시각인공지능프로젝트
최종발표

객체 검출 기법을 활용한 현장 작업자 위험 상황 인지 알고리즘 개발

팀원

20162528 윤성식

20171617 노지민

20172842 옥윤호

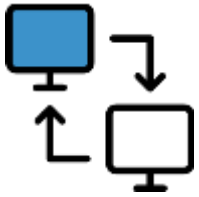
목차

1.서론 및 주제설명

- 연구동기
- 연구주제설명
- 프로젝트 요약

2.선행 연구 조사

- 모델 관련 선행조사
- 위험 알고리즘 관련 선행조사

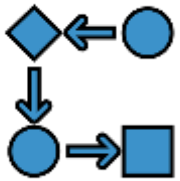


3.데이터 셋

- 데이터셋 소개
- 데이터 샘플링 과정
- 데이터 전처리 과정

4.모델링

- FCOS
- 사용 코드 특징
- 학습 및 결과



5.위험감지 알고리즘

- 정적 객체
- 동적 객체

6.결론 및 시사점

- 결론 및 시사점
- 보완점

1.서론 및 주제설명

연구동기

- 현장에서 종사하는 직원들의 안전이 매우 취약
- 어떠한 현장이던 노동자에게 위험한 장비가 있는 곳 이라면 최소한의 안전대책이 필요

특히나 한번의 실수가 생명에 직접적으로 영향을 주는 곳, 홀로 일하는 환경에서의 대책 필요

연구주제 설명

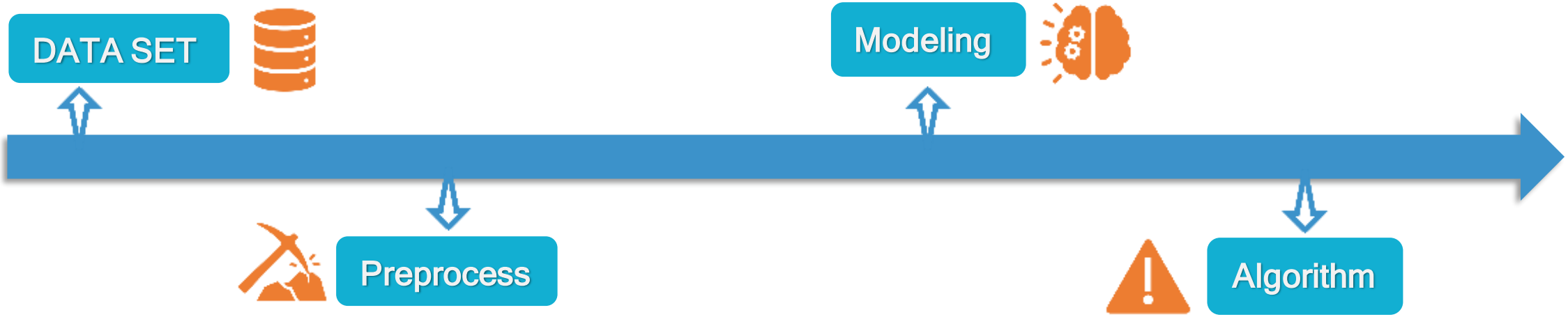
- 최신 Object Detection모델을 통해 실제 작업 현장에서의 객체를 탐지 해낼 것
- 객체를 탐지하는 모델은 1 Stage Model을 활용하며 Anchor Free 방법을 시도 해볼 것
- 객체 검출 이후에는 해당 객체의 위험여부와 동적 / 정적 객체 여부를 고려하여 위험 알고리즘을 제시할 것

1.프로젝트 요약

AI Hub roboflow

다양한 형태를 가진 데이터셋 다운

- FCOS 모델을 활용해 Detection 진행
- Detection 후 Loss와 평가지표 시각화



- 위험상황 하에서 분석에 필요한 객체와 샘플링
- Task에 적합하게 데이터셋을 재구성
- Torchvision패키지 FCOS모델의 Input 형식에 맞게 데이터 전처리

- 정적 / 동적 객체 선정
- 정적 객체에는 착용 유무 판별
- 동적 객체에는 거리계산

2.선행연구 조사

2-1) 모델 관련 선행조사

Animal detection and classification from camera trap image using different mainstream object detection architectures
(Mengyu Tan 외8명)

<Summary>

- Yolov5, Cascade R-CNN, FCOS 3개의 모델을 사용
- 동물 17종에 대해 카메라 트랩에서 캡처한 이미지를 사용
- Acc_0.7일 때 정확도가 87% 정도로 세 모델 비슷함

<한계점>

- 특정 동물에 대한 이미지가 작아 불균형한 데이터를 사용함

2-2) 위험감지 알고리즘 관련 선행조사

YOLO-v3을 활용한 건설 장비 주변 위험상황 인지 알고리즘 개발
(심승보, 최상일)

스마트건설 현장에서 개인 보호장비 검출의 개선 방법
(박용석 외2명)

<Summary>

- 두 논문 모두 YOLO 모델을 사용하여 학습 진행
- 첫 번째 논문의 경우 다짐 장비라는 동적 객체에 대해 작업자와의 거리를 구하는 방식으로 위험상황 알고리즘 구축
- 두 번째 논문의 경우 산업 현장에서 정적 객체에 대한 유무를 파악하는 방식으로 위험상황 알고리즘 구축

<한계점>

- 동적 객체와 정적 객체를 모두 고려하지 못함

3.데이터셋 소개

AI Hub

물류창고 내 작업 안전 데이터

- 물류창고 내 사고 및 위험 발생 가능성을 사전 파악할 수 있도록 작업환경 안의 각 객체에 대한 정보를 기반으로 하는 AI 학습용 데이터를 구축

roboflow

안전모 착용유무 데이터

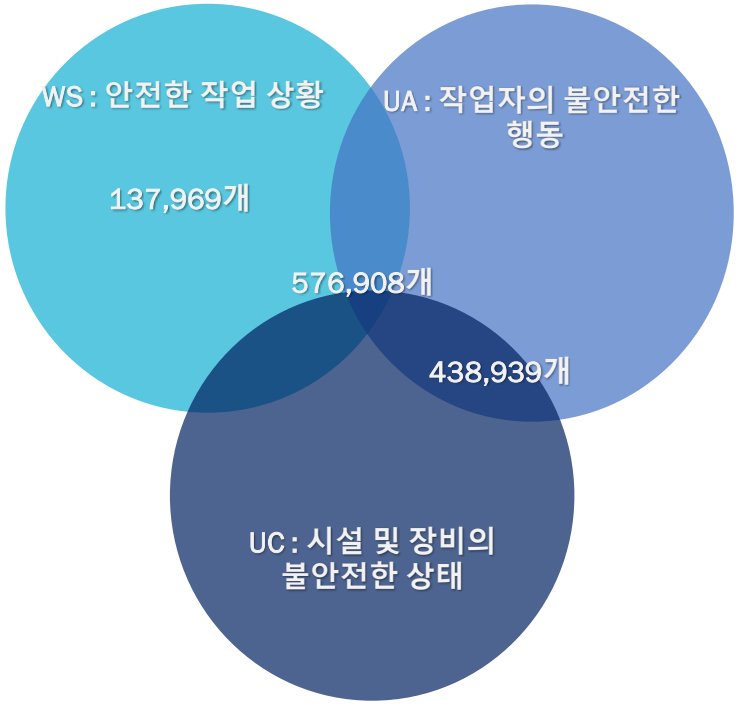
- 작업현장 내에 작업자들의 안전을 위해 안전모 착용유무를 보여주는 데이터 셋

프로젝트의 특성상 다양한 형태를 가진 데이터셋을 적용시켜보면 좋을 것이라고 생각

따라서 같은 주제이지만 다른 annotation을 가진 데이터를 찾아서 적용시킴

3.데이터셋 소개: 물류창고

물류창고 데이터



공정 상황

1. 작업 공정에서의 안전 상황
- 작업 공정 : 11개
  - Ex) 입고, 운반, 보관, ...
- 작업 상황 : 43개
  - Ex) WS-01, WS-02, ...

2. 작업자의 불안전한 행동
- 작업 공정 : 7개
  - Ex) 입고, 운반, 보관, ...
- 작업 상황 : 20개
  - Ex) UA-01, UA-02, ...

3. 시설 및 장비의 불안전한 상태
- 작업 공정 : 7개
  - Ex) 입고, 운반, 보관, ...
- 작업 상황 : 22개
  - Ex) UC-01, UC-02,

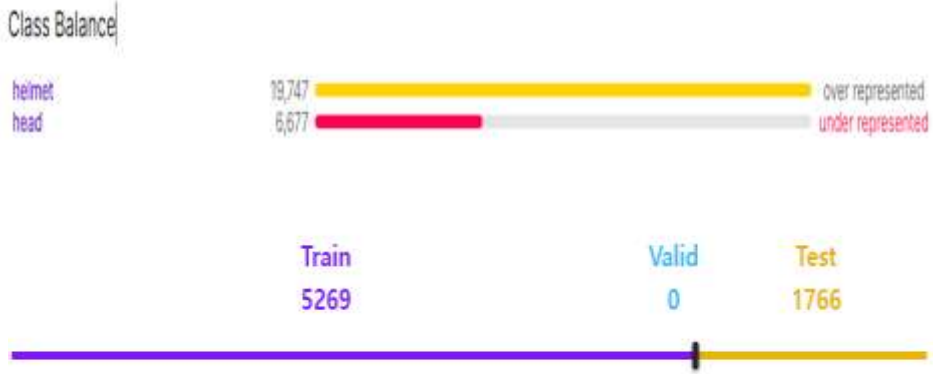
동적 객체, 정적 객체

동적 객체 (8가지)
1. 작업자(작업복 착용)
2. 작업자(작업복 미착용)
3. ...

정적 객체 (20가지)
1. 보관랙
2. 적재물류
3. ...

3.데이터셋 소개: 안전모 데이터

안전모 데이터



이미지 사이즈


```
"id": 5217,
"license": 1,
"file_name": "000001_jpg.rf.fddb09e33a544e332617f8ceb53ee805.jpg",
"height": 375,
"width": 500,
"date_captured": "2020-04-30T03:26:22+00:00"
```

```
"id": 3283,
"license": 1,
"file_name": "000003_jpg.rf.9fc4dfe06775f585757ede9300ba6af9.jpg",
"height": 264,
"width": 400,
"date_captured": "2020-04-30T03:26:22+00:00"
```

3.데이터 샘플링 과정

물류창고 데이터

TS_03_부가가치서비스.zip	19GB
TS_10_피킹,분배.zip	26GB
TS_08_출고.zip	33GB
TS_09_파렛트 렉.zip	33GB
TS_05_운반.zip	34GB
TS_02_보관.zip	34GB
TS_04_설비 및 장비.zip	36GB
TS_06_입고.zip	42GB
TS_01_도크설비.zip	46GB
TS_11_화재.zip	62GB
TS_07_지게차.zip	100GB

데이터 샘플링

```
import os
import shutil
path_dir = 'C:₩출고₩불안전한 상태(UC)'
move_path_dir = 'C:₩출고₩리샘플'
file_list = os.listdir(path_dir)
for item in file_list:
    for i in range(1,300,5):
        if i < 10:
            if item[-8:] == '000'+str(i)+'.jpg':
                shutil.copy(path_dir+'₩'+item,move_path_dir+'₩'+item)
        elif i<100:
            if item[-8:] == '00'+str(i)+'.jpg':
                shutil.copy(path_dir+'₩'+item,move_path_dir+'₩'+item)
        else:
            if item[-8:] == '0'+str(i)+'.jpg':
                shutil.copy(path_dir+'₩'+item,move_path_dir+'₩'+item)
```

```
img_dir = 'C:₩출고₩리샘플'
path_dir = 'C:₩출고 라벨₩불안전한 상태(UC)'
move_path_dir = 'C:₩출고 라벨₩라벨 리샘플'

file_list = os.listdir(path_dir)
img_file_list = os.listdir(img_dir)

for item in file_list:
    for i in img_file_list:
        if item[:29] == i[:29]:
            shutil.copy(path_dir+'₩'+item,move_path_dir+'₩'+item)
```

Train Dataset: 3151

Validation Dataset: 156

Test Dataset: 161

- 안전모 데이터셋: 샘플링을 진행하지 않고 그대로 사용
- 물류창고 데이터셋: 위험 알고리즘 구성과 데이터셋 불균형을 고려하여 위험상황 중 출고상황을 샘플링

3.데이터 전처리 과정

1-1.CoCo Dataset 변환

```
imgTmpDict = []
annoTmpDict = []

idNum = 1
annoIdNum = len(annoList)+1
for anno in annoList:
    with open(annoDir + anno, encoding="utf-8-sig", errors="ignore") as json_file:
        jsonData = json.load(json_file)

    imgPath = anno.replace(".json", ".jpg")

    imgSize = 1920 * 1080

    try :
        images = {
            "license": 1,
            "file_name": jsonData["Source data Info."]["source_data_ID"],
            "coco_url": "",
            "height": jsonData["Raw data Info."]["resolution"][1],
            "width": jsonData["Raw data Info."]["resolution"][0],
            "date_captured": jsonData["Raw data Info."]["date"],
            "flickr_url": "",
            "id": idNum
        }
    except :
        images = {
            "license": 1,
            "file_name": jsonData["Source data Info."]["source_data_ID"],
            "coco_url": "",
            "width": 1920,
            "height": 1080,
            "date_captured": jsonData["Raw data Info."]["date"],
            "flickr_url": "",
```

1-2.CoCo Dataset 변환

```
imgTmpDict.append(images)


for i in range(len(jsonData["Learning data info."]["annotation"])):
    if jsonData["Learning data info."]["annotation"][i]["type"] == 'box':
        #       try: # bbox 정보가 존재할 경우
        bbox = [
            jsonData["Learning data info."]["annotation"][i]["coord"][0],
            jsonData["Learning data info."]["annotation"][i]["coord"][1],
            jsonData["Learning data info."]["annotation"][i]["coord"][2],
            jsonData["Learning data info."]["annotation"][i]["coord"][3]
        ]

        annotation = {
            "segmentation": [],
            "file_name": jsonData["Source data Info."]["source_data_ID"],
            "area": "",
            "iscrowd": "",
            "image_id": idNum,
            "bbox": bbox,
            "category_id": jsonData["Learning data info."]["annotation"][i]["class_id"],
            "id": annoIdNum
            }
    annoTmpDict.append(annotation)
    annoIdNum += 1
idNum += 1
```

- AI Hub Label을 CoCo Dataset Format의 Annotation으로 변형

2.불필요한 Category

```
def del_uc(self, coco):
    dataset = coco.dataset
    for idx, ann in enumerate(dataset['annotations']):
        if ann['category_id'] == "UC-06":
            del dataset['annotations'][idx]
    return dataset
```

- Uc-06은 객체가 아닌 상황 Label

3. 이미지 Resize 및 정규화

```
transforms = Compose([
    Resize((max_size, max_size)),
    ToTensor(),
    Normal()
])
```

- 이미지사이즈를 맞춰주고 정규화 적용

4.모델 구조 설명: FCOS

FCOS 모델의 특징

<모델요약>

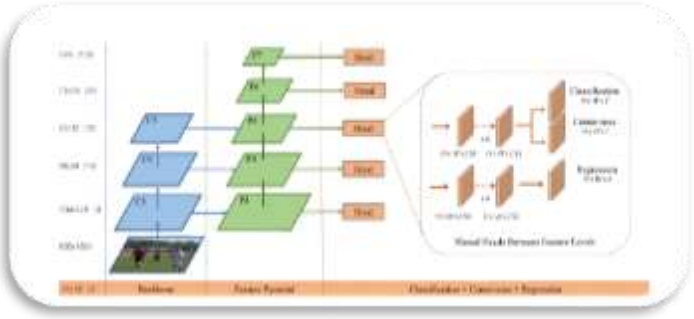
1. 중심점부터 바운딩 박스의 경계까지의 거리 예측
2. Multi-Label Prediction With FPN
3. Center-ness 제안

1. 중심점부터 바운딩 박스 경계까지의 거리 예측

- Anchor-based Detector은 Anchor Box와의 Offset을 예측하지만, FCOS은 중심점 (x,y)부터 예측한 바운딩 박스의 경계까지의 거리를 예측
- 만약 (x,y)가 GT(Ground-Truth) 범위 안에 존재하고, Class가 동일하면 Positive로 간주한 후 (x,y)를 기준으로 바운딩 박스 경계 까지의 거리를 Regress
- 만약 (x,y)가 여러 GT안에 존재한다면 Ambiguous Sample로 간주하고 뒤에서 설명할 Multi-Level Prediction with FPN에서 처리

4.모델 구조 설명: FCOS

2. Multi-Label Prediction With FPN



- Anchor-based Detector은 FPN의 각 Level 마다 서로 다른 크기, 종횡비의 Anchor Box를 할당
- **FCOS는 Bounding Box Regression의 범위를 제한**

3. Center-ness 제안

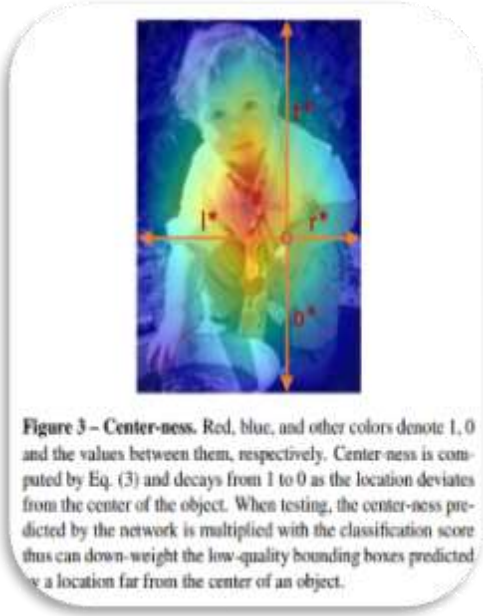


Figure 3 – Center-ness. Red, blue, and other colors denote 1, 0 and the values between them, respectively. Center-ness is computed by Eq. (3) and decays from 1 to 0 as the location deviates from the center of the object. When testing, the center-ness predicted by the network is multiplied with the classification score thus can down-weight the low-quality bounding boxes predicted by a location far from the center of an object.

- Center-ness는 Low-Quility Predicted Bounding Box를 제거하기 위해 사용

$$\text{centerness}^* = \sqrt{\frac{\min(l^*, r^*)}{\max(l^*, r^*)} \times \frac{\min(t^*, b^*)}{\max(t^*, b^*)}}.$$

- Center-ness는 중심점과의 거리를 정규화 만약 중심점과 가깝다면 1에 가까운 값, 중심점과 멀리 존재한다면 0에 가까운 값 산출

<Yolo와 차이점>

- **Anchor Free의 대표적인 방법으로 Anchor Box를 제거함으로써 FCOS는 복잡한 Anchor box 관련 연산을 없앰 또한 어떤 Anchor Box를 사용해야 하는 지 선택하는 고민을 하지 않아도 됨**
  → 따라서 연산량을 줄일 수 있는 장점이 존재

4.모델링:사용코드와 그 특징

코드 특징

<공개코드>

<사용한 코드의 특징>

```
class Helmet_Dataset(Dataset):
    def __init__(self, path, mode, transforms=None):

        # transforms
        self.transforms = transforms
        # file path
        self.path = os.path.join(path, 'data', mode)
        self.ann_path = os.path.join(self.path, '_annotations.coco.json')
        # coco function
        self.coco = COCO(self.ann_path)
        # image num
        self.image_ids = list(self.coco.imgToAnns.keys())
        print(f'{mode}_dataset length : {len(self.image_ids)}')
```

<공개코드 특징>

- Torchvision 사이트에 공개되어있는 FCOS모델 사용
- 복잡한 설치나 환경설정없이 Image 사이즈나 형식을 맞춰주면 바로 사용이 가능
- ResNet-50-FPN 모델을 BackBone 모델로 사용

- Input Data와 Json Annotation 파일을 CoCo Dataset 형태로 맞춰주면 어떠한 데이터셋에 대해서도 모델에 적용할 수 있도록 코드 구성

4.모델링:모델 학습과정 및 결과

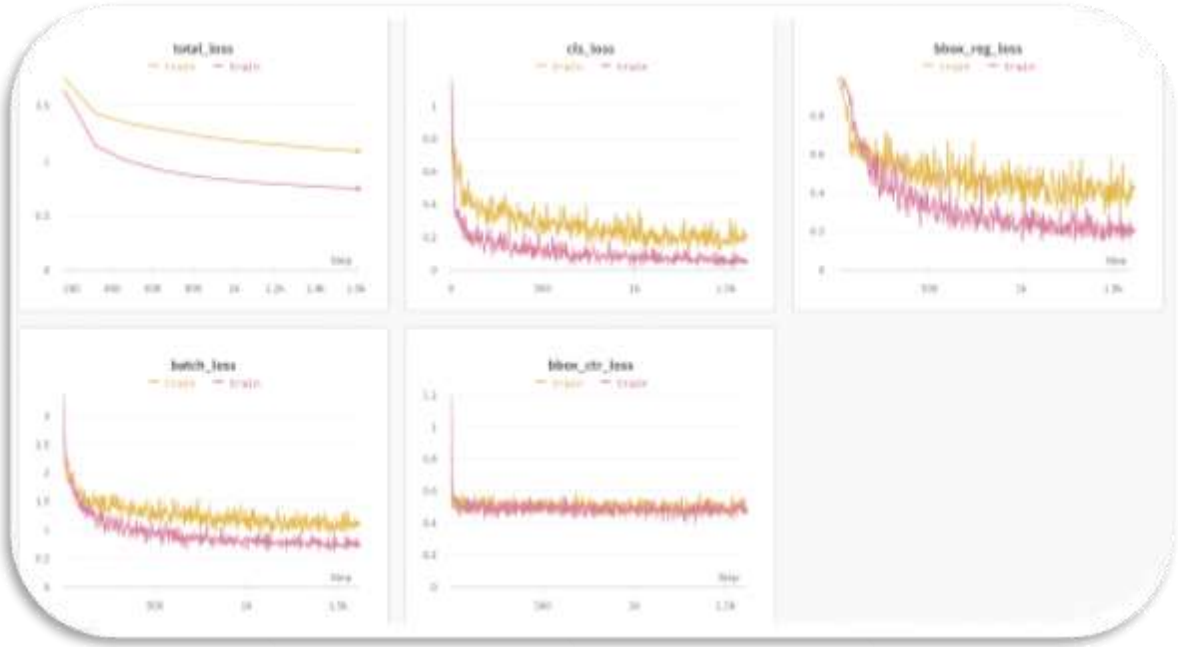
물류창고 데이터셋

<HyperParmeter>

```
# warehouse train
!python3 '/content/drive/MyDrive/KMU/4학년 1학기/사인지/프로젝트/train.py'\
'--batch_size=8'\
'--lr=5e-2'\
'--max_size=400'\
'--num_workers=2'\
'--num_epoch=10'\
'--num_classes=32'\
"--base_path=/content/drive/MyDrive/KMU/4학년 1학기/사인지/프로젝트"\
"--data=warehouse"\
"--save_path=model_3.pth"\
"--wandb_project_name=CV_project"\
"--wandb_entity_name=jimin_"\
"--wandb_name=train"
```

- Batch Size: 8
- Learning Rate: 5e-2
- Num Epoch: 10
- Max Size: 400
- Num Worker: 2

<Training Loss>



- Parameter Tuning 전과 후의 Training Loss
- Tuning 후의 핑크색 Loss가 확연히 더 떨어지는 것을 볼 수 있음
- Total Loss는 0.7 수준까지 학습 수행

4.모델링:모델 학습과정 및 결과

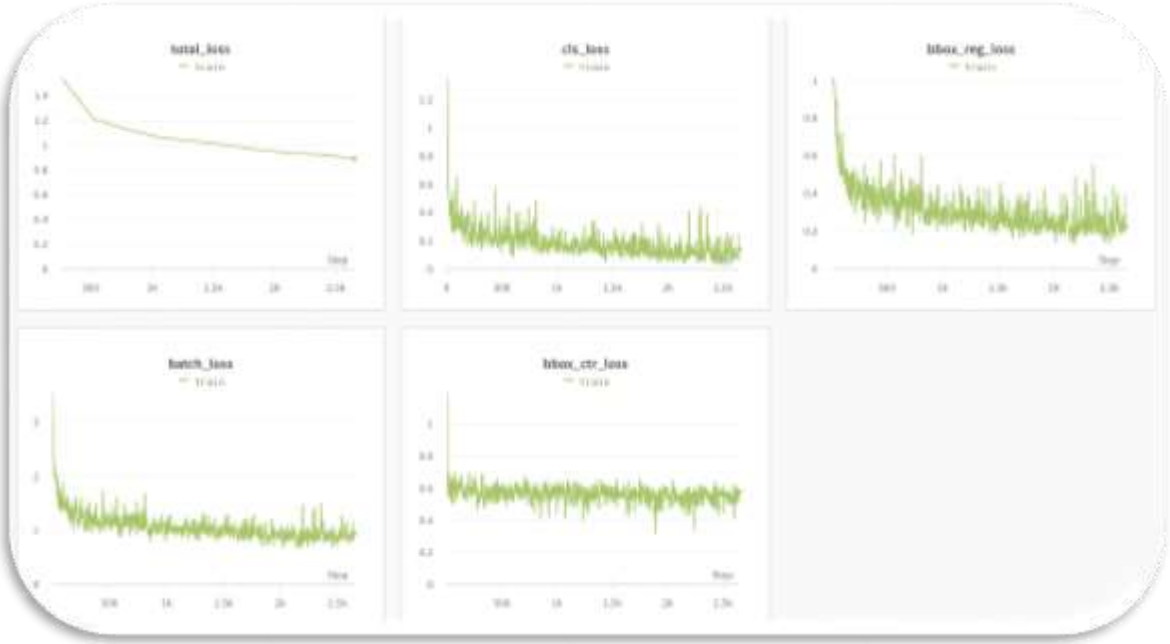
안전모 데이터셋

<HyperParmeter>

```
[4] # helmet train
!python3 './drive/MyDrive/KMU/4학년 1학기/시인지/프로젝트/train.py'\
'--batch_size=8'\
'--lr=1e-3'\
'--max_size=400'\
'--num_workers=2'\
'--num_epoch=10'\
'--num_classes=4'\
"--base_path=/content/drive/MyDrive/KMU/4학년 1학기/시인지/프로젝트"\
"--data=helmet"\
"--save_path=model_1.pth"\
"--wandb_project_name=CV_project"\
"--wandb_entity_name=jimin_"\
"--wandb_name=train"
```

- Batch Size: 8
- Learning Rate: 1e-3
- Num Epoch: 10
- Max Size: 400
- Num Worker: 2

<Training Loss>



- Classification / Regression 전반에 대한 Loss
- Total Loss 0.9 까지 학습 수행

4.모델링:모델 학습과정 및 결과

각 데이터셋에 대한 평가지표: MAP

<Mean Average Precision>

- 각각의 클래스에 대한 AP의 평균
- MAP_50: IOU > 50 이상의 클래스에 대한 평균
- MAP_75: IOU > 75 이상의 클래스에 대한 평균

<안전모 데이터셋 MAP>

```
'map': tensor(0.2507),
'map_50': tensor(0.4927),
'map_75': tensor(0.2239),
'map_large': tensor(0.1898),
'map_medium': tensor(0.3054),
'map_per_class': tensor(-1.),
'map_small': tensor(0.1750),
'mar_1': tensor(0.0979),
'mar_10': tensor(0.3123),
'mar_100': tensor(0.3219),
'mar_100_per_class': tensor(-1.),
'mar_large': tensor(0.2864),
'mar_medium': tensor(0.3919),
'mar_small': tensor(0.2512)}
```

MAP: 0.2507

MAP_50: 0.4927

MAP_75: 0.2239

<물류창고 데이터셋 MAP>

```
{'map': tensor(0.1450),
 'map_50': tensor(0.3910),
 'map_75': tensor(0.0960),
 'map_large': tensor(0.0269),
 'map_medium': tensor(0.1625),
 'map_per_class': tensor(-1.),
 'map_small': tensor(0.1283),
 'mar_1': tensor(0.1201),
 'mar_10': tensor(0.2212),
 'mar_100': tensor(0.2258),
 'mar_100_per_class': tensor(-1.),
 'mar_large': tensor(0.0264),
 'mar_medium': tensor(0.2858),
 'mar_small': tensor(0.1943)}
```

MAP: 0.145

MAP_50: 0.391

MAP_75: 0.096

5.위험 감지 알고리즘:정적

정적객체 위험 알고리즘

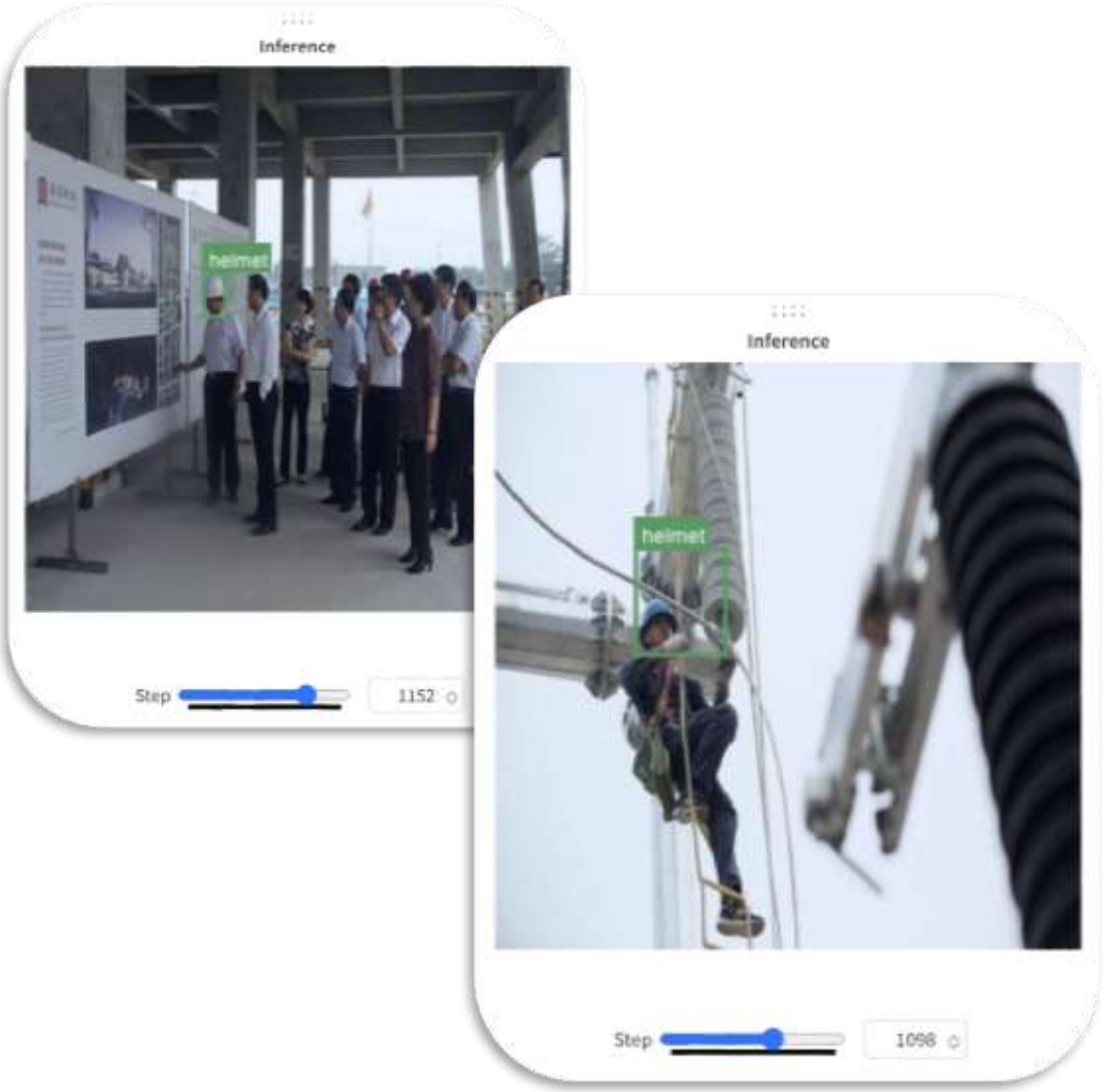
1. 정적객체 선정(안전모, 고글, 장갑 등등)

2. 정적객체 탐지

3. 작업자 Bounding Box에 정적객체 Bounding Box가 포함되어 있는지 판단

4. 포함되어 있는 정도를 임의의 Threshold 설정

5. Threshold이하는 위험하다고 판단



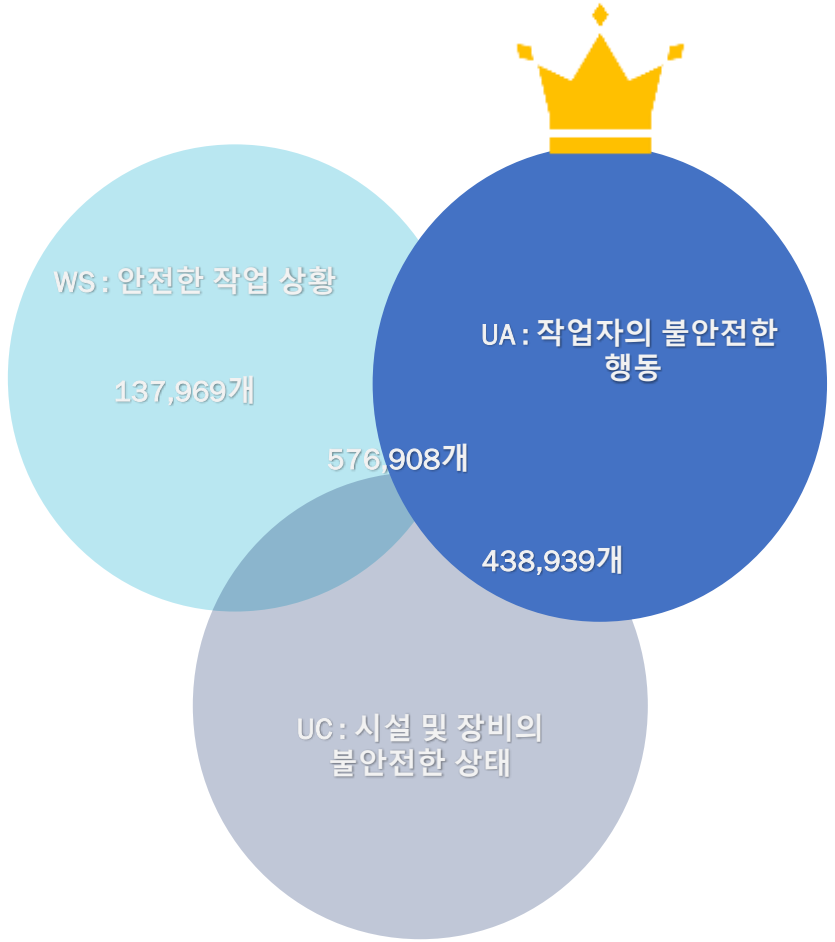
5.위험 감지 알고리즘:동적

동적객체 위험 알고리즘

안전한 행동을 제외하고 위험상황에 대한 시뮬레이션 정보 샘플링 하여 학습 진행

따라서 샘플링한 모든 상황의 객체들에 대해 지게차 같은 각별한 주의가 필요한 객체를 선정한 후,
이미지 내의 특정 객체에 대해 유클리디안 거리를 구하고

산출된 거리가 위험하다고 Baseline 설정 가능



5.위험 감지 알고리즘:동적

동적객체 위험 알고리즘

선정한 동적 객체(지게차)와 다른 객체 사이의 유클리디안 거리 산출

<지게차와 이물질의 경우>

<지게차와 작업자의 경우>

지게차와 작업자의 경우
탑승한 작업자를 고려해야 하므로
작업자의 Center와 지게차의 Center가
어느 정도를 넘어갈 때만 다른 객체로 인식

동적 객체와 비교 객체 선정

↓

동적 객체와 비교객체의 거리 계산

↓

해당 거리를 유클리디안 방식으로 평균

↓

해당 평균값을 Threshold로 설정

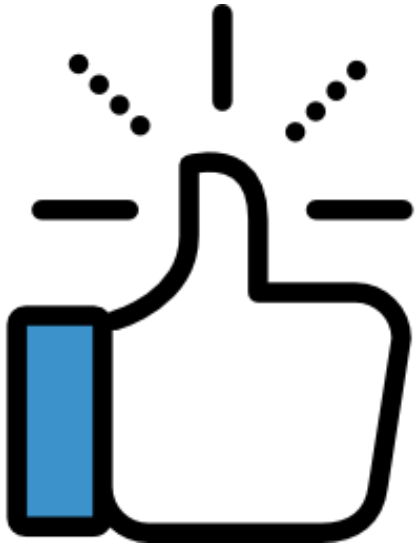
6.결론 및 시사점

결론 및 시사점

- 사람, 트럭, 지게차 등 동적객체와 안전모 정적객체 데이터셋을 구축
- FCOS를 활용하여 정적객체와 동적객체를 동시에 고려하는 위험감지 모델을 만듦
- 위험알고리즘을 적용하여 작업자의 안전을 보장
- 유클리디안 거리 Threshold값이 다른 연구에 기준점 제공

보완점

- 다양한 상황 데이터셋 구축
- 컴퓨팅 파워 부족 아쉬운 성능
- 위험알고리즘 적용



THANK YOU

참고문헌

<참고논문>

1. FCOS: Fully Convolutional One-Stage Object Detection
Zhi Tian, Chunhua Shen, Hao Chen, and Tong He

2. Animal detection and classification from camera trap image using different main stream object detection architectures
(Mengyu Tan 외8명)

3. YOLO-v3을 활용한 건설 장비 주변 위험 상황 인지 알고리즘 개발
심승보(한국건설기술연구원) , **최상일**(한국건설기술연구원)

4. 스마트건설 현장에서 개인 보호장비 검출의 개선 방법
(박용석 외2명)

Appendix

```python
import torch
import numpy as np
import os
import json
from pycocotools.coco import COCO

from torch.utils.data import Dataset

class Helmet_Dataset(Dataset):
    def __init__(self, path, mode, transforms=None):

        # transforms
        self.transforms = transforms
        # file path
        self.path = os.path.join(path, 'data', mode)
        self.ann_path = os.path.join(self.path, '_annotations.coco.json')
        # coco function
        self.coco = COCO(self.ann_path)
        # image num
        self.image_ids = list(self.coco.imgToAnns.keys())
        print(f'[mode]_dataset length : {len(self.image_ids)}')

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

    def __getitem__(self, idx):
        image_id = self.image_ids[idx]
        file_name = self.coco.loadImgs(image_id)[0]['file_name']
        file_name = f'{self.path}/{file_name}'
        image = Image.open(file_name).convert('RGB')

        annot_ids = self.coco.getAnnIds(imgIds=image_id)
        annots = [x for x in self.coco.loadAnns(annot_ids) if x['image_id'] == image_id]

        boxes = np.array([annot['bbox'] for annot in annots], dtype=np.float32)
        boxes[:, 2] = boxes[:, 0] + boxes[:, 2]
```

- Dataset.py 중 일부 코드

```python
import torch
import torchvision
import torchvision.models as models
from torchvision.models.detection import FCOS
from torchvision.models.detection.anchor_utils import AnchorGenerator

def fcos(num_classes):
    backbone = models.mobilenet_v2(weights=models.MobileNet_V2_Weights.DEF
    backbone.out_channels = 1280

    anchor_generator = AnchorGenerator(
        sizes=((8,), (16,), (32,), (64,), (128,)),
        aspect_ratios=((1.0,),)
    )

    model = FCOS(
        backbone,
        num_classes=num_classes,
        anchor_generator=anchor_generator,
    )
    return model

if __name__ == '__main__':
    torch.manual_seed(0)

    model = fcos(3)

    model.eval()
    x = [torch.rand(3, 300, 400)]
    predictions = model(x)

    boxes, scores, labels = predictions[0].values()
    print(len(boxes))
    print(len(scores))
    print(len(labels))
```

- Model.py 중 일부 코드

```python
    train_loader = DataLoader(
        train_dataset, batch_size=batch_size, shuffle=True,
        num_workers=num_workers, collate_fn=collate_fn)
    val_loader = DataLoader(
        val_dataset, batch_size=batch_size, shuffle=False,
        num_workers=num_workers, collate_fn=collate_fn)
else:
    train_dataset = Helmet_Dataset(base_path, 'train', transforms=tra
    # val_dataset = Helmet_Dataset(base_path, 'val', transforms=trans

    train_loader = DataLoader(
        train_dataset, batch_size=batch_size, shuffle=True,
        num_workers=num_workers, collate_fn=collate_fn)
    # val_loader = DataLoader(
    #     val_dataset, batch_size=batch_size, shuffle=False,
    #     num_workers=num_workers, collate_fn=collate_fn)

# ---------- create model ----------
model = fcos(num_classes)

# backbone network(resnet+fpn) requires_grad = False
for name, param in model.named_parameters():
    if name.split('.')[0] in ['backbone']:
        pass
    else:
        param.requires_grad = False

model.to(device)

# ---------- Optimizer ----------
params = [p for p in model.parameters() if p.requires_grad]
optimizer = optim.Adam(
    params, lr=lr, weight_decay=1e-5)

scheduler = optim.lr_scheduler.StepLR(optimizer, step_size=3, gamm
```

- Train.py 중 일부 코드