Detectron2 를 활용한 Object detection 기술 연구

요 약

최근 AI 의 발전과 함께 머신러닝, 딥러닝 등의 학습 방식이 발전하면서 객체 인식 분야도 함께 진보하고 있다. 오늘날 정보화 사회에서 객체 인식 기술은 우리의 삶 모든 부분에 영향을 미치게 될 것이며 많은 분야에서 활용되고 사용될 것이다. 이에 본 논문에서는 객체 인식 기술을 활용하여 차량번호판을 검출해보고자 한다. 현재 많은 종류의 객체인식기술이 있는데 FAIR 에서 개발한 Pytorch 기반의 Object Detection, Segmentation 라이브러리인 Detectron2 를 활용하여 논문을 진행해보고자 한다. 따라서 Detectron2 에 대하여 알아보고 난 후 Detectron2 의 기술을 활용하여 자동차의 번호판을 검출할 것이다.

1. 서론

1.1. 연구배경

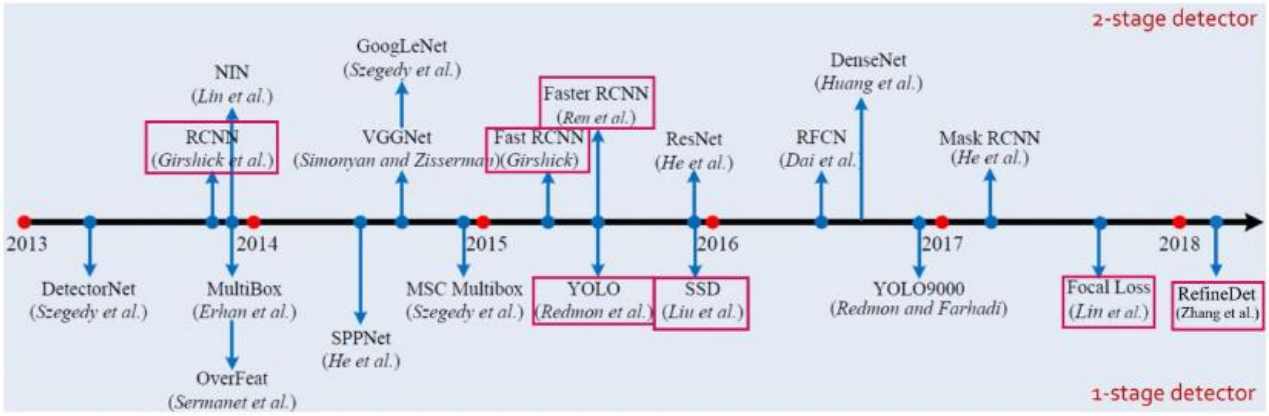
오늘날 정보통신기술의 발전으로 인해 하루에도 수많은 양의 영상들이 발생하고 이를 가공 처리하는 과정이 발생한다. 가공 처리하는 일을 사람이 아닌 컴퓨터가 대신 처리함으로써 우리는 많은 양의 데이터를 빠른 속도로 활용할 수 있었다. 그 중에서 객체 인식은 사람이 필요로 하는 시각 정보를 머신러닝, 딥러닝을 통해 컴퓨터가 대신하여 분석하고 인식하게 하는 분야이다. 객체 인식은 얼굴 인식, IoT, 자율주행, 보안 등의 다양한 곳에서 활용되고 있으며 현대에 중요한 기술로 여겨지고 있다. 지금까지 많은 객체 인식 기술이 소개되었고 발전되어왔다. 그 중에서 FAIR 에서 개발한 Pytorch 기반의 Object Detection, Segmentation 라이브러리인 Detectron2 에 대해서 알게 되었고 관심이 생겨 이를 활용하여 객체 인식 기술에 대하여 알아보고 검출하고 싶은 객체에 대해서 Detectron2 을 활용하여 Mask R-CNN 을 학습시켜 보는 식으로 이번 논문을 진행해볼 것이다.

1.2. 연구목표

본 프로젝트의 연구 목표는 객체인식기술의 원리에 대하여 알아보고 Detectron2 가 기존에 인식할 수 있는 객체들에 대하여 알아본 후, Detectron2 를 활용하여 차량의 번호판을 인식할 수 있게 Mask R-CNN 을 학습을 시킨 후, Loss function graph 를 통해 iteration 횟수를 비교해보며 Overfitting 되지 않는 선에서 학습이 잘 되었는지 확인하는 것까지가 이번 연구의 목표이다.

2. 관련연구

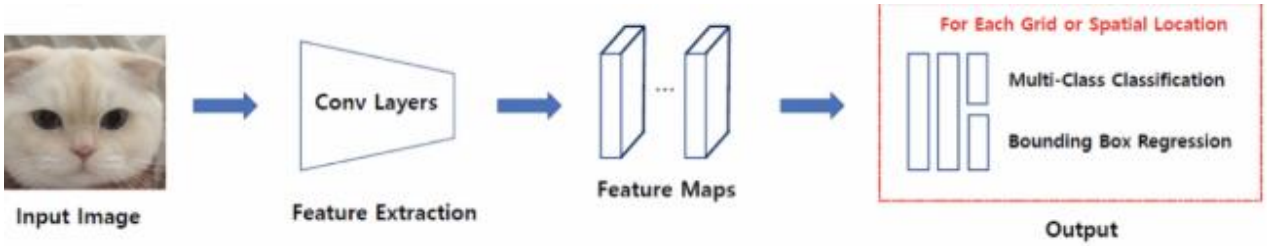
2.1. 1-Stage detector & 2-Stage dector



[그림 1] 1-Stage detector & 2-Stage detector

1- Stage detector

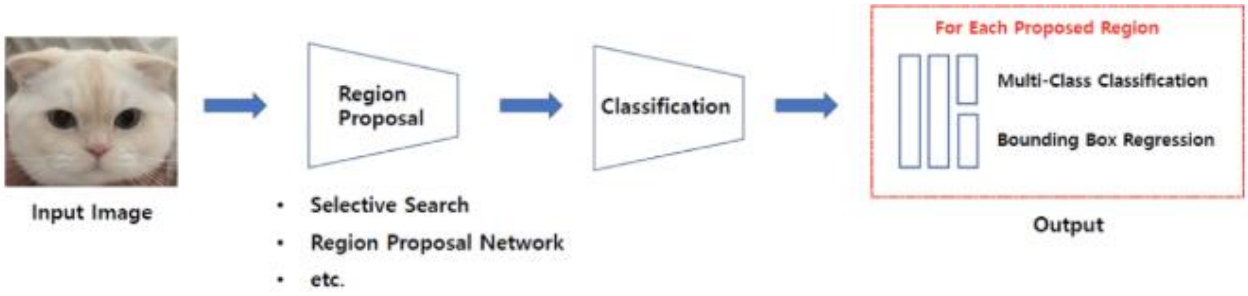
Regional proposal 과 classification 이 동시에 이루어진다.



[그림 2] 1-Stage detector process

2- Stage detector

Regional proposal 과 classification 이 순차적으로 이루어진다.



[그림 3] 2-Stage detector process

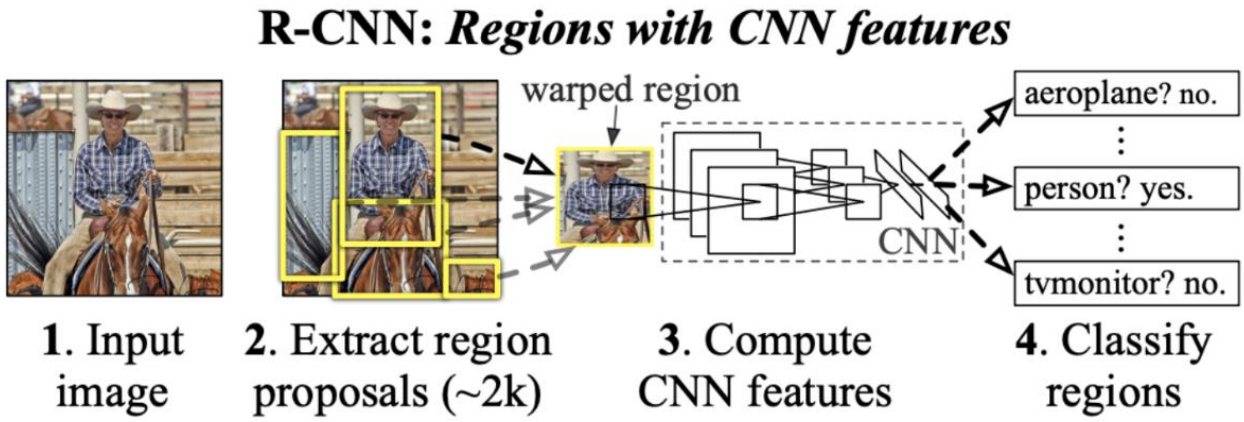
2.2. RCNN

R-CNN은 Image classification을 수행하는 CNN과 localization을 위한 regional proposal 알고리즘을 연결한 모델이다.

RCNN 의 과정은

1. Image를 입력 받는다.
2. Selective search 알고리즘에 의해 regional proposal output 약 2000개를 추출한다.
추출한 regional proposal output을 모두 동일 input size로 만들어주기 위해 warp해준다.
(이전의 Sliding window 방식은 너무 비효율적이어서 Selective search 알고리즘을 사용)
3. 2000개의 warped image를 각각 CNN 모델에 넣는다. 그리고 CNN모델에 들어가 feature vector를 뽑고 각각의 class마다 SVM로 classification을 수행한다.
4. 각각의 Convolution 결과에 대해 classification을 진행하여 결과를 얻는다.

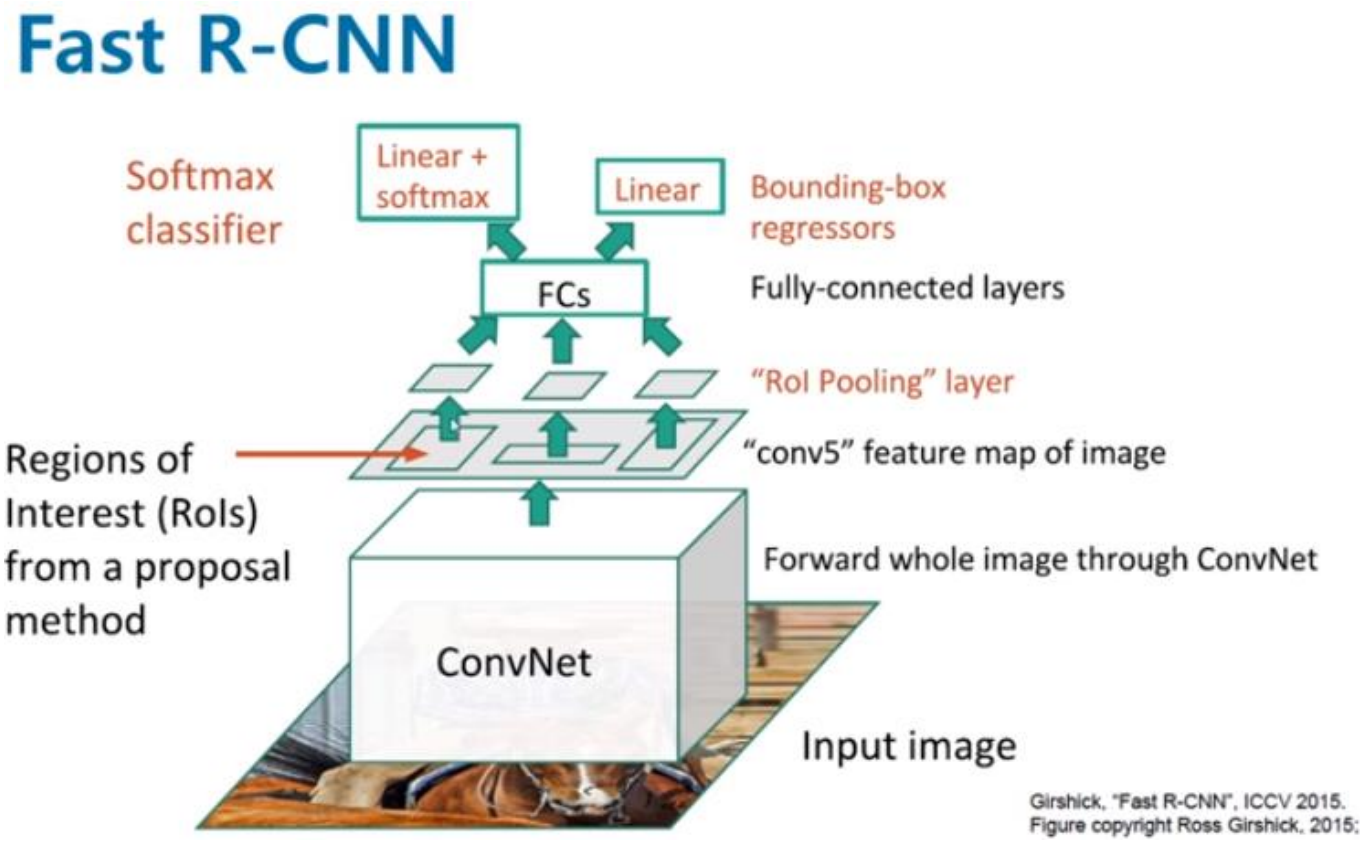
[1]



[그림 4] R-CNN Object detection system overview

RCNN은 selective search로 2000개의 region proposal을 뽑고 각 영역마다 CNN을 수행하기 때문에 CNN연산 * 2000 만큼의 시간이 걸려 수행시간이 매우 느리다.
이것을 Fast RCNN에서 RoI Pooling으로 보완하게 된다.

2.3. Fast RCNN

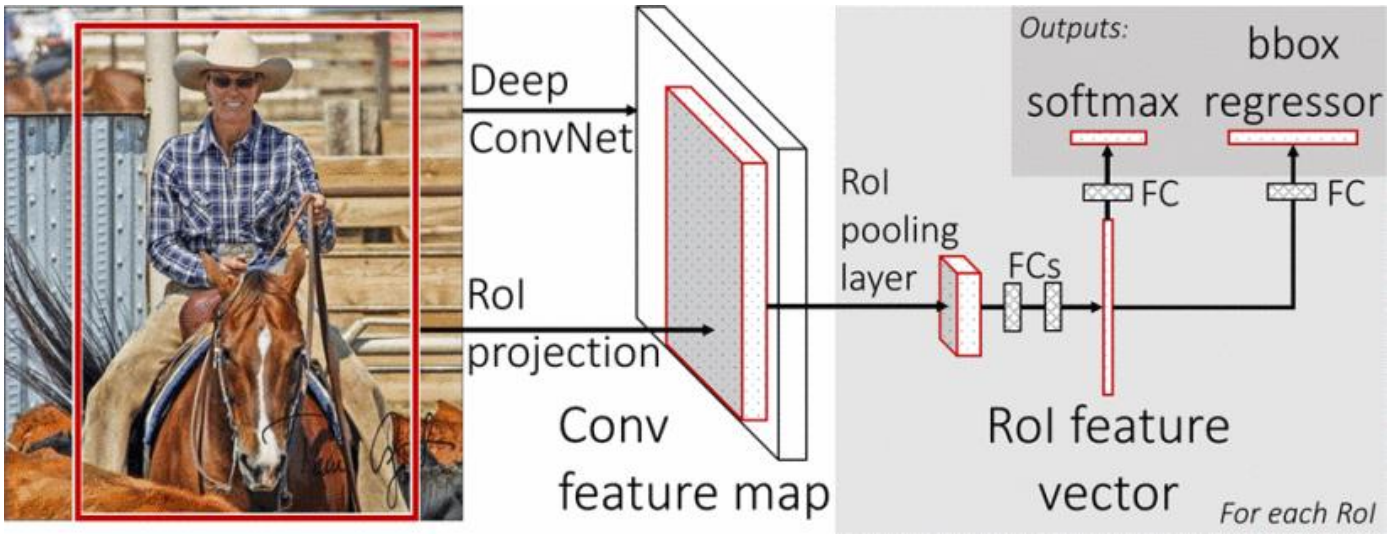


[그림 5] Fast R-CNN system overview

Fast RCNN 의 과정은

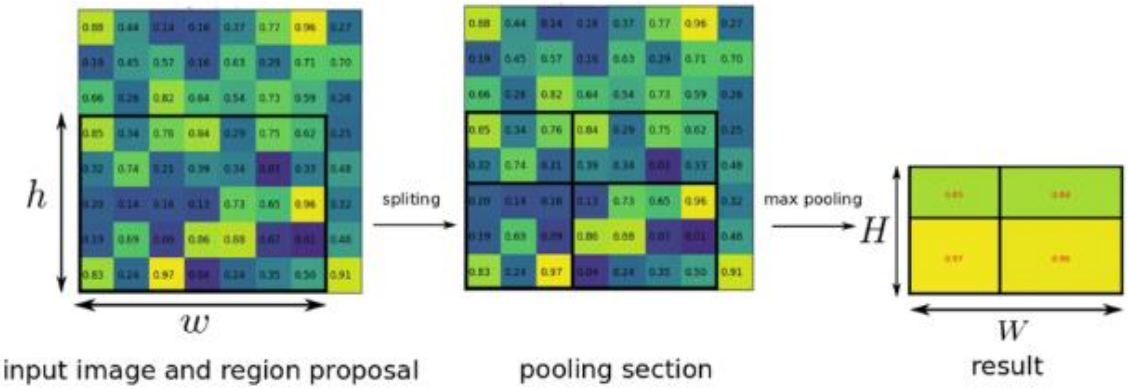
1. R-CNN에서와 마찬가지로 Selective Search를 통해 RoI를 찾는다. 그리고 전체 이미지를 CNN에 통과시켜 feature map을 추출한다.
2. Selective Search로 찾았던 RoI를 feature map크기에 맞춰서 projection시킨다.
3. projection시킨 RoI에 대해 RoI Pooling을 진행하여 고정된 크기의 feature vector를 얻는다.
4. feature vector는 FC layer를 통과한 뒤, 구 브랜치로 나뉘게 된다.
5. 하나는 softmax를 통과하여 RoI에 대해 object classification을 한다. 그리고 bounding box regression을 통해 selective search로 찾은 box의 위치를 조정한다.

[2]



[그림 6] Fast R-CNN architecture

RoI Pooling



[그림 7] RoI overview

Fast R-CNN에서 먼저 입력 이미지를 CNN에 통과시켜 feature map을 추출한다.
그 후 이전에 미리 Selective search로 만들어놨던 RoI(=region proposal)을 feature map에 projection시킨다.

위 그림의 가장 좌측 그림이 feature map이고 그 안에 검은색 box가 투영된 RoI이다.

미리 설정한 HxW크기로 만들어주기 위해서 (h/H) * (w/H) 크기만큼 grid를 RoI위에 만든다.
RoI를 grid크기로 split시킨 뒤 max pooling을 적용시켜 결국 각 grid 칸마다 하나의 값을 추출한다.
위 작업을 통해 feature map에 투영했던 검은색 박스크기의 RoI는 result 크기의 고정된 feature vector로 변환된다.

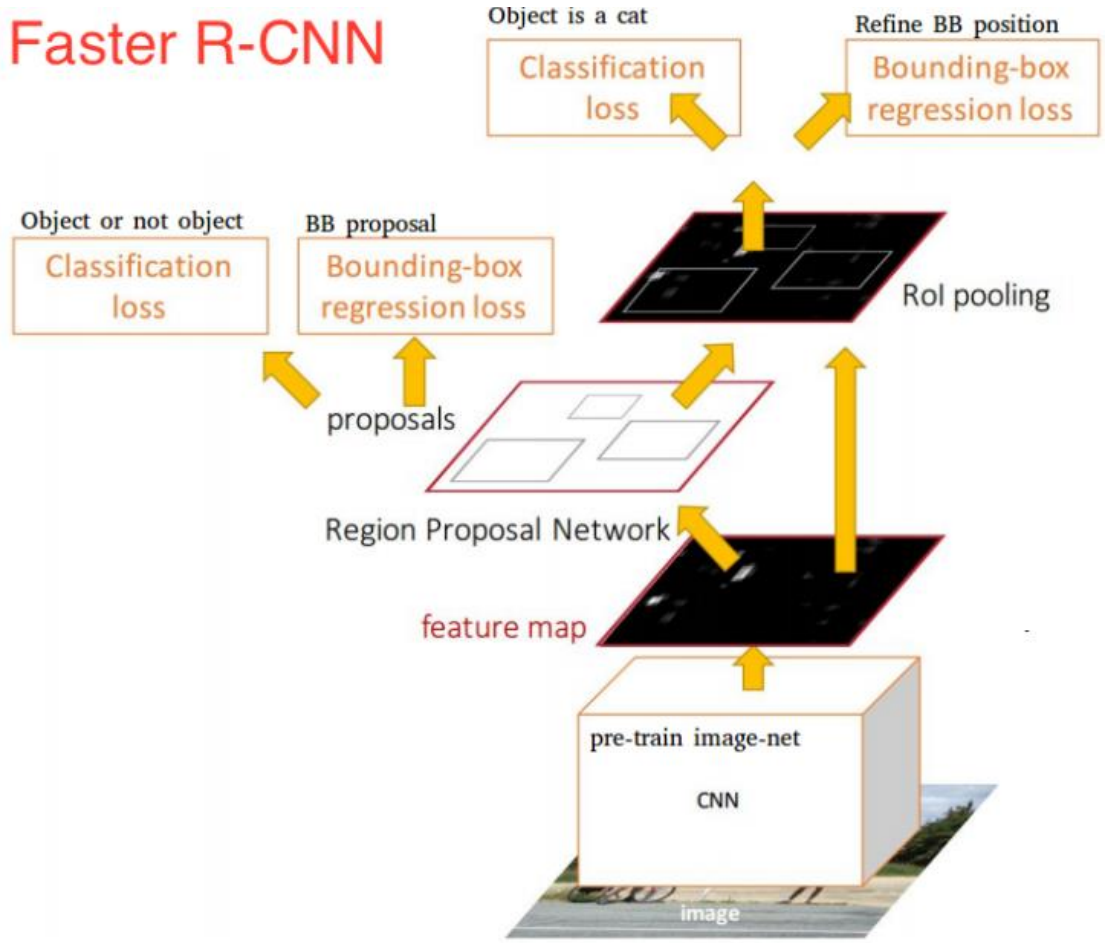
이렇게 RoI pooling을 이용함으로써
원래 이미지를 CNN에 통과시킨 후 나온 feature map에 이전에 생성한 RoI를 projection시키고
이 RoI를 FC layer input 크기에 맞게 고정된 크기로 변형할 수가 있다
따라서 더이상 2000번의 CNN연산이 필요하지 않고 1번의 CNN연산으로 속도를 대폭 높일 수 있다.
Fast RCNN의 경우도 RCNN처럼 RoI를 만드는 Selective search 알고리즘이 CNN외부에서 진행되는데 이것을 내부에서 하는 Faster RCNN이 나오게 된다.

2.4. Faster RCNN

Faster R-CNN은 Fast R-CNN에 RPN 이 결합되었다고 할 수 있다.
Faster R-CNN은 Fast R-CNN구조에서 conv feature map과 RoI Pooling사이에 RoI를 생성하는 Region Proposal Network가 추가된 구조이다.

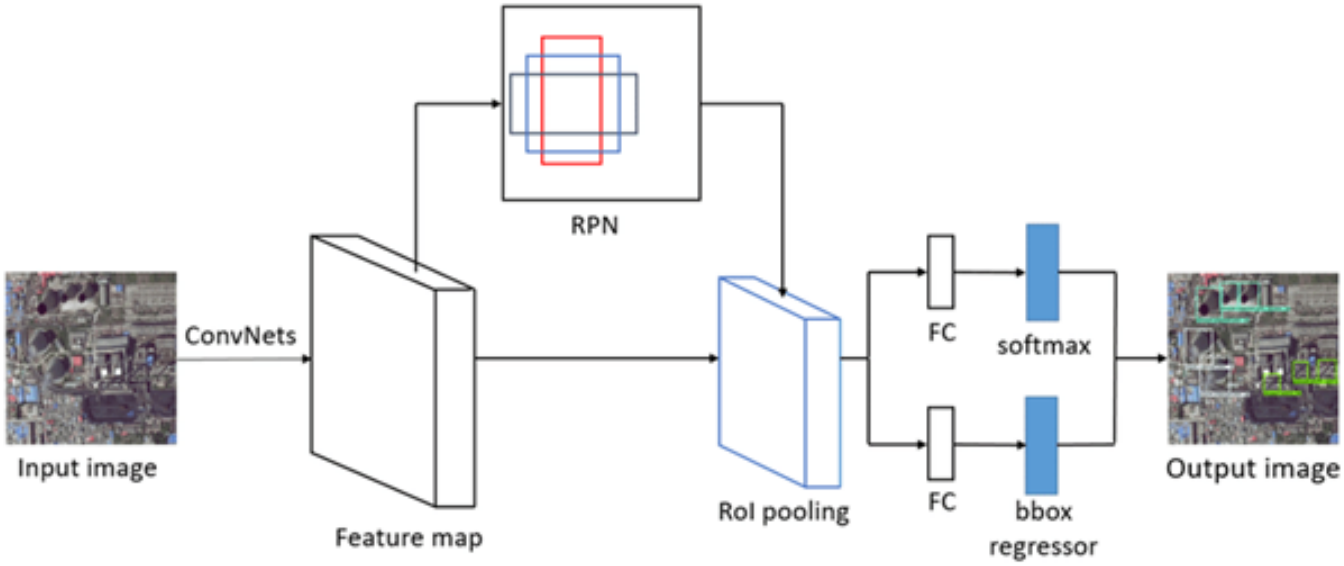
그리고 Faster R-CNN에서는 RPN 네트워크에서 사용할 CNN과 Fast R-CNN에서 classification, bounding box regression을 위해 사용한 CNN 네트워크를 공유하자는 개념에서 나왔다.

[3]



[그림 8] Faster R-CNN is a single, unified network for object detection

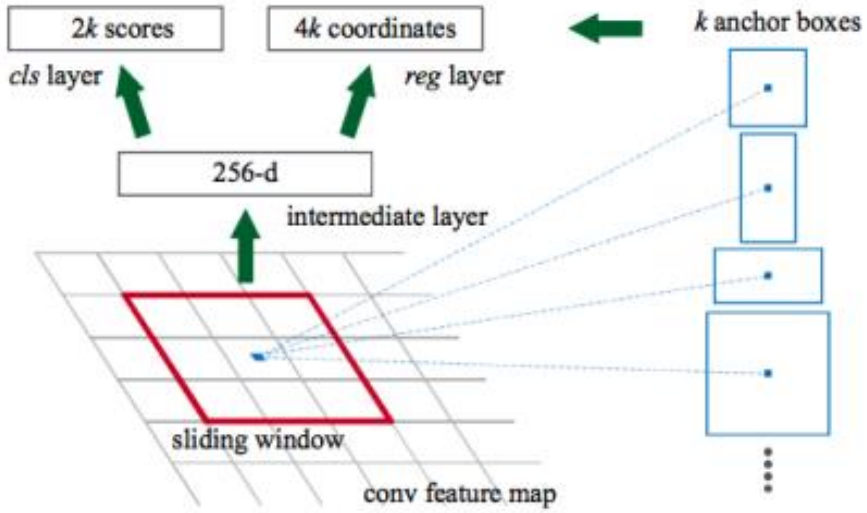
위 그림에서와 같이 CNN을 통과하여 생성된 conv feature map이 RPN에 의해 RoI를 생성한다.
주의해야할 것이 생성된 RoI는 feature map에서의 RoI가 아닌 original image에서의 RoI이다.
따라서 original image위에서 생성된 RoI는 conv feature map의 크기에 맞게 rescaling된다.
feature map 에 RoI 가 투영되고 나면 FC layer 에 의해 classification 과 bounding box regression 이 수행된다.



[그림 9] RoI & Fc layer

위 그림에서 보다시피 마지막에 FC layer를 사용하기에 input size를 맞춰주기 위해 RoI pooling을 사용한다. RoI pooling을 사용하니까 RoI들의 size가 달라도 되는 것처럼 original image의 input size도 달라도 된다.

RPN



[그림 10] RPN

RPN의 input 값은 이전 CNN 모델에서 뽑아낸 feature map이다.
Region proposal을 생성하기 위해 feature map위에 nxn window를 sliding window시킨다.
이때, object의 크기와 비율이 어떻게 될지모르므로 k개의 anchor box를 미리 정의해놓는다.
이 anchor box가 bounding box가 될 수 있는 것이고 미리 가능할만한 box모양 k개를 정의해놓는 것이다.
여기서는 가로세로길이 3종류 x 비율 3종류 = 9개의 anchor box를 이용한다.

이 단계에서 9개의 anchor box를 이용하여 classification과 bbox regression을 먼저 구한다.
CNN에서 뽑아낸 feature map에 대해 3x3 conv filter 256개를 연산하여 depth를 256으로 만든다.
그 후 1x1 conv 두개를 이용하여 각각 classification과 bbox regression을 계산한다.
RPN에서 이렇게 1x1 convolution을 이용하여 classification과 bbox regression을 계산하는데
이때 네트워크를 가볍게 만들기 위해 binary classification으로 bbox에 물체가 있나 없나만 판단한다. 무슨 물체인지 classification하는 것은 마지막 classification 단계에서 한다.
RPN단계에서 classification과 bbox regression을 하는 이유는 결국 학습을 위함이다.
위 단계로부터 positive / negative examples들을 뽑아내는데 다음 기준에 따른다.

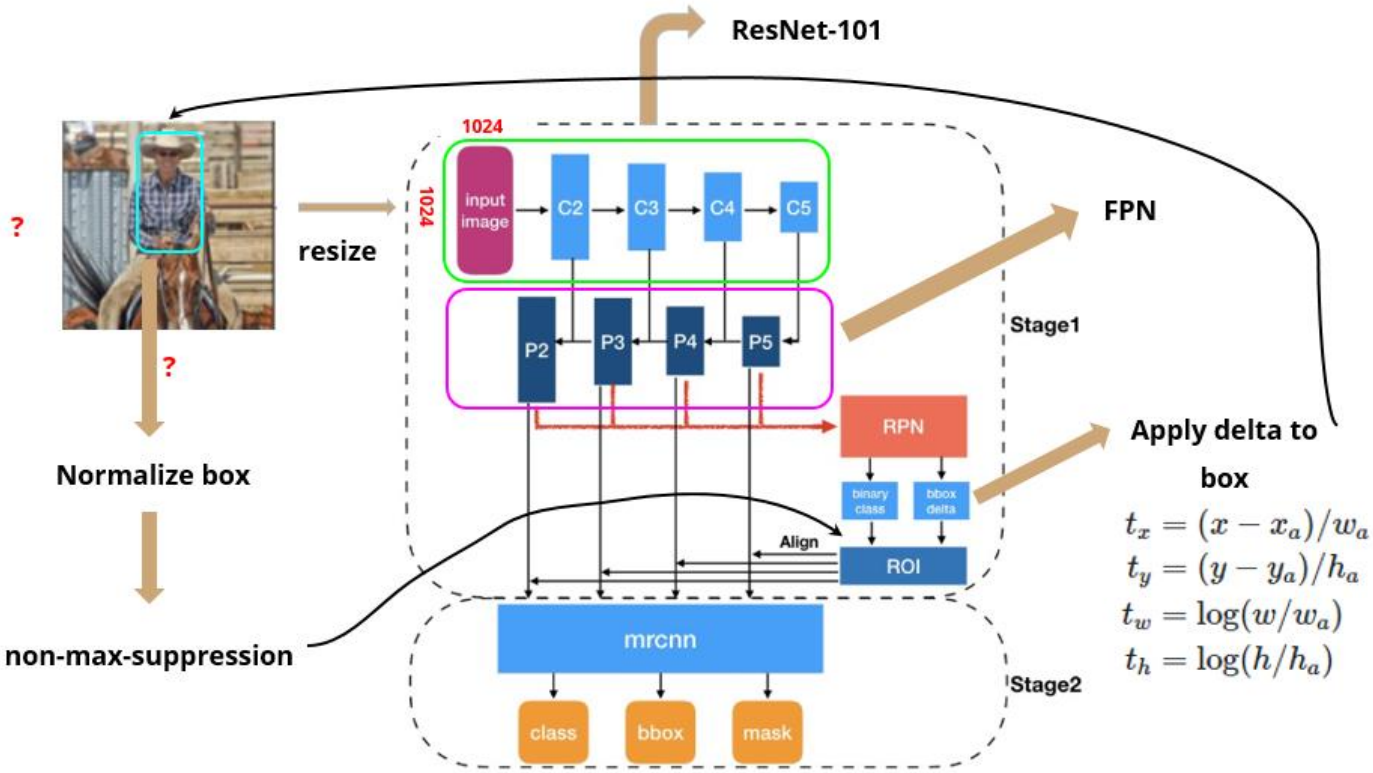
$$p^* = \begin{cases} 1 & \text{if } IoU > 0.7 \\ -1 & \text{if } IoU < 0.3 \\ 0 & \text{if otherwise} \end{cases}$$

IoU가 0.7보다 크거나, 한 지점에서 모든 anchor box중 가장 IoU가 큰 anchor box는 positive example로 만든다.
IoU가 0.3보다 작으면 object가 아닌 background를 뜻하므로 negative example로 만들고
이 사이에 있는 IoU에 대해서는 애매한 값이므로 학습 데이터로 이용하지 않는다.

Faster R-CNN에 대한 학습이 완료된 후 RPN모델을 예측시키며 하마 한 객체당 여러 proposal값이 나올 것이다.

이 문제를 해결하기 위해 NMS알고리즘을 사용하여 proposal의 개수를 줄인다.
NMS알고리즘은 다음과 같다.
1. box들의 score(confidence)를 기준으로 정렬한다.
2. score가 가장 높은 box부터 시작해서 다른 모든 box들과 IoU를 계산해서 0.7이상이면 같은 객체를 detect한 box라고 생각할 수 있기 때문에 해당 box는 지운다.
3. 최종적으로 각 object별로 score가 가장 높은 box 하나씩만 남게 된다.

2.5. Mask RCNN



[그림 11] Mask RCNN architecture

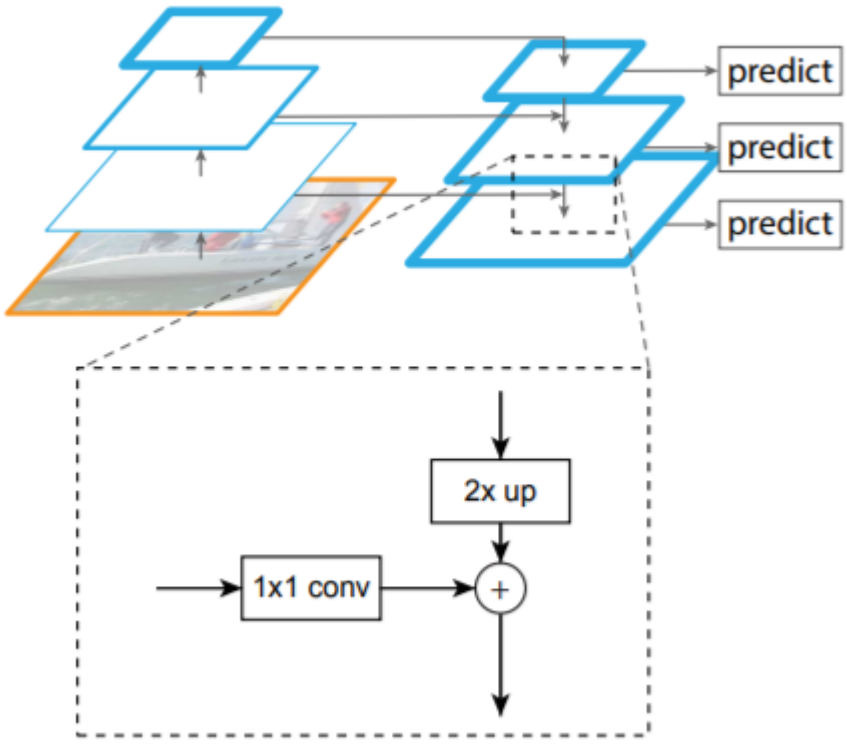
Input image resize

Mask R-CNN에서는 backbone으로 ResNet-101을 사용하는데 ResNet 네트워크에서는 이미지 input size가 800~1024일때 성능이 좋으므로 input image를 이 사이즈로 맞춰준다.
이렇게 resize를 한 후 네트워크의 input size인 1024 x 1024로 맞춰주기 위해
나머지 값들은 zero padding으로 값을 채워준다.
그리고 [4]Mask R-CNN에서는 Backbone으로 ResNet모델을 사용한다.

FPN

FPN에서는 위 그림과 같이 마지막 layer의 feature map에서 점점 이전의 중간 feature map들을 더하면서 이전 정보까지 유지할 수 있도록 한다. 이렇게 함으로써 더이상 여러 scale값으로 anchor를 생성할 필요가 없게 되고 모두 동일한 scale의 anchor를 생성한다. 따라서 작은 feature map에서는 큰 anchor를 생성하여 큰 object를, 큰 feature map에서는 다소 작은 anchor를 생성하여 작은 object를 detect할 수 있도록 설계되었다.

마지막 layer에서의 feature map에서 이전 feature map을 더하는 것은 Upsampling을 통해 이루어진다.



[그림 12] Feature map 관련

먼저 2배로 upsampling을 한 후 이전 layer의 feature map을 1x1 Fully convolution 연산을 통해 filter개수를 똑같이 맞춰준후 더함으로써 새로운 feature map을 생성한다.

RPN

위의 과정을 통해 생성된 내용들을 각각 RPN 모델에 전달하는데 Faster R-CNN 와 달리 이제 각 feature map 에서 1 개 scale 의 anchor 를 생성하므로 결국 각 pyramid feature map 마다 scale 1 개 x ratio 3 개 = 3 개의 anchor 를 생성한다.
RPN 을 통해 output 으로 classification 값, bbox regression 값이 나오는데 이때 bbox regression 값은 delta 값이다.

$$
\begin{aligned}
t_x &= (x - x_a)/w_a \\
t_y &= (y - y_a)/h_a \\
t_w &= \log(w/w_a) \\
t_h &= \log(h/h_a)
\end{aligned}
$$

-.

$$
\begin{aligned}
t_x^* &= (x^* - x_a)/w_a \\
t_y^* &= (y^* - y_a)/h_a \\
t_w^* &= \log(w^*/w_a) \\
t_h^* &= \log(h^*/h_a)
\end{aligned}
$$

t_x, t_y : 박스의 center coordinates
t_w, t_h : 박스의 width, height
x, y, w, h : predicted box
x_a, y_a, w_a, h_a : anchor box
x^*, y^*, w^*, h^* : ground-truth box

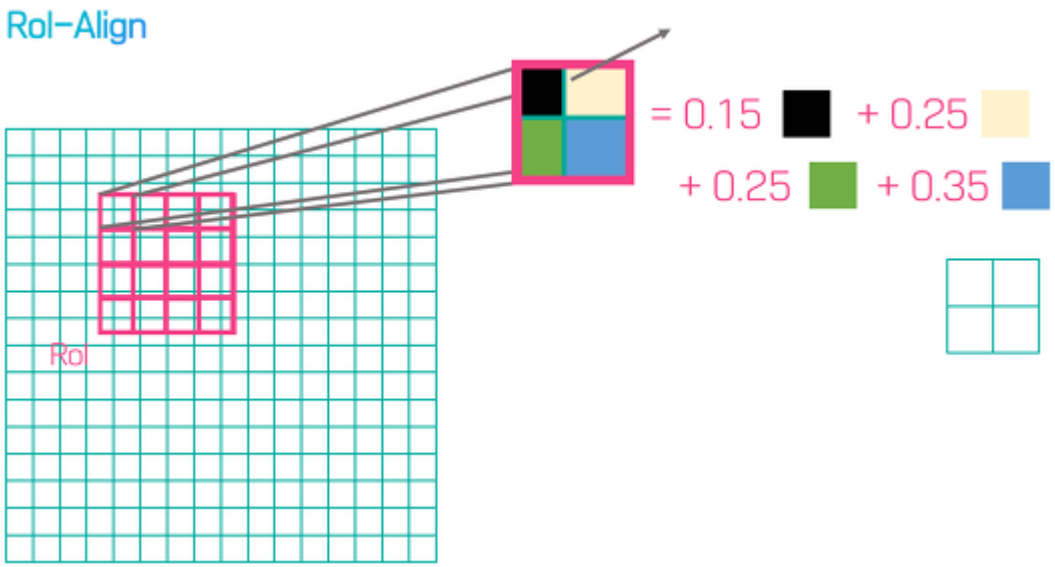
t값들, 즉 delta값을 output 으로 받게 된다. 따라서 이 delta값에 anchor정보를 연산해서 원래 이미지에 대응되는 anchor bounding box 좌표값으로 바꿔주게 된다.

NMS

원래 이미지에 anchor 좌표를 대응시킨 후에는 각각 normalized coordinate로 대응시킨다.
이는 FPN에서 이미 각기 다른 feature map크기를 갖고있기에 모두 통일되게 정규좌표계로 이동시키는 것이다. 이렇게 수천 개의 anchor box가 생성되면 NMS알고리즘을 통해 anchor의 개수를 줄인다. 각 object마다 대응되는 anchor가 수십개 존재하는데 이때 가장 classification score가 높은 anchor를 제외하고 주위에 다른 anchor들은 모두 지우는 것이다.
NMS알고리즘은 anchor bbox들을 score순으로 정렬시킨 후 score가 높은 bbox부터 다른 bbox와 IoU를 계산한다. 이때 **IoU가 해당 bbox와 0.7이 넘어가면 두 bbox는 동일 object를 detect한 것이라 간주**하여 score가 더 낮은 bbox는 지우는 식으로 동작한다. 최종적으로 각 객체마다 score가 가장 큰 box만 남게되고 나머지 box는 제거한다.

RoI align

bilinear interpolation 을 이용해서 위치정보를 담는 RoI align 을 이용한다.



[그림 13] RoI Align

3. 프로젝트 내용

3.1. 접근방식

본 연구의 내용은 Detectron2 를 활용하여 Mask R-CNN 을 학습시켜 Custom Object Detection, Custom Instance Segmentation 을 가능하게 하는 것이다.

프로젝트 과정은 Google Colab 에서 detectron2 를 활용할 수 있게 환경을 만들기 위해 pytorch 와 관련된 것들을 설치한 후 학습을 진행하고 결과물들을 확인한다.

3.2. 요구사항

Dataset 로 쓰여질 차량번호판이 보이는 이미지들을 확보한다.

이미지들을 확보한 후에는 Labelme 라는 도구를 이용하여 내가 인식하고 싶어하는 차량번호판이 어떤 형태와 특징을 가지는지에 대해 각 이미지에서 Polygon 을 형성하여 설정하여 준다.

각각의 이미지에 대하여 Labelme 에서 생성된 json 파일을 이용하여 학습을 진행하면 된다.

3.3. 구현

소스코드는 다음과 같다.

초기환경설정

```
[ ]  1 from google.colab import drive
     2 drive.mount('/content/drive')
```

```
[ ]  1 !pip install pyyaml==5.1
     2 !pip install torch==1.8.0+cu101 torchvision==0.9.0+cu101 -f https://download.pytorch.org/whl/torch_stable.html
     3 #install old version of pytorch since detectron2 hasn't released packages for pytorch 1.9
```

```
[ ]  1 !pip install detectron2 -f https://dl.fbaipublicfiles.com/detectron2/wheels/cu101/torch1.8/index.html
     2 # After this setp it will ask you to restart the runtime, please do it
```

```
[ ]  1 import torch
     2 import torchvision
     3 import cv2
```

[그림 14] 소스코드 1

utils.py

```python
from detectron2.data import DatasetCatalog, MetadataCatalog
from detectron2.utils.visualizer import Visualizer
from detectron2.config import get_cfg
from detectron2 import model_zoo

from detectron2.utils.visualizer import ColorMode

import random
import cv2
import matplotlib.pyplot as plt
%matplotlib inline

#####
from detectron2.engine import HookBase
from detectron2.data import build_detection_train_loader
import detectron2.utils.comm as comm
#####

def plot_samples(dataset_name, n=1):
    dataset_custom = DatasetCatalog.get(dataset_name)
    dataset_custom_metadata = MetadataCatalog.get(dataset_name)

    for s in random.sample(dataset_custom, n):
        img = cv2.imread(s["file_name"])
        v = Visualizer(img[:,:,::-1], metadata=dataset_custom_metadata, scale=1)
        v = v.draw_dataset_dict(s)
        plt.figure(figsize=(14,10))
        plt.imshow(v.get_image())
        plt.show()

def get_train_cfg(config_file_path, checkpoint_url, train_datasetname, test_dataset_name, num_classes, device, output_dir):
    cfg = get_cfg()

    cfg.merge_from_file(model_zoo.get_config_file(config_file_path))
    cfg.MODEL.WEIGHTS = model_zoo.get_checkpoint_url(checkpoint_url)
    cfg.DATASETS.TRAIN = (train_dataset_name,)
    cfg.DATASETS.TEST = (test_dataset_name,)  #  Validation

    #####
    cfg.DATASETS.VAL = (test_dataset_name,)
    #####

    """cfg.TEST.EVAL_PERIOD = 100 # 1000 iteration 중에 100번마다 validation 수행 """

    cfg.DATALOADER.NUM_WORKERS = 1      # class의 개수

    cfg.SOLVER.IMS_PER_BATCH = 2
    cfg.SOLVER.BASE_LR = 0.00025
    cfg.SOLVER.MAX_ITER = 15000        # 학습횟수 iteration
    cfg.SOLVER.STEPS = []

    cfg.MODEL.ROI_HEADS.NUM_CLASSES = num_classes
    cfg.MODEL.DEVICE = device
    cfg.OUTPUT_DIR = output_dir

    return cfg

#####


class ValidationLoss(HookBase):
    def __init__(self, cfg):
        super().__init__()
        self.cfg = cfg.clone()
        self.cfg.DATASETS.TRAIN = cfg.DATASETS.VAL
        self._loader = iter(build_detection_train_loader(self.cfg))

    def after_step(self):
        data = next(self._loader)
        with torch.no_grad():
            loss_dict = self.trainer.model(data)

            losses = sum(loss_dict.values())
            assert torch.isfinite(losses).all(), loss_dict

            loss_dict_reduced = {"val_" + k: v.item() for k, v in
                                 comm.reduce_dict(loss_dict).items()}
            losses_reduced = sum(loss for loss in loss_dict_reduced.values())
            if comm.is_main_process():
                self.trainer.storage.put_scalars(total_val_loss=losses_reduced,
                                                 **loss_dict_reduced)
#####

def on_image(image_path, predictor) :
    im = cv2.imread(image_path)
    outputs = predictor(im)
    v = Visualizer(im[:,:,::-1], metadata={}, scale=1, instance_mode=ColorMode.SEGMENTATION)
    v = v.draw_instance_predictions(outputs["instances"].to("cpu"))

    plt.figure(figsize=(14,10))
    plt.imshow(v.get_image())
    plt.show()

def on_video(videoPath, predictor):
    cap = cv2.VideoCapture(videoPath)
    if (cap.isOpened()==False):
        print("Error opening file...")
        return

    (success,image) = cap.read()
    while success:
        predictions = predictor(image)
        v = Visualizer(image[:,:,::-1], metadata={}, scale=1, instance_mode=ColorMode.SEGMENTATION)
        output = v.draw_instance_predictions(predictions["instances"].to("cpu"))

        cv2.imshow("Result", output.get_image()[:,:,::-1])

        key = cv2.waitKey(1) & 0xFF
        if key == ord("q"):
            break
        (success,image) = cap.read()

```

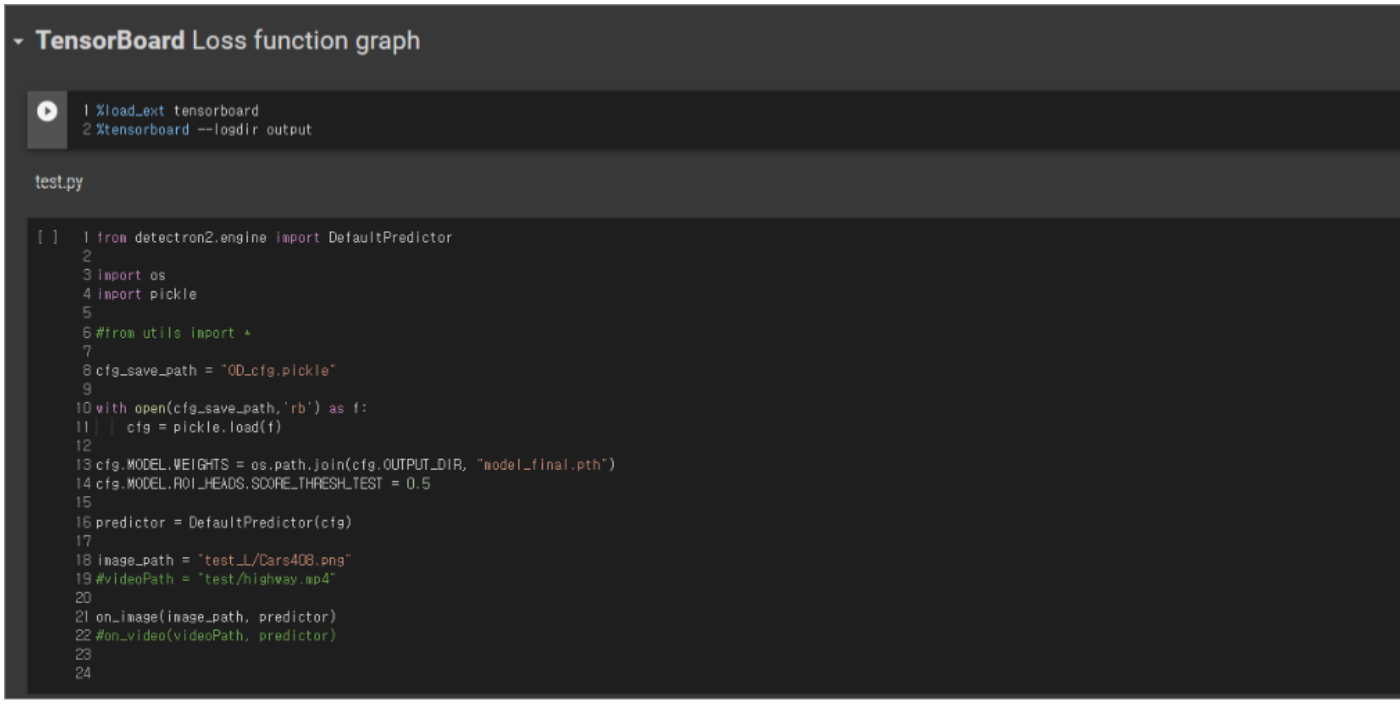
[그림 15] 소스코드 2

train.py

```python
from detectron2.utils.logger import setup_logger

setup_logger()

from detectron2.data.datasets import register_coco_instances
from detectron2.engine import DefaultTrainer


### 모델평가 관련
from detectron2.evaluation import COCOEvaluator, inference_on_dataset
from detectron2.data import build_detection_test_loader


import os
import pickle

#from utils import *

config_file_path = "COCO-Detection/faster_rcnn_R_50_FPN_3x.yaml" #COCO-InstanceSegmentation/mask_rcnn_R_50_FPN_3x.yaml
checkpoint_url = "COCO-Detection/faster_rcnn_R_50_FPN_3x.yaml"

output_dir = "./output/object_detection"
num_classes = 1

device = "cuda" # "cuda" or "cpu"

train_dataset_name = "LP_train"
train_images_path = "train"
train_json_annot_path = "train.json"

test_dataset_name = "LP_test"
test_images_path = "test"
test_json_annot_path = "test.json"

cfg_save_path = "OD_cfg.pickle"

################################################

if train_dataset_name in DatasetCatalog.list():
    DatasetCatalog.remove(train_dataset_name)

if test_dataset_name in DatasetCatalog.list():
    DatasetCatalog.remove(test_dataset_name)

################################################

register_coco_instances(name = train_dataset_name, metadata={},
json_file= train_json_annot_path, image_root=train_images_path)
```

```python
register_coco_instances(name = test_dataset_name, metadata={},
json_file= test_json_annot_path, image_root=test_images_path)

plot_samples(dataset_name=train_dataset_name, n=2)

################################################

def main():
    cfg = get_train_cfg(config_file_path, checkpoint_url, train_dataset_name, test_dataset_name, num_classes, device, output_dir)

    with open(cfg_save_path, 'wb') as f :
        pickle.dump(cfg,f,protocol=pickle.HIGHEST_PROTOCOL)

    os.makedirs(cfg.OUTPUT_DIR,exist_ok=True)
    trainer = DefaultTrainer(cfg)
    #####
    val_loss = ValidationLoss(cfg)
    trainer.register_hooks([val_loss])
    # swap the order of PeriodicWriter and ValidationLoss
    trainer._hooks = trainer._hooks[:-2] + trainer._hooks[-2:][::-1]
    #####

    trainer.resume_or_load(resume=False)

    trainer.train()  # 학습 시작

    ### 모델 평가(mAP)
    evaluator = COCOEvaluator("LP_test", cfg, False, output_dir="./output/")
    val_loader = build_detection_test_loader(cfg, "LP_test")
    print(inference_on_dataset(trainer.model, val_loader, evaluator))
    ###

if __name__ == '__main__' :
    main()

################################################
```

[그림 16] 소스코드 3



[그림 17] 소스코드 4

관련 코드는 https://github.com/sswoo333/2022-1-capstone2-ssw 에서 자세히 볼 수 있다.

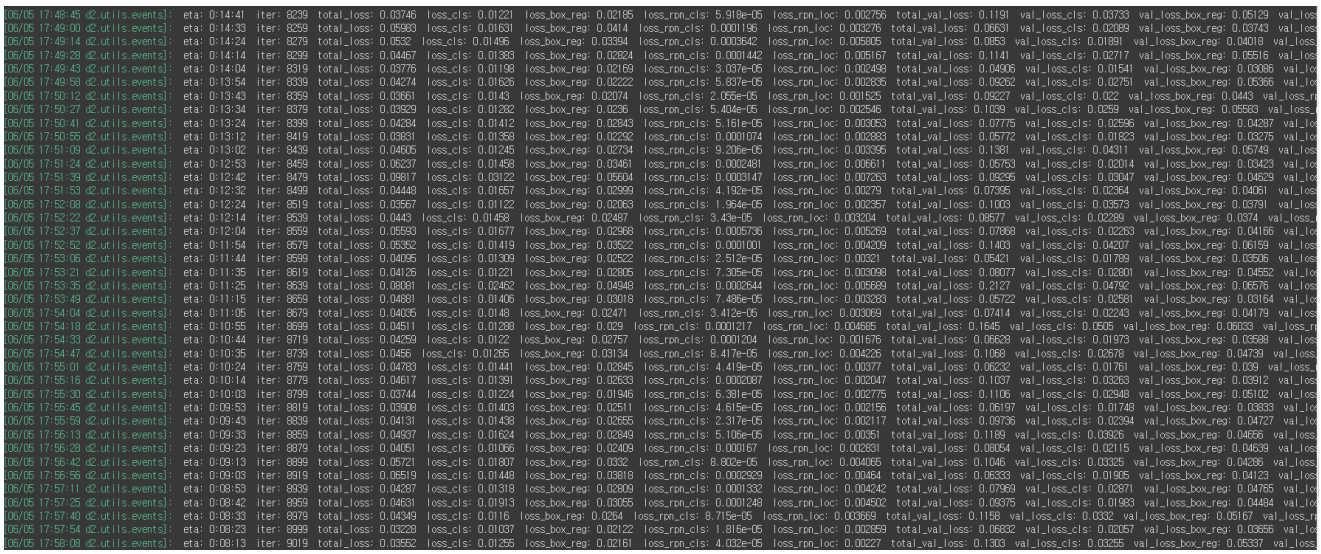
4. 프로젝트 결과

먼저 학습을 시키기 이전에 차량번호판을 잘 나타내고 있는지 확인하는 결과는 아래와 같다.

[그림 18] License plate 예시 1

[그림 19] License plate 예시 2

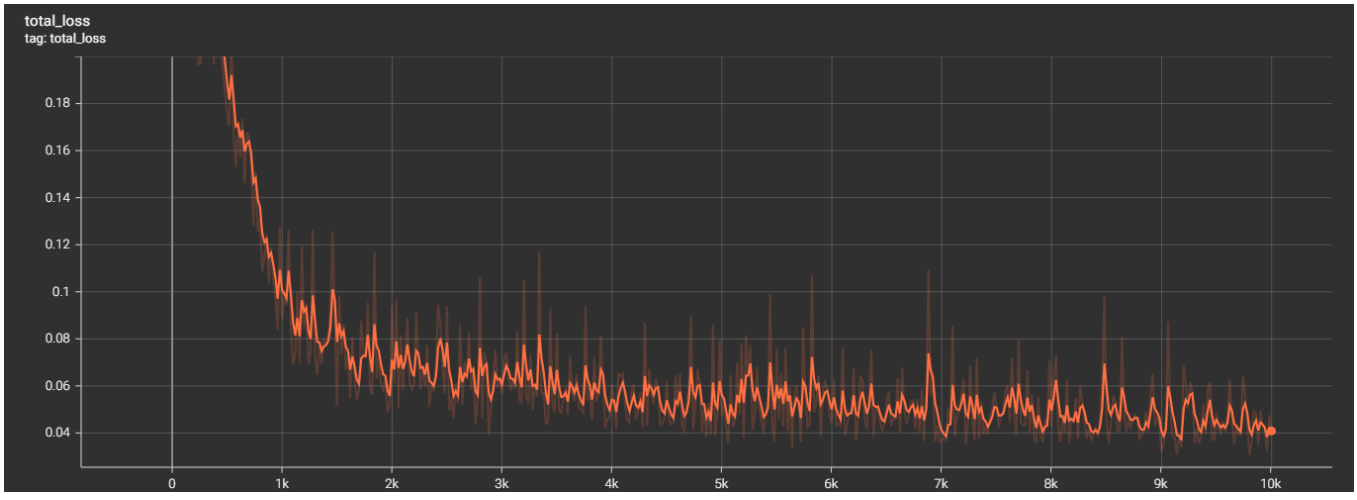
학습을 하는 과정은 다음과 같다.



[그림 20] 학습과정

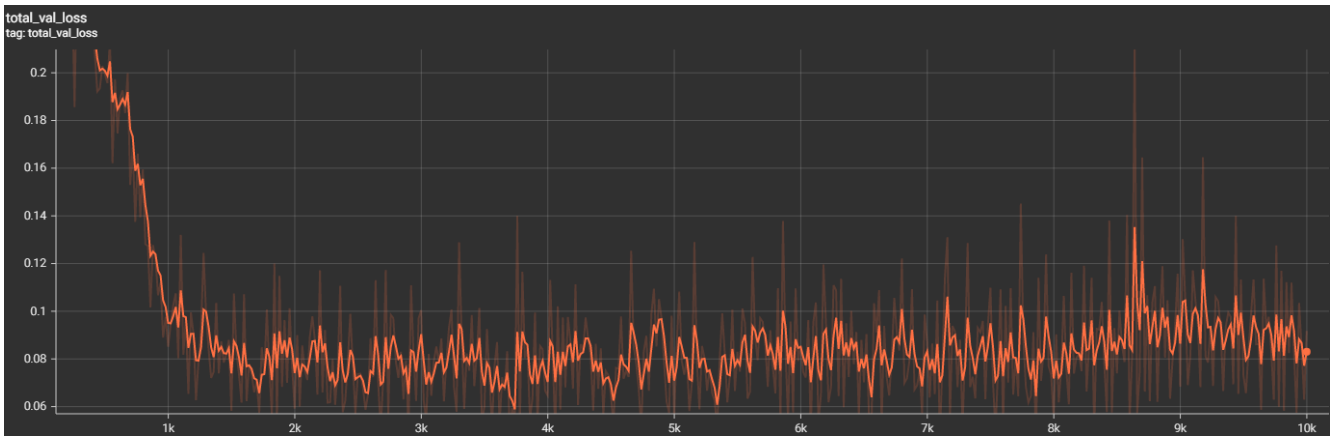
학습을 마치고 나면 우리는 Train loss function graph & Validation loss function graph & mAP 를 얻을 수 있다.

(Train loss function)



[그림 21] 10k train loss function graph

(Validation loss function)



[그림 22] 10k validation loss function graph 1

이 두 그래프를 살펴보면 train loss function 은 계속해서 값이 떨어지는 것을 볼 수 있는데 이게 훈련이 잘 되고 있는지 검증할 필요가 있다.

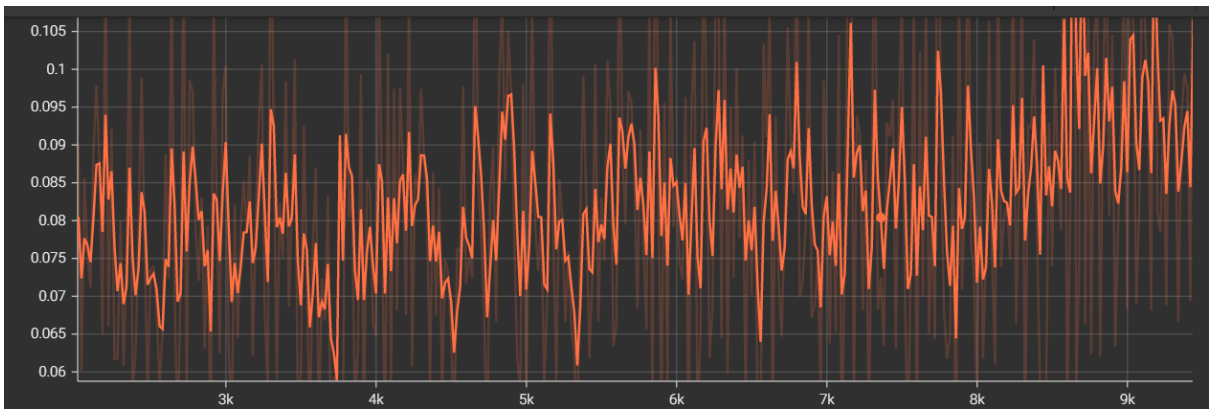
이것을 알아보기 위해서 validation set 으로 loss function graph 를 그려보면 되는데 두 그래프 같이 값이 떨어지고 있을 때는 훈련이 잘 되고 있는 것이다.

하지만 train loss function graph 는 값이 떨어지는데 validation loss function graph 가 값이 떨어지지 않고 우상향하게 되면 이것은 overfitting 이 발생하는 것이며 overfitting 이 발생하기 전에 훈련을 멈추어야만 훈련이 잘 되었다고 할 수 있다.

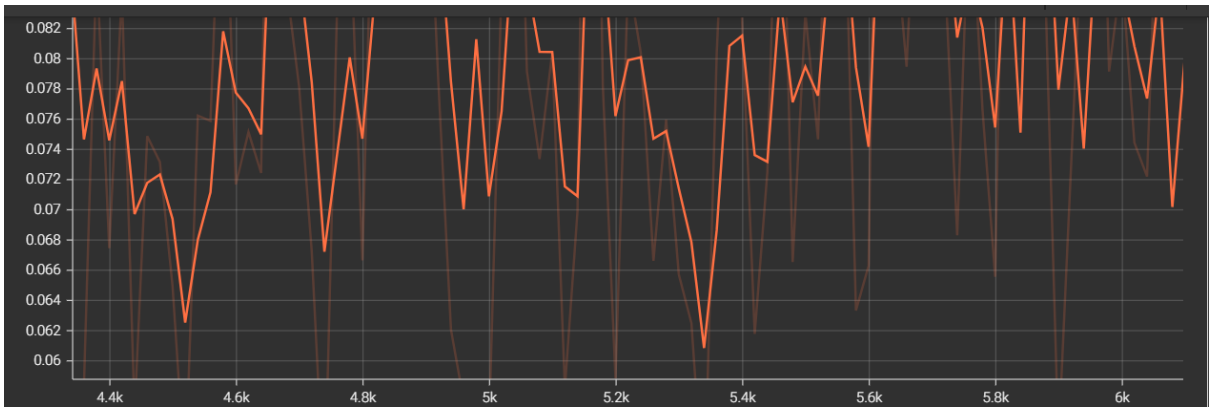
Overfitting 이 발생하는 이유는 train set 에 있는 데이터들을 계속해서 학습하다보니 그 데이터에 대한 정확도는 계속해서 높아져가지만 train set 이외에 다른 데이터를 접했을 때는 원하는 객체를 잘 인식하지 못하게 될 때이다.

본 연구의 목적은 train set 이외에 다른 데이터(이미지)를 접하더라도 정상적으로 차량번호판을 검출해야만 하므로 overfitting 이전에 훈련을 멈추어야 한다.

다시 Validation loss function graph 를 보면



[그림 23] 10k validation loss function graph 2



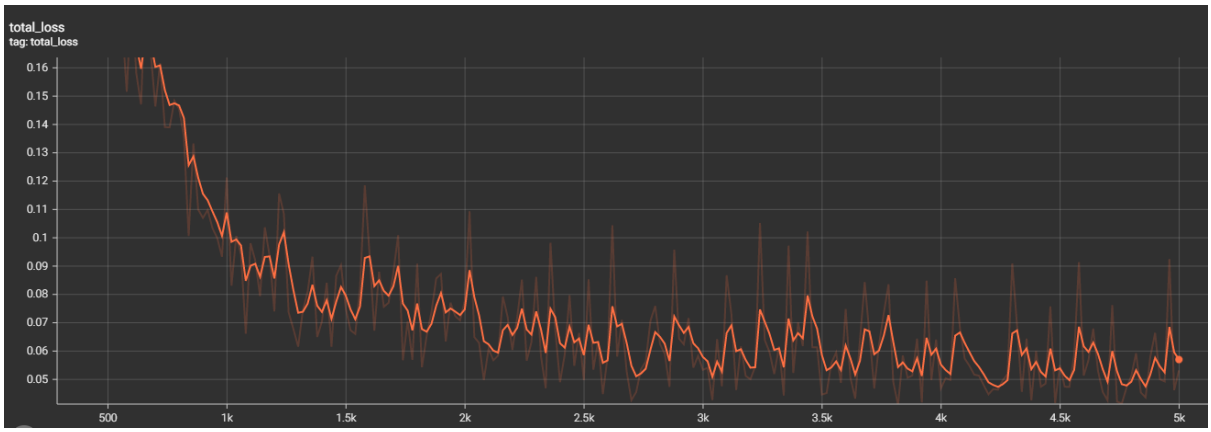
[그림 24] 10k validation loss function graph 3

iteration(X 축)이 5.34k 값을 가지는 지점 이후에서부터 graph 가 조금씩 우상향한다는 것을 알 수 있다.

Overfitting 이 발생한 것이다.

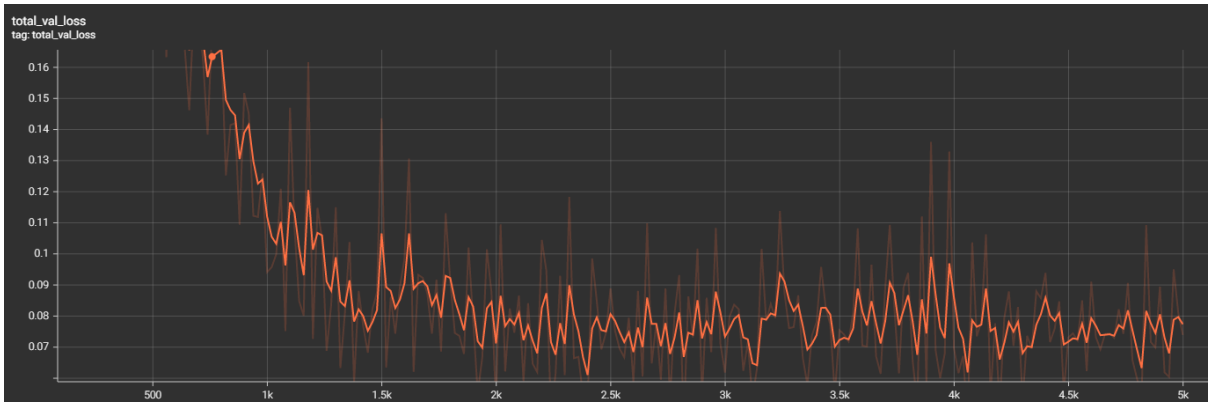
이 경우 validation loss function graph 가 우상향하기 직전까지 학습을 시켰을 때 좋은 객체 인식 모델을 얻을 수 있으므로 따라서 iteration 을 5k 로 설정한 후 학습을 다시 시켜보도록 한다.

(Train loss function)



[그림 25] 5k train loss function graph

(Validation loss function)



[그림 26] 5k train loss function graph

위의 그래프를 보면 5k 까지 train loss function & validation loss function 둘 다 값이 우하향되는 것을 볼 수 있다.

그리고 이 때의 mAP 의 값과 테스트 해본 결과는 아래와 같다.

AP	AP50	AP75	APs	APm	APl
59.236	82.075	69.847	39.613	83.003	82.129

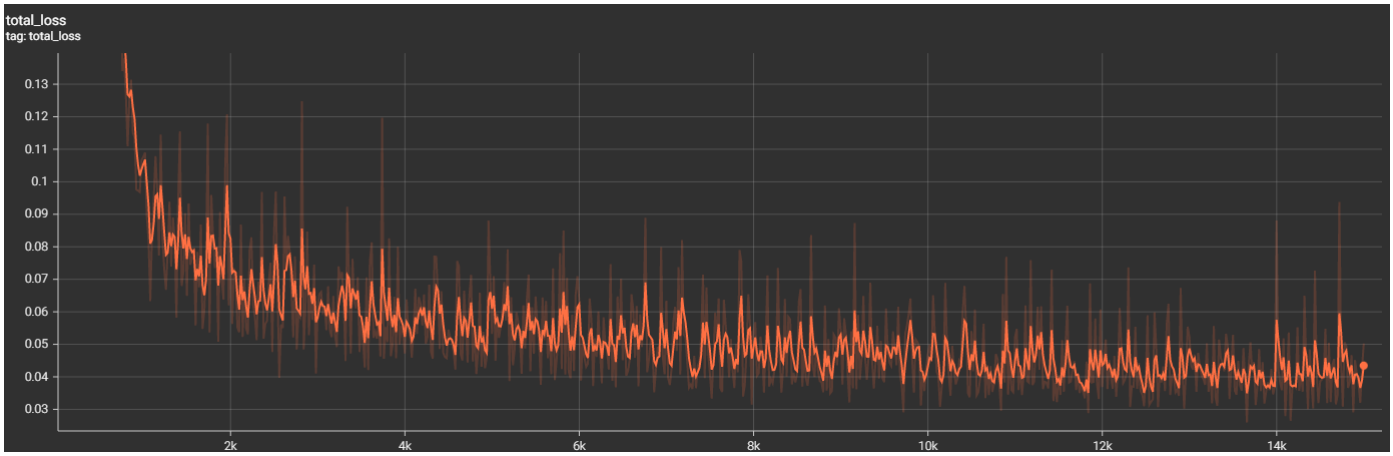
[그림 26] 5k mAP

[그림 27] 5k test picture

위의 결과 사진으로 보아 Detectron2를 활용하여 Mask R-CNN을 학습시켜 자동차 번호판을 잡는 것이 가능하게 되었다는 사실을 알 수 있다.

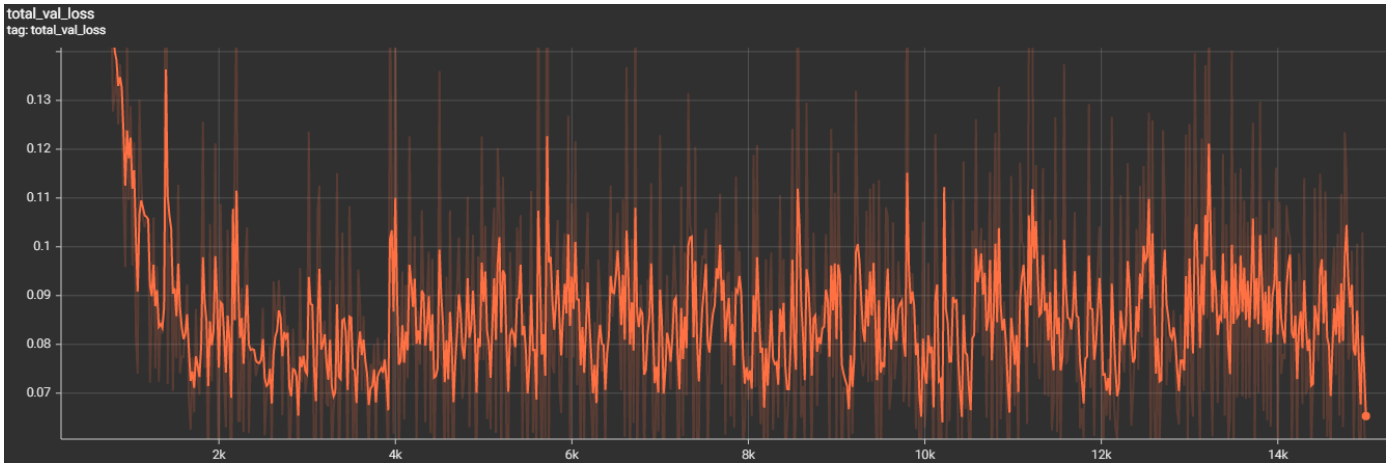
추가로 Iteration 을 15k 로 설정하고 학습한 결과의 loss function graph 는 아래와 같다.

(15k train loss function graph)



[그림 28] 15k train loss function graph

(15k validation loss function graph)



[그림 28] 15k validation loss function graph

5. 결론

5.1 기대효과

본 연구의 목표였던 Detectron2 를 활용하여 Mask R-CNN 을 잘 학습시켜 이미지나 영상에서 자동차 번호판을 인식하게 할 수 있었다.

이번 연구의 결과를 바탕으로 원하는 다른 객체를 검출하는 것도 이와 유사한 방법으로 진행한다면 가능할 것이다.

5.2 추후 연구 방향

이 프로젝트를 시작할 당시에 본 논문의 연구 방향은 이미지나 영상에서 산불이 일어나는 것을 검출하거나 알파벳, 숫자 등과 같은 문자를 검출하는 데에 두었는데, 진행하는 과정에서 산불을 검출하는 경우에는 산불이 일어나는 그 현상은 그 형태가 일정하지 않은 비정형적인 모습이므로 검출하는 것에 어려움이 있었다. 그리고 문자 인식 같은 경우에는 문자는 숫자 10 개 알파벳 26 개를 포함하여 검출해야 하므로 그 객체의 개수가 많은 복잡함이 뒤따라 어려움이 발생했다.
따라서 검출해야 할 객체의 모양이 일정하면서도 종류가 다양하지 않은 자동차 번호판을 객체로 선택하여 프로젝트를 진행하게 되었다.

앞으로의 연구 방향은 산불과 같은 비정형적인 형태의 객체를 인식해도 보고 문자와 같은 객체가 다양한 경우에도 Detectron2 를 이용해 Mask R-CNN 을 학습시켜 도출할 수 있게 하는 것이다.

그리고 자동차 번호판을 인식하는 것에 그치지 않고 자동차 번호판에 있는 문자들을 인식하여 자동차에 있는 고유한 문자와 숫자의 나열을 인식한 후, 그를 처리 가공하여 활용할 수 있게 데이터를 추출해보는 것도 예상해볼 수 있다.

6. 참고문헌

[1] "Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation", Ross Girshick, Jeff Donahue, Trevor Darrell, Jitendra Malik; Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2014, pp. 580-587

[2] "Fast R-CNN", Ross Girshick; 2015 IEEE International Conference on Computer Vision (ICCV), 2015, pp.1440-1448

[3] "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks", Shaoqing Ren, Kaiming He, Ross Girshick, Jian Sun; in *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 39, no. 6, pp. 1137-1149; Part of Advances in Neural Information Processing Systems 28 (NIPS 2015)

[4] "Mask R-CNN", Kaiming He, Georgia Gkioxari, Piotr Dollar, Ross Girshick; Proceedings of the IEEE International Conference on Computer Vision (ICCV), 2017, pp. 2961-2969