

---

카메라를 활용한 도  
로 위 장애물 탐지.

---

*Computer Vision*



**1. 주제**

**2. Model**

**3. Data review**

**4. Code review**

**5. 결과물**

**6. 앞으로의 계획**

## 1. 주제 소개

시선뉴스 2020.09.21.

도로 위 묵숨 위협하는 낙하물 '판스프링'! 트럭, 안전상 문제 없...

빠른 속도로 달리는 도로에서 도로위 낙하물은 언제 어떤 사고를 유발할지 모른다.  
특히 낙하물의 종류에 따라 차량에 크고 작은 파손을 야기하는 것은 물론 심지어...



연합뉴스 2020.10.12 네이버뉴스

최근 5년간 고속도로 차량 낙하물 126만개...낙하물 사고 217건

'화물차 판스프링·페타이어 등 사고 위험성 높아' '도로위 흉기'로 불리는 화물차 판스프링이나 타이어 파편 등 운행 중인 차량에서 떨어지는 차량 낙하물이 최근 ...



KBS 2017.10.20. 네이버뉴스

참사 부른 고속도로 낙하물 '도로위 시한폭탄'

특히 밤길 운전은 도로 위 낙하물을 발견하기 어려워 사고 위험이 더욱 높습니다.  
일본 경찰청 집계 결과, 지난해까지 4년 동안 142건의 낙하물 관련 사고가 일어...



충청타임즈 2020.10.28.

충북지역 '도로위 시한폭탄' 낙하물 여전

도로 위 시한폭탄, '낙하물'이 인명을 위협하고 있다. 27일 충북지방경찰청에 따르면 최근 5년(2015~2019년)간 도내에서 발생한 낙하물(적재물 추락·도로방치물) 원인 교통사고는 19건이다. 사고로 다친 인원만 38명에...



## 기업이 원하는 자격요건.

### 자격요건

- 컴퓨터비전/딥러닝/머신러닝 이론의 전반적인 지식 보유
- 딥러닝 프레임워크(PyTorch, TensorFlow 등) 기반 컴퓨터 비전 연구개발 경험
- 프로그래밍(Python, C/C++ 등) 능력
- AI관련 최신 논문의 이해, 구현 및 활용 능력

### 주요업무

#### 1. 딥러닝/머신러닝 모델 개발

- Classification(분류) / Object Detection(객체 검출)
- 데이터 셋 설계 및 구축 / 데이터 셋 통합 / Semi-Supervised Learning
- Image Augmentation / Auto-Labeling

### 자격요건

- 컴퓨터 비전 관련 프로젝트 경험 있으신 분
- 관련 학과 학사 졸업 이상이신 분
- Machine learning, Deep Learning 개발 경험
- Pytorch, Tensorflow 경험 있으신 분



### 3. Data review



수도권

#객체 검출

#시맨틱 세그멘테이션

#주행 중 이상 상태 인식

## 도로장애물/표면 인지 영상(수도권)

분야

교통물류

유형

이미지

갱신년월 : 2023-05

구축년도 : 2020

조회수 : 2,973

다운로드 : 1,071

용량 : 1.01 TB

이미지 프레임

AI 학습용 데이터 구축량

객체 바운딩박스	동적 객체	예측불가 동적객체	도로상에 출현하는 고라니, 사슴 보행자	5만
	정적 객체	예측불가 정적객체	화물차에서의 낙하물(상자), 라바콘, 공사표지판, 쓰레기	30만
		도로위 낙서	산사태 등의 암석	5만
노면		포트홀	포트홀	10만
		보수완료 포트홀	정상도로에 보수완료된 포트홀	20만
		맨홀	정상도로에 맨홀	
객체 시맨틱 세그멘테이션	노면	크랙	크랙	30만

Images: 50,000

Labels: 50,000

Images: 35,856

Labels: 35,856

## 4. Code review



Json -> txt.

```
import os
import json

def json_to_yolo(json_file, output_folder, label_mapping, image_folder):
    # JSON 파일을 읽어옵니다.
    with open(json_file) as f:
        data = json.load(f)

    # 이미지 정보와 파일 이름을 가져옵니다.
    image_info = data["images"]
    image_name = image_info["file_name"].split('.')[0]
    yolo_labels = []

    # JSON 데이터의 각 주석에 대해 반복합니다.
    for annotation_info in data["annotations"]:
        # 카테고리 ID를 가져옵니다.
        category_id = annotation_info["category_id"]

        # 레이블 매핑에 있는 카테고리가 있으면 해당 정보를 이용해 YOLO 형식으로 변환합니다.
        if category_id in label_mapping:
            bbox = annotation_info["bbox"] # bounding box 좌표를 가져옵니다.
            xmin, ymin, width, height = bbox # bounding box의 xmin, ymin, 폭, 높이를 가져옵니다.

            # 클래스 ID를 레이블 매핑에서 가져옵니다.
            class_id = label_mapping[category_id]

            # 이미지 너비와 높이를 설정합니다.
            image_info["width"] = 1280
            image_info["height"] = 720

            # 카테고리명과 관련된 좌표를 YOLO format으로 변환합니다.
            x_center = (xmin + width / 2) / image_info["width"]
            y_center = (ymin + height / 2) / image_info["height"]
            box_width = width / image_info["width"]
            box_height = height / image_info["height"]
```

```
},
"images": {
  "file_name": "V0F_HY_0262_20210105_093553_E_CH0_Seoul_Sun_Industrialroads_Day_45609.png",
  "height": 0,
  "width": 0,
  "id": 1
},
```

V1F\_HY\_1111\_20160212\_012814\_E\_CH1\_Seoul\_Sun\_Industrialroads\_Day\_92586 - Windows 메모장

파일(F)	편집(E)	서식(O)	보기(V)	도움말(H)
0 0.22355769230769235 0.5089031339031339 0.06570512820512822 0.15028490028490024				
0 0.6045101610429447 0.13910190865712338 0.029476610429447804 0.055811179277436944				
0 0.6234422929447853 0.18106680299931835 0.014139187116564412 0.03323108384458077				
0 0.6381805981595091 0.21472392638036805 0.02779907975460123 0.0408997955010225				
0 0.7052446705426356 0.3569121447028424 0.024830426356589188 0.024763135228251482				
0 0.7259871608527133 0.35233634797588287 0.016654554263565925 0.03929801894918172				



label, x\_center, y\_center, box\_width, box\_height

## 4. Code review



Json -> txt.

```
# YOLO 형식 레이블을 추가합니다.
yolo_labels.append(f"{class_id} {x_center} {y_center} {box_width} {box_height}")

# 출력 파일 경로를 생성합니다.
output_file_path = os.path.join(output_folder, f"{image_name}.txt")

# YOLO 형식 레이블을 txt 파일로 저장합니다.
with open(output_file_path, "w") as f:
    f.write("\n".join(yolo_labels))

# 결과 값 용량이 0KB인 경우 파일을 삭제하고 삭제한 파일 이름을 반환합니다.
if os.path.getsize(output_file_path) == 0:
    os.remove(output_file_path)

# 이미지 파일을 찾기 위해 가능한 확장자를 사용합니다.
image_file_path = None
for image_ext in [".png"]:
    test_path = os.path.join(image_folder, f"{image_name}{image_ext}")
    if os.path.exists(test_path):
        image_file_path = test_path
        break
if image_file_path is not None:
    os.remove(image_file_path)
    return image_name
return None
```

이름	수정된 날짜	유형	크기
V0F_HY_0262_20210105_093553_E_CH0_Seoul_...	2023-07-05 오후 11:20	텍스트 문서	1KB
V0F_HY_0284_20210105_093754_E_CH1_Seoul_...	2023-07-05 오후 11:20	텍스트 문서	1KB
V0F_HY_0323_20210105_093453_E_CH1_Seoul_...	2023-07-05 오후 11:20	텍스트 문서	1KB
V0F_HY_0715_20210105_093453_E_CH1_Seoul_...	2023-07-05 오후 11:20	텍스트 문서	1KB
V0F_HY_0716_20210105_093553_E_CH0_Seoul_...	2023-07-05 오후 11:20	텍스트 문서	1KB
V0F_HY_0765_20210105_093553_E_CH0_Seoul_...	2023-07-05 오후 11:20	텍스트 문서	1KB
V0F_HY_0873_20210105_093754_E_CH0_Seoul_...	2023-07-05 오후 11:20	텍스트 문서	1KB
V0F_HY_0911_20210105_093754_E_CH0_Seoul_...	2023-07-05 오후 11:20	텍스트 문서	0KB
V0F_HY_1426_20210105_093754_E_CH1_Seoul_...	2023-07-05 오후 11:20	텍스트 문서	1KB
V0F_HY_1639_20210105_093453_E_CH1_Seoul_...	2023-07-05 오후 11:20	텍스트 문서	1KB
V0F_HY_1670_20210105_093754_E_CH0_Seoul_...	2023-07-05 오후 11:20	텍스트 문서	0KB
V0F_HY_1703_20210105_093553_E_CH0_Seoul_...	2023-07-05 오후 11:20	텍스트 문서	1KB
V0F_HY_1712_20210105_093553_E_CH1_Seoul_...	2023-07-05 오후 11:20	텍스트 문서	1KB
V0F_HY_2163_20210105_093553_E_CH1_Seoul_...	2023-07-05 오후 11:20	텍스트 문서	1KB
V0F_HY_2321_20210105_093754_E_CH0_Seoul_...	2023-07-05 오후 11:20	텍스트 문서	1KB
V0F_HY_2441_20210105_093654_E_CH0_Seoul_...	2023-07-05 오후 11:20	텍스트 문서	1KB
V0F_HY_2593_20210105_093754_E_CH0_Seoul_...	2023-07-05 오후 11:20	텍스트 문서	1KB
V0F_HY_2639_20210105_093453_E_CH0_Seoul_...	2023-07-05 오후 11:20	텍스트 문서	0KB
V0F_HY_2644_20210105_093453_E_CH1_Seoul_...	2023-07-05 오후 11:20	텍스트 문서	1KB

이름	수정된 날짜	유형	크기
V2F_HY_3479_20201223_142813_E_CH0_Seoul_...	2023-07-05 오후 12:21	텍스트 문서	1KB
V2F_HY_3490_20210105_143208_E_CH0_Seoul_...	2023-07-05 오후 12:21	텍스트 문서	1KB
V2F_HY_3516_20201223_153813_E_CH0_Seoul_...	2023-07-05 오후 12:21	텍스트 문서	1KB
V2F_HY_3526_20210108_112705_E_CH0_Seoul_...	2023-07-05 오후 12:21	텍스트 문서	1KB
V2F_HY_3528_20210105_100640_E_CH0_Seoul_...	2023-07-05 오후 12:21	텍스트 문서	1KB
V2F_HY_3532_20201223_153813_E_CH0_Seoul_...	2023-07-05 오후 12:21	텍스트 문서	1KB
V2F_HY_3533_20201223_153813_E_CH2_Seoul_...	2023-07-05 오후 12:21	텍스트 문서	1KB
V2F_HY_3538_20201222_111207_E_CH1_Seoul_...	2023-07-05 오후 12:20	텍스트 문서	1KB
V2F_HY_3538_20210105_143208_E_CH1_Seoul_...	2023-07-05 오후 12:21	텍스트 문서	1KB
V2F_HY_3541_20210105_104252_E_CH2_Seoul_...	2023-07-05 오후 12:21	텍스트 문서	1KB
V2F_HY_3557_20201223_153913_E_CH1_Seoul_...	2023-07-05 오후 12:21	텍스트 문서	1KB
V2F_HY_3564_20210105_100742_E_CH0_Seoul_...	2023-07-05 오후 12:21	텍스트 문서	1KB
V2F_HY_3578_20210105_100940_E_CH1_Seoul_...	2023-07-05 오후 12:21	텍스트 문서	1KB
V2F_HY_3583_20210105_100940_E_CH1_Seoul_...	2023-07-05 오후 12:20	텍스트 문서	1KB
V2F_HY_3591_20210105_100640_E_CH1_Seoul_...	2023-07-05 오후 12:21	텍스트 문서	1KB
V2F_HY_3593_20201222_105710_E_CH0_Seoul_...	2023-07-05 오후 12:20	텍스트 문서	1KB
V2F_HY_3597_20210105_104252_E_CH2_Seoul_...	2023-07-05 오후 12:21	텍스트 문서	1KB
V2F_HY_3598_20210105_100940_E_CH0_Seoul_...	2023-07-05 오후 12:21	텍스트 문서	1KB
V2F_HY_3603_20201223_154113_E_CH1_Seoul_...	2023-07-05 오후 12:21	텍스트 문서	1KB

## 4. Code review

### Json -> txt.

# 카테고리 목록 정의

```
label_mapping = {1: 0, 3: 1, 6: 2, 7: 3}
```

# JSON 파일과 출력 디렉터리 경로를 지정합니다.

```
input_folder = 'MMPP_data/Validation/Annotations/TOA/5.Mainroad_F01/'
```

```
output_folder = 'MMPP_data/5.Mainroad_F01_del'
```

# 이미지 디렉터리 경로를 지정합니다.

```
image_folder = 'MMPP_data/Validation/Images/TOA/5.Mainroad_F01/'
```

# 출력 디렉터리가 없을 경우 생성합니다.

```
if not os.path.exists(output_folder):  
    os.makedirs(output_folder)
```

# JSON 파일 목록을 가져옵니다.

```
json_files = [f for f in os.listdir(input_folder) if f.endswith('.json')]
```

# 삭제된 파일 목록을 저장할 리스트를 생성합니다.

```
deleted_files = []
```

for json\_file in json\_files:

```
    json_file_path = os.path.join(input_folder, json_file)
```

# image\_folder 매개변수 추가

```
    deleted_file = json_to_yolo(json_file_path, output_folder, label_mapping, image_folder)
```

# 반환된 파일 이름이 None이 아니면 삭제된 파일이므로 목록에 추가합니다.

```
    if deleted_file:
```

```
        deleted_files.append(deleted_file)
```

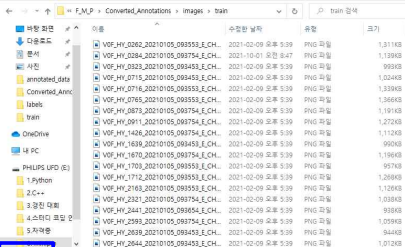
# 삭제된 파일 목록을 출력합니다.

```
print("삭제된 파일 목록:")
```

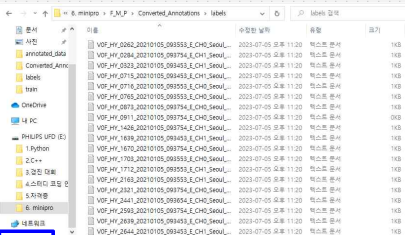
```
for deleted_file in deleted_files:
```

```
    print(deleted_file)
```

0: Animal, 1: Garbage bag, 2: Box, 3: stones



이름	수정된 날짜	유형	크기
VOF_HY_0262_20210105_093553_E_CH...	2021-02-09 오후 5:39	PNG 파일	1,311KB
VOF_HY_0284_20210105_093754_E_CH...	2021-10-01 오전 8:47	PNG 파일	1,139KB
VOF_HY_0323_20210105_093453_E_CH...	2021-02-09 오후 5:39	PNG 파일	993KB
VOF_HY_0715_20210105_093453_E_CH...	2021-02-09 오후 5:39	PNG 파일	1,024KB
VOF_HY_0716_20210105_093553_E_CH...	2021-02-09 오후 5:39	PNG 파일	1,339KB
VOF_HY_0765_20210105_093553_E_CH...	2021-02-09 오후 5:39	PNG 파일	1,366KB
VOF_HY_0873_20210105_093754_E_CH...	2021-02-09 오후 5:39	PNG 파일	1,191KB
VOF_HY_0911_20210105_093754_E_CH...	2021-02-09 오후 5:39	PNG 파일	1,272KB
VOF_HY_1426_20210105_093754_E_CH...	2021-02-09 오후 5:39	PNG 파일	1,112KB
VOF_HY_1639_20210105_093453_E_CH...	2021-02-09 오후 5:39	PNG 파일	990KB
VOF_HY_1670_20210105_093754_E_CH...	2021-02-09 오후 5:39	PNG 파일	1,196KB
VOF_HY_1703_20210105_093553_E_CH...	2021-02-09 오후 5:39	PNG 파일	957KB
VOF_HY_1712_20210105_093553_E_CH...	2021-02-09 오후 5:39	PNG 파일	1,268KB
VOF_HY_2163_20210105_093553_E_CH...	2021-02-09 오후 5:39	PNG 파일	1,126KB
VOF_HY_2321_20210105_093754_E_CH...	2021-02-09 오후 5:39	PNG 파일	1,038KB
VOF_HY_2441_20210105_093654_E_CH...	2021-02-09 오후 5:39	PNG 파일	938KB
VOF_HY_2593_20210105_093754_E_CH...	2021-02-09 오후 5:39	PNG 파일	1,059KB
VOF_HY_2639_20210105_093453_E_CH...	2021-02-09 오후 5:39	PNG 파일	944KB
VOF_HY_2644_20210105_093453_E_CH...	2021-02-09 오후 5:39	PNG 파일	1,012KB



이름	수정된 날짜	유형	크기
VOF_HY_0262_20210105_093553_E_CH0_Seu...	2023-07-05 오후 11:20	텍스트 문서	1KB
VOF_HY_0284_20210105_093754_E_CH1_Seu...	2023-07-05 오후 11:20	텍스트 문서	1KB
VOF_HY_0323_20210105_093453_E_CH1_Seu...	2023-07-05 오후 11:20	텍스트 문서	1KB
VOF_HY_0715_20210105_093453_E_CH1_Seu...	2023-07-05 오후 11:20	텍스트 문서	1KB
VOF_HY_0716_20210105_093553_E_CH0_Seu...	2023-07-05 오후 11:20	텍스트 문서	1KB
VOF_HY_0765_20210105_093553_E_CH0_Seu...	2023-07-05 오후 11:20	텍스트 문서	1KB
VOF_HY_0873_20210105_093754_E_CH0_Seu...	2023-07-05 오후 11:20	텍스트 문서	1KB
VOF_HY_0911_20210105_093754_E_CH0_Seu...	2023-07-05 오후 11:20	텍스트 문서	0KB
VOF_HY_1426_20210105_093754_E_CH1_Seu...	2023-07-05 오후 11:20	텍스트 문서	1KB
VOF_HY_1639_20210105_093453_E_CH1_Seu...	2023-07-05 오후 11:20	텍스트 문서	1KB
VOF_HY_1670_20210105_093754_E_CH0_Seu...	2023-07-05 오후 11:20	텍스트 문서	1KB
VOF_HY_1703_20210105_093553_E_CH0_Seu...	2023-07-05 오후 11:20	텍스트 문서	1KB
VOF_HY_1712_20210105_093553_E_CH1_Seu...	2023-07-05 오후 11:20	텍스트 문서	1KB
VOF_HY_2163_20210105_093553_E_CH1_Seu...	2023-07-05 오후 11:20	텍스트 문서	1KB
VOF_HY_2321_20210105_093754_E_CH0_Seu...	2023-07-05 오후 11:20	텍스트 문서	1KB
VOF_HY_2441_20210105_093654_E_CH0_Seu...	2023-07-05 오후 11:20	텍스트 문서	1KB
VOF_HY_2593_20210105_093754_E_CH0_Seu...	2023-07-05 오후 11:20	텍스트 문서	1KB
VOF_HY_2639_20210105_093453_E_CH0_Seu...	2023-07-05 오후 11:20	텍스트 문서	1KB
VOF_HY_2644_20210105_093453_E_CH1_Seu...	2023-07-05 오후 11:20	텍스트 문서	1KB



## 4. Code review

## YOLOv5lu

```
from ultralytics import YOLO
```

```
# yolo5 사전학습 모델 선택
```

```
model = YOLO("models/yolov5lu.pt") # file size : 약 106.9 MB
```

```
# model = YOLO("models/yolov5l.pt") # file size : 약 92 MB
```

```
# 지정된 사전 학습 모델을 이용하여 지정된 예속으로 train 수행
```

```
train_results = model.train(data="yolov5_train2.yaml", epochs=20) # 20 epochs
```

```
# train_results = model.train(data="yolov5_train2.yaml", epochs=30) # 30 epochs
```

```
# 예측 결과를 확인합니다.
```

```
results = model.predict(source='data/test/images/', save=True)
```



images: 7,673  
test: 36



images: 7,673  
train: 7,673 -> 160m (2.4 h)  
test: 36 -> 1m4s

Epoch	GPU_mem	box_loss	cls_loss	dfl_loss	Instances	Size
17/20	9.99G	0.6968	0.4954	0.8680	5	640: 100% [████████] 752/752 [07:18:00:00, 1.72it/s]
	Class	Images	Instances	Box(P	R	mAP50 mAP50-95): 100% [████████] 3/3 [00:00:00:00, 4.53it/s]
	all	66	44	0.374	0.477	0.314 0.27
Epoch	GPU_mem	box_loss	cls_loss	dfl_loss	Instances	Size
18/20	9.99G	0.6799	0.4862	0.8634	7	640: 100% [████████] 752/752 [07:19:00:00, 1.71it/s]
	Class	Images	Instances	Box(P	R	mAP50 mAP50-95): 100% [████████] 3/3 [00:00:00:00, 4.53it/s]
	all	66	44	0.373	0.514	0.319 0.281
Epoch	GPU_mem	box_loss	cls_loss	dfl_loss	Instances	Size
19/20	10.4G	0.6656	0.4705	0.8621	9	640: 100% [████████] 752/752 [07:18:00:00, 1.71it/s]
	Class	Images	Instances	Box(P	R	mAP50 mAP50-95): 100% [████████] 3/3 [00:00:00:00, 4.65it/s]
	all	66	44	0.336	0.483	0.274 0.24
Epoch	GPU_mem	box_loss	cls_loss	dfl_loss	Instances	Size
20/20	10.1G	0.6562	0.4618	0.8606	6	640: 100% [████████] 752/752 [07:18:00:00, 1.71it/s]
	Class	Images	Instances	Box(P	R	mAP50 mAP50-95): 100% [████████] 3/3 [00:01:00:00, 2.79it/s]
	all	66	44	0.311	0.682	0.289 0.258

20 epochs completed in 2.470 hours.

Optimizer stripped from runs/detect/train15/weights/last.pt, 106.8MB

Optimizer stripped from runs/detect/train15/weights/best.pt, 106.8MB

## 4. Code review

### YOLOv8I

```
1 import os
2 import shutil
3 from ultralytics import YOLO
4 from sklearn.model_selection import train_test_split
```

```
1 # Read images and annotations
2 images = [os.path.join("C:/mini_project/yolov8/Road_Obstacle_HG/images", x) for x in os.listdir("C:/mini_project/yolov8/Road_Obstacle_HG/images") if x[-3:] == ".png"]
3 annotations = [os.path.join("C:/mini_project/yolov8/Road_Obstacle_HG/labels", x) for x in os.listdir("C:/mini_project/yolov8/Road_Obstacle_HG/labels") if x[-3:] == ".txt"]
4
5 images.sort()
6 annotations.sort()
7
8 # Split the dataset
9 # train_images, val_images, train_annotations, val_annotations = train_test_split(images, annotations, test_size=0.2, random_state=1)
10 # val_images, test_images, val_annotations, test_annotations = train_test_split(val_images, val_annotations, test_size=0.5, random_state=1)
11
12 train_images, test_images, train_annotations, test_annotations = train_test_split(images, annotations, test_size=0.1, random_state=1)
```

```
1 # images 폴더 및 labels 폴더에 각각 train, val, test 폴더 생성
2 pass_list = ['C:/mini_project/yolov8/Road_Obstacle_HG/images/train',
3             'C:/mini_project/yolov8/Road_Obstacle_HG/images/val',
4             'C:/mini_project/yolov8/Road_Obstacle_HG/images/test',
5             'C:/mini_project/yolov8/Road_Obstacle_HG/labels/train',
6             'C:/mini_project/yolov8/Road_Obstacle_HG/labels/val',
7             'C:/mini_project/yolov8/Road_Obstacle_HG/labels/test']
8
9 for path in pass_list:
10     os.makedirs(path, exist_ok=True)
```



images: 35,856  
train: 32,370  
test: 3,586



## 4. Code review



### YOLOv8l

```

1 #Utility function to move images
2 def move_files_to_folder(list_of_files, destination_folder):
3     for f in list_of_files:
4         try:
5             shutil.move(f, destination_folder)
6         except:
7             print(f)
8             assert False
9
10 # Move the splits into their folders
11 move_files_to_folder(train_images, 'C:/mini_project/yolov8/Road_Obstacle_HG/images/train/')
12 move_files_to_folder(val_images, 'C:/mini_project/yolov8/Road_Obstacle_HG/images/val/')
13 move_files_to_folder(test_images, 'C:/mini_project/yolov8/Road_Obstacle_HG/images/test/')
14 move_files_to_folder(train_annotations, 'C:/mini_project/yolov8/Road_Obstacle_HG/labels/train/')
15 move_files_to_folder(val_annotations, 'C:/mini_project/yolov8/Road_Obstacle_HG/labels/val/')
16 move_files_to_folder(test_annotations, 'C:/mini_project/yolov8/Road_Obstacle_HG/labels/test/')
17
18 # 20230706_1624_test :- 1500 -> 1200 / -150 / -150 나눠짐

```

```

1 # Yaml 파일 경로를 정의합니다.
2 yaml_path = os.path.join("C:/mini_project/yolov8/data/", "data11.yaml")
3
4 # 설정 파일로 YOLO 객체를 초기화.
5 model = YOLO("yolov8l.yaml")
6
7 # 데이터와 지정한 에폭으로 YOLO 모델을 학습시킵니다.
8 train_results = model.train(data=yaml_path, epochs=20)

```

✓ 204m 2.6s

```

1 # 예측 결과를 확인
2 results = model.predict(source='C:/mini_project/yolov8/Road_Obstacle_HG/images/test', save=True)
3
4 3m 58.0s
5
6 results = model.predict(source='C:/mini_project/yolov8/Road_Obstacle_HG/videos', save=True)

```

```

Epoch 18/20   GPU_mem 10.9G   box_loss 0.5958   cls_loss 0.3717   dfl_loss 0.8602   Instances 46   Size 640: 100%|██████████| 2017/2017 [05:54<00:00, 5.69it/s]
Class        Images  Instances  Box(P   R   mAP50  mAP50-95): 100%|██████████| 1009/1009 [03:41<00:00, 4.56it/s]
all          32270   98660     0.768  0.773  0.802   0.656

Epoch 19/20   GPU_mem 10.9G   box_loss 0.5859   cls_loss 0.3637   dfl_loss 0.8578   Instances 50   Size 640: 100%|██████████| 2017/2017 [05:54<00:00, 5.69it/s]
Class        Images  Instances  Box(P   R   mAP50  mAP50-95): 100%|██████████| 1009/1009 [03:42<00:00, 4.54it/s]
all          32270   98660     0.768  0.775  0.804   0.659

Epoch 20/20   GPU_mem 11G     box_loss 0.5765   cls_loss 0.3562   dfl_loss 0.8545   Instances 41   Size 640: 100%|██████████| 2017/2017 [05:54<00:00, 5.69it/s]
Class        Images  Instances  Box(P   R   mAP50  mAP50-95): 100%|██████████| 1009/1009 [04:16<00:00, 3.93it/s]
all          32270   98660     0.769  0.778  0.808   0.662

20 epochs completed in 3.276 hours.
Optimizer stripped from runs\detect\train18\weights\last.pt, 87.6MB
Optimizer stripped from runs\detect\train18\weights\best.pt, 87.6MB

```



images: 35,856  
train: 32,370 -> 204m 2.6s (3.2 h)  
test: 3,586 -> 3m 58.0s

## 4. Code review



### Resnet50

```
from ultralytics import YOLO
import time
import numpy as np
import pandas as pd
import cv2
import torch
import torch.nn as nn
from torch.utils.data import DataLoader
from torchinfo import summary
from torchvision import models, datasets, transforms
from IPython.display import Image, clear_output
import warnings
import torch.nn.functional as F
import os
from PIL import Image
from torchvision.transforms import transforms
from tqdm import tqdm_notebook as tqdm
import random
import json
import glob
from matplotlib import pyplot as plt
import matplotlib.pyplot as plt
from PIL import Image, ImageDraw

warnings.filterwarnings("ignore")
device = "cuda" if torch.cuda.is_available() else "cpu"
```



### Categort\_ID List 생성.

```
file_list = "C:/Users/chlasltn/Desktop/final_project"
category_list=[]
def load_categories(file_path, category_list):
    with open(file_path) as json_file:
        data = json.load(json_file)
        categories = data["categories"]
        category_list.append(categories)

load_categories(file_list,category_list)
print(category_list)
```



### Categort\_list에서 사용할 Labeling Df 생성.

```
df_label = pd.json_normalize(category_list)
df_label.reset_index(inplace=True)
df_label.drop("index",axis=1,inplace=True)
df_label.drop([1,3,4,7,8,9],axis="columns",inplace=True)

df_label
```

0	2	5	6
0 ['id': 1, 'name': 'Animals(Dolls)']	['id': 3, 'name': 'Garbage bag & sacks']	['id': 6, 'name': 'Box']	['id': 7, 'name': 'Stones on road']

## 4. Code review



## Resnet50



## Images File Name 과 BBox 좌표 리스트 생성.

```

image_file_path = "C:/Users/chlasltn/Desktop/final_project/Validation/Images/TOA/3.Industrialroads_F01/"
annotation_file_path = "C:/Users/chlasltn/Desktop/final_project/Validation/Annotations/TOA/Industrialroads/"
annotation_files = glob.glob(annotation_file_path + "*.json")
dropped_bbox_list = []
image_path_list = []

def get_dropped_bboxes(image_file_path, annotation_file_path):
    annotation_files = glob.glob(annotation_file_path + "*.json")
    dropped_bbox_list = []
    image_path_list = []
    for file in annotation_files:
        with open(file, 'r') as json_file:
            data = json.load(json_file)
            file_name = data["images"]["file_name"]

            if "images" in data:
                img_path = data["images"]["file_name"]
                image_path_list.append(img_path)

            if "annotations" in data:
                cat = data["annotations"]
                for ca in cat:
                    if ca["category_id"] in [1, 3, 6, 7]:
                        dropped_bbox_list.append(ca["bbox"])
    return dropped_bbox_list, image_path_list

dropped_bbox_list, image_path_list = get_dropped_bboxes(image_file_path, annotation_file_path)

```



## Dataset 생성.

```

# dropped_bbox_list
image_file_paths = [os.path.join(image_file_path, f) for f in os.listdir(image_file_path) if f.endswith(".png")]
from torch.utils.data import Dataset
class BBoxDataset(Dataset):
    def __init__(self, image_paths, bbox_list, transform=None):
        self.image_paths = image_paths
        self.image_filenames = [os.path.basename(f) for f in image_paths] # 설명
        self.bbox_list = bbox_list
        self.transform = transform

    def __len__(self):
        return len(self.image_filenames)

    def __getitem__(self, idx):
        image_path = self.image_paths[idx]
        img = Image.open(image_path).convert("RGB")
        bbox = np.array(self.bbox_list[idx], dtype=np.float32)

        if self.transform:
            img = self.transform(img)

        return img, bbox

transform = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.RandomHorizontalFlip(),
    transforms.RandomRotation(10),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
])
dataset = BBoxDataset(image_file_paths,
                      bbox_list = dropped_bbox_list, transform=transform
)
dataset

```

## 4. Code review

## Resnet50



## DataLoader 생성.

```
from torch.utils.data import DataLoader

train_loader = DataLoader(dataset, batch_size=20, shuffle=True)
```

```
from torchvision.models import resnet50
```

```
model = resnet50(pretrained=True) # Resnet50 모델 사용
num_features = model.fc.in_features # Layer의 입력 특성 수 생성
model.fc = torch.nn.Linear(num_features, 4) # FC Layer 생성 후 output의 크기는 4로 설정
```



## Train 과정 총 정리.

```
class BBoxDataset(Dataset):
    def __init__(self, image_paths, bbox_list, transform=None, unnormalize=False):
        self.image_paths = image_paths # 파일경로
        self.image_filenames = [os.path.basename(f) for f in image_paths] # 파일이름만 가져옴
        self.bbox_list = bbox_list # BBOX 좌표
        self.transform = transform # 이미지 변환에 사용
        self.unnormalize = unnormalize # 정규화를 할지 안할지 여부
        self.original_width, self.original_height = 1280, 720 # 원본 이미지 크기
        self.transformed_width, self.transformed_height = 224, 224 # 전처리 후 이미지 크기
        self.mean = [0.485, 0.456, 0.406]
        self.std = [0.229, 0.224, 0.225]
        self.orig_image_paths = image_paths # 원본 이미지 경로
```

```
def __len__(self):
    return len(self.image_filenames)

def __getitem__(self, idx):
    image_path = self.image_paths[idx] # 이미지 파일 경로 불러오기
    img = Image.open(image_path).convert("RGB")
    bbox = np.array(self.bbox_list[idx], dtype=np.float32) # bbox 좌표를 가져옴

    width_ratio = self.transformed_width / self.original_width
    height_ratio = self.transformed_height / self.original_height

    width_ratio = self.transformed_width / self.original_width # width ratio 계산
    height_ratio = self.transformed_height / self.original_height # height ratio 계산

    # bbox 좌표 변환
    bbox[0] = (bbox[0] * width_ratio) / self.transformed_width
    bbox[1] = (bbox[1] * height_ratio) / self.transformed_height
    bbox[2] = (bbox[2] * width_ratio) / self.transformed_width
    bbox[3] = (bbox[3] * height_ratio) / self.transformed_height

    if self.transform:
        img = self.transform(img) # transform 적용
        if self.unnormalize:
            img = img * torch.tensor(self.std).view(3, 1, 1) + torch.tensor(self.mean).view(3, 1, 1)
            img = img.clamp(0, 1) # 비정규화 적용
    return img, bbox
```

## 4. Code review

## Resnet50



Train 과정 총 정리.

```

transform = transforms.Compose([
    transforms.Resize((224,224)),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485,0.456,0.406],std=[0.229,0.224,0.225])
])

# C:/Users/Playdata/Desktop/Final_Project/Validation/Images/T0A/3.Industrialroads_F01/

dataset = BBoxDataset(image_file_paths,
                      bbox_list = dropped_bbox_list,transform=transform
)

num_epochs = 10 # epoch 수 설정
device = "cuda" if torch.cuda.is_available() else "cpu"
model.to(device)

loss_function = torch.nn.SmoothL1Loss() # 손실함수 SmoothL1Loss() 설정
optimizer = torch.optim.Adam(model.parameters(),lr=0.001)

model = resnet50(pretrained=True) # Resnet50 모델 사용
num_features = model.fc.in_features # Layer 의 입력 특성 수 생성
model.fc = torch.nn.Linear(num_features,4) # FC Layer 생성 후 output의 크기는 4로 설정

train_loader = DataLoader(dataset,batch_size=10,shuffle=True)

```

```

for epoch in range(num_epochs):
    for inputs,targets in train_loader:
        inputs = inputs.to(device)
        targets = targets.to(device)

        optimizer.zero_grad() # Gradient 0으로 설정
        outputs = model(inputs)

        loss = loss_function(outputs, targets) # 손실 계산
        loss.backward() # Gradient 계산
        optimizer.step() # Optimizer 업데이트

    print(f"Epoch: {epoch+1}/{num_epochs},loss: {loss.item():.4f}")

```

## 4. Code review

## Resnet50



## Test set의 Images path와 BBox\_list 생성.

```
image_testfile_path = "C:/Users/chlasltn/Desktop/final_project/resnet50_test/test_image/"
image_testfile_paths = [os.path.join(image_testfile_path, f) for f in os.listdir(image_testfile_path) if f.endswith(".png")]

annotation_file_path = "C:/Users/chlasltn/Desktop/final_project/resnet50_test/test_json/"
# annotation_files = glob.glob(annotation_file_path + "*.json")

def get_dropped_bboxes(image_file_path, annotation_file_path):
    annotation_files = glob.glob(annotation_file_path + "*.json")
    dropped_bbox_lists = []
    image_path_list = []
    for file in annotation_files:
        with open(file, 'r') as json_file:
            data = json.load(json_file)
            file_name = data["images"]["file_name"]

            if "images" in data:
                img_path = data["images"]["file_name"]
                image_path_list.append(img_path)

            if "annotations" in data:
                cat = data["annotations"]
                for ca in cat:
                    if ca["category_id"] in [1, 3, 6, 7]:
                        dropped_bbox_lists.append(ca["bbox"])
    return dropped_bbox_list, image_path_list

dropped_bbox_lists, image_file_paths = get_dropped_bboxes(image_testfile_paths, annotation_file_path)
image_testfile_paths
```

```
test_dataset = BBoxDataset(image_testfile_path, dropped_bbox_lists,
                           transform=transform)
test_loader = DataLoader(test_dataset, batch_size=10)
```

```
def add_boxes_to_image(image_path, result_file_path, bboxes, class_names):
    image = cv2.imread(image_path)
    image_copy = image.copy()
    for bbox in bboxes:
        x, y, w, h = [int(i) for i in bbox]
        cv2.rectangle(image_copy, (x, y), (x+w, y+h), (0, 255, 0), 2)
    # result_file_path에 파일이름.png으로 저장
    cv2.imwrite(result_file_path, image_copy)
```



## 4. Code review

## Resnet50



## Test 과정.

```

image_testfile_path = "C:/Users/chlasltn/Desktop/final_project/resnet50_test/test_image/"
image_testfile_paths = [os.path.join(image_testfile_path, f) for f in os.listdir(image_testfile_path) if f.endswith(".png")]

image_result_file = "C:/Users/chlasltn/Desktop/final_project/resnet50_test/test_result/"
annotation_file_path = "C:/Users/chlasltn/Desktop/final_project/resnet50_test/test_json/"
annotation_files = glob.glob(annotation_file_path + "*.json")

if not os.path.exists(image_result_file): # 이미지 결과 생성
    os.makedirs(image_result_file)

dropped_bbox_lists = []
image_path_list = []
for file in annotation_files:
    with open(file, 'r') as json_file:
        data = json.load(json_file)
        file_name = data["images"][0]["file_name"]

        if "images" in data:
            img_path = data["images"][0]["file_name"]
            image_path_list.append(img_path)

        if "annotations" in data:
            cat = data["annotations"]
            for ca in cat:
                if ca["category_id"] in [1, 3, 6, 7]:
                    dropped_bbox_lists.append(ca["bbox"])

```

```

test_losses = [] # 각 배치에서의 손실 저장할 리스트
test_dataset = BBBoxDataset(image_testfile_paths, dropped_bbox_lists,
                             transform=transform)
test_loader = DataLoader(test_dataset, batch_size=10) # DataLoader 설정

model.eval() # 평가모드 전환
total_loss = 0.0

with torch.no_grad():
    for file_idx, (inputs, targets) in enumerate(test_loader):
        inputs = inputs.to(device)
        outputs = model(inputs)

        loss = loss_function(outputs, targets) # 손실 계산
        test_losses.append(loss.item()) # 손실값 추가
        total_loss += loss.item() # 손실 값 누적

average_loss = total_loss / len(test_losses)
print(f"Average loss: {average_loss:.4f}")





```

## 4. Code review

### Resnet50





Resnet 진행하는 과정.

-  Dataset을 만들면서 BBox 좌표까지 비율에 맞춰 설정.
-  Transform.Compose를 통해서 Resnet50에서 기본적으로 제공하는 size (244,244)를 맞췄으며 평균과 표준편차로 정규화.
-  Loss Function을 SmoothL1Loss() 사용.
  -  SmoothL1Loss는 이상치의 영향을 감소시키며 Vanishing Gradient 문제 완화.



Resnet50을 진행하면서 느낀 점.

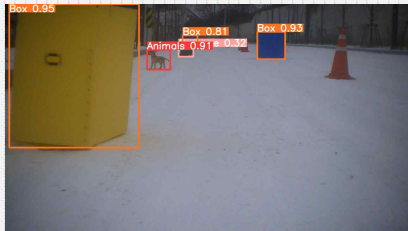
-  Resnet50의 대한 모델 이해도 부족. (Pretrained 모델을 사용한다고 해서 기본적인 모델의 구조를 알아야 함.)
-  BBox 좌표에 대한 이해도가 떨어짐. (Train까진 되나 Test 과정에서 좌표 설정이 어려움.)

5.

## 결과물



YOLOv8l

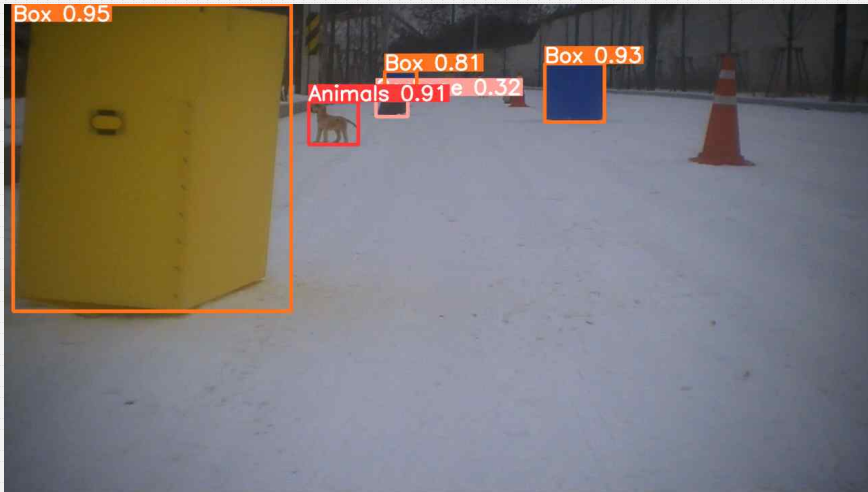


5.

## 결과물



YOLOv8l



5.

## 결과물



YOLOv8l



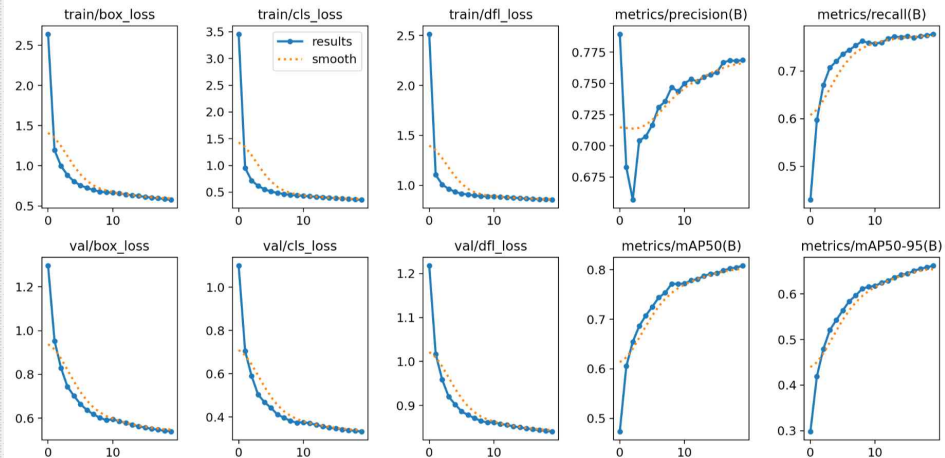
5.

## 결과물



YOLOv8l





5.

## 결과물

YOLOv5lu



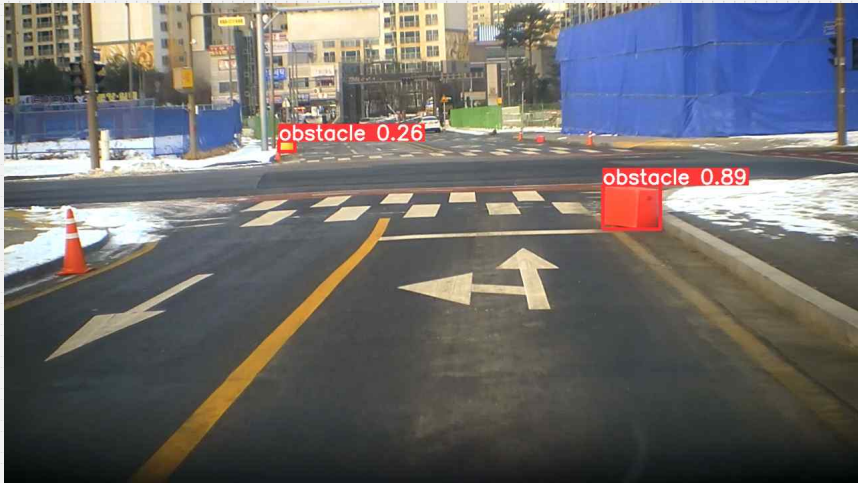


5.

## 결과물



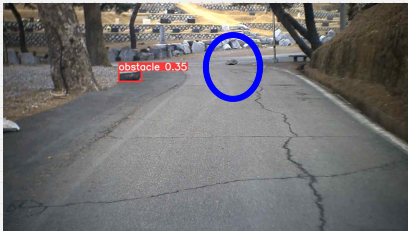
YOLOv5lu



5.

## 결과물

YOLOv5lu



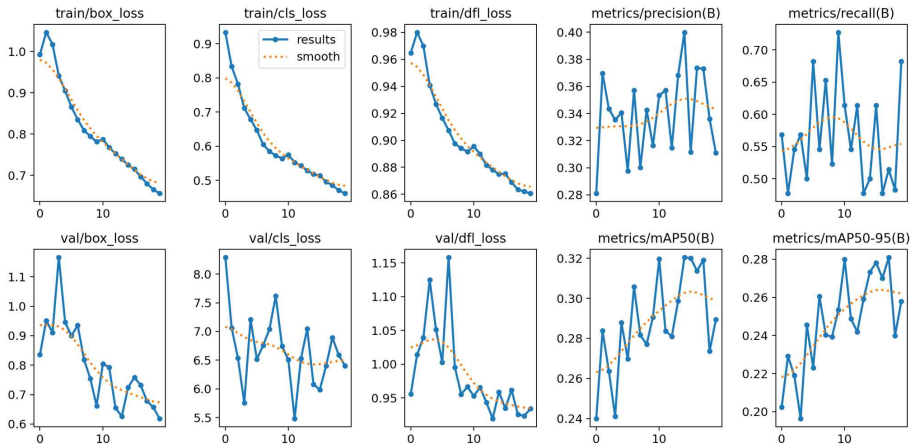
5.

## 결과물



YOLOv5lu



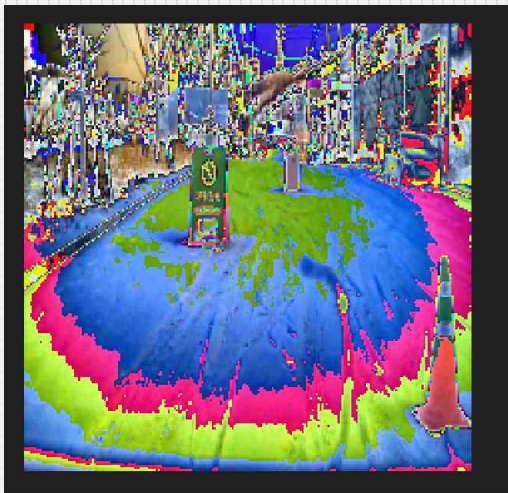


5.

## 결과물



Resnet50



## 6. 앞으로의 계획

**Resnet50  
완성하여 비교.**

**images  
추가 학습.**

**label  
mapping.**

**model  
성능 비교.**

**경고 알림 기능  
추가.**