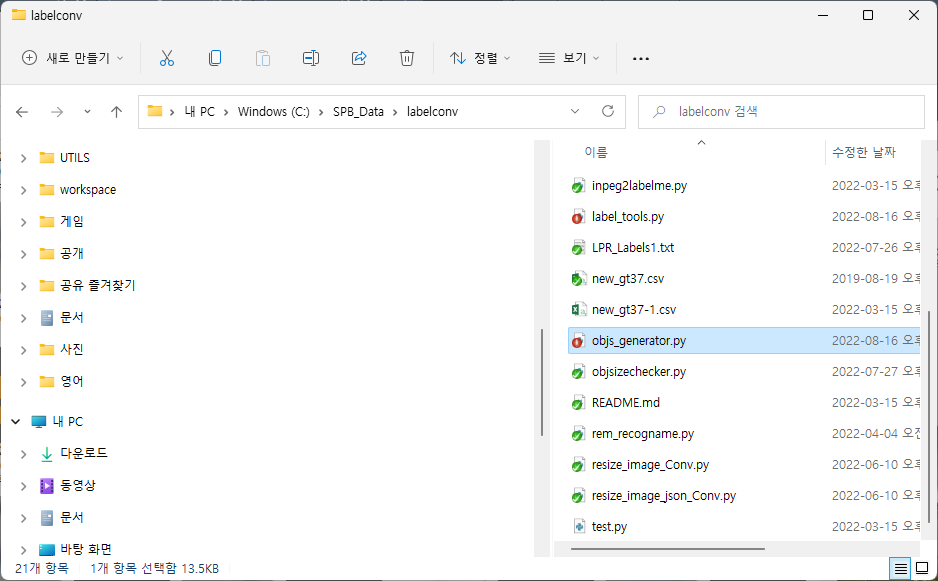
번호인식 알고리즘 개발

작성자 : 윤경섭

작성일 : 2022년 8월 22일



# 용도문자 생성

C:\SPB\_Data\labelconv 로 이동한다.

objs\_generator.py (object generator python) 파일을 사용하여 각 문자, 지역문자를 잘라 낸다.

예) python objs\_generator -l ./LPR\_Labels1.txt -i ./image -j ./annots -o ./result -r 225,225 -f False -t ch

-l label 파일

-i 이미지 위치

-j json 위치

-o 결과 파일 저장 위치

-r resize 만일 이것이 없으면 resize를 하지 않는다.

-f fixratio 영상 가로세로 비율을 고정할지 안할지 여부

-t type ch: 문자 n: number r:지역문자 vr:세로 지역문자 hr:가로지역문자 or: 영지역문자 r6: 6지역문자

C:\SPB\_Data\labelconv\dataset\labels\out 에 image 파일과 json 파일이 생성된다.

이 생성 위치는 지역문자에도 동일하게 적용된다.

원래는 위와 같이 파라미터를 정해 주어야 하나

#------------------------------

# 수정할 내용

OUTPUT\_FOLDER\_NAME = 'out' # labelme로 출력할 디렉토리 이름 (현재 디렉토리 아래로 저장된다.)

IMAGE\_FOLDER\_NAME = 'images' #이미지 파일에 있는 영상 파일이 있는 경로

JSON\_FOLDER\_NAME = 'annots' #json 폴더가 있는 경로

MIDDLE\_PATH =  os.path.join('dataset','labels')

DEFAULT\_LABEL\_FILE = "./LPR\_Labels1.txt"  #라벨 파일이름

CROP\_MARGIN = 0

DEFULT\_FIXED\_RATIO=False         #영상의 크기를 싸이즈에 맞게 늘일지 여부 True : 고정 False : 늘림

DEFAULT\_TYPE='ch' #읽어올 default 타입을 설정한다. ch: 문자  n: number r:지역문자 vr:세로 지역문자 hr:가로지역문자 or: 영지역문자 r6: 6지역문자

#------------------------------

objs\_generator.py 의 상단부 기본적으로 수정할 내용을 두어 필요에 따라 수정 하게 하였다.

# 지역문자 생성

지역문자는 가로지역문자, 세로 지역문자, 특장지역문자, 특장 비슷한 지역문자가 있다.

각각을 hr, vr, or, r6 옵션을 통해서 생성한다.

아래는 hr 옵션을 선택한 예이다.

python objs\_generator -l ./LPR\_Labels1.txt -i ./image -j ./annots -o ./result -r 225,225 -f False -t hr

# 문자 샘플 training, validation 데이터 나누기

1,2에서 생성한 문자를 training validation 비율로 나누려면

C:\SPB\_Data\chardet 로 이동한다. image\_distributer.py가 해당 파일이다.

파일을 열어 상단부분에

MIDDLE\_PATH\_NAME = 'datasets'

OUTPUT\_FOLDER\_NAME = 'out' # labelme로 출력할 디렉토리 이름 (현재 디렉토리 아래로 저장된다.)

DEFAULT\_OBJ\_TYPE = 'vr'

DEFAULT\_LABEL\_FILE = "./LPR\_Labels1.txt"  #라벨 파일이름

option\_move = False # 원 파일을 옮길지 여부

DEFAULT\_OBJ\_TYPE을

ch: 문자 n: number r:지역문자 vr:세로 지역문자 hr:가로지역문자 or: 영지역문자 r6: 6지역문자

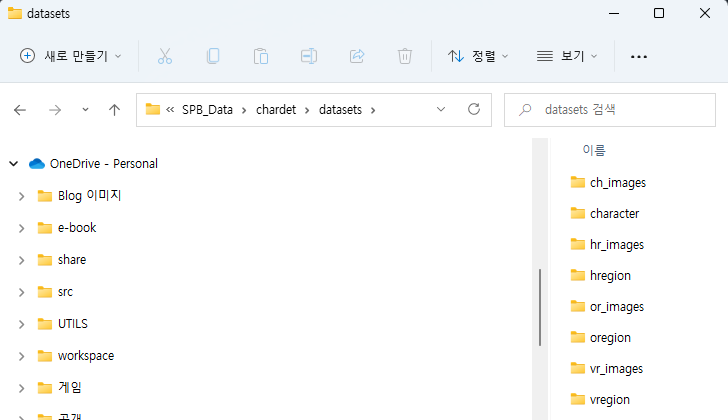
로 구분하여 입력하면 각 영상이 분리 된다.

이때 영상 및 json 파일은

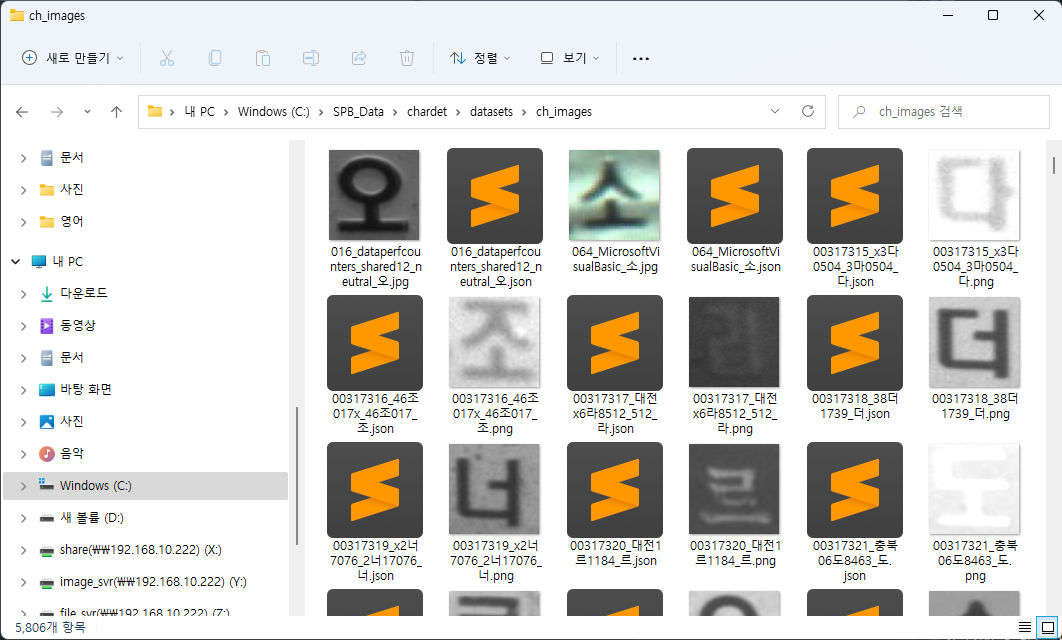
Type 별 영상 디렉토리

|  |  |  |
| --- | --- | --- |
| 타입 | 영상 디렉토리 | 출력 디렉토리 |
| ch | ch\_images | character |
| hr | hr\_images | hregion |
| vr | vr\_images | vregion |
| or | or\_images | oregion |
| r6 | r6\_images | region6 |
| r | r\_images | region |
| n | n\_images | number |

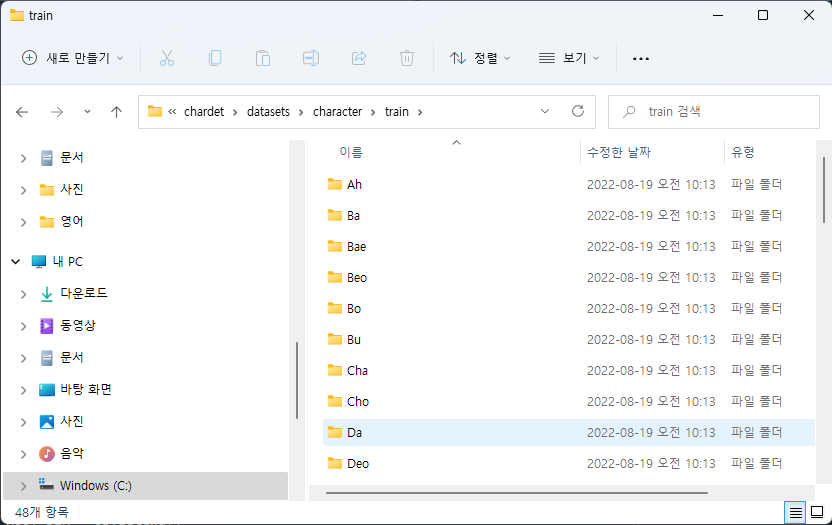
로 매칭 된다.

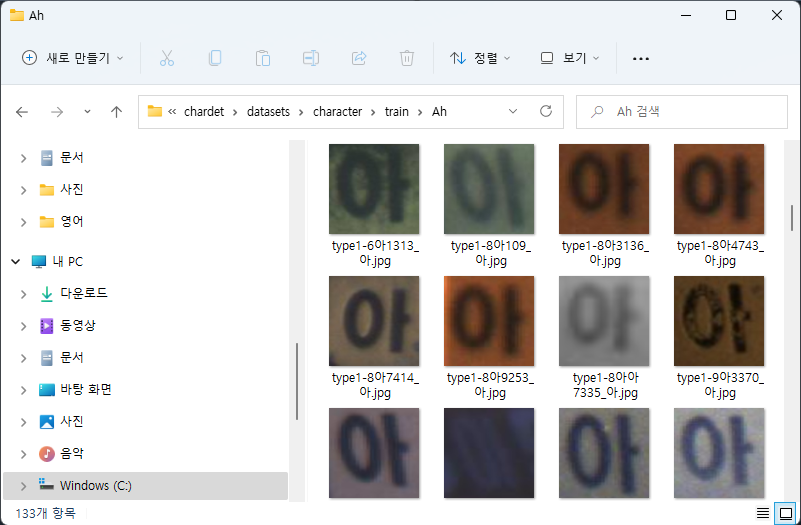


ch\_images 디렉토리를 보면 글자 영상과 거기에 해당하는 json 파일이 있다. 이것은 다른 디렉토리도 동일하다.



Character 이미지는 각 문자별로 구별 되어 저장된다.





인식을 하기 위해서는 train 데이터와 validation 데이터가 필요하므로 동일한 디렉토리 구조를 갖는 validation 디렉토리를 생성한다. 이 코드는 랜덤하게 파일을 섞어서 구성하므로 매번 train과 validation의 파일 구성이 달라질 수 있다. 또한 train에 있는게 validation에 없고 train에 없는게 validation에 있을 수 있다. 동일한 디렉토리 구조를 가져야 하므로 배치 후 다시 한번 검사하여 동일한 디렉토리를 갖도록 후처리를 한다.

Train 데이터와 validation 데이터의 구성비는 보통 7:3 정도인데, 이것은 사용자가 변경 할수 있다.

# training / validateion  비율을 설정한다.

parser.add\_argument("-r",

                    "--ratio", type=float,

                    help="train validation ratio ex[0.7,0.3] ",default=[0.7,0.3], required=False)

# 문자 훈련 (training)

각 문자 및 지역문자들은 chardet 프로젝트의 train.py 함수를 사용하여 훈련한다.

이 파일의 상단에

#---------------------------------------------

#이미지 크기 조정 크기

IMG\_SIZE = 224

#배치 싸이즈

BATCH\_SIZE = 32

#epochs

EPOCHS =  200

backbone = 'resnet50'

DEFAULT\_LABEL\_FILE = "./LPR\_Labels1.txt"  #라벨 파일이름

DEFAULT\_OBJ\_TYPE = 'ch'#'ch'

#---------------------------------------------

부분에서 DEFAULT\_OBJ\_TYPE를 변경하며 훈련한다.

여기서 train 디렉토리의 각 폴더가 하나의 클래스가 되므로 만일 디렉토리가 없거나 비어 있으면 해당 클래스는 인식 범주에서 벗어난다.

# 영상증폭 (augmentation)

다음과 같이 ImageDataGenerator를 사용하여 이미지를 증폭한다.

train\_datagen = ImageDataGenerator(

                            rotation\_range=30,

                             width\_shift\_range=0.2,

                             height\_shift\_range=0.2,

                             shear\_range=0.2,

                             zoom\_range=[0.7,1.0],

                             brightness\_range=[0.2,1.0])

각 입력 값을 통하여 얼마나 이미지의 변형을 줄지를 결정한다.

단 validation 이미지는이미지 증폭을 하지 않는다.

아래 함수를 통하여 이미지를 읽어 온다.

train\_generator = train\_datagen.flow\_from\_directory(train\_dir,

                                                    target\_size=(IMG\_SIZE,IMG\_SIZE),

                                                    batch\_size=BATCH\_SIZE,

                                                    shuffle=True,

                                                    seed=42,

                                                    class\_mode='categorical',

                                                    classes = categories)

모델 컴파일을 하고

model.compile( loss='categorical\_crossentropy', optimizer='adam',metrics=["acc"])

중간에 종료 할수 있게 earlystopping을 설정한다.

earlystopping = tf.keras.callbacks.EarlyStopping(monitor='val\_acc', patience=10)

첵크 포인트를 두어 중간 중간 훈련된 weight 를 저장한다.

checkpoint\_callback = ModelCheckpoint(filepath= model\_sub\_path\_str + "epoch\_{epoch:02d}\_val\_acc\_{val\_acc:.3f}.h5", monitor="val\_acc", save\_freq='epoch',save\_best\_only=True, verbose=1, mode='auto' ,save\_weights\_only=True)

모델을 훈련 시킨다.

history = model.fit(train\_generator,

                              steps\_per\_epoch=steps\_per\_epoch,

                              epochs=EPOCHS,

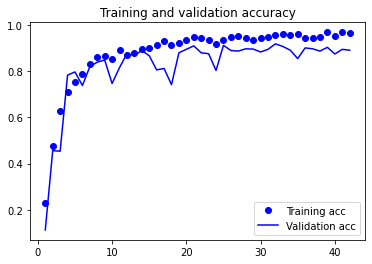
                              validation\_data=validation\_generator,

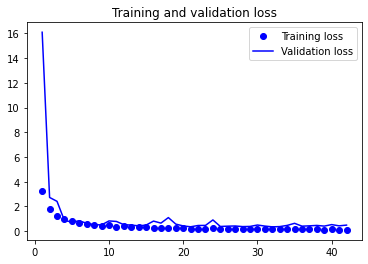
                              validation\_steps=validation\_steps,

                              #class\_weight=class\_weights,

                              callbacks=[checkpoint\_callback, tensorboard\_callback,earlystopping])

아래는 용도 문자를 훈련 시킨 예이다.

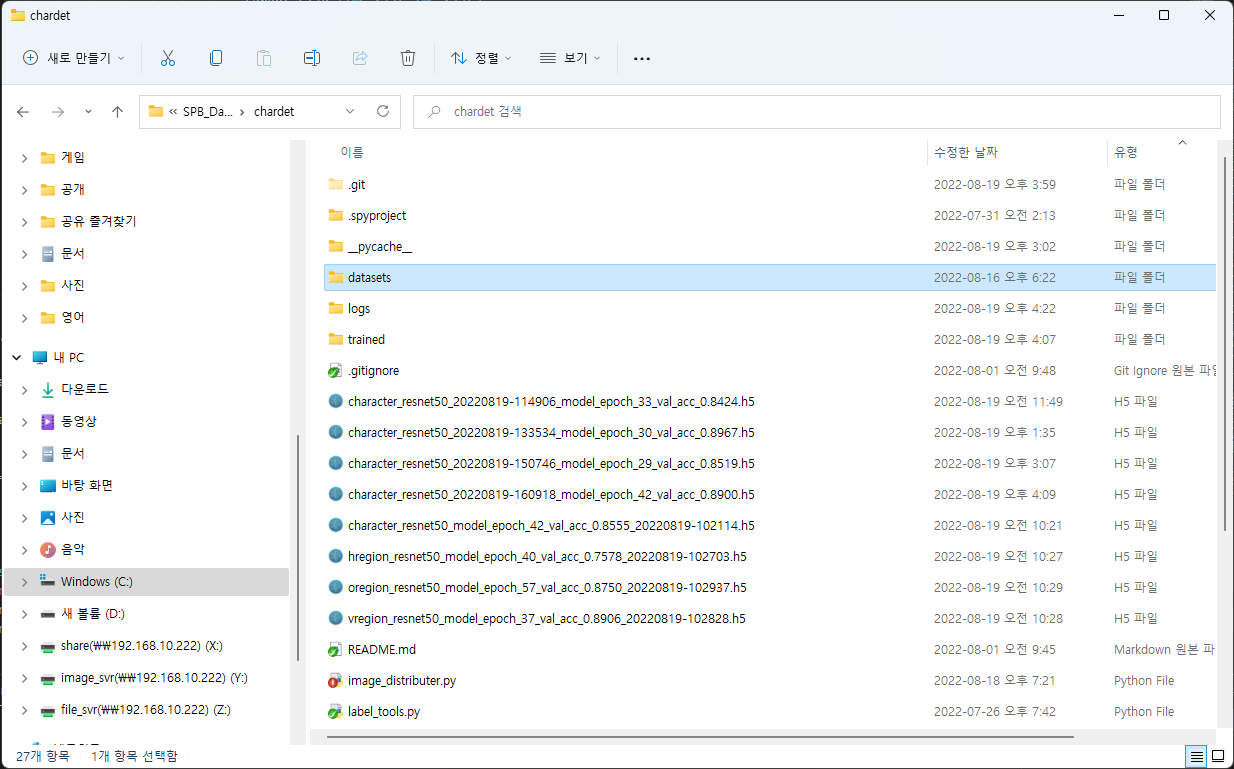




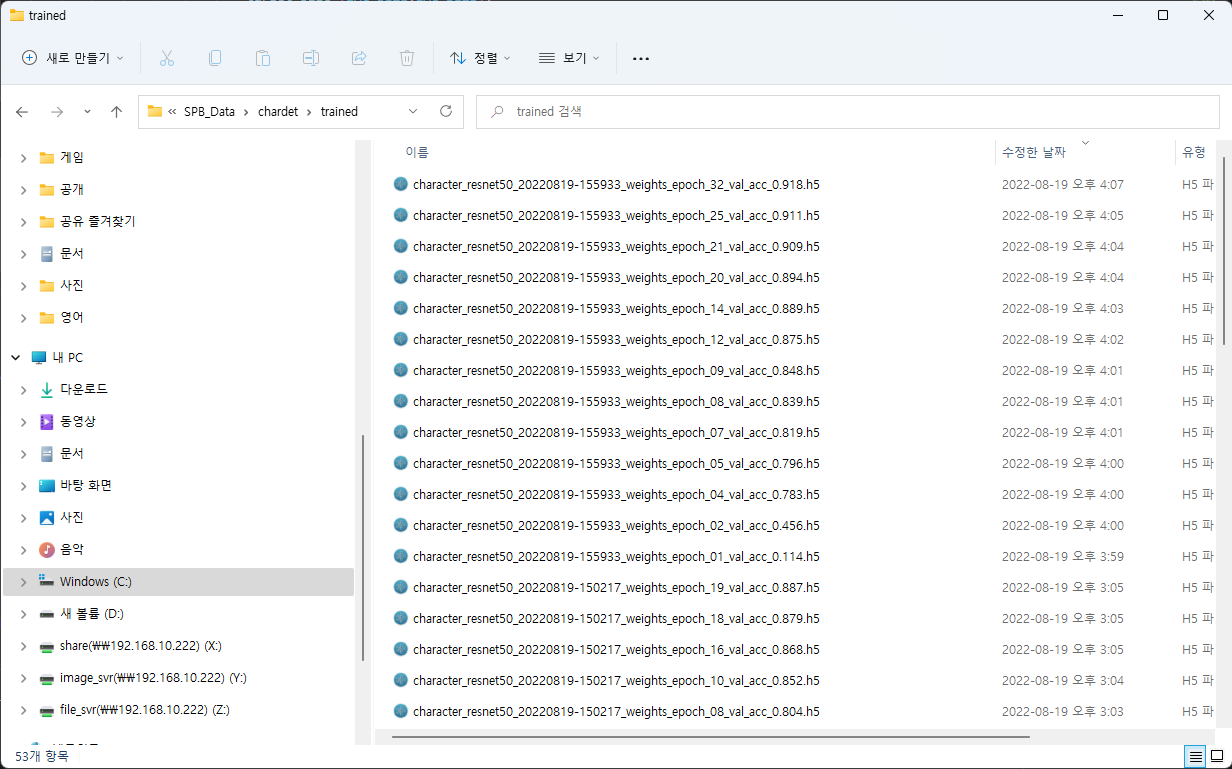
Epoch 32: val\_acc improved from 0.91125 to 0.91750, saving model to trained\character\_resnet50\_20220819-155933\_weights\_epoch\_32\_val\_acc\_0.918.h5

58/58 [==============================] - 12s 203ms/step - loss: 0.1338 - acc: 0.9540 - val\_loss: 0.3443 - val\_acc: 0.9175

훈련이 끝나면 훈련한 날짜별로 모델이 저장된다.



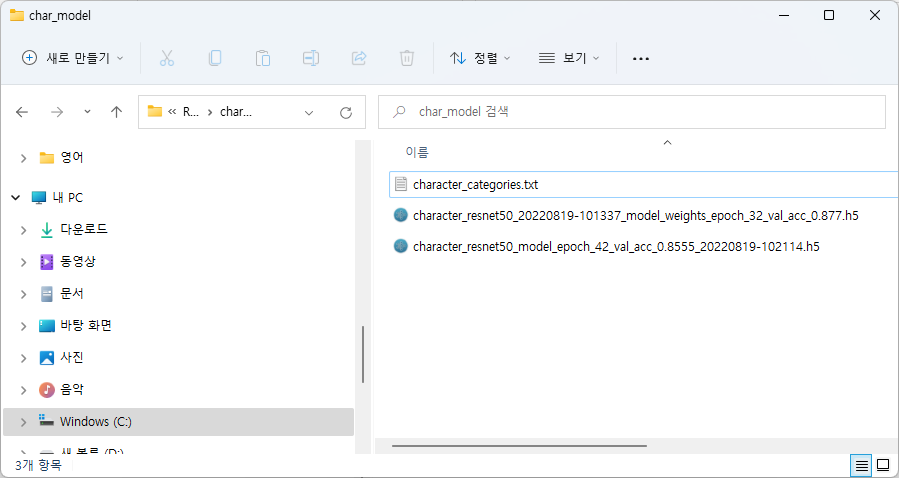
Weight 파일도 trained 디렉토리 아래에 저장 된다.



# Put all together

이제 각 문자별로 인식된 것을 가지고 전체 인식 과정을 수행한다.

RealTimeObjectDetection-main 디렉토리 아래 각각 char\_model, hreg\_model, vreg\_model, oreg\_model 디렉토리에 각각 인식된 모델 및 weight file, category 파일을 모델별 폴더로 복사한다.



각 모델별로 inference를 하는 파일이 있는데, 예를 들어 용도 문자를 인식하는 plate\_char\_infer.py의 경우 최신 모델과 weight file, category 파일을 업데이트 한다.[[1]](#footnote-1)

#----------------------------

DEFAULT\_LABEL\_FILE = "./LPR\_Labels1.txt"  #라벨 파일이름

CHAR\_MODEL\_DIR = 'char\_model'

MODEL\_FILE\_NAME ='character\_resnet50\_model\_epoch\_42\_val\_acc\_0.8555\_20220819-102114.h5'

CMODEL\_PATH = os.path.join(ROOT\_DIR,CHAR\_MODEL\_DIR,MODEL\_FILE\_NAME)

WEIGHT\_FILE\_NAME = 'character\_resnet50\_20220819-101337\_model\_weights\_epoch\_32\_val\_acc\_0.877.h5'

CWEIGHT\_PATH = os.path.join(ROOT\_DIR,CHAR\_MODEL\_DIR,WEIGHT\_FILE\_NAME)

CATEGORIES\_FILE\_NAME = 'character\_categories.txt'

CATEGORIES\_FILE\_PATH = os.path.join(ROOT\_DIR,CHAR\_MODEL\_DIR,CATEGORIES\_FILE\_NAME)

categories = []

CH\_THRESH\_HOLD = 0.8

#----------------------------

**인식의 시작은 plate\_recognition.py 코드를 실행하여 시작한다.** 이는 object detection api 함수를 이용하는 ssd 코드로 번호판 검지 및 숫자 인식은 ssd로 하고 용도 문자 및 지역 문자의 인식은 각 인식 모듈 별로 따로 한다.



아래 코드가 실제 구현 함수이다.

 ch = None

    twoLinePlate = False

    category\_index\_temp = copy.deepcopy(category\_index)

    for index, cindex in enumerate(detections['detection\_classes']+label\_id\_offset) :

        if category\_index[cindex]['name'] == 'Char' :

            det\_image\_np = extract\_sub\_image(image\_np,detections['detection\_boxes'][index],IMG\_SIZE,IMG\_SIZE,fixratio=False)

            ch = char\_det\_fn(cdet\_model,det\_image\_np)

            category\_index\_temp[cindex]['name'] = ch

        if category\_index[cindex]['name'] == 'hReg' :

            det\_image\_np = extract\_sub\_image(image\_np,detections['detection\_boxes'][index],IMG\_SIZE,IMG\_SIZE,fixratio=False)

            ch = hr\_det\_fn(hr\_det\_model,det\_image\_np)

            category\_index\_temp[cindex]['name'] = ch

            twoLinePlate = True

        if category\_index[cindex]['name'] == 'vReg' :

            det\_image\_np = extract\_sub\_image(image\_np,detections['detection\_boxes'][index],IMG\_SIZE,IMG\_SIZE,fixratio=False)

            ch = vr\_det\_fn(vr\_det\_model,det\_image\_np)

            category\_index\_temp[cindex]['name'] = ch

        if category\_index[cindex]['name'] == 'oReg' :

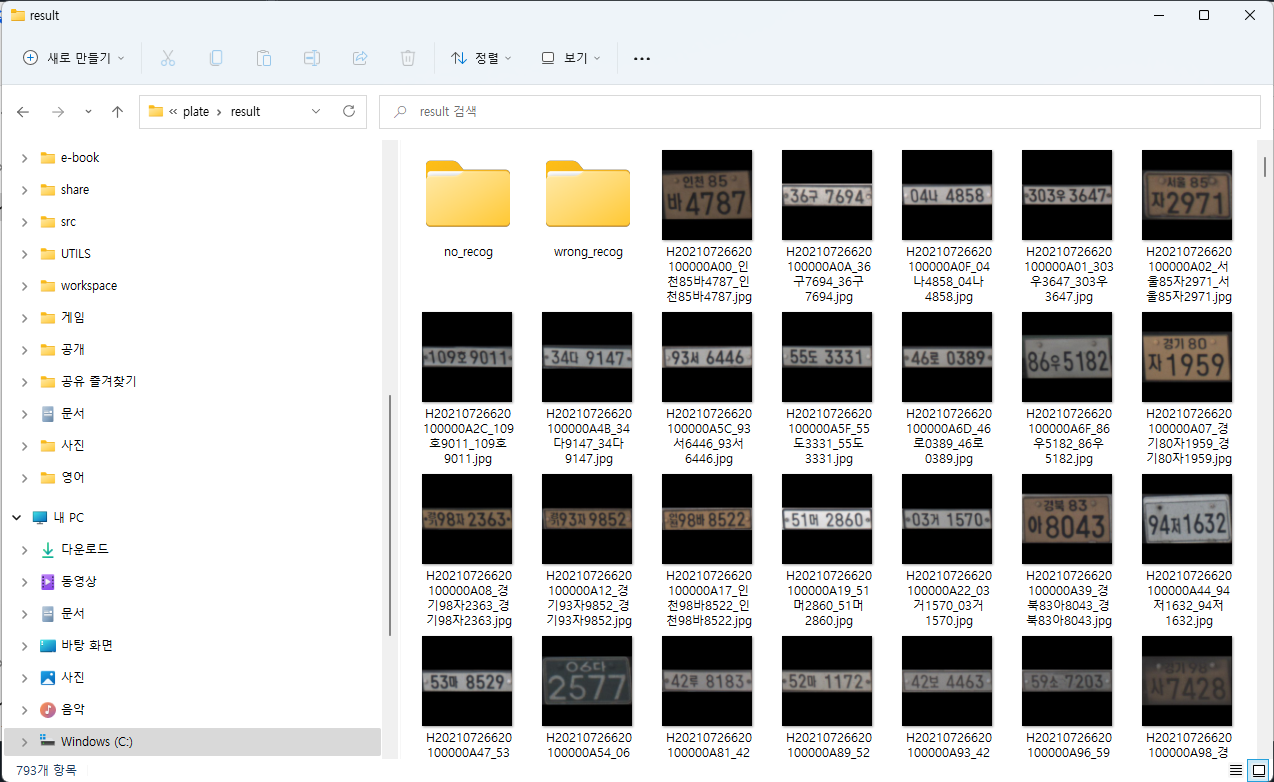
            det\_image\_np = extract\_sub\_image(image\_np,detections['detection\_boxes'][index],IMG\_SIZE,IMG\_SIZE,fixratio=False)

            ch = or\_det\_fn(or\_det\_model,det\_image\_np)

            category\_index\_temp[cindex]['name'] = ch

            twoLinePlate = True

    plate\_str =  predictPlateNumberODAPI(detections,platetype\_index,category\_index\_temp, CLASS\_DIC, twoLinePlate=twoLinePlate)



결과는

C:\SPB\_Data\RealTimeObjectDetection-main\Tensorflow\workspace\images\plate\result 폴더에 저장된다.

# 성능

서울문산 1000장에 대한 인식률 시험에 대한 데이터 이다.

수행시간: 366.55

건당 수행시간 : 0.37

인식률: 890 (89.00) %

정인식: 791 (88.88) %

오인식: 99 (11.12) %

인식실패: 110 (11.00) %

인식률이 무인과속에 적용하려면 95% 이상, 도로공사에 적용하려면 99% 이상, 오인식 2% 이하로 줄여야 하므로 아직 많은 보완이 이루어져야 한다.

# 이슈사항.

낮은 인식률

인식 : 경기91x5889



낮은 인식률은 대부분 문자에서 기인한다. 용도 문자 한글은 총 90개의 카테고리가 존재하고 있으나 라벨링한 종류는 48개 카테고리 정도에 미치고 있어 절대적인 라벨링한 샘플수가 모자라고, 카테고리가 커질수록 인식정확도가 낮아지는 문제가 있어 이에 대한 대비가 필요하다.

높은 오인식률

인식 서울81바3131



인식 87고81720



높은 오인식률은 기본적으로 낮은 인식률에서 기인하나 인식 후 후처리에 의해서 어느 정도 낮출수 있을 것으로 본다.

# 개선 방안

많은 레이블링 확보 (없는 레이블 우선 확보)

오토바이 번호판을 인식하려면 글자 단위로 인식하는 방법으로 가야 할 것 같음.

CRNN(CNN + BiLSTM + CTC) 적용 방안

칼라🡪 흑백 적용방안

전처리 TPS(Thin Plate Spine)

가변적 learning rate 적용 방안





형상관리 서버

Json을 보고 영상에서 필요한 영상을 잘라내는 코드

<https://gitlab.com/ksyoon70/labelconv.git>

잘라낸 영상에서 용도문자, 지역문자를 인식 코드

<https://github.com/ksyoon70/chardet.git>

번호판인식 + 번호인식 + 문자인식을 다 하며 번호판 인식결과를 도출하는 코드

<https://github.com/ksyoon70/RealTimeObjectDetection-main.git>

# 차량, 번호판 검지

이 챕터에서는 영상에서 차량과 번호판을 분리하는 로직을 트레이닝 한다.

레이블 파일을 아래와 같이 생성한다.

LPR\_Car-Plate\_Labels.txt 파일에 아래 내용을 기술한다.

car,승용

truck,트럭

bus,버스

bike,오토바이

bicycle,자전거

plate,번호판

objectDetecssd.py 파일에서

dataset\_category='car-plate' 이렇게 변경한다.

이 파일을 실행 시키면 아래와 같은 실행 명령이 생성 된다.

python c:\spb\_data\models/research/object\_detection/model\_main\_tf2.py --model\_dir=C:\SPB\_Data\RealTimeObjectDetection-main\Tensorflow\workspace\models\car-plate/my\_ssd\_mobnet --pipeline\_config\_path=C:\SPB\_Data\RealTimeObjectDetection-main\Tensorflow\workspace\models\car-plate/my\_ssd\_mobnet/pipeline.config --num\_train\_steps=5000

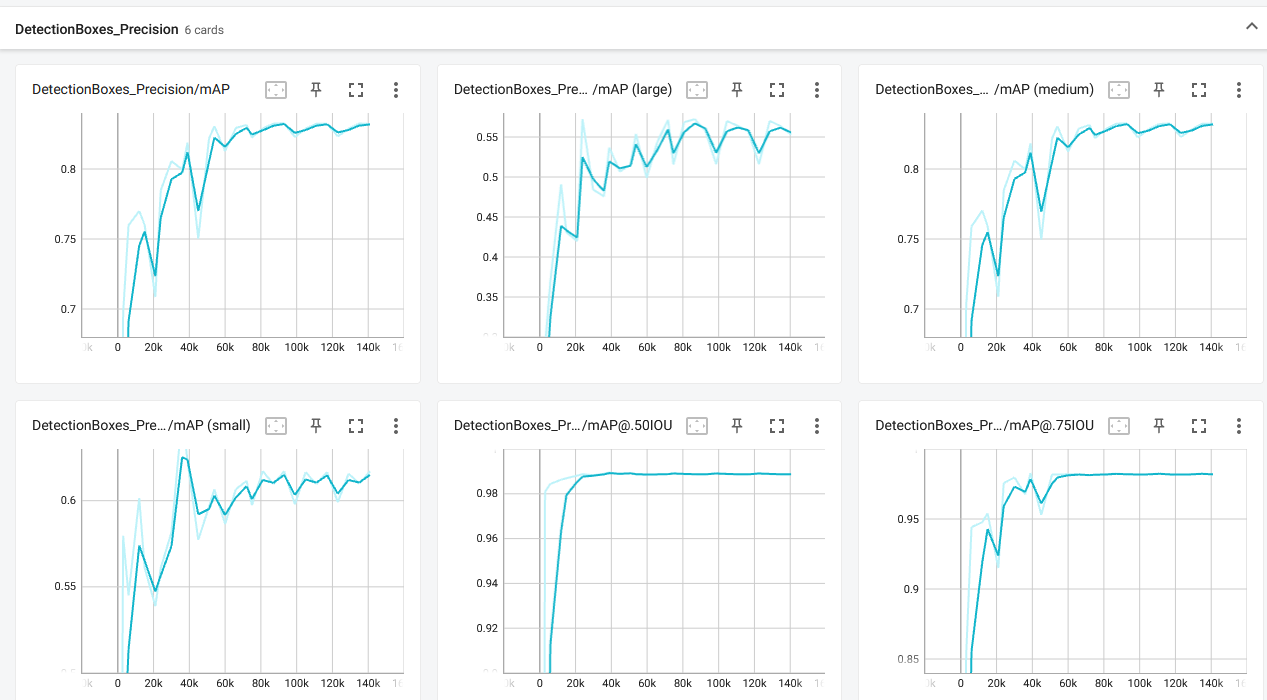
이후 C:\SPB\_Data\RealTimeObjectDetection-main\Tensorflow\workspace\annotations

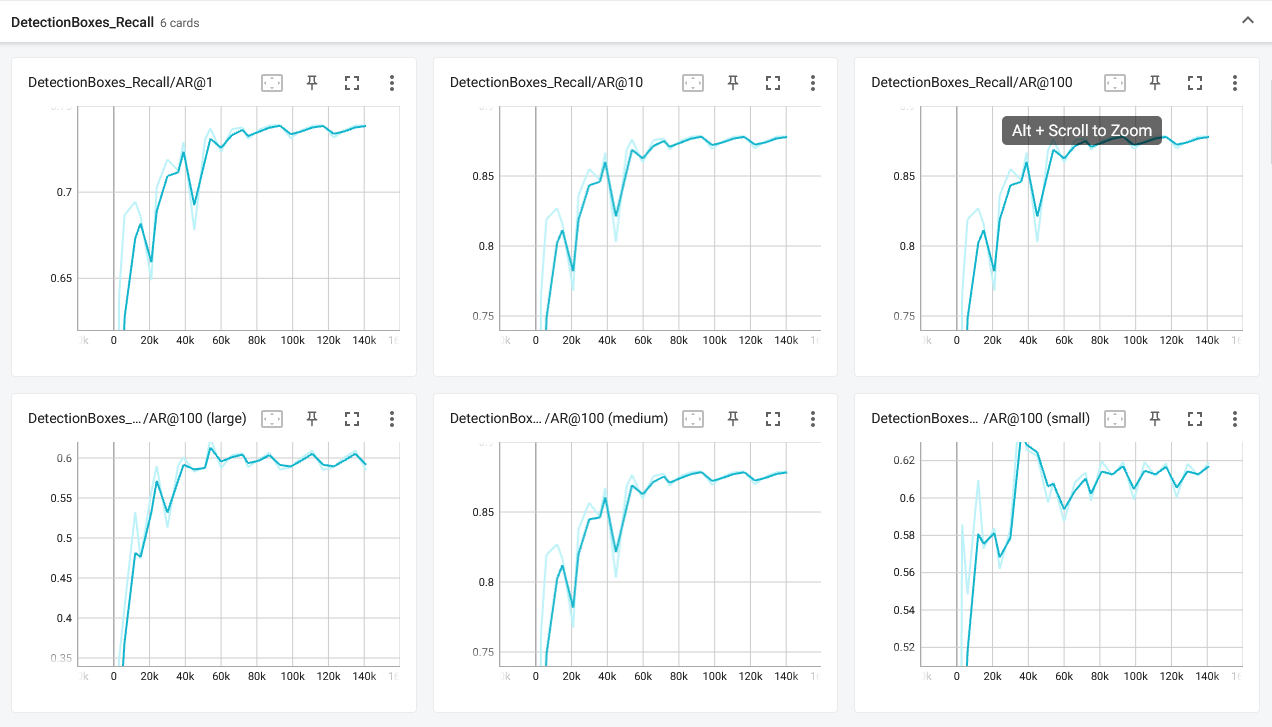
에서 label\_map.pbtxt 파일을 열어보면

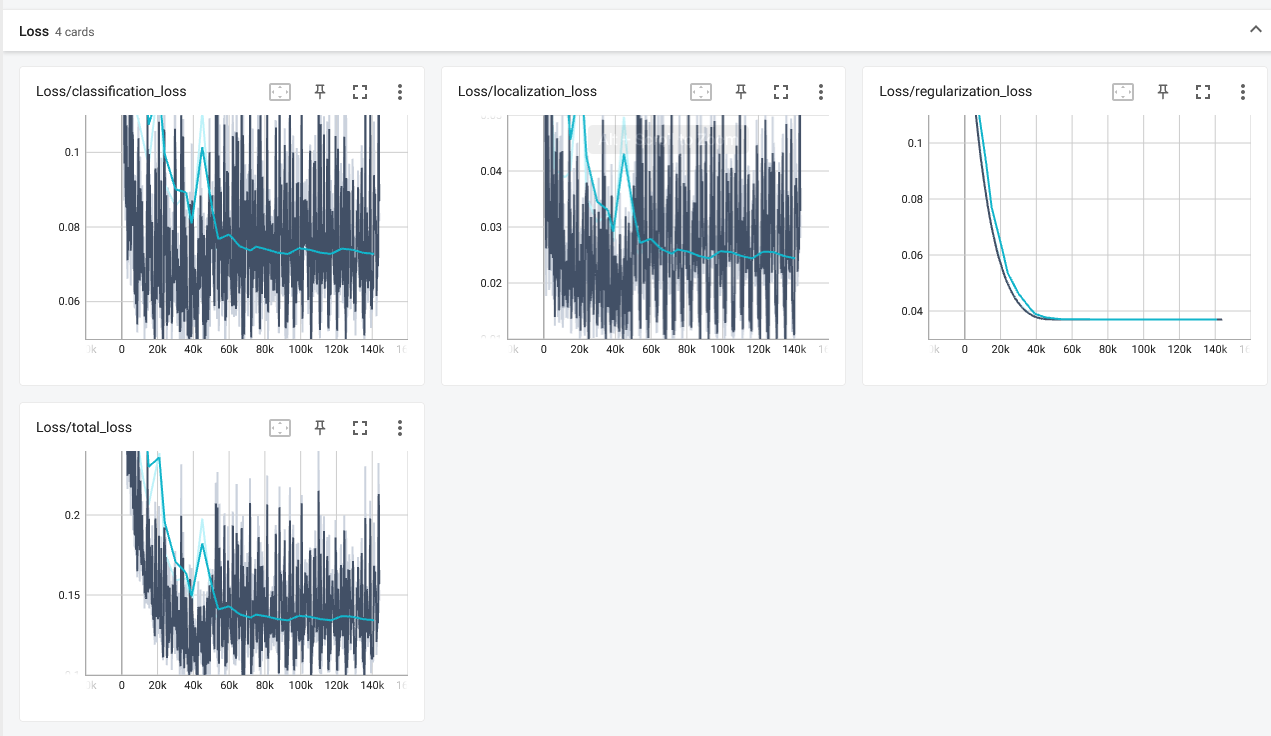
item {   
    name:'car'  
    id:1  
}  
item {   
    name:'truck'  
    id:2  
}  
item {   
    name:'bus'  
    id:3  
}  
item {   
    name:'bike'  
    id:4  
}  
item {   
    name:'bicycle'  
    id:5  
}  
item {   
    name:'plate'  
    id:6  
}

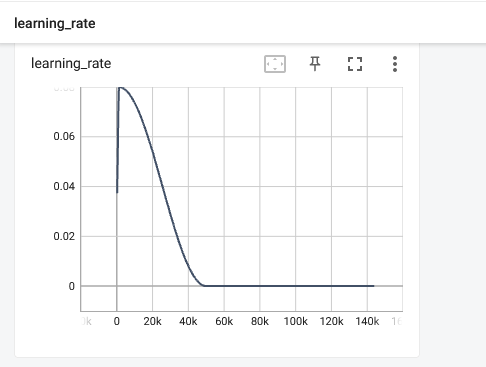
이제 데이터를 생성한다.

텐서 보드 사용





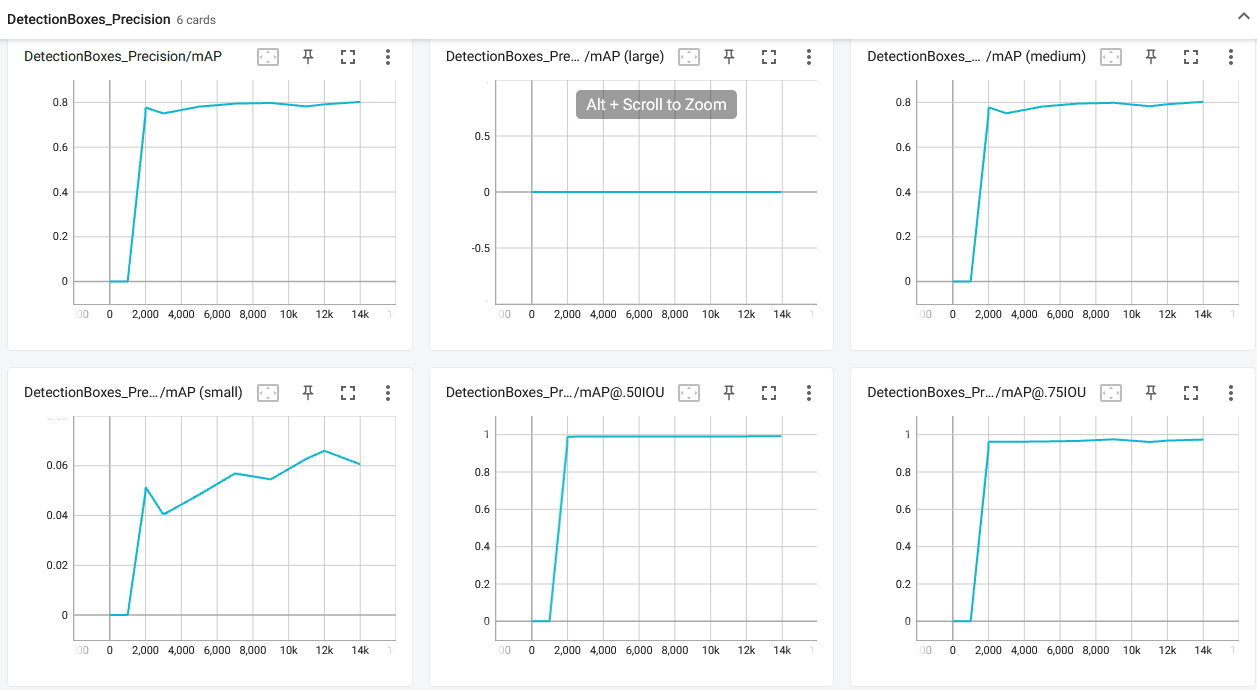




이륜차 번호판 인식

train 6714장

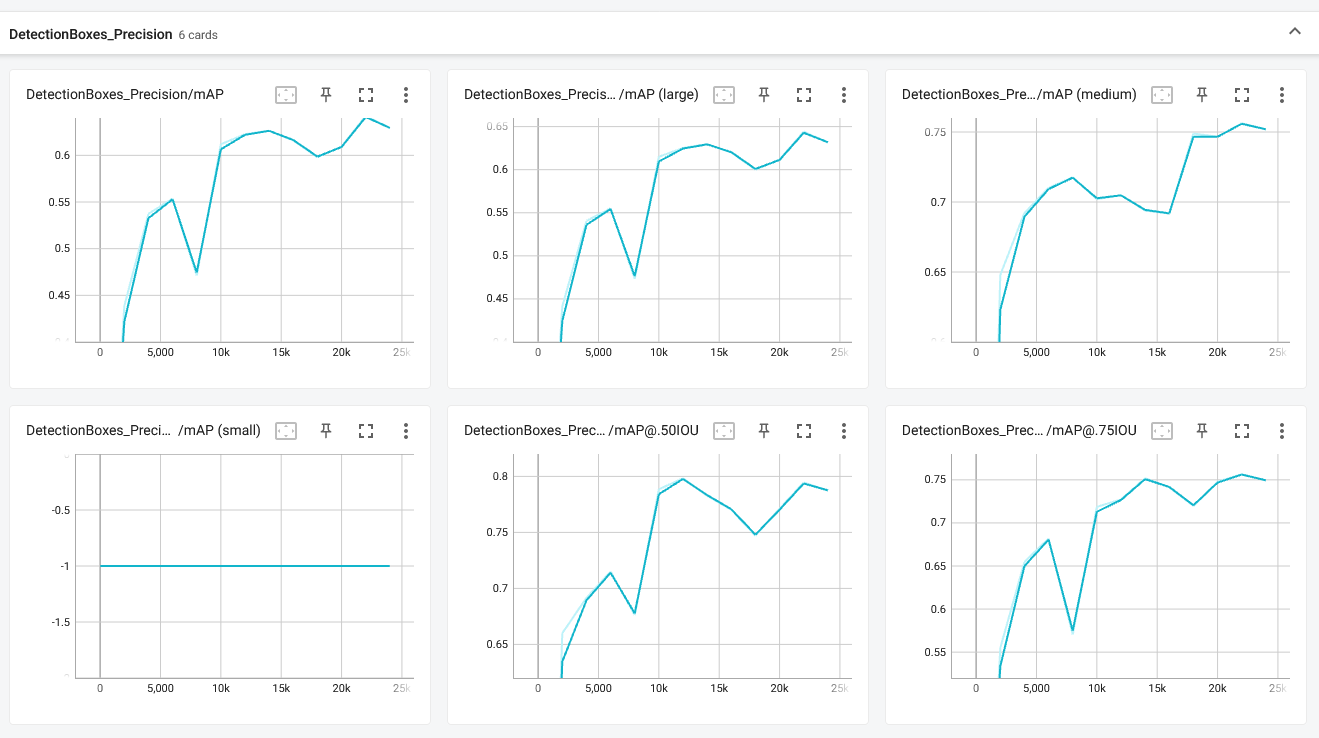
validation 1611장

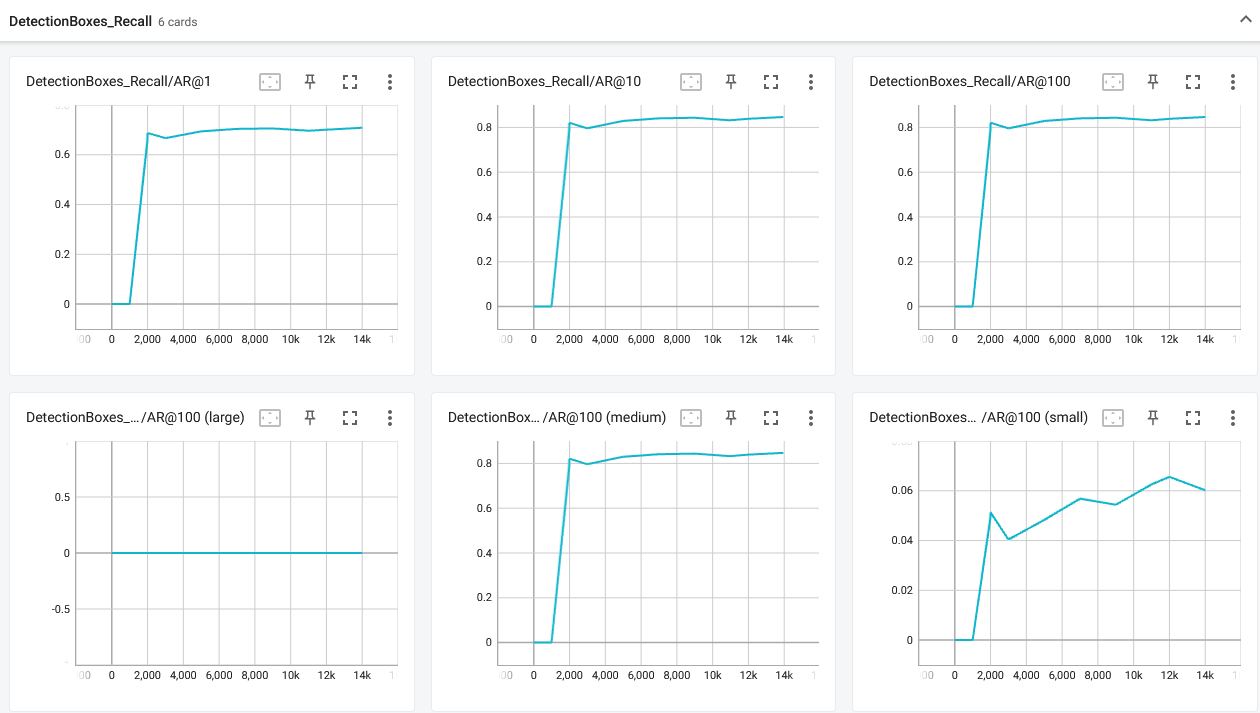


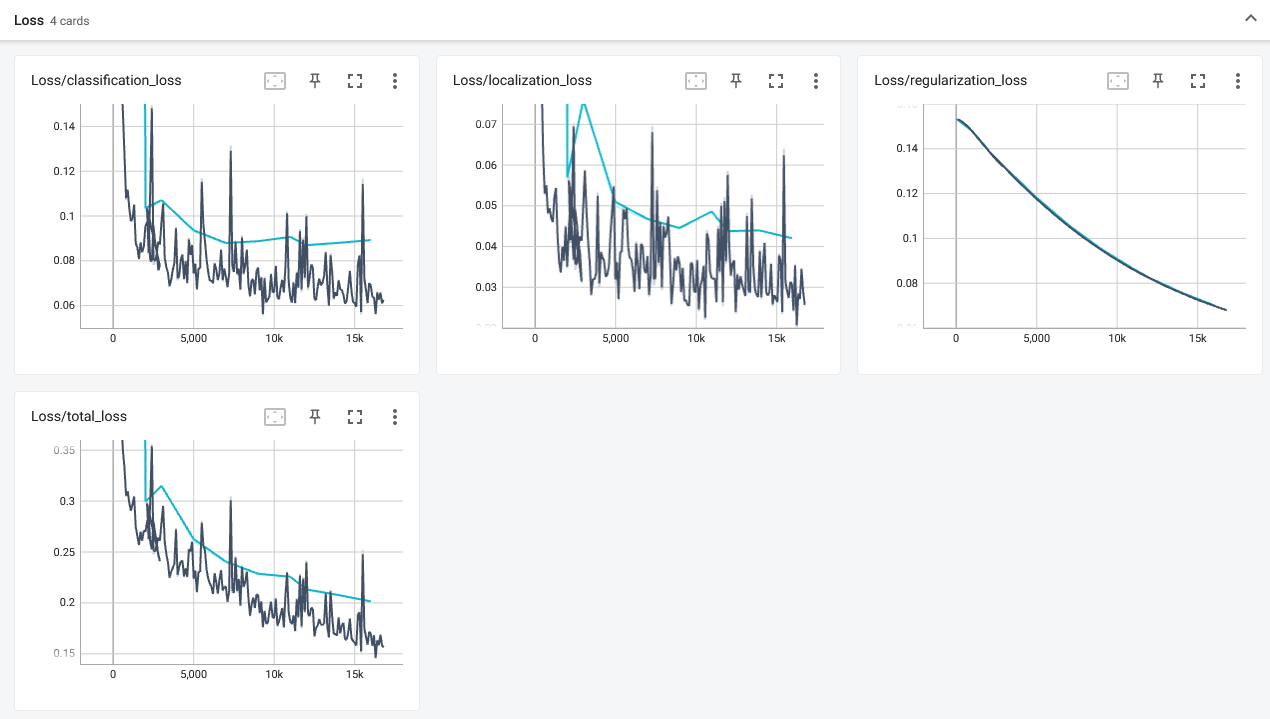
Car-plate

C:\SPB\_Data\RealTimeObjectDetection-main>python ..\models\research\object\_detection\model\_main\_tf2.py --model\_dir=Tensorflow\workspace\models\car-plate\my\_ssd\_mobnet --pipeline\_config\_path=Tensorflow\workspace\models\car-plate\my\_ssd\_mobnet\pipeline.config --num\_train\_steps=24075 --checkpoint\_every\_n=2000 --alsologtostderr

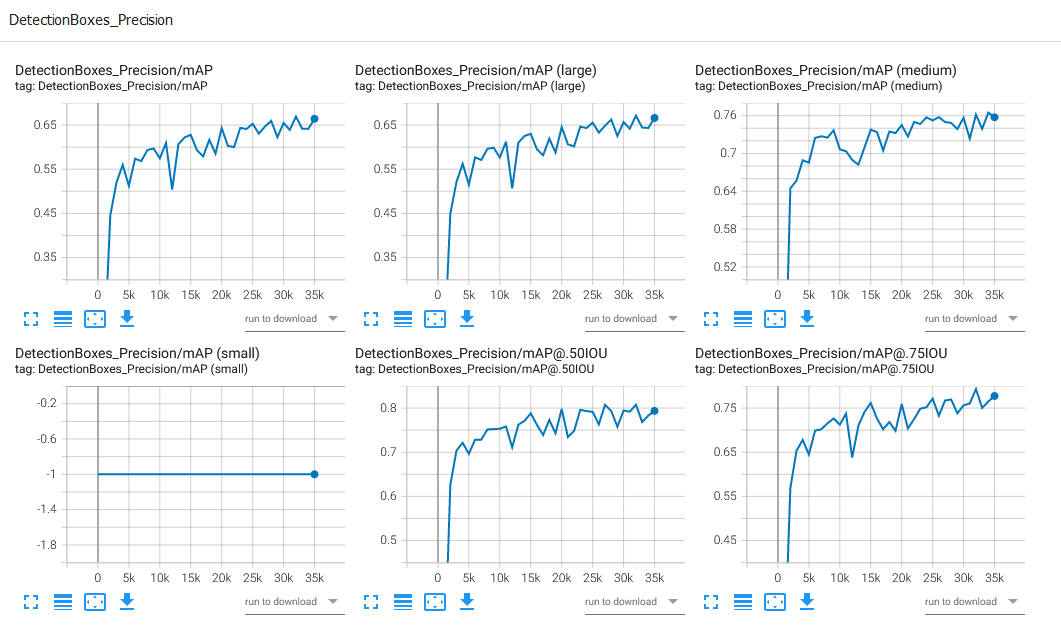
로 훈련함. 25595개 영상 파일

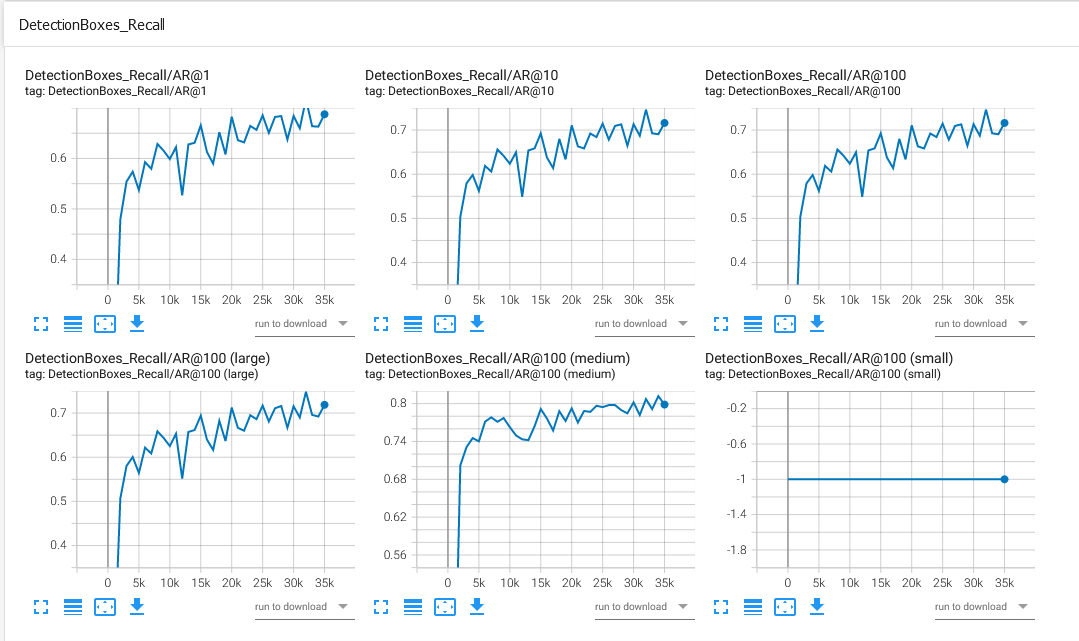


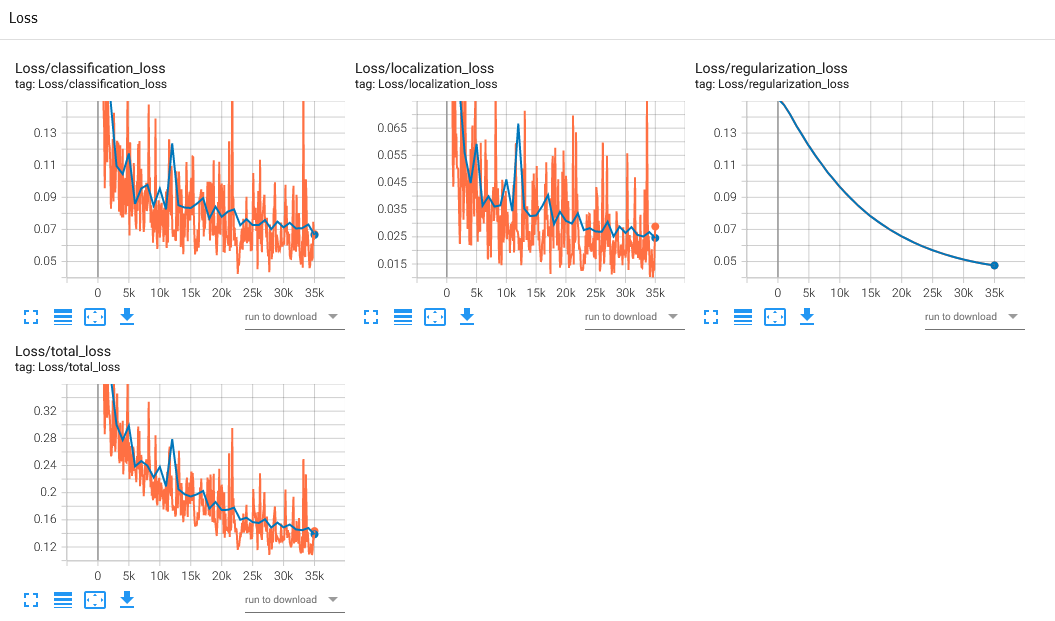




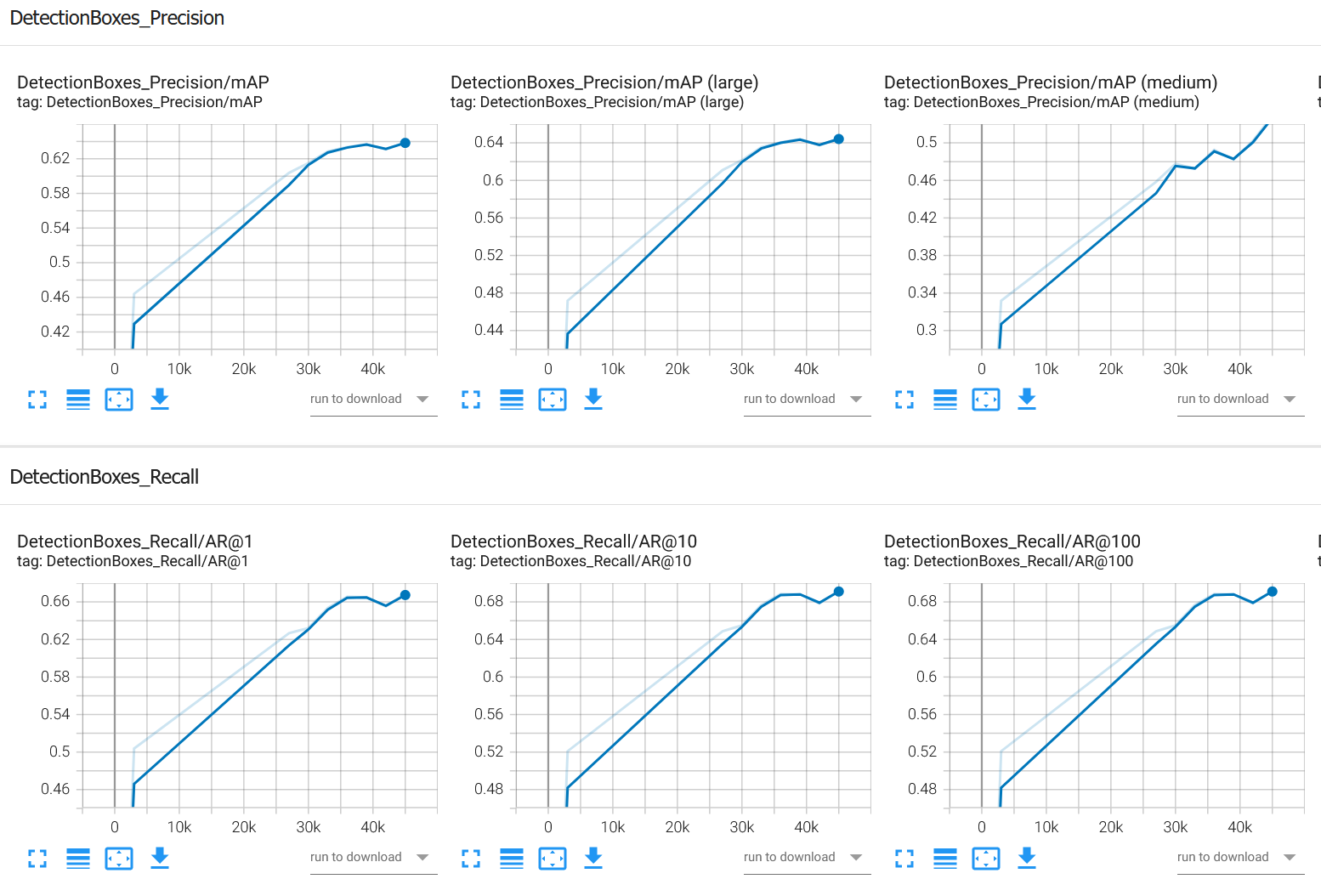
이륜차 헬멧 인식 고도화 (경성고 낮에 헬멧 인식 안되는 문제) 5월25일

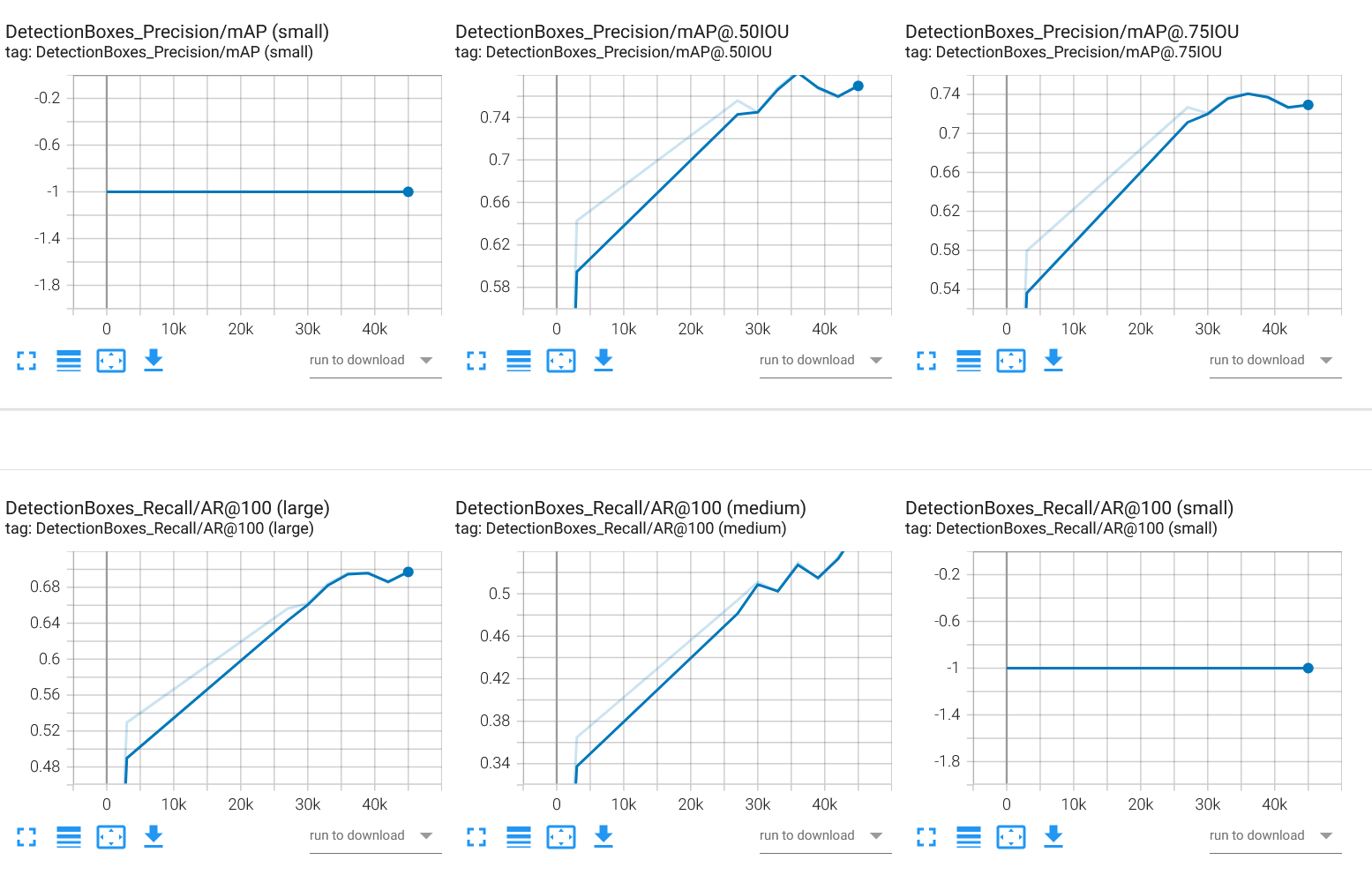


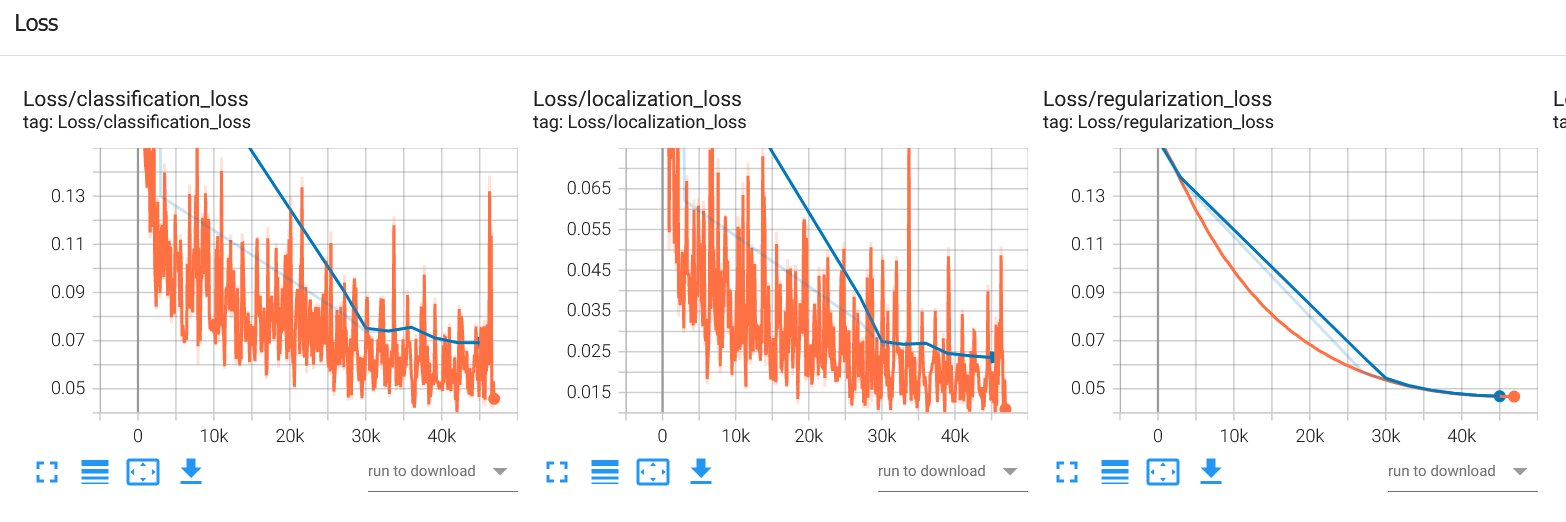


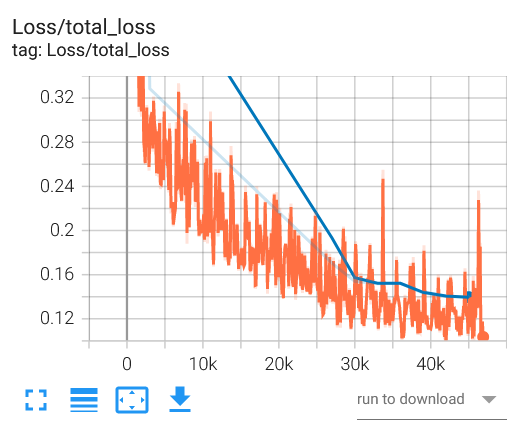


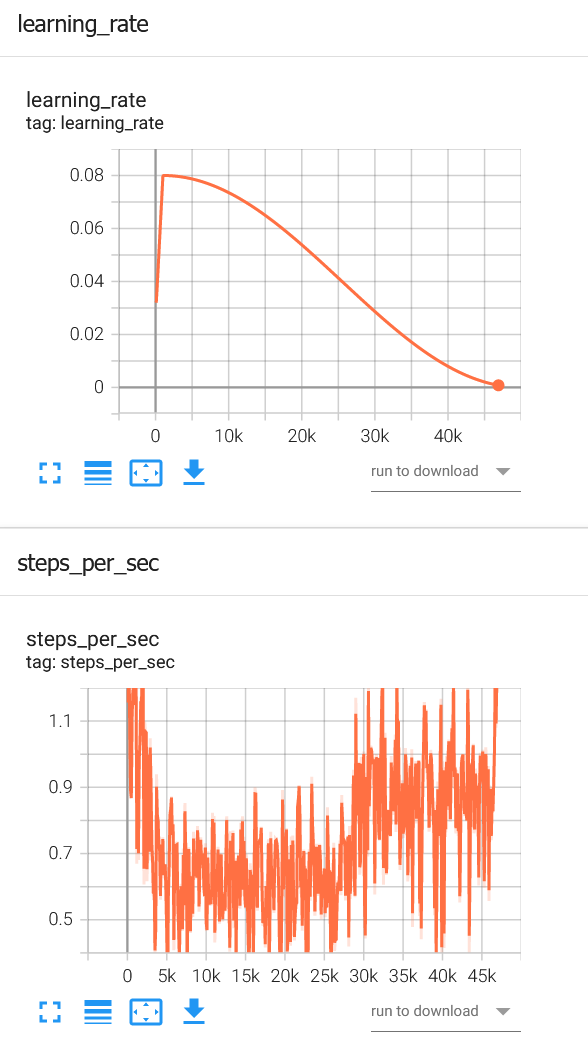
Car-Plate 320x320 23년 8월 28일



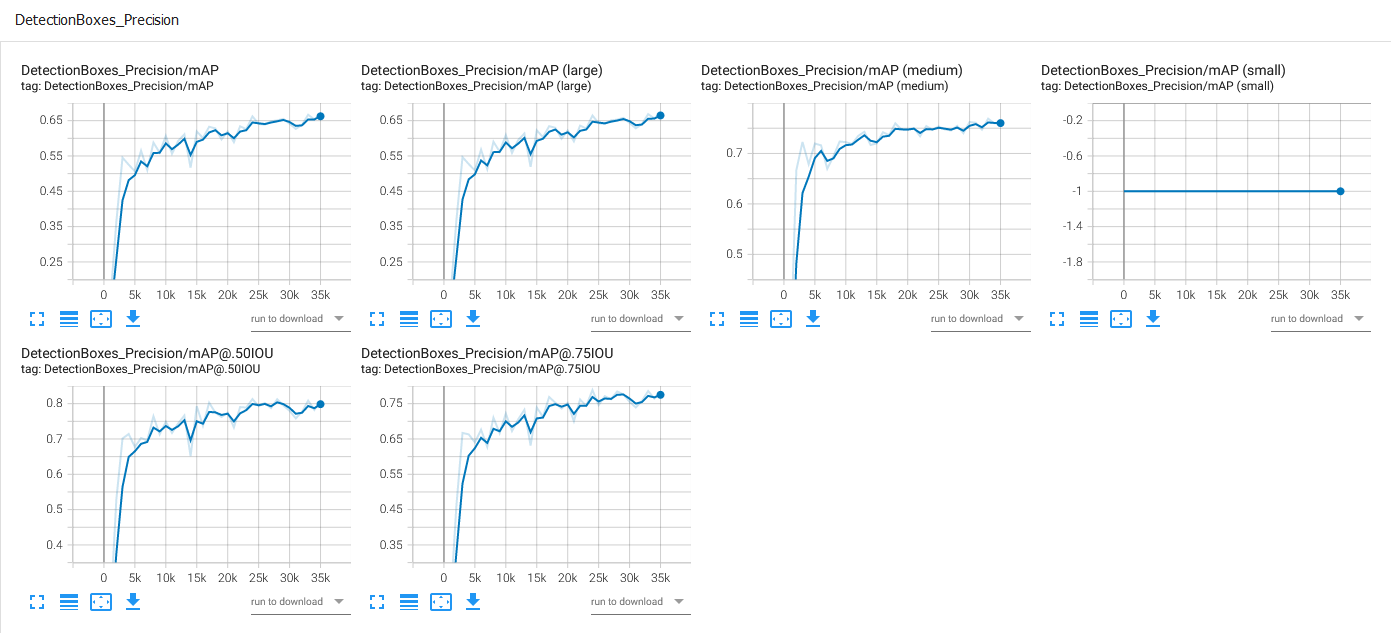


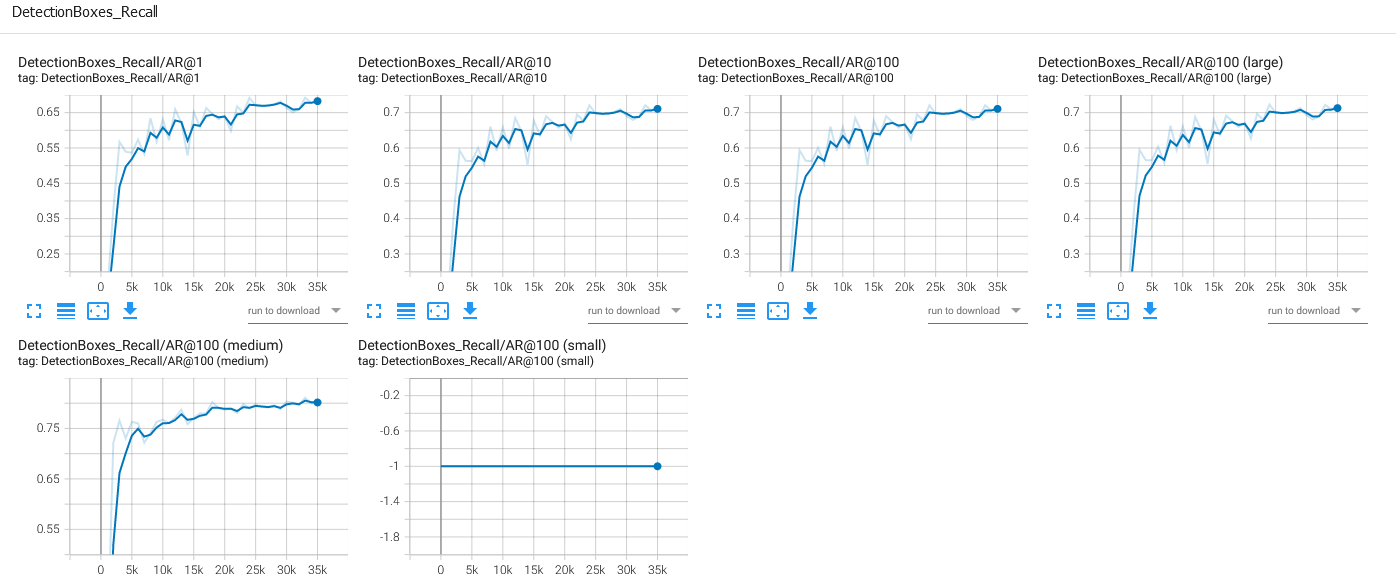


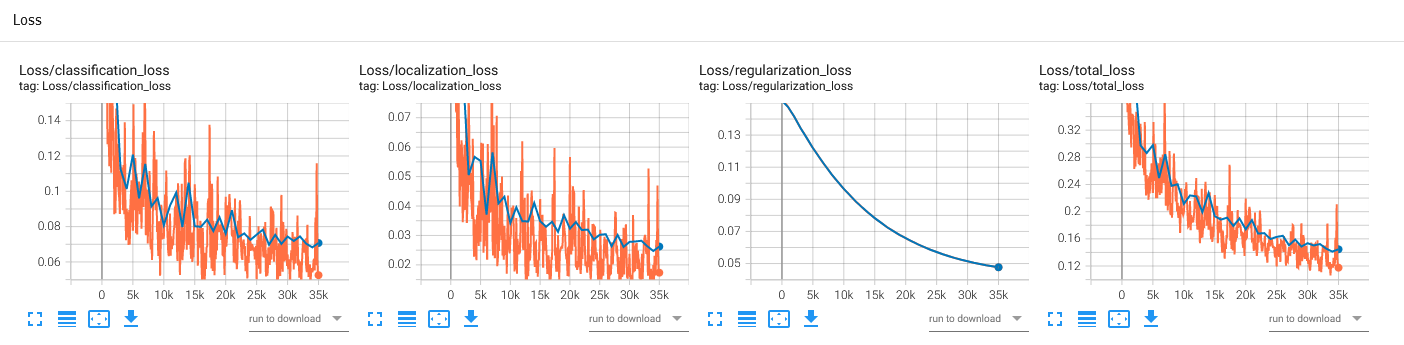




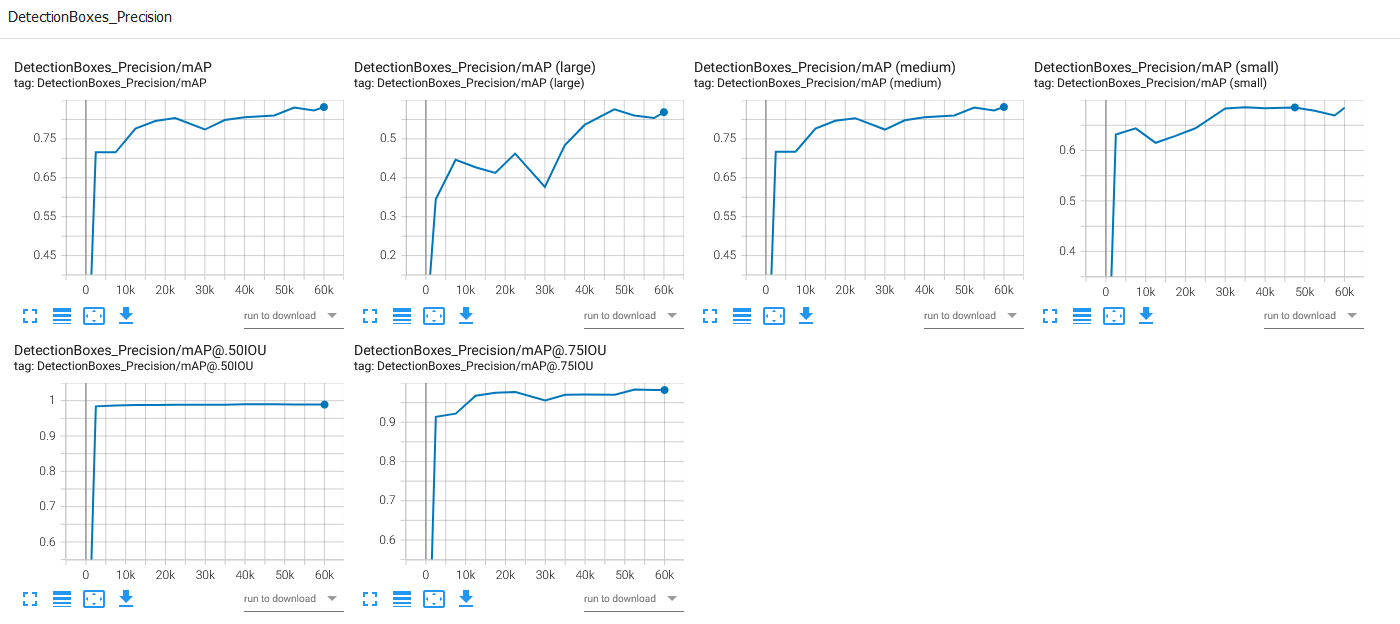
이륜차 헬멧 인식 고도화 (경성고 낮에 헬멧 인식 안되는 문제) 5월30일

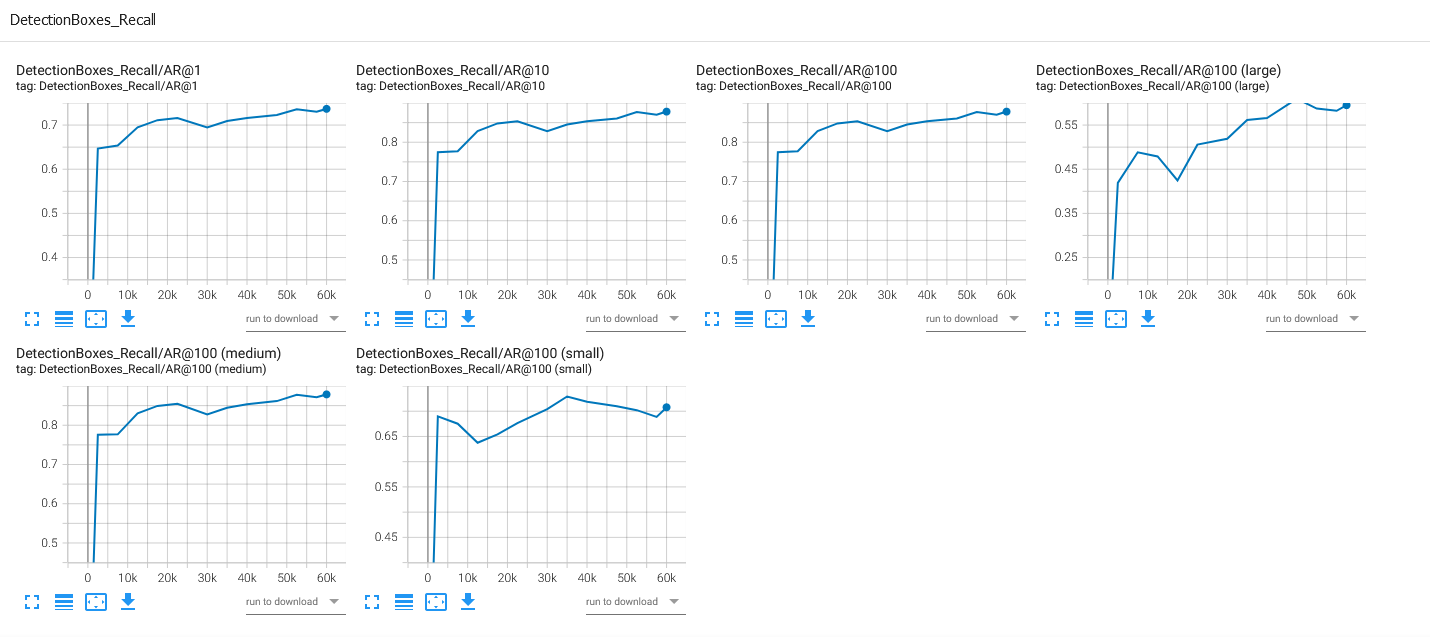


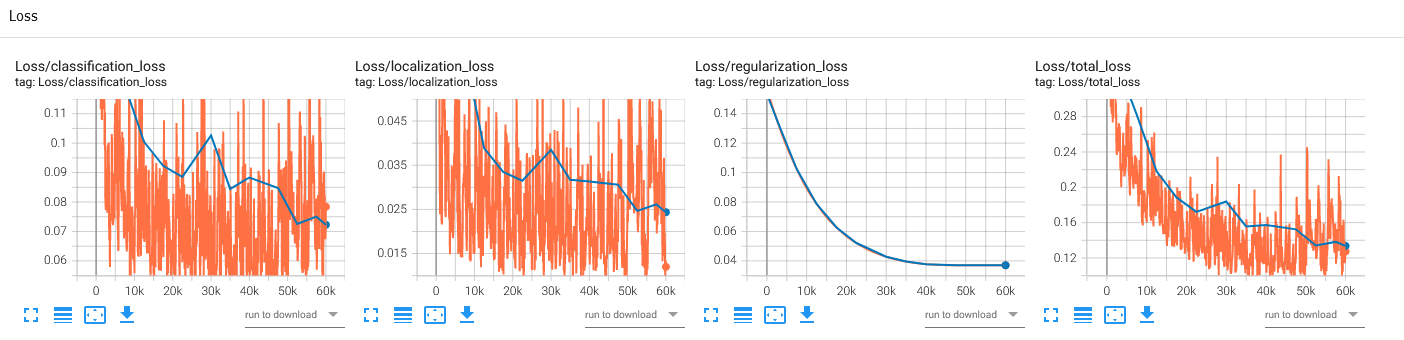




PlateImage training 23년 7월 10일







1. 추후 자동화 하는 코드를 넣을 예정이다. [↑](#footnote-ref-1)