AI_17_김준혁

자율주행 인지판단을 위한 객체 인식

Section 3 Project

CONTENTS

자율주행 인지판단을 위한 객체 인식

CONTENTS 1

 # 자율주행

자율주행 시스템

객체 인식의 필요성

CONTENTS 2

YOLOv5

 # YOLO v5

객체 인식 모델 선정

YOLO v5 구조

CONTENTS 3

 # 학습 # 검증 # 평가

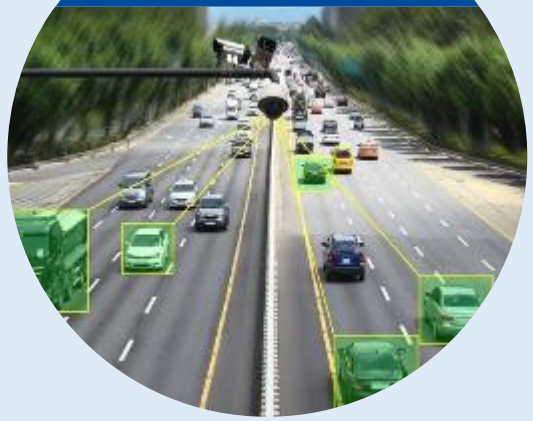
DATASET

Custom Training

Model 성능 분석

Model Test

CONTENTS 4



 # 결론

결론

자율주행 시스템

Contents 1

- 자율주행 시스템 주요 구성 -

- GPS
- RADAR
- LiDAR
- Camera
- 그 외 초음파 센서 등등

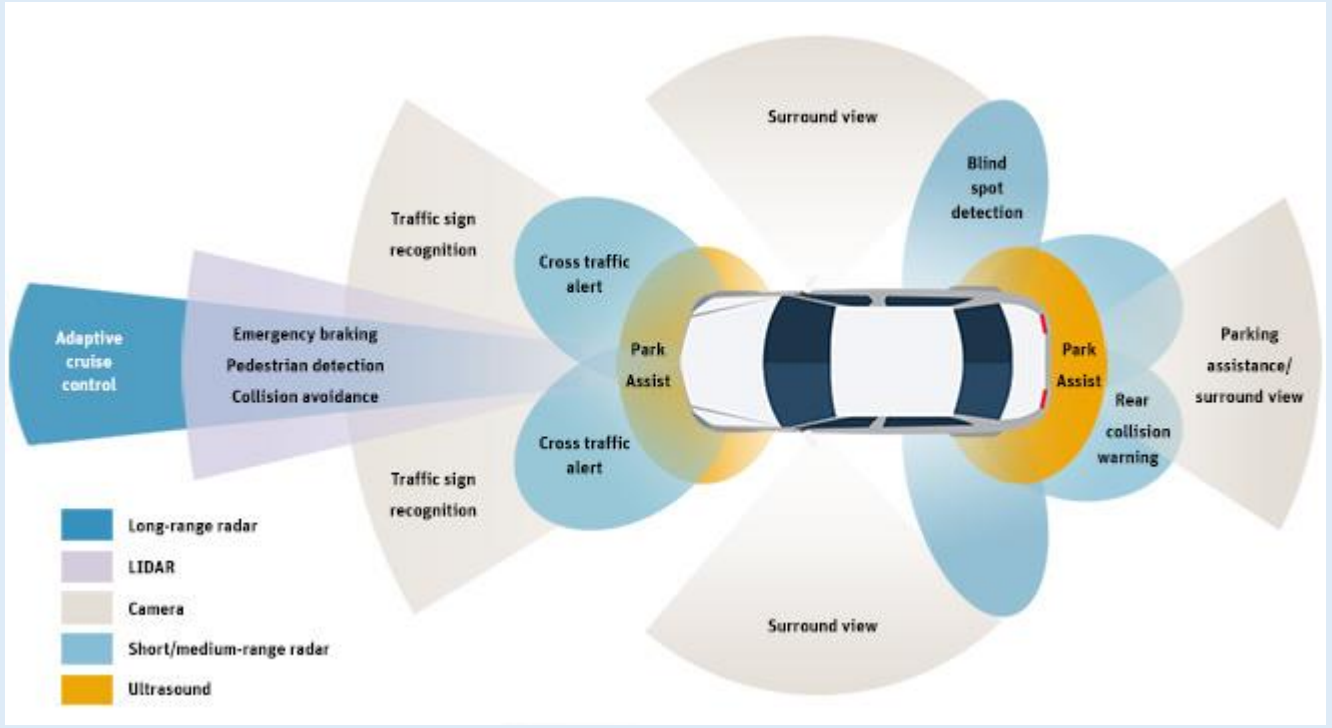
- 자율주행 제어 방식은 차량의 위치, 다른 사물과의 거리를 센서로 측정한 값을 이용해 자율주행 제어를 하는 방식

- 하지만 각 센서별로 한계점이 있어서 사람과 흡사한 주행 시스템을 갖추기는 어려움

(GPS는 터널, 고층 빌딩 주변 등 위치를 판별하기 어려운 경우)

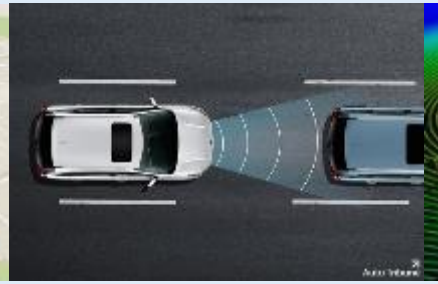
(RADAR, LiDAR 센서로 거리를 판단하지만 센서에서 인식한 물체가 급경사 도로(언덕 등)일 경우)

(Camera는 광도에 따라 인식이 바뀔 수 있음)

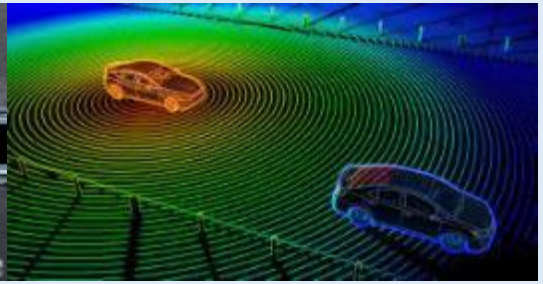


출처 : https://www.e4ds.com/sub_view.asp?ch=31&t=0&idx=12365

GPS



RADAR



LiDAR

객체 인식의 필요성

Contents 1

- 센서를 통한 제어의 한계점을 보완하기 위해 객체 인식을 도입

- 사람이 운전할 때 사주경계를 하는 것과 같이 객체 인식을 통해 주행 경로에 있는 보행자, 차량, 표지판 등등을 탐지하여 주행 시 속도, 방향을 결정할 수 있어 주행 안정성을 높일 수 있음

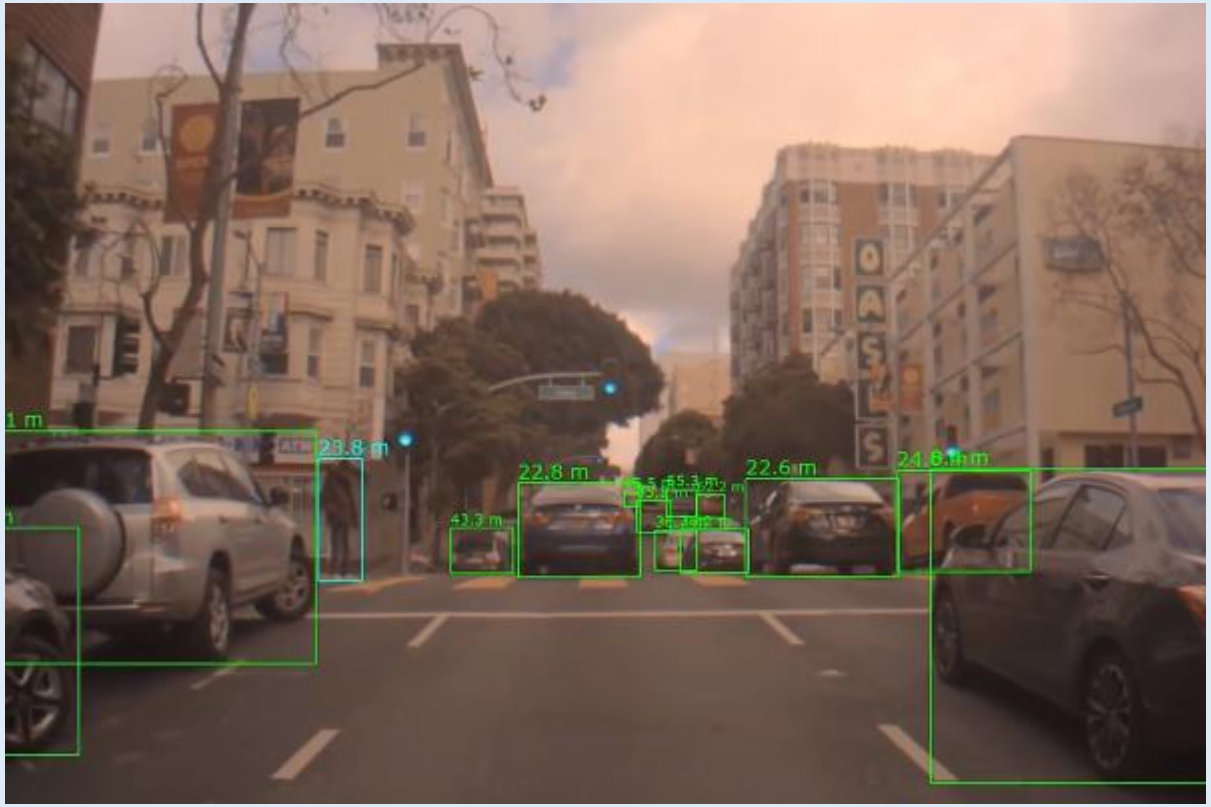
- 현재는 LiDAR의 3D MAP과 CAMERA의 2D IMAGE를 융합하여 객체를 인식하는 기술이 구현되었고 계속 연구 중

- 이번 Project는 차량 주행 IMAGE Data를 이용한 객체 인식을 진행

- 도로의 객체를 인식하기 위해 주행 IMAGE 만으로도 학습 성능이 좋게 나올지에 대한 가설 설정

H0 : (주행 IMAGE만 학습 시 성능 ≥ 다른 IMAGE 포함 학습 시 성능)

H1 : (주행 IMAGE만 학습 시 성능 < 다른 IMAGE 포함 학습 시 성능)



출처 : https://www.e4ds.com/sub_view.asp?ch=31&t=0&idx=12365

객체 인식 모델 선정

Contents 2

- YOLO(You Only Look Once) -

- YOLO 모델 주요 특징

➢ IMAGE(2D) 파일 학습 및 추론 가능

➢ 1-Stage Detection : Regional Proposal과 classification이 동시에 이루어짐. 2-Stage Detection 과의 차이는 이미지 데이터를 한 번만 본다는 점

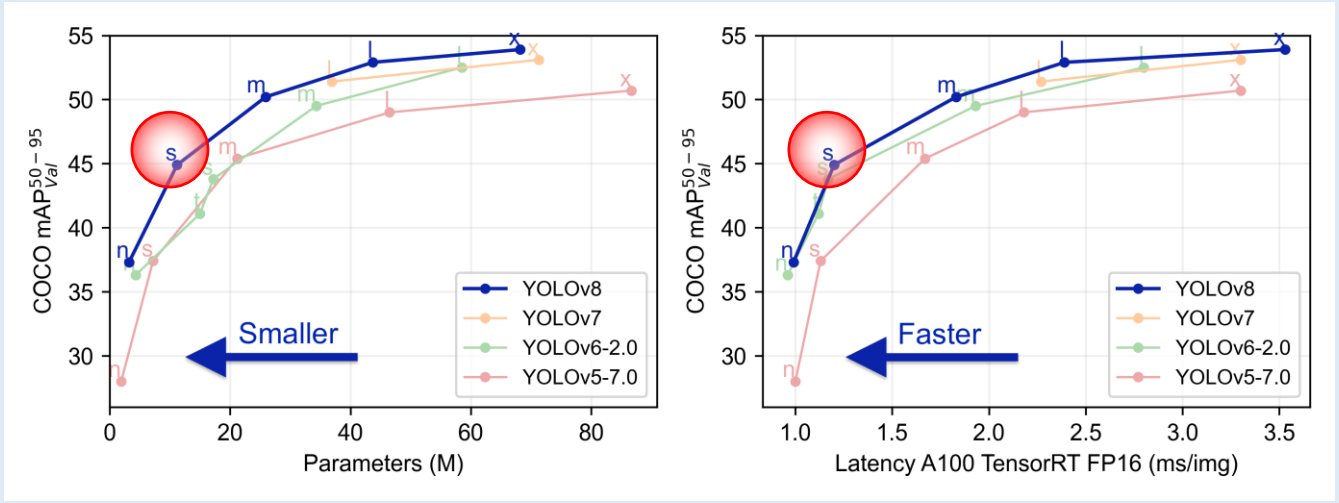
➢ Bounding Box : Labeling 방식 중에서 Bounding Box를 사용 (추론 속도를 빠르게 하기 위해 채택된 것으로 보임)

➢ YOLO v5는 Pytorch 기반 모델

- YOLO v5s 선택 이유 -

- 자율주행 차량에서의 AI 모델은 실시간으로 데이터를 추론해야 하므로 속도가 중요하다고 판단. 따라서 1-Stage Detection 방법을 사용하는 YOLO 모델을 선택
- Version 5의 장점은 모델 구조나 모델 학습 시 Custom 하기 좋게 배포되었다는 점
- 모델명 가장 뒤에 붙는 알파벳은 모델의 성능, 파라미터 개수에 따라 바뀌는데, 추론 성능이 어느 정도 보장되고 가장 빠르다고 판단한 **s**를 선택

출처 : https://github.com/ultralytics/yolov5



출처 : https://github.com/ultralytics/yolov5

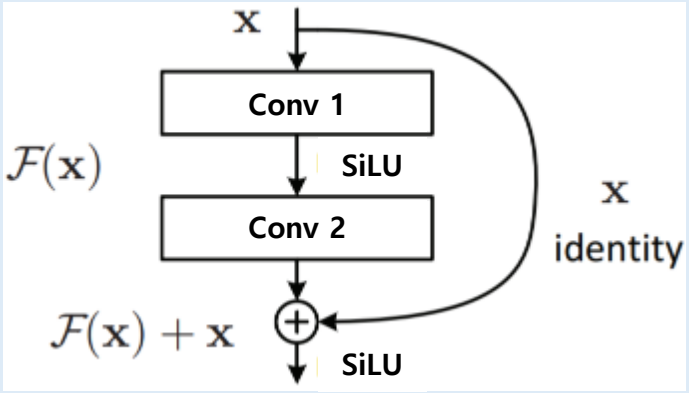
YOLO v5 구조

Contents 2

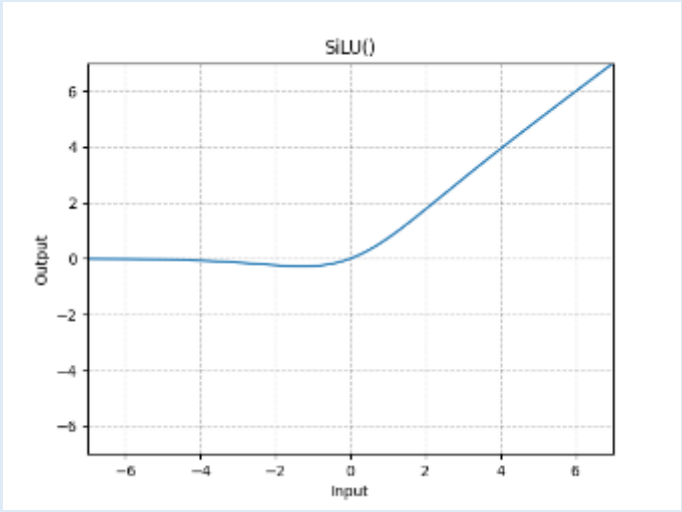
- Backborn -

- CNN을 통해 원본 Image에서 Feature map을 추출 (특징을 찾는 부분, Downsampling)
- C3이라는 이름의 modul은 **BottleneckCSP** Layer
- C3 modul의 number 수에 따라 BottleneckCSP Layer 내의 **Bottleneck** Layer를 얼마나 반복할지가 정해짐 (number=3 -> Bottleneck Layer=1)
- SPPF는 최종적으로 Fully Connected Layer에서 입력으로 들어갈 수 있도록 연산하는 Layer

```yaml
backbone:
  # [from, number, module, args]
  [[-1, 1, Conv, [64, 6, 2, 2]],  # 0-P1/2
   [-1, 1, Conv, [128, 3, 2]],  # 1-P2/4
   [-1, 3, C3, [128]],
   [-1, 1, Conv, [256, 3, 2]],  # 3-P3/8
   [-1, 6, C3, [256]],
   [-1, 1, Conv, [512, 3, 2]],  # 5-P4/16
   [-1, 9, C3, [512]],
   [-1, 1, Conv, [1024, 3, 2]],  # 7-P5/32
   [-1, 3, C3, [1024]],
   [-1, 1, SPPF, [1024, 5]],  # 9
  ]
```



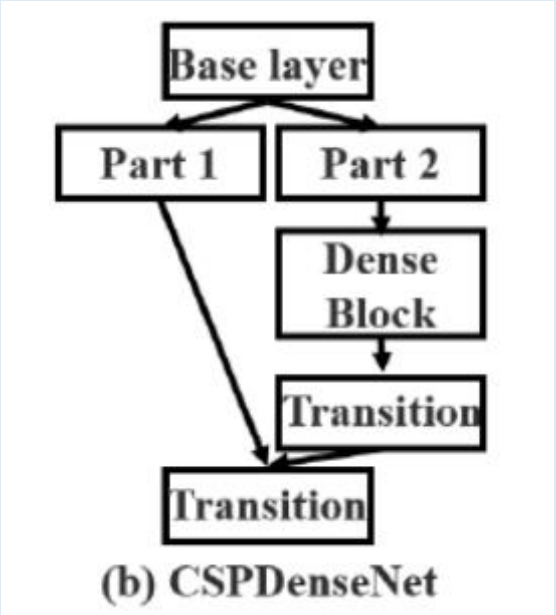
Bottleneck 구조 (skip-connection)



활성화 함수로 사용된 SiLU
(강화 학습, 분류 문제에서 좋다고 함)

$$\text{silu}(x) = x * \sigma(x)$$

$\sigma(x)$ = logistic sigmoid



CSP(Coss Stage Partial Network) 구조

YOLO v5 구조

Contents 2

- Head -

- Backbone에서 추출된 Feature map을 이용해 물체의 위치를 찾는 부분 (Upsampling, Detecting)
- C3 부분은 Backborn에서 설명한 것과 동일한 BottleneckCSP Layer

```
head:
  [[-1, 1, Conv, [512, 1, 1]],
   [-1, 1, nn.Upsample, [None, 2, 'nearest']],
   [[-1, 6], 1, Concat, [1]],  # cat backbone P4
   [-1, 3, C3, [512, False]],  # 13

   [-1, 1, Conv, [256, 1, 1]],
   [-1, 1, nn.Upsample, [None, 2, 'nearest']],
   [[-1, 4], 1, Concat, [1]],  # cat backbone P3
   [-1, 3, C3, [256, False]],  # 17 (P3/8-small)

   [-1, 1, Conv, [256, 3, 2]],
   [[-1, 14], 1, Concat, [1]],  # cat head P4
   [-1, 3, C3, [512, False]],  # 20 (P4/16-medium)

   [-1, 1, Conv, [512, 3, 2]],
   [[-1, 10], 1, Concat, [1]],  # cat head P5
   [-1, 3, C3, [1024, False]],  # 23 (P5/32-large)

   [[17, 20, 23], 1, Segment, [nc, anchors, 32, 256]],  # Detect(P3, P4, P5)
  ]
```

- Concat Layer를 통해 small, medium, large 크기의 물체를 검출
- Feature map이 얼마나 축소되었는가에 따라 물체 검출 크기가 정해지는 것으로 추측

- 가장 마지막에 있는 Detect Layer에서는 1-Stage Detection으로 Regional Proposal과 Classification이 같이 진행
- **Anchor Box**를 통해 예측할 Box 값 도출 (Regression 문제)
- **Loss**
  - **GIoU** : Bounding box 실제와 예측값의 차이에 관한 Loss 값 (IoU에서 파생된 평가 지표에는 GIoU, DIoU, CIoU가 있음, YOLO v5 모델의 Default는 GIoU)
  - **objectness loss** : 객체 탐지 Loss (객체가 있을 경우와 없을 경우의 Loss를 따로 구하고, 각 Loss에 가중치를 곱해서 불균형 문제 해결 후 Loss 값을 합침)
  - **classification loss** : class 실제와 예측값이 맞는지에 대한 Loss

DATASET

Contents 3

- 학습 DATASET -

- 국가 교통 데이터 오픈마켓, 차량 전방 카메라 데이터셋 (강남)
(https://www.bigdata-transportation.kr/frn/prdt/detail?prdtId=PRDTNUM_000000000125)

- 배포한 데이터를 전부 학습하기에는 리소스가 부족하여 일부만 사용

- 이미지 데이터 : 총 8,823개(약 20.2G)
  - ❖ Training(7,058개), Validation(1,765)
  - ❖ Size : 1920 x 1080

- 확장자가 json인 label 파일 포함

g_F_1-000190.json	2020-12-30 오후 5:10	JSON 파일	2KB
g_F_1-000190.png	2020-12-26 오후 6:32	PNG 파일	2,936KB
g_F_1-000191.json	2020-12-30 오후 5:10	JSON 파일	2KB
g_F_1-000191.png	2020-12-26 오후 6:32	PNG 파일	2,934KB
g_F_1-000192.json	2020-12-30 오후 5:10	JSON 파일	2KB
g_F_1-000192.png	2020-12-26 오후 6:32	PNG 파일	2,848KB
g_F_1-000193.json	2020-12-30 오후 5:10	JSON 파일	2KB
g_F_1-000193.png	2020-12-26 오후 6:32	PNG 파일	2,750KB
g_F_1-000194.json	2020-12-30 오후 5:10	JSON 파일	2KB
g_F_1-000194.png	2020-12-26 오후 6:32	PNG 파일	2,676KB

원본 이미지, Label 시각화

- 학습 DATASET -

- YOLO v5 모델로 학습 시키기 위해 DATA Folder 정리
- 기존의 Label 파일을 YOLO v5 용 Label 파일로 변환


```
{
    "imagePath": "g_F_1-000190.jpg",
    "imageData": null,
    "shapes": [
        {
            "shape_type": "rectangle",
            "points": [
                [
                    804.55,
                    383.17
                ],
                [
                    920.46,
                    486.96
                ]
            ],
            "flags": {},
            "group_id": null,
            "label": "truck"
        },
        {
            "shape_type": "rectangle",
            "points": [
                [
                    1279.16,
                    376.5
                ],
                [
                    1318.5,
                    407.7
                ]
            ],
            "flags": {},
            "group_id": null,
            "label": "car"
        },
```


```
### 기존의 .json 파일을 YOLOv5 format의 .txt 파일로 변환
def label_parsing(path):
  labels_list = glob.glob(path + '\\' + '*.json')
  class_list = {}
  i = 0
  path = 'E:\\vehicle_front_camera_20200904_gangnam\\data\\labels\\train\\'

  for label in labels_list:
    name = label.split('\\')[-1].split('.')[0]
    txt = path + name + '.txt'
    txt_file = open(txt, 'w')

    with open(label, 'r') as f:
      file = json.load(f)

    for object in file['shapes']:
      class_name = object['label']

      if class_name not in class_list.keys():
        class_list[class_name] = i
        i += 1

      x1, y1 = object['points'][0]
      x2, y2 = object['points'][1]

      # Yolo v5 Labeling
      c = str(class_list[class_name])
      x = str((max(x2, x1) - (abs(x2 - x1) / 2.)) / 1920.)
      y = str((max(y2, y1) - (abs(y2 - y1) / 2.)) / 1080.)
      w = str(abs(x2 - x1) / 1920.)
      h = str(abs(y2 - y1) / 1080.)

      # save .txt
      txt_file.write(' '.join([c, x, y, w, h]) + '\n')

  txt_file.close()
  f.close()
  return class_list
```

```
custom_dataset
|
├── test/
|   ├── images/
|   └── labels/
|
├── train/
|   ├── images/
|   └── labels/
|
├── valid/
|   ├── images/
|   └── labels/
|
├── data.yaml
└── README.dataset.txt
```

YOLOv5 label FORMAT
(class_id, center_x, center_y, width, height)

```
0 0.44922135416666664 0.402837962962963 0.06036979166666671 0.09610185185185181
1 0.6764739583333333 0.3630555555555556 0.02048958333333329 0.028888888888888877
0 0.20026041666666666 0.4235185185185857 0.4005208333333333 0.620925925925926
1 0.6130104166666667 0.36825 0.023614583333333408 0.024462962962978
1 0.5420390625000001 0.428425925925926 0.097140625 0.11925925925925927
```

Custom Training

Contents 3

- YOLO v5s 모델 학습 –

- YOLO v5는 명령줄로 커스텀 모델 생성 및 학습을 진행할 수 있음
- Github에서 Load 한 YOLO v5의 train.py를 사용하여 학습
- 이미지 데이터 증강, Scaling 등 전처리 과정이 포함되어 학습 진행

- **주요 명령어**

➢ --img : 기존의 1920x1080 이미지를 가로 512의 사이즈로 Scaling

➢ --batch : 학습 시 batch_size

➢ --epochs : 학습 Epochs

➢ --weights : 가중치 초기화를 위한 명령어 (yolov5s.pt는 가중치가 초기화 되어 있는 파일. Pytorch 용)

➢ --hyp : 하이퍼 파라미터 설정 (커스텀 가능한 하이퍼 파라미터가 많아서 세부 내용은 제외)

```
!python yolov5-master\\train.py --img 512 --batch 16 --epochs 50 --data E:\\vehicle_front_camera_20200904_gangnam\\data\\custom_data.yaml --weights yolov5s.pt --evolve 3 --name yolo5 --project train_50epochs
# 하이퍼 파라미터 튜닝
```

```
!python yolov5-master\\train.py --img 512 --batch 16 --epochs 100 --data E:\\vehicle_front_camera_20200904_gangnam\\data\\custom_data.yaml --weights yolov5s.pt --hyp train_50epochs\yolo5\hyp_evolve.yaml --name yolo5 --project train_100epochs
# 튜닝된 하이퍼 파라미터로 다시 학습
```

- 최종 모델–

- YOLOv5s 기본 모델은 213개의 layers, 7225885개의 parameters, 80개의 label class
- YOLOv5s Custom 모델은 157 layers, 7185430개의 parameters, 37개의 label class (best epochs 기준)

```
YOLOv5  2023-3-5 Python-3.10.9 torch-1.13.1 CUDA:0 (NVIDIA GeForce RTX 2080 Ti, 11264MiB)

Fusing layers...
YOLOv5s summary: 213 layers, 7225885 parameters, 0 gradients
Adding AutoShape...
```

```
YOLOv5  2023-3-5 Python-3.10.9 torch-1.13.1 CUDA:0 (NVIDIA GeForce RTX 2080 Ti, 11264MiB)

Fusing layers...
Model summary: 157 layers, 7185430 parameters, 0 gradients, 16.3 GFLOPs
Adding AutoShape...
```

모델 성능 분석

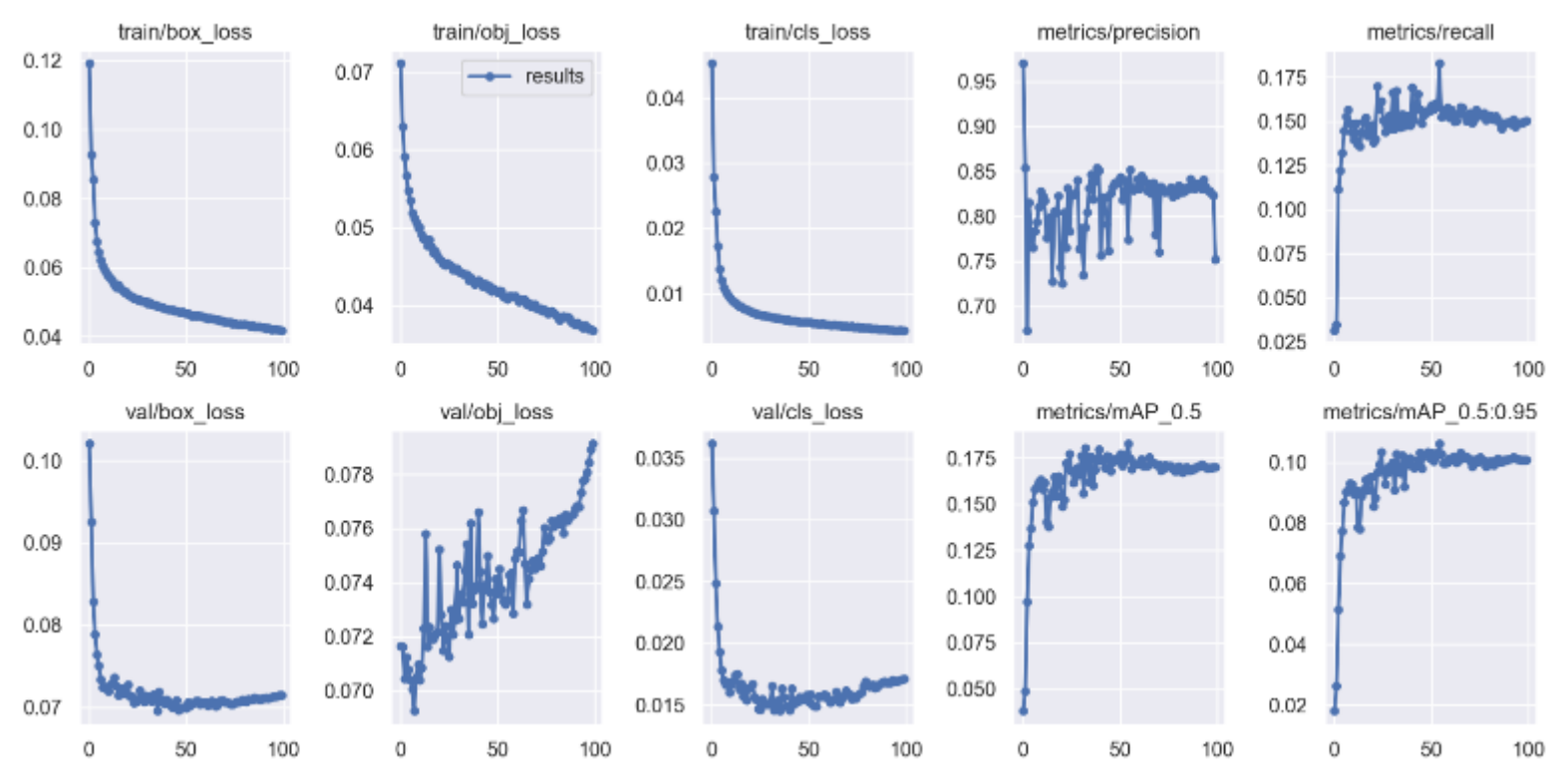
Contents 3

- wandb를 이용한 분석 -

- EPOCH에 따른 변화 (100 EPOCH)

- box_loss, classification_loss, precision 은 의미 있게 나왔지만, objectness_loss, recall, mAP는 성능이 좋지 않은 것을 알 수 있음

- 객체 인식 모델에서 recall 값은 얼마나 Detecting을 잘했는지 알 수 있는 지표인데, 값이 낮게 나온 이유 분석 필요



모델 성능 분석

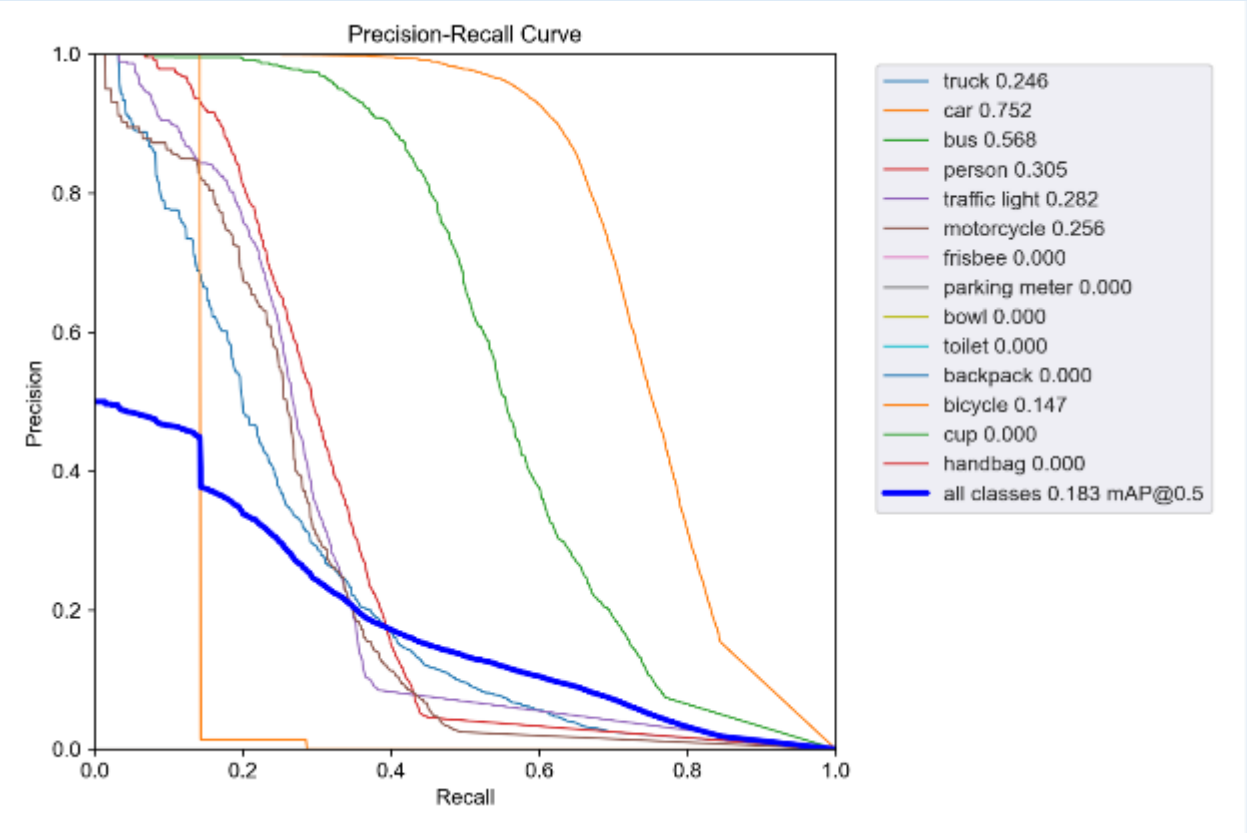
Contents 3

- PR(Precision-Recall) Curve를 통해 확인해 본 결과 -

- 기존 Dataset의 labeling이 도로 상황과 관련된 class로 이루어지지 않은 것을 확인 (backpack, cup, handbag, toilet 등)

- Dataset을 일부만 사용했기 때문에 상대적으로 적게 출현하는 class는 PR Curve 값이 낮게 나온 것을 확인 (bicycle, motorcycle 등)

- 이 평가 지표를 통해 PR Curve의 아래 면적을 계산하는 AP(Average Precision), 그리고 각 class 별 AP의 평균인 mAP(mean Average Precision) 값 또한 낮게 나왔다는 것을 알 수 있음



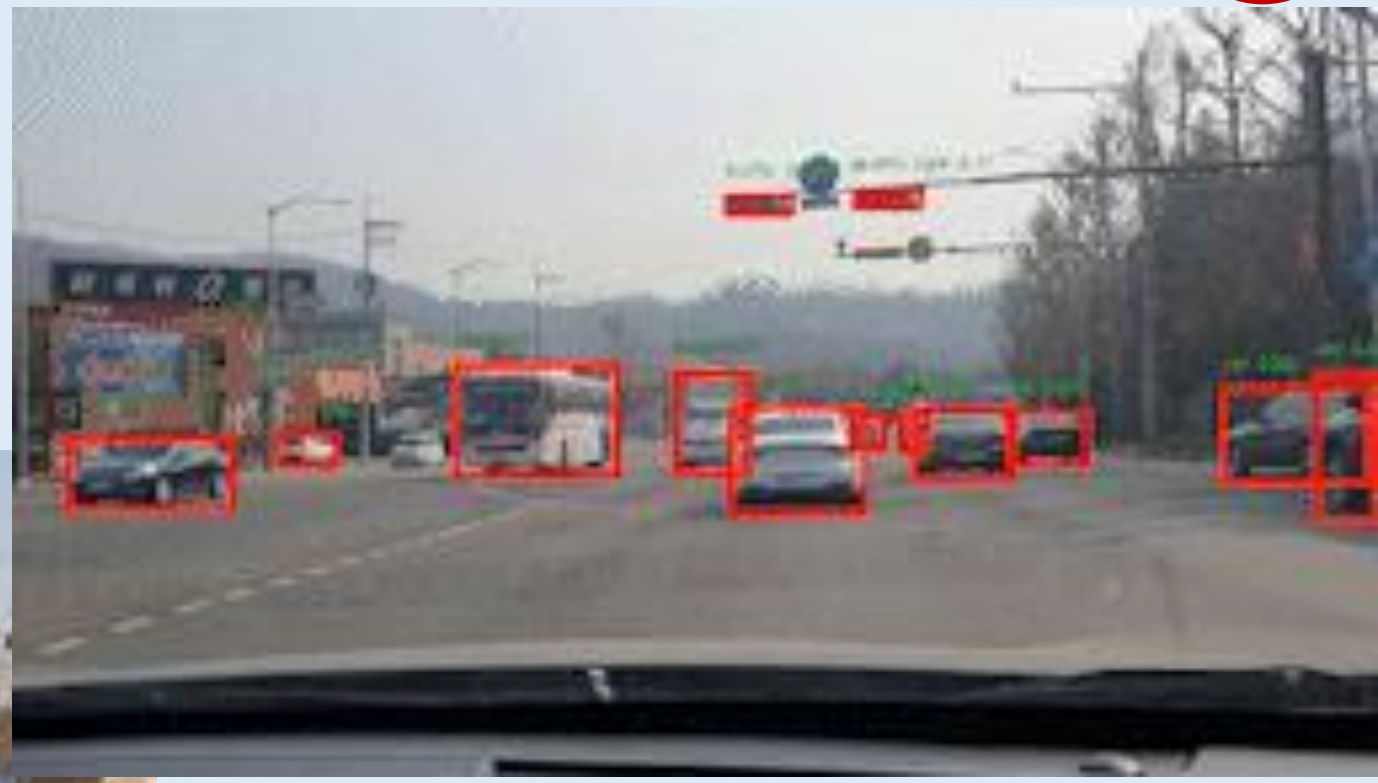
Model Test

Contents 3

- 실제 주행 영상 적용 -

- 성능이 아쉽지만 실제 적용 시를 opencv 패키지를 사용해 시각화

- 자가용으로 주행하며 직접 촬영한 영상 (사당 - 과천)
- 주로 장애물에 가려진 차량, 그늘진 곳에 있는 사람은 제대로 감지하지 못한다는 것을 확인



결론

Contents 4

❖ 현재 모델은 자율 주행에 적용할 수 없음. 그 이유는 안정성을 높이기 위함이 목적인데 Object Detecting이 제대로 되지 않기 때문 (Recall이 너무 낮음)

❖ 편향된 학습이 되지 않도록 여러 가지 상황, 여러 가지 객체들이 들어있는 Image Dataset으로 학습시킬 필요가 있음 (H0 기각, H1 채택)

❖ 추가로 상용화할 자율주행 차량 및 로봇에 AI를 추가시키기 위해서는 그만큼의 추가 리소스가 필요

❖ 필요 리소스를 줄이기 위해 모델을 얼마만큼 경량화할 수 있는지를 연구해야 함

❖ 추후 진행해 보고 싶은 프로젝트

- ➢ YOLO 모델을 기반으로 경량화된 모델을 구현
- ➢ 현재 구현되어 있고 계속해서 연구 중에 있는 Lidar + Camera 융합 객체 인식 모델 구현
- ➢ YOLO v8 모델 구현

- Project 회고 -

❖ 원래 목표는 Pytorch 기반인 YOLO 모델을 Tensorflow로 구현해 보는 것이었으나, 모델 구조 이해, 구현 능력이 갖춰지지 않아서 프로젝트 기간에 맞춰 할 수 없겠다고 생각이 들었고, 결국 현재의 프로젝트로 진행되었습니다.

❖ 여러 가지 수식과 파이썬 코딩이 익숙해져서 커스텀 모델을 만들고 연구하는 것이 목표입니다. (Pytorch도 공부해야 함)

AI_17_김준혁

THANK YOU

kjh970104@gmail.com