EasyCV

Release 0.4.1

EasyCV Authors

Jul 27, 2022

USER GUIDE

EasyCV is an all-in-one computer vision toolbox based on PyTorch, mainly focus on self-supervised learning, image classification, metric-learning, object detection and so on.

CHAPTER

ONE

PREPARE DATASETS

EasyCV provides various datasets for multi tasks. Please refer to the following guide for data preparation and keep the same data structure.

- *Image Classification*
- *Object Detection*
- *Self-Supervised Learning*
- *Pose (Keypoint)*

1.1 Image Classification

- *Cifar10*
- *Cifar100*
- *Imagenet-1k*
- *Imagenet-1k-TFrecords*

1.1.1 Cifar10

The CIFAR-10 are labeled subsets of the 80 million tiny images dataset. They were collected by Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton.

It consists of 60000 32x32 colour images in 10 classes, with 6000 images per class.

There are 50000 training images and 10000 test images.

Here is the list of classes in the CIFAR-10: `airplane, automobile, bird, cat, deer, dog, frog, horse, ship, truck`.

For more detailed information, please refer to CIFAR.

Download

Download data from cifar-10-python.tar.gz (163MB). And uncompress files to `data/cifar10`.

Directory structure is as follows:

```
data/cifar10
└── cifar-10-batches-py
    ├── batches.meta
    ├── data_batch_1
    ├── data_batch_2
    ├── data_batch_3
    ├── data_batch_4
    ├── data_batch_5
    ├── readme.html
    ├── read.py
    └── test_batch
```

1.1.2 Cifar100

The CIFAR-100 are labeled subsets of the 80 million tiny images dataset. They were collected by Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton.

This dataset is just like the CIFAR-10, except it has 100 classes containing 600 images each.

There are 500 training images and 100 testing images per class.

The 100 classes in the CIFAR-100 are grouped into 20 superclasses. Each image comes with a "fine" label (the class to which it belongs) and a "coarse" label (the superclass to which it belongs).

For more detailed information, please refer to CIFAR.

Download

Download data from cifar-100-python.tar.gz (161MB). And uncompress files to `data/cifar100`.

Directory structure should be as follows:

```
data/cifar100
└── cifar-100-python
    ├── file.txt~
    ├── meta
    ├── test
    ├── train
```

1.1.3 Imagenet-1k

ImageNet is an image database organized according to the WordNet hierarchy (currently only the nouns).

It is used in the ImageNet Large Scale Visual Recognition Challenge(ILSVRC) and is a benchmark for image classification.

For more detailed information, please refer to ImageNet.

Download

ILSVRC2012 is widely used, download it as follows:

1. Go to the download-url, Register an account and log in .
2. Recommended ILSVRC2012, download the following files
    - Training images (Task 1 & 2). 138GB.
    - Validation images (all tasks). 6.3GB.
3. Unzip the downloaded file.
4. Using this scrip to get data meta.

Directory structure should be as follows:

```
data/imagenet
└── train
    └── n01440764
    └── n01443537
    └── ...
└── val
    └── n01440764
    └── n01443537
    └── ...
└── meta
    ├── train.txt
    ├── val.txt
    ├── ...
```

1.1.4 Imagenet-1k-TFrecords

Original imagenet raw images packed in TFrecord format.

For more detailed information about Imagenet dataset, please refer to ImageNet.

Download

1. Go to the download-url, Register an account and log in .
2. The dataset is divided into two parts, part0 (79GB) and part1 (75GB), you need download all of them.

Directory structure should be as follows, put the image file and the idx file in the same folder:

```
data/imagenet
└── train
    ├── train-00000-of-01024
    ├── train-00000-of-01024.idx
    ├── train-00001-of-01024
    ├── train-00001-of-01024.idx
    ├── ...
└── validation
    ├── validation-00000-of-00128
    ├── validation-00000-of-00128.idx
    ├── validation-00001-of-00128
    ├── validation-00001-of-00128.idx
    ├── ...
```

1.2 Object Detection

- *PAI-iTAG detection*
- *COCO2017*
- *VOC2007*
- *VOC2012*

1.2.1 PAI-iTAG detection

`PAI-iTAG` is a platform for intelligent data annotation, which supports the annotation of various data types such as images, texts, videos, and audios, as well as multi-modal mixed annotation.

Please refer to iTAG for file format and data annotation.

Download

Download SmallCOCO dataset to `data/demo_itag_coco`, Directory structure should be as follows:

```
data/demo_itag_coco/
├── train2017
├── train2017_20_local.manifest
├── val2017
└── val2017_20_local.manifest
```

1.2.2 COCO2017

The COCO dataset is a large-scale object detection, segmentation, key-point detection, and captioning dataset. The dataset consists of 328K images.

The COCO dataset has been updated for several editions, and coco2017 is widely used. In 2017, the training/validation split was 118K/5K and test set is a subset of 41K images of the 2015 test set.

For more detailed information, please refer to COCO.

Download

Download train2017.zip (18G) ,val2017.zip (1G), annotations_trainval2017.zip (241MB) and uncompress files to to `data/coco2017`.

Directory structure is as follows:

```
data/coco2017
└── annotations
    ├── instances_train2017.json
    ├── instances_val2017.json
└── train2017
    ├── 000000000009.jpg
    ├── 000000000025.jpg
    ├── ...
└── val2017
    ├── 000000000139.jpg
    ├── 000000000285.jpg
    ├── ...
```

1.2.3 VOC2007

PASCAL VOC 2007 is a dataset for image recognition. The twenty object classes that have been selected are:

- *Person:* person
- *Animal:* bird, cat, cow, dog, horse, sheep
- *Vehicle:* aeroplane, bicycle, boat, bus, car, motorbike, train
- *Indoor:* bottle, chair, dining table, potted plant, sofa, tv/monitor

Each image in this dataset has pixel-level segmentation annotations, bounding box annotations, and object class annotations.

For more detailed information, please refer to voc2007.

Download

Download VOCtrainval_06-Nov-2007.tar (439MB) and uncompress files to to `data/VOCdevkit`.

Directory structure is as follows:

```
data/VOCdevkit
└── VOC2007
    └── Annotations
        ├── 000005.xml
        ├── 001010.xml
            ├── ...
    └── JPEGImages
        ├── 000005.jpg
        ├── 001010.jpg
        ├── ...
    └── SegmentationClass
        ├── 000005.png
        ├── 001010.png
        ├── ...
    └── SegmentationObject
        ├── 000005.png
        ├── 001010.png
        ├── ...
    └── ImageSets
        └── Layout
            ├── train.txt
            ├── trainval.txt
            ├── val.txt
        └── Main
            ├── train.txt
            ├── val.txt
            ├── ...
        └── Segmentation
            ├── train.txt
            ├── trainval.txt
            ├── val.txt
```

1.2.4 VOC2012

The PASCAL VOC 2012 dataset contains 20 object categories including:

- *Person:* person
- *Animal:* bird, cat, cow, dog, horse, sheep
- *Vehicle:* aeroplane, bicycle, boat, bus, car, motorbike, train
- *Indoor:* bottle, chair, dining table, potted plant, sofa, tv/monitor

Each image in this dataset has pixel-level segmentation annotations, bounding box annotations, and object class annotations.

For more detailed information, please refer to voc2012.

Download

Download VOCtrainval_11-May-2012.tar (2G) and uncompress files to to `data/VOCdevkit`.

Directory structure is as follows:

```
data/VOCdevkit
└── VOC2012
    └── Annotations
        ├── 000005.xml
        ├── 001010.xml
            ├── ...
    └── JPEGImages
        ├── 000005.jpg
        ├── 001010.jpg
        ├── ...
    └── SegmentationClass
        ├── 000005.png
        ├── 001010.png
        ├── ...
    └── SegmentationObject
        ├── 000005.png
        ├── 001010.png
        ├── ...
    └── ImageSets
        └── Layout
            ├── train.txt
            ├── trainval.txt
            ├── val.txt
        └── Main
            ├── train.txt
            ├── val.txt
            ├── ...
        └── Segmentation
            ├── train.txt
            ├── trainval.txt
            ├── val.txt
```

1.3 Self-Supervised Learning

- *Imagenet-1k*
- *Imagenet-1k-TFrecords*

1.3.1 Imagenet-1k

Refer to *Image Classification: Imagenet-1k*.

1.3.2 Imagenet-1k-TFrecords

Refer to *Image Classification: Imagenet-1k-TFrecords*.

1.4 Pose

- *COCO2017*

1.4.1 COCO2017

The COCO dataset is a large-scale object detection, segmentation, key-point detection, and captioning dataset. The dataset consists of 328K images.

The COCO dataset has been updated for several editions, and coco2017 is widely used. In 2017, the training/validation split was 118K/5K and test set is a subset of 41K images of the 2015 test set.

For more detailed information, please refer to COCO.

Download

Download it as follows:

1. Download data: train2017.zip (18G) , val2017.zip (1G)
2. Download annotations: annotations_trainval2017.zip (241MB)
3. Download person detection results: HRNet-Human-Pose-Estimation provides person detection result of COCO val2017 to reproduce our multi-person pose estimation results. Please download from OneDrive or GoogleDrive (26.2MB).

Then uncompress files to `data/coco2017`, directory structure is as follows:

```
data/coco2017
└── annotations
    ├── person_keypoints_train2017.json
    ├── person_keypoints_val2017.json
└── person_detection_results
    ├── COCO_val2017_detections_AP_H_56_person.json
    ├── COCO_test-dev2017_detections_AP_H_609_person.json
└── train2017
    ├── 000000000009.jpg
    ├── 000000000025.jpg
    ├── ...
└── val2017
    ├── 000000000139.jpg
    ├── 000000000285.jpg
    ├── ...
```

CHAPTER

TWO

QUICK START

2.1 Prerequisites

- python >= 3.6
- Pytorch >= 1.5
- mmcv >= 1.2.0
- nvidia-dali == 0.25.0

2.2 Installation

2.2.1 Prepare environment

1. Create a conda virtual environment and activate it.

```
conda create -n ev python=3.6 -y
conda activate ev
```

2. Install PyTorch and torchvision

   The master branch works with **PyTorch 1.5.1** or higher.

```
conda install pytorch==1.7.0 torchvision==0.8.0 -c pytorch
```

3. Install some python dependencies

   replace {cu_version} and {torch_version} to the version used in your environment

```
# install mmcv
pip install mmcv-full==1.4.4 -f https://download.openmmlab.com/mmcv/dist/{cu_
↪version}/{torch_version}/index.html
# for example, install mmcv-full for cuda10.1 and pytorch 1.7.0
pip install mmcv-full==1.4.4 -f https://download.openmmlab.com/mmcv/dist/cu101/
↪torch1.7.0/index.html

# install nvidia-dali
pip install http://pai-vision-data-hz.oss-cn-zhangjiakou.aliyuncs.com/third_party/
↪nvidia_dali_cuda100-0.25.0-1535750-py3-none-manylinux2014_x86_64.whl
```

(continues on next page)

(continued from previous page)

```
# install common_io for MaxCompute table read (optional)
pip install https://tfsmoke1.oss-cn-zhangjiakou.aliyuncs.com/tunnel_paiio/common_io/
↪py3/common_io-0.1.0-cp36-cp36m-linux_x86_64.whl
```

4. Install EasyCV

   You can simply install easycv with the following command:

```
pip install pai-easycv
```

   or clone the repository and then install it:

```
git clone https://github.com/Alibaba/EasyCV.git
cd easycv
pip install -r requirements.txt
pip install -v -e .  # or "python setup.py develop"
```

5. Install pai_nni and blade_compressin

   When you use model quantize and prune, you need to install pai_nni and blade_compressin with the following command:

```
# install torch >= 1.8.0
pip install torch==1.8.0 torchvision==0.9.0 torchaudio==0.8.0

# install mmcv >= 1.3.0 (torch version >= 1.8.0 does not support mmcv version < 1.3.
↪0)
pip install mmcv-full==1.4.4 -f https://download.openmmlab.com/mmcv/dist/{cu_
↪version}/{torch_version}/index.html

# install onnx and pai_nni
pip install onnx
pip install https://pai-nni.oss-cn-zhangjiakou.aliyuncs.com/release/2.5/pai_nni-2.5-
↪py3-none-manylinux1_x86_64.whl

# install blade_compression
pip install http://pai-vision-data-hz.oss-cn-zhangjiakou.aliyuncs.com/third_party/
↪blade_compression-0.0.1-py3-none-any.whl
```

2.2.2 Verification

```
Simple verification

```python
from easycv.apis import *
```

You can also verify your installation using follwing quick-start examples
```

2.3 Examples

- *Image classification example*
- *Self-supervised learning example*
- *object detection example*

CHAPTER

THREE

SELF-SUPERVISED LEARNING MODEL ZOO

3.1 Pretrained models

3.1.1 MAE

Pretrained on **ImageNet** dataset.

3.1.2 DINO

Pretrained on **ImageNet** dataset.

3.1.3 MoBY

Pretrained on **ImageNet** dataset.

3.1.4 MoCo V2

Pretrained on **ImageNet** dataset.

3.1.5 SwAV

Pretrained on **ImageNet** dataset.

3.2 Benchmarks

For detailed usage of benchmark tools, please refer to benchmark README.md.

3.2.1 ImageNet Linear Evaluation

3.2.2 ImageNet Finetuning

3.2.3 COCO2017 Object Detection

3.2.4 VOC2012 Aug Semantic Segmentation

CHAPTER

FOUR

DETECTION MODEL ZOO

4.1 YOLOX

Pretrained on COCO2017 dataset.

4.2 ViTDet

CHAPTER

FIVE

DEVELOP

5.1 1. Code Style

We adopt PEP8 as the preferred code style.

We use the following toolsseed isortseed isortseed isort for linting and formatting:

- flake8: linter
- yapf: formatter
- isort: sort imports

Style configurations of yapf and isort can be found in setup.cfg. We use pre-commit hook that checks and formats for `flake8`, `yapf`, `seed-isort-config`, `isort`, `trailing whitespaces`, fixes `end-of-files`, sorts `requirments.txt` automatically on every commit. The config for a pre-commit hook is stored in .pre-commit-config. After you clone the repository, you will need to install initialize pre-commit hook.

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

From the repository folder

```
pre-commit install
```

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

If you want to use pre-commit to check all the files, you can run

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

If you only want to format and lint your code, you can run

```
sh scripts/linter.sh
```

5.2 2. Test

5.2.1 2.1 Unit test

```
bash scripts/ci_test.sh
```

5.2.2 2.2 Test data

if you add new data, please do the following to commit it to git-lfs before "git commit":

```
python git-lfs/git_lfs.py add data/test/new_data
python git-lfs/git_lfs.py push
```

5.3 3. Build pip package

```
python setup.py sdist bdist_wheel
```

CHAPTER

SIX

SELF-SUPERVISED LEARNING TUTORIAL

6.1 Data Preparation

To download the dataset, please refer to *prepare_data.md*.

Self-supervised learning support imagenet(raw and tfrecord) format data.

6.1.1 Imagenet format

You can download Imagenet data or use your own unlabeld image data. You should provide a directory which contains images for self-supervised training and a filelist which contains image path to the root directory. For example, the image directory is as follows

```
images/
├── 0001.jpg
├── 0002.jpg
├── 0003.jpg
|...
└── 9999.jpg
```

the content of filelist is

```
0001.jpg
0002.jpg
0003.jpg
...
9999.jpg
```

6.2 Local & PAI-DSW

We use configs/selfsup/mocov2/mocov2_rn50_8xb32_200e_jpg.py as an example config in which two config variable should be modified

```
data_train_list = 'filelist.txt'
data_train_root = 'images'
```

6.2.1 Training

Single gpu:

```
python tools/train.py \
                ${CONFIG_PATH} \
                --work_dir ${WORK_DIR}
```

Multi gpus:

```
bash tools/dist_train.sh \
                ${NUM_GPUS} \
                ${CONFIG_PATH} \
                --work_dir ${WORK_DIR}
```

- NUM_GPUS: number of gpus
- CONFIG_PATH: the config file path of a selfsup method
- WORK_DIR: your path to save models and logs

Examples:

Edit data_rootpath in the ${CONFIG_PATH} to your own data path.

```
GPUS=8
bash tools/dist_train.sh configs/selfsup/mocov2/mocov2_rn50_8xb32_200e_jpg.py $GPUS
```

6.2.2 Export model

```
python tools/export.py \
                ${CONFIG_PATH} \
                ${CHECKPOINT} \
                ${EXPORT_PATH}
```

- CONFIG_PATH: the config file path of a selfsup method
- CHECKPOINT:your checkpoint file of a selfsup method named as epoch_*.pth
- EXPORT_PATH: your path to save export model

Examples:

```
python tools/export.py configs/selfsup/mocov2/mocov2_rn50_8xb32_200e_jpg.py \
    work_dirs/selfsup/mocov2/epoch_200.pth \
    work_dirs/selfsup/mocov2/epoch_200_export.pth
```

6.2.3 Feature extract

Download test_image

```
import cv2
from easycv.predictors.feature_extractor import TorchFeatureExtractor

output_ckpt = 'work_dirs/selfsup/mocov2/epoch_200_export.pth'
fe = TorchFeatureExtractor(output_ckpt)

img = cv2.imread('248347732153_1040.jpg')
img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
feature = fe.predict([img])
print(feature[0]['feature'].shape)
```

CHAPTER

SEVEN

YOLOX TUTORIAL

7.1 Data preparation

To download the dataset, please refer to *prepare_data.md*.

Yolox support both coco format and PAI-Itag detection format,

7.1.1 COCO format

To use coco data to train detection, you can refer to configs/detection/yolox/yolox_s_8xb16_300e_coco.py for more configuration details.

7.1.2 PAI-Itag detection format

To use pai-itag detection format data to train detection, you can refer to configs/detection/yolox/yolox_s_8xb16_300e_coco_pai.py for more configuration details.

7.2 Local & PAI-DSW

To use COCO format data, use config file `configs/detection/yolox/yolox_s_8xb16_300e_coco.py`

To use PAI-Itag format data, use config file `configs/detection/yolox/yolox_s_8xb16_300e_coco_pai.py`

7.2.1 Train

Single gpu:

```
python tools/train.py \
                ${CONFIG_PATH} \
                --work_dir ${WORK_DIR}
```

Multi gpus:

```
bash tools/dist_train.sh \
                ${NUM_GPUS} \
                ${CONFIG_PATH} \
                --work_dir ${WORK_DIR}
```

- NUM_GPUS: number of gpus
- CONFIG_PATH: the config file path of a detection method
- WORK_DIR: your path to save models and logs

Examples:

Edit data_rootpath in the ${CONFIG_PATH} to your own data path.

```
GPUS=8
bash tools/dist_train.sh configs/detection/yolox/yolox_s_8xb16_300e_coco.py $GPUS
```

7.2.2 Evaluation

Single gpu:

```
python tools/eval.py \
                ${CONFIG_PATH} \
                ${CHECKPOINT} \
                --eval
```

Multi gpus:

```
bash tools/dist_test.sh \
                ${CONFIG_PATH} \
                ${NUM_GPUS} \
                ${CHECKPOINT} \
                --eval
```

- CONFIG_PATH: the config file path of a detection method
- NUM_GPUS: number of gpus
- CHECKPOINT: the checkpoint file named as epoch_*.pth.

Examples:

```
GPUS=8
bash tools/dist_test.sh configs/detection/yolox/yolox_s_8xb16_300e_coco.py $GPUS work_
↪dirs/detection/yolox/epoch_300.pth --eval
```

7.2.3 Export model

```
python tools/export.py \
                ${CONFIG_PATH} \
                ${CHECKPOINT} \
                ${EXPORT_PATH}
```

- CONFIG_PATH: the config file path of a detection method
- CHECKPOINT:your checkpoint file of a detection method named as epoch_*.pth.
- EXPORT_PATH: your path to save export model

Examples:

```
python tools/export.py configs/detection/yolox/yolox_s_8xb16_300e_coco.py \
        work_dirs/detection/yolox/epoch_300.pth \
        work_dirs/detection/yolox/epoch_300_export.pth
```

7.2.4 Inference

Download test_image

```
import cv2
from easycv.predictors import TorchYoloXPredictor

output_ckpt = 'work_dirs/detection/yolox/epoch_300.pth'
detector = TorchYoloXPredictor(output_ckpt)

img = cv2.imread('000000017627.jpg')
img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
output = detector.predict([img])
print(output)

# visualize image
from matplotlib import pyplot as plt
image = img.copy()
for box, cls_name in zip(output[0]['detection_boxes'], output[0]['detection_class_names
↪']):
    # box is [x1,y1,x2,y2]
    box = [int(b) for b in box]
    image = cv2.rectangle(image, tuple(box[:2]), tuple(box[2:4]), (0,255,0), 2)
    cv2.putText(image, cls_name, (box[0], box[1]-5), cv2.FONT_HERSHEY_SIMPLEX, 1.0, (0,0,
↪255), 2)
plt.imshow(image)
plt.show()
```

CHAPTER

EIGHT

IMAGE CLASSIFICATION TUTORIAL

8.1 Data Preparation

To download the dataset, please refer to *prepare_data.md*.

Image classification support cifar and imagenet(raw and tfrecord) format data.

8.1.1 Cifar

To use Cifar data to train classification, you can refer to configs/classification/cifar10/swintiny_b64_5e_jpg.py for more configuration details.

8.1.2 Imagenet format

You can also use your self-defined data which follows `imagenet format`, you should provide a root directory which condatains images for classification training and a filelist which contains image path to the root directory. For example, the image root directory is as follows

```
images/
├── 0001.jpg
├── 0002.jpg
├── 0003.jpg
|...
└── 9999.jpg
```

each line of the filelist consists of two parts, subpath to the image files starting from the image root directory, class label string for the corresponding image, which are seperated by space

```
0001.jpg label1
0002.jpg label2
0003.jpg label3
...
9999.jpg label9999
```

To use Imagenet format data to train classification, you can refer to configs/classification/imagenet/imagenet_rn50_jpg.py for more configuration details.

8.2 Local & PAI-DSW

8.2.1 Training

Single gpu:

```
python tools/train.py \
                ${CONFIG_PATH} \
                --work_dir ${WORK_DIR}
```

Multi gpus:

```
bash tools/dist_train.sh \
                ${NUM_GPUS} \
                ${CONFIG_PATH} \
                --work_dir ${WORK_DIR}
```

- NUM_GPUS: number of gpus
- CONFIG_PATH: the config file path of a image classification method
- WORK_DIR: your path to save models and logs

Examples:

Edit data_rootpath in the ${CONFIG_PATH} to your own data path.

```
single gpu training:
```shell
python tools/train.py  configs/classification/cifar10/swintiny_b64_5e_jpg.py --work_dir␣
↪work_dirs/classification/cifar10/swintiny  --fp16
```

multi gpu training
```shell
GPUS=8
bash tools/dist_train.sh configs/classification/cifar10/swintiny_b64_5e_jpg.py $GPUS --
↪fp16
```

training using python api
```python
import easycv.tools

import os
# config_path can be a local file or http url
config_path = 'configs/classification/cifar10/swintiny_b64_5e_jpg.py'
easycv.tools.train(config_path, gpus=8, fp16=False, master_port=29527)
```
```

8.2.2 Evaluation

Single gpu:

```
python tools/eval.py \
                ${CONFIG_PATH} \
                ${CHECKPOINT} \
                --eval
```

Multi gpus:

```
bash tools/dist_test.sh \
                ${CONFIG_PATH} \
                ${NUM_GPUS} \
                ${CHECKPOINT} \
                --eval
```

- CONFIG_PATH: the config file path of a image classification method
- NUM_GPUS: number of gpus
- CHECKPOINT: the checkpoint file named as epoch_*.pth

Examples:

```
single gpu evaluation
```shell
python tools/eval.py configs/classification/cifar10/swintiny_b64_5e_jpg.py work_dirs/
↪classification/cifar10/swintiny/epoch_350.pth --eval --fp16
```

multi-gpu evaluation

```shell
GPUS=8
bash tools/dist_test.sh configs/classification/cifar10/swintiny_b64_5e_jpg.py $GPUS work_
↪dirs/classification/cifar10/swintiny/epoch_350.pth --eval --fp16
```

evaluation using python api
```python
import easycv.tools

import os
os.environ['CUDA_VISIBLE_DEVICES']='3,4,5,6'
config_path = 'configs/classification/cifar10/swintiny_b64_5e_jpg.py'
checkpoint_path = 'work_dirs/classification/cifar10/swintiny/epoch_350.pth'
easycv.tools.eval(config_path, checkpoint_path, gpus=8)
```
```

8.2.3 Export model for inference

````
If SyncBN is configured, we should replace it with BN in config file
```python
# imagenet_rn50.py
model = dict(
    ...
    backbone=dict(
        ...
        norm_cfg=dict(type='BN')), # SyncBN --> BN
    ...)
```

```shell
python tools/export.py configs/classification/cifar10/swintiny_b64_5e_jpg.py \
    work_dirs/classification/cifar10/swintiny/epoch_350.pth \
    work_dirs/classification/cifar10/swintiny/epoch_350_export.pth
```

or using python api
```python
import easycv.tools

config_path = './imagenet_rn50.py'
checkpoint_path = 'oss://pai-vision-data-hz/pretrained_models/easycv/resnet/resnet50.pth'
export_path = './resnet50_export.pt'
easycv.tools.export(config_path, checkpoint_path, export_path)
```
````

8.2.4 Inference

````
Download [test_image](http://pai-vision-data-hz.oss-cn-zhangjiakou.aliyuncs.com/data/
↪cifar10/qince_data/predict/aeroplane_s_000004.png)


```python
import cv2
from easycv.predictors.classifier import TorchClassifier

output_ckpt = 'work_dirs/classification/cifar10/swintiny/epoch_350_export.pth'
tcls = TorchClassifier(output_ckpt)

img = cv2.imread('aeroplane_s_000004.png')
# input image should be RGB order
img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
output = tcls.predict([img])
print(output)
```
````

CHAPTER

NINE

FILE TUTORIAL

The file module of easycv supports operations both on local and oss files, oss introduction please refer to: https://www.aliyun.com/product/oss .

If you operate oss files, you need refer to *access_oss* to authorize oss first.

9.1 Support operations

9.1.1 access_oss

Authorize oss.

Method1:

```
from easycv.file import io
io.access_oss(
ak_id='your_accesskey_id',
ak_secret='your_accesskey_secret',
hosts='your endpoint' or ['your endpoint1', 'your endpoint2'],
buckets='your bucket' or ['your bucket1', 'your bucket2'])
```

Method2:

Add oss config to your local file `~/.ossutilconfig`, as follows: More oss config information, please refer to: https://help.aliyun.com/document_detail/120072.html

```
[Credentials]
language = CH
endpoint = your endpoint
accessKeyID = your_accesskey_id
accessKeySecret = your_accesskey_secret
[Bucket-Endpoint]
bucket1 = endpoint1
bucket2 = endpoint2
```

If you want to modify the path of the default oss config file (`~/.ossutilconfig`), you can do as follows:

```
$ export OSS_CONFIG_FILE='your oss config file path'
```

Then run the following command, the config file will be read by default to authorize oss.

```
from easycv.file import io
io.access_oss()
```

Method3:

Set environment variables as follow, EasyCV will automatically parse environment variables for authorization:

```
import os

os.environ['OSS_ACCESS_KEY_ID'] = 'your_accesskey_id'
os.environ['OSS_ACCESS_KEY_SECRET'] = 'your_accesskey_secret'
os.environ['OSS_ENDPOINTS'] = 'your endpoint1,your endpoint2'  # split with ","
os.environ['OSS_BUCKETS'] = 'your bucket1,your bucket2'  # split with ","
```

9.1.2 open

Support w,wb, a, r, rb modes on oss path. Local path is the same usage as the python build-in open.

Example for oss:

`io.access_oss` please refer to [access_oss](# access_oss).

```
from easycv.file import io

io.access_oss('your oss config')

# Write something to a oss file.
with io.open('oss://bucket_name/demo.txt', 'w') as f:
        f.write("test")

# Read from a oss file.
with io.open('oss://bucket_name/demo.txt', 'r') as f:
  print(f.read())
```

Example for local:

```
from easycv.file import io

# Write something to a oss file.
with io.open('/your/local/path/demo.txt', 'w') as f:
        f.write("test")

# Read from a oss file.
with io.open('/your/local/path/demo.txt', 'r') as f:
  print(f.read())
```

9.1.3 exists

Whether the file exists, same usage as `os.path.exists`. Support local path and oss path.

Example for oss:

`io.access_oss` please refer to [access_oss](# access_oss).

```
from easycv.file import io

io.access_oss('your oss config')

ret = io.exists('oss://bucket_name/dir')
print(ret)
```

Example for Local:

```
from easycv.file import io

ret = io.exists('oss://bucket_name/dir')
print(ret)
```

9.1.4 move

Move src to dst, same usage as shutil.move. Support local path and oss path.

Example for oss:

`io.access_oss` please refer to [access_oss](# access_oss).

```
from easycv.file import io

io.access_oss('your oss config')

# move oss file to local
io.move('oss://bucket_name/file.txt', '/your/local/path/file.txt')
# move oss file to oss
io.move('oss://bucket_name/dir1/file.txt', 'oss://bucket_name/dir2/file.txt')
# move local file to oss
io.move('/your/local/file.txt', 'oss://bucket_name/file.txt')
# move directory
io.move('oss://bucket_name/dir1/', 'oss://bucket_name/dir2/')
```

Example for local:

```
from easycv.file import io

# move local file to local
io.move('/your/local/path1/file.txt', '/your/local/path2/file.txt')
# move local dir to local
io.move('/your/local/dir1', '/your/local/dir2')
```

9.1.5 copy

Copy a file from src to dst. Same usage as `shutil.copyfile`.If you want to copy a directory, please refert to [copytree](# copytree).

Example for oss:

`io.access_oss` please refer to [access_oss](# access_oss).

```
from easycv.file import io

io.access_oss('your oss config')

# Copy a file from local to oss:
io.copy('/your/local/file.txt', 'oss://bucket/dir/file.txt')
# Copy a oss file to local:
io.copy('oss://bucket/dir/file.txt', '/your/local/file.txt')
# Copy a file from oss to oss::
io.copy('oss://bucket/dir/file.txt', 'oss://bucket/dir/file2.txt')
```

Example for local:

```
from easycv.file import io

# Copy a file from local to local:
io.copy('/your/local/path1/file.txt', '/your/local/path2/file.txt'')
```

9.1.6 copytree

Copy files recursively from src to dst. Same usage as `shutil.copytree`.

If you want to copy a file, please use [copy](# copy).

Example for oss:

`io.access_oss` please refer to [access_oss](# access_oss).

```
from easycv.file import io

io.access_oss('your oss config')

# copy files from local to oss
io.copytree(src='/your/local/dir1', dst='oss://bucket_name/dir2')
# copy files from oss to local
io.copytree(src='oss://bucket_name/dir2', dst='/your/local/dir1')
# copy files from oss to oss
io.copytree(src='oss://bucket_name/dir1', dst='oss://bucket_name/dir2')
```

Example for local:

```
from easycv.file import io

# copy files from local to local
io.copytree(src='/your/local/dir1', dst='/your/local/dir2')
```

9.1.7 listdir

List all objects in path. Same usage as `os.listdir`.

Example for oss:

`io.access_oss` please refer to [access_oss](# access_oss).

```
from easycv.file import io

io.access_oss('your oss config')

ret = io.listdir('oss://bucket/dir', recursive=True)
print(ret)
```

Example for local:

```
from easycv.file import io

ret = io.listdir('oss://bucket/dir', recursive=True)
print(ret)
```

9.1.8 remove

Remove a file or a directory recursively. Same usage as `os.remove` or `shutil.rmtree`.

Example for oss:

`io.access_oss` please refer to [access_oss](# access_oss).

```
from easycv.file import io

io.access_oss('your oss config')

# Remove a oss file
io.remove('oss://bucket_name/file.txt')
# Remove a oss directory
io.remove('oss://bucket_name/dir/')
```

Example for local:

```
from easycv.file import io

# Remove a local file
io.remove('/your/local/path/file.txt')
# Remove a local directory
io.remove('/your/local/dir/')
```

9.1.9 rmtree

Remove directory recursively, same usage as `shutil.rmtree`.

Example for oss:

`io.access_oss` please refer to [access_oss](# access_oss).

```
from easycv.file import io

io.access_oss('your oss config')

io.remove('oss://bucket_name/dir_name/')
```

Example for local:

```
from easycv.file import io

io.remove('/your/local/dir/')
```

9.1.10 makedirs

Create directories recursively, same usage as `os.makedirs`.

Example for oss:

`io.access_oss` please refer to [access_oss](# access_oss).

```
from easycv.file import io

io.access_oss('your oss config')

io.makedirs('oss://bucket/new_dir/')
```

Example for local:

```
from easycv.file import io

io.makedirs('/your/local/new_dir/')
```

9.1.11 isdir

Return whether a path is directory, same usage as `os.path.isdir`.

Example for oss:

`io.access_oss` please refer to [access_oss](# access_oss).

```
from easycv.file import io

io.access_oss('your oss config') # only oss file need, refer to `IO.access_oss`
ret = io.isdir('oss://bucket/dir/')
print(ret)
```

Example for local:

```
from easycv.file import io

ret = io.isdir('your/local/dir/')
print(ret)
```

9.1.12 isfile

Return whether a path is file object, same usage as `os.path.isfile`.

Example for oss:

`io.access_oss` please refer to [access_oss](# access_oss).

```
from easycv.file import io

io.access_oss('your oss config')
ret = io.isfile('oss://bucket/file.txt')
print(ret)
```

Example for local:

```
from easycv.file import io

ret = io.isfile('/your/local/path/file.txt')
print(ret)
```

9.1.13 glob

Return a list of paths matching a pathname pattern.

Example for oss:

`io.access_oss` please refer to [access_oss](# access_oss).

```
from easycv.file import io

io.access_oss('your oss config')

ret = io.glob('oss://bucket/dir/*.txt')
print(ret)
```

Example for local:

```
from easycv.file import io

ret = io.glob('/your/local/dir/*.txt')
print(ret)
```

9.1.14 size

Get the size of file path, same usage as `os.path.getsize`.

Example for oss:

`io.access_oss` please refer to [access_oss](# access_oss).

```
from easycv.file import io

io.access_oss('your oss config')

size = io.size('oss://bucket/file.txt')
print(size)
```

Example for local:

```
from easycv.file import io

size = io.size('/your/local/path/file.txt')
print(size)
```

CHAPTER

TEN

V 0.2.2 (07/04/2022)

- initial commit & first release

1. SOTA SSL Algorithms

EasyCV provides state-of-the-art algorithms in self-supervised learning based on contrastive learning such as SimCLR, MoCO V2, Swav, DINO and also MAE based on masked image modeling. We also provides standard benchmark tools for ssl model evaluation.

1. Vision Transformers

EasyCV aims to provide plenty vision transformer models trained either using supervised learning or self-supervised learning, such as ViT, Swin-Transformer and XCit. More models will be added in the future.

1. Functionality & Extensibility

In addition to SSL, EasyCV also support image classification, object detection, metric learning, and more area will be supported in the future. Although convering different area, EasyCV decompose the framework into different componets such as dataset, model, running hook, making it easy to add new compoenets and combining it with existing modules. EasyCV provide simple and comprehensive interface for inference. Additionaly, all models are supported on PAI-EAS, which can be easily deployed as online service and support automatic scaling and service moniting.

1. Efficiency

EasyCV support multi-gpu and multi worker training. EasyCV use DALI to accelerate data io and preprocessing process, and use fp16 to accelerate training process. For inference optimization, EasyCV export model using jit script, which can be optimized by PAI-Blade.

CHAPTER

ELEVEN

V 0.3.0 (05/05/2022)

11.1 Highlights

- Support image visualization for tensorboard and wandb (#15)

11.2 New Features

- Update moby pretrained model to deit small (#10)
- Support image visualization for tensorboard and wandb (#15)
- Add mae vit-large benchmark and pretrained models (#24)

11.3 Bug Fixes

- Fix extract.py for benchmarks (#7)
- Fix inference error of classifier (#19)
- Fix multi-process reading of detection datasource and accelerate data preprocessing (#23)
- Fix torchvision transforms wrapper (#31)

11.4 Improvements

- Add chinese readme (#39)
- Add model compression tutorial (#20)
- Add notebook tutorials (#22)
- Uniform input and output format for transforms (#6)
- Update model zoo link (#8)
- Support readthedocs (#29)
- refine autorelease gitworkflow (#13)

CHAPTER

TWELVE

V 0.4.0 (23/06/2022)

12.1 Highlights

- Add **semantic segmentation** modules, support FCN algorithm (#71)
- Expand classification model zoo (#55)
- Support export model with **blade** for yolox (#66)
- Support **ViTDet** algorithm (#35)
- Add sailfish for extensible fully sharded data parallel training (#97)
- Support run with mmdetection models (#25)

12.2 New Features

- Set multiprocess env for speedup (#77)
- Add data hub, summarized various datasets in different fields (#70)

12.3 Bug Fixes

- Fix the inaccurate accuracy caused by missing the `groundtruth_is_crowd` field in CocoMaskEvaluator (#61)
- Unified the usage of `pretrained` parameter and fix load bugs(#79) (#85) (#95)

12.4 Improvements

- Update MAE pretrained models and benchmark (#50)
- Add detection benchmark for SwAV and MoCo-v2 (#58)
- Add moby swin-tiny pretrained model and benchmark (#72)
- Update prepare_data.md, add more details (#69)
- Optimize quantize code and support to export MNN model (#44)

CHAPTER

THIRTEEN

EASYCV.APIS PACKAGE

13.1 Submodules

13.2 easycv.apis.export module

easycv.apis.export.**export**(*cfg*, *ckpt_path*, *filename*)
export model for inference

Parameters

- **cfg** – Config object
- **ckpt_path** (*str*) – path to checkpoint file
- **filename** (*str*) – filename to save exported models

class easycv.apis.export.**PreProcess**(*target_size: Tuple[int, int] = (640, 640)*, *keep_ratio: bool = True*)
Bases: object

Process the data input to model.

Parameters

- **target_size** (*Tuple[int, int]*) – output spatial size.
- **keep_ratio** (*bool*) – Whether to keep the aspect ratio when resizing the image.

__init__(*target_size: Tuple[int, int] = (640, 640)*, *keep_ratio: bool = True*)
Initialize self. See help(type(self)) for accurate signature.

class easycv.apis.export.**DetPostProcess**(*max_det: int = 100*, *score_thresh: float = 0.5*)
Bases: object

Process output values of detection models.

Parameters **max_det** – max number of detections to keep.

__init__(*max_det: int = 100*, *score_thresh: float = 0.5*)
Initialize self. See help(type(self)) for accurate signature.

class easycv.apis.export.**End2endModelExportWrapper**(*model*, *example_inputs*, *preprocess_fn: Optional[Callable] = None*, *postprocess_fn: Optional[Callable] = None*, *trace_model: bool = True*)
Bases: torch.nn.modules.module.Module

Model export wrapper that supports end-to-end export of pre-processing and post-processing. We support some built-in preprocessing and postprocessing functions. If the requirements are not met, you can customize

the preprocessing and postprocessing functions. The custom functions must support satisfy requirements of *torch.jit.script*, please refer to: https://pytorch.org/docs/stable/jit_language_reference_v2.html

Parameters

- **model** (*torch.nn.Module*) – *torch.nn.Module* that will be run with *example_inputs*. *model* arguments and return values must be tensors or (possibly nested) tuples that contain tensors. When a module is passed *torch.jit.trace*, only the `forward_export` method is run and traced (see `torch.jit.trace` for details).
- **example_inputs** (*tuple or torch.Tensor*) – A tuple of example inputs that will be passed to the function while tracing. The resulting trace can be run with inputs of different types and shapes assuming the traced operations support those types and shapes. *example_inputs* may also be a single Tensor in which case it is automatically wrapped in a tuple.
- **preprocess_fn** (*callable or None*) – A Python function for processing example_input. If there is only one return value, it will be passed to *model.forward_export*. If there are multiple return values, the first return value will be passed to *model.forward_export*, and the remaining return values will be passed to *postprocess_fn*.
- **postprocess_fn** (*callable or None*) – A Python function for processing the output value of the model. If *preprocess_fn* has multiple outputs, the output value of *preprocess_fn* will also be passed to *postprocess_fn*. For details, please refer to: *preprocess_fn*.
- **trace_model** (*bool*) – If True, before exporting the end-to-end model, *torch.jit.trace* will be used to export the *model* first. Traceing an `nn.Module` by default will compile the `forward_export` method and recursively.

Examples

import torch

batch_size = 1 example_inputs = 255 * torch.rand((batch_size, 3, 640, 640), device='cuda') end2end_model = End2endModelExportWrapper(

model, example_inputs, preprocess_fn=PreProcess(target_size=(640, 640)), # *PreProcess* refer to ev_torch.apis.export.PreProcess postprocess_fn=DetPostProcess() # *DetPostProcess* refer to ev_torch.apis.export.DetPostProcess trace_model=True)

model_script = torch.jit.script(end2end_model) with io.open('/tmp/model.jit', 'wb') as f:

torch.jit.save(model_script, f)

__init__(*model*, *example_inputs*, *preprocess_fn: Optional[Callable] = None*, *postprocess_fn: Optional[Callable] = None*, *trace_model: bool = True*) → None

Initializes internal Module state, shared by both nn.Module and ScriptModule.

trace_module(***kwargs*)

training: bool

forward(*image*)

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.

13.3 easycv.apis.test module

easycv.apis.test.**single_cpu_test**(*model*, *data_loader*, *mode='test'*, *show=False*, *out_dir=None*, *show_score_thr=0.3*, ***kwargs*)

easycv.apis.test.**single_gpu_test**(*model*, *data_loader*, *mode='test'*, *use_fp16=False*, ***kwargs*)

Test model with single.

This method tests model with single

Parameters

- **model** (*str*) – Model to be tested.
- **data_loader** (*nn.Dataloader*) – Pytorch data loader.
- **model** – mode for model to forward
- **use_fp16** – Use fp16 inference

Returns The prediction results.

Return type list

easycv.apis.test.**multi_gpu_test**(*model*, *data_loader*, *mode='test'*, *tmpdir=None*, *gpu_collect=False*, *use_fp16=False*, ***kwargs*)

Test model with multiple gpus.

This method tests model with multiple gpus and collects the results under two different modes: gpu and cpu modes. By setting 'gpu_collect=True' it encodes results to gpu tensors and use gpu communication for results collection. On cpu mode it saves the results on different gpus to 'tmpdir' and collects them by the rank 0 worker.

Parameters

- **model** (*str*) – Model to be tested.
- **data_loader** (*nn.Dataloader*) – Pytorch data loader.
- **model** – mode for model to forward
- **tmpdir** (*str*) – Path of directory to save the temporary results from different gpus under cpu mode.
- **gpu_collect** (*bool*) – Option to use either gpu or cpu to collect results.
- **use_fp16** – Use fp16 inference

Returns The prediction results.

Return type list

easycv.apis.test.**collect_results_cpu**(*result_part*, *size*, *tmpdir=None*)

easycv.apis.test.**serialize_tensor**(*tensor_collection*)

easycv.apis.test.**collect_results_gpu**(*result_part*, *size*)

13.4 easycv.apis.train module

easycv.apis.train.**set_random_seed**(*seed*, *deterministic=False*)
Set random seed.

Parameters

- **seed** (*int*) – Seed to be used.
- **deterministic** (*bool*) – Whether to set the deterministic option for CUDNN backend, i.e., set *torch.backends.cudnn.deterministic* to True and *torch.backends.cudnn.benchmark* to False. Default: False.

easycv.apis.train.**train_model**(*model*, *data_loaders*, *cfg*, *distributed=False*, *timestamp=None*, *meta=None*, *use_fp16=False*, *validate=True*, *gpu_collect=True*)
Training API.

Parameters

- **model** (nn.Module) – user defined model
- **data_loaders** – a list of dataloader for training data
- **cfg** – config object
- **distributed** – distributed training or not
- **timestamp** – time str formated as '%Y%m%d_%H%M%S'
- **meta** – a dict containing meta data info, such as env_info, seed, iter, epoch
- **use_fp16** – use fp16 training or not
- **validate** – do evaluation while training
- **gpu_collect** – use gpu collect or cpu collect for tensor gathering

easycv.apis.train.**get_skip_list_keywords**(*model*)

easycv.apis.train.**build_optimizer**(*model*, *optimizer_cfg*)
Build optimizer from configs.

Parameters

- **model** (nn.Module) – The model with parameters to be optimized.
- **optimizer_cfg** (*dict*) – The config dict of the optimizer.

  Positional fields are:

  - type: class name of the optimizer.
  - lr: base learning rate.

  Optional fields are:

  - any arguments of the corresponding optimizer type, e.g., weight_decay, momentum, etc.
  - paramwise_options: a dict with regular expression as keys to match parameter names and a dict containing options as values. Options include 6 fields: lr, lr_mult, momentum, momentum_mult, weight_decay, weight_decay_mult.

Returns The initialized optimizer.

Return type torch.optim.Optimizer

Example

```
>>> model = torch.nn.modules.Conv1d(1, 1, 1)
>>> paramwise_options = {
>>>     '(bn|gn)(\d+)?.(weight|bias)': dict(weight_decay_mult=0.1),
>>>     '\Ahead.': dict(lr_mult=10, momentum=0)}
>>> optimizer_cfg = dict(type='SGD', lr=0.01, momentum=0.9,
>>>                      weight_decay=0.0001,
>>>                      paramwise_options=paramwise_options)
>>> optimizer = build_optimizer(model, optimizer_cfg)
```

13.5 easycv.apis.train_misc module

easycv.apis.train_misc.**build_yolo_optimizer**(*model*, *optimizer_cfg*)
build optimizer for yolo.

CHAPTER

FOURTEEN

EASYCV.DATASETS PACKAGE

14.1 Subpackages

14.1.1 easycv.datasets.classification package

class easycv.datasets.classification.**ClsDataset**(*data_source*, *pipeline*)

Bases: Generic[torch.utils.data.dataset.T_co]

Dataset for classification

Parameters

- **data_source** – data source to parse input data
- **pipeline** – transforms list

__init__(*data_source*, *pipeline*)

Initialize self. See help(type(self)) for accurate signature.

evaluate(*results*, *evaluators*, *logger=None*, *topk=(1, 5)*)

evaluate classification task

Parameters

- **results** – a dict of list of tensor, including prediction and groundtruth info, where prediction tensor is NxCand the same with groundtruth labels.
- **evaluators** – a list of evaluator

Returns a dict of float, different metric values

Return type eval_result

visualize(*results*, *vis_num=10*, ***kwargs*)

Visulaize the model output on validation data. :param results: A dictionary containing

class: List of length number of test images. img_metas: List of length number of test images,

dict of image meta info, containing filename, img_shape, origin_img_shape and so on.

Parameters **vis_num** – number of images visualized

Returns: A dictionary containing images: Visulaized images, list of np.ndarray. img_metas: List of length number of test images,

dict of image meta info, containing filename, img_shape, origin_img_shape and so on.

class easycv.datasets.classification.**ClsOdpsDataset**(*data_source*, *pipeline*, *image_key='url_image'*, *label_key='label'*, ***kwargs*)

Bases: Generic[torch.utils.data.dataset.T_co]

Dataset for rotation prediction

__init__(*data_source*, *pipeline*, *image_key='url_image'*, *label_key='label'*, ***kwargs*)
Initialize self. See help(type(self)) for accurate signature.

evaluate(*results*, *evaluators*, *logger=None*)

Subpackages

easycv.datasets.classification.data_sources package

class easycv.datasets.classification.data_sources.**ClsSourceCifar10**(*root*, *split*)

Bases: object

CLASSES = ['airplane', 'automobile', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', 'truck']

__init__(*root*, *split*)
Initialize self. See help(type(self)) for accurate signature.

get_length()

get_sample(*idx*)

class easycv.datasets.classification.data_sources.**ClsSourceCifar100**(*root*, *split*)

Bases: object

CLASSES = None

__init__(*root*, *split*)
Initialize self. See help(type(self)) for accurate signature.

get_length()

get_sample(*idx*)

class easycv.datasets.classification.data_sources.**ClsSourceImageListByClass**(*root*, *list_file*, *m_per_class=2*, *delimeter=' '*, *split_huge_listfile_byrank=False*, *cache_path='data/'*, *max_try=20*)

Bases: object

Get the same *m_per_class* samples by the label idx.

Parameters

- **list_file** – str / list(str), str means a input image list file path, this file contains records as *image_path label* in list_file list(str) means multi image list, each one contains some records as *image_path label*
- **root** – str / list(str), root path for image_path, each list_file will need a root.
- **m_per_class** – num of samples for each class.
- **delimeter** – str, delimeter of each line in the *list_file*

- **split_huge_listfile_byrank** – Adapt to the situation that the memory cannot fully load a huge amount of data list. If split, data list will be split to each rank.
- **cache_path** – if *split_huge_listfile_byrank* is true, cache list_file will be saved to cache_path.
- **max_try** – int, max try numbers of reading image

__init__(*root*, *list_file*, *m_per_class=2*, *delimeter=' '*, *split_huge_listfile_byrank=False*, *cache_path='data/'*, *max_try=20*)
Initialize self. See help(type(self)) for accurate signature.

get_length()

get_sample(*idx*)

class easycv.datasets.classification.data_sources.**ClsSourceImageList**(*list_file*, *root=''*, *delimeter=' '*, *split_huge_listfile_byrank=False*, *split_label_balance=False*, *cache_path='data/'*, *max_try=20*)

Bases: object

data source for classification

Parameters

- **list_file** – str / list(str), str means a input image list file path, this file contains records as *image_path label* in list_file list(str) means multi image list, each one contains some records as *image_path label*
- **root** – str / list(str), root path for image_path, each list_file will need a root, if len(root) < len(list_file), we will use root[-1] to fill root list.
- **delimeter** – str, delimeter of each line in the *list_file*
- **split_huge_listfile_byrank** – Adapt to the situation that the memory cannot fully load a huge amount of data list. If split, data list will be split to each rank.
- **split_label_balance** – if *split_huge_listfile_byrank* is true, whether split with label balance
- **cache_path** – if *split_huge_listfile_byrank* is true, cache list_file will be saved to cache_path.
- **max_try** – int, max try numbers of reading image

__init__(*list_file*, *root=''*, *delimeter=' '*, *split_huge_listfile_byrank=False*, *split_label_balance=False*, *cache_path='data/'*, *max_try=20*)
Initialize self. See help(type(self)) for accurate signature.

static parse_list_file(*list_file*, *root*, *delimeter*)

get_length()

get_sample(*idx*)

class easycv.datasets.classification.data_sources.**ClsSourceImageNetTFRecord**(*list_file=''*, *root=''*, *file_pattern=None*, *cache_path='data/cache/'*, *max_try=10*)

Bases: object

data source for imagenet tfrecord.

__init__(*list_file=''*, *root=''*, *file_pattern=None*, *cache_path='data/cache/'*, *max_try=10*)
Initialize self. See help(type(self)) for accurate signature.

class easycv.datasets.classification.data_sources.**ClsSourceCUB**(**args*, *ann_file*, *image_class_labels_file*, *train_test_split_file*, *test_mode*, *data_prefix*, ***kwargs*)

Bases: object

The CUB-200-2011 Dataset. Support the CUB-200-2011 Dataset. Comparing with the CUB-200 Dataset, there are much more pictures in *CUB-200-2011*. :param ann_file: the annotation file.

images.txt in CUB.

Parameters

- **image_class_labels_file** (*str*) – the label file. image_class_labels.txt in CUB.
- **train_test_split_file** (*str*) – the split file. train_test_split_file.txt in CUB.

```
CLASSES = ['Black_footed_Albatross', 'Laysan_Albatross', 'Sooty_Albatross',
'Groove_billed_Ani', 'Crested_Auklet', 'Least_Auklet', 'Parakeet_Auklet',
'Rhinoceros_Auklet', 'Brewer_Blackbird', 'Red_winged_Blackbird', 'Rusty_Blackbird',
'Yellow_headed_Blackbird', 'Bobolink', 'Indigo_Bunting', 'Lazuli_Bunting',
'Painted_Bunting', 'Cardinal', 'Spotted_Catbird', 'Gray_Catbird',
'Yellow_breasted_Chat', 'Eastern_Towhee', 'Chuck_will_Widow', 'Brandt_Cormorant',
'Red_faced_Cormorant', 'Pelagic_Cormorant', 'Bronzed_Cowbird', 'Shiny_Cowbird',
'Brown_Creeper', 'American_Crow', 'Fish_Crow', 'Black_billed_Cuckoo',
'Mangrove_Cuckoo', 'Yellow_billed_Cuckoo', 'Gray_crowned_Rosy_Finch',
'Purple_Finch', 'Northern_Flicker', 'Acadian_Flycatcher',
'Great_Crested_Flycatcher', 'Least_Flycatcher', 'Olive_sided_Flycatcher',
'Scissor_tailed_Flycatcher', 'Vermilion_Flycatcher', 'Yellow_bellied_Flycatcher',
'Frigatebird', 'Northern_Fulmar', 'Gadwall', 'American_Goldfinch',
'European_Goldfinch', 'Boat_tailed_Grackle', 'Eared_Grebe', 'Horned_Grebe',
'Pied_billed_Grebe', 'Western_Grebe', 'Blue_Grosbeak', 'Evening_Grosbeak',
'Pine_Grosbeak', 'Rose_breasted_Grosbeak', 'Pigeon_Guillemot', 'California_Gull',
'Glaucous_winged_Gull', 'Heermann_Gull', 'Herring_Gull', 'Ivory_Gull',
'Ring_billed_Gull', 'Slaty_backed_Gull', 'Western_Gull', 'Anna_Hummingbird',
'Ruby_throated_Hummingbird', 'Rufous_Hummingbird', 'Green_Violetear',
'Long_tailed_Jaeger', 'Pomarine_Jaeger', 'Blue_Jay', 'Florida_Jay', 'Green_Jay',
'Dark_eyed_Junco', 'Tropical_Kingbird', 'Gray_Kingbird', 'Belted_Kingfisher',
'Green_Kingfisher', 'Pied_Kingfisher', 'Ringed_Kingfisher',
'White_breasted_Kingfisher', 'Red_legged_Kittiwake', 'Horned_Lark', 'Pacific_Loon',
'Mallard', 'Western_Meadowlark', 'Hooded_Merganser', 'Red_breasted_Merganser',
'Mockingbird', 'Nighthawk', 'Clark_Nutcracker', 'White_breasted_Nuthatch',
'Baltimore_Oriole', 'Hooded_Oriole', 'Orchard_Oriole', 'Scott_Oriole', 'Ovenbird',
'Brown_Pelican', 'White_Pelican', 'Western_Wood_Pewee', 'Sayornis',
'American_Pipit', 'Whip_poor_Will', 'Horned_Puffin', 'Common_Raven',
'White_necked_Raven', 'American_Redstart', 'Geococcyx', 'Loggerhead_Shrike',
'Great_Grey_Shrike', 'Baird_Sparrow', 'Black_throated_Sparrow', 'Brewer_Sparrow',
'Chipping_Sparrow', 'Clay_colored_Sparrow', 'House_Sparrow', 'Field_Sparrow',
'Fox_Sparrow', 'Grasshopper_Sparrow', 'Harris_Sparrow', 'Henslow_Sparrow',
'Le_Conte_Sparrow', 'Lincoln_Sparrow', 'Nelson_Sharp_tailed_Sparrow',
'Savannah_Sparrow', 'Seaside_Sparrow', 'Song_Sparrow', 'Tree_Sparrow',
'Vesper_Sparrow', 'White_crowned_Sparrow', 'White_throated_Sparrow',
'Cape_Glossy_Starling', 'Bank_Swallow', 'Barn_Swallow', 'Cliff_Swallow',
'Tree_Swallow', 'Scarlet_Tanager', 'Summer_Tanager', 'Artic_Tern', 'Black_Tern',
'Caspian_Tern', 'Common_Tern', 'Elegant_Tern', 'Forsters_Tern', 'Least_Tern',
'Green_tailed_Towhee', 'Brown_Thrasher', 'Sage_Thrasher', 'Black_capped_Vireo',
'Blue_headed_Vireo', 'Philadelphia_Vireo', 'Red_eyed_Vireo', 'Warbling_Vireo',
'White_eyed_Vireo', 'Yellow_throated_Vireo', 'Bay_breasted_Warbler',
'Black_and_white_Warbler', 'Black_throated_Blue_Warbler', 'Blue_winged_Warbler',
'Canada_Warbler', 'Cape_May_Warbler', 'Cerulean_Warbler', 'Chestnut_sided_Warbler',
'Golden_winged_Warbler', 'Hooded_Warbler', 'Kentucky_Warbler', 'Magnolia_Warbler',
'Mourning_Warbler', 'Myrtle_Warbler', 'Nashville_Warbler', 'Orange_crowned_Warbler',
'Palm_Warbler', 'Pine_Warbler', 'Prairie_Warbler', 'Prothonotary_Warbler',
'Swainson_Warbler', 'Tennessee_Warbler', 'Wilson_Warbler', 'Worm_eating_Warbler',
'Yellow_Warbler', 'Northern_Waterthrush', 'Louisiana_Waterthrush',
'Bohemian_Waxwing', 'Cedar_Waxwing', 'American_Three_toed_Woodpecker',
'Pileated_Woodpecker', 'Red_bellied_Woodpecker', 'Red_cockaded_Woodpecker',
'Red_headed_Woodpecker', 'Downy_Woodpecker', 'Bewick_Wren', 'Cactus_Wren',
'Carolina_Wren', 'House_Wren', 'Marsh_Wren', 'Rock_Wren', 'Winter_Wren',
'Common_Yellowthroat']
```

__init__(**args*, *ann_file*, *image_class_labels_file*, *train_test_split_file*, *test_mode*, *data_prefix*, ***kwargs*)
Initialize self. See help(type(self)) for accurate signature.

load_annotations()

get_length()

get_sample(*idx*)

Submodules

easycv.datasets.classification.data_sources.cifar module

class easycv.datasets.classification.data_sources.cifar.ClsSourceCifar10(*root*, *split*)
Bases: object

CLASSES = ['airplane', 'automobile', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', 'truck']

__init__(*root*, *split*)
Initialize self. See help(type(self)) for accurate signature.

get_length()

get_sample(*idx*)

class easycv.datasets.classification.data_sources.cifar.ClsSourceCifar100(*root*, *split*)
Bases: object

CLASSES = None

__init__(*root*, *split*)
Initialize self. See help(type(self)) for accurate signature.

get_length()

get_sample(*idx*)

easycv.datasets.classification.data_sources.class_list module

class easycv.datasets.classification.data_sources.class_list.ClsSourceImageListByClass(*root*, *list_file*, *m_per_class=2*, *delimeter=' '*, *split_huge_listfile_by_class=False*, *cache_path='data/'*, *max_try=20*)
Bases: object

Get the same *m_per_class* samples by the label idx.

Parameters

- **list_file** – str / list(str), str means a input image list file path, this file contains records as *image_path label* in list_file list(str) means multi image list, each one contains some records as *image_path label*

- **root** – str / list(str), root path for image_path, each list_file will need a root.
- **m_per_class** – num of samples for each class.
- **delimeter** – str, delimeter of each line in the *list_file*
- **split_huge_listfile_byrank** – Adapt to the situation that the memory cannot fully load a huge amount of data list. If split, data list will be split to each rank.
- **cache_path** – if *split_huge_listfile_byrank* is true, cache list_file will be saved to cache_path.
- **max_try** – int, max try numbers of reading image

__init__(*root*, *list_file*, *m_per_class=2*, *delimeter=' '*, *split_huge_listfile_byrank=False*, *cache_path='data/'*, *max_try=20*)
Initialize self. See help(type(self)) for accurate signature.

get_length()

get_sample(*idx*)

easycv.datasets.classification.data_sources.fashiongen_h5 module

class easycv.datasets.classification.data_sources.fashiongen_h5.**FashionGenH5**(*h5file_path*, *return_label=True*, *cache_path='data/fashionGenH5'*)

Bases: object

__init__(*h5file_path*, *return_label=True*, *cache_path='data/fashionGenH5'*)
Initialize self. See help(type(self)) for accurate signature.

get_length()

get_sample(*idx*)

easycv.datasets.classification.data_sources.image_list module

class easycv.datasets.classification.data_sources.image_list.**ClsSourceImageList**(*list_file*, *root=''*, *delimeter=' '*, *split_huge_listfile_byrank=False*, *split_label_balance=False*, *cache_path='data/'*, *max_try=20*)

Bases: object

data source for classification

Parameters

- **list_file** – str / list(str), str means a input image list file path, this file contains records as *image_path label* in list_file list(str) means multi image list, each one contains some records as *image_path label*
- **root** – str / list(str), root path for image_path, each list_file will need a root, if len(root) < len(list_file), we will use root[-1] to fill root list.
- **delimeter** – str, delimeter of each line in the *list_file*

- **split_huge_listfile_byrank** – Adapt to the situation that the memory cannot fully load a huge amount of data list. If split, data list will be split to each rank.
- **split_label_balance** – if *split_huge_listfile_byrank* is true, whether split with label balance
- **cache_path** – if *split_huge_listfile_byrank* is true, cache list_file will be saved to cache_path.
- **max_try** – int, max try numbers of reading image

__init__(*list_file*, *root=''*, *delimeter=' '*, *split_huge_listfile_byrank=False*, *split_label_balance=False*, *cache_path='data/'*, *max_try=20*)
Initialize self. See help(type(self)) for accurate signature.

static parse_list_file(*list_file*, *root*, *delimeter*)

get_length()

get_sample(*idx*)

easycv.datasets.classification.data_sources.imagenet_tfrecord module

class easycv.datasets.classification.data_sources.imagenet_tfrecord.**ClsSourceImageNetTFRecord**(*list_file=''*, *root=''*, *file_pattern=*, *cache_path=*, *max_try=10*)

Bases: object

data source for imagenet tfrecord.

__init__(*list_file=''*, *root=''*, *file_pattern=None*, *cache_path='data/cache/'*, *max_try=10*)
Initialize self. See help(type(self)) for accurate signature.

easycv.datasets.classification.data_sources.utils module

easycv.datasets.classification.data_sources.utils.**split_listfile_byrank**(*list_file*, *label_balance*, *save_path='data/'*, *delimeter=' '*)

easycv.datasets.classification.pipelines package

class easycv.datasets.classification.pipelines.**MMAutoAugment**(*policies=[[{'type': 'Posterize', 'bits': 4, 'prob': 0.4}, {'type': 'Rotate', 'angle': 30.0, 'prob': 0.6}], [{'type': 'Solarize', 'thr': 113.77777777777777, 'prob': 0.6}, {'type': 'AutoContrast', 'prob': 0.6}], [{'type': 'Equalize', 'prob': 0.8}, {'type': 'Equalize', 'prob': 0.6}], [{'type': 'Posterize', 'bits': 5, 'prob': 0.6}, {'type': 'Posterize', 'bits': 5, 'prob': 0.6}], [{'type': 'Equalize', 'prob': 0.4}, {'type': 'Solarize', 'thr': 142.22222222222223, 'prob': 0.2}], [{'type': 'Equalize', 'prob': 0.4}, {'type': 'Rotate', 'angle': 26.666666666666668, 'prob': 0.8}], [{'type': 'Solarize', 'thr': 170.66666666666666, 'prob': 0.6}, {'type': 'Equalize', 'prob': 0.6}], [{'type': 'Posterize', 'bits': 6, 'prob': 0.8}, {'type': 'Equalize', 'prob': 1.0}], [{'type': 'Rotate', 'angle': 10.0, 'prob': 0.2}, {'type': 'Solarize', 'thr': 28.444444444444443, 'prob': 0.6}], [{'type': 'Equalize', 'prob': 0.6}, {'type': 'Posterize', 'bits': 5, 'prob': 0.4}], [{'type': 'Rotate', 'angle': 26.666666666666668, 'prob': 0.8}, {'type': 'ColorTransform', 'magnitude': 0.0, 'prob': 0.4}], [{'type': 'Rotate', 'angle': 30.0, 'prob': 0.4}, {'type': 'Equalize', 'prob': 0.6}], [{'type': 'Equalize', 'prob': 0.0}, {'type': 'Equalize', 'prob': 0.8}], [{'type': 'Invert', 'prob': 0.6}, {'type': 'Equalize', 'prob': 1.0}], [{'type': 'ColorTransform', 'magnitude': 0.4, 'prob': 0.6}, {'type': 'Contrast', 'magnitude': 0.8, 'prob': 1.0}], [{'type': 'Rotate', 'angle': 26.666666666666668, 'prob': 0.8}, {'type': 'ColorTransform', 'magnitude': 0.2, 'prob': 1.0}], [{'type': 'ColorTransform', 'magnitude': 0.8, 'prob': 0.8}, {'type': 'Solarize', 'thr': 56.888888888888886, 'prob': 0.8}], [{'type': 'Sharpness', 'magnitude': 0.7, 'prob': 0.4}, {'type': 'Invert', 'prob': 0.6}], [{'type': 'Shear', 'magnitude': 0.16666666666666666, 'prob': 0.6, 'direction': 'horizontal'}, {'type': 'Equalize', 'prob': 1.0}], [{'type': 'ColorTransform', 'magnitude': 0.0, 'prob': 0.4}, {'type': 'Equalize', 'prob': 0.6}], [{'type': 'Equalize', 'prob': 0.4}, {'type': 'Solarize', 'thr':*

Auto augmentation. This data augmentation is proposed in AutoAugment: Learning Augmentation Policies from Data. :param policies: The policies of auto augmentation. Each

policy in `policies` is a specific augmentation policy, and is composed by several augmentations (dict). When AutoAugment is called, a random policy in `policies` will be selected to augment images.

Parameters **`hparams`** (*`dict`*) – Configs of hyperparameters. Hyperparameters will be used in policies that require these arguments if these arguments are not set in policy dicts. Defaults to use _HPARAMS_DEFAULT.

`__init__`(*policies=[[{'type': 'Posterize', 'bits': 4, 'prob': 0.4}, {'type': 'Rotate', 'angle': 30.0, 'prob': 0.6}], [{'type': 'Solarize', 'thr': 113.77777777777777, 'prob': 0.6}, {'type': 'AutoContrast', 'prob': 0.6}], [{'type': 'Equalize', 'prob': 0.8}, {'type': 'Equalize', 'prob': 0.6}], [{'type': 'Posterize', 'bits': 5, 'prob': 0.6}, {'type': 'Posterize', 'bits': 5, 'prob': 0.6}], [{'type': 'Equalize', 'prob': 0.4}, {'type': 'Solarize', 'thr': 142.22222222222223, 'prob': 0.2}], [{'type': 'Equalize', 'prob': 0.4}, {'type': 'Rotate', 'angle': 26.666666666666668, 'prob': 0.8}], [{'type': 'Solarize', 'thr': 170.66666666666666, 'prob': 0.6}, {'type': 'Equalize', 'prob': 0.6}], [{'type': 'Posterize', 'bits': 6, 'prob': 0.8}, {'type': 'Equalize', 'prob': 1.0}], [{'type': 'Rotate', 'angle': 10.0, 'prob': 0.2}, {'type': 'Solarize', 'thr': 28.444444444444443, 'prob': 0.6}], [{'type': 'Equalize', 'prob': 0.6}, {'type': 'Posterize', 'bits': 5, 'prob': 0.4}], [{'type': 'Rotate', 'angle': 26.666666666666668, 'prob': 0.8}, {'type': 'ColorTransform', 'magnitude': 0.0, 'prob': 0.4}], [{'type': 'Rotate', 'angle': 30.0, 'prob': 0.4}, {'type': 'Equalize', 'prob': 0.6}], [{'type': 'Equalize', 'prob': 0.0}, {'type': 'Equalize', 'prob': 0.8}], [{'type': 'Invert', 'prob': 0.6}, {'type': 'Equalize', 'prob': 1.0}], [{'type': 'ColorTransform', 'magnitude': 0.4, 'prob': 0.6}, {'type': 'Contrast', 'magnitude': 0.8, 'prob': 1.0}], [{'type': 'Rotate', 'angle': 26.666666666666668, 'prob': 0.8}, {'type': 'ColorTransform', 'magnitude': 0.2, 'prob': 1.0}], [{'type': 'ColorTransform', 'magnitude': 0.8, 'prob': 0.8}, {'type': 'Solarize', 'thr': 56.888888888888886, 'prob': 0.8}], [{'type': 'Sharpness', 'magnitude': 0.7, 'prob': 0.4}, {'type': 'Invert', 'prob': 0.6}], [{'type': 'Shear', 'magnitude': 0.16666666666666666, 'prob': 0.6, 'direction': 'horizontal'}, {'type': 'Equalize', 'prob': 1.0}], [{'type': 'ColorTransform', 'magnitude': 0.0, 'prob': 0.4}, {'type': 'Equalize', 'prob': 0.6}], [{'type': 'Equalize', 'prob': 0.4}, {'type': 'Solarize', 'thr': 142.22222222222223, 'prob': 0.2}], [{'type': 'Solarize', 'thr': 113.77777777777777, 'prob': 0.6}, {'type': 'AutoContrast', 'prob': 0.6}], [{'type': 'Invert', 'prob': 0.6}, {'type': 'Equalize', 'prob': 1.0}], [{'type': 'ColorTransform', 'magnitude': 0.4, 'prob': 0.6}, {'type': 'Contrast', 'magnitude': 0.8, 'prob': 1.0}], [{'type': 'Equalize', 'prob': 0.8}, {'type': 'Equalize', 'prob': 0.6}]]*, *hparams={'pad_val': 128}*)

Initialize self. See help(type(self)) for accurate signature.

class easycv.datasets.classification.pipelines.**MMRandAugment**(*num_policies*, *magnitude_level*, *magnitude_std=0.0*, *total_level=30*, *policies=[{'type': 'AutoContrast'}, {'type': 'Equalize'}, {'type': 'Invert'}, {'type': 'Rotate', 'magnitude_key': 'angle', 'magnitude_range': (0, 30)}, {'type': 'Posterize', 'magnitude_key': 'bits', 'magnitude_range': (4, 0)}, {'type': 'Solarize', 'magnitude_key': 'thr', 'magnitude_range': (256, 0)}, {'type': 'SolarizeAdd', 'magnitude_key': 'magnitude', 'magnitude_range': (0, 110)}, {'type': 'ColorTransform', 'magnitude_key': 'magnitude', 'magnitude_range': (0, 0.9)}, {'type': 'Contrast', 'magnitude_key': 'magnitude', 'magnitude_range': (0, 0.9)}, {'type': 'Brightness', 'magnitude_key': 'magnitude', 'magnitude_range': (0, 0.9)}, {'type': 'Sharpness', 'magnitude_key': 'magnitude', 'magnitude_range': (0, 0.9)}, {'type': 'Shear', 'magnitude_key': 'magnitude', 'magnitude_range': (0, 0.3), 'direction': 'horizontal'}, {'type': 'Shear', 'magnitude_key': 'magnitude', 'magnitude_range': (0, 0.3), 'direction': 'vertical'}, {'type': 'Translate', 'magnitude_key': 'magnitude', 'magnitude_range': (0, 0.45), 'direction': 'horizontal'}, {'type': 'Translate', 'magnitude_key': 'magnitude', 'magnitude_range': (0, 0.45), 'direction': 'vertical'}]*, *hparams={'pad_val': 128}*)

Bases: `object`

Random augmentation. This data augmentation is proposed in RandAugment: Practical automated data augmentation with a reduced search space. :param policies: The policies of random augmentation. Each

policy in `policies` is one specific augmentation policy (dict). The policy shall at least have key *type*, indicating the type of augmentation. For those which have magnitude, (given to the fact they are named differently in different augmentation,) *magnitude_key* and *magnitude_range* shall be the magnitude argument (str) and the range of magnitude (tuple in the format of (val1, val2)), respectively. Note that val1 is not necessarily less than val2.

Parameters

- **num_policies** (*int*) – Number of policies to select from policies each time.
- **magnitude_level** (*int | float*) – Magnitude level for all the augmentation selected.
- **total_level** (*int | float*) – Total level for the magnitude. Defaults to 30.
- **magnitude_std** (*Number | str*) – Deviation of magnitude noise applied. - If positive number, magnitude is sampled from normal distribution

(mean=magnitude, std=magnitude_std).

- If 0 or negative number, magnitude remains unchanged.
- If str "inf", magnitude is sampled from uniform distribution (range=[min, magnitude]).

- **hparams** (*dict*) – Configs of hyperparameters. Hyperparameters will be used in policies that require these arguments if these arguments are not set in policy dicts. Defaults to use _HPARAMS_DEFAULT.

Note: *magnitude_std* will introduce some randomness to policy, modified by https://github.com/rwightman/pytorch-image-models. When magnitude_std=0, we calculate the magnitude as follows: .. math:

```
\text{magnitude} = \frac{\text{magnitude\_level}}
{\text{total\_level}} \times (\text{val2} - \text{val1})
+ \text{val1}
```

__init__(*num_policies*, *magnitude_level*, *magnitude_std=0.0*, *total_level=30*, *policies=[{'type': 'AutoContrast'}, {'type': 'Equalize'}, {'type': 'Invert'}, {'type': 'Rotate', 'magnitude_key': 'angle', 'magnitude_range': (0, 30)}, {'type': 'Posterize', 'magnitude_key': 'bits', 'magnitude_range': (4, 0)}, {'type': 'Solarize', 'magnitude_key': 'thr', 'magnitude_range': (256, 0)}, {'type': 'SolarizeAdd', 'magnitude_key': 'magnitude', 'magnitude_range': (0, 110)}, {'type': 'ColorTransform', 'magnitude_key': 'magnitude', 'magnitude_range': (0, 0.9)}, {'type': 'Contrast', 'magnitude_key': 'magnitude', 'magnitude_range': (0, 0.9)}, {'type': 'Brightness', 'magnitude_key': 'magnitude', 'magnitude_range': (0, 0.9)}, {'type': 'Sharpness', 'magnitude_key': 'magnitude', 'magnitude_range': (0, 0.9)}, {'type': 'Shear', 'magnitude_key': 'magnitude', 'magnitude_range': (0, 0.3), 'direction': 'horizontal'}, {'type': 'Shear', 'magnitude_key': 'magnitude', 'magnitude_range': (0, 0.3), 'direction': 'vertical'}, {'type': 'Translate', 'magnitude_key': 'magnitude', 'magnitude_range': (0, 0.45), 'direction': 'horizontal'}, {'type': 'Translate', 'magnitude_key': 'magnitude', 'magnitude_range': (0, 0.45), 'direction': 'vertical'}]*, *hparams={'pad_val': 128}*)

Initialize self. See help(type(self)) for accurate signature.

class easycv.datasets.classification.pipelines.**MMRandomErasing**(*erase_prob=0.5*, *min_area_ratio=0.02*, *max_area_ratio=0.4*, *aspect_range=(0.3, 3.3333333333333335)*, *mode='const'*, *fill_color=(128, 128, 128)*, *fill_std=None*)

Bases: object

Randomly selects a rectangle region in an image and erase pixels. :param erase_prob: Probability that image will be randomly erased.

Default: 0.5

Parameters

- **min_area_ratio** (*float*) – Minimum erased area / input image area Default: 0.02
- **max_area_ratio** (*float*) – Maximum erased area / input image area Default: 0.4
- **aspect_range** (*sequence | float*) – Aspect ratio range of erased area. if float, it will be converted to (aspect_ratio, 1/aspect_ratio) Default: (3/10, 10/3)

- **mode** (*str*) – Fill method in erased area, can be: - const (default): All pixels are assign with the same value. - rand: each pixel is assigned with a random value in [0, 255]
- **fill_color** (*sequence | Number*) – Base color filled in erased area. Defaults to (128, 128, 128).
- **fill_std** (*sequence | Number, optional*) – If set and mode is 'rand', fill erased area with random color from normal distribution (mean=fill_color, std=fill_std); If not set, fill erased area with random color from uniform distribution (0~255). Defaults to None.

Note: See Random Erasing Data Augmentation This paper provided 4 modes: RE-R, RE-M, RE-0, RE-255, and use RE-M as default. The config of these 4 modes are: - RE-R: RandomErasing(mode='rand') - RE-M: RandomErasing(mode='const', fill_color=(123.67, 116.3, 103.5)) - RE-0: RandomErasing(mode='const', fill_color=0) - RE-255: RandomErasing(mode='const', fill_color=255)

__init__(*erase_prob=0.5*, *min_area_ratio=0.02*, *max_area_ratio=0.4*, *aspect_range=(0.3, 3.333333333333335)*, *mode='const'*, *fill_color=(128, 128, 128)*, *fill_std=None*)
Initialize self. See help(type(self)) for accurate signature.

Submodules

easycv.datasets.classification.pipelines.auto_augment module

easycv.datasets.classification.pipelines.auto_augment.**random_negative**(*value*, *random_negative_prob*)
Randomly negate value based on random_negative_prob.

easycv.datasets.classification.pipelines.auto_augment.**merge_hparams**(*policy: dict*, *hparams: dict*)
Merge hyperparameters into policy config. Only merge partial hyperparameters required of the policy. :param policy: Original policy config dict. :type policy: dict :param hparams: Hyperparameters need to be merged. :type hparams: dict

Returns Policy config dict after adding hparams.

Return type dict

class easycv.datasets.classification.pipelines.auto_augment.**MMAutoAugment**(*policies=[[{'type': 'Posterize', 'bits': 4, 'prob': 0.4}, {'type': 'Rotate', 'angle': 30.0, 'prob': 0.6}], [{'type': 'Solarize', 'thr': 113.77777777777777, 'prob': 0.6}, {'type': 'AutoContrast', 'prob': 0.6}], [{'type': 'Equalize', 'prob': 0.8}, {'type': 'Equalize', 'prob': 0.6}], [{'type': 'Posterize', 'bits': 5, 'prob': 0.6}, {'type': 'Posterize', 'bits': 5, 'prob': 0.6}], [{'type': 'Equalize', 'prob': 0.4}, {'type': 'Solarize', 'thr': 142.22222222222223, 'prob': 0.2}], [{'type': 'Equalize', 'prob': 0.4}, {'type': 'Rotate', 'angle': 26.666666666666668, 'prob': 0.8}], [{'type': 'Solarize', 'thr': 170.66666666666666, 'prob': 0.6}, {'type': 'Equalize', 'prob': 0.6}], [{'type': 'Posterize', 'bits': 6, 'prob': 0.8}, {'type': 'Equalize', 'prob': 1.0}], [{'type': 'Rotate', 'angle': 10.0, 'prob': 0.2}, {'type': 'Solarize', 'thr': 28.444444444444443, 'prob': 0.6}], [{'type': 'Equalize', 'prob': 0.6}, {'type': 'Posterize', 'bits': 5, 'prob': 0.4}], [{'type': 'Rotate', 'angle': 26.666666666666668, 'prob': 0.8}, {'type': 'ColorTransform', 'magnitude': 0.0, 'prob': 0.4}], [{'type': 'Rotate', 'angle': 30.0, 'prob': 0.4}, {'type': 'Equalize', 'prob':*

Auto augmentation. This data augmentation is proposed in AutoAugment: Learning Augmentation Policies from Data. :param policies: The policies of auto augmentation. Each

policy in `policies` is a specific augmentation policy, and is composed by several augmentations (dict). When AutoAugment is called, a random policy in `policies` will be selected to augment images.

Parameters **`hparams`** (*`dict`*) – Configs of hyperparameters. Hyperparameters will be used in policies that require these arguments if these arguments are not set in policy dicts. Defaults to use _HPARAMS_DEFAULT.

`__init__`(*policies=[[{'type': 'Posterize', 'bits': 4, 'prob': 0.4}, {'type': 'Rotate', 'angle': 30.0, 'prob': 0.6}], [{'type': 'Solarize', 'thr': 113.77777777777777, 'prob': 0.6}, {'type': 'AutoContrast', 'prob': 0.6}], [{'type': 'Equalize', 'prob': 0.8}, {'type': 'Equalize', 'prob': 0.6}], [{'type': 'Posterize', 'bits': 5, 'prob': 0.6}, {'type': 'Posterize', 'bits': 5, 'prob': 0.6}], [{'type': 'Equalize', 'prob': 0.4}, {'type': 'Solarize', 'thr': 142.22222222222223, 'prob': 0.2}], [{'type': 'Equalize', 'prob': 0.4}, {'type': 'Rotate', 'angle': 26.666666666666668, 'prob': 0.8}], [{'type': 'Solarize', 'thr': 170.66666666666666, 'prob': 0.6}, {'type': 'Equalize', 'prob': 0.6}], [{'type': 'Posterize', 'bits': 6, 'prob': 0.8}, {'type': 'Equalize', 'prob': 1.0}], [{'type': 'Rotate', 'angle': 10.0, 'prob': 0.2}, {'type': 'Solarize', 'thr': 28.444444444444443, 'prob': 0.6}], [{'type': 'Equalize', 'prob': 0.6}, {'type': 'Posterize', 'bits': 5, 'prob': 0.4}], [{'type': 'Rotate', 'angle': 26.666666666666668, 'prob': 0.8}, {'type': 'ColorTransform', 'magnitude': 0.0, 'prob': 0.4}], [{'type': 'Rotate', 'angle': 30.0, 'prob': 0.4}, {'type': 'Equalize', 'prob': 0.6}], [{'type': 'Equalize', 'prob': 0.0}, {'type': 'Equalize', 'prob': 0.8}], [{'type': 'Invert', 'prob': 0.6}, {'type': 'Equalize', 'prob': 1.0}], [{'type': 'ColorTransform', 'magnitude': 0.4, 'prob': 0.6}, {'type': 'Contrast', 'magnitude': 0.8, 'prob': 1.0}], [{'type': 'Rotate', 'angle': 26.666666666666668, 'prob': 0.8}, {'type': 'ColorTransform', 'magnitude': 0.2, 'prob': 1.0}], [{'type': 'ColorTransform', 'magnitude': 0.8, 'prob': 0.8}, {'type': 'Solarize', 'thr': 56.888888888888886, 'prob': 0.8}], [{'type': 'Sharpness', 'magnitude': 0.7, 'prob': 0.4}, {'type': 'Invert', 'prob': 0.6}], [{'type': 'Shear', 'magnitude': 0.16666666666666666, 'prob': 0.6, 'direction': 'horizontal'}, {'type': 'Equalize', 'prob': 1.0}], [{'type': 'ColorTransform', 'magnitude': 0.0, 'prob': 0.4}, {'type': 'Equalize', 'prob': 0.6}], [{'type': 'Equalize', 'prob': 0.4}, {'type': 'Solarize', 'thr': 142.22222222222223, 'prob': 0.2}], [{'type': 'Solarize', 'thr': 113.77777777777777, 'prob': 0.6}, {'type': 'AutoContrast', 'prob': 0.6}], [{'type': 'Invert', 'prob': 0.6}, {'type': 'Equalize', 'prob': 1.0}], [{'type': 'ColorTransform', 'magnitude': 0.4, 'prob': 0.6}, {'type': 'Contrast', 'magnitude': 0.8, 'prob': 1.0}], [{'type': 'Equalize', 'prob': 0.8}, {'type': 'Equalize', 'prob': 0.6}]]*, *hparams={'pad_val': 128}*)

Initialize self. See help(type(self)) for accurate signature.

class easycv.datasets.classification.pipelines.auto_augment.**MMRandAugment**(*num_policies*, *magnitude_level*, *magnitude_std=0.0*, *total_level=30*, *policies=[{'type': 'AutoContrast'}, {'type': 'Equalize'}, {'type': 'Invert'}, {'type': 'Rotate', 'magnitude_key': 'angle', 'magnitude_range': (0, 30)}, {'type': 'Posterize', 'magnitude_key': 'bits', 'magnitude_range': (4, 0)}, {'type': 'Solarize', 'magnitude_key': 'thr', 'magnitude_range': (256, 0)}, {'type': 'SolarizeAdd', 'magnitude_key': 'magnitude', 'magnitude_range': (0, 110)}, {'type': 'ColorTransform', 'magnitude_key': 'magnitude', 'magnitude_range': (0, 0.9)}, {'type': 'Contrast', 'magnitude_key': 'magnitude', 'magnitude_range': (0, 0.9)}, {'type': 'Brightness', 'magnitude_key': 'magnitude', 'magnitude_range': (0, 0.9)}, {'type': 'Sharpness', 'magnitude_key': 'magnitude', 'magnitude_range': (0, 0.9)}, {'type': 'Shear', 'magnitude_key': 'magnitude', 'magnitude_range': (0, 0.3), 'direction': 'horizontal'}, {'type': 'Shear', 'magnitude_key': 'magnitude', 'magnitude_range': (0, 0.3), 'direction': 'vertical'}, {'type':*

Random augmentation. This data augmentation is proposed in RandAugment: Practical automated data augmentation with a reduced search space. :param policies: The policies of random augmentation. Each

policy in `policies` is one specific augmentation policy (dict). The policy shall at least have key *type*, indicating the type of augmentation. For those which have magnitude, (given to the fact they are named differently in different augmentation,) *magnitude_key* and *magnitude_range* shall be the magnitude argument (str) and the range of magnitude (tuple in the format of (val1, val2)), respectively. Note that val1 is not necessarily less than val2.

Parameters

- **num_policies** (*int*) – Number of policies to select from policies each time.
- **magnitude_level** (*int | float*) – Magnitude level for all the augmentation selected.
- **total_level** (*int | float*) – Total level for the magnitude. Defaults to 30.
- **magnitude_std** (*Number | str*) – Deviation of magnitude noise applied. - If positive number, magnitude is sampled from normal distribution

  (mean=magnitude, std=magnitude_std).

  - If 0 or negative number, magnitude remains unchanged.
  - If str "inf", magnitude is sampled from uniform distribution (range=[min, magnitude]).

- **hparams** (*dict*) – Configs of hyperparameters. Hyperparameters will be used in policies that require these arguments if these arguments are not set in policy dicts. Defaults to use _HPARAMS_DEFAULT.

Note: *magnitude_std* will introduce some randomness to policy, modified by https://github.com/rwightman/pytorch-image-models. When magnitude_std=0, we calculate the magnitude as follows: .. math:

```
\text{magnitude} = \frac{\text{magnitude\_level}}
{\text{total\_level}} \times (\text{val2} - \text{val1})
+ \text{val1}
```

__init__(*num_policies*, *magnitude_level*, *magnitude_std=0.0*, *total_level=30*, *policies=[{'type': 'AutoContrast'}, {'type': 'Equalize'}, {'type': 'Invert'}, {'type': 'Rotate', 'magnitude_key': 'angle', 'magnitude_range': (0, 30)}, {'type': 'Posterize', 'magnitude_key': 'bits', 'magnitude_range': (4, 0)}, {'type': 'Solarize', 'magnitude_key': 'thr', 'magnitude_range': (256, 0)}, {'type': 'SolarizeAdd', 'magnitude_key': 'magnitude', 'magnitude_range': (0, 110)}, {'type': 'ColorTransform', 'magnitude_key': 'magnitude', 'magnitude_range': (0, 0.9)}, {'type': 'Contrast', 'magnitude_key': 'magnitude', 'magnitude_range': (0, 0.9)}, {'type': 'Brightness', 'magnitude_key': 'magnitude', 'magnitude_range': (0, 0.9)}, {'type': 'Sharpness', 'magnitude_key': 'magnitude', 'magnitude_range': (0, 0.9)}, {'type': 'Shear', 'magnitude_key': 'magnitude', 'magnitude_range': (0, 0.3), 'direction': 'horizontal'}, {'type': 'Shear', 'magnitude_key': 'magnitude', 'magnitude_range': (0, 0.3), 'direction': 'vertical'}, {'type': 'Translate', 'magnitude_key': 'magnitude', 'magnitude_range': (0, 0.45), 'direction': 'horizontal'}, {'type': 'Translate', 'magnitude_key': 'magnitude', 'magnitude_range': (0, 0.45), 'direction': 'vertical'}]*, *hparams={'pad_val': 128}*)

Initialize self. See help(type(self)) for accurate signature.

class easycv.datasets.classification.pipelines.auto_augment.**Shear**(*magnitude*, *pad_val=128*, *prob=0.5*, *direction='horizontal'*, *random_negative_prob=0.5*, *interpolation='bicubic'*)

Bases: object

Shear images. :param magnitude: The magnitude used for shear. :type magnitude: int | float :param pad_val: Pixel pad_val value for constant fill.

If a sequence of length 3, it is used to pad_val R, G, B channels respectively. Defaults to 128.

Parameters

- **prob** (*float*) – The probability for performing Shear therefore should be in range [0, 1]. Defaults to 0.5.
- **direction** (*str*) – The shearing direction. Options are 'horizontal' and 'vertical'. Defaults to 'horizontal'.
- **random_negative_prob** (*float*) – The probability that turns the magnitude negative, which should be in range [0,1]. Defaults to 0.5.
- **interpolation** (*str*) – Interpolation method. Options are 'nearest', 'bilinear', 'bicubic', 'area', 'lanczos'. Defaults to 'bicubic'.

__init__(*magnitude*, *pad_val=128*, *prob=0.5*, *direction='horizontal'*, *random_negative_prob=0.5*, *interpolation='bicubic'*)

Initialize self. See help(type(self)) for accurate signature.

class easycv.datasets.classification.pipelines.auto_augment.**Translate**(*magnitude*, *pad_val=128*, *prob=0.5*, *direction='horizontal'*, *random_negative_prob=0.5*, *interpolation='nearest'*)

Bases: object

Translate images. :param magnitude: The magnitude used for translate. Note that

the offset is calculated by magnitude * size in the corresponding direction. With a magnitude of 1, the whole image will be moved out of the range.

Parameters

- **pad_val** (*int, Sequence[int]*) – Pixel pad_val value for constant fill. If a sequence of length 3, it is used to pad_val R, G, B channels respectively. Defaults to 128.
- **prob** (*float*) – The probability for performing translate therefore should be in range [0, 1]. Defaults to 0.5.
- **direction** (*str*) – The translating direction. Options are 'horizontal' and 'vertical'. Defaults to 'horizontal'.
- **random_negative_prob** (*float*) – The probability that turns the magnitude negative, which should be in range [0,1]. Defaults to 0.5.
- **interpolation** (*str*) – Interpolation method. Options are 'nearest', 'bilinear', 'bicubic', 'area', 'lanczos'. Defaults to 'nearest'.

__init__(*magnitude*, *pad_val=128*, *prob=0.5*, *direction='horizontal'*, *random_negative_prob=0.5*, *interpolation='nearest'*)
Initialize self. See help(type(self)) for accurate signature.

class easycv.datasets.classification.pipelines.auto_augment.**Rotate**(*angle*, *center=None*, *scale=1.0*, *pad_val=128*, *prob=0.5*, *random_negative_prob=0.5*, *interpolation='nearest'*)
Bases: object

Rotate images. :param angle: The angle used for rotate. Positive values stand for

clockwise rotation.

Parameters

- **center** (*tuple[float], optional*) – Center point (w, h) of the rotation in the source image. If None, the center of the image will be used. Defaults to None.
- **scale** (*float*) – Isotropic scale factor. Defaults to 1.0.
- **pad_val** (*int, Sequence[int]*) – Pixel pad_val value for constant fill. If a sequence of length 3, it is used to pad_val R, G, B channels respectively. Defaults to 128.
- **prob** (*float*) – The probability for performing Rotate therefore should be in range [0, 1]. Defaults to 0.5.
- **random_negative_prob** (*float*) – The probability that turns the angle negative, which should be in range [0,1]. Defaults to 0.5.
- **interpolation** (*str*) – Interpolation method. Options are 'nearest', 'bilinear', 'bicubic', 'area', 'lanczos'. Defaults to 'nearest'.

__init__(*angle*, *center=None*, *scale=1.0*, *pad_val=128*, *prob=0.5*, *random_negative_prob=0.5*, *interpolation='nearest'*)
Initialize self. See help(type(self)) for accurate signature.

class easycv.datasets.classification.pipelines.auto_augment.**AutoContrast**(*prob=0.5*)
Bases: object

Auto adjust image contrast. :param prob: The probability for performing invert therefore should

be in range [0, 1]. Defaults to 0.5.

__init__(*prob=0.5*)
Initialize self. See help(type(self)) for accurate signature.

class easycv.datasets.classification.pipelines.auto_augment.**Invert**(*prob=0.5*)
Bases: object

Invert images. :param prob: The probability for performing invert therefore should

be in range [0, 1]. Defaults to 0.5.

__init__(*prob=0.5*)
Initialize self. See help(type(self)) for accurate signature.

class easycv.datasets.classification.pipelines.auto_augment.**Equalize**(*prob=0.5*)
Bases: object

Equalize the image histogram. :param prob: The probability for performing invert therefore should

be in range [0, 1]. Defaults to 0.5.

__init__(*prob=0.5*)
Initialize self. See help(type(self)) for accurate signature.

class easycv.datasets.classification.pipelines.auto_augment.**Solarize**(*thr*, *prob=0.5*)
Bases: object

Solarize images (invert all pixel values above a threshold). :param thr: The threshold above which the pixels value will be

inverted.

Parameters **prob** (*float*) – The probability for solarizing therefore should be in range [0, 1]. Defaults to 0.5.

__init__(*thr*, *prob=0.5*)
Initialize self. See help(type(self)) for accurate signature.

class easycv.datasets.classification.pipelines.auto_augment.**SolarizeAdd**(*magnitude*, *thr=128*, *prob=0.5*)
Bases: object

SolarizeAdd images (add a certain value to pixels below a threshold). :param magnitude: The value to be added to pixels below the thr. :type magnitude: int | float :param thr: The threshold below which the pixels value will be

adjusted.

Parameters **prob** (*float*) – The probability for solarizing therefore should be in range [0, 1]. Defaults to 0.5.

__init__(*magnitude*, *thr=128*, *prob=0.5*)
Initialize self. See help(type(self)) for accurate signature.

class easycv.datasets.classification.pipelines.auto_augment.**Posterize**(*bits*, *prob=0.5*)
Bases: object

Posterize images (reduce the number of bits for each color channel). :param bits: Number of bits for each pixel in the output img,

which should be less or equal to 8.

Parameters **prob** (*float*) – The probability for posterizing therefore should be in range [0, 1]. Defaults to 0.5.

__init__(*bits*, *prob=0.5*)
Initialize self. See help(type(self)) for accurate signature.

class easycv.datasets.classification.pipelines.auto_augment.**Contrast**(*magnitude*, *prob=0.5*, *random_negative_prob=0.5*)
Bases: object

Adjust images contrast. :param magnitude: The magnitude used for adjusting contrast. A

positive magnitude would enhance the contrast and a negative magnitude would make the image grayer. A magnitude=0 gives the origin img.

Parameters

- **prob** (*float*) – The probability for performing contrast adjusting therefore should be in range [0, 1]. Defaults to 0.5.
- **random_negative_prob** (*float*) – The probability that turns the magnitude negative, which should be in range [0,1]. Defaults to 0.5.

__init__(*magnitude*, *prob=0.5*, *random_negative_prob=0.5*)
Initialize self. See help(type(self)) for accurate signature.

class easycv.datasets.classification.pipelines.auto_augment.**ColorTransform**(*magnitude*, *prob=0.5*, *random_negative_prob=0.5*)

Bases: object

Adjust images color balance. :param magnitude: The magnitude used for color transform. A

positive magnitude would enhance the color and a negative magnitude would make the image grayer. A magnitude=0 gives the origin img.

Parameters

- **prob** (*float*) – The probability for performing ColorTransform therefore should be in range [0, 1]. Defaults to 0.5.
- **random_negative_prob** (*float*) – The probability that turns the magnitude negative, which should be in range [0,1]. Defaults to 0.5.

__init__(*magnitude*, *prob=0.5*, *random_negative_prob=0.5*)
Initialize self. See help(type(self)) for accurate signature.

class easycv.datasets.classification.pipelines.auto_augment.**Brightness**(*magnitude*, *prob=0.5*, *random_negative_prob=0.5*)

Bases: object

Adjust images brightness. :param magnitude: The magnitude used for adjusting brightness. A

positive magnitude would enhance the brightness and a negative magnitude would make the image darker. A magnitude=0 gives the origin img.

Parameters

- **prob** (*float*) – The probability for performing contrast adjusting therefore should be in range [0, 1]. Defaults to 0.5.
- **random_negative_prob** (*float*) – The probability that turns the magnitude negative, which should be in range [0,1]. Defaults to 0.5.

__init__(*magnitude*, *prob=0.5*, *random_negative_prob=0.5*)
Initialize self. See help(type(self)) for accurate signature.

class easycv.datasets.classification.pipelines.auto_augment.**Sharpness**(*magnitude*, *prob=0.5*, *random_negative_prob=0.5*)

Bases: object

Adjust images sharpness. :param magnitude: The magnitude used for adjusting sharpness. A

positive magnitude would enhance the sharpness and a negative magnitude would make the image bulr. A magnitude=0 gives the origin img.

Parameters

- **prob** (*float*) – The probability for performing contrast adjusting therefore should be in range [0, 1]. Defaults to 0.5.
- **random_negative_prob** (*float*) – The probability that turns the magnitude negative, which should be in range [0,1]. Defaults to 0.5.

__init__(*magnitude*, *prob=0.5*, *random_negative_prob=0.5*)
Initialize self. See help(type(self)) for accurate signature.

class easycv.datasets.classification.pipelines.auto_augment.**Cutout**(*shape*, *pad_val=128*, *prob=0.5*)

Bases: object

Cutout images. :param shape: Expected cutout shape (h, w).

If given as a single value, the value will be used for both h and w.

Parameters

- **pad_val** (*int, Sequence[int]*) – Pixel pad_val value for constant fill. If it is a sequence, it must have the same length with the image channels. Defaults to 128.
- **prob** (*float*) – The probability for performing cutout therefore should be in range [0, 1]. Defaults to 0.5.

__init__(*shape*, *pad_val=128*, *prob=0.5*)
Initialize self. See help(type(self)) for accurate signature.

easycv.datasets.classification.pipelines.transform module

class easycv.datasets.classification.pipelines.transform.**MMRandomErasing**(*erase_prob=0.5*, *min_area_ratio=0.02*, *max_area_ratio=0.4*, *aspect_range=(0.3, 3.3333333333333335)*, *mode='const'*, *fill_color=(128, 128, 128)*, *fill_std=None*)

Bases: object

Randomly selects a rectangle region in an image and erase pixels. :param erase_prob: Probability that image will be randomly erased.

Default: 0.5

Parameters

- **min_area_ratio** (*float*) – Minimum erased area / input image area Default: 0.02
- **max_area_ratio** (*float*) – Maximum erased area / input image area Default: 0.4
- **aspect_range** (*sequence | float*) – Aspect ratio range of erased area. if float, it will be converted to (aspect_ratio, 1/aspect_ratio) Default: (3/10, 10/3)
- **mode** (*str*) – Fill method in erased area, can be: - const (default): All pixels are assign with the same value. - rand: each pixel is assigned with a random value in [0, 255]
- **fill_color** (*sequence | Number*) – Base color filled in erased area. Defaults to (128, 128, 128).
- **fill_std** (*sequence | Number, optional*) – If set and `mode` is 'rand', fill erased area with random color from normal distribution (mean=fill_color, std=fill_std); If not set, fill erased area with random color from uniform distribution (0~255). Defaults to None.

Note: See Random Erasing Data Augmentation This paper provided 4 modes: RE-R, RE-M, RE-0, RE-255, and use RE-M as default. The config of these 4 modes are: - RE-R: RandomErasing(mode='rand') - RE-M: RandomErasing(mode='const', fill_color=(123.67, 116.3, 103.5)) - RE-0: RandomErasing(mode='const', fill_color=0) - RE-255: RandomErasing(mode='const', fill_color=255)

__init__(*erase_prob=0.5*, *min_area_ratio=0.02*, *max_area_ratio=0.4*, *aspect_range=(0.3, 3.3333333333333335)*, *mode='const'*, *fill_color=(128, 128, 128)*, *fill_std=None*)
Initialize self. See help(type(self)) for accurate signature.

Submodules

easycv.datasets.classification.odps module

class easycv.datasets.classification.odps.**ClsOdpsDataset**(*data_source*, *pipeline*, *image_key='url_image'*, *label_key='label'*, ***kwargs*)

Bases: `Generic[torch.utils.data.dataset.T_co]`

Dataset for rotation prediction

__init__(*data_source*, *pipeline*, *image_key='url_image'*, *label_key='label'*, ***kwargs*)
Initialize self. See help(type(self)) for accurate signature.

evaluate(*results*, *evaluators*, *logger=None*)

easycv.datasets.classification.raw module

class easycv.datasets.classification.raw.**ClsDataset**(*data_source*, *pipeline*)
Bases: `Generic[torch.utils.data.dataset.T_co]`

Dataset for classification

Parameters

- **data_source** – data source to parse input data
- **pipeline** – transforms list

__init__(*data_source*, *pipeline*)
Initialize self. See help(type(self)) for accurate signature.

evaluate(*results*, *evaluators*, *logger=None*, *topk=(1, 5)*)
evaluate classification task

Parameters

- **results** – a dict of list of tensor, including prediction and groundtruth info, where prediction tensor is NxCand the same with groundtruth labels.
- **evaluators** – a list of evaluator

Returns a dict of float, different metric values

Return type eval_result

visualize(*results*, *vis_num=10*, ***kwargs*)
Visulaize the model output on validation data. :param results: A dictionary containing

class: List of length number of test images. img_metas: List of length number of test images,

dict of image meta info, containing filename, img_shape, origin_img_shape and so on.

Parameters **vis_num** – number of images visualized

Returns: A dictionary containing images: Visulaized images, list of np.ndarray. img_metas: List of length number of test images,

dict of image meta info, containing filename, img_shape, origin_img_shape and so on.

14.1.2 easycv.datasets.detection package

class easycv.datasets.detection.**DetDataset**(*data_source*, *pipeline*, *profiling=False*, *classes=None*)
Bases: Generic[torch.utils.data.dataset.T_co]

Dataset for Detection

__init__(*data_source*, *pipeline*, *profiling=False*, *classes=None*)

Parameters

- **data_source** – Data_source config dict
- **pipeline** – Pipeline config list
- **profiling** – If set True, will print pipeline time
- **classes** – A list of class names, used in evaluation for result and groundtruth visualization

evaluate(*results*, *evaluators=None*, *logger=None*)
Evaluates the detection boxes. :param results: A dictionary containing

detection_boxes: List of length number of test images. Float32 numpy array of shape [num_boxes, 4] and format [ymin, xmin, ymax, xmax] in absolute image coordinates.

detection_scores: List of length number of test images, detection scores for the boxes, float32 numpy array of shape [num_boxes].

detection_classes: List of length number of test images, integer numpy array of shape [num_boxes] containing 1-indexed detection classes for the boxes.

img_metas: List of length number of test images, dict of image meta info, containing filename, img_shape, origin_img_shape, scale_factor and so on.

Parameters **evaluators** – evaluators to calculate metric with results and groundtruth_dict

visualize(*results*, *vis_num=10*, *score_thr=0.3*, ***kwargs*)

Visulaize the model output on validation data. :param results: A dictionary containing

detection_boxes: List of length number of test images. Float32 numpy array of shape [num_boxes, 4] and format [ymin, xmin, ymax, xmax] in absolute image coordinates.

detection_scores: List of length number of test images, detection scores for the boxes, float32 numpy array of shape [num_boxes].

detection_classes: List of length number of test images, integer numpy array of shape [num_boxes] containing 1-indexed detection classes for the boxes.

img_metas: List of length number of test images, dict of image meta info, containing filename, img_shape, origin_img_shape, scale_factor and so on.

Parameters

- **vis_num** – number of images visualized
- **score_thr** – The threshold to filter box, boxes with scores greater than score_thr will be kept.

Returns: A dictionary containing images: Visulaized images. img_metas: List of length number of test images,

dict of image meta info, containing filename, img_shape, origin_img_shape, scale_factor and so on.

class easycv.datasets.detection.**DetImagesMixDataset**(*data_source*, *pipeline*, *dynamic_scale=None*, *skip_type_keys=None*, *profiling=False*, *classes=None*, *yolo_format=True*, *label_padding=True*)

Bases: Generic[torch.utils.data.dataset.T_co]

A wrapper of multiple images mixed dataset.

Suitable for training on multiple images mixed data augmentation like mosaic and mixup. For the augmentation pipeline of mixed image data, the *get_indexes* method needs to be provided to obtain the image indexes, and you can set *skip_flags* to change the pipeline running process. At the same time, we provide the *dynamic_scale* parameter to dynamically change the output image size.

output boxes format: cx, cy, w, h

Parameters

- **data_source** (DetSourceCoco) – The dataset to be mixed.
- **pipeline** (*Sequence[dict]*) – Sequence of transform object or config dict to be composed.
- **dynamic_scale** (*tuple[int], optional*) – The image scale can be changed dynamically. Default to None.
- **skip_type_keys** (*list[str], optional*) – Sequence of type string to be skip pipeline. Default to None.
- **label_padding** – out labeling padding [N, 120, 5]

__init__(*data_source*, *pipeline*, *dynamic_scale=None*, *skip_type_keys=None*, *profiling=False*, *classes=None*, *yolo_format=True*, *label_padding=True*)
Args: data_source: Data_source config dict pipeline: Pipeline config list profiling: If set True, will print pipeline time classes: A list of class names, used in evaluation for result and groundtruth visualization

update_skip_type_keys(*skip_type_keys*)
Update skip_type_keys. It is called by an external hook.

Parameters **skip_type_keys** (*list[str], optional*) – Sequence of type string to be skip pipeline.

update_dynamic_scale(*dynamic_scale*)
Update dynamic_scale. It is called by an external hook.

Parameters **dynamic_scale** (*tuple[int]*) – The image scale can be changed dynamically.

results2json(*results*, *outfile_prefix*)
Dump the detection results to a COCO style json file.

There are 3 types of results: proposals, bbox predictions, mask predictions, and they have different data types. This method will automatically recognize the type, and dump them to json files.

Parameters

- **results** (*list[list | tuple | ndarray]*) – Testing results of the dataset.
- **outfile_prefix** (*str*) – The filename prefix of the json files. If the prefix is "somepath/xxx", the json files will be named "somepath/xxx.bbox.json", "somepath/xxx.segm.json", "somepath/xxx.proposal.json".

Returns str]: Possible keys are "bbox", "segm", "proposal", and values are corresponding filenames.

Return type dict[str

format_results(*results*, *jsonfile_prefix=None*, ***kwargs*)
Format the results to json (standard format for COCO evaluation).

Parameters

- **results** (*list[tuple | numpy.ndarray]*) – Testing results of the dataset.
- **jsonfile_prefix** (*str | None*) – The prefix of json files. It includes the file path and the prefix of filename, e.g., "a/b/prefix". If not specified, a temp file will be created. Default: None.

Returns (result_files, tmp_dir), result_files is a dict containing the json filepaths, tmp_dir is the temporal directory created for saving json files when jsonfile_prefix is not specified.

Return type tuple

Subpackages

easycv.datasets.detection.data_sources package

class easycv.datasets.detection.data_sources.**DetSourceCoco**(*ann_file*, *img_prefix*, *pipeline*, *test_mode=False*, *filter_empty_gt=False*, *classes=None*, *iscrowd=False*)

Bases: object

coco data source

__init__(*ann_file*, *img_prefix*, *pipeline*, *test_mode=False*, *filter_empty_gt=False*, *classes=None*, *iscrowd=False*)

Parameters

- **ann_file** – Path of annotation file.
- **img_prefix** – coco path prefix
- **test_mode** (*bool, optional*) – If set True, *self._filter_imgs* will not works.
- **filter_empty_gt** (*bool, optional*) – If set true, images without bounding boxes of the dataset's classes will be filtered out. This option only works when *test_mode=False*, i.e., we never filter images during tests.
- **iscrowd** – when traing setted as False, when val setted as Tre

get_length()
Total number of samples of data.

load_annotations(*ann_file*)
Load annotation from COCO style annotation file.

Parameters **ann_file** (*str*) – Path of annotation file.

Returns Annotation info from COCO api.

Return type list[dict]

get_ann_info(*idx*)
Get COCO annotation by index.

Parameters **idx** (*int*) – Index of data.

Returns Annotation info of specified index.

Return type dict

get_cat_ids(*idx*)
Get COCO category ids by index.

Parameters **idx** (*int*) – Index of data.

Returns All categories in the image of specified index.

Return type list[int]

xyxy2xywh(*bbox*)
Convert xyxy style bounding boxes to xywh style for COCO evaluation.

Parameters **bbox** (*numpy.ndarray*) – The bounding boxes, shape (4,), in xyxy order.

Returns The converted bounding boxes, in xywh order.

Return type list[float]

pre_pipeline(*results*)
Prepare results dict for pipeline.

prepare_train_img(*idx*)
Get training data and annotations after pipeline.

Parameters **idx** (*int*) – Index of data.

Returns Training data and annotation after pipeline with new keys introduced by pipeline.

Return type dict

get_sample(*idx*)

class easycv.datasets.detection.data_sources.**DetSourcePAI**(*path*, *classes=[]*, *cache_at_init=False*, *cache_on_the_fly=False*, *parse_fn=<function parser_manifest_row_str>*, *num_processes=1*, ***kwargs*)

Bases: easycv.datasets.detection.data_sources.base.DetSourceBase

data format please refer to: https://help.aliyun.com/document_detail/311173.html

__init__(*path*, *classes=[]*, *cache_at_init=False*, *cache_on_the_fly=False*, *parse_fn=<function parser_manifest_row_str>*, *num_processes=1*, ***kwargs*)

Parameters

- **path** – Path of manifest path with pai label format
- **classes** – classes list
- **cache_at_init** – if set True, will cache in memory in __init__ for faster training
- **cache_on_the_fly** – if set True, will cache in memroy during training
- **parse_fn** – parse function to parse item of source iterator
- **num_processes** – number of processes to parse samples

get_source_iterator()

Return data list iterator, source iterator will be passed to parse_fn, and parse_fn will receive params of item of source iter and classes for parsing. What does parse_fn need, what does source iterator returns.

class easycv.datasets.detection.data_sources.**DetSourceRaw**(*img_root_path*, *label_root_path*, *classes=[]*, *cache_at_init=False*, *cache_on_the_fly=False*, *delimeter=' '*, *parse_fn=<function parse_raw>*, *num_processes=1*, ***kwargs*)

Bases: easycv.datasets.detection.data_sources.base.DetSourceBase

data dir is as follows: ``` |- data_dir

|**-images** |-1.jpg |-...

|**-labels** |-1.txt |-...

` Label txt file is as follows: The first column is the label id, and columns 2 to 5 are coordinates relative to the image width and height [x_center, y_center, bbox_w, bbox_h]. ` 15 0.519398 0.544087 0.476359 0.572061 2 0.501859 0.820726 0.996281 0.332178 ... ``` .. rubric:: Example

data_source = DetSourceRaw(img_root_path='/your/data_dir/images', label_root_path='/your/data_dir/labels',

)

__init__(*img_root_path*, *label_root_path*, *classes=[]*, *cache_at_init=False*, *cache_on_the_fly=False*, *delimeter=' '*, *parse_fn=<function parse_raw>*, *num_processes=1*, ***kwargs*)

Parameters

- **img_root_path** – images dir path
- **label_root_path** – labels dir path

- **classes** (*list, optional*) – classes list
- **cache_at_init** – if set True, will cache in memory in __init__ for faster training
- **cache_on_the_fly** – if set True, will cache in memroy during training
- **delimeter** – delimeter of txt file
- **parse_fn** – parse function to parse item of source iterator
- **num_processes** – number of processes to parse samples

get_source_iterator()

Return data list iterator, source iterator will be passed to parse_fn, and parse_fn will receive params of item of source iter and classes for parsing. What does parse_fn need, what does source iterator returns.

post_process_fn(*result_dict*)

class easycv.datasets.detection.data_sources.**DetSourceVOC**(*path*, *classes=[]*, *img_root_path=None*, *label_root_path=None*, *cache_at_init=False*, *cache_on_the_fly=False*, *img_suffix='.jpg'*, *label_suffix='.xml'*, *parse_fn=<function parse_xml>*, *num_processes=1*, ***kwargs*)

Bases: easycv.datasets.detection.data_sources.base.DetSourceBase

data dir is as follows: ``` |- voc_data

|-ImageSets

|-Main |-train.txt |-...

|-JPEGImages |-00001.jpg |-...

|-Annotations |-00001.xml |-...

``` Example1:

**data_source = DetSourceVOC**( path='/your/voc_data/ImageSets/Main/train.txt', classes=${VOC_CLASSES},

)

**Example1:**

**data_source = DetSourceVOC**( path='/your/voc_data/train.txt', classes=${VOC_CLASSES}, img_root_path='/your/voc_data/images', img_root_path='/your/voc_data/annotations'

)

**__init__**(*path*, *classes=[]*, *img_root_path=None*, *label_root_path=None*, *cache_at_init=False*, *cache_on_the_fly=False*, *img_suffix='.jpg'*, *label_suffix='.xml'*, *parse_fn=<function parse_xml>*, *num_processes=1*, *\*\*kwargs*)

**Parameters**

- **path** – path of img id list file in ImageSets/Main/
- **classes** – classes list
- **img_root_path** – image dir path, if None, default to detect the image dir by the relative path of the *path* according to the VOC data format.
```

- **label_root_path** – label dir path, if None, default to detect the label dir by the relative path of the *path* according to the VOC data format.
- **cache_at_init** – if set True, will cache in memory in __init__ for faster training
- **cache_on_the_fly** – if set True, will cache in memroy during training
- **img_suffix** – suffix of image file
- **label_suffix** – suffix of label file
- **parse_fn** – parse function to parse item of source iterator
- **num_processes** – number of processes to parse samples

get_source_iterator()
Return data list iterator, source iterator will be passed to parse_fn, and parse_fn will receive params of item of source iter and classes for parsing. What does parse_fn need, what does source iterator returns.

Submodules

easycv.datasets.detection.data_sources.coco module

class easycv.datasets.detection.data_sources.coco.**DetSourceCoco**(*ann_file*, *img_prefix*, *pipeline*, *test_mode=False*, *filter_empty_gt=False*, *classes=None*, *iscrowd=False*)

Bases: object

coco data source

__init__(*ann_file*, *img_prefix*, *pipeline*, *test_mode=False*, *filter_empty_gt=False*, *classes=None*, *iscrowd=False*)

Parameters

- **ann_file** – Path of annotation file.
- **img_prefix** – coco path prefix
- **test_mode** (*bool, optional*) – If set True, *self._filter_imgs* will not works.
- **filter_empty_gt** (*bool, optional*) – If set true, images without bounding boxes of the dataset's classes will be filtered out. This option only works when *test_mode=False*, i.e., we never filter images during tests.
- **iscrowd** – when traing setted as False, when val setted as Tre

get_length()
Total number of samples of data.

load_annotations(*ann_file*)
Load annotation from COCO style annotation file.

Parameters **ann_file** (*str*) – Path of annotation file.

Returns Annotation info from COCO api.

Return type list[dict]

get_ann_info(*idx*)
Get COCO annotation by index.

Parameters **idx** (*int*) – Index of data.

Returns Annotation info of specified index.

Return type dict

get_cat_ids(*idx*)
Get COCO category ids by index.

Parameters **idx** (*int*) – Index of data.

Returns All categories in the image of specified index.

Return type list[int]

xyxy2xywh(*bbox*)
Convert xyxy style bounding boxes to xywh style for COCO evaluation.

Parameters **bbox** (*numpy.ndarray*) – The bounding boxes, shape (4,), in xyxy order.

Returns The converted bounding boxes, in xywh order.

Return type list[float]

pre_pipeline(*results*)
Prepare results dict for pipeline.

prepare_train_img(*idx*)
Get training data and annotations after pipeline.

Parameters **idx** (*int*) – Index of data.

Returns Training data and annotation after pipeline with new keys introduced by pipeline.

Return type dict

get_sample(*idx*)

easycv.datasets.detection.data_sources.pai_format module

easycv.datasets.detection.data_sources.pai_format.**get_prior_task_id**(*keys*)
"The task id ends with *check* is the highest priority.

easycv.datasets.detection.data_sources.pai_format.**is_itag_v2**(*row*)
The keyword of the data source is *picUrl* in v1, but is *source* in v2

easycv.datasets.detection.data_sources.pai_format.**parser_manifest_row_str**(*row_str*, *classes*)

class easycv.datasets.detection.data_sources.pai_format.**DetSourcePAI**(*path*, *classes=[]*, *cache_at_init=False*, *cache_on_the_fly=False*, *parse_fn=<function parser_manifest_row_str>*, *num_processes=1*, ***kwargs*)

Bases: easycv.datasets.detection.data_sources.base.DetSourceBase

data format please refer to: https://help.aliyun.com/document_detail/311173.html

__init__(*path*, *classes=[]*, *cache_at_init=False*, *cache_on_the_fly=False*, *parse_fn=<function parser_manifest_row_str>*, *num_processes=1*, ***kwargs*)

Parameters

- **path** – Path of manifest path with pai label format
- **classes** – classes list
- **cache_at_init** – if set True, will cache in memory in __init__ for faster training
- **cache_on_the_fly** – if set True, will cache in memroy during training
- **parse_fn** – parse function to parse item of source iterator
- **num_processes** – number of processes to parse samples

get_source_iterator()
Return data list iterator, source iterator will be passed to parse_fn, and parse_fn will receive params of item of source iter and classes for parsing. What does parse_fn need, what does source iterator returns.

easycv.datasets.detection.data_sources.raw module

easycv.datasets.detection.data_sources.raw.**parse_raw**(*source_iter*, *classes=None*, *delimeter=' '*)

class easycv.datasets.detection.data_sources.raw.**DetSourceRaw**(*img_root_path*, *label_root_path*, *classes=[]*, *cache_at_init=False*, *cache_on_the_fly=False*, *delimeter=' '*, *parse_fn=<function parse_raw>*, *num_processes=1*, ***kwargs*)

Bases: easycv.datasets.detection.data_sources.base.DetSourceBase

data dir is as follows: ``` |- data_dir

|-images |-1.jpg |-...

|-labels |-1.txt |-...

` Label txt file is as follows: The first column is the label id, and columns 2 to 5 are coordinates relative to the image width and height [x_center, y_center, bbox_w, bbox_h]. ` 15 0.519398 0.544087 0.476359 0.572061 2 0.501859 0.820726 0.996281 0.332178 ... ``` .. rubric:: Example

data_source = DetSourceRaw(img_root_path='/your/data_dir/images', label_root_path='/your/data_dir/labels',

)

__init__(*img_root_path*, *label_root_path*, *classes=[]*, *cache_at_init=False*, *cache_on_the_fly=False*, *delimeter=' '*, *parse_fn=<function parse_raw>*, *num_processes=1*, ***kwargs*)

Parameters

- **img_root_path** – images dir path
- **label_root_path** – labels dir path
- **classes** (*list, optional*) – classes list
- **cache_at_init** – if set True, will cache in memory in __init__ for faster training
- **cache_on_the_fly** – if set True, will cache in memroy during training
- **delimeter** – delimeter of txt file
- **parse_fn** – parse function to parse item of source iterator
- **num_processes** – number of processes to parse samples

get_source_iterator()
Return data list iterator, source iterator will be passed to parse_fn, and parse_fn will receive params of item of source iter and classes for parsing. What does parse_fn need, what does source iterator returns.

post_process_fn(*result_dict*)

easycv.datasets.detection.data_sources.utils module

easycv.datasets.detection.data_sources.utils.**exif_size**(*img*)

easycv.datasets.detection.data_sources.voc module

easycv.datasets.detection.data_sources.voc.**parse_xml**(*source_item*, *classes*)

class easycv.datasets.detection.data_sources.voc.**DetSourceVOC**(*path*, *classes=[]*, *img_root_path=None*, *label_root_path=None*, *cache_at_init=False*, *cache_on_the_fly=False*, *img_suffix='.jpg'*, *label_suffix='.xml'*, *parse_fn=<function parse_xml>*, *num_processes=1*, ***kwargs*)

Bases: easycv.datasets.detection.data_sources.base.DetSourceBase

data dir is as follows: ``` |- voc_data

|-ImageSets

|-Main |-train.txt |-…

|-JPEGImages |-00001.jpg |-…

|-Annotations |-00001.xml |-…

``` Example1:

**data_source = DetSourceVOC(** path='/your/voc_data/ImageSets/Main/train.txt', classes=${VOC_CLASSES},

)

**Example1:**

**data_source = DetSourceVOC(** path='/your/voc_data/train.txt', classes=${VOC_CLASSES}, img_root_path='/your/voc_data/images', img_root_path='/your/voc_data/annotations'

)

**__init__**(*path*, *classes=[]*, *img_root_path=None*, *label_root_path=None*, *cache_at_init=False*, *cache_on_the_fly=False*, *img_suffix='.jpg'*, *label_suffix='.xml'*, *parse_fn=<function parse_xml>*, *num_processes=1*, *\*\*kwargs*)

**Parameters**

- **path** – path of img id list file in ImageSets/Main/
- **classes** – classes list
```

- **img_root_path** – image dir path, if None, default to detect the image dir by the relative path of the *path* according to the VOC data format.
- **label_root_path** – label dir path, if None, default to detect the label dir by the relative path of the *path* according to the VOC data format.
- **cache_at_init** – if set True, will cache in memory in __init__ for faster training
- **cache_on_the_fly** – if set True, will cache in memroy during training
- **img_suffix** – suffix of image file
- **label_suffix** – suffix of label file
- **parse_fn** – parse function to parse item of source iterator
- **num_processes** – number of processes to parse samples

get_source_iterator()
Return data list iterator, source iterator will be passed to parse_fn, and parse_fn will receive params of item of source iter and classes for parsing. What does parse_fn need, what does source iterator returns.

easycv.datasets.detection.pipelines package

class easycv.datasets.detection.pipelines.**MMToTensor**
Bases: object

Transform image to Tensor.

Required key: 'img'. Modifies key: 'img'.

Parameters **results** (*dict*) – contain all information about training.

class easycv.datasets.detection.pipelines.**NormalizeTensor**(*mean*, *std*)
Bases: object

Normalize the Tensor image (CxHxW), with mean and std.

Required key: 'img'. Modifies key: 'img'.

Parameters

- **mean** (*list[float]*) – Mean values of 3 channels.
- **std** (*list[float]*) – Std values of 3 channels.

__init__(*mean*, *std*)
Initialize self. See help(type(self)) for accurate signature.

class easycv.datasets.detection.pipelines.**MMMosaic**(*img_scale=(640, 640)*, *center_ratio_range=(0.5, 1.5)*, *pad_val=114*)
Bases: object

Mosaic augmentation.

Given 4 images, mosaic transform combines them into one output image. The output image is composed of the parts from each sub- image.

```
                mosaic transform
                   center_x
          +------------------------------+
          |       pad        |  pad      |
          |      +-----------+           |
```

(continues on next page)

(continued from previous page)

```
           |      |           |           |
           |      |  image1   |--------+  |
           |      |           |        |  |
           |      |           | image2 |  |
center_y   |----+-------------+-----------|
           |    |   cropped   |           |
           |pad |   image3    |  image4   |
           |    |             |           |
           +----|-------------+-----------+
                |             |
                +-------------+

The mosaic transform steps are as follows:

    1. Choose the mosaic center as the intersections of 4 images
    2. Get the left top image according to the index, and randomly
       sample another 3 images from the custom dataset.
    3. Sub image will be cropped if image is larger than mosaic patch
```

Parameters

- **img_scale** (*Sequence[int]*) – Image size after mosaic pipeline of single image. Default to (640, 640).
- **center_ratio_range** (*Sequence[float]*) – Center ratio range of mosaic output. Default to (0.5, 1.5).
- **pad_val** (*int*) – Pad value. Default to 114.

__init__(*img_scale=(640, 640)*, *center_ratio_range=(0.5, 1.5)*, *pad_val=114*)
Initialize self. See help(type(self)) for accurate signature.

get_indexes(*dataset*)
Call function to collect indexes.

Parameters **dataset** (DetImagesMixDataset) – The dataset.

Returns indexes.

Return type list

class easycv.datasets.detection.pipelines.**MMMixUp**(*img_scale=(640, 640)*, *ratio_range=(0.5, 1.5)*, *flip_ratio=0.5*, *pad_val=114*, *max_iters=15*, *min_bbox_size=5*, *min_area_ratio=0.2*, *max_aspect_ratio=20*)

Bases: object

MixUp data augmentation.

```
                  mixup transform
           +------------------------------+
           | mixup image   |              |
           |      +--------|--------+     |
           |      |        |        |     |
           |------------+  |        |     |
           |      |     |  |        |     |
```

(continues on next page)

(continued from previous page)

```
            |      |      image      |     |
            |      |                 |     |
            |      |                 |     |
            |      |-----------------+     |
            |      |             pad       |
            |      +-------------------------+

The mixup transform steps are as follows::

   1. Another random image is picked by dataset and embedded in
      the top left patch(after padding and resizing)
   2. The target of mixup transform is the weighted average of mixup
      image and origin image.
```

Parameters

- **img_scale** (*Sequence[int]*) – Image output size after mixup pipeline. Default: (640, 640).
- **ratio_range** (*Sequence[float]*) – Scale ratio of mixup image. Default: (0.5, 1.5).
- **flip_ratio** (*float*) – Horizontal flip ratio of mixup image. Default: 0.5.
- **pad_val** (*int*) – Pad value. Default: 114.
- **max_iters** (*int*) – The maximum number of iterations. If the number of iterations is greater than *max_iters*, but gt_bbox is still empty, then the iteration is terminated. Default: 15.
- **min_bbox_size** (*float*) – Width and height threshold to filter bboxes. If the height or width of a box is smaller than this value, it will be removed. Default: 5.
- **min_area_ratio** (*float*) – Threshold of area ratio between original bboxes and wrapped bboxes. If smaller than this value, the box will be removed. Default: 0.2.
- **max_aspect_ratio** (*float*) – Aspect ratio of width and height threshold to filter bboxes. If max(h/w, w/h) larger than this value, the box will be removed. Default: 20.

__init__(*img_scale=(640, 640)*, *ratio_range=(0.5, 1.5)*, *flip_ratio=0.5*, *pad_val=114*, *max_iters=15*, *min_bbox_size=5*, *min_area_ratio=0.2*, *max_aspect_ratio=20*)

Initialize self. See help(type(self)) for accurate signature.

get_indexes(*dataset*)

Call function to collect indexes.

Parameters **dataset** (DetImagesMixDataset) – The dataset.

Returns indexes.

Return type list

class easycv.datasets.detection.pipelines.**MMRandomAffine**(*max_rotate_degree=10.0*, *max_translate_ratio=0.1*, *scaling_ratio_range=(0.5, 1.5)*, *max_shear_degree=2.0*, *border=(0, 0)*, *border_val=(114, 114, 114)*, *min_bbox_size=2*, *min_area_ratio=0.2*, *max_aspect_ratio=20*)

Bases: object

Random affine transform data augmentation. for yolox

This operation randomly generates affine transform matrix which including rotation, translation, shear and scaling transforms.

Parameters

- **max_rotate_degree** (*float*) – Maximum degrees of rotation transform. Default: 10.
- **max_translate_ratio** (*float*) – Maximum ratio of translation. Default: 0.1.
- **scaling_ratio_range** (*tuple[float]*) – Min and max ratio of scaling transform. Default: (0.5, 1.5).
- **max_shear_degree** (*float*) – Maximum degrees of shear transform. Default: 2.
- **border** (*tuple[int]*) – Distance from height and width sides of input image to adjust output shape. Only used in mosaic dataset. Default: (0, 0).
- **border_val** (*tuple[int]*) – Border padding values of 3 channels. Default: (114, 114, 114).
- **min_bbox_size** (*float*) – Width and height threshold to filter bboxes. If the height or width of a box is smaller than this value, it will be removed. Default: 2.
- **min_area_ratio** (*float*) – Threshold of area ratio between original bboxes and wrapped bboxes. If smaller than this value, the box will be removed. Default: 0.2.
- **max_aspect_ratio** (*float*) – Aspect ratio of width and height threshold to filter bboxes. If max(h/w, w/h) larger than this value, the box will be removed.

__init__(*max_rotate_degree=10.0*, *max_translate_ratio=0.1*, *scaling_ratio_range=(0.5, 1.5)*, *max_shear_degree=2.0*, *border=(0, 0)*, *border_val=(114, 114, 114)*, *min_bbox_size=2*, *min_area_ratio=0.2*, *max_aspect_ratio=20*)

Initialize self. See help(type(self)) for accurate signature.

filter_gt_bboxes(*origin_bboxes*, *wrapped_bboxes*)

class easycv.datasets.detection.pipelines.**MMPhotoMetricDistortion**(*brightness_delta=32*, *contrast_range=(0.5, 1.5)*, *saturation_range=(0.5, 1.5)*, *hue_delta=18*)

Bases: object

Apply photometric distortion to image sequentially, every transformation is applied with a probability of 0.5. The position of random contrast is in second or second to last.

1. random brightness
2. random contrast (mode 0)
3. convert color from BGR to HSV
4. random saturation
5. random hue
6. convert color from HSV to BGR
7. random contrast (mode 1)
8. randomly swap channels

Parameters

- **brightness_delta** (*int*) – delta of brightness.

- **contrast_range** (*tuple*) – range of contrast.
- **saturation_range** (*tuple*) – range of saturation.
- **hue_delta** (*int*) – delta of hue.

__init__(*brightness_delta=32*, *contrast_range=(0.5, 1.5)*, *saturation_range=(0.5, 1.5)*, *hue_delta=18*)
Initialize self. See help(type(self)) for accurate signature.

class easycv.datasets.detection.pipelines.**MMResize**(*img_scale=None*, *multiscale_mode='range'*, *ratio_range=None*, *keep_ratio=True*, *bbox_clip_border=True*, *backend='cv2'*, *override=False*)

Bases: object

Resize images & bbox & mask. This transform resizes the input image to some scale. Bboxes and masks are then resized with the same scale factor. If the input dict contains the key "scale", then the scale in the input dict is used, otherwise the specified scale in the init method is used. If the input dict contains the key "scale_factor" (if MultiScaleFlipAug does not give img_scale but scale_factor), the actual scale will be computed by image shape and scale_factor. *img_scale* can either be a tuple (single-scale) or a list of tuple (multi-scale). There are 3 multiscale modes: - `ratio_range is not None`: randomly sample a ratio from the ratio range and multiply it with the image scale. - `ratio_range is None` and `multiscale_mode == "range"`: randomly sample a scale from the multiscale range. - `ratio_range is None` and `multiscale_mode == "value"`: randomly sample a scale from multiple scales. :param img_scale: Images scales for resizing. :type img_scale: tuple or list[tuple] :param multiscale_mode: Either "range" or "value". :type multiscale_mode: str :param ratio_range: (min_ratio, max_ratio) :type ratio_range: tuple[float] :param keep_ratio: Whether to keep the aspect ratio when resizing the

image.

Parameters

- **bbox_clip_border** (*bool, optional*) – Whether clip the objects outside the border of the image. Defaults to True.
- **backend** (*str*) – Image resize backend, choices are 'cv2' and 'pillow'. These two backends generates slightly different results. Defaults to 'cv2'.
- **override** (*bool, optional*) – Whether to override *scale* and *scale_factor* so as to call resize twice. Default False. If True, after the first resizing, the existed *scale* and *scale_factor* will be ignored so the second resizing can be allowed. This option is a work-around for multiple times of resize in DETR. Defaults to False.

__init__(*img_scale=None*, *multiscale_mode='range'*, *ratio_range=None*, *keep_ratio=True*, *bbox_clip_border=True*, *backend='cv2'*, *override=False*)
Initialize self. See help(type(self)) for accurate signature.

static random_select(*img_scales*)
Randomly select an img_scale from given candidates. :param img_scales: Images scales for selection. :type img_scales: list[tuple]

Returns Returns a tuple `(img_scale, scale_dix)`, where `img_scale` is the selected image scale and `scale_idx` is the selected index in the given candidates.

Return type (tuple, int)

static random_sample(*img_scales*)
Randomly sample an img_scale when `multiscale_mode=='range'`. :param img_scales: Images scale range for sampling.

There must be two tuples in img_scales, which specify the lower and upper bound of image scales.

Returns Returns a tuple `(img_scale, None)`, where `img_scale` is sampled scale and None is just a placeholder to be consistent with *random_select()*.

Return type (tuple, None)

static random_sample_ratio(*img_scale*, *ratio_range*)

Randomly sample an img_scale when `ratio_range` is specified. A ratio will be randomly sampled from the range specified by `ratio_range`. Then it would be multiplied with `img_scale` to generate sampled scale. :param img_scale: Images scale base to multiply with ratio. :type img_scale: tuple :param ratio_range: The minimum and maximum ratio to scale

the `img_scale`.

Returns Returns a tuple `(scale, None)`, where `scale` is sampled ratio multiplied with `img_scale` and None is just a placeholder to be consistent with *random_select()*.

Return type (tuple, None)

class easycv.datasets.detection.pipelines.**MMRandomFlip**(*flip_ratio=None*, *direction='horizontal'*)

Bases: `object`

Flip the image & bbox & mask. If the input dict contains the key "flip", then the flag will be used, otherwise it will be randomly decided by a ratio specified in the init method. When random flip is enabled, `flip_ratio/direction` can either be a float/string or tuple of float/string. There are 3 flip modes: - `flip_ratio` is float, `direction` is string: the image will be

`direction``ly flipped with probability of ``flip_ratio` . E.g., `flip_ratio=0.5`, `direction='horizontal'`, then image will be horizontally flipped with probability of 0.5.

- **`flip_ratio` is float, `direction` is list of string: the image wil** be `direction[i]``ly flipped with probability of ``flip_ratio/len(direction)`. E.g., `flip_ratio=0.5`, `direction=['horizontal', 'vertical']`, then image will be horizontally flipped with probability of 0.25, vertically with probability of 0.25.
- **`flip_ratio` is list of float, `direction` is list of string:** given `len(flip_ratio) == len(direction)`, the image wil be `direction[i]``ly flipped with probability of ``flip_ratio[i]`. E.g., `flip_ratio=[0.3, 0.5]`, `direction=['horizontal', 'vertical']`, then image will be horizontally flipped with probability of 0.3, vertically with probability of 0.5.

Parameters

- **`flip_ratio`** (*float | list[float], optional*) – The flipping probability. Default: None.
- **`direction`** (*str | list[str], optional*) – The flipping direction. Options are 'horizontal', 'vertical', 'diagonal'. Default: 'horizontal'. If input is a list, the length must equal `flip_ratio`. Each element in `flip_ratio` indicates the flip probability of corresponding direction.

__init__(*flip_ratio=None*, *direction='horizontal'*)

Initialize self. See help(type(self)) for accurate signature.

bbox_flip(*bboxes*, *img_shape*, *direction*)
Flip bboxes horizontally. :param bboxes: Bounding boxes, shape (..., 4*k) :type bboxes: numpy.ndarray :param img_shape: Image shape (height, width) :type img_shape: tuple[int] :param direction: Flip direction. Options are 'horizontal',

'vertical'.

Returns Flipped bounding boxes.

Return type numpy.ndarray

class easycv.datasets.detection.pipelines.**MMPad**(*size=None*, *size_divisor=None*, *pad_to_square=False*, *pad_val=0*, *seg_pad_val=255*)

Bases: object

Pad the image & mask. There are two padding modes: (1) pad to a fixed size and (2) pad to the minimum size that is divisible by some number. Added keys are "pad_shape", "pad_fixed_size", "pad_size_divisor", :param size: Fixed padding size. :type size: tuple, optional :param size_divisor: The divisor of padded size. :type size_divisor: int, optional :param pad_to_square: Whether to pad the image into a square.

Currently only used for YOLOX. Default: False.

Parameters

- **pad_val** (*float, optional*) – Padding value, 0 by default.
- **seg_pad_val** (*float, optional*) – Padding value of segmentation map. Default: 255.

__init__(*size=None*, *size_divisor=None*, *pad_to_square=False*, *pad_val=0*, *seg_pad_val=255*)
Initialize self. See help(type(self)) for accurate signature.

class easycv.datasets.detection.pipelines.**MMNormalize**(*mean*, *std*, *to_rgb=True*)

Bases: object

Normalize the image.

Added key is "img_norm_cfg".

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, default is true.

__init__(*mean*, *std*, *to_rgb=True*)
Initialize self. See help(type(self)) for accurate signature.

class easycv.datasets.detection.pipelines.**LoadImageFromFile**(*to_float32=False*, *color_type='color'*, *file_client_args={'backend': 'disk'}*)

Bases: object

Load an image from file. Required keys are "img_prefix" and "img_info" (a dict that must contain the key "filename"). Added or updated keys are "filename", "img", "img_shape", "ori_shape" (same as *img_shape*), "pad_shape" (same as *img_shape*), "scale_factor" (1.0) and "img_norm_cfg" (means=0 and stds=1). :param to_float32: 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.

Parameters

- **color_type** (*str*) – The flag argument for `mmcv.imfrombytes()`. Defaults to 'color'.
- **file_client_args** (*dict*) – Arguments to instantiate a FileClient. See `mmcv.fileio.FileClient` for details. Defaults to `dict(backend='disk')`.

__init__(*to_float32=False*, *color_type='color'*, *file_client_args={'backend': 'disk'}*)
Initialize self. See help(type(self)) for accurate signature.

class easycv.datasets.detection.pipelines.**LoadMultiChannelImageFromFiles**(*to_float32=False*, *color_type='unchanged'*, *file_client_args={'backend': 'disk'}*)

Bases: `object`

Load multi-channel images from a list of separate channel files.

Required keys are "img_prefix" and "img_info" (a dict that must contain the key "filename", which is expected to be a list of filenames). Added or updated keys are "filename", "img", "img_shape", "ori_shape" (same as *img_shape*), "pad_shape" (same as *img_shape*), "scale_factor" (1.0) and "img_norm_cfg" (means=0 and stds=1).

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'.
- **file_client_args** (*dict*) – Arguments to instantiate a FileClient. See `mmcv.fileio.FileClient` for details. Defaults to `dict(backend='disk')`.

__init__(*to_float32=False*, *color_type='unchanged'*, *file_client_args={'backend': 'disk'}*)
Initialize self. See help(type(self)) for accurate signature.

class easycv.datasets.detection.pipelines.**LoadAnnotations**(*with_bbox=True*, *with_label=True*, *with_mask=False*, *with_seg=False*, *poly2mask=True*, *file_client_args={'backend': 'disk'}*)

Bases: `object`

Load multiple types of annotations. :param with_bbox: Whether to parse and load the bbox annotation.

Default: True.

Parameters

- **with_label** (*bool*) – Whether to parse and load the label annotation. Default: True.
- **with_mask** (*bool*) – Whether to parse and load the mask annotation. Default: False.
- **with_seg** (*bool*) – Whether to parse and load the semantic segmentation annotation. Default: False.
- **poly2mask** (*bool*) – Whether to convert the instance masks from polygons to bitmaps. Default: True.
- **file_client_args** (*dict*) – Arguments to instantiate a FileClient. See `mmcv.fileio.FileClient` for details. Defaults to `dict(backend='disk')`.

__init__(*with_bbox=True*, *with_label=True*, *with_mask=False*, *with_seg=False*, *poly2mask=True*, *file_client_args={'backend': 'disk'}*)
Initialize self. See help(type(self)) for accurate signature.

process_polygons(*polygons*)
Convert polygons to list of ndarray and filter invalid polygons. :param polygons: Polygons of one instance. :type polygons: list[list]

Returns Processed polygons.

Return type list[numpy.ndarray]

class easycv.datasets.detection.pipelines.**MMMultiScaleFlipAug**(*transforms*, *img_scale=None*, *scale_factor=None*, *flip=False*, *flip_direction='horizontal'*)

Bases: object

Test-time augmentation with multiple scales and flipping.

An example configuration is as followed:

```
img_scale=[(1333, 400), (1333, 800)],
flip=True,
transforms=[
    dict(type='Resize', keep_ratio=True),
    dict(type='RandomFlip'),
    dict(type='Normalize', **img_norm_cfg),
    dict(type='Pad', size_divisor=32),
    dict(type='ImageToTensor', keys=['img']),
    dict(type='Collect', keys=['img']),
]
```

After MultiScaleFLipAug with above configuration, the results are wrapped into lists of the same length as followed:

```
dict(
    img=[...],
    img_shape=[...],
    scale=[(1333, 400), (1333, 400), (1333, 800), (1333, 800)]
    flip=[False, True, False, True]
    ...
)
```

Parameters

- **transforms** (*list[dict]*) – Transforms to apply in each augmentation.
- **img_scale** (*tuple | list[tuple] | None*) – Images scales for resizing.
- **scale_factor** (*float | list[float] | None*) – Scale factors for resizing.
- **flip** (*bool*) – Whether apply flip augmentation. Default: 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. Default: "horizontal".

__init__(*transforms*, *img_scale=None*, *scale_factor=None*, *flip=False*, *flip_direction='horizontal'*)
Initialize self. See help(type(self)) for accurate signature.

class easycv.datasets.detection.pipelines.**MMRandomCrop**(*crop_size*, *crop_type='absolute'*, *allow_negative_crop=False*, *recompute_bbox=False*, *bbox_clip_border=True*)

Bases: object

Random crop the image & bboxes & masks.

The absolute *crop_size* is sampled based on *crop_type* and *image_size*, then the cropped results are generated.

Parameters

- **crop_size** (*tuple*) – The relative ratio or absolute pixels of height and width.
- **crop_type** (*str, optional*) – one of "relative_range", "relative", "absolute", "absolute_range". "relative" randomly crops (h * crop_size[0], w * crop_size[1]) part from an input of size (h, w). "relative_range" uniformly samples relative crop size from range [crop_size[0], 1] and [crop_size[1], 1] for height and width respectively. "absolute" crops from an input with absolute size (crop_size[0], crop_size[1]). "absolute_range" uniformly samples crop_h in range [crop_size[0], min(h, crop_size[1])] and crop_w in range [crop_size[0], min(w, crop_size[1])]. Default "absolute".
- **allow_negative_crop** (*bool, optional*) – Whether to allow a crop that does not contain any bbox area. Default False.
- **recompute_bbox** (*bool, optional*) – Whether to re-compute the boxes based on cropped instance masks. Default False.
- **bbox_clip_border** (*bool, optional*) – Whether clip the objects outside the border of the image. Defaults to True.

Note:

- **If the image is smaller than the absolute crop size, return the** original image.
- The keys for bboxes, labels and masks must be aligned. That is, *gt_bboxes* corresponds to *gt_labels* and *gt_masks*, and *gt_bboxes_ignore* corresponds to *gt_labels_ignore* and *gt_masks_ignore*.
- If the crop does not contain any gt-bbox region and *allow_negative_crop* is set to False, skip this image.

__init__(*crop_size*, *crop_type='absolute'*, *allow_negative_crop=False*, *recompute_bbox=False*, *bbox_clip_border=True*)

Initialize self. See help(type(self)) for accurate signature.

class easycv.datasets.detection.pipelines.**MMFilterAnnotations**(*min_gt_bbox_wh*, *keep_empty=True*)

Bases: object

Filter invalid annotations. :param min_gt_bbox_wh: Minimum width and height of ground truth

boxes.

Parameters **keep_empty** (*bool*) – Whether to return None when it becomes an empty bbox after filtering. Default: True

__init__(*min_gt_bbox_wh*, *keep_empty=True*)

Initialize self. See help(type(self)) for accurate signature.

Submodules

easycv.datasets.detection.pipelines.mm_transforms module

class easycv.datasets.detection.pipelines.mm_transforms.**MMToTensor**
Bases: object

Transform image to Tensor.

Required key: 'img'. Modifies key: 'img'.

Parameters **results** (*dict*) – contain all information about training.

class easycv.datasets.detection.pipelines.mm_transforms.**NormalizeTensor**(*mean*, *std*)
Bases: object

Normalize the Tensor image (CxHxW), with mean and std.

Required key: 'img'. Modifies key: 'img'.

Parameters

- **mean** (*list[float]*) – Mean values of 3 channels.
- **std** (*list[float]*) – Std values of 3 channels.

__init__(*mean*, *std*)
Initialize self. See help(type(self)) for accurate signature.

class easycv.datasets.detection.pipelines.mm_transforms.**MMMosaic**(*img_scale=(640, 640)*, *center_ratio_range=(0.5, 1.5)*, *pad_val=114*)
Bases: object

Mosaic augmentation.

Given 4 images, mosaic transform combines them into one output image. The output image is composed of the parts from each sub- image.

```
                mosaic transform
                   center_x
         +------------------------------+
         |       pad        |  pad      |
         |      +-----------+           |
         |      |           |           |
         |      |  image1   |--------+  |
         |      |           |        |  |
         |      |           | image2 |  |
center_y |----+-------------+-----------|
         |    |   cropped   |           |
         |pad |   image3    |  image4   |
         |    |             |           |
         +----|-------------+-----------+
              |             |
              +-------------+

The mosaic transform steps are as follows:

    1. Choose the mosaic center as the intersections of 4 images
```

(continues on next page)

(continued from previous page)

```
    2. Get the left top image according to the index, and randomly
       sample another 3 images from the custom dataset.
    3. Sub image will be cropped if image is larger than mosaic patch
```

Parameters

- **img_scale** (*Sequence[int]*) – Image size after mosaic pipeline of single image. Default to (640, 640).
- **center_ratio_range** (*Sequence[float]*) – Center ratio range of mosaic output. Default to (0.5, 1.5).
- **pad_val** (*int*) – Pad value. Default to 114.

__init__(*img_scale=(640, 640)*, *center_ratio_range=(0.5, 1.5)*, *pad_val=114*)
Initialize self. See help(type(self)) for accurate signature.

get_indexes(*dataset*)
Call function to collect indexes.

Parameters **dataset** (DetImagesMixDataset) – The dataset.

Returns indexes.

Return type list

class easycv.datasets.detection.pipelines.mm_transforms.**MMMixUp**(*img_scale=(640, 640)*, *ratio_range=(0.5, 1.5)*, *flip_ratio=0.5*, *pad_val=114*, *max_iters=15*, *min_bbox_size=5*, *min_area_ratio=0.2*, *max_aspect_ratio=20*)

Bases: object

MixUp data augmentation.

```
                 mixup transform
            +------------------------------+
            | mixup image   |              |
            |      +--------|--------+     |
            |      |        |        |     |
            |------------------+     |     |
            |      |                 |     |
            |      |      image      |     |
            |      |                 |     |
            |      |                 |     |
            |      |-----------------+     |
            |             pad              |
            +------------------------------+

The mixup transform steps are as follows::

   1. Another random image is picked by dataset and embedded in
      the top left patch(after padding and resizing)
```

(continues on next page)

(continued from previous page)

```
2. The target of mixup transform is the weighted average of mixup
   image and origin image.
```

Parameters

- **img_scale** (*Sequence[int]*) – Image output size after mixup pipeline. Default: (640, 640).
- **ratio_range** (*Sequence[float]*) – Scale ratio of mixup image. Default: (0.5, 1.5).
- **flip_ratio** (*float*) – Horizontal flip ratio of mixup image. Default: 0.5.
- **pad_val** (*int*) – Pad value. Default: 114.
- **max_iters** (*int*) – The maximum number of iterations. If the number of iterations is greater than *max_iters*, but gt_bbox is still empty, then the iteration is terminated. Default: 15.
- **min_bbox_size** (*float*) – Width and height threshold to filter bboxes. If the height or width of a box is smaller than this value, it will be removed. Default: 5.
- **min_area_ratio** (*float*) – Threshold of area ratio between original bboxes and wrapped bboxes. If smaller than this value, the box will be removed. Default: 0.2.
- **max_aspect_ratio** (*float*) – Aspect ratio of width and height threshold to filter bboxes. If max(h/w, w/h) larger than this value, the box will be removed. Default: 20.

__init__(*img_scale=(640, 640)*, *ratio_range=(0.5, 1.5)*, *flip_ratio=0.5*, *pad_val=114*, *max_iters=15*, *min_bbox_size=5*, *min_area_ratio=0.2*, *max_aspect_ratio=20*)

Initialize self. See help(type(self)) for accurate signature.

get_indexes(*dataset*)

Call function to collect indexes.

Parameters **dataset** (DetImagesMixDataset) – The dataset.

Returns indexes.

Return type list

class easycv.datasets.detection.pipelines.mm_transforms.**MMRandomAffine**(*max_rotate_degree=10.0*, *max_translate_ratio=0.1*, *scaling_ratio_range=(0.5, 1.5)*, *max_shear_degree=2.0*, *border=(0, 0)*, *border_val=(114, 114, 114)*, *min_bbox_size=2*, *min_area_ratio=0.2*, *max_aspect_ratio=20*)

Bases: object

Random affine transform data augmentation. for yolox

This operation randomly generates affine transform matrix which including rotation, translation, shear and scaling transforms.

Parameters

- **max_rotate_degree** (*float*) – Maximum degrees of rotation transform. Default: 10.

- **max_translate_ratio** (*float*) – Maximum ratio of translation. Default: 0.1.
- **scaling_ratio_range** (*tuple[float]*) – Min and max ratio of scaling transform. Default: (0.5, 1.5).
- **max_shear_degree** (*float*) – Maximum degrees of shear transform. Default: 2.
- **border** (*tuple[int]*) – Distance from height and width sides of input image to adjust output shape. Only used in mosaic dataset. Default: (0, 0).
- **border_val** (*tuple[int]*) – Border padding values of 3 channels. Default: (114, 114, 114).
- **min_bbox_size** (*float*) – Width and height threshold to filter bboxes. If the height or width of a box is smaller than this value, it will be removed. Default: 2.
- **min_area_ratio** (*float*) – Threshold of area ratio between original bboxes and wrapped bboxes. If smaller than this value, the box will be removed. Default: 0.2.
- **max_aspect_ratio** (*float*) – Aspect ratio of width and height threshold to filter bboxes. If max(h/w, w/h) larger than this value, the box will be removed.

__init__(*max_rotate_degree=10.0*, *max_translate_ratio=0.1*, *scaling_ratio_range=(0.5, 1.5)*, *max_shear_degree=2.0*, *border=(0, 0)*, *border_val=(114, 114, 114)*, *min_bbox_size=2*, *min_area_ratio=0.2*, *max_aspect_ratio=20*)

Initialize self. See help(type(self)) for accurate signature.

filter_gt_bboxes(*origin_bboxes*, *wrapped_bboxes*)

class easycv.datasets.detection.pipelines.mm_transforms.**MMPhotoMetricDistortion**(*brightness_delta=32*, *contrast_range=(0.5, 1.5)*, *saturation_range=(0.5, 1.5)*, *hue_delta=18*)

Bases: object

Apply photometric distortion to image sequentially, every transformation is applied with a probability of 0.5. The position of random contrast is in second or second to last.

1. random brightness
2. random contrast (mode 0)
3. convert color from BGR to HSV
4. random saturation
5. random hue
6. convert color from HSV to BGR
7. random contrast (mode 1)
8. randomly swap channels

Parameters

- **brightness_delta** (*int*) – delta of brightness.
- **contrast_range** (*tuple*) – range of contrast.
- **saturation_range** (*tuple*) – range of saturation.

- **hue_delta** (*int*) – delta of hue.

__init__(*brightness_delta=32*, *contrast_range=(0.5, 1.5)*, *saturation_range=(0.5, 1.5)*, *hue_delta=18*)
Initialize self. See help(type(self)) for accurate signature.

class easycv.datasets.detection.pipelines.mm_transforms.**MMResize**(*img_scale=None*, *multiscale_mode='range'*, *ratio_range=None*, *keep_ratio=True*, *bbox_clip_border=True*, *backend='cv2'*, *override=False*)

Bases: object

Resize images & bbox & mask. This transform resizes the input image to some scale. Bboxes and masks are then resized with the same scale factor. If the input dict contains the key "scale", then the scale in the input dict is used, otherwise the specified scale in the init method is used. If the input dict contains the key "scale_factor" (if MultiScaleFlipAug does not give img_scale but scale_factor), the actual scale will be computed by image shape and scale_factor. *img_scale* can either be a tuple (single-scale) or a list of tuple (multi-scale). There are 3 multiscale modes: - `ratio_range is not None`: randomly sample a ratio from the ratio range and multiply it with the image scale. - `ratio_range is None` and `multiscale_mode == "range"`: randomly sample a scale from the multiscale range. - `ratio_range is None` and `multiscale_mode == "value"`: randomly sample a scale from multiple scales. :param img_scale: Images scales for resizing. :type img_scale: tuple or list[tuple] :param multiscale_mode: Either "range" or "value". :type multiscale_mode: str :param ratio_range: (min_ratio, max_ratio) :type ratio_range: tuple[float] :param keep_ratio: Whether to keep the aspect ratio when resizing the

image.

Parameters

- **bbox_clip_border** (*bool, optional*) – Whether clip the objects outside the border of the image. Defaults to True.
- **backend** (*str*) – Image resize backend, choices are 'cv2' and 'pillow'. These two backends generates slightly different results. Defaults to 'cv2'.
- **override** (*bool, optional*) – Whether to override *scale* and *scale_factor* so as to call resize twice. Default False. If True, after the first resizing, the existed *scale* and *scale_factor* will be ignored so the second resizing can be allowed. This option is a work-around for multiple times of resize in DETR. Defaults to False.

__init__(*img_scale=None*, *multiscale_mode='range'*, *ratio_range=None*, *keep_ratio=True*, *bbox_clip_border=True*, *backend='cv2'*, *override=False*)
Initialize self. See help(type(self)) for accurate signature.

static random_select(*img_scales*)
Randomly select an img_scale from given candidates. :param img_scales: Images scales for selection. :type img_scales: list[tuple]

Returns Returns a tuple `(img_scale, scale_dix)`, where `img_scale` is the selected image scale and `scale_idx` is the selected index in the given candidates.

Return type (tuple, int)

static random_sample(*img_scales*)
Randomly sample an img_scale when `multiscale_mode=='range'`. :param img_scales: Images scale range for sampling.

There must be two tuples in img_scales, which specify the lower and upper bound of image scales.

Returns Returns a tuple `(img_scale, None)`, where `img_scale` is sampled scale and None is just a placeholder to be consistent with *random_select()*.

Return type (tuple, None)

static random_sample_ratio(*img_scale*, *ratio_range*)

Randomly sample an img_scale when `ratio_range` is specified. A ratio will be randomly sampled from the range specified by `ratio_range`. Then it would be multiplied with `img_scale` to generate sampled scale. :param img_scale: Images scale base to multiply with ratio. :type img_scale: tuple :param ratio_range: The minimum and maximum ratio to scale

the `img_scale`.

Returns Returns a tuple `(scale, None)`, where `scale` is sampled ratio multiplied with `img_scale` and None is just a placeholder to be consistent with *random_select()*.

Return type (tuple, None)

class easycv.datasets.detection.pipelines.mm_transforms.**MMRandomFlip**(*flip_ratio=None*, *direction='horizontal'*)

Bases: `object`

Flip the image & bbox & mask. If the input dict contains the key "flip", then the flag will be used, otherwise it will be randomly decided by a ratio specified in the init method. When random flip is enabled, `flip_ratio/direction` can either be a float/string or tuple of float/string. There are 3 flip modes: - `flip_ratio` is float, `direction` is string: the image will be

`direction``ly flipped with probability of ``flip_ratio` . E.g., `flip_ratio=0.5`, `direction='horizontal'`, then image will be horizontally flipped with probability of 0.5.

- **`flip_ratio` is float, `direction` is list of string: the image wil** be `direction[i]``ly flipped with probability of ``flip_ratio/len(direction)`. E.g., `flip_ratio=0.5`, `direction=['horizontal', 'vertical']`, then image will be horizontally flipped with probability of 0.25, vertically with probability of 0.25.
- **`flip_ratio` is list of float, `direction` is list of string:** given `len(flip_ratio) == len(direction)`, the image wil be `direction[i]``ly flipped with probability of ``flip_ratio[i]`. E.g., `flip_ratio=[0.3, 0.5]`, `direction=['horizontal', 'vertical']`, then image will be horizontally flipped with probability of 0.3, vertically with probability of 0.5.

Parameters

- **`flip_ratio`** (*float | list[float], optional*) – The flipping probability. Default: None.
- **`direction`** (*str | list[str], optional*) – The flipping direction. Options are 'horizontal', 'vertical', 'diagonal'. Default: 'horizontal'. If input is a list, the length must equal `flip_ratio`. Each element in `flip_ratio` indicates the flip probability of corresponding direction.

__init__(*flip_ratio=None*, *direction='horizontal'*)

Initialize self. See help(type(self)) for accurate signature.

bbox_flip(*bboxes*, *img_shape*, *direction*)

Flip bboxes horizontally. :param bboxes: Bounding boxes, shape (..., 4*k) :type bboxes: numpy.ndarray :param img_shape: Image shape (height, width) :type img_shape: tuple[int] :param direction: Flip direction. Options are 'horizontal',

'vertical'.

Returns Flipped bounding boxes.

Return type numpy.ndarray

class easycv.datasets.detection.pipelines.mm_transforms.**MMRandomCrop**(*crop_size*, *crop_type='absolute'*, *allow_negative_crop=False*, *recompute_bbox=False*, *bbox_clip_border=True*)

Bases: object

Random crop the image & bboxes & masks.

The absolute *crop_size* is sampled based on *crop_type* and *image_size*, then the cropped results are generated.

Parameters

- **crop_size** (*tuple*) – The relative ratio or absolute pixels of height and width.
- **crop_type** (*str, optional*) – one of "relative_range", "relative", "absolute", "absolute_range". "relative" randomly crops (h * crop_size[0], w * crop_size[1]) part from an input of size (h, w). "relative_range" uniformly samples relative crop size from range [crop_size[0], 1] and [crop_size[1], 1] for height and width respectively. "absolute" crops from an input with absolute size (crop_size[0], crop_size[1]). "absolute_range" uniformly samples crop_h in range [crop_size[0], min(h, crop_size[1])] and crop_w in range [crop_size[0], min(w, crop_size[1])]. Default "absolute".
- **allow_negative_crop** (*bool, optional*) – Whether to allow a crop that does not contain any bbox area. Default False.
- **recompute_bbox** (*bool, optional*) – Whether to re-compute the boxes based on cropped instance masks. Default False.
- **bbox_clip_border** (*bool, optional*) – Whether clip the objects outside the border of the image. Defaults to True.

Note:

- **If the image is smaller than the absolute crop size, return the** original image.
- The keys for bboxes, labels and masks must be aligned. That is, *gt_bboxes* corresponds to *gt_labels* and *gt_masks*, and *gt_bboxes_ignore* corresponds to *gt_labels_ignore* and *gt_masks_ignore*.
- If the crop does not contain any gt-bbox region and *allow_negative_crop* is set to False, skip this image.

__init__(*crop_size*, *crop_type='absolute'*, *allow_negative_crop=False*, *recompute_bbox=False*, *bbox_clip_border=True*)

Initialize self. See help(type(self)) for accurate signature.

class easycv.datasets.detection.pipelines.mm_transforms.**MMPad**(*size=None*, *size_divisor=None*, *pad_to_square=False*, *pad_val=0*, *seg_pad_val=255*)

Bases: object

Pad the image & mask. There are two padding modes: (1) pad to a fixed size and (2) pad to the minimum size that is divisible by some number. Added keys are "pad_shape", "pad_fixed_size", "pad_size_divisor", :param size: Fixed padding size. :type size: tuple, optional :param size_divisor: The divisor of padded size. :type size_divisor: int, optional :param pad_to_square: Whether to pad the image into a square.

Currently only used for YOLOX. Default: False.

Parameters

- **pad_val** (*float, optional*) – Padding value, 0 by default.
- **seg_pad_val** (*float, optional*) – Padding value of segmentation map. Default: 255.

__init__(*size=None*, *size_divisor=None*, *pad_to_square=False*, *pad_val=0*, *seg_pad_val=255*)
Initialize self. See help(type(self)) for accurate signature.

class easycv.datasets.detection.pipelines.mm_transforms.**MMNormalize**(*mean*, *std*, *to_rgb=True*)
Bases: object

Normalize the image.

Added key is "img_norm_cfg".

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, default is true.

__init__(*mean*, *std*, *to_rgb=True*)
Initialize self. See help(type(self)) for accurate signature.

class easycv.datasets.detection.pipelines.mm_transforms.**LoadImageFromFile**(*to_float32=False*, *color_type='color'*, *file_client_args={'backend': 'disk'}*)
Bases: object

Load an image from file. Required keys are "img_prefix" and "img_info" (a dict that must contain the key "filename"). Added or updated keys are "filename", "img", "img_shape", "ori_shape" (same as *img_shape*), "pad_shape" (same as *img_shape*), "scale_factor" (1.0) and "img_norm_cfg" (means=0 and stds=1). :param to_float32: 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.

Parameters

- **color_type** (*str*) – The flag argument for mmcv.imfrombytes(). Defaults to 'color'.
- **file_client_args** (*dict*) – Arguments to instantiate a FileClient. See mmcv.fileio.FileClient for details. Defaults to dict(backend='disk').

__init__(*to_float32=False*, *color_type='color'*, *file_client_args={'backend': 'disk'}*)
Initialize self. See help(type(self)) for accurate signature.

class easycv.datasets.detection.pipelines.mm_transforms.**LoadMultiChannelImageFromFiles**(*to_float32=False*, *color_type='unchang file_client_args={'ba 'disk'}*)
Bases: object

Load multi-channel images from a list of separate channel files.

Required keys are "img_prefix" and "img_info" (a dict that must contain the key "filename", which is expected to be a list of filenames). Added or updated keys are "filename", "img", "img_shape", "ori_shape" (same as *img_shape*), "pad_shape" (same as *img_shape*), "scale_factor" (1.0) and "img_norm_cfg" (means=0 and stds=1).

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'.
- **file_client_args** (*dict*) – Arguments to instantiate a FileClient. See `mmcv.fileio.FileClient` for details. Defaults to `dict(backend='disk')`.

__init__(*to_float32=False*, *color_type='unchanged'*, *file_client_args={'backend': 'disk'}*)
Initialize self. See help(type(self)) for accurate signature.

class easycv.datasets.detection.pipelines.mm_transforms.**LoadAnnotations**(*with_bbox=True*, *with_label=True*, *with_mask=False*, *with_seg=False*, *poly2mask=True*, *file_client_args={'backend': 'disk'}*)

Bases: `object`

Load multiple types of annotations. :param with_bbox: Whether to parse and load the bbox annotation.

Default: True.

Parameters

- **with_label** (*bool*) – Whether to parse and load the label annotation. Default: True.
- **with_mask** (*bool*) – Whether to parse and load the mask annotation. Default: False.
- **with_seg** (*bool*) – Whether to parse and load the semantic segmentation annotation. Default: False.
- **poly2mask** (*bool*) – Whether to convert the instance masks from polygons to bitmaps. Default: True.
- **file_client_args** (*dict*) – Arguments to instantiate a FileClient. See `mmcv.fileio.FileClient` for details. Defaults to `dict(backend='disk')`.

__init__(*with_bbox=True*, *with_label=True*, *with_mask=False*, *with_seg=False*, *poly2mask=True*, *file_client_args={'backend': 'disk'}*)
Initialize self. See help(type(self)) for accurate signature.

process_polygons(*polygons*)
Convert polygons to list of ndarray and filter invalid polygons. :param polygons: Polygons of one instance. :type polygons: list[list]

Returns Processed polygons.

Return type list[numpy.ndarray]

class easycv.datasets.detection.pipelines.mm_transforms.**MMFilterAnnotations**(*min_gt_bbox_wh*, *keep_empty=True*)

Bases: `object`

Filter invalid annotations. :param min_gt_bbox_wh: Minimum width and height of ground truth boxes.

Parameters **keep_empty** (*bool*) – Whether to return None when it becomes an empty bbox after filtering. Default: True

__init__(*min_gt_bbox_wh*, *keep_empty=True*)
Initialize self. See help(type(self)) for accurate signature.

class easycv.datasets.detection.pipelines.mm_transforms.**MMMultiScaleFlipAug**(*transforms*, *img_scale=None*, *scale_factor=None*, *flip=False*, *flip_direction='horizontal'*)

Bases: object

Test-time augmentation with multiple scales and flipping.

An example configuration is as followed:

```
img_scale=[(1333, 400), (1333, 800)],
flip=True,
transforms=[
    dict(type='Resize', keep_ratio=True),
    dict(type='RandomFlip'),
    dict(type='Normalize', **img_norm_cfg),
    dict(type='Pad', size_divisor=32),
    dict(type='ImageToTensor', keys=['img']),
    dict(type='Collect', keys=['img']),
]
```

After MultiScaleFLipAug with above configuration, the results are wrapped into lists of the same length as followed:

```
dict(
    img=[...],
    img_shape=[...],
    scale=[(1333, 400), (1333, 400), (1333, 800), (1333, 800)]
    flip=[False, True, False, True]
    ...
)
```

Parameters

- **transforms** (*list[dict]*) – Transforms to apply in each augmentation.
- **img_scale** (*tuple | list[tuple] | None*) – Images scales for resizing.
- **scale_factor** (*float | list[float] | None*) – Scale factors for resizing.
- **flip** (*bool*) – Whether apply flip augmentation. Default: 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. Default: "horizontal".

__init__(*transforms*, *img_scale=None*, *scale_factor=None*, *flip=False*, *flip_direction='horizontal'*)
Initialize self. See help(type(self)) for accurate signature.

Submodules

easycv.datasets.detection.mix module

class easycv.datasets.detection.mix.**DetImagesMixDataset**(*data_source*, *pipeline*, *dynamic_scale=None*, *skip_type_keys=None*, *profiling=False*, *classes=None*, *yolo_format=True*, *label_padding=True*)

Bases: `Generic[torch.utils.data.dataset.T_co]`

A wrapper of multiple images mixed dataset.

Suitable for training on multiple images mixed data augmentation like mosaic and mixup. For the augmentation pipeline of mixed image data, the *get_indexes* method needs to be provided to obtain the image indexes, and you can set *skip_flags* to change the pipeline running process. At the same time, we provide the *dynamic_scale* parameter to dynamically change the output image size.

output boxes format: cx, cy, w, h

Parameters

- **data_source** (`DetSourceCoco`) – The dataset to be mixed.
- **pipeline** (*Sequence[dict]*) – Sequence of transform object or config dict to be composed.
- **dynamic_scale** (*tuple[int], optional*) – The image scale can be changed dynamically. Default to None.
- **skip_type_keys** (*list[str], optional*) – Sequence of type string to be skip pipeline. Default to None.
- **label_padding** – out labeling padding [N, 120, 5]

__init__(*data_source*, *pipeline*, *dynamic_scale=None*, *skip_type_keys=None*, *profiling=False*, *classes=None*, *yolo_format=True*, *label_padding=True*)
Args: data_source: Data_source config dict pipeline: Pipeline config list profiling: If set True, will print pipeline time classes: A list of class names, used in evaluation for result and groundtruth visualization

update_skip_type_keys(*skip_type_keys*)
Update skip_type_keys. It is called by an external hook.

Parameters **skip_type_keys** (*list[str], optional*) – Sequence of type string to be skip pipeline.

update_dynamic_scale(*dynamic_scale*)
Update dynamic_scale. It is called by an external hook.

Parameters **dynamic_scale** (*tuple[int]*) – The image scale can be changed dynamically.

results2json(*results*, *outfile_prefix*)
Dump the detection results to a COCO style json file.

There are 3 types of results: proposals, bbox predictions, mask predictions, and they have different data types. This method will automatically recognize the type, and dump them to json files.

Parameters

- **results** (*list[list | tuple | ndarray]*) – Testing results of the dataset.
- **outfile_prefix** (*str*) – The filename prefix of the json files. If the prefix is "somepath/xxx", the json files will be named "somepath/xxx.bbox.json", "somepath/xxx.segm.json", "somepath/xxx.proposal.json".

Returns str]: Possible keys are "bbox", "segm", "proposal", and values are corresponding filenames.

Return type dict[str

format_results(*results*, *jsonfile_prefix=None*, ***kwargs*)
Format the results to json (standard format for COCO evaluation).

Parameters

- **results** (*list[tuple | numpy.ndarray]*) – Testing results of the dataset.
- **jsonfile_prefix** (*str | None*) – The prefix of json files. It includes the file path and the prefix of filename, e.g., "a/b/prefix". If not specified, a temp file will be created. Default: None.

Returns (result_files, tmp_dir), result_files is a dict containing the json filepaths, tmp_dir is the temporal directory created for saving json files when jsonfile_prefix is not specified.

Return type tuple

easycv.datasets.detection.raw module

class easycv.datasets.detection.raw.**DetDataset**(*data_source*, *pipeline*, *profiling=False*, *classes=None*)
Bases: Generic[torch.utils.data.dataset.T_co]

Dataset for Detection

__init__(*data_source*, *pipeline*, *profiling=False*, *classes=None*)

Parameters

- **data_source** – Data_source config dict
- **pipeline** – Pipeline config list
- **profiling** – If set True, will print pipeline time
- **classes** – A list of class names, used in evaluation for result and groundtruth visualization

evaluate(*results*, *evaluators=None*, *logger=None*)
Evaluates the detection boxes. :param results: A dictionary containing

detection_boxes: List of length number of test images. Float32 numpy array of shape [num_boxes, 4] and format [ymin, xmin, ymax, xmax] in absolute image coordinates.

detection_scores: List of length number of test images, detection scores for the boxes, float32 numpy array of shape [num_boxes].

detection_classes: List of length number of test images, integer numpy array of shape [num_boxes] containing 1-indexed detection classes for the boxes.

img_metas: List of length number of test images, dict of image meta info, containing filename, img_shape, origin_img_shape, scale_factor and so on.

Parameters **evaluators** – evaluators to calculate metric with results and groundtruth_dict

visualize(*results*, *vis_num=10*, *score_thr=0.3*, ***kwargs*)
Visulaize the model output on validation data. :param results: A dictionary containing

detection_boxes: List of length number of test images. Float32 numpy array of shape [num_boxes, 4] and format [ymin, xmin, ymax, xmax] in absolute image coordinates.

detection_scores: List of length number of test images, detection scores for the boxes, float32 numpy array of shape [num_boxes].

detection_classes: List of length number of test images, integer numpy array of shape [num_boxes] containing 1-indexed detection classes for the boxes.

img_metas: List of length number of test images, dict of image meta info, containing filename, img_shape, origin_img_shape, scale_factor and so on.

Parameters

- **vis_num** – number of images visualized
- **score_thr** – The threshold to filter box, boxes with scores greater than score_thr will be kept.

Returns: A dictionary containing images: Visulaized images. img_metas: List of length number of test images,

dict of image meta info, containing filename, img_shape, origin_img_shape, scale_factor and so on.

14.1.3 easycv.datasets.loader package

class easycv.datasets.loader.**GroupSampler**(*dataset*, *samples_per_gpu=1*)
Bases: Generic[torch.utils.data.sampler.T_co]

__init__(*dataset*, *samples_per_gpu=1*)
Initialize self. See help(type(self)) for accurate signature.

class easycv.datasets.loader.**DistributedGroupSampler**(*dataset*, *samples_per_gpu=1*, *num_replicas=None*, *rank=None*)
Bases: Generic[torch.utils.data.sampler.T_co]

Sampler that restricts data loading to a subset of the dataset. It is especially useful in conjunction with torch.nn.parallel.DistributedDataParallel. In such case, each process can pass a DistributedSampler instance as a DataLoader sampler, and load a subset of the original dataset that is exclusive to it. .. note:

```
Dataset is assumed to be of constant size.
```

Parameters

- **dataset** – Dataset used for sampling.
- **num_replicas** (*optional*) – Number of processes participating in distributed training.
- **rank** (*optional*) – Rank of the current process within num_replicas.

__init__(*dataset*, *samples_per_gpu=1*, *num_replicas=None*, *rank=None*)
Initialize self. See help(type(self)) for accurate signature.

set_epoch(*epoch*)

easycv.datasets.loader.**build_dataloader**(*dataset*, *imgs_per_gpu*, *workers_per_gpu*, *num_gpus=1*, *dist=True*, *shuffle=True*, *replace=False*, *seed=None*, *reuse_worker_cache=False*, *odps_config=None*, *persistent_workers=False*, ***kwargs*)

Build PyTorch DataLoader. In distributed training, each GPU/process has a dataloader. In non-distributed training, there is only one dataloader for all GPUs. :param dataset: A PyTorch dataset. :type dataset: Dataset :param imgs_per_gpu: Number of images on each GPU, i.e., batch size of

each GPU.

Parameters

- **workers_per_gpu** (*int*) – How many subprocesses to use for data loading for each GPU.
- **num_gpus** (*int*) – Number of GPUs. Only used in non-distributed training.
- **dist** (*bool*) – Distributed training/test or not. Default: True.
- **shuffle** (*bool*) – Whether to shuffle the data at every epoch. Default: True.
- **replace** (*bool*) – Replace or not in random shuffle. It works on when shuffle is True.
- **reuse_worker_cache** (*bool*) – If set true, will reuse worker process so that cached data in worker process can be reused.
- **persistent_workers** (*bool*) – After pytorch1.7, could use persistent_workers=True to avoid reconstruct dataworker before each epoch, speed up before epoch
- **kwargs** – any keyword argument to be used to initialize DataLoader

Returns A PyTorch dataloader.

Return type DataLoader

class easycv.datasets.loader.**DistributedGivenIterationSampler**(*dataset*, *total_iter*, *batch_size*, *num_replicas=None*, *rank=None*, *last_iter=- 1*)

Bases: Generic[torch.utils.data.sampler.T_co]

__init__(*dataset*, *total_iter*, *batch_size*, *num_replicas=None*, *rank=None*, *last_iter=- 1*)

Initialize self. See help(type(self)) for accurate signature.

set_uniform_indices(*labels*, *num_classes*)

gen_new_list()

set_epoch(*epoch*)

Submodules

easycv.datasets.loader.build_loader module

easycv.datasets.loader.build_loader.**build_dataloader**(*dataset*, *imgs_per_gpu*, *workers_per_gpu*, *num_gpus=1*, *dist=True*, *shuffle=True*, *replace=False*, *seed=None*, *reuse_worker_cache=False*, *odps_config=None*, *persistent_workers=False*, ***kwargs*)

Build PyTorch DataLoader. In distributed training, each GPU/process has a dataloader. In non-distributed training, there is only one dataloader for all GPUs. :param dataset: A PyTorch dataset. :type dataset: Dataset :param imgs_per_gpu: Number of images on each GPU, i.e., batch size of

each GPU.

Parameters

- **workers_per_gpu** (*int*) – How many subprocesses to use for data loading for each GPU.
- **num_gpus** (*int*) – Number of GPUs. Only used in non-distributed training.
- **dist** (*bool*) – Distributed training/test or not. Default: True.
- **shuffle** (*bool*) – Whether to shuffle the data at every epoch. Default: True.
- **replace** (*bool*) – Replace or not in random shuffle. It works on when shuffle is True.
- **reuse_worker_cache** (*bool*) – If set true, will reuse worker process so that cached data in worker process can be reused.
- **persistent_workers** (*bool*) – After pytorch1.7, could use persistent_workers=True to avoid reconstruct dataworker before each epoch, speed up before epoch
- **kwargs** – any keyword argument to be used to initialize DataLoader

Returns A PyTorch dataloader.

Return type DataLoader

easycv.datasets.loader.build_loader.**worker_init_fn**(*worker_id*, *seed=None*, *odps_config=None*)

class easycv.datasets.loader.build_loader.**InfiniteDataLoader**(**args*, ***kwargs*)

Bases: Generic[torch.utils.data.dataloader.T_co]

Dataloader that reuses workers. https://github.com/pytorch/pytorch/issues/15849 Uses same syntax as vanilla DataLoader.

__init__(**args*, ***kwargs*)

Initialize self. See help(type(self)) for accurate signature.

dataset: torch.utils.data.dataset.Dataset[torch.utils.data.dataloader.T_co]

batch_size: Optional[int]

num_workers: int

pin_memory: bool

drop_last: bool

timeout: float

sampler: Union[torch.utils.data.sampler.Sampler, Iterable]

pin_memory_device: str

prefetch_factor: int

easycv.datasets.loader.sampler module

class easycv.datasets.loader.sampler.**DistributedMPSampler**(*dataset*, *num_replicas=None*, *rank=None*, *shuffle=True*, *split_huge_listfile_byrank=False*)

Bases: torch.utils.data.sampler.Sampler[torch.utils.data.distributed.T_co]

__init__(*dataset*, *num_replicas=None*, *rank=None*, *shuffle=True*, *split_huge_listfile_byrank=False*)

A Distribute sampler which support sample m instance from one class once for classification dataset dataset: pytorch dataset object num_replicas (optional): Number of processes participating in

distributed training.

rank (optional): Rank of the current process within num_replicas. shuffle (optional): If true (default), sampler will shuffle the indices split_huge_listfile_byrank: if split, return all indice for each rank, because list for each rank has been

split before build dataset in dist training

generate_indice()

get_label_dict()

calculate_this_label_list()

class easycv.datasets.loader.sampler.**DistributedSampler**(*dataset*, *num_replicas=None*, *rank=None*, *shuffle=True*, *replace=False*, *split_huge_listfile_byrank=False*)

Bases: torch.utils.data.sampler.Sampler[torch.utils.data.distributed.T_co]

__init__(*dataset*, *num_replicas=None*, *rank=None*, *shuffle=True*, *replace=False*, *split_huge_listfile_byrank=False*)

A Distribute sampler which support sample m instance from one class once for classification dataset :param dataset: pytorch dataset object :param num_replicas: Number of processes participating in

distributed training.

Parameters

- **rank** (*optional*) – Rank of the current process within num_replicas.
- **shuffle** (*optional*) – If true (default), sampler will shuffle the indices
- **split_huge_listfile_byrank** – if split, return all indice for each rank, because list for each rank has been split before build dataset in dist training

generate_new_list()

set_uniform_indices(*labels*, *num_classes*)

class easycv.datasets.loader.sampler.**GroupSampler**(*dataset*, *samples_per_gpu=1*)

Bases: Generic[torch.utils.data.sampler.T_co]

__init__(*dataset*, *samples_per_gpu=1*)

Initialize self. See help(type(self)) for accurate signature.

class easycv.datasets.loader.sampler.**DistributedGroupSampler**(*dataset*, *samples_per_gpu=1*, *num_replicas=None*, *rank=None*)

Bases: Generic[torch.utils.data.sampler.T_co]

Sampler that restricts data loading to a subset of the dataset. It is especially useful in conjunction with torch.nn.parallel.DistributedDataParallel. In such case, each process can pass a DistributedSampler instance as a DataLoader sampler, and load a subset of the original dataset that is exclusive to it. .. note:

```
Dataset is assumed to be of constant size.
```

Parameters

- **dataset** – Dataset used for sampling.
- **num_replicas** (*optional*) – Number of processes participating in distributed training.
- **rank** (*optional*) – Rank of the current process within num_replicas.

__init__(*dataset*, *samples_per_gpu=1*, *num_replicas=None*, *rank=None*)
Initialize self. See help(type(self)) for accurate signature.

set_epoch(*epoch*)

class easycv.datasets.loader.sampler.**DistributedGivenIterationSampler**(*dataset*, *total_iter*, *batch_size*, *num_replicas=None*, *rank=None*, *last_iter=-1*)

Bases: Generic[torch.utils.data.sampler.T_co]

__init__(*dataset*, *total_iter*, *batch_size*, *num_replicas=None*, *rank=None*, *last_iter=- 1*)
Initialize self. See help(type(self)) for accurate signature.

set_uniform_indices(*labels*, *num_classes*)

gen_new_list()

set_epoch(*epoch*)

14.1.4 easycv.datasets.pose package

class easycv.datasets.pose.**PoseTopDownDataset**(*data_source*, *pipeline*, *profiling=False*)
Bases: Generic[torch.utils.data.dataset.T_co]

PoseTopDownDataset dataset for top-down pose estimation. The dataset loads raw features and apply specified transforms to return a dict containing the image tensors and other information.

Parameters

- **data_source** – Data_source config dict
- **pipeline** – Pipeline config list
- **profiling** – If set True, will print pipeline time

__init__(*data_source*, *pipeline*, *profiling=False*)
Initialize self. See help(type(self)) for accurate signature.

evaluate(*outputs*, *evaluators*, ***kwargs*)

Subpackages

easycv.datasets.pose.data_sources package

class easycv.datasets.pose.data_sources.**PoseTopDownSourceCoco**(*ann_file*, *img_prefix*, *data_cfg*, *dataset_info=None*, *test_mode=False*)

Bases: *easycv.datasets.pose.data_sources.top_down.PoseTopDownSource*

CocoSource for top-down pose estimation.

Microsoft COCO: Common Objects in Context' ECCV'2014 More details can be found in the `paper .

The source loads raw features to build a data meta object containing the image info, annotation info and others.

COCO keypoint indexes:

```
0: 'nose',
1: 'left_eye',
2: 'right_eye',
3: 'left_ear',
4: 'right_ear',
5: 'left_shoulder',
6: 'right_shoulder',
7: 'left_elbow',
8: 'right_elbow',
9: 'left_wrist',
10: 'right_wrist',
11: 'left_hip',
12: 'right_hip',
13: 'left_knee',
14: 'right_knee',
15: 'left_ankle',
16: 'right_ankle'
```

Parameters

- **ann_file** (*str*) – Path to the annotation file.
- **img_prefix** (*str*) – Path to a directory where images are held. Default: None.
- **data_cfg** (*dict*) – config
- **dataset_info** (DatasetInfo) – A class containing all dataset info.
- **test_mode** (*bool*) – Store True when building test or
- **dataset. Default** (*validation*) – False.

__init__(*ann_file*, *img_prefix*, *data_cfg*, *dataset_info=None*, *test_mode=False*)
Initialize self. See help(type(self)) for accurate signature.

class easycv.datasets.pose.data_sources.**PoseTopDownSource**(*ann_file*, *img_prefix*, *data_cfg*, *dataset_info*, *coco_style=True*, *test_mode=False*)

Bases: object

Class for keypoint 2D top-down pose estimation with single-view RGB image as the data source.

Parameters

- **ann_file** (*str*) – Path to the annotation file.
- **img_prefix** (*str*) – Path to a directory where images are held. Default: None.
- **data_cfg** (*dict*) – config
- **dataset_info** (DatasetInfo) – A class containing all dataset info.
- **coco_style** (*bool*) – Whether the annotation json is coco-style. Default: True
- **test_mode** (*bool*) – Store True when building test or validation dataset. Default: False.

__init__(*ann_file*, *img_prefix*, *data_cfg*, *dataset_info*, *coco_style=True*, *test_mode=False*)
Initialize self. See help(type(self)) for accurate signature.

load_image(*image_file*)

get_length()
Get the size of the dataset.

get_sample(*idx*)

Submodules

easycv.datasets.pose.data_sources.coco module

class easycv.datasets.pose.data_sources.coco.**PoseTopDownSourceCoco**(*ann_file*, *img_prefix*, *data_cfg*, *dataset_info=None*, *test_mode=False*)

Bases: *easycv.datasets.pose.data_sources.top_down.PoseTopDownSource*

CocoSource for top-down pose estimation.

Microsoft COCO: Common Objects in Context' ECCV'2014 More details can be found in the `paper .

The source loads raw features to build a data meta object containing the image info, annotation info and others.

COCO keypoint indexes:

```
0: 'nose',
1: 'left_eye',
2: 'right_eye',
3: 'left_ear',
4: 'right_ear',
5: 'left_shoulder',
6: 'right_shoulder',
7: 'left_elbow',
8: 'right_elbow',
9: 'left_wrist',
10: 'right_wrist',
11: 'left_hip',
12: 'right_hip',
13: 'left_knee',
14: 'right_knee',
15: 'left_ankle',
16: 'right_ankle'
```

Parameters

- **ann_file** (*str*) – Path to the annotation file.
- **img_prefix** (*str*) – Path to a directory where images are held. Default: None.
- **data_cfg** (*dict*) – config
- **dataset_info** (DatasetInfo) – A class containing all dataset info.
- **test_mode** (*bool*) – Store True when building test or
- **dataset. Default** (*validation*) – False.

__init__(*ann_file*, *img_prefix*, *data_cfg*, *dataset_info=None*, *test_mode=False*)
Initialize self. See help(type(self)) for accurate signature.

easycv.datasets.pose.data_sources.top_down module

class easycv.datasets.pose.data_sources.top_down.**DatasetInfo**(*dataset_info*)
Bases: object

__init__(*dataset_info*)
Initialize self. See help(type(self)) for accurate signature.

class easycv.datasets.pose.data_sources.top_down.**PoseTopDownSource**(*ann_file*, *img_prefix*, *data_cfg*, *dataset_info*, *coco_style=True*, *test_mode=False*)
Bases: object

Class for keypoint 2D top-down pose estimation with single-view RGB image as the data source.

Parameters

- **ann_file** (*str*) – Path to the annotation file.
- **img_prefix** (*str*) – Path to a directory where images are held. Default: None.
- **data_cfg** (*dict*) – config
- **dataset_info** (DatasetInfo) – A class containing all dataset info.
- **coco_style** (*bool*) – Whether the annotation json is coco-style. Default: True
- **test_mode** (*bool*) – Store True when building test or validation dataset. Default: False.

__init__(*ann_file*, *img_prefix*, *data_cfg*, *dataset_info*, *coco_style=True*, *test_mode=False*)
Initialize self. See help(type(self)) for accurate signature.

load_image(*image_file*)

get_length()
Get the size of the dataset.

get_sample(*idx*)

easycv.datasets.pose.pipelines package

class easycv.datasets.pose.pipelines.**PoseCollect**(*keys*, *meta_keys*, *meta_name='img_metas'*)
Bases: object

Collect data from the loader relevant to the specific task.

This keeps the items in *keys* as it is, and collect items in *meta_keys* into a meta item called *meta_name*.This is usually the last stage of the data loader pipeline. For example, when keys='imgs', meta_keys=('filename', 'label', 'original_shape'), meta_name='img_metas', the results will be a dict with keys 'imgs' and 'img_metas', where 'img_metas' is a DataContainer of another dict with keys 'filename', 'label', 'original_shape'.

Parameters

- **keys** (*Sequence[str|tuple]*) – Required keys to be collected. If a tuple (key, key_new) is given as an element, the item retrieved by key will be renamed as key_new in collected data.
- **meta_name** (*str*) – The name of the key that contains meta information. This key is always populated. Default: "img_metas".
- **meta_keys** (*Sequence[str|tuple]*) – Keys that are collected under meta_name. The contents of the *meta_name* dictionary depends on *meta_keys*.

__init__(*keys*, *meta_keys*, *meta_name='img_metas'*)
Initialize self. See help(type(self)) for accurate signature.

class easycv.datasets.pose.pipelines.**TopDownRandomFlip**(*flip_prob=0.5*)
Bases: object

Data augmentation with random image flip.

Required keys: 'img', 'joints_3d', 'joints_3d_visible', 'center' and 'ann_info'. Modifies key: 'img', 'joints_3d', 'joints_3d_visible', 'center' and 'flipped'.

Parameters

- **flip** (*bool*) – Option to perform random flip.
- **flip_prob** (*float*) – Probability of flip.

__init__(*flip_prob=0.5*)
Initialize self. See help(type(self)) for accurate signature.

class easycv.datasets.pose.pipelines.**TopDownHalfBodyTransform**(*num_joints_half_body=8*, *prob_half_body=0.3*)
Bases: object

Data augmentation with half-body transform. Keep only the upper body or the lower body at random.

Required keys: 'joints_3d', 'joints_3d_visible', and 'ann_info'. Modifies key: 'scale' and 'center'.

Parameters

- **num_joints_half_body** (*int*) – Threshold of performing half-body transform. If the body has fewer number of joints (< num_joints_half_body), ignore this step.
- **prob_half_body** (*float*) – Probability of half-body transform.

__init__(*num_joints_half_body=8*, *prob_half_body=0.3*)
Initialize self. See help(type(self)) for accurate signature.

static **half_body_transform**(*cfg*, *joints_3d*, *joints_3d_visible*)
Get center&scale for half-body transform.

class easycv.datasets.pose.pipelines.**TopDownGetRandomScaleRotation**(*rot_factor=40*, *scale_factor=0.5*, *rot_prob=0.6*)

Bases: object

Data augmentation with random scaling & rotating.

Required key: 'scale'. Modifies key: 'scale' and 'rotation'.

Parameters

- **rot_factor** (*int*) – Rotating to [-2*rot_factor, 2*rot_factor].
- **scale_factor** (*float*) – Scaling to [1-scale_factor, 1+scale_factor].
- **rot_prob** (*float*) – Probability of random rotation.

__init__(*rot_factor=40*, *scale_factor=0.5*, *rot_prob=0.6*)
Initialize self. See help(type(self)) for accurate signature.

class easycv.datasets.pose.pipelines.**TopDownAffine**(*use_udp=False*)
Bases: object

Affine transform the image to make input.

Required keys:'img', 'joints_3d', 'joints_3d_visible', 'ann_info','scale', 'rotation' and 'center'. Modified keys:'img', 'joints_3d', and 'joints_3d_visible'.

Parameters **use_udp** (*bool*) – To use unbiased data processing. Paper ref: Huang et al. The Devil is in the Details: Delving into Unbiased Data Processing for Human Pose Estimation (CVPR 2020).

__init__(*use_udp=False*)
Initialize self. See help(type(self)) for accurate signature.

class easycv.datasets.pose.pipelines.**TopDownGenerateTarget**(*sigma=2*, *kernel=(11, 11)*, *valid_radius_factor=0.0546875*, *target_type='GaussianHeatmap'*, *encoding='MSRA'*, *unbiased_encoding=False*)

Bases: object

Generate the target heatmap.

Required keys: 'joints_3d', 'joints_3d_visible', 'ann_info'. Modified keys: 'target', and 'target_weight'.

Parameters

- **sigma** – Sigma of heatmap gaussian for 'MSRA' approach.
- **kernel** – Kernel of heatmap gaussian for 'Megvii' approach.
- **encoding** (*str*) – Approach to generate target heatmaps. Currently supported approaches: 'MSRA', 'Megvii', 'UDP'. Default:'MSRA'
- **unbiased_encoding** (*bool*) – Option to use unbiased encoding methods. Paper ref: Zhang et al. Distribution-Aware Coordinate Representation for Human Pose Estimation (CVPR 2020).
- **keypoint_pose_distance** – Keypoint pose distance for UDP. Paper ref: Huang et al. The Devil is in the Details: Delving into Unbiased Data Processing for Human Pose Estimation (CVPR 2020).

- **target_type** (*str*) – supported targets: 'GaussianHeatmap', 'CombinedTarget'. Default:'GaussianHeatmap' CombinedTarget: The combination of classification target (response map) and regression target (offset map). Paper ref: Huang et al. The Devil is in the Details: Delving into Unbiased Data Processing for Human Pose Estimation (CVPR 2020).

__init__(*sigma=2*, *kernel=(11, 11)*, *valid_radius_factor=0.0546875*, *target_type='GaussianHeatmap'*, *encoding='MSRA'*, *unbiased_encoding=False*)
Initialize self. See help(type(self)) for accurate signature.

class easycv.datasets.pose.pipelines.**TopDownGenerateTargetRegression**
Bases: object

Generate the target regression vector (coordinates).

Required keys: 'joints_3d', 'joints_3d_visible', 'ann_info'. Modified keys: 'target', and 'target_weight'.

__init__()
Initialize self. See help(type(self)) for accurate signature.

class easycv.datasets.pose.pipelines.**TopDownRandomTranslation**(*trans_factor=0.15*, *trans_prob=1.0*)
Bases: object

Data augmentation with random translation.

Required key: 'scale' and 'center'. Modifies key: 'center'.

Notes

bbox height: H bbox width: W

Parameters

- **trans_factor** (*float*) – Translating center to `[-trans_factor, trans_factor] * [W, H] + center`.
- **trans_prob** (*float*) – Probability of random translation.

__init__(*trans_factor=0.15*, *trans_prob=1.0*)
Initialize self. See help(type(self)) for accurate signature.

Submodules

easycv.datasets.pose.pipelines.transforms module

class easycv.datasets.pose.pipelines.transforms.**PoseCollect**(*keys*, *meta_keys*, *meta_name='img_metas'*)
Bases: object

Collect data from the loader relevant to the specific task.

This keeps the items in *keys* as it is, and collect items in *meta_keys* into a meta item called *meta_name*.This is usually the last stage of the data loader pipeline. For example, when keys='imgs', meta_keys=('filename', 'label', 'original_shape'), meta_name='img_metas', the results will be a dict with keys 'imgs' and 'img_metas', where 'img_metas' is a DataContainer of another dict with keys 'filename', 'label', 'original_shape'.

Parameters

- **keys** (*Sequence[str|tuple]*) – Required keys to be collected. If a tuple (key, key_new) is given as an element, the item retrieved by key will be renamed as key_new in collected data.
- **meta_name** (*str*) – The name of the key that contains meta information. This key is always populated. Default: "img_metas".
- **meta_keys** (*Sequence[str|tuple]*) – Keys that are collected under meta_name. The contents of the *meta_name* dictionary depends on *meta_keys*.

__init__(*keys*, *meta_keys*, *meta_name='img_metas'*)
Initialize self. See help(type(self)) for accurate signature.

class easycv.datasets.pose.pipelines.transforms.**TopDownRandomFlip**(*flip_prob=0.5*)
Bases: object

Data augmentation with random image flip.

Required keys: 'img', 'joints_3d', 'joints_3d_visible', 'center' and 'ann_info'. Modifies key: 'img', 'joints_3d', 'joints_3d_visible', 'center' and 'flipped'.

Parameters

- **flip** (*bool*) – Option to perform random flip.
- **flip_prob** (*float*) – Probability of flip.

__init__(*flip_prob=0.5*)
Initialize self. See help(type(self)) for accurate signature.

class easycv.datasets.pose.pipelines.transforms.**TopDownHalfBodyTransform**(*num_joints_half_body=8*, *prob_half_body=0.3*)
Bases: object

Data augmentation with half-body transform. Keep only the upper body or the lower body at random.

Required keys: 'joints_3d', 'joints_3d_visible', and 'ann_info'. Modifies key: 'scale' and 'center'.

Parameters

- **num_joints_half_body** (*int*) – Threshold of performing half-body transform. If the body has fewer number of joints (< num_joints_half_body), ignore this step.
- **prob_half_body** (*float*) – Probability of half-body transform.

__init__(*num_joints_half_body=8*, *prob_half_body=0.3*)
Initialize self. See help(type(self)) for accurate signature.

static half_body_transform(*cfg*, *joints_3d*, *joints_3d_visible*)
Get center&scale for half-body transform.

class easycv.datasets.pose.pipelines.transforms.**TopDownGetRandomScaleRotation**(*rot_factor=40*, *scale_factor=0.5*, *rot_prob=0.6*)
Bases: object

Data augmentation with random scaling & rotating.

Required key: 'scale'. Modifies key: 'scale' and 'rotation'.

Parameters

- **rot_factor** (*int*) – Rotating to `[-2*rot_factor, 2*rot_factor]`.
- **scale_factor** (*float*) – Scaling to `[1-scale_factor, 1+scale_factor]`.

- **rot_prob** (*float*) – Probability of random rotation.

__init__(*rot_factor=40*, *scale_factor=0.5*, *rot_prob=0.6*)
Initialize self. See help(type(self)) for accurate signature.

class easycv.datasets.pose.pipelines.transforms.**TopDownAffine**(*use_udp=False*)
Bases: object

Affine transform the image to make input.

Required keys:'img', 'joints_3d', 'joints_3d_visible', 'ann_info','scale', 'rotation' and 'center'. Modified keys:'img', 'joints_3d', and 'joints_3d_visible'.

Parameters **use_udp** (*bool*) – To use unbiased data processing. Paper ref: Huang et al. The Devil is in the Details: Delving into Unbiased Data Processing for Human Pose Estimation (CVPR 2020).

__init__(*use_udp=False*)
Initialize self. See help(type(self)) for accurate signature.

class easycv.datasets.pose.pipelines.transforms.**TopDownGenerateTarget**(*sigma=2*, *kernel=(11, 11)*, *valid_radius_factor=0.0546875*, *target_type='GaussianHeatmap'*, *encoding='MSRA'*, *unbiased_encoding=False*)
Bases: object

Generate the target heatmap.

Required keys: 'joints_3d', 'joints_3d_visible', 'ann_info'. Modified keys: 'target', and 'target_weight'.

Parameters

- **sigma** – Sigma of heatmap gaussian for 'MSRA' approach.
- **kernel** – Kernel of heatmap gaussian for 'Megvii' approach.
- **encoding** (*str*) – Approach to generate target heatmaps. Currently supported approaches: 'MSRA', 'Megvii', 'UDP'. Default:'MSRA'
- **unbiased_encoding** (*bool*) – Option to use unbiased encoding methods. Paper ref: Zhang et al. Distribution-Aware Coordinate Representation for Human Pose Estimation (CVPR 2020).
- **keypoint_pose_distance** – Keypoint pose distance for UDP. Paper ref: Huang et al. The Devil is in the Details: Delving into Unbiased Data Processing for Human Pose Estimation (CVPR 2020).
- **target_type** (*str*) – supported targets: 'GaussianHeatmap', 'CombinedTarget'. Default:'GaussianHeatmap' CombinedTarget: The combination of classification target (response map) and regression target (offset map). Paper ref: Huang et al. The Devil is in the Details: Delving into Unbiased Data Processing for Human Pose Estimation (CVPR 2020).

__init__(*sigma=2*, *kernel=(11, 11)*, *valid_radius_factor=0.0546875*, *target_type='GaussianHeatmap'*, *encoding='MSRA'*, *unbiased_encoding=False*)
Initialize self. See help(type(self)) for accurate signature.

class easycv.datasets.pose.pipelines.transforms.**TopDownGenerateTargetRegression**
Bases: object

Generate the target regression vector (coordinates).

Required keys: 'joints_3d', 'joints_3d_visible', 'ann_info'. Modified keys: 'target', and 'target_weight'.

__init__()
Initialize self. See help(type(self)) for accurate signature.

class easycv.datasets.pose.pipelines.transforms.**TopDownRandomTranslation**(*trans_factor=0.15*, *trans_prob=1.0*)

Bases: object

Data augmentation with random translation.

Required key: 'scale' and 'center'. Modifies key: 'center'.

Notes

bbox height: H bbox width: W

Parameters

- **trans_factor** (*float*) – Translating center to [-trans_factor, trans_factor] * [W, H] + center.
- **trans_prob** (*float*) – Probability of random translation.

__init__(*trans_factor=0.15*, *trans_prob=1.0*)
Initialize self. See help(type(self)) for accurate signature.

Submodules

easycv.datasets.pose.top_down module

class easycv.datasets.pose.top_down.**PoseTopDownDataset**(*data_source*, *pipeline*, *profiling=False*)
Bases: Generic[torch.utils.data.dataset.T_co]

PoseTopDownDataset dataset for top-down pose estimation. The dataset loads raw features and apply specified transforms to return a dict containing the image tensors and other information.

Parameters

- **data_source** – Data_source config dict
- **pipeline** – Pipeline config list
- **profiling** – If set True, will print pipeline time

__init__(*data_source*, *pipeline*, *profiling=False*)
Initialize self. See help(type(self)) for accurate signature.

evaluate(*outputs*, *evaluators*, ***kwargs*)

14.1.5 easycv.datasets.selfsup package

Subpackages

easycv.datasets.selfsup.data_sources package

class easycv.datasets.selfsup.data_sources.**SSLSourceImageList**(*list_file*, *root=''*, *max_try=20*)

Bases: object

datasource for classification

Parameters

- **list_file** – str / list(str), str means a input image list file path, this file contains records as *image_path label* in list_file list(str) means multi image list, each one contains some records as *image_path label*
- **root** – str / list(str), root path for image_path, each list_file will need a root, if len(root) < len(list_file), we will use root[-1] to fill root list.
- **max_try** – int, max try numbers of reading image

__init__(*list_file*, *root=''*, *max_try=20*)

Initialize self. See help(type(self)) for accurate signature.

static parse_list_file(*list_file*, *root*)

get_length()

get_sample(*idx*)

class easycv.datasets.selfsup.data_sources.**SSLSourceImageNetFeature**(*root_path*, *training=True*, *data_keyword='feat1'*, *label_keyword='label'*, *dynamic_load=True*)

Bases: object

__init__(*root_path*, *training=True*, *data_keyword='feat1'*, *label_keyword='label'*, *dynamic_load=True*)

Initialize self. See help(type(self)) for accurate signature.

get_sample(*idx*)

get_length()

Submodules

easycv.datasets.selfsup.data_sources.image_list module

class easycv.datasets.selfsup.data_sources.image_list.**SSLSourceImageList**(*list_file*, *root=''*, *max_try=20*)

Bases: object

datasource for classification

Parameters

- **list_file** – str / list(str), str means a input image list file path, this file contains records as *image_path label* in list_file list(str) means multi image list, each one contains some records as *image_path label*

- **root** – str / list(str), root path for image_path, each list_file will need a root, if len(root) < len(list_file), we will use root[-1] to fill root list.
- **max_try** – int, max try numbers of reading image

__init__(*list_file*, *root=''*, *max_try=20*)
Initialize self. See help(type(self)) for accurate signature.

static parse_list_file(*list_file*, *root*)

get_length()

get_sample(*idx*)

easycv.datasets.selfsup.data_sources.imagenet_feature module

class easycv.datasets.selfsup.data_sources.imagenet_feature.**SSLSourceImageNetFeature**(*root_path*, *training=True*, *data_keyword='feat1'*, *label_keyword='label'*, *dynamic_load=True*)
Bases: object

__init__(*root_path*, *training=True*, *data_keyword='feat1'*, *label_keyword='label'*, *dynamic_load=True*)
Initialize self. See help(type(self)) for accurate signature.

get_sample(*idx*)

get_length()

easycv.datasets.selfsup.pipelines package

class easycv.datasets.selfsup.pipelines.**RandomAppliedTrans**(*transforms*, *p=0.5*)
Bases: object

Randomly applied transformations. :param transforms: List of transformations in dictionaries. :type transforms: List[Dict]

__init__(*transforms*, *p=0.5*)
Initialize self. See help(type(self)) for accurate signature.

class easycv.datasets.selfsup.pipelines.**Lighting**
Bases: object

Lighting noise(AlexNet - style PCA - based noise)

__init__()
Initialize self. See help(type(self)) for accurate signature.

class easycv.datasets.selfsup.pipelines.**Solarization**(*threshold=128*)
Bases: object

__init__(*threshold=128*)
Initialize self. See help(type(self)) for accurate signature.

Submodules

easycv.datasets.selfsup.pipelines.transforms module

class easycv.datasets.selfsup.pipelines.transforms.**MAEFtAugment**(*input_size=None*, *color_jitter=None*, *auto_augment=None*, *interpolation=None*, *re_prob=None*, *re_mode=None*, *re_count=None*, *mean=None*, *std=None*, *is_train=True*)

Bases: object

RandAugment data augmentation method based on "RandAugment: Practical automated data augmentation with a reduced search space". This code is borrowed from <https://github.com/pengzhiliang/MAE-pytorch> :param input_size: images input size :type input_size: int :param color_jitter: Color jitter factor :type color_jitter: float :param auto_augment: Use AutoAugment policy :param iterpolation: Training interpolation :param re_prob: Random erase prob :param re_mode: Random erase mode :param re_count: Random erase count :param mean: mean used for normalization :param std: std used for normalization :param is_train: If True use all augmentation strategy

__init__(*input_size=None*, *color_jitter=None*, *auto_augment=None*, *interpolation=None*, *re_prob=None*, *re_mode=None*, *re_count=None*, *mean=None*, *std=None*, *is_train=True*)

Initialize self. See help(type(self)) for accurate signature.

class easycv.datasets.selfsup.pipelines.transforms.**RandomAppliedTrans**(*transforms*, *p=0.5*)

Bases: object

Randomly applied transformations. :param transforms: List of transformations in dictionaries. :type transforms: List[Dict]

__init__(*transforms*, *p=0.5*)

Initialize self. See help(type(self)) for accurate signature.

class easycv.datasets.selfsup.pipelines.transforms.**Lighting**

Bases: object

Lighting noise(AlexNet - style PCA - based noise)

__init__()

Initialize self. See help(type(self)) for accurate signature.

class easycv.datasets.selfsup.pipelines.transforms.**Solarization**(*threshold=128*)

Bases: object

__init__(*threshold=128*)

Initialize self. See help(type(self)) for accurate signature.

14.1.6 easycv.datasets.shared package

class easycv.datasets.shared.**ConcatDataset**(*datasets*)
Bases: torch.utils.data.dataset.Dataset[torch.utils.data.dataset.T_co]

A wrapper of concatenated dataset.

Same as torch.utils.data.dataset.ConcatDataset, but concat the group flag for image aspect ratio.

Parameters **datasets** (list[Dataset]) – A list of datasets.

__init__(*datasets*)
Initialize self. See help(type(self)) for accurate signature.

datasets: List[torch.utils.data.dataset.Dataset[torch.utils.data.dataset.T_co]]

cumulative_sizes: List[int]

class easycv.datasets.shared.**RepeatDataset**(*dataset*, *times*)
Bases: object

A wrapper of repeated dataset.

The length of repeated dataset will be *times* larger than the original dataset. This is useful when the data loading time is long but the dataset is small. Using RepeatDataset can reduce the data loading time between epochs.

Parameters

- **dataset** (Dataset) – The dataset to be repeated.
- **times** (*int*) – Repeat times.

__init__(*dataset*, *times*)
Initialize self. See help(type(self)) for accurate signature.

class easycv.datasets.shared.**OdpsReader**(*table_name*, *selected_cols=[]*, *excluded_cols=[]*, *random_start=False*, *odps_io_config=None*, *image_col=['url_image']*, *image_type=['url']*)
Bases: object

__init__(*table_name*, *selected_cols=[]*, *excluded_cols=[]*, *random_start=False*, *odps_io_config=None*, *image_col=['url_image']*, *image_type=['url']*)
Init odps reader and datasource set to load data from odps table

Parameters

- **table_name** (*str*) – odps table to load
- **selected_cols** (*list(str)*) – select column
- **excluded_cols** (*list(str)*) – exclude column
- **random_start** (*bool*) – random start for odps table
- **odps_io_config** (*dict*) – odps config contains access_id, access_key, endpoint
- **image_col** (*list(str)*) – image column names
- **image_type** (*list(str)*) – image column types support url/base64, must be same length with image type or 0

Returns : None

get_length()

reset_reader(*dataloader_workid*, *dataloader_worknum*)

get_sample(*idx*)

b64_decode()

class easycv.datasets.shared.**RawDataset**(*data_source*, *pipeline*)
Bases: Generic[torch.utils.data.dataset.T_co]

__init__(*data_source*, *pipeline*)
Initialize self. See help(type(self)) for accurate signature.

evaluate(*scores*, *keyword*, *logger=None*)

class easycv.datasets.shared.**BaseDataset**(*data_source*, *pipeline*, *profiling=False*)
Bases: Generic[torch.utils.data.dataset.T_co]

Base Dataset

__init__(*data_source*, *pipeline*, *profiling=False*)
Initialize self. See help(type(self)) for accurate signature.

abstract evaluate(*results*, *evaluators*, *logger=None*, ***kwargs*)

visualize(*results*, ***kwargs*)
Visulaize the model output results on validation data. Returns: A dictionary

If add image visualization, return dict containing images: List of visulaized images. img_metas: List of length number of test images,

dict of image meta info, containing filename, img_shape, origin_img_shape, scale_factor and so on.

class easycv.datasets.shared.**MultiViewDataset**(*data_source*, *num_views*, *pipelines*)
Bases: Generic[torch.utils.data.dataset.T_co]

The dataset outputs multiple views of an image. The number of views in the output dict depends on *num_views*. The image can be processed by one pipeline or multiple piepelines. :param num_views: The number of different views. :type num_views: list :param pipelines: A list of pipelines. :type pipelines: list[list[dict]]

__init__(*data_source*, *num_views*, *pipelines*)
Initialize self. See help(type(self)) for accurate signature.

evaluate(*results*, *evaluators*, *logger=None*)

Subpackages

easycv.datasets.shared.data_sources package

class easycv.datasets.shared.data_sources.**ImageNpy**(*image_file*, *label_file=None*, *cache_root='data_cache/'*)
Bases: object

__init__(*image_file*, *label_file=None*, *cache_root='data_cache/'*)
image_file: (local or oss) image data saved in one .npy data [cv2.img, cv2.img,...] label_file: (local or oss) label data saved in one .npy data

get_length()

get_sample(*idx*)

class easycv.datasets.shared.data_sources.**SourceConcat**(*data_source_list*)
Bases: object

Concat multi data source config.

__init__(*data_source_list*)
Initialize self. See help(type(self)) for accurate signature.

get_length()

get_sample(*idx*)

cumsum_length()

Submodules

easycv.datasets.shared.data_sources.concat module

class easycv.datasets.shared.data_sources.concat.SourceConcat(*data_source_list*)
Bases: object

Concat multi data source config.

__init__(*data_source_list*)
Initialize self. See help(type(self)) for accurate signature.

get_length()

get_sample(*idx*)

cumsum_length()

easycv.datasets.shared.data_sources.image_npy module

class easycv.datasets.shared.data_sources.image_npy.ImageNpy(*image_file*, *label_file=None*, *cache_root='data_cache/'*)
Bases: object

__init__(*image_file*, *label_file=None*, *cache_root='data_cache/'*)
image_file: (local or oss) image data saved in one .npy data [cv2.img, cv2.img,...] label_file: (local or oss) label data saved in one .npy data

get_length()

get_sample(*idx*)

easycv.datasets.shared.pipelines package

Submodules

easycv.datasets.shared.pipelines.dali_transforms module

class easycv.datasets.shared.pipelines.dali_transforms.DaliImageDecoder(*device='mixed'*, ***kwargs*)
Bases: object

refer to: https://docs.nvidia.com/deeplearning/dali/archives/dali_0250/user-guide/docs/supported_ops.html#nvidia.dali.ops.ImageDecoder

__init__(*device='mixed'*, ***kwargs*)
Initialize self. See help(type(self)) for accurate signature.

class easycv.datasets.shared.pipelines.dali_transforms.**DaliRandomResizedCrop**(*size*, *random_area*, *device='gpu'*, ***kwargs*)

Bases: object

refer to: https://docs.nvidia.com/deeplearning/dali/archives/dali_0250/user-guide/docs/supported_ops.html#nvidia.dali.ops.RandomResizedCrop

__init__(*size*, *random_area*, *device='gpu'*, ***kwargs*)
Initialize self. See help(type(self)) for accurate signature.

class easycv.datasets.shared.pipelines.dali_transforms.**DaliResize**(*resize_shorter*, *device='gpu'*, ***kwargs*)

Bases: object

refer to: https://docs.nvidia.com/deeplearning/dali/archives/dali_0250/user-guide/docs/supported_ops.html#nvidia.dali.ops.Resize

__init__(*resize_shorter*, *device='gpu'*, ***kwargs*)
Initialize self. See help(type(self)) for accurate signature.

class easycv.datasets.shared.pipelines.dali_transforms.**DaliColorTwist**(*prob*, *saturation*, *contrast*, *brightness*, *hue*, *device='gpu'*, *center=1*)

Bases: object

refer to: https://docs.nvidia.com/deeplearning/dali/archives/dali_0250/user-guide/docs/supported_ops.html#nvidia.dali.ops.ColorTwist

__init__(*prob*, *saturation*, *contrast*, *brightness*, *hue*, *device='gpu'*, *center=1*)
Initialize self. See help(type(self)) for accurate signature.

class easycv.datasets.shared.pipelines.dali_transforms.**DaliRandomGrayscale**(*prob*, *device='gpu'*)

Bases: object

refer to: https://docs.nvidia.com/deeplearning/dali/archives/dali_0250/user-guide/docs/supported_ops.html#nvidia.dali.ops.Hsv Create *RandomGrayscale* op with ops.Hsv. when saturation=0, it represents a grayscale image

__init__(*prob*, *device='gpu'*)
Initialize self. See help(type(self)) for accurate signature.

class easycv.datasets.shared.pipelines.dali_transforms.**DaliCropMirrorNormalize**(*crop*, *mean*, *std*, *prob=0.0*, *device='gpu'*, *crop_pos_x=0.5*, *crop_pos_y=0.5*, ***kwargs*)

Bases: object

refer to: https://docs.nvidia.com/deeplearning/dali/archives/dali_0250/user-guide/docs/supported_ops.html#nvidia.dali.ops.CropMirrorNormalize

__init__(*crop*, *mean*, *std*, *prob=0.0*, *device='gpu'*, *crop_pos_x=0.5*, *crop_pos_y=0.5*, ***kwargs*)
Initialize self. See help(type(self)) for accurate signature.

easycv.datasets.shared.pipelines.format module

easycv.datasets.shared.pipelines.format.**to_tensor**(*data*)
Convert objects of various python types to `torch.Tensor`.

Supported types are: `numpy.ndarray`, `torch.Tensor`, `Sequence`, `int` and `float`.

Parameters **data** (*torch.Tensor | numpy.ndarray | Sequence | int | float*) – Data to be converted.

class easycv.datasets.shared.pipelines.format.**ImageToTensor**(*keys*)
Bases: `object`

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).

Parameters **keys** (*Sequence[str]*) – Key of images to be converted to Tensor.

__init__(*keys*)
Initialize self. See help(type(self)) for accurate signature.

class easycv.datasets.shared.pipelines.format.**Collect**(*keys*, *meta_keys=('filename', 'ori_filename', 'ori_img_shape', 'img_shape', 'scale_factor', 'pad', 'flip', 'flip_direction', 'img_norm_cfg')*)
Bases: `object`

Collect data from the loader relevant to the specific task.

This is usually the last stage of the data loader pipeline. Typically keys is set to some subset of "img", "proposals", "gt_bboxes", "gt_bboxes_ignore", "gt_labels", and/or "gt_masks".

The "img_meta" item is always populated. The contents of the "img_meta" dictionary depends on "meta_keys". By default this includes:

- "img_shape": shape of the image input to the network as a tuple (h, w). Note that images may be zero padded on the bottom/right if the batch tensor is larger than this shape.
- "scale_factor": a float indicating the preprocessing scale
- "flip": a boolean indicating if image flip transform was used
- "filename": path to the image file
- "ori_img_shape": original shape of the image as a tuple (h, w, c)
- "img_norm_cfg": a dict of normalization information:
  - mean - per channel mean subtraction
  - std - per channel std divisor
  - to_rgb - bool indicating if bgr was converted to rgb

Parameters

- **keys** (*Sequence[str]*) – Keys of results to be collected in `data`.
- **meta_keys** (*Sequence[str], optional*) – Meta keys to be converted to `mmcv.DataContainer` and collected in `data[img_metas]`. Default: ``('filename', 'ori_filename', 'ori_img_shape', 'img_shape', 'scale_factor', 'flip', 'flip_direction', 'img_norm_cfg')``

__init__(*keys*, *meta_keys=('filename', 'ori_filename', 'ori_img_shape', 'img_shape', 'scale_factor', 'pad', 'flip', 'flip_direction', 'img_norm_cfg')*)
Initialize self. See help(type(self)) for accurate signature.

class easycv.datasets.shared.pipelines.format.**DefaultFormatBundle**
Bases: object

Default formatting bundle. It simplifies the pipeline of formatting common fields, including "img", "proposals", "gt_bboxes", "gt_labels", "gt_masks" and "gt_semantic_seg". These fields are formatted as follows. - img: (1)transpose, (2)to tensor, (3)to DataContainer (stack=True) - proposals: (1)to tensor, (2)to DataContainer - gt_bboxes: (1)to tensor, (2)to DataContainer - gt_bboxes_ignore: (1)to tensor, (2)to DataContainer - gt_labels: (1)to tensor, (2)to DataContainer - gt_masks: (1)to tensor, (2)to DataContainer (cpu_only=True) - gt_semantic_seg: (1)unsqueeze dim-0 (2)to tensor, (3)to DataContainer (stack=True)

easycv.datasets.shared.pipelines.third_transforms_wrapper module

easycv.datasets.shared.pipelines.third_transforms_wrapper.**is_child_of**(*obj*, *cls*)

easycv.datasets.shared.pipelines.third_transforms_wrapper.**get_args**(*obj*)

easycv.datasets.shared.pipelines.third_transforms_wrapper.**wrap_torchvision_transforms**(*transform_obj*)

class easycv.datasets.shared.pipelines.third_transforms_wrapper.**AugMix**(*severity: int = 3*, *mixture_width: int = 3*, *chain_depth: int = -1*, *alpha: float = 1.0*, *all_ops: bool = True*, *interpolation: torchvision.transforms.functional.InterpolationM = <InterpolationMode.BILINEAR: 'bilinear'>*, *fill: Optional[List[float]] = None*)
Bases: torchvision.transforms.autoaugment.AugMix

forward(*results*)
img (PIL Image or Tensor): Image to be transformed.

Returns Transformed image.

Return type PIL Image or Tensor

training: bool

class easycv.datasets.shared.pipelines.third_transforms_wrapper.**AutoAugment**(*policy: torchvision.transforms.autoaugment.AutoA = <AutoAugmentPolicy.IMAGENET: 'imagenet'>, interpolation: torchvision.transforms.functional.Interpola = <InterpolationMode.NEAREST: 'nearest'>, fill: Optional[List[float]] = None*)

Bases: torchvision.transforms.autoaugment.AutoAugment

forward(*results*)

img (PIL Image or Tensor): Image to be transformed.

Returns AutoAugmented image.

Return type PIL Image or Tensor

training: bool

class easycv.datasets.shared.pipelines.third_transforms_wrapper.**CenterCrop**(*size*)

Bases: torchvision.transforms.transforms.CenterCrop

forward(*results*)

Parameters **img** (*PIL Image or Tensor*) – Image to be cropped.

Returns Cropped image.

Return type PIL Image or Tensor

training: bool

class easycv.datasets.shared.pipelines.third_transforms_wrapper.**ColorJitter**(*brightness=0, contrast=0, saturation=0, hue=0*)

Bases: torchvision.transforms.transforms.ColorJitter

forward(*results*)

Parameters **img** (*PIL Image or Tensor*) – Input image.

Returns Color jittered image.

Return type PIL Image or Tensor

training: bool

class easycv.datasets.shared.pipelines.third_transforms_wrapper.**ConvertImageDtype**(*dtype: torch.dtype*)

Bases: torchvision.transforms.transforms.ConvertImageDtype

forward(*results*)

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.

---

training: bool

class easycv.datasets.shared.pipelines.third_transforms_wrapper.**FiveCrop**(*size*)

Bases: torchvision.transforms.transforms.FiveCrop

forward(*results*)

Parameters **img** (*PIL Image or Tensor*) – Image to be cropped.

Returns tuple of 5 images. Image can be PIL Image or Tensor

training: bool

class easycv.datasets.shared.pipelines.third_transforms_wrapper.**GaussianBlur**(*kernel_size*, *sigma=(0.1, 2.0)*)

Bases: torchvision.transforms.transforms.GaussianBlur

forward(*results*)

Parameters **img** (*PIL Image or Tensor*) – image to be blurred.

Returns Gaussian blurred image

Return type PIL Image or Tensor

training: bool

class easycv.datasets.shared.pipelines.third_transforms_wrapper.**Grayscale**(*num_output_channels=1*)

Bases: torchvision.transforms.transforms.Grayscale

forward(*results*)

Parameters **img** (*PIL Image or Tensor*) – Image to be converted to grayscale.

Returns Grayscaled image.

Return type PIL Image or Tensor

training: bool

class easycv.datasets.shared.pipelines.third_transforms_wrapper.**Lambda**(*lambd*)

Bases: torchvision.transforms.transforms.Lambda

class easycv.datasets.shared.pipelines.third_transforms_wrapper.**LinearTransformation**(*transformation_matrix*, *mean_vector*)

Bases: torchvision.transforms.transforms.LinearTransformation

forward(*results*)

Parameters **tensor** (*Tensor*) – Tensor image to be whitened.

Returns Transformed image.

Return type Tensor

training: bool

class easycv.datasets.shared.pipelines.third_transforms_wrapper.**Normalize**(*mean*, *std*, *inplace=False*)

Bases: torchvision.transforms.transforms.Normalize

forward(*results*)

Parameters **tensor** (*Tensor*) – Tensor image to be normalized.

Returns Normalized Tensor image.

Return type Tensor

training: bool

class easycv.datasets.shared.pipelines.third_transforms_wrapper.**PILToTensor**

Bases: torchvision.transforms.transforms.PILToTensor

class easycv.datasets.shared.pipelines.third_transforms_wrapper.**Pad**(*padding*, *fill=0*, *padding_mode='constant'*)

Bases: torchvision.transforms.transforms.Pad

forward(*results*)

Parameters **img** (*PIL Image or Tensor*) – Image to be padded.

Returns Padded image.

Return type PIL Image or Tensor

training: bool

class easycv.datasets.shared.pipelines.third_transforms_wrapper.**RandAugment**(*num_ops: int = 2*, *magnitude: int = 9*, *num_magnitude_bins: int = 31*, *interpolation: torchvision.transforms.functional.Interpola = <InterpolationMode.NEAREST: 'nearest'>*, *fill: Optional[List[float]] = None*)

Bases: torchvision.transforms.autoaugment.RandAugment

forward(*results*)

img (PIL Image or Tensor): Image to be transformed.

Returns Transformed image.

Return type PIL Image or Tensor

training: bool

class easycv.datasets.shared.pipelines.third_transforms_wrapper.**RandomAdjustSharpness**(*sharpness_factor*, *p=0.5*)

Bases: torchvision.transforms.transforms.RandomAdjustSharpness

forward(*results*)

Parameters **img** (*PIL Image or Tensor*) – Image to be sharpened.

Returns Randomly sharpened image.

Return type PIL Image or Tensor

training: bool

class easycv.datasets.shared.pipelines.third_transforms_wrapper.**RandomAffine**(*degrees*, *translate=None*, *scale=None*, *shear=None*, *interpolation=<InterpolationMode.NEARE 'nearest'>*, *fill=0*, *fillcolor=None*, *resample=None*, *center=None*)

Bases: torchvision.transforms.transforms.RandomAffine

forward(*results*)

img (PIL Image or Tensor): Image to be transformed.

Returns Affine transformed image.

Return type PIL Image or Tensor

training: bool

class easycv.datasets.shared.pipelines.third_transforms_wrapper.**RandomAutocontrast**(*p=0.5*)

Bases: torchvision.transforms.transforms.RandomAutocontrast

forward(*results*)

Parameters **img** (*PIL Image or Tensor*) – Image to be autocontrasted.

Returns Randomly autocontrasted image.

Return type PIL Image or Tensor

training: bool

class easycv.datasets.shared.pipelines.third_transforms_wrapper.**RandomChoice**(*transforms*, *p=None*)

Bases: torchvision.transforms.transforms.RandomChoice

class easycv.datasets.shared.pipelines.third_transforms_wrapper.**RandomCrop**(*size*, *padding=None*, *pad_if_needed=False*, *fill=0*, *padding_mode='constant'*)

Bases: torchvision.transforms.transforms.RandomCrop

forward(*results*)

Parameters **img** (*PIL Image or Tensor*) – Image to be cropped.

Returns Cropped image.

Return type PIL Image or Tensor

training: bool

class easycv.datasets.shared.pipelines.third_transforms_wrapper.RandomEqualize(*p=0.5*)

Bases: torchvision.transforms.transforms.RandomEqualize

forward(*results*)

Parameters **img** (*PIL Image or Tensor*) – Image to be equalized.

Returns Randomly equalized image.

Return type PIL Image or Tensor

training: bool

class easycv.datasets.shared.pipelines.third_transforms_wrapper.RandomErasing(*p=0.5*, *scale=(0.02, 0.33)*, *ratio=(0.3, 3.3)*, *value=0*, *inplace=False*)

Bases: torchvision.transforms.transforms.RandomErasing

forward(*results*)

Parameters **img** (*Tensor*) – Tensor image to be erased.

Returns Erased Tensor image.

Return type img (Tensor)

training: bool

class easycv.datasets.shared.pipelines.third_transforms_wrapper.RandomGrayscale(*p=0.1*)

Bases: torchvision.transforms.transforms.RandomGrayscale

forward(*results*)

Parameters **img** (*PIL Image or Tensor*) – Image to be converted to grayscale.

Returns Randomly grayscaled image.

Return type PIL Image or Tensor

training: bool

class easycv.datasets.shared.pipelines.third_transforms_wrapper.RandomHorizontalFlip(*p=0.5*)

Bases: torchvision.transforms.transforms.RandomHorizontalFlip

forward(*results*)

Parameters **img** (*PIL Image or Tensor*) – Image to be flipped.

Returns Randomly flipped image.

Return type PIL Image or Tensor

training: bool

class easycv.datasets.shared.pipelines.third_transforms_wrapper.**RandomInvert**(*p=0.5*)

Bases: torchvision.transforms.transforms.RandomInvert

forward(*results*)

Parameters **img** (*PIL Image or Tensor*) – Image to be inverted.

Returns Randomly color inverted image.

Return type PIL Image or Tensor

training: bool

class easycv.datasets.shared.pipelines.third_transforms_wrapper.**RandomOrder**(*transforms*)

Bases: torchvision.transforms.transforms.RandomOrder

class easycv.datasets.shared.pipelines.third_transforms_wrapper.**RandomPerspective**(*distortion_scale=0.5*, *p=0.5*, *interpolation=<InterpolationMode.B 'bilinear'>*, *fill=0*)

Bases: torchvision.transforms.transforms.RandomPerspective

forward(*results*)

Parameters **img** (*PIL Image or Tensor*) – Image to be Perspectively transformed.

Returns Randomly transformed image.

Return type PIL Image or Tensor

training: bool

class easycv.datasets.shared.pipelines.third_transforms_wrapper.**RandomPosterize**(*bits*, *p=0.5*)

Bases: torchvision.transforms.transforms.RandomPosterize

forward(*results*)

Parameters **img** (*PIL Image or Tensor*) – Image to be posterized.

Returns Randomly posterized image.

Return type PIL Image or Tensor

training: bool

class easycv.datasets.shared.pipelines.third_transforms_wrapper.**RandomResizedCrop**(*size*, *scale=(0.08, 1.0)*, *ratio=(0.75, 1.3333333333333333)*, *interpolation=<InterpolationMode.B 'bilinear'>*)

Bases: torchvision.transforms.transforms.RandomResizedCrop

forward(*results*)

Parameters **img** (*PIL Image or Tensor*) – Image to be cropped and resized.

Returns Randomly cropped and resized image.

Return type PIL Image or Tensor

training: bool

class easycv.datasets.shared.pipelines.third_transforms_wrapper.**RandomRotation**(*degrees*, *interpolation=<InterpolationMode.NEA 'nearest'>*, *expand=False*, *center=None*, *fill=0*, *resample=None*)

Bases: torchvision.transforms.transforms.RandomRotation

forward(*results*)

Parameters **img** (*PIL Image or Tensor*) – Image to be rotated.

Returns Rotated image.

Return type PIL Image or Tensor

training: bool

class easycv.datasets.shared.pipelines.third_transforms_wrapper.**RandomSolarize**(*threshold*, *p=0.5*)

Bases: torchvision.transforms.transforms.RandomSolarize

forward(*results*)

Parameters **img** (*PIL Image or Tensor*) – Image to be solarized.

Returns Randomly solarized image.

Return type PIL Image or Tensor

training: bool

class easycv.datasets.shared.pipelines.third_transforms_wrapper.**RandomVerticalFlip**(*p=0.5*)

Bases: torchvision.transforms.transforms.RandomVerticalFlip

forward(*results*)

Parameters **img** (*PIL Image or Tensor*) – Image to be flipped.

Returns Randomly flipped image.

Return type PIL Image or Tensor

training: bool

class easycv.datasets.shared.pipelines.third_transforms_wrapper.**Resize**(*size*, *interpolation=<InterpolationMode.BILINEAR: 'bilinear'>*, *max_size=None*, *antialias=None*)

Bases: torchvision.transforms.transforms.Resize

forward(*results*)

Parameters **img** (*PIL Image or Tensor*) – Image to be scaled.

Returns Rescaled image.

Return type PIL Image or Tensor

training: bool

class easycv.datasets.shared.pipelines.third_transforms_wrapper.**TenCrop**(*size*, *vertical_flip=False*)

Bases: torchvision.transforms.transforms.TenCrop

forward(*results*)

Parameters **img** (*PIL Image or Tensor*) – Image to be cropped.

Returns tuple of 10 images. Image can be PIL Image or Tensor

training: bool

class easycv.datasets.shared.pipelines.third_transforms_wrapper.**ToPILImage**(*mode=None*)

Bases: torchvision.transforms.transforms.ToPILImage

class easycv.datasets.shared.pipelines.third_transforms_wrapper.**ToTensor**

Bases: torchvision.transforms.transforms.ToTensor

class easycv.datasets.shared.pipelines.third_transforms_wrapper.**TrivialAugmentWide**(*num_magnitude_bins: int = 31*, *interpolation: torchvision.transforms.functional. = <InterpolationMode.NEAREST: 'nearest'>*, *fill: Optional[List[float]] = None*)

Bases: torchvision.transforms.autoaugment.TrivialAugmentWide

forward(*results*)

img (PIL Image or Tensor): Image to be transformed.

Returns Transformed image.

Return type PIL Image or Tensor

training: bool

easycv.datasets.shared.pipelines.transforms module

class easycv.datasets.shared.pipelines.transforms.**Compose**(*transforms*, *profiling=False*)

Bases: object

Compose a data pipeline with a sequence of transforms. :param transforms: Either config dicts of transforms or transform objects. :type transforms: list[dict | callable]

__init__(*transforms*, *profiling=False*)

Initialize self. See help(type(self)) for accurate signature.

Submodules

easycv.datasets.shared.base module

class easycv.datasets.shared.base.**BaseDataset**(*data_source*, *pipeline*, *profiling=False*)

Bases: Generic[torch.utils.data.dataset.T_co]

Base Dataset

__init__(*data_source*, *pipeline*, *profiling=False*)

Initialize self. See help(type(self)) for accurate signature.

abstract evaluate(*results*, *evaluators*, *logger=None*, ***kwargs*)

visualize(*results*, ***kwargs*)

Visulaize the model output results on validation data. Returns: A dictionary

If add image visualization, return dict containing images: List of visulaized images. img_metas: List of length number of test images,

dict of image meta info, containing filename, img_shape, origin_img_shape, scale_factor and so on.

easycv.datasets.shared.dali_tfrecord_imagenet module

class easycv.datasets.shared.dali_tfrecord_imagenet.**DaliLoaderWrapper**(*dali_loader*, *batch_size*, *label_offset=0*)

Bases: object

__init__(*dali_loader*, *batch_size*, *label_offset=0*)
Initialize self. See help(type(self)) for accurate signature.

evaluate(*results*, *evaluators*, *logger=None*)
evaluate classification task

Parameters

- **results** – a dict of list of tensor, including prediction and groundtruth info, where prediction tensor is NxCand the same with groundtruth labels.
- **evaluators** – a list of evaluator

Returns a dict of float, different metric values

Return type eval_result

class easycv.datasets.shared.dali_tfrecord_imagenet.**DaliImageNetTFRecordDataSet**(*data_source*, *pipeline*, *distributed*, *batch_size*, *label_offset=0*, *random_shuffle=True*, *workers_per_gpu=2*)

Bases: object

__init__(*data_source*, *pipeline*, *distributed*, *batch_size*, *label_offset=0*, *random_shuffle=True*, *workers_per_gpu=2*)
Initialize self. See help(type(self)) for accurate signature.

get_dataloader()

easycv.datasets.shared.dali_tfrecord_multi_view module

class easycv.datasets.shared.dali_tfrecord_multi_view.**DaliLoaderWrapper**(*dali_loader*, *batch_size*, *return_list*)

Bases: object

__init__(*dali_loader*, *batch_size*, *return_list*)
Initialize self. See help(type(self)) for accurate signature.

class easycv.datasets.shared.dali_tfrecord_multi_view.**DaliTFRecordMultiViewDataset**(*data_source*, *num_views*, *pipelines*, *distributed*, *batch_size*, *random_shuffle=True*, *workers_per_gpu=2*)

Bases: object

Adapt to dali, the dataset outputs multiple views of an image. The number of views in the output dict depends on *num_views*. The image can be processed by one pipeline or multiple piepelines. :param num_views: The number of different views. :type num_views: list :param pipelines: A list of pipelines. :type pipelines: list[list[dict]]

__init__(*data_source*, *num_views*, *pipelines*, *distributed*, *batch_size*, *random_shuffle=True*, *workers_per_gpu=2*)

Initialize self. See help(type(self)) for accurate signature.

get_dataloader()

easycv.datasets.shared.dataset_wrappers module

class easycv.datasets.shared.dataset_wrappers.**ConcatDataset**(*datasets*)

Bases: torch.utils.data.dataset.Dataset[torch.utils.data.dataset.T_co]

A wrapper of concatenated dataset.

Same as torch.utils.data.dataset.ConcatDataset, but concat the group flag for image aspect ratio.

Parameters **datasets** (list[Dataset]) – A list of datasets.

__init__(*datasets*)

Initialize self. See help(type(self)) for accurate signature.

datasets: List[torch.utils.data.dataset.Dataset[torch.utils.data.dataset.T_co]]

cumulative_sizes: List[int]

class easycv.datasets.shared.dataset_wrappers.**RepeatDataset**(*dataset*, *times*)

Bases: object

A wrapper of repeated dataset.

The length of repeated dataset will be *times* larger than the original dataset. This is useful when the data loading time is long but the dataset is small. Using RepeatDataset can reduce the data loading time between epochs.

Parameters

- **dataset** (Dataset) – The dataset to be repeated.
- **times** (*int*) – Repeat times.

__init__(*dataset*, *times*)

Initialize self. See help(type(self)) for accurate signature.

easycv.datasets.shared.multi_view module

class easycv.datasets.shared.multi_view.**MultiViewDataset**(*data_source*, *num_views*, *pipelines*)
Bases: Generic[torch.utils.data.dataset.T_co]

The dataset outputs multiple views of an image. The number of views in the output dict depends on *num_views*. The image can be processed by one pipeline or multiple piepelines. :param num_views: The number of different views. :type num_views: list :param pipelines: A list of pipelines. :type pipelines: list[list[dict]]

__init__(*data_source*, *num_views*, *pipelines*)
Initialize self. See help(type(self)) for accurate signature.

evaluate(*results*, *evaluators*, *logger=None*)

easycv.datasets.shared.odps_reader module

easycv.datasets.shared.odps_reader.**set_dataloader_workid**(*value*)

easycv.datasets.shared.odps_reader.**set_dataloader_worknum**(*value*)

easycv.datasets.shared.odps_reader.**get_dist_image**(*img_url*, *max_try=10*)

class easycv.datasets.shared.odps_reader.**OdpsReader**(*table_name*, *selected_cols=[]*, *excluded_cols=[]*, *random_start=False*, *odps_io_config=None*, *image_col=['url_image']*, *image_type=['url']*)
Bases: object

__init__(*table_name*, *selected_cols=[]*, *excluded_cols=[]*, *random_start=False*, *odps_io_config=None*, *image_col=['url_image']*, *image_type=['url']*)
Init odps reader and datasource set to load data from odps table

Parameters

- **table_name** (*str*) – odps table to load
- **selected_cols** (*list(str)*) – select column
- **excluded_cols** (*list(str)*) – exclude column
- **random_start** (*bool*) – random start for odps table
- **odps_io_config** (*dict*) – odps config contains access_id, access_key, endpoint
- **image_col** (*list(str)*) – image column names
- **image_type** (*list(str)*) – image column types support url/base64, must be same length with image type or 0

Returns : None

get_length()

reset_reader(*dataloader_workid*, *dataloader_worknum*)

get_sample(*idx*)

b64_decode()

easycv.datasets.shared.raw module

class easycv.datasets.shared.raw.**RawDataset**(*data_source*, *pipeline*)
Bases: Generic[torch.utils.data.dataset.T_co]

__init__(*data_source*, *pipeline*)
Initialize self. See help(type(self)) for accurate signature.

evaluate(*scores*, *keyword*, *logger=None*)

14.1.7 easycv.datasets.utils package

Submodules

easycv.datasets.utils.tfrecord_util module

easycv.datasets.utils.tfrecord_util.**get_imagenet_dali_tfrecord_feature**()

easycv.datasets.utils.tfrecord_util.**get_path_and_index**(*file_list_or_path*)

easycv.datasets.utils.tfrecord_util.**download_tfrecord**(*file_list_or_path*, *target_path*, *slice_count=1*, *slice_id=0*, *force=False*)
Download data from oss. Use the processes on the gpus to slice download, each gpu process downloads part of the data. The number of slices is the same as the number of gpu processes. Support tfrecord of ImageNet style. tfrecord_dir

|—train1 |—train1.idx |—train2 |—train2.idx |—...

Parameters

- **file_list_or_path** – A list of absolute data path or a path str type(file_list) == list means this is the list type(file_list) == str means open(file_list).readlines()
- **target_path** – A str, download path
- **slice_count** – Download worker num
- **slice_id** – Download worker ID
- **force** – If false, skip download if the file already exists in the target path. If true, recopy and replace the original file.

Returns list of str, download tfrecord path index_path: list of str, download tfrecord idx path

Return type path

easycv.datasets.utils.type_util module

easycv.datasets.utils.type_util.**is_dali_dataset_type**(*type_name*)

14.2 Submodules

14.3 easycv.datasets.builder module

easycv.datasets.builder.**build_dataset**(*cfg*, *default_args=None*)

easycv.datasets.builder.**build_dali_dataset**(*cfg*, *default_args=None*)

easycv.datasets.builder.**build_datasource**(*cfg*)

14.4 easycv.datasets.registry module

CHAPTER

FIFTEEN

EASYCV.HOOKS PACKAGE

class easycv.hooks.**BestCkptSaverHook**(*by_epoch=True*, *save_optimizer=True*, *best_metric_name=[]*, *best_metric_type=[]*, ***kwargs*)

Bases: mmcv.runner.hooks.hook.Hook

Save checkpoints periodically.

Parameters

- **by_epoch** (*bool*) – Saving checkpoints by epoch or by iteration. Default: True.
- **save_optimizer** (*bool*) – Whether to save optimizer state_dict in the checkpoint. It is usually used for resuming experiments. Default: True.
- **best_metric_name** (*List(str)*) – metric name to save best, such as "neck_top1"… Default: [], do not save anything
- **best_metric_type** (*List(str)*) – metric type to define best, should be "max", "min" if len(best_metric_type) <= len(best_metric_type), use "max" to append.

__init__(*by_epoch=True*, *save_optimizer=True*, *best_metric_name=[]*, *best_metric_type=[]*, ***kwargs*)

Initialize self. See help(type(self)) for accurate signature.

before_run(*runner*)

after_train_epoch(*runner*)

easycv.hooks.**build_hook**(*cfg*, *default_args=None*)

class easycv.hooks.**BYOLHook**(*end_momentum=1.0*, ***kwargs*)

Bases: mmcv.runner.hooks.hook.Hook

Hook in BYOL

This hook including momentum adjustment in BYOL following: m = 1 - (1- m_0) * (cos(pi * k / K) + 1) / 2, k: current step, K: total steps.

__init__(*end_momentum=1.0*, ***kwargs*)

Initialize self. See help(type(self)) for accurate signature.

before_train_iter(*runner*)

class easycv.hooks.**DINOHook**(*momentum_teacher=0.996*, *weight_decay=0.04*, *weight_decay_end=0.4*, ***kwargs*)

Bases: mmcv.runner.hooks.hook.Hook

Hook in DINO

__init__(*momentum_teacher=0.996*, *weight_decay=0.04*, *weight_decay_end=0.4*, ***kwargs*)

Initialize self. See help(type(self)) for accurate signature.

before_run(*runner*)

before_train_iter(*runner*)

after_train_iter(*runner*)

before_train_epoch(*runner*)

class easycv.hooks.**EMAHook**(*decay=0.9999*, *copy_model_attr=()*)

Bases: mmcv.runner.hooks.hook.Hook

Hook to carry out Exponential Moving Average

__init__(*decay=0.9999*, *copy_model_attr=()*)

Parameters

- **decay** – decay rate for exponetial moving average
- **copy_model_attr** – attribute to copy from origin model to ema model

before_run(*runner*)

before_train_epoch(*runner*)

after_train_iter(*runner*)

class easycv.hooks.**DistEvalHook**(*dataloader*, *interval=1*, *mode='test'*, *initial=False*, *gpu_collect=False*, *flush_buffer=True*, ***eval_kwargs*)

Bases: *easycv.hooks.eval_hook.EvalHook*

Distributed evaluation hook.

dataloader
A PyTorch dataloader.

Type DataLoader

interval
Evaluation interval (by epochs). Default: 1.

Type int

mode
model forward mode

Type str

tmpdir
Temporary directory to save the results of all processes. Default: None.

Type str | None

gpu_collect
Whether to use gpu or cpu to collect results. Default: False.

Type bool

__init__(*dataloader*, *interval=1*, *mode='test'*, *initial=False*, *gpu_collect=False*, *flush_buffer=True*, ***eval_kwargs*)
Initialize self. See help(type(self)) for accurate signature.

before_run(*runner*)

after_train_epoch(*runner*)

class easycv.hooks.**EvalHook**(*dataloader*, *initial=False*, *interval=1*, *mode='test'*, *flush_buffer=True*, ***eval_kwargs*)

Bases: mmcv.runner.hooks.hook.Hook

Evaluation hook.

dataloader
A PyTorch dataloader.

Type DataLoader

interval
Evaluation interval (by epochs). Default: 1.

Type int

mode
model forward mode

Type str

flush_buffer
flush log buffer

Type bool

__init__(*dataloader*, *initial=False*, *interval=1*, *mode='test'*, *flush_buffer=True*, ***eval_kwargs*)
Initialize self. See help(type(self)) for accurate signature.

before_run(*runner*)

after_train_epoch(*runner*)

add_visualization_info(*runner*, *results*)

evaluate(*runner*, *results*)

class easycv.hooks.**ExportHook**(*cfg*, *ckpt_filename_tmpl='epoch_{}.pth'*, *export_ckpt_filename_tmpl='epoch_{}_export.pt'*, *export_after_each_ckpt=False*)

Bases: mmcv.runner.hooks.hook.Hook

export model when training on pai

__init__(*cfg*, *ckpt_filename_tmpl='epoch_{}.pth'*, *export_ckpt_filename_tmpl='epoch_{}_export.pt'*, *export_after_each_ckpt=False*)

Parameters

- **cfg** – config dict
- **ckpt_filename_tmpl** – checkpoint filename template

export_model(*runner*, *epoch*)

after_train_iter(*runner*)

after_train_epoch(*runner*)

after_run(*runner*)

class easycv.hooks.**Extractor**(*dataset*, *imgs_per_gpu*, *workers_per_gpu*, *dist_mode=False*)

Bases: object

__init__(*dataset*, *imgs_per_gpu*, *workers_per_gpu*, *dist_mode=False*)
Initialize self. See help(type(self)) for accurate signature.

class easycv.hooks.**OptimizerHook**(*update_interval=1*, *grad_clip=None*, *coalesce=True*, *bucket_size_mb=-1*, *ignore_key=[]*, *ignore_key_epoch=[]*, *multiply_key=[]*, *multiply_rate=[]*)

Bases: mmcv.runner.hooks.optimizer.OptimizerHook

__init__(*update_interval=1*, *grad_clip=None*, *coalesce=True*, *bucket_size_mb=-1*, *ignore_key=[]*, *ignore_key_epoch=[]*, *multiply_key=[]*, *multiply_rate=[]*)

ignore_key: [str,...], ignore_key[i], name of parameters, which's gradient will be set to zero before every optimizer step when epoch < ignore_key_epoch[i] ignore_key_epoch: [int,...], epoch < ignore_key_epoch[i], ignore_key[i]'s gradient will be set to zero.

multiply_key:[str,...] multiply_key[i], name of parameters, which will set different learning rate ratio by multipy_rate multiply_rate:[float,...] multiply_rate[i], different ratio

before_run(*runner*)

after_train_iter(*runner*)

class easycv.hooks.**OSSSyncHook**(*work_dir*, *oss_work_dir*, *interval=1*, *ckpt_filename_tmpl='epoch_{}.pth'*, *export_ckpt_filename_tmpl='epoch_{}_export.pt'*, *other_file_list=[]*, *iter_interval=None*)

Bases: mmcv.runner.hooks.hook.Hook

upload log files and checkpoints to oss when training on pai

__init__(*work_dir*, *oss_work_dir*, *interval=1*, *ckpt_filename_tmpl='epoch_{}.pth'*, *export_ckpt_filename_tmpl='epoch_{}_export.pt'*, *other_file_list=[]*, *iter_interval=None*)

Parameters

- **work_dir** – work_dir in cfg
- **oss_work_dir** – oss directory where to upload local files in work_dir
- **interval** – upload frequency
- **ckpt_filename_tmpl** – checkpoint filename template
- **other_file_list** – other file need to be upload to oss
- **iter_interval** – upload frequency by iter interval, default to be None, means do it with certain assignment

upload_file(*runner*)

after_train_iter(*runner*)

after_train_epoch(*runner*)

after_run(*runner*)

class easycv.hooks.**TIMEHook**(*end_momentum=1.0*, ***kwargs*)

Bases: mmcv.runner.hooks.hook.Hook

This hook to show time for runner running process

__init__(*end_momentum=1.0*, ***kwargs*)

Initialize self. See help(type(self)) for accurate signature.

before_train_iter(*runner*)

after_train_iter(*runner*)

class easycv.hooks.**SWAVHook**(*gpu_batch_size=32*, *dump_path='data/'*, ***kwargs*)
Bases: mmcv.runner.hooks.hook.Hook

Hook in SWAV

__init__(*gpu_batch_size=32*, *dump_path='data/'*, ***kwargs*)
Initialize self. See help(type(self)) for accurate signature.

before_run(*runner*)

before_train_epoch(*runner*)

after_train_epoch(*runner*)

class easycv.hooks.**SyncNormHook**(*no_aug_epochs=15*, *interval=1*, ***kwargs*)
Bases: mmcv.runner.hooks.hook.Hook

Synchronize Norm states after training epoch, currently used in YOLOX.

Parameters

- **no_aug_epochs** (*int*) – The number of latter epochs in the end of the training to switch to synchronizing norm interval. Default: 15.
- **interval** (*int*) – Synchronizing norm interval. Default: 1.

__init__(*no_aug_epochs=15*, *interval=1*, ***kwargs*)
Initialize self. See help(type(self)) for accurate signature.

before_train_epoch(*runner*)

after_train_epoch(*runner*)
Synchronizing norm.

class easycv.hooks.**SyncRandomSizeHook**(*ratio_range=(14, 26)*, *img_scale=(640, 640)*, *interval=10*, *device='cuda'*, ***kwargs*)
Bases: mmcv.runner.hooks.hook.Hook

Change and synchronize the random image size across ranks, currently used in YOLOX.

Parameters

- **ratio_range** (*tuple[int]*) – Random ratio range. It will be multiplied by 32, and then change the dataset output image size. Default: (14, 26).
- **img_scale** (*tuple[int]*) – Size of input image. Default: (640, 640).
- **interval** (*int*) – The interval of change image size. Default: 10.
- **device** (*torch.device | str*) – device for returned tensors. Default: 'cuda'.

__init__(*ratio_range=(14, 26)*, *img_scale=(640, 640)*, *interval=10*, *device='cuda'*, ***kwargs*)
Initialize self. See help(type(self)) for accurate signature.

after_train_iter(*runner*)
Change the dataset output image size.

class easycv.hooks.**TensorboardLoggerHookV2**(*log_dir=None*, *interval=10*, *ignore_last=True*, *reset_flag=False*, *by_epoch=True*)
Bases: mmcv.runner.hooks.logger.tensorboard.TensorboardLoggerHook

visualization_log(*runner*)
Images Visulization. *visualization_buffer* is a dictionary containing:

images (list): list of visulaized images. img_metas (list of dict, optional): dict containing ori_filename and so on.

ori_filename will be displayed as the tag of the image by default.

log(*runner*)

after_train_iter(*runner*)

class easycv.hooks.**WandbLoggerHookV2**(*init_kwargs=None*, *interval=10*, *ignore_last=True*, *reset_flag=False*, *commit=True*, *by_epoch=True*, *with_step=True*)

Bases: mmcv.runner.hooks.logger.wandb.WandbLoggerHook

visualization_log(*runner*)

Images Visulization. *visualization_buffer* is a dictionary containing:

images (list): list of visulaized images. img_metas (list of dict, optional): dict containing ori_filename and so on.

ori_filename will be displayed as the tag of the image by default.

log(*runner*)

after_train_iter(*runner*)

class easycv.hooks.**YOLOXLrUpdaterHook**(*num_last_epochs*, ***kwargs*)

Bases: mmcv.runner.hooks.lr_updater.CosineAnnealingLrUpdaterHook

YOLOX learning rate scheme.

There are two main differences between YOLOXLrUpdaterHook and CosineAnnealingLrUpdaterHook.

1. **When the current running epoch is greater than** *max_epoch-last_epoch*, a fixed learning rate will be used
2. The exp warmup scheme is different with LrUpdaterHook in MMCV

Parameters **num_last_epochs** (*int*) – The number of epochs with a fixed learning rate before the end of the training.

__init__(*num_last_epochs*, ***kwargs*)

Initialize self. See help(type(self)) for accurate signature.

get_warmup_lr(*cur_iters*)

get_lr(*runner*, *base_lr*)

class easycv.hooks.**YOLOXModeSwitchHook**(*no_aug_epochs=15*, *skip_type_keys=('MMMosaic', 'MMRandomAffine', 'MMMixUp')*, ***kwargs*)

Bases: mmcv.runner.hooks.hook.Hook

Switch the mode of YOLOX during training.

This hook turns off the mosaic and mixup data augmentation and switches to use L1 loss in bbox_head.

Parameters **no_aug_epochs** – The number of latter epochs in the end of the training to close the data augmentation and switch to L1 loss. Default: 15.

__init__(*no_aug_epochs=15*, *skip_type_keys=('MMMosaic', 'MMRandomAffine', 'MMMixUp')*, ***kwargs*)

Initialize self. See help(type(self)) for accurate signature.

before_train_epoch(*runner*)

Close mosaic and mixup augmentation and switches to use L1 loss.

class easycv.hooks.**AMPFP16OptimizerHook**(*update_interval=1*, *grad_clip=None*, *coalesce=True*, *bucket_size_mb=- 1*, *ignore_key=[]*, *ignore_key_epoch=[]*, *loss_scale={}*)

Bases: *easycv.hooks.optimizer_hook.OptimizerHook*

__init__(*update_interval=1*, *grad_clip=None*, *coalesce=True*, *bucket_size_mb=- 1*, *ignore_key=[]*, *ignore_key_epoch=[]*, *loss_scale={}*)

ignore_key: [str,...], ignore_key[i], name of parameters, which's gradient will be set to zero before every optimizer step when epoch < ignore_key_epoch[i] ignore_key_epoch: [int,...], epoch < ignore_key_epoch[i], ignore_key[i]'s gradient will be set to zero. loss_scale (float | dict): grade scale config. If loss_scale is a float, static loss scaling will be used with the specified scale.

It can also be a dict containing arguments of GradScalar. For Pytorch >= 1.6, we use official torch.cuda.amp.GradScaler. please refer to: https://pytorch.org/docs/stable/amp.html#torch.cuda.amp.GradScaler for the parameters.

before_run(*runner*)

after_train_iter(*runner*)

15.1 Submodules

15.2 easycv.hooks.best_ckpt_saver_hook module

class easycv.hooks.best_ckpt_saver_hook.**BestCkptSaverHook**(*by_epoch=True*, *save_optimizer=True*, *best_metric_name=[]*, *best_metric_type=[]*, ***kwargs*)

Bases: mmcv.runner.hooks.hook.Hook

Save checkpoints periodically.

Parameters

- **by_epoch** (*bool*) – Saving checkpoints by epoch or by iteration. Default: True.
- **save_optimizer** (*bool*) – Whether to save optimizer state_dict in the checkpoint. It is usually used for resuming experiments. Default: True.
- **best_metric_name** (*List(str)*) – metric name to save best, such as "neck_top1"... Default: [], do not save anything
- **best_metric_type** (*List(str)*) – metric type to define best, should be "max", "min" if len(best_metric_type) <= len(best_metric_type), use "max" to append.

__init__(*by_epoch=True*, *save_optimizer=True*, *best_metric_name=[]*, *best_metric_type=[]*, ***kwargs*)

Initialize self. See help(type(self)) for accurate signature.

before_run(*runner*)

after_train_epoch(*runner*)

15.3 easycv.hooks.builder module

easycv.hooks.builder.**build_hook**(*cfg*, *default_args=None*)

15.4 easycv.hooks.byol_hook module

class easycv.hooks.byol_hook.**BYOLHook**(*end_momentum=1.0*, ***kwargs*)
Bases: mmcv.runner.hooks.hook.Hook

Hook in BYOL

This hook including momentum adjustment in BYOL following: m = 1 - (1- m_0) * (cos(pi * k / K) + 1) / 2, k: current step, K: total steps.

__init__(*end_momentum=1.0*, ***kwargs*)
Initialize self. See help(type(self)) for accurate signature.

before_train_iter(*runner*)

15.5 easycv.hooks.dino_hook module

easycv.hooks.dino_hook.**cosine_scheduler**(*base_value*, *final_value*, *epochs*, *niter_per_ep*, *warmup_epochs=0*, *start_warmup_value=0*)

class easycv.hooks.dino_hook.**DINOHook**(*momentum_teacher=0.996*, *weight_decay=0.04*, *weight_decay_end=0.4*, ***kwargs*)
Bases: mmcv.runner.hooks.hook.Hook

Hook in DINO

__init__(*momentum_teacher=0.996*, *weight_decay=0.04*, *weight_decay_end=0.4*, ***kwargs*)
Initialize self. See help(type(self)) for accurate signature.

before_run(*runner*)

before_train_iter(*runner*)

after_train_iter(*runner*)

before_train_epoch(*runner*)

15.6 easycv.hooks.ema_hook module

class easycv.hooks.ema_hook.**ModelEMA**(*model*, *decay=0.9999*, *updates=0*)
Bases: object

Model Exponential Moving Average from https://github.com/rwightman/pytorch-image-models Keep a moving average of everything in the model state_dict (parameters and buffers). This is intended to allow functionality like https://www.tensorflow.org/api_docs/python/tf/train/ExponentialMovingAverage A smoothed version of the weights is necessary for some training schemes to perform well. This class is sensitive where it is initialized in the sequence of model init, GPU assignment and distributed training wrappers.

In Yolo5s, ema help increase mAP from 0.27 to 0.353

__init__(*model*, *decay=0.9999*, *updates=0*)
Initialize self. See help(type(self)) for accurate signature.

update(*model*)

update_attr(*model*, *include=()*, *exclude=('process_group', 'reducer')*)

class easycv.hooks.ema_hook.**EMAHook**(*decay=0.9999*, *copy_model_attr=()*)

Bases: mmcv.runner.hooks.hook.Hook

Hook to carry out Exponential Moving Average

__init__(*decay=0.9999*, *copy_model_attr=()*)

Parameters

- **decay** – decay rate for exponetial moving average
- **copy_model_attr** – attribute to copy from origin model to ema model

before_run(*runner*)

before_train_epoch(*runner*)

after_train_iter(*runner*)

15.7 easycv.hooks.eval_hook module

class easycv.hooks.eval_hook.**EvalHook**(*dataloader*, *initial=False*, *interval=1*, *mode='test'*, *flush_buffer=True*, ***eval_kwargs*)

Bases: mmcv.runner.hooks.hook.Hook

Evaluation hook.

dataloader
A PyTorch dataloader.

Type DataLoader

interval
Evaluation interval (by epochs). Default: 1.

Type int

mode
model forward mode

Type str

flush_buffer
flush log buffer

Type bool

__init__(*dataloader*, *initial=False*, *interval=1*, *mode='test'*, *flush_buffer=True*, ***eval_kwargs*)
Initialize self. See help(type(self)) for accurate signature.

before_run(*runner*)

after_train_epoch(*runner*)

add_visualization_info(*runner*, *results*)

evaluate(*runner*, *results*)

class easycv.hooks.eval_hook.**DistEvalHook**(*dataloader*, *interval=1*, *mode='test'*, *initial=False*, *gpu_collect=False*, *flush_buffer=True*, ***eval_kwargs*)

Bases: *easycv.hooks.eval_hook.EvalHook*

Distributed evaluation hook.

dataloader
A PyTorch dataloader.

Type DataLoader

interval
Evaluation interval (by epochs). Default: 1.

Type int

mode
model forward mode

Type str

tmpdir
Temporary directory to save the results of all processes. Default: None.

Type str | None

gpu_collect
Whether to use gpu or cpu to collect results. Default: False.

Type bool

__init__(*dataloader*, *interval=1*, *mode='test'*, *initial=False*, *gpu_collect=False*, *flush_buffer=True*, ***eval_kwargs*)
Initialize self. See help(type(self)) for accurate signature.

before_run(*runner*)

after_train_epoch(*runner*)

15.8 easycv.hooks.export_hook module

class easycv.hooks.export_hook.**ExportHook**(*cfg*, *ckpt_filename_tmpl='epoch_{}.pth'*, *export_ckpt_filename_tmpl='epoch_{}_export.pt'*, *export_after_each_ckpt=False*)

Bases: mmcv.runner.hooks.hook.Hook

export model when training on pai

__init__(*cfg*, *ckpt_filename_tmpl='epoch_{}.pth'*, *export_ckpt_filename_tmpl='epoch_{}_export.pt'*, *export_after_each_ckpt=False*)

Parameters

- **cfg** – config dict
- **ckpt_filename_tmpl** – checkpoint filename template

export_model(*runner*, *epoch*)

after_train_iter(*runner*)

after_train_epoch(*runner*)

after_run(*runner*)

15.9 easycv.hooks.extractor module

class easycv.hooks.extractor.**Extractor**(*dataset*, *imgs_per_gpu*, *workers_per_gpu*, *dist_mode=False*)

Bases: object

__init__(*dataset*, *imgs_per_gpu*, *workers_per_gpu*, *dist_mode=False*)

Initialize self. See help(type(self)) for accurate signature.

15.10 easycv.hooks.optimizer_hook module

class easycv.hooks.optimizer_hook.**OptimizerHook**(*update_interval=1*, *grad_clip=None*, *coalesce=True*, *bucket_size_mb=- 1*, *ignore_key=[]*, *ignore_key_epoch=[]*, *multiply_key=[]*, *multiply_rate=[]*)

Bases: mmcv.runner.hooks.optimizer.OptimizerHook

__init__(*update_interval=1*, *grad_clip=None*, *coalesce=True*, *bucket_size_mb=- 1*, *ignore_key=[]*, *ignore_key_epoch=[]*, *multiply_key=[]*, *multiply_rate=[]*)

ignore_key: [str,...], ignore_key[i], name of parameters, which's gradient will be set to zero before every optimizer step when epoch < ignore_key_epoch[i] ignore_key_epoch: [int,...], epoch < ignore_key_epoch[i], ignore_key[i]'s gradient will be set to zero.

multiply_key:[str,...] multiply_key[i], name of parameters, which will set different learning rate ratio by multipy_rate multiply_rate:[float,...] multiply_rate[i], different ratio

before_run(*runner*)

after_train_iter(*runner*)

class easycv.hooks.optimizer_hook.**AMPFP16OptimizerHook**(*update_interval=1*, *grad_clip=None*, *coalesce=True*, *bucket_size_mb=- 1*, *ignore_key=[]*, *ignore_key_epoch=[]*, *loss_scale={}*)

Bases: *easycv.hooks.optimizer_hook.OptimizerHook*

__init__(*update_interval=1*, *grad_clip=None*, *coalesce=True*, *bucket_size_mb=- 1*, *ignore_key=[]*, *ignore_key_epoch=[]*, *loss_scale={}*)

ignore_key: [str,...], ignore_key[i], name of parameters, which's gradient will be set to zero before every optimizer step when epoch < ignore_key_epoch[i] ignore_key_epoch: [int,...], epoch < ignore_key_epoch[i], ignore_key[i]'s gradient will be set to zero. loss_scale (float | dict): grade scale config. If loss_scale is a float, static loss scaling will be used with the specified scale.

It can also be a dict containing arguments of GradScalar. For Pytorch >= 1.6, we use official torch.cuda.amp.GradScaler. please refer to: https://pytorch.org/docs/stable/amp.html#torch.cuda.amp.GradScaler for the parameters.

before_run(*runner*)

after_train_iter(*runner*)

15.11 easycv.hooks.oss_sync_hook module

class easycv.hooks.oss_sync_hook.**OSSSyncHook**(*work_dir*, *oss_work_dir*, *interval=1*, *ckpt_filename_tmpl='epoch_{}.pth'*, *export_ckpt_filename_tmpl='epoch_{}_export.pt'*, *other_file_list=[]*, *iter_interval=None*)

Bases: mmcv.runner.hooks.hook.Hook

upload log files and checkpoints to oss when training on pai

__init__(*work_dir*, *oss_work_dir*, *interval=1*, *ckpt_filename_tmpl='epoch_{}.pth'*, *export_ckpt_filename_tmpl='epoch_{}_export.pt'*, *other_file_list=[]*, *iter_interval=None*)

Parameters

- **work_dir** – work_dir in cfg
- **oss_work_dir** – oss directory where to upload local files in work_dir
- **interval** – upload frequency
- **ckpt_filename_tmpl** – checkpoint filename template
- **other_file_list** – other file need to be upload to oss
- **iter_interval** – upload frequency by iter interval, default to be None, means do it with certain assignment

upload_file(*runner*)

after_train_iter(*runner*)

after_train_epoch(*runner*)

after_run(*runner*)

15.12 easycv.hooks.registry module

15.13 easycv.hooks.show_time_hook module

class easycv.hooks.show_time_hook.**TIMEHook**(*end_momentum=1.0*, ***kwargs*)

Bases: mmcv.runner.hooks.hook.Hook

This hook to show time for runner running process

__init__(*end_momentum=1.0*, ***kwargs*)

Initialize self. See help(type(self)) for accurate signature.

before_train_iter(*runner*)

after_train_iter(*runner*)

15.14 easycv.hooks.swav_hook module

class easycv.hooks.swav_hook.**SWAVHook**(*gpu_batch_size=32*, *dump_path='data/'*, ***kwargs*)
Bases: mmcv.runner.hooks.hook.Hook

Hook in SWAV

__init__(*gpu_batch_size=32*, *dump_path='data/'*, ***kwargs*)
Initialize self. See help(type(self)) for accurate signature.

before_run(*runner*)

before_train_epoch(*runner*)

after_train_epoch(*runner*)

15.15 easycv.hooks.sync_norm_hook module

easycv.hooks.sync_norm_hook.**get_norm_states**(*module*)

class easycv.hooks.sync_norm_hook.**SyncNormHook**(*no_aug_epochs=15*, *interval=1*, ***kwargs*)
Bases: mmcv.runner.hooks.hook.Hook

Synchronize Norm states after training epoch, currently used in YOLOX.

Parameters

- **no_aug_epochs** (*int*) – The number of latter epochs in the end of the training to switch to synchronizing norm interval. Default: 15.
- **interval** (*int*) – Synchronizing norm interval. Default: 1.

__init__(*no_aug_epochs=15*, *interval=1*, ***kwargs*)
Initialize self. See help(type(self)) for accurate signature.

before_train_epoch(*runner*)

after_train_epoch(*runner*)
Synchronizing norm.

15.16 easycv.hooks.sync_random_size_hook module

class easycv.hooks.sync_random_size_hook.**SyncRandomSizeHook**(*ratio_range=(14, 26)*, *img_scale=(640, 640)*, *interval=10*, *device='cuda'*, ***kwargs*)
Bases: mmcv.runner.hooks.hook.Hook

Change and synchronize the random image size across ranks, currently used in YOLOX.

Parameters

- **ratio_range** (*tuple[int]*) – Random ratio range. It will be multiplied by 32, and then change the dataset output image size. Default: (14, 26).
- **img_scale** (*tuple[int]*) – Size of input image. Default: (640, 640).
- **interval** (*int*) – The interval of change image size. Default: 10.
- **device** (*torch.device | str*) – device for returned tensors. Default: 'cuda'.

__init__(*ratio_range=(14, 26)*, *img_scale=(640, 640)*, *interval=10*, *device='cuda'*, ***kwargs*)
Initialize self. See help(type(self)) for accurate signature.

after_train_iter(*runner*)
Change the dataset output image size.

15.17 easycv.hooks.tensorboard module

class easycv.hooks.tensorboard.**TensorboardLoggerHookV2**(*log_dir=None*, *interval=10*, *ignore_last=True*, *reset_flag=False*, *by_epoch=True*)
Bases: mmcv.runner.hooks.logger.tensorboard.TensorboardLoggerHook

visualization_log(*runner*)
Images Visulization. *visualization_buffer* is a dictionary containing:

images (list): list of visulaized images. img_metas (list of dict, optional): dict containing ori_filename and so on.

ori_filename will be displayed as the tag of the image by default.

log(*runner*)

after_train_iter(*runner*)

15.18 easycv.hooks.wandb module

class easycv.hooks.wandb.**WandbLoggerHookV2**(*init_kwargs=None*, *interval=10*, *ignore_last=True*, *reset_flag=False*, *commit=True*, *by_epoch=True*, *with_step=True*)
Bases: mmcv.runner.hooks.logger.wandb.WandbLoggerHook

visualization_log(*runner*)
Images Visulization. *visualization_buffer* is a dictionary containing:

images (list): list of visulaized images. img_metas (list of dict, optional): dict containing ori_filename and so on.

ori_filename will be displayed as the tag of the image by default.

log(*runner*)

after_train_iter(*runner*)

15.19 easycv.hooks.yolox_lr_hook module

class easycv.hooks.yolox_lr_hook.**YOLOXLrUpdaterHook**(*num_last_epochs*, ***kwargs*)
Bases: mmcv.runner.hooks.lr_updater.CosineAnnealingLrUpdaterHook

YOLOX learning rate scheme.

There are two main differences between YOLOXLrUpdaterHook and CosineAnnealingLrUpdaterHook.

1. **When the current running epoch is greater than** *max_epoch-last_epoch*, a fixed learning rate will be used

2. The exp warmup scheme is different with LrUpdaterHook in MMCV

Parameters **num_last_epochs** (*int*) – The number of epochs with a fixed learning rate before the end of the training.

__init__(*num_last_epochs*, ***kwargs*)
Initialize self. See help(type(self)) for accurate signature.

get_warmup_lr(*cur_iters*)

get_lr(*runner*, *base_lr*)

15.20 easycv.hooks.yolox_mode_switch_hook module

class easycv.hooks.yolox_mode_switch_hook.**YOLOXModeSwitchHook**(*no_aug_epochs=15*, *skip_type_keys=('MMMosaic', 'MMRandomAffine', 'MMMixUp')*, ***kwargs*)

Bases: mmcv.runner.hooks.hook.Hook

Switch the mode of YOLOX during training.

This hook turns off the mosaic and mixup data augmentation and switches to use L1 loss in bbox_head.

Parameters **no_aug_epochs** – The number of latter epochs in the end of the training to close the data augmentation and switch to L1 loss. Default: 15.

__init__(*no_aug_epochs=15*, *skip_type_keys=('MMMosaic', 'MMRandomAffine', 'MMMixUp')*, ***kwargs*)
Initialize self. See help(type(self)) for accurate signature.

before_train_epoch(*runner*)
Close mosaic and mixup augmentation and switches to use L1 loss.

CHAPTER

SIXTEEN

EASYCV.PREDICTORS PACKAGE

16.1 Submodules

16.2 easycv.predictors.base module

class easycv.predictors.base.**NumpyToPIL**
Bases: object

class easycv.predictors.base.**Predictor**(*model_path*, *numpy_to_pil=True*)
Bases: object

__init__(*model_path*, *numpy_to_pil=True*)
Initialize self. See help(type(self)) for accurate signature.

preprocess(*image_list*)

predict_batch(*image_batch*, ***forward_kwargs*)
predict using batched data

Parameters

- **image_batch** (*torch.Tensor*) – tensor with shape [N, 3, H, W]
- **forward_kwargs** – kwargs for additional parameters

Returns the output of model.forward, list or tuple

Return type output

16.3 easycv.predictors.builder module

easycv.predictors.builder.**build_predictor**(*cfg*)

16.4 easycv.predictors.classifier module

class easycv.predictors.classifier.**TorchClassifier**(*model_path*, *model_config=None*, *topk=1*, *label_map_path=None*)

Bases: *easycv.predictors.interface.PredictorInterface*

__init__(*model_path*, *model_config=None*, *topk=1*, *label_map_path=None*)

init model

Parameters

- **model_path** – model file path
- **model_config** – config string for model to init, in json format

get_output_type()

in this function user should return a type dict, which indicates which type of data should the output of predictor be converted to * type json, data will be serialized to json str

- type image, data will be converted to encode image binary and write to oss file, whose name is output_dir/${key}/${input_filename}_${idx}.jpg, where input_filename is the base filename extracted from url, key corresponds to the key in the dict of output_type, if the type of data indexed by key is a list, idx is the index of element in list, otherwhile ${idx} will be empty
- type video, data will be converted to encode video binary and write to oss file,

:: return { 'image': 'image', 'feature': 'json'

}

indicating that the image data in the output dict will be save to image file and feature in output dict will be converted to json

batch(*image_tensor_list*)

predict(*input_data_list*, *batch_size=- 1*)

using session run predict a number of samples using batch_size

Parameters

- **input_data_list** – a list of numpy array, each array is a sample to be predicted
- **batch_size** – batch_size passed by the caller, you can also ignore this param and use a fixed number if you do not want to adjust batch_size in runtime

Returns

a list of dict, each dict is the prediction result of one sample eg, {"output1": value1, "output2": value2}, the value type can be python int str float, and numpy array

Return type result

16.5 easycv.predictors.detector module

class easycv.predictors.detector.**TorchYoloXPredictor**(*model_path*, *max_det=100*, *score_thresh=0.5*, *model_config=None*)

Bases: *easycv.predictors.interface.PredictorInterface*

__init__(*model_path*, *max_det=100*, *score_thresh=0.5*, *model_config=None*)

init model

Parameters

- **model_path** – model file path
- **max_det** – maximum number of detection
- **score_thresh** – score_thresh to filter box
- **model_config** – config string for model to init, in json format

post_assign(*outputs*, *img_metas*)

predict(*input_data_list*, *batch_size=- 1*, *to_numpy=True*)

using session run predict a number of samples using batch_size

Parameters

- **input_data_list** – a list of numpy array(in rgb order), each array is a sample to be predicted
- **batch_size** – batch_size passed by the caller, you can also ignore this param and use a fixed number if you do not want to adjust batch_size in runtime

Returns

a list of dict, each dict is the prediction result of one sample eg, {"output1": value1, "output2": value2}, the value type can be python int str float, and numpy array

Return type result

class easycv.predictors.detector.**TorchViTDetPredictor**(*model_path*)

Bases: *easycv.predictors.interface.PredictorInterface*

__init__(*model_path*)

init model

Parameters

- **model_path** – init model from this directory
- **model_config** – config string for model to init, in json format

predict(*imgs*)

Inference image(s) with the detector. :param model: The loaded detector. :type model: nn.Module :param imgs: :type imgs: str/ndarray or list[str/ndarray] or tuple[str/ndarray] :param Either image files or loaded images.:

Returns If imgs is a list or tuple, the same length list type results will be returned, otherwise return the detection results directly.

show_result_pyplot(*img*, *results*, *score_thr=0.3*, *show=False*, *out_file=None*)

class easycv.predictors.detector.**TorchFaceDetector**(*model_path=None*, *model_config=None*)

Bases: *easycv.predictors.interface.PredictorInterface*

__init__(*model_path=None*, *model_config=None*)
init model, add a facedetect and align for img input.

Parameters

- **model_path** – model file path
- **model_config** – config string for model to init, in json format

get_output_type()
in this function user should return a type dict, which indicates which type of data should the output of predictor be converted to * type json, data will be serialized to json str

- type image, data will be converted to encode image binary and write to oss file, whose name is output_dir/${key}/${input_filename}_${idx}.jpg, where input_filename is the base filename extracted from url, key corresponds to the key in the dict of output_type, if the type of data indexed by key is a list, idx is the index of element in list, otherwhile ${idx} will be empty
- type video, data will be converted to encode video binary and write to oss file,

:: return { 'image': 'image', 'feature': 'json'

} indicating that the image data in the output dict will be save to image file and feature in output dict will be converted to json

batch(*image_tensor_list*)

predict(*input_data_list*, *batch_size=- 1*, *threshold=0.95*)
using session run predict a number of samples using batch_size

Parameters

- **input_data_list** – a list of numpy array, each array is a sample to be predicted
- **batch_size** – batch_size passed by the caller, you can also ignore this param and use a fixed number if you do not want to adjust batch_size in runtime

Returns

a list of dict, each dict is the prediction result of one sample eg, {"output1": value1, "output2": value2}, the value type can be python int str float, and numpy array

Return type result

Raises **if detect !=1 face in a img, then do nothing for this image** –

class easycv.predictors.detector.**TorchYoloXClassifierPredictor**(*models_root_dir*, *max_det=100*, *cls_score_thresh=0.01*, *det_model_config=None*, *cls_model_config=None*)

Bases: *easycv.predictors.interface.PredictorInterface*

__init__(*models_root_dir*, *max_det=100*, *cls_score_thresh=0.01*, *det_model_config=None*, *cls_model_config=None*)
init model, add a yolox and classification predictor for img input.

Parameters

- **models_root_dir** – models_root_dir/detection/*.pth and models_root_dir/classification/*.pth
- **det_model_config** – config string for detection model to init, in json format
- **cls_model_config** – config string for classification model to init, in json format

predict(*input_data_list*, *batch_size=- 1*)
using session run predict a number of samples using batch_size

Parameters

- **input_data_list** – a list of numpy array(in rgb order), each array is a sample to be predicted
- **batch_size** – batch_size passed by the caller, you can also ignore this param and use a fixed number if you do not want to adjust batch_size in runtime

Returns

a list of dict, each dict is the prediction result of one sample eg, {"output1": value1, "output2": value2}, the value type can be python int str float, and numpy array

Return type result

16.6 easycv.predictors.feature_extractor module

class easycv.predictors.feature_extractor.**TorchFeatureExtractor**(*model_path*, *model_config=None*)

Bases: *easycv.predictors.interface.PredictorInterface*

__init__(*model_path*, *model_config=None*)
init model

Parameters

- **model_path** – model file path
- **model_config** – config string for model to init, in json format

get_output_type()
in this function user should return a type dict, which indicates which type of data should the output of predictor be converted to * type json, data will be serialized to json str

- type image, data will be converted to encode image binary and write to oss file, whose name is output_dir/${key}/${input_filename}_${idx}.jpg, where input_filename is the base filename extracted from url, key corresponds to the key in the dict of output_type, if the type of data indexed by key is a list, idx is the index of element in list, otherwhile ${idx} will be empty
- type video, data will be converted to encode video binary and write to oss file,

:: return { 'image': 'image', 'feature': 'json'

} indicating that the image data in the output dict will be save to image file and feature in output dict will be converted to json

batch(*image_tensor_list*)

predict(*input_data_list*, *batch_size=- 1*)
using session run predict a number of samples using batch_size

Parameters

- **input_data_list** – a list of numpy array, each array is a sample to be predicted
- **batch_size** – batch_size passed by the caller, you can also ignore this param and use a fixed number if you do not want to adjust batch_size in runtime

Returns

a list of dict, each dict is the prediction result of one sample eg, {"output1": value1, "output2": value2}, the value type can be python int str float, and numpy array

Return type result

class easycv.predictors.feature_extractor.**TorchFaceFeatureExtractor**(*model_path*, *model_config=None*)

Bases: *easycv.predictors.interface.PredictorInterface*

__init__(*model_path*, *model_config=None*)
init model, add a facedetect and align for img input.

Parameters

- **model_path** – model file path
- **model_config** – config string for model to init, in json format

get_output_type()
in this function user should return a type dict, which indicates which type of data should the output of predictor be converted to * type json, data will be serialized to json str

- type image, data will be converted to encode image binary and write to oss file, whose name is output_dir/${key}/${input_filename}_${idx}.jpg, where input_filename is the base filename extracted from url, key corresponds to the key in the dict of output_type, if the type of data indexed by key is a list, idx is the index of element in list, otherwhile ${idx} will be empty
- type video, data will be converted to encode video binary and write to oss file,

:: return { 'image': 'image', 'feature': 'json'

} indicating that the image data in the output dict will be save to image file and feature in output dict will be converted to json

batch(*image_tensor_list*)

predict(*input_data_list*, *batch_size=- 1*, *detect_and_align=True*)
using session run predict a number of samples using batch_size

Parameters

- **input_data_list** – a list of numpy array or PIL.Image, each array is a sample to be predicted
- **batch_size** – batch_size passed by the caller, you can also ignore this param and use a fixed number if you do not want to adjust batch_size in runtime
- **detect_and_align** – True to detect and align before feature extractor

Returns

a list of dict, each dict is the prediction result of one sample eg, {"output1": value1, "output2": value2}, the value type can be python int str float, and numpy array

Return type result

Raises **if detect !=1 face in a img, then do nothing for this image** –

class easycv.predictors.feature_extractor.**TorchMultiFaceFeatureExtractor**(*model_path*, *model_config=None*)

Bases: *easycv.predictors.interface.PredictorInterface*

__init__(*model_path*, *model_config=None*)
init model, add a facedetect and align for img input.

Parameters

- **model_path** – model file path
- **model_config** – config string for model to init, in json format

get_output_type()
in this function user should return a type dict, which indicates which type of data should the output of predictor be converted to * type json, data will be serialized to json str

- type image, data will be converted to encode image binary and write to oss file, whose name is output_dir/${key}/${input_filename}_${idx}.jpg, where input_filename is the base filename extracted from url, key corresponds to the key in the dict of output_type, if the type of data indexed by key is a list, idx is the index of element in list, otherwhile ${idx} will be empty
- type video, data will be converted to encode video binary and write to oss file,

:: return { 'image': 'image', 'feature': 'json'

} indicating that the image data in the output dict will be save to image file and feature in output dict will be converted to json

batch(*image_tensor_list*)

predict(*input_data_list*, *batch_size=- 1*, *detect_and_align=True*)
using session run predict a number of samples using batch_size

Parameters

- **input_data_list** – a list of numpy array or PIL.Image, each array is a sample to be predicted
- **batch_size** – batch_size passed by the caller, you can also ignore this param and use a fixed number if you do not want to adjust batch_size in runtime
- **detect_and_align** – True to detect and align before feature extractor

Returns

a list of dict, each dict is the prediction result of one sample eg, {"output1": value1, "output2": value2}, the value type can be python int str float, and numpy array

Return type result

Raises **if detect !=1 face in a img, then do nothing for this image** –

class easycv.predictors.feature_extractor.**TorchFaceAttrExtractor**(*model_path*, *model_config=None*, *face_threshold=0.95*, *attr_method=['distribute_sum', 'softmax', 'softmax']*, *attr_name=['age', 'gender', 'emo']*)

Bases: *easycv.predictors.interface.PredictorInterface*

__init__(*model_path*, *model_config=None*, *face_threshold=0.95*, *attr_method=['distribute_sum', 'softmax', 'softmax']*, *attr_name=['age', 'gender', 'emo']*)
init model

Parameters

- **model_path** – model file path
- **model_config** – config string for model to init, in json format

- **attr_method** –
  - softmax: do softmax for feature_dim 1
  - distribute_sum: do softmax and prob sum

get_output_type()
in this function user should return a type dict, which indicates which type of data should the output of predictor be converted to * type json, data will be serialized to json str

- type image, data will be converted to encode image binary and write to oss file, whose name is output_dir/${key}/${input_filename}_${idx}.jpg, where input_filename is the base filename extracted from url, key corresponds to the key in the dict of output_type, if the type of data indexed by key is a list, idx is the index of element in list, otherwhile ${idx} will be empty
- type video, data will be converted to encode video binary and write to oss file,

:: return { 'image': 'image', 'feature': 'json'

} indicating that the image data in the output dict will be save to image file and feature in output dict will be converted to json

batch(*image_tensor_list*)

predict(*input_data_list*, *batch_size=- 1*)
using session run predict a number of samples using batch_size

Parameters

- **input_data_list** – a list of numpy array, each array is a sample to be predicted
- **batch_size** – batch_size passed by the caller, you can also ignore this param and use a fixed number if you do not want to adjust batch_size in runtime

Returns

a list of dict, each dict is the prediction result of one sample eg, {"output1": value1, "output2": value2}, the value type can be python int str float, and numpy array

Return type result

16.7 easycv.predictors.interface module

class easycv.predictors.interface.**PredictorInterface**(*model_path*, *model_config=None*)
Bases: object

version = 1

__init__(*model_path*, *model_config=None*)
init model

Parameters

- **model_path** – init model from this directory
- **model_config** – config string for model to init, in json format

abstract predict(*input_data*, *batch_size*)
using session run predict a number of samples using batch_size

Parameters

- **input_data** – a list of numpy array, each array is a sample to be predicted

- **batch_size** – batch_size passed by the caller, you can also ignore this param and use a fixed number if you do not want to adjust batch_size in runtime

Returns

a list of dict, each dict is the prediction result of one sample eg, {"output1": value1, "output2": value2}, the value type can be python int str float, and numpy array

Return type result

get_output_type()
in this function user should return a type dict, which indicates which type of data should the output of predictor be converted to * type json, data will be serialized to json str

- type image, data will be converted to encode image binary and write to oss file, whose name is output_dir/${key}/${input_filename}_${idx}.jpg, where input_filename is extracted from url, key corresponds to the key in the dict of output_type, if the type of data indexed by key is a list, idx is the index of element in list, otherwhile ${idx} will be empty
- type video, data will be converted to encode video binary and write to oss file,

:: return { 'image': 'image', 'feature': 'json'

} indicating that the image data in the output dict will be save to image file and feature in output dict will be converted to json

class easycv.predictors.interface.**PredictorInterfaceV2**(*model_path*, *model_config=None*)
Bases: *easycv.predictors.interface.PredictorInterface*

version = 2

__init__(*model_path*, *model_config=None*)
init model

Parameters

- **model_path** – init model from this directory
- **model_config** – config string for model to init, in json format

get_output_type()
in this function user should return a type dict, which indicates which type of data should the output of predictor be converted to * type json, data will be serialized to json str

- type image, data will be converted to encode image binary and write to oss file, whose name is output_dir/${key}/${input_filename}_${idx}.jpg, where input_filename is the base filename extracted from url, key corresponds to the key in the dict of output_type, if the type of data indexed by key is a list, idx is the index of element in list, otherwhile ${idx} will be empty
- type video, data will be converted to encode video binary and write to oss file,

:: return { 'image': 'image', 'feature': 'json'

} indicating that the image data in the output dict will be save to image file and feature in output dict will be converted to json

abstract predict(*input_data_dict_list*, *batch_size*)
using session run predict a number of samples using batch_size

Parameters

- **input_data_dict_list** – a list of dict, each dict is a sample data to be predicted

- **batch_size** – batch_size passed by the caller, you can also ignore this param and use a fixed number if you do not want to adjust batch_size in runtime

Returns

a list of dict, each dict is the prediction result of one sample eg, {"output1": value1, "output2": value2}, the value type can be python int str float, and numpy array

Return type result

16.8 easycv.predictors.pose_predictor module

class easycv.predictors.pose_predictor.**LoadImage**(*color_type='color'*, *channel_order='rgb'*)

Bases: object

A simple pipeline to load image.

__init__(*color_type='color'*, *channel_order='rgb'*)

Initialize self. See help(type(self)) for accurate signature.

easycv.predictors.pose_predictor.**rgetattr**(*obj*, *attr*, **args*)

class easycv.predictors.pose_predictor.**OutputHook**(*module*, *outputs=None*, *as_tensor=False*)

Bases: object

__init__(*module*, *outputs=None*, *as_tensor=False*)

Initialize self. See help(type(self)) for accurate signature.

register(*module*)

remove()

class easycv.predictors.pose_predictor.**TorchPoseTopDownPredictor**(*model_path*, *model_config=None*)

Bases: *easycv.predictors.interface.PredictorInterface*

Inference a single image with a list of bounding boxes.

__init__(*model_path*, *model_config=None*)

init model

Parameters

- **model_path** – model file path
- **model_config** – config string for model to init, in json format

predict(*input_data_list*, *batch_size=- 1*, *return_heatmap=False*)

Inference pose.

Parameters

- **input_data_list** – A list of image infos, like: [

  {

  'img' (str | np.ndarray, RGB): Image filename or loaded image.

  'detection_results'(list | np.ndarray): All bounding boxes (with scores), shaped (N, 4) or (N, 5). (left, top, width, height, [score]) where N is number of bounding boxes.

  ]

- **batch_size** – batch size
- **return_heatmap** – return heatmap value or not, default false.

Returns

{ 'pose_results': list of ndarray[NxKx3]: Predicted pose x, y, score 'pose_heatmap' (optional): list of heatmap[N, K, H, W]: Model output heatmap

}

class easycv.predictors.pose_predictor.**TorchPoseTopDownPredictorWithDetector**(*model_path*, *model_config={'detection': {'model_type': None, 'reserved_classes': [], 'score_thresh': 0.0}, 'pose': {'bbox_thr': 0.3, 'format': 'xywh'}}*)

Bases: *easycv.predictors.interface.PredictorInterface*

SUPPORT_DETECTION_PREDICTORS = {'TorchYoloXPredictor': <class 'easycv.predictors.detector.TorchYoloXPredictor'>}

__init__(*model_path*, *model_config={'detection': {'model_type': None, 'reserved_classes': [], 'score_thresh': 0.0}, 'pose': {'bbox_thr': 0.3, 'format': 'xywh'}}*)

init model

Parameters

- **model_path** – pose and detection model file path, split with ,, make sure the first is pose model, second is detection model
- **model_config** – config string for model to init, in json format

process_det_results(*outputs*, *input_data_list*, *reserved_classes=[]*)

predict(*input_data_list*, *batch_size=- 1*, *return_heatmap=False*)

Inference with pose model and detection model.

Parameters

- **input_data_list** – A list of images(np.ndarray, RGB)
- **batch_size** – batch size
- **return_heatmap** – return heatmap value or not, default false.

Returns

{ 'pose_results': list of ndarray[NxKx3]: Predicted pose x, y, score 'pose_heatmap' (optional): list of heatmap[N, K, H, W]: Model output heatmap

}

easycv.predictors.pose_predictor.**vis_pose_result**(*model*, *img*, *result*, *radius=4*, *thickness=1*, *kpt_score_thr=0.3*, *bbox_color='green'*, *dataset_info=None*, *show=False*, *out_file=None*)

Visualize the detection results on the image.

Parameters

- **model** (*nn.Module*) – The loaded detector.
- **img** (*str | np.ndarray*) – Image filename or loaded image.
- **result** (*list[dict]*) – The results to draw over *img* (bbox_result, pose_result).
- **radius** (*int*) – Radius of circles.
- **thickness** (*int*) – Thickness of lines.
- **kpt_score_thr** (*float*) – The threshold to visualize the keypoints.
- **skeleton** (*list[tuple()]*) – Default None.
- **show** (*bool*) – Whether to show the image. Default True.
- **out_file** (*str|None*) – The filename of the output visualization image.

CHAPTER

SEVENTEEN

EASYCV.CORE PACKAGE

17.1 Subpackages

17.1.1 easycv.core.evaluation package

Subpackages

easycv.core.evaluation.custom_cocotools package

Submodules

easycv.core.evaluation.custom_cocotools.cocoeval module

class easycv.core.evaluation.custom_cocotools.cocoeval.**COCOeval**(*cocoGt=None*, *cocoDt=None*, *iouType='segm'*, *sigmas=None*)

Bases: object

__init__(*cocoGt=None*, *cocoDt=None*, *iouType='segm'*, *sigmas=None*)

Initialize CocoEval using coco APIs for gt and dt :param cocoGt: coco object with ground truth annotations :param cocoDt: coco object with detection results :param iouType: type of iou to be computed, bbox for detection task,

segm for segmentation task

Parameters **sigmas** – keypoint labelling sigmas.

Returns None

evaluate()

Run per image evaluation on given images and store results (a list of dict) in self.evalImgs :returns: None

computeIoU(*imgId*, *catId*)

computeOks(*imgId*, *catId*)

evaluateImg(*imgId*, *catId*, *aRng*, *maxDet*)

perform evaluation for single category and image :param imgId: image id, string :param catId: category id, string :param aRng: area range, tuple :param maxDet: maximum detection number

Returns dict (single image results)

accumulate(*p=None*)

Accumulate per image evaluation results and store the result in self.eval :param param p: input params for evaluation

Returns None

summarize()
Compute and display summary metrics for evaluation results. Note this functin can *only* be applied on the default parameter setting

summarize_per_category()
Compute and display summary metrics for evaluation results *per category*. Note this functin can *only* be applied on the default parameter setting

filter_annotations(*annotations*, *catIds*)

makeplot(*recThrs*, *precisions*, *name*, *save_dir=None*)

analyze()
Analyze errors

class easycv.core.evaluation.custom_cocotools.cocoeval.**Params**(*iouType='segm'*)
Bases: object

Params for coco evaluation api

setDetParams()

setKpParams()

__init__(*iouType='segm'*)
Initialize self. See help(type(self)) for accurate signature.

Submodules

easycv.core.evaluation.ap module

easycv.core.evaluation.auc_eval module

class easycv.core.evaluation.auc_eval.**AucEvaluator**(*dataset_name=None*, *metric_names=['neck_auc']*, *neck_num=None*)
Bases: *easycv.core.evaluation.base_evaluator.Evaluator*

AUC evaluator for binary classification only.

__init__(*dataset_name=None*, *metric_names=['neck_auc']*, *neck_num=None*)

Parameters

- **dataset_name** – eval dataset name
- **metric_names** – eval metrics name
- **neck_num** – some model contains multi-neck to support multitask, neck_num means use the no.neck_num neck output of model to eval

easycv.core.evaluation.base_evaluator module

class easycv.core.evaluation.base_evaluator.**Evaluator**(*dataset_name=None*, *metric_names=[]*)
Bases: object

Evaluator interface

__init__(*dataset_name=None*, *metric_names=[]*)
Construct eval ops from tensor

Parameters

- **dataset_name** (*str*) – dataset name to be evaluated
- **metric_names** (*List[str]*) – metric names this evaluator will return

evaluate(*prediction_dict*, *groundtruth_dict*, ***kwargs*)

property metric_names

easycv.core.evaluation.builder module

easycv.core.evaluation.builder.**build_evaluator**(*evaluator_cfg_list*)
build evaluator according to metric name

Parameters **evaluator_cfg_list** – list of evaluator config dict

Returns return a list of evaluator

easycv.core.evaluation.classification_eval module

class easycv.core.evaluation.classification_eval.**ClsEvaluator**(*topk=(1, 5)*, *dataset_name=None*, *metric_names=['neck_top1']*, *neck_num=None*)
Bases: *easycv.core.evaluation.base_evaluator.Evaluator*

Classification evaluator.

__init__(*topk=(1, 5)*, *dataset_name=None*, *metric_names=['neck_top1']*, *neck_num=None*)

Parameters

- **top_k** – tuple of int, evaluate top_k acc
- **dataset_name** – eval dataset name
- **metric_names** – eval metrics name
- **neck_num** – some model contains multi-neck to support multitask, neck_num means use the no.neck_num neck output of model to eval

easycv.core.evaluation.coco_evaluation module

Class for evaluating object detections with COCO metrics.

class easycv.core.evaluation.coco_evaluation.**CocoDetectionEvaluator**(*classes*, *include_metrics_per_category=False*, *all_metrics_per_category=False*, *coco_analyze=False*, *dataset_name=None*, *metric_names=['DetectionBoxes_Precision/mAP'*

Bases: *easycv.core.evaluation.base_evaluator.Evaluator*

Class to evaluate COCO detection metrics.

__init__(*classes*, *include_metrics_per_category=False*, *all_metrics_per_category=False*, *coco_analyze=False*, *dataset_name=None*, *metric_names=['DetectionBoxes_Precision/mAP']*)

Constructor.

Parameters

- **classes** – a list of class name
- **include_metrics_per_category** – If True, include metrics for each category.
- **all_metrics_per_category** – Whether to include all the summary metrics for each category in per_category_ap. Be careful with setting it to true if you have more than handful of , because it will pollute your mldash.
- **coco_analyze** – If True, will analyze the detection result using coco analysis.
- **dataset_name** – If not None, dataset_name will be inserted to each metric name.

clear()

Clears the state to prepare for a fresh evaluation.

add_single_ground_truth_image_info(*image_id*, *groundtruth_dict*)

Adds groundtruth for a single image to be used for evaluation.

If the image has already been added, a warning is logged, and groundtruth is ignored.

Parameters

- **image_id** – A unique string/integer identifier for the image.
- **groundtruth_dict** – A dictionary containing

  InputDataFields.groundtruth_boxes float32 numpy array of shape [num_boxes, 4] containing *num_boxes* groundtruth boxes of the format [ymin, xmin, ymax, xmax] in absolute image coordinates.

  InputDataFields.groundtruth_classes integer numpy array of shape [num_boxes] containing 1-indexed groundtruth classes for the boxes. InputDataFields.groundtruth_is_crowd (optional): integer numpy array of shape [num_boxes] containing iscrowd flag for groundtruth boxes.

add_single_detected_image_info(*image_id*, *detections_dict*)

Adds detections for a single image to be used for evaluation.

If a detection has already been added for this image id, a warning is logged, and the detection is skipped.

Parameters

- **image_id** – A unique string/integer identifier for the image.
- **detections_dict** – A dictionary containing

DetectionResultFields.detection_boxes float32 numpy array of shape [num_boxes, 4] containing *num_boxes* detection boxes of the format [ymin, xmin, ymax, xmax] in absolute image coordinates.

DetectionResultFields.detection_scores float32 numpy array of shape [num_boxes] containing detection scores for the boxes.

DetectionResultFields.detection_classes integer numpy array of shape [num_boxes] containing 1-indexed detection classes for the boxes.

Raises `ValueError` – If groundtruth for the image_id is not available.

class easycv.core.evaluation.coco_evaluation.**CocoMaskEvaluator**(*classes*, *include_metrics_per_category=False*, *dataset_name=None*, *metric_names=['DetectionMasks_Precision/mAP']*)

Bases: *easycv.core.evaluation.base_evaluator.Evaluator*

Class to evaluate COCO detection metrics.

`__init__`(*classes*, *include_metrics_per_category=False*, *dataset_name=None*, *metric_names=['DetectionMasks_Precision/mAP']*)

Constructor.

Parameters

- `categories` – A list of dicts, each of which has the following keys :id: (required) an integer id uniquely identifying this category. :name: (required) string representing category name e.g., 'cat', 'dog'.
- `include_metrics_per_category` – If True, include metrics for each category.

`clear`()

Clears the state to prepare for a fresh evaluation.

`add_single_ground_truth_image_info`(*image_id*, *groundtruth_dict*)

Adds groundtruth for a single image to be used for evaluation.

If the image has already been added, a warning is logged, and groundtruth is ignored.

Parameters

- `image_id` – A unique string/integer identifier for the image.
- `groundtruth_dict` – A dictionary containing :InputDataFields.groundtruth_boxes: float32 numpy array of shape

  [num_boxes, 4] containing *num_boxes* groundtruth boxes of the format [ymin, xmin, ymax, xmax] in absolute image coordinates.

  InputDataFields.groundtruth_classes integer numpy array of shape [num_boxes] containing 1-indexed groundtruth classes for the boxes.

  InputDataFields.groundtruth_instance_masks uint8 numpy array of shape [num_boxes, image_height, image_width] containing groundtruth masks corresponding to the boxes. The elements of the array must be in {0, 1}.

`add_single_detected_image_info`(*image_id*, *detections_dict*)

Adds detections for a single image to be used for evaluation.

If a detection has already been added for this image id, a warning is logged, and the detection is skipped.

Parameters

- **image_id** – A unique string/integer identifier for the image.
- **detections_dict** – A dictionary containing - DetectionResultFields.detection_scores: float32 numpy array of shape [num_boxes] containing detection scores for the boxes. DetectionResultFields.detection_classes: integer numpy array of shape [num_boxes] containing 1-indexed detection classes for the boxes. DetectionResultFields.detection_masks: optional uint8 numpy array of shape [num_boxes, image_height, image_width] containing instance masks corresponding to the boxes. The elements of the array must be in {0, 1}.

Raises **ValueError** – If groundtruth for the image_id is not available or if spatial shapes of groundtruth_instance_masks and detection_masks are incompatible.

class easycv.core.evaluation.coco_evaluation.**CoCoPoseTopDownEvaluator**(*dataset_name=None*, *metric_names=['AP']*, ***kwargs*)

Bases: *easycv.core.evaluation.base_evaluator.Evaluator*

Class to evaluate COCO keypoint topdown metrics.

__init__(*dataset_name=None*, *metric_names=['AP']*, ***kwargs*)
Construct eval ops from tensor

Parameters

- **dataset_name** (*str*) – dataset name to be evaluated
- **metric_names** (*List[str]*) – metric names this evaluator will return

easycv.core.evaluation.coco_tools module

Wrappers for third party pycocotools to be used within object_detection.

Note that nothing in this file is tensorflow related and thus cannot be called directly as a slim metric, for example.

TODO(jonathanhuang): wrap as a slim metric in metrics.py

Usage example: given a set of images with ids in the list image_ids and corresponding lists of numpy arrays encoding groundtruth (boxes and classes) and detections (boxes, scores and classes), where elements of each list correspond to detections/annotations of a single image, then evaluation (in multi-class mode) can be invoked as follows:

groundtruth_dict = coco_tools.ExportGroundtruthToCOCO(image_ids, groundtruth_boxes_list, groundtruth_classes_list, max_num_classes, output_path=None)

detections_list = coco_tools.ExportDetectionsToCOCO(image_ids, detection_boxes_list, detection_scores_list, detection_classes_list, output_path=None)

groundtruth = coco_tools.COCOWrapper(groundtruth_dict) detections = groundtruth.LoadAnnotations(detections_list) evaluator = coco_tools.COCOEvalWrapper(groundtruth, detections,

agnostic_mode=False)

metrics = evaluator.ComputeMetrics()

class easycv.core.evaluation.coco_tools.**COCOWrapper**(*dataset*, *detection_type='bbox'*)
Bases: xtcocotools.coco.COCO

Wrapper for the pycocotools COCO class.

__init__(*dataset*, *detection_type='bbox'*)
COCOWrapper constructor.

See http://mscoco.org/dataset/#format for a description of the format. By default, the coco.COCO class constructor reads from a JSON file. This function duplicates the same behavior but loads from a dictionary, allowing us to perform evaluation without writing to external storage.

Parameters

- **dataset** – a dictionary holding bounding box annotations in the COCO format.
- **detection_type** – type of detections being wrapped. Can be one of ['bbox', 'segmentation']

Raises **ValueError** – if detection_type is unsupported.

LoadAnnotations(*annotations*)
Load annotations dictionary into COCO datastructure.

See http://mscoco.org/dataset/#format for a description of the annotations format. As above, this function replicates the default behavior of the API but does not require writing to external storage.

Parameters **annotations** – python list holding object detection results where each detection is encoded as a dict with required keys ['image_id', 'category_id', 'score'] and one of ['bbox', 'segmentation'] based on *detection_type*.

Returns a coco.COCO datastructure holding object detection annotations results

Raises

- **ValueError** – if annotations is not a list
- **ValueError** – if annotations do not correspond to the images contained in self.

class easycv.core.evaluation.coco_tools.**COCOEvalWrapper**(*groundtruth=None*, *detections=None*, *agnostic_mode=False*, *iou_type='bbox'*)

Bases: *easycv.core.evaluation.custom_cocotools.cocoeval.COCOeval*

Wrapper for the pycocotools COCOeval class.

To evaluate, create two objects (groundtruth_dict and detections_list) using the conventions listed at http://mscoco.org/dataset/#format. Then call evaluation as follows:
groundtruth = coco_tools.COCOWrapper(groundtruth_dict) detections = groundtruth.LoadAnnotations(detections_list) evaluator = coco_tools.COCOEvalWrapper(groundtruth, detections,

agnostic_mode=False)

metrics = evaluator.ComputeMetrics()

__init__(*groundtruth=None*, *detections=None*, *agnostic_mode=False*, *iou_type='bbox'*)
COCOEvalWrapper constructor.

Note that for the area-based metrics to be meaningful, detection and groundtruth boxes must be in image coordinates measured in pixels.

Parameters

- **groundtruth** – a coco.COCO (or coco_tools.COCOWrapper) object holding groundtruth annotations
- **detections** – a coco.COCO (or coco_tools.COCOWrapper) object holding detections
- **agnostic_mode** – boolean (default: False). If True, evaluation ignores class labels, treating all detections as proposals.

- **iou_type** – IOU type to use for evaluation. Supports *bbox* or *segm*.

GetCategory(*category_id*)
Fetches dictionary holding category information given category id.

Parameters **category_id** – integer id

Returns dictionary holding 'id', 'name'.

GetAgnosticMode()
Returns true if COCO Eval is configured to evaluate in agnostic mode.

GetCategoryIdList()
Returns list of valid category ids.

ComputeMetrics(*include_metrics_per_category=False*, *all_metrics_per_category=False*)
Computes detection metrics.

Parameters

- **include_metrics_per_category** – If True, will include metrics per category.
- **all_metrics_per_category** – If true, include all the summery metrics for each category in per_category_ap. Be careful with setting it to true if you have more than handful of categories, because it will pollute your mldash.

Returns

a dictionary holding:

'Precision/mAP': mean average precision over classes averaged over IOU thresholds ranging from .5 to .95 with .05 increments

'Precision/mAP@.50IOU': mean average precision at 50% IOU 'Precision/mAP@.75IOU': mean average precision at 75% IOU 'Precision/mAP (small)': mean average precision for small objects

(area < 32^2 pixels)

'Precision/mAP (medium)': mean average precision for medium sized objects (32^2 pixels < area < 96^2 pixels)

'Precision/mAP (large)': mean average precision for large objects (96^2 pixels < area < 10000^2 pixels)

'Recall/AR@1': average recall with 1 detection 'Recall/AR@10': average recall with 10 detections 'Recall/AR@100': average recall with 100 detections 'Recall/AR@100 (small)': average recall for small objects with 100

detections

'Recall/AR@100 (medium)': average recall for medium objects with 100 detections

'Recall/AR@100 (large)': average recall for large objects with 100 detections

2. per_category_ap: a dictionary holding category specific results with

keys of the form: 'Precision mAP ByCategory/category' (without the supercategory part if no supercategories exist). For backward compatibility 'PerformanceByCategory' is included in the output regardless of

all_metrics_per_category. If evaluating class-agnostic mode, per_category_ap is an empty dictionary.

Return type

1. summary_metrics

Raises `ValueError` – If category_stats does not exist.

`Analyze()`
Analyze detection results.

Args:

Returns

A dictionary containing images of analyzing result images, key is the image name, value is a [H,W,3] numpy array which represent the image content. You can refer to http://cocodataset.org/#detection-eval section 4 Analysis code.

`easycv.core.evaluation.coco_tools.ExportSingleImageGroundtruthToCoco(`*image_id*, *next_annotation_id*, *category_id_set*, *groundtruth_boxes*, *groundtruth_classes*, *groundtruth_masks=None*, *groundtruth_is_crowd=None*, *super_categories=None*`)`

Export groundtruth of a single image to COCO format.

This function converts groundtruth detection annotations represented as numpy arrays to dictionaries that can be ingested by the COCO evaluation API. Note that the image_ids provided here must match the ones given to ExportSingleImageDetectionsToCoco. We assume that boxes and classes are in correspondence - that is: groundtruth_boxes[i, :], and groundtruth_classes[i] are associated with the same groundtruth annotation.

In the exported result, "area" fields are always set to the area of the groundtruth bounding box.

Parameters

- `image_id` – a unique image identifier either of type integer or string.
- `next_annotation_id` – integer specifying the first id to use for the groundtruth annotations. All annotations are assigned a continuous integer id starting from this value.
- `category_id_set` – A set of valid class ids. Groundtruth with classes not in category_id_set are dropped.
- `groundtruth_boxes` – numpy array (float32) with shape [num_gt_boxes, 4]
- `groundtruth_classes` – numpy array (int) with shape [num_gt_boxes]
- `groundtruth_masks` – optional uint8 numpy array of shape [num_detections, image_height, image_width] containing detection_masks.
- `groundtruth_is_crowd` – optional numpy array (int) with shape [num_gt_boxes] indicating whether groundtruth boxes are crowd.
- `super_categories` – optional list of str indicating each box super category

Returns a list of groundtruth annotations for a single image in the COCO format.

Raises `ValueError` – if (1) groundtruth_boxes and groundtruth_classes do not have the right lengths or (2) if each of the elements inside these lists do not have the correct shapes or (3) if image_ids are not integers

easycv.core.evaluation.coco_tools.**ExportGroundtruthToCOCO**(*image_ids*, *groundtruth_boxes*, *groundtruth_classes*, *categories*, *output_path=None*)

Export groundtruth detection annotations in numpy arrays to COCO API.

This function converts a set of groundtruth detection annotations represented as numpy arrays to dictionaries that can be ingested by the COCO API. Inputs to this function are three lists: image ids for each groundtruth image, groundtruth boxes for each image and groundtruth classes respectively. Note that the image_ids provided here must match the ones given to the ExportDetectionsToCOCO function in order for evaluation to work properly. We assume that for each image, boxes, scores and classes are in correspondence — that is: image_id[i], groundtruth_boxes[i, :] and groundtruth_classes[i] are associated with the same groundtruth annotation.

In the exported result, "area" fields are always set to the area of the groundtruth bounding box and "iscrowd" fields are always set to 0. TODO(jonathanhuang): pass in "iscrowd" array for evaluating on COCO dataset.

Parameters

- **image_ids** – a list of unique image identifier either of type integer or string.
- **groundtruth_boxes** – list of numpy arrays with shape [num_gt_boxes, 4] (note that num_gt_boxes can be different for each entry in the list)
- **groundtruth_classes** – list of numpy arrays (int) with shape [num_gt_boxes] (note that num_gt_boxes can be different for each entry in the list)
- **categories** – a list of dictionaries representing all possible categories. Each dict in this list has the following keys:

  'id': (required) an integer id uniquely identifying this category 'name': (required) string representing category name

  e.g., 'cat', 'dog', 'pizza'

  'supercategory': (optional) string representing the supercategory e.g., 'animal', 'vehicle', 'food', etc

- **output_path** – (optional) path for exporting result to JSON

Returns dictionary that can be read by COCO API

Raises **ValueError** – if (1) groundtruth_boxes and groundtruth_classes do not have the right lengths or (2) if each of the elements inside these lists do not have the correct shapes or (3) if image_ids are not integers

easycv.core.evaluation.coco_tools.**ExportSingleImageDetectionBoxesToCoco**(*image_id*, *category_id_set*, *detection_boxes*, *detection_scores*, *detection_classes*)

Export detections of a single image to COCO format.

This function converts detections represented as numpy arrays to dictionaries that can be ingested by the COCO evaluation API. Note that the image_ids provided here must match the ones given to the ExporSingleImageDetectionBoxesToCoco. We assume that boxes, and classes are in correspondence - that is: boxes[i, :], and classes[i] are associated with the same groundtruth annotation.

Parameters

- **image_id** – unique image identifier either of type integer or string.
- **category_id_set** – A set of valid class ids. Detections with classes not in category_id_set are dropped.

- **detection_boxes** – float numpy array of shape [num_detections, 4] containing detection boxes.
- **detection_scores** – float numpy array of shape [num_detections] containing scored for the detection boxes.
- **detection_classes** – integer numpy array of shape [num_detections] containing the classes for detection boxes.

Returns a list of detection annotations for a single image in the COCO format.

Raises **ValueError** – if (1) detection_boxes, detection_scores and detection_classes do not have the right lengths or (2) if each of the elements inside these lists do not have the correct shapes or (3) if image_ids are not integers.

easycv.core.evaluation.coco_tools.**ExportSingleImageDetectionMasksToCoco**(*image_id*, *category_id_set*, *detection_masks*, *detection_scores*, *detection_classes*)

Export detection masks of a single image to COCO format.

This function converts detections represented as numpy arrays to dictionaries that can be ingested by the COCO evaluation API. We assume that detection_masks, detection_scores, and detection_classes are in correspondence - that is: detection_masks[i, :], detection_classes[i] and detection_scores[i]
are associated with the same annotation.

Parameters

- **image_id** – unique image identifier either of type integer or string.
- **category_id_set** – A set of valid class ids. Detections with classes not in category_id_set are dropped.
- **detection_masks** – uint8 numpy array of shape [num_detections, image_height, image_width] containing detection_masks.
- **detection_scores** – float numpy array of shape [num_detections] containing scores for detection masks.
- **detection_classes** – integer numpy array of shape [num_detections] containing the classes for detection masks.

Returns a list of detection mask annotations for a single image in the COCO format.

Raises **ValueError** – if (1) detection_masks, detection_scores and detection_classes do not have the right lengths or (2) if each of the elements inside these lists do not have the correct shapes or (3) if image_ids are not integers.

easycv.core.evaluation.coco_tools.**ExportDetectionsToCOCO**(*image_ids*, *detection_boxes*, *detection_scores*, *detection_classes*, *categories*, *output_path=None*)

Export detection annotations in numpy arrays to COCO API.

This function converts a set of predicted detections represented as numpy arrays to dictionaries that can be ingested by the COCO API. Inputs to this function are lists, consisting of boxes, scores and classes, respectively, corresponding to each image for which detections have been produced. Note that the image_ids provided here must match the ones given to the ExportGroundtruthToCOCO function in order for evaluation to work properly.

We assume that for each image, boxes, scores and classes are in correspondence — that is: detection_boxes[i, :], detection_scores[i] and detection_classes[i] are associated with the same detection.

Parameters

- **image_ids** – a list of unique image identifier either of type integer or string.
- **detection_boxes** – list of numpy arrays with shape [num_detection_boxes, 4]
- **detection_scores** – list of numpy arrays (float) with shape [num_detection_boxes]. Note that num_detection_boxes can be different for each entry in the list.
- **detection_classes** – list of numpy arrays (int) with shape [num_detection_boxes]. Note that num_detection_boxes can be different for each entry in the list.
- **categories** – a list of dictionaries representing all possible categories. Each dict in this list must have an integer 'id' key uniquely identifying this category.
- **output_path** – (optional) path for exporting result to JSON

Returns list of dictionaries that can be read by COCO API, where each entry corresponds to a single detection and has keys from: ['image_id', 'category_id', 'bbox', 'score'].

Raises **ValueError** – if (1) detection_boxes and detection_classes do not have the right lengths or (2) if each of the elements inside these lists do not have the correct shapes or (3) if image_ids are not integers.

easycv.core.evaluation.coco_tools.**ExportSegmentsToCOCO**(*image_ids*, *detection_masks*, *detection_scores*, *detection_classes*, *categories*, *output_path=None*)

Export segmentation masks in numpy arrays to COCO API.

This function converts a set of predicted instance masks represented as numpy arrays to dictionaries that can be ingested by the COCO API. Inputs to this function are lists, consisting of segments, scores and classes, respectively, corresponding to each image for which detections have been produced.

Note this function is recommended to use for small dataset. For large dataset, it should be used with a merge function (e.g. in map reduce), otherwise the memory consumption is large.

We assume that for each image, masks, scores and classes are in correspondence — that is: detection_masks[i, :, :, :], detection_scores[i] and detection_classes[i] are associated with the same detection.

Parameters

- **image_ids** – list of image ids (typically ints or strings)
- **detection_masks** – list of numpy arrays with shape [num_detection, h, w, 1] and type uint8. The height and width should match the shape of corresponding image.
- **detection_scores** – list of numpy arrays (float) with shape [num_detection]. Note that num_detection can be different for each entry in the list.
- **detection_classes** – list of numpy arrays (int) with shape [num_detection]. Note that num_detection can be different for each entry in the list.
- **categories** – a list of dictionaries representing all possible categories. Each dict in this list must have an integer 'id' key uniquely identifying this category.
- **output_path** – (optional) path for exporting result to JSON

Returns list of dictionaries that can be read by COCO API, where each entry corresponds to a single detection and has keys from: ['image_id', 'category_id', 'segmentation', 'score'].

Raises **ValueError** – if detection_masks and detection_classes do not have the right lengths or if each of the elements inside these lists do not have the correct shapes.

easycv.core.evaluation.coco_tools.**ExportKeypointsToCOCO**(*image_ids*, *detection_keypoints*, *detection_scores*, *detection_classes*, *categories*, *output_path=None*)

Exports keypoints in numpy arrays to COCO API.

This function converts a set of predicted keypoints represented as numpy arrays to dictionaries that can be ingested by the COCO API. Inputs to this function are lists, consisting of keypoints, scores and classes, respectively, corresponding to each image for which detections have been produced.

We assume that for each image, keypoints, scores and classes are in correspondence — that is: detection_keypoints[i, :, :, :], detection_scores[i] and detection_classes[i] are associated with the same detection.

Parameters

- **image_ids** – list of image ids (typically ints or strings)
- **detection_keypoints** – list of numpy arrays with shape [num_detection, num_keypoints, 2] and type float32 in absolute x-y coordinates.
- **detection_scores** – list of numpy arrays (float) with shape [num_detection]. Note that num_detection can be different for each entry in the list.
- **detection_classes** – list of numpy arrays (int) with shape [num_detection]. Note that num_detection can be different for each entry in the list.
- **categories** – a list of dictionaries representing all possible categories. Each dict in this list must have an integer 'id' key uniquely identifying this category and an integer 'num_keypoints' key specifying the number of keypoints the category has.
- **output_path** – (optional) path for exporting result to JSON

Returns list of dictionaries that can be read by COCO API, where each entry corresponds to a single detection and has keys from: ['image_id', 'category_id', 'keypoints', 'score'].

Raises **ValueError** – if detection_keypoints and detection_classes do not have the right lengths or if each of the elements inside these lists do not have the correct shapes.

easycv.core.evaluation.faceid_pair_eval module

easycv.core.evaluation.faceid_pair_eval.**calculate_roc**(*thresholds*, *embeddings1*, *embeddings2*, *actual_issame*, *nrof_folds=10*, *pca=0*)

easycv.core.evaluation.faceid_pair_eval.**calculate_accuracy**(*threshold*, *dist*, *actual_issame*)

easycv.core.evaluation.faceid_pair_eval.**calculate_val**(*thresholds*, *embeddings1*, *embeddings2*, *actual_issame*, *far_target*, *nrof_folds=10*)

easycv.core.evaluation.faceid_pair_eval.**calculate_val_far**(*threshold*, *dist*, *actual_issame*)

easycv.core.evaluation.faceid_pair_eval.**faceid_evaluate**(*embeddings*, *actual_issame*, *nrof_folds=10*, *pca=0*)

Do Kfold=nrof_folds faceid pair-match test for embeddings :param embeddings: [N x C] inputs embedding of all dataset :param actual_issame: [N/2, 1] label of is match :param nrof_folds: KFold number :param pca: > 0 means, do pca and trans embedding to [N, pca] feature

Returns KFold average best accuracy and best threshold

class easycv.core.evaluation.faceid_pair_eval.**FaceIDPairEvaluator**(*dataset_name=None*, *metric_names=['acc']*, *kfold=10*, *pca=0*)

Bases: *easycv.core.evaluation.base_evaluator.Evaluator*

FaceIDPairEvaluator evaluator. Input nx2 pairs and label, kfold thresholds search and return average best accuracy

__init__(*dataset_name=None*, *metric_names=['acc']*, *kfold=10*, *pca=0*)

Faceid small dataset evaluator, do pair match validation :param dataset_name: faceid small validate set name, include [lfw, agedb_30, cfp_ff, cfp_fw, calfw] :param kfold: Kfold for train/val split :param pca:

pca dimensions, if > 0, do PCA for input feature, transfer to [n, pca]

Returns None

easycv.core.evaluation.metric_registry module

class easycv.core.evaluation.metric_registry.**MetricRegistry**

Bases: object

__init__()

Initialize self. See help(type(self)) for accurate signature.

get(*evaluator_type*)

register_default_best_metric(*cls*, *metric_name*, *metric_cmp_op='max'*)

Register default best metric for each evaluator

Parameters

- **cls** (*object*) – class object
- **metric_name** (*str or List[str]*) – default best metric name
- **metric_cmp_op** (*str or List[str]*) – metric compare operation, should be one of ["max", "min"]

easycv.core.evaluation.mse_eval module

class easycv.core.evaluation.mse_eval.**MSEEvaluator**(*dataset_name=None*, *metric_names=['avg_mse']*, *neck_num=None*)

Bases: *easycv.core.evaluation.base_evaluator.Evaluator*

MSEEvaluator evaluator,

__init__(*dataset_name=None*, *metric_names=['avg_mse']*, *neck_num=None*)

easycv.core.evaluation.retrival_topk_eval module

class easycv.core.evaluation.retrival_topk_eval.**RetrivalTopKEvaluator**(*topk=(1, 2, 4, 8)*, *norm=0*, *metric='cos'*, *pca=0*, *dataset_name=None*, *metric_names=['R@K=1']*, *save_results=False*, *save_results_dir=''*, *feature_keyword=['neck']*)

Bases: *easycv.core.evaluation.base_evaluator.Evaluator*

RetrivalTopK evaluator, Retrival evaluate do the topK retrival, by measuring the distance of every 1 vs other. get the topK nearest, and count the match of ID. if Retrival = 1, Miss = 0. Finally average all RetrivalRate.

__init__(*topk=(1, 2, 4, 8)*, *norm=0*, *metric='cos'*, *pca=0*, *dataset_name=None*, *metric_names=['R@K=1']*, *save_results=False*, *save_results_dir=''*, *feature_keyword=['neck']*)

Parameters **top_k** – tuple of int, evaluate top_k acc

easycv.core.evaluation.top_down_eval module

easycv.core.evaluation.top_down_eval.**pose_pck_accuracy**(*output*, *target*, *mask*, *thr=0.05*, *normalize=None*)

Calculate the pose accuracy of PCK for each individual keypoint and the averaged accuracy across all keypoints from heatmaps.

Note: PCK metric measures accuracy of the localization of the body joints. The distances between predicted positions and the ground-truth ones are typically normalized by the bounding box size. The threshold (thr) of the normalized distance is commonly set as 0.05, 0.1 or 0.2 etc.

batch_size: N num_keypoints: K heatmap height: H heatmap width: W

Parameters

- **output** (*np.ndarray[N, K, H, W]*) – Model output heatmaps.
- **target** (*np.ndarray[N, K, H, W]*) – Groundtruth heatmaps.
- **mask** (*np.ndarray[N, K]*) – Visibility of the target. False for invisible joints, and True for visible. Invisible joints will be ignored for accuracy calculation.
- **thr** (*float*) – Threshold of PCK calculation. Default 0.05.
- **normalize** (*np.ndarray[N, 2]*) – Normalization factor for H&W.

Returns

A tuple containing keypoint accuracy.

- np.ndarray[K]: Accuracy of each keypoint.
- float: Averaged accuracy across all keypoints.
- int: Number of valid keypoints.

Return type tuple

easycv.core.evaluation.top_down_eval.**keypoint_pck_accuracy**(*pred*, *gt*, *mask*, *thr*, *normalize*)

Calculate the pose accuracy of PCK for each individual keypoint and the averaged accuracy across all keypoints for coordinates.

Note: PCK metric measures accuracy of the localization of the body joints. The distances between predicted positions and the ground-truth ones are typically normalized by the bounding box size. The threshold (thr) of the normalized distance is commonly set as 0.05, 0.1 or 0.2 etc.

batch_size: N num_keypoints: K

Parameters

- **pred** (*np.ndarray[N, K, 2]*) – Predicted keypoint location.
- **gt** (*np.ndarray[N, K, 2]*) – Groundtruth keypoint location.
- **mask** (*np.ndarray[N, K]*) – Visibility of the target. False for invisible joints, and True for visible. Invisible joints will be ignored for accuracy calculation.
- **thr** (*float*) – Threshold of PCK calculation.
- **normalize** (*np.ndarray[N, 2]*) – Normalization factor for H&W.

Returns

A tuple containing keypoint accuracy.

- acc (np.ndarray[K]): Accuracy of each keypoint.
- avg_acc (float): Averaged accuracy across all keypoints.
- cnt (int): Number of valid keypoints.

Return type tuple

easycv.core.evaluation.top_down_eval.**post_dark_udp**(*coords*, *batch_heatmaps*, *kernel=3*)

DARK post-pocessing. Implemented by udp. Paper ref: Huang et al. The Devil is in the Details: Delving into Unbiased Data Processing for Human Pose Estimation (CVPR 2020). Zhang et al. Distribution-Aware Coordinate Representation for Human Pose Estimation (CVPR 2020).

Note: batch size: B num keypoints: K num persons: N height of heatmaps: H width of heatmaps: W B=1 for bottom_up paradigm where all persons share the same heatmap. B=N for top_down paradigm where each person has its own heatmaps.

Parameters

- **coords** (*np.ndarray[N, K, 2]*) – Initial coordinates of human pose.
- **batch_heatmaps** (*np.ndarray[B, K, H, W]*) – batch_heatmaps
- **kernel** (*int*) – Gaussian kernel size (K) for modulation.

Returns Refined coordinates.

Return type res (np.ndarray[N, K, 2])

easycv.core.evaluation.top_down_eval.**keypoints_from_heatmaps**(*heatmaps*, *center*, *scale*, *unbiased=False*, *post_process='default'*, *kernel=11*, *valid_radius_factor=0.0546875*, *use_udp=False*, *target_type='GaussianHeatmap'*)

Get final keypoint predictions from heatmaps and transform them back to the image.

Note: batch size: N num keypoints: K heatmap height: H heatmap width: W

Parameters

- **heatmaps** (*np.ndarray[N, K, H, W], dtype=float32*) – model predicted heatmaps.
- **center** (*np.ndarray[N, 2]*) – Center of the bounding box (x, y).
- **scale** (*np.ndarray[N, 2]*) – Scale of the bounding box wrt height/width.
- **post_process** (*str/None*) – Choice of methods to post-process heatmaps. Currently supported: None, 'default', 'unbiased', 'megvii'.
- **unbiased** (*bool*) – Option to use unbiased decoding. Mutually exclusive with megvii. Note: this arg is deprecated and unbiased=True can be replaced by post_process='unbiased' Paper ref: Zhang et al. Distribution-Aware Coordinate Representation for Human Pose Estimation (CVPR 2020).

- **kernel** (*int*) – Gaussian kernel size (K) for modulation, which should match the heatmap gaussian sigma when training. K=17 for sigma=3 and k=11 for sigma=2.
- **valid_radius_factor** (*float*) – The radius factor of the positive area in classification heatmap for UDP.
- **use_udp** (*bool*) – Use unbiased data processing.
- **target_type** (*str*) – 'GaussianHeatmap' or 'CombinedTarget'. GaussianHeatmap: Classification target with gaussian distribution. CombinedTarget: The combination of classification target (response map) and regression target (offset map). Paper ref: Huang et al. The Devil is in the Details: Delving into Unbiased Data Processing for Human Pose Estimation (CVPR 2020).

Returns

A tuple containing keypoint predictions and scores.

- preds (np.ndarray[N, K, 2]): Predicted keypoint location in images.
- maxvals (np.ndarray[N, K, 1]): Scores (confidence) of the keypoints.

Return type tuple

17.1.2 easycv.core.optimizer package

Submodules

easycv.core.optimizer.lars module

class easycv.core.optimizer.lars.**LARS**(*params*, *lr=<required parameter>*, *momentum=0*, *dampening=0*, *weight_decay=0*, *eta=0.001*, *nesterov=False*)

Bases: torch.optim.optimizer.Optimizer

Implements layer-wise adaptive rate scaling for SGD.

Parameters

- **params** (*iterable*) – iterable of parameters to optimize or dicts defining parameter groups
- **lr** (*float*) – base learning rate (gamma_0)
- **momentum** (*float, optional*) – momentum factor (default: 0) ("m")
- **weight_decay** (*float, optional*) – weight decay (L2 penalty) (default: 0) ("beta")
- **dampening** (*float, optional*) – dampening for momentum (default: 0)
- **eta** (*float, optional*) – LARS coefficient
- **nesterov** (*bool, optional*) – enables Nesterov momentum (default: False)

Based on Algorithm 1 of the following paper by You, Gitman, and Ginsburg. Large Batch Training of Convolutional Networks:

https://arxiv.org/abs/1708.03888

Example

```
>>> optimizer = LARS(model.parameters(), lr=0.1, momentum=0.9,
>>>                  weight_decay=1e-4, eta=1e-3)
>>> optimizer.zero_grad()
>>> loss_fn(model(input), target).backward()
>>> optimizer.step()
```

__init__(*params*, *lr=<required parameter>*, *momentum=0*, *dampening=0*, *weight_decay=0*, *eta=0.001*, *nesterov=False*)
Initialize self. See help(type(self)) for accurate signature.

step(*closure=None*)
Performs a single optimization step.

Parameters **closure** (*callable, optional*) – A closure that reevaluates the model and returns the loss.

easycv.core.optimizer.ranger module

easycv.core.optimizer.ranger.**centralized_gradient**(*x*, *use_gc=True*, *gc_conv_only=False*)
credit - https://github.com/Yonghongwei/Gradient-Centralization

class easycv.core.optimizer.ranger.**Ranger**(*params*, *lr=0.001*, *alpha=0.5*, *k=6*, *N_sma_threshhold=5*, *betas=(0.95, 0.999)*, *eps=1e-05*, *weight_decay=0*, *use_gc=True*, *gc_conv_only=False*, *gc_loc=True*)
Bases: torch.optim.optimizer.Optimizer

Adam+LookAhead: refer to https://github.com/lessw2020/Ranger-Deep-Learning-Optimizer

__init__(*params*, *lr=0.001*, *alpha=0.5*, *k=6*, *N_sma_threshhold=5*, *betas=(0.95, 0.999)*, *eps=1e-05*, *weight_decay=0*, *use_gc=True*, *gc_conv_only=False*, *gc_loc=True*)
Initialize self. See help(type(self)) for accurate signature.

step(*closure=None*)
Performs a single optimization step (parameter update).

Parameters **closure** (*callable*) – A closure that reevaluates the model and returns the loss. Optional for most optimizers.

Note: Unless otherwise specified, this function should not modify the `.grad` field of the parameters.

17.1.3 easycv.core.post_processing package

easycv.core.post_processing.**affine_transform**(*pt*, *trans_mat*)
Apply an affine transformation to the points.

Parameters

- **pt** (*np.ndarray*) – a 2 dimensional point to be transformed
- **trans_mat** (*np.ndarray*) – 2x3 matrix of an affine transform

Returns Transformed points.

Return type np.ndarray

easycv.core.post_processing.**flip_back**(*output_flipped*, *flip_pairs*, *target_type='GaussianHeatmap'*)
Flip the flipped heatmaps back to the original form.

Note: batch_size: N num_keypoints: K heatmap height: H heatmap width: W

Parameters

- **output_flipped** (*np.ndarray[N, K, H, W]*) – The output heatmaps obtained from the flipped images.
- **flip_pairs** (*list[tuple()*) – Pairs of keypoints which are mirrored (for example, left ear – right ear).
- **target_type** (*str*) – GaussianHeatmap or CombinedTarget

Returns heatmaps that flipped back to the original image

Return type np.ndarray

easycv.core.post_processing.**fliplr_joints**(*joints_3d*, *joints_3d_visible*, *img_width*, *flip_pairs*)
Flip human joints horizontally.

Note: num_keypoints: K

Parameters

- **joints_3d** (*np.ndarray([K, 3])*) – Coordinates of keypoints.
- **joints_3d_visible** (*np.ndarray([K, 1])*) – Visibility of keypoints.
- **img_width** (*int*) – Image width.
- **flip_pairs** (*list[tuple()]*) – Pairs of keypoints which are mirrored (for example, left ear – right ear).

Returns

Flipped human joints.

- joints_3d_flipped (np.ndarray([K, 3])): Flipped joints.
- joints_3d_visible_flipped (np.ndarray([K, 1])): Joint visibility.

Return type tuple

easycv.core.post_processing.**fliplr_regression**(*regression*, *flip_pairs*, *center_mode='static'*, *center_x=0.5*, *center_index=0*)
Flip human joints horizontally.

Note: batch_size: N num_keypoint: K

Parameters

- **regression** (*np.ndarray([..., K, C])*) – Coordinates of keypoints, where K is the joint number and C is the dimension. Example shapes are: - [N, K, C]: a batch of keypoints where N is the batch size. - [N, T, K, C]: a batch of pose sequences, where T is the frame

  number.

- **flip_pairs** (*list[tuple()]*) – Pairs of keypoints which are mirrored (for example, left ear – right ear).
- **center_mode** (*str*) – The mode to set the center location on the x-axis to flip around. Options are: - static: use a static x value (see center_x also) - root: use a root joint (see center_index also)
- **center_x** (*float*) – Set the x-axis location of the flip center. Only used when center_mode=static.
- **center_index** (*int*) – Set the index of the root joint, whose x location will be used as the flip center. Only used when center_mode=root.

Returns

Flipped human joints.

- regression_flipped (np.ndarray([..., K, C])): Flipped joints.

Return type tuple

easycv.core.post_processing.**get_affine_transform**(*center*, *scale*, *rot*, *output_size*, *shift=(0.0, 0.0)*, *inv=False*)

Get the affine transform matrix, given the center/scale/rot/output_size.

Parameters

- **center** (*np.ndarray[2,]*) – Center of the bounding box (x, y).
- **scale** (*np.ndarray[2,]*) – Scale of the bounding box wrt [width, height].
- **rot** (*float*) – Rotation angle (degree).
- **output_size** (*np.ndarray[2,] | list(2,)*) – Size of the destination heatmaps.
- **shift** (*0-100%*) – Shift translation ratio wrt the width/height. Default (0., 0.).
- **inv** (*bool*) – Option to inverse the affine transform direction. (inv=False: src->dst or inv=True: dst->src)

Returns The transform matrix.

Return type np.ndarray

easycv.core.post_processing.**get_warp_matrix**(*theta*, *size_input*, *size_dst*, *size_target*)

Calculate the transformation matrix under the constraint of unbiased. Paper ref: Huang et al. The Devil is in the Details: Delving into Unbiased Data Processing for Human Pose Estimation (CVPR 2020).

Parameters

- **theta** (*float*) – Rotation angle in degrees.
- **size_input** (*np.ndarray*) – Size of input image [w, h].
- **size_dst** (*np.ndarray*) – Size of output image [w, h].
- **size_target** (*np.ndarray*) – Size of ROI in input plane [w, h].

Returns A matrix for transformation.

Return type matrix (np.ndarray)

easycv.core.post_processing.**rotate_point**(*pt*, *angle_rad*)

Rotate a point by an angle.

Parameters

- **pt** (*list[float]*) – 2 dimensional point to be rotated
- **angle_rad** (*float*) – rotation angle by radian

Returns Rotated point.

Return type list[float]

easycv.core.post_processing.**transform_preds**(*coords*, *center*, *scale*, *output_size*, *use_udp=False*)
Get final keypoint predictions from heatmaps and apply scaling and translation to map them back to the image.

Note: num_keypoints: K

Parameters

- **coords** (*np.ndarray[K, ndims]*) –
  - If ndims=2, corrds are predicted keypoint location.
  - If ndims=4, corrds are composed of (x, y, scores, tags)
  - If ndims=5, corrds are composed of (x, y, scores, tags, flipped_tags)
- **center** (*np.ndarray[2,]*) – Center of the bounding box (x, y).
- **scale** (*np.ndarray[2,]*) – Scale of the bounding box wrt [width, height].
- **output_size** (*np.ndarray[2,] | list(2,)*) – Size of the destination heatmaps.
- **use_udp** (*bool*) – Use unbiased data processing

Returns Predicted coordinates in the images.

Return type np.ndarray

easycv.core.post_processing.**warp_affine_joints**(*joints*, *mat*)
Apply affine transformation defined by the transform matrix on the joints.

Parameters

- **joints** (*np.ndarray[..., 2]*) – Origin coordinate of joints.
- **mat** (*np.ndarray[3, 2]*) – The affine matrix.

Returns Result coordinate of joints.

Return type matrix (np.ndarray[..., 2])

easycv.core.post_processing.**oks_nms**(*kpts_db*, *thr*, *sigmas=None*, *vis_thr=None*)
OKS NMS implementations.

Parameters

- **kpts_db** – keypoints.
- **thr** – Retain overlap < thr.
- **sigmas** – standard deviation of keypoint labelling.
- **vis_thr** – threshold of the keypoint visibility.

Returns indexes to keep.

Return type np.ndarray

easycv.core.post_processing.**soft_oks_nms**(*kpts_db*, *thr*, *max_dets=20*, *sigmas=None*, *vis_thr=None*)
Soft OKS NMS implementations.

Parameters

- **kpts_db** –
- **thr** – retain oks overlap < thr.

- **max_dets** – max number of detections to keep.
- **sigmas** – Keypoint labelling uncertainty.

Returns indexes to keep.

Return type np.ndarray

Submodules

easycv.core.post_processing.nms module

easycv.core.post_processing.nms.**oks_iou**(*g*, *d*, *a_g*, *a_d*, *sigmas=None*, *vis_thr=None*)
Calculate oks ious.

Parameters

- **g** – Ground truth keypoints.
- **d** – Detected keypoints.
- **a_g** – Area of the ground truth object.
- **a_d** – Area of the detected object.
- **sigmas** – standard deviation of keypoint labelling.
- **vis_thr** – threshold of the keypoint visibility.

Returns The oks ious.

Return type list

easycv.core.post_processing.nms.**oks_nms**(*kpts_db*, *thr*, *sigmas=None*, *vis_thr=None*)
OKS NMS implementations.

Parameters

- **kpts_db** – keypoints.
- **thr** – Retain overlap < thr.
- **sigmas** – standard deviation of keypoint labelling.
- **vis_thr** – threshold of the keypoint visibility.

Returns indexes to keep.

Return type np.ndarray

easycv.core.post_processing.nms.**soft_oks_nms**(*kpts_db*, *thr*, *max_dets=20*, *sigmas=None*, *vis_thr=None*)
Soft OKS NMS implementations.

Parameters

- **kpts_db** –
- **thr** – retain oks overlap < thr.
- **max_dets** – max number of detections to keep.
- **sigmas** – Keypoint labelling uncertainty.

Returns indexes to keep.

Return type np.ndarray

easycv.core.post_processing.pose_transforms module

easycv.core.post_processing.pose_transforms.**fliplr_joints**(*joints_3d*, *joints_3d_visible*, *img_width*, *flip_pairs*)

Flip human joints horizontally.

Note: num_keypoints: K

Parameters

- **joints_3d** (*np.ndarray([K, 3])*) – Coordinates of keypoints.
- **joints_3d_visible** (*np.ndarray([K, 1])*) – Visibility of keypoints.
- **img_width** (*int*) – Image width.
- **flip_pairs** (*list[tuple()]*) – Pairs of keypoints which are mirrored (for example, left ear – right ear).

Returns

Flipped human joints.

- joints_3d_flipped (np.ndarray([K, 3])): Flipped joints.
- joints_3d_visible_flipped (np.ndarray([K, 1])): Joint visibility.

Return type tuple

easycv.core.post_processing.pose_transforms.**fliplr_regression**(*regression*, *flip_pairs*, *center_mode='static'*, *center_x=0.5*, *center_index=0*)

Flip human joints horizontally.

Note: batch_size: N num_keypoint: K

Parameters

- **regression** (*np.ndarray([..., K, C])*) – Coordinates of keypoints, where K is the joint number and C is the dimension. Example shapes are: - [N, K, C]: a batch of keypoints where N is the batch size. - [N, T, K, C]: a batch of pose sequences, where T is the frame

  number.
- **flip_pairs** (*list[tuple()]*) – Pairs of keypoints which are mirrored (for example, left ear – right ear).
- **center_mode** (*str*) – The mode to set the center location on the x-axis to flip around. Options are: - static: use a static x value (see center_x also) - root: use a root joint (see center_index also)
- **center_x** (*float*) – Set the x-axis location of the flip center. Only used when center_mode=static.
- **center_index** (*int*) – Set the index of the root joint, whose x location will be used as the flip center. Only used when center_mode=root.

Returns

Flipped human joints.

- regression_flipped (np.ndarray([..., K, C])): Flipped joints.

Return type tuple

easycv.core.post_processing.pose_transforms.**flip_back**(*output_flipped*, *flip_pairs*, *target_type='GaussianHeatmap'*)

Flip the flipped heatmaps back to the original form.

Note: batch_size: N num_keypoints: K heatmap height: H heatmap width: W

Parameters

- **output_flipped** (*np.ndarray[N, K, H, W]*) – The output heatmaps obtained from the flipped images.
- **flip_pairs** (*list[tuple()]*) – Pairs of keypoints which are mirrored (for example, left ear – right ear).
- **target_type** (*str*) – GaussianHeatmap or CombinedTarget

Returns heatmaps that flipped back to the original image

Return type np.ndarray

easycv.core.post_processing.pose_transforms.**transform_preds**(*coords*, *center*, *scale*, *output_size*, *use_udp=False*)

Get final keypoint predictions from heatmaps and apply scaling and translation to map them back to the image.

Note: num_keypoints: K

Parameters

- **coords** (*np.ndarray[K, ndims]*) –
  - If ndims=2, corrds are predicted keypoint location.
  - If ndims=4, corrds are composed of (x, y, scores, tags)
  - If ndims=5, corrds are composed of (x, y, scores, tags, flipped_tags)
- **center** (*np.ndarray[2,]*) – Center of the bounding box (x, y).
- **scale** (*np.ndarray[2,]*) – Scale of the bounding box wrt [width, height].
- **output_size** (*np.ndarray[2,] | list(2,)*) – Size of the destination heatmaps.
- **use_udp** (*bool*) – Use unbiased data processing

Returns Predicted coordinates in the images.

Return type np.ndarray

easycv.core.post_processing.pose_transforms.**get_affine_transform**(*center*, *scale*, *rot*, *output_size*, *shift=(0.0, 0.0)*, *inv=False*)

Get the affine transform matrix, given the center/scale/rot/output_size.

Parameters

- **center** (*np.ndarray[2,]*) – Center of the bounding box (x, y).
- **scale** (*np.ndarray[2,]*) – Scale of the bounding box wrt [width, height].
- **rot** (*float*) – Rotation angle (degree).

- **output_size** (*np.ndarray[2,] | list(2,)*) – Size of the destination heatmaps.
- **shift** (*0-100%*) – Shift translation ratio wrt the width/height. Default (0., 0.).
- **inv** (*bool*) – Option to inverse the affine transform direction. (inv=False: src->dst or inv=True: dst->src)

Returns The transform matrix.

Return type np.ndarray

easycv.core.post_processing.pose_transforms.**affine_transform**(*pt*, *trans_mat*)
Apply an affine transformation to the points.

Parameters

- **pt** (*np.ndarray*) – a 2 dimensional point to be transformed
- **trans_mat** (*np.ndarray*) – 2x3 matrix of an affine transform

Returns Transformed points.

Return type np.ndarray

easycv.core.post_processing.pose_transforms.**rotate_point**(*pt*, *angle_rad*)
Rotate a point by an angle.

Parameters

- **pt** (*list[float]*) – 2 dimensional point to be rotated
- **angle_rad** (*float*) – rotation angle by radian

Returns Rotated point.

Return type list[float]

easycv.core.post_processing.pose_transforms.**get_warp_matrix**(*theta*, *size_input*, *size_dst*, *size_target*)
Calculate the transformation matrix under the constraint of unbiased. Paper ref: Huang et al. The Devil is in the Details: Delving into Unbiased Data Processing for Human Pose Estimation (CVPR 2020).

Parameters

- **theta** (*float*) – Rotation angle in degrees.
- **size_input** (*np.ndarray*) – Size of input image [w, h].
- **size_dst** (*np.ndarray*) – Size of output image [w, h].
- **size_target** (*np.ndarray*) – Size of ROI in input plane [w, h].

Returns A matrix for transformation.

Return type matrix (np.ndarray)

easycv.core.post_processing.pose_transforms.**warp_affine_joints**(*joints*, *mat*)
Apply affine transformation defined by the transform matrix on the joints.

Parameters

- **joints** (*np.ndarray[..., 2]*) – Origin coordinate of joints.
- **mat** (*np.ndarray[3, 2]*) – The affine matrix.

Returns Result coordinate of joints.

Return type matrix (np.ndarray[..., 2])

17.1.4 easycv.core.visualization package

easycv.core.visualization.**imshow_bboxes**(*img*, *bboxes*, *labels=None*, *colors='green'*, *text_color='white'*, *font_size=20*, *thickness=1*, *font_scale=0.5*, *show=True*, *win_name=''*, *wait_time=0*, *out_file=None*)

Draw bboxes with labels (optional) on an image. This is a wrapper of mmcv.imshow_bboxes.

Parameters

- **img** (*str or ndarray*) – The image to be displayed.
- **bboxes** (*ndarray*) – ndarray of shape (k, 4), each row is a bbox in format [x1, y1, x2, y2].
- **labels** (*str or list[str], optional*) – labels of each bbox.
- **colors** (list[str or tuple or Color]) – A list of colors.
- **text_color** (str or tuple or Color) – Color of texts.
- **font_size** (*int*) – Size of font.
- **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, optional*) – The filename to write the image.

Returns The image with bboxes drawn on it.

Return type ndarray

easycv.core.visualization.**imshow_keypoints**(*img*, *pose_result*, *skeleton=None*, *kpt_score_thr=0.3*, *pose_kpt_color=None*, *pose_link_color=None*, *radius=4*, *thickness=1*, *show_keypoint_weight=False*)

Draw keypoints and links on an image.

Parameters

- **img** (*str or Tensor*) – The image to draw poses on. If an image array is given, id will be modified in-place.
- **pose_result** (*list[kpts]*) – The poses to draw. Each element kpts is a set of K keypoints as an Kx3 numpy.ndarray, where each keypoint is represented as x, y, score.
- **kpt_score_thr** (*float, optional*) – Minimum score of keypoints to be shown. Default: 0.3.
- **pose_kpt_color** (*np.array[Nx3]`*) – Color of N keypoints. If None, the keypoint will not be drawn.
- **pose_link_color** (*np.array[Mx3]*) – Color of M links. If None, the links will not be drawn.
- **thickness** (*int*) – Thickness of lines.

easycv.core.visualization.**imshow_label**(*img*, *labels*, *text_color='blue'*, *font_size=20*, *thickness=1*, *font_scale=0.5*, *intervel=5*, *show=True*, *win_name=''*, *wait_time=0*, *out_file=None*)

Draw images with labels on an image.

Parameters

- **img** (*str or ndarray*) – The image to be displayed.
- **labels** (*str or list[str]*) – labels of each image.
- **text_color** (str or tuple or Color) – Color of texts.
- **font_size** (*int*) – Size of font.
- **thickness** (*int*) – Thickness of lines.
- **font_scale** (*float*) – Font scales of texts.
- **intervelint)** – interval pixels between multiple labels
- **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

Submodules

easycv.core.visualization.image module

easycv.core.visualization.image.**get_font_path**()

easycv.core.visualization.image.**put_text**(*img*, *xy*, *text*, *fill*, *size=20*)
support chinese text

easycv.core.visualization.image.**imshow_label**(*img*, *labels*, *text_color='blue'*, *font_size=20*, *thickness=1*, *font_scale=0.5*, *intervel=5*, *show=True*, *win_name=''*, *wait_time=0*, *out_file=None*)

Draw images with labels on an image.

Parameters

- **img** (*str or ndarray*) – The image to be displayed.
- **labels** (*str or list[str]*) – labels of each image.
- **text_color** (str or tuple or Color) – Color of texts.
- **font_size** (*int*) – Size of font.
- **thickness** (*int*) – Thickness of lines.
- **font_scale** (*float*) – Font scales of texts.
- **intervelint)** – interval pixels between multiple labels
- **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

easycv.core.visualization.image.**imshow_bboxes**(*img*, *bboxes*, *labels=None*, *colors='green'*, *text_color='white'*, *font_size=20*, *thickness=1*, *font_scale=0.5*, *show=True*, *win_name=''*, *wait_time=0*, *out_file=None*)

Draw bboxes with labels (optional) on an image. This is a wrapper of mmcv.imshow_bboxes.

Parameters

- **img** (*str or ndarray*) – The image to be displayed.
- **bboxes** (*ndarray*) – ndarray of shape (k, 4), each row is a bbox in format [x1, y1, x2, y2].
- **labels** (*str or list[str], optional*) – labels of each bbox.
- **colors** (list[str or tuple or Color]) – A list of colors.
- **text_color** (str or tuple or Color) – Color of texts.
- **font_size** (*int*) – Size of font.
- **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, optional*) – The filename to write the image.

Returns The image with bboxes drawn on it.

Return type ndarray

easycv.core.visualization.image.**imshow_keypoints**(*img*, *pose_result*, *skeleton=None*, *kpt_score_thr=0.3*, *pose_kpt_color=None*, *pose_link_color=None*, *radius=4*, *thickness=1*, *show_keypoint_weight=False*)

Draw keypoints and links on an image.

Parameters

- **img** (*str or Tensor*) – The image to draw poses on. If an image array is given, id will be modified in-place.
- **pose_result** (*list[kpts]*) – The poses to draw. Each element kpts is a set of K keypoints as an Kx3 numpy.ndarray, where each keypoint is represented as x, y, score.
- **kpt_score_thr** (*float, optional*) – Minimum score of keypoints to be shown. Default: 0.3.
- **pose_kpt_color** (*np.array[Nx3]`*) – Color of N keypoints. If None, the keypoint will not be drawn.
- **pose_link_color** (*np.array[Mx3]*) – Color of M links. If None, the links will not be drawn.
- **thickness** (*int*) – Thickness of lines.

17.2 Submodules

17.3 easycv.core.standard_fields module

Contains classes specifying naming conventions used for object detection.

Specifies: InputDataFields: standard fields used by reader/preprocessor/batcher. DetectionResultFields: standard fields returned by object detector. BoxListFields: standard field used by BoxList TfExampleFields: standard fields for tf-example data format (go/tf-example).

class easycv.core.standard_fields.**InputDataFields**
Bases: object

Names for the input tensors.

Holds the standard data field names to use for identifying input tensors. This should be used by the decoder to identify keys for the returned tensor_dict containing input tensors. And it should be used by the model to identify the tensors it needs.

image
image.

original_image
image in the original input size.

key
unique key corresponding to image.

source_id
source of the original image.

filename
original filename of the dataset (without common path).

groundtruth_image_classes
image-level class labels.

groundtruth_boxes
coordinates of the ground truth boxes in the image.

groundtruth_classes
box-level class labels.

groundtruth_label_types
box-level label types (e.g. explicit negative).

groundtruth_is_crowd
[DEPRECATED, use groundtruth_group_of instead] is the groundtruth a single object or a crowd.

groundtruth_area
area of a groundtruth segment.

groundtruth_difficult
is a *difficult* object

groundtruth_group_of
is a *group_of* objects, e.g. multiple objects of the same class, forming a connected group, where instances are heavily occluding each other.

proposal_boxes
coordinates of object proposal boxes.

proposal_objectness
: objectness score of each proposal.

groundtruth_instance_masks
: ground truth instance masks.

groundtruth_instance_boundaries
: ground truth instance boundaries.

groundtruth_instance_classes
: instance mask-level class labels.

groundtruth_keypoints
: ground truth keypoints.

groundtruth_keypoint_visibilities
: ground truth keypoint visibilities.

groundtruth_label_scores
: groundtruth label scores.

groundtruth_weights
: groundtruth weight factor for bounding boxes.

num_groundtruth_boxes
: number of groundtruth boxes.

true_image_shapes
: true shapes of images in the resized images, as resized images can be padded with zeros.

image = 'image'

mask = 'mask'

width = 'width'

height = 'height'

original_image = 'original_image'

optical_flow = 'optical_flow'

key = 'key'

source_id = 'source_id'

filename = 'filename'

dataset_name = 'dataset_name'

groundtruth_image_classes = 'groundtruth_image_classes'

groundtruth_image_classes_num = 'groundtruth_image_classes_num'

groundtruth_boxes = 'groundtruth_boxes'

groundtruth_classes = 'groundtruth_classes'

groundtruth_label_types = 'groundtruth_label_types'

groundtruth_is_crowd = 'groundtruth_is_crowd'

groundtruth_area = 'groundtruth_area'

groundtruth_difficult = 'groundtruth_difficult'

groundtruth_group_of = 'groundtruth_group_of'

proposal_boxes = 'proposal_boxes'

proposal_objectness = 'proposal_objectness'

groundtruth_instance_masks = 'groundtruth_instance_masks'

groundtruth_instance_boundaries = 'groundtruth_instance_boundaries'

groundtruth_instance_classes = 'groundtruth_instance_classes'

groundtruth_keypoints = 'groundtruth_keypoints'

groundtruth_keypoint_visibilities = 'groundtruth_keypoint_visibilities'

groundtruth_label_scores = 'groundtruth_label_scores'

groundtruth_weights = 'groundtruth_weights'

num_groundtruth_boxes = 'num_groundtruth_boxes'

true_image_shape = 'true_image_shape'

original_image_shape = 'original_image_shape'

original_instance_masks = 'original_instance_masks'

groundtruth_boxes_absolute = 'groundtruth_boxes_absolute'

groundtruth_keypoints_absolute = 'groundtruth_keypoints_absolute'

label_map = 'label_map'

char_dict = 'char_dict'

class easycv.core.standard_fields.**DetectionResultFields**

Bases: object

Naming conventions for storing the output of the detector.

source_id
source of the original image.

key
unique key corresponding to image.

detection_boxes
coordinates of the detection boxes in the image.

detection_scores
detection scores for the detection boxes in the image.

detection_classes
detection-level class labels.

detection_masks
contains a segmentation mask for each detection box.

detection_boundaries
contains an object boundary for each detection box.

detection_keypoints
contains detection keypoints for each detection box.

num_detections
number of detections in the batch.

source_id = 'source_id'

key = 'key'

detection_boxes = 'detection_boxes'

detection_scores = 'detection_scores'

detection_classes = 'detection_classes'

detection_masks = 'detection_masks'

detection_boundaries = 'detection_boundaries'

detection_keypoints = 'detection_keypoints'

num_detections = 'num_detections'

class easycv.core.standard_fields.**TfExampleFields**

Bases: object

TF-example proto feature names for object detection.

Holds the standard feature names to load from an Example proto for object detection.

image_encoded
JPEG encoded string

image_format
image format, e.g. "JPEG"

filename
filename

channels
number of channels of image

colorspace
colorspace, e.g. "RGB"

height
height of image in pixels, e.g. 462

width
width of image in pixels, e.g. 581

source_id
original source of the image

object_class_text
labels in text format, e.g. ["person", "cat"]

object_class_label
labels in numbers, e.g. [16, 8]

object_bbox_xmin
xmin coordinates of groundtruth box, e.g. 10, 30

object_bbox_xmax
xmax coordinates of groundtruth box, e.g. 50, 40

object_bbox_ymin
ymin coordinates of groundtruth box, e.g. 40, 50

object_bbox_ymax
ymax coordinates of groundtruth box, e.g. 80, 70

object_view
viewpoint of object, e.g. ["frontal", "left"]

object_truncated
is object truncated, e.g. [true, false]

object_occluded
is object occluded, e.g. [true, false]

object_difficult
is object difficult, e.g. [true, false]

object_group_of
is object a single object or a group of objects

object_depiction
is object a depiction

object_is_crowd
[DEPRECATED, use object_group_of instead] is the object a single object or a crowd

object_segment_area
the area of the segment.

object_weight
a weight factor for the object's bounding box.

instance_masks
instance segmentation masks.

instance_boundaries
instance boundaries.

instance_classes
Classes for each instance segmentation mask.

detection_class_label
class label in numbers.

detection_bbox_ymin
ymin coordinates of a detection box.

detection_bbox_xmin
xmin coordinates of a detection box.

detection_bbox_ymax
ymax coordinates of a detection box.

detection_bbox_xmax
xmax coordinates of a detection box.

detection_score
detection score for the class label and box.

image_encoded = 'image/encoded'

image_format = 'image/format'

filename = 'image/filename'

channels = 'image/channels'

colorspace = 'image/colorspace'

height = 'image/height'

```
width = 'image/width'
source_id = 'image/source_id'
object_class_text = 'image/object/class/text'
object_class_label = 'image/object/class/label'
object_bbox_ymin = 'image/object/bbox/ymin'
object_bbox_xmin = 'image/object/bbox/xmin'
object_bbox_ymax = 'image/object/bbox/ymax'
object_bbox_xmax = 'image/object/bbox/xmax'
object_view = 'image/object/view'
object_truncated = 'image/object/truncated'
object_occluded = 'image/object/occluded'
object_difficult = 'image/object/difficult'
object_group_of = 'image/object/group_of'
object_depiction = 'image/object/depiction'
object_is_crowd = 'image/object/is_crowd'
object_segment_area = 'image/object/segment/area'
object_weight = 'image/object/weight'
instance_masks = 'image/segmentation/object'
instance_boundaries = 'image/boundaries/object'
instance_classes = 'image/segmentation/object/class'
detection_class_label = 'image/detection/label'
detection_bbox_ymin = 'image/detection/bbox/ymin'
detection_bbox_xmin = 'image/detection/bbox/xmin'
detection_bbox_ymax = 'image/detection/bbox/ymax'
detection_bbox_xmax = 'image/detection/bbox/xmax'
detection_score = 'image/detection/score'
```

CHAPTER

EIGHTEEN

EASYCV.MODELS PACKAGE

18.1 Subpackages

18.1.1 easycv.models.backbones package

Submodules

easycv.models.backbones.benchmark_mlp module

class easycv.models.backbones.benchmark_mlp.**BenchMarkMLP**(*feature_num*, *num_classes=1000*, *avg_pool=False*, ***kwargs*)

Bases: torch.nn.modules.module.Module

__init__(*feature_num*, *num_classes=1000*, *avg_pool=False*, ***kwargs*)
Initializes internal Module state, shared by both nn.Module and ScriptModule.

init_weights()

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.

---

training: bool

easycv.models.backbones.bninception module

This model is taken from the official PyTorch model zoo. - torchvision.models.mobilenet.py on 31th Aug, 2019

class easycv.models.backbones.bninception.**BNInception**(*num_classes=0*)

Bases: torch.nn.modules.module.Module

__init__(*num_classes=0*)
Initializes internal Module state, shared by both nn.Module and ScriptModule.

init_weights()

features(*input*)

logits(*features*)

forward(*input*)

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.

---

training: bool

easycv.models.backbones.darknet module

class easycv.models.backbones.darknet.**Darknet**(*depth*, *in_channels=3*, *stem_out_channels=32*, *out_features=('dark3', 'dark4', 'dark5')*)

Bases: `torch.nn.modules.module.Module`

depth2blocks = {21: [1, 2, 2, 1], 53: [2, 8, 8, 4]}

__init__(*depth*, *in_channels=3*, *stem_out_channels=32*, *out_features=('dark3', 'dark4', 'dark5')*)

Parameters

- **depth** (*int*) – depth of darknet used in model, usually use [21, 53] for this param.
- **in_channels** (*int*) – number of input channels, for example, use 3 for RGB image.
- **stem_out_channels** (*int*) – number of output chanels of darknet stem. It decides channels of darknet layer2 to layer5.
- **out_features** (*Tuple[str]*) – desired output layer name.

make_group_layer(*in_channels: int*, *num_blocks: int*, *stride: int = 1*)

starts with conv layer then has *num_blocks ResLayer*

make_spp_block(*filters_list*, *in_filters*)

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.

---

training: bool

class easycv.models.backbones.darknet.**CSPDarknet**(*dep_mul*, *wid_mul*, *out_features=('dark3', 'dark4', 'dark5')*, *depthwise=False*, *act='silu'*)

Bases: `torch.nn.modules.module.Module`

__init__(*dep_mul*, *wid_mul*, *out_features=('dark3', 'dark4', 'dark5')*, *depthwise=False*, *act='silu'*)

Initializes internal Module state, shared by both nn.Module and ScriptModule.

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.

---

training: bool

easycv.models.backbones.genet module

easycv.models.backbones.genet.**remove_bn_in_superblock**(*super_block*)

easycv.models.backbones.genet.**fuse_bn**(*model*)

class easycv.models.backbones.genet.**PlainNetBasicBlockClass**(*in_channels=0*, *out_channels=0*, *stride=1*, *no_create=False*, *block_name=None*, ***kwargs*)

Bases: `torch.nn.modules.module.Module`

__init__(*in_channels=0*, *out_channels=0*, *stride=1*, *no_create=False*, *block_name=None*, ***kwargs*)
Initializes internal Module state, shared by both nn.Module and ScriptModule.

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.

---

static create_from_str(*s*, *no_create=False*)

static is_instance_from_str(*s*)

training: bool

class easycv.models.backbones.genet.**AdaptiveAvgPool**(*out_channels*, *output_size*, *no_create=False*, *block_name=None*, ***kwargs*)

Bases: *easycv.models.backbones.genet.PlainNetBasicBlockClass*

__init__(*out_channels*, *output_size*, *no_create=False*, *block_name=None*, ***kwargs*)
Initializes internal Module state, shared by both nn.Module and ScriptModule.

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.

---

static create_from_str(*s*, *no_create=False*)

static is_instance_from_str(*s*)

training: bool

class easycv.models.backbones.genet.**BN**(*out_channels=None*, *copy_from=None*, *no_create=False*, *block_name=None*, ***kwargs*)

Bases: *easycv.models.backbones.genet.PlainNetBasicBlockClass*

__init__(*out_channels=None*, *copy_from=None*, *no_create=False*, *block_name=None*, ***kwargs*)
Initializes internal Module state, shared by both nn.Module and ScriptModule.

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.

---

static create_from_str(*s*, *no_create=False*)

static is_instance_from_str(*s*)

training: bool

class easycv.models.backbones.genet.**ConvDW**(*out_channels=None*, *kernel_size=None*, *stride=None*, *copy_from=None*, *no_create=False*, *block_name=None*, ***kwargs*)

Bases: *easycv.models.backbones.genet.PlainNetBasicBlockClass*

__init__(*out_channels=None*, *kernel_size=None*, *stride=None*, *copy_from=None*, *no_create=False*, *block_name=None*, ***kwargs*)
Initializes internal Module state, shared by both nn.Module and ScriptModule.

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.

---

static create_from_str(*s*, *no_create=False*)

static is_instance_from_str(*s*)

training: bool

class easycv.models.backbones.genet.**ConvKX**(*in_channels=None*, *out_channels=None*, *kernel_size=None*, *stride=None*, *copy_from=None*, *no_create=False*, *block_name=None*, ***kwargs*)

Bases: *easycv.models.backbones.genet.PlainNetBasicBlockClass*

__init__(*in_channels=None*, *out_channels=None*, *kernel_size=None*, *stride=None*, *copy_from=None*, *no_create=False*, *block_name=None*, ***kwargs*)
Initializes internal Module state, shared by both nn.Module and ScriptModule.

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.

---

static create_from_str(*s*, *no_create=False*)

static is_instance_from_str(*s*)

training: bool

class easycv.models.backbones.genet.**Flatten**(*out_channels*, *no_create=False*, *block_name=None*, ***kwargs*)

Bases: *easycv.models.backbones.genet.PlainNetBasicBlockClass*

__init__(*out_channels*, *no_create=False*, *block_name=None*, ***kwargs*)

Initializes internal Module state, shared by both nn.Module and ScriptModule.

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.

---

static create_from_str(*s*, *no_create=False*)

static is_instance_from_str(*s*)

training: bool

class easycv.models.backbones.genet.**Linear**(*in_channels=None*, *out_channels=None*, *bias=None*, *copy_from=None*, *no_create=False*, *block_name=None*, ***kwargs*)

Bases: *easycv.models.backbones.genet.PlainNetBasicBlockClass*

__init__(*in_channels=None*, *out_channels=None*, *bias=None*, *copy_from=None*, *no_create=False*, *block_name=None*, ***kwargs*)

Initializes internal Module state, shared by both nn.Module and ScriptModule.

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.

---

static create_from_str(*s*, *no_create=False*)

static is_instance_from_str(*s*)

training: bool

class easycv.models.backbones.genet.**MaxPool**(*out_channels*, *kernel_size*, *stride*, *no_create=False*, *block_name=None*, ***kwargs*)

Bases: *easycv.models.backbones.genet.PlainNetBasicBlockClass*

__init__(*out_channels*, *kernel_size*, *stride*, *no_create=False*, *block_name=None*, ***kwargs*)
Initializes internal Module state, shared by both nn.Module and ScriptModule.

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.

---

static create_from_str(*s*, *no_create=False*)

static is_instance_from_str(*s*)

training: bool

class easycv.models.backbones.genet.**MultiSumBlock**(*inner_block_list*, *no_create=False*, *block_name=None*, ***kwargs*)

Bases: *easycv.models.backbones.genet.PlainNetBasicBlockClass*

__init__(*inner_block_list*, *no_create=False*, *block_name=None*, ***kwargs*)
Initializes internal Module state, shared by both nn.Module and ScriptModule.

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.

---

static create_from_str(*s*, *no_create=False*)

static is_instance_from_str(*s*)

training: bool

class easycv.models.backbones.genet.**RELU**(*out_channels*, *no_create=False*, *block_name=None*, ***kwargs*)

Bases: *easycv.models.backbones.genet.PlainNetBasicBlockClass*

__init__(*out_channels*, *no_create=False*, *block_name=None*, ***kwargs*)
Initializes internal Module state, shared by both nn.Module and ScriptModule.

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.

static create_from_str(*s*, *no_create=False*)

static is_instance_from_str(*s*)

training: bool

class easycv.models.backbones.genet.**ResBlock**(*inner_block_list*, *in_channels=None*, *stride=None*, *no_create=False*, *block_name=None*, ***kwargs*)

Bases: *easycv.models.backbones.genet.PlainNetBasicBlockClass*

ResBlock(in_channles, inner_blocks_str). If in_channels is missing, use inner_block_list[0].in_channels as in_channels

__init__(*inner_block_list*, *in_channels=None*, *stride=None*, *no_create=False*, *block_name=None*, ***kwargs*)

Initializes internal Module state, shared by both nn.Module and ScriptModule.

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.

static create_from_str(*s*, *no_create=False*)

static is_instance_from_str(*s*)

training: bool

class easycv.models.backbones.genet.**Sequential**(*inner_block_list*, *no_create=False*, *block_name=None*, ***kwargs*)

Bases: *easycv.models.backbones.genet.PlainNetBasicBlockClass*

__init__(*inner_block_list*, *no_create=False*, *block_name=None*, ***kwargs*)

Initializes internal Module state, shared by both nn.Module and ScriptModule.

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.

static create_from_str(*s*, *no_create=False*)

static is_instance_from_str(*s*)

training: bool

class easycv.models.backbones.genet.**SuperResKXKX**(*in_channels=0*, *out_channels=0*, *kernel_size=3*, *stride=1*, *expansion=1.0*, *sublayers=1*, *no_create=False*, *block_name=None*, ***kwargs*)

Bases: *easycv.models.backbones.genet.PlainNetBasicBlockClass*

__init__(*in_channels=0*, *out_channels=0*, *kernel_size=3*, *stride=1*, *expansion=1.0*, *sublayers=1*, *no_create=False*, *block_name=None*, ***kwargs*)

Initializes internal Module state, shared by both nn.Module and ScriptModule.

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.

---

static create_from_str(*s*, *no_create=False*)

static is_instance_from_str(*s*)

training: bool

class easycv.models.backbones.genet.**SuperResK1KX**(*in_channels=0*, *out_channels=0*, *kernel_size=3*, *stride=1*, *expansion=1.0*, *sublayers=1*, *no_create=False*, *block_name=None*, ***kwargs*)

Bases: *easycv.models.backbones.genet.PlainNetBasicBlockClass*

__init__(*in_channels=0*, *out_channels=0*, *kernel_size=3*, *stride=1*, *expansion=1.0*, *sublayers=1*, *no_create=False*, *block_name=None*, ***kwargs*)

Initializes internal Module state, shared by both nn.Module and ScriptModule.

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.

---

static create_from_str(*s*, *no_create=False*)

static is_instance_from_str(*s*)

training: bool

class easycv.models.backbones.genet.**SuperResK1KXK1**(*in_channels=0*, *out_channels=0*, *kernel_size=3*, *stride=1*, *expansion=1.0*, *sublayers=1*, *no_create=False*, *block_name=None*, ***kwargs*)

Bases: *easycv.models.backbones.genet.PlainNetBasicBlockClass*

__init__(*in_channels=0*, *out_channels=0*, *kernel_size=3*, *stride=1*, *expansion=1.0*, *sublayers=1*, *no_create=False*, *block_name=None*, ***kwargs*)

Initializes internal Module state, shared by both nn.Module and ScriptModule.

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.

---

static create_from_str(*s*, *no_create=False*)

static is_instance_from_str(*s*)

training: bool

class easycv.models.backbones.genet.**SuperResK1DWK1**(*in_channels=0*, *out_channels=0*, *kernel_size=3*, *stride=1*, *expansion=1.0*, *sublayers=1*, *no_create=False*, *block_name=None*, ***kwargs*)

Bases: *easycv.models.backbones.genet.PlainNetBasicBlockClass*

__init__(*in_channels=0*, *out_channels=0*, *kernel_size=3*, *stride=1*, *expansion=1.0*, *sublayers=1*, *no_create=False*, *block_name=None*, ***kwargs*)

Initializes internal Module state, shared by both nn.Module and ScriptModule.

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.

---

static create_from_str(*s*, *no_create=False*)

static is_instance_from_str(*s*)

training: bool

class easycv.models.backbones.genet.**SuperResK1DW**(*in_channels=0*, *out_channels=0*, *kernel_size=3*, *stride=1*, *expansion=1.0*, *sublayers=1*, *no_create=False*, *block_name=None*, ***kwargs*)

Bases: *easycv.models.backbones.genet.PlainNetBasicBlockClass*

__init__(*in_channels=0*, *out_channels=0*, *kernel_size=3*, *stride=1*, *expansion=1.0*, *sublayers=1*, *no_create=False*, *block_name=None*, ***kwargs*)

Initializes internal Module state, shared by both nn.Module and ScriptModule.

training: bool

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.

---

static create_from_str(*s*, *no_create=False*)

static is_instance_from_str(*s*)

class easycv.models.backbones.genet.**PlainNet**(*plainnet_struct_idx=None*, *num_classes=0*, *no_create=False*, ***kwargs*)

Bases: torch.nn.modules.module.Module

training: bool

__init__(*plainnet_struct_idx=None*, *num_classes=0*, *no_create=False*, ***kwargs*)
Initializes internal Module state, shared by both nn.Module and ScriptModule.

init_weights()

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.

---

easycv.models.backbones.hrnet module

easycv.models.backbones.hrnet.**get_expansion**(*block*, *expansion=None*)
Get the expansion of a residual block.

The block expansion will be obtained by the following order:

1. If `expansion` is given, just return it.
2. If `block` has the attribute `expansion`, then return `block.expansion`.
3. Return the default value according the the block type: 1 for `BasicBlock` and 4 for `Bottleneck`.

Parameters

- **block** (*class*) – The block class.
- **expansion** (*int | None*) – The given expansion ratio.

Returns The expansion of the block.

Return type int

class easycv.models.backbones.hrnet.**Bottleneck**(*in_channels*, *out_channels*, *expansion=4*, *stride=1*, *dilation=1*, *downsample=None*, *style='pytorch'*, *with_cp=False*, *conv_cfg=None*, *norm_cfg={'type': 'BN'}*)

Bases: torch.nn.modules.module.Module

Bottleneck block for ResNet.

Parameters

- **in_channels** (*int*) – Input channels of this block.
- **out_channels** (*int*) – Output channels of this block.
- **expansion** (*int*) – The ratio of `out_channels/mid_channels` where `mid_channels` is the input/output channels of conv2. Default: 4.
- **stride** (*int*) – stride of the block. Default: 1
- **dilation** (*int*) – dilation of convolution. Default: 1

- **downsample** (*nn.Module*) – downsample operation on identity branch. Default: None.
- **style** (*str*) – "pytorch" or "caffe". If set to "pytorch", the stride-two layer is the 3x3 conv layer, otherwise the stride-two layer is the first 1x1 conv layer. Default: "pytorch".
- **with_cp** (*bool*) – Use checkpoint or not. Using checkpoint will save some memory while slowing down the training speed.
- **conv_cfg** (*dict*) – dictionary to construct and config conv layer. Default: None
- **norm_cfg** (*dict*) – dictionary to construct and config norm layer. Default: dict(type='BN')

__init__(*in_channels*, *out_channels*, *expansion=4*, *stride=1*, *dilation=1*, *downsample=None*, *style='pytorch'*, *with_cp=False*, *conv_cfg=None*, *norm_cfg={'type': 'BN'}*)

Initializes internal Module state, shared by both nn.Module and ScriptModule.

property norm1

the normalization layer named "norm1"

Type nn.Module

property norm2

the normalization layer named "norm2"

Type nn.Module

property norm3

the normalization layer named "norm3"

Type nn.Module

forward(*x*)

Forward function.

training: bool

class easycv.models.backbones.hrnet.**HRModule**(*num_branches*, *blocks*, *num_blocks*, *in_channels*, *num_channels*, *multiscale_output=False*, *with_cp=False*, *conv_cfg=None*, *norm_cfg={'type': 'BN'}*, *upsample_cfg={'align_corners': None, 'mode': 'nearest'}*)

Bases: torch.nn.modules.module.Module

High-Resolution Module for HRNet.

In this module, every branch has 4 BasicBlocks/Bottlenecks. Fusion/Exchange is in this module.

__init__(*num_branches*, *blocks*, *num_blocks*, *in_channels*, *num_channels*, *multiscale_output=False*, *with_cp=False*, *conv_cfg=None*, *norm_cfg={'type': 'BN'}*, *upsample_cfg={'align_corners': None, 'mode': 'nearest'}*)

Initializes internal Module state, shared by both nn.Module and ScriptModule.

forward(*x*)

Forward function.

training: bool

class easycv.models.backbones.hrnet.**HRNet**(*arch='w32'*, *extra=None*, *in_channels=3*, *conv_cfg=None*, *norm_cfg={'type': 'BN'}*, *norm_eval=False*, *with_cp=False*, *zero_init_residual=False*, *multi_scale_output=False*)

Bases: torch.nn.modules.module.Module

HRNet backbone.

High-Resolution Representations for Labeling Pixels and Regions

Parameters

- **extra** (*dict*) – detailed configuration for each stage of HRNet.
- **in_channels** (*int*) – Number of input image channels. Default: 3.
- **conv_cfg** (*dict*) – dictionary to construct and config conv layer.
- **norm_cfg** (*dict*) – dictionary to construct and config norm layer.
- **norm_eval** (*bool*) – Whether to set norm layers to eval mode, namely, freeze running stats (mean and var). Note: Effect on Batch Norm and its variants only. Default: False
- **with_cp** (*bool*) – Use checkpoint or not. Using checkpoint will save some memory while slowing down the training speed.
- **zero_init_residual** (*bool*) – whether to use zero init for last norm layer in resblocks to let them behave as identity.

Example

```
>>> from mmpose.models import HRNet
>>> import torch
>>> extra = dict(
>>>     stage1=dict(
>>>         num_modules=1,
>>>         num_branches=1,
>>>         block='BOTTLENECK',
>>>         num_blocks=(4, ),
>>>         num_channels=(64, )),
>>>     stage2=dict(
>>>         num_modules=1,
>>>         num_branches=2,
>>>         block='BASIC',
>>>         num_blocks=(4, 4),
>>>         num_channels=(32, 64)),
>>>     stage3=dict(
>>>         num_modules=4,
>>>         num_branches=3,
>>>         block='BASIC',
>>>         num_blocks=(4, 4, 4),
>>>         num_channels=(32, 64, 128)),
>>>     stage4=dict(
>>>         num_modules=3,
>>>         num_branches=4,
>>>         block='BASIC',
>>>         num_blocks=(4, 4, 4, 4),
>>>         num_channels=(32, 64, 128, 256)))
>>> self = HRNet(extra, in_channels=1)
>>> self.eval()
>>> inputs = torch.rand(1, 1, 32, 32)
>>> level_outputs = self.forward(inputs)
>>> for level_out in level_outputs:
...     print(tuple(level_out.shape))
(1, 32, 8, 8)
```

blocks_dict = {'BASIC': <class 'easycv.models.backbones.resnet.BasicBlock'>, 'BOTTLENECK': <class 'easycv.models.backbones.hrnet.Bottleneck'>}

arch_zoo = {'w18': [[1, 1, 'BOTTLENECK', (4,), (64,)], [1, 2, 'BASIC', (4, 4), (18, 36)], [4, 3, 'BASIC', (4, 4, 4), (18, 36, 72)], [3, 4, 'BASIC', (4, 4, 4, 4), (18, 36, 72, 144)]], 'w30': [[1, 1, 'BOTTLENECK', (4,), (64,)], [1, 2, 'BASIC', (4, 4), (30, 60)], [4, 3, 'BASIC', (4, 4, 4), (30, 60, 120)], [3, 4, 'BASIC', (4, 4, 4, 4), (30, 60, 120, 240)]], 'w32': [[1, 1, 'BOTTLENECK', (4,), (64,)], [1, 2, 'BASIC', (4, 4), (32, 64)], [4, 3, 'BASIC', (4, 4, 4), (32, 64, 128)], [3, 4, 'BASIC', (4, 4, 4, 4), (32, 64, 128, 256)]], 'w40': [[1, 1, 'BOTTLENECK', (4,), (64,)], [1, 2, 'BASIC', (4, 4), (40, 80)], [4, 3, 'BASIC', (4, 4, 4), (40, 80, 160)], [3, 4, 'BASIC', (4, 4, 4, 4), (40, 80, 160, 320)]], 'w44': [[1, 1, 'BOTTLENECK', (4,), (64,)], [1, 2, 'BASIC', (4, 4), (44, 88)], [4, 3, 'BASIC', (4, 4, 4), (44, 88, 176)], [3, 4, 'BASIC', (4, 4, 4, 4), (44, 88, 176, 352)]], 'w48': [[1, 1, 'BOTTLENECK', (4,), (64,)], [1, 2, 'BASIC', (4, 4), (48, 96)], [4, 3, 'BASIC', (4, 4, 4), (48, 96, 192)], [3, 4, 'BASIC', (4, 4, 4, 4), (48, 96, 192, 384)]], 'w64': [[1, 1, 'BOTTLENECK', (4,), (64,)], [1, 2, 'BASIC', (4, 4), (64, 128)], [4, 3, 'BASIC', (4, 4, 4), (64, 128, 256)], [3, 4, 'BASIC', (4, 4, 4, 4), (64, 128, 256, 512)]]}

__init__(*arch='w32'*, *extra=None*, *in_channels=3*, *conv_cfg=None*, *norm_cfg={'type': 'BN'}*, *norm_eval=False*, *with_cp=False*, *zero_init_residual=False*, *multi_scale_output=False*)
Initializes internal Module state, shared by both nn.Module and ScriptModule.

property norm1
the normalization layer named "norm1"

Type nn.Module

property norm2
the normalization layer named "norm2"

Type nn.Module

init_weights()

forward(*x*)
Forward function.

train(*mode=True*)
Convert the model into training mode.

training: bool

parse_arch(*arch*, *extra=None*)

easycv.models.backbones.inceptionv3 module

This model is taken from the official PyTorch model zoo. - torchvision.models.inception.py on 31th Aug, 2019

class easycv.models.backbones.inceptionv3.**Inception3**(*num_classes: int = 0*, *aux_logits: bool = True*, *transform_input: bool = False*)

Bases: torch.nn.modules.module.Module

__init__(*num_classes: int = 0*, *aux_logits: bool = True*, *transform_input: bool = False*) → None

Parameters

- **num_classes** – number of classes based on dataset.

- **aux_logits** – If True, adds two auxiliary branches that can improve training. Default: *False* when pretrained is True otherwise *True*
- **transform_input** – If True, preprocesses the input according to the method with which it was trained on ImageNet. Default: *False*

init_weights()

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.

---

training: bool

easycv.models.backbones.lighthrnet module

easycv.models.backbones.lighthrnet.**channel_shuffle**(*x*, *groups*)

Channel Shuffle operation.

This function enables cross-group information flow for multiple groups convolution layers.

Parameters

- **x** (*Tensor*) – The input tensor.
- **groups** (*int*) – The number of groups to divide the input tensor in the channel dimension.

Returns The output tensor after channel shuffle operation.

Return type Tensor

class easycv.models.backbones.lighthrnet.**SpatialWeighting**(*channels*, *ratio=16*, *conv_cfg=None*, *norm_cfg=None*, *act_cfg=({'type': 'ReLU'}, {'type': 'Sigmoid'})*)

Bases: `torch.nn.modules.module.Module`

Spatial weighting module.

Parameters

- **channels** (*int*) – The channels of the module.
- **ratio** (*int*) – channel reduction ratio.
- **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'), dict(type='Sigmoid')). The last ConvModule uses Sigmoid by default.

__init__(*channels*, *ratio=16*, *conv_cfg=None*, *norm_cfg=None*, *act_cfg=({'type': 'ReLU'}, {'type': 'Sigmoid'})*)

Initializes internal Module state, shared by both nn.Module and ScriptModule.

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.

---

training: bool

class easycv.models.backbones.lighthrnet.**CrossResolutionWeighting**(*channels*, *ratio=16*, *conv_cfg=None*, *norm_cfg=None*, *act_cfg=({'type': 'ReLU'}, {'type': 'Sigmoid'})*)

Bases: `torch.nn.modules.module.Module`

Cross-resolution channel weighting module.

Parameters

- **channels** (*int*) – The channels of the module.
- **ratio** (*int*) – channel reduction ratio.
- **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'), dict(type='Sigmoid')). The last ConvModule uses Sigmoid by default.

__init__(*channels*, *ratio=16*, *conv_cfg=None*, *norm_cfg=None*, *act_cfg=({'type': 'ReLU'}, {'type': 'Sigmoid'})*)

Initializes internal Module state, shared by both nn.Module and ScriptModule.

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.

---

training: bool

class easycv.models.backbones.lighthrnet.**ConditionalChannelWeighting**(*in_channels*, *stride*, *reduce_ratio*, *conv_cfg=None*, *norm_cfg={'type': 'BN'}*, *with_cp=False*)

Bases: `torch.nn.modules.module.Module`

Conditional channel weighting block.

Parameters

- **in_channels** (*int*) – The input channels of the block.

- **stride** (*int*) – Stride of the 3x3 convolution layer.
- **reduce_ratio** (*int*) – channel reduction ratio.
- **conv_cfg** (*dict*) – Config dict for convolution layer. Default: None, which means using conv2d.
- **norm_cfg** (*dict*) – Config dict for normalization layer. Default: dict(type='BN').
- **with_cp** (*bool*) – Use checkpoint or not. Using checkpoint will save some memory while slowing down the training speed. Default: False.

__init__(*in_channels*, *stride*, *reduce_ratio*, *conv_cfg=None*, *norm_cfg={'type': 'BN'}*, *with_cp=False*)
Initializes internal Module state, shared by both nn.Module and ScriptModule.

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.

---

training: bool

class easycv.models.backbones.lighthrnet.**Stem**(*in_channels*, *stem_channels*, *out_channels*, *expand_ratio*, *conv_cfg=None*, *norm_cfg={'type': 'BN'}*, *with_cp=False*)

Bases: `torch.nn.modules.module.Module`

Stem network block.

Parameters

- **in_channels** (*int*) – The input channels of the block.
- **stem_channels** (*int*) – Output channels of the stem layer.
- **out_channels** (*int*) – The output channels of the block.
- **expand_ratio** (*int*) – adjusts number of channels of the hidden layer in InvertedResidual by this amount.
- **conv_cfg** (*dict*) – Config dict for convolution layer. Default: None, which means using conv2d.
- **norm_cfg** (*dict*) – Config dict for normalization layer. Default: dict(type='BN').
- **with_cp** (*bool*) – Use checkpoint or not. Using checkpoint will save some memory while slowing down the training speed. Default: False.

__init__(*in_channels*, *stem_channels*, *out_channels*, *expand_ratio*, *conv_cfg=None*, *norm_cfg={'type': 'BN'}*, *with_cp=False*)
Initializes internal Module state, shared by both nn.Module and ScriptModule.

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.

training: bool

class easycv.models.backbones.lighthrnet.**IterativeHead**(*in_channels*, *norm_cfg={'type': 'BN'}*)

Bases: torch.nn.modules.module.Module

Extra iterative head for feature learning.

Parameters

- **in_channels** (*int*) – The input channels of the block.
- **norm_cfg** (*dict*) – Config dict for normalization layer. Default: dict(type='BN').

__init__(*in_channels*, *norm_cfg={'type': 'BN'}*)

Initializes internal Module state, shared by both nn.Module and ScriptModule.

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.

training: bool

class easycv.models.backbones.lighthrnet.**ShuffleUnit**(*in_channels*, *out_channels*, *stride=1*, *conv_cfg=None*, *norm_cfg={'type': 'BN'}*, *act_cfg={'type': 'ReLU'}*, *with_cp=False*)

Bases: torch.nn.modules.module.Module

InvertedResidual block for ShuffleNetV2 backbone.

Parameters

- **in_channels** (*int*) – The input channels of the block.
- **out_channels** (*int*) – The output channels of the block.
- **stride** (*int*) – Stride of the 3x3 convolution layer. Default: 1
- **conv_cfg** (*dict*) – Config dict for convolution layer. Default: None, which means using conv2d.
- **norm_cfg** (*dict*) – Config dict for normalization layer. Default: dict(type='BN').
- **act_cfg** (*dict*) – Config dict for activation layer. Default: dict(type='ReLU').
- **with_cp** (*bool*) – Use checkpoint or not. Using checkpoint will save some memory while slowing down the training speed. Default: False.

__init__(*in_channels*, *out_channels*, *stride=1*, *conv_cfg=None*, *norm_cfg={'type': 'BN'}*, *act_cfg={'type': 'ReLU'}*, *with_cp=False*)

Initializes internal Module state, shared by both nn.Module and ScriptModule.

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.

training: bool

class easycv.models.backbones.lighthrnet.**LiteHRModule**(*num_branches*, *num_blocks*, *in_channels*, *reduce_ratio*, *module_type*, *multiscale_output=False*, *with_fuse=True*, *conv_cfg=None*, *norm_cfg={'type': 'BN'}*, *with_cp=False*)

Bases: `torch.nn.modules.module.Module`

High-Resolution Module for LiteHRNet.

It contains conditional channel weighting blocks and shuffle blocks.

Parameters

- **num_branches** (*int*) – Number of branches in the module.
- **num_blocks** (*int*) – Number of blocks in the module.
- **in_channels** (*list(int)*) – Number of input image channels.
- **reduce_ratio** (*int*) – Channel reduction ratio.
- **module_type** (*str*) – 'LITE' or 'NAIVE'
- **multiscale_output** (*bool*) – Whether to output multi-scale features.
- **with_fuse** (*bool*) – Whether to use fuse layers.
- **conv_cfg** (*dict*) – dictionary to construct and config conv layer.
- **norm_cfg** (*dict*) – dictionary to construct and config norm layer.
- **with_cp** (*bool*) – Use checkpoint or not. Using checkpoint will save some memory while slowing down the training speed.

__init__(*num_branches*, *num_blocks*, *in_channels*, *reduce_ratio*, *module_type*, *multiscale_output=False*, *with_fuse=True*, *conv_cfg=None*, *norm_cfg={'type': 'BN'}*, *with_cp=False*)

Initializes internal Module state, shared by both nn.Module and ScriptModule.

forward(*x*)

Forward function.

training: bool

class easycv.models.backbones.lighthrnet.**LiteHRNet**(*extra*, *in_channels=3*, *conv_cfg=None*, *norm_cfg={'type': 'BN'}*, *norm_eval=False*, *with_cp=False*)

Bases: `torch.nn.modules.module.Module`

Lite-HRNet backbone.

Lite-HRNet: A Lightweight High-Resolution Network

Code adapted from 'https://github.com/HRNet/Lite-HRNet/' 'blob/hrnet/models/backbones/litehrnet.py'

Parameters

- **extra** (*dict*) – detailed configuration for each stage of HRNet.
- **in_channels** (*int*) – Number of input image channels. Default: 3.

- **conv_cfg** (*dict*) – dictionary to construct and config conv layer.
- **norm_cfg** (*dict*) – dictionary to construct and config norm layer.
- **norm_eval** (*bool*) – Whether to set norm layers to eval mode, namely, freeze running stats (mean and var). Note: Effect on Batch Norm and its variants only. Default: False
- **with_cp** (*bool*) – Use checkpoint or not. Using checkpoint will save some memory while slowing down the training speed.

Example

```
>>> from mmpose.models import LiteHRNet
>>> import torch
>>> extra=dict(
>>>    stem=dict(stem_channels=32, out_channels=32, expand_ratio=1),
>>>    num_stages=3,
>>>    stages_spec=dict(
>>>        num_modules=(2, 4, 2),
>>>        num_branches=(2, 3, 4),
>>>        num_blocks=(2, 2, 2),
>>>        module_type=('LITE', 'LITE', 'LITE'),
>>>        with_fuse=(True, True, True),
>>>        reduce_ratios=(8, 8, 8),
>>>        num_channels=(
>>>            (40, 80),
>>>            (40, 80, 160),
>>>            (40, 80, 160, 320),
>>>        )),
>>>    with_head=False)
>>> self = LiteHRNet(extra, in_channels=1)
>>> self.eval()
>>> inputs = torch.rand(1, 1, 32, 32)
>>> level_outputs = self.forward(inputs)
>>> for level_out in level_outputs:
...     print(tuple(level_out.shape))
(1, 40, 8, 8)
```

__init__(*extra*, *in_channels=3*, *conv_cfg=None*, *norm_cfg={'type': 'BN'}*, *norm_eval=False*, *with_cp=False*)

Initializes internal Module state, shared by both nn.Module and ScriptModule.

init_weights()

Initialize the weights in backbone.

Parameters **pretrained** (*str, optional*) – Path to pre-trained weights. Defaults to None.

forward(*x*)

Forward function.

train(*mode=True*)

Convert the model into training mode.

training: bool

easycv.models.backbones.mae_vit_transformer module

Mostly copy-paste from https://github.com/facebookresearch/mae/blob/main/models_mae.py

class easycv.models.backbones.mae_vit_transformer.**MaskedAutoencoderViT**(*img_size=224*, *patch_size=16*, *in_chans=3*, *embed_dim=1024*, *depth=24*, *num_heads=16*, *mlp_ratio=4.0*, *norm_layer=functools.partial(<class 'torch.nn.modules.normalization.LayerNor*, *eps=1e-06)*)

Bases: torch.nn.modules.module.Module

Masked Autoencoder with VisionTransformer backbone. MaskedAutoencoderViT is mostly same as vit_tranformer_dynamic, but with a random_masking func. MaskedAutoencoderViT model can be loaded by vit_tranformer_dynamic.

Parameters

- **img_size** (*int*) – input image size
- **patch_size** (*int*) – patch size
- **in_chans** (*int*) – input image channels
- **embed_dim** (*int*) – feature dimensions
- **depth** (*int*) – number of encoder layers
- **num_heads** (*int*) – Parallel attention heads
- **mlp_ratio** (*float*) – mlp ratio
- **norm_layer** – type of normalization layer

__init__(*img_size=224*, *patch_size=16*, *in_chans=3*, *embed_dim=1024*, *depth=24*, *num_heads=16*, *mlp_ratio=4.0*, *norm_layer=functools.partial(<class 'torch.nn.modules.normalization.LayerNorm'>*, *eps=1e-06)*)

Initializes internal Module state, shared by both nn.Module and ScriptModule.

init_weights()

random_masking(*x*, *mask_ratio*)

Perform per-sample random masking by per-sample shuffling. Per-sample shuffling is done by argsort random noise. x: [N, L, D], sequence

forward(*x*, *mask_ratio*)

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.

---

training: bool

easycv.models.backbones.mnasnet module

This model is taken from the official PyTorch model zoo. - torchvision.models.mnasnet.py on 31th Aug, 2019

class easycv.models.backbones.mnasnet.**MNASNet**(*alpha*, *num_classes=0*, *dropout=0.2*)

Bases: torch.nn.modules.module.Module

MNASNet, as described in https://arxiv.org/pdf/1807.11626.pdf. >>> model = MNASNet(1000, 1.0) >>> x = torch.rand(1, 3, 224, 224) >>> y = model(x) >>> y.dim() 1 >>> y.nelement() 1000

__init__(*alpha*, *num_classes=0*, *dropout=0.2*)

Initializes internal Module state, shared by both nn.Module and ScriptModule.

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.

---

init_weights()

training: bool

easycv.models.backbones.mobilenetv2 module

This model is taken from the official PyTorch model zoo. - torchvision.models.mobilenet.py on 31th Aug, 2019

class easycv.models.backbones.mobilenetv2.**MobileNetV2**(*num_classes=0*, *width_multi=1.0*, *inverted_residual_setting=None*, *round_nearest=8*)

Bases: torch.nn.modules.module.Module

__init__(*num_classes=0*, *width_multi=1.0*, *inverted_residual_setting=None*, *round_nearest=8*)

MobileNet V2 main class :param num_classes: Number of classes :type num_classes: int :param width_multi: Width multiplier - adjusts number of channels in each layer by this amount :type width_multi: float :param inverted_residual_setting: Network structure :param round_nearest: Round the number of channels in each layer to be a multiple of this number :type round_nearest: int :param Set to 1 to turn off rounding:

init_weights()

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.

---

training: bool

easycv.models.backbones.network_blocks module

class easycv.models.backbones.network_blocks.**SiLU**(*inplace=True*)
Bases: torch.nn.modules.module.Module

export-friendly inplace version of nn.SiLU()

__init__(*inplace=True*)
Initializes internal Module state, shared by both nn.Module and ScriptModule.

static 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.

---

training: bool

class easycv.models.backbones.network_blocks.**HSiLU**(*inplace=True*)
Bases: torch.nn.modules.module.Module

export-friendly inplace version of nn.SiLU() hardsigmoid is better than sigmoid when used for edge model

__init__(*inplace=True*)
Initializes internal Module state, shared by both nn.Module and ScriptModule.

static 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.

---

training: bool

easycv.models.backbones.network_blocks.**get_activation**(*name='silu'*, *inplace=True*)

class easycv.models.backbones.network_blocks.**BaseConv**(*in_channels*, *out_channels*, *ksize*, *stride*, *groups=1*, *bias=False*, *act='silu'*)
Bases: torch.nn.modules.module.Module

A Conv2d -> Batchnorm -> silu/leaky relu block

__init__(*in_channels*, *out_channels*, *ksize*, *stride*, *groups=1*, *bias=False*, *act='silu'*)
Initializes internal Module state, shared by both nn.Module and ScriptModule.

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.

fuseforward(*x*)

training: bool

class easycv.models.backbones.network_blocks.**DWConv**(*in_channels*, *out_channels*, *ksize*, *stride=1*, *act='silu'*)

Bases: torch.nn.modules.module.Module

Depthwise Conv + Conv

__init__(*in_channels*, *out_channels*, *ksize*, *stride=1*, *act='silu'*)
Initializes internal Module state, shared by both nn.Module and ScriptModule.

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.

training: bool

class easycv.models.backbones.network_blocks.**Bottleneck**(*in_channels*, *out_channels*, *shortcut=True*, *expansion=0.5*, *depthwise=False*, *act='silu'*)

Bases: torch.nn.modules.module.Module

__init__(*in_channels*, *out_channels*, *shortcut=True*, *expansion=0.5*, *depthwise=False*, *act='silu'*)
Initializes internal Module state, shared by both nn.Module and ScriptModule.

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.

training: bool

class easycv.models.backbones.network_blocks.**ResLayer**(*in_channels: int*)
Bases: torch.nn.modules.module.Module

Residual layer with *in_channels* inputs.

__init__(*in_channels: int*)
Initializes internal Module state, shared by both nn.Module and ScriptModule.

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.

training: bool

class easycv.models.backbones.network_blocks.**SPPBottleneck**(*in_channels*, *out_channels*, *kernel_sizes=(5, 9, 13)*, *activation='silu'*)

Bases: torch.nn.modules.module.Module

Spatial pyramid pooling layer used in YOLOv3-SPP

__init__(*in_channels*, *out_channels*, *kernel_sizes=(5, 9, 13)*, *activation='silu'*)
Initializes internal Module state, shared by both nn.Module and ScriptModule.

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.

training: bool

class easycv.models.backbones.network_blocks.**CSPLayer**(*in_channels*, *out_channels*, *n=1*, *shortcut=True*, *expansion=0.5*, *depthwise=False*, *act='silu'*)

Bases: torch.nn.modules.module.Module

CSP Bottleneck with 3 convolutions

__init__(*in_channels*, *out_channels*, *n=1*, *shortcut=True*, *expansion=0.5*, *depthwise=False*, *act='silu'*)

Parameters

- **in_channels** (*int*) – input channels.
- **out_channels** (*int*) – output channels.
- **n** (*int*) – number of Bottlenecks. Default value: 1.

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.

training: bool

class easycv.models.backbones.network_blocks.**Focus**(*in_channels*, *out_channels*, *ksize=1*, *stride=1*, *act='silu'*)

Bases: torch.nn.modules.module.Module

Focus width and height information into channel space.

__init__(*in_channels*, *out_channels*, *ksize=1*, *stride=1*, *act='silu'*)
Initializes internal Module state, shared by both nn.Module and ScriptModule.

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.

---

training: bool

easycv.models.backbones.pytorch_image_models_wrapper module

class easycv.models.backbones.pytorch_image_models_wrapper.**PytorchImageModelWrapper**(*model_name='resnet50'*, *scriptable=None*, *exportable=None*, *no_jit=None*, ***kwargs*)

Bases: torch.nn.modules.module.Module

Support Backbones From pytorch-image-models.

The PyTorch community has lots of awesome contributions for image models. PyTorch Image Models (timm) is a collection of image models, aim to pull together a wide variety of SOTA models with ability to reproduce ImageNet training results.

Model pages can be found at https://rwightman.github.io/pytorch-image-models/models/

References: https://github.com/rwightman/pytorch-image-models

__init__(*model_name='resnet50'*, *scriptable=None*, *exportable=None*, *no_jit=None*, ***kwargs*)
Inits PytorchImageModelWrapper by timm.create_models :param model_name: name of model to instantiate :type model_name: str :param scriptable: set layer config so that model is jit scriptable (not working for all models yet) :type scriptable: bool :param exportable: set layer config so that model is traceable / ONNX exportable (not fully impl/obeyed yet) :type exportable: bool :param no_jit: set layer config so that model doesn't utilize jit scripted layers (so far activations only) :type no_jit: bool

init_weights(*pretrained=None*)

Parameters

- **pretrained == True** (*if*) –
- **model from default path;** (*load*) –
- **pretrained == False or None** (*if*) –
- **from init weights.** (*load*) –

- **model_name in timm_model_names** (*if*) –
- **model from timm default path;** (*load*) –
- **model_name in _MODEL_MAP** (*if*) –
- **model from easycv default path** (*load*) –

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.

training: bool

easycv.models.backbones.resnest module

ResNet variants

class easycv.models.backbones.resnest.**SplAtConv2d**(*in_channels*, *channels*, *kernel_size*, *stride=(1, 1)*, *padding=(0, 0)*, *dilation=(1, 1)*, *groups=1*, *bias=True*, *radix=2*, *reduction_factor=4*, *rectify=False*, *rectify_avg=False*, *norm_layer=None*, *dropblock_prob=0.0*, ***kwargs*)

Bases: `torch.nn.modules.module.Module`

Split-Attention Conv2d

__init__(*in_channels*, *channels*, *kernel_size*, *stride=(1, 1)*, *padding=(0, 0)*, *dilation=(1, 1)*, *groups=1*, *bias=True*, *radix=2*, *reduction_factor=4*, *rectify=False*, *rectify_avg=False*, *norm_layer=None*, *dropblock_prob=0.0*, ***kwargs*)

Initializes internal Module state, shared by both nn.Module and ScriptModule.

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.

training: bool

class easycv.models.backbones.resnest.**rSoftMax**(*radix*, *cardinality*)

Bases: `torch.nn.modules.module.Module`

__init__(*radix*, *cardinality*)

Initializes internal Module state, shared by both nn.Module and ScriptModule.

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.

---

training: bool

class easycv.models.backbones.resnest.**DropBlock2D**(**args*, ***kwargs*)

Bases: `object`

__init__(**args*, ***kwargs*)

Initialize self. See help(type(self)) for accurate signature.

class easycv.models.backbones.resnest.**GlobalAvgPool2d**

Bases: `torch.nn.modules.module.Module`

__init__()

Global average pooling over the input's spatial dimensions

forward(*inputs*)

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.

---

training: bool

class easycv.models.backbones.resnest.**Bottleneck**(*inplanes*, *planes*, *stride=1*, *downsample=None*, *radix=1*, *cardinality=1*, *bottleneck_width=64*, *avd=False*, *avd_first=False*, *dilation=1*, *is_first=False*, *rectified_conv=False*, *rectify_avg=False*, *norm_layer=None*, *dropblock_prob=0.0*, *last_gamma=False*)

Bases: `torch.nn.modules.module.Module`

ResNet Bottleneck

expansion = 4

__init__(*inplanes*, *planes*, *stride=1*, *downsample=None*, *radix=1*, *cardinality=1*, *bottleneck_width=64*, *avd=False*, *avd_first=False*, *dilation=1*, *is_first=False*, *rectified_conv=False*, *rectify_avg=False*, *norm_layer=None*, *dropblock_prob=0.0*, *last_gamma=False*)

Initializes internal Module state, shared by both nn.Module and ScriptModule.

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.

---

training: bool

class easycv.models.backbones.resnest.ResNeSt(*depth=None, block=<class 'easycv.models.backbones.resnest.Bottleneck'>, layers=[3, 4, 6, 3], radix=2, groups=1, bottleneck_width=64, num_classes=0, dilated=False, dilation=1, deep_stem=True, stem_width=32, avg_down=True, rectified_conv=False, rectify_avg=False, avd=False, avd_first=False, final_drop=0.0, dropblock_prob=0, last_gamma=False, norm_layer=<class 'torch.nn.modules.batchnorm.BatchNorm2d'>*)

Bases: torch.nn.modules.module.Module

ResNet Variants

Parameters

- **block** (Block) – Class for the residual block. Options are BasicBlockV1, BottleneckV1.
- **layers** (*list of int*) – Numbers of layers in each block
- **classes** (*int, default 1000*) – Number of classification classes.
- **dilated** (*bool, default False*) – Applying dilation strategy to pretrained ResNet yielding a stride-8 model, typically used in Semantic Segmentation.
- **norm_layer** (*object*) – Normalization layer used in backbone network (default: mxnet.gluon.nn.BatchNorm; for Synchronized Cross-GPU BachNormalization).
- **Reference** –
  - He, Kaiming, et al. "Deep residual learning for image recognition." Proceedings of the IEEE conference on computer vision and pattern recognition. 2016.
  - Yu, Fisher, and Vladlen Koltun. "Multi-scale context aggregation by dilated convolutions."

arch_settings = {50: ((3, 4, 6, 3), 32), 101: ((3, 4, 23, 3), 64), 200: ((3, 24, 36, 3), 64), 269: ((3, 30, 48, 8), 64)}

__init__(*depth=None, block=<class 'easycv.models.backbones.resnest.Bottleneck'>, layers=[3, 4, 6, 3], radix=2, groups=1, bottleneck_width=64, num_classes=0, dilated=False, dilation=1, deep_stem=True, stem_width=32, avg_down=True, rectified_conv=False, rectify_avg=False, avd=False, avd_first=False, final_drop=0.0, dropblock_prob=0, last_gamma=False, norm_layer=<class 'torch.nn.modules.batchnorm.BatchNorm2d'>*)

Initializes internal Module state, shared by both nn.Module and ScriptModule.

training: bool

init_weights()

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.

---

easycv.models.backbones.resnet module

class easycv.models.backbones.resnet.**BasicBlock**(*inplanes*, *planes*, *stride=1*, *dilation=1*, *downsample=None*, *style='pytorch'*, *with_cp=False*, *conv_cfg=None*, *norm_cfg={'type': 'BN'}*, *frelu=False*)

Bases: torch.nn.modules.module.Module

expansion = 1

__init__(*inplanes*, *planes*, *stride=1*, *dilation=1*, *downsample=None*, *style='pytorch'*, *with_cp=False*, *conv_cfg=None*, *norm_cfg={'type': 'BN'}*, *frelu=False*)

Initializes internal Module state, shared by both nn.Module and ScriptModule.

property norm1

property norm2

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.

---

training: bool

class easycv.models.backbones.resnet.**Bottleneck**(*inplanes*, *planes*, *stride=1*, *dilation=1*, *downsample=None*, *style='pytorch'*, *with_cp=False*, *conv_cfg=None*, *norm_cfg={'type': 'BN'}*, *frelu=False*)

Bases: torch.nn.modules.module.Module

expansion = 4

__init__(*inplanes*, *planes*, *stride=1*, *dilation=1*, *downsample=None*, *style='pytorch'*, *with_cp=False*, *conv_cfg=None*, *norm_cfg={'type': 'BN'}*, *frelu=False*)

Bottleneck block for ResNet. If style is "pytorch", the stride-two layer is the 3x3 conv layer, if it is "caffe", the stride-two layer is the first 1x1 conv layer.

property norm1

property norm2

property norm3

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.

---

training: bool

easycv.models.backbones.resnet.**make_res_layer**(*block*, *inplanes*, *planes*, *blocks*, *stride=1*, *dilation=1*, *style='pytorch'*, *avg_down=False*, *with_cp=False*, *conv_cfg=None*, *norm_cfg={'type': 'BN'}*, *frelu=False*, *multi_grid=None*, *contract_dilation=False*)

class easycv.models.backbones.resnet.**ResNet**(*depth*, *in_channels=3*, *num_stages=4*, *strides=(1, 2, 2, 2)*, *dilations=(1, 1, 1, 1)*, *out_indices=(0, 1, 2, 3, 4)*, *style='pytorch'*, *deep_stem=False*, *avg_down=False*, *num_classes=0*, *frozen_stages=- 1*, *conv_cfg=None*, *norm_cfg={'requires_grad': True, 'type': 'BN'}*, *norm_eval=False*, *with_cp=False*, *frelu=False*, *original_inplanes=64*, *stem_channels=64*, *zero_init_residual=False*, *multi_grid=None*, *contract_dilation=False*)

Bases: torch.nn.modules.module.Module

ResNet backbone.

Parameters

- **depth** (*int*) – Depth of resnet, from {18, 34, 50, 101, 152}.
- **in_channels** (*int*) – Number of input image channels. Normally 3.
- **num_stages** (*int*) – Resnet stages, normally 4.
- **strides** (*Sequence[int]*) – Strides of the first block of each stage.
- **dilations** (*Sequence[int]*) – Dilation of each stage.
- **out_indices** (*Sequence[int]*) – Output from which stages.
- **style** (*str*) – *pytorch* or *caffe*. If set to "pytorch", the stride-two layer is the 3x3 conv layer, otherwise the stride-two layer is the first 1x1 conv layer.
- **deep_stem** (*bool*) – Replace 7x7 conv in input stem with 3 3x3 conv. Default: False.
- **avg_down** (*bool*) – Use AvgPool instead of stride conv when downsampling in the bottleneck. Default: False.
- **frozen_stages** (*int*) – Stages to be frozen (stop grad and set eval mode). -1 means not freezing any parameters.
- **norm_cfg** (*dict*) – dictionary to construct and config norm layer.
- **norm_eval** (*bool*) – Whether to set norm layers to eval mode, namely, freeze running stats (mean and var). Note: Effect on Batch Norm and its variants only.
- **with_cp** (*bool*) – Use checkpoint or not. Using checkpoint will save some memory while slowing down the training speed.
- **original_inplanes** – start channel for first block, default=64
- **stem_channels** (*int*) – Number of stem channels. Default: 64.
- **zero_init_residual** (*bool*) – whether to use zero init for last norm layer in resblocks to let them behave as identity.
- **multi_grid** (*Sequence[int] | None*) – Multi grid dilation rates of last stage. Default: None.
- **contract_dilation** (*bool*) – Whether contract first dilation of each layer Default: False.

Example

```
>>> from easycv.models import ResNet
>>> import torch
>>> self = ResNet(depth=18)
>>> self.eval()
>>> inputs = torch.rand(1, 3, 32, 32)
>>> level_outputs = self.forward(inputs)
>>> for level_out in level_outputs:
...     print(tuple(level_out.shape))
(1, 64, 8, 8)
(1, 128, 4, 4)
(1, 256, 2, 2)
(1, 512, 1, 1)
```

arch_settings = {10: (<class 'easycv.models.backbones.resnet.BasicBlock'>, (1, 1, 1, 1)), 18: (<class 'easycv.models.backbones.resnet.BasicBlock'>, (2, 2, 2, 2)), 34: (<class 'easycv.models.backbones.resnet.BasicBlock'>, (3, 4, 6, 3)), 50: (<class 'easycv.models.backbones.resnet.Bottleneck'>, (3, 4, 6, 3)), 101: (<class 'easycv.models.backbones.resnet.Bottleneck'>, (3, 4, 23, 3)), 152: (<class 'easycv.models.backbones.resnet.Bottleneck'>, (3, 8, 36, 3))}

__init__(*depth*, *in_channels=3*, *num_stages=4*, *strides=(1, 2, 2, 2)*, *dilations=(1, 1, 1, 1)*, *out_indices=(0, 1, 2, 3, 4)*, *style='pytorch'*, *deep_stem=False*, *avg_down=False*, *num_classes=0*, *frozen_stages=- 1*, *conv_cfg=None*, *norm_cfg={'requires_grad': True, 'type': 'BN'}*, *norm_eval=False*, *with_cp=False*, *frelu=False*, *original_inplanes=64*, *stem_channels=64*, *zero_init_residual=False*, *multi_grid=None*, *contract_dilation=False*)

Initializes internal Module state, shared by both nn.Module and ScriptModule.

property norm1

init_weights()

forward(*x*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

Note: Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

train(*mode=True*)

Sets the module in training mode.

This has any effect only on certain modules. See documentations of particular modules for details of their behaviors in training/evaluation mode, if they are affected, e.g. `Dropout`, `BatchNorm`, etc.

Parameters **mode** (*bool*) – whether to set training mode (`True`) or evaluation mode (`False`). Default: `True`.

Returns self

Return type Module

training: bool

class easycv.models.backbones.resnet.**ResNetV1c**(***kwargs*)
Bases: *easycv.models.backbones.resnet.ResNet*

Compared to ResNet, ResNetV1c replaces the 7x7 conv in the input stem with three 3x3 convs. For more details please refer to <https://arxiv.org/abs/1812.01187>.

__init__(***kwargs*)
Initializes internal Module state, shared by both nn.Module and ScriptModule.

training: bool

class easycv.models.backbones.resnet.**ResNetV1d**(***kwargs*)
Bases: *easycv.models.backbones.resnet.ResNet*

Compared to ResNet, ResNetV1d replaces the 7x7 conv in the input stem with three 3x3 convs. And in the downsampling block, a 2x2 avg_pool with stride 2 is added before conv, whose stride is changed to 1.

training: bool

__init__(***kwargs*)
Initializes internal Module state, shared by both nn.Module and ScriptModule.

easycv.models.backbones.resnet_jit module

class easycv.models.backbones.resnet_jit.**BasicBlock**(*inplanes*, *planes*, *stride=1*, *dilation=1*, *downsample=None*, *style='pytorch'*, *with_cp=False*, *conv_cfg=None*, *norm_cfg={'type': 'BN'}*)
Bases: torch.nn.modules.module.Module

expansion = 1

__init__(*inplanes*, *planes*, *stride=1*, *dilation=1*, *downsample=None*, *style='pytorch'*, *with_cp=False*, *conv_cfg=None*, *norm_cfg={'type': 'BN'}*)
Initializes internal Module state, shared by both nn.Module and ScriptModule.

property norm1

property norm2

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.

---

training: bool

class easycv.models.backbones.resnet_jit.**Bottleneck**(*inplanes*, *planes*, *stride=1*, *dilation=1*, *downsample=None*, *style='pytorch'*, *with_cp=False*, *conv_cfg=None*, *norm_cfg={'type': 'BN'}*)
Bases: torch.nn.modules.module.Module

expansion = 4

__init__(*inplanes*, *planes*, *stride=1*, *dilation=1*, *downsample=None*, *style='pytorch'*, *with_cp=False*, *conv_cfg=None*, *norm_cfg={'type': 'BN'}*)

Bottleneck block for ResNet. If style is "pytorch", the stride-two layer is the 3x3 conv layer, if it is "caffe", the stride-two layer is the first 1x1 conv layer.

property norm1

property norm2

property norm3

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.

---

training: bool

easycv.models.backbones.resnet_jit.**make_res_layer**(*block*, *inplanes*, *planes*, *blocks*, *stride=1*, *dilation=1*, *style='pytorch'*, *with_cp=False*, *conv_cfg=None*, *norm_cfg={'type': 'BN'}*)

class easycv.models.backbones.resnet_jit.**ResNetJIT**(*depth*, *in_channels=3*, *num_stages=4*, *strides=(1, 2, 2, 2)*, *dilations=(1, 1, 1, 1)*, *out_indices=(0, 1, 2, 3, 4)*, *style='pytorch'*, *frozen_stages=- 1*, *conv_cfg=None*, *norm_cfg={'requires_grad': True, 'type': 'BN'}*, *norm_eval=False*, *with_cp=False*, *zero_init_residual=False*)

Bases: `torch.nn.modules.module.Module`

ResNet backbone.

Parameters

- **depth** (*int*) – Depth of resnet, from {18, 34, 50, 101, 152}.
- **in_channels** (*int*) – Number of input image channels. Normally 3.
- **num_stages** (*int*) – Resnet stages, normally 4.
- **strides** (*Sequence[int]*) – Strides of the first block of each stage.
- **dilations** (*Sequence[int]*) – Dilation of each stage.
- **out_indices** (*Sequence[int]*) – Output from which stages.
- **style** (*str*) – *pytorch* or *caffe*. If set to "pytorch", the stride-two layer is the 3x3 conv layer, otherwise the stride-two layer is the first 1x1 conv layer.
- **frozen_stages** (*int*) – Stages to be frozen (stop grad and set eval mode). -1 means not freezing any parameters.
- **norm_cfg** (*dict*) – dictionary to construct and config norm layer.
- **norm_eval** (*bool*) – Whether to set norm layers to eval mode, namely, freeze running stats (mean and var). Note: Effect on Batch Norm and its variants only.
- **with_cp** (*bool*) – Use checkpoint or not. Using checkpoint will save some memory while slowing down the training speed.

- **zero_init_residual** (*bool*) – whether to use zero init for last norm layer in resblocks to let them behave as identity.

Example

```
>>> from easycv.models import ResNet
>>> import torch
>>> self = ResNet(depth=18)
>>> self.eval()
>>> inputs = torch.rand(1, 3, 32, 32)
>>> level_outputs = self.forward(inputs)
>>> for level_out in level_outputs:
...     print(tuple(level_out.shape))
(1, 64, 8, 8)
(1, 128, 4, 4)
(1, 256, 2, 2)
(1, 512, 1, 1)
```

arch_settings = {18: (<class 'easycv.models.backbones.resnet_jit.BasicBlock'>, (2, 2, 2, 2)), 34: (<class 'easycv.models.backbones.resnet_jit.BasicBlock'>, (3, 4, 6, 3)), 50: (<class 'easycv.models.backbones.resnet_jit.Bottleneck'>, (3, 4, 6, 3)), 101: (<class 'easycv.models.backbones.resnet_jit.Bottleneck'>, (3, 4, 23, 3)), 152: (<class 'easycv.models.backbones.resnet_jit.Bottleneck'>, (3, 8, 36, 3))}

__init__(*depth*, *in_channels=3*, *num_stages=4*, *strides=(1, 2, 2, 2)*, *dilations=(1, 1, 1, 1)*, *out_indices=(0, 1, 2, 3, 4)*, *style='pytorch'*, *frozen_stages=- 1*, *conv_cfg=None*, *norm_cfg={'requires_grad': True, 'type': 'BN'}*, *norm_eval=False*, *with_cp=False*, *zero_init_residual=False*)

Initializes internal Module state, shared by both nn.Module and ScriptModule.

property norm1

init_weights()

training: bool

forward(*x: torch.Tensor*) → List[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.

---

train(*mode=True*)

Sets the module in training mode.

This has any effect only on certain modules. See documentations of particular modules for details of their behaviors in training/evaluation mode, if they are affected, e.g. `Dropout`, `BatchNorm`, etc.

Parameters **mode** (*bool*) – whether to set training mode (`True`) or evaluation mode (`False`). Default: `True`.

Returns self

Return type Module

easycv.models.backbones.resnext module

class easycv.models.backbones.resnext.**Bottleneck**(*inplanes*, *planes*, *groups=1*, *base_width=4*, ***kwargs*)

Bases: *easycv.models.backbones.resnet.Bottleneck*

__init__(*inplanes*, *planes*, *groups=1*, *base_width=4*, ***kwargs*)

Bottleneck block for ResNeXt. If style is "pytorch", the stride-two layer is the 3x3 conv layer, if it is "caffe", the stride-two layer is the first 1x1 conv layer.

training: bool

easycv.models.backbones.resnext.**make_res_layer**(*block*, *inplanes*, *planes*, *blocks*, *stride=1*, *dilation=1*, *groups=1*, *base_width=4*, *style='pytorch'*, *with_cp=False*, *conv_cfg=None*, *norm_cfg={'type': 'BN'}*)

class easycv.models.backbones.resnext.**ResNeXt**(*groups=1*, *base_width=4*, ***kwargs*)

Bases: *easycv.models.backbones.resnet.ResNet*

ResNeXt backbone.

Parameters

- **depth** (*int*) – Depth of resnet, from {18, 34, 50, 101, 152}.
- **in_channels** (*int*) – Number of input image channels. Normally 3.
- **num_stages** (*int*) – Resnet stages, normally 4.
- **groups** (*int*) – Group of resnext.
- **base_width** (*int*) – Base width of resnext.
- **strides** (*Sequence[int]*) – Strides of the first block of each stage.
- **dilations** (*Sequence[int]*) – Dilation of each stage.
- **out_indices** (*Sequence[int]*) – Output from which stages.
- **style** (*str*) – *pytorch* or *caffe*. If set to "pytorch", the stride-two layer is the 3x3 conv layer, otherwise the stride-two layer is the first 1x1 conv layer.
- **frozen_stages** (*int*) – Stages to be frozen (all param fixed). -1 means not freezing any parameters.
- **norm_cfg** (*dict*) – dictionary to construct and config norm layer.
- **norm_eval** (*bool*) – Whether to set norm layers to eval mode, namely, freeze running stats (mean and var). Note: Effect on Batch Norm and its variants only.
- **with_cp** (*bool*) – Use checkpoint or not. Using checkpoint will save some memory while slowing down the training speed.
- **zero_init_residual** (*bool*) – whether to use zero init for last norm layer in resblocks to let them behave as identity.

Example

```
>>> from easycv.models import ResNeXt
>>> import torch
>>> self = ResNeXt(depth=50)
>>> self.eval()
>>> inputs = torch.rand(1, 3, 32, 32)
>>> level_outputs = self.forward(inputs)
>>> for level_out in level_outputs:
...     print(tuple(level_out.shape))
(1, 256, 8, 8)
(1, 512, 4, 4)
(1, 1024, 2, 2)
(1, 2048, 1, 1)
```

arch_settings = {50: (<class 'easycv.models.backbones.resnext.Bottleneck'>, (3, 4, 6, 3)), 101: (<class 'easycv.models.backbones.resnext.Bottleneck'>, (3, 4, 23, 3)), 152: (<class 'easycv.models.backbones.resnext.Bottleneck'>, (3, 8, 36, 3))}

__init__(*groups=1*, *base_width=4*, ***kwargs*)
Initializes internal Module state, shared by both nn.Module and ScriptModule.

training: bool

easycv.models.backbones.shuffle_transformer module

class easycv.models.backbones.shuffle_transformer.**Mlp**(*in_features*, *hidden_features=None*, *out_features=None*, *act_layer=<class 'torch.nn.modules.activation.ReLU6'>*, *drop=0.0*, *stride=False*)

Bases: torch.nn.modules.module.Module

__init__(*in_features*, *hidden_features=None*, *out_features=None*, *act_layer=<class 'torch.nn.modules.activation.ReLU6'>*, *drop=0.0*, *stride=False*)
Initializes internal Module state, shared by both nn.Module and ScriptModule.

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.

---

training: bool

class easycv.models.backbones.shuffle_transformer.**Attention**(*dim*, *num_heads*, *window_size=1*, *shuffle=False*, *qkv_bias=False*, *qk_scale=None*, *attn_drop=0.0*, *proj_drop=0.0*, *relative_pos_embedding=False*)

Bases: torch.nn.modules.module.Module

__init__(*dim*, *num_heads*, *window_size=1*, *shuffle=False*, *qkv_bias=False*, *qk_scale=None*, *attn_drop=0.0*, *proj_drop=0.0*, *relative_pos_embedding=False*)
Initializes internal Module state, shared by both nn.Module and ScriptModule.

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.

---

training: bool

class easycv.models.backbones.shuffle_transformer.**Block**(*dim*, *out_dim*, *num_heads*, *window_size=1*, *shuffle=False*, *mlp_ratio=4.0*, *qkv_bias=False*, *qk_scale=None*, *drop=0.0*, *attn_drop=0.0*, *drop_path=0.0*, *act_layer=<class 'torch.nn.modules.activation.ReLU6'>*, *norm_layer=<class 'torch.nn.modules.batchnorm.BatchNorm2d'>*, *stride=False*, *relative_pos_embedding=False*)

Bases: torch.nn.modules.module.Module

__init__(*dim*, *out_dim*, *num_heads*, *window_size=1*, *shuffle=False*, *mlp_ratio=4.0*, *qkv_bias=False*, *qk_scale=None*, *drop=0.0*, *attn_drop=0.0*, *drop_path=0.0*, *act_layer=<class 'torch.nn.modules.activation.ReLU6'>*, *norm_layer=<class 'torch.nn.modules.batchnorm.BatchNorm2d'>*, *stride=False*, *relative_pos_embedding=False*)
Initializes internal Module state, shared by both nn.Module and ScriptModule.

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.

---

training: bool

class easycv.models.backbones.shuffle_transformer.**PatchMerging**(*dim*, *out_dim*, *norm_layer=<class 'torch.nn.modules.batchnorm.BatchNorm2d'>*)

Bases: torch.nn.modules.module.Module

__init__(*dim*, *out_dim*, *norm_layer=<class 'torch.nn.modules.batchnorm.BatchNorm2d'>*)
Initializes internal Module state, shared by both nn.Module and ScriptModule.

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.

extra_repr() → str
Set the extra representation of the module

To print customized extra information, you should re-implement this method in your own modules. Both single-line and multi-line strings are acceptable.

training: bool

class easycv.models.backbones.shuffle_transformer.**StageModule**(*layers*, *dim*, *out_dim*, *num_heads*, *window_size=1*, *shuffle=True*, *mlp_ratio=4.0*, *qkv_bias=False*, *qk_scale=None*, *drop=0.0*, *attn_drop=0.0*, *drop_path=0.0*, *act_layer=<class 'torch.nn.modules.activation.ReLU6'>*, *norm_layer=<class 'torch.nn.modules.batchnorm.BatchNorm2d'>*, *relative_pos_embedding=False*)

Bases: `torch.nn.modules.module.Module`

__init__(*layers*, *dim*, *out_dim*, *num_heads*, *window_size=1*, *shuffle=True*, *mlp_ratio=4.0*, *qkv_bias=False*, *qk_scale=None*, *drop=0.0*, *attn_drop=0.0*, *drop_path=0.0*, *act_layer=<class 'torch.nn.modules.activation.ReLU6'>*, *norm_layer=<class 'torch.nn.modules.batchnorm.BatchNorm2d'>*, *relative_pos_embedding=False*)
Initializes internal Module state, shared by both nn.Module and ScriptModule.

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.

training: bool

class easycv.models.backbones.shuffle_transformer.**PatchEmbedding**(*inter_channel=32*, *out_channels=48*)

Bases: `torch.nn.modules.module.Module`

__init__(*inter_channel=32*, *out_channels=48*)
Initializes internal Module state, shared by both nn.Module and ScriptModule.

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.

training: bool

class easycv.models.backbones.shuffle_transformer.**ShuffleTransformer**(*img_size=224*, *in_chans=3*, *num_classes=1000*, *token_dim=32*, *embed_dim=96*, *mlp_ratio=4.0*, *layers=[2, 2, 6, 2]*, *num_heads=[3, 6, 12, 24]*, *relative_pos_embedding=True*, *shuffle=True*, *window_size=7*, *qkv_bias=True*, *qk_scale=None*, *drop_rate=0.0*, *attn_drop_rate=0.0*, *drop_path_rate=0.0*, *has_pos_embed=False*, ***kwargs*)

Bases: torch.nn.modules.module.Module

__init__(*img_size=224*, *in_chans=3*, *num_classes=1000*, *token_dim=32*, *embed_dim=96*, *mlp_ratio=4.0*, *layers=[2, 2, 6, 2]*, *num_heads=[3, 6, 12, 24]*, *relative_pos_embedding=True*, *shuffle=True*, *window_size=7*, *qkv_bias=True*, *qk_scale=None*, *drop_rate=0.0*, *attn_drop_rate=0.0*, *drop_path_rate=0.0*, *has_pos_embed=False*, ***kwargs*)

Initializes internal Module state, shared by both nn.Module and ScriptModule.

init_weights()

no_weight_decay()

no_weight_decay_keywords()

get_classifier()

reset_classifier(*num_classes*, *global_pool=''*)

forward_features(*x*)

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.

training: bool

easycv.models.backbones.shuffle_transformer.**shuffletrans_base_p4_w7_224**(*pretrained=False*, ***kwargs*)

easycv.models.backbones.shuffle_transformer.**shuffletrans_small_p4_w7_224**(*pretrained=False*, ***kwargs*)

easycv.models.backbones.shuffle_transformer.**shuffletrans_tiny_p4_w7_224**(*pretrained=False*, ***kwargs*)

easycv.models.backbones.swin_transformer_dynamic module

Borrow this code from https://github.com/microsoft/esvit/blob/main/models/swin_transformer.py To support dynamic swin-transformer for ssl!

class easycv.models.backbones.swin_transformer_dynamic.**Mlp**(*in_features*, *hidden_features=None*, *out_features=None*, *act_layer=<class 'torch.nn.modules.activation.GELU'>*, *drop=0.0*)

Bases: torch.nn.modules.module.Module

__init__(*in_features*, *hidden_features=None*, *out_features=None*, *act_layer=<class 'torch.nn.modules.activation.GELU'>*, *drop=0.0*)

Initializes internal Module state, shared by both nn.Module and ScriptModule.

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.

---

training: bool

easycv.models.backbones.swin_transformer_dynamic.**window_partition**(*x*, *window_size*)

Parameters

- **x** – (B, H, W, C)
- **window_size** (*int*) – window size

Returns (num_windows*B, window_size, window_size, C)

Return type windows

easycv.models.backbones.swin_transformer_dynamic.**window_reverse**(*windows*, *window_size*, *H*, *W*)

Parameters

- **windows** – (num_windows*B, window_size, window_size, C)
- **window_size** (*int*) – Window size
- **H** (*int*) – Height of image
- **W** (*int*) – Width of image

Returns (B, H, W, C)

Return type x

class easycv.models.backbones.swin_transformer_dynamic.**WindowAttention**(*dim*, *window_size*, *num_heads*, *qkv_bias=True*, *qk_scale=None*, *attn_drop=0.0*, *proj_drop=0.0*)

Bases: torch.nn.modules.module.Module

Window based multi-head self attention (W-MSA) module with relative position bias. It supports both of shifted and non-shifted window.

Parameters

- **dim** (*int*) – Number of input channels.
- **window_size** (*tuple[int]*) – The height and width of the window.
- **num_heads** (*int*) – Number of attention heads.
- **qkv_bias** (*bool, optional*) – If True, add a learnable bias to query, key, value. Default: True
- **qk_scale** (*float | None, optional*) – Override default qk scale of head_dim ** -0.5 if set
- **attn_drop** (*float, optional*) – Dropout ratio of attention weight. Default: 0.0
- **proj_drop** (*float, optional*) – Dropout ratio of output. Default: 0.0

__init__(*dim*, *window_size*, *num_heads*, *qkv_bias=True*, *qk_scale=None*, *attn_drop=0.0*, *proj_drop=0.0*)

Initializes internal Module state, shared by both nn.Module and ScriptModule.

forward(*x*, *mask=None*)

Parameters

- **x** – input features with shape of (num_windows*B, N, C)
- **mask** – (0/-inf) mask with shape of (num_windows, Wh*Ww, Wh*Ww) or None

extra_repr() → str

Set the extra representation of the module

To print customized extra information, you should re-implement this method in your own modules. Both single-line and multi-line strings are acceptable.

flops(*N*)

static compute_macs(*module*, *input*, *output*)

training: bool

class easycv.models.backbones.swin_transformer_dynamic.**SwinTransformerBlock**(*dim*, *input_resolution*, *num_heads*, *window_size=7*, *shift_size=0*, *mlp_ratio=4.0*, *qkv_bias=True*, *qk_scale=None*, *drop=0.0*, *attn_drop=0.0*, *drop_path=0.0*, *act_layer=<class 'torch.nn.modules.activation.GELU*, *norm_layer=<class 'torch.nn.modules.normalization.La*

Bases: torch.nn.modules.module.Module

Swin Transformer Block.

Parameters

- **dim** (*int*) – Number of input channels.
- **input_resolution** (*tuple[int]*) – Input resulotion.
- **num_heads** (*int*) – Number of attention heads.
- **window_size** (*int*) – Window size.
- **shift_size** (*int*) – Shift size for SW-MSA.
- **mlp_ratio** (*float*) – Ratio of mlp hidden dim to embedding dim.
- **qkv_bias** (*bool, optional*) – If True, add a learnable bias to query, key, value. Default: True
- **qk_scale** (*float | None, optional*) – Override default qk scale of head_dim ** -0.5 if set.
- **drop** (*float, optional*) – Dropout rate. Default: 0.0
- **attn_drop** (*float, optional*) – Attention dropout rate. Default: 0.0
- **drop_path** (*float, optional*) – Stochastic depth rate. Default: 0.0
- **act_layer** (*nn.Module, optional*) – Activation layer. Default: nn.GELU
- **norm_layer** (*nn.Module, optional*) – Normalization layer. Default: nn.LayerNorm

__init__(*dim*, *input_resolution*, *num_heads*, *window_size=7*, *shift_size=0*, *mlp_ratio=4.0*, *qkv_bias=True*, *qk_scale=None*, *drop=0.0*, *attn_drop=0.0*, *drop_path=0.0*, *act_layer=<class 'torch.nn.modules.activation.GELU'>*, *norm_layer=<class 'torch.nn.modules.normalization.LayerNorm'>*)

Initializes internal Module state, shared by both nn.Module and ScriptModule.

create_attn_mask(*H*, *W*)

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.

---

extra_repr() → str
Set the extra representation of the module

To print customized extra information, you should re-implement this method in your own modules. Both single-line and multi-line strings are acceptable.

flops()

training: bool

class easycv.models.backbones.swin_transformer_dynamic.**PatchMerging**(*input_resolution*, *dim*, *norm_layer=<class 'torch.nn.modules.normalization.LayerNorm'>*

Bases: torch.nn.modules.module.Module

Patch Merging Layer.
Parameters

- **input_resolution** (*tuple[int]*) – Resolution of input feature.
- **dim** (*int*) – Number of input channels.
- **norm_layer** (*nn.Module, optional*) – Normalization layer. Default: nn.LayerNorm

__init__(*input_resolution*, *dim*, *norm_layer=<class 'torch.nn.modules.normalization.LayerNorm'>*)
Initializes internal Module state, shared by both nn.Module and ScriptModule.

forward(*x*)
Forward function. :param x: Input feature, tensor size (B, H*W, C). :param H: Spatial resolution of the input feature. :param W: Spatial resolution of the input feature.

extra_repr() → str
Set the extra representation of the module

To print customized extra information, you should re-implement this method in your own modules. Both single-line and multi-line strings are acceptable.

flops()

training: bool

class easycv.models.backbones.swin_transformer_dynamic.**BasicLayer**(*dim*, *input_resolution*, *depth*, *num_heads*, *window_size*, *mlp_ratio=4.0*, *qkv_bias=True*, *qk_scale=None*, *drop=0.0*, *attn_drop=0.0*, *drop_path=0.0*, *norm_layer=<class 'torch.nn.modules.normalization.LayerNorm'>*, *downsample=None*)

Bases: torch.nn.modules.module.Module

A basic Swin Transformer layer for one stage.
Parameters

- **dim** (*int*) – Number of input channels.
- **input_resolution** (*tuple[int]*) – Input resulotion.
- **depth** (*int*) – Number of blocks.

- **num_heads** (*int*) – Number of attention heads.
- **window_size** (*int*) – Window size.
- **mlp_ratio** (*float*) – Ratio of mlp hidden dim to embedding dim.
- **qkv_bias** (*bool, optional*) – If True, add a learnable bias to query, key, value. Default: True
- **qk_scale** (*float | None, optional*) – Override default qk scale of head_dim ** -0.5 if set.
- **drop** (*float, optional*) – Dropout rate. Default: 0.0
- **attn_drop** (*float, optional*) – Attention dropout rate. Default: 0.0
- **drop_path** (*float | tuple[float], optional*) – Stochastic depth rate. Default: 0.0
- **norm_layer** (*nn.Module, optional*) – Normalization layer. Default: nn.LayerNorm
- **downsample** (*nn.Module | None, optional*) – Downsample layer at the end of the layer. Default: None

__init__(*dim*, *input_resolution*, *depth*, *num_heads*, *window_size*, *mlp_ratio=4.0*, *qkv_bias=True*, *qk_scale=None*, *drop=0.0*, *attn_drop=0.0*, *drop_path=0.0*, *norm_layer=<class 'torch.nn.modules.normalization.LayerNorm'>*, *downsample=None*)

Initializes internal Module state, shared by both nn.Module and ScriptModule.

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.

---

forward_with_features(*x*)

forward_with_attention(*x*)

extra_repr() → str

Set the extra representation of the module

To print customized extra information, you should re-implement this method in your own modules. Both single-line and multi-line strings are acceptable.

flops()

training: bool

class easycv.models.backbones.swin_transformer_dynamic.**PatchEmbed**(*img_size=224*, *patch_size=16*, *in_chans=3*, *embed_dim=768*, *norm_layer=None*)

Bases: `torch.nn.modules.module.Module`

Image to Patch Embedding

__init__(*img_size=224*, *patch_size=16*, *in_chans=3*, *embed_dim=768*, *norm_layer=None*)

Initializes internal Module state, shared by both nn.Module and ScriptModule.

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.

---

flops()

training: bool

class easycv.models.backbones.swin_transformer_dynamic.**SwinTransformer**(*img_size=224, patch_size=4, in_chans=3, num_classes=1000, embed_dim=96, depths=[2, 2, 6, 2], num_heads=[3, 6, 12, 24], window_size=7, mlp_ratio=4.0, qkv_bias=True, qk_scale=None, drop_rate=0.0, attn_drop_rate=0.0, drop_path_rate=0.1, norm_layer=<class 'torch.nn.modules.normalization.LayerNor ape=False, patch_norm=True, use_dense_prediction=False, **kwargs*)

Bases: `torch.nn.modules.module.Module`

Swin Transformer

A PyTorch impl of [*Swin Transformer: Hierarchical Vision Transformer using Shifted Windows* -] https://arxiv.org/pdf/2103.14030

Parameters

- **img_size** (*int | tuple(int)*) – Input image size.
- **patch_size** (*int | tuple(int)*) – Patch size.
- **in_chans** (*int*) – Number of input channels.
- **num_classes** (*int*) – Number of classes for classification head.
- **embed_dim** (*int*) – Embedding dimension.
- **depths** (*tuple(int)*) – Depth of Swin Transformer layers.
- **num_heads** (*tuple(int)*) – Number of attention heads in different layers.
- **window_size** (*int*) – Window size.
- **mlp_ratio** (*float*) – Ratio of mlp hidden dim to embedding dim.
- **qkv_bias** (*bool*) – If True, add a learnable bias to query, key, value. Default: Truee

- **qk_scale** (*float*) – Override default qk scale of head_dim ** -0.5 if set.
- **drop_rate** (*float*) – Dropout rate.
- **attn_drop_rate** (*float*) – Attention dropout rate.
- **drop_path_rate** (*float*) – Stochastic depth rate.
- **norm_layer** (*nn.Module*) – normalization layer.
- **ape** (*bool*) – If True, add absolute position embedding to the patch embedding.
- **patch_norm** (*bool*) – If True, add normalization after patch embedding.

__init__(*img_size=224, patch_size=4, in_chans=3, num_classes=1000, embed_dim=96, depths=[2, 2, 6, 2], num_heads=[3, 6, 12, 24], window_size=7, mlp_ratio=4.0, qkv_bias=True, qk_scale=None, drop_rate=0.0, attn_drop_rate=0.0, drop_path_rate=0.1, norm_layer=<class 'torch.nn.modules.normalization.LayerNorm'>, ape=False, patch_norm=True, use_dense_prediction=False, **kwargs*)

Initializes internal Module state, shared by both nn.Module and ScriptModule.

init_weights()

no_weight_decay()

no_weight_decay_keywords()

forward_features(*x*)

forward_feature_maps(*x*)

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.

---

forward_selfattention(*x*, *n=1*)

forward_last_selfattention(*x*)

forward_all_selfattention(*x*)

forward_return_n_last_blocks(*x*, *n=1*, *return_patch_avgpool=False*, *depth=[]*)

flops()

freeze_pretrained_layers(*frozen_layers=[]*)

training: bool

easycv.models.backbones.swin_transformer_dynamic.**dynamic_swin_tiny_p4_w7_224**(*pretrained=False*, ***kwargs*)

easycv.models.backbones.swin_transformer_dynamic.**dynamic_swin_small_p4_w7_224**(*pretrained=False*, ***kwargs*)

easycv.models.backbones.swin_transformer_dynamic.**dynamic_swin_base_p4_w7_224**(*pretrained=False*, ***kwargs*)

easycv.models.backbones.vit_transfomer_dynamic module

Mostly copy-paste from timm library. https://github.com/rwightman/pytorch-image-models/blob/master/timm/models/vision_transformer.py

dynamic Input support borrow from https://github.com/microsoft/esvit/blob/main/models/vision_transformer.py

easycv.models.backbones.vit_transfomer_dynamic.**drop_path**(*x*, *drop_prob: float = 0.0*, *training: bool = False*)

class easycv.models.backbones.vit_transfomer_dynamic.**DropPath**(*drop_prob=None*)

Bases: torch.nn.modules.module.Module

Drop paths (Stochastic Depth) per sample (when applied in main path of residual blocks).

__init__(*drop_prob=None*)

Initializes internal Module state, shared by both nn.Module and ScriptModule.

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.

---

training: bool

class easycv.models.backbones.vit_transfomer_dynamic.**Mlp**(*in_features*, *hidden_features=None*, *out_features=None*, *act_layer=<class 'torch.nn.modules.activation.GELU'>*, *drop=0.0*)

Bases: torch.nn.modules.module.Module

__init__(*in_features*, *hidden_features=None*, *out_features=None*, *act_layer=<class 'torch.nn.modules.activation.GELU'>*, *drop=0.0*)

Initializes internal Module state, shared by both nn.Module and ScriptModule.

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.

---

training: bool

class easycv.models.backbones.vit_transfomer_dynamic.**Attention**(*dim*, *num_heads=8*, *qkv_bias=False*, *qk_scale=None*, *attn_drop=0.0*, *proj_drop=0.0*)

Bases: torch.nn.modules.module.Module

__init__(*dim*, *num_heads=8*, *qkv_bias=False*, *qk_scale=None*, *attn_drop=0.0*, *proj_drop=0.0*)

Initializes internal Module state, shared by both nn.Module and ScriptModule.

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.

---

training: bool

class easycv.models.backbones.vit_transfomer_dynamic.**Block**(*dim*, *num_heads*, *mlp_ratio=4.0*, *qkv_bias=False*, *qk_scale=None*, *drop=0.0*, *attn_drop=0.0*, *drop_path=0.0*, *act_layer=<class 'torch.nn.modules.activation.GELU'>*, *norm_layer=<class 'torch.nn.modules.normalization.LayerNorm'>*)

Bases: `torch.nn.modules.module.Module`

__init__(*dim*, *num_heads*, *mlp_ratio=4.0*, *qkv_bias=False*, *qk_scale=None*, *drop=0.0*, *attn_drop=0.0*, *drop_path=0.0*, *act_layer=<class 'torch.nn.modules.activation.GELU'>*, *norm_layer=<class 'torch.nn.modules.normalization.LayerNorm'>*)

Initializes internal Module state, shared by both nn.Module and ScriptModule.

forward(*x*, *return_attention=False*)

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.

---

forward_fea_and_attn(*x*)

training: bool

class easycv.models.backbones.vit_transfomer_dynamic.**PatchEmbed**(*img_size=224*, *patch_size=16*, *in_chans=3*, *embed_dim=768*)

Bases: `torch.nn.modules.module.Module`

Image to Patch Embedding

__init__(*img_size=224*, *patch_size=16*, *in_chans=3*, *embed_dim=768*)

Initializes internal Module state, shared by both nn.Module and ScriptModule.

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.

---

training: bool

class easycv.models.backbones.vit_transfomer_dynamic.**VisionTransformer**(*img_size=[224], patch_size=16, in_chans=3, num_classes=0, embed_dim=768, depth=12, num_heads=12, mlp_ratio=4.0, qkv_bias=False, qk_scale=None, drop_rate=0.0, attn_drop_rate=0.0, drop_path_rate=0.0, norm_layer=<class 'torch.nn.modules.normalization.LayerNor use_dense_prediction=False, global_pool=False, **kwargs*)

Bases: torch.nn.modules.module.Module

Vision Transformer

__init__(*img_size=[224], patch_size=16, in_chans=3, num_classes=0, embed_dim=768, depth=12, num_heads=12, mlp_ratio=4.0, qkv_bias=False, qk_scale=None, drop_rate=0.0, attn_drop_rate=0.0, drop_path_rate=0.0, norm_layer=<class 'torch.nn.modules.normalization.LayerNorm'>, use_dense_prediction=False, global_pool=False, **kwargs*)

Initializes internal Module state, shared by both nn.Module and ScriptModule.

init_weights()

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.

---

forward_features(*x*)

forward_feature_maps(*x*)

interpolate_pos_encoding(*x*, *pos_embed*)

forward_selfattention(*x*, *n=1*)

forward_last_selfattention(*x*)

forward_all_selfattention(*x*)

forward_return_n_last_blocks(*x*, *n=1*, *return_patch_avgpool=False*, *depths=[]*)

training: bool

easycv.models.backbones.vit_transfomer_dynamic.**dynamic_deit_tiny_p16**(*patch_size=16*, ***kwargs*)

easycv.models.backbones.vit_transfomer_dynamic.**dynamic_deit_small_p16**(*patch_size=16*, ***kwargs*)

easycv.models.backbones.vit_transfomer_dynamic.**dynamic_vit_base_p16**(*patch_size=16*, ***kwargs*)

easycv.models.backbones.vit_transfomer_dynamic.**dynamic_vit_large_p16**(*patch_size=16*, ***kwargs*)

easycv.models.backbones.vit_transfomer_dynamic.**dynamic_vit_huge_p14**(*patch_size=14*, ***kwargs*)

easycv.models.backbones.xcit_transformer module

Implementation of Cross-Covariance Image Transformer (XCiT) Based on timm and DeiT code bases https://github.com/rwightman/pytorch-image-models/tree/master/timm https://github.com/facebookresearch/deit/

XCiT Transformer. Part of the code is borrowed from: https://github.com/facebookresearch/xcit/blob/master/xcit.py

class easycv.models.backbones.xcit_transformer.**PositionalEncodingFourier**(*hidden_dim=32*, *dim=768*, *temperature=10000*)

Bases: torch.nn.modules.module.Module

Positional encoding relying on a fourier kernel matching the one used in the "Attention is all of Need" paper. The implementation builds on DeTR code https://github.com/facebookresearch/detr/blob/master/models/position_encoding.py

__init__(*hidden_dim=32*, *dim=768*, *temperature=10000*)
Initializes internal Module state, shared by both nn.Module and ScriptModule.

forward(*B*, *H*, *W*)
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.

---

training: bool

easycv.models.backbones.xcit_transformer.**conv3x3**(*in_planes*, *out_planes*, *stride=1*)
3x3 convolution with padding

class easycv.models.backbones.xcit_transformer.**ConvPatchEmbed**(*img_size=224*, *patch_size=16*, *in_chans=3*, *embed_dim=768*)

Bases: torch.nn.modules.module.Module

Image to Patch Embedding using multiple convolutional layers

__init__(*img_size=224*, *patch_size=16*, *in_chans=3*, *embed_dim=768*)
Initializes internal Module state, shared by both nn.Module and ScriptModule.

forward(*x*, *padding_size=None*)
Defines the computation performed at every call.

Should be overridden by all subclasses.

Note: Although the recipe for forward pass needs to be defined within this function, one should call the Module instance afterwards instead of this since the former takes care of running the registered hooks

while the latter silently ignores them.

training: bool

class easycv.models.backbones.xcit_transformer.**LPI**(*in_features*, *hidden_features=None*, *out_features=None*, *act_layer=<class 'torch.nn.modules.activation.GELU'>*, *drop=0.0*, *kernel_size=3*)

Bases: torch.nn.modules.module.Module

Local Patch Interaction module that allows explicit communication between tokens in 3x3 windows to augment the implicit communcation performed by the block diagonal scatter attention. Implemented using 2 layers of separable 3x3 convolutions with GeLU and BatchNorm2d

__init__(*in_features*, *hidden_features=None*, *out_features=None*, *act_layer=<class 'torch.nn.modules.activation.GELU'>*, *drop=0.0*, *kernel_size=3*)

Initializes internal Module state, shared by both nn.Module and ScriptModule.

forward(*x*, *H*, *W*)

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.

training: bool

class easycv.models.backbones.xcit_transformer.**ClassAttention**(*dim*, *num_heads=8*, *qkv_bias=False*, *qk_scale=None*, *attn_drop=0.0*, *proj_drop=0.0*)

Bases: torch.nn.modules.module.Module

Class Attention Layer as in CaiT https://arxiv.org/abs/2103.17239

__init__(*dim*, *num_heads=8*, *qkv_bias=False*, *qk_scale=None*, *attn_drop=0.0*, *proj_drop=0.0*)

Initializes internal Module state, shared by both nn.Module and ScriptModule.

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.

training: bool

class easycv.models.backbones.xcit_transformer.**ClassAttentionBlock**(*dim*, *num_heads*, *mlp_ratio=4.0*, *qkv_bias=False*, *qk_scale=None*, *drop=0.0*, *attn_drop=0.0*, *drop_path=0.0*, *act_layer=<class 'torch.nn.modules.activation.GELU'>*, *norm_layer=<class 'torch.nn.modules.normalization.LayerNorm'>*, *eta=None*, *tokens_norm=False*)

Bases: torch.nn.modules.module.Module

Class Attention Layer as in CaiT https://arxiv.org/abs/2103.17239

__init__(*dim*, *num_heads*, *mlp_ratio=4.0*, *qkv_bias=False*, *qk_scale=None*, *drop=0.0*, *attn_drop=0.0*, *drop_path=0.0*, *act_layer=<class 'torch.nn.modules.activation.GELU'>*, *norm_layer=<class 'torch.nn.modules.normalization.LayerNorm'>*, *eta=None*, *tokens_norm=False*)

Initializes internal Module state, shared by both nn.Module and ScriptModule.

forward(*x*, *H*, *W*, *mask=None*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

Note: Although the recipe for forward pass needs to be defined within this function, one should call the Module instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

training: bool

class easycv.models.backbones.xcit_transformer.**XCA**(*dim*, *num_heads=8*, *qkv_bias=False*, *qk_scale=None*, *attn_drop=0.0*, *proj_drop=0.0*)

Bases: torch.nn.modules.module.Module

Cross-Covariance Attention (XCA) operation where the channels are updated using a weighted sum.

The weights are obtained from the (softmax normalized) Cross-covariance matrix (Q^T K in d_h times d_h)

__init__(*dim*, *num_heads=8*, *qkv_bias=False*, *qk_scale=None*, *attn_drop=0.0*, *proj_drop=0.0*)

Initializes internal Module state, shared by both nn.Module and ScriptModule.

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.

---

no_weight_decay()

training: bool

class easycv.models.backbones.xcit_transformer.**XCABlock**(*dim*, *num_heads*, *mlp_ratio=4.0*, *qkv_bias=False*, *qk_scale=None*, *drop=0.0*, *attn_drop=0.0*, *drop_path=0.0*, *act_layer=<class 'torch.nn.modules.activation.GELU'>*, *norm_layer=<class 'torch.nn.modules.normalization.LayerNorm'>*, *num_tokens=196*, *eta=None*)

Bases: torch.nn.modules.module.Module

__init__(*dim*, *num_heads*, *mlp_ratio=4.0*, *qkv_bias=False*, *qk_scale=None*, *drop=0.0*, *attn_drop=0.0*, *drop_path=0.0*, *act_layer=<class 'torch.nn.modules.activation.GELU'>*, *norm_layer=<class 'torch.nn.modules.normalization.LayerNorm'>*, *num_tokens=196*, *eta=None*)

Initializes internal Module state, shared by both nn.Module and ScriptModule.

forward(*x*, *H*, *W*)

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.

---

training: bool

class easycv.models.backbones.xcit_transformer.**XCiT**(*img_size=224*, *patch_size=16*, *in_chans=3*, *num_classes=1000*, *embed_dim=768*, *depth=12*, *num_heads=12*, *mlp_ratio=4.0*, *qkv_bias=True*, *qk_scale=None*, *drop_rate=0.0*, *attn_drop_rate=0.0*, *drop_path_rate=0.0*, *norm_layer=None*, *cls_attn_layers=2*, *use_pos=True*, *patch_proj='linear'*, *eta=None*, *tokens_norm=False*)

Bases: torch.nn.modules.module.Module

Based on timm and DeiT code bases https://github.com/rwightman/pytorch-image-models/tree/master/timm https://github.com/facebookresearch/deit/

__init__(*img_size=224*, *patch_size=16*, *in_chans=3*, *num_classes=1000*, *embed_dim=768*, *depth=12*, *num_heads=12*, *mlp_ratio=4.0*, *qkv_bias=True*, *qk_scale=None*, *drop_rate=0.0*, *attn_drop_rate=0.0*, *drop_path_rate=0.0*, *norm_layer=None*, *cls_attn_layers=2*, *use_pos=True*, *patch_proj='linear'*, *eta=None*, *tokens_norm=False*)

Parameters

- **img_size** (*int, tuple*) – input image size
- **patch_size** (*int, tuple*) – patch size
- **in_chans** (*int*) – number of input channels
- **num_classes** (*int*) – number of classes for classification head
- **embed_dim** (*int*) – embedding dimension
- **depth** (*int*) – depth of transformer
- **num_heads** (*int*) – number of attention heads

- **mlp_ratio** (*int*) – ratio of mlp hidden dim to embedding dim
- **qkv_bias** (*bool*) – enable bias for qkv if True
- **qk_scale** (*float*) – override default qk scale of head_dim ** -0.5 if set
- **drop_rate** (*float*) – dropout rate
- **attn_drop_rate** (*float*) – attention dropout rate
- **drop_path_rate** (*float*) – stochastic depth rate
- **norm_layer** – (nn.Module): normalization layer
- **cls_attn_layers** – (int) Depth of Class attention layers
- **use_pos** – (bool) whether to use positional encoding
- **eta** – (float) layerscale initialization value
- **tokens_norm** – (bool) Whether to normalize all tokens or just the cls_token in the CA

init_weights()

no_weight_decay()

forward_features(*x*)

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.

---

training: bool

easycv.models.backbones.xcit_transformer.**xcit_small_12_p16**(*pretrained=False*, ***kwargs*)

easycv.models.backbones.xcit_transformer.**xcit_small_24_p16**(*pretrained=False*, ***kwargs*)

easycv.models.backbones.xcit_transformer.**xcit_medium_24_p16**(*pretrained=False*, ***kwargs*)

easycv.models.backbones.xcit_transformer.**xcit_small_12_p8**(*pretrained=False*, ***kwargs*)

easycv.models.backbones.xcit_transformer.**xcit_small_24_p8**(*pretrained=False*, ***kwargs*)

easycv.models.backbones.xcit_transformer.**xcit_medium_24_p8**(*pretrained=False*, ***kwargs*)

easycv.models.backbones.xcit_transformer.**xcit_large_24_p8**(*pretrained=False*, ***kwargs*)

18.1.2 easycv.models.classification package

Submodules

easycv.models.classification.classification module

class easycv.models.classification.classification.**Classification**(*backbone*, *train_preprocess=[]*, *with_sobel=False*, *head=None*, *neck=None*, *pretrained=True*, *mixup_cfg=None*)

Bases: *easycv.models.base.BaseModel*

Parameters

- **pretrained** – Select one {str or True or False/None}.
- **pretrained == str** (*if*) –
- **model from specified path;** (*load*) –
- **pretrained == True** (*if*) –
- **model from default path** (*load*) –
- **pretrained == False or None** (*if*) –
- **from init weights.** (*load*) –

__init__(*backbone*, *train_preprocess=[]*, *with_sobel=False*, *head=None*, *neck=None*, *pretrained=True*, *mixup_cfg=None*)

Initializes internal Module state, shared by both nn.Module and ScriptModule.

init_weights()

forward_backbone(*img: torch.Tensor*) → List[torch.Tensor]

Forward backbone

Returns backbone outputs

Return type x (tuple)

forward_train(*img*, *gt_labels*) → Dict[str, torch.Tensor]

In forward train, model will forward backbone + neck / multi-neck, get alist of output tensor, and put this list to head / multi-head, to compute each loss

forward_test(*img: torch.Tensor*) → Dict[str, torch.Tensor]

forward_test means generate prob/class from image only support one neck + one head

forward_test_label(*img*, *gt_labels*) → Dict[str, torch.Tensor]

forward_test_label means generate prob/class from image only support one neck + one head ps : head init need set the input feature idx

aug_test(*imgs*)

training: bool

forward_feature(*img*) → Dict[str, torch.Tensor]

Forward feature means forward backbone + neck/multineck ,get dict of output feature,

self.neck_num = 0: means only forward backbone, output backbone feature with avgpool, with key neck, self.neck_num > 0: means has 1/multi neck, output neck's feature with key neck_neckidx_featureidx, suck as neck_0_0

Returns feature tensor

Return type x (torch.Tensor)

update_extract_list(*key*)

forward(*img: torch.Tensor*, *mode: str = 'train'*, *gt_labels: Optional[torch.Tensor] = None*, *img_metas: Optional[torch.Tensor] = None*) → Dict[str, 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.

---

easycv.models.classification.necks module

class easycv.models.classification.necks.**LinearNeck**(*in_channels*, *out_channels*, *with_avg_pool=True*, *with_norm=False*)

Bases: `torch.nn.modules.module.Module`

Linear neck: fc only

__init__(*in_channels*, *out_channels*, *with_avg_pool=True*, *with_norm=False*)

Initializes internal Module state, shared by both nn.Module and ScriptModule.

init_weights(*init_linear='normal'*)

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.

---

training: bool

class easycv.models.classification.necks.**RetrivalNeck**(*in_channels*, *out_channels*, *with_avg_pool=True*, *cdg_config=['G', 'M']*)

Bases: `torch.nn.modules.module.Module`

RetrivalNeck: refer, Combination of Multiple Global Descriptors for Image Retrieval https://arxiv.org/pdf/1903.10663.pdf

CGD feature : only use avg pool + gem pooling + max pooling, by pool -> fc -> norm -> concat -> norm Avg feature : use avg pooling, avg pool -> syncbn -> fc

len(cgd_config) > 0: return [CGD, Avg] len(cgd_config) = 0 : return [Avg]

__init__(*in_channels*, *out_channels*, *with_avg_pool=True*, *cdg_config=['G', 'M']*)

Init RetrivalNeck, faceid neck doesn't pool for input feature map, doesn't support dynamic input

Parameters

- **in_channels** – Int - input feature map channels
- **out_channels** – Int - output feature map channels
- **with_avg_pool** – bool do avg pool for BNneck or not

- **cdg_config** – list('G','M','S'), to configure output feature, CGD = [gempooling] + [maxpooling] + [meanpooling], if len(cgd_config) > 0: return [CGD, Avg] if len(cgd_config) = 0 : return [Avg]

init_weights(*init_linear='normal'*)

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.

---

training: bool

class easycv.models.classification.necks.**FaceIDNeck**(*in_channels*, *out_channels*, *map_shape=1*, *dropout_ratio=0.4*, *with_norm=False*, *bn_type='SyncBN'*)

Bases: `torch.nn.modules.module.Module`

FaceID neck: Include BN, dropout, flatten, linear, bn

__init__(*in_channels*, *out_channels*, *map_shape=1*, *dropout_ratio=0.4*, *with_norm=False*, *bn_type='SyncBN'*)

Init FaceIDNeck, faceid neck doesn't pool for input feature map, doesn't support dynamic input

Parameters

- **in_channels** – Int - input feature map channels
- **out_channels** – Int - output feature map channels
- **map_shape** – Int or list(int,...), input feature map (w,h) or w when w=h,
- **dropout_ratio** – float, drop out ratio
- **with_norm** – normalize output feature or not
- **bn_type** – SyncBN or BN

init_weights(*init_linear='normal'*)

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.

---

training: bool

class easycv.models.classification.necks.**MultiLinearNeck**(*in_channels*, *out_channels*, *num_layers=1*, *with_avg_pool=True*)

Bases: `torch.nn.modules.module.Module`

MultiLinearNeck neck: MultiFc head

__init__(*in_channels*, *out_channels*, *num_layers=1*, *with_avg_pool=True*)

Parameters

- **in_channels** – int or list[int]
- **out_channels** – int or list[int]
- **num_layers** – total fc num
- **with_avg_pool** – input will be avgPool if True

Returns None

Raises

- **len(in_channel) != len(out_channels)** –
- **len(in_channel) != len(num_layers)** –

init_weights(*init_linear='normal'*)

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.

---

training: bool

class easycv.models.classification.necks.**HRFuseScales**(*in_channels*, *out_channels=2048*, *norm_cfg={'momentum': 0.1, 'type': 'BN'}*)

Bases: `torch.nn.modules.module.Module`

Fuse feature map of multiple scales in HRNet. :param in_channels: The input channels of all scales. :type in_channels: list[int] :param out_channels: The channels of fused feature map.
Defaults to 2048.

Parameters

- **norm_cfg** (*dict*) – dictionary to construct norm layers. Defaults to `dict(type='BN', momentum=0.1)`.
- **init_cfg** (*dict | list[dict], optional*) – Initialization config dict. Defaults to `dict(type='Normal', layer='Linear', std=0.01))`.

__init__(*in_channels*, *out_channels=2048*, *norm_cfg={'momentum': 0.1, 'type': 'BN'}*)

Initializes internal Module state, shared by both nn.Module and ScriptModule.

training: bool

init_weights(*init_linear='normal'*)

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.

18.1.3 easycv.models.detection package

Subpackages

easycv.models.detection.utils package

Submodules

easycv.models.detection.utils.boxes module

easycv.models.detection.utils.boxes.**bboxes_iou**(*bboxes_a*, *bboxes_b*, *xyxy=True*)

easycv.models.detection.utils.boxes.**postprocess**(*prediction*, *num_classes*, *conf_thre=0.7*, *nms_thre=0.45*)

easycv.models.detection.yolox package

Submodules

easycv.models.detection.yolox.yolo_head module

class easycv.models.detection.yolox.yolo_head.**YOLOXHead**(*num_classes*, *width=1.0*, *strides=[8, 16, 32]*, *in_channels=[256, 512, 1024]*, *act='silu'*, *depthwise=False*, *stage='CLOUD'*)

Bases: `torch.nn.modules.module.Module`

__init__(*num_classes*, *width=1.0*, *strides=[8, 16, 32]*, *in_channels=[256, 512, 1024]*, *act='silu'*, *depthwise=False*, *stage='CLOUD'*)

Parameters

- **num_classes** (*int*) – detection class numbers.
- **width** (*float*) – model width. Default value: 1.0.
- **strides** (*list*) – expanded strides. Default value: [8, 16, 32].
- **in_channels** (*list*) – model conv channels set. Default value: [256, 512, 1024].
- **act** (*str*) – activation type of conv. Defalut value: “silu”.
- **depthwise** (*bool*) – whether apply depthwise conv in conv branch. Default value: False.
- **stage** (*str*) – model stage, distinguish edge head to cloud head. Default value: CLOUD.

initialize_biases(*prior_prob*)

forward(*xin*, *labels=None*, *imgs=None*)
Defines the computation performed at every call.

Should be overridden by all subclasses.

Note: Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

get_output_and_grid(*output*, *k*, *stride*, *dtype*)

decode_outputs(*outputs*, *dtype*)

get_losses(*imgs*, *x_shifts*, *y_shifts*, *expanded_strides*, *labels*, *outputs*, *origin_preds*, *dtype*)

get_l1_target(*l1_target*, *gt*, *stride*, *x_shifts*, *y_shifts*, *eps=1e-08*)

get_assignments(*batch_idx*, *num_gt*, *total_num_anchors*, *gt_bboxes_per_image*, *gt_classes*, *bboxes_preds_per_image*, *expanded_strides*, *x_shifts*, *y_shifts*, *cls_preds*, *bbox_preds*, *obj_preds*, *labels*, *imgs*, *mode='gpu'*)

get_in_boxes_info(*gt_bboxes_per_image*, *expanded_strides*, *x_shifts*, *y_shifts*, *total_num_anchors*, *num_gt*)

dynamic_k_matching(*cost*, *pair_wise_ious*, *gt_classes*, *num_gt*, *fg_mask*)

training: bool

easycv.models.detection.yolox.yolo_pafpn module

class easycv.models.detection.yolox.yolo_pafpn.**YOLOPAFPN**(*depth=1.0*, *width=1.0*, *in_features=('dark3', 'dark4', 'dark5')*, *in_channels=[256, 512, 1024]*, *depthwise=False*, *act='silu'*)

Bases: `torch.nn.modules.module.Module`

YOLOv3 model. Darknet 53 is the default backbone of this model.

__init__(*depth=1.0*, *width=1.0*, *in_features=('dark3', 'dark4', 'dark5')*, *in_channels=[256, 512, 1024]*, *depthwise=False*, *act='silu'*)
Initializes internal Module state, shared by both nn.Module and ScriptModule.

forward(*input*)

Parameters **inputs** – input images.

Returns FPN feature.

Return type Tuple[Tensor]

training: bool

easycv.models.detection.yolox.yolox module

easycv.models.detection.yolox.yolox.**init_yolo**(*M*)

class easycv.models.detection.yolox.yolox.**YOLOX**(*model_type: str = 's'*, *num_classes: int = 80*, *test_size: tuple = (640, 640)*, *test_conf: float = 0.01*, *nms_thre: float = 0.65*, *pretrained: Optional[str] = None*)

Bases: *easycv.models.base.BaseModel*

YOLOX model module. The module list is defined by create_yolov3_modules function. The network returns loss values from three YOLO layers during training and detection results during test.

param_map = {'l': [1.0, 1.0], 'm': [0.67, 0.75], 'nano': [0.33, 0.25], 's': [0.33, 0.5], 'tiny': [0.33, 0.375], 'x': [1.33, 1.25]}

__init__(*model_type: str = 's'*, *num_classes: int = 80*, *test_size: tuple = (640, 640)*, *test_conf: float = 0.01*, *nms_thre: float = 0.65*, *pretrained: Optional[str] = None*)

Initializes internal Module state, shared by both nn.Module and ScriptModule.

forward_train(*img: torch.Tensor*, *gt_bboxes: torch.Tensor*, *gt_labels: torch.Tensor*, *img_metas=None*, *scale=None*) → Dict[str, torch.Tensor]

Abstract interface for model forward in training

Parameters

- **img** (*Tensor*) – image tensor, NxCxHxW
- **target** (*List[Tensor]*) – list of target tensor, NTx5 [class,x_c,y_c,w,h]

forward_test(*img: torch.Tensor*, *img_metas=None*) → torch.Tensor

Abstract interface for model forward in training

Parameters

- **img** (*Tensor*) – image tensor, NxCxHxW
- **target** (*List[Tensor]*) – list of target tensor, NTx5 [class,x_c,y_c,w,h]

forward(*img*, *mode='compression'*, ***kwargs*)

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.

---

forward_compression(*x*)

forward_export(*img*)

training: bool

easycv.models.detection.yolox_edge package

Submodules

easycv.models.detection.yolox_edge.yolox_edge module

easycv.models.detection.yolox_edge.yolox_edge.**init_yolo**(*M*)

class easycv.models.detection.yolox_edge.yolox_edge.**YOLOX_EDGE**(*stage: str = 'EDGE'*, *model_type: str = 's'*, *num_classes: int = 80*, *test_size: tuple = (640, 640)*, *test_conf: float = 0.01*, *nms_thre: float = 0.65*, *pretrained: Optional[str] = None*, *depth: float = 1.0*, *width: float = 1.0*, *max_model_params: float = 0.0*, *max_model_flops: float = 0.0*, *activation: str = 'silu'*, *in_channels: list = [256, 512, 1024]*, *backbone=None*, *head=None*)

Bases: *easycv.models.detection.yolox.yolox.YOLOX*

YOLOX model module. The module list is defined by create_yolov3_modules function. The network returns loss values from three YOLO layers during training and detection results during test.

__init__(*stage: str = 'EDGE'*, *model_type: str = 's'*, *num_classes: int = 80*, *test_size: tuple = (640, 640)*, *test_conf: float = 0.01*, *nms_thre: float = 0.65*, *pretrained: Optional[str] = None*, *depth: float = 1.0*, *width: float = 1.0*, *max_model_params: float = 0.0*, *max_model_flops: float = 0.0*, *activation: str = 'silu'*, *in_channels: list = [256, 512, 1024]*, *backbone=None*, *head=None*)

Initializes internal Module state, shared by both nn.Module and ScriptModule.

training: bool

18.1.4 easycv.models.heads package

Submodules

easycv.models.heads.cls_head module

class easycv.models.heads.cls_head.**ClsHead**(*with_avg_pool=False*, *label_smooth=0.0*, *in_channels=2048*, *with_fc=True*, *num_classes=1000*, *loss_config={'type': 'CrossEntropyLossWithLabelSmooth'}*, *input_feature_index=[0]*)

Bases: torch.nn.modules.module.Module

Simplest classifier head, with only one fc layer. Should Notice Evtorch module design input always be feature_list = [tensor, tensor,...]

__init__(*with_avg_pool=False*, *label_smooth=0.0*, *in_channels=2048*, *with_fc=True*, *num_classes=1000*, *loss_config={'type': 'CrossEntropyLossWithLabelSmooth'}*, *input_feature_index=[0]*)

Initializes internal Module state, shared by both nn.Module and ScriptModule.

init_weights(*pretrained=None*, *init_linear='normal'*, *std=0.01*, *bias=0.0*)

forward(*x: List[torch.Tensor]*) → List[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.

---

loss(*cls_score: List[torch.Tensor]*, *labels: torch.Tensor*) → Dict[str, torch.Tensor]

Parameters

- **cls_score** – [N x num_classes]
- **labels** – if don't use mixup, shape is [N],else [N x num_classes]

mixup_loss(*cls_score*, *labels_1*, *labels_2*, *lam*) → Dict[str, torch.Tensor]

training: bool

easycv.models.heads.contrastive_head module

class easycv.models.heads.contrastive_head.**ContrastiveHead**(*temperature=0.1*)
Bases: torch.nn.modules.module.Module

Head for contrastive learning.

__init__(*temperature=0.1*)
Initializes internal Module state, shared by both nn.Module and ScriptModule.

forward(*pos*, *neg*)

Parameters

- **pos** (*Tensor*) – Nx1 positive similarity
- **neg** (*Tensor*) – Nxk negative similarity

training: bool

class easycv.models.heads.contrastive_head.**DebiasedContrastiveHead**(*temperature=0.1*, *tau=0.1*)
Bases: torch.nn.modules.module.Module

__init__(*temperature=0.1*, *tau=0.1*)
Initializes internal Module state, shared by both nn.Module and ScriptModule.

forward(*pos*, *neg*)

Parameters

- **pos** (*Tensor*) – Nx1 positive similarity
- **neg** (*Tensor*) – Nxk negative similarity

training: bool

easycv.models.heads.latent_pred_head module

class easycv.models.heads.latent_pred_head.**LatentPredictHead**(*predictor*, *size_average=True*)
Bases: torch.nn.modules.module.Module

Head for contrastive learning.

__init__(*predictor*, *size_average=True*)
Initializes internal Module state, shared by both nn.Module and ScriptModule.

init_weights(*init_linear='normal'*)

forward(*input*, *target*)

Parameters

- **input** (*Tensor*) – NxC input features.
- **target** (*Tensor*) – NxC target features.

training: bool

class easycv.models.heads.latent_pred_head.**LatentClsHead**(*predictor*)
Bases: torch.nn.modules.module.Module

Head for contrastive learning.

__init__(*predictor*)
Initializes internal Module state, shared by both nn.Module and ScriptModule.

init_weights(*init_linear='normal'*)

forward(*input*, *target*)

Parameters

- **input** (*Tensor*) – NxC input features.
- **target** (*Tensor*) – NxC target features.

training: bool

easycv.models.heads.mp_metric_head module

easycv.models.heads.mp_metric_head.**EmbeddingExplansion**(*embs*, *labels*, *explanion_rate=4*, *alpha=1.0*)
Expand embedding: CVPR refer to https://github.com/clovaai/embedding-expansion combine PK sampled data, mixup anchor positive pair to generate more features, always combine with BatchHardminer. result on SOP and CUB need to be add

Parameters

- **embs** – [N , dims] tensor
- **labels** – [N] tensor
- **explanion_rate** – to expand N to explanion_rate * N
- **alpha** – beta distribution parameter for mixup

Returns [N*explanion_rate , dims]

Return type embs

class easycv.models.heads.mp_metric_head.**MpMetrixHead**(*with_avg_pool=False*, *in_channels=2048*, *loss_config=[{'type': 'CircleLoss', 'loss_weight': 1.0, 'norm': True, 'ddp': True, 'm': 0.4, 'gamma': 80}]*, *input_feature_index=[0]*, *input_label_index=0*, *ignore_label=None*)

Bases: torch.nn.modules.module.Module

Simplest classifier head, with only one fc layer.

__init__(*with_avg_pool=False*, *in_channels=2048*, *loss_config=[{'type': 'CircleLoss', 'loss_weight': 1.0, 'norm': True, 'ddp': True, 'm': 0.4, 'gamma': 80}]*, *input_feature_index=[0]*, *input_label_index=0*, *ignore_label=None*)

Initializes internal Module state, shared by both nn.Module and ScriptModule.

init_weights(*pretrained=None*, *init_linear='normal'*, *std=0.01*, *bias=0.0*)

forward(*x: List[torch.Tensor]*) → List[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.

---

loss(*cls_score*, *labels*) → Dict[str, torch.Tensor]

training: bool

easycv.models.heads.multi_cls_head module

class easycv.models.heads.multi_cls_head.**MultiClsHead**(*pool_type='adaptive'*, *in_indices=(0)*, *with_last_layer_unpool=False*, *backbone='resnet50'*, *norm_cfg={'type': 'BN'}*, *num_classes=1000*)

Bases: torch.nn.modules.module.Module

Multiple classifier heads.

FEAT_CHANNELS = {'resnet50': [64, 256, 512, 1024, 2048]}

FEAT_LAST_UNPOOL = {'resnet50': 100352}

__init__(*pool_type='adaptive'*, *in_indices=(0)*, *with_last_layer_unpool=False*, *backbone='resnet50'*, *norm_cfg={'type': 'BN'}*, *num_classes=1000*)

Initializes internal Module state, shared by both nn.Module and ScriptModule.

init_weights()

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.

loss(*cls_score*, *labels*)

training: bool

18.1.5 easycv.models.loss package

Submodules

easycv.models.loss.iou_loss module

class easycv.models.loss.iou_loss.**IOUloss**(*reduction='none'*, *loss_type='iou'*)
Bases: torch.nn.modules.module.Module

__init__(*reduction='none'*, *loss_type='iou'*)
Initializes internal Module state, shared by both nn.Module and ScriptModule.

forward(*pred*, *target*)
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.

training: bool

easycv.models.loss.mse_loss module

class easycv.models.loss.mse_loss.**JointsMSELoss**(*use_target_weight=False*, *loss_weight=1.0*)
Bases: torch.nn.modules.module.Module

MSE loss for heatmaps.
Parameters

- **use_target_weight** (*bool*) – Option to use weighted MSE loss. Different joint types may have different target weights.
- **loss_weight** (*float*) – Weight of the loss. Default: 1.0.

__init__(*use_target_weight=False*, *loss_weight=1.0*)
Initializes internal Module state, shared by both nn.Module and ScriptModule.

forward(*output*, *target*, *target_weight*)
Forward function.

training: bool

easycv.models.loss.pytorch_metric_learning module

class easycv.models.loss.pytorch_metric_learning.**FocalLoss2d**(*gamma=2*, *weight=None*, *size_average=None*, *reduce=None*, *reduction='mean'*, *num_classes=2*)

Bases: torch.nn.modules.loss._WeightedLoss

__init__(*gamma=2*, *weight=None*, *size_average=None*, *reduce=None*, *reduction='mean'*, *num_classes=2*)

FocalLoss2d, loss solve 2-class classification unbalance problem

Parameters

- **gamma** – focal loss param Gamma
- **weight** – weight same as loss._WeightedLoss
- **size_average** – size_average same as loss._WeightedLoss
- **reduce** – reduce same as loss._WeightedLoss
- **reduction** – reduce same as loss._WeightedLoss
- **num_classes** – fix num 2

Returns Focalloss nn.module.loss object

forward(*input*, *target*)

input: [N * num_classes] target : [N * num_classes] one-hot

reduction: str

class easycv.models.loss.pytorch_metric_learning.**DistributeMSELoss**

Bases: torch.nn.modules.module.Module

__init__()

DistributeMSELoss : for faceid age, score predict (regression by softmax)

forward(*input*, *target*)

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.

---

training: bool

class easycv.models.loss.pytorch_metric_learning.**CrossEntropyLossWithLabelSmooth**(*label_smooth=0.1*, *temperature=1.0*, *with_cls=False*, *embedding_size=512*, *num_classes=10000*)

Bases: torch.nn.modules.module.Module

__init__(*label_smooth=0.1*, *temperature=1.0*, *with_cls=False*, *embedding_size=512*, *num_classes=10000*)

A softmax loss , with label_smooth and fc(to fit pytorch metric learning interface) :param label_smooth: label_smooth args, default=0.1 :param with_cls: if True, will generate a nn.Linear to trans input embedding from embedding_size to num_classes :param embedding_size: if input is feature not logits, then need this

to indicate embedding shape :param num_classes: if input is feature not logits, then need this to indicate classification num_classes

Returns None

Raises **IOError** – An error occurred accessing the bigtable.Table object.

forward(*input*, *target*)

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.

---

training: bool

class easycv.models.loss.pytorch_metric_learning.**AMSoftmaxLoss**(*embedding_size=512*, *num_classes=100000*, *margin=0.35*, *scale=30*)

Bases: `torch.nn.modules.module.Module`

__init__(*embedding_size=512*, *num_classes=100000*, *margin=0.35*, *scale=30*)

AMsoftmax loss , with fc(to fit pytorch metric learning interface), paper: https://arxiv.org/pdf/1801.05599.pdf :param embedding_size: forward input [N, embedding_size] :param num_classes: classification num_classes :param margin: AMSoftmax param :param scale: AMSoftmax param, should increase num_classes

forward(*x*, *lb*)

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.

---

training: bool

class easycv.models.loss.pytorch_metric_learning.**ModelParallelSoftmaxLoss**(*embedding_size=512*, *num_classes=100000*, *scale=None*, *margin=None*, *bias=True*)

Bases: `torch.nn.modules.module.Module`

__init__(*embedding_size=512*, *num_classes=100000*, *scale=None*, *margin=None*, *bias=True*)

ModelParallel Softmax by sailfish :param embedding_size: forward input [N, embedding_size] :param num_classes: classification num_classes

forward(*x*, *lb*)

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.

training: bool

class easycv.models.loss.pytorch_metric_learning.**ModelParallelAMSoftmaxLoss**(*embedding_size=512*, *num_classes=100000*, *margin=0.35*, *scale=30*)

Bases: torch.nn.modules.module.Module

__init__(*embedding_size=512*, *num_classes=100000*, *margin=0.35*, *scale=30*)

ModelParallel AMSoftmax by sailfish :param embedding_size: forward input [N, embedding_size] :param num_classes: classification num_classes

forward(*x*, *lb*)

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.

training: bool

class easycv.models.loss.pytorch_metric_learning.**SoftTargetCrossEntropy**(*num_classes=1000*, ***kwargs*)

Bases: torch.nn.modules.module.Module

__init__(*num_classes=1000*, ***kwargs*)

Initializes internal Module state, shared by both nn.Module and ScriptModule.

forward(*x: torch.Tensor*, *target: 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.

training: bool

18.1.6 easycv.models.pose package

Subpackages

easycv.models.pose.heads package

Submodules

easycv.models.pose.heads.topdown_heatmap_base_head module

class easycv.models.pose.heads.topdown_heatmap_base_head.**TopdownHeatmapBaseHead**
Bases: torch.nn.modules.module.Module

Base class for top-down heatmap heads.

All top-down heatmap heads should subclass it. All subclass should overwrite:

Methods:*get_loss*, supporting to calculate loss. Methods:*get_accuracy*, supporting to calculate accuracy. Methods:*forward*, supporting to forward model. Methods:*inference_model*, supporting to inference model.

abstract get_loss(***kwargs*)
Gets the loss.

abstract get_accuracy(***kwargs*)
Gets the accuracy.

abstract forward(***kwargs*)
Forward function.

abstract inference_model(***kwargs*)
Inference function.

decode(*img_metas*, *output*, ***kwargs*)
Decode keypoints from heatmaps.

Parameters

- **img_metas** (*list(dict)*) – Information about data augmentation By default this includes: - "image_file: path to the image file - "center": center of the bbox - "scale": scale of the bbox - "rotation": rotation of the bbox - "bbox_score": score of bbox
- **output** (*np.ndarray[N, K, H, W]*) – model predicted heatmaps.

training: bool

easycv.models.pose.heads.topdown_heatmap_simple_head module

class easycv.models.pose.heads.topdown_heatmap_simple_head.**TopdownHeatmapSimpleHead**(*in_channels*, *out_channels*, *num_deconv_layers=3*, *num_deconv_filters=(256, 256, 256)*, *num_deconv_kernels=(4, 4, 4)*, *extra=None*, *in_index=0*, *input_transform=None*, *align_corners=False*, *loss_keypoint=None*, *train_cfg=None*, *test_cfg=None*)
Bases: *easycv.models.pose.heads.topdown_heatmap_base_head.TopdownHeatmapBaseHead*

Top-down heatmap simple head. paper ref: Bin Xiao et al. Simple Baselines for Human Pose Estimation and Tracking.

TopdownHeatmapSimpleHead is consisted of (>=0) number of deconv layers and a simple conv2d layer.

Parameters

- **in_channels** (*int*) – Number of input channels
- **out_channels** (*int*) – Number of output channels
- **num_deconv_layers** (*int*) – Number of deconv layers. num_deconv_layers should >= 0. Note that 0 means no deconv layers.
- **num_deconv_filters** (*list|tuple*) – Number of filters. If num_deconv_layers > 0, the length of
- **num_deconv_kernels** (*list|tuple*) – Kernel sizes.
- **in_index** (*int|Sequence[int]*) – Input feature index. Default: 0
- **input_transform** (*str|None*) – Transformation type of input features. Options: 'resize_concat', 'multiple_select', None. Default: None.
  - **'resize_concat': Multiple feature maps will be resized to the** same size as the first one and then concat together. Usually used in FCN head of HRNet.
  - **'multiple_select': Multiple feature maps will be bundle into** a list and passed into decode head.
  - None: Only one select feature map is allowed.
- **align_corners** (*bool*) – align_corners argument of F.interpolate. Default: False.
- **loss_keypoint** (*dict*) – Config for keypoint loss. Default: None.

__init__(*in_channels*, *out_channels*, *num_deconv_layers=3*, *num_deconv_filters=(256, 256, 256)*, *num_deconv_kernels=(4, 4, 4)*, *extra=None*, *in_index=0*, *input_transform=None*, *align_corners=False*, *loss_keypoint=None*, *train_cfg=None*, *test_cfg=None*)

Initializes internal Module state, shared by both nn.Module and ScriptModule.

get_loss(*output*, *target*, *target_weight*)

Calculate top-down keypoint loss.

Note: batch_size: N num_keypoints: K heatmaps height: H heatmaps weight: W

Parameters

- **output** (*torch.Tensor[NxKxHxW]*) – Output heatmaps.
- **target** (*torch.Tensor[NxKxHxW]*) – Target heatmaps.
- **target_weight** (*torch.Tensor[NxKx1]*) – Weights across different joint types.

get_accuracy(*output*, *target*, *target_weight*)

Calculate accuracy for top-down keypoint loss.

Note: batch_size: N num_keypoints: K heatmaps height: H heatmaps weight: W

Parameters

- **output** (*torch.Tensor[NxKxHxW]*) – Output heatmaps.

- **target** (*torch.Tensor[NxKxHxW]*) – Target heatmaps.
- **target_weight** (*torch.Tensor[NxKx1]*) – Weights across different joint types.

forward(*x*)
Forward function.

inference_model(*x*, *flip_pairs=None*)
Inference function.

Returns Output heatmaps.

Return type output_heatmap (np.ndarray)

Parameters

- **x** (*torch.Tensor[NxKxHxW]*) – Input features.
- **flip_pairs** (*None | list[tuple()]*) – Pairs of keypoints which are mirrored.

training: bool

init_weights()
Initialize model weights.

Submodules

easycv.models.pose.top_down module

class easycv.models.pose.top_down.**TopDown**(*backbone*, *neck=None*, *keypoint_head=None*, *train_cfg=None*, *test_cfg=None*, *pretrained=None*, *loss_pose=None*)

Bases: *easycv.models.base.BaseModel*

Top-down pose detectors.

Parameters

- **backbone** (*dict*) – Backbone modules to extract feature.
- **keypoint_head** (*dict*) – Keypoint head to process feature.
- **train_cfg** (*dict*) – Config for training. Default: None.
- **test_cfg** (*dict*) – Config for testing. Default: None.
- **pretrained** (*str*) – Path to the pretrained models.
- **loss_pose** (*None*) – Deprecated arguments. Please use *loss_keypoint* for heads instead.

__init__(*backbone*, *neck=None*, *keypoint_head=None*, *train_cfg=None*, *test_cfg=None*, *pretrained=None*, *loss_pose=None*)
Initializes internal Module state, shared by both nn.Module and ScriptModule.

property with_neck
Check if has keypoint_head.

property with_keypoint
Check if has keypoint_head.

init_weights()
Weight initialization for model.

forward_train(*img*, *target*, *target_weight*, *img_metas*, ***kwargs*)
Defines the computation performed at every call when training.

forward_test(*img*, *img_metas*, *return_heatmap=False*, ***kwargs*)
Defines the computation performed at every call when testing.

show_result(*img*, *result*, *skeleton=None*, *kpt_score_thr=0.3*, *bbox_color='green'*, *pose_kpt_color=None*, *pose_link_color=None*, *text_color='white'*, *radius=4*, *thickness=1*, *font_scale=0.5*, *bbox_thickness=1*, *win_name=''*, *show=False*, *show_keypoint_weight=False*, *wait_time=0*, *out_file=None*)
Draw *result* over *img*.

Parameters

- **img** (*str or Tensor*) – The image to be displayed.
- **result** (*list[dict]*) – The results to draw over *img* (bbox_result, pose_result).
- **skeleton** (*list[list]*) – The connection of keypoints. skeleton is 0-based indexing.
- **kpt_score_thr** (*float, optional*) – Minimum score of keypoints to be shown. Default: 0.3.
- **bbox_color** (str or tuple or Color) – Color of bbox lines.
- **pose_kpt_color** (*np.array[Nx3]`*) – Color of N keypoints. If None, do not draw keypoints.
- **pose_link_color** (*np.array[Mx3]*) – Color of M links. If None, do not draw links.
- **text_color** (str or tuple or Color) – Color of texts.
- **radius** (*int*) – Radius of circles.
- **thickness** (*int*) – Thickness of lines.
- **font_scale** (*float*) – Font scales of texts.
- **win_name** (*str*) – The window name.
- **show** (*bool*) – Whether to show the image. Default: False.
- **show_keypoint_weight** (*bool*) – Whether to change the transparency using the predicted confidence scores of keypoints.
- **wait_time** (*int*) – Value of waitKey param. Default: 0.
- **out_file** (*str or None*) – The filename to write the image. Default: None.

Returns Visualized img, only if not *show* or *out_file*.

Return type Tensor

training: bool

18.1.7 easycv.models.selfsup package

Submodules

easycv.models.selfsup.byol module

class easycv.models.selfsup.byol.**BYOL**(*backbone*, *neck=None*, *head=None*, *pretrained=None*, *base_momentum=0.996*, ***kwargs*)

Bases: *easycv.models.base.BaseModel*

BYOL unofficial implementation. Paper: https://arxiv.org/abs/2006.07733

__init__(*backbone*, *neck=None*, *head=None*, *pretrained=None*, *base_momentum=0.996*, ***kwargs*)

Initializes internal Module state, shared by both nn.Module and ScriptModule.

init_weights()

forward_train(*img*, ***kwargs*)

Abstract interface for model forward in training

Parameters

- **img** (*Tensor*) – image tensor
- **kwargs** (*keyword arguments*) – Specific to concrete implementation.

forward_test(*img*, ***kwargs*)

Abstract interface for model forward in testing

Parameters

- **img** (*Tensor*) – image tensor
- **kwargs** (*keyword arguments*) – Specific to concrete implementation.

forward(*img*, *mode='train'*, ***kwargs*)

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.

---

training: bool

easycv.models.selfsup.dino module

class easycv.models.selfsup.dino.**MultiCropWrapper**(*backbone*, *head*)

Bases: `torch.nn.modules.module.Module`

Perform forward pass separately on each resolution input. The inputs corresponding to a single resolution are clubbed and single forward is run on the same resolution inputs. Hence we do several forward passes = number of different resolutions used. We then concatenate all the output features and run the head forward on these concatenated features.

__init__(*backbone*, *head*)

Initializes internal Module state, shared by both nn.Module and ScriptModule.

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.

---

training: bool

class easycv.models.selfsup.dino.**DINOLoss**(*out_dim*, *ncrops*, *warmup_teacher_temp*, *teacher_temp*, *warmup_teacher_temp_epochs*, *nepochs*, *device*, *student_temp=0.1*, *center_momentum=0.9*)

Bases: `torch.nn.modules.module.Module`

__init__(*out_dim*, *ncrops*, *warmup_teacher_temp*, *teacher_temp*, *warmup_teacher_temp_epochs*, *nepochs*, *device*, *student_temp=0.1*, *center_momentum=0.9*)
Initializes internal Module state, shared by both nn.Module and ScriptModule.

forward(*student_output*, *teacher_output*, *epoch*)
Cross-entropy between softmax outputs of the teacher and student networks.

update_center(*teacher_output*)
Update center used for teacher output.

training: bool

easycv.models.selfsup.dino.**has_batchnorms**(*model*)

easycv.models.selfsup.dino.**get_params_groups**(*model*)

class easycv.models.selfsup.dino.**DINO**(*backbone*, *train_preprocess=[]*, *neck=None*, *config=None*, *pretrained=None*)

Bases: *easycv.models.base.BaseModel*

__init__(*backbone*, *train_preprocess=[]*, *neck=None*, *config=None*, *pretrained=None*)
Init Moby

Parameters

- **backbone** – backbone config to build vision backbone
- **train_preprocess** – [gaussBlur, mixUp, solarize]
- **neck** – neck config to build Moby Neck
- **config** – DINO parameter config

get_params_groups()

init_weights(*pretrained=None*)

init_before_train()

momentum_update_key_encoder(*m=0.999*)
ema for dino

forward_train(*inputs*)
Abstract interface for model forward in training

Parameters

- **img** (*Tensor*) – image tensor

- **kwargs** (*keyword arguments*) – Specific to concrete implementation.

forward_test(*img*, ***kwargs*)
Abstract interface for model forward in testing

Parameters

- **img** (*Tensor*) – image tensor
- **kwargs** (*keyword arguments*) – Specific to concrete implementation.

forward_feature(*img*, ***kwargs*)
Forward backbone

Returns feature tensor

Return type x (torch.Tensor)

training: bool

forward(*img*, *gt_label=None*, *mode='train'*, *extract_list=['neck']*, ***kwargs*)
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.

---

easycv.models.selfsup.mae module

class easycv.models.selfsup.mae.**MAE**(*backbone*, *neck*, *mask_ratio=0.75*, *norm_pix_loss=True*, ***kwargs*)
Bases: *easycv.models.base.BaseModel*

__init__(*backbone*, *neck*, *mask_ratio=0.75*, *norm_pix_loss=True*, ***kwargs*)
Initializes internal Module state, shared by both nn.Module and ScriptModule.

patchify(*imgs*)
convert image to patch

Parameters **imgs** – (N, 3, H, W)

Returns (N, L, patch_size**2 *3)

Return type x

forward_loss(*imgs*, *pred*, *mask*)
compute loss

Parameters

- **imgs** – (N, 3, H, W)
- **pred** – (N, L, p*p*3)
- **mask** – (N, L), 0 is keep, 1 is remove,

forward_train(*img*, ***kwargs*)
Abstract interface for model forward in training

Parameters

- **img** (*Tensor*) – image tensor

- **kwargs** (*keyword arguments*) – Specific to concrete implementation.

forward_test(*img*, ***kwargs*)
Abstract interface for model forward in testing

Parameters

- **img** (*Tensor*) – image tensor
- **kwargs** (*keyword arguments*) – Specific to concrete implementation.

forward(*img*, *mode='train'*, ***kwargs*)
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.

---

training: bool

easycv.models.selfsup.mixco module

class easycv.models.selfsup.mixco.**MIXCO**(*backbone*, *train_preprocess=[]*, *neck=None*, *head=None*, *mixco_head=None*, *pretrained=None*, *queue_len=65536*, *feat_dim=128*, *momentum=0.999*, ***kwargs*)
Bases: *easycv.models.selfsup.moco.MOCO*

MOCO.

A mixup version moco https://arxiv.org/pdf/2010.06300.pdf

__init__(*backbone*, *train_preprocess=[]*, *neck=None*, *head=None*, *mixco_head=None*, *pretrained=None*, *queue_len=65536*, *feat_dim=128*, *momentum=0.999*, ***kwargs*)
Initializes internal Module state, shared by both nn.Module and ScriptModule.

forward_train(*img*, ***kwargs*)
Abstract interface for model forward in training

Parameters

- **img** (*Tensor*) – image tensor
- **kwargs** (*keyword arguments*) – Specific to concrete implementation.

training: bool

easycv.models.selfsup.moby module

class easycv.models.selfsup.moby.**MoBY**(*backbone*, *train_preprocess=[]*, *neck=None*, *head=None*, *pretrained=None*, *queue_len=4096*, *contrast_temperature=0.2*, *momentum=0.99*, *online_drop_path_rate=0.2*, *target_drop_path_rate=0.0*, ***kwargs*)
Bases: *easycv.models.base.BaseModel*

MoBY. Part of the code is borrowed from: https://github.com/SwinTransformer/Transformer-SSL/blob/main/models/moby.py.

__init__(*backbone*, *train_preprocess=[]*, *neck=None*, *head=None*, *pretrained=None*, *queue_len=4096*, *contrast_temperature=0.2*, *momentum=0.99*, *online_drop_path_rate=0.2*, *target_drop_path_rate=0.0*, ***kwargs*)

Init Moby

Parameters

- **backbone** – backbone config to build vision backbone
- **train_preprocess** – [gaussBlur, mixUp, solarize]
- **neck** – neck config to build Moby Neck
- **head** – head config to build Moby Neck
- **pretrained** – pretrained weight for backbone
- **queue_len** – moby queue length
- **contrast_temperature** – contrastive_loss temperature
- **momentum** – ema target weights momentum
- **online_drop_path_rate** – for transformer based backbone, set online model drop_path_rate
- **target_drop_path_rate** – for transformer based backbone, set target model drop_path_rate

init_weights()

forward_backbone(*img*)

contrastive_loss(*q*, *k*, *queue*)

forward_train(*img*, ***kwargs*)

Abstract interface for model forward in training

Parameters

- **img** (*Tensor*) – image tensor
- **kwargs** (*keyword arguments*) – Specific to concrete implementation.

forward_test(*img*, ***kwargs*)

Abstract interface for model forward in testing

Parameters

- **img** (*Tensor*) – image tensor
- **kwargs** (*keyword arguments*) – Specific to concrete implementation.

forward_feature(*img*, ***kwargs*)

Forward backbone

Returns feature tensor

Return type x (torch.Tensor)

forward(*img*, *gt_label=None*, *mode='train'*, *extract_list=['neck']*, ***kwargs*)

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.

training: bool

easycv.models.selfsup.moby.**concat_all_gather**(*tensor*)
Performs all_gather operation on the provided tensors. * **Warning** *: torch.distributed.all_gather has no gradient.

easycv.models.selfsup.moco module

class easycv.models.selfsup.moco.**MOCO**(*backbone*, *train_preprocess=[]*, *neck=None*, *head=None*, *pretrained=None*, *queue_len=65536*, *feat_dim=128*, *momentum=0.999*, ***kwargs*)
Bases: *easycv.models.base.BaseModel*

MOCO. Part of the code is borrowed from: https://github.com/facebookresearch/moco/blob/master/moco/builder.py.

__init__(*backbone*, *train_preprocess=[]*, *neck=None*, *head=None*, *pretrained=None*, *queue_len=65536*, *feat_dim=128*, *momentum=0.999*, ***kwargs*)
Initializes internal Module state, shared by both nn.Module and ScriptModule.

init_weights()

forward_backbone(*img*)

forward_train(*img*, ***kwargs*)
Abstract interface for model forward in training

Parameters

- **img** (*Tensor*) – image tensor
- **kwargs** (*keyword arguments*) – Specific to concrete implementation.

forward_test(*img*, ***kwargs*)
Abstract interface for model forward in testing

Parameters

- **img** (*Tensor*) – image tensor
- **kwargs** (*keyword arguments*) – Specific to concrete implementation.

forward_feature(*img*, ***kwargs*)
Forward backbone

Returns feature tensor

Return type x (torch.Tensor)

forward(*img*, *gt_label=None*, *mode='train'*, *extract_list=['neck']*, ***kwargs*)
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.

training: bool

easycv.models.selfsup.moco.**concat_all_gather**(*tensor*)
Performs all_gather operation on the provided tensors. * **Warning** *: torch.distributed.all_gather has no gradient.

easycv.models.selfsup.necks module

class easycv.models.selfsup.necks.**DINONeck**(*in_dim*, *out_dim*, *use_bn=False*, *norm_last_layer=True*, *nlayers=3*, *hidden_dim=2048*, *bottleneck_dim=256*)
Bases: torch.nn.modules.module.Module

__init__(*in_dim*, *out_dim*, *use_bn=False*, *norm_last_layer=True*, *nlayers=3*, *hidden_dim=2048*, *bottleneck_dim=256*)
Initializes internal Module state, shared by both nn.Module and ScriptModule.

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.

training: bool

class easycv.models.selfsup.necks.**MoBYMLP**(*in_channels=256*, *hid_channels=4096*, *out_channels=256*, *num_layers=2*, *with_avg_pool=True*)
Bases: torch.nn.modules.module.Module

__init__(*in_channels=256*, *hid_channels=4096*, *out_channels=256*, *num_layers=2*, *with_avg_pool=True*)
Initializes internal Module state, shared by both nn.Module and ScriptModule.

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.

init_weights(*init_linear='normal'*)

training: bool

class easycv.models.selfsup.necks.**NonLinearNeckSwav**(*in_channels*, *hid_channels*, *out_channels*, *with_avg_pool=True*, *export=False*)
Bases: torch.nn.modules.module.Module

The non-linear neck in byol: fc-syncbn-relu-fc

__init__(*in_channels*, *hid_channels*, *out_channels*, *with_avg_pool=True*, *export=False*)
Initializes internal Module state, shared by both nn.Module and ScriptModule.

init_weights(*init_linear='normal'*)

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.

---

training: bool

class easycv.models.selfsup.necks.**NonLinearNeckV0**(*in_channels*, *hid_channels*, *out_channels*, *sync_bn=False*, *with_avg_pool=True*)

Bases: `torch.nn.modules.module.Module`

The non-linear neck in ODC, fc-bn-relu-dropout-fc-relu

__init__(*in_channels*, *hid_channels*, *out_channels*, *sync_bn=False*, *with_avg_pool=True*)
Initializes internal Module state, shared by both nn.Module and ScriptModule.

init_weights(*init_linear='normal'*)

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.

---

training: bool

class easycv.models.selfsup.necks.**NonLinearNeckV1**(*in_channels*, *hid_channels*, *out_channels*, *with_avg_pool=True*)

Bases: `torch.nn.modules.module.Module`

The non-linear neck in MoCO v2: fc-relu-fc

__init__(*in_channels*, *hid_channels*, *out_channels*, *with_avg_pool=True*)
Initializes internal Module state, shared by both nn.Module and ScriptModule.

init_weights(*init_linear='normal'*)

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.

---

training: bool

class easycv.models.selfsup.necks.**NonLinearNeckV2**(*in_channels*, *hid_channels*, *out_channels*, *with_avg_pool=True*)

Bases: torch.nn.modules.module.Module

The non-linear neck in byol: fc-bn-relu-fc

__init__(*in_channels*, *hid_channels*, *out_channels*, *with_avg_pool=True*)
Initializes internal Module state, shared by both nn.Module and ScriptModule.

init_weights(*init_linear='normal'*)

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.

---

training: bool

class easycv.models.selfsup.necks.**NonLinearNeckSimCLR**(*in_channels*, *hid_channels*, *out_channels*, *num_layers=2*, *with_avg_pool=True*)

Bases: torch.nn.modules.module.Module

SimCLR non-linear neck.

Structure: fc(no_bias)-bn(has_bias)-[relu-fc(no_bias)-bn(no_bias)]. The substructures in [] can be repeated. For the SimCLR default setting, the repeat time is 1.

However, PyTorch does not support to specify (weight=True, bias=False). It only support "affine" including the weight and bias. Hence, the second BatchNorm has bias in this implementation. This is different from the offical implementation of SimCLR.

Since SyncBatchNorm in pytorch<1.4.0 does not support 2D input, the input is expanded to 4D with shape: (N,C,1,1). I am not sure if this workaround has no bugs. See the pull request here: https://github.com/pytorch/pytorch/pull/29626

Parameters

- **in_channels** – input channel number
- **hid_channels** – hidden channels
- **out_channels** – output channel number
- **num_layers** (*int*) – number of fc layers, it is 2 in the SimCLR default setting.
- **with_avg_pool** – output with average pooling

__init__(*in_channels*, *hid_channels*, *out_channels*, *num_layers=2*, *with_avg_pool=True*)
Initializes internal Module state, shared by both nn.Module and ScriptModule.

init_weights(*init_linear='normal'*)

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.

training: bool

class easycv.models.selfsup.necks.**RelativeLocNeck**(*in_channels*, *out_channels*, *sync_bn=False*, *with_avg_pool=True*)

Bases: torch.nn.modules.module.Module

Relative patch location neck: fc-bn-relu-dropout

__init__(*in_channels*, *out_channels*, *sync_bn=False*, *with_avg_pool=True*)
Initializes internal Module state, shared by both nn.Module and ScriptModule.

init_weights(*init_linear='normal'*)

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.

training: bool

class easycv.models.selfsup.necks.**MAENeck**(*num_patches*, *embed_dim=768*, *patch_size=16*, *in_chans=3*, *decoder_embed_dim=512*, *decoder_depth=8*, *decoder_num_heads=16*, *mlp_ratio=4.0*, *norm_layer=functools.partial(<class 'torch.nn.modules.normalization.LayerNorm'>, eps=1e-06)*)

Bases: torch.nn.modules.module.Module

MAE decoder

Parameters

- **num_patches** (*int*) – number of patches from encoder
- **embed_dim** (*int*) – encoder embedding dimension
- **patch_size** (*int*) – encoder patch size
- **in_chans** (*int*) – input image channels
- **decoder_embed_dim** (*int*) – decoder embedding dimension
- **decoder_depth** (*int*) – number of decoder layers
- **decoder_num_heads** (*int*) – Parallel attention heads
- **mlp_ratio** (*float*) – mlp ratio
- **norm_layer** – type of normalization layer

__init__(*num_patches*, *embed_dim=768*, *patch_size=16*, *in_chans=3*, *decoder_embed_dim=512*, *decoder_depth=8*, *decoder_num_heads=16*, *mlp_ratio=4.0*, *norm_layer=functools.partial(<class 'torch.nn.modules.normalization.LayerNorm'>, eps=1e-06)*)
Initializes internal Module state, shared by both nn.Module and ScriptModule.

training: bool

forward(*x*, *ids_restore*)
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.

---

easycv.models.selfsup.simclr module

class easycv.models.selfsup.simclr.**SimCLR**(*backbone*, *train_preprocess=[]*, *neck=None*, *head=None*, *pretrained=None*)

Bases: *easycv.models.base.BaseModel*

__init__(*backbone*, *train_preprocess=[]*, *neck=None*, *head=None*, *pretrained=None*)
Initializes internal Module state, shared by both nn.Module and ScriptModule.

init_weights()

forward_backbone(*img*)
Forward backbone

Returns backbone outputs

Return type x (tuple)

forward_train(*img*, ***kwargs*)
Abstract interface for model forward in training

Parameters

- **img** (*Tensor*) – image tensor
- **kwargs** (*keyword arguments*) – Specific to concrete implementation.

forward_test(*img*, ***kwargs*)
Abstract interface for model forward in testing

Parameters

- **img** (*Tensor*) – image tensor
- **kwargs** (*keyword arguments*) – Specific to concrete implementation.

forward(*img*, *mode='train'*, ***kwargs*)
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.

---

training: bool

easycv.models.selfsup.swav module

class easycv.models.selfsup.swav.**SWAV**(*backbone*, *train_preprocess=[]*, *neck=None*, *config=None*, *pretrained=None*)

Bases: *easycv.models.base.BaseModel*

__init__(*backbone*, *train_preprocess=[]*, *neck=None*, *config=None*, *pretrained=None*)
Initializes internal Module state, shared by both nn.Module and ScriptModule.

init_weights()

forward_backbone(*img*)

forward_train_model(*inputs*)

forward_train(*inputs*)
Abstract interface for model forward in training

Parameters

- **img** (*Tensor*) – image tensor
- **kwargs** (*keyword arguments*) – Specific to concrete implementation.

forward_test(*img*, ***kwargs*)
Abstract interface for model forward in testing

Parameters

- **img** (*Tensor*) – image tensor
- **kwargs** (*keyword arguments*) – Specific to concrete implementation.

forward_feature(*img*, ***kwargs*)
Forward backbone

Returns feature tensor

Return type x (torch.Tensor)

forward(*img*, *gt_label=None*, *mode='train'*, *extract_list=['neck']*, ***kwargs*)
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.

---

training: bool

class easycv.models.selfsup.swav.**MultiPrototypes**(*output_dim*, *nmb_prototypes*)

Bases: `torch.nn.modules.module.Module`

__init__(*output_dim*, *nmb_prototypes*)
Initializes internal Module state, shared by both nn.Module and ScriptModule.

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.

training: bool

easycv.models.selfsup.swav.**distributed_sinkhorn**(*Q*, *nmb_iters*)

18.1.8 easycv.models.utils package

Submodules

easycv.models.utils.accuracy module

easycv.models.utils.activation module

class easycv.models.utils.activation.**FReLU**(*in_channel*)

Bases: torch.nn.modules.module.Module

__init__(*in_channel*)

Initializes internal Module state, shared by both nn.Module and ScriptModule.

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.

training: bool

easycv.models.utils.activation.**build_activation_layer**(*cfg*)

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

easycv.models.utils.conv_module module

easycv.models.utils.conv_module.**build_conv_layer**(*cfg*, **args*, ***kwargs*)

Build convolution layer

Parameters **cfg** (*None or dict*) – cfg should contain: type (str): identify conv layer type. layer args: args needed to instantiate a conv layer.

Returns created conv layer

Return type layer (nn.Module)

class easycv.models.utils.conv_module.**ConvModule**(*in_channels*, *out_channels*, *kernel_size*, *stride=1*, *padding=0*, *dilation=1*, *groups=1*, *bias='auto'*, *conv_cfg=None*, *norm_cfg=None*, *act_cfg={'type': 'ReLU'}*, *inplace=True*, *order=('conv', 'norm', 'act')*)

Bases: torch.nn.modules.module.Module

A conv block that contains conv/norm/activation 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 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.
- **conv_cfg** (*dict*) – Config dict for convolution layer.
- **norm_cfg** (*dict*) – Config dict for normalization layer.
- **act_cfg** (*dict*) – Config of activation layers. Default: dict(type='ReLU')
- **inplace** (*bool*) – Whether to use inplace mode for activation.
- **order** (*tuple[str]*) – The order of conv/norm/activation layers. It is a sequence of "conv", "norm" and "act". Examples are ("conv", "norm", "act") and ("act", "conv", "norm").

__init__(*in_channels*, *out_channels*, *kernel_size*, *stride=1*, *padding=0*, *dilation=1*, *groups=1*, *bias='auto'*, *conv_cfg=None*, *norm_cfg=None*, *act_cfg={'type': 'ReLU'}*, *inplace=True*, *order=('conv', 'norm', 'act')*)

Initializes internal Module state, shared by both nn.Module and ScriptModule.

property norm

init_weights()

forward(*x*, *activate=True*, *norm=True*)

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.

---

training: bool

easycv.models.utils.conv_ws module

easycv.models.utils.conv_ws.**conv_ws_2d**(*input*, *weight*, *bias=None*, *stride=1*, *padding=0*, *dilation=1*, *groups=1*, *eps=1e-05*)

class easycv.models.utils.conv_ws.**ConvWS2d**(*in_channels*, *out_channels*, *kernel_size*, *stride=1*, *padding=0*, *dilation=1*, *groups=1*, *bias=True*, *eps=1e-05*)

Bases: torch.nn.modules.conv.Conv2d

__init__(*in_channels*, *out_channels*, *kernel_size*, *stride=1*, *padding=0*, *dilation=1*, *groups=1*, *bias=True*, *eps=1e-05*)

Initializes internal Module state, shared by both nn.Module and ScriptModule.

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.

---

bias: Optional[torch.Tensor]

out_channels: int

kernel_size: Tuple[int, ...]

stride: Tuple[int, ...]

padding: Union[str, Tuple[int, ...]]

dilation: Tuple[int, ...]

transposed: bool

output_padding: Tuple[int, ...]

groups: int

padding_mode: str

weight: torch.Tensor

easycv.models.utils.dist_utils module

easycv.models.utils.dist_utils.**is_distributed**()

easycv.models.utils.dist_utils.**all_gather**(*embeddings*, *labels*)

easycv.models.utils.dist_utils.**all_gather_embeddings_labels**(*embeddings*, *labels*)

class easycv.models.utils.dist_utils.**DistributedLossWrapper**(*loss*, ***kwargs*)

Bases: torch.nn.modules.module.Module

__init__(*loss*, ***kwargs*)

Initializes internal Module state, shared by both nn.Module and ScriptModule.

forward(*embeddings*, *labels*, **args*, ***kwargs*)

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.

training: bool

class easycv.models.utils.dist_utils.**DistributedMinerWrapper**(*miner*)

Bases: torch.nn.modules.module.Module

__init__(*miner*)

Initializes internal Module state, shared by both nn.Module and ScriptModule.

forward(*embeddings*, *labels*, *ref_emb=None*, *ref_labels=None*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

Note: Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

training: bool

easycv.models.utils.gather_layer module

class easycv.models.utils.gather_layer.**GatherLayer**(**args*, ***kwargs*)

Bases: torch.autograd.function.Function

Gather tensors from all process, supporting backward propagation.

static forward(*ctx*, *input*)

Performs the operation.

This function is to be overridden by all subclasses.

It must accept a context ctx as the first argument, followed by any number of arguments (tensors or other types).

The context can be used to store arbitrary data that can be then retrieved during the backward pass. Tensors should not be stored directly on *ctx* (though this is not currently enforced for backward compatibility). Instead, tensors should be saved either with `ctx.save_for_backward()` if they are intended to be used in `backward` (equivalently, `vjp`) or `ctx.save_for_forward()` if they are intended to be used for in `jvp`.

static backward(*ctx*, **grads*)

Defines a formula for differentiating the operation with backward mode automatic differentiation (alias to the vjp function).

This function is to be overridden by all subclasses.

It must accept a context `ctx` as the first argument, followed by as many outputs as the *forward()* returned (None will be passed in for non tensor outputs of the forward function), and it should return as many tensors, as there were inputs to *forward()*. Each argument is the gradient w.r.t the given output, and each returned value should be the gradient w.r.t. the corresponding input. If an input is not a Tensor or is a Tensor not requiring grads, you can just pass None as a gradient for that input.

The context can be used to retrieve tensors saved during the forward pass. It also has an attribute `ctx.needs_input_grad` as a tuple of booleans representing whether each input needs gradient. E.g.,

backward() will have `ctx.needs_input_grad[0] = True` if the first input to *forward()* needs gradient computated w.r.t. the output.

easycv.models.utils.init_weights module

easycv.models.utils.init_weights.**trunc_normal_**(*tensor*, *mean=0.0*, *std=1.0*, *a=- 2.0*, *b=2.0*)

easycv.models.utils.multi_pooling module

class easycv.models.utils.multi_pooling.**GeMPooling**(*p=3*, *eps=1e-06*)

Bases: torch.nn.modules.module.Module

GemPooling used for image retrival p = 1, avgpooling p > 1 : increases the contrast of the pooled feature map and focuses on the salient features of the image p = infinite : spatial max-pooling layer

__init__(*p=3*, *eps=1e-06*)

Initializes internal Module state, shared by both nn.Module and ScriptModule.

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.

---

gem(*x*, *p=3*, *eps=1e-06*)

training: bool

class easycv.models.utils.multi_pooling.**MultiPooling**(*pool_type='adaptive'*, *in_indices=(0)*, *backbone='resnet50'*)

Bases: torch.nn.modules.module.Module

Pooling layers for features from multiple depth.

POOL_PARAMS = {'resnet50': [{'kernel_size': 10, 'stride': 10, 'padding': 4}, {'kernel_size': 16, 'stride': 8, 'padding': 0}, {'kernel_size': 13, 'stride': 5, 'padding': 0}, {'kernel_size': 8, 'stride': 3, 'padding': 0}, {'kernel_size': 6, 'stride': 1, 'padding': 0}]}

POOL_SIZES = {'resnet50': [12, 6, 4, 3, 2]}

POOL_DIMS = {'resnet50': [9216, 9216, 8192, 9216, 8192]}

__init__(*pool_type='adaptive'*, *in_indices=(0)*, *backbone='resnet50'*)

Initializes internal Module state, shared by both nn.Module and ScriptModule.

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.

---

training: bool

class easycv.models.utils.multi_pooling.**MultiAvgPooling**(*pool_type='adaptive'*, *in_indices=(0)*, *backbone='resnet50'*)

Bases: torch.nn.modules.module.Module

Pooling layers for features from multiple depth.

POOL_PARAMS = {'resnet50': [{'kernel_size': 10, 'stride': 10, 'padding': 4}, {'kernel_size': 16, 'stride': 8, 'padding': 0}, {'kernel_size': 13, 'stride': 5, 'padding': 0}, {'kernel_size': 8, 'stride': 3, 'padding': 0}, {'kernel_size': 7, 'stride': 1, 'padding': 0}]}

POOL_SIZES = {'resnet50': [12, 6, 4, 3, 1]}

POOL_DIMS = {'resnet50': [9216, 9216, 8192, 9216, 2048]}

__init__(*pool_type='adaptive'*, *in_indices=(0)*, *backbone='resnet50'*)
Initializes internal Module state, shared by both nn.Module and ScriptModule.

training: bool

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.

---

easycv.models.utils.norm module

class easycv.models.utils.norm.**SyncIBN**(*planes*, *ratio=0.5*, *eps=1e-05*)
Bases: torch.nn.modules.module.Module

Instance-Batch Normalization layer from *"Two at Once: Enhancing Learning and Generalization Capacities via IBN-Net"* <*https://arxiv.org/pdf/1807.09441.pdf*> :param planes: Number of channels for the input tensor :type planes: int :param ratio: Ratio of instance normalization in the IBN layer :type ratio: float

__init__(*planes*, *ratio=0.5*, *eps=1e-05*)
Initializes internal Module state, shared by both nn.Module and ScriptModule.

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.

---

training: bool

class easycv.models.utils.norm.**IBN**(*planes*, *ratio=0.5*, *eps=1e-05*)
Bases: torch.nn.modules.module.Module

Instance-Batch Normalization layer from *"Two at Once: Enhancing Learning and Generalization Capacities via IBN-Net" <https://arxiv.org/pdf/1807.09441.pdf>* :param planes: Number of channels for the input tensor :type planes: int :param ratio: Ratio of instance normalization in the IBN layer :type ratio: float

__init__(*planes*, *ratio=0.5*, *eps=1e-05*)
Initializes internal Module state, shared by both nn.Module and ScriptModule.

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.

---

training: bool

easycv.models.utils.norm.**build_norm_layer**(*cfg*, *num_features*, *postfix=''*)
Build normalization layer

Parameters

- **cfg** (*dict*) – cfg should contain: type (str): identify norm layer type. layer args: args needed to instantiate a norm layer. requires_grad (bool): [optional] whether stop gradient updates
- **num_features** (*int*) – number of channels from input.
- **postfix** (*int, str*) – appended into norm abbreviation to create named layer.

Returns abbreviation + postfix layer (nn.Module): created norm layer

Return type name (str)

easycv.models.utils.ops module

easycv.models.utils.ops.**resize_tensor**(*input*, *size=None*, *scale_factor=None*, *mode='nearest'*, *align_corners=None*, *warning=True*)
Resize tensor with F.interpolate.

Parameters

- **input** (*Tensor*) – the input tensor.
- **size** (*Tuple[int, int]*) – output spatial size.
- **scale_factor** (*float or Tuple[float]*) – multiplier for spatial size. If scale_factor is a tuple, its length has to match input.dim().
- **mode** (*str*) – algorithm used for upsampling: 'nearest' | 'linear' | 'bilinear' | 'bicubic' | 'trilinear' | 'area'. Default: 'nearest'
- **align_corners** (*bool*) – Geometrically, we consider the pixels of the input and output as squares rather than points. If set to True, the input and output tensors are aligned by the center points of their corner pixels, preserving the values at the corner pixels.

  If set to False, the input and output tensors are aligned by the corner points of their corner pixels, and the interpolation uses edge value padding for out-of-boundary values, making this operation independent of input size when scale_factor is kept the same. This only has an effect when mode is 'linear', 'bilinear', 'bicubic' or 'trilinear'.

easycv.models.utils.pos_embed module

easycv.models.utils.pos_embed.**get_2d_sincos_pos_embed**(*embed_dim*, *grid_size*, *cls_token=False*)
grid_size: int of the grid height and width return: pos_embed: [grid_size*grid_size, embed_dim] or [1+grid_size*grid_size, embed_dim] (w/ or w/o cls_token)

easycv.models.utils.pos_embed.**get_2d_sincos_pos_embed_from_grid**(*embed_dim*, *grid*)

easycv.models.utils.pos_embed.**get_1d_sincos_pos_embed_from_grid**(*embed_dim*, *pos*)
embed_dim: output dimension for each position pos: a list of positions to be encoded: size (M,) out: (M, D)

easycv.models.utils.pos_embed.**interpolate_pos_embed**(*model*, *checkpoint_model*)

easycv.models.utils.res_layer module

class easycv.models.utils.res_layer.**ResLayer**(*block*, *num_blocks*, *in_channels*, *out_channels*, *expansion=None*, *stride=1*, *avg_down=False*, *conv_cfg=None*, *norm_cfg={'type': 'BN'}*, ***kwargs*)
Bases: `torch.nn.modules.container.Sequential`

ResLayer to build ResNet style backbone. :param block: Residual block used to build ResLayer. :type block: nn.Module :param num_blocks: Number of blocks. :type num_blocks: int :param in_channels: Input channels of this block. :type in_channels: int :param out_channels: Output channels of this block. :type out_channels: int :param expansion: The expansion for BasicBlock/Bottleneck.
If not specified, it will firstly be obtained via `block.expansion`. If the block has no attribute "expansion", the following default values will be used: 1 for BasicBlock and 4 for Bottleneck. Default: None.

Parameters

- **stride** (*int*) – stride of the first block. Default: 1.
- **avg_down** (*bool*) – Use AvgPool instead of stride conv when downsampling in the bottleneck. Default: False
- **conv_cfg** (*dict, optional*) – dictionary to construct and config conv layer. Default: None
- **norm_cfg** (*dict*) – dictionary to construct and config norm layer. Default: dict(type='BN')

__init__(*block*, *num_blocks*, *in_channels*, *out_channels*, *expansion=None*, *stride=1*, *avg_down=False*, *conv_cfg=None*, *norm_cfg={'type': 'BN'}*, ***kwargs*)
Initializes internal Module state, shared by both nn.Module and ScriptModule.

easycv.models.utils.scale module

class easycv.models.utils.scale.**Scale**(*scale=1.0*)
Bases: `torch.nn.modules.module.Module`

A learnable scale parameter

__init__(*scale=1.0*)
Initializes internal Module state, shared by both nn.Module and ScriptModule.

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.

training: bool

easycv.models.utils.sobel module

class easycv.models.utils.sobel.**Sobel**

Bases: torch.nn.modules.module.Module

__init__()

Initializes internal Module state, shared by both nn.Module and ScriptModule.

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.

training: bool

18.2 Submodules

18.3 easycv.models.base module

class easycv.models.base.**BaseModel**

Bases: torch.nn.modules.module.Module

base class for model.

__init__()

Initializes internal Module state, shared by both nn.Module and ScriptModule.

abstract forward_train(*img: torch.Tensor*, ***kwargs*) → Dict[str, torch.Tensor]

Abstract interface for model forward in training

Parameters

- **img** (*Tensor*) – image tensor
- **kwargs** (*keyword arguments*) – Specific to concrete implementation.

forward_test(*img: torch.Tensor*, ***kwargs*) → Dict[str, torch.Tensor]

Abstract interface for model forward in testing

Parameters

- **img** (*Tensor*) – image tensor
- **kwargs** (*keyword arguments*) – Specific to concrete implementation.

forward(*img*, *mode='train'*, ***kwargs*)
Defines the computation performed at every call.

Should be overridden by all subclasses.

Note: Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

train_step(*data*, *optimizer*)
The iteration step during training.

This method defines an iteration step during training, except for the back propagation and optimizer updating, which are done in an optimizer hook. Note that in some complicated cases or models, the whole process including back propagation and optimizer updating is also defined in this method, such as GAN.

Parameters

- **data** (*dict*) – The output of dataloader.
- **optimizer** (`torch.optim.Optimizer` | dict) – The optimizer of runner is passed to `train_step()`. This argument is unused and reserved.

Returns

It should contain at least 3 keys: `loss`, `log_vars`, `num_samples`.

- `loss` is a tensor for back propagation, which can be a weighted sum of multiple losses.
- `log_vars` contains all the variables to be sent to the logger.
- `num_samples` indicates the batch size (when the model is DDP, it means the batch size on each GPU), which is used for averaging the logs.

Return type dict

val_step(*data*, *optimizer*)
The iteration step during validation.

This method shares the same signature as *train_step()*, but used during val epochs. Note that the evaluation after training epochs is not implemented with this method, but an evaluation hook.

show_result(***kwargs*)
Visualize the results.

training: bool

18.4 easycv.models.builder module

easycv.models.builder.**build**(*cfg*, *registry*, *default_args=None*)

easycv.models.builder.**build_backbone**(*cfg*)

easycv.models.builder.**build_neck**(*cfg*)

easycv.models.builder.**build_memory**(*cfg*)

easycv.models.builder.**build_head**(*cfg*)

easycv.models.builder.**build_loss**(*cfg*)

easycv.models.builder.**build_model**(*cfg*)

18.5 easycv.models.modelzoo module

18.6 easycv.models.registry module

CHAPTER

NINETEEN

EASYCV.UTILS PACKAGE

19.1 Submodules

19.2 easycv.utils.alias_multinomial module

class easycv.utils.alias_multinomial.**AliasMethod**(*probs*)
Bases: object

https://hips.seas.harvard.edu/blog/2013/03/03/the-alias-method-efficient-sampling-with-many-discrete-outcomes/

__init__(*probs*)
Initialize self. See help(type(self)) for accurate signature.

cuda()

draw(*N*)
Draw N samples from multinomial

19.3 easycv.utils.bbox_util module

easycv.utils.bbox_util.**bound_limits**(*v*)

easycv.utils.bbox_util.**bound_limits_for_list**(*xywh*)

easycv.utils.bbox_util.**xyxy2xywh_with_shape**(*x*, *shape*)

easycv.utils.bbox_util.**batched_cxcywh2xyxy_with_shape**(*bboxes*, *shape*)
reverse of *xyxy2xywh_with_shape* transform normalized points *[[x_center, y_center, box_w, box_h],...]* to standard [[x1, y1, x2, y2],...] :param bboxes: np.array or tensor like [[x_center, y_center, box_w, box_h],...], all value is normalized

Parameters **shape** – img shape: [h, w]

return: np.array or tensor like [[x1, y1, x2, y2],...]

easycv.utils.bbox_util.**batched_xyxy2cxcywh_with_shape**(*bboxes*, *shape*)

easycv.utils.bbox_util.**xyxy2xywh_coco**(*bboxes*, *offset=0*)

easycv.utils.bbox_util.**xywh2xyxy_coco**(*bboxes*, *offset=0*)

easycv.utils.bbox_util.**xyxy2xywh**(*x*)

easycv.utils.bbox_util.**xywh2xyxy**(*x*)

easycv.utils.bbox_util.**bbox_iou**(*box1*, *box2*, *x1y1x2y2=True*, *GIoU=False*, *DIoU=False*, *CIoU=False*, *eps=1e-09*)

easycv.utils.bbox_util.**box_iou**(*box1*, *box2*)

Return intersection-over-union (Jaccard index) of boxes. Both sets of boxes are expected to be in (x1, y1, x2, y2) format. :param box1: :type box1: Tensor[N, 4] :param box2: :type box2: Tensor[M, 4]

Returns

the NxM matrix containing the pairwise IoU values for every element in boxes1 and boxes2

Return type iou (Tensor[N, M])

easycv.utils.bbox_util.**box_candidates**(*box1*, *box2*, *wh_thr=2*, *ar_thr=20*, *area_thr=0.1*)

Compute candidate boxes: box1 before augment, box2 after augment, wh_thr (pixels), aspect_ratio_thr, area_ratio

easycv.utils.bbox_util.**clip_coords**(*boxes: torch.Tensor*, *img_shape: Tuple[int, int]*) → None

Clip bounding xyxy bounding boxes to image shape

Parameters

- **boxes** – tensor with shape Nx4 (x1,y1,x2,y2)
- **img_shape** – image size tuple, (height, width)

easycv.utils.bbox_util.**scale_coords**(*img1_shape: Tuple[int, int]*, *coords: torch.Tensor*, *img0_shape: Tuple[int, int]*, *ratio_pad: Optional[Tuple[Tuple[float, float], Tuple[float, float]]] = None*)

19.4 easycv.utils.checkpoint module

easycv.utils.checkpoint.**load_checkpoint**(*model*, *filename*, *map_location='cpu'*, *strict=False*, *logger=None*)

Load checkpoint from a file or URI.

Parameters

- **model** (*Module*) – Module to load checkpoint.
- **filename** (*str*) – Accept local filepath, URL, `torchvision://xxx`, `open-mmlab://xxx`. Please refer to `docs/model_zoo.md` for details.
- **map_location** (*str*) – Same as `torch.load()`.
- **strict** (*bool*) – Whether to allow different params for the model and checkpoint.
- **logger** (`logging.Logger` or None) – The logger for error message.

Returns The loaded checkpoint.

Return type dict or OrderedDict

easycv.utils.checkpoint.**save_checkpoint**(*model*, *filename*, *optimizer=None*, *meta=None*)

Save checkpoint to file.

The checkpoint will have 3 fields: `meta`, `state_dict` and `optimizer`. By default `meta` will contain version and time info.

Parameters

- **model** (*Module*) – Module whose params are to be saved.
- **filename** (*str*) – Checkpoint filename.

- **optimizer** (Optimizer, optional) – Optimizer to be saved.
- **meta** (*dict, optional*) – Metadata to be saved in checkpoint.

19.5 easycv.utils.collect module

easycv.utils.collect.**nondist_forward_collect**(*func*, *data_loader*, *length*)
Forward and collect network outputs.

This function performs forward propagation and collects outputs. It can be used to collect results, features, losses, etc.

Parameters

- **func** (*function*) – The function to process data. The output must be a dictionary of CPU tensors.
- **length** (*int*) – Expected length of output arrays.

Returns The concatenated outputs.

Return type results_all (dict(np.ndarray))

easycv.utils.collect.**dist_forward_collect**(*func*, *data_loader*, *rank*, *length*, *ret_rank=-1*)
Forward and collect network outputs in a distributed manner.

This function performs forward propagation and collects outputs. It can be used to collect results, features, losses, etc.

Parameters

- **func** (*function*) – The function to process data. The output must be a dictionary of CPU tensors.
- **rank** (*int*) – This process id.
- **length** (*int*) – Expected length of output arrays.
- **ret_rank** (*int*) – The process that returns. Other processes will return None.

Returns The concatenated outputs.

Return type results_all (dict(np.ndarray))

19.6 easycv.utils.collect_env module

easycv.utils.collect_env.**collect_env**()

19.7 easycv.utils.config_tools module

easycv.utils.config_tools.**traverse_replace**(*d*, *key*, *value*)

easycv.utils.config_tools.**check_base_cfg_path**(*base_cfg_name='configs/base.py'*, *ori_filename=None*)

easycv.utils.config_tools.**mmcv_file2dict_raw**(*ori_filename*)

easycv.utils.config_tools.**mmcv_file2dict_base**(*ori_filename*)

easycv.utils.config_tools.**mmcv_config_fromfile**(*ori_filename*)

easycv.utils.config_tools.**get_config_class_value**(*cfg_dict*, *ori_key*, *dict_mem_helper*)

easycv.utils.config_tools.**config_dict_edit**(*ori_cfg_dict*, *cfg_dict*, *reg*, *dict_mem_helper*)
edit ${configs.variables} in config dict to solve dependicies in config

ori_cfg_dict: to find the true value of ${configs.variables} cfg_dict: for find leafs of dict by recursive reg: Regular expression pattern for find all ${configs.variables} in leafs of dict dict_mem_helper: to store the true value of ${configs.variables} which have been found

easycv.utils.config_tools.**rebuild_config**(*cfg*, *user_config_params*)
rebuild config by user config params, modify config by user config params & replace ${configs.variables} by true value

return: Config

easycv.utils.config_tools.**validate_export_config**(*cfg*)

19.8 easycv.utils.constant module

19.9 easycv.utils.dist_utils module

easycv.utils.dist_utils.**is_master**()

easycv.utils.dist_utils.**local_rank**()

easycv.utils.dist_utils.**dist_zero_exec**(*rank=0*)

easycv.utils.dist_utils.**get_num_gpu_per_node**()
get number of gpu per node

easycv.utils.dist_utils.**barrier**()

easycv.utils.dist_utils.**is_parallel**(*model*)

easycv.utils.dist_utils.**obj2tensor**(*pyobj*, *device='cuda'*)
Serialize picklable python object to tensor.

easycv.utils.dist_utils.**tensor2obj**(*tensor*)
Deserialize tensor to picklable python object.

easycv.utils.dist_utils.**all_reduce_dict**(*py_dict*, *op='sum'*, *group=None*, *to_float=True*)
Apply all reduce function for python dict object.

The code is modified from https://github.com/Megvii- BaseDetection/YOLOX/blob/main/yolox/utils/allreduce_norm.py.

NOTE: make sure that py_dict in different ranks has the same keys and the values should be in the same shape.

Parameters

- **py_dict** (*dict*) – Dict to be applied all reduce op.
- **op** (*str*) – Operator, could be 'sum' or 'mean'. Default: 'sum'
- **group** (torch.distributed.group, optional) – Distributed group, Default: None.
- **to_float** (*bool*) – Whether to convert all values of dict to float. Default: True.

Returns reduced python dict object.

Return type OrderedDict

19.10 easycv.utils.eval_utils module

easycv.utils.eval_utils.**generate_best_metric_name**(*evaluate_type*, *dataset_name*, *metric_names*)
Generate best metric name for different evaluator / different dataset / different metric_names evaluate_type: str dataset_name: None or str metric_names: None str or list[str] or tuple(str)
Returns list[str]

19.11 easycv.utils.flops_counter module

easycv.utils.flops_counter.**get_model_info**(*model*, *input_size*, *model_config*, *logger*)
get_model_info, check model parameters and Gflops

easycv.utils.flops_counter.**get_model_complexity_info**(*model*, *input_res*, *print_per_layer_stat=True*, *as_strings=True*, *input_constructor=None*, *ost=<_io.TextIOWrapper name='<stdout>' mode='w' encoding='UTF-8'>*)

easycv.utils.flops_counter.**flops_to_string**(*flops*, *units='GMac'*, *precision=2*)

easycv.utils.flops_counter.**params_to_string**(*params_num*)
converting number to string
Parameters **params_num** (*float*) – number

Returns str number

```
>>> params_to_string(1e9)
'1000.0 M'
>>> params_to_string(2e5)
'200.0 k'
>>> params_to_string(3e-9)
'3e-09'
```

easycv.utils.flops_counter.**print_model_with_flops**(*model*, *units='GMac'*, *precision=3*, *ost=<_io.TextIOWrapper name='<stdout>' mode='w' encoding='UTF-8'>*)

easycv.utils.flops_counter.**get_model_parameters_number**(*model*)

easycv.utils.flops_counter.**add_flops_counting_methods**(*net_main_module*)

easycv.utils.flops_counter.**compute_average_flops_cost**(*self*)
A method that will be available after add_flops_counting_methods() is called on a desired net object. Returns current mean flops consumption per image.

easycv.utils.flops_counter.**start_flops_count**(*self*)
A method that will be available after add_flops_counting_methods() is called on a desired net object. Activates the computation of mean flops consumption per image. Call it before you run the network.

easycv.utils.flops_counter.**stop_flops_count**(*self*)
A method that will be available after add_flops_counting_methods() is called on a desired net object. Stops computing the mean flops consumption per image. Call whenever you want to pause the computation.

easycv.utils.flops_counter.**reset_flops_count**(*self*)
A method that will be available after add_flops_counting_methods() is called on a desired net object. Resets statistics computed so far.

easycv.utils.flops_counter.**add_flops_mask**(*module*, *mask*)

easycv.utils.flops_counter.**remove_flops_mask**(*module*)

easycv.utils.flops_counter.**is_supported_instance**(*module*)

easycv.utils.flops_counter.**empty_flops_counter_hook**(*module*, *input*, *output*)

easycv.utils.flops_counter.**upsample_flops_counter_hook**(*module*, *input*, *output*)

easycv.utils.flops_counter.**relu_flops_counter_hook**(*module*, *input*, *output*)

easycv.utils.flops_counter.**linear_flops_counter_hook**(*module*, *input*, *output*)

easycv.utils.flops_counter.**pool_flops_counter_hook**(*module*, *input*, *output*)

easycv.utils.flops_counter.**bn_flops_counter_hook**(*module*, *input*, *output*)

easycv.utils.flops_counter.**gn_flops_counter_hook**(*module*, *input*, *output*)

easycv.utils.flops_counter.**deconv_flops_counter_hook**(*conv_module*, *input*, *output*)

easycv.utils.flops_counter.**conv_flops_counter_hook**(*conv_module*, *input*, *output*)

easycv.utils.flops_counter.**batch_counter_hook**(*module*, *input*, *output*)

easycv.utils.flops_counter.**add_batch_counter_variables_or_reset**(*module*)

easycv.utils.flops_counter.**add_batch_counter_hook_function**(*module*)

easycv.utils.flops_counter.**remove_batch_counter_hook_function**(*module*)

easycv.utils.flops_counter.**add_flops_counter_variable_or_reset**(*module*)

easycv.utils.flops_counter.**add_flops_counter_hook_function**(*module*)

easycv.utils.flops_counter.**remove_flops_counter_hook_function**(*module*)

easycv.utils.flops_counter.**add_flops_mask_variable_or_reset**(*module*)

19.12 easycv.utils.gather module

easycv.utils.gather.**gather_tensors**(*input_array*)

easycv.utils.gather.**gather_tensors_batch**(*input_array*, *part_size=100*, *ret_rank=- 1*)

19.13 easycv.utils.json_utils module

Utilities for dealing with writing json strings.

json_utils wraps json.dump and json.dumps so that they can be used to safely control the precision of floats when writing to json strings or files.

class easycv.utils.json_utils.**MyEncoder**(**, skipkeys=False, ensure_ascii=True, check_circular=True, allow_nan=True, sort_keys=False, indent=None, separators=None, default=None*)

Bases: `json.encoder.JSONEncoder`

default(*o*)

Implement this method in a subclass such that it returns a serializable object for `o`, or calls the base implementation (to raise a `TypeError`).

For example, to support arbitrary iterators, you could implement default like this:

```
def default(self, o):
    try:
        iterable = iter(o)
    except TypeError:
        pass
    else:
        return list(iterable)
    # Let the base class default method raise the TypeError
    return JSONEncoder.default(self, o)
```

iterencode(*o*, *_one_shot=False*)

Encode the given object and yield each string representation as available.

For example:

```
for chunk in JSONEncoder().iterencode(bigobject):
    mysocket.write(chunk)
```

easycv.utils.json_utils.**dump**(*obj*, *fid*, *float_digits=- 1*, ***params*)

Wrapper of json.dump that allows specifying the float precision used.

Parameters

- **obj** – The object to dump.
- **fid** – The file id to write to.
- **float_digits** – The number of digits of precision when writing floats out.
- ****params** – Additional parameters to pass to json.dumps.

easycv.utils.json_utils.**dumps**(*obj*, *float_digits=- 1*, ***params*)

Wrapper of json.dumps that allows specifying the float precision used.

Parameters

- **obj** – The object to dump.
- **float_digits** – The number of digits of precision when writing floats out.
- ****params** – Additional parameters to pass to json.dumps.

Returns JSON string representation of obj.

Return type output

easycv.utils.json_utils.**compat_dumps**(*data*, *float_digits=- 1*)

handle json dumps chinese and numpy data :param data python data structure: :param float_digits: The number of digits of precision when writing floats out.

Returns

json str, in python2 , the str is encoded with utf8 in python3, the str is unicode type(python3 str)

easycv.utils.json_utils.**PrettyParams**(***params*)

Returns parameters for use with Dump and Dumps to output pretty json.

Example usage: `` `json_str = json_utils.Dumps(obj, **json_utils.PrettyParams())` ``

```json_str = json_utils.Dumps(

obj, **json_utils.PrettyParams(allow_nans=False))```

**Parameters** **\*\*params** – Additional params to pass to json.dump or json.dumps.

**Returns**
```

Parameters that are compatible with json_utils.Dump and json_utils.Dumps.

Return type params

19.14 easycv.utils.logger module

easycv.utils.logger.**get_root_logger**(*log_file=None*, *log_level=20*)

Get the root logger.

The logger will be initialized if it has not been initialized. By default a StreamHandler will be added. If *log_file* is specified, a FileHandler will also be added. The name of the root logger is the top-level package name, e.g., "easycv".

Parameters

- **log_file** (*str | None*) – The log filename. If specified, a FileHandler will be added to the root logger.
- **log_level** (*int*) – The root logger level. Note that only the process of rank 0 is affected, while other processes will set the level to "Error" and be silent most of the time.

Returns The root logger.

Return type logging.Logger

easycv.utils.logger.**print_log**(*msg*, *logger=None*, *level=20*)

Print a log message.

Parameters

- **msg** (*str*) – The message to be logged.
- **logger** (*logging.Logger | str | None*) – The logger to be used. Some special loggers are: - "root": the root logger obtained with *get_root_logger()*. - "silent": no message will be printed. - None: The *print()* method will be used to print log messages.
- **level** (*int*) – Logging level. Only available when *logger* is a Logger object or "root".

19.15 easycv.utils.metric_distance module

easycv.utils.metric_distance.**LpDistance**(*query_emb*, *ref_emb*, *p=2*)

Input: query_emb: [n, dims] tensor ref_emb: [m, dims] tensor p : p normalize
Output: distance_matrix: [n, m] tensor
distance_matrix_i_j = (sigma_k(a_i_k**p - b_j_k**p))**(1/p)

easycv.utils.metric_distance.**DotproductSimilarity**(*query_emb*, *ref_emb*)

easycv.utils.metric_distance.**CosineSimilarity**(*query_emb*, *ref_emb*)

Input: query_emb: [n, dims] tensor ref_emb: [m, dims] tensor
Output: distance_matrix: [n, m] tensor

19.16 easycv.utils.misc module

easycv.utils.misc.**tensor2imgs**(*tensor*, *mean=(0, 0, 0)*, *std=(1, 1, 1)*, *to_rgb=True*)

easycv.utils.misc.**multi_apply**(*func*, **args*, ***kwargs*)

easycv.utils.misc.**unmap**(*data*, *count*, *inds*, *fill=0*)
Unmap a subset of item (data) back to the original set of items (of size count)

easycv.utils.misc.**add_prefix**(*inputs*, *prefix*)
Add prefix for dict key.

Parameters

- **inputs** (*dict*) – The input dict with str keys.
- **prefix** (*str*) – The prefix add to key name.

Returns The dict with keys wrapped with `prefix`.

Return type dict

19.17 easycv.utils.preprocess_function module

easycv.utils.preprocess_function.**bninceptionPre**(*image*, *mean=[104, 117, 128]*, *std=[1, 1, 1]*)

Parameters

- **image** – pytorch Image tensor from PIL (range 0~1), bgr format
- **mean** – norm mean
- **std** – norm val

Returns A image norm in 0~255, rgb format

easycv.utils.preprocess_function.**randomErasing**(*image*, *probability=0.5*, *sl=0.02*, *sh=0.2*, *r1=0.3*, *mean=[0.4914, 0.4822, 0.4465]*)

easycv.utils.preprocess_function.**solarize**(*tensor*, *threshold=0.5*, *apply_prob=0.2*)
tensor : pytorch tensor

easycv.utils.preprocess_function.**gaussianBlurDynamic**(*image*, *apply_prob=0.5*)

easycv.utils.preprocess_function.**gaussianBlur**(*image*, *kernel_size=22*, *apply_prob=0.5*)

easycv.utils.preprocess_function.**randomGrayScale**(*image*, *apply_prob=0.2*)

easycv.utils.preprocess_function.**mixUp**(*image*, *alpha=0.2*)

easycv.utils.preprocess_function.**mixUpCls**(*data*, *alpha=0.2*)

19.18 easycv.utils.profiling module

easycv.utils.profiling.**profile_time**(*trace_name*, *name*, *enabled=True*, *stream=None*, *end_stream=None*)
Print time spent by CPU and GPU.

Useful as a temporary context manager to find sweet spots of code suitable for async implementation.

easycv.utils.profiling.**benchmark_torch_function**(*iters*, *f*, **args*)

easycv.utils.profiling.**time_synchronized**()

19.19 easycv.utils.py_util module

easycv.utils.py_util.**copy_attr**(*a*, *b*, *include=()*, *exclude=()*)

easycv.utils.py_util.**get_parent_path**(*path: str*)
get parent path, support oss-style path

19.20 easycv.utils.registry module

class easycv.utils.registry.**Registry**(*name*)
Bases: object

__init__(*name*)
Initialize self. See help(type(self)) for accurate signature.

property name

property module_dict

get(*key*)

register_module(*cls=None*, *force=False*)

easycv.utils.registry.**build_from_cfg**(*cfg*, *registry*, *default_args=None*)
Build a module from config dict.

Parameters

- **cfg** (*dict*) – Config dict. It should at least contain the key "type".
- **registry** (*Registry*) – The registry to search the type from.
- **default_args** (*dict, optional*) – Default initialization arguments.

Returns The constructed object.

Return type obj

19.21 easycv.utils.test_util module

Contains functions which are convenient for unit testing.

easycv.utils.test_util.**get_tmp_dir**()

easycv.utils.test_util.**clear_all_tmp_dirs**()

easycv.utils.test_util.**replace_data_for_test**(*cfg*)
replace real data with test data
Parameters **cfg** – Config object

easycv.utils.test_util.**RunAsSubprocess**(*f*)

easycv.utils.test_util.**clean_up**(*test_dir*)

easycv.utils.test_util.**run_in_subprocess**(*cmd*)

easycv.utils.test_util.**dist_exec_wrapper**(*cmd*, *nproc_per_node*, *node_rank=0*, *nnodes=1*, *port='29527'*, *addr='127.0.0.1'*, *python_path=None*)
donot forget init dist in your function or script of cmd `python from mmcv.runner import init_dist init_dist(launcher='pytorch') `

easycv.utils.test_util.**is_port_used**(*port*, *host='127.0.0.1'*)

easycv.utils.test_util.**get_random_port**()

easycv.utils.test_util.**pseudo_dist_init**()

easycv.utils.test_util.**computeStats**(*backend*, *timings*, *batch_size=1*, *model_name='default'*)
compute the statistical metric of time and speed

easycv.utils.test_util.**benchmark**(*predictor*, *input_data_list*, *backend='BACKEND'*, *batch_size=1*, *model_name='default'*, *num=200*)
evaluate the time and speed of different models

19.22 easycv.utils.user_config_params_utils module

easycv.utils.user_config_params_utils.**check_value_type**(*replacement*, *original*)
convert replacement's type to original's type, support converting str to int or float or list or tuple

CHAPTER

TWENTY

EASYCV PACKAGE

20.1 Subpackages

20.1.1 easycv.file package

Submodules

easycv.file.base module

class easycv.file.base.**IOBase**

Bases: object

static register(*options*)

open(*path: str*, *mode: str = 'r'*)

exists(*path: str*) → bool

move(*src: str*, *dst: str*)

copy(*src: str*, *dst: str*)

copytree(*src: str*, *dst: str*)

makedirs(*path: str*, *exist_ok=True*)

remove(*path: str*)

rmtree(*path: str*)

listdir(*path: str*, *recursive=False*, *full_path=False*, *contains=None*)

isdir(*path: str*) → bool

isfile(*path: str*) → bool

abspath(*path: str*) → str

last_modified(*path: str*) → datetime.datetime

last_modified_str(*path: str*) → str

size(*path: str*) → int

md5(*path: str*) → str

re_remote = re.compile('(oss|https?)://')

islocal(*path: str*) → bool

is_writable(*path*)

class easycv.file.base.**IOLocal**

Bases: *easycv.file.base.IOBase*

open(*path*, *mode='r'*)

exists(*path*)

move(*src*, *dst*)

copy(*src*, *dst*)

copytree(*src*, *dst*)

makedirs(*path*, *exist_ok=True*)

remove(*path*)

rmtree(*path*)

listdir(*path*, *recursive=False*, *full_path=False*, *contains: Optional[Union[str, List[str]]] = None*)

isdir(*path*)

isfile(*path*)

glob(*path*)

abspath(*path*)

last_modified(*path*)

last_modified_str(*path*)

size(*path: str*) → int

easycv.file.file_io module

easycv.file.file_io.**set_oss_env**(*ak_id: str*, *ak_secret: str*, *hosts: Union[str, List[str]]*, *buckets: Union[str, List[str]]*)

class easycv.file.file_io.**IO**(*max_retry=10*)

Bases: *easycv.file.base.IOLocal*

IO module to support both local and oss io. If access oss file, you need to authorize OSS, please refer to *IO.access_oss*.

__init__(*max_retry=10*)

Initialize self. See help(type(self)) for accurate signature.

access_oss(*ak_id: str = ''*, *ak_secret: str = ''*, *hosts: Union[str, List[str]] = ''*, *buckets: Union[str, List[str]] = ''*)

If access oss file, you need to authorize OSS as follows:

Method1: from easycv.file import io io.access_oss(

ak_id='your_accesskey_id', ak_secret='your_accesskey_secret', hosts='your endpoint' or ['your endpoint1', 'your endpoint2'], buckets='your bucket' or ['your bucket1', 'your bucket2'])

Method2: Add oss config to your local file *~/.ossutilconfig*, as follows: More oss config information, please refer to: https://help.aliyun.com/document_detail/120072.html ``` [Credentials]

language = CH endpoint = your endpoint accessKeyID = your_accesskey_id accessKeySecret = your_accesskey_secret

[Bucket-Endpoint] bucket1 = endpoint1 bucket2 = endpoint2

``` Then run the following command, the config file will be read by default to authorize oss.

from easycv.file import io io.access_oss()

**open**(*full_path*, *mode='r'*)
Same usage as the python build-in *open*. Support local path and oss path.

### Example

from easycv.file import io io.access_oss(your oss config) # only oss file need, refer to *IO.access_oss* # Write something to a oss file. with io.open('oss://bucket_name/demo.txt', 'w') as f:

f.write("test")

# Read from a oss file. with io.open('oss://bucket_name/demo.txt', 'r') as f:

print(f.read())

**Parameters** **full_path** – absolute oss path

**exists**(*path*)
Whether the file exists, same usage as *os.path.exists*. Support local path and oss path.

### Example

from easycv.file import io io.access_oss(your oss config) # only oss file need, refer to *IO.access_oss* ret = io.exists('oss://bucket_name/dir') print(ret)

**Parameters** **path** – oss path or local path

**move**(*src*, *dst*)
Move src to dst, same usage as shutil.move. Support local path and oss path.

### Example

from easycv.file import io io.access_oss(your oss config) # only oss file need, refer to *IO.access_oss* # move oss file to local io.move('oss://bucket_name/file.txt', '/your/local/path/file.txt') # move oss file to oss io.move('oss://bucket_name/file.txt', 'oss://bucket_name/file.txt') # move local file to oss io.move('/your/local/file.txt', 'oss://bucket_name/file.txt') # move directory io.move('oss://bucket_name/dir1', 'oss://bucket_name/dir2')

**Parameters**

- **src** – oss path or local path
- **dst** – oss path or local path

**safe_copy**(*src*, *dst*, *try_max=3*)
oss always bug, we need safe_copy!

**copy**(*src*, *dst*)
Copy a file from src to dst. Same usage as *shutil.copyfile*. If you want to copy a directory, please use *easycv.io.copytree*
```

Example

from easycv.file import io io.access_oss(your oss config) # only oss file need, refer to *IO.access_oss* # Copy a file from local to oss: io.copy('/your/local/file.txt', 'oss://bucket/dir/file.txt')

Copy a oss file to local: io.copy('oss://bucket/dir/file.txt', '/your/local/file.txt')

Copy a file from oss to oss:: io.copy('oss://bucket/dir/file.txt', 'oss://bucket/dir/file2.txt')

Parameters

- **src** – oss path or local path
- **dst** – oss path or local path

copytree(*src*, *dst*)

Copy files recursively from src to dst. Same usage as *shutil.copytree*. If you want to copy a file, please use *easycv.io.copy*.

Example

from easycv.file import io io.access_oss(your oss config) # only oss file need, refer to *IO.access_oss* # copy files from local to oss io.copytree(src='/your/local/dir1', dst='oss://bucket_name/dir2') # copy files from oss to local io.copytree(src='oss://bucket_name/dir2', dst='/your/local/dir1') # copy files from oss to oss io.copytree(src='oss://bucket_name/dir1', dst='oss://bucket_name/dir2')

Parameters

- **src** – oss path or local path
- **dst** – oss path or local path

listdir(*path*, *recursive=False*, *full_path=False*, *contains: Optional[Union[str, List[str]]] = None*)

List all objects in path. Same usage as *os.listdir*.

Example

from easycv.file import io io.access_oss(your oss config) # only oss file need, refer to *IO.access_oss* ret = io.listdir('oss://bucket/dir', recursive=True) print(ret)

Parameters

- **path** – local file path or oss path.
- **recursive** – If False, only list the top level objects. If True, recursively list all objects.
- **full_path** – if full path, return files with path prefix.
- **contains** – substr to filter list files.

return: A list of path.

remove(*path*)

Remove a file or a directory recursively. Same usage as *os.remove* or *shutil.rmtree*.

Example

from easycv.file import io io.access_oss(your oss config) # only oss file need, refer to *IO.access_oss* # Remove a oss file io.remove('oss://bucket_name/file.txt')

Remove a oss directory io.remove('oss://bucket_name/dir/')

Parameters **path** – local or oss path, file or directory

rmtree(*path*)
Remove directory recursively, same usage as *shutil.rmtree*

Example

from easycv.file import io io.access_oss(your oss config) # only oss file need, refer to *IO.access_oss* io.remove('oss://bucket_name/dir_name') # Or io.remove('oss://bucket_name/dir_name/')

Parameters **path** – oss path

makedirs(*path*, *exist_ok=True*)
Create directories recursively, same usage as *os.makedirs*

Example

from easycv.file import io io.access_oss(your oss config) # only oss file need, refer to *IO.access_oss* io.makedirs('oss://bucket/new_dir/')

Parameters **path** – local or oss dir path

isdir(*path*)
Return whether a path is directory, same usage as *os.path.isdir*

Example

from easycv.file import io io.access_oss(your oss config) # only oss file need, refer to *IO.access_oss* io.isdir('oss://bucket/dir/')

Parameters **path** – local or oss path

Return: bool, True or False.

isfile(*path*)
Return whether a path is file object, same usage as *os.path.isfile*

Example

from easycv.file import io io.access_oss(your oss config) # only oss file need, refer to *IO.access_oss* io.isfile('oss://bucket/file.txt')

Parameters **path** – local or oss path

Return: bool, True or False.

glob(*file_path*)
Return a list of paths matching a pathname pattern. .. rubric:: Example

from easycv.file import io io.access_oss(your oss config) # only oss file need, refer to *IO.access_oss* io.glob('oss://bucket/dir/*.txt')

Parameters **path** – local or oss file pattern

Return: list, a list of paths.

abspath(*path*)

authorize(*path*)

last_modified(*path*)

last_modified_str(*path*)

size(*path: str*) → int
Get the size of file path, same usage as *os.path.getsize*

Example

from easycv.file import io io.access_oss(your oss config) # only oss file need, refer to *IO.access_oss* size = io.size('oss://bucket/file.txt') print(size)

Parameters **path** – local or oss path.

Return: size of file in bytes

class easycv.file.file_io.**OSSFile**(*bucket*, *path*, *position=0*)
Bases: object

__init__(*bucket*, *path*, *position=0*)
Initialize self. See help(type(self)) for accurate signature.

write(*content*)

flush(*retry=0*)

close()

seek(*position*)

class easycv.file.file_io.**BinaryOSSFile**(*bucket*, *path*)
Bases: object

__init__(*bucket*, *path*)
Initialize self. See help(type(self)) for accurate signature.

class easycv.file.file_io.**NullContextWrapper**(*obj*)
Bases: object

__init__(*obj*)
Initialize self. See help(type(self)) for accurate signature.

easycv.file.utils module

easycv.file.utils.**create_namedtuple**(***kwargs*)

easycv.file.utils.**is_oss_path**(*s*)

easycv.file.utils.**is_url_path**(*s*)

easycv.file.utils.**url_path_exists**(*url*)

easycv.file.utils.**get_oss_config**()
Get oss config file from env *OSS_CONFIG_FILE*, default file is *~/.ossutilconfig*.

easycv.file.utils.**oss_progress**(*desc*)

easycv.file.utils.**ignore_oss_error**(*msg=''*)

easycv.file.utils.**mute_stderr**()

20.1.2 easycv.runner package

Submodules

easycv.runner.ev_runner module

class easycv.runner.ev_runner.**EVRunner**(*model*, *batch_processor=None*, *optimizer=None*, *work_dir=None*, *logger=None*, *meta=None*, *fp16_enable=False*)

Bases: mmcv.runner.epoch_based_runner.EpochBasedRunner

__init__(*model*, *batch_processor=None*, *optimizer=None*, *work_dir=None*, *logger=None*, *meta=None*, *fp16_enable=False*)

Epoch Runner for easycv, add support for oss IO and file sync.

Parameters

- **model** (torch.nn.Module) – The model to be run.
- **batch_processor** (*callable*) – A callable method that process a data batch. The interface of this method should be *batch_processor(model, data, train_mode) -> dict*
- **optimizer** (dict or torch.optim.Optimizer) – It can be either an optimizer (in most cases) or a dict of optimizers (in models that requires more than one optimizer, e.g., GAN).
- **work_dir** (*str, optional*) – The working directory to save checkpoints and logs. Defaults to None.
- **logger** (logging.Logger) – Logger used during training. Defaults to None. (The default value is just for backward compatibility)
- **meta** (*dict | None*) – A dict records some import information such as environment info and seed, which will be logged in logger hook. Defaults to None.
- **fp16_enable** (*bool*) – if use fp16

run_iter(*data_batch*, *train_mode*, ***kwargs*)

process for each iteration.

Parameters

- **data_batch** – Batch of dict of data.
- **train_model** (*bool*) – If set True, run training step else validation step.

train(*data_loader*, ***kwargs*)

Training process for one epoch which will iterate through all training data and call hooks at different stages.

Parameters **data_loader** – data loader object for training

val(*data_loader*, ***kwargs*)

Validation step which Deprecated, using evaluation hook instead.

save_checkpoint(*out_dir*, *filename_tmpl='epoch_{}.pth'*, *save_optimizer=True*, *meta=None*, *create_symlink=True*)

Save checkpoint to file.

Parameters

- **out_dir** – Directory where checkpoint files are to be saved.
- **filename_tmpl** (*str, optional*) – Checkpoint filename pattern.
- **save_optimizer** (*bool, optional*) – save optimizer state.
- **meta** (*dict, optional*) – Metadata to be saved in checkpoint.

`current_lr()`
Get current learning rates.

Returns

Current learning rates of all param groups. If the runner has a dict of optimizers, this method will return a dict.

Return type list[float] | dict[str, list[float]]

`load_checkpoint(filename, map_location=device(type='cpu'), strict=False, logger=None)`
Load checkpoint from a file or URL.

Parameters

- **filename** (*str*) – Accept local filepath, URL, `torchvision://xxx`, `open-mmlab://xxx`, `oss://xxx`. Please refer to `docs/source/model_zoo.md` for details.
- **map_location** (*str*) – Same as `torch.load()`.
- **strict** (*bool*) – Whether to allow different params for the model and checkpoint.
- **logger** (`logging.Logger` or None) – The logger for error message.

Returns The loaded checkpoint.

Return type dict or OrderedDict

`resume(checkpoint, resume_optimizer=True, map_location='default')`
Resume state dict from checkpoint.

Parameters

- **checkpoint** – Checkpoint path
- **resume_optimizer** – Whether to resume optimizer state
- **map_location** (*str*) – Same as `torch.load()`.

20.1.3 easycv.toolkit package

Subpackages

easycv.toolkit.prune package

Submodules

easycv.toolkit.prune.prune_utils module

easycv.toolkit.quantize package

20.2 Submodules

20.3 easycv.version module

CHAPTER

TWENTYONE

EASYCV

CHAPTER

TWENTYTWO

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

e

INDEX

Symbols

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

R

S

T

U

V

W

X

Y