

CP1 – yolov5를 이용한 영상 내 과적차량 감지 모델

AI 16기 – 한승희

목차

01. 프로젝트 개요

02. yolov5 모델 선택

03. 데이터 소개

04. 모델 학습 및 성능평가

05. 패키지 소개 및 사용

06. 한계점 및 향후 개선방향

1. 프로젝트 개요

❖ 프로젝트 배경


영상에서 과적차량을 인식하여 단속하는 서비스를 개발하고자 한다

❖ 이용 데이터

과적 차량(500장)과 정상 차량(500장)의 이미지(jpg) 및 라벨링 데이터(json)


❖ 사용 모델 : yolov5s6

02. yolov5 모델 선택




Small
YOLOv5s

14 MB_{FP16}
2.2 ms_{V100}
36.8 mAP_{COCO}



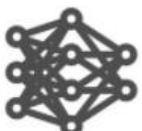
Medium
YOLOv5m

41 MB_{FP16}
2.9 ms_{V100}
44.5 mAP_{COCO}



Large
YOLOv5l

90 MB_{FP16}
3.8 ms_{V100}
48.1 mAP_{COCO}



XLarge
YOLOv5x

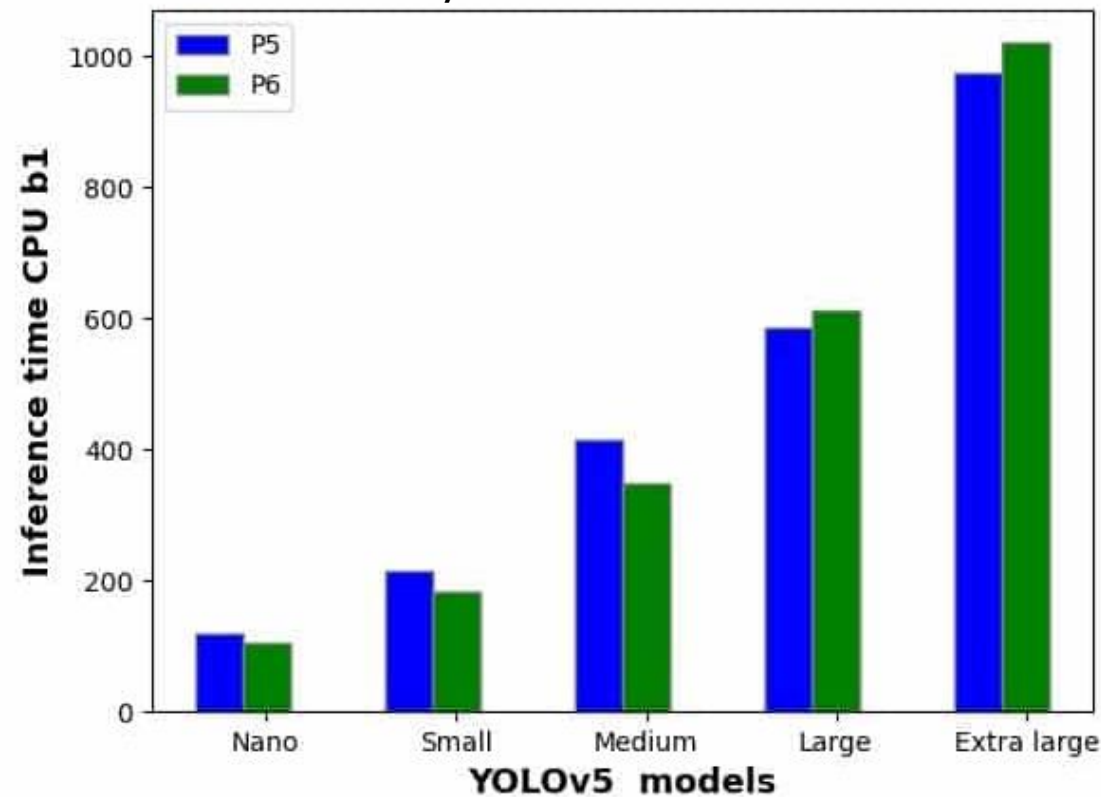
168 MB_{FP16}
6.0 ms_{V100}
50.1 mAP_{COCO}

<https://github.com/ultralytics/yolov5/wiki/Train-Custom-Data>

✓ YOLOv5의 모델 중 비교적 작으면서 가장 빠른 모델인 YOLOv5s 를 사용

Model	size(pixels)	mAPval50-95	mAPval50	SpeedCPU b1(ms)
YOLOv5s	640	37.4	56.8	98
YOLOv5s6	1280	44.8	63.7	385

yolov5 의 P5 및 P6 비교



<https://learnopencv.com/object-detection-using-yolov5-and-opencv-dnn-in-c-and-python/#YOLOv5-Speed-test-with-input-size-variations>

- ✓ 또한, YOLOv5s의 P5와 P6을 비교해보면 처리시간은 오른쪽 그래프와 같이 P6이 더 짧다고 하는 경우도 있어 장치에 따라 차이가 나는 것으로 생각됨
- ✓ P6인 YOLOv5s6은 YOLOv5s에 비해 더 큰 객체를 감지할 수 있으며 이에 따라 약간 향상된 성능을 나타내므로, 본 프로젝트에서 YOLOv5s6 를 선택함

03. 데이터 소개

학습에 이용한 데이터 정보

❖ 데이터 준비

label (json)

1,000 개



image (jpg)

1,000 개

- Data : train 900 개, valiation 80 개, test 20 개
(이미지 하나당 json 라벨링 데이터 1건)
- 이미지 해상도 : 1920 x 1080,
- label 좌표 : xmin, ymin, width, height
(→ yolo 모델에 맞도록 좌표 변환해주는 작업 필요 ►)

```
# 좌표변환 & 정규화 함수
def convert(size, box):
    dw = 1/size[0]
    dh = 1/size[1]
    w = round(box[2]*dw,7)
    h = round(box[3]*dh,7)
    x = round(box[0]*dw+w/2,7)
    y = round(box[1]*dh+h/2,7)
    if w < 0 or h < 0:
        return False
    return (x,y,w,h)
```

❖ 폴더 구조 구성

yolo 모델에 맞도록 폴더 구조 구성 필요



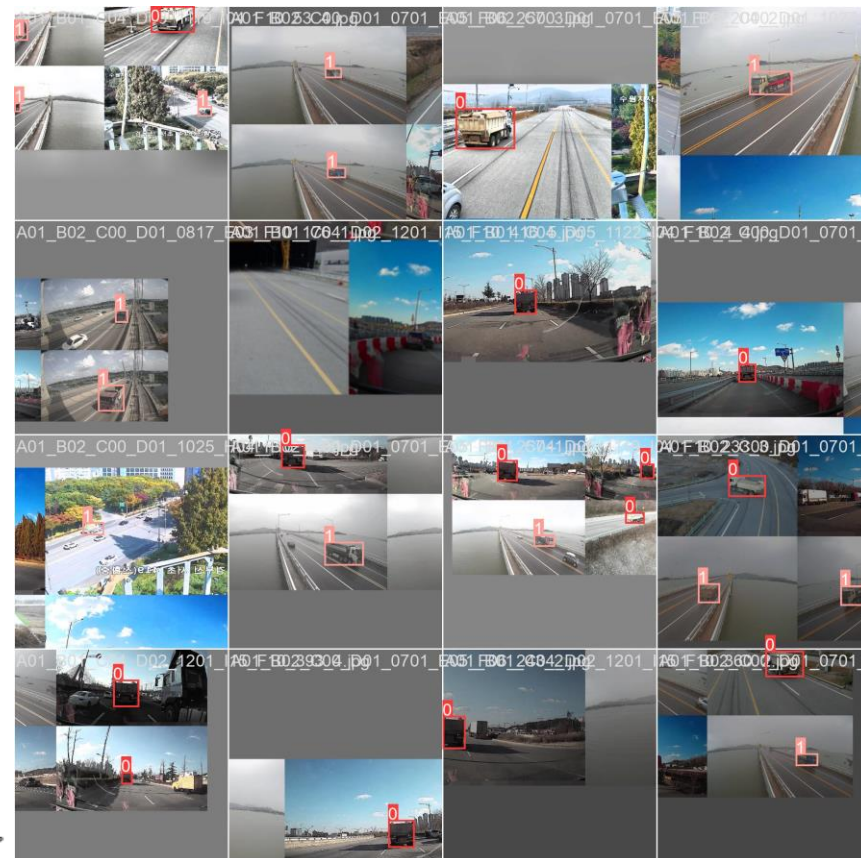
04. 모델 학습 및 성능평가

❖ 모델 학습

```
!python train.py --img 832 --batch 32 --epochs 50 --data ./vehicle/dataset.yaml\  
--cfg ./models/custom_yolov5s6.yaml --weights ''\
```

	from	n	params	module	arguments
0	-1	1	3520	models.common.Conv	[3, 32, 6, 2, 2]
1	-1	1	18560	models.common.Conv	[32, 64, 3, 2]
2	-1	1	18816	models.common.C3	[64, 64, 1]
3	-1	1	73984	models.common.Conv	[64, 128, 3, 2]
4	-1	2	115712	models.common.C3	[128, 128, 2]
5	-1	1	295424	models.common.Conv	[128, 256, 3, 2]
6	-1	3	625152	models.common.C3	[256, 256, 3]
7	-1	1	885504	models.common.Conv	[256, 384, 3, 2]
8	-1	1	665856	models.common.C3	[384, 384, 1]
9	-1	1	1770496	models.common.Conv	[384, 512, 3, 2]
10	-1	1	1182720	models.common.C3	[512, 512, 1]
11	-1	1	656896	models.common.SPPF	[512, 512, 5]
12	-1	1	197376	models.common.Conv	[512, 384, 1, 1]
13	-1	1	0	torch.nn.modules.upsampling.Upsample	[None, 2, 'nearest']
14	[-1, 8]	1	0	models.common.Concat	[1]
15	-1	1	813312	models.common.C3	[768, 384, 1, False]
16	-1	1	98816	models.common.Conv	[384, 256, 1, 1]
17	-1	1	0	torch.nn.modules.upsampling.Upsample	[None, 2, 'nearest']
18	[-1, 6]	1	0	models.common.Concat	[1]
19	-1	1	361984	models.common.C3	[512, 256, 1, False]
20	-1	1	33024	models.common.Conv	[256, 128, 1, 1]
21	-1	1	0	torch.nn.modules.upsampling.Upsample	[None, 2, 'nearest']
22	[-1, 4]	1	0	models.common.Concat	[1]
23	-1	1	90880	models.common.C3	[256, 128, 1, False]
24	-1	1	147712	models.common.Conv	[128, 128, 3, 2]
25	[-1, 20]	1	0	models.common.Concat	[1]
26	-1	1	296448	models.common.C3	[256, 256, 1, False]
27	-1	1	590336	models.common.Conv	[256, 256, 3, 2]
28	[-1, 16]	1	0	models.common.Concat	[1]
29	-1	1	715008	models.common.C3	[512, 384, 1, False]
30	-1	1	1327872	models.common.Conv	[384, 384, 3, 2]
31	[-1, 12]	1	0	models.common.Concat	[1]
32	-1	1	1313792	models.common.C3	[768, 512, 1, False]
33	[23, 26, 29, 32]	1	26964	models.yolo.Detect	[2, [[19, 27, 44, 40, 38, 94],

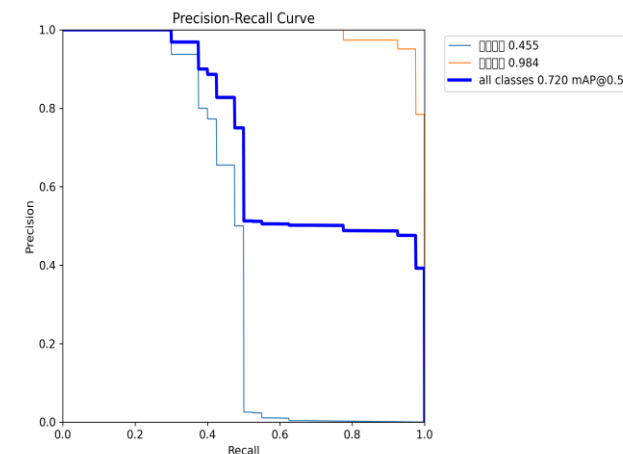
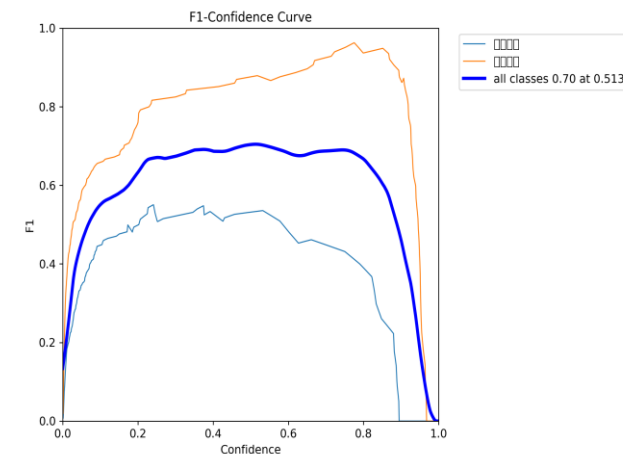
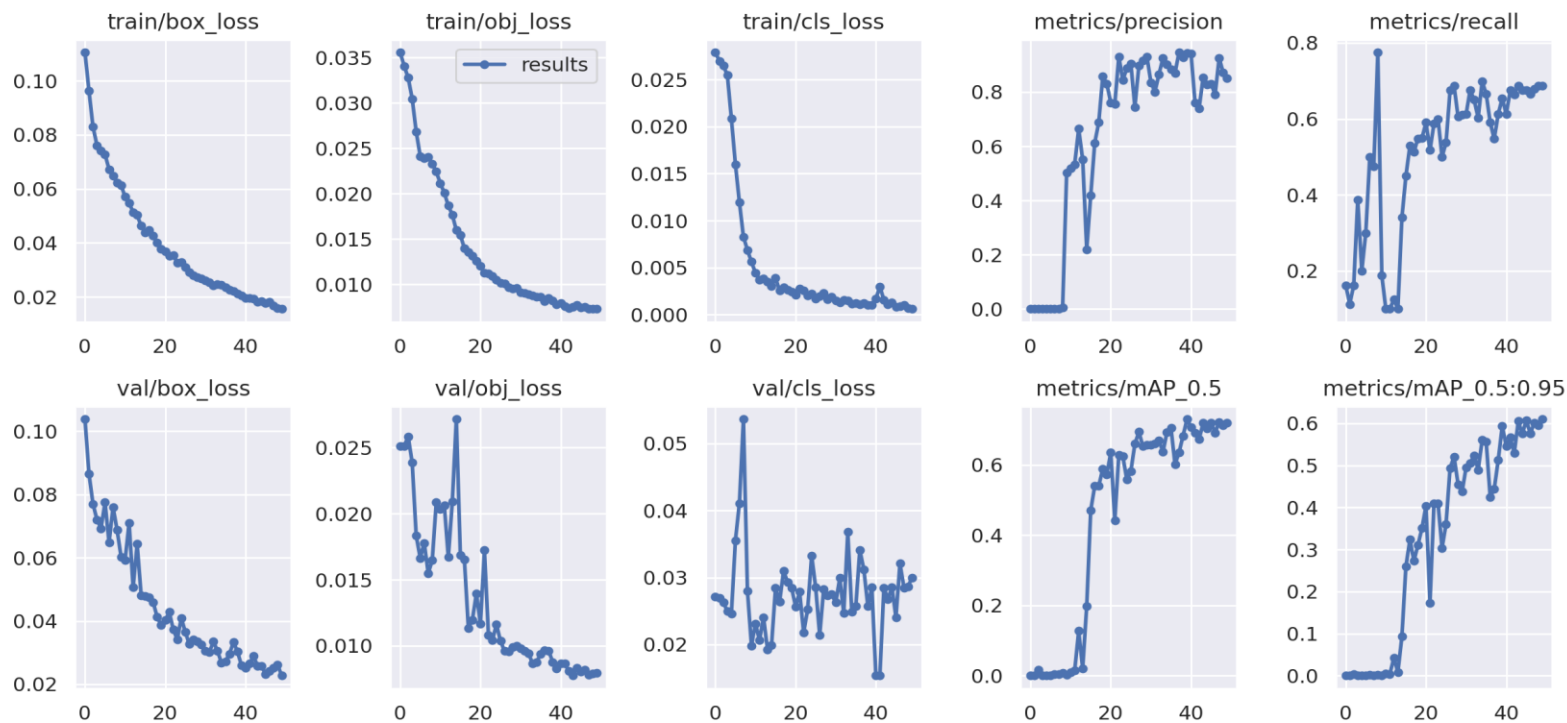
custom_YOLOv5s6 summary: 281 layers, 12326164 parameters, 12326164 gradients, 16.3 GFLOPs



04. 모델 학습 및 성능평가

❖ 성능 평가

- mAP_0.5 : 0.7194, mAP_0.5:0.95 : 0.6101, precision : 0.851, recall : 0.6875
- 약 20 epoch 부터 성능 지표의 추이가 일관된 패턴을 보이기 시작함
- 다른 지표에 비해 class loss의 편차는 크게 나타나는 것으로 보아 class 구분의 성능이 좋지 않음을 알 수 있고, 이에 따라 더 많은(다양한) 학습데이터로 훈련하는 작업이 필요해보임



05. 패키지 소개 및 사용

❖ 패키지 소개

사전 학습한 모델을 사용해 영상 내 다양한 객체를 탐지하고 결과 영상을 만들어내는 기능 구현

```
├── data
│   ├── A01_B02_C00_D01_0703_E08_F03_554_1.jpg
│   ├── A01_B02_C00_D01_0703_E08_F03_568_3.jpg
│   ├── sample.mp4
│   └── sample2.mp4
├── models
│   └── best.onnx
├── results
│   ├── img_554.jpg
│   ├── img_568.jpg
│   ├── output.mp4
│   └── output2.mp4
├── test
│   └── test.ipynb
├── utils
│   ├── __init__.py
│   └── detection.py
├── AI_16기_한승희_CP1_DS.ipynb
├── README.md
├── detect.py
└── requirements.txt
```

data

모델을 test 할 수 있는 sample data 입니다.

models

onnx 형태의 사전학습된 model (best.onnx) 이 저장되어 있습니다.

모델 학습과정은 AI_16기_한승희_CP1_DS.ipynb 에서 볼 수 있습니다.

results

detection.py에 의해 생성된 결과 영상/이미지가 해당 폴더에 저장됩니다.

영상은 .mp4, 이미지는 jpg 형태로 저장됩니다.

test

해당 패키지가 정상적으로 작동하는지 확인할 수 있는 test.ipynb 파일을 포함하고 있습니다.

utils

best.onnx 모델을 통해 객체탐지를 수행하고 결과를 저장하는 detection.py 가 저장되어 있습니다.

05. 패키지 소개 및 사용

- label(=class name) 의 입력값이 한글일 때와 그렇지 않을 때로 구분 (문자 깨짐 문제)

```
label = f"{self.labels[class_ids[i]]} {confidences[i]:.3f}"
    if not is_ascii(label): # 한글 label
        frame = Image.fromarray(frame)
        draw = ImageDraw.Draw(frame)
        FONT = ImageFont.truetype('Fonts/gulim.ttc',
                                max(round(sum(frame.size) / 2 * 0.03), 12))
        (생략)
        draw.text((left-1, top-th-1 if outside else top), label, fill=WHITE, font=FONT)
    else: # cv2
        FONT_FACE= cv2.FONT_HERSHEY_SIMPLEX # 폰트 종류
        FONT_SCALE = 0.7 # 폰트 크기
        (생략)
        # Text 넣기
        cv2.putText(frame, label, (left, top-2 if outside else top+th+2),
                   FONT_FACE, FONT_SCALE, WHITE, thickness=1, lineType=cv2.LINE_AA)
```

- input 파일(jpg/mp4)에 따른 output 설정

```
if input_path.endswith(".mp4"):
    codec =cv2.VideoWriter_fourcc(*"mp4v")
    vid_fps = VideoSignal.get(cv2.CAP_PROP_FPS)
    vid_size = (int(VideoSignal.get(cv2.CAP_PROP_FRAME_WIDTH)), int(VideoSignal.get(cv2.CAP_PROP_FRAME_HEIGHT)))
    vid_writer = cv2.VideoWriter(output_path, codec, vid_fps, vid_size)
elif input_path.endswith(".jpg"):
    frame = cv2.imread(input_path)
    img = self.process(frame)
    cv2.imwrite(output_path, img)
```

❖ 패키지 사용

터미널 실행

```
git clone https://github.com/H-Seung/AI_16_HanSeungHee_CP1_DS.git
cd AI_16_HanSeungHee_CP1_DS
pip install -r requirements.txt
python detect.py
```

06. 한계점 및 향후 개선방향

❖ 한계점

- colab에서 yolo 패키지를 이용해 detect 한 결과와 본 패키지의 detect 결과(box 탐지, class 구분 둘 다)가 다르게 나타난다.
→ 임계값 설정이 다른 점이 가장 큰 원인일 것으로 추측하고, 또한 그 외 세부적인 코드 내용이 다름으로 인해 발생한다고 생각

❖ 향후 개선방향

- class 구분의 성능이 좋지 않아 더 많은 다양한 학습데이터로 훈련이 필요하다.
- 학습한 데이터에 거의 유사한 사진들이 5~6장씩 존재한다. 유사한 데이터들을 하나로 줄이면 학습 시간의 단축에 도움이 될 것이다.
- 터미널에서 파라미터 인자를 넘겨줄 수 있도록 코드 보완이 필요하다. (ex. `python detect.py --<input_path> --<output_path>`)
- yolov5s6 외에 다른 모델로 객체탐지를 수행하여 더 나은 성능을 보이는 모델을 찾는다.