

합생데이터를 활용한 자동차 탐지 AI 모델 개발



이게 왜 되지? 팀 (딥러닝 프로젝트 1팀)

팀장
이혜원

이선정

김민영

목차

1 대회 설명

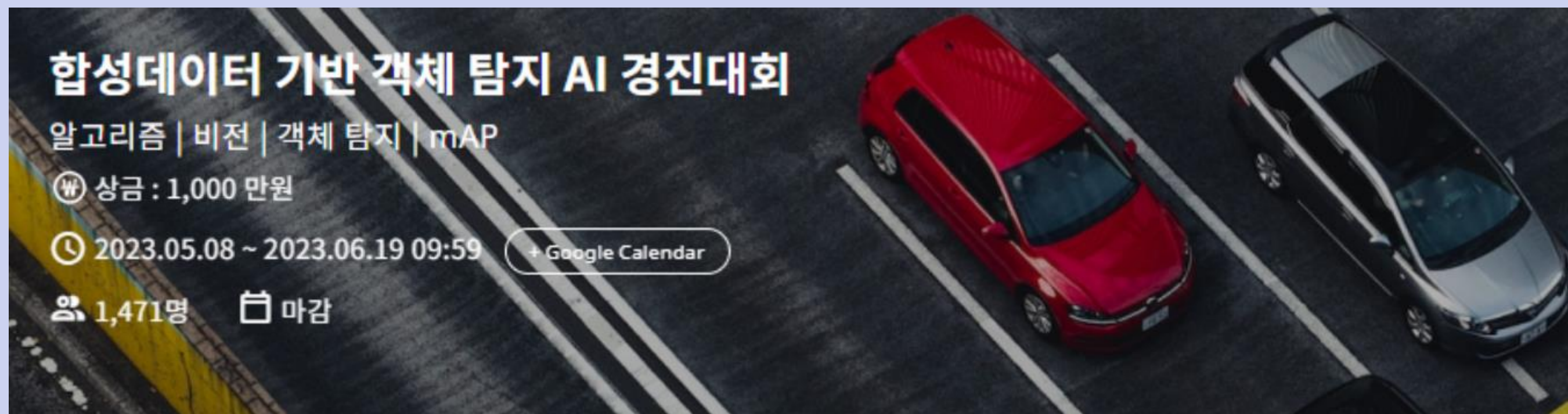
4 모델 학습

2 모델 선정

5 결과

3 데이터 전처리

1 대회 설명



[주제]

합성 데이터를 활용한 자동차 탐지 AI 모델 개발

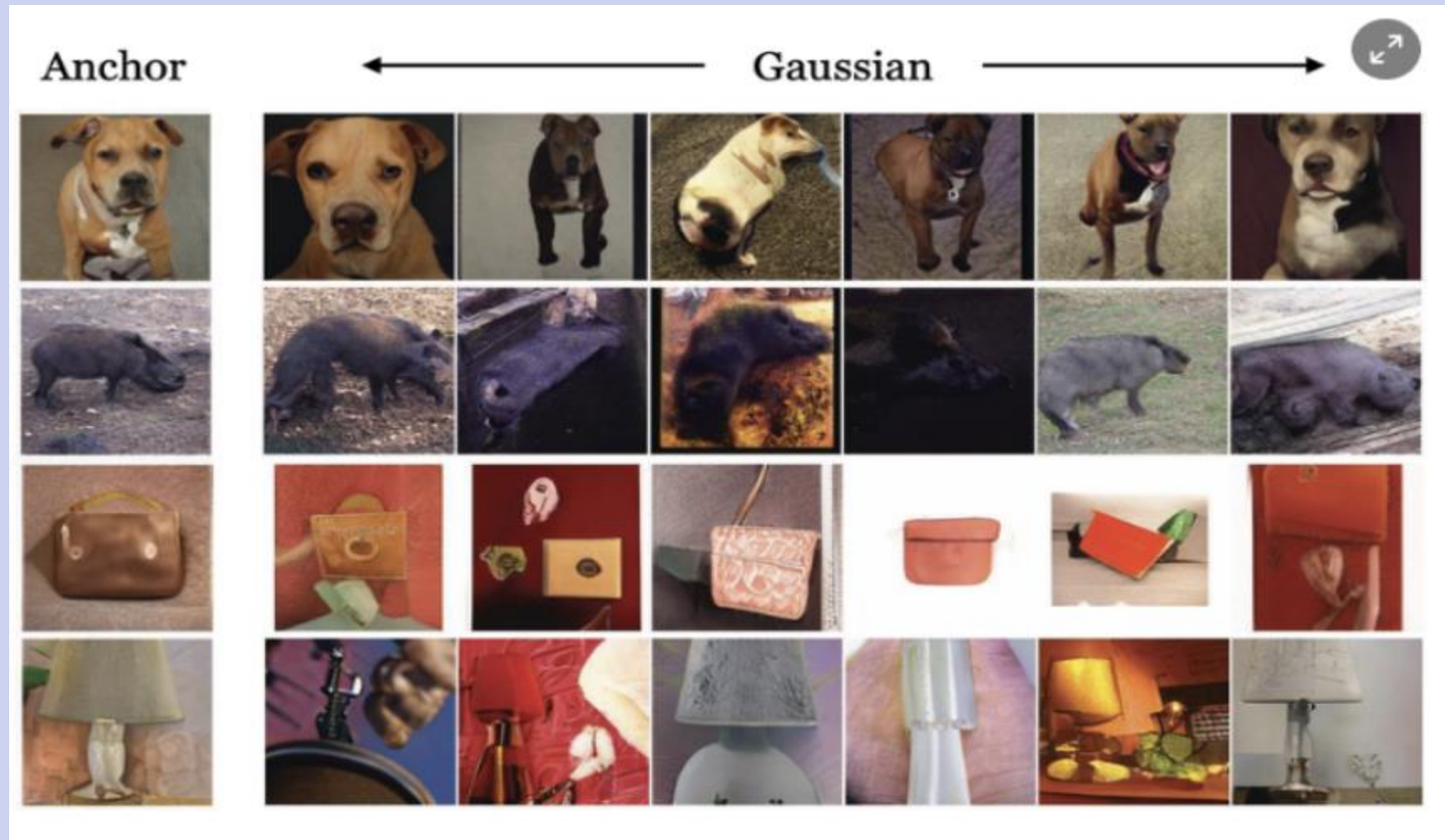
[설명]

학습용 합성데이터를 활용하여 자동차 탐지를 수행하는 AI 모델을 개발
자동차 탐지 & 34가지의 자동차 세부모델까지 판별

1 대회 설명

합성데이터

- 합성데이터
 - 컴퓨터 알고리즘에 의해 생성되는 데이터로 실제 데이터 세트의 통계 패턴을 모방하여 인공적으로 만들어진 데이터
 - 발생할 수 있는 모든 시나리오나 흔히 발생하지 않는 상황에 대한 데이터를 생성, 학습에 활용함
- 합성데이터 특징
 - real world 이미지로 pretrained된 weight를 사용하여 finetuning 하는 것이 효과적임
 - 합성데이터는 real world 이미지에 비해 노이즈가 작음
 - 다양한 augmentation이 효과적임



1 대회 설명

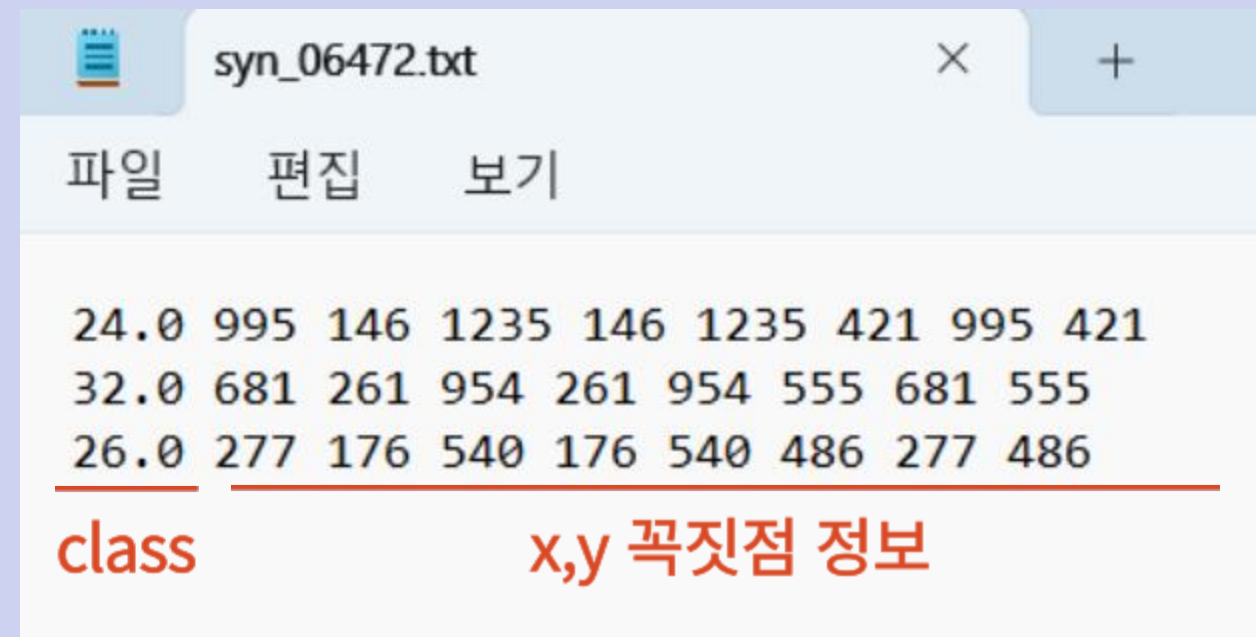
데이터 설명

[데이터]

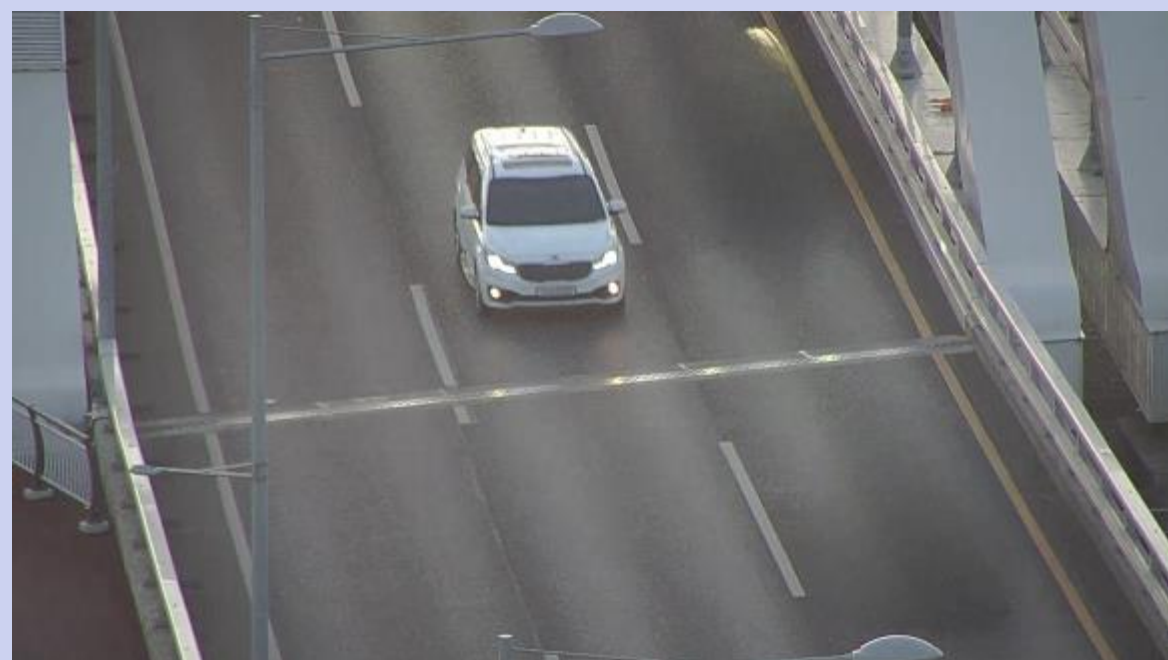
Train - 6480개의 합성 이미지 파일, 정보 txt 파일 → class 별 갯수는 500개로 균형함

Test - 3400개의 평가 이미지 파일

- train 이미지, txt 파일



- test 평가 이미지



2 모델 선정

Yolov5


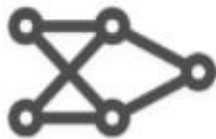

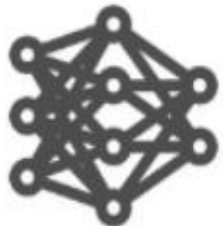
- Yolov5를 베이스 모델로 잡고 이후에 정확도를 높이는 방향으로 진행

1. 높은 탐지의 정확도

2. 경량화 특성

3. 빠른 속도로 탐지

실제로 활용시 카메라로 촬영한 자동차의 영상이나 이미지를 실시간으로 판별해 내야 하기 때문에 동영상이나 이미지에서 객체를 빠르게 인식할 수 있는 yolov5 를 사용함

			
Small YOLOv5s	Medium YOLOv5m	Large YOLOv5l	XLarge YOLOv5x
14 MB _{FP16} 2.2 ms _{V100} 36.8 mAP _{COCO}	41 MB _{FP16} 2.9 ms _{V100} 44.5 mAP _{COCO}	90 MB _{FP16} 3.8 ms _{V100} 48.1 mAP _{COCO}	168 MB _{FP16} 6.0 ms _{V100} 50.1 mAP _{COCO}

Backbone 종류

- yolo v5는 s, m, l, x의 4가지 버전이 있음
- s가 가장 가벼운 모델
- x가 가장 무거운 모델
- s가 성능이 제일 낮지만 FPS가 가장 높음
- x가 성능이 제일 높지만 FPS는 가장 낮음

2 모델 선정

<Yolov5 모델 선정>

- 배치 사이즈 64
- 에폭 20

동일한 조건에서 Yolov5s와 Yolov5m 성능을 비교

Yolov5s

2023-06-28 17:18:27	0.2381382225	□
	0.2287507276	

LB : 0.2381382225 (public)
0.2287507276 (private)

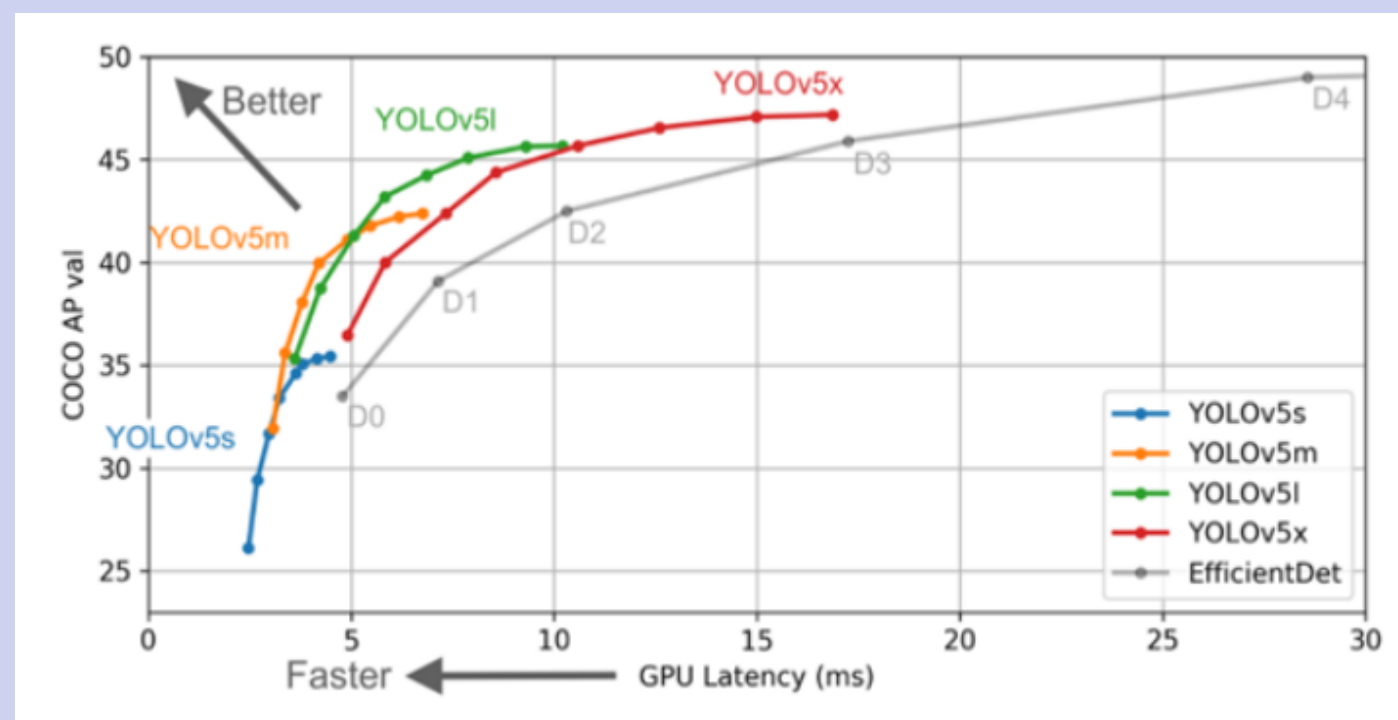
Yolov5m

2023-06-30 05:38:40	0.3173657043	□
	0.3176007773	

LB : 0.3173657043 (public)
0.3176007773 (private)

Yolov5m 모델의 정확도가 더 높고, Yolov5s는 시간 당 프레임 처리 속도가 빠름

▶ 정확도가 더 높은 Yolov5m을 사용해서 베이스 라인 진행



3 데이터 전처리

데이터 라벨링

Yolo_labelMe

```
def yolo_labelMe(line, img_width, img_height, txt_file_name):
    line = line.split(' ')
    file_name = txt_file_name.split('/')[-1].split('.')[0] + '.png'
    class_id = line[0]
    x_center = float(line[1])
    y_center = float(line[2])
    width = float(line[3])
    height = float(line[4])
    confidence = float(line[5])

    point1_x = x_center - (width / 2) # box의 왼쪽 위 꼭지점
    point1_y = y_center - (height / 2) # box의 왼쪽 위 꼭지점
    point2_x = x_center + (width / 2) # box의 오른쪽 위 꼭지점
    point2_y = y_center - (height / 2) # box의 오른쪽 위 꼭지점
    point3_x = x_center + (width / 2) # box의 오른쪽 아래 꼭지점
    point3_y = y_center + (height / 2) # box의 오른쪽 아래 꼭지점
    point4_x = x_center - (width / 2) # box의 왼쪽 아래 꼭지점
    point4_y = y_center + (height / 2) # box의 왼쪽 아래 꼭지점

    point1_x = int(point1_x * img_width)
    point1_y = int(point1_y * img_height)
    point2_x = int(point2_x * img_width)
    point2_y = int(point2_y * img_height)
    point3_x = int(point3_x * img_width)
    point3_y = int(point3_y * img_height)
    point4_x = int(point4_x * img_width)
    point4_y = int(point4_y * img_height)

    return file_name, class_id, confidence, point1_x, point1_y, point2_x, point2_y, point3_x, point3_y, point4_x, point4_y
```

▶ 텍스트 파일에 있는 좌표를 YOLOv5 형식으로 변화해줌

3 데이터 전처리

Grayscale

Grayscale

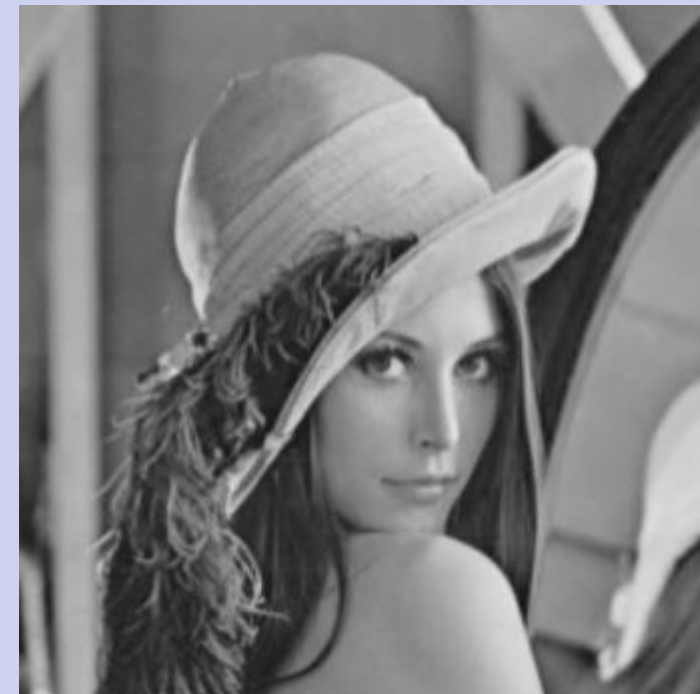
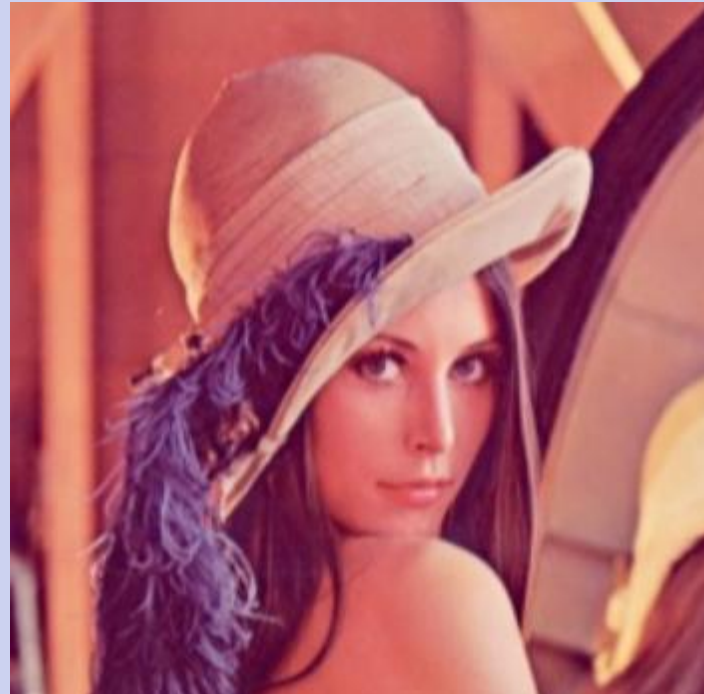
1. 계산량 감소

RGB는 3차원의 색공간 $255 \times 255 \times 255$

Grayscale은 1차원의 색공간 0~255 → 연산량 대폭 감소

2. 명암의 차이

명암의 차이에 집중하기 때문에 이미지에서의 구조와 패턴을 더 쉽게 인식할 수 있음
컬러 정보에 영향을 받지 않아 색감에 따른 차이를 줄일 수 있음



3 데이터 전처리

Grayscale

- train 파일 grayscale 처리

```
for f in tqdm(txt_file_list):  
    file_name = os.path.basename(f)  
    file_name = file_name.split('.')[0]  
    img=Image.open('/content/detect_car/train/' + file_name + '.png').convert('L')  
    img_numpy = np.array(img, 'uint8')  
    cv2.imwrite("/content/detect_car/train/" + file_name + ".png", img_numpy)
```

- test 파일 grayscale 처리

```
path = '/content/detect_car/test/'  
imagePaths = [os.path.join(path,file_name) for file_name in os.listdir(path)]  
for imagePath in imagePaths:  
    img = Image.open(imagePath).convert('L')  
    img_numpy = np.array(img, 'uint8')  
    cv2.imwrite(imagePath, img_numpy)
```



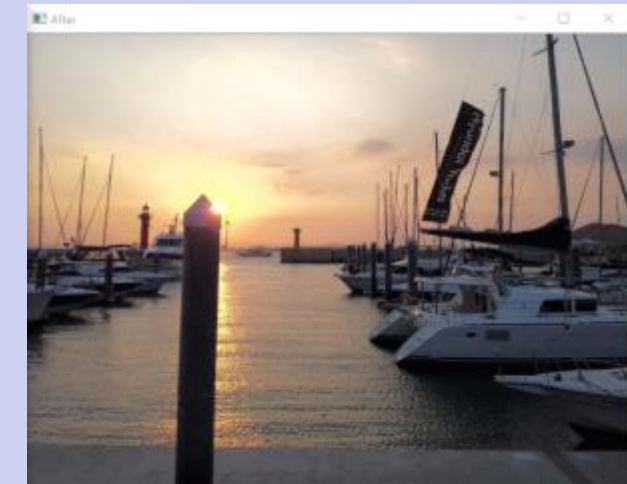
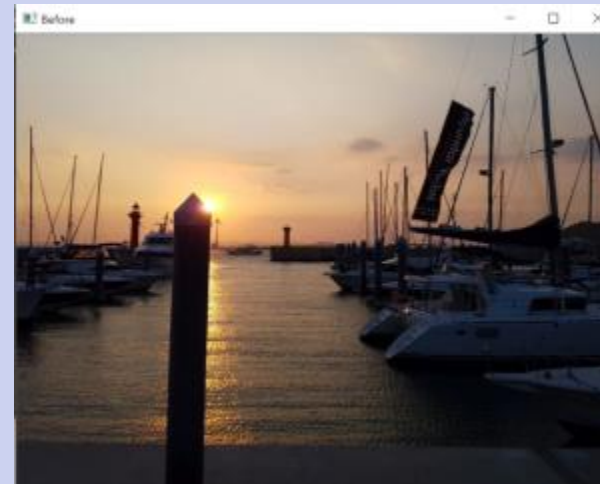
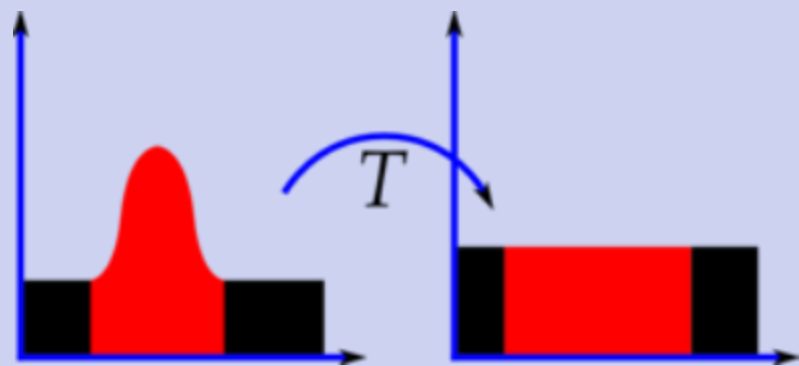
3 데이터 전처리

Clahe

Clahe

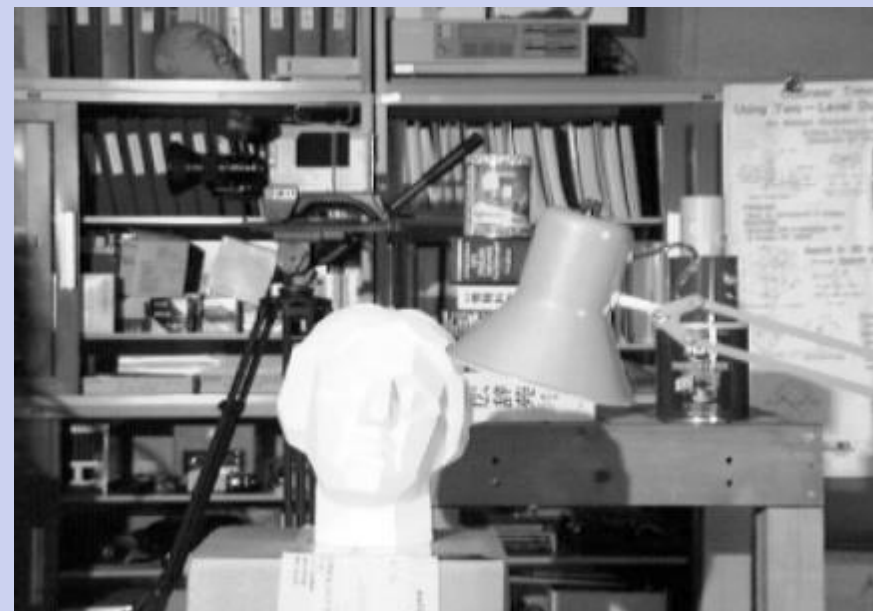
1. Equalization(평탄화)의 문제점 개선

평탄화 - 특정 영역에 집중되어 있는 분포를 골고루 분포하여 명암 대비를 개선함



이미지의 밝은 부분이 날아가는 현상이 발생함

▷ 이미지를 일정한 영역으로 나누어 평탄화를 적용함



평탄화 적용



clahe 적용

3 데이터 전처리

Clahe

- train 파일 clahe 처리

```
for f in tqdm(txt_file_list):  
    file_name = os.path.basename(f)  
    file_name = file_name.split('.')[0]  
    img=cv2.imread('/content/detect_car/train/' + file_name + '.png', cv2.IMREAD_GRAYSCALE)  
    clahe = cv2.createCLAHE(clipLimit=2.0, tileGridSize=(8,8))  
    img2 = clahe.apply(img)  
    cv2.imwrite("/content/detect_car/train/" + file_name + ".png", img2)
```

- test 파일 clahe 처리

```
path = '/content/detect_car/test/'  
imagePaths = [os.path.join(path,file_name) for file_name in os.listdir(path)]  
for imagePath in imagePaths:  
    img=cv2.imread(imagePath, cv2.IMREAD_GRAYSCALE)  
    clahe = cv2.createCLAHE(clipLimit=2.0, tileGridSize=(8,8))  
    img2 = clahe.apply(img)  
    cv2.imwrite(imagePath, img2)
```



4 모델 학습

하이퍼파라미터 조정

image size = 736 (image size 키울수록 성능 증가)

batch size = 32 (batch size 키울수록 성능 증가)

epoch = 기본 50 -> 80

- hyp.yaml~ 파일을 통해 하이퍼파라미터를 조정함

```
6 lr0: 0.01 # initial learning rate (SGD=1E-2, Adam=1E-3)
7 lrf: 0.1 # final OneCycleLR learning rate (lr0 * lrf)
8 momentum: 0.937 # SGD momentum/Adam beta1
9 weight_decay: 0.0005 # optimizer weight decay 5e-4
10 warmup_epochs: 3.0 # warmup epochs (fractions ok)
11 warmup_momentum: 0.8 # warmup initial momentum
12 warmup_bias_lr: 0.1 # warmup initial bias lr
13 box: 0.05 # box loss gain
14 cls: 0.3 # cls loss gain
15 cls_pw: 1.0 # cls BCELoss positive_weight
16 obj: 0.7 # obj loss gain (scale with pixels)
17 obj_pw: 1.0 # obj BCELoss positive_weight
18 iou_t: 0.50 # IoU training threshold
19 anchor_t: 5.0 # anchor-multiple threshold
20 # anchors: 3 # anchors per output layer (0 to ignore)
21 fl_gamma: 0.0 # focal loss gamma (efficientDet default gamma=1.5)
22 hsv_h: 0.015 # image HSV-Hue augmentation (fraction)
23 hsv_s: 0.7 # image HSV-Saturation augmentation (fraction)
24 hsv_v: 0.4 # image HSV-Value augmentation (fraction)
25 degrees: 0.5 # image rotation (+/- deg)
26 translate: 0.1 # image translation (+/- fraction)
27 scale: 0.9 # image scale (+/- gain)
28 shear: 0.0 # image shear (+/- deg)
29 perspective: 0.0 # image perspective (+/- fraction), range 0-0.001
30 flipud: 0.5 # image flip up-down (probability)
31 fliplr: 0.5 # image flip left-right (probability)
32 mosaic: 1.0 # image mosaic (probability)
33 mixup: 0.5 # image mixup (probability)
34 copy_paste: 0.1 # segment copy-paste (probability)
```

hyperparameter	설명
degrees	이미지 회전, 사진마다, epoch마다 랜덤으로 주어 학습
flipud, fliplr	상하 변환
mixup	두 이미지를 사용하여 중첩시켜 ratio로 레이블 조정
iou_t	임계값
anchor_t	anchor박스의 multiple 임계값

4 모델 학습

결과

image size = 736

batch size = 32

epoch = 50

Grayscale 결과

2023-06-27	0.5267316889
06:36:33	0.5021634492



LB : 0.5267316889 (public)
0.5021634492 (private)

Clahe 결과

2023-07-01	0.5818246825
07:18:41	0.5870166353



LB : 0.5818246825 (public)
0.5870166353 (private)



grayscale 후 Clahe 적용했을 때 성능이 더 높아짐을 확인할 수 있음

4 모델 학습

Data Augmentation

- Data Augmentation(데이터 증강)은 원본 데이터를 변형하여 데이터의 다양성을 증가시키는 것을 의미함
- 더 다양한 환경에서 자동차를 탐지하고 모델의 성능을 높이고자 augmentation을 진행하였음
- augmentation은 albumentations.py 모듈을 사용함

```
31 T = [  
32     A.RandomResizedCrop(height=size, width=size, scale=(0.8, 1.0), ratio=(0.9, 1.11), p=0.0),  
33     A.Blur(p=0.1),  
34     A.MedianBlur(p=0.1),  
35     A.ToGray(p=0.01),  
36     A.CLAHE(p=0.01),  
37     A.RandomBrightnessContrast(p=0.1),  
38     A.RandomGamma(p=0.0),  
39     A.ImageCompression(quality_lower=75, p=0.0)] # transforms  
40 self.transform = A.Compose(T, bbox_params=A.BboxParams(format='yolo', label_fields=['class_labels']))  
41
```

4 모델 학습

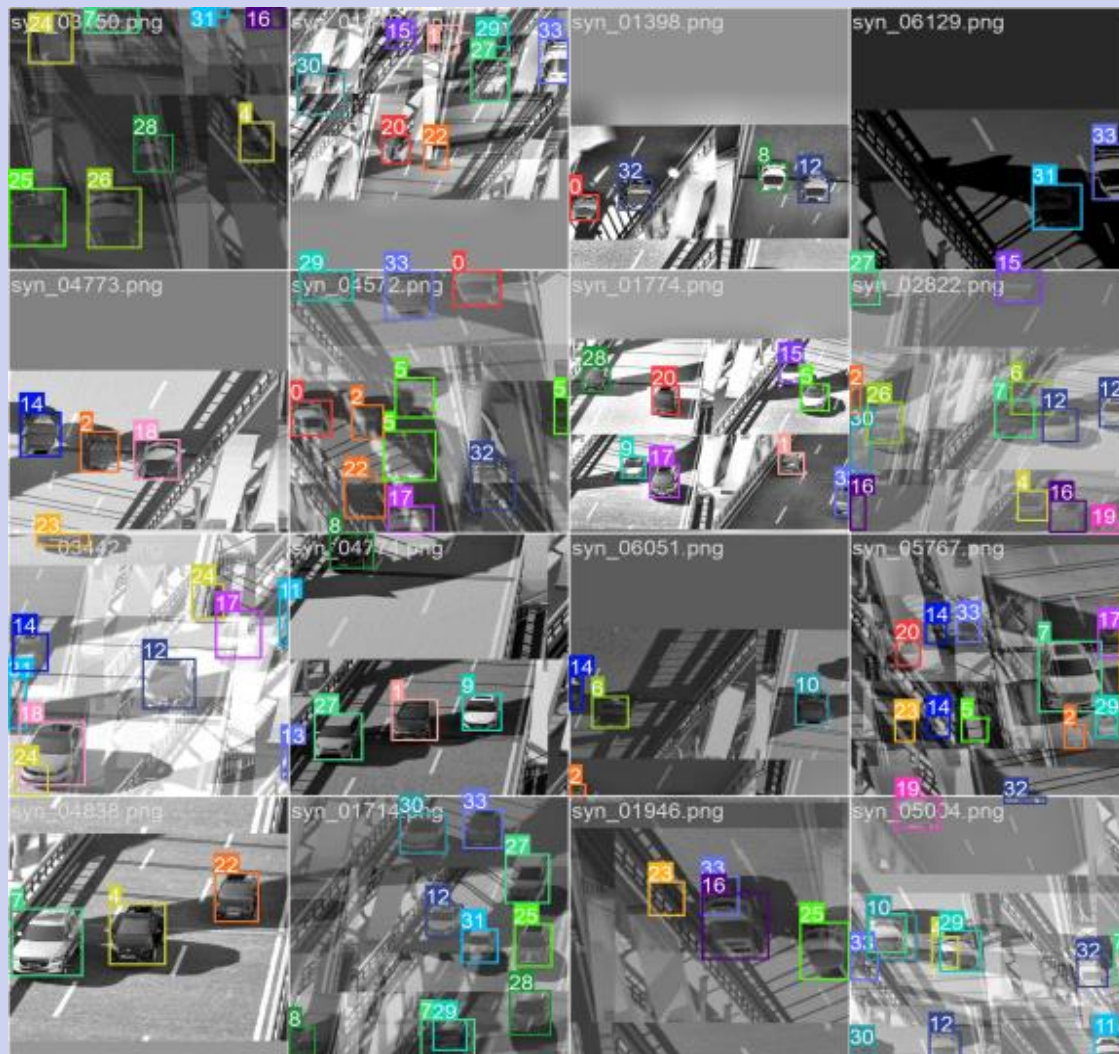
Albumentations 소개

- Albumentations은 데이터 증강을 위한 라이브러리로 이미지 회전, 이동, 밝기와 대비 조정 등 다양한 이미지 증강 기법을 제공하여 효과적인 데이터 증강을 편리하게 수행할 수 있도록 도와줌

list	p	설명	조정 이유
Blur	0.1	이미지에 블러처리를 적용함	이미지의 노이즈를 제거하기 위함
MedianBlur	0.1	이미지에 선형 크기의 중앙값 필터를 사용하여 블러처리를 적용함	이미지의 노이즈를 제거하기 위함
RandomBrightnessContrast	0.1	이미지에 임의의 밝기와 대비 변화를 적용함	실제 상황에서 기후, 시간 등으로 인한 밝기와 대조의 영향을 받을 수 있으므로 다양한 상황을 만들기 위함

4 모델 학습

Data Augmentaion 결과



➔ train batch 이미지를 확인한 결과,
증강된 이미지로 학습하는 것을
확인할 수 있음

● 증강 전

2023-07-07 23:52:39 0.6837643801
 0.6640316866

LB : 0.6837643801 (public)
0.6640316866 (private)



● 증강 후

2023-07-08 10:30:09 0.7204895463
 0.6862235612

LB : 0.7204895463 (public)
0.6862235612 (private)

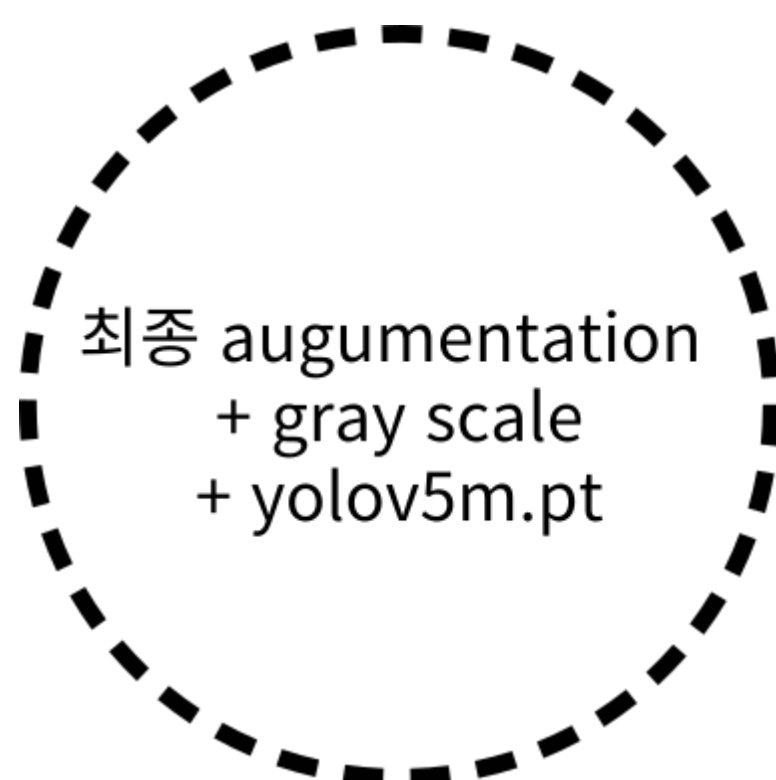
4 모델 학습

ensemble

- 다양한 모델을 사용하여 결과를 예측하기 위해 여러 다양한 모델을 만드는 프로세스
- 각 기본 모델의 예측을 집계하고 최종 예측을 한 번 더 생성함



+

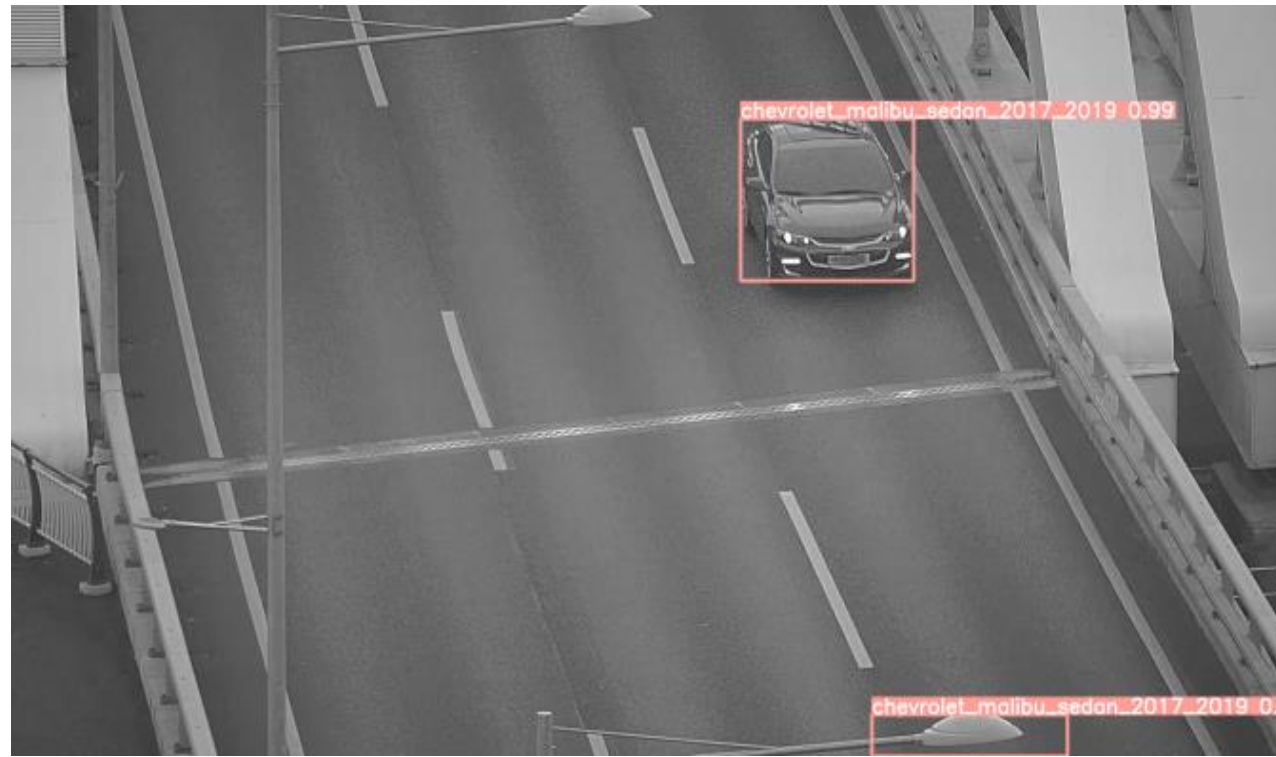


2023-07-10 0.7435127638
02:16:27 0.7198356693

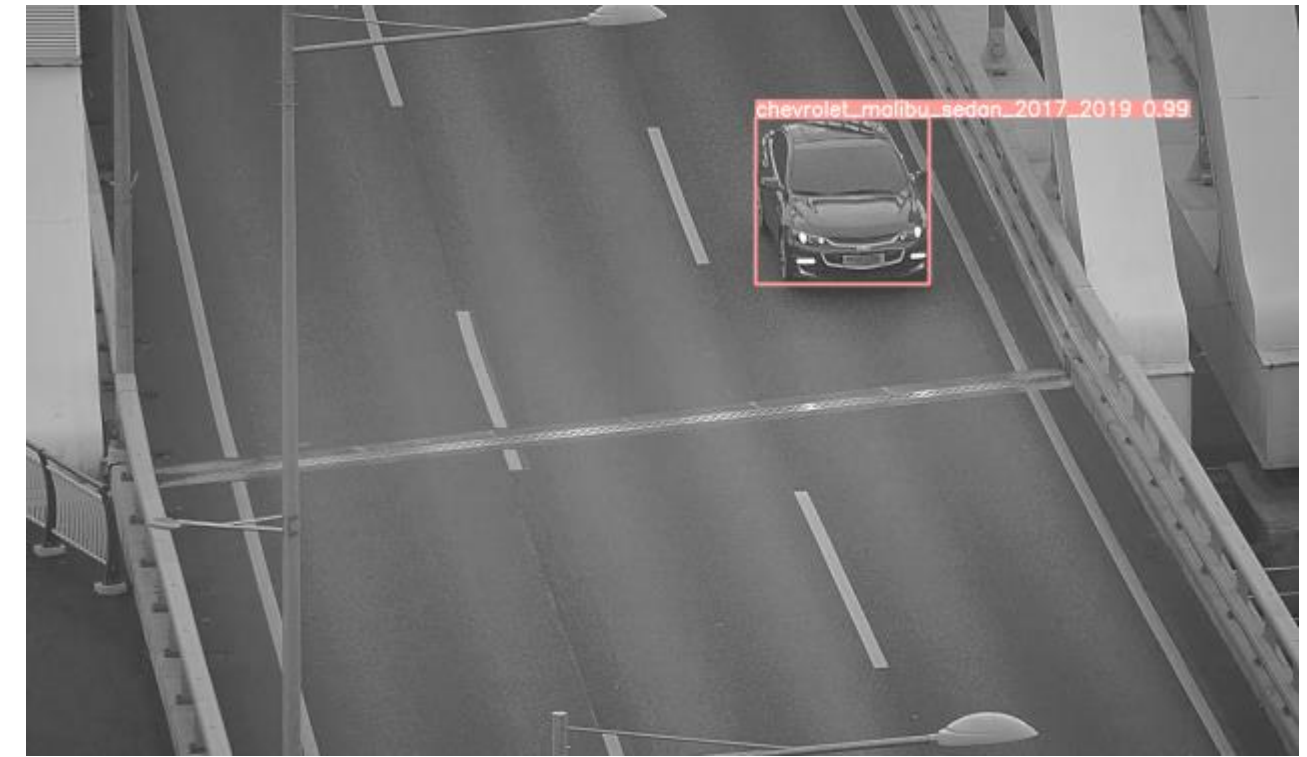
LB : 0.7435127638 (public)
0.7198356693 (private)

- 앙상블 했을 때와 성능이 가장 좋은 (yolo5m.pt) 모델에서 epochs를 10 증가한 경우를 비교했을 때 후자의 성능이 더 높게 나옴

앙상블 했을 때



가장 좋은 pt에서 epoch 수 높였을 때



예측 시 전등을 자동차로 인식하는 상황 발생

4 모델 학습

Fine Tuning

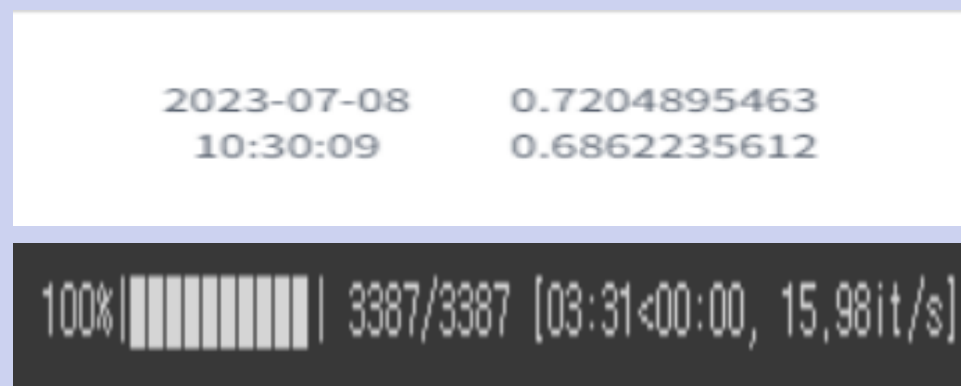
- 기존에 학습되어져 있는 모델을 기반으로 아키텍처를 새로운 목적으로 변형하고 이미 학습된 모델 weights로 부터 학습을 업데이트하는 방법
- 모델의 파라미터를 미세하게 조정함
- 다음과 같이 최종 train된 모든 weights(best.pt) 를 한번 더 train 시킴

```
1 !python train.py --img 736 #
2 --batch 32 #
3 --epochs 80 #
4 --data "/content/detect_car/data.yaml" #
5 --cfg './models/yolov5m.yaml' #
6 --weights '/content/yolov5/runs/train/yolo5s_results/weights/best.pt' #
7 --name yolov5m_results
```

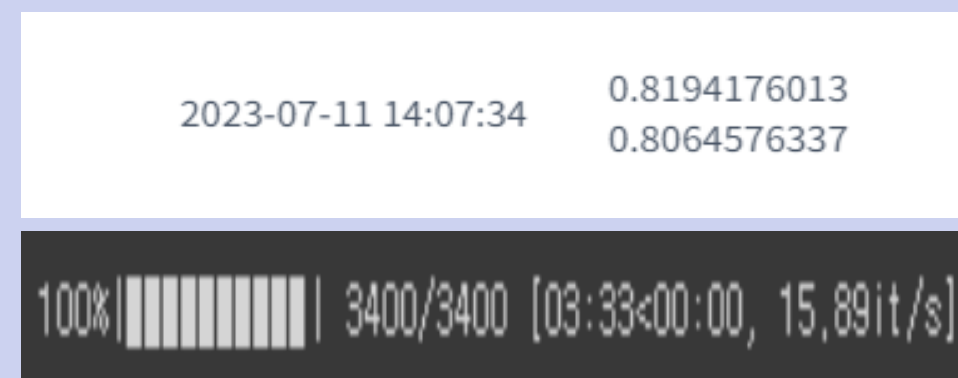

4 모델 학습

TTA

- TTA는 Test Time Augmentations의 약자로 컴퓨터 비전에서 테스트 작업 중 더 좋은 작업을 보장하기 위해 사용함
- Test set 이미지 양이 보통 부족하기 때문에 test 이미지를 transformations 함으로써 정확도를 향상시킴
- 최종 모델을 예측하는 코드 detect.py 에 --augment 를 더하여 사용함



LB : 7204895463 (public)
0.6862235612 (private)

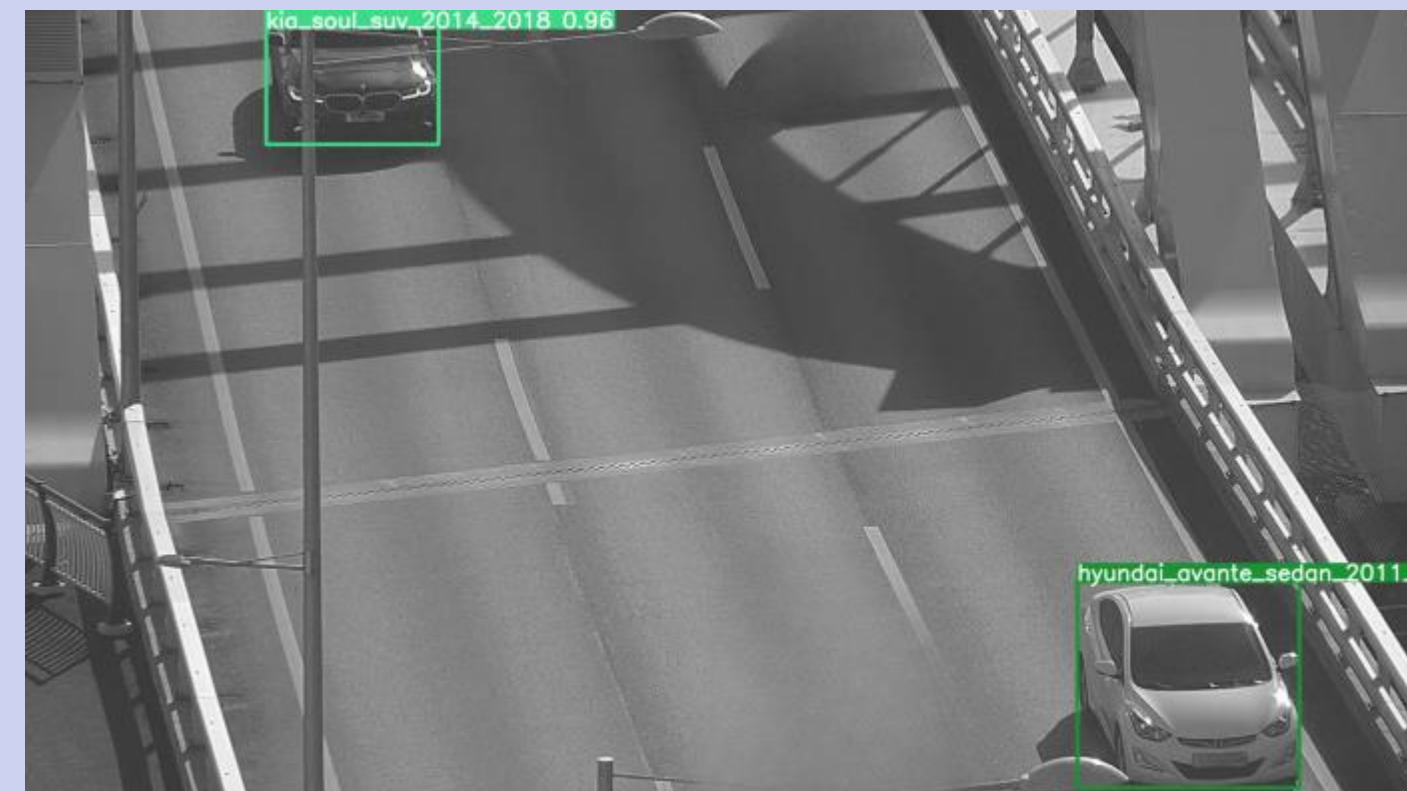


LB : 0.8194176013 (public)
0.8064576337 (private)

결과적으로 TTA를 적용시킨 전 후로 비교해 봤을 때 TTA를 한 결과가 더 높은 private score를 기록하였으며, detect 또한 test 개수의 결과와 맞게 수행함

5 결과

object detection 결과



요약

- Grayscale + Clahe 를 사용하여 데이터 전처리 후 데이터 증강
- Train을 진행시킨 후 best.pt를 사용하여 다시 fine tuning을 실행 -> test 파일을 detect 할 때 TTA를 적용
- 최종적으로 LB : 0.8194176013 (public), 0.8064576337(private) 점수를 도출함

5 결과

한계 및 개선 방향

- 한계

GPU 제한으로 인한 런타임 끊김 문제 발생 및 메모리 부족 문제 발생

-> epoch, batch size, image size 등 다양한 파라미터 등을 조정하고 싶었으나,
GPU 자원의 한계로 모델 학습에 어려움을 겪음

- 개선 방향

1. 차량 종류, 날씨 변화에 맞는 데이터 전처리 및 증강 적용 : 현실세계 문제를 반영하여
흑백처리를 넘어서 채도, 명도, 색상 등을 변경해보고자 함

2. 다양한 모델 적용 : YOLOv5에 모델만 가지고 진행했지만 추후에 YOLOv6, YOLOv8과
같은 다양한 모델도 적용해봄으로써 성능을 높임

3. 파라미터 변경 : 실험이 부족했기에 같은 epoch, batch size의 조건 하에 파라미터
수정 후 최적의 파라미터를 찾을 예정

4. Bounding box 검출 : WBF를 사용하여 nms나 soft-NMS와 같은 서로 다른 모델들에
서 얻어진 모든 bounding box들의 confidence score를 활용해 평균적인 box의 위치를
구하여 bounding box의 정확도를 높임

😊 감사합니다 😊