




Project 4

전동킥보드 사업에서의
Object Detection 활용방안

AI_01_장형준

- 
- 전동킥보드의 경우, 반납시 사진을 찍어서 반납하는데 이 때 제대로 주차해놨는지 확인하는 방법?
 - 헬멧을 썼는지 확인하는 방법?

< 반납 이미지 체크에서 예상되는 흐름 >

1. 사진 이미지를 통한 킥보드 검출 + 우리 회사의 킥보드가 맞는지 확인
2. 이미지 속의 킥보드 위치 판별 + 킥보드가 누여있지 않은지 확인
3. 도로 형태 구분 (인도, 차도, 비포장도로 등)
4. 사람 또는 차량의 통행을 방해할 가능성 측정을 통한

< 헬멧 썼는지 체크하는 방법 >

1. 사용자가 헬멧을 쓴 이미지를 업로드해서 검사하는 방식

Idea 1 - 킥보드 사진이 맞게 업로드 되었는지?

이것을 테스트하는데 있어 충분한 데이터를 구하지 못하여, 비슷한 다른 테스트로 변경

나름 비슷한 외관을 가진 자전거와 오토바이의 사진을 분류해낼 수 있다면, 우리회사의 킥보드 사진또한 작 후련시켜서 분류하는것이 가능할 것





자전거 vs 오토바이 테스트

이번 분류에 사용한 **MobilNet**은 적은 연산량과, 모델의 사이즈가 상대적으로 작아, 모바일 디바이스와 같은 제한된 환경에서도 사용하기 적합함

기존에 학습된 **MobileNetV2** 를 전이학습하여, 최종 분류에서 자전거인지, 오토바이인지만 분류하도록 활용함으로써, 새로운 학습에 대한 시간적 손실을 줄일 수 있음.

활용된 데이터 개수: 552개

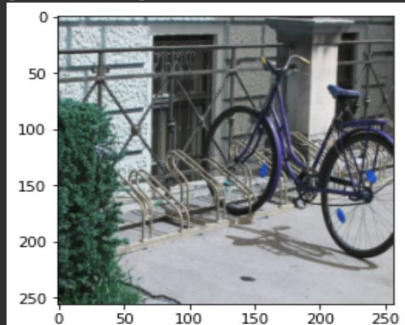
Bike = 1 Motorcycle = 0

정확도 98%

loss: 0.1660 - accuracy: 0.9909 - val_loss: 0.1718 - val_accuracy: 0.9820

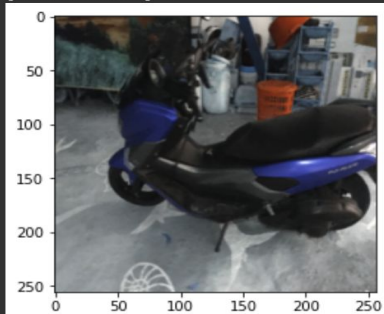
4 check_for_pred(pred, 7)

[0.8789812] 1



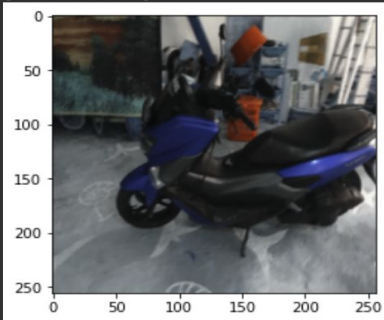
1 check_for_pred(pred, 15)

[0.11020851] 0



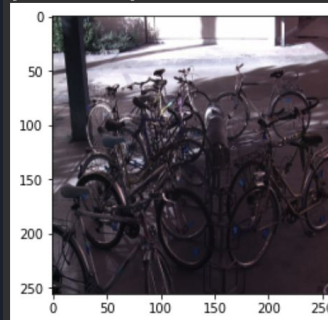
1 check_for_pred(pred, 27)

[0.08744252] 0



1 check_for_pred(pred, 37)

[0.82025254] 1



우리 회사의 전동킥보드 이미지를 학습시킨다면, 사용자가 반납시 업로드한 사진을 제대로 분류할 수 있을것으로 기대됨

Idea 2 - 제대로 주차 되었는지 확인

초기의 아이디어처럼 누여있는지, 도로위에서의 위치가 어디인지, 차도인지 인도인지 등의 분류를 하기위해 학습시킬 데이터셋을 마련하지 못하였으며, 새로운 학습을 시킬 수 없는 상황이므로, 이중 누여있는지에 대한 확인 방법의 테스트를 진행함





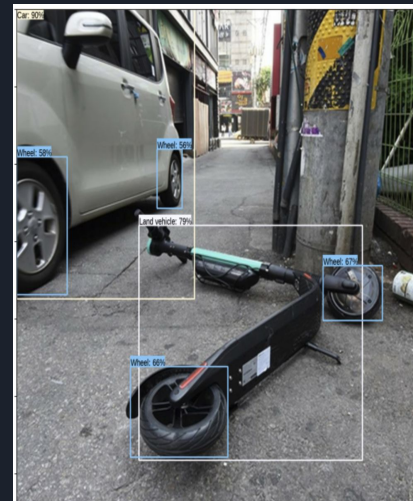
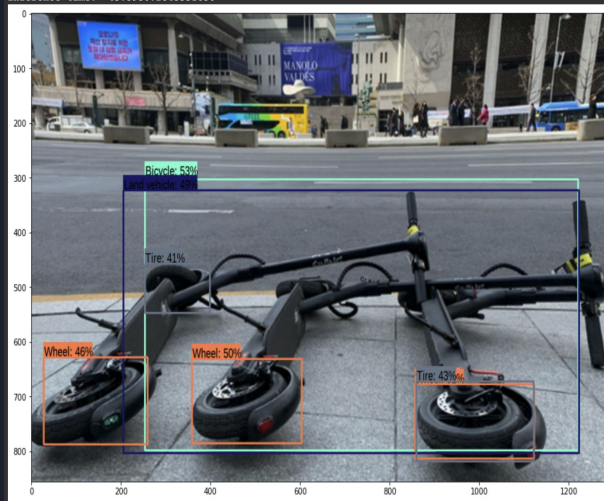
어떻게 뉘여있는지 판단할 것인가

기존의 **Object Detection**에서 모델의 **Output**은 사진속 검출된 물체의 이름(**Class**)과 각각의 위치좌표가 주어짐

이렇게 주어지는 물체 **Class**와 위치좌표를 사용하여 뉘여져있는지 테스트를 진행

기존에 잘 학습되어있는 **Inception_resnet_v2**를 사용하여, **Object Detection**된 결과를 해석하는 방식으로 진행

Inference time: 45.393072843551636



이렇게 검출된 이미지의 output을 통하여, “누어져있을 경우, 가로(Width)의 길이가 세로(Height)의 길이보다 길지 않을까?”

라는 아이디어로 시작하여, 실제 계산을 해본 결과

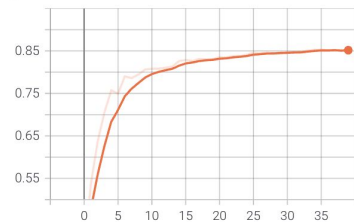
몇몇의 이미지에서는 쓰러져있는지 여부가 잘 검출되는가 싶었지만, 세번째 이미지처럼, 누여있음에도 카메라의 각도에 따라 킥보드가 감지된 박스 영역의 크기가 제각각이라는 점에서, 위 아이디어는 사용 불가하다고 판단

Idea 3 - 헬멧을 쓴 사진을 제출하여 이를 통해 헬멧을 썼다고 판단하자

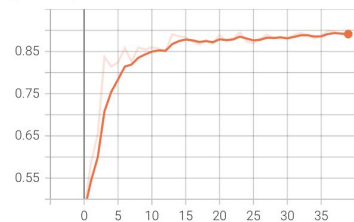
3712개의 이미지를 Yolo v5 모델에 학습시켰으며, 헬멧을 쓰지않은 사람과, 헬멧을 쓴 사람의 라벨을 분류하여 40번 학습시킨 결과

30/39	1.34G	0.03502	0.02649	0.00121	0.06272	207	416:	100%	232/232	[02:35<00:00, 1.49it/s]
Class	Images	Labels	P	R			mAP@0.5	mAP@0.5:95:	100%	30/30 [00:13<00:00, 2.18it/s]
all	929	7691	0.878	0.799	0.846	0.472				
Epoch	gpu_mem	box	obj	cls	total	labels	img_size			
31/39	1.34G	0.03504	0.02622	0.001078	0.06235	246	416:	100%	232/232	[02:31<00:00, 1.53it/s]
Class	Images	Labels	P	R			mAP@0.5	mAP@0.5:95:	100%	30/30 [00:12<00:00, 2.36it/s]
all	929	7691	0.889	0.791	0.847	0.472				
Epoch	gpu_mem	box	obj	cls	total	labels	img_size			
32/39	1.34G	0.03497	0.02659	0.001071	0.06263	244	416:	100%	232/232	[02:36<00:00, 1.48it/s]
Class	Images	Labels	P	R			mAP@0.5	mAP@0.5:95:	100%	30/30 [00:12<00:00, 2.33it/s]
all	929	7691	0.894	0.799	0.847	0.472				
Epoch	gpu_mem	box	obj	cls	total	labels	img_size			
33/39	1.34G	0.03473	0.02626	0.001101	0.06209	325	416:	100%	232/232	[02:32<00:00, 1.52it/s]
Class	Images	Labels	P	R			mAP@0.5	mAP@0.5:95:	100%	30/30 [00:13<00:00, 2.27it/s]
all	929	7691	0.889	0.802	0.851	0.481				
Epoch	gpu_mem	box	obj	cls	total	labels	img_size			
34/39	1.34G	0.03489	0.02586	0.001078	0.06193	156	416:	100%	232/232	[02:34<00:00, 1.50it/s]
Class	Images	Labels	P	R			mAP@0.5	mAP@0.5:95:	100%	30/30 [00:12<00:00, 2.31it/s]
all	929	7691	0.881	0.81	0.852	0.482				
Epoch	gpu_mem	box	obj	cls	total	labels	img_size			
35/39	1.34G	0.03418	0.02579	0.0008756	0.06084	255	416:	100%	232/232	[02:35<00:00, 1.49it/s]
Class	Images	Labels	P	R			mAP@0.5	mAP@0.5:95:	100%	30/30 [00:13<00:00, 2.23it/s]
all	929	7691	0.886	0.805	0.853	0.48				
Epoch	gpu_mem	box	obj	cls	total	labels	img_size			
36/39	1.34G	0.0342	0.02602	0.0009792	0.0612	252	416:	100%	232/232	[02:34<00:00, 1.50it/s]
Class	Images	Labels	P	R			mAP@0.5	mAP@0.5:95:	100%	30/30 [00:12<00:00, 2.32it/s]
all	929	7691	0.899	0.798	0.851	0.481				
Epoch	gpu_mem	box	obj	cls	total	labels	img_size			
37/39	1.34G	0.03456	0.02625	0.001008	0.06182	157	416:	100%	232/232	[02:34<00:00, 1.50it/s]
Class	Images	Labels	P	R			mAP@0.5	mAP@0.5:95:	100%	30/30 [00:13<00:00, 2.23it/s]
all	929	7691	0.897	0.799	0.852	0.479				
Epoch	gpu_mem	box	obj	cls	total	labels	img_size			
38/39	1.34G	0.03481	0.02749	0.0009192	0.06222	227	416:	55%	128/232	[01:25<00:56, 1.86it/s]

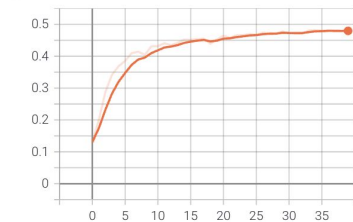
mAP_0.5
tag: metrics/mAP_0.5



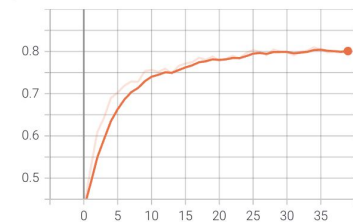
precision
tag: metrics/precision



mAP_0.5:0.95
tag: metrics/mAP_0.5:0.95



recall
tag: metrics/recall



학습에 따른 Detection 결과



헬멧이 아닌 일반 모자(후드)를 뒤집어 쓴 이미지 또한, 헬멧을 쓰지 않았다고 분류할 수 있을만큼 높은 성능을 보여줌

충분한 데이터와 라벨링 작업에 대한 시간만 주어진다면, 다른 Object에 대해서도 학습이 가능할 것이며,

이를 통해, 향후 전동킥보드의 파손여부 또한 학습을 통해 검출할 수 있을 것으로 생각합니다.



Reference

<https://github.com/ultralytics/yolov5>

<https://www.kaggle.com/brendan45774/bike-helmets-detection>

https://tfhub.dev/google/faster_rcnn/openimages_v4/inception_resnet_v2/1

<https://www.hankookilbo.com/News/Read/201912261654784606>



감사합니다