

소프트웨어융합최신기술 실습과제(이재구 교수님)

: 데이터 전처리 및 Mmdetection

20191987 이세희

1. 주어진 데이터를 전처리 과정을 통하여 **COCO** 데이터 포맷으로 전환.(20점)

1) 변환 로직

<pre> "file_name": "i0303827.jpg", "width": 1936, "height": 1464, "id": 0 </pre>	<pre> "segmentation": [[]], "image_id": 0, "bbox": [540, 519, 86, 66], "category_id": 1, "id": 0, "area": 5675, "iscrowd": 0 </pre>	<pre> "id": 0, "name": "traffic_lighjt" </pre>
"images": []	"annotations": []	"categories": []

coco 데이터 포맷은 총 위처럼 3가지 정보로 이루어져 있다.

원본 데이터 포맷은 아래 사진과 같이 **annotation** 부분과 **image**부분으로 구성되어 있으며, 각 하나의 이미지에 대한 정보만 담고 있다.

원본데이터의 **annotation**부분에는 같은 카테고리의 여러 정보가 담길 수도 있고, 카테고리 아이디가 다른 여러 정보가 담길 수도 있다. 그래서 리스트로 감싸져 있는 것을 볼 수 있다.

원본데이터의 **image**의 부분에는 **annotation**의 정보들을 가져온 원본이미지에 대한 정보를 가지고 있다.

```

{
  "annotation": [
    {
      "shape": "circle",
      "color": "red",
      "kind": "normal",
      "box": [642, 530, 663, 552],
      "text": "50",
      "type": "restriction",
      "class": "traffic_sign"
    }
  ],
  "image": { "filename": "i0659222.jpg", "imsize": [1936, 1464] }
}

```

원본 데이터를 **coco** 데이터셋 포맷으로 바꾸려면 아래의 로직대로 실행한다.

1. coco 데이터에서 **categories** 부분은 원본 데이터 설명서를 참고하여 3개의 클래스로 미리 지정하고 각 카테고리에 고유 id를 부여한다.
 - a. traffic_sign: 0
 - b. traffic_light: 1
 - c. **traffic_information: 2 (사용 x)**
2. **test** 원본 데이터를 for문으로 하나씩 가져온다. (반복)
3. 원본데이터의 이미지 정보 변환
 - a. 고유 이미지 id를 부여한다.
 - b. coco 데이터 포맷에서 **images** 원소의 json 형식으로 변환한다.
 - c. coco 데이터 포맷에서 **images** 될 딕셔너리에 **append** 한다.
4. 원본데이터의 여러 **annotation**의 정보 변환 (여러개)
 - a. **여러개일 수도 있는 annotation 정보들을 딕셔너리로 변환한다. (반복)**
 - b. **annotation** 정보 for문으로 하나씩 가져온다. (**카테고리 id가 2면 건너뛰다.**)
 - c. **하나 annotation의 정보 변환(한 개)**
 - d. 하나의 **annotation**의 정보에서 가져올 정보는 아래와 같다.
 - i. 어떤 이미지의 데이터인지(위에서 부여한 이미지 id)
 - ii. 박스의 좌표값 + 연산
 - iii. 박스의 클래스(처음에 지정한 카테고리에서 id 찾아 적용)
 - e. 이걸 조건에 맞는 하나의 json 형식으로 변환한다.
 - f. coco 데이터 포맷에서 **annotations**가 될 부분에 **append** 한다.
5. 마무리 작업
 - a. 이렇게 하면 coco 데이터의 일부가 될 딕셔너리가 총 3개가 생긴다.
 - i. categories
 - ii. images
 - iii. annotations
 - b. 위의 세 딕셔너리를 각각 json 형식으로 변환하고 하나의 json 파일로 만든다.
6. train 원본 데이터에도 동일한 로직으로 json 파일을 만든다. (반복)

2) 코드 및 json 데이터

파일명: [testCoco.json](#)(Github 링크)

파일명: [trainCoco.json](#)(Github 링크)

파일명: [toCoco.py](#)(Github 링크)

```
# coding:utf-8
import json
import os

# 파일 리스트 불러오기, 0: test, 1: train
name = ["test", "train"]
```

```

originAnnoFilesDir = ["sycData\\test_road_information\\label\\",
"sycData\\train_road_information\\label\\"]
# originAnnoFilesDir = ["sample\\test\\label\\", "sample\\train\\label\\"]
originAnnoFilesList = [os.listdir(originAnnoFilesDir[0]), os.listdir(originAnnoFilesDir[1])]

# test, train 공통부분
categoriyDict = {"traffic_sign": 0, "traffic_light": 1, "traffic_information":2} # 2는 사용 x
common_categories = [
    {"id": 0, "name": "traffic_sign"},
    {"id": 1, "name": "traffic_light"},
    {"id": 2, "name": "traffic_information"}
] # 2는 사용 x

for i in range(len(originAnnoFilesList)): # 0: test, 1: train
    CocoDict = {} # coco 데이터셋이 될 딕셔너리

    # 공통부분 처리
    CocoDict["categories"] = common_categories

    # 다음 for문에서 채울 정보들
    cocoImgs = []
    cocoAnnos = []
    annotationCount = 0; # annotation 각 원소 id 부여
    for imgId in range(len(originAnnoFilesList[i])): # 원본 annotation 파일 불러오기, j는 이미지 id
        with open(originAnnoFilesDir[i] + originAnnoFilesList[i][imgId], encoding="utf-8-sig") as
originData:
            jsonData = json.load(originData) # 파싱

            # 이미지 정보 저장
            imgPath = originAnnoFilesList[i][imgId].replace(".json", ".jpg")
            images = {
                "file_name": jsonData["image"]["filename"],
                "width": jsonData["image"]["imsize"][0],
                "height": jsonData["image"]["imsize"][1],
                "id": imgId
            }
            cocoImgs.append(images)

            # 여러개 annotation 정보 하나씩 가공
            for k in range(len(jsonData["annotation"])): # 원본데이터 안에서 하나의 annotation 정보
처리
                categoryId = categoriyDict[jsonData["annotation"][k]["class"]]
                if categoryId != 2:
                    width =
jsonData["annotation"][k]["box"][2]-jsonData["annotation"][k]["box"][0]
                    height =
jsonData["annotation"][k]["box"][3]-jsonData["annotation"][k]["box"][1]
                    area = width*height
                    bbox = [

```

```

        jsonData["annotation"][k]["box"][0],
        jsonData["annotation"][k]["box"][1],
        width,
        height
    ]
    annotation = {
        "segmentation": [[]],
        "image_id": imgId,
        "bbox": bbox,
        "category_id": categoryId,
        "id": annotationCount,
        "area": area,
        "iscrowd": 0,
    }
    cocoAnnos.append(annotation)
    annotationCount += 1

CocoDict["images"] = cocoImgs # coco 데이터 images 부분
CocoDict["annotations"] = cocoAnnos # coco 데이터 annotations 부분

# json 파일 저장
with open("./" + name[i] + "Coco.json", 'w') as coco :
    json.dump(CocoDict, coco, ensure_ascii=False)
    print("finish making ./" + name[i] + "Coco.json")
    print(name[i] + "annotation count: " + str(annotationCount))
    print(name[i] + "image count: " + str(len(originAnnoFilesList[i])))

```

2. 전처리를 진행한 데이터를 토대로 **MM detection**을 활용하여 학습을 진행하여 성능을 보이시오. (**Pretrained 사용x**).(20점)

1) 코드

※ 사전 설명

코드에서 언급한 아래 변수 값에 따라 **Pretrained** 사용 여부가 달라집니다. 코드에서 아래 변수에 대한 조건문이 나오는 경우, **False** 부분을 확인해주시면 됩니다!
아래 변수에 대한 조건문이 나오는 코드블록은 구분하기 쉽게 빨간 테두리로 설정했습니다.

변수명	use_pre_trained	
Value	True	False
조건에 따른 실행	Pretrained 사용	Pretrained 미사용

```
# Check nvcc version
```

```
!nvcc -V
# Check GCC version
!gcc --version
```

```
# 구글 드라이브 데이터 불러오기
from google.colab import drive
drive.mount('/content/drive')
```

```
# 용량이 큰 데이터 압축 풀기
%cd /content/drive/MyDrive/mmdec
!unzip -qq "/content/drive/MyDrive/mmdec/train_image.zip"
```

```
# install dependencies: (use cu111 because colab has CUDA 11.1)
!pip install torch==1.9.0+cu111 torchvision==0.10.0+cu111 -f
https://download.pytorch.org/whl/torch_stable.html

# install mmdcv-full thus we could use CUDA operators
!pip install mmdcv-full -f https://download.openmmlab.com/mmdcv/dist/cu111/torch1.9.0/index.html

# Install mmdetection
!rm -rf mmdetection
!git clone https://github.com/open-mmlab/mmdetection.git
%cd mmdetection

!pip install -e .
```

```
from mmdcv import collect_env
collect_env()
```

```
# Check Pytorch installation
import torch, torchvision
print(torch.__version__, torch.cuda.is_available())

# Check MMDetection installation
import mmdet
print(mmdet.__version__)

# Check mmdcv installation
from mmdcv.ops import get_compiling_cuda_version, get_compiler_version
print(get_compiling_cuda_version())
print(get_compiler_version())
```

```
# We download the pre-trained checkpoints for inference and finetuning.
!mkdir checkpoints
!wget -c
https://download.openmmlab.com/mmdetection/v2.0/faster_rcnn/faster_rcnn_r50_caffe_fpn_mstrain_
3x_coco/faster_rcnn_r50_caffe_fpn_mstrain_3x_coco_20210526_095054-1f77628b.pth \
-O checkpoints/faster_rcnn_r50_caffe_fpn_mstrain_3x_coco_20210526_095054-1f77628b.pth
```

```
from mmdcv import Config
```

```
cfg = Config.fromfile('./configs/faster_rcnn/faster_rcnn_r50_caffe_fpn_mstrain_1x_coco.py')
```

```
from mmdet.apis import set_random_seed
```

```
cfg.dataset_type = 'CocoDataset'
```

```
cfg.classes = ('traffic_sign', 'traffic_light')
```

```
cfg.data_root = '/content/drive/MyDrive/mmdec/'
```

```
cfg.model.roi_head.bbox_head.num_classes = 2
```

```
cfg.data.test.type = 'CocoDataset'
```

```
cfg.data.test.data_root = '/content/drive/MyDrive/mmdec/'
```

```
cfg.data.test.ann_file = 'testCoCo.json'
```

```
cfg.data.test.img_prefix = 'test_image/'
```

```
cfg.data.test.classes = ('traffic_sign', 'traffic_light')
```

```
cfg.data.train.type = 'CocoDataset'
```

```
cfg.data.train.data_root = '/content/drive/MyDrive/mmdec/'
```

```
cfg.data.train.ann_file = 'trainCoco.json'
```

```
cfg.data.train.img_prefix = 'train_image/'
```

```
cfg.data.train.classes = ('traffic_sign', 'traffic_light')
```

```
cfg.data.val.type = 'CocoDataset'
```

```
cfg.data.val.data_root = '/content/drive/MyDrive/mmdec/'
```

```
cfg.data.val.ann_file = 'trainCoco.json'
```

```
cfg.data.val.img_prefix = 'train_image/'
```

```
cfg.data.val.classes = ('traffic_sign', 'traffic_light')
```

```
use_pre_trained = False
```

```
# If we need to finetune a model based on a pre-trained detector, we need to
```

```
# use load_from to set the path of checkpoints.
```

```
if (use_pre_trained): ##### pretrained 사용할 때
```

```
    cfg.load_from =
```

```
'checkpoints/faster_rcnn_r50_caffe_fpn_mstrain_3x_coco_20210526_095054-1f77628b.pth'
```

```
else: ##### pretrained 사용 안 할 때
```

```
    cfg.load_from = None
```

```
# Set up working dir to save files and logs.
```

```
cfg.work_dir = '/content/drive/MyDrive/mmdec/backup'
```

```
# The original learning rate (LR) is set for 8-GPU training.
```

```
# We divide it by 8 since we only use one GPU.
```

```
cfg.optimizer.lr = 0.02 / 8
```

```
cfg.lr_config.warmup="linear"
```

```
cfg.lr_config.warmup_iters=1000
```

```
cfg.lr_config.warmup_ratio=0.001
```

```
cfg.optimizer_config.grad_clip = dict(max_norm=35, norm_type=2)
```

```

cfg.log_config.interval = 10
cfg.lr_config.policy = 'step'
cfg.lr_config.step = 7

# Change the evaluation metric since we use customized dataset.
cfg.evaluation.metric = ['bbox']
# We can set the evaluation interval to reduce the evaluation times
cfg.evaluation.interval = 1

# We can set the checkpoint saving interval to reduce the storage cost
cfg.checkpoint_config.interval = 1
cfg.runner.max_epochs = 1

cfg.seed = 2022
set_random_seed(2022, deterministic=False)
cfg.gpu_ids = range(1)
cfg.device = 'cuda'

# We can also use tensorboard to log the training process
cfg.log_config.hooks = [
    dict(type='TextLoggerHook'),
    dict(type='TensorboardLoggerHook')]

# We can initialize the logger for training and have a look
# at the final config used for training
print(f'Config:\n{cfg.pretty_text}')

```

```

from mmdet.datasets import build_dataset
from mmdet.models import build_detector
from mmdet.apis import train_detector
import os.path as osp

# Build dataset
datasets = [build_dataset(cfg.data.train)]

# Build the detector
model = build_detector(
    cfg.model, train_cfg=cfg.get('train_cfg'), test_cfg=cfg.get('test_cfg'))
# Add an attribute for visualization convenience
model.CLASSES = datasets[0].CLASSES

# Create work_dir
mmdcv.mkdir_or_exist(osp.abspath(cfg.work_dir))
train_detector(model, datasets, cfg, distributed=False, validate=True)

```

2) Mmdetection 성능표(Non Pretrained)

```

Accumulating evaluation results...
2022-11-21 12:03:43,915 - mmdet - INFO -
Average Precision (AP) @[ IoU=0.50:0.95 | area= all | maxDets=100 ] = 0.015
Average Precision (AP) @[ IoU=0.50 | area= all | maxDets=1000 ] = 0.041
Average Precision (AP) @[ IoU=0.75 | area= all | maxDets=1000 ] = 0.008
Average Precision (AP) @[ IoU=0.50:0.95 | area= small | maxDets=1000 ] = 0.002
Average Precision (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=1000 ] = 0.041
Average Precision (AP) @[ IoU=0.50:0.95 | area= large | maxDets=1000 ] = 0.022
Average Recall (AR) @[ IoU=0.50:0.95 | area= all | maxDets=100 ] = 0.029
Average Recall (AR) @[ IoU=0.50:0.95 | area= all | maxDets=300 ] = 0.029
Average Recall (AR) @[ IoU=0.50:0.95 | area= all | maxDets=1000 ] = 0.029
Average Recall (AR) @[ IoU=0.50:0.95 | area= small | maxDets=1000 ] = 0.003
Average Recall (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=1000 ] = 0.089
Average Recall (AR) @[ IoU=0.50:0.95 | area= large | maxDets=1000 ] = 0.040

2022-11-21 12:03:43,955 - mmdet - INFO - Epoch(val) [1][2000] bbox_mAP: 0.0150, bbox_mAP_50: 0.0410, bbox_mAP_75: 0.0080, bbox_mAP_s: 0.0020,
DONE (t=0.93s).

```

3. Pretrained model을 가져와서 학습을 진행하여 성능을 보이고 결과에 대한 의견을 서술하시오. (**Pretrained 사용**).**(10점)**

1) 코드

※ 사전 설명

2번 코드에서 언급한 아래 변수(**use_pre_trained**) 값에 따라 **Pretrained** 사용 여부가 달라집니다. **2번 코드**와 동일한 부분이 많아, 아래 변수의 조건문이 있는 코드블럭(**코드 2와의 차이점**)만 제시합니다!

변수명	use_pre_trained	
Value	True	False
조건에 따른 실행	Pretrained 사용	Pretrained 미사용

```

from mmdet.apis import set_random_seed

cfg.dataset_type = 'CocoDataset'
cfg.classes = ('traffic_sign', 'traffic_light')
cfg.data_root = '/content/drive/MyDrive/mmdet/'

cfg.model.roi_head.bbox_head.num_classes = 2

cfg.data.test.type = 'CocoDataset'
cfg.data.test.data_root = '/content/drive/MyDrive/mmdet/'
cfg.data.test.ann_file = 'testCoCo.json'
cfg.data.test.img_prefix = 'test_image/'
cfg.data.test.classes = ('traffic_sign', 'traffic_light')

cfg.data.train.type = 'CocoDataset'
cfg.data.train.data_root = '/content/drive/MyDrive/mmdet/'
cfg.data.train.ann_file = 'trainCoco.json'
cfg.data.train.img_prefix = 'train_image/'
cfg.data.train.classes = ('traffic_sign', 'traffic_light')

cfg.data.val.type = 'CocoDataset'
cfg.data.val.data_root = '/content/drive/MyDrive/mmdet/'

```



```

cfg.data.val.ann_file = 'trainCoco.json'
cfg.data.val.img_prefix = 'train_image/'
cfg.data.val.classes = ('traffic_sign', 'traffic_light')

use_pre_trained = True

# If we need to finetune a model based on a pre-trained detector, we need to
# use load_from to set the path of checkpoints.
if (use_pre_trained): ##### pretrained 사용할 때
    cfg.load_from =
'checkpoints/faster_rcnn_r50_caffe_fpn_mstrain_3x_coco_20210526_095054-1f77628b.pth'
else: ##### pretrained 사용 안 할 때
    cfg.load_from = None

# Set up working dir to save files and logs.
cfg.work_dir = '/content/drive/MyDrive/mmdet/backup'

# The original learning rate (LR) is set for 8-GPU training.
# We divide it by 8 since we only use one GPU.
cfg.optimizer.lr = 0.02 / 8

cfg.lr_config.warmup="linear"
cfg.lr_config.warmup_iters=1000
cfg.lr_config.warmup_ratio=0.001
cfg.optimizer_config.grad_clip = dict(max_norm=35, norm_type=2)
cfg.log_config.interval = 10
cfg.lr_config.policy = 'step'
cfg.lr_config.step = 7

# Change the evaluation metric since we use customized dataset.
cfg.evaluation.metric = ['bbox']
# We can set the evaluation interval to reduce the evaluation times
cfg.evaluation.interval = 1

# We can set the checkpoint saving interval to reduce the storage cost
cfg.checkpoint_config.interval = 1
cfg.runner.max_epochs = 1

cfg.seed = 2022
set_random_seed(2022, deterministic=False)
cfg.gpu_ids = range(1)
cfg.device = 'cuda'

# We can also use tensorboard to log the training process
cfg.log_config.hooks = [
    dict(type='TextLoggerHook'),
    dict(type='TensorboardLoggerHook')]

# We can initialize the logger for training and have a look

```

```
# at the final config used for training
print(f'Config:\n{cfg.pretty_text}')
```

2) Mmdetection 성능표(Pretrained)

```
2022-11-21 12:41:08,734 - mmdet - INFO -
Average Precision (AP) @[ IoU=0.50:0.95 | area= all | maxDets=100 ] = 0.243
Average Precision (AP) @[ IoU=0.50 | area= all | maxDets=1000 ] = 0.435
Average Precision (AP) @[ IoU=0.75 | area= all | maxDets=1000 ] = 0.241
Average Precision (AP) @[ IoU=0.50:0.95 | area= small | maxDets=1000 ] = 0.093
Average Precision (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=1000 ] = 0.498
Average Precision (AP) @[ IoU=0.50:0.95 | area= large | maxDets=1000 ] = 0.626
Average Recall (AR) @[ IoU=0.50:0.95 | area= all | maxDets=100 ] = 0.308
Average Recall (AR) @[ IoU=0.50:0.95 | area= all | maxDets=300 ] = 0.308
Average Recall (AR) @[ IoU=0.50:0.95 | area= all | maxDets=1000 ] = 0.308
Average Recall (AR) @[ IoU=0.50:0.95 | area= small | maxDets=1000 ] = 0.151
Average Recall (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=1000 ] = 0.599
Average Recall (AR) @[ IoU=0.50:0.95 | area= large | maxDets=1000 ] = 0.708

2022-11-21 12:41:08,769 - mmdet - INFO - Epoch(val) [1][2000] bbox_mAP: 0.2430, bbox_mAP_50: 0.4350, bbox_mAP_75: 0.2410, bbox_mAP_s: 0.0930, b
DONE (t=0.56s).
```

3) 결과에 대한 의견 및 2번과의 비교

Non Pretrained 성능표

```
cfg.load_from = None
```

위처럼 코드를 설정하고 Non Pretrained로 설정하여 학습을 진행했다.

```
Accumulating evaluation results...
2022-11-21 12:03:43,915 - mmdet - INFO -
Average Precision (AP) @[ IoU=0.50:0.95 | area= all | maxDets=100 ] = 0.015
Average Precision (AP) @[ IoU=0.50 | area= all | maxDets=1000 ] = 0.041
Average Precision (AP) @[ IoU=0.75 | area= all | maxDets=1000 ] = 0.008
Average Precision (AP) @[ IoU=0.50:0.95 | area= small | maxDets=1000 ] = 0.002
Average Precision (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=1000 ] = 0.041
Average Precision (AP) @[ IoU=0.50:0.95 | area= large | maxDets=1000 ] = 0.022
Average Recall (AR) @[ IoU=0.50:0.95 | area= all | maxDets=100 ] = 0.029
Average Recall (AR) @[ IoU=0.50:0.95 | area= all | maxDets=300 ] = 0.029
Average Recall (AR) @[ IoU=0.50:0.95 | area= all | maxDets=1000 ] = 0.029
Average Recall (AR) @[ IoU=0.50:0.95 | area= small | maxDets=1000 ] = 0.003
Average Recall (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=1000 ] = 0.089
Average Recall (AR) @[ IoU=0.50:0.95 | area= large | maxDets=1000 ] = 0.040

2022-11-21 12:03:43,955 - mmdet - INFO - Epoch(val) [1][2000] bbox_mAP: 0.0150, bbox_mAP_50: 0.0410, bbox_mAP_75: 0.0080, bbox_mAP_s: 0.0020,
DONE (t=0.93s).
```

Pretrained 성능표

```
cfg.load_from =
```

```
'checkpoints/faster_rcnn_r50_caffe_fpn_mstrain_3x_coco_20210526_095054-1f77628b.pth'
```

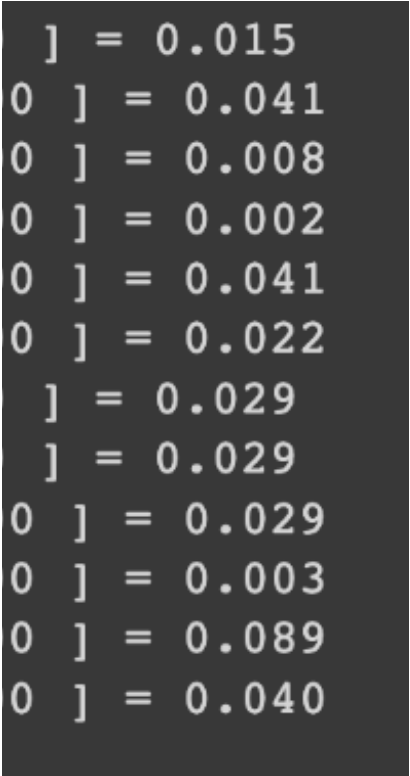
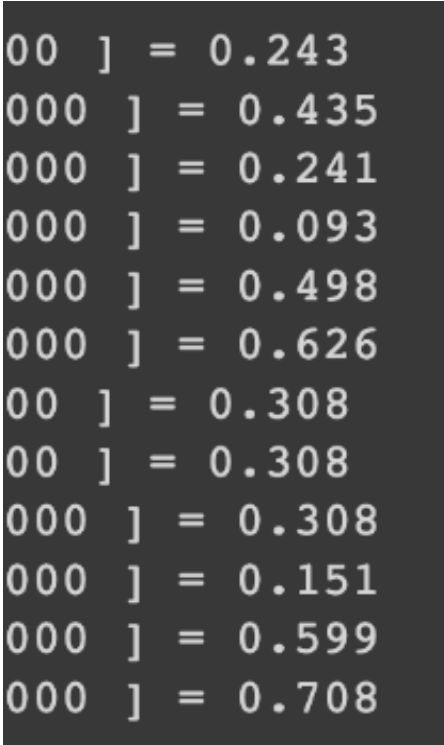
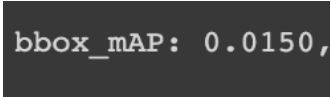
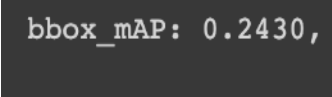
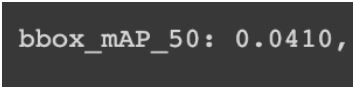

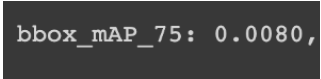
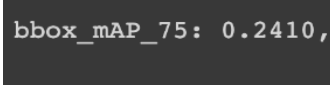
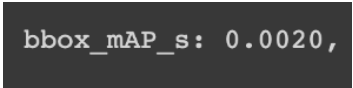
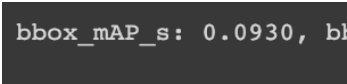
위처럼 코드를 설정하고 Pretrained로 설정하여 학습을 진행했다.

```
Accumulating evaluation results...
2022-11-21 12:41:08,734 - mmdet - INFO -
Average Precision (AP) @[ IoU=0.50:0.95 | area= all | maxDets=100 ] = 0.243
Average Precision (AP) @[ IoU=0.50 | area= all | maxDets=1000 ] = 0.435
Average Precision (AP) @[ IoU=0.75 | area= all | maxDets=1000 ] = 0.241
Average Precision (AP) @[ IoU=0.50:0.95 | area= small | maxDets=1000 ] = 0.093
Average Precision (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=1000 ] = 0.498
Average Precision (AP) @[ IoU=0.50:0.95 | area= large | maxDets=1000 ] = 0.626
Average Recall (AR) @[ IoU=0.50:0.95 | area= all | maxDets=100 ] = 0.308
Average Recall (AR) @[ IoU=0.50:0.95 | area= all | maxDets=300 ] = 0.308
Average Recall (AR) @[ IoU=0.50:0.95 | area= all | maxDets=1000 ] = 0.308
Average Recall (AR) @[ IoU=0.50:0.95 | area= small | maxDets=1000 ] = 0.151
Average Recall (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=1000 ] = 0.599
Average Recall (AR) @[ IoU=0.50:0.95 | area= large | maxDets=1000 ] = 0.708

2022-11-21 12:41:08,769 - mmdet - INFO - Epoch(val) [1][2000] bbox_mAP: 0.2430, bbox_mAP_50: 0.4350, bbox_mAP_75: 0.2410, bbox_mAP_s: 0.0930, b
DONE (t=0.56s).
```

결과에 대한 의견

Pretrained로 설정하여 학습을 진행한 경우가 Non Pretrained로 설정하여 학습을 진행한 경우보다 성능표의 수치가 전체적으로 높아진 것을 알 수 있다.

Non Pretrained	Pretrained
	
	
	
	
	

4. 참고자료

- 원본 데이터 출처
[데이터 분야 - AI 데이터 찾기 - AI-Hub \(aihub.or.kr\)](https://aihub.or.kr)
- Mm Detection 공식 문서

- 1) 시작하는 법
[mmdetection/get_started.md at master](#)
- 2) 사용자가 가진 데이터셋으로 이용하는 법
[mmdetection/2_new_data_model.md at master](#)

- 블로그

- 1) 사용법 1
[MMDetection 사용법 1\(Quick Run\)](#)
- 2) 사용법 2
[MMDetection 사용법 2\(Tutorial\)](#)