오픈소스AI응용

Non-Helmet Kickboard Rider Detector

Git Hub :
https://github.com/Kkkimdongju/DKU_termproject_15

32170436 김 동 주
32211675 박 성 철
32217495 윤 택 민
32217789 최 진 혁

목 차

Step 01

프로젝트
소개 및 준비

Step 02

프로젝트 진행

Step 03

프로젝트 결론

Step 01

프로젝트
소개 및 준비

Step 02

프로젝트 진행

Step 03

프로젝트 결론

Step 01

프로젝트 소개

공용킥보드 상용화 이후 전동킥보드의 헬멧 착용 의무화 시행

오늘부터 킥보드 범칙금…헬멧 안 쓰고 '쌩쌩'

"심야 킥보드, 90%가 헬멧안써…인도주행 …법따로 현실따로 교통법규

"얼굴이 몇개냐?" 학생 4명이 'NO헬멧' 킥보드 역주행 아찔한 질주

'도로 위 무법자'로 등극한 전동 킥보드…응답자 10명 중 6명 "단속 제대로 안 된다" 불만 토로

한 밤 '거리의 무법자' 킥보드…헬멧 규제 '1년만에 유명무실'

'시행 1년' 전동킥보드 안전모 착용 의무화…'실효성' 논란 여전

전동킥보드 헬멧 의무화 실효성 '글쎄'…"단속도 쉽지 않

경찰의 불필요한 현장 출동을 줄이기 위한 대책 필요

Step 01

프로젝트 소개

헬멧 탐지기를 기반으로 사진 촬영 후 킥보드 이용자의 헬멧 착용 유무를 판별하는 솔루션 기획

사진 촬영

별도의 탐지기(카메라)로
킥보드 이용자를 촬영

유형 분류

헬멧을 쓴 이용자와 헬멧을
쓰지 않은 이용자를 분류

Step 01

프로젝트 준비

교수님 피드백을 반영하여 데이터, 모델 구성

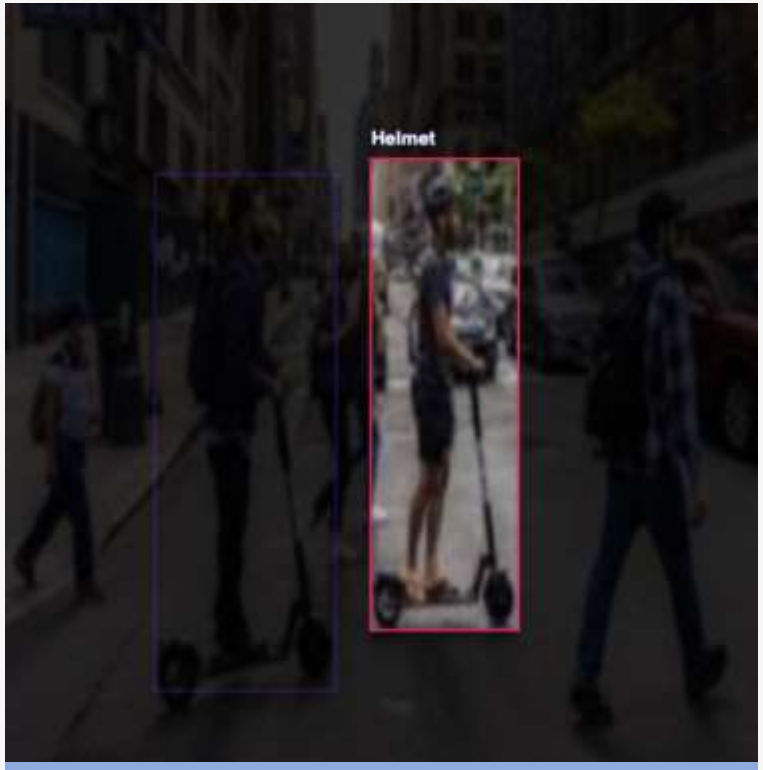
그리고 실제로 테스트할때는 모든 이미지가 킥보드를 타는 이미지가 아니잖아요? 예를 들어 CCTV 같은거에 적용된다고 했을때 사람들이 대부분은 킥보드를 타고 있지 않음에도 불구하고 (그냥 걸어다니고 있음에도 불구하고) 헬멧을 쓰지 않았다고 평가를 내리게 되면 조금 의미가 떨어질 수 있겠죠. 이런걸 해결하기 위해서 예를 들어서 Yolo와 같은 모델로 공유 킥보드를 사용하는 사람이 있는 box를 찾고, 그 box 내에서 헬멧을 썼는지 안썼는지를 보는 것이 더 좋을수 있죠.

Step 01

프로젝트 준비

사람, 킥보드, 헬멧 등을 클래스화 하는 방식이 아닌 **헬멧 착용 이용자**와 **헬멧 미착용 이용자** 2개의 클래스로 바운딩박스 라벨링

사람, 킥보드, 헬멧 분류 (X)



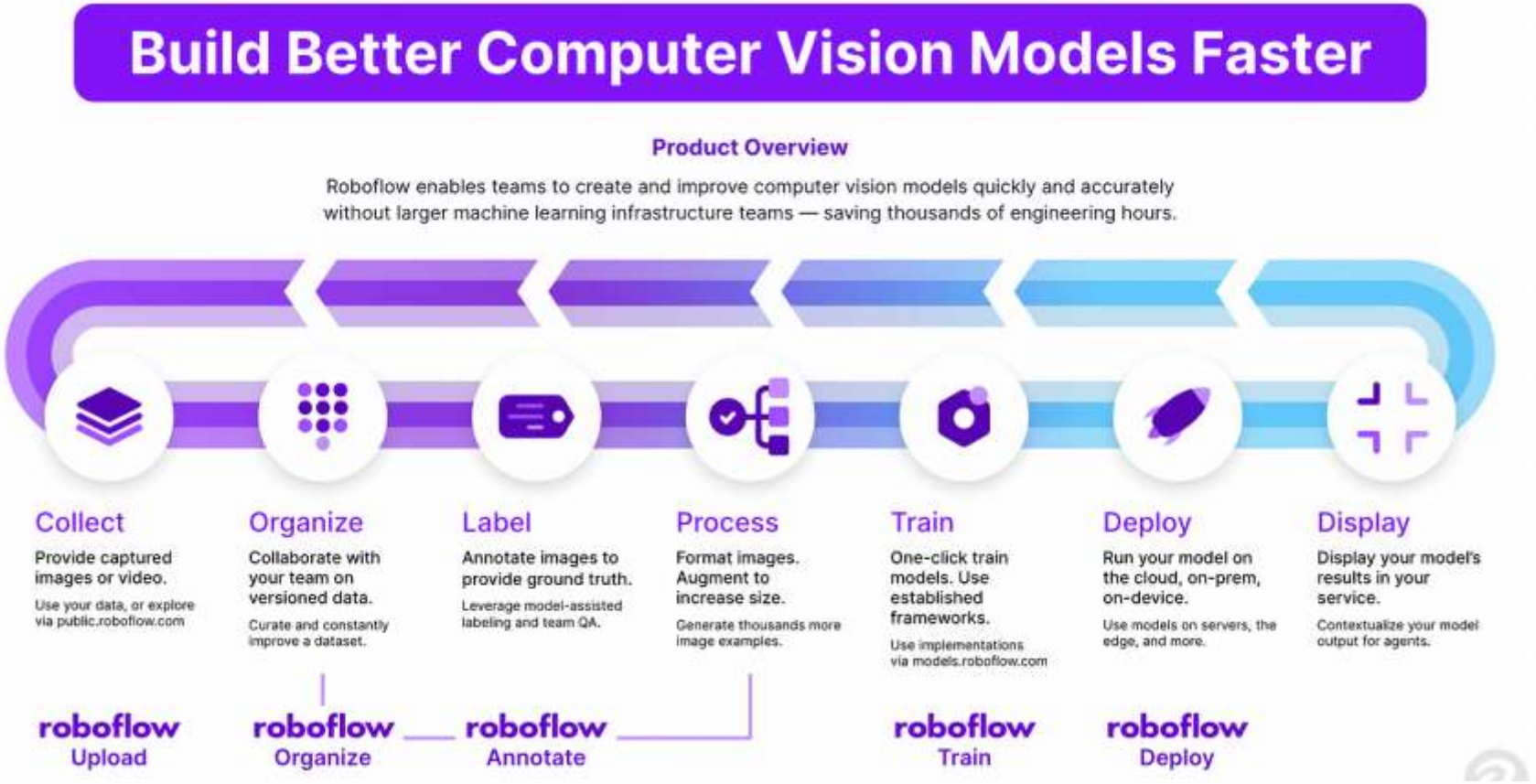
헬멧 착용, 헬멧 미착용 분류 (O)

Step 01

프로젝트 준비

roboflow를 이용하여 이미지 데이터 Labeling, Augmentation 수행

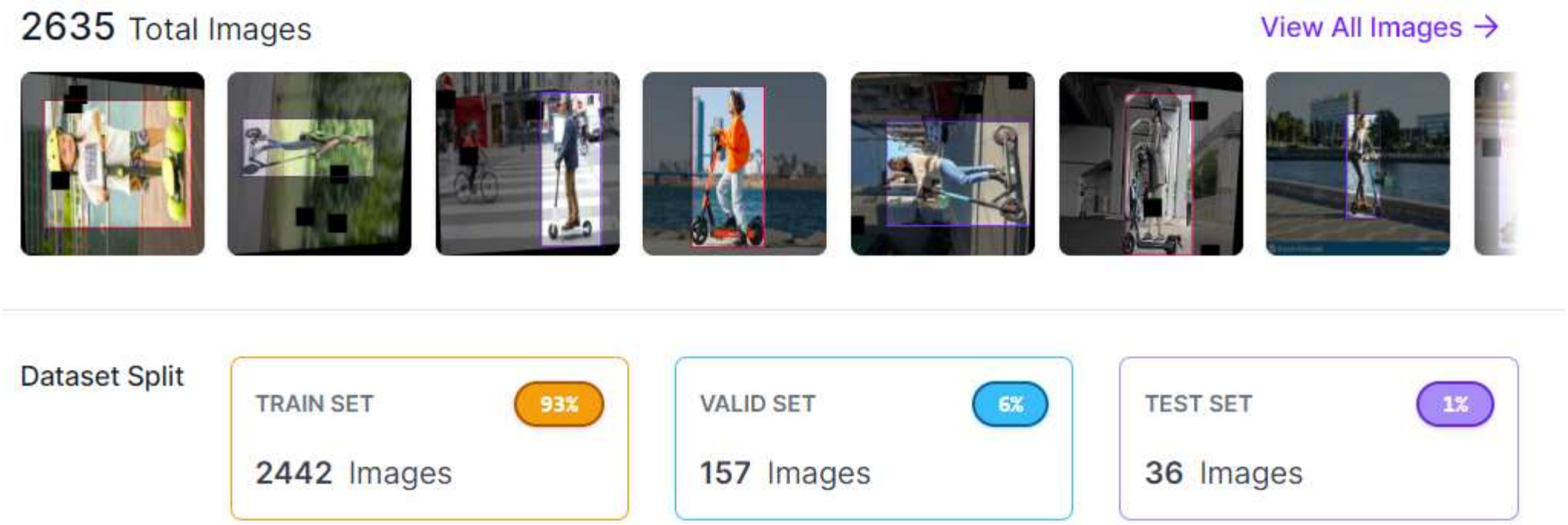
- 라벨링 조건 : 킥보드 전체 모습과 이용자의 머리를 합쳐 하나의 클래스로 라벨링



Step 01

프로젝트 준비

Labeling, Augmentation을 끝마친 2635개의 이미지 준비



https://app.roboflow.com/helmet-detector/dku-opensourceai-15-helmet/deploy/1

Step 01

프로젝트
소개 및 준비

Step 02

프로젝트 진행

Step 03

프로젝트 결론

Step 02

프로젝트 진행 (YOLOv8)

roboflow의 데이터를 colab으로 다운로드받고 YOLOv8 모델을 사용하여 데이터 학습

```
[1]   1 !wget -O Helmet_data.zip https://app.roboflow.com/ds/Hry4ceH5Vc?key=ebkn8Ldswl

--2023-12-04 14:52:54--  https://app.roboflow.com/ds/Hry4ceH5Vc?key=ebkn8Ldswl
Resolving app.roboflow.com (app.roboflow.com)... 151.101.1.195, 151.101.65.195
Connecting to app.roboflow.com (app.roboflow.com)|151.101.1.195|:443... connected.
HTTP request sent, awaiting response... 302 Found
Location: https://storage.googleapis.com/roboflow-platform-exports/suBjbML0TFOE1NsPhNvcuv
--2023-12-04 14:52:55--  https://storage.googleapis.com/roboflow-platform-exports/suBjbML
Resolving storage.googleapis.com (storage.googleapis.com)... 142.250.148.207, 142.251.171
Connecting to storage.googleapis.com (storage.googleapis.com)|142.250.148.207|:443... con
HTTP request sent, awaiting response... 200 OK
Length: 125492866 (120M) [application/zip]
Saving to: 'Helmet_data.zip'

Helmet_data.zip     100%[===================>] 119.68M   176MB/s    in 0.7s

2023-12-04 14:52:56 (176 MB/s) - 'Helmet_data.zip' saved [125492866/125492866]
```

```
10 model.train(data = '/content/Helmet_Data/Helmet_Data.yaml', epochs=70, patience=25, batch=16, imgsz=320)
11
```

6	-1	2	197632	ultralytics.nn.modules.block.C2f	[128, 128, 2, True]
7	-1	1	295424	ultralytics.nn.modules.conv.Conv	[128, 256, 3, 2]
8	-1	1	460288	ultralytics.nn.modules.block.C2f	[256, 256, 1, True]
9	-1	1	164608	ultralytics.nn.modules.block.SPPF	[256, 256, 5]
10	-1	1	0	torch.nn.modules.upsampling.Upsample	[None, 2, 'nearest']
11	[-1, 6]	1	0	ultralytics.nn.modules.conv.Concat	[1]
12	-1	1	148224	ultralytics.nn.modules.block.C2f	[384, 128, 1]
13	-1	1	0	torch.nn.modules.upsampling.Upsample	[None, 2, 'nearest']
14	[-1, 4]	1	0	ultralytics.nn.modules.conv.Concat	[1]
15	-1	1	37248	ultralytics.nn.modules.block.C2f	[192, 64, 1]
16	-1	1	36992	ultralytics.nn.modules.conv.Conv	[64, 64, 3, 2]
17	[-1, 12]	1	0	ultralytics.nn.modules.conv.Concat	[1]
18	-1	1	123648	ultralytics.nn.modules.block.C2f	[192, 128, 1]
19	-1	1	147712	ultralytics.nn.modules.conv.Conv	[128, 128, 3, 2]
20	[-1, 9]	1	0	ultralytics.nn.modules.conv.Concat	[1]
21	-1	1	493056	ultralytics.nn.modules.block.C2f	[384, 256, 1]
22	[15, 18, 21]	1	751702	ultralytics.nn.modules.head.Detect	[2, [64, 128, 256]]

```
Model summary: 225 layers, 3011238 parameters, 3011222 gradients, 8.2 GFLOPs

Transferred 319/355 items from pretrained weights
TensorBoard: Start with 'tensorboard --logdir runs/detect/train', view at http://localhost:6006/
Freezing layer 'model.22.dfl.conv.weight'
AMP: running Automatic Mixed Precision (AMP) checks with YOLOv8n...
```

Step 02

프로젝트 진행 (YOLOv8)

epochs에 따라 달라지는 결과

epochs = 30

epochs = 50

epochs = 100

Step 02

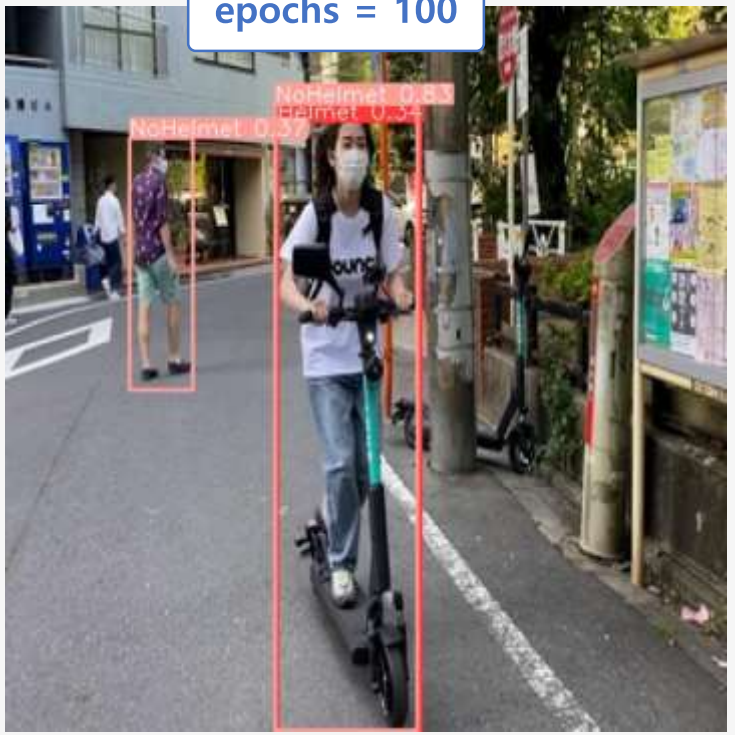
프로젝트 진행 (YOLOv8)

epochs가 100이 넘어가면서 과적합(Overfitting) 발생

epochs = 30

epochs = 50

epochs = 100



epochs 100부터 행인을 이용자로 인식

Step 02

프로젝트 진행 (YOLOv8)

epochs가 100이 넘어가면서 과적합(Overfitting) 발생

epochs = 100

epochs = 200

100일때보다 성능 하락

Step 02

프로젝트 진행 (YOLOv8)

모델 학습 epochs를 70 ~ 80으로 설정하였을 때가 가장 적합하다고 판단

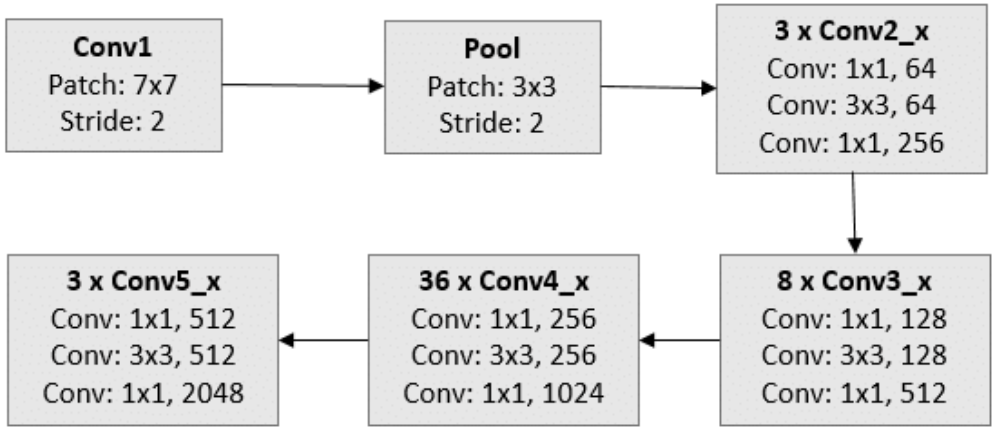
epochs = 70

epochs = 80

Step 02

프로젝트 진행 (ResNet-152 , MobileNetV3)

1. roboflow의 데이터를 json 형식으로 다운로드 하여, 데이터셋 구축
2. 실시간 Object Detection 에 최적화된 YOLOv8 모델과 비교할 모델 선정
3. 높은 정확도를 목표로하는 ResNet-152 , 임베디드 시스템에서 효율적인 MobileNetV3 모델 선정



ResNet-152 Layer

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 MobilenetV3small (Function  (None, 7, 7, 576)         939120
 al)

 global_average_pooling2d_1  (None, 576)               0
  (GlobalAveragePooling2D)

 dense_3 (Dense)             (None, 256)               147712

 dense_4 (Dense)             (None, 128)               32896

 dense_5 (Dense)             (None, 3)                 387
```

MobileNetV3 Layer

Step 02

프로젝트 진행 (ResNet-152)

```
[ ] # 텐서보드를 활용한 학습과정 모니터링
    import tensorflow as tf

    # Define ResNet152
    base_model = ResNet152(include_top=False, weights='imagenet', input_shape=(224, 224, 3))
    for layer in base_model.layers:
        layer.trainable = False

    # 모델 구성
    x = Flatten()(base_model.output)
    x = Dense(128, activation='relu')(x)
    output = Dense(len(coco_data["categories"]), activation='softmax')(x)
    model = Model(inputs=base_model.input, outputs=output)

    # 모델 컴파일
    model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

    # 텐서보드 로그 디렉토리 지정
    log_dir = "logs/weights_visualization"

    # 텐서보드 콜백 설정
    tensorboard_callback = tf.keras.callbacks.TensorBoard(log_dir=log_dir, histogram_freq=1)
```

```
[ ] # 모델 학습
    num_epochs = 200
    batch_size = 50
    model.fit(images, labels, epochs=num_epochs, batch_size=batch_size, callbacks=[tensorboard_callback])
```

< ResNet-152 >

- Epochs = 200
- Batch_size = 50

Step 02

프로젝트 진행 (MobileNetV3)

```
# 모델 구성
base_model = MobileNetV3Small(weights='imagenet',
                              include_top=False,
                              input_shape=(224, 224, 3))

model = Sequential()
model.add(base_model)
model.add(GlobalAveragePooling2D())  # 메모리 효율을 위해 Global Average Pooling 레이어 사용
model.add(Dense(256, activation='relu'))
model.add(Dense(128, activation='relu'))
model.add(Dense(3, activation='softmax'))

# 모델 컴파일
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

# 텐서보드 로그 디렉토리 지정
log_dir = "logs/weights_visualization"

# 텐서보드 콜백 설정
tensorboard_callback = tf.keras.callbacks.TensorBoard(log_dir=log_dir, histogram_freq=1)

import numpy as np
from tensorflow.keras.utils import to_categorical

# 이미지 데이터를 numpy 배열로 변환
images = np.array(images)
# 라벨을 one-hot 인코딩 된 텐서 형태로 변환
labels = to_categorical(labels, num_classes=len(coco_data["categories"]))

# 모델 학습
num_epochs = 70
model.fit(images, labels, epochs=num_epochs, batch_size=batch_size, callbacks=[tensorboard_callback])
```

< MobileNetV3 >

- 메모리 효율을 위한 별도의 풀링기법 사용
- Output Layer 별도 설정
- Epochs = 70
- Batch_size = 32

Step 01

프로젝트
소개 및 준비

Step 02

프로젝트 진행

Step 03

프로젝트 결론

Step 03

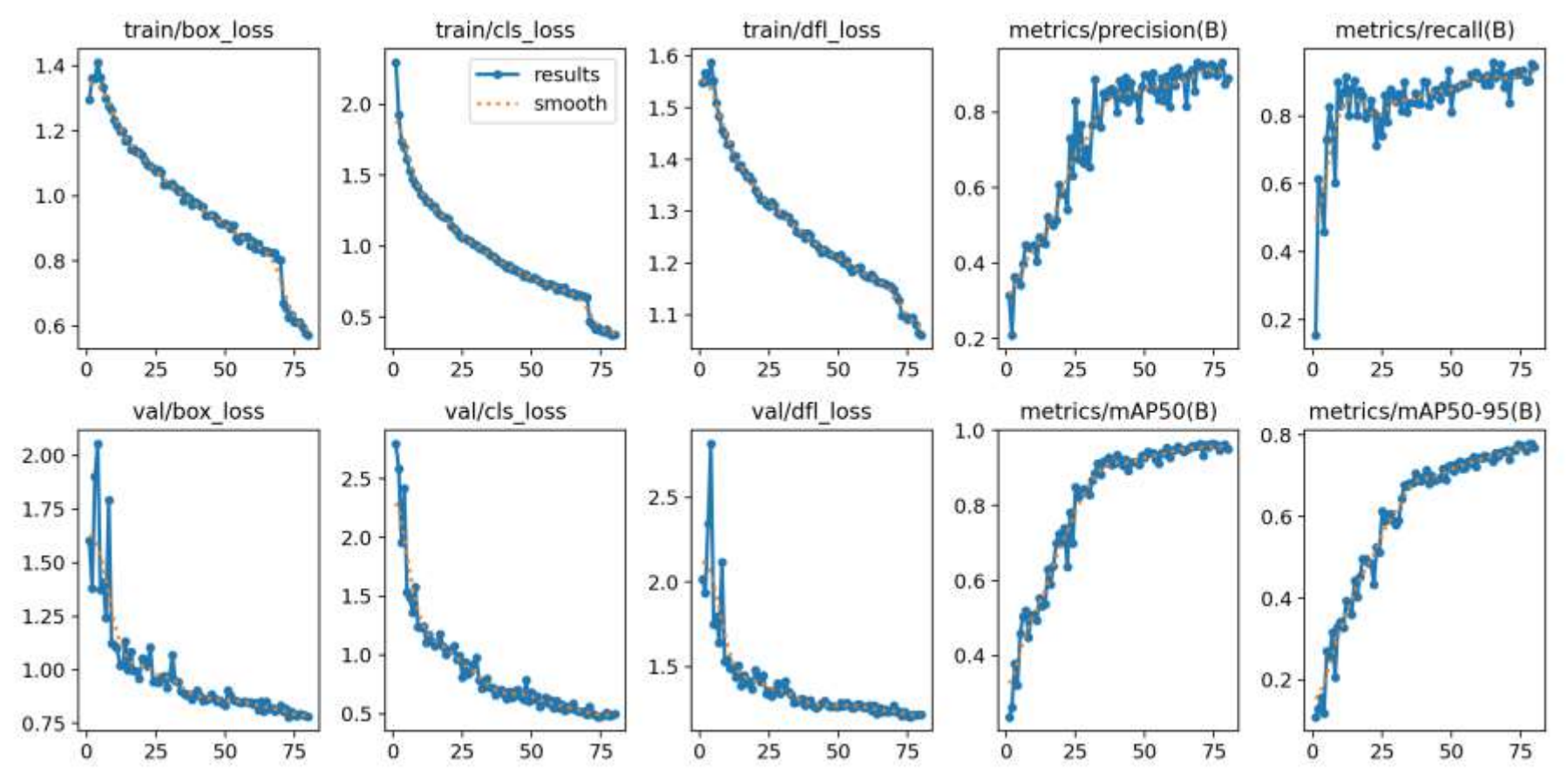
프로젝트 결과 (YOLOv8)

예측값의 범위는 최소/최대 0.70 ~ 0.90이고 킥보드 이용자를 잘 인식하는 것을 볼 수 있다

행인은 인식하지 않는다

Step 03

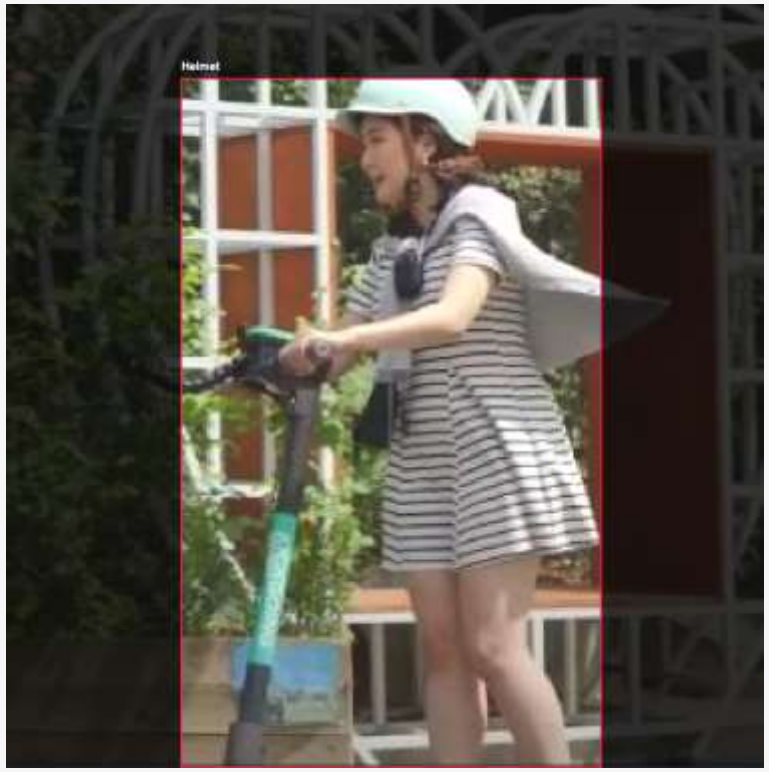
각 항목별 결과 그래프(YOLOv8)



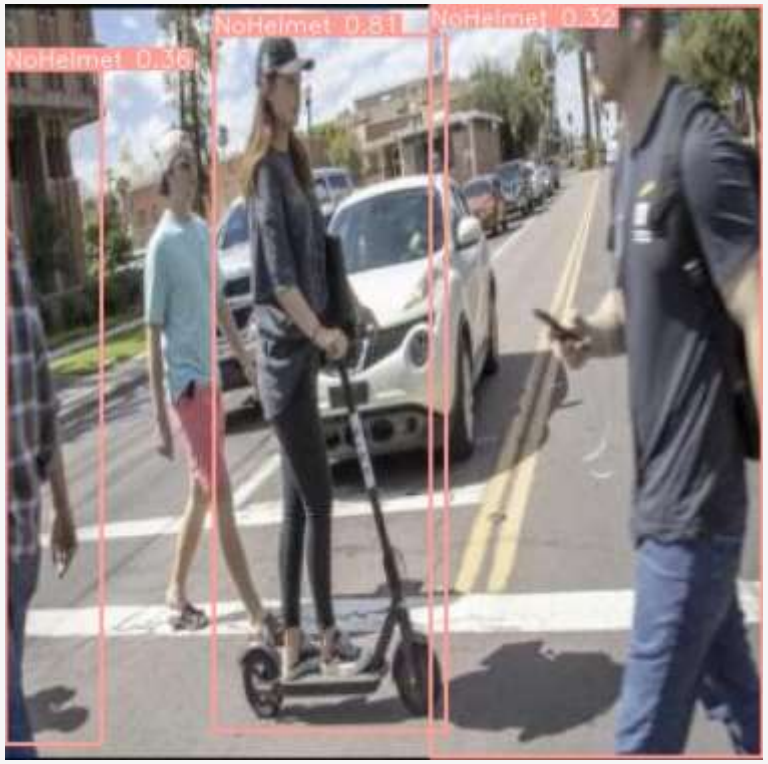
Step 03

프로젝트 특이사항 (YOLOv8)

킥보드 전체 모습을 묶지 않으면 유사한 모습의 행인도 인식



킥보드가 잘린 사진으로 라벨링



사진이 잘린 행인들도 인식

라벨링 작업을 다시 수행해야 한다

Step 03

프로젝트 특이사항 (YOLOv8)

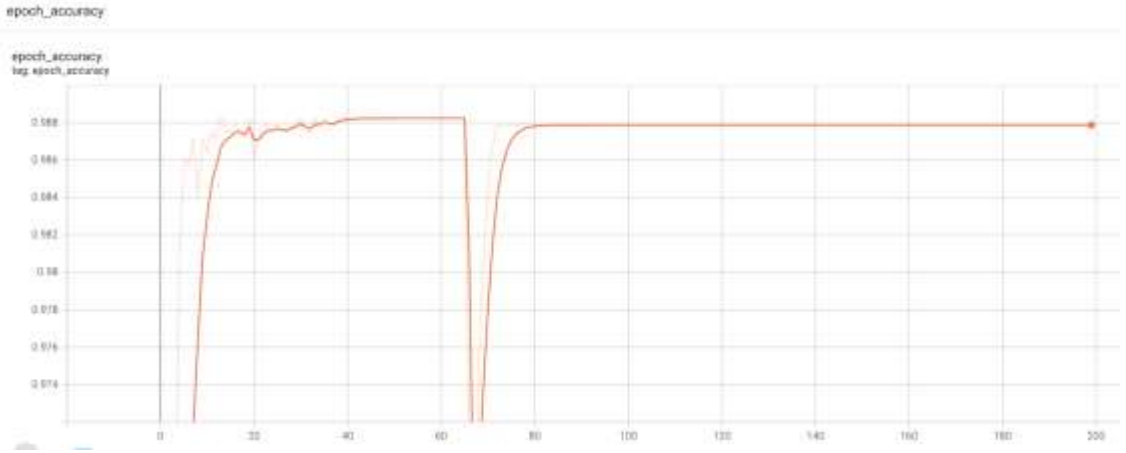
자전거 이용자의 뒷모습이나 앞모습은 킥보드 이용자와 유사하게 학습된다

킥보드 이용자와 유사하게 인식되는 자전거 이용자

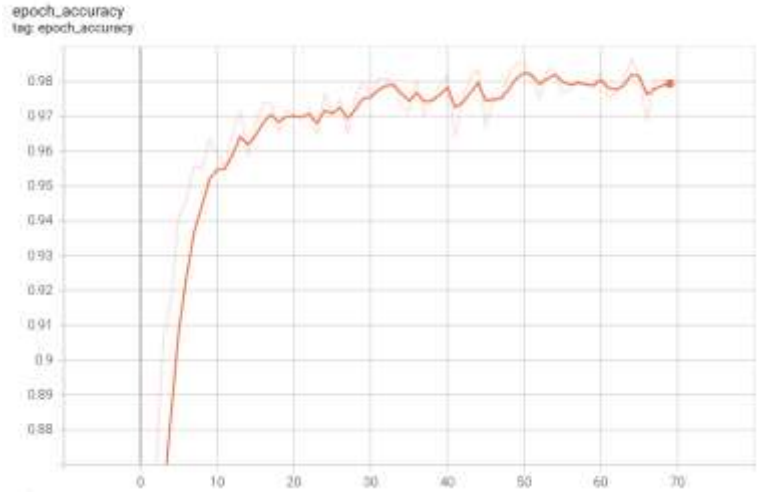
대신 자전거의 **옆모습**은 킥보드로 인식하지 않는다

Step 03

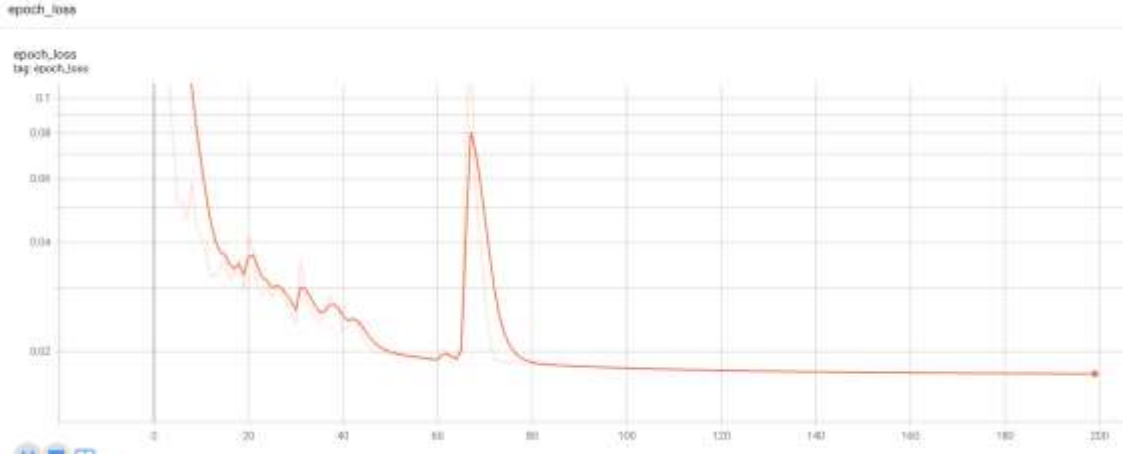
프로젝트 결과 (ResNet-152 , MobileNetV3)



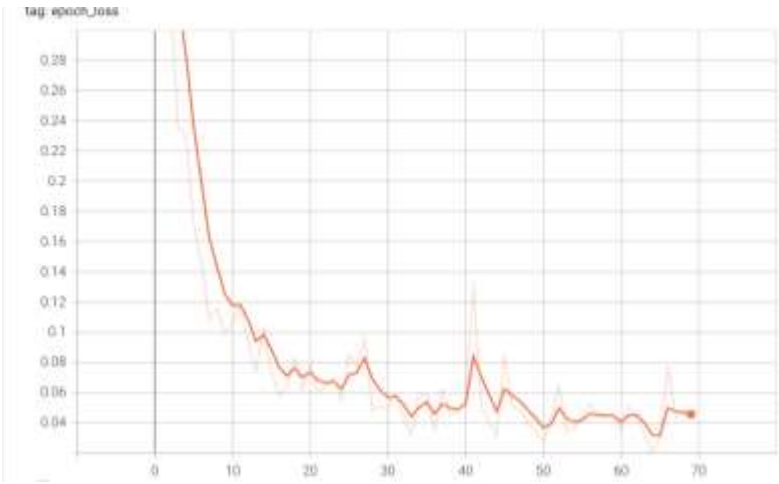
ResNet-152 Epoch_ACC



MobileNetV3 Epoch_ACC



ResNet-152 Epoch_loss



MobileNetV3 Epoch_loss

Step 03

프로젝트 결과 (ResNet-152)

1. 36 개의 테스트 데이터를 통해 결과 확인
2. 실제 데이터 VS 예측 데이터
3. 예측 데이터가 몇 %의 신뢰를 갖고 분류했는가 ?
4. 예측 실패한 데이터 이미지 출력

ResNet-152 결과 ::
36개의 테스트 중 6 번 예측실패
Accuracy : 83 %

예측 성공했을 때의 신뢰도는 최저 60% 이며, 이외에 98% 이상의 신뢰를 갖고 분류한다 !

예측 실패했을 때의 특이점 우측 이미지처럼 이용자가 마스크와 같은, 얼굴이 가려졌을 때 분류를 실패

```
True Label: Helmet, Predicted Label: NoHelmet
Helmet: 38.75%
NoHelmet: 61.25%
```

예측 실패한 이미지 출력

Step 03

프로젝트 결과 (MobileNetV3)

1. 36 개의 테스트 데이터를 통해 결과 확인
2. 실제 데이터 VS 예측 데이터
3. 예측 데이터가 몇 %의 신뢰를 갖고 분류했는가 ?
4. 예측 실패한 데이터 이미지 출력

MobileNetV3 결과 ::
36개의 테스트 중 9 번 예측실패
Accuracy : 75 %

예측 성공했을 때의 신뢰도는 최저 65% 이며, 이외에 98% 이상의 신뢰를 갖고 분류한다 !

예측 실패했을 때의 특이점은 우측 이미지처럼 이용자가
- 선글라스 끼고있거나
- 2인 이상 탑승하거나

```
1/1 [==============================] - 0s 23ms/step
True Label: NoHelmet, Predicted Label: Helmet
Helmet: 97.78%
NoHelmet: 2.22%
예측 실패한 이미지 출력
```

프로젝트 결론 1

무슨 모델 써야될까 ?
- 사진촬영 후 사후처리에 달려있다

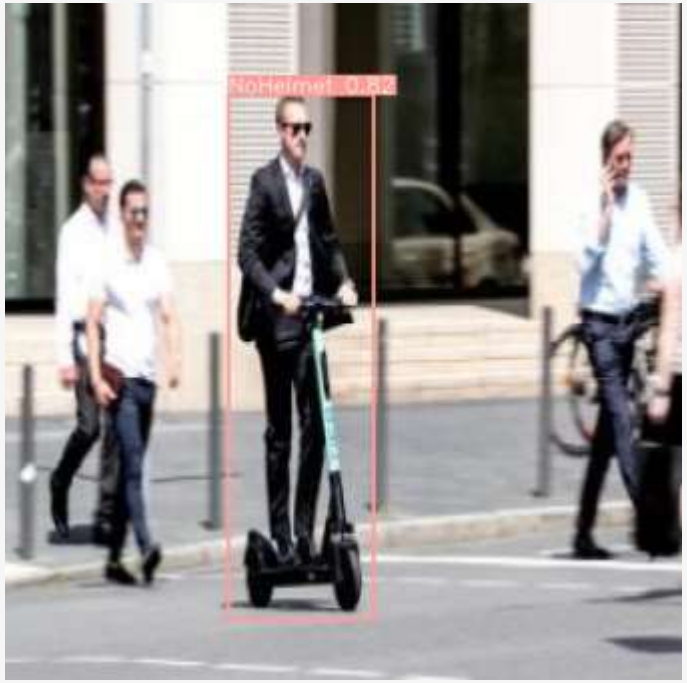
YOLOv8 :: 실시간 객체탐지 가능
ResNet-125 :: 보다 정확하지만 느림
MobileNetV3 :: 메모리 절약할 수 있지만 정확도가 비교적 낮음

행정과 연결될 가능성 때문에 모델성능이 준수하고 실시간 객체탐지까지 가능한
YOLOv8 모델이 느린 ResNet-125, 멍청한 MobileNetV3 보다는 적합하다

Step 03

프로젝트 결론 2

행인은 인식하지 않은 상태로, 헬멧을 쓰지 않는 킥보드 이용자를 AI를 활용하여 찾을 수 있게 된다.



이 AI 모델을 활용한 시스템이 사용된다면 국내 헬멧 미착용 이용자의 수를 줄일 수 있게 된다

Step 03

프로젝트 기대효과 - 경찰 전용 SW

경찰이 매번 현장에 출동하지 않고 도로에 카메라를 설치하여 헬멧을 쓰지 않은 이용자를 자동으로 검거한다.

01: 헬멧 미착용 이용자 촬영

02: 카메라 현재 위치와 시간, 사진을 공유킥보드 회사에 전송

03: 공유킥보드 회사는 당시 사용자의 정보를 경찰에 전송

04: 경찰은 전송 받은 사용자의 정보를 통해 벌금 부과

Step 03

프로젝트 기대효과 - 신고 어플리케이션

일반 시민이 헬멧 미착용 이용자를 어플로 촬영하여 검거 시 신고 포상금 수령

01: 지나가던 시민이 해당 어플로 헬멧 미착용 이용자 촬영

02: 카메라 현재 위치와 시간, 사진을 공유킥보드 회사에 전송

03: 공유킥보드 회사는 당시 사용자의 정보를 경찰에 전송

04: 경찰은 전송 받은 사용자의 정보를 통해 벌금 부과

05: 시민은 신고 포상금 수령

감 사 합 니 다