

「실시간 객체 인식을 통한 이륜차 위험요소 탐지 서비스」 결과보고서



KDT2기 T3Q 1팀 임재원, 신민수, 박병준, 김지윤

Contents

1	분석요약	0
2	분석 실습	
1.	분석 개요	0
1.1	분석 배경	0
1)	개요	
2)	이슈(Pain Point)	
1.2	분석 목표	0
1)	분석 목표	
2)	데이터셋 정의 및 소개	
2.	분석 실습	0
2.1	위험물 탐지 데이터 소개	0
1)	데이터 유형/구조	
2)	데이터 정제(전처리)	
2.2	분석 모델 소개	0
1)	AI 학습모델 분석 및 비교	
2)	학습모델 평가지표	
3)	이론차 위험물 탐지 모델 평가	
2.3	분석 체험	0
1)	필요 SW, 패키지	
2)	추론 및 추론 방법	
3)	프로젝트 결과 평가	

「실시간 객체 인식을 통한 이륜차 위험요소 탐지 서비스」 결과보고서

- 필요 S W : Python, Anaconda, VS code 등
- 필요 패키지 : Tensorflow, Keras, matplotlib, Pandas, sklearn, numpy, seaborn, opencv-python, Pillow, PyYAML, requests, scipy, torch, torchvision, tqdm, tensorboard
- 분석 환경 : [CPU] 11th Gen Intel(R) Core(TM) i5-1155G7 @ 2.50GHz 2.50GHz [RAM] 16GB
- 필요 S W : Python, Anaconda, VS code 등
- 필요 패키지 : Tensorflow, Keras, matplotlib, Pandas, sklearn, numpy, seaborn, opencv-python, Pillow, PyYAML, requests, scipy, torch, torchvision, tqdm, tensorboard
- 분석 환경 : [CPU] 11th Gen Intel(R) Core(TM) i5-1155G7 @ 2.50GHz 2.50GHz [RAM] 16GB
- 필요 데이터 : *.png, *.png, *.json, *.txt

1 분석요약

No	구분		내용
1	분석 목적		<ul style="list-style-type: none"> - 이륜차 사고는 꾸준히 증가하고 있으며, 이륜자동차의 경우 자동차에 비해 도로 환경에 대한 위험 노출도와 부상 위험이 매우 크다. - 이륜차들이 주로 다니는 골목길, 이면도로와 같은 생활도로의 위험요소는 즉각적인 파악 및 보수가 잘 이루어지지 않고 있다. - 이륜차 주행 중 위험요소를 탐지하고 사용자에게 피드백하는 동시에, 해당 데이터를 수집하여 지자체에 전송한다.
2	데이터셋 형태 및 수집방법		1) 분석에 사용된 변수명 : PotHole, Person, Traffic Cone 외 4개, 이미지 내 객체의 바운딩 박스 좌표값 4가지(x1, y1, x2, y2) 및 해당 객체의 라벨 2) 데이터 수집 방법 : AI Hub에서 제작한 이륜자동차 안전 위험 시설물 데이터, Roboflow에서 제작한 Pothole Detection 3) 데이터셋 파일 확장자 : *.png, *.jpg, *.json, *.txt
3	데이터 개수 데이터셋 총		<ul style="list-style-type: none"> - 데이터 개수 : 영상 데이터 수(4,308개), 라벨링 데이터 수(4,308개), 총 8,616개 - 데이터셋 총량 : 4.13GB
4	분석적용 알고리즘	알고리즘	라벨이 있는 데이터를 Supervised learning(지도학습)의 객체 탐지(Object Detection) YOLO 모델을 통해 학습시키는 알고리즘을 선택한다.

		알고리즘 간략소개	<ul style="list-style-type: none"> - Supervised learning: Supervised learning(지도학습)이란, '목표치가 주어진 데이터'들을 모두 활용하여 학습하는 방법이다. 라벨을 가진 데이터에서 훈련을 위한 학습 셋과 테스트를 위한 테스트셋을 나누어 각각 구성한다. 전처리 과정을 통해 데이터를 가공하여 사용한다. - YOLO 네트워크: 데이터 학습 및 테스트를 위해 YOLO 네트워크를 활용한다. YOLO는 영상 기반 객체 검출에 주로 사용되는 네트워크로, 반복 학습 및 업데이트를 통해 최종적으로 객체의 종류 및 위치 정보를 결과로 출력한다.
5	분석결과 및 시사점		<ul style="list-style-type: none"> - YOLO AI 모델을 통해 객체 검출 및 위치 정보를 획득 및 추정하였다. - YOLO AI 모델을 통해 검출된 객체가 위험물 일시 사용자에게 피드백을 반영하고, 위치 및 이미지 정보를 지자체에 전송하는 서비스 개발을 진행하였다. - Streamlit 웹 서비스 구현, T3Q 플랫폼 구현을 진행하였다.

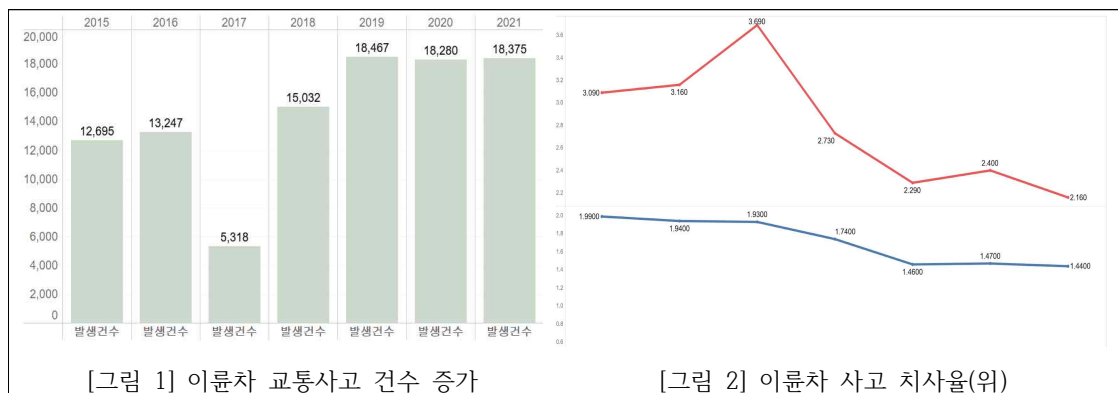
2. 분석 실습

1. 분석 개요

1.1 분석 배경

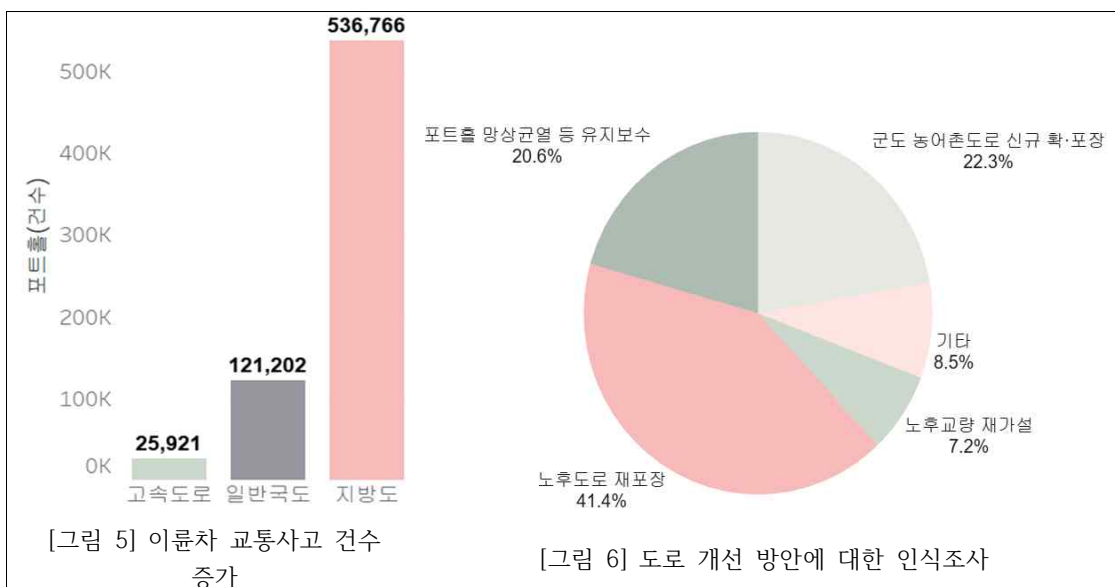
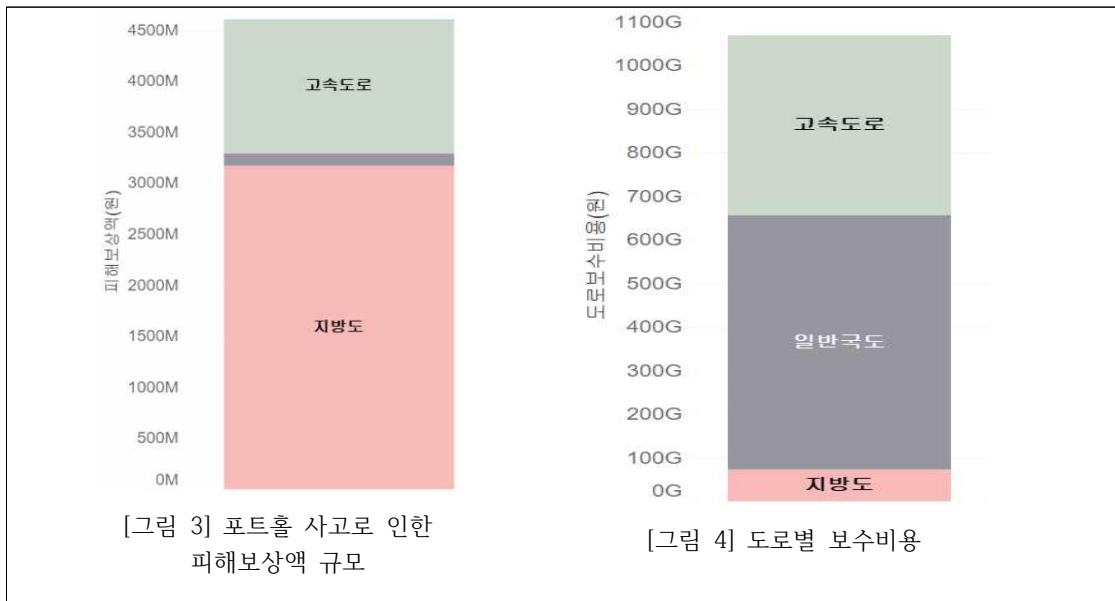
▶ 이륜자동차 위험요소 현황

- 도로교통공단에 따르면 최근 5년간 이륜자동차의 교통사고 건수가 꾸준히 증가했고, 특히 코로나 이후 배달에 수요에 따른 이륜자동차 사고 비율은 더욱 증가하였다. 또한 이륜자동차는 다른 교통수단에 비해 도로 환경에 대한 위험 노출도와 부상 위험이 매우 크고 치사율이 1.6배가량 높다.
- 하지만 이륜자동차들이 주로 다니는 골목길, 이면도로, 지방도와 같은 생활도로의 위험 요소는 즉각적인 파악 및 보수가 잘 이루어지지 않고 있다.



▶ 이슈(Pain Point)

- 포트홀 등 도로 손상으로 인한 대부분의 물적사고 건수는 생활도로에서 발생한다. 또한 이로 인해 발생하는 피해보상액은 70%에 육박한다. 하지만 고속도로와 국도에 비해 투입되는 유지보수비용이 극히 적다.
- 도로이용자들의 지방도 등 생활도로 보수의 필요성에 대해 인식하고 있다. 도로 노면 손상으로 인한 사고는 판결 선례에 의해 국가가 배상하지만, 모든 보상을 받을 수 없다. 많은 도로이용자들이 도로 노면 손상으로 인한 교통사고가 발생함을 인지하고 해결할 것을 희망한다.
- 지방도로에 대한 지속적인 정비가 이루어진다면 적은 비용으로도 큰 비용 절감을 얻을 수 있을 것으로 기대한다.



1.2 분석 목표

1) 분석 목표

- 이륜차 주행 중 위험요소 데이터를 실시간 객체 탐지하고 수집한다.
- 사용자에게 탐지된 데이터를 피드백한다.
- 탐지된 위험물 데이터를 수집 및 적재하고 전송하는 웹 서비스를 구현한다.

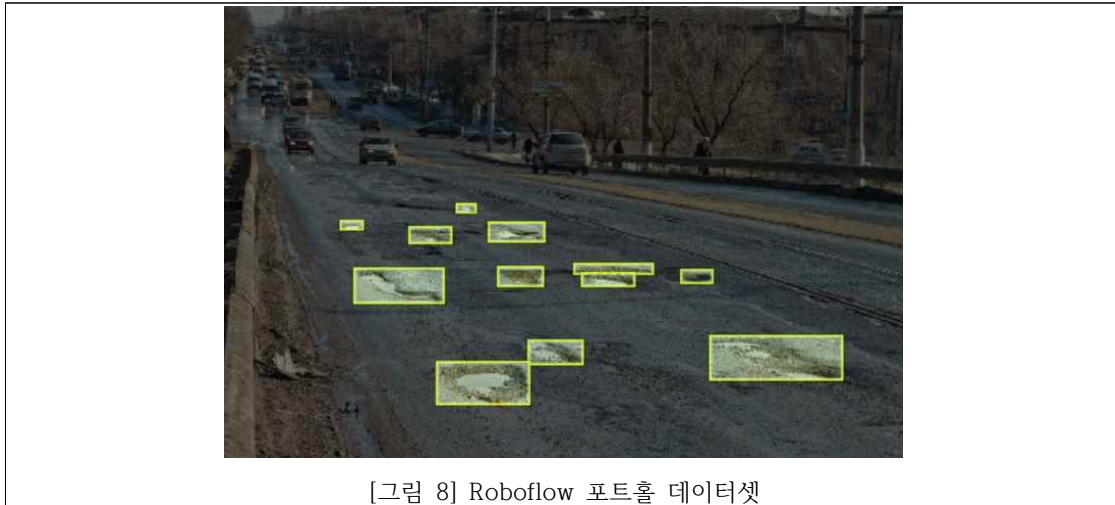
2) 데이터 정의 및 소개

▶ AI Hub에서 제작한 ‘도로 장애물 표면 인지 영상(수도권 외)’ 데이터



- ‘도로 장애물 표면 인지 영상(수도권 외)’라는 광역시, 고속도로, 국도 등 도로상의 장애물 및 도로 표면의 이상 상태 인지를 위한 영상 및 이미지 데이터셋으로 이중 6가지 카테고리 이미지 프레임을 선택한다.
- 해당 카테고리는 Person(사람), Traffic cone(라바콘), Construction signs & Parking prohibited board(공사표지판), Garbage bag & sacks(쓰레기), Filled pothole(정상수리된 포트홀), Manhole(정상도로의 맨홀)이다.
- 해당 데이터셋은 *.png형태의 원본 이미지와 *.json형태의 라벨링 이미지로 구성되어 있으며, 경계 박스(Bounding Box)의 좌표는 COCO형식이다.

▶ Roboflow에서 제작한 Public Dataset인 Pothole Dataset



- AI Hub의 포트홀 이미지는 위험요소로 보기 힘든 매우 작은 균열, 웅덩이를 포함하고 있어 Roboflow의 데이터로 대체한다. 해당 데이터셋은 포트홀이 표시된 665개의 도로 이미지이다.
- 해당 카테고리는 Pothole(포트홀)이다.
- 해당 데이터셋은 *.jpg형태의 원본 이미지와 *.txt형태의 라벨링 이미지로 구성되어 있으며, 경계 박스의 좌표는 YOLOv5 Pytorch Txt형식이다.

▶ 사용할 7개의 위험물 카테고리

- AI Hub와 Roboflow에서 수집한 데이터셋을 합하여 이륜자동차에게 위험요소가 될 수 있는 ‘위험요소’와 오탐지를 방지하기 위한 ‘오탐지 방지용’으로 두 가지의 기준에 부합하는 요소들은 선정하였으며 다음과 같다.
- 1. 위험요소(라바콘, 공사표지판, 쓰레기, 포트홀), 2. 오탐지방지용(사람, 정상수리된 포트홀, 맨홀)

2. 분석 실습

2.1 이륜차 위험물 탐지 데이터 분석

1) 데이터 유형/구조

▶ Original Data

- AI Hub와 Roboflow의 Opendataset을 활용하여 이륜차 주행 중 위험요소인 이미지와 라벨링 데이터의 구조는 다음과 같다. 데이터 용량을 축소하기 위해 손실 압축 방식을 사용, *.png 포맷을 *.jpg로 변환하였다.

데이터명	이륜자동차 안전 위험 시설물 데이터 & 도로 장애물/표면 인지 데이터	데이터 유형	이미지
데이터 영역	교통물류	데이터 출처	AI HUB
데이터 형식	.png	데이터 구축년도	2021
라벨링 유형	바운딩 박스, 세그멘테이션	데이터 구축량	42만장(수정예정)

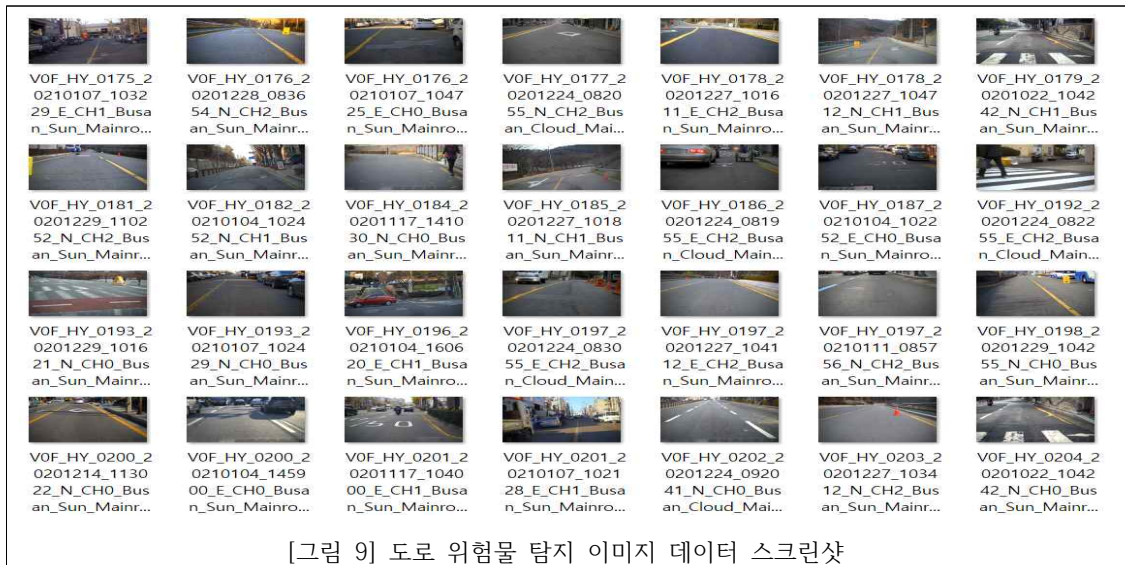
[표 1] 데이터 구조 요약

이미지 프레임			데이터 구축량	라벨링 유형
동적 객체	예측불가 동적객체	보행자	5만	바운딩박스
정적 객체	예측불가 정적객체	낙하물, 라바콘, 공사표지판, 쓰레기	30만	
노면	포트홀	포트홀	10만	
	맨홀	맨홀	20만	시맨틱 세그멘테이션
	크랙	크랙	30만	

[표 2] 데이터 카테고리

구분	이름	설명
차량구분	V0F, V1F, V2F, ... VnF	차량 및 운전자 고유번호
영상장치	HY_0002, HY_0015, ...	동영상(1분, MP4) 고유 No.
촬영일시	촬영일 : YYYY/MM/DD	_2020118
	촬영시간 : hh/mm/ss	_13958
비식별화	N / E	N : 비식별 X / E : 비식별 O
채널	CH0, CH1, ...	카메라 위치, 채널 No.
촬영지역	Seoul / Busan	수도권, 광역시 구분
날씨구분	Sun / Cloud / Rain / Fog / Snow	맑음 / 흐림 / 비 / 안개 / 눈 등
도로상태	Frontback / Highway / Kidzone / Mainroad / Industrialroads	도심(골목길), 고속도로 어린이보호구역), 국(지방)도, 항만 / 공단
촬영시간 구분	Day / Night / Sunrise / Sunset	낮 / 밤 / 일출 / 일몰 등
PNG No.	_0005	이미지 생성시 PNG 번호

[표 3] 데이터 Naming



[그림 9] 도로 위험물 탐지 이미지 데이터 스크린샷


json

```


{
  "info": {
    "description": "V0F_HY_0073_20201214_104232_N_CH0_Busan_Sun_Highway_Day_07341_BBOX_JSON_file",
    "url": "",
    "version": "1.0",
    "year": 2021,
    "contributor": "Konkuk_university",
    "date_created": "2021/05/12"
  },
  "images": {
    "file_name": "V0F_HY_0073_20201214_104232_N_CH0_Busan_Sun_Highway_Day_07341.png",
    "height": 720,
    "width": 1280,
    "id": 1
  },
  "annotations": [
    {
      "segmentation": [],
      "polyline": [],
      "image_id": 1,
      "bbox": [
        737.5,
        548.5,
        105,
        33
      ],
      "category_id": 8,
      "area": 3465,
      "is_crowd": 0,
      "id": 1
    }
  ],
  "categories": [
    {
      "id": 1,
      "name": "Animals(Dolls)"
    },
    {
      "id": 2,
      "name": "Person"
    },
    {
      "id": 3,
      "name": "Garbage bag & sacks"
    },
    {
      "id": 4,

```

바운딩박스



시맨틱 세그멘테이션(polyline)



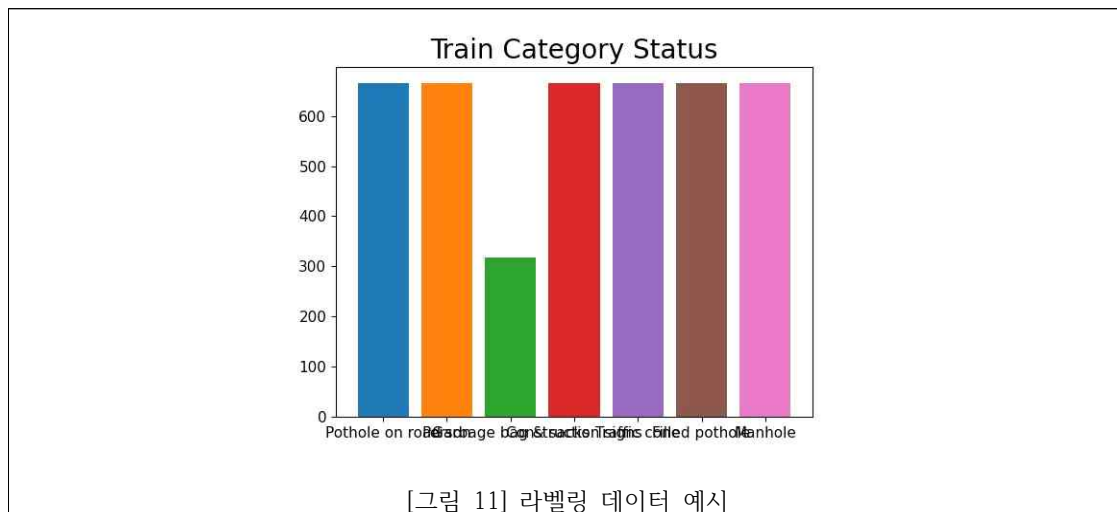
[그림 10] 라벨링 데이터 예시

2) 데이터 정제(전처리)

▶ 데이터셋 준비 (Data Setup)

○ 학습 카테고리를 균일화 시키는 과정

- 해당 과정은 사용하는 원본 데이터셋 용량 업로드 문제를 해결하기 위해 학습할 분량의 데이터만 추출하여 images, Annotation 폴더에 저장한다. 앞서 확인한 AI HUB와 roboflow의 데이터를 훈련에 사용할 수 있는 형태로 바꾸고자 한다.
- 카테고리별 사용한 이미지 개수
 - 665장: Person(사람)
 - 665장: Traffic cone(라바콘)
 - 665장: Construction signs & Parking prohibited board(공사표지판)
 - 318장: Garbage bag & sacks(쓰레기)
 - 665장: Filled pothole(정상수리된 포트홀)
 - 665장: Manhole(정상도로의 맨홀)
 - 665장: Pothole(포트홀)
- 다음과 같이 추출한 이유는 roboflow의 포트홀 이미지가 665장이므로 다른 데이터셋도 같은 개수로 맞추어 객체 탐지의 정확도를 높이하고자 한다.
- 쓰레기 카테고리의 경우, 절대적인 이미지 수가 부족하여 318장만 추출한다.



[그림 11] 라벨링 데이터 예시

▶ 데이터셋 전처리 (Data Preprocessing)

○ 라벨링 데이터 변환

- YOLOv5에서 라벨링 학습을 위해서는 YOLO방식의 라벨링 방식과, txt type 데이터가 요구되므로 json type, COCO방식의 라벨링 방식을 가진 라벨링 데이터를 전부 txt type과 YOLO방식의 라벨링 방식으로 변환한다.
- COCO 라벨링 방식(x,y,w,h) -> x: 좌상단x, y: 좌상단y, w: bounding box의 width, h: bounding box의 height

- YOLO 라벨링 방식(x,y,w,h) -> x: bounding box 중심점의 x, y: bounding box 중심점의 y, w: bounding box의 width, h: bounding box의 h

```

for file in tqdm(json_list):
    dataset = importer.ImportCoco(LABEL_PATH + file)

    # 좌표 변경을 위한 변수 지정
    bbox_x = int(dataset.df.iloc[0].ann_bbox_xmin)
    bbox_y = int(dataset.df.iloc[0].ann_bbox_ymin)
    height = int(dataset.df.iloc[0].ann_bbox_height)
    width = int(dataset.df.iloc[0].ann_bbox_width)
    img_height = int(dataset.df.iloc[0].img_height)
    img_width = int(dataset.df.iloc[0].img_width)

    # 카테고리 id, 해당하는 이미지의 이름 정보
    img_category = int(dataset.df.iloc[0].cat_id)
    img_name = dataset.df.iloc[0].img_filename[:-3]

    # 카테고리 id가 0~6안에 있어야하기 때문에 카테고리 id 재설정
    if img_category == 2:
        img_category = img_category - 1
    elif img_category == 3:
        img_category = img_category - 1
    elif img_category == 4:
        img_category = img_category - 1
    elif img_category == 5:
        img_category = img_category - 1
    elif img_category == 9:
        img_category = img_category - 4
    elif img_category == 10:
        img_category = img_category - 4

    # img_height, img_width 정보 누락 방지
    if img_height != 720:
        img_height = 720
    if img_width != 1280:
        img_width = 1280

    # COCO -> YOLO 포맷으로 바운딩 박스 좌표 기준 변경
    dw = 1.0 / img_width
    dh = 1.0 / img_height
    x_center = bbox_x + width / 2.0
    y_center = bbox_y + height / 2.0
    x = x_center * dw
    y = y_center * dh
    w = width * dw
    h = height * dh

    # SAVE_PATH가 없으면 생성
    if not os.path.exists(LABEL_SAVE_PATH):
        os.makedirs(LABEL_SAVE_PATH)

    # txt파일 이름이 없으면 생성
    if not os.path.isfile(LABEL_SAVE_PATH + '/' + img_name + 'txt'):
        f = open(LABEL_SAVE_PATH + '/' + img_name + 'txt', 'w')
        f.close()

    # 변경된 좌표값, 카테고리 id를 이용해 txt포맷의 라벨링 데이터 생성
    f = open(LABEL_SAVE_PATH + '/' + img_name + 'txt', 'a')
    f.write(str(img_category) + ' ')
    f.write(str(x) + ' ')
    f.write(str(y) + ' ')
    f.write(str(w) + ' ')
    f.write(str(h) + '\n')
    f.close()

```

100%|██████████| 3643/3643 [01:03<00:00, 57.48it/s]

[그림 12] 라벨링 데이터 변환 예시 스크린샷

○ 훈련/평가 데이터셋 생성

- 전체 데이터셋 중 9:1의 비율로 훈련/평가 데이터셋을 생성한다.
 - 전체 데이터 중 일부는 훈련 (train)에 사용하고, 나머지 일부는 훈련된 모델의 성능을 검증(val)하기 위해 사용하고자 한다. ('train_test_split')

```
# train, validation 데이터 분리
train_img_list, val_img_list = train_test_split(train_img_list,
                                                train_size=0.9,
                                                random_state=42)

print(f'Train Image 갯수: {len(train_img_list)}')
print(f'Validation Image 갯수: {len(val_img_list)}')

Train Image 갯수: 3877
Validation Image 갯수: 431
```

[그림 13] 훈련/평가 데이터셋 생성 예시 스크린샷

○ train/val 이미지 경로를 txt파일로 저장

- YOLOv5가 동작에 사용되는 yaml파일 실행을 위해 txt type으로 이미지 경로를 저장한다. ('with open(): f.write()')

```
# train/val 이미지 경로 txt파일로 저장
with open(f'{DIR_PATH}\\train.txt', 'w') as f:
    f.write('\n'.join(train_img_list) + '\n')

with open(f'{DIR_PATH}\\val.txt', 'w') as f:
    f.write('\n'.join(val_img_list) + '\n')
```

[그림 14] txt 파일 저장 예시 스크린샷

○ yaml 파일 생성

- YOLOv5 모델 학습에 필요한 yml 파일 생성하고 최신화한다.
 - names : 카테고리명
 - nc : 카테고리 개수(카테고리명 순서대로 0 ~ nc-1의 인덱스 설정됨.)
 - train : train이미지 경로(train.txt)
 - val : val이미지 경로(val.txt)

```
# yaml 생성
with open(f'{DIR_PATH}\\data.yaml', 'r') as f:
    data = yaml.full_load(f)

data['train'] = f'{DIR_PATH}\\train.txt'
data['val'] = f'{DIR_PATH}\\val.txt'

with open(f'{DIR_PATH}\\data.yaml', 'w') as f:
    yaml.dump(data, f)

print(f'data.yaml 파일 정보: {data}')

data.yaml 파일 정보: {'names': ['Pothole on road', 'Person', 'Garbage bag & sacks', 'Construction signs & Parking prohibited board', 'Traffic cone', 'Filled pothole', 'Manhole'], 'nc': 7, 'train': 'C:\\\\DataScience\\\\T3Q\\\\플랫폼 최종 제출 양식\\\\T3Q_1_PLATFORM\\\\meta_data\\\\dataset\\\\train.txt', 'val': 'C:\\\\DataScience\\\\T3Q\\\\플랫폼 최종 제출 양식\\\\T3Q_1_PLATFORM\\\\meta_data\\\\dataset\\\\val.txt'}
```

[그림 15] yaml 파일 생성 예시 스크린샷

2. 분석 실습

2.2 AI 분석모델 소개

1) AI 학습모델 분석 및 비교

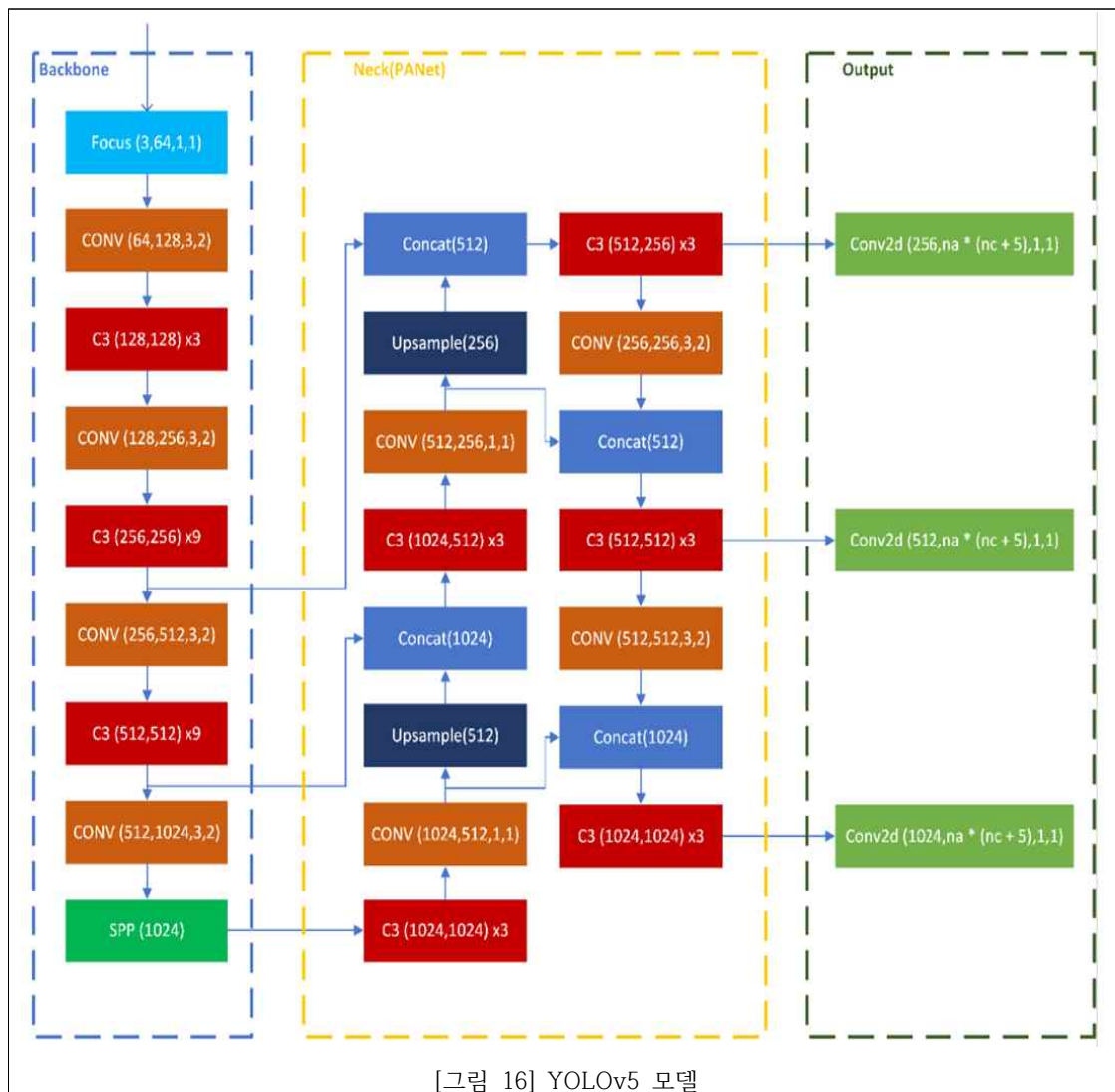
▶ 해당 AI 방법론(알고리즘) 선정 이유 기술

- 이륜자동차 위험물 탐지 분석 방법 : 지도 학습(supervised learning)의 실시간 다중 객체 탐지(Unified, Real-Time Object Detection)
- 운전 현장의 안전을 위해 입력 영상에서 위험물을 검출해야 하며, 7개 라벨로 설정된 데이터셋을 실시간 다중 객체 탐지 모델(YOLO v5)에 학습시켜 결과를 얻을 수 있다. 또한 단순히 운전 중 위험물의 존재 유무를 검출하는 것이 아니라, 영상 내 위험물의 위치 좌표를 획득하여야 한다. 해당 위치좌표와 이미지 데이터를 확인할 수 있어야 하며, 이 정보를 도로관리사업소 등 지자체에 전송하여 도로 노면의 원활한 보수가 이루어지도록 한다. 따라서 해당 작업에 높은 성능을 보이는 옴로 네트워크(YOLOv5 Network)에 학습시켜 해당 결과를 얻었고, 탐지 기술에 적용하였다.

▶ 적용하고자 하는 AI 분석 방법론(알고리즘)의 구체적 소개

○ 옴로 네트워크 (YOLOv5 Network)

- 객체 탐지(Object detection)란 이미지 내에서 물체의 위치와 그 종류를 찾는 것으로, 어떤 물체가 있는지 분류(Classification)하고 분류된 물체가 어디에 있는지 경계 박스(Bounding box)를 통해 위치를 특정하여 파악(Localization)한다.
- 객체 탐지 모델인 YOLOv5는 하나의 심층 신경망에서 경계 박스 탐색과 클래스 분류가 동시에 이뤄지는 객체 탐지 모델(1-Stage object detection)이다. 반면 R-CNN 기반 모델은 경계 박스의 추출과 해당 클래스 추측을 단계로 나누어 진행한다. 또, R-CNN은 이미지를 여러 장으로 분할하기 때문에, 한 장을 여러 장으로 탐지하는 것과 같은 효과를 낼 수 있어 정확도가 높지만 처리 속도가 느리다. 하지만 옴로 모델은 이미지 전체를 한 번에 탐색하여 클래스에 대한 맥락적 이해도가 높기 때문에 상대적으로 낮은 오탐(False-Positive)을 보인다. 또한 전처리 모델과 인공신경망을 통합하였기 때문에 탐색 속도가 기존의 R-CNN보다 6배 빠르고, 정확도가 상대적으로 낮지만 준수한 성능을 보인다. 옴로 네트워크의 전체적인 과정은 다음과 같다.
- YOLOv5는 특징(feature)를 추출하는 백본(backbone), 추출된 특징을 융합(fusion)하여 성능을 높이는 넥(neck), 특징을 경계 박스 파라미터로 변환하는 헤드(Head)와 같이 3종류 부분으로 이루어져 있다. 옴로 모델의 Backbone은 이미지로부터 Feature map을 추출하는 부분으로, CSP-Darknet를 사용한다.



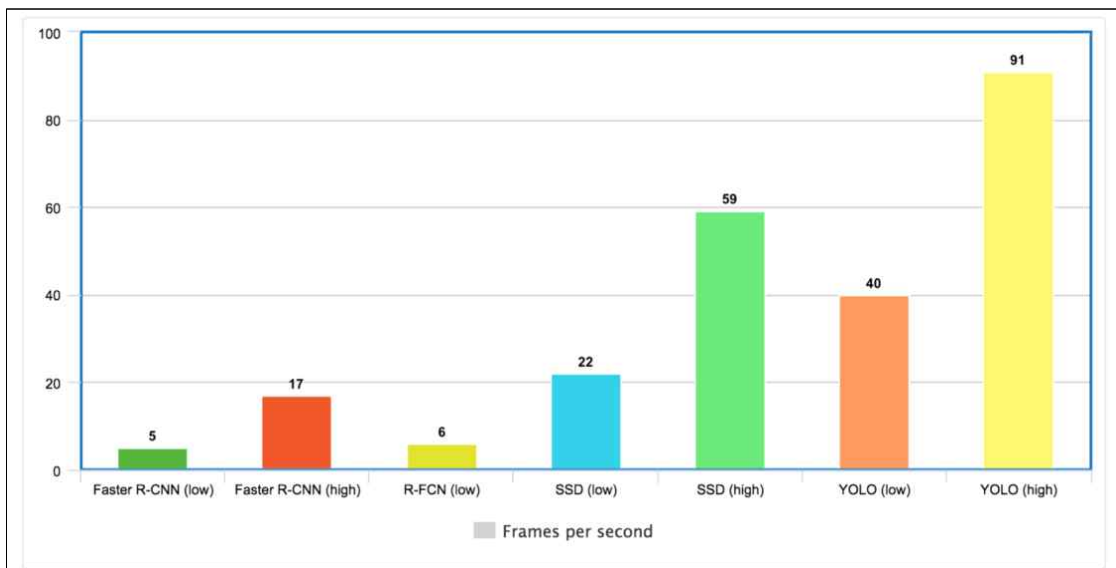
- 백본은 여러 층의 합성곱 신경망(Convolution Layer)와 풀링 레이어(Pooling Layer)를 통해 입력 이미지로부터 다양한 크기의 특징 맵(Feature Map)을 추출하는 부분이다. Convolution with Batch normalization and Leaky Relu(CBL), Cross Stage Partial(CSP), Spatial Pyramid Pooling(SPP)의 기법이 사용되었다. CBL은 컨볼루션 레이어, 배치 정규화, Leaky-relu의 활성 함수로 이루어진 블록으로 피처를 추출하는데 기본적으로 사용되는 블록이다. CSP는 특징 맵의 일부에만 합성곱 연산을 수행하고, 나머지 부분과 통합시키는 방법이다. 일부의 특징 맵만 합성곱 신경망을 통과시키기 때문에 연산량을 줄일 수 있고, 역전파 과정에서 기울기(gradient)의 흐름을 효율적으로 수행하게 되어 성능이 개선될 수 있다. SPP는 특징 맵을 다양한 크기의 필터로 풀링한 후 다시 합쳐줌으로써 성능 향상을 가져온다.
- 넥 부분에서는 다양한 크기의 특징 맵을 융합하는데 path aggregation network(PAN)을 사용하여 낮은 레벨의 피처와 높은 레벨의 피처를 섞어주어 성능을 향상시킨다.

- 헤드는 합성곱 신경망을 이용해 Neck로부터 출력된 특징 맵을 바탕으로 네트워크의 최종 출력으로 변환하고 위치를 찾는 부분으로 경계 박스 파라미터(x,y,w,h), 물체가 존재할 확률, 클래스가 존재할 확률로 변환해준다. 8 pixel 정보를 가진 작은 물체, 16 pixel 정보를 가진 중간 물체, 32 pixel 정보를 가진 큰 물체를 인식 가능한 3가지의 스케일에서 바운딩 박스를 생성하고 다시 각 스케일에서 3개의 앵커 박스를 사용한다. 즉, 총 9개의 앵커 박스가 생성된다.

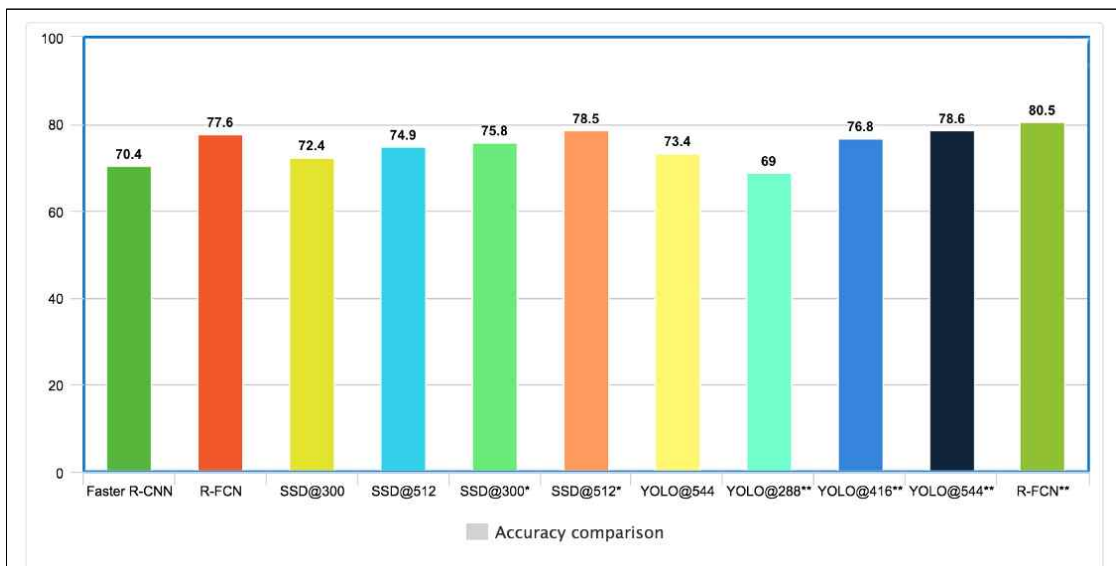
▶ 객체탐지 모델과의 비교

○ YOLOv5 모델과 타 객체탐지 모델 비교

- Faster R-CNN은 일반 이미지에서 합성곱 신경망들을 적용하여 특징 맵(feature map)을 뽑아낸 다음, 여기서 객체가 있을 것 같은 영역(RoI)을 뽑아주고 해당 영역들을 RoI Pooling을 통해 사이즈를 통일시킨 뒤, 각각 영역들을 분류를 통해 최종 이미지 라벨과 경계 박스를 추출한다. Faster R-CNN은 기본적으로 정확도가 가장 높은 편이라고 알려져 있다. 또 개선을 거듭할수록 처리속도가 상승하고 있지만, 특징 맵의 각 영역마다 k개의 앵커 박스가 생성되기 때문에 속도가 여전히 느린 편이다.
- SSD 모델은 YOLO와 같은 하나의 심층 신경망에서 경계 박스 탐색과 클래스 분류가 동시에 이뤄지는 객체 탐지 모델(1-Stage object detection)이다. YOLO에서는 이미지를 grid로 나누어서 각 영역에 대해 경계 박스를 예측했다면, SSD는 CNN pyramidal feature hierarchy를 이용해 예측한다. SSD는 YOLO에 비해 높은 정확도를 보이지만 처리속도가 낮다. 한편 SSD는 계산이 용이하다는 장점이 있다.
- 다른 객체탐지 모델과 처리속도, 정확도, 메모리 등을 비교하였을 때 본 프로젝트는 YOLO 모델이 가장 적합하다고 판단한다. 정확도가 상대적으로 낮지만 여러 기법들을 통해 극복이 가능하고, 로컬 환경에서 적은 메모리로 빠른 성능을 보여줄 수 있기 때문이다.



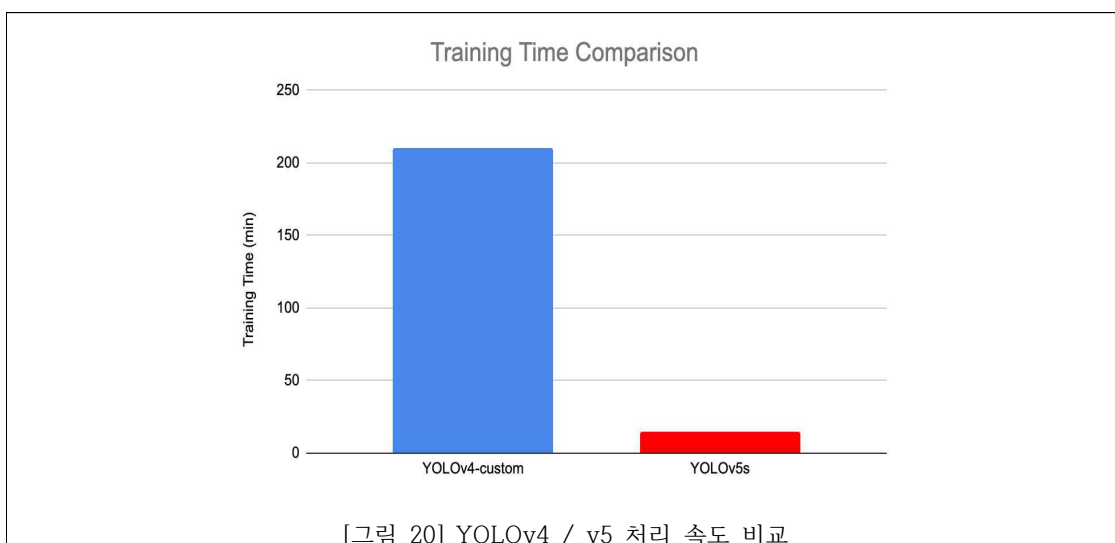
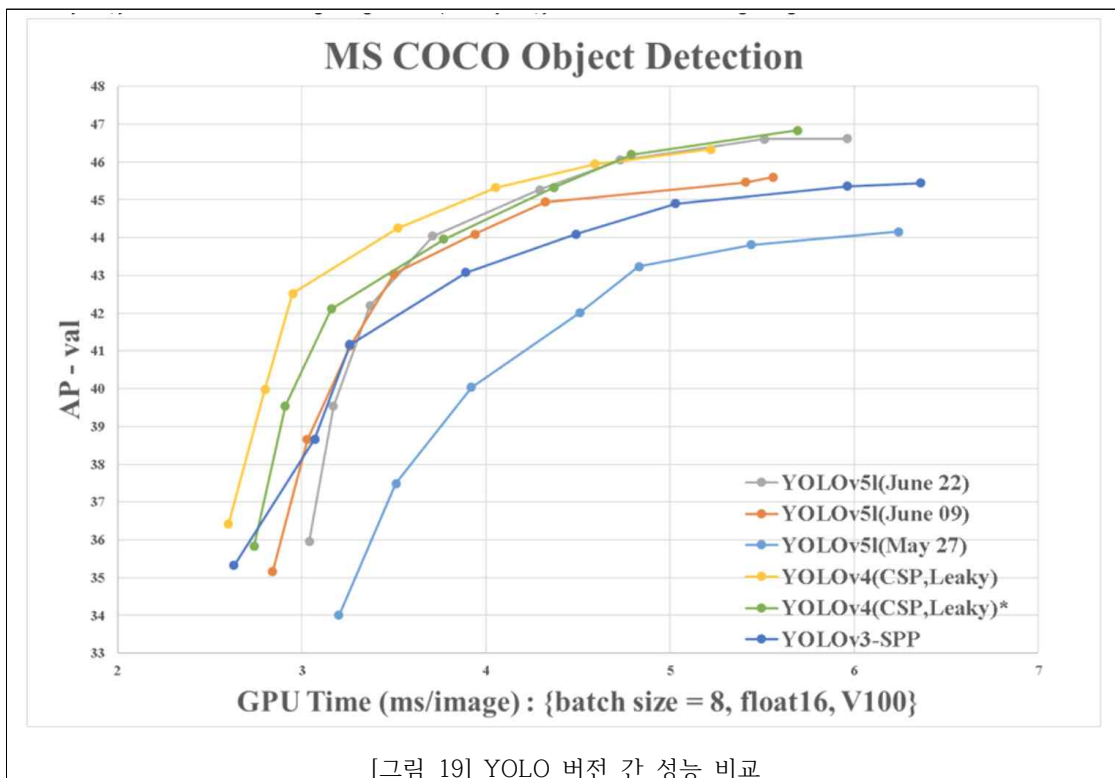
[그림 17] 객체 탐지 모델과 YOLOv5 프레임 당 속도 비교



[그림 18] 객체 탐지 모델과 YOLOv5 정확도 비교

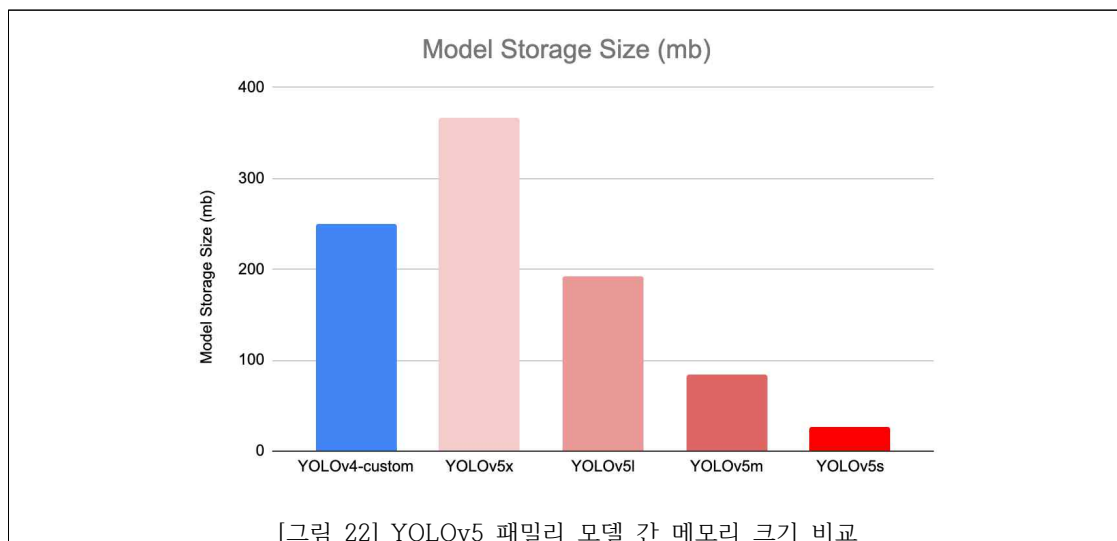
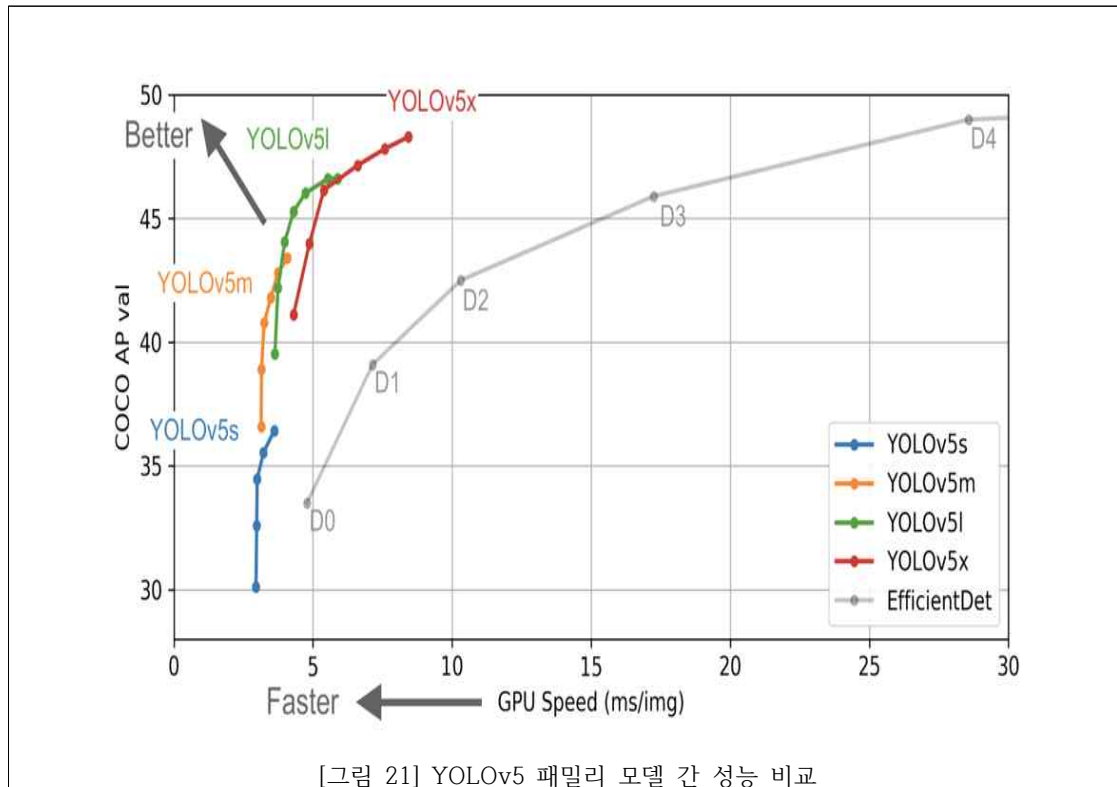
○ YOLOv5 버전 간 비교

- YOLO v1과 v2는 옴니 모델의 초기 버전이기 때문에 다른 버전들에 비해 전체적인 성능이 낮다. 크기가 작거나 거리가 먼 객체, 경계 박스가 겹쳐진 객체를 탐지하기 어려운 것은 옴니 모델의 단점 중 하나이지만, 초기 버전은 그 단점이 더 부각된다.
- YOLO v4와 v5는 v3의 백본을 기반으로 발전한 모델이다. v3의 발전형인 만큼 v4와 v5는 성능과 처리속도가 비슷하지만 v5는 메모리 사용량이 월등히 낮다. tensorflow를 기반으로 한다면 메모리 사용량을 감안해서라도 v4를 사용할 수도 있다. 하지만 본 프로젝트는 pytorch를 기반으로 진행하며, 낮은 메모리 대비 높은 처리속도와 성능을 지향하기 때문에 v5를 선택한다.



○ YOLOv5 패밀리 모델 간 비교

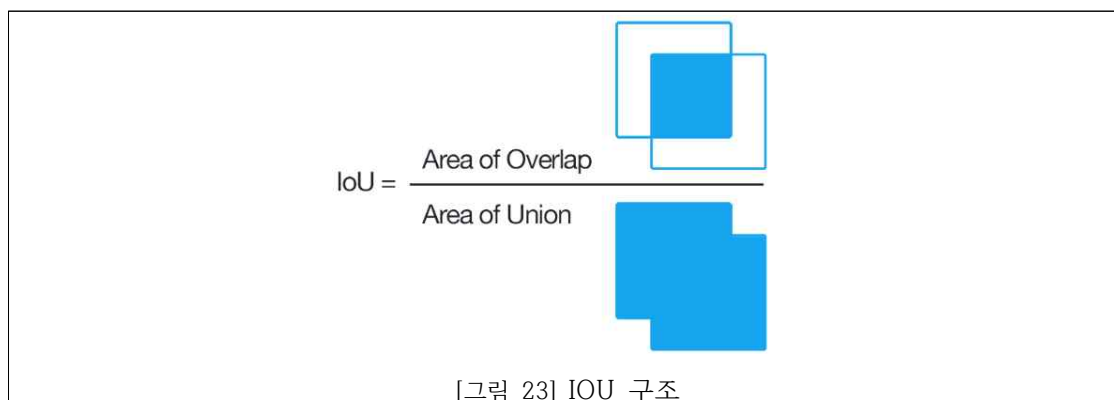
- YOLOv5은 구동 환경에 따라 적합한 모델을 다르게 적용할 수 있는데 크기별로 nano, small, medium, large, extra Large로 나뉘는 패밀리 모델이다. v5m부터는 메모리 문제로 인해 사실상 GPU환경에서 학습할 수 있다. 패밀리 모델이 클수록 정확도는 높지만 초당 프레임(fps)는 낮다. 본 프로젝트는 로컬 기반에서 빠른 처리속도(fps)와 동시에 메모리 크기가 낮은 v5s를 선택한다.



2) 학습모델 평가 지표

○ IOU(Intersection over Union)

- 옴로 네트워크에서 성능 지표로는 IOU(Intersection over Union), 재현율 (Recall), 정밀도(Precision), P-curve, R-curve, PR-curve, F1-curve. mAP(mean Average Precision) 등이 쓰인다. 그 중 IOU는 mAP를 이해하는데 IoU가 필요하며, 재현율과 정밀도를 계산할 때 물체를 검출했을 때 ‘옳게 검출되었다’ / ‘옳게 검출되지 않았다’를 구분하는 기준이다. 두 경계 박스가 있을 때, 겹친 영역의 넓이 / 합친 영역의 넓이를 나타낸다. 두 경계 박스가 많은 영역이 겹칠수록 큰 값이 나오며, 최댓값은 1이다. 해당 값이 높을 수록 예측이 잘되었다고 말할 수 있다. 설정한 threshold보다 IoU값이 높거나 같으면 해당 predicted bounding box는 positive, 즉 객체가 존재한다고 label된다. 반대로 임계값(threshold)보다 낮다면 해당 bounding box는 negative로 label된다.



○ AP (Average Precision)

- Precision-Recall curve는 하나의 알고리즘 성능을 파악하고, (보통 이러한 curve는 threshold 등을 결정하는 데에 사용하는 듯), 서로 다른 알고리즘의 성능을 정량적으로 비교하는 것에 불편함이 있다. 따라서 알고리즘 간 성능 비교를 용이하게 하기 위해 precision-recall curve를 하나로 요약한 값인 AP를 사용한다. 이때 AP는 각 임계값(threshold)에서의 정밀도(precision)의 가중합을 의미하며, 이때의 가중치는 이전 임계값과 현재의 임계값에서의 재현율(recall) 값의 차이이다. 결국에 AP는 정밀도의 평균(Average)를 대표하는 값으로 볼 수 있다. 수식으로 표현하면 다음과 같다.

$$AP = \sum_{k=0}^{k=n-1} [Recalls(k) - Recalls(k+1)] * Precisions(k)$$

$Recalls(n) = 0, Precisions(n) = 1$
 $n = \text{Number of thresholds.}$

[그림 24] AP 수식

○ **mAP (mean Average Precision)**

- AP는 여러 알고리즘 간의 성능 비교가 가능한 알고리즘 성능 평가 지표이다. 객체 탐지에서는 하나의 객체(binary class)만을 탐지하는 것이 아닌, 여러 객체를 탐지하는 것이므로(multi-class) AP를 모든 클래스에 대해 계산할 필요가 있다. 이때 mAP는 AP를 클래스에 대해 평균한 결과이다. 각 클래스별로 AP를 계산하여 해당 클래스에 대해 정밀도 값의 정확도를 구한 후, 이를 모든 클래스에 대해 평균을 내어 해당 모델이 모든 클래스에 대해 어느 정도의 탐지 정확도를 기록하는지 판단할 수 있다. 다음은 mAP의 수식이다. k 는 클래스에 대한 인덱스이고, n 은 이미지 셋 내의 전체 객체 클래스의 개수이다.

$$mAP = \frac{1}{n} \sum_{k=1}^{k=n} AP_k$$

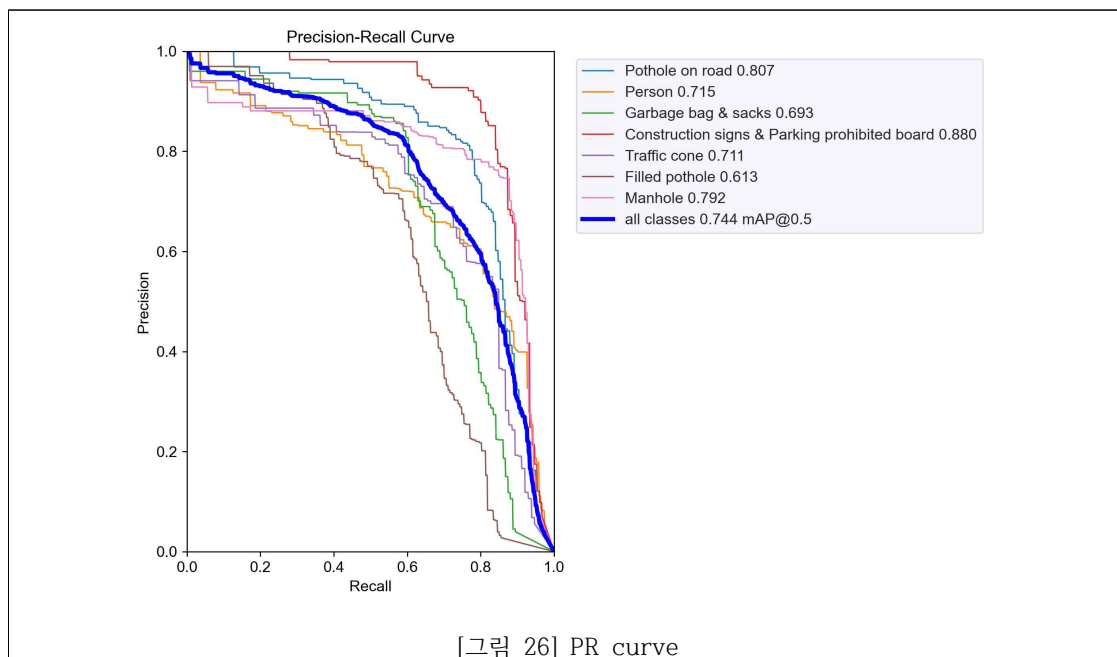
$AP_k = \text{the AP of class } k$
 $n = \text{the number of classes}$

[그림 25] mAP 수식

3) 이론차 위험물 탐지 모델 평가

▶ PR-curve(Precision-Recall Curve)

- 정밀도/재현율 곡선(PR-curve)은 오브젝트가 특정 클래스일 확률(confidence score)을 조정하면서 얻은 재현율(Recall) 값의 변화에 따른 정밀도(Precision)을 나타낸 곡선으로 모델(Object detector)의 성능을 평가하는 방법이다. 일반적으로 PR 곡선의 면적(AOU, Area under curve) 값으로 계산된다.
- PR-curve의 면적 값이 클수록 모델 성능이 높은 것으로 평가할 수 있는데, Construction signs & Parking prohibited board가 0.880으로 가장 좋은 결과를 보이고 있다. 한편, Filled potHole은 0.613으로 가장 낮은 성능으로 평가된다. 하지만 모든 클래스에 대한 평균(mAP)는 0.744로 준수한 모델 성능으로 평가된다.
- 실제 오브젝트를 모두 찾아내기 위해 오브젝트 수를 많이 탐지(detect)하는 경우, 일반적으로 PR-curve가 높은 정밀도로 시작하지만 재현율이 증가함에 따라 감소한다. 본 프로젝트의 PR-curve 또한 재현율이 증가함에 따라 정밀도가 감소하므로 적절한 성능의 모델이라고 할 수 있다.

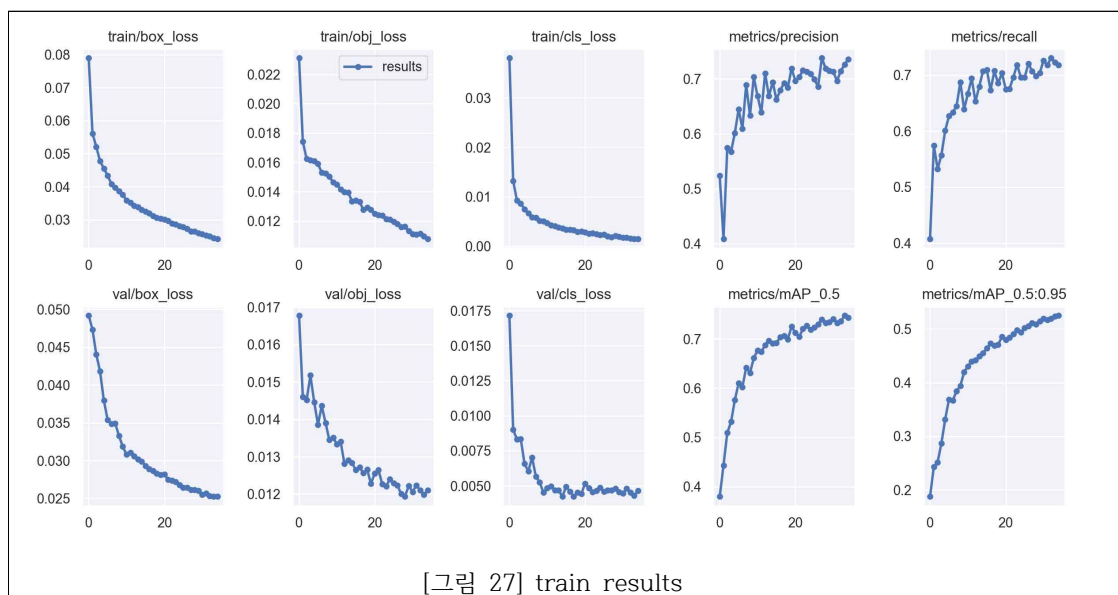


▶ train results 평가지표 분석

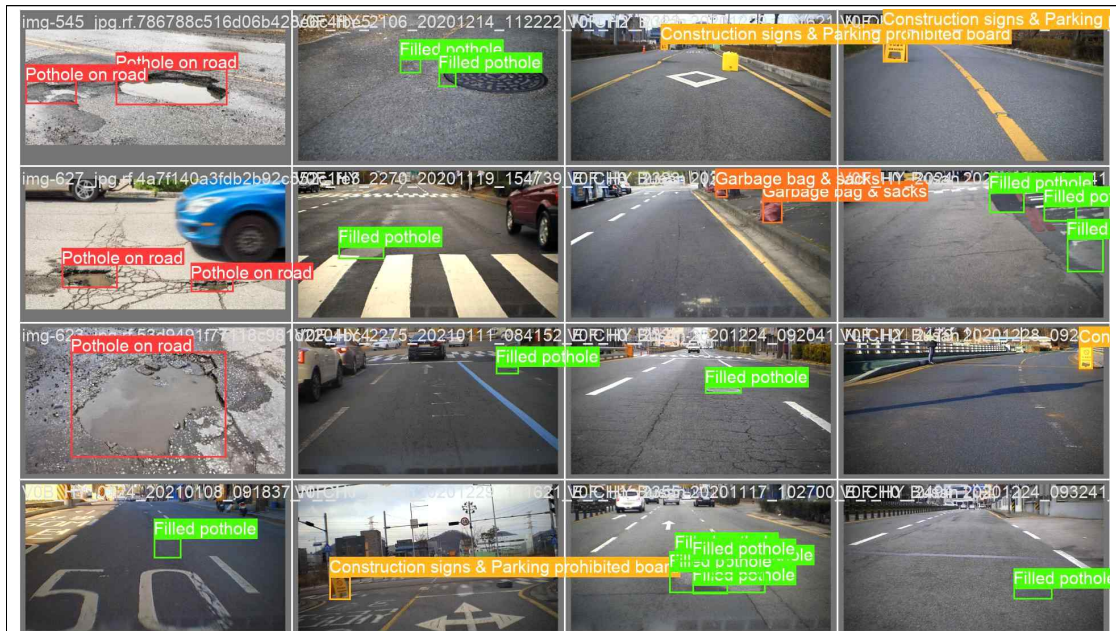
- train results는 본 프로젝트의 커스텀 데이터를 YOLOv5 모델로 학습하고 반환한 평가 지표이다. YOLOv5 커스텀 모델 학습이 얼마나 잘 이루어졌는지, 새로운 데이터를 입력했을 때 잘 기능할 수 있는지를 평가하는 지표들이다. train/box_loss, val/box_loss, train/obj_loss, val/obj_loss, train/cls_loss, val/cls_loss, metrics/precision, metrics/mAP_0.5, metrics/recall, metrics/mAP_0.5:0.95 지표로 구성한다.
- train/box_loss, val/box_loss, train/obj_loss, val/obj_loss, train/cls_loss,

val/cls_loss은 훈련 데이터와 평가 데이터의 경계 박스, 객체 탐지, 클래스 분류의 결과가 과적합이 아님을 보여준다.

- metrics/mAP_0.5는 IOU를 0.5로 설정하여 모델 성능을 평가할 경우, 원래 정답 박스 (Ground true Box)와 절반 이상만 겹치는 예측을 하면 맞는 걸로 평가했을 때 나타내는 지표이다. 즉 절반만 겹치던, 완전히 동일하게 겹치던 모두 옳게 예측한 것으로 인정한다.
- metrics/mAP_0.5:95는 IOU가 0.95 정도 일치해야 한다. 즉, 정답 박스가 거의 동일한 수준으로 예측되어야 한다. 이 경우는 상대적으로 IOU가 0.5일때 보다 모델 성능 평가가 떨어지게 되는 두 지표를 종합하였을 때 준수한 모델 성능으로 평가한다.



- 아래의 val_batch(0,1,2)_labels.jpg와 val_batch(0,1,2)_pred.jpg는 기존에 설정한 바운딩 박스와 실제 탐지를 실행했을 때의 경계 박스 예측 결과를 보여준다. 즉 label과 pred가 일치할수록 모델의 정확도가 높다.



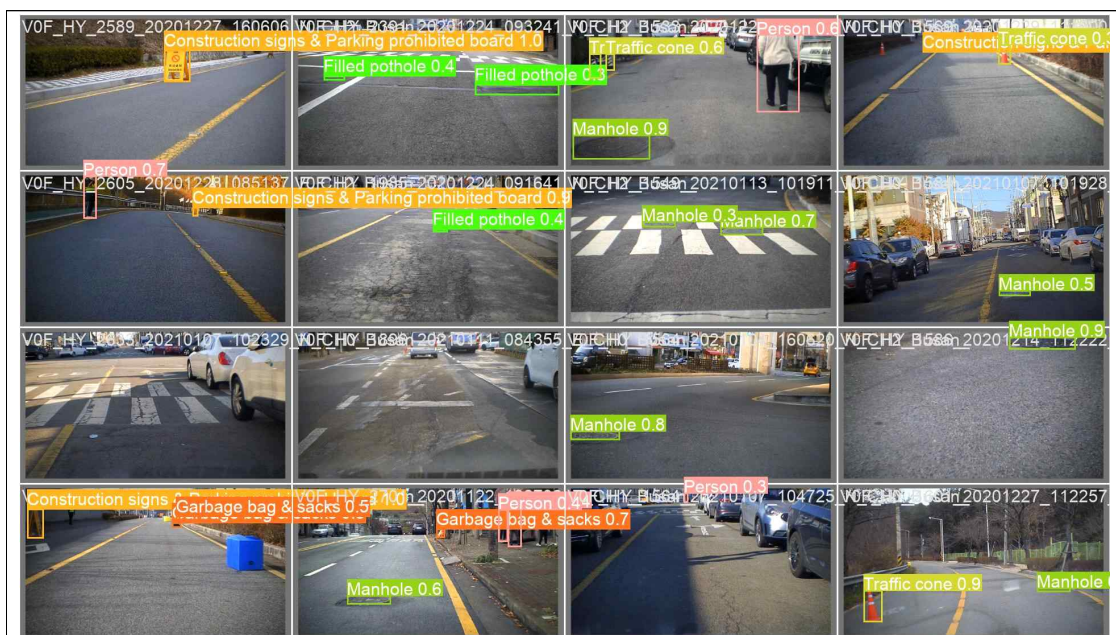
[그림 28] val_batch0_labels.jpg



[그림 29] val_batch0_pred.jpg



[그림 30] val_batch1_labels.jpg



[그림 31] val_batch1_pred.jpg



[그림 32] val_batch2_labels.jpg



[그림 33] val_batch2_labels.jpg

2.3 분석 체험

1) 필요 SW, 패키지

- YOLOv5를 통한 이륜차 위험물 탐지 서비스의 추론은 다음과 같은 패키지를 필요로 한다.

	라이브러리명	버전
1	matplotlib	3.2.2
2	numpy	1.18.5
3	pandas	1.1.4
4	seaborn	0.11.0
5	opencv-python	4.1.1
6	Pillow	7.1.2
7	PyYAML	5.3.1
8	requests	2.23.0
9	scipy	1.4.1
10	torch	1.7.0
11	torchvision	0.8.1
12	tqdm	4.64.0
13	tensorboard	2.4.1

2) 추론 및 추론방법

▶ 요약

- 훈련시킨 모델을 직접 사용해보고자 한다. 잘 훈련된 모델이라면 우리가 촬영한 이미지나 동영상에 주어진 이미지나, 동영상의 카테고리를 bounding box로 detecting할 것이다. 그 과정을 진행한다. 그 과정은 다음과 같다.
 - 이미지 또는 동영상 준비
 - YOLOv5 모델에 위에서 학습한 모델 가중치를 설정하여 테스트
 - 결과 확인

▶ 이미지 또는 동영상 준비

- 예제에서는 미리 준비해둔 이미지와 동영상 파일을 대신 받아 추론을 진행해보고자 한다.
 - 실제 사용자가 자신의 이미지 또는 동영상을 만들어 실행해볼 수도 있다.

- 이미지, 동영상 생성법

1. 직접 촬영한 이미지 또는 동영상을 저장한다.
2. 지원 이미지 포맷: 'bmp', 'dng', 'jpeg', 'jpg', 'mpo', 'png', 'tif', 'tiff', 'webp', 'pfm'
3. 지원 동영상 포맷: 'asf', 'avi', 'gif', 'm4v', 'mkv', 'mov', 'mp4', 'mpeg', 'mpg', 'ts', 'wmv'
3. bounding box로 객체 탐지할 카테고리로는 다음과 같은 목록이 있다.
 - Person(사람)
 - Traffic cone(라바콘)
 - Construction signs & Parking prohibited board(공사표지판)
 - Garbage bag & sacks(쓰레기)
 - Filled pothole(정상수리된 포트홀)
 - Manhole(정상도로의 맨홀)
 - Pothole(포트홀)

```
# 추론 데이터셋 준비 / test_dataset.zip 파일 압축 풀기
zip_source_path = './test_dataset.zip'
zip_target_path = './meta_data'

extract_zip_file = zipfile.ZipFile(zip_source_path)
extract_zip_file.extractall(zip_target_path)

extract_zip_file.close()
```

[그림 34] 추론 데이터셋 준비 예시

▶ 추론 실행

- detect.py의 run 함수를 임포트 하여 사용한다.
- weights : 가장 잘 학습된 가중치 best.pt를 사용한다.
- conf-thres : cofidence score가 0.3 이상인 detecting box만 시각화(0~1까지 설정 가능)
- source: 테스트할 이미지나 동영상 경로
- line_thickness: bounding box의 line 두께를 설정한다.
- project: detecting 결과가 저장되는 경로
- view_img: 추론 결과 시각화 여부
- 더 많은 parameter설명은 model\yolov5\detect.py line 54 run()함수 참고
- 프레임마다 인식되는 카테고리명과 해당 카테고리 갯수를 확인할 수 있다. 결과 이미지, 동영상은 지정한 경로 안에 저장된다.

```
# 테스트할 이미지 또는 영상 source 경로 설정
source = r'..\meta_data\test_dataset\V0F_HY_1505_20201227_101811_N_CH2_Busan_Sun_Mainroad_Day_51810.png'
```

[그림 35] 추론 실행 예시

▶ 추론 실행

- bounding box와 conf_score가 시각화되어 표시된다.

```
# 추론 결과: runs\detect 폴더 안에 저장됨
run(weights=r'..\model\yolov5\runs\train\train_results\weights\best.pt',
     source=source,
     conf_thres=0.3,
     line_thickness=2,
     project=r'..\meta_data\test_result',
     view_img=True)
```

YOLOv5 2022-12-4 Python-3.8.13 torch-1.13.0+cpu CPU

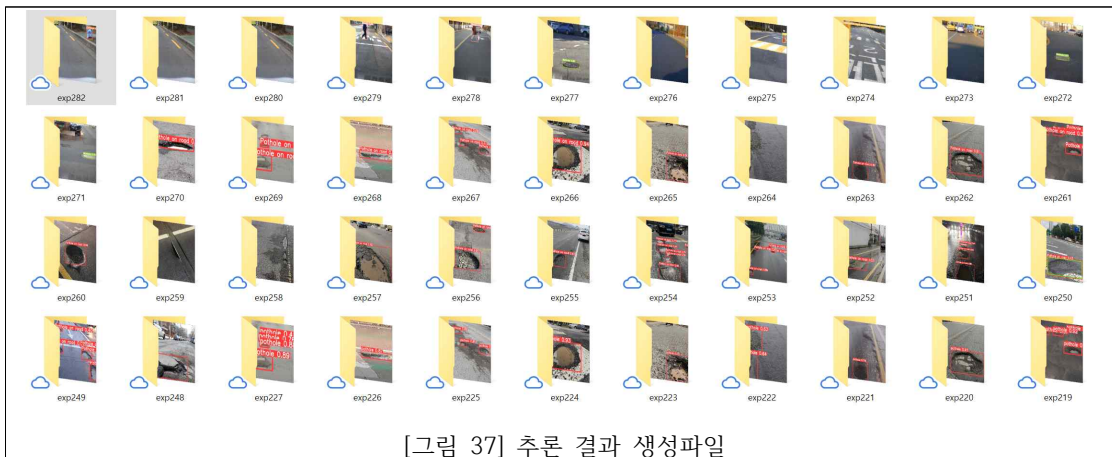
Fusing layers...

YOLOv5s summary: 157 layers, 7029004 parameters, 0 gradients, 15.8 GFLOPs

image 1/1 C:\DataScience\T3Q\ \T3Q_1_PLATFORM\meta_data\test_dataset\V0F_HY_1505_20201227_101811_N_CH
2_Busan_Sun_Mainroad_Day_51810.png: 384x640 1 Garbage bag & sacks, 98.9ms

Results saved to meta_data\test_result\exp3

[그림 36] 추론 결과 예시

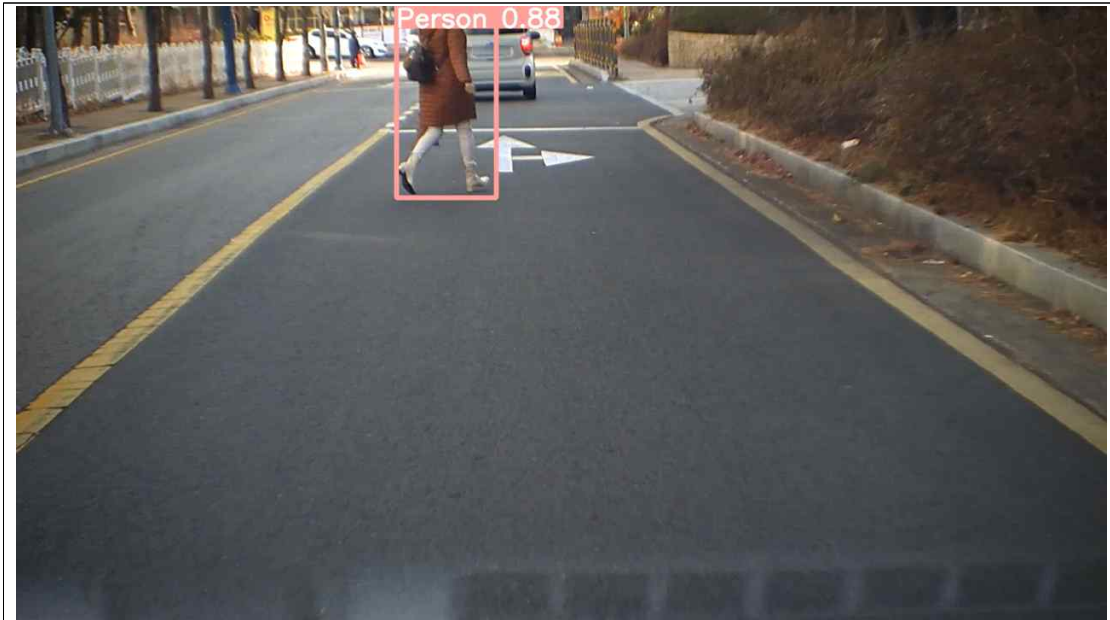


[그림 37] 추론 결과 생성파일

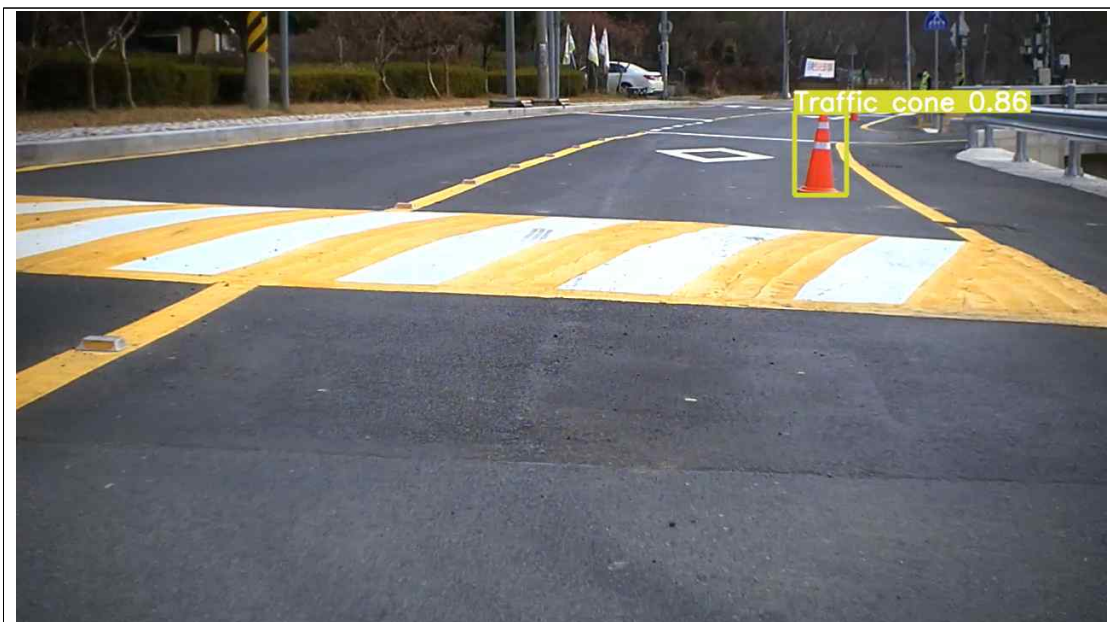
3) 프로젝트 결과 평가

▶ 이미지 탐지 결과

◦ 사람, 라바콘, 공사표지판, 쓰레기, 정상도로의 맨홀, 포트홀 카테고리 에 대한 YOLOv5 커스텀 데이터 모델 학습의 예측 결과 예시이다.



[그림 38] 사람 예측 결과



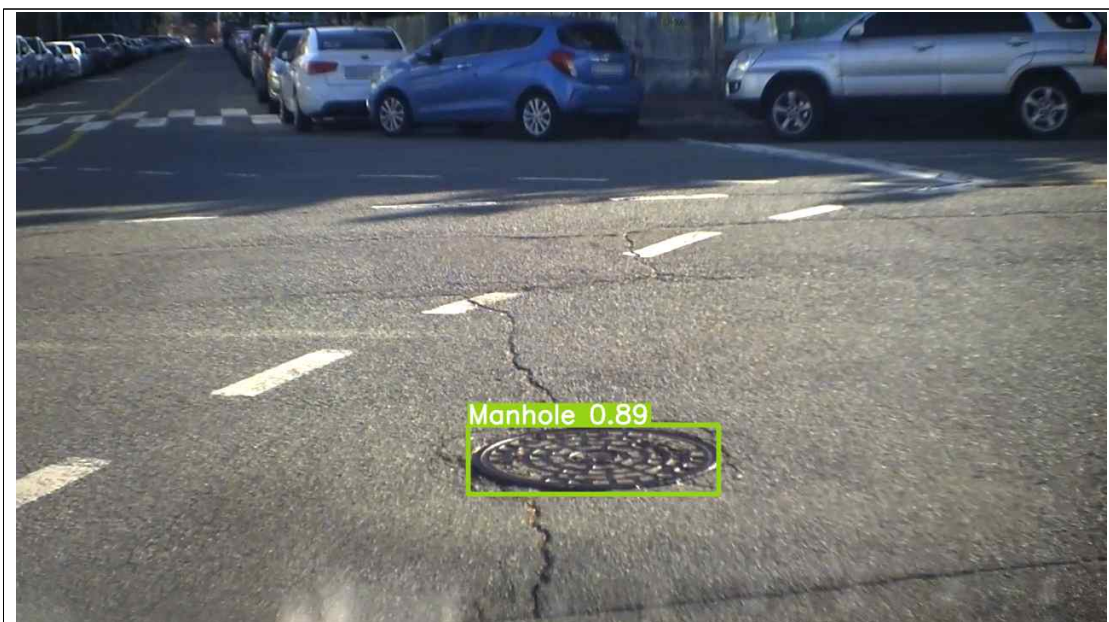
[그림 39] 라바콘 예측 결과



[그림 40] 공사표지판 예측 결과



[그림 41] 쓰레기 예측 결과



[그림 42] 맨홀 예측 결과



[그림 43] 포트홀 예측 결과

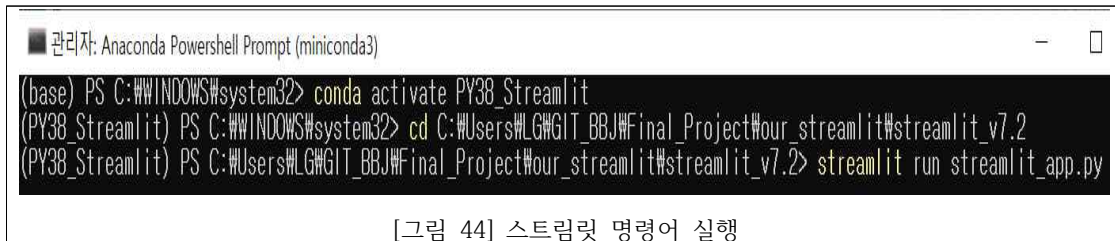
▶ 스트림릿(Streamlit) 시연

○ 스트림릿(Streamlit)

- 스트림릿은 빅데이터와 머신러닝, 딥러닝 등 AI 모델을 배포할 수 있는 파이썬 기반의 웹 어플리케이션이다. 웹 개발에 대한 배경지식이 없어도 쉽게 접근이 가능하고, 간결하고 명확한 API가 특징이다. 또 전달력이 좋아 탐색적 데이터 분석(EDA) 결과를 공유하는데 적합하다.

○ 필요 패키지 설치 및 실행

- 아나콘다 파워셸 프롬프트(Anaconda Powershell Prompt)를 관리자 권한으로 실행하여 새로운 가상환경을 생성한다. (conda create -n 가상환경이름 python=3.8)
- cd 명령어를 통해 지정경로로 이동한다.
- pip install -r requirements.txt 등의 방법을 통해 필요한 패키지를 다운로드 한다.
- cmd 명령어 창에 streamlit run 파일이름.py를 입력하여 실행한다.



[그림 44] 스트림릿 명령어 실행

- '실시간 객체 인식을 통한 이륜차 위험물 탐지 서비스'의 스트림릿 웹 구현에는 다음과 같은 패키지가 필요하다.

```
altair==4.2.0
asttokens==2.2.0
attrs==22.1.0
backcall==0.2.0
backports.zoneinfo==0.2.1
blinker==1.5
cachetools==5.2.0
certifi @ file:///C:/b/abs_ac29jvt43w/croot/certifi_1665076682579/work/certifi
charset-normalizer==2.1.1
click==8.1.3
colorama==0.4.6
commonmark==0.9.1
contourpy==1.0.6
cycler==0.11.0
decorator==5.1.1
entrypoints==0.4
executing==1.2.0
```



```
fonttools==4.38.0
geographiclib==2.0
geopy==2.3.0
gitdb==4.0.10
GitPython==3.1.29
idna==3.4
importlib-metadata==5.1.0
importlib-resources==5.10.0
ipython==8.7.0
jedi==0.18.2
Jinja2==3.1.2
jsonschema==4.17.3
kiwisolver==1.4.4
MarkupSafe==2.1.1
matplotlib==3.6.2
matplotlib-inline==0.1.6
numpy==1.23.5
opencv-python==4.6.0.66
packaging==21.3
pandas==1.5.2
parso==0.8.3
pickleshare==0.7.5
Pillow==9.3.0
pkgutil_resolve_name==1.3.10
prompt-toolkit==3.0.33
protobuf==3.20.3
psutil==5.9.4
pure-eval==0.2.2
pyarrow==10.0.1
pydeck==0.8.0
Pygments==2.13.0
Pympler==1.0.1
pyparsing==3.0.9
pyrsistent==0.19.2
python-dateutil==2.8.2
python-decouple==3.6
pytz==2022.6
pytz-deprecation-shim==0.1.0.post0
PyYAML==6.0
requests==2.28.1
```

```
rich==12.6.0
seaborn==0.12.1
semver==2.13.0
six==1.16.0
smmap==5.0.0
stack-data==0.6.2
streamlit==1.15.2
streamlit-aggrid==0.3.3
toml==0.10.2
toolz==0.12.0
torch==1.13.0
torchvision==0.14.0
tornado==6.2
tqdm==4.64.1
traitlets==5.6.0
typing_extensions==4.4.0
tzdata==2022.7
tzlocal==4.2
urllib3==1.26.13
validators==0.20.0
watchdog==2.1.9
wcwidth==0.2.5
wincertstore==0.2
zipp==3.11.0
```

[표 4] requirements.txt

○ 회원가입, 로그인 기능

- 무분별한 데이터 등록을 방지하고, 향후 사용자가 업로드한 데이터를 확인할 수 있도록 회원가입과 로그인 기능을 구현한다. 아이디와 비밀번호를 입력되면 database.db에 저장한다.

WO-T3Q1

MENU

회원가입

새 계정을 만듭니다

유저명을 입력해주세요

ABC123

비밀번호를 입력해주세요

.....

signUp

Watch Out 서비스를 이용해주셔서 감사합니다.

로그인 화면에서 로그인 해주세요.

[그림 45] 스트림릿 회원가입 화면 예시

- 가입한 아이디와 비밀번호를 입력하여 로그인한다. 회원가입하지 않은 아이디는 로그인을 거부하는 메시지를 출력한다.

WO-T3Q1

MENU

로그인

로그인 해주세요

ID를 입력해주세요

ABC123123

비밀번호를 입력해주세요

.....

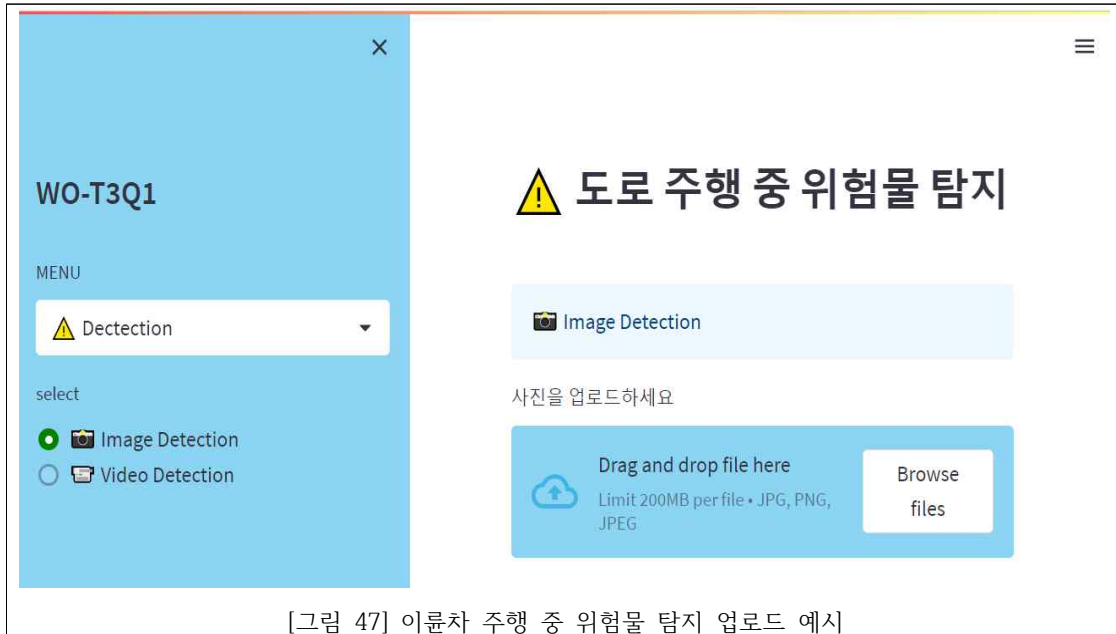
Login

사용자 이름이나 비밀번호가 잘못되었습니다.

[그림 46] 스트림릿 로그인 화면 예시

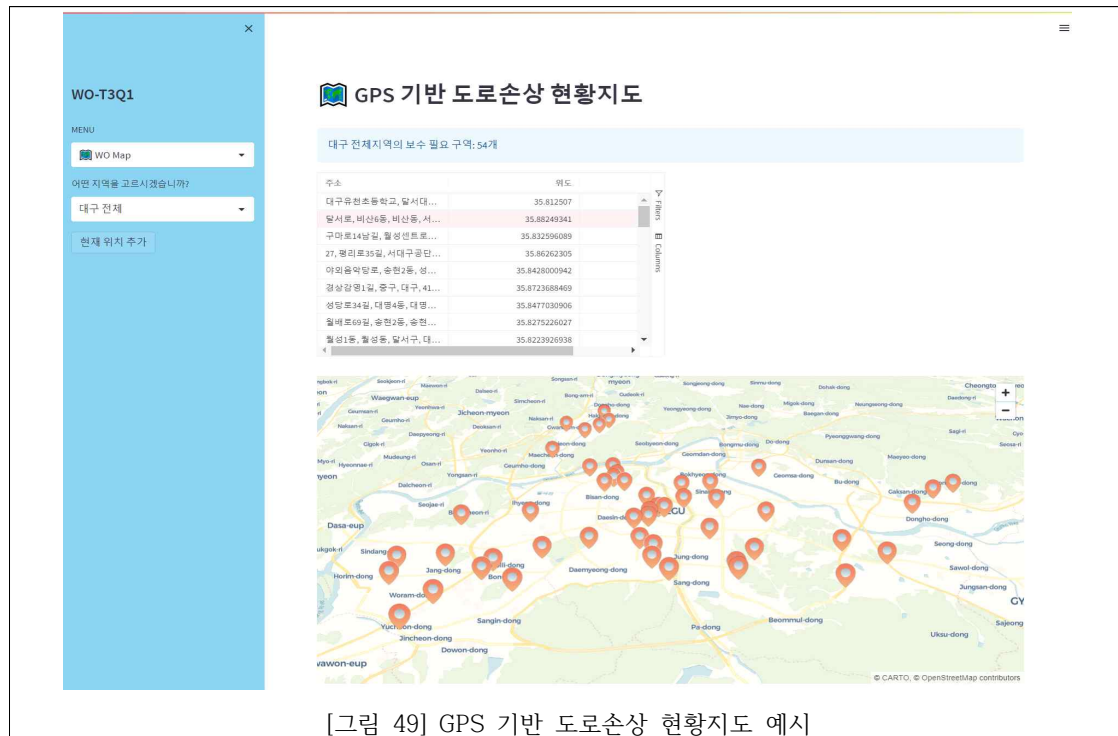
○ 이륜차 위험물 이미지, 영상 탐지

- 이륜차 주행 중 수집된 데이터를 웹 서비스에 적재한다. 적재 후 사진과 영상 데이터를 YOLOv5 모델로 객체 탐지한다. 그 후 탐지된 데이터를 사용자에게 출력한다.



○ GPS 기반 도로손상 현황지도

- 주행 중 탐지된 위험물 이미지 데이터의 위치정보를 수집하여 웹 서비스에 적재한다. 등록된 위치정보는 지도의 GPS 아이콘으로 표시되어 사용자에게 손상(포트홀 등)된 도로의 위치와 주소명, 위도, 경도를 확인할 수 있도록 한다. 또한 현재 웹 서비스에 등록된 현재 손상된 도로의 수를 출력한다.



- 수집된 위험물 데이터의 주소명, 위도, 경도를 클릭하면 해당하는 위치로 이동하여 사용자에게 보여준다. 또한 해당 위치에서 탐지한 이미지 데이터를 사용자에게 출력한다.
- 좌측 사이드바의 지역 선택 카테고리를 클릭하여 해당 지역(구)에 분포한 도로 손상 여부를 출력하고 해당 위치로 이동한다.

