2024 세종테크노파크 대학생 AI 경진대회

< SafeWalk AI: 도로 돌발상황 관제를 위한 AI 기반 보행자 감지 시스템 >

-최종 보고서

소 속	국립 창원대학교
팀 원	박진성 김성연 박혜성

창원대학교

목 차

1. 서 론

1. 1 배경 및 목적

21 세기의 진입과 함께 우리는 기술과 데이터가 주도하는 새로운 시대의 문턱에 서 있음. 특히, 인공지능(AI)은 모든 산업 분야에 걸쳐 혁신적인 변화를 가져오고 있으며, 그 중심에는 '데이터'가 자리를 잡는 중임. 본 공모전인 '2024 세종테크노파크 대학생 AI 경진대회'는 이러한 시대적 변화를 반영하여, 미래의 주역이 될 대학생들에게 실질적인 데이터 분석 경험을 제공하고 AI 기술의 발전을 촉진하는데 중요한 목적을 두고 있음. 우리는 우리의 기술적 능력을 시험해볼 수 있는 기회를 갈망하고 있었고, 이 대회는 그러한 우리의 열정에 부응하는 완벽한 기회를 제공해주고 있음.

이 공모전은 세종테크노파크에서 주최하였으며, 세종 자율주행 시범운행지구 운영을 통해 자율주행 차량 데이터와 관제 데이터를 보유하고 있음. 이번 공모전에는 이러한 데이터셋을 제공하여 도로 돌발 상황 관제 데이터를 활용하여 횡단, 무단횡단 보행자를 감시하는 AI 경진 대회를 진행함.

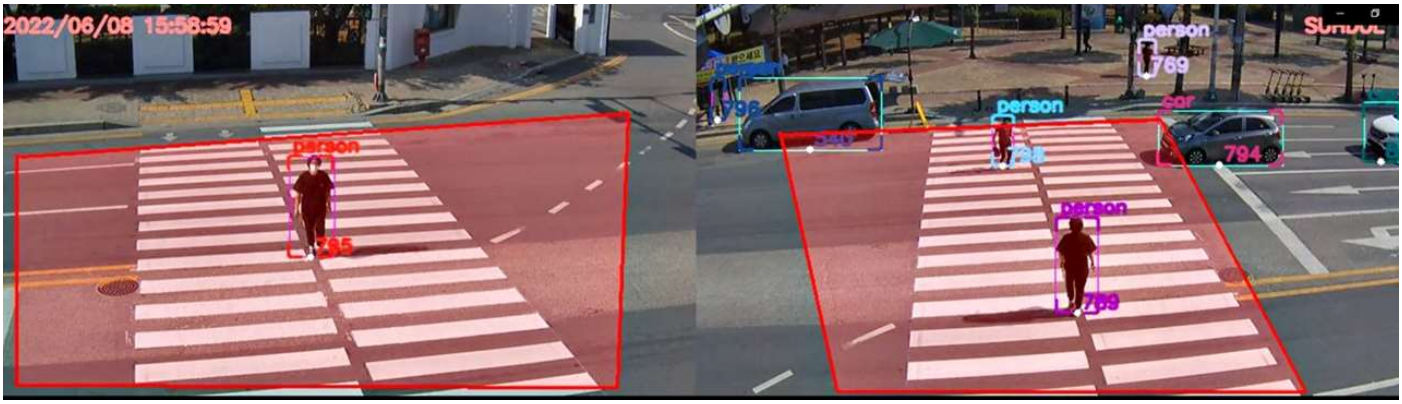


그림 1 - 보행자 감지 예시

그림 2 - 킥보드 교통사고 통계

출처: https://kwnews.co.kr/page/view/2023040610360389448 – 강원일보 (2023.04,06)

시간이 지남에 따라 보행자의 종류도 다양해 지고 있음.(ex/ 킥보드, 자전거 등등) 이에 발 맞춰 횡단보도를 킥보드와 자전거 등으로 횡단하는 경우도 빈번한데, 이 경우에도 구별할 수 있는 모델이 필요하다고 판단함.

그러므로 제공되어지는 데이터 셋 외에도 추가적인 데이터를 첨가하여 킥보드와 자전거를 구별하는 객체 탐지 능력을 학습시키고자 함

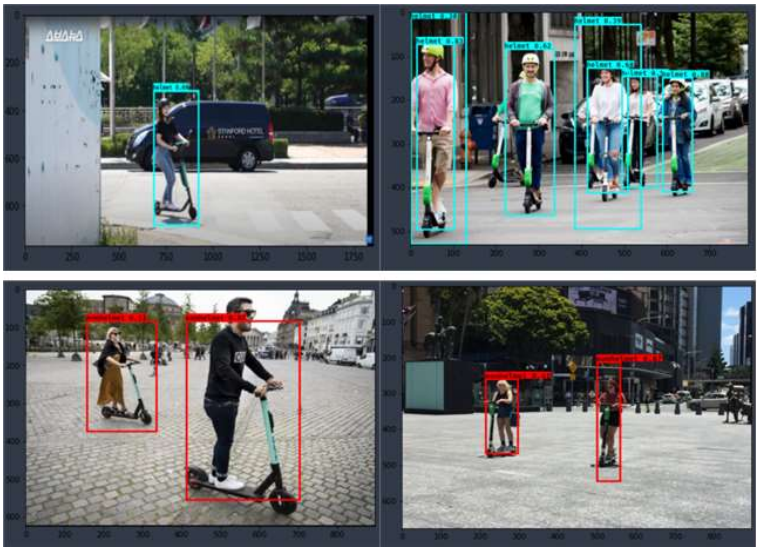


그림 3 - 객체 탐지 예시

1. 2 데이터셋의 구성

제공된 데이터셋은 크게 두 가지의 형태로 이루어져 있음. 첫 번째는 객체 탐지(Object Detection)데이터로 원천데이터, 라벨데이터 1, 라벨데이터 2 로 구성되어 있음.
원천 데이터는 JPG 형식으로 1920x1080 의 해상도를 가지며, 총 600 장의 데이터가 존재함.

라벨데이터들은 보행자 객체의 Bounding Box 데이터로써 데이터는
[라벨] [객체 중심좌표(X)] [객체 중심좌표(Y)] [객체 너비(W)] [객체 높이(H)]의 형식을 가짐.

라벨 데이터 1 은 모든 보행자를 나타내며 라벨 데이터 2 는 횡단보도 보행자와 무단횡단 보행자를 구별하여 나타내고 있음.

$$[X\ Y\ W\ H] = \left[\frac{X_0}{1920}\ \frac{Y_0}{1080}\ \frac{W_0}{1920}\ \frac{H_0}{1080}\right]$$

이러한 형식으로 정규화 되어 TXT 형식으로 저장되어 있음.

총 1592 + 889 개의 라벨링 데이터가 존재함.

두 번째는 영상 분할(Image Segmentation) 데이터로 원천데이터, 라벨데이터로 구성되어 있음. 원천 데이터는 JPG 형식으로 1920x1080 의 해상도를 가지며, 총 396 장의 데이터가 존재함.

라벨 데이터는 화소마다 라벨 인덱스 값을 가지는 영상 분할 라벨 지도로써 PNG 형식으로 총 396 장의 데이터가 존재하며, 0 은 배경, 1 은 도로, 2 는 횡단보도, 3 은 캡션으로 라벨링되어 있음.

1. 3 개발 일정

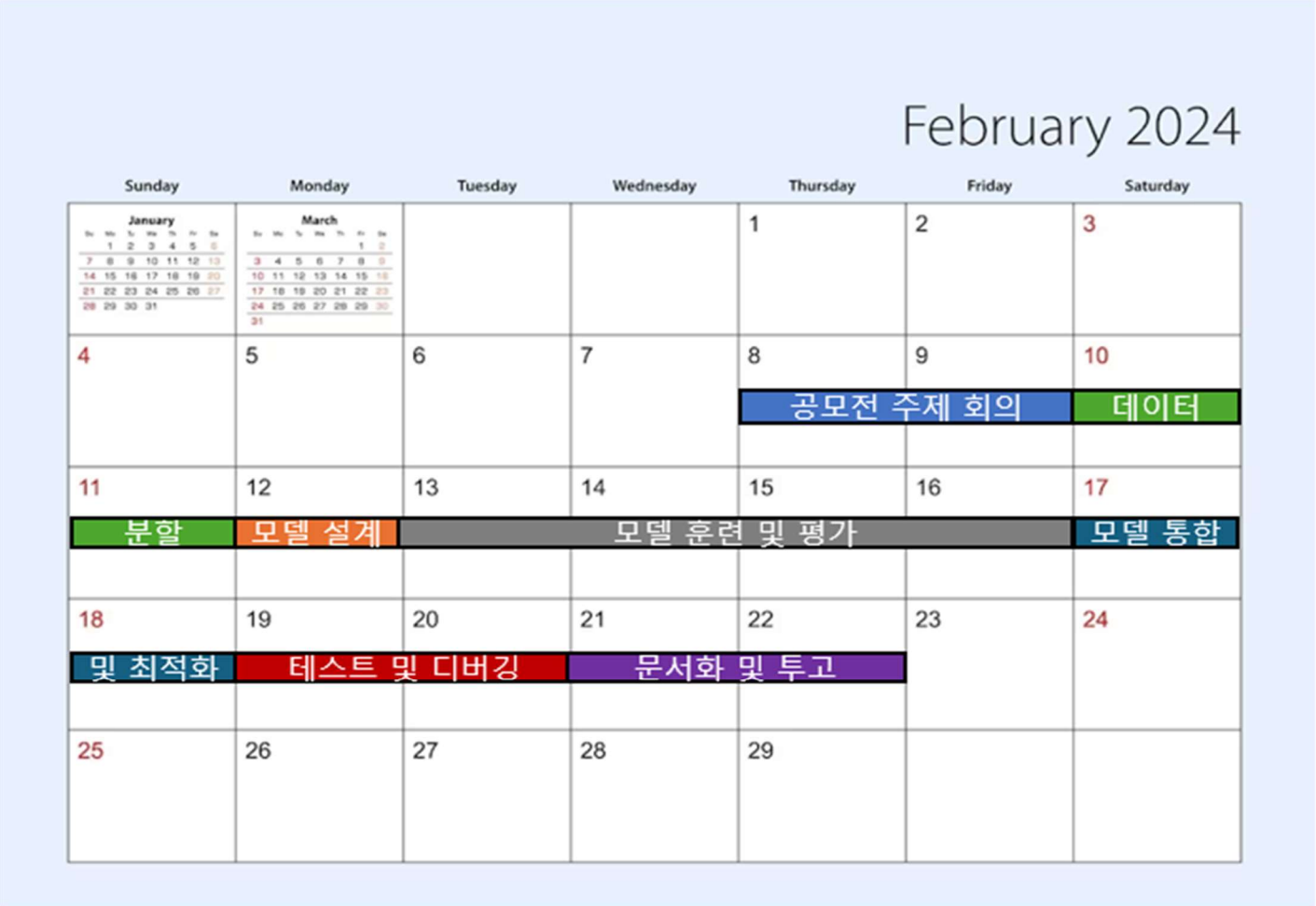


그림 4 - 개발 일정표

1)	요구사항 정의 및 리서치 (2월 8일 – 2월 9일)
목표	프로젝트의 범위, 목표, 필요한 데이터셋, 기술 스텍을 정의함.
작업	프로젝트 목표 및 요구사항 명확하게 하기 사용할 영상 분할 및 객체 탐지 기술에 대한 리서치 필요한 데이터셋 조사 및 수집 계획 수립

2)	데이터 준비 (2월 10일 – 2월 11일)
목표	프로젝트에 사용될 데이터셋을 준비하고, 초기 데이터 처리를 수행함.
작업	데이터셋 다운로드 및 정리 데이터 전처리 및 분할 (훈련, 검증, 테스트 세트)

3)	모델 설계 (2월 12일)
목표	영상 분할과 객체 탐지를 위한 모델 아키텍처 설계함.
작업	사용할 모델 아키텍처 결정 모델 구현을 위한 프로그래밍 환경 설정

4)	모델 훈련 및 평가 (2월 13일 – 2월 16일)
목표	설계한 모델을 훈련시키고, 성능을 평가함.
작업	모델 훈련을 위한 코드 구현 훈련 과정 모니터링 및 하이퍼파라미터 조정 검증 데이터셋을 사용한 모델 성능 평가

5)	모델 통합 및 최적화 (2월 17일 – 2월 18일)
목표	영상 분할과 객체 탐지 모델을 통합하고, 시스템 전체의 성능 최적화를 수행함.
작업	두 모델의 결과를 결합하는 로직 구현 성능 최적화 및 버그 수정

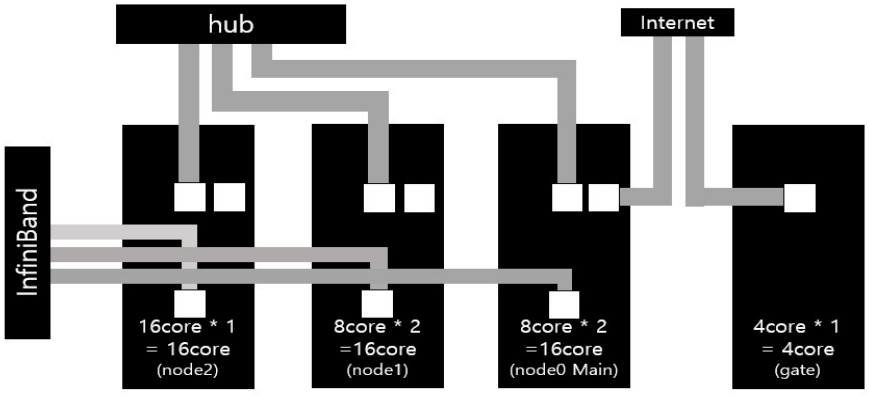
6)	테스트 및 디버깅 (2월 19일 – 2월 20일)
목표	완성된 시스템을 테스트하고, 문제가 있는 부분을 수정함.
작업	테스트 케이스 실행 및 결과 분석 발견된 문제 수정

7)	문서화 및 배포 준비 (2월 21일 – 2월 22일)
목표	프로젝트 문서를 작성하고, 사용자가 시스템을 사용할 수 있도록 배포 준비.
작업	코드 주석 추가 및 문서화 사용 설명서 작성 배포를 위한 패키징 및 가이드 준비

1. 4 개발 환경 소개

1. 4. 1 시스템사양

Laboratory Hardware Spec



Name	Hardware Specification
Node 0 main	CPU: Intel® Xeon Silver 4208 (8 cores 16 threads) * 2 EA GPU: NVIDIA GeForce GTX 3090 * 2 EA RAM: 128GB(16GB * 8EA) SSD: 960GB HDD: 4TB OS: CentOS 7
Node 1	CPU: Intel® Xeon Silver 4208 (8 cores 16 threads) * 2EA GPU: NVIDIA GeForce GTX 3090 * 2 EA RAM: 128GB(16GB * 8EA)
Node 2	CPU: Intel® Xeon SPS-4314 (16 cores 32 threads) * 1EA GPU: NVIDIA GeForce GTX 3090 * 2 EA RAM: 128GB(16GB * 8EA)
Gate	CPU: Intel® i7-7700 CPU (4 cores 8 threads) GPU: GeForce GTX 1060 3GB RAM: 8GB HDD: 1TB SDD: 100GB OS: CentOS 7

그림 5 – LAB 서버

FEATURE	CAE-100SWE24UF1
Product Description	Intel Omni-Path Edge Switch 100 Series - switch - 24 ports - rack-mountable
Device Type	Switch - 24 ports
Port/Expansion Slot Details:	24 x 100 Gigabit Ethernet Expansion Slot
Ethernet Technology:	100 Gigabit Ethernet
Media Type Supported:	*Optical Fiber*
Power (Typ./Max) - Watts DC Copper	AC 120/230 V (50/60 Hz)
Performance	Switching capacity: 4.8 Tbps ¦ Latency: 100 - 110 ns

Name	Hardware Specification
Node 0 main	CPU: Intel® Xeon Silver 4208 (8 cores 16 threads) * 2 EA GPU: NVIDIA GeForce GTX 3090 * 2 RAM: 128GB(16GB * 8EA) SSD: 960GB HDD: 4TB OS: CentOS 7
Node 1	CPU: Intel® Xeon Silver 4208 (8 cores 16 threads) * 2EA GPU: NVIDIA GeForce GTX 3090 * 2 RAM: 128GB(16GB * 8EA)
Node 2	CPU: Intel® Xeon SPS-4314 (16 cores 32 threads) * 1EA GPU: NVIDIA GeForce GTX 3090 * 2 EA RAM: 128GB(16GB * 8EA)
Gate	CPU: Intel® i7-7700 CPU (4 cores 8 threads) GPU: GeForce GTX 1060 3GB RAM: 8GB HDD: 1TB SDD: 100GB OS: CentOS 7

표 1 - 서버 스펙

1) 개발 도구 및 소프트웨어

Development Tool/Software	Version	Installation Command
Visual Studio Code	1.85.2	-
Extension: Remote – SSH	0.107.01	-
Python	3.10.13	-
Anaconda	-	Install Anaconda
-	-	conda create -n 'your_virtual_env_name' python==3.10.13
Jupyter	-	pip install jupyter
-	-	conda activate 'your_virtual_env_name'
ultralytics	-	pip install ultralytics
OpenCV	-	pip install opencv-python
Matplotlib	-	pip install matplotlib
PyTorch	1.11.0	pip install torch==1.11.0
Torchvision	0.12.0	pip install torchvision==0.12.0
NumPy	1.23.5	pip install numpy==1.23.5

표 2 - Library Requirements

2) 개발 환경 구축

※ 아나콘다 설치 -> conda create -n '원하는 가상환경 이름' python==3.10.13

※ pip install jupyter

※ conda activate '원하는 가상환경 이름'

※ pip install ultralytics

※ pip install opencv-python

※ pip install matplotlib

※ pip install torch==1.11.0

※ pip install torchvision==0.12.0

※ pip install numpy==1.23.5

2. AI 학습모델 개발 과정 (본 론)

2. 1 목표

공모전에서 최종적으로 바라는 것은 도로 돌발상황 관제 데이터를 활용하여 횡단, 무단횡단 보행자를 검지하는 알고리즘이므로 객체 탐지 모델 하나, 영상 분할 모델 하나, 총 두 개의 모델을 사용하여 횡단, 무단횡단 보행자를 검지할 수 있도록 만들고자 함.

사진을 입력을 받으면 객체 탐지 모델을 통해 객체를 먼저 탐지한 다음 좌표 값을 저장하고 그와 동시에 영상 분할 모델도 같이 작동하여 사진에서 도로와 횡단보도를 구별해서 해당 이미지의 픽셀마다 라벨링을 하여 객체의 좌표 값과 해당 픽셀의 위치가 도로에 존재하는지 혹은, 횡단보도에 존재하는지를 판단하여 보행자의 횡단, 무단횡단을 판별하는 모델을 만들고자 함.

데이터셋이 제공된 이상 서로 비슷한 알고리즘이 많이 투고될 것을 예상함. 이를 위한 차이점을 두고자 앞서 서술했듯, 최종적으로 나온 방안으로는 보행자뿐만이 아닌 자전거, 킥보드 등 다양한 탈 것을 구별할 수 있도록 하고자 하며, 정확성에 중점을 두고 개발하고자 함.

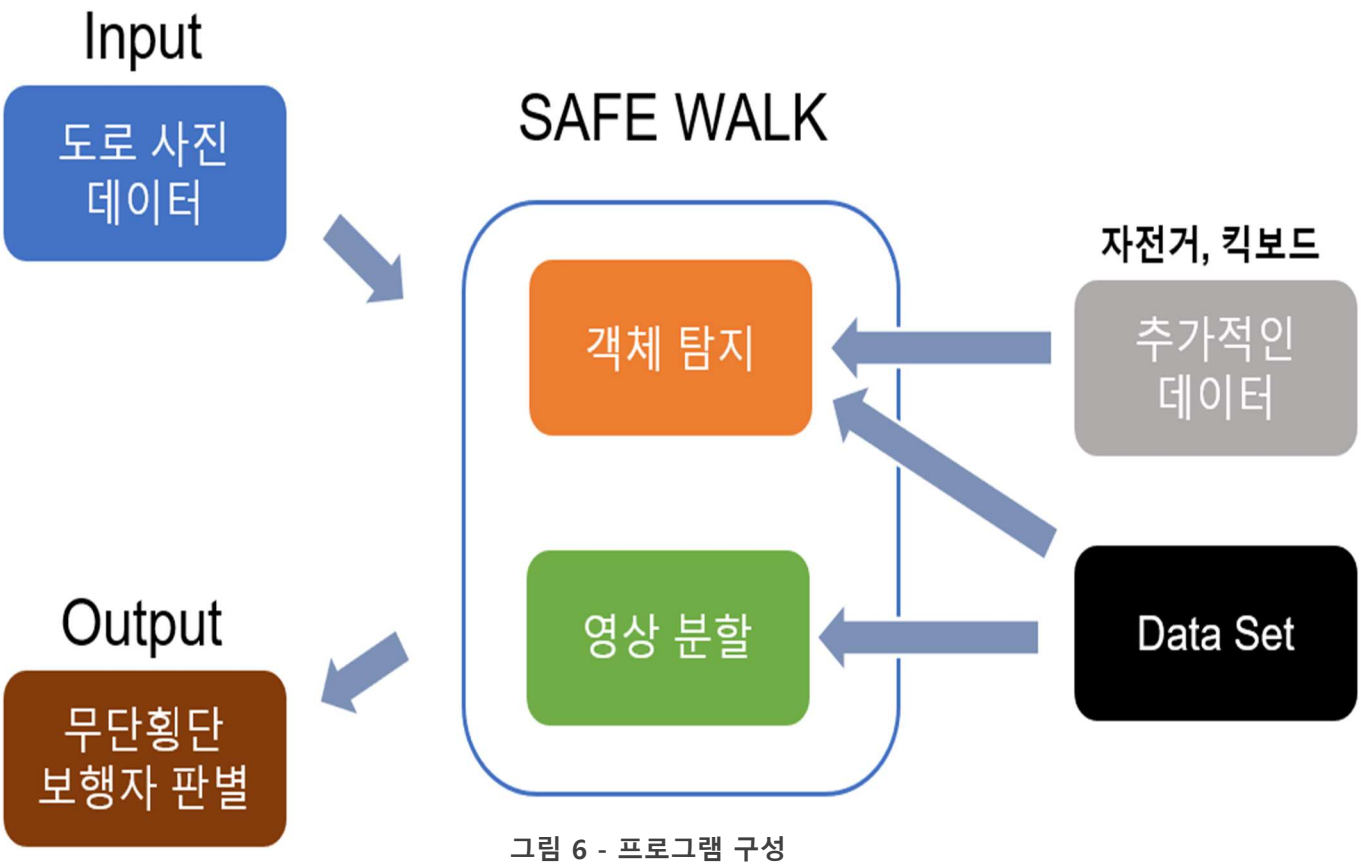


그림 6 - 프로그램 구성

2. 2 데이터 전처리 및 분석 방법

2. 2. 1 객체 탐지 (Object Detection)

최신의 객체 감지 모델인 YOLOv8X 를 활용하여, 실시간으로 다양한 객체를 정확하게 감지하는 시스템을 구축함. RoboFlow 에서 제공받은 데이터와 자체 수집한 커스텀 데이터를 결합하여 모델을 학습시켜, 기존 모델 대비 더 높은 학습률과 달성하는 것을 목표로 함.

1) YOLOv8 모델 소개

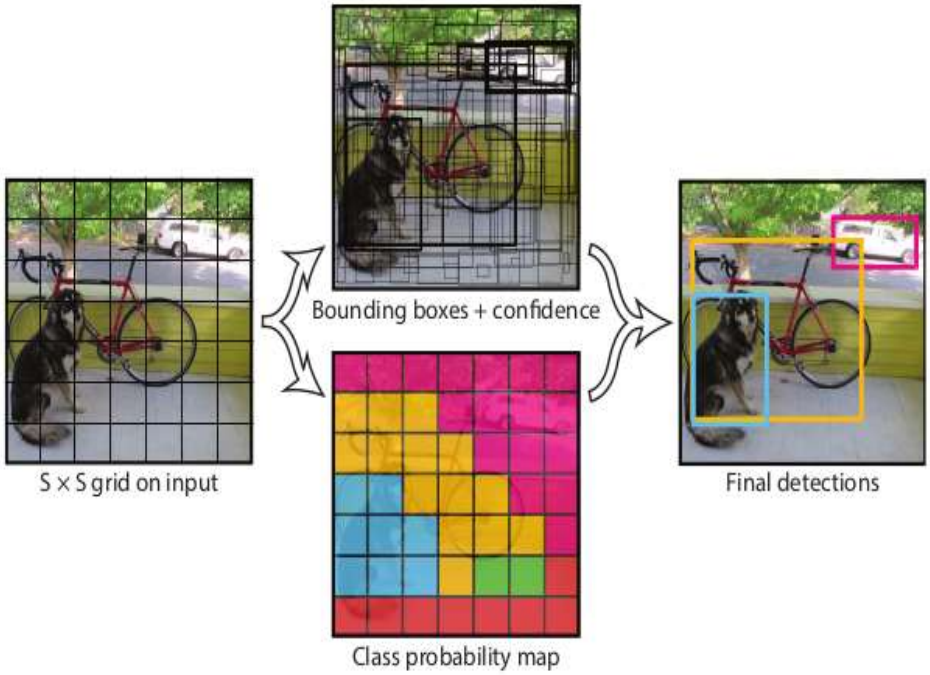


그림 7 – YOLO 학습 원리

출처: https://dotiromoook.tistory.com/24

YOLOv8 은 실시간 객체 감지를 위해 설계된 딥러닝 모델로, 높은 속도와 정확도를 동시에 제공함. 특히 YOLOv8X 버전은 모델의 크기와 아키텍처를 최적화하여 이전 버전보다 향상된 성능을 보임. 자동화된 라벨링, 다양한 객체 감지 능력 등은 YOLOv8X 를 현대 컴퓨터 비전 문제에 이상적인 선택으로 만듦.

1. **S x S 그리드 분할**: 입력 이미지는 S x S 그리드로 나누어집니다. 이 예에서, 각 셀은 이미지의 특정 영역을 대표하며, 객체의 중심이 그 셀에 놓이면 그 셀은 객체를 감지할 책임이 있었음.

2. **Bounding Boxes + Confidence**: 각 그리드 셀은 객체의 경계 상자(bounding boxes)를 예측하고, 이들 각각에 대한 신뢰도(confidence score)를 계산함. 신뢰도는 그 상자가 객체를 포함하고 있을 확률과 예측된 상자와 실제 객체의 상자가 얼마나 잘 일치하는지를 나타냄.

3. **Class Probability Map**: 또한, 네트워크는 각 셀에 대한 클래스 확률을 예측함. 이는 각 셀이 특정 클래스의 객체(예: 자전거, 개)를 포함할 확률임.

4. **Final Detections**: 마지막으로, 신뢰도와 클래스 확률을 결합하여 최종 감지를 수행함. 높은 신뢰도를 가진 상자와 높은 클래스 확률을 가진 셀이 결합되어 객체의 위치와 클래스를 결정함.

2) 데이터 전처리

본 객체 탐지 모델에서는 세종테크노파크 공모전 주최 측 제공 데이터와 자체 수집한 커스텀 데이터를 사용함. 커스텀 데이터에 대한 정밀 라벨링은 RoboFlow 의 라벨링 도구를 통해 수행함. 데이터 증강 기술을 적용하여 모델의 일반화 능력을 향상시키는 전략을 채택함.

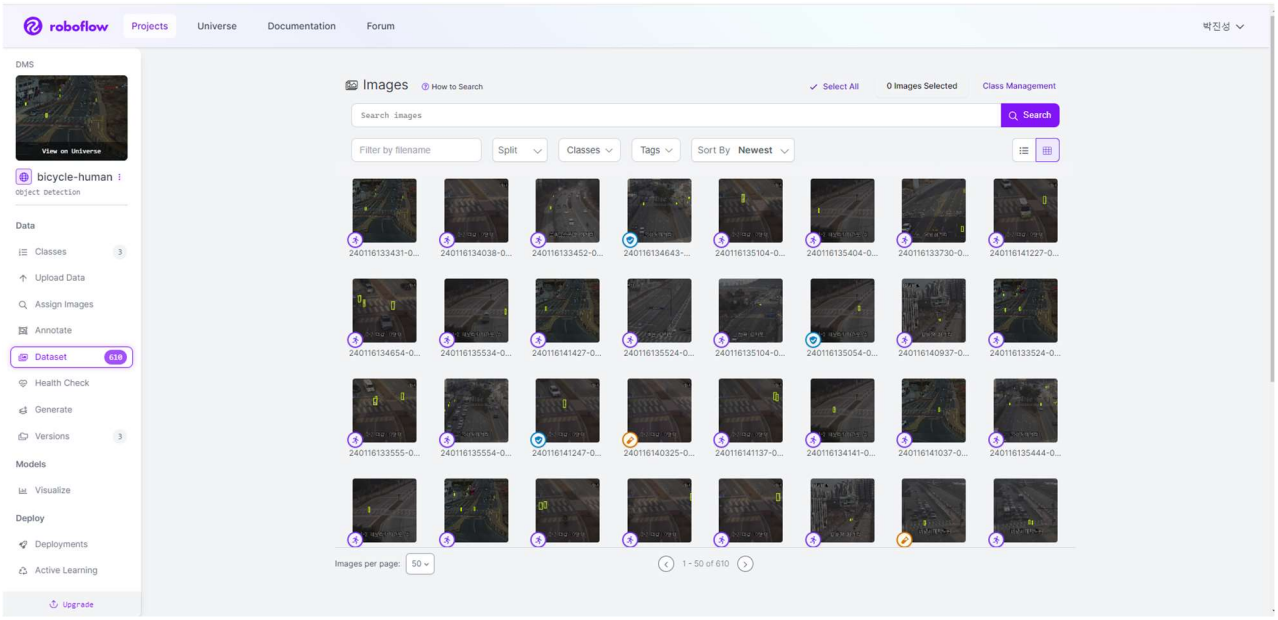


그림 8 - RoboFlow 를 활용한 라벨링 처리

3) 학습 과정

테크노파크 공모전 주최 측 제공 데이터를 학습용 데이터 70%, 검증용 데이터 20%, 테스트용 데이터 10%로 나누어 학습을 진행함.

```
from ultralytics import YOLO

# Load a pre-trained YOLOv8x model
model = YOLO('yolov8x.yaml').load('yolov8x.pt')

# Train the model with your dataset
results = model.train(data='/home/pjs/workspace/SEJONG/bicycle-human-3/data.yaml',
epochs=100, imgsz=640, patience=10)
```

그림 9 - YOLO 활용 코드

기존 yolov8x 에는 80 개의 클래스가 지정되어 있으나 커스터마이징을 함으로 써

Class 0 = person, Class 80 = bicycle, Class 81 = kick board 를 추가하여 총 82 개의 클래스가 존재함. 그 중에 추가한 클래스만 사용함.

yolov8x 모델의 가중치인 yolov8x.pt 파일을 불러와 커스터마이징 한 data.yaml 데이터셋을 추가로 학습시키는 코드임.

그림 10 - YOLO 를 사용한 객체 탐지

2. 2. 2 영상 분할 (Image Segmentation)

여기서 사용된 모델은 수정된 U-Net 임. U-Net 은 인코더(다운 샘플러)와 디코더(업샘플러)로 구성되는데 강력한 기능을 학습하고 학습 가능한 매개변수의 수를 줄이기 위해 사전 학습된 모델인 MobileNetV2 를 인코더로 사용함. 디코더의 경우 TensorFlow 예제 리포지토리의 pix2pix 예제에서 이미 구현된 업샘플 블록을 사용함.

```python
class Encoder(nn.Module):
    def __init__(self):
        super(Encoder, self).__init__()
        mobilenet_v2 = models.mobilenet_v2(pretrained=True)
        self.features = mobilenet_v2.features
        for param in self.features.parameters():
            param.requires_grad = False

    def forward(self, x):
        return self.features(x)
```

```python
class MobileNetV2_UNet(nn.Module):
    def __init__(self, num_classes=4):
        super(MobileNetV2_UNet, self).__init__()
        self.encoder = Encoder()
        self.decoder = nn.Sequential(
            UpSampleBlock(1280, 512, apply_dropout=True),
            UpSampleBlock(512, 256),
            UpSampleBlock(256, 128),
            UpSampleBlock(128, 64),
            nn.Conv2d(64, num_classes, kernel_size=1)
        )

    def forward(self, x):
        x = self.encoder(x)
        x = self.decoder(x)
        # 출력 크기 조정
        x = F.interpolate(x, size=(960, 544), mode='bilinear', align_corners=False)
        return x
```

```python
class UpSampleBlock(nn.Module):
    def __init__(self, in_channels, out_channels, norm_type='batchnorm', apply_dropout=False):
        super(UpSampleBlock, self).__init__()
        layers = [nn.ConvTranspose2d(in_channels, out_channels, kernel_size=4, stride=2, padding=1, bias=False)]
        if norm_type.lower() == 'batchnorm':
            layers.append(nn.BatchNorm2d(out_channels))
        if apply_dropout:
            layers.append(nn.Dropout(0.5))
        layers.append(nn.ReLU(inplace=True))
        self.upsample = nn.Sequential(*layers)

    def forward(self, x):
        return self.upsample(x)
```

그림 11 - 영상 분할 모델 코드

1) 데이터 전처리

이미지 데이터는 픽셀 값 범위를 모델 학습에 적합하게 조정하기 위해 정규화 됨. 정규화는 transforms.Normalize 를 사용하여 수행되며, 이는 이미지의 각 채널에 대해 평균과 표준편차를 사용하여 데이터를 정규화 함.

여기서 사용된 평균과 표준편차는 [0.485, 0.456, 0.406]과 [0.229, 0.224, 0.225]임. 이 값들은 일반적으로 ImageNet 데이터셋에 대한 사전 학습 모델에서 사용되는 값들임.

CustomTransform 클래스를 통해 이미지와 마스크에 대한 전처리 과정을 커스터마이징 함. 이 클래스는 리사이즈, 텐서 변환, 그리고 이미지 정규화를 포함한 전처리 파이프라인을 구현함. 마스크에 대해서는 크기 조정과 텐서 변환만 수행되며, 정규화는 적용되지 않음.

```python
class CustomTransform:
    def __init__(self):
        self.resize = transforms.Resize((960, 544))
        self.to_tensor = transforms.ToTensor()
        self.normalize = transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])

    def __call__(self, image, mask):
        image = self.resize(image)
        image = self.to_tensor(image)
        image = self.normalize(image)

        # 마스크는 크기만 조정하고, 텐서로 변환합니다. Normalize는 적용하지 않습니다.
        mask = self.resize(mask)
        mask = torch.from_numpy(np.array(mask, dtype=np.int64))
        return image, mask
```

그림 12 - CustomTransform 클래스 코드

2) 학습 과정

세종테크노파크 공모전 주최 측 제공 데이터를 학습용 데이터 70%, 검증용 데이터 25%,테스트용 데이터 5%로 나누어 학습을 진행함.

손실 함수로는 'nn.CrossEntropyLoss'가 사용되어 모델의 예측과 실제 마스크 간의 차이를 계산함. 최적화 알고리즘으로는 'Adam'이 사용되며, 학습률은 0.001 로 설정함.

```python
# 모델, 손실 함수 및 최적화 알고리즘 설정
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model = MobileNetV2_UNet(num_classes=4).to(device)
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)
```

그림 13 - 손실 함수 및 최적화 알고리즘

학습 과정은 여러 에폭(epoch)에 걸쳐 반복됨. 각 에폭은 모든 학습 데이터를 한 번씩 모델에 공급하는 과정으로 구성됨.

학습 단계: 각 배치에서, 모델의 예측을 계산하고, 손실 함수를 사용하여 손실을 계산함. 그런 다음 역전파를 수행하여 모델의 가중치를 조정함.

검증 단계: 학습 후, 검증 데이터셋을 사용하여 모델의 성능을 평가함. 이 단계에서는 모델의 가중치를 업데이트하지 않으며, 손실과 성능 지표만 계산함.

모델의 성능은 검증 손실을 기준으로 평가됨. 검증 손실이 이전 에폭보다 개선되면, 해당 모델의 가중치가 저장됨.

```python
# 학습 및 검증
best_loss = float('inf')
best_model_wts = copy.deepcopy(model.state_dict())

num_epochs = 15
for epoch in range(num_epochs):
    model.train()
    train_loss = 0.0
    for images, masks in train_loader:
        images = images.to(device)
        masks = masks.to(device)

        optimizer.zero_grad()
        outputs = model(images)
        loss = criterion(outputs, masks)
        loss.backward()
        optimizer.step()
        train_loss += loss.item() * images.size(0)
    train_loss /= len(train_loader.dataset)

    # 검증 과정
    model.eval()
    val_loss = 0.0
    with torch.no_grad():
        for images, masks in val_loader:
            images = images.to(device)
            masks = masks.to(device)
            outputs = model(images)
            loss = criterion(outputs, masks)
            val_loss += loss.item() * images.size(0)
    val_loss /= len(val_loader.dataset)
```

```python
# 데이터셋 및 DataLoader 설정
resize_transform = transforms.Compose([
    transforms.Resize((960, 544)),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
])

train_dataset = RoadSegmentationDataset(
    image_dir='/home/ksy/WorkData/segmentation/trainImage',
    mask_dir='/home/ksy/WorkData/segmentation/trainLabels',
    transform=CustomTransform()  # CustomTransform 인스턴스로 수정
)

val_dataset = RoadSegmentationDataset(
    image_dir='/home/ksy/WorkData/segmentation/verification',
    mask_dir='/home/ksy/WorkData/segmentation/verificationLabel',
    transform=CustomTransform()  # CustomTransform 인스턴스로 수정
)

train_loader = DataLoader(train_dataset, batch_size=4, shuffle=True)
val_loader = DataLoader(val_dataset, batch_size=4, shuffle=False)
```

```python
# 모델 저장
if val_loss < best_loss:
    print(f'Validation loss decreased from {best_loss:.4f} to {val_loss:.4f}. Saving model...')
    best_loss = val_loss
    best_model_wts = copy.deepcopy(model.state_dict())
```

그림 14 - 모델 학습 코드

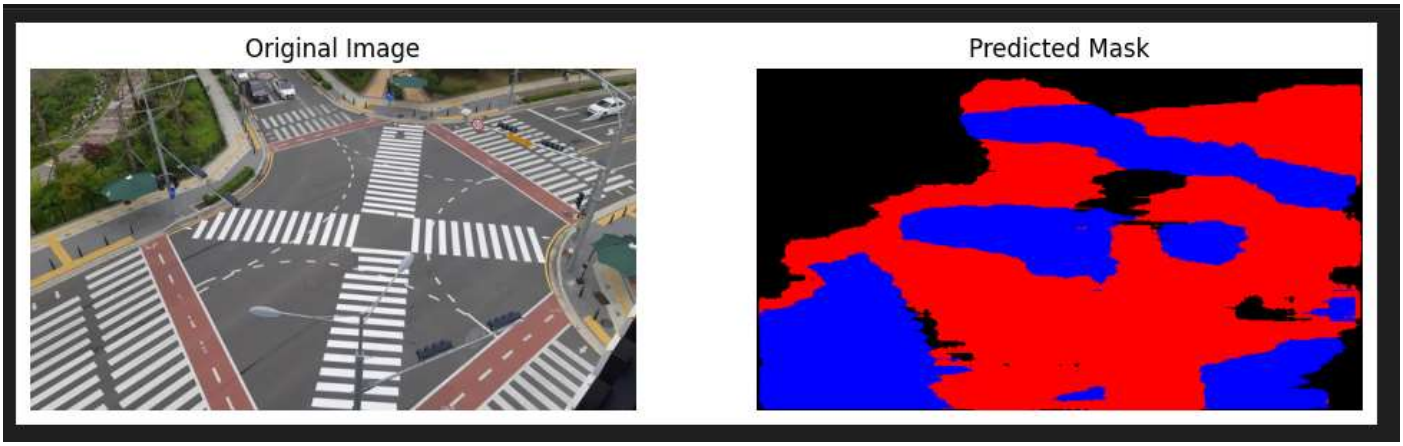


그림 15 - 영상 분할 결과물

배경은 검은색, 도로는 빨간색, 횡단보도는 파란색으로 표현함.

2. 3 분석 프로그램 제작 및 실행 방법

1) 요구사항

이미지 파일 하나에 분석결과 텍스트 파일 하나를 생성해야 함.
형식은 [라벨] [객체 중심좌표(X)] [객체 중심좌표(Y)] [객체 너비(W)] [객체 높이(H)]
띄어쓰기로 구분하며, 라벨 0 은 무단횡단 보행자, 라벨 1 은 횡단 보행자로 표시함.

2) 프로그램 순서도

1. 인자 파싱과 초기 설정

인자 파싱: argparse 라이브러리를 사용하여 명령줄 인터페이스를 통해 data_dir (이미지 폴더 경로) 및 result_dir (결과가 저장될 폴더 경로)를 입력받음.

```
# 인자 파싱을 위한 설정
parser = argparse.ArgumentParser(description='모델 실행을 위한 인자 파싱')
parser.add_argument('--data_dir', required=True, help='이미지 폴더 경로를 입력하세요.')
parser.add_argument('--result_dir', required=True, help='분석 결과가 저장될 폴더 경로를 입력하세요.')
args = parser.parse_args
```

그림 16 - 인자 파싱 코드

2. 모델 인스턴스 생성

MobileNetV2_UNet 모델을 초기화하고, 필요한 클래스 수에 맞게 조정함.

모델 설정: 모델을 CPU 나 GPU 에 로드하고, 사전 학습된 가중치를 불러온 후, 모델을 평가 모드로 설정함.

```python
# 모델 인스턴스 생성
model = MobileNetV2_UNet(num_classes=4)  # 클래스 수에 맞게 조정해야 할 수 있습니다.

# 모델을 CPU나 GPU로 이동
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model.to(device)

# 모델 가중치 로드 (학습된 모델 가중치 경로 지정 필요)
model.load_state_dict(torch.load(os.path.join(args.data_dir, '/home/pjs/workspace/best_model.pth'), map_location=device))

# 모델을 평가 모드로 설정
model.eval()
```

그림 17 - 모델 인스턴스 생성 코드

3. 이미지 전처리 및 예측 마스크 생성

이미지를 로드하고, 사이즈 조정, 텐서 변환, 정규화 등의 전처리 과정을 수행함.

전처리된 이미지를 모델에 입력하여 예측을 실행함. 출력된 예측값으로부터 마스크 이미지를 생성함.

예측된 마스크를 색상 이미지로 변환하여, 다른 클래스에 대해 서로 다른 색상을 할당함.

변환된 색상 마스크 이미지를 지정된 디렉토리에 저장하고, 파일 경로를 반환함.

```python
def visualize_prediction_with_image(model, image_path, device, result_dir, predicted_mask_dir):
    # 이미지 전처리 및 모델을 통한 예측
    image = Image.open(image_path).convert("RGB")
    preprocess = transforms.Compose([
        transforms.Resize((960, 544)),
        transforms.ToTensor(),
        transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
    ])
    image_tensor = preprocess(image).unsqueeze(0).to(device)
    $
    with torch.no_grad():
        output = model(image_tensor)
        output = F.interpolate(output, size=(640, 640), mode='bilinear', align_corners=False)
        predicted_mask = torch.argmax(output, dim=1).cpu()

    color_mask = mask_to_color_image(predicted_mask.numpy())

    # 640 x 640 크기로 이미지 조정
    color_mask_resized = Image.fromarray(color_mask).resize((640, 640), Image.NEAREST)

    # 저장할 파일명과 경로 설정
    image_name = os.path.basename(image_path).split('.')[0] + '_predicted_mask.png'
    save_path = os.path.join(predicted_mask_dir, image_name)

    # 예측된 색상 마스크 시각화 및 저장
    color_mask_resized.save(save_path)
    print(f"Saved predicted mask to {save_path}")

    # 저장된 파일 경로 반환
    return save_path
```

그림 18 - 이미지 전처리 및 마스크 생성 코드

4. 객체 감지 및 행동 판단

객체 감지 모델 실행: YOLO 객체 감지 모델을 사용하여 이미지 내의 객체를 감지함. 각 객체에 대한 바운딩 박스, 클래스 ID, 신뢰도 등의 정보를 추출함.

감지된 객체 정보 저장: 감지된 객체들의 정보를 텍스트 파일로 저장함.

감지된 객체에 대한 행동 판단:

감지된 각 객체의 바닥 지점 좌표를 계산함.

마스크 이미지에서 해당 좌표의 픽셀 색상을 확인함.

특정 색상(예: 빨간색)에 해당하는 경우, 특정 행동(예: 무단횡단)으로 판단함.

```
# YOLO 모델 로드 및 객체 탐지 실행
model_yolo = YOLO(os.path.join(args.data_dir, '/home/pjs/workspace/SEJONG/realrun/detect/train38/weights/best.pt'))
results = model_yolo(image_path)
```

그림 19 - YOLO 객체 탐지 실행 코드

```
xywh_tensor = results[0].boxes.xywh
red_value = [255, 0, 0]  # 실제 마스크의 파란색 값을 기준으로 설정하세요

# 중심 좌표를 추출함.
x_center = xywh_tensor[0][0].item()
y_center = xywh_tensor[0][1].item()
width = xywh_tensor[0][2].item()
height = xywh_tensor[0][3].item()

# 중심에서 바닥으로 절반 내린 지점의 y 좌표를 계산함.
y_lower_half = y_center + (height / 2)

# 결과 좌표를 반환함.
result_coordinates = (x_center, y_lower_half)
```

그림 20 - 객체 좌표 값 계산 코드

그림 21 - 객체 탐지 예시

객체의 중심에서 바닥으로 절반 내린 지점의 y 좌표를 계산 후 해당 좌표를 사용하여 후에 무단횡단 유무를 구별함.

```
# 사람의 좌표를 기반으로 바닥 지점 계산
x_center = xywh_tensor[0][0].item()
y_center = xywh_tensor[0][1].item()
width = xywh_tensor[0][2].item()
height = xywh_tensor[0][3].item()
y_lower_half = y_center + (height / 2)

# 좌표가 정수가 되도록 반올림
x_center_int = int(round(x_center))
y_lower_half_int = int(round(y_lower_half))
# 해당 좌표의 픽셀 값 확인
pixel_value = mask_array[y_lower_half_int, x_center_int]
# 해당 좌표의 픽셀이 파란색인지 확인
is_in_blue_area = np.all(pixel_value == red_value)

# xywh_tensor 에서 모든 객체에 대해 반복하여 처리함.
for i in range(len(xywh_tensor)):
    # 중심 좌표와 바운딩 박스 크기 추출
    x_center = xywh_tensor[i][0].item()
    y_center = xywh_tensor[i][1].item()
    width = xywh_tensor[i][2].item()
    height = xywh_tensor[i][3].item()

    # 중심에서 바닥으로 절반 내린 지점의 y 좌표를 계산함.
    y_lower_half = y_center + (height / 2)

    # 좌표가 정수가 되도록 반올림
    x_center_int = int(round(x_center))
    y_lower_half_int = int(round(y_lower_half))

    # 좌표가 이미지 범위 내에 있는지 확인함.
    if 0 <= x_center_int < mask_array.shape[1] and 0 <= y_lower_half_int <
mask_array.shape[0]:
        # 해당 좌표의 픽셀 값 확인
        pixel_value = mask_array[y_lower_half_int, x_center_int]

        # 해당 좌표의 픽셀이 빨간색(도로)인지 확인
        is_in_red_area = np.all(pixel_value == red_value)

        print(f"The person at ({x_center}, {y_lower_half})")

        if is_in_red_area:
            print("The person is jaywalking")
        else:
            print("The person is a normal pedestrian.")
    else:
        print(f"The person at ({x_center}, {y_lower_half}) is outside the image
boundary.")
```

그림 22 - 객체 탐지 좌표 계산 코드

5. ‘Predicte_mask_png’저장 폴더 및 디렉토리 구성

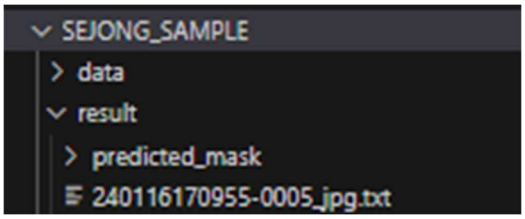
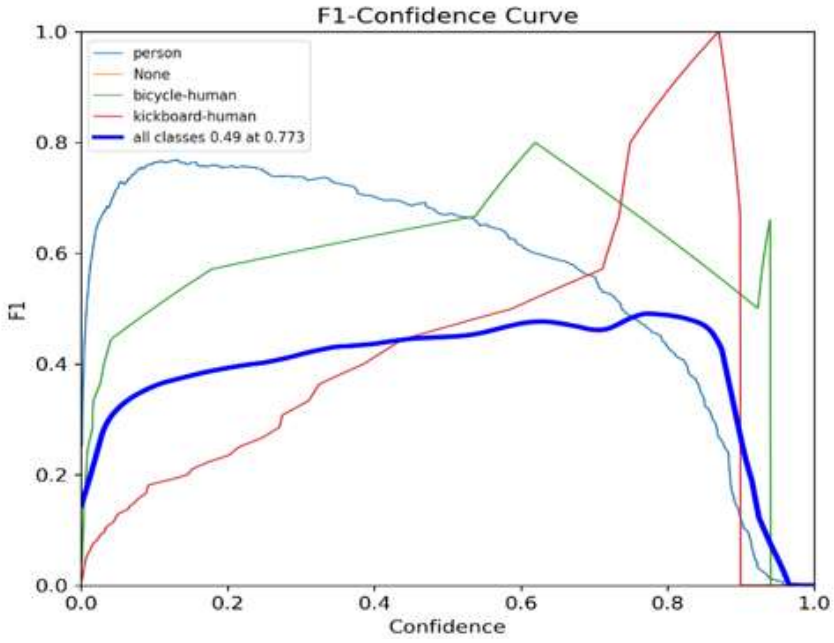


그림 23 - 저장 폴더 예시

Result 폴더안 predicted_mask (unet 모델로 라벨링된 Mask 이미지) 폴더 생성 및 파일 저장

3. 결 론

3. 1 모델 평가(Customizing yolov8x)



Result 1 - Customizing yolov8x F1 신뢰도 그래프

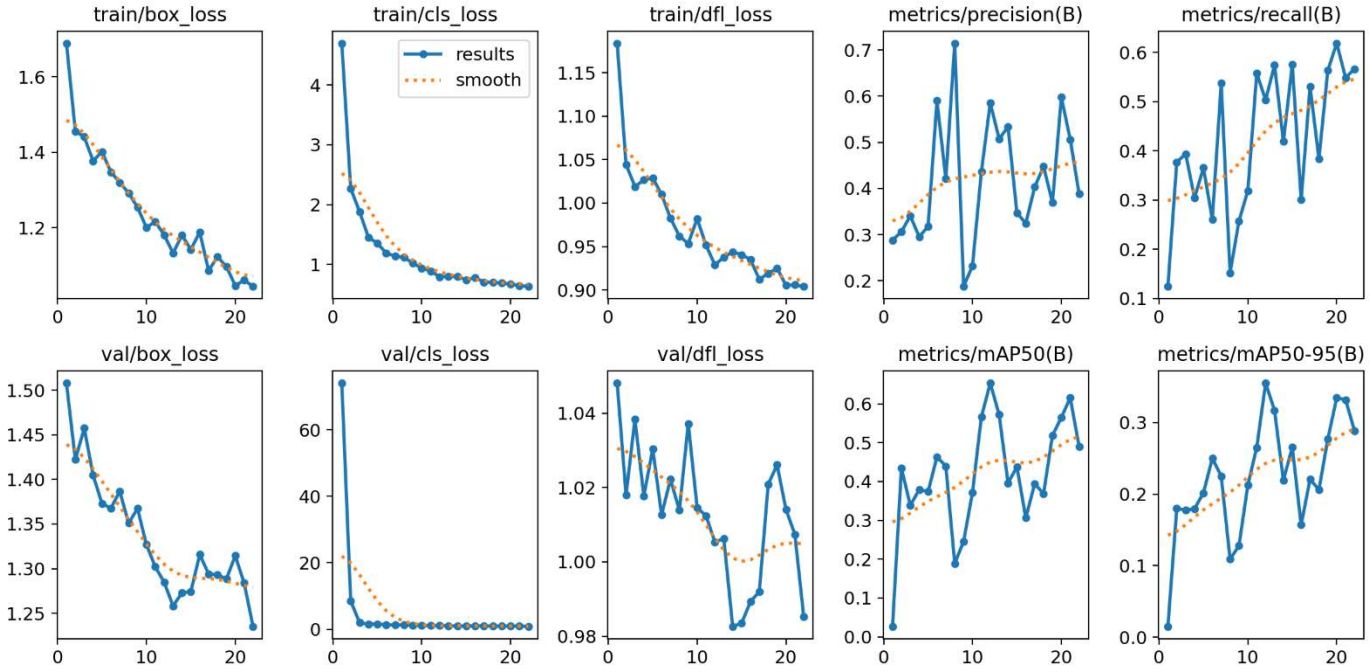
F1-신뢰도 성능 평가

본 연구에서 사용된 객체 감지 모델의 성능은 F1-신뢰도 곡선을 통해 평가하였음. 이 그래프는 다양한 신뢰도 임계값에서 모델의 F1 점수를 나타냄. F1 점수는 모델의 정밀도와 재현율을 조화롭게 고려한 지표로, 감지된 객체의 정확도와 관련성을 종합적으로 평가함.

각 클래스별로 관찰된 성능은 다음과 같다: 'bicycle-human' 클래스는 높은 F1 점수를 달성하여 모델이 이 클래스를 감지하는데 높은 정확성을 보였음을 나타냄. 반면, 'person'과 'kickboard-human 클래스는 상대적으로 낮은 F1 점수를 보여줌으로써 모델이 해당 객체들을 감지하는 데 있어 일부 한계를 가지고 있음을 시사함. 특히 'kickboard-human' 클래스의 경우, 신뢰도 임계값이

높아질수록 F1 점수가 급격히 감소하는 현상이 관찰되었음 이는 모델이 'kickboard-human' 객체들을 높은 신뢰도로 예측하는 것이 어렵다는 것을 의미할 수 있음.

전체 클래스에 대한 성능을 종합적으로 나타내는 'all classes' 곡선은 최적의 신뢰도 임계값인 0.773 에서 평균 F1 점수 0.49 를 달성하였음. 이는 모델이 모든 클래스를 종합적으로 고려했을 때, 이 임계값에서 가장 균형잡힌 성능을 보인다는 것을 나타냄. 하지만, 이 임계값에서조차 모델의 성능은 완벽하지 않으며, 특히 개별 클래스에서의 성능 차이가 뚜렷하게 나타남.



Result 2 - Customizing yolov8x 성능 그래프

- train/box_loss 와 val/box_loss: 이 그래프들은 학습 및 검증 데이터에 대한 바운딩 박스 손실(bounding box loss)을 나타냄. 이 손실은 모델이 예측한 바운딩 박스의 위치가 실제 객체의 위치와 얼마나 잘 일치하는지를 측정함. 값이 낮을수록 성능이 좋다는 것을 의미함.

- train/cls_loss 와 val/cls_loss: 이들은 학습 및 검증 데이터에 대한 클래스 손실(class loss)을 보여줍니다. 이는 모델이 각 객체의 클래스를 얼마나 정확하게 예측하는지를 평가함.

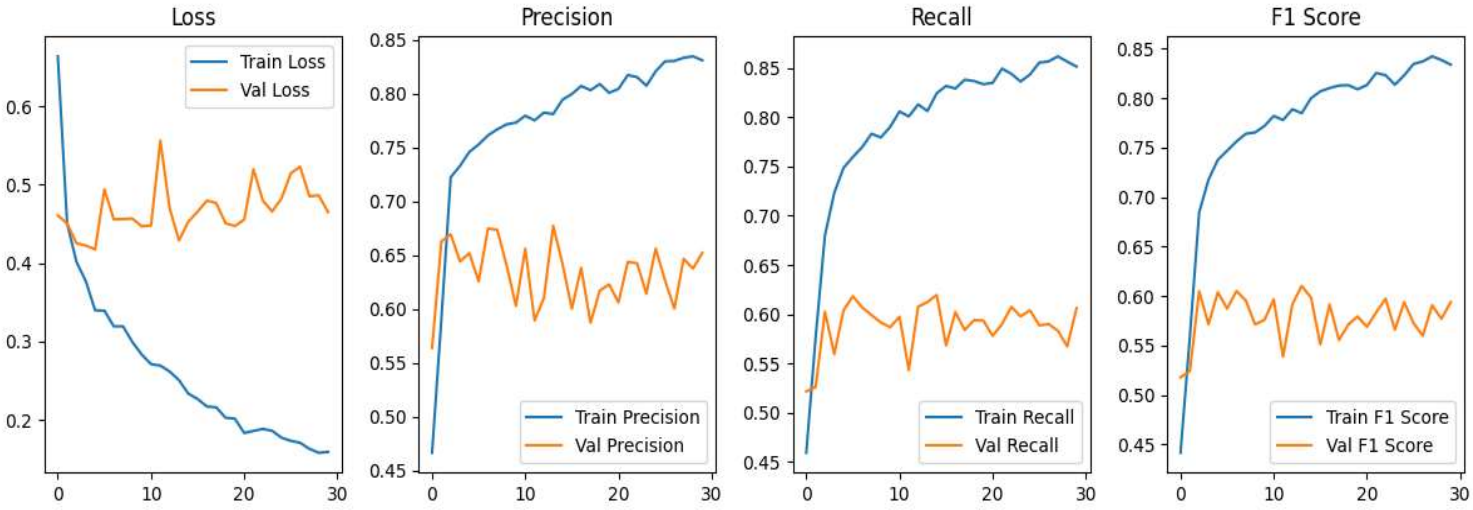
- train/obj_loss 와 val/obj_loss: 이 그래프들은 객체 손실(object loss)을 나타내며, 모델이 실제 객체를 얼마나 잘 감지하는지를 나타냄.

- metrics/precision(B)와 metrics/recall(B): 정밀도(precision)는 모델이 True Positive 로 분류한 예측의 비율이고, 재현율(recall)은 실제 Positive 중 모델이 True Positive 로 정확하게 분류한 비율임. 이 두 지표는 모델의 예측 성능을 나타냄.

- metrics/mAP50(B)와 metrics/mAP50-95(B): 평균 정밀도(mean Average Precision)는 모델이 다양한 임계값에서 객체를 얼마나 잘 감지하는지를 종합적으로 평가하는 지표임.

mAP50 은 IoU(Intersection over Union) 임계값이 0.5 일 때의 값이고, mAP50-95 는 IoU 가 0.5 에서 0.95 까지의 평균을 나타냄.

3. 2 모델 평가(MobileNetV2_Unet)



Result 3 - MobileNetV2_Unet 성능 그래프

1) 손실(Loss) 그래프:

- 훈련 손실(Train Loss)은 에폭이 진행됨에 따라 지속적으로 감소하고 있어 모델이 학습 데이터에 잘 적응하고 있음을 나타냄.
- 검증 손실(Val Loss)은 불규칙한 변동을 보이고 있지만 전반적으로 감소 추세는 아닌 것으로 보임. 이는 모델이 검증 데이터에 대해서는 일반화를 잘 하지 못하고 있음을 시사함.

2) 정밀도(Precision) 그래프:

- 훈련 정밀도(Train Precision)는 상당히 높은 수준으로, 모델이 학습 데이터에서 정확한 예측을 많이 하고 있음을 나타냄.
- 검증 정밀도(Val Precision)는 매우 변동이 심하며, 훈련 정밀도에 비해 낮음. 이는 모델이 학습 데이터에 과적합(overfitting)되었을 가능성을 나타냄.

3) 재현율(Recall) 그래프:

- 훈련 재현율(Train Recall)은 시간이 지남에 따라 증가하고 있으며, 모델이 학습 데이터의 관련 샘플들을 잘 찾아내고 있음을 나타냄.
- 검증 재현율(Val Recall)은 변동성이 크며 훈련 재현율보다 낮음. 이는 모델이 일반화하는 데 어려움을 겪고 있음을 의미함.

4) F1 점수(F1 Score) 그래프:

- 훈련 F1 점수(Train F1 Score)는 매우 높으며, 모델이 학습 데이터에서 정밀도와 재현율 사이의 균형이 잘 잡혀있음을 의미함.
- 검증 F1 점수(Val F1 Score)는 다른 검증 지표들처럼 변동성이 크며, 훈련 F1 점수에 비해 현저히 낮음. 이는 모델이 학습 데이터에 비해 검증 데이터에서 낮은 성능을 보이고 있음을 나타냄.

3. 3 프로젝트 결과

본 프로젝트는 U-Net 과 YOLO 모델을 활용하여 도로 환경에서의 객체 탐지와 영상 분할을 성공적으로 수행하였음. U-Net 구조는 MobileNetV2 를 기반으로 한 인코더를 포함하고 있으며, 사전 학습된 모델의 가중치는 best_model.pth 에서 가져와 적용하였음. 반면, YOLO 모델은 도로 상의 행인, 차량 등의 객체를 식별하는 데 사용되었고, 그 가중치는 last.pt 에서 로드하였음.

프로젝트의 실행 과정에서는 아나콘다를 통해 관리되는 파이썬 가상 환경을 구축하였고, 필수적인 패키지들을 설치하여 딥러닝 모델의 학습과 검증을 위한 환경을 마련하였음. 이 과정에서 파이토치와 오픈 CV, 그리고 Ultralytics YOLO 라이브러리가 주요 도구로 사용되었음

프로젝트에 사용된 파일들은 모두 Git 으로 저장되어 있음.

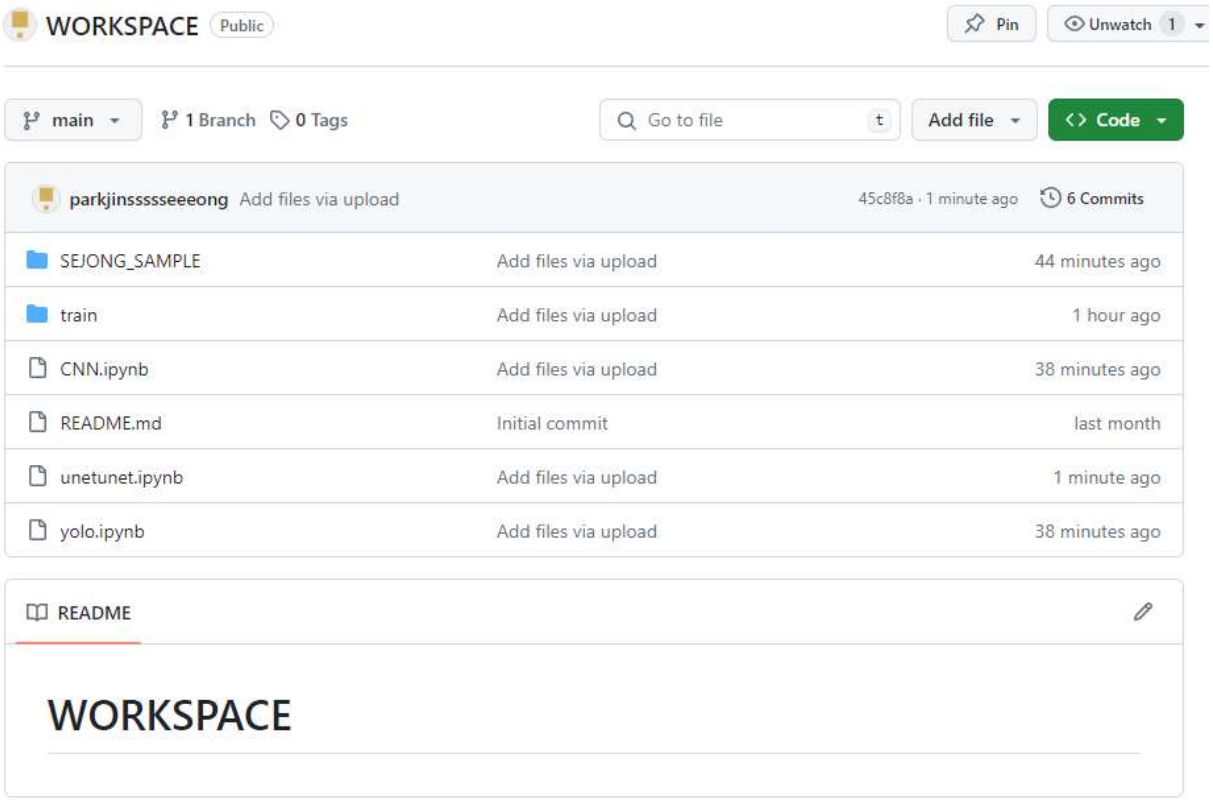


그림 24 - GitHub 저장소

출처:https://github.com/parkjinsssssseeeong/WORKSPACE

데이터 전처리를 위한 CustomTransform 클래스를 정의하여 입력 이미지와 레이블 마스크를 적절한 크기와 텐서 형태로 변환하였으며, 평균과 표준 편차를 사용하여 정규화를 수행하였음. RoadSegmentationDataset 클래스는 파이토치의 DataLoader 를 통해 학습 및 검증 데이터로 쉽게 통합될 수 있도록 설계되었음

모델의 추론 단계에서는 가중치가 로드된 후 평가 모드로 전환하여, 신규 데이터에 대한 예측을 수행하였음. 예측된 마스크는 색상 이미지로 변환하여 시각적으로 분석이 가능했으며, 이를 통해 모델이 객체의 위치를 정확히 파악하고 올바르게 분류할 수 있음을 확인하였음.

마지막으로, 결과 디렉토리에는 예측된 마스크와 객체 탐지 정보가 저장되었고, 이는 프로젝트의 성과를 나타내는 중요한 산출물로 활용되었음 프로젝트의 결과물은 도로 안전을 강화하고 자율 주행 차량의 인지 시스템을 향상하는 데 기여할 잠재력을 지니고 있으며, 향후 연구와 개발에 있어 중요한 기초 자료로 활용될 수 있음. 이를 통해 보행자의 안전을 보장하고 또한, 수집된 데이터와 분석 결과는 도로 교통 환경에 대한 이해를 깊이 있게 하고, 자율 주행 알고리즘의 정확도를 높이는 데 중요한 역할을 할 것으로 기대됨.