#conversions of all the dataset into coco format or middle format,

1. Here I will be using coco format

import os

import json

# Define the paths to your image and annotation directories

image\_dir = "/content/drive/MyDrive/kitti\_dataset/raw/image\_2"

annotation\_dir = "/content/drive/MyDrive/kitti\_dataset/raw/label\_2"

# Initialize the COCO annotation dictionary

coco\_data = {

    "info": {},

    "licenses": [],

    "images": [],

    "annotations": [],

    "categories": []

}

# Define the categories in your dataset

categories = [

    {"id": 0, "name": "car"},

    {"id": 1, "name": "Van"},

    {"id": 2, "name": "Truck"},

    {"id": 3, "name": "Pedestrian"},

    {"id": 4, "name": "Person\_sitting"},

    {"id": 5, "name": "Cyclist"},

    {"id": 6, "name": "Tram"},

    {"id": 7, "name": "Misc"}

]

# Add categories to the COCO data

coco\_data["categories"] = categories

# Iterate over the annotation files

for filename in os.listdir(annotation\_dir):

    if filename.endswith(".json"):

        annotation\_path = os.path.join(annotation\_dir, filename)

        image\_path = os.path.join(image\_dir, os.path.splitext(filename)[0] + ".jpg")

        # Load annotation file

        with open(annotation\_path, "r") as file:

            annotation = json.load(file)

        # Exclude "label don't care" annotations

        if "label" in annotation and annotation["label"] == "dontcare":

            continue

        # Create COCO image entry

        image\_id = len(coco\_data["images"]) + 1

        image\_entry = {

            "id": image\_id,

            "width": annotation["size"][1],

            "height": annotation["size"][0],

            "file\_name": os.path.basename(image\_path)

        }

        coco\_data["images"].append(image\_entry)

        # Create COCO annotation entry

        annotation\_entry = {

            "id": image\_id,  # Use the same ID as the image

            "image\_id": image\_id,

            "category\_id": annotation["class\_ids"][0],

            "segmentation": [],  # Replace with the actual segmentation data if available

            "area": (annotation["bboxes"][0][2] - annotation["bboxes"][0][0]) \* (annotation["bboxes"][0][3] - annotation["bboxes"][0][1]),

            "bbox": annotation["bboxes"][0],

            "iscrowd": 0

        }

        coco\_data["annotations"].append(annotation\_entry)

# Save COCO data to a JSON file

coco\_output\_path = "/content/drive/MyDrive/kitti\_dataset/coco\_annotations.json"

with open(coco\_output\_path, "w") as output\_file:

    json.dump(coco\_data, output\_file)

the output will be a single annotation file in this case, it takes input as image directory and corresponding image annotation files.

1. Importing the nessecary libraries for mmdetection

!pip3 install openmim

!mim install mmengine

!mim install "mmcv>=2.0.0,<2.1.0"

1. Cloning the respository

#@title clone repository

!git clone https://github.com/open-mmlab/mmdetection.git

%cd mmdetection

!pip install -e .

1. Checking the version should be 3.0.0

import mmdet

print(mmdet.\_\_version\_\_)

# Example output: 3.0.0, or an another version.

* Very.Imp step, changing the classes according to our dataset of the file in this path “mmdetection/mmdet/datasets/coco.py”

I have changed this according to my dataset,

As my dataset only contains 8 classes.

# Copyright (c) OpenMMLab. All rights reserved.

import copy

import os.path as osp

from typing import List, Union

from mmengine.fileio import get\_local\_path

from mmdet.registry import DATASETS

from .api\_wrappers import COCO

from .base\_det\_dataset import BaseDetDataset

@DATASETS.register\_module()

class CocoDataset(BaseDetDataset):

    """Dataset for COCO."""

    METAINFO = {

        'classes':

        ('Car', 'Van', 'Truck', 'Pedestrian', 'Person\_sitting', 'Cyclist', 'Tram', 'Misc')

    }

    COCOAPI = COCO

    # ann\_id is unique in coco dataset.

    ANN\_ID\_UNIQUE = True

    def load\_data\_list(self) -> List[dict]:

        """Load annotations from an annotation file named as ``self.ann\_file``

        Returns:

            List[dict]: A list of annotation.

        """  # noqa: E501

        with get\_local\_path(

                self.ann\_file, backend\_args=self.backend\_args) as local\_path:

            self.coco = self.COCOAPI(local\_path)

        # The order of returned `cat\_ids` will not

        # change with the order of the `classes`

        self.cat\_ids = self.coco.get\_cat\_ids(

            cat\_names=self.metainfo['classes'])

        self.cat2label = {cat\_id: i for i, cat\_id in enumerate(self.cat\_ids)}

        self.cat\_img\_map = copy.deepcopy(self.coco.cat\_img\_map)

        img\_ids = self.coco.get\_img\_ids()

        data\_list = []

        total\_ann\_ids = []

        for img\_id in img\_ids:

            raw\_img\_info = self.coco.load\_imgs([img\_id])[0]

            raw\_img\_info['img\_id'] = img\_id

            ann\_ids = self.coco.get\_ann\_ids(img\_ids=[img\_id])

            raw\_ann\_info = self.coco.load\_anns(ann\_ids)

            total\_ann\_ids.extend(ann\_ids)

            parsed\_data\_info = self.parse\_data\_info({

                'raw\_ann\_info':

                raw\_ann\_info,

                'raw\_img\_info':

                raw\_img\_info

            })

            data\_list.append(parsed\_data\_info)

        if self.ANN\_ID\_UNIQUE:

            assert len(set(total\_ann\_ids)) == len(

                total\_ann\_ids

            ), f"Annotation ids in '{self.ann\_file}' are not unique!"

        del self.coco

        return data\_list

    def parse\_data\_info(self, raw\_data\_info: dict) -> Union[dict, List[dict]]:

        """Parse raw annotation to target format.

        Args:

            raw\_data\_info (dict): Raw data information load from ``ann\_file``

        Returns:

            Union[dict, List[dict]]: Parsed annotation.

        """

        img\_info = raw\_data\_info['raw\_img\_info']

        ann\_info = raw\_data\_info['raw\_ann\_info']

        data\_info = {}

        # TODO: need to change data\_prefix['img'] to data\_prefix['img\_path']

        img\_path = osp.join(self.data\_prefix['img'], img\_info['file\_name'])

        if self.data\_prefix.get('seg', None):

            seg\_map\_path = osp.join(

                self.data\_prefix['seg'],

                img\_info['file\_name'].rsplit('.', 1)[0] + self.seg\_map\_suffix)

        else:

            seg\_map\_path = None

        data\_info['img\_path'] = img\_path

        data\_info['img\_id'] = img\_info['img\_id']

        data\_info['seg\_map\_path'] = seg\_map\_path

        data\_info['height'] = img\_info['height']

        data\_info['width'] = img\_info['width']

        instances = []

        for i, ann in enumerate(ann\_info):

            instance = {}

            if ann.get('ignore', False):

                continue

            x1, y1, w, h = ann['bbox']

            inter\_w = max(0, min(x1 + w, img\_info['width']) - max(x1, 0))

            inter\_h = max(0, min(y1 + h, img\_info['height']) - max(y1, 0))

            if inter\_w \* inter\_h == 0:

                continue

            if ann['area'] <= 0 or w < 1 or h < 1:

                continue

            if ann['category\_id'] not in self.cat\_ids:

                continue

            bbox = [x1, y1, x1 + w, y1 + h]

            if ann.get('iscrowd', False):

                instance['ignore\_flag'] = 1

            else:

                instance['ignore\_flag'] = 0

            instance['bbox'] = bbox

            instance['bbox\_label'] = self.cat2label[ann['category\_id']]

            if ann.get('segmentation', None):

                instance['mask'] = ann['segmentation']

            instances.append(instance)

        data\_info['instances'] = instances

        return data\_info

    def filter\_data(self) -> List[dict]:

        """Filter annotations according to filter\_cfg.

        Returns:

            List[dict]: Filtered results.

        """

        if self.test\_mode:

            return self.data\_list

        if self.filter\_cfg is None:

            return self.data\_list

        filter\_empty\_gt = self.filter\_cfg.get('filter\_empty\_gt', False)

        min\_size = self.filter\_cfg.get('min\_size', 0)

        # obtain images that contain annotation

        ids\_with\_ann = set(data\_info['img\_id'] for data\_info in self.data\_list)

        # obtain images that contain annotations of the required categories

        ids\_in\_cat = set()

        for i, class\_id in enumerate(self.cat\_ids):

            ids\_in\_cat |= set(self.cat\_img\_map[class\_id])

        # merge the image id sets of the two conditions and use the merged set

        # to filter out images if self.filter\_empty\_gt=True

        ids\_in\_cat &= ids\_with\_ann

        valid\_data\_infos = []

        for i, data\_info in enumerate(self.data\_list):

            img\_id = data\_info['img\_id']

            width = data\_info['width']

            height = data\_info['height']

            if filter\_empty\_gt and img\_id not in ids\_in\_cat:

                continue

            if min(width, height) >= min\_size:

                valid\_data\_infos.append(data\_info)

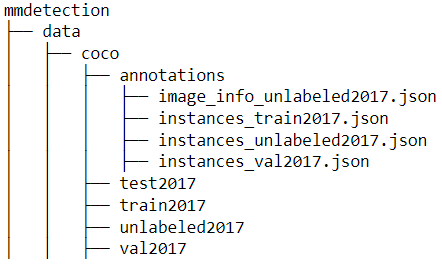
        return valid\_data\_infos

since I converted the dataset to coco format fore hand, I proceeded with the upcoming steps, if you haven’t converted to coco format, you could use this py file and give your dataset path and convert to coco format (in built, which I don’t suggest but you could) .

1. Identify the config file required in mmdetection/config

Here my task is “SOFT TEACHER ON CUSTOM DATASET”

1. Kept my dataset directory in the following order as order plays a very important role in these kinda repository referenced training.



Here, annotation folder will contain single-filed annotation that is the collection of all the annotations of the images. (train annotation and valid annotation)

Below is how I replicated this



Train and val folder are folders containing images, train contains around 500 images and val contains around 6891 images.

1. Make sure you move this whole “DATA directory” to the mmdetection (imported from git) folder, this is nessecary as we set path to “root directory” in the config and it will take the “pwd” so if all are in one directory it will be convenient.



You can see here I have moved it to mmdetection.

1. Since soft teacher is a semi supervised learning, there is a config file called “semi\_coco\_detection.py” in “mmdetection/configs/\_base\_/datasets”

# dataset settings

dataset\_type = 'CocoDataset'

data\_root = 'data/coco/'

# Example to use different file client

# Method 1: simply set the data root and let the file I/O module

# automatically infer from prefix (not support LMDB and Memcache yet)

# data\_root = 's3://openmmlab/datasets/detection/coco/'

# Method 2: Use `backend\_args`, `file\_client\_args` in versions before 3.0.0rc6

# backend\_args = dict(

#     backend='petrel',

#     path\_mapping=dict({

#         './data/': 's3://openmmlab/datasets/detection/',

#         'data/': 's3://openmmlab/datasets/detection/'

#     }))

backend\_args = None

color\_space = [

    [dict(type='ColorTransform')],

    [dict(type='AutoContrast')],

    [dict(type='Equalize')],

    [dict(type='Sharpness')],

    [dict(type='Posterize')],

    [dict(type='Solarize')],

    [dict(type='Color')],

    [dict(type='Contrast')],

    [dict(type='Brightness')],

]

geometric = [

    [dict(type='Rotate')],

    [dict(type='ShearX')],

    [dict(type='ShearY')],

    [dict(type='TranslateX')],

    [dict(type='TranslateY')],

]

scale = [(1333, 400), (1333, 1200)]

branch\_field = ['sup', 'unsup\_teacher', 'unsup\_student']

# pipeline used to augment labeled data,

# which will be sent to student model for supervised training.

sup\_pipeline = [

    dict(type='LoadImageFromFile', backend\_args=backend\_args),

    dict(type='LoadAnnotations', with\_bbox=True),

    dict(type='RandomResize', scale=scale, keep\_ratio=True),

    dict(type='RandomFlip', prob=0.5),

    dict(type='RandAugment', aug\_space=color\_space, aug\_num=1),

    dict(type='FilterAnnotations', min\_gt\_bbox\_wh=(1e-2, 1e-2)),

    dict(

        type='MultiBranch',

        branch\_field=branch\_field,

        sup=dict(type='PackDetInputs'))

]

# pipeline used to augment unlabeled data weakly,

# which will be sent to teacher model for predicting pseudo instances.

weak\_pipeline = [

    dict(type='RandomResize', scale=scale, keep\_ratio=True),

    dict(type='RandomFlip', prob=0.5),

    dict(

        type='PackDetInputs',

        meta\_keys=('img\_id', 'img\_path', 'ori\_shape', 'img\_shape',

                   'scale\_factor', 'flip', 'flip\_direction',

                   'homography\_matrix')),

]

# pipeline used to augment unlabeled data strongly,

# which will be sent to student model for unsupervised training.

strong\_pipeline = [

    dict(type='RandomResize', scale=scale, keep\_ratio=True),

    dict(type='RandomFlip', prob=0.5),

    dict(

        type='RandomOrder',

        transforms=[

            dict(type='RandAugment', aug\_space=color\_space, aug\_num=1),

            dict(type='RandAugment', aug\_space=geometric, aug\_num=1),

        ]),

    dict(type='RandomErasing', n\_patches=(1, 5), ratio=(0, 0.2)),

    dict(type='FilterAnnotations', min\_gt\_bbox\_wh=(1e-2, 1e-2)),

    dict(

        type='PackDetInputs',

        meta\_keys=('img\_id', 'img\_path', 'ori\_shape', 'img\_shape',

                   'scale\_factor', 'flip', 'flip\_direction',

                   'homography\_matrix')),

]

# pipeline used to augment unlabeled data into different views

unsup\_pipeline = [

    dict(type='LoadImageFromFile', backend\_args=backend\_args),

    dict(type='LoadEmptyAnnotations'),

    dict(

        type='MultiBranch',

        branch\_field=branch\_field,

        unsup\_teacher=weak\_pipeline,

        unsup\_student=strong\_pipeline,

    )

]

test\_pipeline = [

    dict(type='LoadImageFromFile', backend\_args=backend\_args),

    dict(type='Resize', scale=(1333, 800), keep\_ratio=True),

    dict(

        type='PackDetInputs',

        meta\_keys=('img\_id', 'img\_path', 'ori\_shape', 'img\_shape',

                   'scale\_factor'))

]

batch\_size = 5

num\_workers = 5

# There are two common semi-supervised learning settings on the coco dataset：

# (1) Divide the train2017 into labeled and unlabeled datasets

# by a fixed percentage, such as 1%, 2%, 5% and 10%.

# The format of labeled\_ann\_file and unlabeled\_ann\_file are

# instances\_train2017.{fold}@{percent}.json, and

# instances\_train2017.{fold}@{percent}-unlabeled.json

# `fold` is used for cross-validation, and `percent` represents

# the proportion of labeled data in the train2017.

# (2) Choose the train2017 as the labeled dataset

# and unlabeled2017 as the unlabeled dataset.

# The labeled\_ann\_file and unlabeled\_ann\_file are

# instances\_train2017.json and image\_info\_unlabeled2017.json

# We use this configuration by default.

labeled\_dataset = dict(

    type=dataset\_type,

    data\_root=data\_root,

    ann\_file='annotations/coco\_annotations.json',

    data\_prefix=dict(img='train/'),

    filter\_cfg=dict(filter\_empty\_gt=True, min\_size=32),

    pipeline=sup\_pipeline,

    backend\_args=backend\_args)

unlabeled\_dataset = dict(

    type=dataset\_type,

    data\_root=data\_root,

    ann\_file='annotations/val\_coco\_annotations.json',

    data\_prefix=dict(img='val/'),

    filter\_cfg=dict(filter\_empty\_gt=False),

    pipeline=unsup\_pipeline,

    backend\_args=backend\_args)

train\_dataloader = dict(

    batch\_size=batch\_size,

    num\_workers=num\_workers,

    persistent\_workers=True,

    sampler=dict(

        type='GroupMultiSourceSampler',

        batch\_size=batch\_size,

        source\_ratio=[1, 4]),

    dataset=dict(

        type='ConcatDataset', datasets=[labeled\_dataset, unlabeled\_dataset]))

val\_dataloader = dict(

    batch\_size=1,

    num\_workers=2,

    persistent\_workers=True,

    drop\_last=False,

    sampler=dict(type='DefaultSampler', shuffle=False),

    dataset=dict(

        type=dataset\_type,

        data\_root=data\_root,

        ann\_file='annotations/val\_coco\_annotations.json',

        data\_prefix=dict(img='val/'),

        test\_mode=True,

        pipeline=test\_pipeline,

        backend\_args=backend\_args))

test\_dataloader = val\_dataloader

val\_evaluator = dict(

    type='CocoMetric',

    ann\_file=data\_root + 'annotations/val\_coco\_annotations.json',

    metric='bbox',

    format\_only=False,

    backend\_args=backend\_args)

test\_evaluator = val\_evaluator

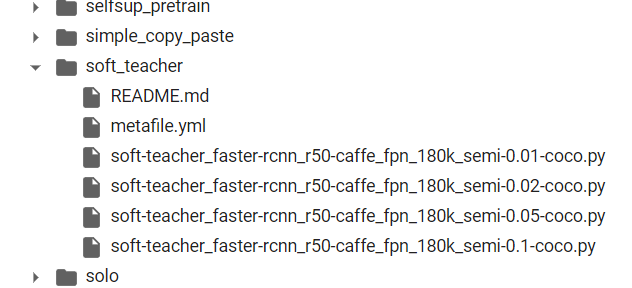
this file has the entire pipeline for soft-teacher(semi supervised training).

You can change the path of annotation files and images according to your directory/folder names.

* As you can see we have used “data\_root” here
* Metrics we use here is cocometric, (MAP is our main concern)

1. After this, we will dig into our soft-teacher.py config files from the config , “mmdetection/configs/soft\_teacher”

Softteacher will have 4 files, first 3 are supporting files, we will deal with the last file which is for changing configuration according to our perspective or task or objective.



We will open the last file

“mmdetection/configs/soft\_teacher/soft-teacher\_faster-rcnn\_r50-caffe\_fpn\_180k\_semi-0.1-coco.py”

1. The file shows the abstraction of the architecture of soft-teacher, it uses faster RCNN and retina NET as their backbone and feature extraction nets.

* We will modify this file according to our convenience,

1. Modify paths to our dataset (basic and enough for learning)

2. change the type epoch and optimizer. (not recommended)

I have modified according to my dataset path below.

\_base\_ = [

    '../\_base\_/models/faster-rcnn\_r50\_fpn.py', '../\_base\_/default\_runtime.py',

    '../\_base\_/datasets/semi\_coco\_detection.py'

]

detector = \_base\_.model

detector.data\_preprocessor = dict(

    type='DetDataPreprocessor',

    mean=[103.530, 116.280, 123.675],

    std=[1.0, 1.0, 1.0],

    bgr\_to\_rgb=False,

    pad\_size\_divisor=32)

detector.backbone = dict(

    type='ResNet',

    depth=50,

    num\_stages=4,

    out\_indices=(0, 1, 2, 3),

    frozen\_stages=1,

    norm\_cfg=dict(type='BN', requires\_grad=False),

    norm\_eval=True,

    style='caffe',

    init\_cfg=dict(

        type='Pretrained',

        checkpoint='open-mmlab://detectron2/resnet50\_caffe'))

model = dict(

    \_delete\_=True,

    type='SoftTeacher',

    detector=detector,

    data\_preprocessor=dict(

        type='MultiBranchDataPreprocessor',

        data\_preprocessor=detector.data\_preprocessor),

    semi\_train\_cfg=dict(

        freeze\_teacher=True,

        sup\_weight=1.0,

        unsup\_weight=4.0,

        pseudo\_label\_initial\_score\_thr=0.5,

        rpn\_pseudo\_thr=0.9,

        cls\_pseudo\_thr=0.9,

        reg\_pseudo\_thr=0.02,

        jitter\_times=10,

        jitter\_scale=0.06,

        min\_pseudo\_bbox\_wh=(1e-2, 1e-2)),

    semi\_test\_cfg=dict(predict\_on='teacher'))

# 10% coco train2017 is set as labeled dataset

labeled\_dataset = \_base\_.labeled\_dataset

unlabeled\_dataset = \_base\_.unlabeled\_dataset

labeled\_dataset.ann\_file = 'annotations/coco\_annotations.json'

unlabeled\_dataset.ann\_file = 'annotations/val\_coco\_annotations.json'

labeled\_dataset.data\_prefix = dict(img='train/')

unlabeled\_dataset.data\_prefix = dict(img='val/')

train\_dataloader = dict(

    dataset=dict(datasets=[labeled\_dataset, unlabeled\_dataset]))

# training schedule for 180k

train\_cfg = dict(

    type='IterBasedTrainLoop', max\_iters=180000, val\_interval=5000)

val\_cfg = dict(type='TeacherStudentValLoop')

test\_cfg = dict(type='TestLoop')

# learning rate policy

param\_scheduler = [

    dict(

        type='LinearLR', start\_factor=0.001, by\_epoch=False, begin=0, end=500),

    dict(

        type='MultiStepLR',

        begin=0,

        end=180000,

        by\_epoch=False,

        milestones=[120000, 160000],

        gamma=0.1)

]

# optimizer

optim\_wrapper = dict(

    type='OptimWrapper',

    optimizer=dict(type='SGD', lr=0.01, momentum=0.9, weight\_decay=0.0001))

default\_hooks = dict(

    checkpoint=dict(by\_epoch=False, interval=10000, max\_keep\_ckpts=2))

log\_processor = dict(by\_epoch=False)

custom\_hooks = [dict(type='MeanTeacherHook')]

\*modification if required by the user\*

I found out that training will require more time, storage and resources. Hence I had to checkpoint every 1500 interations, if yours is a powerful machine, you can make more iterations.

Here is my modified code,

# Copyright (c) OpenMMLab. All rights reserved.

**import argparse**

**import logging**

**import os**

**import os.path as osp**

**from mmengine.config import Config, DictAction**

**from mmengine.logging import print\_log**

**from mmengine.registry import RUNNERS**

**from mmengine.runner import Runner**

**from mmdet.utils import setup\_cache\_size\_limit\_of\_dynamo**

**def parse\_args():**

**parser = argparse.ArgumentParser(description='Train a detector')**

**parser.add\_argument('config', help='train config file path')**

**parser.add\_argument('--work-dir', help='the dir to save logs and models')**

**parser.add\_argument(**

**'--amp',**

**action='store\_true',**

**default=False,**

**help='enable automatic-mixed-precision training')**

**parser.add\_argument(**

**'--auto-scale-lr',**

**action='store\_true',**

**help='enable automatically scaling LR.')**

**parser.add\_argument(**

**'--resume',**

**nargs='?',**

**type=str,**

**const='auto',**

**help='If specify checkpoint path, resume from it, while if not '**

**'specify, try to auto resume from the latest checkpoint '**

**'in the work directory.')**

**parser.add\_argument(**

**'--cfg-options',**

**nargs='+',**

**action=DictAction,**

**help='override some settings in the used config, the key-value pair '**

**'in xxx=yyy format will be merged into config file. If the value to '**

**'be overwritten is a list, it should be like key="[a,b]" or key=a,b '**

**'It also allows nested list/tuple values, e.g. key="[(a,b),(c,d)]" '**

**'Note that the quotation marks are necessary and that no white space '**

**'is allowed.')**

**parser.add\_argument(**

**'--launcher',**

**choices=['none', 'pytorch', 'slurm', 'mpi'],**

**default='none',**

**help='job launcher')**

**# When using PyTorch version >= 2.0.0, the `torch.distributed.launch`**

**# will pass the `--local-rank` parameter to `tools/train.py` instead**

**# of `--local\_rank`.**

**parser.add\_argument('--local\_rank', '--local-rank', type=int, default=0)**

**args = parser.parse\_args()**

**if 'LOCAL\_RANK' not in os.environ:**

**os.environ['LOCAL\_RANK'] = str(args.local\_rank)**

**return args**

**def main():**

**args = parse\_args()**

**# Reduce the number of repeated compilations and improve**

**# training speed.**

**setup\_cache\_size\_limit\_of\_dynamo()**

**# load config**

**cfg = Config.fromfile(args.config)**

**cfg.launcher = args.launcher**

**if args.cfg\_options is not None:**

**cfg.merge\_from\_dict(args.cfg\_options)**

**# work\_dir is determined in this priority: CLI > segment in file > filename**

**if args.work\_dir is not None:**

**# update configs according to CLI args if args.work\_dir is not None**

**cfg.work\_dir = args.work\_dir**

**elif cfg.get('work\_dir', None) is None:**

**# use config filename as default work\_dir if cfg.work\_dir is None**

**cfg.work\_dir = osp.join('./work\_dirs',**

**osp.splitext(osp.basename(args.config))[0])**

**# enable automatic-mixed-precision training**

**if args.amp is True:**

**optim\_wrapper = cfg.optim\_wrapper.type**

**if optim\_wrapper == 'AmpOptimWrapper':**

**print\_log(**

**'AMP training is already enabled in your config.',**

**logger='current',**

**level=logging.WARNING)**

**else:**

**assert optim\_wrapper == 'OptimWrapper', (**

**'`--amp` is only supported when the optimizer wrapper type is '**

**f'`OptimWrapper` but got {optim\_wrapper}.')**

**cfg.optim\_wrapper.type = 'AmpOptimWrapper'**

**cfg.optim\_wrapper.loss\_scale = 'dynamic'**

**# enable automatically scaling LR**

**if args.auto\_scale\_lr:**

**if 'auto\_scale\_lr' in cfg and \**

**'enable' in cfg.auto\_scale\_lr and \**

**'base\_batch\_size' in cfg.auto\_scale\_lr:**

**cfg.auto\_scale\_lr.enable = True**

**else:**

**raise RuntimeError('Can not find "auto\_scale\_lr" or '**

**'"auto\_scale\_lr.enable" or '**

**'"auto\_scale\_lr.base\_batch\_size" in your'**

**' configuration file.')**

**# resume is determined in this priority: resume from > auto\_resume**

**if args.resume == 'auto':**

**cfg.resume = True**

**cfg.load\_from = None**

**elif args.resume is not None:**

**cfg.resume = True**

**cfg.load\_from = args.resume**

**# build the runner from config**

**if 'runner\_type' not in cfg:**

**# build the default runner**

**runner = Runner.from\_cfg(cfg)**

**else:**

**# build customized runner from the registry**

**# if 'runner\_type' is set in the cfg**

**runner = RUNNERS.build(cfg)**

**save\_interval = 1500 # Save a checkpoint every 1500 iterations**

**checkpoint\_dir = "/content/drive/MyDrive/checkpoints" # Specify the desired directory to save checkpoints**

**# Create the checkpoint directory if it doesn't exist**

**os.makedirs(checkpoint\_dir, exist\_ok=True)**

**for epoch in range(runner.max\_epochs):**

**runner.train()**

**# Save checkpoint at specified intervals**

**if (epoch + 1) % save\_interval == 0:**

**checkpoint\_path = osp.join(checkpoint\_dir, f'epoch\_{epoch + 1}.pth')**

**runner.save\_checkpoint(checkpoint\_path)**

**if \_\_name\_\_ == '\_\_main\_\_':**

**main()**

1. Now everythings ready to start the training

!python tools/train.py configs/soft\_teacher/soft-teacher\_faster-rcnn\_r50-caffe\_fpn\_180k\_semi-0.1-coco.py

06/20 12:39:58 - mmengine - INFO -

------------------------------------------------------------

System environment:

sys.platform: linux

Python: 3.10.12 (main, Jun 7 2023, 12:45:35) [GCC 9.4.0]

CUDA available: False

numpy\_random\_seed: 1525478322

GCC: x86\_64-linux-gnu-gcc (Ubuntu 9.4.0-1ubuntu1~20.04.1) 9.4.0

PyTorch: 2.0.1+cu118

PyTorch compiling details: PyTorch built with:

- GCC 9.3

- C++ Version: 201703

- Intel(R) oneAPI Math Kernel Library Version 2022.2-Product Build 20220804 for Intel(R) 64 architecture applications

- Intel(R) MKL-DNN v2.7.3 (Git Hash 6dbeffbae1f23cbbeae17adb7b5b13f1f37c080e)

- OpenMP 201511 (a.k.a. OpenMP 4.5)

- LAPACK is enabled (usually provided by MKL)

- NNPACK is enabled

- CPU capability usage: AVX2

- Build settings: BLAS\_INFO=mkl, BUILD\_TYPE=Release, CUDA\_VERSION=11.8, CUDNN\_VERSION=8.7.0, CXX\_COMPILER=/opt/rh/devtoolset-9/root/usr/bin/c++, CXX\_FLAGS= -D\_GLIBCXX\_USE\_CXX11\_ABI=0 -fabi-version=11 -Wno-deprecated -fvisibility-inlines-hidden -DUSE\_PTHREADPOOL -DNDEBUG -DUSE\_KINETO -DLIBKINETO\_NOROCTRACER -DUSE\_FBGEMM -DUSE\_QNNPACK -DUSE\_PYTORCH\_QNNPACK -DUSE\_XNNPACK -DSYMBOLICATE\_MOBILE\_DEBUG\_HANDLE -O2 -fPIC -Wall -Wextra -Werror=return-type -Werror=non-virtual-dtor -Werror=bool-operation -Wnarrowing -Wno-missing-field-initializers -Wno-type-limits -Wno-array-bounds -Wno-unknown-pragmas -Wunused-local-typedefs -Wno-unused-parameter -Wno-unused-function -Wno-unused-result -Wno-strict-overflow -Wno-strict-aliasing -Wno-error=deprecated-declarations -Wno-stringop-overflow -Wno-psabi -Wno-error=pedantic -Wno-error=redundant-decls -Wno-error=old-style-cast -fdiagnostics-color=always -faligned-new -Wno-unused-but-set-variable -Wno-maybe-uninitialized -fno-math-errno -fno-trapping-math -Werror=format -Werror=cast-function-type -Wno-stringop-overflow, LAPACK\_INFO=mkl, PERF\_WITH\_AVX=1, PERF\_WITH\_AVX2=1, PERF\_WITH\_AVX512=1, TORCH\_DISABLE\_GPU\_ASSERTS=ON, TORCH\_VERSION=2.0.1, USE\_CUDA=ON, USE\_CUDNN=ON, USE\_EXCEPTION\_PTR=1, USE\_GFLAGS=OFF, USE\_GLOG=OFF, USE\_MKL=ON, USE\_MKLDNN=ON, USE\_MPI=OFF, USE\_NCCL=1, USE\_NNPACK=ON, USE\_OPENMP=ON, USE\_ROCM=OFF,

TorchVision: 0.15.2+cu118

OpenCV: 4.7.0

MMEngine: 0.7.4

Runtime environment:

cudnn\_benchmark: False

mp\_cfg: {'mp\_start\_method': 'fork', 'opencv\_num\_threads': 0}

dist\_cfg: {'backend': 'nccl'}

seed: 1525478322

Distributed launcher: none

Distributed training: False

GPU number: 1

------------------------------------------------------------

06/20 12:40:06 - mmengine - INFO - Config:

model = dict(

type='SoftTeacher',

detector=dict(

type='FasterRCNN',

data\_preprocessor=dict(

type='DetDataPreprocessor',

mean=[103.53, 116.28, 123.675],

std=[1.0, 1.0, 1.0],

bgr\_to\_rgb=False,

pad\_size\_divisor=32),

backbone=dict(

type='ResNet',

depth=50,

num\_stages=4,

out\_indices=(0, 1, 2, 3),

frozen\_stages=1,

norm\_cfg=dict(type='BN', requires\_grad=False),

norm\_eval=True,

style='caffe',

init\_cfg=dict(

type='Pretrained',

checkpoint='open-mmlab://detectron2/resnet50\_caffe')),

neck=dict(

type='FPN',

in\_channels=[256, 512, 1024, 2048],

out\_channels=256,

num\_outs=5),

rpn\_head=dict(

type='RPNHead',

in\_channels=256,

feat\_channels=256,

anchor\_generator=dict(

type='AnchorGenerator',

scales=[8],

ratios=[0.5, 1.0, 2.0],

strides=[4, 8, 16, 32, 64]),

bbox\_coder=dict(

type='DeltaXYWHBBoxCoder',

target\_means=[0.0, 0.0, 0.0, 0.0],

target\_stds=[1.0, 1.0, 1.0, 1.0]),

loss\_cls=dict(

type='CrossEntropyLoss', use\_sigmoid=True, loss\_weight=1.0),

loss\_bbox=dict(type='L1Loss', loss\_weight=1.0)),

roi\_head=dict(

type='StandardRoIHead',

bbox\_roi\_extractor=dict(

type='SingleRoIExtractor',

roi\_layer=dict(

type='RoIAlign', output\_size=7, sampling\_ratio=0),

out\_channels=256,

featmap\_strides=[4, 8, 16, 32]),

bbox\_head=dict(

type='Shared2FCBBoxHead',

in\_channels=256,

fc\_out\_channels=1024,

roi\_feat\_size=7,

num\_classes=80,

bbox\_coder=dict(

type='DeltaXYWHBBoxCoder',

target\_means=[0.0, 0.0, 0.0, 0.0],

target\_stds=[0.1, 0.1, 0.2, 0.2]),

reg\_class\_agnostic=False,

loss\_cls=dict(

type='CrossEntropyLoss',

use\_sigmoid=False,

loss\_weight=1.0),

loss\_bbox=dict(type='L1Loss', loss\_weight=1.0))),

train\_cfg=dict(

rpn=dict(

assigner=dict(

type='MaxIoUAssigner',

pos\_iou\_thr=0.7,

neg\_iou\_thr=0.3,

min\_pos\_iou=0.3,

match\_low\_quality=True,

ignore\_iof\_thr=-1),

sampler=dict(

type='RandomSampler',

num=256,

pos\_fraction=0.5,

neg\_pos\_ub=-1,

add\_gt\_as\_proposals=False),

allowed\_border=-1,

pos\_weight=-1,

debug=False),

rpn\_proposal=dict(

nms\_pre=2000,

max\_per\_img=1000,

nms=dict(type='nms', iou\_threshold=0.7),

min\_bbox\_size=0),

rcnn=dict(

assigner=dict(

type='MaxIoUAssigner',

pos\_iou\_thr=0.5,

neg\_iou\_thr=0.5,

min\_pos\_iou=0.5,

match\_low\_quality=False,

ignore\_iof\_thr=-1),

sampler=dict(

type='RandomSampler',

num=512,

pos\_fraction=0.25,

neg\_pos\_ub=-1,

add\_gt\_as\_proposals=True),

pos\_weight=-1,

debug=False)),

test\_cfg=dict(

rpn=dict(

nms\_pre=1000,

max\_per\_img=1000,

nms=dict(type='nms', iou\_threshold=0.7),

min\_bbox\_size=0),

rcnn=dict(

score\_thr=0.05,

nms=dict(type='nms', iou\_threshold=0.5),

max\_per\_img=100))),

data\_preprocessor=dict(

type='MultiBranchDataPreprocessor',

data\_preprocessor=dict(

type='DetDataPreprocessor',

mean=[103.53, 116.28, 123.675],

std=[1.0, 1.0, 1.0],

bgr\_to\_rgb=False,

pad\_size\_divisor=32)),

semi\_train\_cfg=dict(

freeze\_teacher=True,

sup\_weight=1.0,

unsup\_weight=4.0,

pseudo\_label\_initial\_score\_thr=0.5,

rpn\_pseudo\_thr=0.9,

cls\_pseudo\_thr=0.9,

reg\_pseudo\_thr=0.02,

jitter\_times=10,

jitter\_scale=0.06,

min\_pseudo\_bbox\_wh=(0.01, 0.01)),

semi\_test\_cfg=dict(predict\_on='teacher'))

default\_scope = 'mmdet'

default\_hooks = dict(

timer=dict(type='IterTimerHook'),

logger=dict(type='LoggerHook', interval=50),

param\_scheduler=dict(type='ParamSchedulerHook'),

checkpoint=dict(

type='CheckpointHook',

interval=10000,

by\_epoch=False,

max\_keep\_ckpts=2),

sampler\_seed=dict(type='DistSamplerSeedHook'),

visualization=dict(type='DetVisualizationHook'))

env\_cfg = dict(

cudnn\_benchmark=False,

mp\_cfg=dict(mp\_start\_method='fork', opencv\_num\_threads=0),

dist\_cfg=dict(backend='nccl'))

vis\_backends = [dict(type='LocalVisBackend')]

visualizer = dict(

type='DetLocalVisualizer',

vis\_backends=[dict(type='LocalVisBackend')],

name='visualizer')

log\_processor = dict(type='LogProcessor', window\_size=50, by\_epoch=False)

log\_level = 'INFO'

load\_from = None

resume = False

dataset\_type = 'CocoDataset'

data\_root = 'data/coco/'

backend\_args = None

color\_space = [[{

'type': 'ColorTransform'

}], [{

'type': 'AutoContrast'

}], [{

'type': 'Equalize'

}], [{

'type': 'Sharpness'

}], [{

'type': 'Posterize'

}], [{

'type': 'Solarize'

}], [{

'type': 'Color'

}], [{

'type': 'Contrast'

}], [{

'type': 'Brightness'

}]]

geometric = [[{

'type': 'Rotate'

}], [{

'type': 'ShearX'

}], [{

'type': 'ShearY'

}], [{

'type': 'TranslateX'

}], [{

'type': 'TranslateY'

}]]

scale = [(1333, 400), (1333, 1200)]

branch\_field = ['sup', 'unsup\_teacher', 'unsup\_student']

sup\_pipeline = [

dict(type='LoadImageFromFile', backend\_args=None),

dict(type='LoadAnnotations', with\_bbox=True),

dict(

type='RandomResize',

scale=[(1333, 400), (1333, 1200)],

keep\_ratio=True),

dict(type='RandomFlip', prob=0.5),

dict(

type='RandAugment',

aug\_space=[[{

'type': 'ColorTransform'

}], [{

'type': 'AutoContrast'

}], [{

'type': 'Equalize'

}], [{

'type': 'Sharpness'

}], [{

'type': 'Posterize'

}], [{

'type': 'Solarize'

}], [{

'type': 'Color'

}], [{

'type': 'Contrast'

}], [{

'type': 'Brightness'

}]],

aug\_num=1),

dict(type='FilterAnnotations', min\_gt\_bbox\_wh=(0.01, 0.01)),

dict(

type='MultiBranch',

branch\_field=['sup', 'unsup\_teacher', 'unsup\_student'],

sup=dict(type='PackDetInputs'))

]

weak\_pipeline = [

dict(

type='RandomResize',

scale=[(1333, 400), (1333, 1200)],

keep\_ratio=True),

dict(type='RandomFlip', prob=0.5),

dict(

type='PackDetInputs',

meta\_keys=('img\_id', 'img\_path', 'ori\_shape', 'img\_shape',

'scale\_factor', 'flip', 'flip\_direction',

'homography\_matrix'))

]

strong\_pipeline = [

dict(

type='RandomResize',

scale=[(1333, 400), (1333, 1200)],

keep\_ratio=True),

dict(type='RandomFlip', prob=0.5),

dict(

type='RandomOrder',

transforms=[

dict(

type='RandAugment',

aug\_space=[[{

'type': 'ColorTransform'

}], [{

'type': 'AutoContrast'

}], [{

'type': 'Equalize'

}], [{

'type': 'Sharpness'

}], [{

'type': 'Posterize'

}], [{

'type': 'Solarize'

}], [{

'type': 'Color'

}], [{

'type': 'Contrast'

}], [{

'type': 'Brightness'

}]],

aug\_num=1),

dict(

type='RandAugment',

aug\_space=[[{

'type': 'Rotate'

}], [{

'type': 'ShearX'

}], [{

'type': 'ShearY'

}], [{

'type': 'TranslateX'

}], [{

'type': 'TranslateY'

}]],

aug\_num=1)

]),

dict(type='RandomErasing', n\_patches=(1, 5), ratio=(0, 0.2)),

dict(type='FilterAnnotations', min\_gt\_bbox\_wh=(0.01, 0.01)),

dict(

type='PackDetInputs',

meta\_keys=('img\_id', 'img\_path', 'ori\_shape', 'img\_shape',

'scale\_factor', 'flip', 'flip\_direction',

'homography\_matrix'))

]

unsup\_pipeline = [

dict(type='LoadImageFromFile', backend\_args=None),

dict(type='LoadEmptyAnnotations'),

dict(

type='MultiBranch',

branch\_field=['sup', 'unsup\_teacher', 'unsup\_student'],

unsup\_teacher=[

dict(

type='RandomResize',

scale=[(1333, 400), (1333, 1200)],

keep\_ratio=True),

dict(type='RandomFlip', prob=0.5),

dict(

type='PackDetInputs',

meta\_keys=('img\_id', 'img\_path', 'ori\_shape', 'img\_shape',

'scale\_factor', 'flip', 'flip\_direction',

'homography\_matrix'))

],

unsup\_student=[

dict(

type='RandomResize',

scale=[(1333, 400), (1333, 1200)],

keep\_ratio=True),

dict(type='RandomFlip', prob=0.5),

dict(

type='RandomOrder',

transforms=[

dict(

type='RandAugment',

aug\_space=[[{

'type': 'ColorTransform'

}], [{

'type': 'AutoContrast'

}], [{

'type': 'Equalize'

}], [{

'type': 'Sharpness'

}], [{

'type': 'Posterize'

}], [{

'type': 'Solarize'

}], [{

'type': 'Color'

}], [{

'type': 'Contrast'

}], [{

'type': 'Brightness'

}]],

aug\_num=1),

dict(

type='RandAugment',

aug\_space=[[{

'type': 'Rotate'

}], [{

'type': 'ShearX'

}], [{

'type': 'ShearY'

}], [{

'type': 'TranslateX'

}], [{

'type': 'TranslateY'

}]],

aug\_num=1)

]),

dict(type='RandomErasing', n\_patches=(1, 5), ratio=(0, 0.2)),

dict(type='FilterAnnotations', min\_gt\_bbox\_wh=(0.01, 0.01)),

dict(

type='PackDetInputs',

meta\_keys=('img\_id', 'img\_path', 'ori\_shape', 'img\_shape',

'scale\_factor', 'flip', 'flip\_direction',

'homography\_matrix'))

])

]

test\_pipeline = [

dict(type='LoadImageFromFile', backend\_args=None),

dict(type='Resize', scale=(1333, 800), keep\_ratio=True),

dict(

type='PackDetInputs',

meta\_keys=('img\_id', 'img\_path', 'ori\_shape', 'img\_shape',

'scale\_factor'))

]

batch\_size = 5

num\_workers = 5

labeled\_dataset = dict(

type='CocoDataset',

data\_root='data/coco/',

ann\_file='annotations/coco\_annotations.json',

data\_prefix=dict(img='train/'),

filter\_cfg=dict(filter\_empty\_gt=True, min\_size=32),

pipeline=[

dict(type='LoadImageFromFile', backend\_args=None),

dict(type='LoadAnnotations', with\_bbox=True),

dict(

type='RandomResize',

scale=[(1333, 400), (1333, 1200)],

keep\_ratio=True),

dict(type='RandomFlip', prob=0.5),

dict(

type='RandAugment',

aug\_space=[[{

'type': 'ColorTransform'

}], [{

'type': 'AutoContrast'

}], [{

'type': 'Equalize'

}], [{

'type': 'Sharpness'

}], [{

'type': 'Posterize'

}], [{

'type': 'Solarize'

}], [{

'type': 'Color'

}], [{

'type': 'Contrast'

}], [{

'type': 'Brightness'

}]],

aug\_num=1),

dict(type='FilterAnnotations', min\_gt\_bbox\_wh=(0.01, 0.01)),

dict(

type='MultiBranch',

branch\_field=['sup', 'unsup\_teacher', 'unsup\_student'],

sup=dict(type='PackDetInputs'))

],

backend\_args=None)

unlabeled\_dataset = dict(

type='CocoDataset',

data\_root='data/coco/',

ann\_file='annotations/val\_coco\_annotations.json',

data\_prefix=dict(img='val/'),

filter\_cfg=dict(filter\_empty\_gt=False),

pipeline=[

dict(type='LoadImageFromFile', backend\_args=None),

dict(type='LoadEmptyAnnotations'),

dict(

type='MultiBranch',

branch\_field=['sup', 'unsup\_teacher', 'unsup\_student'],

unsup\_teacher=[

dict(

type='RandomResize',

scale=[(1333, 400), (1333, 1200)],

keep\_ratio=True),

dict(type='RandomFlip', prob=0.5),

dict(

type='PackDetInputs',

meta\_keys=('img\_id', 'img\_path', 'ori\_shape', 'img\_shape',

'scale\_factor', 'flip', 'flip\_direction',

'homography\_matrix'))

],

unsup\_student=[

dict(

type='RandomResize',

scale=[(1333, 400), (1333, 1200)],

keep\_ratio=True),

dict(type='RandomFlip', prob=0.5),

dict(

type='RandomOrder',

transforms=[

dict(

type='RandAugment',

aug\_space=[[{

'type': 'ColorTransform'

}], [{

'type': 'AutoContrast'

}], [{

'type': 'Equalize'

}], [{

'type': 'Sharpness'

}], [{

'type': 'Posterize'

}], [{

'type': 'Solarize'

}], [{

'type': 'Color'

}], [{

'type': 'Contrast'

}], [{

'type': 'Brightness'

}]],

aug\_num=1),

dict(

type='RandAugment',

aug\_space=[[{

'type': 'Rotate'

}], [{

'type': 'ShearX'

}], [{

'type': 'ShearY'

}], [{

'type': 'TranslateX'

}], [{

'type': 'TranslateY'

}]],

aug\_num=1)

]),

dict(type='RandomErasing', n\_patches=(1, 5), ratio=(0, 0.2)),

dict(type='FilterAnnotations', min\_gt\_bbox\_wh=(0.01, 0.01)),

dict(

type='PackDetInputs',

meta\_keys=('img\_id', 'img\_path', 'ori\_shape', 'img\_shape',

'scale\_factor', 'flip', 'flip\_direction',

'homography\_matrix'))

])

],

backend\_args=None)

train\_dataloader = dict(

batch\_size=5,

num\_workers=5,

persistent\_workers=True,

sampler=dict(

type='GroupMultiSourceSampler', batch\_size=5, source\_ratio=[1, 4]),

dataset=dict(

type='ConcatDataset',

datasets=[

dict(

type='CocoDataset',

data\_root='data/coco/',

ann\_file='annotations/coco\_annotations.json',

data\_prefix=dict(img='train/'),

filter\_cfg=dict(filter\_empty\_gt=True, min\_size=32),

pipeline=[

dict(type='LoadImageFromFile', backend\_args=None),

dict(type='LoadAnnotations', with\_bbox=True),

dict(

type='RandomResize',

scale=[(1333, 400), (1333, 1200)],

keep\_ratio=True),

dict(type='RandomFlip', prob=0.5),

dict(

type='RandAugment',

aug\_space=[[{

'type': 'ColorTransform'

}], [{

'type': 'AutoContrast'

}], [{

'type': 'Equalize'

}], [{

'type': 'Sharpness'

}], [{

'type': 'Posterize'

}], [{

'type': 'Solarize'

}], [{

'type': 'Color'

}], [{

'type': 'Contrast'

}], [{

'type': 'Brightness'

}]],

aug\_num=1),

dict(

type='FilterAnnotations', min\_gt\_bbox\_wh=(0.01, 0.01)),

dict(

type='MultiBranch',

branch\_field=['sup', 'unsup\_teacher', 'unsup\_student'],

sup=dict(type='PackDetInputs'))

],

backend\_args=None),

dict(

type='CocoDataset',

data\_root='data/coco/',

ann\_file='annotations/val\_coco\_annotations.json',

data\_prefix=dict(img='val/'),

filter\_cfg=dict(filter\_empty\_gt=False),

pipeline=[

dict(type='LoadImageFromFile', backend\_args=None),

dict(type='LoadEmptyAnnotations'),

dict(

type='MultiBranch',

branch\_field=['sup', 'unsup\_teacher', 'unsup\_student'],

unsup\_teacher=[

dict(

type='RandomResize',

scale=[(1333, 400), (1333, 1200)],

keep\_ratio=True),

dict(type='RandomFlip', prob=0.5),

dict(

type='PackDetInputs',

meta\_keys=('img\_id', 'img\_path', 'ori\_shape',

'img\_shape', 'scale\_factor', 'flip',

'flip\_direction',

'homography\_matrix'))

],

unsup\_student=[

dict(

type='RandomResize',

scale=[(1333, 400), (1333, 1200)],

keep\_ratio=True),

dict(type='RandomFlip', prob=0.5),

dict(

type='RandomOrder',

transforms=[

dict(

type='RandAugment',

aug\_space=[[{

'type': 'ColorTransform'

}], [{

'type': 'AutoContrast'

}], [{

'type': 'Equalize'

}], [{

'type': 'Sharpness'

}], [{

'type': 'Posterize'

}], [{

'type': 'Solarize'

}], [{

'type': 'Color'

}], [{

'type': 'Contrast'

}], [{

'type': 'Brightness'

}]],

aug\_num=1),

dict(

type='RandAugment',

aug\_space=[[{

'type': 'Rotate'

}], [{

'type': 'ShearX'

}], [{

'type': 'ShearY'

}], [{

'type': 'TranslateX'

}], [{

'type': 'TranslateY'

}]],

aug\_num=1)

]),

dict(

type='RandomErasing',

n\_patches=(1, 5),

ratio=(0, 0.2)),

dict(

type='FilterAnnotations',

min\_gt\_bbox\_wh=(0.01, 0.01)),

dict(

type='PackDetInputs',

meta\_keys=('img\_id', 'img\_path', 'ori\_shape',

'img\_shape', 'scale\_factor', 'flip',

'flip\_direction',

'homography\_matrix'))

])

],

backend\_args=None)

]))

val\_dataloader = dict(

batch\_size=1,

num\_workers=2,

persistent\_workers=True,

drop\_last=False,

sampler=dict(type='DefaultSampler', shuffle=False),

dataset=dict(

type='CocoDataset',

data\_root='data/coco/',

ann\_file='annotations/val\_coco\_annotations.json',

data\_prefix=dict(img='val/'),

test\_mode=True,

pipeline=[

dict(type='LoadImageFromFile', backend\_args=None),

dict(type='Resize', scale=(1333, 800), keep\_ratio=True),

dict(

type='PackDetInputs',

meta\_keys=('img\_id', 'img\_path', 'ori\_shape', 'img\_shape',

'scale\_factor'))

],

backend\_args=None))

test\_dataloader = dict(

batch\_size=1,

num\_workers=2,

persistent\_workers=True,

drop\_last=False,

sampler=dict(type='DefaultSampler', shuffle=False),

dataset=dict(

type='CocoDataset',

data\_root='data/coco/',

ann\_file='annotations/val\_coco\_annotations.json',

data\_prefix=dict(img='val/'),

test\_mode=True,

pipeline=[

dict(type='LoadImageFromFile', backend\_args=None),

dict(type='Resize', scale=(1333, 800), keep\_ratio=True),

dict(

type='PackDetInputs',

meta\_keys=('img\_id', 'img\_path', 'ori\_shape', 'img\_shape',

'scale\_factor'))

],

backend\_args=None))

val\_evaluator = dict(

type='CocoMetric',

ann\_file='data/coco/annotations/val\_coco\_annotations.json',

metric='bbox',

format\_only=False,

backend\_args=None)

test\_evaluator = dict(

type='CocoMetric',

ann\_file='data/coco/annotations/val\_coco\_annotations.json',

metric='bbox',

format\_only=False,

backend\_args=None)

detector = dict(

type='FasterRCNN',

data\_preprocessor=dict(

type='DetDataPreprocessor',

mean=[103.53, 116.28, 123.675],

std=[1.0, 1.0, 1.0],

bgr\_to\_rgb=False,

pad\_size\_divisor=32),

backbone=dict(

type='ResNet',

depth=50,

num\_stages=4,

out\_indices=(0, 1, 2, 3),

frozen\_stages=1,

norm\_cfg=dict(type='BN', requires\_grad=False),

norm\_eval=True,

style='caffe',

init\_cfg=dict(

type='Pretrained',

checkpoint='open-mmlab://detectron2/resnet50\_caffe')),

neck=dict(

type='FPN',

in\_channels=[256, 512, 1024, 2048],

out\_channels=256,

num\_outs=5),

rpn\_head=dict(

type='RPNHead',

in\_channels=256,

feat\_channels=256,

anchor\_generator=dict(

type='AnchorGenerator',

scales=[8],

ratios=[0.5, 1.0, 2.0],

strides=[4, 8, 16, 32, 64]),

bbox\_coder=dict(

type='DeltaXYWHBBoxCoder',

target\_means=[0.0, 0.0, 0.0, 0.0],

target\_stds=[1.0, 1.0, 1.0, 1.0]),

loss\_cls=dict(

type='CrossEntropyLoss', use\_sigmoid=True, loss\_weight=1.0),

loss\_bbox=dict(type='L1Loss', loss\_weight=1.0)),

roi\_head=dict(

type='StandardRoIHead',

bbox\_roi\_extractor=dict(

type='SingleRoIExtractor',

roi\_layer=dict(type='RoIAlign', output\_size=7, sampling\_ratio=0),

out\_channels=256,

featmap\_strides=[4, 8, 16, 32]),

bbox\_head=dict(

type='Shared2FCBBoxHead',

in\_channels=256,

fc\_out\_channels=1024,

roi\_feat\_size=7,

num\_classes=80,

bbox\_coder=dict(

type='DeltaXYWHBBoxCoder',

target\_means=[0.0, 0.0, 0.0, 0.0],

target\_stds=[0.1, 0.1, 0.2, 0.2]),

reg\_class\_agnostic=False,

loss\_cls=dict(

type='CrossEntropyLoss', use\_sigmoid=False, loss\_weight=1.0),

loss\_bbox=dict(type='L1Loss', loss\_weight=1.0))),

train\_cfg=dict(

rpn=dict(

assigner=dict(

type='MaxIoUAssigner',

pos\_iou\_thr=0.7,

neg\_iou\_thr=0.3,

min\_pos\_iou=0.3,

match\_low\_quality=True,

ignore\_iof\_thr=-1),

sampler=dict(

type='RandomSampler',

num=256,

pos\_fraction=0.5,

neg\_pos\_ub=-1,

add\_gt\_as\_proposals=False),

allowed\_border=-1,

pos\_weight=-1,

debug=False),

rpn\_proposal=dict(

nms\_pre=2000,

max\_per\_img=1000,

nms=dict(type='nms', iou\_threshold=0.7),

min\_bbox\_size=0),

rcnn=dict(

assigner=dict(

type='MaxIoUAssigner',

pos\_iou\_thr=0.5,

neg\_iou\_thr=0.5,

min\_pos\_iou=0.5,

match\_low\_quality=False,

ignore\_iof\_thr=-1),

sampler=dict(

type='RandomSampler',

num=512,

pos\_fraction=0.25,

neg\_pos\_ub=-1,

add\_gt\_as\_proposals=True),

pos\_weight=-1,

debug=False)),

test\_cfg=dict(

rpn=dict(

nms\_pre=1000,

max\_per\_img=1000,

nms=dict(type='nms', iou\_threshold=0.7),

min\_bbox\_size=0),

rcnn=dict(

score\_thr=0.05,

nms=dict(type='nms', iou\_threshold=0.5),

max\_per\_img=100)))

train\_cfg = dict(

type='IterBasedTrainLoop', max\_iters=180000, val\_interval=5000)

val\_cfg = dict(type='TeacherStudentValLoop')

test\_cfg = dict(type='TestLoop')

param\_scheduler = [

dict(

type='LinearLR', start\_factor=0.001, by\_epoch=False, begin=0, end=500),

dict(

type='MultiStepLR',

begin=0,

end=180000,

by\_epoch=False,

milestones=[120000, 160000],

gamma=0.1)

]

optim\_wrapper = dict(

type='OptimWrapper',

optimizer=dict(type='SGD', lr=0.01, momentum=0.9, weight\_decay=0.0001))

custom\_hooks = [dict(type='MeanTeacherHook')]

launcher = 'none'

work\_dir = './work\_dirs/soft-teacher\_faster-rcnn\_r50-caffe\_fpn\_180k\_semi-0.1-coco'

06/20 12:40:18 - mmengine - INFO - Distributed training is not used, all SyncBatchNorm (SyncBN) layers in the model will be automatically reverted to BatchNormXd layers if they are used.

06/20 12:40:18 - mmengine - INFO - Hooks will be executed in the following order:

before\_run:

(VERY\_HIGH ) RuntimeInfoHook

(BELOW\_NORMAL) LoggerHook

--------------------

before\_train:

(VERY\_HIGH ) RuntimeInfoHook

(NORMAL ) IterTimerHook

(NORMAL ) MeanTeacherHook

(VERY\_LOW ) CheckpointHook

--------------------

before\_train\_epoch:

(VERY\_HIGH ) RuntimeInfoHook

(NORMAL ) IterTimerHook

(NORMAL ) DistSamplerSeedHook

--------------------

before\_train\_iter:

(VERY\_HIGH ) RuntimeInfoHook

(NORMAL ) IterTimerHook

--------------------

after\_train\_iter:

(VERY\_HIGH ) RuntimeInfoHook

(NORMAL ) IterTimerHook

(NORMAL ) MeanTeacherHook

(BELOW\_NORMAL) LoggerHook

(LOW ) ParamSchedulerHook

(VERY\_LOW ) CheckpointHook

--------------------

after\_train\_epoch:

(NORMAL ) IterTimerHook

(LOW ) ParamSchedulerHook

(VERY\_LOW ) CheckpointHook

--------------------

before\_val\_epoch:

(NORMAL ) IterTimerHook

--------------------

before\_val\_iter:

(NORMAL ) IterTimerHook

--------------------

after\_val\_iter:

(NORMAL ) IterTimerHook

(NORMAL ) DetVisualizationHook

(BELOW\_NORMAL) LoggerHook

--------------------

after\_val\_epoch:

(VERY\_HIGH ) RuntimeInfoHook

(NORMAL ) IterTimerHook

(BELOW\_NORMAL) LoggerHook

(LOW ) ParamSchedulerHook

(VERY\_LOW ) CheckpointHook

--------------------

after\_train:

(VERY\_LOW ) CheckpointHook

--------------------

before\_test\_epoch:

(NORMAL ) IterTimerHook

--------------------

before\_test\_iter:

(NORMAL ) IterTimerHook

--------------------

after\_test\_iter:

(NORMAL ) IterTimerHook

(NORMAL ) DetVisualizationHook

(BELOW\_NORMAL) LoggerHook

--------------------

after\_test\_epoch:

(VERY\_HIGH ) RuntimeInfoHook

(NORMAL ) IterTimerHook

(BELOW\_NORMAL) LoggerHook

--------------------

after\_run:

(BELOW\_NORMAL) LoggerHook

--------------------

loading annotations into memory...

Done (t=0.00s)

creating index...

index created!

loading annotations into memory...

Done (t=0.03s)

creating index...

index created!

/usr/local/lib/python3.10/dist-packages/torch/utils/data/dataloader.py:560: UserWarning: This DataLoader will create 5 worker processes in total. Our suggested max number of worker in current system is 2, which is smaller than what this DataLoader is going to create. Please be aware that excessive worker creation might get DataLoader running slow or even freeze, lower the worker number to avoid potential slowness/freeze if necessary.

warnings.warn(\_create\_warning\_msg(

loading annotations into memory...

Done (t=0.15s)

creating index...

index created!

loading annotations into memory...

Done (t=0.14s)

creating index...

index created!

06/20 12:40:38 - mmengine - INFO - load model from: open-mmlab://detectron2/resnet50\_caffe

06/20 12:40:38 - mmengine - INFO - Loads checkpoint by openmmlab backend from path: open-mmlab://detectron2/resnet50\_caffe

06/20 12:40:39 - mmengine - WARNING - The model and loaded state dict do not match exactly

unexpected key in source state\_dict: conv1.bias

06/20 12:40:40 - mmengine - INFO - load model from: open-mmlab://detectron2/resnet50\_caffe

06/20 12:40:40 - mmengine - INFO - Loads checkpoint by openmmlab backend from path: open-mmlab://detectron2/resnet50\_caffe

06/20 12:40:40 - mmengine - WARNING - "FileClient" will be deprecated in future. Please use io functions in <https://mmengine.readthedocs.io/en/latest/api/fileio.html#file-io>

06/20 12:40:40 - mmengine - WARNING - "HardDiskBackend" is the alias of "LocalBackend" and the former will be deprecated in future.

06/20 12:40:40 - mmengine - INFO - Checkpoints will be saved to /content/mmdetection/work\_dirs/soft-teacher\_faster-rcnn\_r50-caffe\_fpn\_180k\_semi-0.1-coco.



this is how the process should look, kindly zoom and take reference.

**if you require some more study about the architecture of soft teacher, you can refer it in this directory, once you finish or start your train you’ll find this.**

We will then look into “mmdetection/work\_dirs/soft-teacher\_faster-rcnn\_r50-caffe\_fpn\_180k\_semi-0.1-coco/soft-teacher\_faster-rcnn\_r50-caffe\_fpn\_180k\_semi-0.1-coco.py”

This work\_dirs is where all the running of your model gets stored.

This file contains the actual architecture of soft teacher algorithm.

model = dict(

    type='SoftTeacher',

    detector=dict(

        type='FasterRCNN',

        data\_preprocessor=dict(

            type='DetDataPreprocessor',

            mean=[103.53, 116.28, 123.675],

            std=[1.0, 1.0, 1.0],

            bgr\_to\_rgb=False,

            pad\_size\_divisor=32),

        backbone=dict(

            type='ResNet',

            depth=50,

            num\_stages=4,

            out\_indices=(0, 1, 2, 3),

            frozen\_stages=1,

            norm\_cfg=dict(type='BN', requires\_grad=False),

            norm\_eval=True,

            style='caffe',

            init\_cfg=dict(

                type='Pretrained',

                checkpoint='open-mmlab://detectron2/resnet50\_caffe')),

        neck=dict(

            type='FPN',

            in\_channels=[256, 512, 1024, 2048],

            out\_channels=256,

            num\_outs=5),

        rpn\_head=dict(

            type='RPNHead',

            in\_channels=256,

            feat\_channels=256,

            anchor\_generator=dict(

                type='AnchorGenerator',

                scales=[8],

                ratios=[0.5, 1.0, 2.0],

                strides=[4, 8, 16, 32, 64]),

            bbox\_coder=dict(

                type='DeltaXYWHBBoxCoder',

                target\_means=[0.0, 0.0, 0.0, 0.0],

                target\_stds=[1.0, 1.0, 1.0, 1.0]),

            loss\_cls=dict(

                type='CrossEntropyLoss', use\_sigmoid=True, loss\_weight=1.0),

            loss\_bbox=dict(type='L1Loss', loss\_weight=1.0)),

        roi\_head=dict(

            type='StandardRoIHead',

            bbox\_roi\_extractor=dict(

                type='SingleRoIExtractor',

                roi\_layer=dict(

                    type='RoIAlign', output\_size=7, sampling\_ratio=0),

                out\_channels=256,

                featmap\_strides=[4, 8, 16, 32]),

            bbox\_head=dict(

                type='Shared2FCBBoxHead',

                in\_channels=256,

                fc\_out\_channels=1024,

                roi\_feat\_size=7,

                num\_classes=80,

                bbox\_coder=dict(

                    type='DeltaXYWHBBoxCoder',

                    target\_means=[0.0, 0.0, 0.0, 0.0],

                    target\_stds=[0.1, 0.1, 0.2, 0.2]),

                reg\_class\_agnostic=False,

                loss\_cls=dict(

                    type='CrossEntropyLoss',

                    use\_sigmoid=False,

                    loss\_weight=1.0),

                loss\_bbox=dict(type='L1Loss', loss\_weight=1.0))),

        train\_cfg=dict(

            rpn=dict(

                assigner=dict(

                    type='MaxIoUAssigner',

                    pos\_iou\_thr=0.7,

                    neg\_iou\_thr=0.3,

                    min\_pos\_iou=0.3,

                    match\_low\_quality=True,

                    ignore\_iof\_thr=-1),

                sampler=dict(

                    type='RandomSampler',

                    num=256,

                    pos\_fraction=0.5,

                    neg\_pos\_ub=-1,

                    add\_gt\_as\_proposals=False),

                allowed\_border=-1,

                pos\_weight=-1,

                debug=False),

            rpn\_proposal=dict(

                nms\_pre=2000,

                max\_per\_img=1000,

                nms=dict(type='nms', iou\_threshold=0.7),

                min\_bbox\_size=0),

            rcnn=dict(

                assigner=dict(

                    type='MaxIoUAssigner',

                    pos\_iou\_thr=0.5,

                    neg\_iou\_thr=0.5,

                    min\_pos\_iou=0.5,

                    match\_low\_quality=False,

                    ignore\_iof\_thr=-1),

                sampler=dict(

                    type='RandomSampler',

                    num=512,

                    pos\_fraction=0.25,

                    neg\_pos\_ub=-1,

                    add\_gt\_as\_proposals=True),

                pos\_weight=-1,

                debug=False)),

        test\_cfg=dict(

            rpn=dict(

                nms\_pre=1000,

                max\_per\_img=1000,

                nms=dict(type='nms', iou\_threshold=0.7),

                min\_bbox\_size=0),

            rcnn=dict(

                score\_thr=0.05,

                nms=dict(type='nms', iou\_threshold=0.5),

                max\_per\_img=100))),

    data\_preprocessor=dict(

        type='MultiBranchDataPreprocessor',

        data\_preprocessor=dict(

            type='DetDataPreprocessor',

            mean=[103.53, 116.28, 123.675],

            std=[1.0, 1.0, 1.0],

            bgr\_to\_rgb=False,

            pad\_size\_divisor=32)),

    semi\_train\_cfg=dict(

        freeze\_teacher=True,

        sup\_weight=1.0,

        unsup\_weight=4.0,

        pseudo\_label\_initial\_score\_thr=0.5,

        rpn\_pseudo\_thr=0.9,

        cls\_pseudo\_thr=0.9,

        reg\_pseudo\_thr=0.02,

        jitter\_times=10,

        jitter\_scale=0.06,

        min\_pseudo\_bbox\_wh=(0.01, 0.01)),

    semi\_test\_cfg=dict(predict\_on='teacher'))

default\_scope = 'mmdet'

default\_hooks = dict(

    timer=dict(type='IterTimerHook'),

    logger=dict(type='LoggerHook', interval=50),

    param\_scheduler=dict(type='ParamSchedulerHook'),

    checkpoint=dict(

        type='CheckpointHook',

        interval=10000,

        by\_epoch=False,

        max\_keep\_ckpts=2),

    sampler\_seed=dict(type='DistSamplerSeedHook'),

    visualization=dict(type='DetVisualizationHook'))

env\_cfg = dict(

    cudnn\_benchmark=False,

    mp\_cfg=dict(mp\_start\_method='fork', opencv\_num\_threads=0),

    dist\_cfg=dict(backend='nccl'))

vis\_backends = [dict(type='LocalVisBackend')]

visualizer = dict(

    type='DetLocalVisualizer',

    vis\_backends=[dict(type='LocalVisBackend')],

    name='visualizer')

log\_processor = dict(type='LogProcessor', window\_size=50, by\_epoch=False)

log\_level = 'INFO'

load\_from = None

resume = False

dataset\_type = 'CocoDataset'

data\_root = 'data/coco/'

backend\_args = None

color\_space = [[{

    'type': 'ColorTransform'

}], [{

    'type': 'AutoContrast'

}], [{

    'type': 'Equalize'

}], [{

    'type': 'Sharpness'

}], [{

    'type': 'Posterize'

}], [{

    'type': 'Solarize'

}], [{

    'type': 'Color'

}], [{

    'type': 'Contrast'

}], [{

    'type': 'Brightness'

}]]

geometric = [[{

    'type': 'Rotate'

}], [{

    'type': 'ShearX'

}], [{

    'type': 'ShearY'

}], [{

    'type': 'TranslateX'

}], [{

    'type': 'TranslateY'

}]]

scale = [(1333, 400), (1333, 1200)]

branch\_field = ['sup', 'unsup\_teacher', 'unsup\_student']

sup\_pipeline = [

    dict(type='LoadImageFromFile', backend\_args=None),

    dict(type='LoadAnnotations', with\_bbox=True),

    dict(

        type='RandomResize',

        scale=[(1333, 400), (1333, 1200)],

        keep\_ratio=True),

    dict(type='RandomFlip', prob=0.5),

    dict(

        type='RandAugment',

        aug\_space=[[{

            'type': 'ColorTransform'

        }], [{

            'type': 'AutoContrast'

        }], [{

            'type': 'Equalize'

        }], [{

            'type': 'Sharpness'

        }], [{

            'type': 'Posterize'

        }], [{

            'type': 'Solarize'

        }], [{

            'type': 'Color'

        }], [{

            'type': 'Contrast'

        }], [{

            'type': 'Brightness'

        }]],

        aug\_num=1),

    dict(type='FilterAnnotations', min\_gt\_bbox\_wh=(0.01, 0.01)),

    dict(

        type='MultiBranch',

        branch\_field=['sup', 'unsup\_teacher', 'unsup\_student'],

        sup=dict(type='PackDetInputs'))

]

weak\_pipeline = [

    dict(

        type='RandomResize',

        scale=[(1333, 400), (1333, 1200)],

        keep\_ratio=True),

    dict(type='RandomFlip', prob=0.5),

    dict(

        type='PackDetInputs',

        meta\_keys=('img\_id', 'img\_path', 'ori\_shape', 'img\_shape',

                   'scale\_factor', 'flip', 'flip\_direction',

                   'homography\_matrix'))

]

strong\_pipeline = [

    dict(

        type='RandomResize',

        scale=[(1333, 400), (1333, 1200)],

        keep\_ratio=True),

    dict(type='RandomFlip', prob=0.5),

    dict(

        type='RandomOrder',

        transforms=[

            dict(

                type='RandAugment',

                aug\_space=[[{

                    'type': 'ColorTransform'

                }], [{

                    'type': 'AutoContrast'

                }], [{

                    'type': 'Equalize'

                }], [{

                    'type': 'Sharpness'

                }], [{

                    'type': 'Posterize'

                }], [{

                    'type': 'Solarize'

                }], [{

                    'type': 'Color'

                }], [{

                    'type': 'Contrast'

                }], [{

                    'type': 'Brightness'

                }]],

                aug\_num=1),

            dict(

                type='RandAugment',

                aug\_space=[[{

                    'type': 'Rotate'

                }], [{

                    'type': 'ShearX'

                }], [{

                    'type': 'ShearY'

                }], [{

                    'type': 'TranslateX'

                }], [{

                    'type': 'TranslateY'

                }]],

                aug\_num=1)

        ]),

    dict(type='RandomErasing', n\_patches=(1, 5), ratio=(0, 0.2)),

    dict(type='FilterAnnotations', min\_gt\_bbox\_wh=(0.01, 0.01)),

    dict(

        type='PackDetInputs',

        meta\_keys=('img\_id', 'img\_path', 'ori\_shape', 'img\_shape',

                   'scale\_factor', 'flip', 'flip\_direction',

                   'homography\_matrix'))

]

unsup\_pipeline = [

    dict(type='LoadImageFromFile', backend\_args=None),

    dict(type='LoadEmptyAnnotations'),

    dict(

        type='MultiBranch',

        branch\_field=['sup', 'unsup\_teacher', 'unsup\_student'],

        unsup\_teacher=[

            dict(

                type='RandomResize',

                scale=[(1333, 400), (1333, 1200)],

                keep\_ratio=True),

            dict(type='RandomFlip', prob=0.5),

            dict(

                type='PackDetInputs',

                meta\_keys=('img\_id', 'img\_path', 'ori\_shape', 'img\_shape',

                           'scale\_factor', 'flip', 'flip\_direction',

                           'homography\_matrix'))

        ],

        unsup\_student=[

            dict(

                type='RandomResize',

                scale=[(1333, 400), (1333, 1200)],

                keep\_ratio=True),

            dict(type='RandomFlip', prob=0.5),

            dict(

                type='RandomOrder',

                transforms=[

                    dict(

                        type='RandAugment',

                        aug\_space=[[{

                            'type': 'ColorTransform'

                        }], [{

                            'type': 'AutoContrast'

                        }], [{

                            'type': 'Equalize'

                        }], [{

                            'type': 'Sharpness'

                        }], [{

                            'type': 'Posterize'

                        }], [{

                            'type': 'Solarize'

                        }], [{

                            'type': 'Color'

                        }], [{

                            'type': 'Contrast'

                        }], [{

                            'type': 'Brightness'

                        }]],

                        aug\_num=1),

                    dict(

                        type='RandAugment',

                        aug\_space=[[{

                            'type': 'Rotate'

                        }], [{

                            'type': 'ShearX'

                        }], [{

                            'type': 'ShearY'

                        }], [{

                            'type': 'TranslateX'

                        }], [{

                            'type': 'TranslateY'

                        }]],

                        aug\_num=1)

                ]),

            dict(type='RandomErasing', n\_patches=(1, 5), ratio=(0, 0.2)),

            dict(type='FilterAnnotations', min\_gt\_bbox\_wh=(0.01, 0.01)),

            dict(

                type='PackDetInputs',

                meta\_keys=('img\_id', 'img\_path', 'ori\_shape', 'img\_shape',

                           'scale\_factor', 'flip', 'flip\_direction',

                           'homography\_matrix'))

        ])

]

test\_pipeline = [

    dict(type='LoadImageFromFile', backend\_args=None),

    dict(type='Resize', scale=(1333, 800), keep\_ratio=True),

    dict(

        type='PackDetInputs',

        meta\_keys=('img\_id', 'img\_path', 'ori\_shape', 'img\_shape',

                   'scale\_factor'))

]

batch\_size = 5

num\_workers = 5

labeled\_dataset = dict(

    type='CocoDataset',

    data\_root='data/coco/',

    ann\_file='annotations/coco\_annotations.json',

    data\_prefix=dict(img='train/'),

    filter\_cfg=dict(filter\_empty\_gt=True, min\_size=32),

    pipeline=[

        dict(type='LoadImageFromFile', backend\_args=None),

        dict(type='LoadAnnotations', with\_bbox=True),

        dict(

            type='RandomResize',

            scale=[(1333, 400), (1333, 1200)],

            keep\_ratio=True),

        dict(type='RandomFlip', prob=0.5),

        dict(

            type='RandAugment',

            aug\_space=[[{

                'type': 'ColorTransform'

            }], [{

                'type': 'AutoContrast'

            }], [{

                'type': 'Equalize'

            }], [{

                'type': 'Sharpness'

            }], [{

                'type': 'Posterize'

            }], [{

                'type': 'Solarize'

            }], [{

                'type': 'Color'

            }], [{

                'type': 'Contrast'

            }], [{

                'type': 'Brightness'

            }]],

            aug\_num=1),

        dict(type='FilterAnnotations', min\_gt\_bbox\_wh=(0.01, 0.01)),

        dict(

            type='MultiBranch',

            branch\_field=['sup', 'unsup\_teacher', 'unsup\_student'],

            sup=dict(type='PackDetInputs'))

    ],

    backend\_args=None)

unlabeled\_dataset = dict(

    type='CocoDataset',

    data\_root='data/coco/',

    ann\_file='annotations/val\_coco\_annotations.json',

    data\_prefix=dict(img='val/'),

    filter\_cfg=dict(filter\_empty\_gt=False),

    pipeline=[

        dict(type='LoadImageFromFile', backend\_args=None),

        dict(type='LoadEmptyAnnotations'),

        dict(

            type='MultiBranch',

            branch\_field=['sup', 'unsup\_teacher', 'unsup\_student'],

            unsup\_teacher=[

                dict(

                    type='RandomResize',

                    scale=[(1333, 400), (1333, 1200)],

                    keep\_ratio=True),

                dict(type='RandomFlip', prob=0.5),

                dict(

                    type='PackDetInputs',

                    meta\_keys=('img\_id', 'img\_path', 'ori\_shape', 'img\_shape',

                               'scale\_factor', 'flip', 'flip\_direction',

                               'homography\_matrix'))

            ],

            unsup\_student=[

                dict(

                    type='RandomResize',

                    scale=[(1333, 400), (1333, 1200)],

                    keep\_ratio=True),

                dict(type='RandomFlip', prob=0.5),

                dict(

                    type='RandomOrder',

                    transforms=[

                        dict(

                            type='RandAugment',

                            aug\_space=[[{

                                'type': 'ColorTransform'

                            }], [{

                                'type': 'AutoContrast'

                            }], [{

                                'type': 'Equalize'

                            }], [{

                                'type': 'Sharpness'

                            }], [{

                                'type': 'Posterize'

                            }], [{

                                'type': 'Solarize'

                            }], [{

                                'type': 'Color'

                            }], [{

                                'type': 'Contrast'

                            }], [{

                                'type': 'Brightness'

                            }]],

                            aug\_num=1),

                        dict(

                            type='RandAugment',

                            aug\_space=[[{

                                'type': 'Rotate'

                            }], [{

                                'type': 'ShearX'

                            }], [{

                                'type': 'ShearY'

                            }], [{

                                'type': 'TranslateX'

                            }], [{

                                'type': 'TranslateY'

                            }]],

                            aug\_num=1)

                    ]),

                dict(type='RandomErasing', n\_patches=(1, 5), ratio=(0, 0.2)),

                dict(type='FilterAnnotations', min\_gt\_bbox\_wh=(0.01, 0.01)),

                dict(

                    type='PackDetInputs',

                    meta\_keys=('img\_id', 'img\_path', 'ori\_shape', 'img\_shape',

                               'scale\_factor', 'flip', 'flip\_direction',

                               'homography\_matrix'))

            ])

    ],

    backend\_args=None)

train\_dataloader = dict(

    batch\_size=5,

    num\_workers=5,

    persistent\_workers=True,

    sampler=dict(

        type='GroupMultiSourceSampler', batch\_size=5, source\_ratio=[1, 4]),

    dataset=dict(

        type='ConcatDataset',

        datasets=[

            dict(

                type='CocoDataset',

                data\_root='data/coco/',

                ann\_file='annotations/coco\_annotations.json',

                data\_prefix=dict(img='train/'),

                filter\_cfg=dict(filter\_empty\_gt=True, min\_size=32),

                pipeline=[

                    dict(type='LoadImageFromFile', backend\_args=None),

                    dict(type='LoadAnnotations', with\_bbox=True),

                    dict(

                        type='RandomResize',

                        scale=[(1333, 400), (1333, 1200)],

                        keep\_ratio=True),

                    dict(type='RandomFlip', prob=0.5),

                    dict(

                        type='RandAugment',

                        aug\_space=[[{

                            'type': 'ColorTransform'

                        }], [{

                            'type': 'AutoContrast'

                        }], [{

                            'type': 'Equalize'

                        }], [{

                            'type': 'Sharpness'

                        }], [{

                            'type': 'Posterize'

                        }], [{

                            'type': 'Solarize'

                        }], [{

                            'type': 'Color'

                        }], [{

                            'type': 'Contrast'

                        }], [{

                            'type': 'Brightness'

                        }]],

                        aug\_num=1),

                    dict(

                        type='FilterAnnotations', min\_gt\_bbox\_wh=(0.01, 0.01)),

                    dict(

                        type='MultiBranch',

                        branch\_field=['sup', 'unsup\_teacher', 'unsup\_student'],

                        sup=dict(type='PackDetInputs'))

                ],

                backend\_args=None),

            dict(

                type='CocoDataset',

                data\_root='data/coco/',

                ann\_file='annotations/val\_coco\_annotations.json',

                data\_prefix=dict(img='val/'),

                filter\_cfg=dict(filter\_empty\_gt=False),

                pipeline=[

                    dict(type='LoadImageFromFile', backend\_args=None),

                    dict(type='LoadEmptyAnnotations'),

                    dict(

                        type='MultiBranch',

                        branch\_field=['sup', 'unsup\_teacher', 'unsup\_student'],

                        unsup\_teacher=[

                            dict(

                                type='RandomResize',

                                scale=[(1333, 400), (1333, 1200)],

                                keep\_ratio=True),

                            dict(type='RandomFlip', prob=0.5),

                            dict(

                                type='PackDetInputs',

                                meta\_keys=('img\_id', 'img\_path', 'ori\_shape',

                                           'img\_shape', 'scale\_factor', 'flip',

                                           'flip\_direction',

                                           'homography\_matrix'))

                        ],

                        unsup\_student=[

                            dict(

                                type='RandomResize',

                                scale=[(1333, 400), (1333, 1200)],

                                keep\_ratio=True),

                            dict(type='RandomFlip', prob=0.5),

                            dict(

                                type='RandomOrder',

                                transforms=[

                                    dict(

                                        type='RandAugment',

                                        aug\_space=[[{

                                            'type': 'ColorTransform'

                                        }], [{

                                            'type': 'AutoContrast'

                                        }], [{

                                            'type': 'Equalize'

                                        }], [{

                                            'type': 'Sharpness'

                                        }], [{

                                            'type': 'Posterize'

                                        }], [{

                                            'type': 'Solarize'

                                        }], [{

                                            'type': 'Color'

                                        }], [{

                                            'type': 'Contrast'

                                        }], [{

                                            'type': 'Brightness'

                                        }]],

                                        aug\_num=1),

                                    dict(

                                        type='RandAugment',

                                        aug\_space=[[{

                                            'type': 'Rotate'

                                        }], [{

                                            'type': 'ShearX'

                                        }], [{

                                            'type': 'ShearY'

                                        }], [{

                                            'type': 'TranslateX'

                                        }], [{

                                            'type': 'TranslateY'

                                        }]],

                                        aug\_num=1)

                                ]),

                            dict(

                                type='RandomErasing',

                                n\_patches=(1, 5),

                                ratio=(0, 0.2)),

                            dict(

                                type='FilterAnnotations',

                                min\_gt\_bbox\_wh=(0.01, 0.01)),

                            dict(

                                type='PackDetInputs',

                                meta\_keys=('img\_id', 'img\_path', 'ori\_shape',

                                           'img\_shape', 'scale\_factor', 'flip',

                                           'flip\_direction',

                                           'homography\_matrix'))

                        ])

                ],

                backend\_args=None)

        ]))

val\_dataloader = dict(

    batch\_size=1,

    num\_workers=2,

    persistent\_workers=True,

    drop\_last=False,

    sampler=dict(type='DefaultSampler', shuffle=False),

    dataset=dict(

        type='CocoDataset',

        data\_root='data/coco/',

        ann\_file='annotations/val\_coco\_annotations.json',

        data\_prefix=dict(img='val/'),

        test\_mode=True,

        pipeline=[

            dict(type='LoadImageFromFile', backend\_args=None),

            dict(type='Resize', scale=(1333, 800), keep\_ratio=True),

            dict(

                type='PackDetInputs',

                meta\_keys=('img\_id', 'img\_path', 'ori\_shape', 'img\_shape',

                           'scale\_factor'))

        ],

        backend\_args=None))

test\_dataloader = dict(

    batch\_size=1,

    num\_workers=2,

    persistent\_workers=True,

    drop\_last=False,

    sampler=dict(type='DefaultSampler', shuffle=False),

    dataset=dict(

        type='CocoDataset',

        data\_root='data/coco/',

        ann\_file='annotations/val\_coco\_annotations.json',

        data\_prefix=dict(img='val/'),

        test\_mode=True,

        pipeline=[

            dict(type='LoadImageFromFile', backend\_args=None),

            dict(type='Resize', scale=(1333, 800), keep\_ratio=True),

            dict(

                type='PackDetInputs',

                meta\_keys=('img\_id', 'img\_path', 'ori\_shape', 'img\_shape',

                           'scale\_factor'))

        ],

        backend\_args=None))

val\_evaluator = dict(

    type='CocoMetric',

    ann\_file='data/coco/annotations/val\_coco\_annotations.json',

    metric='bbox',

    format\_only=False,

    backend\_args=None)

test\_evaluator = dict(

    type='CocoMetric',

    ann\_file='data/coco/annotations/val\_coco\_annotations.json',

    metric='bbox',

    format\_only=False,

    backend\_args=None)

detector = dict(

    type='FasterRCNN',

    data\_preprocessor=dict(

        type='DetDataPreprocessor',

        mean=[103.53, 116.28, 123.675],

        std=[1.0, 1.0, 1.0],

        bgr\_to\_rgb=False,

        pad\_size\_divisor=32),

    backbone=dict(

        type='ResNet',

        depth=50,

        num\_stages=4,

        out\_indices=(0, 1, 2, 3),

        frozen\_stages=1,

        norm\_cfg=dict(type='BN', requires\_grad=False),

        norm\_eval=True,

        style='caffe',

        init\_cfg=dict(

            type='Pretrained',

            checkpoint='open-mmlab://detectron2/resnet50\_caffe')),

    neck=dict(

        type='FPN',

        in\_channels=[256, 512, 1024, 2048],

        out\_channels=256,

        num\_outs=5),

    rpn\_head=dict(

        type='RPNHead',

        in\_channels=256,

        feat\_channels=256,

        anchor\_generator=dict(

            type='AnchorGenerator',

            scales=[8],

            ratios=[0.5, 1.0, 2.0],

            strides=[4, 8, 16, 32, 64]),

        bbox\_coder=dict(

            type='DeltaXYWHBBoxCoder',

            target\_means=[0.0, 0.0, 0.0, 0.0],

            target\_stds=[1.0, 1.0, 1.0, 1.0]),

        loss\_cls=dict(

            type='CrossEntropyLoss', use\_sigmoid=True, loss\_weight=1.0),

        loss\_bbox=dict(type='L1Loss', loss\_weight=1.0)),

    roi\_head=dict(

        type='StandardRoIHead',

        bbox\_roi\_extractor=dict(

            type='SingleRoIExtractor',

            roi\_layer=dict(type='RoIAlign', output\_size=7, sampling\_ratio=0),

            out\_channels=256,

            featmap\_strides=[4, 8, 16, 32]),

        bbox\_head=dict(

            type='Shared2FCBBoxHead',

            in\_channels=256,

            fc\_out\_channels=1024,

            roi\_feat\_size=7,

            num\_classes=80,

            bbox\_coder=dict(

                type='DeltaXYWHBBoxCoder',

                target\_means=[0.0, 0.0, 0.0, 0.0],

                target\_stds=[0.1, 0.1, 0.2, 0.2]),

            reg\_class\_agnostic=False,

            loss\_cls=dict(

                type='CrossEntropyLoss', use\_sigmoid=False, loss\_weight=1.0),

            loss\_bbox=dict(type='L1Loss', loss\_weight=1.0))),

    train\_cfg=dict(

        rpn=dict(

            assigner=dict(

                type='MaxIoUAssigner',

                pos\_iou\_thr=0.7,

                neg\_iou\_thr=0.3,

                min\_pos\_iou=0.3,

                match\_low\_quality=True,

                ignore\_iof\_thr=-1),

            sampler=dict(

                type='RandomSampler',

                num=256,

                pos\_fraction=0.5,

                neg\_pos\_ub=-1,

                add\_gt\_as\_proposals=False),

            allowed\_border=-1,

            pos\_weight=-1,

            debug=False),

        rpn\_proposal=dict(

            nms\_pre=2000,

            max\_per\_img=1000,

            nms=dict(type='nms', iou\_threshold=0.7),

            min\_bbox\_size=0),

        rcnn=dict(

            assigner=dict(

                type='MaxIoUAssigner',

                pos\_iou\_thr=0.5,

                neg\_iou\_thr=0.5,

                min\_pos\_iou=0.5,

                match\_low\_quality=False,

                ignore\_iof\_thr=-1),

            sampler=dict(

                type='RandomSampler',

                num=512,

                pos\_fraction=0.25,

                neg\_pos\_ub=-1,

                add\_gt\_as\_proposals=True),

            pos\_weight=-1,

            debug=False)),

    test\_cfg=dict(

        rpn=dict(

            nms\_pre=1000,

            max\_per\_img=1000,

            nms=dict(type='nms', iou\_threshold=0.7),

            min\_bbox\_size=0),

        rcnn=dict(

            score\_thr=0.05,

            nms=dict(type='nms', iou\_threshold=0.5),

            max\_per\_img=100)))

train\_cfg = dict(

    type='IterBasedTrainLoop', max\_iters=180000, val\_interval=5000)

val\_cfg = dict(type='TeacherStudentValLoop')

test\_cfg = dict(type='TestLoop')

param\_scheduler = [

    dict(

        type='LinearLR', start\_factor=0.001, by\_epoch=False, begin=0, end=500),

    dict(

        type='MultiStepLR',

        begin=0,

        end=180000,

        by\_epoch=False,

        milestones=[120000, 160000],

        gamma=0.1)

]

optim\_wrapper = dict(

    type='OptimWrapper',

    optimizer=dict(type='SGD', lr=0.01, momentum=0.9, weight\_decay=0.0001))

custom\_hooks = [dict(type='MeanTeacherHook')]

launcher = 'none'

work\_dir = './work\_dirs/soft-teacher\_faster-rcnn\_r50-caffe\_fpn\_180k\_semi-0.1-coco'

For more refernce look into the “semi\_det.md” file, attached to git.