소프트웨어융합최신기술 Part2. 실습과제

20153163 김정환

1. 실행환경 설명

데이터 전처리를 위하여 Python 3.9를 사용하고 IDE로 Pycharm community edition 2022.2.1을 사용

MMDetection을 사용한 Image Detection을 위하여 구글 Colab Pro 환경을 사용

2. 주어진 데이터의 전처리 과정 및 결과

```python
import json
import collections
import glob
import os
import numpy as np
from collections import OrderedDict
from pycocotools.coco import COCO

#폴더내 json파일을 하나의 파일로 합치는 부분.
path_dir = 'data/train_road_information/label'
file_list = os.listdir(path_dir)
merged = ""
p = open('data/train_road_information/label/merged.json', 'w')
for i in file_list:
    with open(path_dir+'/'+i, 'r') as f:
        data = f.read()
        merged += data + '\n'

p.write(merged)
```

먼저 path_dir를 통해 변환하고자 하는 json데이터 파일의 위치를 지정하고

(train, test) file_list를 통해 폴더내의 파일이름 리스트를 받아온다.

Merged.json파일을 만들어서 여러 개의 json파일을 한줄씩 merged에 합친다.

```
#하나로 합친json파일을 coco파일로 변환하는 부분.
coco_group = OrderedDict()
categories = OrderedDict()
images = OrderedDict()
annotations = OrderedDict()


images = {}
annotations = {}
#category 설정
coco_group["categories"] = []
coco_group["images"] = []
coco_group["annotations"] = []

coco_group["categories"].append({
    "id": 0,
    "name": "traffic_light"
})
coco_group["categories"].append({
    "id": 1,
    "name": "traffic_sign"
})
coco_group["categories"].append({
    "id": 2,
    "name": "traffic_information"
})

```

그뒤 COCO데이터로의 변환을 위해 Dict형을 선언한뒤 그 안에 categories, images, annotation, 를 선언한뒤 3개밖에 없는 categories들을 삽입한다.

```
image_name = 'data/train_road_information/image'
find_ext = '*jpg'

images_list = []
ann_list = []
category_id = []
category_id_set = []
inner1_list = []

inner1 = {}
```

Annotation data와 image 데이터를 처리하기 위한 리스트들과 image파일들의 이름을 받아오기위한 directory를 선언해준다.

```
with open('data/train_road_information/label/merged.json', 'r+') as f:
    json_list = f.readlines()
    print(len(json_list))
#annotation 데이터 처리부분
annotation_id = 0
img_id = 0
for i in np.arange(len(json_list)):
    if( i  >= len(json_list)-1):
        break

```

한줄씩 합쳐놓았던 merged.json에서 한줄씩 읽어오고 annotation_id와 image_id를 0으로 초기화해준다.

```
for i in np.arange(len(json_list)):
    if( i  >= len(json_list)-1):
        break

    data = json.loads(json_list[i])

    # annotation Data 추출 & 가공파트
    annotation_Data = data['annotation']
    for j in np.arange(len(annotation_Data)):
        category_id = 0 # 기본은 0 traffic light
        if(annotation_Data[j]['class'] == "traffic_sign"):
            category_id = 1 # sign이면 1로 변환
        coco_group["annotations"].append({"segmentation": [[0]],
                                          "image_id": img_id,
                                          "bbox": [annotation_Data[j]["box"][0],annotation_Data[j]["box"][1],annotation_Data[j]["box"][2],annotation_Data[j]["box"][3]],
                                          "category_id": category_id,
                                          "id": annotation_id,
                                          "area": annotation_Data[j]["box"][2] * annotation_Data[j]["box"][3],#Width * Height
                                          "iscrowd": 0}) # 0으로 고정
        annotation_id = annotation_id+1
    # image_id추출 & 가공파트
    img_Data = data['image']
    coco_group["images"].append(({
        "file_name": img_Data['filename'],
        "width": img_Data['imsize'][0],
        "height": img_Data['imsize'][1],
        "id": img_id
    }))
    img_id = img_id + 1

```

Data를 통해 merged.json에서 한줄씩 받아오고 annotation은 필요한 부분대로 가공을 하고 한줄의 json에 여러 개의 annotation이 존재할 수 있으므로 for문을 하나 더 사용하여 모두 추가해준다. 마찬가지로 img_data로 image정보를 받아온뒤 가공해준다.

```

print(json.dumps(coco_group, ensure_ascii=False, indent="\t"))
with open('data/coco_road_info.json', 'w') as outfile:
    json.dump(coco_group, outfile, indent="\t")
print(len(coco_group['categories']))

```

끝으로 가공한 데이터를 coco_road_info.json파일에 써주고 출력하여 확인한다.

```
{
    "categories": [
        {
            "id": 0,
            "name": "traffic_light"
        },
        {
            "id": 1,
            "name": "traffic_sign"
        },
        {
            "id": 2,
            "name": "traffic_information"
        }
    ],
    "images": [
        {
            "file_name": "i0001205.jpg",
            "width": 1936,
            "height": 1464,
            "id": 0
        },
        {
            "file_name": "i0001206.jpg",
            "width": 1936,
            "height": 1464,
            "id": 1
        },
        {
            "file_name": "i0001207.jpg",
            "width": 1936,
            "height": 1464,
            "id": 2
        },
```


결과로 categories와 images도 알맞게 들어가고

```
            "segmentation": [
                [
                    0
                ]
            ],
            "image_id": 1422,
            "bbox": [
                1268,
                586,
                1313,
                705
            ],
            "category_id": 0,
            "id": 6083,
            "area": 925665,
            "iscrowd": 0
        },
        {
            "segmentation": [
                [
                    0
                ]
            ],
            "image_id": 1423,
            "bbox": [
                826,
                588,
                850,
                612
            ],
            "category_id": 1,
            "id": 6084,
            "area": 520200,
            "iscrowd": 0
        },
```


중간의 annotation 부분도 잘 들어간 것을 확인할 수 있다. 위의 받아오는 파일 location만 바꾸면 train, road모두 잘 되는 것을 확인할 수 있다.

3. Pretrained 모델을 사용한 학습

Customized model을 사용하기 전에 전처리를 한 COCO_Data를 이미 모델이 만들어져있는 모델로 학습시켜 결과를 확인하고자 한다.

모델은 예제에 있는 cascade_mask_rcnn을 사용하고 싶었지만 변환한 COCO에 segmentation 데이터 셋이 없어 cascade_rcnn을 사용하였다.

```
_base_ = '../cascade_rcnn/cascade_rcnn_r50_fpn_1x_coco.py'

# 1. dataset settings
dataset_type = 'CocoDataset'
classes = ('traffic_light', 'traffic_sign', 'traffic_information')
data = dict(
    samples_per_gpu=2,
    workers_per_gpu=2,
    train=dict(
        type=dataset_type,
        classes=classes,
        ann_file='train_road/label/coco_road_info.json',
        img_prefix='train_road/image'),
    val=dict(
        type=dataset_type,
        classes=classes,
        ann_file='test_road/label/coco_road_info.json',
        img_prefix='test_road/image'),
    test=dict(
        type=dataset_type,
        classes=classes,
        ann_file='test_road/label/coco_road_info.json',
        img_prefix='test_road/image'))
```

config파일을 작성해 train, var, dict의 파일위치를 지정해준뒤

```
    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.],
            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))),
# model training and testing settings
```

__base__의 RCNN model 주소로 들어가 num_classes의 숫자를 3개로 바꾸어준다.

그뒤 train.py를 참조해 train을 진행해주면 1x 모델이기에 12번의 epoch를 돌리게 된다.

```
2022-11-12 14:30:37,044 - mmdet - INFO - Epoch [1][50/962]      lr: 1.978e-03, eta: 2:03:32, time: 0.645,
2022-11-12 14:31:08,047 - mmdet - INFO - Epoch [1][100/962]     lr: 3.976e-03, eta: 2:00:37, time: 0.620,
2022-11-12 14:31:39,626 - mmdet - INFO - Epoch [1][150/962]     lr: 5.974e-03, eta: 2:00:02, time: 0.632,
2022-11-12 14:32:13,058 - mmdet - INFO - Epoch [1][200/962]     lr: 7.972e-03, eta: 2:01:14, time: 0.669,
2022-11-12 14:32:47,381 - mmdet - INFO - Epoch [1][250/962]     lr: 9.970e-03, eta: 2:02:24, time: 0.686,
2022-11-12 14:33:20,572 - mmdet - INFO - Epoch [1][300/962]     lr: 1.197e-02, eta: 2:02:17, time: 0.664,
2022-11-12 14:33:54,976 - mmdet - INFO - Epoch [1][350/962]     lr: 1.397e-02, eta: 2:02:41, time: 0.688,
2022-11-12 14:34:29,967 - mmdet - INFO - Epoch [1][400/962]     lr: 1.596e-02, eta: 2:03:07, time: 0.700,
2022-11-12 14:35:04,482 - mmdet - INFO - Epoch [1][450/962]     lr: 1.796e-02, eta: 2:03:07, time: 0.690,
2022-11-12 14:35:38,898 - mmdet - INFO - Epoch [1][500/962]     lr: 1.996e-02, eta: 2:02:59, time: 0.688,
2022-11-12 14:36:12,395 - mmdet - INFO - Epoch [1][550/962]     lr: 2.000e-02, eta: 2:02:27, time: 0.670,
2022-11-12 14:36:46,457 - mmdet - INFO - Epoch [1][600/962]     lr: 2.000e-02, eta: 2:02:05, time: 0.681,
2022-11-12 14:37:20,454 - mmdet - INFO - Epoch [1][650/962]     lr: 2.000e-02, eta: 2:01:41, time: 0.680,
2022-11-12 14:37:53,813 - mmdet - INFO - Epoch [1][700/962]     lr: 2.000e-02, eta: 2:01:05, time: 0.667,
2022-11-12 14:38:28,360 - mmdet - INFO - Epoch [1][750/962]     lr: 2.000e-02, eta: 2:00:47, time: 0.691,
2022-11-12 14:39:01,794 - mmdet - INFO - Epoch [1][800/962]     lr: 2.000e-02, eta: 2:00:11, time: 0.669,
2022-11-12 14:39:35,674 - mmdet - INFO - Epoch [1][850/962]     lr: 2.000e-02, eta: 1:59:42, time: 0.678,
2022-11-12 14:40:09,717 - mmdet - INFO - Epoch [1][900/962]     lr: 2.000e-02, eta: 1:59:13, time: 0.681,
2022-11-12 14:40:43,618 - mmdet - INFO - Epoch [1][950/962]     lr: 2.000e-02, eta: 1:58:43, time: 0.678,
2022-11-12 14:40:51,858 - mmdet - INFO - Saving checkpoint at 1 epochs
[>>] 200/200, 6.4 task/s, elapsed: 31s, ETA:     0s2022-11-12 14:41:27,968 - mmdet - INFO - Evaluating bb
Loading and preparing results...
DONE (t=0.13s)
```

첫번째 epoch후의 결과는

```
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.031
 Average Precision  (AP) @[ IoU=0.50      | area=   all | maxDets=1000 ] = 0.119
 Average Precision  (AP) @[ IoU=0.75      | area=   all | maxDets=1000 ] = 0.009
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small | maxDets=1000 ] = -1.000
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=1000 ] = -1.000
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large | maxDets=1000 ] = 0.031
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.169
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=300 ] = 0.169
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=1000 ] = 0.169
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= small | maxDets=1000 ] = -1.000
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=1000 ] = -1.000
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large | maxDets=1000 ] = 0.169
```

두번째 epoch후의 결과는

```
2022-11-12 14:53:03,897 - mmdet - INFO -
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.059
 Average Precision  (AP) @[ IoU=0.50      | area=   all | maxDets=1000 ] = 0.198
 Average Precision  (AP) @[ IoU=0.75      | area=   all | maxDets=1000 ] = 0.020
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small | maxDets=1000 ] = -1.000
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=1000 ] = -1.000
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large | maxDets=1000 ] = 0.059
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.191
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=300 ] = 0.191
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=1000 ] = 0.191
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= small | maxDets=1000 ] = -1.000
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=1000 ] = -1.000
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large | maxDets=1000 ] = 0.191
```

매 epoch마다 조금씩 검출률의 향상이 이루어 지고 있는 것을 확인할 수 있다.

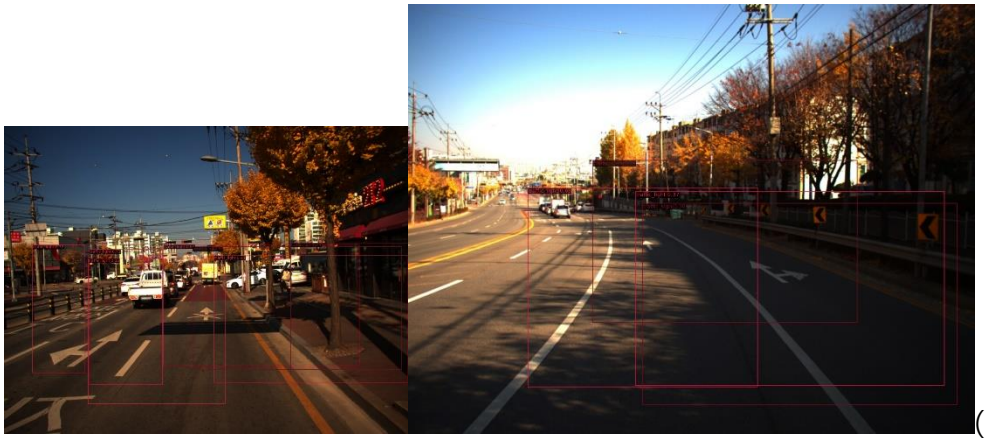
```
!python tools/test.py ₩
    configs/COCO_Road/rcnn_r50_caffe_fpn_mstrain-poly_1x_COCO.py ₩
    work_dirs/rcnn_r50_caffe_fpn_mstrain-poly_1x_COCO/latest.pth ₩
    --eval bbox segm ₩
    --show-dir results/rcnn_r50_caffe_fpn_mstrain-poly_1x_COCO

/usr/local/lib/python3.7/dist-packages/mmcv/__init__.py:21: UserWarning: On January 1, 2023, MMCV will release v2.0.0, in which it will remove components
  'On January 1, 2023, MMCV will release v2.0.0, in which it will remove '
/content/mmdetection/mmdet/utils/setup_env.py:39: UserWarning: Setting OMP_NUM_THREADS environment variable for each process to be 1 in default, to avoid
  f'Setting OMP_NUM_THREADS environment variable for each process '
/content/mmdetection/mmdet/utils/setup_env.py:49: UserWarning: Setting MKL_NUM_THREADS environment variable for each process to be 1 in default, to avoid
  f'Setting MKL_NUM_THREADS environment variable for each process '
loading annotations into memory...
Done (t=0.01s)
creating index...
index created!
load checkpoint from local path: work_dirs/rcnn_r50_caffe_fpn_mstrain-poly_1x_COCO/latest.pth
[                                                  ] 0/200, elapsed: 0s, ETA:/usr/local/lib/python3.7/dist-packages/torch/nn/functional.py:718: UserWarni
  return torch.max_pool2d(input, kernel_size, stride, padding, dilation, ceil_mode)
[>>] 200/200, 1.5 task/s, elapsed: 135s, ETA:     0s
Evaluating bbox...
Loading and preparing results...
DONE (t=0.00s)
creating index...
index created!
Running per image evaluation...
Evaluate annotation type *bbox*
DONE (t=0.23s).
Accumulating evaluation results...
DONE (t=0.04s).

 Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.136
 Average Precision  (AP) @[ IoU=0.50      | area=   all | maxDets=1000 ] = 0.312
 Average Precision  (AP) @[ IoU=0.75      | area=   all | maxDets=1000 ] = 0.103
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small | maxDets=1000 ] = -1.000
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=1000 ] = -1.000
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large | maxDets=1000 ] = 0.136
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.242
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=300 ] = 0.242
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=1000 ] = 0.242
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= small | maxDets=1000 ] = -1.000
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=1000 ] = -1.000
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large | maxDets=1000 ] = 0.242
```

최종결과가 나왔는데 결과 폴더의 이름은 rcnn_r50_caffe로 되어있지만 실제론 cascade_rcnn을 돌린것이고 예제를 따라하느라 이름만 저렇게 된것이다.

Precision과 Recall에서 Epoch횟수가 증가할수록 유의미한 향상이 이루어졌으며 IoU50 area=all에서 31.2%의 Precision을 기록했으며 IoU에서 13.6%를 기록했다. 다만 Precision보다는 Recall의 확률이 적었는데, Cascade모델에서는 검출되어야 할것이 검출되어야 할 확률인 Recall의 학습이 조금 저조한 것을 보였다.

 (결과사진)

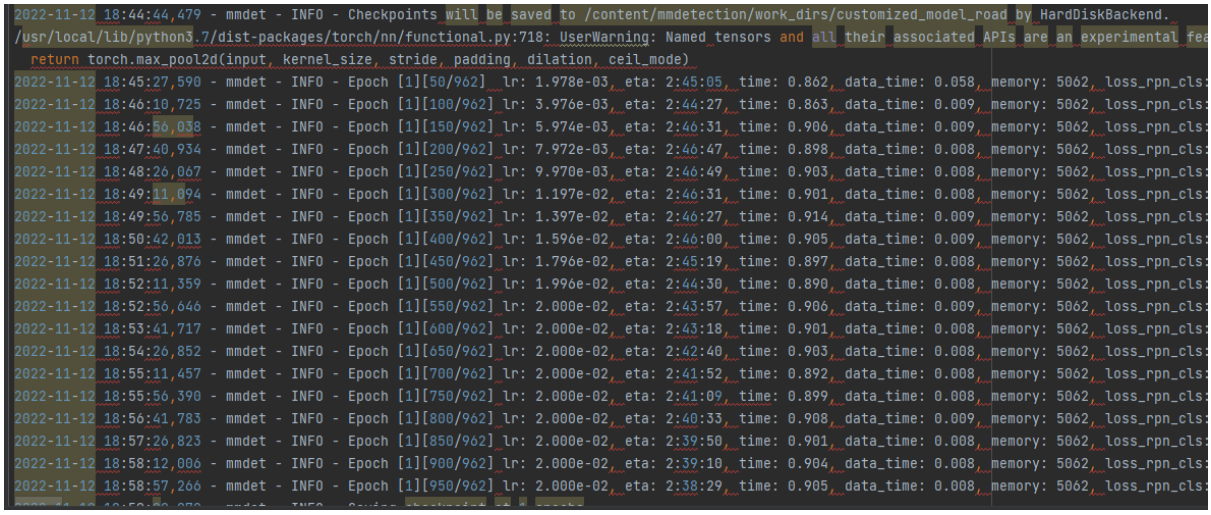
4. Customized Model의 학습

실제 Customized된 학습 Model을 처음부터 작성하기는 무리가 있다고 판단되어 요인에 의한 결과를 바꾸기 위해 같은 모델인 Cascade_RCNN모델의 요소를 바꾸어 학습해 어떤 변화가 있는지를 판단하고자 했다.

```
# model settings
model = dict(
    type='CascadeRCNN',
    backbone=dict(
        type='ResNet',
        depth=101,      #50 > 101으로 변경
        num_stages=4,
        out_indices=(0, 1, 2, 3),
        frozen_stages=1,
        norm_cfg=dict(type='BN', requires_grad=True),
        norm_eval=True,
        style='pytorch',
        init_cfg=dict(type='Pretrained', checkpoint='torchvision://resnet50')),
)
```

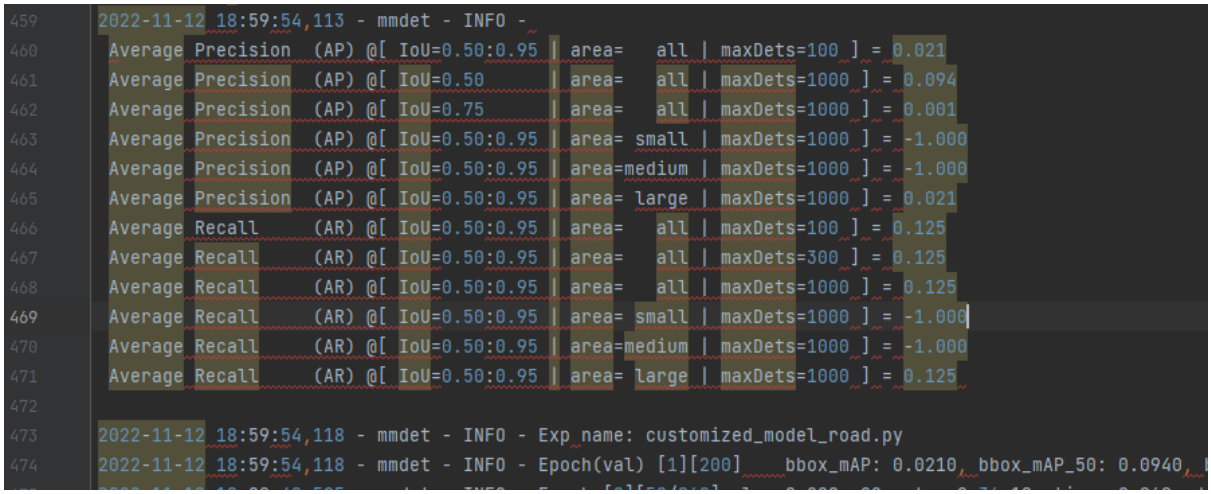
Cascade_RCNN의 depth는 특정값만을 가질 수 있기에 custom config파일에서 backbone부분을 불러와서 depth를 50이 아닌 101로 overwrite한 모델을 가지고 학습을 진행하였다.

실제 depth 수가 늘어났기 때문에 걸리는 시간도 오래걸리고 학습율은 높아질 것 이라고 예측했는데 실제로



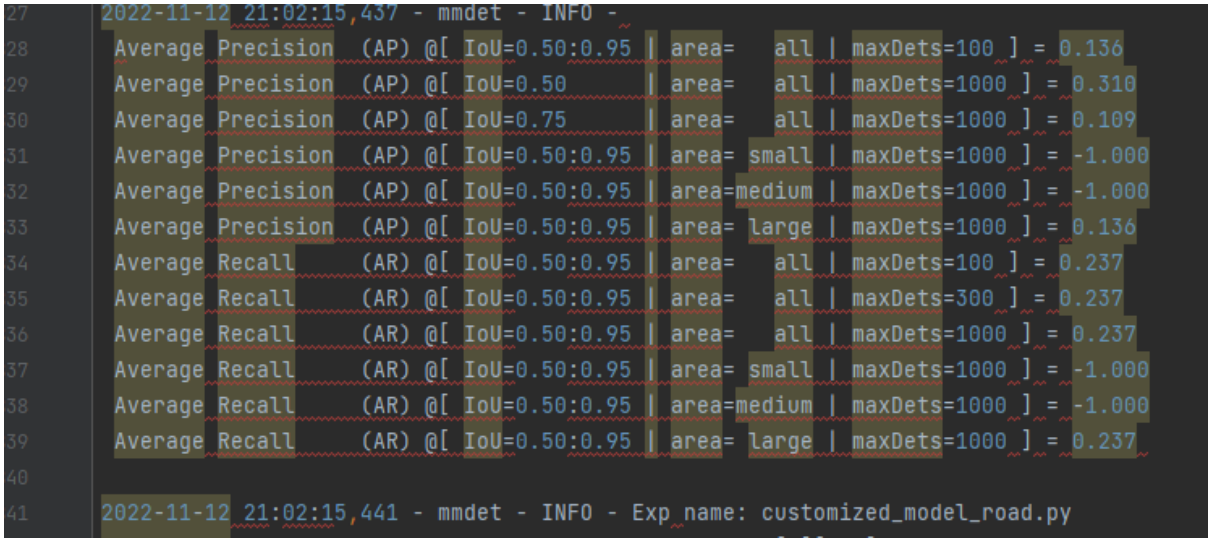
첫번째 epoch의 eta부터 depth50일 당시는 2:00이었지만 2:45초로 45초 가량 늘어난 것을 볼 수 있다.

하지만 학습률에서는 depth50이 IoU50 area=all에서 0.119를 기록했던것에 비해서



0.094를 기록하며 depth수가 증가한다고 선형적으로 AI의 학습률이 증가 하지 않않는 것 보였다.

하지만 Epoch10회에서 이미



pretrained모델에서 기록했던 결과보다 향상된 결과를 보여주었다.

5. 결론

Depth의 차등을 두어 두가지 모델의 결과를 비교하자면 Pretrained모델의 학습률 자체가 31퍼센트대로 낮았기 때문에 depth수를 두배로 늘린다고 overfitting과 같은 문제는 나타나기 어려웠을 것이다. 따라서 단순히 depth수를 늘리는 것만으로도 학습률의 성장을 이루어 낼 수 있었다.