters

- **img** (*str or ndarray*) – The image to be displayed.
- **bboxes** (*ndarray*) – Bounding boxes (with scores), shaped (n, 4) or (n, 5).
- **labels** (*ndarray*) – Labels of bboxes.
- **class_names** (*list[str]*) – Names of each classes.
- **score_thr** (*float*) – Minimum score of bboxes to be shown.
- **bbox_color** (*Color or str or tuple or int or ndarray*) – Color of bbox lines.
- **text_color** (*Color or str or tuple or int or ndarray*) – Color of texts.
- **thickness** (*int*) – Thickness of lines.
- **font_scale** (*float*) – Font scales of texts.
- **show** (*bool*) – Whether to show the image.
- **win_name** (*str*) – The window name.
- **wait_time** (*int*) – Value of waitKey param.
- **out_file** (*str or None*) – The filename to write the image.

Returns The image with bboxes drawn on it.

Return type ndarray

21.3 Optical Flow

flow2rgb	Convert flow map to RGB image.
flowshow	Show optical flow.
make_color_wheel	Build a color wheel.

21.3.1 mmcv.visualization.flow2rgb

mmcv.visualization.**flow2rgb**(*flow: numpy.ndarray*, *color_wheel: Optional[numpy.ndarray] = None*, *unknown_thr: float = 1000000.0*) → numpy.ndarray

Convert flow map to RGB image.

Parameters

- **flow** (*ndarray*) – Array of optical flow.
- **color_wheel** (*ndarray or None*) – Color wheel used to map flow field to RGB colorspace. Default color wheel will be used if not specified.
- **unknown_thr** (*float*) – Values above this threshold will be marked as unknown and thus ignored.

Returns RGB image that can be visualized.

Return type ndarray

21.3.2 mmcv.visualization.flowshow

mmcv.visualization.**flowshow**(*flow: Union[numpy.ndarray, str]*, *win_name: str = ''*, *wait_time: int = 0*) → None

Show optical flow.

Parameters

- **flow** (*ndarray or str*) – The optical flow to be displayed.
- **win_name** (*str*) – The window name.
- **wait_time** (*int*) – Value of waitKey param.

21.3.3 mmcv.visualization.make_color_wheel

mmcv.visualization.**make_color_wheel**(*bins: Optional[Union[list, tuple]] = None*) → numpy.ndarray

Build a color wheel.

Parameters **bins** (*list or tuple, optional*) – Specify the number of bins for each color range, corresponding to six ranges: red -> yellow, yellow -> green, green -> cyan, cyan -> blue, blue -> magenta, magenta -> red. [15, 6, 4, 11, 13, 6] is used for default (see Middlebury).

Returns Color wheel of shape (total_bins, 3).

Return type ndarray

CHAPTER

TWENTYTWO

MMCV.CNN

mmcv.cnn

22.1 Module

ContextBlock	ContextBlock module in GCNet.
Conv2d	
Conv3d	
ConvAWS2d	AWS (Adaptive Weight Standardization)
ConvModule	A conv block that bundles conv/norm/activation layers.
ConvTranspose2d	
ConvTranspose3d	
ConvWS2d	
DepthwiseSeparableConvModule	Depthwise separable convolution module.
GeneralizedAttention	GeneralizedAttention module.
HSigmoid	Hard Sigmoid Module.
HSwish	Hard Swish Module.
Linear	
MaxPool2d	
MaxPool3d	
NonLocal1d	1D Non-local module.
NonLocal2d	2D Non-local module.
NonLocal3d	3D Non-local module.

continues on next page

Table 1 – continued from previous page

Scale	A learnable scale parameter.
Swish	Swish Module.
Conv2dRFSearchOp	Enable Conv2d with receptive field searching ability.

22.1.1 ContextBlock

class mmcv.cnn.**ContextBlock**(*in_channels: int*, *ratio: float*, *pooling_type: str = 'att'*, *fusion_types: tuple = ('channel_add')*)

ContextBlock module in GCNet.

See 'GCNet: Non-local Networks Meet Squeeze-Excitation Networks and Beyond' (https://arxiv.org/abs/1904.11492) for details.

Parameters

- **in_channels** (*int*) – Channels of the input feature map.
- **ratio** (*float*) – Ratio of channels of transform bottleneck
- **pooling_type** (*str*) – Pooling method for context modeling. Options are 'att' and 'avg', stand for attention pooling and average pooling respectively. Default: 'att'.
- **fusion_types** (*Sequence[str]*) – Fusion method for feature fusion, Options are 'channels_add', 'channel_mul', stand for channelwise addition and multiplication respectively. Default: ('channel_add',)

forward(*x: torch.Tensor*) → torch.Tensor

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.

22.1.2 Conv2d

class mmcv.cnn.**Conv2d**(*in_channels: int*, *out_channels: int*, *kernel_size: Union[int, Tuple[int, int]]*, *stride: Union[int, Tuple[int, int]] = 1*, *padding: Union[str, int, Tuple[int, int]] = 0*, *dilation: Union[int, Tuple[int, int]] = 1*, *groups: int = 1*, *bias: bool = True*, *padding_mode: str = 'zeros'*, *device=None*, *dtype=None*)

forward(*x: torch.Tensor*) → torch.Tensor

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.

22.1.3 Conv3d

class mmcv.cnn.**Conv3d**(*in_channels: int*, *out_channels: int*, *kernel_size: Union[int, Tuple[int, int, int]]*, *stride: Union[int, Tuple[int, int, int]] = 1*, *padding: Union[str, int, Tuple[int, int, int]] = 0*, *dilation: Union[int, Tuple[int, int, int]] = 1*, *groups: int = 1*, *bias: bool = True*, *padding_mode: str = 'zeros'*, *device=None*, *dtype=None*)

forward(*x: torch.Tensor*) → torch.Tensor

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.

22.1.4 ConvAWS2d

class mmcv.cnn.**ConvAWS2d**(*in_channels: int*, *out_channels: int*, *kernel_size: Union[int, Tuple[int, int]]*, *stride: Union[int, Tuple[int, int]] = 1*, *padding: Union[int, Tuple[int, int]] = 0*, *dilation: Union[int, Tuple[int, int]] = 1*, *groups: int = 1*, *bias: bool = True*)

AWS (Adaptive Weight Standardization)

This is a variant of Weight Standardization (https://arxiv.org/pdf/1903.10520.pdf) It is used in DetectoRS to avoid NaN (https://arxiv.org/pdf/2006.02334.pdf)

Parameters

- **in_channels** (*int*) – Number of channels in the input image
- **out_channels** (*int*) – Number of channels produced by the convolution
- **kernel_size** (*int or tuple*) – Size of the conv kernel
- **stride** (*int or tuple, optional*) – Stride of the convolution. Default: 1
- **padding** (*int or tuple, optional*) – Zero-padding added to both sides of the input. Default: 0
- **dilation** (*int or tuple, optional*) – Spacing between kernel elements. Default: 1
- **groups** (*int, optional*) – Number of blocked connections from input channels to output channels. Default: 1
- **bias** (*bool, optional*) – If set True, adds a learnable bias to the output. Default: True

forward(*x: torch.Tensor*) → torch.Tensor

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.

22.1.5 ConvModule

class mmcv.cnn.**ConvModule**(*in_channels: int*, *out_channels: int*, *kernel_size: Union[int, Tuple[int, int]]*, *stride: Union[int, Tuple[int, int]] = 1*, *padding: Union[int, Tuple[int, int]] = 0*, *dilation: Union[int, Tuple[int, int]] = 1*, *groups: int = 1*, *bias: Union[bool, str] = 'auto'*, *conv_cfg: Optional[Dict] = None*, *norm_cfg: Optional[Dict] = None*, *act_cfg: Optional[Dict] = {'type': 'ReLU'}*, *inplace: bool = True*, *with_spectral_norm: bool = False*, *padding_mode: str = 'zeros'*, *order: tuple = ('conv', 'norm', 'act')*, *efficient_conv_bn_eval: bool = False*)

A conv block that bundles conv/norm/activation layers.

This block simplifies the usage of convolution layers, which are commonly used with a norm layer (e.g., BatchNorm) and activation layer (e.g., ReLU). It is based upon three build methods: *build_conv_layer()*, *build_norm_layer()* and *build_activation_layer()*.

Besides, we add some additional features in this module. 1. Automatically set *bias* of the conv layer. 2. Spectral norm is supported. 3. More padding modes are supported. Before PyTorch 1.5, nn.Conv2d only supports zero and circular padding, and we add "reflect" padding mode.

Parameters

- **in_channels** (*int*) – Number of channels in the input feature map. Same as that in nn._ConvNd.
- **out_channels** (*int*) – Number of channels produced by the convolution. Same as that in nn._ConvNd.
- **kernel_size** (*int | tuple[int]*) – Size of the convolving kernel. Same as that in nn._ConvNd.
- **stride** (*int | tuple[int]*) – Stride of the convolution. Same as that in nn._ConvNd.
- **padding** (*int | tuple[int]*) – Zero-padding added to both sides of the input. Same as that in nn._ConvNd.
- **dilation** (*int | tuple[int]*) – Spacing between kernel elements. Same as that in nn._ConvNd.
- **groups** (*int*) – Number of blocked connections from input channels to output channels. Same as that in nn._ConvNd.
- **bias** (*bool | str*) – If specified as *auto*, it will be decided by the norm_cfg. Bias will be set as True if *norm_cfg* is None, otherwise False. Default: "auto".
- **conv_cfg** (*dict*) – Config dict for convolution layer. Default: None, which means using conv2d.
- **norm_cfg** (*dict*) – Config dict for normalization layer. Default: None.
- **act_cfg** (*dict*) – Config dict for activation layer. Default: dict(type='ReLU').
- **inplace** (*bool*) – Whether to use inplace mode for activation. Default: True.
- **with_spectral_norm** (*bool*) – Whether use spectral norm in conv module. Default: False.
- **padding_mode** (*str*) – If the *padding_mode* has not been supported by current *Conv2d* in PyTorch, we will use our own padding layer instead. Currently, we support ['zeros', 'circular'] with official implementation and ['reflect'] with our own implementation. Default: 'zeros'.
- **order** (*tuple[str]*) – The order of conv/norm/activation layers. It is a sequence of "conv", "norm" and "act". Common examples are ("conv", "norm", "act") and ("act", "conv", "norm"). Default: ('conv', 'norm', 'act').

- **efficient_conv_bn_eval** (*bool*) – Whether use efficient conv when the consecutive bn is in eval mode (either training or testing), as proposed in https://arxiv.org/abs/2305.11624 . Default: *False*.

static create_from_conv_bn(*conv: torch.nn.modules.conv._ConvNd*, *bn: torch.nn.modules.batchnorm._BatchNorm*, *efficient_conv_bn_eval=True*) → *mmcv.cnn.bricks.conv_module.ConvModule*

Create a ConvModule from a conv and a bn module.

forward(*x: torch.Tensor*, *activate: bool = True*, *norm: bool = True*) → torch.Tensor

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.

22.1.6 ConvTranspose2d

class mmcv.cnn.**ConvTranspose2d**(*in_channels: int*, *out_channels: int*, *kernel_size: Union[int, Tuple[int, int]]*, *stride: Union[int, Tuple[int, int]] = 1*, *padding: Union[int, Tuple[int, int]] = 0*, *output_padding: Union[int, Tuple[int, int]] = 0*, *groups: int = 1*, *bias: bool = True*, *dilation: Union[int, Tuple[int, int]] = 1*, *padding_mode: str = 'zeros'*, *device=None*, *dtype=None*)

forward(*x: torch.Tensor*) → torch.Tensor

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.

22.1.7 ConvTranspose3d

class mmcv.cnn.**ConvTranspose3d**(*in_channels: int*, *out_channels: int*, *kernel_size: Union[int, Tuple[int, int, int]]*, *stride: Union[int, Tuple[int, int, int]] = 1*, *padding: Union[int, Tuple[int, int, int]] = 0*, *output_padding: Union[int, Tuple[int, int, int]] = 0*, *groups: int = 1*, *bias: bool = True*, *dilation: Union[int, Tuple[int, int, int]] = 1*, *padding_mode: str = 'zeros'*, *device=None*, *dtype=None*)

forward(*x: torch.Tensor*) → torch.Tensor

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.

22.1.8 ConvWS2d

class mmcv.cnn.**ConvWS2d**(*in_channels: int*, *out_channels: int*, *kernel_size: Union[int, Tuple[int, int]]*, *stride: Union[int, Tuple[int, int]] = 1*, *padding: Union[int, Tuple[int, int]] = 0*, *dilation: Union[int, Tuple[int, int]] = 1*, *groups: int = 1*, *bias: bool = True*, *eps: float = 1e-05*)

forward(*x: torch.Tensor*) → torch.Tensor

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.

22.1.9 DepthwiseSeparableConvModule

class mmcv.cnn.**DepthwiseSeparableConvModule**(*in_channels: int*, *out_channels: int*, *kernel_size: Union[int, Tuple[int, int]]*, *stride: Union[int, Tuple[int, int]] = 1*, *padding: Union[int, Tuple[int, int]] = 0*, *dilation: Union[int, Tuple[int, int]] = 1*, *norm_cfg: Optional[Dict] = None*, *act_cfg: Dict = {'type': 'ReLU'}*, *dw_norm_cfg: Union[Dict, str] = 'default'*, *dw_act_cfg: Union[Dict, str] = 'default'*, *pw_norm_cfg: Union[Dict, str] = 'default'*, *pw_act_cfg: Union[Dict, str] = 'default'*, ***kwargs*)

Depthwise separable convolution module.

See https://arxiv.org/pdf/1704.04861.pdf for details.

This module can replace a ConvModule with the conv block replaced by two conv block: depthwise conv block and pointwise conv block. The depthwise conv block contains depthwise-conv/norm/activation layers. The pointwise conv block contains pointwise-conv/norm/activation layers. It should be noted that there will be norm/activation layer in the depthwise conv block if *norm_cfg* and *act_cfg* are specified.

Parameters

- **in_channels** (*int*) – Number of channels in the input feature map. Same as that in `nn._ConvNd`.
- **out_channels** (*int*) – Number of channels produced by the convolution. Same as that in `nn._ConvNd`.
- **kernel_size** (*int | tuple[int]*) – Size of the convolving kernel. Same as that in `nn._ConvNd`.
- **stride** (*int | tuple[int]*) – Stride of the convolution. Same as that in `nn._ConvNd`. Default: 1.
- **padding** (*int | tuple[int]*) – Zero-padding added to both sides of the input. Same as that in `nn._ConvNd`. Default: 0.

- **dilation** (*int | tuple[int]*) – Spacing between kernel elements. Same as that in nn._ConvNd. Default: 1.
- **norm_cfg** (*dict*) – Default norm config for both depthwise ConvModule and pointwise ConvModule. Default: None.
- **act_cfg** (*dict*) – Default activation config for both depthwise ConvModule and pointwise ConvModule. Default: dict(type='ReLU').
- **dw_norm_cfg** (*dict*) – Norm config of depthwise ConvModule. If it is 'default', it will be the same as *norm_cfg*. Default: 'default'.
- **dw_act_cfg** (*dict*) – Activation config of depthwise ConvModule. If it is 'default', it will be the same as *act_cfg*. Default: 'default'.
- **pw_norm_cfg** (*dict*) – Norm config of pointwise ConvModule. If it is 'default', it will be the same as *norm_cfg*. Default: 'default'.
- **pw_act_cfg** (*dict*) – Activation config of pointwise ConvModule. If it is 'default', it will be the same as *act_cfg*. Default: 'default'.
- **kwargs** (*optional*) – Other shared arguments for depthwise and pointwise ConvModule. See ConvModule for ref.

forward(*x: torch.Tensor*) → torch.Tensor

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.

22.1.10 GeneralizedAttention

class mmcv.cnn.**GeneralizedAttention**(*in_channels: int*, *spatial_range: int = - 1*, *num_heads: int = 9*, *position_embedding_dim: int = - 1*, *position_magnitude: int = 1*, *kv_stride: int = 2*, *q_stride: int = 1*, *attention_type: str = '1111'*)

GeneralizedAttention module.

See 'An Empirical Study of Spatial Attention Mechanisms in Deep Networks' (https://arxiv.org/abs/1904.05873) for details.

Parameters

- **in_channels** (*int*) – Channels of the input feature map.
- **spatial_range** (*int*) – The spatial range. -1 indicates no spatial range constraint. Default: -1.
- **num_heads** (*int*) – The head number of empirical_attention module. Default: 9.
- **position_embedding_dim** (*int*) – The position embedding dimension. Default: -1.
- **position_magnitude** (*int*) – A multiplier acting on coord difference. Default: 1.
- **kv_stride** (*int*) – The feature stride acting on key/value feature map. Default: 2.
- **q_stride** (*int*) – The feature stride acting on query feature map. Default: 1.

- **attention_type** (*str*) – A binary indicator string for indicating which items in generalized empirical_attention module are used. Default: '1111'.
  - '1000' indicates 'query and key content' (appr - appr) item,
  - '0100' indicates 'query content and relative position' (appr - position) item,
  - '0010' indicates 'key content only' (bias - appr) item,
  - '0001' indicates 'relative position only' (bias - position) item.

forward(*x_input: torch.Tensor*) → torch.Tensor
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.

22.1.11 HSigmoid

class mmcv.cnn.**HSigmoid**(*bias: float = 3.0*, *divisor: float = 6.0*, *min_value: float = 0.0*, *max_value: float = 1.0*)
Hard Sigmoid Module. Apply the hard sigmoid function: Hsigmoid(x) = min(max((x + bias) / divisor, min_value), max_value) Default: Hsigmoid(x) = min(max((x + 3) / 6, 0), 1)

Note: In MMCV v1.4.4, we modified the default value of args to align with PyTorch official.

Parameters

- **bias** (*float*) – Bias of the input feature map. Default: 3.0.
- **divisor** (*float*) – Divisor of the input feature map. Default: 6.0.
- **min_value** (*float*) – Lower bound value. Default: 0.0.
- **max_value** (*float*) – Upper bound value. Default: 1.0.

Returns The output tensor.

Return type Tensor

forward(*x: torch.Tensor*) → torch.Tensor
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.

22.1.12 HSwish

class mmcv.cnn.**HSwish**(*inplace: bool = False*)

Hard Swish Module.

This module applies the hard swish function:

$$Hswish(x) = x * ReLU6(x + 3)/6$$

Parameters **inplace** (*bool*) – can optionally do the operation in-place. Default: False.

Returns The output tensor.

Return type Tensor

forward(*x: torch.Tensor*) → torch.Tensor

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.

22.1.13 Linear

class mmcv.cnn.**Linear**(*in_features: int*, *out_features: int*, *bias: bool = True*, *device=None*, *dtype=None*)

forward(*x: torch.Tensor*) → torch.Tensor

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.

22.1.14 MaxPool2d

class mmcv.cnn.**MaxPool2d**(*kernel_size: Union[int, Tuple[int, ...]]*, *stride: Optional[Union[int, Tuple[int, ...]]] = None*, *padding: Union[int, Tuple[int, ...]] = 0*, *dilation: Union[int, Tuple[int, ...]] = 1*, *return_indices: bool = False*, *ceil_mode: bool = False*)

forward(*x: torch.Tensor*) → torch.Tensor

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.

22.1.15 MaxPool3d

class mmcv.cnn.**MaxPool3d**(*kernel_size: Union[int, Tuple[int, ...]]*, *stride: Optional[Union[int, Tuple[int, ...]]] = None*, *padding: Union[int, Tuple[int, ...]] = 0*, *dilation: Union[int, Tuple[int, ...]] = 1*, *return_indices: bool = False*, *ceil_mode: bool = False*)

forward(*x: torch.Tensor*) → torch.Tensor

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.

22.1.16 NonLocal1d

class mmcv.cnn.**NonLocal1d**(*in_channels: int*, *sub_sample: bool = False*, *conv_cfg: Dict = {'type': 'Conv1d'}*, ***kwargs*)

1D Non-local module.

Parameters

- **in_channels** (*int*) – Same as *NonLocalND*.
- **sub_sample** (*bool*) – Whether to apply max pooling after pairwise function (Note that the *sub_sample* is applied on spatial only). Default: False.
- **conv_cfg** (*None | dict*) – Same as *NonLocalND*. Default: dict(type='Conv1d').

22.1.17 NonLocal2d

class mmcv.cnn.**NonLocal2d**(*in_channels: int*, *sub_sample: bool = False*, *conv_cfg: Dict = {'type': 'Conv2d'}*, ***kwargs*)

2D Non-local module.

Parameters

- **in_channels** (*int*) – Same as *NonLocalND*.
- **sub_sample** (*bool*) – Whether to apply max pooling after pairwise function (Note that the *sub_sample* is applied on spatial only). Default: False.
- **conv_cfg** (*None | dict*) – Same as *NonLocalND*. Default: dict(type='Conv2d').

22.1.18 NonLocal3d

class mmcv.cnn.**NonLocal3d**(*in_channels: int*, *sub_sample: bool = False*, *conv_cfg: Dict = {'type': 'Conv3d'}*, ***kwargs*)

3D Non-local module.

Parameters

- **in_channels** (*int*) – Same as *NonLocalND*.
- **sub_sample** (*bool*) – Whether to apply max pooling after pairwise function (Note that the *sub_sample* is applied on spatial only). Default: False.
- **conv_cfg** (*None | dict*) – Same as *NonLocalND*. Default: dict(type='Conv3d').

22.1.19 Scale

class mmcv.cnn.**Scale**(*scale: float = 1.0*)

A learnable scale parameter.

This layer scales the input by a learnable factor. It multiplies a learnable scale parameter of shape (1,) with input of any shape.

Parameters **scale** (*float*) – Initial value of scale factor. Default: 1.0

forward(*x: torch.Tensor*) → torch.Tensor

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.

22.1.20 Swish

class mmcv.cnn.**Swish**

Swish Module.

This module applies the swish function:

$$Swish(x) = x * Sigmoid(x)$$

Returns The output tensor.

Return type Tensor

forward(*x: torch.Tensor*) → torch.Tensor

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.

22.1.21 Conv2dRFSearchOp

class mmcv.cnn.**Conv2dRFSearchOp**(*op_layer: torch.nn.modules.module.Module*, *global_config: dict*, *verbose: bool = True*)

Enable Conv2d with receptive field searching ability.

Parameters

- **op_layer** (*nn.Module*) – pytorch module, e,g, Conv2d
- **global_config** (*dict*) – config dict. Defaults to None. By default this must include:
  - "init_alphas": The value for initializing weights of each branch.
  - "num_branches": The controller of the size of search space (the number of branches).
  - "exp_rate": The controller of the sparsity of search space.
  - "mmin": The minimum dilation rate.
  - "mmax": The maximum dilation rate.

  Extra keys may exist, but are used by RFSearchHook, e.g., "step", "max_step", "search_interval", and "skip_layer".
- **verbose** (*bool*) – Determines whether to print rf-next related logging messages. Defaults to True.

estimate_rates() → None
Estimate new dilation rate based on trained branch_weights.

expand_rates() → None
Expand dilation rate.

forward(*input: torch.Tensor*) → torch.Tensor
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.

22.2 Build Function

build_activation_layer	Build activation layer.
build_conv_layer	Build convolution layer.
build_norm_layer	Build normalization layer.
build_padding_layer	Build padding layer.
build_plugin_layer	Build plugin layer.
build_upsample_layer	Build upsample layer.

22.2.1 mmcv.cnn.build_activation_layer

mmcv.cnn.**build_activation_layer**(*cfg: Dict*) → torch.nn.modules.module.Module
Build activation layer.

Parameters **cfg** (*dict*) – The activation layer config, which should contain:

- type (str): Layer type.
- layer args: Args needed to instantiate an activation layer.

Returns Created activation layer.

Return type nn.Module

22.2.2 mmcv.cnn.build_conv_layer

mmcv.cnn.**build_conv_layer**(*cfg: Optional[Dict]*, **args*, ***kwargs*) → torch.nn.modules.module.Module
Build convolution layer.

Parameters

- **cfg** (*None or dict*) – The conv layer config, which should contain: - type (str): Layer type. - layer args: Args needed to instantiate an conv layer.
- **args** (*argument list*) – Arguments passed to the *__init__* method of the corresponding conv layer.
- **kwargs** (*keyword arguments*) – Keyword arguments passed to the *__init__* method of the corresponding conv layer.

Returns Created conv layer.

Return type nn.Module

22.2.3 mmcv.cnn.build_norm_layer

mmcv.cnn.**build_norm_layer**(*cfg: Dict*, *num_features: int*, *postfix: Union[int, str] = ''*) → Tuple[str, torch.nn.modules.module.Module]
Build normalization layer.

Parameters

- **cfg** (*dict*) – The norm layer config, which should contain:
  - type (str): Layer type.
  - layer args: Args needed to instantiate a norm layer.
  - requires_grad (bool, optional): Whether stop gradient updates.
- **num_features** (*int*) – Number of input channels.
- **postfix** (*int | str*) – The postfix to be appended into norm abbreviation to create named layer.

Returns The first element is the layer name consisting of abbreviation and postfix, e.g., bn1, gn. The second element is the created norm layer.

Return type tuple[str, nn.Module]

22.2.4 mmcv.cnn.build_padding_layer

mmcv.cnn.**build_padding_layer**(*cfg: Dict*, **args*, ***kwargs*) → torch.nn.modules.module.Module
Build padding layer.

Parameters **cfg** (*dict*) – The padding layer config, which should contain: - type (str): Layer type. - layer args: Args needed to instantiate a padding layer.

Returns Created padding layer.

Return type nn.Module

22.2.5 mmcv.cnn.build_plugin_layer

mmcv.cnn.**build_plugin_layer**(*cfg: Dict*, *postfix: Union[int, str] = ''*, ***kwargs*) → Tuple[str, torch.nn.modules.module.Module]
Build plugin layer.

Parameters

- **cfg** (*dict*) – cfg should contain:
  - type (str): identify plugin layer type.
  - layer args: args needed to instantiate a plugin layer.
- **postfix** (*int, str*) – appended into norm abbreviation to create named layer. Default: ‘’.

Returns The first one is the concatenation of abbreviation and postfix. The second is the created plugin layer.

Return type tuple[str, nn.Module]

22.2.6 mmcv.cnn.build_upsample_layer

mmcv.cnn.**build_upsample_layer**(*cfg: Dict*, **args*, ***kwargs*) → torch.nn.modules.module.Module
Build upsample layer.

Parameters

- **cfg** (*dict*) – The upsample layer config, which should contain:
  - type (str): Layer type.
  - scale_factor (int): Upsample ratio, which is not applicable to deconv.
  - layer args: Args needed to instantiate a upsample layer.
- **args** (*argument list*) – Arguments passed to the `__init__` method of the corresponding conv layer.
- **kwargs** (*keyword arguments*) – Keyword arguments passed to the `__init__` method of the corresponding conv layer.

Returns Created upsample layer.

Return type nn.Module

22.3 Miscellaneous

fuse_conv_bn	Recursively fuse conv and bn in a module.
conv_ws_2d	
is_norm	Check if a layer is a normalization layer.
make_res_layer	
make_vgg_layer	
get_model_complexity_info	Get complexity information of a model.

22.3.1 mmcv.cnn.fuse_conv_bn

mmcv.cnn.**fuse_conv_bn**(*module: torch.nn.modules.module.Module*) → torch.nn.modules.module.Module

Recursively fuse conv and bn in a module.

During inference, the functionary of batch norm layers is turned off but only the mean and var alone channels are used, which exposes the chance to fuse it with the preceding conv layers to save computations and simplify network structures.

Parameters **module** (*nn.Module*) – Module to be fused.

Returns Fused module.

Return type nn.Module

22.3.2 mmcv.cnn.conv_ws_2d

mmcv.cnn.**conv_ws_2d**(*input: torch.Tensor*, *weight: torch.Tensor*, *bias: Optional[torch.Tensor] = None*, *stride: Union[int, Tuple[int, int]] = 1*, *padding: Union[int, Tuple[int, int]] = 0*, *dilation: Union[int, Tuple[int, int]] = 1*, *groups: int = 1*, *eps: float = 1e-05*) → torch.Tensor

22.3.3 mmcv.cnn.is_norm

mmcv.cnn.**is_norm**(*layer: torch.nn.modules.module.Module*, *exclude: Optional[Union[type, tuple]] = None*) → bool

Check if a layer is a normalization layer.

Parameters

- **layer** (*nn.Module*) – The layer to be checked.
- **exclude** (*type | tuple[type]*) – Types to be excluded.

Returns Whether the layer is a norm layer.

Return type bool

22.3.4 mmcv.cnn.make_res_layer

mmcv.cnn.**make_res_layer**(*block: torch.nn.modules.module.Module*, *inplanes: int*, *planes: int*, *blocks: int*, *stride: int = 1*, *dilation: int = 1*, *style: str = 'pytorch'*, *with_cp: bool = False*) → torch.nn.modules.module.Module

22.3.5 mmcv.cnn.make_vgg_layer

mmcv.cnn.**make_vgg_layer**(*inplanes: int*, *planes: int*, *num_blocks: int*, *dilation: int = 1*, *with_bn: bool = False*, *ceil_mode: bool = False*) → List[torch.nn.modules.module.Module]

22.3.6 mmcv.cnn.get_model_complexity_info

mmcv.cnn.**get_model_complexity_info**(*model: torch.nn.modules.module.Module*, *input_shape: tuple*, *print_per_layer_stat: bool = True*, *as_strings: bool = True*, *input_constructor: Optional[Callable] = None*, *flush: bool = False*, *ost: TextIO = <_io.TextIOWrapper name='<stdout>' mode='w' encoding='UTF-8'>*) → tuple

Get complexity information of a model.

This method can calculate FLOPs and parameter counts of a model with corresponding input shape. It can also print complexity information for each layer in a model.

Supported layers are listed as below:

- Convolutions: `nn.Conv1d`, `nn.Conv2d`, `nn.Conv3d`.
- Activations: `nn.ReLU`, `nn.PReLU`, `nn.ELU`, `nn.LeakyReLU`, `nn.ReLU6`.
- Poolings: `nn.MaxPool1d`, `nn.MaxPool2d`, `nn.MaxPool3d`, `nn.AvgPool1d`, `nn.AvgPool2d`, `nn.AvgPool3d`, `nn.AdaptiveMaxPool1d`, `nn.AdaptiveMaxPool2d`, `nn.AdaptiveMaxPool3d`, `nn.AdaptiveAvgPool1d`, `nn.AdaptiveAvgPool2d`, `nn.AdaptiveAvgPool3d`.
- BatchNorms: `nn.BatchNorm1d`, `nn.BatchNorm2d`, `nn.BatchNorm3d`, `nn.GroupNorm`, `nn.InstanceNorm1d`, `InstanceNorm2d`, `InstanceNorm3d`, `nn.LayerNorm`.
- Linear: `nn.Linear`.
- Deconvolution: `nn.ConvTranspose2d`.
- Upsample: `nn.Upsample`.

Parameters

- **model** (*nn.Module*) – The model for complexity calculation.
- **input_shape** (*tuple*) – Input shape used for calculation.
- **print_per_layer_stat** (*bool*) – Whether to print complexity information for each layer in a model. Default: True.
- **as_strings** (*bool*) – Output FLOPs and params counts in a string form. Default: True.
- **input_constructor** (*None | callable*) – If specified, it takes a callable method that generates input. otherwise, it will generate a random tensor with input shape to calculate FLOPs. Default: None.
- **flush** (*bool*) – same as that in `print()`. Default: False.
- **ost** (*stream*) – same as `file` param in `print()`. Default: sys.stdout.

Returns If `as_strings` is set to True, it will return FLOPs and parameter counts in a string format. otherwise, it will return those in a float number format.

Return type tuple[float | str]

CHAPTER

TWENTYTHREE

MMCV.OPS

BorderAlign	Border align pooling layer.
CARAFE	CARAFE: Content-Aware ReAssembly of FEatures
CARAFENaive	
CARAFEPack	A unified package of CARAFE upsampler that contains: 1) channel compressor 2) content encoder 3) CARAFE op.
Conv2d	alias of `mmcv.ops.deprecated_wrappers.Conv2d_deprecated`
ConvTranspose2d	alias of `mmcv.ops.deprecated_wrappers.ConvTranspose2d_deprecated`
CornerPool	Corner Pooling.
Correlation	Correlation operator
CrissCrossAttention	Criss-Cross Attention Module.
DeformConv2d	Deformable 2D convolution.
DeformConv2dPack	A Deformable Conv Encapsulation that acts as normal Conv layers.
DeformRoIPool	
DeformRoIPoolPack	
DynamicScatter	Scatters points into voxels, used in the voxel encoder with dynamic voxelization.
FusedBiasLeakyReLU	Fused bias leaky ReLU.
GroupAll	Group xyz with feature.
Linear	alias of `mmcv.ops.deprecated_wrappers.Linear_deprecated`
MaskedConv2d	A MaskedConv2d which inherits the official Conv2d.
MaxPool2d	alias of `mmcv.ops.deprecated_wrappers.MaxPool2d_deprecated`
ModulatedDeformConv2d	
ModulatedDeformConv2dPack	A ModulatedDeformable Conv Encapsulation that acts as normal Conv layers.
ModulatedDeformRoIPoolPack	
MultiScaleDeformableAttention	An attention module used in Deformable-Detr.

continues on next page

Table 1 – continued from previous page

PSAMask	
PointsSampler	Points sampling.
PrRoIPool	The operation of precision RoI pooling.
QueryAndGroup	Groups points with a ball query of radius.
RiRoIAlignRotated	Rotation-invariant RoI align pooling layer for rotated proposals.
RoIAlign	RoI align pooling layer.
RoIAlignRotated	RoI align pooling layer for rotated proposals.
RoIAwarePool3d	Encode the geometry-specific features of each 3D proposal.
RoIPointPool3d	Encode the geometry-specific features of each 3D proposal.
RoIPool	
SAConv2d	SAC (Switchable Atrous Convolution)
SigmoidFocalLoss	
SimpleRoIAlign	
SoftmaxFocalLoss	
SparseConv2d	
SparseConv3d	
SparseConvTensor	
SparseConvTranspose2d	
SparseConvTranspose3d	
SparseInverseConv2d	
SparseInverseConv3d	
SparseMaxPool2d	
SparseMaxPool3d	
SparseModule	place holder, All module subclass from this will take sptensor in SparseSequential.
SparseSequential	A sequential container.
SubMConv2d	
SubMConv3d	
SyncBatchNorm	Synchronized Batch Normalization.
TINShift	Temporal Interlace Shift.
Voxelization	Convert kitti points(N, >=3) to voxels.

23.1 BorderAlign

class mmcv.ops.**BorderAlign**(*pool_size: int*)

Border align pooling layer.

Applies border_align over the input feature based on predicted bboxes. The details were described in the paper BorderDet: Border Feature for Dense Object Detection.

For each border line (e.g. top, left, bottom or right) of each box, border_align does the following:

1. uniformly samples `pool_size` +1 positions on this line, involving the start and end points.
2. the corresponding features on these points are computed by bilinear interpolation.
3. max pooling over all the `pool_size` +1 positions are used for computing pooled feature.

Parameters **pool_size** (*int*) – number of positions sampled over the boxes' borders (e.g. top, bottom, left, right).

forward(*input: torch.Tensor*, *boxes: torch.Tensor*) → torch.Tensor

Parameters

- **input** – Features with shape [N,4C,H,W]. Channels ranged in [0,C), [C,2C), [2C,3C), [3C,4C) represent the top, left, bottom, right features respectively.
- **boxes** – Boxes with shape [N,H*W,4]. Coordinate format (x1,y1,x2,y2).

Returns Pooled features with shape [N,C,H*W,4]. The order is (top,left,bottom,right) for the last dimension.

Return type torch.Tensor

23.2 CARAFE

class mmcv.ops.**CARAFE**(*kernel_size: int*, *group_size: int*, *scale_factor: int*)

CARAFE: Content-Aware ReAssembly of FEatures

Please refer to CARAFE: Content-Aware ReAssembly of FEatures for more details.

Parameters

- **kernel_size** (*int*) – reassemble kernel size
- **group_size** (*int*) – reassemble group size
- **scale_factor** (*int*) – upsample ratio

Returns upsampled feature map

forward(*features: torch.Tensor*, *masks: torch.Tensor*) → torch.Tensor

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.

23.3 CARAFENaive

class mmcv.ops.**CARAFENaive**(*kernel_size: int*, *group_size: int*, *scale_factor: int*)

forward(*features: torch.Tensor*, *masks: torch.Tensor*) → torch.Tensor
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.

23.4 CARAFEPack

class mmcv.ops.**CARAFEPack**(*channels: int*, *scale_factor: int*, *up_kernel: int = 5*, *up_group: int = 1*, *encoder_kernel: int = 3*, *encoder_dilation: int = 1*, *compressed_channels: int = 64*)

A unified package of CARAFE upsampler that contains: 1) channel compressor 2) content encoder 3) CARAFE op.

Official implementation of ICCV 2019 paper CARAFE: Content-Aware ReAssembly of FEatures.

Parameters

- **channels** (*int*) – input feature channels
- **scale_factor** (*int*) – upsample ratio
- **up_kernel** (*int*) – kernel size of CARAFE op
- **up_group** (*int*) – group size of CARAFE op
- **encoder_kernel** (*int*) – kernel size of content encoder
- **encoder_dilation** (*int*) – dilation of content encoder
- **compressed_channels** (*int*) – output channels of channels compressor

Returns upsampled feature map

forward(*x: torch.Tensor*) → torch.Tensor
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.

23.5 Conv2d

mmcv.ops.**Conv2d**
alias of mmcv.ops.deprecated_wrappers.Conv2d_deprecated

23.6 ConvTranspose2d

mmcv.ops.**ConvTranspose2d**
alias of mmcv.ops.deprecated_wrappers.ConvTranspose2d_deprecated

23.7 CornerPool

class mmcv.ops.**CornerPool**(*mode: str*)
Corner Pooling.

Corner Pooling is a new type of pooling layer that helps a convolutional network better localize corners of bounding boxes.

Please refer to CornerNet: Detecting Objects as Paired Keypoints for more details.

Code is modified from https://github.com/princeton-vl/CornerNet-Lite.

Parameters **mode** (*str*) – Pooling orientation for the pooling layer

- 'bottom': Bottom Pooling
- 'left': Left Pooling
- 'right': Right Pooling
- 'top': Top Pooling

Returns Feature map after pooling.

forward(*x: torch.Tensor*) → torch.Tensor
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.

23.8 Correlation

class mmcv.ops.**Correlation**(*kernel_size: int = 1*, *max_displacement: int = 1*, *stride: int = 1*, *padding: int = 0*, *dilation: int = 1*, *dilation_patch: int = 1*)
Correlation operator

This correlation operator works for optical flow correlation computation.

There are two batched tensors with shape (N, C, H, W), and the correlation output's shape is $(N, max\_displacement \times 2 + 1, max\_displacement * 2 + 1, H_{out}, W_{out})$

where

$$H_{out} = \left\lfloor \frac{H_{in} + 2 \times padding - dilation \times (kernel\_size - 1) - 1}{stride} + 1 \right\rfloor$$

$$W_{out} = \left\lfloor \frac{W_{in} + 2 \times padding - dilation \times (kernel\_size - 1) - 1}{stride} + 1 \right\rfloor$$

the correlation item (N_i, dy, dx) is formed by taking the sliding window convolution between input1 and shifted input2,

$$Corr(N_i, dx, dy) = \sum_{c=0}^{C-1} input1(N_i, c) \star \mathcal{S}(input2(N_i, c), dy, dx)$$

where $\star$ is the valid 2d sliding window convolution operator, and $\mathcal{S}$ means shifting the input features (auto-complete zero marginal), and dx, dy are shifting distance, $dx, dy \in [-max\_displacement \times dilation\_patch, max\_displacement \times dilation\_patch]$.

Parameters

- **kernel_size** (*int*) – The size of sliding window i.e. local neighborhood representing the center points and involved in correlation computation. Defaults to 1.
- **max_displacement** (*int*) – The radius for computing correlation volume, but the actual working space can be dilated by dilation_patch. Defaults to 1.
- **stride** (*int*) – The stride of the sliding blocks in the input spatial dimensions. Defaults to 1.
- **padding** (*int*) – Zero padding added to all four sides of the input1. Defaults to 0.
- **dilation** (*int*) – The spacing of local neighborhood that will involved in correlation. Defaults to 1.
- **dilation_patch** (*int*) – The spacing between position need to compute correlation. Defaults to 1.

forward(*input1: torch.Tensor*, *input2: torch.Tensor*) → torch.Tensor

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.

23.9 CrissCrossAttention

class mmcv.ops.**CrissCrossAttention**(*in_channels: int*)

Criss-Cross Attention Module.

Note: Before v1.3.13, we use a CUDA op. Since v1.3.13, we switch to a pure PyTorch and equivalent implementation. For more details, please refer to https://github.com/open-mmlab/mmcv/pull/1201.

Speed comparison for one forward pass

- Input size: [2,512,97,97]

- Device: 1 NVIDIA GeForce RTX 2080 Ti

	PyTorch version	CUDA version	Relative speed
with torch.no_grad()	0.00554402 s	0.0299619 s	5.4x
no with torch.no_grad()	0.00562803 s	0.0301349 s	5.4x

Parameters **in_channels** (*int*) – Channels of the input feature map.

forward(*x: torch.Tensor*) → torch.Tensor
forward function of Criss-Cross Attention.

Parameters **x** (*torch.Tensor*) – Input feature with the shape of (batch_size, in_channels, height, width).

Returns Output of the layer, with the shape of (batch_size, in_channels, height, width)

Return type torch.Tensor

23.10 DeformConv2d

class mmcv.ops.**DeformConv2d**(*in_channels: int*, *out_channels: int*, *kernel_size: Union[int, Tuple[int, ...]]*, *stride: Union[int, Tuple[int, ...]] = 1*, *padding: Union[int, Tuple[int, ...]] = 0*, *dilation: Union[int, Tuple[int, ...]] = 1*, *groups: int = 1*, *deform_groups: int = 1*, *bias: bool = False*, *im2col_step: int = 32*)
Deformable 2D convolution.

Applies a deformable 2D convolution over an input signal composed of several input planes. DeformConv2d was described in the paper Deformable Convolutional Networks

---

Note: The argument `im2col_step` was added in version 1.3.17, which means number of samples processed by the `im2col_cuda_kernel` per call. It enables users to define `batch_size` and `im2col_step` more flexibly and solved issue mmcv#1440.

Parameters

- **in_channels** (*int*) – Number of channels in the input image.
- **out_channels** (*int*) – Number of channels produced by the convolution.
- **kernel_size** (*int, tuple*) – Size of the convolving kernel.
- **stride** (*int, tuple*) – Stride of the convolution. Default: 1.
- **padding** (*int or tuple*) – Zero-padding added to both sides of the input. Default: 0.
- **dilation** (*int or tuple*) – Spacing between kernel elements. Default: 1.
- **groups** (*int*) – Number of blocked connections from input. channels to output channels. Default: 1.
- **deform_groups** (*int*) – Number of deformable group partitions.
- **bias** (*bool*) – If True, adds a learnable bias to the output. Default: False.

- **im2col_step** (*int*) – Number of samples processed by im2col_cuda_kernel per call. It will work when `batch_size > im2col_step`, but `batch_size` must be divisible by `im2col_step`. Default: 32. *New in version 1.3.17.*

forward(*x: torch.Tensor*, *offset: torch.Tensor*) → torch.Tensor
Deformable Convolutional forward function.

Parameters

- **x** (*Tensor*) – Input feature, shape (B, C_in, H_in, W_in)
- **offset** (*Tensor*) – Offset for deformable convolution, shape (B, deform_groups*kernel_size[0]*kernel_size[1]*2, H_out, W_out), H_out, W_out are equal to the output's.

  An offset is like *[y0, x0, y1, x1, y2, x2, …, y8, x8]*. The spatial arrangement is like:

```
(x0, y0) (x1, y1) (x2, y2)
(x3, y3) (x4, y4) (x5, y5)
(x6, y6) (x7, y7) (x8, y8)
```

Returns Output of the layer.

Return type Tensor

23.11 DeformConv2dPack

class mmcv.ops.**DeformConv2dPack**(**args*, ***kwargs*)
A Deformable Conv Encapsulation that acts as normal Conv layers.

The offset tensor is like *[y0, x0, y1, x1, y2, x2, …, y8, x8]*. The spatial arrangement is like:

```
(x0, y0) (x1, y1) (x2, y2)
(x3, y3) (x4, y4) (x5, y5)
(x6, y6) (x7, y7) (x8, y8)
```

Parameters

- **in_channels** (*int*) – Same as nn.Conv2d.
- **out_channels** (*int*) – Same as nn.Conv2d.
- **kernel_size** (*int or tuple[int]*) – Same as nn.Conv2d.
- **stride** (*int or tuple[int]*) – Same as nn.Conv2d.
- **padding** (*int or tuple[int]*) – Same as nn.Conv2d.
- **dilation** (*int or tuple[int]*) – Same as nn.Conv2d.
- **groups** (*int*) – Same as nn.Conv2d.
- **bias** (*bool or str*) – If specified as *auto*, it will be decided by the norm_cfg. Bias will be set as True if norm_cfg is None, otherwise False.

forward(*x: torch.Tensor*) → torch.Tensor
Deformable Convolutional forward function.

Parameters

- **x** (*Tensor*) – Input feature, shape (B, C_in, H_in, W_in)
- **offset** (*Tensor*) – Offset for deformable convolution, shape (B, deform_groups*kernel_size[0]*kernel_size[1]*2, H_out, W_out), H_out, W_out are equal to the output's.

  An offset is like *[y0, x0, y1, x1, y2, x2, ..., y8, x8]*. The spatial arrangement is like:

```
(x0, y0) (x1, y1) (x2, y2)
(x3, y3) (x4, y4) (x5, y5)
(x6, y6) (x7, y7) (x8, y8)
```

Returns Output of the layer.

Return type Tensor

23.12 DeformRoIPool

class mmcv.ops.**DeformRoIPool**(*output_size: Tuple[int, ...]*, *spatial_scale: float = 1.0*, *sampling_ratio: int = 0*, *gamma: float = 0.1*)

forward(*input: torch.Tensor*, *rois: torch.Tensor*, *offset: Optional[torch.Tensor] = None*) → torch.Tensor

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.

23.13 DeformRoIPoolPack

class mmcv.ops.**DeformRoIPoolPack**(*output_size: Tuple[int, ...]*, *output_channels: int*, *deform_fc_channels: int = 1024*, *spatial_scale: float = 1.0*, *sampling_ratio: int = 0*, *gamma: float = 0.1*)

forward(*input: torch.Tensor*, *rois: torch.Tensor*) → torch.Tensor

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.

23.14 DynamicScatter

class mmcv.ops.**DynamicScatter**(*voxel_size: List*, *point_cloud_range: List*, *average_points: bool*)
Scatters points into voxels, used in the voxel encoder with dynamic voxelization.

Note: The CPU and GPU implementation get the same output, but have numerical difference after summation and division (e.g., 5e-7).

Parameters

- **voxel_size** (*list*) – list [x, y, z] size of three dimension.
- **point_cloud_range** (*list*) – The coordinate range of points, [x_min, y_min, z_min, x_max, y_max, z_max].
- **average_points** (*bool*) – whether to use avg pooling to scatter points into voxel.

forward(*points: torch.Tensor*, *coors: torch.Tensor*) → Tuple[torch.Tensor, torch.Tensor]
Scatters points/features into voxels.

Parameters

- **points** (*torch.Tensor*) – Points to be reduced into voxels.
- **coors** (*torch.Tensor*) – Corresponding voxel coordinates (specifically multi-dim voxel index) of each points.

Returns A tuple contains two elements. The first one is the voxel features with shape [M, C] which are respectively reduced from input features that share the same voxel coordinates. The second is voxel coordinates with shape [M, ndim].

Return type tuple[torch.Tensor]

forward_single(*points: torch.Tensor*, *coors: torch.Tensor*) → Tuple[torch.Tensor, torch.Tensor]
Scatters points into voxels.

Parameters

- **points** (*torch.Tensor*) – Points to be reduced into voxels.
- **coors** (*torch.Tensor*) – Corresponding voxel coordinates (specifically multi-dim voxel index) of each points.

Returns A tuple contains two elements. The first one is the voxel features with shape [M, C] which are respectively reduced from input features that share the same voxel coordinates. The second is voxel coordinates with shape [M, ndim].

Return type tuple[torch.Tensor]

23.15 FusedBiasLeakyReLU

class mmcv.ops.**FusedBiasLeakyReLU**(*num_channels: int*, *negative_slope: float = 0.2*, *scale: float = 1.4142135623730951*)

Fused bias leaky ReLU.

This function is introduced in the StyleGAN2: Analyzing and Improving the Image Quality of StyleGAN

The bias term comes from the convolution operation. In addition, to keep the variance of the feature map or gradients unchanged, they also adopt a scale similarly with Kaiming initialization. However, since the $1+alpha^2$ is too small, we can just ignore it. Therefore, the final scale is just $\sqrt{2}$. Of course, you may change it with your own scale.

TODO: Implement the CPU version.

Parameters

- **num_channels** (*int*) – The channel number of the feature map.
- **negative_slope** (*float, optional*) – Same as nn.LeakyRelu. Defaults to 0.2.
- **scale** (*float, optional*) – A scalar to adjust the variance of the feature map. Defaults to 2**0.5.

forward(*input: torch.Tensor*) → torch.Tensor

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.

23.16 GroupAll

class mmcv.ops.**GroupAll**(*use_xyz: bool = True*)

Group xyz with feature.

Parameters **use_xyz** (*bool*) – Whether to use xyz.

forward(*xyz: torch.Tensor*, *new_xyz: torch.Tensor*, *features: Optional[torch.Tensor] = None*) → torch.Tensor

Parameters

- **xyz** (*Tensor*) – (B, N, 3) xyz coordinates of the features.
- **new_xyz** (*Tensor*) – new xyz coordinates of the features.
- **features** (*Tensor*) – (B, C, N) features to group.

Returns (B, C + 3, 1, N) Grouped feature.

Return type Tensor

23.17 Linear

mmcv.ops.**Linear**
alias of mmcv.ops.deprecated_wrappers.Linear_deprecated

23.18 MaskedConv2d

class mmcv.ops.**MaskedConv2d**(*in_channels: int*, *out_channels: int*, *kernel_size: Union[int, Tuple[int, ...]]*, *stride: int = 1*, *padding: int = 0*, *dilation: int = 1*, *groups: int = 1*, *bias: bool = True*)

A MaskedConv2d which inherits the official Conv2d.

The masked forward doesn't implement the backward function and only supports the stride parameter to be 1 currently.

forward(*input: torch.Tensor*, *mask: Optional[torch.Tensor] = None*) → torch.Tensor

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.

23.19 MaxPool2d

mmcv.ops.**MaxPool2d**
alias of mmcv.ops.deprecated_wrappers.MaxPool2d_deprecated

23.20 ModulatedDeformConv2d

class mmcv.ops.**ModulatedDeformConv2d**(*in_channels: int*, *out_channels: int*, *kernel_size: Union[int, Tuple[int]]*, *stride: int = 1*, *padding: int = 0*, *dilation: int = 1*, *groups: int = 1*, *deform_groups: int = 1*, *bias: Union[bool, str] = True*)

forward(*x: torch.Tensor*, *offset: torch.Tensor*, *mask: torch.Tensor*) → torch.Tensor

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.

23.21 ModulatedDeformConv2dPack

class mmcv.ops.**ModulatedDeformConv2dPack**(**args*, ***kwargs*)

A ModulatedDeformable Conv Encapsulation that acts as normal Conv layers.

Parameters

- **in_channels** (*int*) – Same as nn.Conv2d.
- **out_channels** (*int*) – Same as nn.Conv2d.
- **kernel_size** (*int or tuple[int]*) – Same as nn.Conv2d.
- **stride** (*int*) – Same as nn.Conv2d, while tuple is not supported.
- **padding** (*int*) – Same as nn.Conv2d, while tuple is not supported.
- **dilation** (*int*) – Same as nn.Conv2d, while tuple is not supported.
- **groups** (*int*) – Same as nn.Conv2d.
- **bias** (*bool or str*) – If specified as *auto*, it will be decided by the norm_cfg. Bias will be set as True if norm_cfg is None, otherwise False.

forward(*x: torch.Tensor*) → torch.Tensor

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.

23.22 ModulatedDeformRoIPoolPack

class mmcv.ops.**ModulatedDeformRoIPoolPack**(*output_size: Tuple[int, ...]*, *output_channels: int*, *deform_fc_channels: int = 1024*, *spatial_scale: float = 1.0*, *sampling_ratio: int = 0*, *gamma: float = 0.1*)

forward(*input: torch.Tensor*, *rois: torch.Tensor*) → torch.Tensor

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.

23.23 MultiScaleDeformableAttention

class mmcv.ops.**MultiScaleDeformableAttention**(*embed_dims: int = 256*, *num_heads: int = 8*, *num_levels: int = 4*, *num_points: int = 4*, *im2col_step: int = 64*, *dropout: float = 0.1*, *batch_first: bool = False*, *norm_cfg: Optional[dict] = None*, *init_cfg: Optional[mmengine.config.config.ConfigDict] = None*, *value_proj_ratio: float = 1.0*)

An attention module used in Deformable-Detr.

Deformable DETR: Deformable Transformers for End-to-End Object Detection..

Parameters

- **embed_dims** (*int*) – The embedding dimension of Attention. Default: 256.
- **num_heads** (*int*) – Parallel attention heads. Default: 8.
- **num_levels** (*int*) – The number of feature map used in Attention. Default: 4.
- **num_points** (*int*) – The number of sampling points for each query in each head. Default: 4.
- **im2col_step** (*int*) – The step used in image_to_column. Default: 64.
- **dropout** (*float*) – A Dropout layer on *inp_identity*. Default: 0.1.
- **batch_first** (*bool*) – Key, Query and Value are shape of (batch, n, embed_dim) or (n, batch, embed_dim). Default to False.
- **norm_cfg** (*dict*) – Config dict for normalization layer. Default: None.
- **(obj** (*init_cfg*) – *mmcv.ConfigDict*): The Config for initialization. Default: None.
- **value_proj_ratio** (*float*) – The expansion ratio of value_proj. Default: 1.0.

forward(*query: torch.Tensor*, *key: Optional[torch.Tensor] = None*, *value: Optional[torch.Tensor] = None*, *identity: Optional[torch.Tensor] = None*, *query_pos: Optional[torch.Tensor] = None*, *key_padding_mask: Optional[torch.Tensor] = None*, *reference_points: Optional[torch.Tensor] = None*, *spatial_shapes: Optional[torch.Tensor] = None*, *level_start_index: Optional[torch.Tensor] = None*, ***kwargs*) → torch.Tensor

Forward Function of MultiScaleDeformAttention.

Parameters

- **query** (*torch.Tensor*) – Query of Transformer with shape (num_query, bs, embed_dims).
- **key** (*torch.Tensor*) – The key tensor with shape *(num_key, bs, embed_dims)*.
- **value** (*torch.Tensor*) – The value tensor with shape *(num_key, bs, embed_dims)*.
- **identity** (*torch.Tensor*) – The tensor used for addition, with the same shape as *query*. Default None. If None, *query* will be used.
- **query_pos** (*torch.Tensor*) – The positional encoding for *query*. Default: None.
- **key_padding_mask** (*torch.Tensor*) – ByteTensor for *query*, with shape [bs, num_key].
- **reference_points** (*torch.Tensor*) – The normalized reference points with shape (bs, num_query, num_levels, 2), all elements is range in [0, 1], top-left (0,0), bottom-right (1, 1), including padding area. or (N, Length_{query}, num_levels, 4), add additional two dimensions is (w, h) to form reference boxes.

- **spatial_shapes** (*torch.Tensor*) – Spatial shape of features in different levels. With shape (num_levels, 2), last dimension represents (h, w).
- **level_start_index** (*torch.Tensor*) – The start index of each level. A tensor has shape `(num_levels, )` and can be represented as [0, h_0*w_0, h_0*w_0+h_1*w_1, ...].

Returns forwarded results with shape [num_query, bs, embed_dims].

Return type torch.Tensor

init_weights() → None
Default initialization for Parameters of Module.

23.24 PSAMask

class mmcv.ops.**PSAMask**(*psa_type: str*, *mask_size: Optional[tuple] = None*)

forward(*input: torch.Tensor*) → torch.Tensor
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.

23.25 PointsSampler

class mmcv.ops.**PointsSampler**(*num_point: List[int]*, *fps_mod_list: List[str] = ['D-FPS']*, *fps_sample_range_list: List[int] = [- 1]*)
Points sampling.

Parameters

- **num_point** (*list[int]*) – Number of sample points.
- **fps_mod_list** (*list[str], optional*) – Type of FPS method, valid mod ['F-FPS', 'D-FPS', 'FS'], Default: ['D-FPS']. F-FPS: using feature distances for FPS. D-FPS: using Euclidean distances of points for FPS. FS: using F-FPS and D-FPS simultaneously.
- **fps_sample_range_list** (*list[int], optional*) – Range of points to apply FPS. Default: [-1].

forward(*points_xyz: torch.Tensor*, *features: torch.Tensor*) → torch.Tensor

Parameters

- **points_xyz** (*torch.Tensor*) – (B, N, 3) xyz coordinates of the points.
- **features** (*torch.Tensor*) – (B, C, N) features of the points.

Returns (B, npoint, sample_num) Indices of sampled points.

Return type torch.Tensor

23.26 PrRoIPool

class mmcv.ops.PrRoIPool(*output_size: Union[int, tuple]*, *spatial_scale: float = 1.0*)

The operation of precision RoI pooling. The implementation of PrRoIPool is modified from https://github.com/vacancy/PreciseRoIPooling/

Precise RoI Pooling (PrRoIPool) is an integration-based (bilinear interpolation) average pooling method for RoI Pooling. It avoids any quantization and has a continuous gradient on bounding box coordinates. It is:

1. different from the original RoI Pooling proposed in Fast R-CNN. PrRoI Pooling uses average pooling instead of max pooling for each bin and has a continuous gradient on bounding box coordinates. That is, one can take the derivatives of some loss function w.r.t the coordinates of each RoI and optimize the RoI coordinates. 2. different from the RoI Align proposed in Mask R-CNN. PrRoI Pooling uses a full integration-based average pooling instead of sampling a constant number of points. This makes the gradient w.r.t. the coordinates continuous.

Parameters

- **output_size** (*Union[int, tuple]*) – h, w.
- **spatial_scale** (*float, optional*) – scale the input boxes by this number. Defaults to 1.0.

forward(*features: torch.Tensor*, *rois: torch.Tensor*) → torch.Tensor

Forward function.

Parameters

- **features** (*torch.Tensor*) – The feature map.
- **rois** (*torch.Tensor*) – The RoI bboxes in [tl_x, tl_y, br_x, br_y] format.

Returns The pooled results.

Return type torch.Tensor

23.27 QueryAndGroup

class mmcv.ops.QueryAndGroup(*max_radius: float*, *sample_num: int*, *min_radius: float = 0.0*, *use_xyz: bool = True*, *return_grouped_xyz: bool = False*, *normalize_xyz: bool = False*, *uniform_sample: bool = False*, *return_unique_cnt: bool = False*, *return_grouped_idx: bool = False*)

Groups points with a ball query of radius.

Parameters

- **max_radius** (*float*) – The maximum radius of the balls. If None is given, we will use kNN sampling instead of ball query.
- **sample_num** (*int*) – Maximum number of features to gather in the ball.
- **min_radius** (*float, optional*) – The minimum radius of the balls. Default: 0.
- **use_xyz** (*bool, optional*) – Whether to use xyz. Default: True.
- **return_grouped_xyz** (*bool, optional*) – Whether to return grouped xyz. Default: False.
- **normalize_xyz** (*bool, optional*) – Whether to normalize xyz. Default: False.
- **uniform_sample** (*bool, optional*) – Whether to sample uniformly. Default: False

- **return_unique_cnt** (*bool, optional*) – Whether to return the count of unique samples. Default: False.
- **return_grouped_idx** (*bool, optional*) – Whether to return grouped idx. Default: False.

forward(*points_xyz: torch.Tensor*, *center_xyz: torch.Tensor*, *features: Optional[torch.Tensor] = None*) → Union[torch.Tensor, Tuple]

Parameters

- **points_xyz** (*torch.Tensor*) – (B, N, 3) xyz coordinates of the points.
- **center_xyz** (*torch.Tensor*) – (B, npoint, 3) coordinates of the centriods.
- **features** (*torch.Tensor*) – (B, C, N) The features of grouped points.

Returns (B, 3 + C, npoint, sample_num) Grouped concatenated coordinates and features of points.

Return type Tuple | torch.Tensor

23.28 RiRoIAlignRotated

class mmcv.ops.**RiRoIAlignRotated**(*out_size: tuple*, *spatial_scale: float*, *num_samples: int = 0*, *num_orientations: int = 8*, *clockwise: bool = False*)

Rotation-invariant RoI align pooling layer for rotated proposals.

It accepts a feature map of shape (N, C, H, W) and rois with shape (n, 6) with each roi decoded as (batch_index, center_x, center_y, w, h, angle). The angle is in radian.

The details are described in the paper ReDet: A Rotation-equivariant Detector for Aerial Object Detection.

Parameters

- **out_size** (*tuple*) – fixed dimensional RoI output with shape (h, w).
- **spatial_scale** (*float*) – scale the input boxes by this number
- **num_samples** (*int*) – number of inputs samples to take for each output sample. 0 to take samples densely for current models.
- **num_orientations** (*int*) – number of oriented channels.
- **clockwise** (*bool*) – If True, the angle in each proposal follows a clockwise fashion in image space, otherwise, the angle is counterclockwise. Default: False.

forward(*features: torch.Tensor*, *rois: torch.Tensor*) → torch.Tensor

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.

23.29 RoIAlign

class mmcv.ops.**RoIAlign**(*output_size: tuple*, *spatial_scale: float = 1.0*, *sampling_ratio: int = 0*, *pool_mode: str = 'avg'*, *aligned: bool = True*, *use_torchvision: bool = False*)

RoI align pooling layer.

Parameters

- **output_size** (*tuple*) – h, w
- **spatial_scale** (*float*) – scale the input boxes by this number
- **sampling_ratio** (*int*) – number of inputs samples to take for each output sample. 0 to take samples densely for current models.
- **pool_mode** (*str, 'avg' or 'max'*) – pooling mode in each bin.
- **aligned** (*bool*) – if False, use the legacy implementation in MMDetection. If True, align the results more perfectly.
- **use_torchvision** (*bool*) – whether to use roi_align from torchvision.

Note: The implementation of RoIAlign when aligned=True is modified from https://github.com/facebookresearch/detectron2/

The meaning of aligned=True:

Given a continuous coordinate c, its two neighboring pixel indices (in our pixel model) are computed by floor(c - 0.5) and ceil(c - 0.5). For example, c=1.3 has pixel neighbors with discrete indices [0] and [1] (which are sampled from the underlying signal at continuous coordinates 0.5 and 1.5). But the original roi_align (aligned=False) does not subtract the 0.5 when computing neighboring pixel indices and therefore it uses pixels with a slightly incorrect alignment (relative to our pixel model) when performing bilinear interpolation.

With *aligned=True*, we first appropriately scale the ROI and then shift it by -0.5 prior to calling roi_align. This produces the correct neighbors;

The difference does not make a difference to the model's performance if ROIAlign is used together with conv layers.

---

forward(*input: torch.Tensor*, *rois: torch.Tensor*) → torch.Tensor

Parameters

- **input** – NCHW images
- **rois** – Bx5 boxes. First column is the index into N. The other 4 columns are xyxy.

23.30 RoIAlignRotated

class mmcv.ops.**RoIAlignRotated**(*output_size: Union[int, tuple]*, *spatial_scale: float*, *sampling_ratio: int = 0*, *aligned: bool = True*, *clockwise: bool = False*)

RoI align pooling layer for rotated proposals.

It accepts a feature map of shape (N, C, H, W) and rois with shape (n, 6) with each roi decoded as (batch_index, center_x, center_y, w, h, angle). The angle is in radian.

Parameters

- **output_size** (*tuple*) – h, w
- **spatial_scale** (*float*) – scale the input boxes by this number
- **sampling_ratio** (*int*) – number of inputs samples to take for each output sample. 0 to take samples densely for current models.
- **aligned** (*bool*) – if False, use the legacy implementation in MMDetection. If True, align the results more perfectly. Default: True.
- **clockwise** (*bool*) – If True, the angle in each proposal follows a clockwise fashion in image space, otherwise, the angle is counterclockwise. Default: False.

Note: The implementation of RoIAlign when aligned=True is modified from https://github.com/facebookresearch/detectron2/

The meaning of aligned=True:

Given a continuous coordinate c, its two neighboring pixel indices (in our pixel model) are computed by floor(c - 0.5) and ceil(c - 0.5). For example, c=1.3 has pixel neighbors with discrete indices [0] and [1] (which are sampled from the underlying signal at continuous coordinates 0.5 and 1.5). But the original roi_align (aligned=False) does not subtract the 0.5 when computing neighboring pixel indices and therefore it uses pixels with a slightly incorrect alignment (relative to our pixel model) when performing bilinear interpolation.

With *aligned=True*, we first appropriately scale the ROI and then shift it by -0.5 prior to calling roi_align. This produces the correct neighbors;

The difference does not make a difference to the model's performance if ROIAlign is used together with conv layers.

---

forward(*input: torch.Tensor*, *rois: torch.Tensor*) → torch.Tensor

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.

23.31 RoIAwarePool3d

class mmcv.ops.**RoIAwarePool3d**(*out_size: Union[int, tuple]*, *max_pts_per_voxel: int = 128*, *mode: str = 'max'*)

Encode the geometry-specific features of each 3D proposal.

Please refer to PartA2 for more details.

Parameters

- **out_size** (*int or tuple*) – The size of output features. n or [n1, n2, n3].
- **max_pts_per_voxel** (*int, optional*) – The maximum number of points per voxel. Default: 128.
- **mode** (*str, optional*) – Pooling method of RoIAware, 'max' or 'avg'. Default: 'max'.

forward(*rois: torch.Tensor*, *pts: torch.Tensor*, *pts_feature: torch.Tensor*) → torch.Tensor

Parameters

- **rois** (*torch.Tensor*) – [N, 7], in LiDAR coordinate, (x, y, z) is the bottom center of rois.
- **pts** (*torch.Tensor*) – [npoints, 3], coordinates of input points.
- **pts_feature** (*torch.Tensor*) – [npoints, C], features of input points.

Returns Pooled features whose shape is [N, out_x, out_y, out_z, C].

Return type torch.Tensor

23.32 RoIPointPool3d

class mmcv.ops.**RoIPointPool3d**(*num_sampled_points: int = 512*)

Encode the geometry-specific features of each 3D proposal.

Please refer to Paper of PartA2 for more details.

Parameters **num_sampled_points** (*int, optional*) – Number of samples in each roi. Default: 512.

forward(*points: torch.Tensor*, *point_features: torch.Tensor*, *boxes3d: torch.Tensor*) → Tuple[torch.Tensor]

Parameters

- **points** (*torch.Tensor*) – Input points whose shape is (B, N, C).
- **point_features** (*torch.Tensor*) – Features of input points whose shape is (B, N, C).
- **boxes3d** (*B, M, 7), Input bounding boxes whose shape is (B, M, 7*) –

Returns A tuple contains two elements. The first one is the pooled features whose shape is (B, M, 512, 3 + C). The second is an empty flag whose shape is (B, M).

Return type tuple[torch.Tensor]

23.33 RoIPool

class mmcv.ops.**RoIPool**(*output_size: Union[int, tuple]*, *spatial_scale: float = 1.0*)

forward(*input: torch.Tensor*, *rois: torch.Tensor*) → torch.Tensor

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.

23.34 SAConv2d

class mmcv.ops.**SAConv2d**(*in_channels*, *out_channels*, *kernel_size*, *stride=1*, *padding=0*, *dilation=1*, *groups=1*, *bias=True*, *use_deform=False*)

SAC (Switchable Atrous Convolution)

This is an implementation of DetectoRS: Detecting Objects with Recursive Feature Pyramid and Switchable Atrous Convolution.

Parameters

- **in_channels** (*int*) – Number of channels in the input image
- **out_channels** (*int*) – Number of channels produced by the convolution
- **kernel_size** (*int or tuple*) – Size of the convolving kernel
- **stride** (*int or tuple, optional*) – Stride of the convolution. Default: 1
- **padding** (*int or tuple, optional*) – Zero-padding added to both sides of the input. Default: 0
- **padding_mode** (*string, optional*) – `'zeros'`, `'reflect'`, `'replicate'` or `'circular'`. Default: `'zeros'`
- **dilation** (*int or tuple, optional*) – Spacing between kernel elements. Default: 1
- **groups** (*int, optional*) – Number of blocked connections from input channels to output channels. Default: 1
- **bias** (*bool, optional*) – If `True`, adds a learnable bias to the output. Default: `True`
- **use_deform** – If `True`, replace convolution with deformable convolution. Default: `False`.

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.

23.35 SigmoidFocalLoss

class mmcv.ops.**SigmoidFocalLoss**(*gamma: float*, *alpha: float*, *weight: Optional[torch.Tensor] = None*, *reduction: str = 'mean'*)

forward(*input: torch.Tensor*, *target: Union[torch.LongTensor, torch.cuda.LongTensor]*) → torch.Tensor

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.

23.36 SimpleRoIAlign

class mmcv.ops.**SimpleRoIAlign**(*output_size: Tuple[int]*, *spatial_scale: float*, *aligned: bool = True*)

forward(*features: torch.Tensor*, *rois: torch.Tensor*) → torch.Tensor
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.

23.37 SoftmaxFocalLoss

class mmcv.ops.**SoftmaxFocalLoss**(*gamma: float*, *alpha: float*, *weight: Optional[torch.Tensor] = None*, *reduction: str = 'mean'*)

forward(*input: torch.Tensor*, *target: Union[torch.LongTensor, torch.cuda.LongTensor]*) → torch.Tensor
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.

23.38 SparseConv2d

class mmcv.ops.**SparseConv2d**(*in_channels*, *out_channels*, *kernel_size*, *stride=1*, *padding=0*, *dilation=1*, *groups=1*, *bias=True*, *indice_key=None*)

23.39 SparseConv3d

class mmcv.ops.**SparseConv3d**(*in_channels*, *out_channels*, *kernel_size*, *stride=1*, *padding=0*, *dilation=1*, *groups=1*, *bias=True*, *indice_key=None*)

23.40 SparseConvTensor

class mmcv.ops.**SparseConvTensor**(*features: torch.Tensor*, *indices: torch.Tensor*, *spatial_shape: Union[List, Tuple]*, *batch_size: int*, *grid: Optional[torch.Tensor] = None*)

23.41 SparseConvTranspose2d

class mmcv.ops.**SparseConvTranspose2d**(*in_channels*, *out_channels*, *kernel_size*, *stride=1*, *padding=0*, *dilation=1*, *groups=1*, *bias=True*, *indice_key=None*)

23.42 SparseConvTranspose3d

class mmcv.ops.**SparseConvTranspose3d**(*in_channels*, *out_channels*, *kernel_size*, *stride=1*, *padding=0*, *dilation=1*, *groups=1*, *bias=True*, *indice_key=None*)

23.43 SparseInverseConv2d

class mmcv.ops.**SparseInverseConv2d**(*in_channels*, *out_channels*, *kernel_size*, *indice_key=None*, *bias=True*)

23.44 SparseInverseConv3d

class mmcv.ops.**SparseInverseConv3d**(*in_channels*, *out_channels*, *kernel_size*, *indice_key=None*, *bias=True*)

23.45 SparseMaxPool2d

class mmcv.ops.**SparseMaxPool2d**(*kernel_size*, *stride=1*, *padding=0*, *dilation=1*)

23.46 SparseMaxPool3d

class mmcv.ops.**SparseMaxPool3d**(*kernel_size*, *stride=1*, *padding=0*, *dilation=1*)

23.47 SparseModule

class mmcv.ops.**SparseModule**

place holder, All module subclass from this will take sptensor in SparseSequential.

23.48 SparseSequential

class mmcv.ops.**SparseSequential**(**args*, ***kwargs*)

A sequential container. Modules will be added to it in the order they are passed in the constructor. Alternatively, an ordered dict of modules can also be passed in.

To make it easier to understand, given is a small example:

```
.. rubric:: Example
```

```
>>> # using Sequential:
>>> from mmcv.ops import SparseSequential
>>> model = SparseSequential(
            SparseConv2d(1,20,5),
            nn.ReLU(),
            SparseConv2d(20,64,5),
            nn.ReLU()
            )
```

```
>>> # using Sequential with OrderedDict
>>> model = SparseSequential(OrderedDict([
              ('conv1', SparseConv2d(1,20,5)),
              ('relu1', nn.ReLU()),
              ('conv2', SparseConv2d(20,64,5)),
              ('relu2', nn.ReLU())
            ]))
```

```
>>> # using Sequential with kwargs(python 3.6+)
>>> model = SparseSequential(
              conv1=SparseConv2d(1,20,5),
              relu1=nn.ReLU(),
              conv2=SparseConv2d(20,64,5),
              relu2=nn.ReLU()
            )
```

forward(*input: torch.Tensor*) → torch.Tensor

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.

23.49 SubMConv2d

class mmcv.ops.**SubMConv2d**(*in_channels*, *out_channels*, *kernel_size*, *stride=1*, *padding=0*, *dilation=1*, *groups=1*, *bias=True*, *indice_key=None*)

23.50 SubMConv3d

class mmcv.ops.**SubMConv3d**(*in_channels*, *out_channels*, *kernel_size*, *stride=1*, *padding=0*, *dilation=1*, *groups=1*, *bias=True*, *indice_key=None*)

23.51 SyncBatchNorm

class mmcv.ops.**SyncBatchNorm**(*num_features: int*, *eps: float = 1e-05*, *momentum: float = 0.1*, *affine: bool = True*, *track_running_stats: bool = True*, *group: Optional[int] = None*, *stats_mode: str = 'default'*)

Synchronized Batch Normalization.

Parameters

- **num_features** (*int*) – number of features/chennels in input tensor
- **eps** (*float, optional*) – a value added to the denominator for numerical stability. Defaults to 1e-5.
- **momentum** (*float, optional*) – the value used for the running_mean and running_var computation. Defaults to 0.1.
- **affine** (*bool, optional*) – whether to use learnable affine parameters. Defaults to True.
- **track_running_stats** (*bool, optional*) – whether to track the running mean and variance during training. When set to False, this module does not track such statistics, and initializes statistics buffers `running_mean` and `running_var` as `None`. When these buffers are `None`, this module always uses batch statistics in both training and eval modes. Defaults to True.
- **group** (*int, optional*) – synchronization of stats happen within each process group individually. By default it is synchronization across the whole world. Defaults to None.
- **stats_mode** (*str, optional*) – The statistical mode. Available options includes `'default'` and `'N'`. Defaults to 'default'. When `stats_mode=='default'`, it computes the overall statistics using those from each worker with equal weight, i.e., the statistics are synchronized and simply divied by `group`. This mode will produce inaccurate statistics when empty tensors occur. When `stats_mode=='N'`, it compute the overall statistics using the total number of batches in each worker ignoring the number of group, i.e., the statistics are synchronized and then divied by the total batch `N`. This mode is beneficial when empty tensors occur during training, as it average the total mean by the real number of batch.

forward(*input: torch.Tensor*) → torch.Tensor

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.

23.52 TINShift

class mmcv.ops.**TINShift**

Temporal Interlace Shift.

Temporal Interlace shift is a differentiable temporal-wise frame shifting which is proposed in "Temporal Interlacing Network"

Please refer to Temporal Interlacing Network for more details.

Code is modified from https://github.com/mit-han-lab/temporal-shift-module

forward(*input*, *shift*)

Perform temporal interlace shift.

Parameters

- **input** (*torch.Tensor*) – Feature map with shape [N, num_segments, C, H * W].
- **shift** (*torch.Tensor*) – Shift tensor with shape [N, num_segments].

Returns Feature map after temporal interlace shift.

23.53 Voxelization

class mmcv.ops.**Voxelization**(*voxel_size: List*, *point_cloud_range: List*, *max_num_points: int*, *max_voxels: Union[tuple, int] = 20000*, *deterministic: bool = True*)

Convert kitti points(N, >=3) to voxels.

Please refer to Point-Voxel CNN for Efficient 3D Deep Learning for more details.

Parameters

- **voxel_size** (*tuple or float*) – The size of voxel with the shape of [3].
- **point_cloud_range** (*tuple or float*) – The coordinate range of voxel with the shape of [6].
- **max_num_points** (*int*) – maximum points contained in a voxel. if max_points=-1, it means using dynamic_voxelize.
- **max_voxels** (*int, optional*) – maximum voxels this function create. for second, 20000 is a good choice. Users should shuffle points before call this function because max_voxels may drop points. Default: 20000.

forward(*input: torch.Tensor*) → torch.Tensor

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.

active_rotated_filter	
assign_score_withk	
ball_query	
batched_nms	Performs non-maximum suppression in a batched fashion.
bbox_overlaps	Calculate overlap between two set of bboxes.
border_align	
box_iou_rotated	Return intersection-over-union (Jaccard index) of boxes.
boxes_iou3d	Calculate boxes 3D IoU.
boxes_iou_bev	Calculate boxes IoU in the Bird's Eye View.
boxes_overlap_bev	Calculate boxes BEV overlap.
carafe	
carafe_naive	
chamfer_distance	
contour_expand	Expand kernel contours so that foreground pixels are assigned into instances.
convex_giou	Return generalized intersection-over-union (Jaccard index) between point sets and polygons.
convex_iou	Return intersection-over-union (Jaccard index) between point sets and polygons.
deform_conv2d	
deform_roi_pool	
diff_iou_rotated_2d	Calculate differentiable iou of rotated 2d boxes.
diff_iou_rotated_3d	Calculate differentiable iou of rotated 3d boxes.
dynamic_scatter	
furthest_point_sample	
furthest_point_sample_with_dist	
fused_bias_leakyrelu	Fused bias leaky ReLU function.
gather_points	
grouping_operation	
knn	
masked_conv2d	
min_area_polygons	Find the smallest polygons that surrounds all points in the point sets.

continues on next page

Table 2 – continued from previous page

modulated_deform_conv2d	
nms	Dispatch to either CPU or GPU NMS implementations.
nms3d	3D NMS function GPU implementation (for BEV boxes).
nms3d_normal	Normal 3D NMS function GPU implementation.
nms_bev	NMS function GPU implementation (for BEV boxes).
nms_match	Matched dets into different groups by NMS.
nms_normal_bev	Normal NMS function GPU implementation (for BEV boxes).
nms_rotated	Performs non-maximum suppression (NMS) on the rotated boxes according to their intersection-over-union (IoU).
pixel_group	Group pixels into text instances, which is widely used text detection methods.
point_sample	A wrapper around `grid_sample()` to support 3D point_coords tensors Unlike `torch.nn.functional.grid_sample()` it assumes point_coords to lie inside `[0, 1] x [0, 1]` square.
points_in_boxes_all	Find all boxes in which each point is (CUDA).
points_in_boxes_cpu	Find all boxes in which each point is (CPU).
points_in_boxes_part	Find the box in which each point is (CUDA).
points_in_polygons	Judging whether points are inside polygons, which is used in the ATSS assignment for the rotated boxes.
prroi_pool	
rel_roi_point_to_rel_img_point	Convert roi based relative point coordinates to image based absolute point coordinates.
riroi_align_rotated	
roi_align	
roi_align_rotated	
roi_pool	
rotated_feature_align	
scatter_nd	pytorch edition of tensorflow scatter_nd.
sigmoid_focal_loss	
soft_nms	Dispatch to only CPU Soft NMS implementations.
softmax_focal_loss	
three_interpolate	
three_nn	
tin_shift	

continues on next page

Table 2 – continued from previous page

upfirdn2d	Pad, upsample, filter, and downsample a batch of 2D images.
voxelization	

23.54 mmcv.ops.active_rotated_filter

mmcv.ops.**active_rotated_filter**()

23.55 mmcv.ops.assign_score_withk

mmcv.ops.**assign_score_withk**()

23.56 mmcv.ops.ball_query

mmcv.ops.**ball_query**()

23.57 mmcv.ops.batched_nms

mmcv.ops.**batched_nms**(*boxes: torch.Tensor*, *scores: torch.Tensor*, *idxs: torch.Tensor*, *nms_cfg: Optional[Dict]*, *class_agnostic: bool = False*) → Tuple[torch.Tensor, torch.Tensor]

Performs non-maximum suppression in a batched fashion.

Modified from torchvision/ops/boxes.py#L39. In order to perform NMS independently per class, we add an offset to all the boxes. The offset is dependent only on the class idx, and is large enough so that boxes from different classes do not overlap.

Note: In v1.4.1 and later, `batched_nms` supports skipping the NMS and returns sorted raw results when *nms_cfg* is None.

Parameters

- **boxes** (*torch.Tensor*) – boxes in shape (N, 4) or (N, 5).
- **scores** (*torch.Tensor*) – scores in shape (N,).
- **idxs** (*torch.Tensor*) – each index value correspond to a bbox cluster, and NMS will not be applied between elements of different idxs, shape (N,).
- **nms_cfg** (*dict | optional*) – Supports skipping the nms when *nms_cfg* is None, otherwise it should specify nms type and other parameters like *iou_thr*. Possible keys includes the following.
  - iou_threshold (float): IoU threshold used for NMS.

  - split_thr (float): threshold number of boxes. In some cases the number of boxes is large (e.g., 200k). To avoid OOM during training, the users could set *split_thr* to a small value. If the number of boxes is greater than the threshold, it will perform NMS on each group of boxes separately and sequentially. Defaults to 10000.

- **class_agnostic** (*bool*) – if true, nms is class agnostic, i.e. IoU thresholding happens over all boxes, regardless of the predicted class. Defaults to False.

Returns

kept dets and indice.

- boxes (Tensor): Bboxes with score after nms, has shape (num_bboxes, 5). last dimension 5 arrange as (x1, y1, x2, y2, score)
- keep (Tensor): The indices of remaining boxes in input boxes.

Return type tuple

23.58 mmcv.ops.bbox_overlaps

mmcv.ops.**bbox_overlaps**(*bboxes1: torch.Tensor*, *bboxes2: torch.Tensor*, *mode: str = 'iou'*, *aligned: bool = False*, *offset: int = 0*) → torch.Tensor

Calculate overlap between two set of bboxes.

If `aligned` is `False`, then calculate the ious between each bbox of bboxes1 and bboxes2, otherwise the ious between each aligned pair of bboxes1 and bboxes2.

Parameters

- **bboxes1** (*torch.Tensor*) – shape (m, 4) in <x1, y1, x2, y2> format or empty.
- **bboxes2** (*torch.Tensor*) – shape (n, 4) in <x1, y1, x2, y2> format or empty. If aligned is `True`, then m and n must be equal.
- **mode** (*str*) – "iou" (intersection over union) or iof (intersection over foreground).

Returns Return the ious betweens boxes. If `aligned` is `False`, the shape of ious is (m, n) else (m, 1).

Return type torch.Tensor

Example

```
>>> bboxes1 = torch.FloatTensor([
>>>     [0, 0, 10, 10],
>>>     [10, 10, 20, 20],
>>>     [32, 32, 38, 42],
>>> ])
>>> bboxes2 = torch.FloatTensor([
>>>     [0, 0, 10, 20],
>>>     [0, 10, 10, 19],
>>>     [10, 10, 20, 20],
>>> ])
>>> bbox_overlaps(bboxes1, bboxes2)
tensor([[0.5000, 0.0000, 0.0000],
```

(continues on next page)

(continued from previous page)

```
        [0.0000, 0.0000, 1.0000],
        [0.0000, 0.0000, 0.0000]])
```

Example

```
>>> empty = torch.FloatTensor([])
>>> nonempty = torch.FloatTensor([
>>>     [0, 0, 10, 9],
>>> ])
>>> assert tuple(bbox_overlaps(empty, nonempty).shape) == (0, 1)
>>> assert tuple(bbox_overlaps(nonempty, empty).shape) == (1, 0)
>>> assert tuple(bbox_overlaps(empty, empty).shape) == (0, 0)
```

23.59 mmcv.ops.border_align

mmcv.ops.**border_align**()

23.60 mmcv.ops.box_iou_rotated

mmcv.ops.**box_iou_rotated**(*bboxes1: torch.Tensor*, *bboxes2: torch.Tensor*, *mode: str = 'iou'*, *aligned: bool = False*, *clockwise: bool = True*) → torch.Tensor

Return intersection-over-union (Jaccard index) of boxes.

Both sets of boxes are expected to be in (x_center, y_center, width, height, angle) format.

If `aligned` is `False`, then calculate the ious between each bbox of bboxes1 and bboxes2, otherwise the ious between each aligned pair of bboxes1 and bboxes2.

Note: The operator assumes:

1) The positive direction along x axis is left -> right.
2) The positive direction along y axis is top -> down.
3) The w border is in parallel with x axis when angle = 0.

However, there are 2 opposite definitions of the positive angular direction, clockwise (CW) and counter-clockwise (CCW). MMCV supports both definitions and uses CW by default.

Please set `clockwise=False` if you are using the CCW definition.

The coordinate system when `clockwise` is `True` (default)

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

In such coordination system the rotation matrix is

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

The coordinates of the corner point A can be calculated as:

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

The coordinate system when `clockwise` is `False`

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

In such coordination system the rotation matrix is

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

The coordinates of the corner point A can be calculated as:

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

Parameters

- **boxes1** (*torch.Tensor*) – rotated bboxes 1. It has shape (N, 5), indicating (x, y, w, h, theta) for each row. Note that theta is in radian.
- **boxes2** (*torch.Tensor*) – rotated bboxes 2. It has shape (M, 5), indicating (x, y, w, h, theta) for each row. Note that theta is in radian.
- **mode** (*str*) – "iou" (intersection over union) or iof (intersection over foreground).
- **clockwise** (*bool*) – flag indicating whether the positive angular orientation is clockwise. default True. *New in version 1.4.3.*

Returns Return the ious betweens boxes. If `aligned` is `False`, the shape of ious is (N, M) else (N,).

Return type torch.Tensor

23.61 mmcv.ops.boxes_iou3d

mmcv.ops.**boxes_iou3d**(*boxes_a: torch.Tensor*, *boxes_b: torch.Tensor*) → torch.Tensor
Calculate boxes 3D IoU.

Parameters

- **boxes_a** (*torch.Tensor*) – Input boxes a with shape (M, 7).
- **boxes_b** (*torch.Tensor*) – Input boxes b with shape (N, 7).

Returns 3D IoU result with shape (M, N).

Return type torch.Tensor

23.62 mmcv.ops.boxes_iou_bev

mmcv.ops.**boxes_iou_bev**(*boxes_a: torch.Tensor*, *boxes_b: torch.Tensor*) → torch.Tensor
Calculate boxes IoU in the Bird's Eye View.

Parameters

- **boxes_a** (*torch.Tensor*) – Input boxes a with shape (M, 5) ([x1, y1, x2, y2, ry]).
- **boxes_b** (*torch.Tensor*) – Input boxes b with shape (N, 5) ([x1, y1, x2, y2, ry]).

Returns IoU result with shape (M, N).

Return type torch.Tensor

23.63 mmcv.ops.boxes_overlap_bev

mmcv.ops.**boxes_overlap_bev**(*boxes_a: torch.Tensor*, *boxes_b: torch.Tensor*) → torch.Tensor
Calculate boxes BEV overlap.

Parameters

- **boxes_a** (*torch.Tensor*) – Input boxes a with shape (M, 7).
- **boxes_b** (*torch.Tensor*) – Input boxes b with shape (N, 7).

Returns BEV overlap result with shape (M, N).

Return type torch.Tensor

23.64 mmcv.ops.carafe

mmcv.ops.**carafe**()

23.65 mmcv.ops.carafe_naive

mmcv.ops.**carafe_naive**()

23.66 mmcv.ops.chamfer_distance

mmcv.ops.**chamfer_distance**()

23.67 mmcv.ops.contour_expand

mmcv.ops.**contour_expand**(*kernel_mask: Union[numpy.array, torch.Tensor]*, *internal_kernel_label: Union[numpy.array, torch.Tensor]*, *min_kernel_area: int*, *kernel_num: int*) → list

Expand kernel contours so that foreground pixels are assigned into instances.

Parameters

- **kernel_mask** (*np.array or torch.Tensor*) – The instance kernel mask with size hxw.
- **internal_kernel_label** (*np.array or torch.Tensor*) – The instance internal kernel label with size hxw.
- **min_kernel_area** (*int*) – The minimum kernel area.
- **kernel_num** (*int*) – The instance kernel number.

Returns The instance index map with size hxw.

Return type list

23.68 mmcv.ops.convex_giou

mmcv.ops.**convex_giou**(*pointsets: torch.Tensor*, *polygons: torch.Tensor*) → Tuple[torch.Tensor, torch.Tensor]

Return generalized intersection-over-union (Jaccard index) between point sets and polygons.

Parameters

- **pointsets** (*torch.Tensor*) – It has shape (N, 18), indicating (x1, y1, x2, y2, ..., x9, y9) for each row.
- **polygons** (*torch.Tensor*) – It has shape (N, 8), indicating (x1, y1, x2, y2, x3, y3, x4, y4) for each row.

Returns The first element is the gious between point sets and polygons with the shape (N,). The second element is the gradient of point sets with the shape (N, 18).

Return type tuple[torch.Tensor, torch.Tensor]

23.69 mmcv.ops.convex_iou

mmcv.ops.**convex_iou**(*pointsets: torch.Tensor*, *polygons: torch.Tensor*) → torch.Tensor
Return intersection-over-union (Jaccard index) between point sets and polygons.

Parameters

- **pointsets** (*torch.Tensor*) – It has shape (N, 18), indicating (x1, y1, x2, y2, ..., x9, y9) for each row.
- **polygons** (*torch.Tensor*) – It has shape (K, 8), indicating (x1, y1, x2, y2, x3, y3, x4, y4) for each row.

Returns Return the ious between point sets and polygons with the shape (N, K).

Return type torch.Tensor

23.70 mmcv.ops.deform_conv2d

mmcv.ops.**deform_conv2d**()

23.71 mmcv.ops.deform_roi_pool

mmcv.ops.**deform_roi_pool**()

23.72 mmcv.ops.diff_iou_rotated_2d

mmcv.ops.**diff_iou_rotated_2d**(*box1: torch.Tensor*, *box2: torch.Tensor*) → torch.Tensor
Calculate differentiable iou of rotated 2d boxes.

Parameters

- **box1** (*Tensor*) – (B, N, 5) First box.
- **box2** (*Tensor*) – (B, N, 5) Second box.

Returns (B, N) IoU.

Return type Tensor

23.73 mmcv.ops.diff_iou_rotated_3d

mmcv.ops.**diff_iou_rotated_3d**(*box3d1: torch.Tensor*, *box3d2: torch.Tensor*) → torch.Tensor
Calculate differentiable iou of rotated 3d boxes.

Parameters

- **box3d1** (*Tensor*) – (B, N, 3+3+1) First box (x,y,z,w,h,l,alpha).
- **box3d2** (*Tensor*) – (B, N, 3+3+1) Second box (x,y,z,w,h,l,alpha).

Returns (B, N) IoU.

Return type Tensor

23.74 mmcv.ops.dynamic_scatter

mmcv.ops.**dynamic_scatter**()

23.75 mmcv.ops.furthest_point_sample

mmcv.ops.**furthest_point_sample**()

23.76 mmcv.ops.furthest_point_sample_with_dist

mmcv.ops.**furthest_point_sample_with_dist**()

23.77 mmcv.ops.fused_bias_leakyrelu

mmcv.ops.**fused_bias_leakyrelu**(*input: torch.Tensor*, *bias: torch.nn.parameter.Parameter*, *negative_slope: float = 0.2*, *scale: float = 1.4142135623730951*) → torch.Tensor

Fused bias leaky ReLU function.

This function is introduced in the StyleGAN2: Analyzing and Improving the Image Quality of StyleGAN

The bias term comes from the convolution operation. In addition, to keep the variance of the feature map or gradients unchanged, they also adopt a scale similarly with Kaiming initialization. However, since the $1+alpha^2$ is too small, we can just ignore it. Therefore, the final scale is just $\sqrt{2}$. Of course, you may change it with your own scale.

Parameters

- **input** (*torch.Tensor*) – Input feature map.
- **bias** (*nn.Parameter*) – The bias from convolution operation.
- **negative_slope** (*float, optional*) – Same as nn.LeakyRelu. Defaults to 0.2.
- **scale** (*float, optional*) – A scalar to adjust the variance of the feature map. Defaults to 2**0.5.

Returns Feature map after non-linear activation.

Return type torch.Tensor

23.78 mmcv.ops.gather_points

mmcv.ops.**gather_points**()

23.79 mmcv.ops.grouping_operation

mmcv.ops.**grouping_operation**()

23.80 mmcv.ops.knn

mmcv.ops.**knn**()

23.81 mmcv.ops.masked_conv2d

mmcv.ops.**masked_conv2d**()

23.82 mmcv.ops.min_area_polygons

mmcv.ops.**min_area_polygons**(*pointsets: torch.Tensor*) → torch.Tensor

Find the smallest polygons that surrounds all points in the point sets.

Parameters **pointsets** (*Tensor*) – point sets with shape (N, 18).

Returns Return the smallest polygons with shape (N, 8).

Return type torch.Tensor

23.83 mmcv.ops.modulated_deform_conv2d

mmcv.ops.**modulated_deform_conv2d**()

23.84 mmcv.ops.nms

mmcv.ops.**nms**(*boxes: Union[torch.Tensor, numpy.ndarray]*, *scores: Union[torch.Tensor, numpy.ndarray]*, *iou_threshold: float*, *offset: int = 0*, *score_threshold: float = 0*, *max_num: int = - 1*) → Tuple[Union[torch.Tensor, numpy.ndarray], Union[torch.Tensor, numpy.ndarray]]

Dispatch to either CPU or GPU NMS implementations.

The input can be either torch tensor or numpy array. GPU NMS will be used if the input is gpu tensor, otherwise CPU NMS will be used. The returned type will always be the same as inputs.

Parameters

- **boxes** (*torch.Tensor or np.ndarray*) – boxes in shape (N, 4).
- **scores** (*torch.Tensor or np.ndarray*) – scores in shape (N,).

- **iou_threshold** (*float*) – IoU threshold for NMS.
- **offset** (*int, 0 or 1*) – boxes' width or height is (x2 - x1 + offset).
- **score_threshold** (*float*) – score threshold for NMS.
- **max_num** (*int*) – maximum number of boxes after NMS.

Returns kept dets (boxes and scores) and indice, which always have the same data type as the input.

Return type tuple

Example

```
>>> boxes = np.array([[49.1, 32.4, 51.0, 35.9],
>>>                   [49.3, 32.9, 51.0, 35.3],
>>>                   [49.2, 31.8, 51.0, 35.4],
>>>                   [35.1, 11.5, 39.1, 15.7],
>>>                   [35.6, 11.8, 39.3, 14.2],
>>>                   [35.3, 11.5, 39.9, 14.5],
>>>                   [35.2, 11.7, 39.7, 15.7]], dtype=np.float32)
>>> scores = np.array([0.9, 0.9, 0.5, 0.5, 0.5, 0.4, 0.3],                dtype=np.
↪float32)
>>> iou_threshold = 0.6
>>> dets, inds = nms(boxes, scores, iou_threshold)
>>> assert len(inds) == len(dets) == 3
```

23.85 mmcv.ops.nms3d

mmcv.ops.**nms3d**(*boxes: torch.Tensor*, *scores: torch.Tensor*, *iou_threshold: float*) → torch.Tensor

3D NMS function GPU implementation (for BEV boxes).

Parameters

- **boxes** (*torch.Tensor*) – Input boxes with the shape of (N, 7) ([x, y, z, dx, dy, dz, heading]).
- **scores** (*torch.Tensor*) – Scores of boxes with the shape of (N).
- **iou_threshold** (*float*) – Overlap threshold of NMS.

Returns Indexes after NMS.

Return type torch.Tensor

23.86 mmcv.ops.nms3d_normal

mmcv.ops.**nms3d_normal**(*boxes: torch.Tensor*, *scores: torch.Tensor*, *iou_threshold: float*) → torch.Tensor

Normal 3D NMS function GPU implementation. The overlap of two boxes for IoU calculation is defined as the exact overlapping area of the two boxes WITH their yaw angle set to 0.

Parameters

- **boxes** (*torch.Tensor*) – Input boxes with shape (N, 7). ([x, y, z, dx, dy, dz, heading]).
- **scores** (*torch.Tensor*) – Scores of predicted boxes with shape (N).

- **iou_threshold** (*float*) – Overlap threshold of NMS.

Returns Remaining indices with scores in descending order.

Return type torch.Tensor

23.87 mmcv.ops.nms_bev

mmcv.ops.**nms_bev**(*boxes: torch.Tensor*, *scores: torch.Tensor*, *thresh: float*, *pre_max_size: Optional[int] = None*, *post_max_size: Optional[int] = None*) → torch.Tensor

NMS function GPU implementation (for BEV boxes).

The overlap of two boxes for IoU calculation is defined as the exact overlapping area of the two boxes. In this function, one can also set `pre_max_size` and `post_max_size`.

Parameters

- **boxes** (*torch.Tensor*) – Input boxes with the shape of (N, 5) ([x1, y1, x2, y2, ry]).
- **scores** (*torch.Tensor*) – Scores of boxes with the shape of (N,).
- **thresh** (*float*) – Overlap threshold of NMS.
- **pre_max_size** (*int, optional*) – Max size of boxes before NMS. Default: None.
- **post_max_size** (*int, optional*) – Max size of boxes after NMS. Default: None.

Returns Indexes after NMS.

Return type torch.Tensor

23.88 mmcv.ops.nms_match

mmcv.ops.**nms_match**(*dets: Union[torch.Tensor, numpy.ndarray]*, *iou_threshold: float*) → List[Union[numpy.ndarray, torch.Tensor]]

Matched dets into different groups by NMS.

NMS match is Similar to NMS but when a bbox is suppressed, nms match will record the indice of suppressed bbox and form a group with the indice of kept bbox. In each group, indice is sorted as score order.

Parameters

- **dets** (*torch.Tensor | np.ndarray*) – Det boxes with scores, shape (N, 5).
- **iou_threshold** (*float*) – IoU thresh for NMS.

Returns The outer list corresponds different matched group, the inner Tensor corresponds the indices for a group in score order.

Return type list[torch.Tensor | np.ndarray]

23.89 mmcv.ops.nms_normal_bev

mmcv.ops.**nms_normal_bev**(*boxes: torch.Tensor*, *scores: torch.Tensor*, *thresh: float*) → torch.Tensor

Normal NMS function GPU implementation (for BEV boxes).

The overlap of two boxes for IoU calculation is defined as the exact overlapping area of the two boxes WITH their yaw angle set to 0.

Parameters

- **boxes** (*torch.Tensor*) – Input boxes with shape (N, 5) ([x1, y1, x2, y2, ry]).
- **scores** (*torch.Tensor*) – Scores of predicted boxes with shape (N,).
- **thresh** (*float*) – Overlap threshold of NMS.

Returns Remaining indices with scores in descending order.

Return type torch.Tensor

23.90 mmcv.ops.nms_rotated

mmcv.ops.**nms_rotated**(*dets: torch.Tensor*, *scores: torch.Tensor*, *iou_threshold: float*, *labels: Optional[torch.Tensor] = None*, *clockwise: bool = True*) → Tuple[torch.Tensor, torch.Tensor]

Performs non-maximum suppression (NMS) on the rotated boxes according to their intersection-over-union (IoU).

Rotated NMS iteratively removes lower scoring rotated boxes which have an IoU greater than iou_threshold with another (higher scoring) rotated box.

Parameters

- **dets** (*torch.Tensor*) – Rotated boxes in shape (N, 5). They are expected to be in (x_ctr, y_ctr, width, height, angle_radian) format.
- **scores** (*torch.Tensor*) – scores in shape (N,).
- **iou_threshold** (*float*) – IoU thresh for NMS.
- **labels** (*torch.Tensor, optional*) – boxes' label in shape (N,).
- **clockwise** (*bool*) – flag indicating whether the positive angular orientation is clockwise. default True. *New in version 1.4.3.*

Returns kept dets(boxes and scores) and indice, which is always the same data type as the input.

Return type tuple

23.91 mmcv.ops.pixel_group

mmcv.ops.**pixel_group**(*score: Union[numpy.ndarray, torch.Tensor]*, *mask: Union[numpy.ndarray, torch.Tensor]*, *embedding: Union[numpy.ndarray, torch.Tensor]*, *kernel_label: Union[numpy.ndarray, torch.Tensor]*, *kernel_contour: Union[numpy.ndarray, torch.Tensor]*, *kernel_region_num: int*, *distance_threshold: float*) → List[List[float]]

Group pixels into text instances, which is widely used text detection methods.

Parameters

- **score** (*np.array or torch.Tensor*) – The foreground score with size hxw.
- **mask** (*np.array or Tensor*) – The foreground mask with size hxw.
- **embedding** (*np.array or torch.Tensor*) – The embedding with size hxwxc to distinguish instances.
- **kernel_label** (*np.array or torch.Tensor*) – The instance kernel index with size hxw.
- **kernel_contour** (*np.array or torch.Tensor*) – The kernel contour with size hxw.
- **kernel_region_num** (*int*) – The instance kernel region number.
- **distance_threshold** (*float*) – The embedding distance threshold between kernel and pixel in one instance.

Returns The instance coordinates and attributes list. Each element consists of averaged confidence, pixel number, and coordinates (x_i, y_i for all pixels) in order.

Return type list[list[float]]

23.92 mmcv.ops.point_sample

mmcv.ops.**point_sample**(*input: torch.Tensor*, *points: torch.Tensor*, *align_corners: bool = False*, ***kwargs*) → torch.Tensor

A wrapper around `grid_sample()` to support 3D point_coords tensors Unlike `torch.nn.functional.grid_sample()` it assumes point_coords to lie inside `[0, 1] x [0, 1]` square.

Parameters

- **input** (*torch.Tensor*) – Feature map, shape (N, C, H, W).
- **points** (*torch.Tensor*) – Image based absolute point coordinates (normalized), range [0, 1] x [0, 1], shape (N, P, 2) or (N, Hgrid, Wgrid, 2).
- **align_corners** (*bool, optional*) – Whether align_corners. Default: False

Returns Features of *point* on *input*, shape (N, C, P) or (N, C, Hgrid, Wgrid).

Return type torch.Tensor

23.93 mmcv.ops.points_in_boxes_all

mmcv.ops.**points_in_boxes_all**(*points: torch.Tensor*, *boxes: torch.Tensor*) → torch.Tensor
Find all boxes in which each point is (CUDA).

Parameters

- **points** (*torch.Tensor*) – [B, M, 3], [x, y, z] in LiDAR/DEPTH coordinate
- **boxes** (*torch.Tensor*) – [B, T, 7], num_valid_boxes <= T, [x, y, z, x_size, y_size, z_size, rz], (x, y, z) is the bottom center.

Returns Return the box indices of points with the shape of (B, M, T). Default background = 0.

Return type torch.Tensor

23.94 mmcv.ops.points_in_boxes_cpu

mmcv.ops.**points_in_boxes_cpu**(*points: torch.Tensor*, *boxes: torch.Tensor*) → torch.Tensor
Find all boxes in which each point is (CPU). The CPU version of *points_in_boxes_all()*.

Parameters

- **points** (*torch.Tensor*) – [B, M, 3], [x, y, z] in LiDAR/DEPTH coordinate
- **boxes** (*torch.Tensor*) – [B, T, 7], num_valid_boxes <= T, [x, y, z, x_size, y_size, z_size, rz], (x, y, z) is the bottom center.

Returns Return the box indices of points with the shape of (B, M, T). Default background = 0.

Return type torch.Tensor

23.95 mmcv.ops.points_in_boxes_part

mmcv.ops.**points_in_boxes_part**(*points: torch.Tensor*, *boxes: torch.Tensor*) → torch.Tensor
Find the box in which each point is (CUDA).

Parameters

- **points** (*torch.Tensor*) – [B, M, 3], [x, y, z] in LiDAR/DEPTH coordinate.
- **boxes** (*torch.Tensor*) – [B, T, 7], num_valid_boxes <= T, [x, y, z, x_size, y_size, z_size, rz] in LiDAR/DEPTH coordinate, (x, y, z) is the bottom center.

Returns Return the box indices of points with the shape of (B, M). Default background = -1.

Return type torch.Tensor

23.96 mmcv.ops.points_in_polygons

mmcv.ops.**points_in_polygons**(*points: torch.Tensor*, *polygons: torch.Tensor*) → torch.Tensor

Judging whether points are inside polygons, which is used in the ATSS assignment for the rotated boxes.

It should be noted that when the point is just at the polygon boundary, the judgment will be inaccurate, but the effect on assignment is limited.

Parameters

- **points** (*torch.Tensor*) – It has shape (B, 2), indicating (x, y). M means the number of predicted points.
- **polygons** (*torch.Tensor*) – It has shape (M, 8), indicating (x1, y1, x2, y2, x3, y3, x4, y4). M means the number of ground truth polygons.

Returns Return the result with the shape of (B, M), 1 indicates that the point is inside the polygon, 0 indicates that the point is outside the polygon.

Return type torch.Tensor

23.97 mmcv.ops.prroi_pool

mmcv.ops.**prroi_pool**()

23.98 mmcv.ops.rel_roi_point_to_rel_img_point

mmcv.ops.**rel_roi_point_to_rel_img_point**(*rois: torch.Tensor*, *rel_roi_points: torch.Tensor*, *img: Union[tuple, torch.Tensor]*, *spatial_scale: float = 1.0*) → torch.Tensor

Convert roi based relative point coordinates to image based absolute point coordinates.

Parameters

- **rois** (*torch.Tensor*) – RoIs or BBoxes, shape (N, 4) or (N, 5)
- **rel_roi_points** (*torch.Tensor*) – Point coordinates inside RoI, relative to RoI, location, range (0, 1), shape (N, P, 2)
- **img** (*tuple or torch.Tensor*) – (height, width) of image or feature map.
- **spatial_scale** (*float, optional*) – Scale points by this factor. Default: 1.

Returns Image based relative point coordinates for sampling, shape (N, P, 2).

Return type torch.Tensor

23.99 mmcv.ops.riroi_align_rotated

mmcv.ops.**riroi_align_rotated**()

23.100 mmcv.ops.roi_align

mmcv.ops.**roi_align**()

23.101 mmcv.ops.roi_align_rotated

mmcv.ops.**roi_align_rotated**()

23.102 mmcv.ops.roi_pool

mmcv.ops.**roi_pool**()

23.103 mmcv.ops.rotated_feature_align

mmcv.ops.**rotated_feature_align**(*features: torch.Tensor*, *best_rbboxes: torch.Tensor*, *spatial_scale: float = 0.125*, *points: int = 1*) → torch.Tensor

23.104 mmcv.ops.scatter_nd

mmcv.ops.**scatter_nd**(*indices: torch.Tensor*, *updates: torch.Tensor*, *shape: torch.Tensor*) → torch.Tensor

pytorch edition of tensorflow scatter_nd.

this function don't contain except handle code. so use this carefully when indice repeats, don't support repeat add which is supported in tensorflow.

23.105 mmcv.ops.sigmoid_focal_loss

mmcv.ops.**sigmoid_focal_loss**()

23.106 mmcv.ops.soft_nms

mmcv.ops.**soft_nms**(*boxes: Union[torch.Tensor, numpy.ndarray]*, *scores: Union[torch.Tensor, numpy.ndarray]*, *iou_threshold: float = 0.3*, *sigma: float = 0.5*, *min_score: float = 0.001*, *method: str = 'linear'*, *offset: int = 0*) → Tuple[Union[torch.Tensor, numpy.ndarray], Union[torch.Tensor, numpy.ndarray]]

Dispatch to only CPU Soft NMS implementations.

The input can be either a torch tensor or numpy array. The returned type will always be the same as inputs.

Parameters

- **boxes** (*torch.Tensor or np.ndarray*) – boxes in shape (N, 4).
- **scores** (*torch.Tensor or np.ndarray*) – scores in shape (N,).
- **iou_threshold** (*float*) – IoU threshold for NMS.
- **sigma** (*float*) – hyperparameter for gaussian method
- **min_score** (*float*) – score filter threshold
- **method** (*str*) – either 'linear' or 'gaussian'
- **offset** (*int, 0 or 1*) – boxes' width or height is (x2 - x1 + offset).

Returns kept dets (boxes and scores) and indice, which always have the same data type as the input.

Return type tuple

Example

```
>>> boxes = np.array([[4., 3., 5., 3.],
>>>                   [4., 3., 5., 4.],
>>>                   [3., 1., 3., 1.],
>>>                   [3., 1., 3., 1.],
>>>                   [3., 1., 3., 1.],
>>>                   [3., 1., 3., 1.]], dtype=np.float32)
>>> scores = np.array([0.9, 0.9, 0.5, 0.5, 0.4, 0.0], dtype=np.float32)
>>> iou_threshold = 0.6
>>> dets, inds = soft_nms(boxes, scores, iou_threshold, sigma=0.5)
>>> assert len(inds) == len(dets) == 5
```

23.107 mmcv.ops.softmax_focal_loss

mmcv.ops.**softmax_focal_loss**()

23.108 mmcv.ops.three_interpolate

mmcv.ops.**three_interpolate**()

23.109 mmcv.ops.three_nn

mmcv.ops.**three_nn**()

23.110 mmcv.ops.tin_shift

mmcv.ops.**tin_shift**()

23.111 mmcv.ops.upfirdn2d

mmcv.ops.**upfirdn2d**(*input: torch.Tensor*, *filter: torch.Tensor*, *up: int = 1*, *down: int = 1*, *padding: Union[int, List[int]] = 0*, *flip_filter: bool = False*, *gain: Union[float, int] = 1*, *use_custom_op: bool = True*)

Pad, upsample, filter, and downsample a batch of 2D images.

Performs the following sequence of operations for each channel:

1. Upsample the image by inserting N-1 zeros after each pixel (*up*).

2. Pad the image with the specified number of zeros on each side (*padding*). Negative padding corresponds to cropping the image.

3. Convolve the image with the specified 2D FIR filter (*f*), shrinking it so that the footprint of all output pixels lies within the input image.

4. Downsample the image by keeping every Nth pixel (*down*).

This sequence of operations bears close resemblance to scipy.signal.upfirdn().

The fused op is considerably more efficient than performing the same calculation using standard PyTorch ops. It supports gradients of arbitrary order.

Parameters

- **input** (*torch.Tensor*) – Float32/float64/float16 input tensor of the shape *[batch_size, num_channels, in_height, in_width]*.
- **filter** (*torch.Tensor*) – Float32 FIR filter of the shape *[filter_height, filter_width]* (non-separable), *[filter_taps]* (separable), or *None* (identity).
- **up** (*int*) – Integer upsampling factor. Can be a single int or a list/tuple *[x, y]*. Defaults to 1.
- **down** (*int*) – Integer downsampling factor. Can be a single int or a list/tuple *[x, y]*. Defaults to 1.
- **padding** (*int | tuple[int]*) – Padding with respect to the upsampled image. Can be a single number or a list/tuple *[x, y]* or *[x_before, x_after, y_before, y_after]*. Defaults to 0.
- **flip_filter** (*bool*) – False = convolution, True = correlation. Defaults to False.

- **gain** (*int*) – Overall scaling factor for signal magnitude. Defaults to 1.
- **use_custom_op** (*bool*) – Whether to use customized op. Defaults to True.

Returns Tensor of the shape *[batch_size, num_channels, out_height, out_width]*

23.112 mmcv.ops.voxelization

mmcv.ops.**voxelization**()

CHAPTER

TWENTYFOUR

MMCV.TRANSFORMS

BaseTransform	Base class for all transformations.
TestTimeAug	Test-time augmentation transform.

24.1 BaseTransform

class mmcv.transforms.**BaseTransform**

Base class for all transformations.

abstract transform(*results: Dict*) → Optional[Union[Dict, Tuple[List, List]]]

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

24.2 TestTimeAug

class mmcv.transforms.**TestTimeAug**(*transforms: list*)

Test-time augmentation transform.

An example configuration is as followed:

```
dict(type='TestTimeAug',
     transforms=[
        [dict(type='Resize', scale=(1333, 400), keep_ratio=True),
         dict(type='Resize', scale=(1333, 800), keep_ratio=True)],
        [dict(type='RandomFlip', prob=1.),
         dict(type='RandomFlip', prob=0.)],
        [dict(type='PackDetInputs',
              meta_keys=('img_id', 'img_path', 'ori_shape',
                         'img_shape', 'scale_factor', 'flip',
                         'flip_direction'))]])
```

`results` will be transformed using all transforms defined in `transforms` arguments.

For the above configuration, there are four combinations of resize and flip:

- Resize to (1333, 400) + no flip
- Resize to (1333, 400) + flip
- Resize to (1333, 800) + no flip
- resize to (1333, 800) + flip

After that, results are wrapped into lists of the same length as below:

```
dict(
    inputs=[...],
    data_samples=[...]
)
```

The length of `inputs` and `data_samples` are both 4.

Required Keys:

- Depending on the requirements of the `transforms` parameter.

Modified Keys:

- All output keys of each transform.

Parameters **transforms** (*list[list[dict]]*) – Transforms to be applied to data sampled from dataset. `transforms` is a list of list, and each list element usually represents a series of transforms with the same type and different arguments. Data will be processed by each list elements sequentially. See more information in *transform()*.

transform(*results: dict*) → dict

Apply all transforms defined in `transforms` to the results.

As the example given in *TestTimeAug*, `transforms` consists of 2 `Resize`, 2 `RandomFlip` and 1 `PackDetInputs`. The data sampled from dataset will be processed as follows:

1. Data will be processed by 2 `Resize` and return a list of 2 results.
2. Each result in list will be further passed to 2 `RandomFlip`, and aggregates into a list of 4 results.
3. Each result will be processed by `PackDetInputs`, and return a list of dict.
4. Aggregates the same fields of results, and finally returns a dict. Each value of the dict represents 4 transformed results.

Parameters **results** (*dict*) – Result dict contains the data to transform.

Returns The augmented data, where each value is wrapped into a list.

Return type dict

24.3 Loading

LoadAnnotations	Load and process the `instances` and `seg_map` annotation provided by dataset.
LoadImageFromFile	Load an image from file.

24.3.1 LoadAnnotations

class mmcv.transforms.**LoadAnnotations**(*with_bbox: bool = True*, *with_label: bool = True*, *with_seg: bool = False*, *with_keypoints: bool = False*, *imdecode_backend: str = 'cv2'*, *file_client_args: Optional[dict] = None*, ***, *backend_args: Optional[dict] = None*)

Load and process the `instances` and `seg_map` 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.
        'bbox': [x1, y1, x2, y2],

        # Label of image classification.
        'bbox_label': 1,

        # Used in key point detection.
        # Can only load the format of [x1, y1, v1,..., xn, yn, vn]. v[i]
        # means the visibility of this keypoint. n must be equal to the
        # number of keypoint categories.
        'keypoints': [x1, y1, v1, ..., xn, yn, vn]
        }
    ]
    # Filename of semantic or panoptic segmentation ground truth file.
    'seg_map_path': 'a/b/c'
}
```

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 uint8 type.
    'gt_seg_map': np.ndarray (H, W)
     # with (x, y, v) order, in np.float32 type.
    'gt_keypoints': np.ndarray(N, NK, 3)
}
```

Required Keys:

- instances
  - bbox (optional)
  - bbox_label
  - keypoints (optional)
- seg_map_path (optional)

Added Keys:

- gt_bboxes (np.float32)
- gt_bboxes_labels (np.int64)
- gt_seg_map (np.uint8)
- gt_keypoints (np.float32)

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_seg** (*bool*) – Whether to parse and load the semantic segmentation annotation. Defaults to False.
- **with_keypoints** (*bool*) – Whether to parse and load the keypoints annotation. Defaults to False.
- **imdecode_backend** (*str*) – The image decoding backend type. The backend argument for `mmcv.imfrombytes()`. See `mmcv.imfrombytes()` for details. Defaults to 'cv2'.
- **file_client_args** (*dict, optional*) – 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 2.0.0rc4.
- **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 2.0.0rc4.

transform(*results: dict*) → dict

Function to load multiple types annotations.

Parameters **results** (*dict*) – Result dict from `mmengine.dataset.BaseDataset`.

Returns The dict contains loaded bounding box, label and semantic segmentation and keypoints annotations.

Return type dict

24.3.2 LoadImageFromFile

class mmcv.transforms.**LoadImageFromFile**(*to_float32: bool = False*, *color_type: str = 'color'*, *imdecode_backend: str = 'cv2'*, *file_client_args: Optional[dict] = None*, *ignore_empty: bool = False*, ***, *backend_args: Optional[dict] = 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 `mmcv.imfrombytes()`. Defaults to 'color'.
- **imdecode_backend** (*str*) – The image decoding backend type. The backend argument for `mmcv.imfrombytes()`. See `mmcv.imfrombytes()` for details. Defaults to 'cv2'.
- **file_client_args** (*dict, optional*) – 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 2.0.0rc4.
- **ignore_empty** (*bool*) – Whether to allow loading empty image or file path not existent. Defaults to False.
- **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 2.0.0rc4.

transform(*results: dict*) → Optional[dict]

Functions to load image.

Parameters **results** (*dict*) – Result dict from `mmengine.dataset.BaseDataset`.

Returns The dict contains loaded image and meta information.

Return type dict

24.4 Processing

CenterCrop	Crop the center of the image, segmentation masks, bounding boxes and key points.
MultiScaleFlipAug	Test-time augmentation with multiple scales and flipping.
Normalize	Normalize the image.
Pad	Pad the image & segmentation map.
RandomChoiceResize	Resize images & bbox & mask from a list of multiple scales.
RandomFlip	Flip the image & bbox & keypoints & segmentation map.
RandomGrayscale	Randomly convert image to grayscale with a probability.
RandomResize	Random resize images & bbox & keypoints.
Resize	Resize images & bbox & seg & keypoints.
ToTensor	Convert some results to `torch.Tensor` by given keys.
ImageToTensor	Convert image to `torch.Tensor` by given keys.

24.4.1 CenterCrop

class mmcv.transforms.**CenterCrop**(*crop_size: Union[int, Tuple[int, int]]*, *auto_pad: bool = False*, *pad_cfg: dict = {'type': 'Pad'}*, *clip_object_border: bool = True*)

Crop the center of the image, segmentation masks, bounding boxes and key points. If the crop area exceeds the original image and `auto_pad` is True, the original image will be padded before cropping.

Required Keys:

- img
- gt_seg_map (optional)
- gt_bboxes (optional)
- gt_keypoints (optional)

Modified Keys:

- img
- img_shape
- gt_seg_map (optional)
- gt_bboxes (optional)
- gt_keypoints (optional)

Added Key:

- pad_shape

Parameters

- **crop_size** (*Union[int, Tuple[int, int]]*) – Expected size after cropping with the format of (w, h). If set to an integer, then cropping width and height are equal to this integer.
- **auto_pad** (*bool*) – Whether to pad the image if it's smaller than the `crop_size`. Defaults to False.

- **pad_cfg** (*dict*) – Base config for padding. Refer to `mmcv.Pad` for detail. Defaults to `dict(type='Pad')`.
- **clip_object_border** (*bool*) – Whether to clip the objects outside the border of the image. In some dataset like MOT17, the gt bboxes are allowed to cross the border of images. Therefore, we don't need to clip the gt bboxes in these cases. Defaults to True.

transform(*results: dict*) → dict

Apply center crop on results.

Parameters **results** (*dict*) – Result dict contains the data to transform.

Returns Results with CenterCropped image and semantic segmentation map.

Return type dict

24.4.2 MultiScaleFlipAug

class mmcv.transforms.**MultiScaleFlipAug**(*transforms: List[dict]*, *scales: Optional[Union[Tuple, List[Tuple]]] = None*, *scale_factor: Optional[Union[float, List[float]]] = None*, *allow_flip: bool = False*, *flip_direction: Union[str, List[str]] = 'horizontal'*, *resize_cfg: dict = {'keep_ratio': True, 'type': 'Resize'}*, *flip_cfg: dict = {'type': 'RandomFlip'}*)

Test-time augmentation with multiple scales and flipping.

An example configuration is as followed:

```
dict(
    type='MultiScaleFlipAug',
    scales=[(1333, 400), (1333, 800)],
    flip=True,
    transforms=[
        dict(type='Normalize', **img_norm_cfg),
        dict(type='Pad', size_divisor=1),
        dict(type='ImageToTensor', keys=['img']),
        dict(type='Collect', keys=['img'])
    ])
```

`results` will be resized using all the sizes in `scales`. If `flip` is True, then flipped results will also be added into output list.

For the above configuration, there are four combinations of resize and flip:

- Resize to (1333, 400) + no flip
- Resize to (1333, 400) + flip
- Resize to (1333, 800) + no flip
- resize to (1333, 800) + flip

The four results are then transformed with `transforms` argument. After that, results are wrapped into lists of the same length as below:

```
dict(
    inputs=[...],
    data_samples=[...]
)
```

Where the length of `inputs` and `data_samples` are both 4.

Required Keys:

- Depending on the requirements of the `transforms` parameter.

Modified Keys:

- All output keys of each transform.

Parameters

- **transforms** (*list[dict]*) – Transforms to be applied to each resized and flipped data.
- **scales** (*tuple | list[tuple] | None*) – Images scales for resizing.
- **scale_factor** (*float or tuple[float]*) – Scale factors for resizing. Defaults to None.
- **allow_flip** (*bool*) – Whether apply flip augmentation. Defaults to False.
- **flip_direction** (*str | list[str]*) – Flip augmentation directions, options are "horizontal", "vertical" and "diagonal". If flip_direction is a list, multiple flip augmentations will be applied. It has no effect when flip == False. Defaults to "horizontal".
- **resize_cfg** (*dict*) – Base config for resizing. Defaults to `dict(type='Resize', keep_ratio=True)`.
- **flip_cfg** (*dict*) – Base config for flipping. Defaults to `dict(type='RandomFlip')`.

transform(*results: dict*) → Dict

Apply test time augment transforms on results.

Parameters **results** (*dict*) – Result dict contains the data to transform.

Returns The augmented data, where each value is wrapped into a list.

Return type dict

24.4.3 Normalize

class mmcv.transforms.**Normalize**(*mean: Sequence[Union[int, float]]*, *std: Sequence[Union[int, float]]*, *to_rgb: bool = True*)

Normalize the image.

Required Keys:

- img

Modified Keys:

- img

Added Keys:

- img_norm_cfg
  - mean
  - std
  - to_rgb

Parameters

- **mean** (*sequence*) – Mean values of 3 channels.

- **std** (*sequence*) – Std values of 3 channels.
- **to_rgb** (*bool*) – Whether to convert the image from BGR to RGB before normlizing the image. If `to_rgb=True`, the order of mean and std should be RGB. If `to_rgb=False`, the order of mean and std should be the same order of the image. Defaults to True.

transform(*results: dict*) → dict
Function to normalize images.

Parameters **results** (*dict*) – Result dict from loading pipeline.

Returns Normalized results, key 'img_norm_cfg' key is added in to result dict.

Return type dict

24.4.4 Pad

class mmcv.transforms.**Pad**(*size: Optional[Tuple[int, int]] = None*, *size_divisor: Optional[int] = None*, *pad_to_square: bool = False*, *pad_val: Union[int, float, dict] = {'img': 0, 'seg': 255}*, *padding_mode: str = 'constant'*)
Pad the image & segmentation map.

There are three padding modes: (1) pad to a fixed size and (2) pad to the minimum size that is divisible by some number. and (3)pad to square. Also, pad to square and pad to the minimum size can be used as the same time.

Required Keys:

- img
- gt_bboxes (optional)
- gt_seg_map (optional)

Modified Keys:

- img
- gt_seg_map
- img_shape

Added Keys:

- pad_shape
- pad_fixed_size
- pad_size_divisor

Parameters

- **size** (*tuple, optional*) – Fixed padding size. Expected padding shape (w, h). Defaults to None.
- **size_divisor** (*int, optional*) – The divisor of padded size. Defaults to None.
- **pad_to_square** (*bool*) – Whether to pad the image into a square. Currently only used for YOLOX. Defaults to False.
- **pad_val** (*Number | dict[str, Number], optional*) – Padding value for if the pad_mode is "constant". If it is a single number, the value to pad the image is the number and to pad the semantic segmentation map is 255. If it is a dict, it should have the following keys:

  - img: The value to pad the image.
  - seg: The value to pad the semantic segmentation map.

  Defaults to dict(img=0, seg=255).

- **padding_mode** (*str*) – Type of padding. Should be: constant, edge, reflect or symmetric. Defaults to 'constant'.
  - constant: pads with a constant value, this value is specified with pad_val.
  - edge: pads with the last value at the edge of the image.
  - reflect: pads with reflection of image without repeating the last value on the edge. For example, padding [1, 2, 3, 4] with 2 elements on both sides in reflect mode will result in [3, 2, 1, 2, 3, 4, 3, 2].
  - symmetric: pads with reflection of image repeating the last value on the edge. For example, padding [1, 2, 3, 4] with 2 elements on both sides in symmetric mode will result in [2, 1, 1, 2, 3, 4, 4, 3]

transform(*results: dict*) → dict

Call function to pad images, masks, semantic segmentation maps.

Parameters **results** (*dict*) – Result dict from loading pipeline.

Returns Updated result dict.

Return type dict

24.4.5 RandomChoiceResize

class mmcv.transforms.**RandomChoiceResize**(*scales: Sequence[Union[int, Tuple]]*, *resize_type: str = 'Resize'*, ***resize_kwargs*)

Resize images & bbox & mask from a list of multiple scales.

This transform resizes the input image to some scale. Bboxes and masks are then resized with the same scale factor. Resize scale will be randomly selected from `scales`.

How to choose the target scale to resize the image will follow the rules below:

- if *scale* is a list of tuple, the target scale is sampled from the list uniformally.
- if *scale* is a tuple, the target scale will be set to the tuple.

Required Keys:

- img
- gt_bboxes (optional)
- gt_seg_map (optional)
- gt_keypoints (optional)

Modified Keys:

- img
- img_shape
- gt_bboxes (optional)
- gt_seg_map (optional)
- gt_keypoints (optional)

Added Keys:

- scale
- scale_factor
- scale_idx
- keep_ratio

Parameters

- **scales** (*Union[list, Tuple]*) – Images scales for resizing.
- **resize_type** (*str*) – The type of resize class to use. Defaults to "Resize".
- ****resize_kwargs** – Other keyword arguments for the `resize_type`.

Note: By defaults, the `resize_type` is "Resize", if it's not overwritten by your registry, it indicates the `mmcv.Resize`. And therefore, `resize_kwargs` accepts any keyword arguments of it, like `keep_ratio`, `interpolation` and so on.

If you want to use your custom resize class, the class should accept `scale` argument and have `scale` attribution which determines the resize shape.

---

transform(*results: dict*) → dict

Apply resize transforms on results from a list of scales.

Parameters **results** (*dict*) – Result dict contains the data to transform.

Returns Resized results, 'img', 'gt_bboxes', 'gt_seg_map', 'gt_keypoints', 'scale', 'scale_factor', 'img_shape', and 'keep_ratio' keys are updated in result dict.

Return type dict

24.4.6 RandomFlip

class mmcv.transforms.**RandomFlip**(*prob: Optional[Union[float, Iterable[float]]] = None*, *direction: Union[str, Sequence[Optional[str]]] = 'horizontal'*, *swap_seg_labels: Optional[Sequence] = None*)

Flip the image & bbox & keypoints & segmentation map. Added or Updated keys: flip, flip_direction, img, gt_bboxes, gt_seg_map, and gt_keypoints. 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_seg_map (optional)
- gt_keypoints (optional)

Modified Keys:

- img
- gt_bboxes (optional)
- gt_seg_map (optional)
- gt_keypoints (optional)

Added Keys:

- flip
- flip_direction
- swap_seg_labels (optional)

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** (*list, optional*) – The label pair need to be swapped for ground truth, like 'left arm' and 'right arm' need to be swapped after horizontal flipping. For example, `[(1, 5)]`, where 1/5 is the label of the left/right arm. Defaults to None.

transform(*results: dict*) → dict

Transform function to flip images, bounding boxes, semantic segmentation map and keypoints.

Parameters **results** (*dict*) – Result dict from loading pipeline.

Returns Flipped results, 'img', 'gt_bboxes', 'gt_seg_map', 'gt_keypoints', 'flip', and 'flip_direction' keys are updated in result dict.

Return type dict

24.4.7 RandomGrayscale

class mmcv.transforms.**RandomGrayscale**(*prob: float = 0.1*, *keep_channels: bool = False*, *channel_weights: Sequence[float] = (1.0, 1.0, 1.0)*, *color_format: str = 'bgr'*)

Randomly convert image to grayscale with a probability.

Required Key:

- img

Modified Key:

- img

Added Keys:

- grayscale
- grayscale_weights

Parameters

- **prob** (*float*) – Probability that image should be converted to grayscale. Defaults to 0.1.
- **keep_channels** (*bool*) – Whether keep channel number the same as input. Defaults to False.
- **channel_weights** (*tuple*) – The grayscale weights of each channel, and the weights will be normalized. For example, (1, 2, 1) will be normalized as (0.25, 0.5, 0.25). Defaults to (1., 1., 1.).
- **color_format** (*str*) – Color format set to be any of 'bgr', 'rgb', 'hsv'. Note: 'hsv' image will be transformed into 'bgr' format no matter whether it is grayscaled. Defaults to 'bgr'.

transform(*results: dict*) → dict

Apply random grayscale on results.

Parameters **results** (*dict*) – Result dict contains the data to transform.

Returns Results with grayscale image.

Return type dict

24.4.8 RandomResize

class mmcv.transforms.**RandomResize**(*scale: Union[Tuple[int, int], Sequence[Tuple[int, int]]]*, *ratio_range: Optional[Tuple[float, float]] = None*, *resize_type: str = 'Resize'*, ***resize_kwargs*)

Random resize images & bbox & keypoints.

How to choose the target scale to resize the image will follow the rules below:

- if `scale` is a sequence of tuple

$$target\_scale[0] \sim Uniform([scale[0][0], scale[1][0]])$$

$$target\_scale[1] \sim Uniform([scale[0][1], scale[1][1]])$$

Following the resize order of weight and height in cv2, `scale[i][0]` is for width, and `scale[i][1]` is for height.

- if `scale` is a tuple

$$target\_scale[0] \sim Uniform([ratio\_range[0], ratio\_range[1]]) * scale[0]$$

$$target\_scale[1] \sim Uniform([ratio\_range[0], ratio\_range[1]]) * scale[1]$$

Following the resize order of weight and height in cv2, `ratio_range[0]` is for width, and `ratio_range[1]` is for height.

- if `keep_ratio` is True, the minimum value of `target_scale` will be used to set the shorter side and the maximum value will be used to set the longer side.
- if `keep_ratio` is False, the value of `target_scale` will be used to reisze the width and height accordingly.

Required Keys:

- img
- gt_bboxes
- gt_seg_map
- gt_keypoints

Modified Keys:

- img
- gt_bboxes
- gt_seg_map
- gt_keypoints
- img_shape

Added Keys:

- scale
- scale_factor
- keep_ratio

Parameters

- **scale** (*tuple or Sequence[tuple]*) – Images scales for resizing. Defaults to None.
- **ratio_range** (*tuple[float], optional*) – (min_ratio, max_ratio). Defaults to None.
- **resize_type** (*str*) – The type of resize class to use. Defaults to "Resize".
- ****resize_kwargs** – Other keyword arguments for the `resize_type`.

Note: By defaults, the `resize_type` is "Resize", if it's not overwritten by your registry, it indicates the `mmcv.Resize`. And therefore, `resize_kwargs` accepts any keyword arguments of it, like `keep_ratio`, `interpolation` and so on.

If you want to use your custom resize class, the class should accept `scale` argument and have `scale` attribution which determines the resize shape.

---

transform(*results: dict*) → dict

Transform function to resize images, bounding boxes, semantic segmentation map.

Parameters **results** (*dict*) – Result dict from loading pipeline.

Returns Resized results, `img`, `gt_bboxes`, `gt_semantic_seg`, `gt_keypoints`, `scale`, `scale_factor`, `img_shape`, and `keep_ratio` keys are updated in result dict.

Return type dict

24.4.9 Resize

class mmcv.transforms.**Resize**(*scale: Optional[Union[int, Tuple[int, int]]] = None*, *scale_factor: Optional[Union[float, Tuple[float, float]]] = None*, *keep_ratio: bool = False*, *clip_object_border: bool = True*, *backend: str = 'cv2'*, *interpolation='bilinear'*)

Resize images & bbox & seg & keypoints.

This transform resizes the input image according to `scale` or `scale_factor`. Bboxes, seg map and keypoints 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

- gt_bboxes (optional)
- gt_seg_map (optional)
- gt_keypoints (optional)

Modified Keys:

- img
- gt_bboxes
- gt_seg_map
- gt_keypoints
- img_shape

Added Keys:

- scale
- scale_factor
- keep_ratio

Parameters

- **scale** (*int or tuple*) – Images scales for resizing. Defaults to None
- **scale_factor** (*float or tuple[float]*) – Scale factors for resizing. 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. In some dataset like MOT17, the gt bboxes are allowed to cross the border of images. Therefore, we don't need to clip the gt bboxes in these cases. 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'.

transform(*results: dict*) → dict

Transform function to resize images, bounding boxes, semantic segmentation map and keypoints.

Parameters **results** (*dict*) – Result dict from loading pipeline.

Returns Resized results, 'img', 'gt_bboxes', 'gt_seg_map', 'gt_keypoints', 'scale', 'scale_factor', 'img_shape', and 'keep_ratio' keys are updated in result dict.

Return type dict

24.4.10 ToTensor

class mmcv.transforms.**ToTensor**(*keys: Sequence[str]*)

Convert some results to `torch.Tensor` by given keys.

Required keys:

- all these keys in *keys*

Modified Keys:

- all these keys in *keys*

Parameters **keys** (*Sequence[str]*) – Keys that need to be converted to Tensor.

transform(*results: dict*) → dict

Transform function to convert data to *torch.Tensor*.

Parameters **results** (*dict*) – Result dict from loading pipeline.

Returns *keys* in results will be updated.

Return type dict

24.4.11 ImageToTensor

class mmcv.transforms.**ImageToTensor**(*keys: dict*)

Convert image to `torch.Tensor` by given keys.

The dimension order of input image is (H, W, C). The pipeline will convert it to (C, H, W). If only 2 dimension (H, W) is given, the output would be (1, H, W).

Required keys:

- all these keys in *keys*

Modified Keys:

- all these keys in *keys*

Parameters **keys** (*Sequence[str]*) – Key of images to be converted to Tensor.

transform(*results: dict*) → dict

Transform function to convert image in results to `torch.Tensor` and transpose the channel order. :param results: Result dict contains the image data to convert. :type results: dict

Returns The result dict contains the image converted to :obj:`torch.Tensor` and transposed to (C, H, W) order.

Return type dict

24.5 Wrapper

Compose	Compose multiple transforms sequentially.
KeyMapper	A transform wrapper to map and reorganize the input/output of the wrapped transforms (or sub-pipeline).
RandomApply	Apply transforms randomly with a given probability.
RandomChoice	Process data with a randomly chosen transform from given candidates.
TransformBroadcaster	A transform wrapper to apply the wrapped transforms to multiple data items.

24.5.1 Compose

class mmcv.transforms.**Compose**(*transforms: Union[Dict, Callable[[Dict], Dict], Sequence[Union[Dict, Callable[[Dict], Dict]]]]*)

Compose multiple transforms sequentially.

Parameters **transforms** (*list[dict | callable]*) – Sequence of transform object or config dict to be composed.

Examples

```
>>> pipeline = [
>>>     dict(type='Compose',
>>>         transforms=[
>>>             dict(type='LoadImageFromFile'),
>>>             dict(type='Normalize')
>>>         ]
>>>     )
>>> ]
```

transform(*results: Dict*) → Optional[Dict]

Call function to apply transforms sequentially.

Parameters **results** (*dict*) – A result dict contains the results to transform.

Returns Transformed results.

Return type dict or None

24.5.2 KeyMapper

class mmcv.transforms.**KeyMapper**(*transforms: Optional[Union[Dict, Callable[[Dict], Dict], List[Union[Dict, Callable[[Dict], Dict]]]]] = None*, *mapping: Optional[Dict] = None*, *remapping: Optional[Dict] = None*, *auto_remap: Optional[bool] = None*, *allow_nonexist_keys: bool = False*)

A transform wrapper to map and reorganize the input/output of the wrapped transforms (or sub-pipeline).

Parameters

- **transforms** (*list[dict | callable], optional*) – Sequence of transform object or config dict to be wrapped.

- **mapping** (*dict*) – A dict that defines the input key mapping. The keys corresponds to the inner key (i.e., kwargs of the `transform` method), and should be string type. The values corresponds to the outer keys (i.e., the keys of the data/results), and should have a type of string, list or dict. None means not applying input mapping. Default: None.
- **remapping** (*dict*) – A dict that defines the output key mapping. The keys and values have the same meanings and rules as in the `mapping`. Default: None.
- **auto_remap** (*bool, optional*) – If True, an inverse of the mapping will be used as the remapping. If auto_remap is not given, it will be automatically set True if 'remapping' is not given, and vice versa. Default: None.
- **allow_nonexist_keys** (*bool*) – If False, the outer keys in the mapping must exist in the input data, or an exception will be raised. Default: False.

Examples

```
>>> # Example 1: KeyMapper 'gt_img' to 'img'
>>> pipeline = [
>>>     # Use KeyMapper to convert outer (original) field name
>>>     # 'gt_img' to inner (used by inner transforms) filed name
>>>     # 'img'
>>>     dict(type='KeyMapper',
>>>         mapping={'img': 'gt_img'},
>>>         # auto_remap=True means output key mapping is the revert of
>>>         # the input key mapping, e.g. inner 'img' will be mapped
>>>         # back to outer 'gt_img'
>>>         auto_remap=True,
>>>         transforms=[
>>>             # In all transforms' implementation just use 'img'
>>>             # as a standard field name
>>>             dict(type='Crop', crop_size=(384, 384)),
>>>             dict(type='Normalize'),
>>>         ])
>>> ]
```

```
>>> # Example 2: Collect and structure multiple items
>>> pipeline = [
>>>     # The inner field 'imgs' will be a dict with keys 'img_src'
>>>     # and 'img_tar', whose values are outer fields 'img1' and
>>>     # 'img2' respectively.
>>>     dict(type='KeyMapper',
>>>         dict(
>>>             type='KeyMapper',
>>>             mapping=dict(
>>>                 imgs=dict(
>>>                     img_src='img1',
>>>                     img_tar='img2')),
>>>         transforms=...)
>>> ]
```

```
>>> # Example 3: Manually set ignored keys by "..."
>>> pipeline = [
```

(continues on next page)

(continued from previous page)

```
>>>     ...
>>>     dict(type='KeyMapper',
>>>         mapping={
>>>             # map outer key "gt_img" to inner key "img"
>>>             'img': 'gt_img',
>>>             # ignore outer key "mask"
>>>             'mask': ...,
>>>         },
>>>         transforms=[
>>>             dict(type='RandomFlip'),
>>>         ])
>>>     ...
>>> ]
```

transform(*results: Dict*) → Dict
Apply mapping, wrapped transforms and remapping.

24.5.3 RandomApply

class mmcv.transforms.**RandomApply**(*transforms: Union[Dict, Callable[[Dict], Dict], List[Union[Dict, Callable[[Dict], Dict]]]]*, *prob: float = 0.5*)
Apply transforms randomly with a given probability.

Parameters

- **transforms** (*list[dict | callable]*) – The transform or transform list to randomly apply.
- **prob** (*float*) – The probability to apply transforms. Default: 0.5

Examples

```
>>> # config
>>> pipeline = [
>>>     dict(type='RandomApply',
>>>         transforms=[dict(type='HorizontalFlip')],
>>>         prob=0.3)
>>> ]
```

transform(*results: Dict*) → Optional[Dict]
Randomly apply the transform.

24.5.4 RandomChoice

class mmcv.transforms.**RandomChoice**(*transforms: List[Union[Dict, Callable[[Dict], Dict], List[Union[Dict, Callable[[Dict], Dict]]]]]*, *prob: Optional[List[float]] = None*)

Process data with a randomly chosen transform from given candidates.

Parameters

- **transforms** (*list[list]*) – A list of transform candidates, each is a sequence of transforms.
- **prob** (*list[float], optional*) – The probabilities associated with each pipeline. The length should be equal to the pipeline number and the sum should be 1. If not given, a uniform distribution will be assumed.

Examples

```
>>> # config
>>> pipeline = [
>>>     dict(type='RandomChoice',
>>>         transforms=[
>>>             [dict(type='RandomHorizontalFlip')],  # subpipeline 1
>>>             [dict(type='RandomRotate')],  # subpipeline 2
>>>         ]
>>>     )
>>> ]
```

transform(*results: Dict*) → Optional[Dict]

Randomly choose a transform to apply.

24.5.5 TransformBroadcaster

class mmcv.transforms.**TransformBroadcaster**(*transforms: List[Union[Dict, Callable[[Dict], Dict]]]*, *mapping: Optional[Dict] = None*, *remapping: Optional[Dict] = None*, *auto_remap: Optional[bool] = None*, *allow_nonexist_keys: bool = False*, *share_random_params: bool = False*)

A transform wrapper to apply the wrapped transforms to multiple data items. For example, apply Resize to multiple images.

Parameters

- **transforms** (*list[dict | callable]*) – Sequence of transform object or config dict to be wrapped.
- **mapping** (*dict*) – A dict that defines the input key mapping. Note that to apply the transforms to multiple data items, the outer keys of the target items should be remapped as a list with the standard inner key (The key required by the wrapped transform). See the following example and the document of `mmcv.transforms.wrappers.KeyMapper` for details.
- **remapping** (*dict*) – A dict that defines the output key mapping. The keys and values have the same meanings and rules as in the `mapping`. Default: None.
- **auto_remap** (*bool, optional*) – If True, an inverse of the mapping will be used as the remapping. If auto_remap is not given, it will be automatically set True if 'remapping' is not given, and vice versa. Default: None.

- **allow_nonexist_keys** (*bool*) – If False, the outer keys in the mapping must exist in the input data, or an exception will be raised. Default: False.
- **share_random_params** (*bool*) – If True, the random transform (e.g., RandomFlip) will be conducted in a deterministic way and have the same behavior on all data items. For example, to randomly flip either both input image and ground-truth image, or none. Default: False.

Note: To apply the transforms to each elements of a list or tuple, instead of separating data items, you can map the outer key of the target sequence to the standard inner key. See example 2. example.

Examples

```
>>> # Example 1: Broadcast to enumerated keys, each contains a single
>>> # data element
>>> pipeline = [
>>>     dict(type='LoadImageFromFile', key='lq'),  # low-quality img
>>>     dict(type='LoadImageFromFile', key='gt'),  # ground-truth img
>>>     # TransformBroadcaster maps multiple outer fields to standard
>>>     # the inner field and process them with wrapped transforms
>>>     # respectively
>>>     dict(type='TransformBroadcaster',
>>>         # case 1: from multiple outer fields
>>>         mapping={'img': ['lq', 'gt']},
>>>         auto_remap=True,
>>>         # share_random_param=True means using identical random
>>>         # parameters in every processing
>>>         share_random_param=True,
>>>         transforms=[
>>>             dict(type='Crop', crop_size=(384, 384)),
>>>             dict(type='Normalize'),
>>>         ])
>>> ]
```

```
>>> # Example 2: Broadcast to keys that contains data sequences
>>> pipeline = [
>>>     dict(type='LoadImageFromFile', key='lq'),  # low-quality img
>>>     dict(type='LoadImageFromFile', key='gt'),  # ground-truth img
>>>     # TransformBroadcaster maps multiple outer fields to standard
>>>     # the inner field and process them with wrapped transforms
>>>     # respectively
>>>     dict(type='TransformBroadcaster',
>>>         # case 2: from one outer field that contains multiple
>>>         # data elements (e.g. a list)
>>>         # mapping={'img': 'images'},
>>>         auto_remap=True,
>>>         share_random_param=True,
>>>         transforms=[
>>>             dict(type='Crop', crop_size=(384, 384)),
>>>             dict(type='Normalize'),
>>>         ])
>>> ]
```

```
>>> Example 3: Set ignored keys in broadcasting
>>> pipeline = [
>>>        dict(type='TransformBroadcaster',
>>>            # Broadcast the wrapped transforms to multiple images
>>>            # 'lq' and 'gt, but only update 'img_shape' once
>>>            mapping={
>>>                'img': ['lq', 'gt'],
>>>                'img_shape': ['img_shape', ...],
>>>             },
>>>            auto_remap=True,
>>>            share_random_params=True,
>>>            transforms=[
>>>                # `RandomCrop` will modify the field "img",
>>>                # and optionally update "img_shape" if it exists
>>>                dict(type='RandomCrop'),
>>>            ])
>>>    ]
```

scatter_sequence(*data: Dict*) → List[Dict]
Scatter the broadcasting targets to a list of inputs of the wrapped transforms.

transform(*results: Dict*)
Broadcast wrapped transforms to multiple targets.

CHAPTER

TWENTYFIVE

MMCV.ARRAYMISC

quantize	Quantize an array of (-inf, inf) to [0, levels-1].
dequantize	Dequantize an array.

25.1 mmcv.arraymisc.quantize

mmcv.arraymisc.**quantize**(*arr: numpy.ndarray, min_val: Union[int, float], max_val: Union[int, float], levels: int, dtype=<class 'numpy.int64'>*) → tuple

Quantize an array of (-inf, inf) to [0, levels-1].

Parameters

- **arr** (*ndarray*) – Input array.
- **min_val** (*int or float*) – Minimum value to be clipped.
- **max_val** (*int or float*) – Maximum value to be clipped.
- **levels** (*int*) – Quantization levels.
- **dtype** (*np.type*) – The type of the quantized array.

Returns Quantized array.

Return type tuple

25.2 mmcv.arraymisc.dequantize

mmcv.arraymisc.**dequantize**(*arr: numpy.ndarray, min_val: Union[int, float], max_val: Union[int, float], levels: int, dtype=<class 'numpy.float64'>*) → tuple

Dequantize an array.

Parameters

- **arr** (*ndarray*) – Input array.
- **min_val** (*int or float*) – Minimum value to be clipped.
- **max_val** (*int or float*) – Maximum value to be clipped.
- **levels** (*int*) – Quantization levels.
- **dtype** (*np.type*) – The type of the dequantized array.

Returns Dequantized array.

Return type tuple

CHAPTER

TWENTYSIX

MMCV.UTILS

IS_CUDA_AVAILABLE	bool(x) -> bool
IS_MLU_AVAILABLE	bool(x) -> bool
IS_MPS_AVAILABLE	bool(x) -> bool
collect_env	Collect the information of the running environments.
jit	
skip_no_elena	

26.1 mmcv.utils.IS_CUDA_AVAILABLE

mmcv.utils.**IS_CUDA_AVAILABLE = False**
bool(x) -> bool

Returns True when the argument x is true, False otherwise. The builtins True and False are the only two instances of the class bool. The class bool is a subclass of the class int, and cannot be subclassed.

26.2 mmcv.utils.IS_MLU_AVAILABLE

mmcv.utils.**IS_MLU_AVAILABLE = False**
bool(x) -> bool

Returns True when the argument x is true, False otherwise. The builtins True and False are the only two instances of the class bool. The class bool is a subclass of the class int, and cannot be subclassed.

26.3 mmcv.utils.IS_MPS_AVAILABLE

mmcv.utils.**IS_MPS_AVAILABLE = False**
bool(x) -> bool

Returns True when the argument x is true, False otherwise. The builtins True and False are the only two instances of the class bool. The class bool is a subclass of the class int, and cannot be subclassed.

26.4 mmcv.utils.collect_env

mmcv.utils.**collect_env**()
Collect the information of the running environments.

Returns

The environment information. The following fields are contained.

- sys.platform: The variable of `sys.platform`.
- Python: Python version.
- CUDA available: Bool, indicating if CUDA is available.
- GPU devices: Device type of each GPU.
- CUDA_HOME (optional): The env var `CUDA_HOME`.
- NVCC (optional): NVCC version.
- GCC: GCC version, "n/a" if GCC is not installed.
- MSVC: Microsoft Virtual C++ Compiler version, Windows only.
- PyTorch: PyTorch version.
- PyTorch compiling details: The output of `torch.__config__.show()`.
- TorchVision (optional): TorchVision version.
- OpenCV: OpenCV version.
- MMEngine: MMEngine version.
- MMCV: MMCV version.
- MMCV Compiler: The GCC version for compiling MMCV ops.
- MMCV CUDA Compiler: The CUDA version for compiling MMCV ops.

Return type dict

26.5 mmcv.utils.jit

mmcv.utils.**jit**(*func=None*, *check_input=None*, *full_shape=True*, *derivate=False*, *coderize=False*, *optimize=False*)

26.6 mmcv.utils.skip_no_elena

mmcv.utils.**skip_no_elena**(*func*)

CHAPTER

TWENTYSEVEN

INDICES AND TABLES

- genindex
- search

INDEX

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

Y