제목: 보행자 안전을 위한 실시간 객체 탐지 및 분할 기술 개발

1. 서론

- 보행자 안전의 중요성
- 연구 배경 및 필요성
- 보행자 안전과 관련된 이론적 개념 및 선행 연구 고찰
- 딥러닝 기반 객체 탐지 및 분할 기술의 원리 및 장단점

2. 연구 방법

- 데이터 수집 및 전처리
- 딥러닝 모델 학습 및 평가
- 실험 환경 구성 및 실험 절차

3. 연구 결과

- 딥러닝 모델의 성능 평가 결과 제시
- 실제 도로 환경에서의 적용 가능성 검토

4. 결론

- 연구 결과의 요약 및 시사점 도출
- 향후 연구 방향 및 제언

<서론>

최근 보행자 교통사고의 증가로 인해 보행자 안전에 대한 관심이 높아지고 있으며, 자율 주행 기술의 발전과 스마트 시티 구현과 함께 보행자 안전 문제는 더욱 중요한 이슈로 부각되고 있습니다. 이러한 문제를 해결하기 위한 다양한 기술들이 개발되고 있지만, 보행자 안전 문제는 여전히 완전히 해소되지 않고 있습니다. 이에 따라 보행자 안전을 위한 실시간 객체 검출 및 분할 기술 개발에 대한 연구를 수행하고자 합니다.

실시간 보행자 안전 모니터링은 교통 관리 및 자율 주행 시스템에서 중요한 요소입니다. 특히 금천구 독산동과 시흥동 같은 인구 밀집 주거 지역에서 발생하는 보행자와 차량 간의 충돌은 사회적 문제로 대두되고 있으며, 이 지역에서의 보행자 사고를 예방하기 위한 실질적인 대책이 요구됩니다. 실제 도로에서 보행자를 정확하게 인식하고 교통 신호 및 인프라와 상호 작용할 수 있는 자율 주행 기술은 아직 개발 단계에 있으며, 좁은 골목이나 복잡한 도로 환경에서는 보행자와 차량 간의 사고 위험이 크게 존재합니다.

이러한 위험 지역에서는 실시간 객체 검출과 이미지 분할을 통해 보행자의 위치와 차량의 움직임을 정확하게 분석하고, 이를 자율 주행 시스템과 교통 관리 시스템에 반영하는 것이 필수적입니다. 따라서 실시간 객체 검출과 이미지 분할 기술을 기반으로 한 데이터 중심의 보행자 안전 개선 솔루션을 제안하고자 합니다.

자율 주행 기술의 급속한 발전과 스마트 시티 구현은 교통 환경에 새로운 기회를 제공하는 동시에 다양한 안전 관련 이슈를 부각시키고 있습니다. 이러한 이슈에는 보행자 안전뿐만 아니라 차량 간 충돌, 도로 인프라의 안정성, 교통 신호 시스템의 신뢰성, 그리고 도로 이용자의 다양성(자전거, 오토바이, 전동 킥보드 등)으로 인한 안전 문제 등이 포함됩니다. 특히 금천구와 같은 인구 밀집 지역에서는 교통량 증가와 복잡한 도로 구조로 인해 다양한 유형의 교통사고 위험이 증가하고 있습니다. 좁은 도로, 불법 주차, 신호 위반, 도로 공사로 인한 우회로 등은 운전자와 보행자 모두에게 위험을 초래합니다.

이러한 복잡한 문제를 해결하기 위해 본 프로젝트는 인공지능 모델을 활용한 종합적인 접근을 통해 보행자 안전을 획기적으로 향상시키고자 합니다. 먼저 연구 배경과 필요성을 다음과 같이 살펴보겠습니다.

현재 대부분의 교통사고는 운전자의 부주의나 보행자의 무단횡단으로 인해 발생하며, 이러한 사고를 방지하기 위해 정확한 객체 검출 및 분할 기술이 필요합니다. 또한 최근 자율 주행 차량의 보급으로 인해 더욱 정확한 객체 검출 및 분할 기술이 요구되고 있습니다.

연구의 목적은 다음과 같습니다.

본 연구에서는 보행자 안전과 관련된 다양한 요소를 분석하고, 딥러닝 기반의 객체 검출 및 분할 기술을 활용하여 보행자를 정확하게 감지하고 분할하는 기술을 개발하는 것을 목표로 합니다. 이는 보행자 안전을 향상시키고 자율 주행 차량의 상용화를 촉진할 것으로 기대됩니다.

본 연구에서는 먼저 기존의 객체 검출 및 분할 기술을 분석하고, 이를 기반으로 딥러닝 기반의 새로운 객체 검출 및 분할 기술을 개발합니다. 그런 다음 실제 도로 환경에서 실험을 진행하여 성능을 검증하고, 이를 토대로 개선 방안을 모색합니다. 마지막으로 개발된 기술을 실제로 적용함으로써 보행자 안전을 향상시키는 방안을 제시합니다.

결론적으로, 본 연구는 보행자 안전을 위한 실시간 객체 검출 및 분할 기술의 개발을 통해 보행자 안전 향상에 기여할 것이며, 이는 자율 주행 차량의 상용화에도 큰 도움이 될 것으로 기대됩니다.

<배경>

1. 보행자 안전과 관련된 이론적 개념 및 선행 연구 고찰

보행자 안전은 교통사고 중에서도 가장 많은 사망자를 발생시키는 분야 중 하나입니다. 특히, 도심에서는 차량과 보행자가 공존하기 때문에 더욱 위험한 상황이 발생할 수 있습니다.

본 연구에서는 보행자 안전을 위해 다양한 기술이 개발되었습니다. 예를 들어, CCTV나 센서를 이용하여 보행자의 위치를 파악하고, 이를 바탕으로 차량 운전자에게 경고를 보내는 기술이 대표적입니다. 하지만 CCTV나 센서는 날씨나 환경에 따라 성능이 저하될 수 있으며, 정확한 객체 인식이 어렵습니다.

2. 딥러닝 기반 객체 탐지 및 분할 기술의 원리 및 장단점

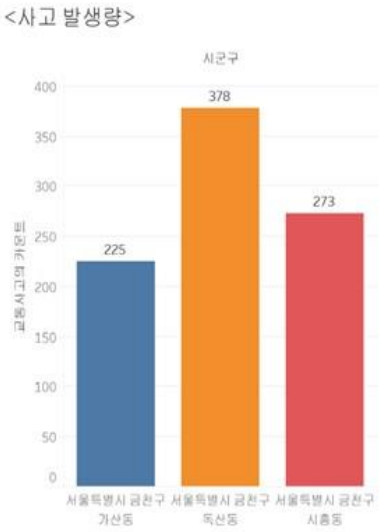
딥러닝 기반 객체 탐지 및 분할 기술은 인공지능 기술 중 하나로, 이미지나 영상에서 객체를 탐지하고 분할하는 데 사용됩니다.

이 방법은 딥러닝 모델을 이용하여 이미지나 영상을 학습하고, 이를 바탕으로 객체를 탐지하고 분할합니다. 높은 정확도와 빠른 처리 속도를 가지고 있으며. 또한, 다양한 환경에서도 안정적인 성능을 발휘하며, 기존의 기술보다 더 정확한 객체 인식이 가능합니다.

1.프로젝트 개요

보행자 안전을 증진하기 위한 본 프로젝트는 자율 주행 및 도시 계획에 활용될 수 있는 데이터를 구축하는 것을 목표로 합니다. 특히, 좁은 골목길과 보행자 사고가 빈번하게 발생하는 환경에서 보행자와 차량 간의 충돌을 방지하기 위한 인공지능 모델을 개발하는 데 중점을 두고 있습니다. 본 프로젝트에서는 금천구의 거리 데이터를 활용하여 보행자 안전과 관련된 다양한 요소를 분석하고, 물체 탐지(Object Detection) 및 분할(Segmentation) 기법을 적용하여 안전 요소들을 모델링합니다.

2. 금천구 데이터 선정 과정



금천구는 본 프로젝트의 대상으로 선정되었습니다. 금천구의 시흥동, 독산동은 좁은 골목과 주거 지역이 혼합된 환경으로, 보행자와 차량 간의 충돌 사고 가능성이 높은 구역입니다. 이러한 특성 때문에 보행자 안전을 위한 데이터 분석이 중요한 구역으로 간주되었습니다.

프로젝트는 금천구 시흥동, 독산동 내의 특정 구역을 대상으로 하여, 해당 지역의 도로, 보행로, 교통 신호 등의 인프라를 분석합니다. Google Maps API를 사용하여 금천구의 거리뷰 이미지를 수집하고, 해당 이미지에서 보행자와 차량의 위험 요소를 탐지하는 방식으로 데이터 수집 및 처리를 진행하였습니다

Google Maps API를 이용하여 금천구 독산동과 시흥동 전역에서 30m 간격으로 거리뷰 이미지를 수집하고, 이들 이미지를 이용하여 딥러닝 모델을 학습시키고 평가한다.

```
# Google Street View API 키
API_KEY = 'your api'

# Google Street View API의 기본 URL
STREET_VIEW_URL = 'https://maps.googleapis.com/maps/api/streetview'
```

api key 와 url 입력.

```
# CSV 파일 로드
data = pd.read_csv(r'C:\Users\user\Desktop\kn\독산,시흥.csv', encoding='cp949')
print(data.head())  # 데이터가 올바르게 로드되었는지 확인
```

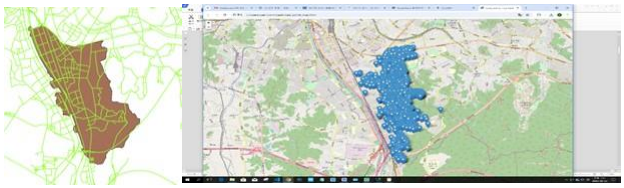
데이터 csv 파일 로드.

```python
# 30미터마다 중간 지점을 생성하는 함수
def generate_points_every_30m(start_lat, start_lon, end_lat, end_lon, interval=30):
    points = []
    start = (start_lat, start_lon)
    end = (end_lat, end_lon)
    total_distance = geodesic(start, end).meters

    # 총 거리를 기반으로 간격 수 계산
    num_intervals = int(total_distance // interval)

    for i in range(1, num_intervals + 1):
        fraction = i / num_intervals
        lat = start_lat + fraction * (end_lat - start_lat)
        lon = start_lon + fraction * (end_lon - start_lon)
        points.append((lat, lon))

    return points
```

함수 generate_points_every_30m 정의.

from geopy.distance import geodesic 라이브러리를 사용(두 지점사이 거리를 구할 때 곡률 같은 것 고려하여 계산을 해준다). generate_points_every_30m 함수를 사용하여 시작지점과 종료지점 사이 총 거리를 계산. 해당 거리를 30m 간격마다 나누어 포인트를 저장.

->Google maps api로 해당 포인트들의 거리뷰 이미지를 불러와 저장함

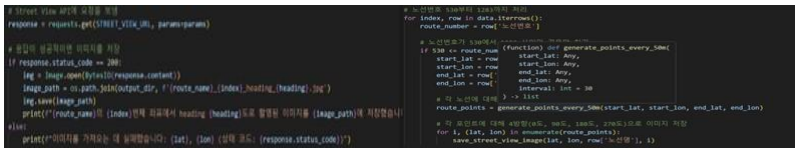


QGIS 사용: 금천구의 경계를 나타내는GeoJSON 데이터를 활용하여 QGIS에서 구역 분할.

30미터 간격 노드 생성: 거리 간격을 설정하여 금천구 독산동, 시흥동 전역에서 이미지를 수집할 지점을 설정.

Google Maps API 활용: 설정된 좌표에 대해 구글 지도 거리뷰API를 통해 거리 이미지 데이터를 자동으로 수집.

이러한 과정을 통해 약5000장의 이미지 데이터를 수집하였고, 이후 이를 기반으로 분석 작업을 진행했습니다.



분석에 불필요한 이미지는 제거하고, 모델 학습에 이용하기 위해 라벨링 작업을 진행

클래스: 인도, 도로, 보행자, 차량, 신호등, 안전 표지판, 장애물 등 총 25개 클래스.

라벨링 도구: Label Studio 및 roboflow 활용.

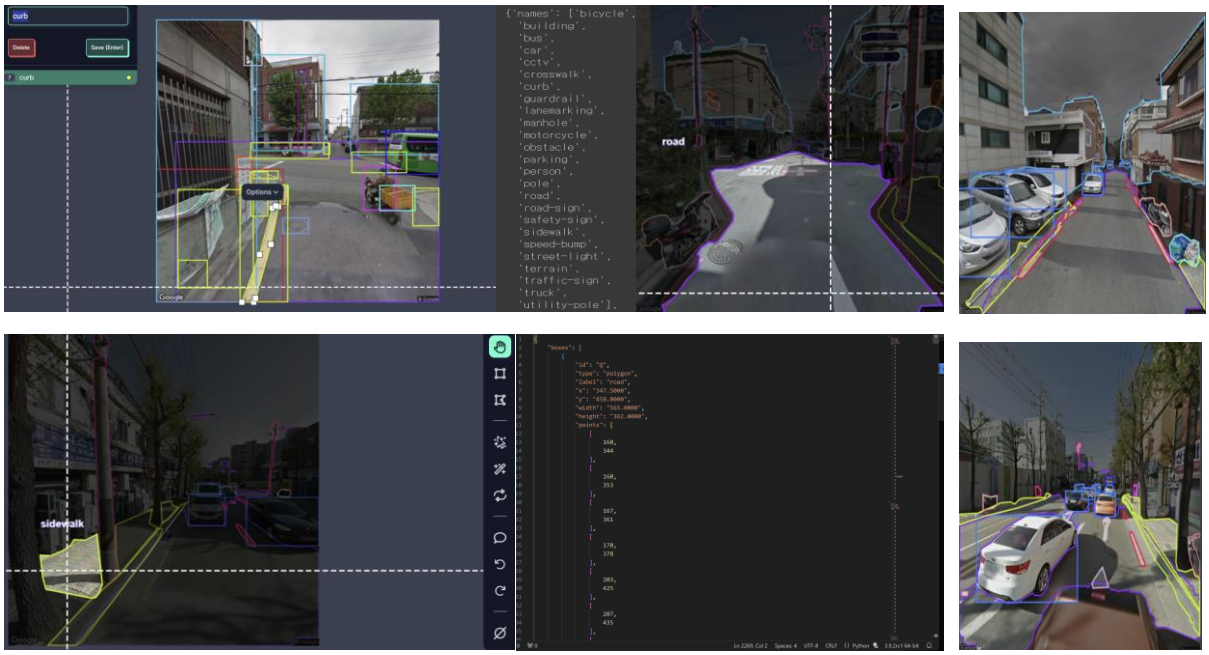
이미지 라벨링 이란?

이미지 라벨링은 데이터 라벨링 작업자가 태그나 메타데이터를 사용하여 AI 모델이 학습하는 데이터 특징을 표시하는 과정을 의미

수집한 거리뷰 이미지 데이터셋을 통한 딥러닝 모델 학습을 위해 라벨링을 진행

클래스 : Sidewalk, road, person, Traffic sign, obstacle 등 총 25개 클래스

분석에 불필요한 이미지는 제거하고, 모델 학습에 이용하기 위해 라벨링 작업을 진행.



연구 방법 :

1. Google Maps API 기반 데이터 수집 : 먼저, Google Maps API를 이용하여 금천구 전역에서 30m 간격으로 거리뷰 이미지를 수집한다. 이렇게 수집한 이미지는 보행자와 차량 등 다양한 객체들이 포함되어 있으며, 이들 객체를 탐지하고 분할하기 위해서는 딥러닝 기술이 필요하다.

2. 딥러닝 모델 학습 및 평가 : 다음으로, 수집한 이미지를 이용하여 딥러닝 모델을 학습시킨다. 이때, 객체 탐지 및 분할에 적합한 딥러닝 모델로는 YOLO(You Only Look Once), Faster R-CNN(Region Convolutional Neural Network) 등이 있다. 이러한 모델들은 이미지 내에서 객체를 탐지하고 그 위치와 크기를 정확하게 예측할 수 있다. 또한, 학습된 모델을 평가하기 위해 다양한 지표를 사용한다. 예를 들어, mAP(mean Average Precision), IoU(Intersection over Union) 등이 있다.

3. 실험 환경 구성 및 실험 절차

코랩 , label studio 등

구글 코랩을 통한 거리뷰 이미지 오브젝트 분류

Colaboratory란? 브라우저에서 Python 코드를 실행할 수 있는 클라우드 기반 플랫폼입니다. 클라우드 기반의 시스템을 이용한Jupyter Notebook로 줄여서 Colab이라고도 하는 Colaboratory를 사용하면 브라우저에서 Python을 작성하고 실행할 수 있음. 무료로 GPU에 대한 액세스를 제공하므로 AI 모델 개발 및 이미지 처리에 이상적입니다. 사용자 친화적이며 설치나 구성이 필요 없으며 간편한 공유 및 협업 기능을 갖추고 있습니다.

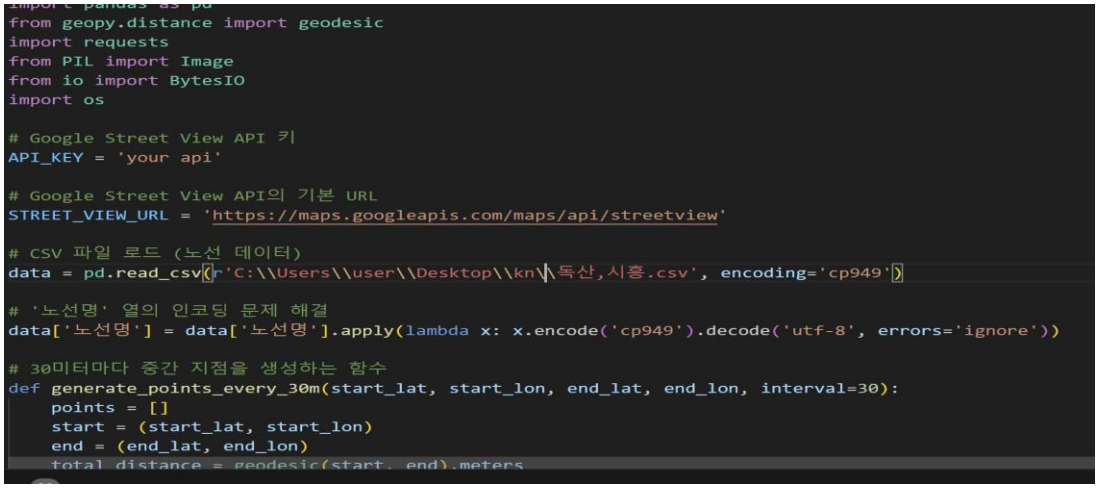
설정: Google 계정을 만들고 Colab에서 Google Colab에 액세스

Google Chrome 등 지원되는 웹 브라우저를 사용하고 있는지 확인.

Google Drive에서 '더보기' -> 'Google Colaboratory'를 선택하여 새 노트를 생성.

'Google 드라이브에 연결'을 클릭하여 데이터 저장을 위해 Google 드라이브를 Colab 환경에 연결. 인증이 필요한 경우, 필요한 코드를 입력하여 연결을 완료.

Google Maps API: 스트리트 뷰 이미지에 액세스하고 지리 데이터(위도 및 경도)를 검색하고 보행자 안전 연구를 위한 관심 영역을 탐색하는 데 사용됩니다.

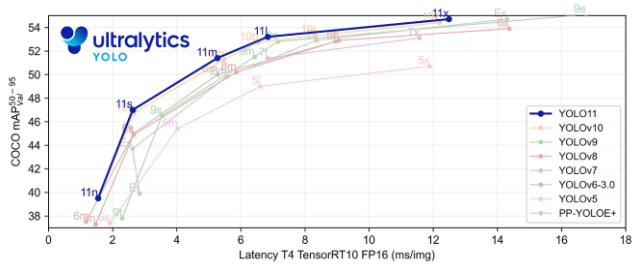


```
import pandas as pd
from geopy.distance import geodesic
import requests
from PIL import Image
from io import BytesIO
import os

# Google Street View API 키
API_KEY = 'your api'

# Google Street View API의 기본 URL
STREET_VIEW_URL = 'https://maps.googleapis.com/maps/api/streetview'

# CSV 파일 로드 (노선 데이터)
data = pd.read_csv(r'C:\\Users\\user\\Desktop\\kn\\독산,시흥.csv', encoding='cp949')

# '노선명' 열의 인코딩 문제 해결
data['노선명'] = data['노선명'].apply(lambda x: x.encode('cp949').decode('utf-8', errors='ignore'))

# 30미터마다 중간 지점을 생성하는 함수
def generate_points_every_30m(start_lat, start_lon, end_lat, end_lon, interval=30):
    points = []
    start = (start_lat, start_lon)
    end = (end_lat, end_lon)
    total_distance = geodesic(start, end).meters
```

YOLOv11:

YOLO(You Only Look Once)는 워싱턴 대학교의 Joseph Redmon과 Ali Farhadi가 개발한 가장 인기 있는 객체 탐지 및 이미지 분할 모델로 2015년에 처음 출시되어 빠른 속도와 정확성이 특징인 AI 모델- YOLOv4까지는 C언어 기반의 Darknet 구조가 사용되었지만, v5 부터는 PyTorch로 개발이 이루어져 서 24년 현재 YOLOv11 (SOTA) 아키텍처로 발전하여 빠른 속도와 사용자 친화적인 파이썬 인터페이스 를 제공하는 형태로 발전. 실시간 객체 감지에 사용되는 고급 AI 모델로, 특히 스트리트 뷰 이미지에서 보행자, 차량, 신호등 및 기타 관련 기능을 감지하는 데 유용합니다. YOLO11은 아키텍처와 학습 방법에서 중요한 개선을 도입하여, 다양한 컴퓨터 비전 작업에 적합한 다재다능한 선택지를 제공한다.

YOLOv11은 Object Detection과 함께 이미지나 동영상의 Image Segmention 또한 동일한 API를 이용하여 구현이 가능함



현재 금천구 이미지 데이터 셋이 크지는 않은데, yolov11을 사용하면 정확도가 2-3% 이상 향상된다. 정확도, 파라미터, flop 으로부터 이점이 있고 실행시간이 많이 단축됩니다.

Why using YOLOv11 is helpful?

연산 부하 감소:

더 적은 매개변수: 모든 모델 크기에 걸쳐 YOLOv11은 매개변수 수를 크게 줄입니다. 예를 들어 대형(L) 모델은 YOLOv8-L에 비해 42.09% 감소합니다.

낮은 FLOP: 초당 부동 소수점 연산(FLOP)이 줄어들어 계산 속도가 빨라집니다. YOLOv11-L 모델은 YOLOv8-L에 비해 FLOP를 47.40% 줄입니다.

복잡성을 줄이면서 정확도 향상:

매개변수 및 FLOP의 감소에도 불구하고 YOLOv11은 더 좋거나 비슷한 mAP 점수를 달성합니다.

이는 복잡한 계산 없이도 물체를 보다 정확하게 감지할 수 있음을 의미합니다.

최적화된 아키텍처

C3K2 및 C2PSA와 같은 효율적인 블록을 도입하면 모델을 경량화하면서 성능이 향상됩니다.

YOLOv8 vs YOLOv11

Model Size	YOLOv8 (mAP50-95)	YOLO11 (mAP50-95)	mAP50-95 Improvement (YOLO11 - YOLOv8)
N	0.371	0.392	0.021
S	0.447	0.467	0.020
M	0.501	0.514	0.013
L	0.529	0.532	0.003
X	0.540	0.547	0.007

Evaluation Results

YOLOv8 vs YOLO11

Model Size	YOLOv8 Parameters (M)	YOLO11 Parameters (M)	Reduction Rate (%)
n	3.2	2.6	18.75%
s	11.2	9.4	16.07%
m	25.9	20.1	22.39%
l	43.7	25.3	42.09%
x	68.2	56.9	16.55%

Parameters

Model Size	YOLOv8 FLOPs (B)	YOLO11 FLOPs (B)	Reduction Rate (%)
n	8.7	6.5	25.29%
s	28.6	21.5	24.83%
m	78.9	68.0	13.81%
l	165.2	86.9	47.40%
x	257.8	194.9	24.40%

YOLOv8 vs YOLO11 FLOPs

데이터 전처리:

- **이미지 입력**: 이미지를 받아 표준 입력 크기로 리사이즈하고 정규화를 적용합니다.
- **증강 (Optional)**: 모델의 일반화 성능을 향상시키기 위해 이미지에 뒤집기, 회전, 스케일링 등의 기법을 적용합니다.

- **특징 추출 (Backbone):**
  - **초기 합성곱**: 초기 합성곱 레이어를 통해 저수준 특징을 추출합니다.
  - **C3K2 블록**: 이 블록을 사용하여 더 깊은 특징을 효율적으로 추출합니다.
- **특징 집계 (Neck):**
  - **SPPF 블록**: 여러 스케일에서 특징을 집계합니다.
  - **C2PSA 블록**: 중요한 영역에 집중하기 위해 공간적 주의 메커니즘을 적용합니다.
- **예측 (Head):**
  - **탐지 레이어**: 바운딩 박스, 객체 점수, 클래스 확률에 대한 예측을 출력합니다.
  - **앵커 박스 (사용 시)**: 다양한 크기의 객체를 탐지하기 위해 사전 정의된 박스를 사용합니다.

- **후처리:**
  - **비최대 억제 (NMS)**: 중복된 바운딩 박스를 제거하여 탐지 결과를 정제합니다.
  - **임계값 처리**: 낮은 확률의 탐지 결과를 필터링하기 위해 신뢰도 임계값을 적용합니다.
- **출력:**
  - **최종 탐지 결과**: 탐지된 객체의 위치, 크기, 클래스 라벨을 제공합니다.

YOLOv11 아키텍쳐

1.backbone

백본은 입력 이미지에서 여러 축척으로 특징을 추출하는 모델의 일부입니다. 일반적으로 컨볼루셔널 레이어와 블록을 쌓아 다양한 해상도에서 기능 맵을 만드는 작업이 포함됩니다.

```yaml
backbone:
  # [from, repeats, module, args]
  - [-1, 1, Conv, [64, 3, 2]] # 0-P1/2
  - [-1, 1, Conv, [128, 3, 2]] # 1-P2/4
  - [-1, 2, C3k2, [256, False, 0.25]]
  - [-1, 1, Conv, [256, 3, 2]] # 3-P3/8
  - [-1, 2, C3k2, [512, False, 0.25]]
  - [-1, 1, Conv, [512, 3, 2]] # 5-P4/16
  - [-1, 2, C3k2, [512, True]]
  - [-1, 1, Conv, [1024, 3, 2]] # 7-P5/32
  - [-1, 2, C3k2, [1024, True]]
  - [-1, 1, SPPF, [1024, 5]] # 9
  - [-1, 2, C2PSA, [1024]] # 10
```

변환 레이어: YOLO11은 초기 컨벌루션 계층을 사용하여 이미지를 다운샘플링하는 유사한 구조를 가지고 있습니다.

```yaml
- [-1, 1, Conv, [64, 3, 2]] # 0-P1/2
- [-1, 1, Conv, [128, 3, 2]] # 1-P2/4
```

C3K2(**C3 Block with k=2)** 블록: YOLO11은 C2f 대신 계산 측면에서 더 효율적인 C3k2 블록을 도입했습니다. C3k2는 여러 개의 경량화된 합성곱(convolution) 레이어로 구성된 블록이다. 이 블록은 주로 C3의 변형으로, 각 레이어가 특정 수의 필터를 사용해 정보를 효과적으로 압축하고, 동시에 깊이(depth)를 늘려 다양한 특징을 학습할 수 있게 설계되어있다. 이 레이어는 데이터 흐름을 원할하게 하면서도 정보 손실을 줄여, 작은 객체를 보다 잘 탐지할 수 있도록 돕는다.

CSP (Cross Stage Partial): CSP networks split the feature map and process one part through a bottleneck layer while merging the other part with the output of the

C2PSA : C2PSA는 두 개의 독립된 브랜치로 구성되는데, 각각의 브랜치는 서로 다른 공간적 정보와 채널 간 상호작용을 처리한다. 이로 인해 객체의 다양한 특징을 더 잘 포착

할 수 있다. 즉, 하나의 브랜치가 객체의 전반적인 형태와 위치 정보를 포착하는 반면, 다른 브랜치는 더 미세한 세부 정보를 추출하는 역할을 한다.

이 방식은 개체의 여러 해상도에서 중요한 특징을 더 잘 추출할 수 있게 하며, 특히 크기가 작은 객체를 탐지하는 데 유리하다. 기존 PSA는 모든 정보를 한 번에 처리했기 때문에 정보 손실이 있을 수 있었으나, C2PSA는 이를 분리해 보다 세밀하게 정보를 처리할 수 있다.

```
# YOLO11n head
head:
  - [-1, 1, nn.Upsample, [None, 2, "nearest"]]
  - [[-1, 6], 1, Concat, [1]] # cat backbone P4
  - [-1, 2, C3k2, [512, False]] # 13

  - [-1, 1, nn.Upsample, [None, 2, "nearest"]]
  - [[-1, 4], 1, Concat, [1]] # cat backbone P3
  - [-1, 2, C3k2, [256, False]] # 16 (P3/8-small)

  - [-1, 1, Conv, [256, 3, 2]]
  - [[-1, 13], 1, Concat, [1]] # cat head P4
  - [-1, 2, C3k2, [512, False]] # 19 (P4/16-medium)

  - [-1, 1, Conv, [512, 3, 2]]
  - [[-1, 10], 1, Concat, [1]] # cat head P5
  - [-1, 2, C3k2, [1024, True]] # 22 (P5/32-large)

  - [[16, 19, 22], 1, Detect, [nc]] # Detect(P3, P4, P5)
```

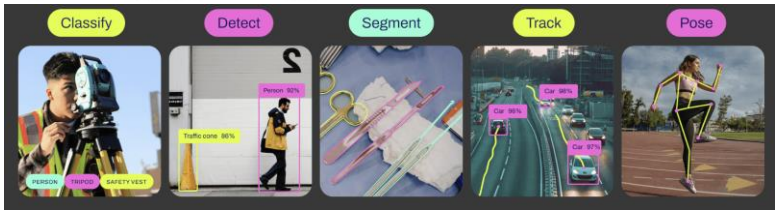
v11에서는 업샘플링된 피처맵을 C3k2블록과 결합해 사용.

v11의 head는 P3, P4, P5의 세 가지 피처맵을 기반으로 탐지를 수행

v11은 C3k2 블록에서 반복 횟수가 상대적으로 적고 피처의 상세한 처리가 이루어진다. 따라서 v11은 보다 경량화된 구조와 새로운 모듈을 도입해 성능을 높였다.

YOLOv11 개발 환경

학습데이터를 구글 코랩에 직접 업로드하거나 대용량인 경우에는 구글 드라이브를 코랩으로 마운트 시켜서 데이터를 업로드 하는 방식으로 사용



YOLOv11 프로젝트 개발 프로세스: 총 5단계로 구성된다.

1. 학습 데이터 만들기

2. 시스템에 업로드

3. YOLOv11 설치

4. 학습

5. 예측

본 프로젝트에서는 Custom Dataset를 사용하므로 모델을 재학습(Fine Tuning) 하는 경우에는 Image/Annotation으로 이루어진 Dataset을 준비해야 함.

MS COCO Dataset으로 사전학습된 YOLOv11 모델의 prediction 기능만을 이용하는 경우라 면 예측에 사용할 Image를 준비함 (즉, 주어진 Image 정답에 해당하는 Annotation은 필요치 않음)

직접 labeling 한 Image/Annotation 으로 이루어진 데이터를 이용하여 Traing Dataset을 구축.

1. 학습데이터 준비

```
from google.colab import drive
drive.mount('/content/drive')
```

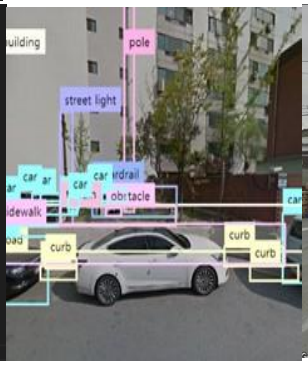
```
Mounted at /content/drive
```

```
%pip install ultralytics
import ultralytics
ultralytics.checks()
```

구글 드라이브를 마운트 하고

학습할 데이터를 준비

```
import yaml

data = {
    'train': '/content/dataset/train/images',
    'val': '/content/dataset/valid/images',
    'test': '/content/dataset/test/images',
    'names': [
        'bicycle', 'building', 'bus', 'car', 'cctv', 'crosswalk',
        'curb', 'guardrail', 'lanemarking', 'manhole', 'motorcycle',
        'obstacle', 'parking', 'person', 'pole', 'road', 'road-sign',
        'safety-sign', 'sidewalk', 'speed-bump', 'street-light',
        'terrain', 'traffic-sign', 'truck', 'utility-pole'
    ]
}

# 데이터를 yolov11 학습에 필요한 data.yaml 저장
with open('/content/dataset/data.yaml', 'w') as file:
    yaml.dump(data, file)
# 화면 출력
with open('/content/dataset/data.yaml','r') as file :
    street_yaml =yaml.safe_load(file)
    display(street_yaml)
```



2. 시스템에 업로드

```
import zipfile
import os

# 데이터 셋 경로
zip_file_path = '/content/drive/MyDrive/street view-obb.zip'
unzip_folder = '/content/dataset'

# Unzip 데이터셋
with zipfile.ZipFile(zip_file_path, 'r') as zip_ref:
    zip_ref.extractall(unzip_folder)

# 확인
os.listdir(unzip_folder)
```

```
['README.dataset.txt',
 'README.roboflow.txt',
 'test',
 'train',
 'data.yaml',
 'valid']
```

전처리, 라벨링 한 데이터를 구글드라이브에 업로드.

코랩에서 경로를 지정해준 뒤 압축해제.

데이터 셋의 경우 아래와 같은 형식으로 구성

Street_view_project/

├──Train/

| ├── images/

| └── labels/

|

├──Test/

| ├── images/

| └── labels/

|

├──Valid/

| ├── images/

| └── labels/

|

└──data.yaml

3. YOLOv11 설치

```
%pip install ultralytics
import ultralytics
ultralytics.checks()
```

```
Ultralytics 8.3.15 🚀 Python-3.10.12 torch-2.4.1+cu121 CUDA:0 (Tesla T4, 15102MiB)
Setup complete ✅ (2 CPUs, 12.7 GB RAM, 29.2/112.6 GB disk)
```

4. 학습

YOLOv11 커스텀 데이터 학습하기

```
from ultralytics import YOLO
model=YOLO('yolo11n.pt')
model.train(data='/content/dataset/data.yaml',epochs=100,patience=30,batch=32,imgsz=416)
```

```
Ultralytics 8.3.15 🚀 Python-3.10.12 torch-2.4.1+cu121 CUDA:0 (Tesla T4, 15102MiB)
engine/trainer: task=detect, mode=train, model=yolo11n.pt, data=/content/dataset/data.yaml, epochs=100, time=None,
Overriding model.yaml nc=80 with nc=25
```

```
23        [16, 19, 22]  1    435547  ultralytics.nn.modules.head.Detect           [25, [64, 128, 256]]
YOLO11n summary: 319 layers, 2,594,715 parameters, 2,594,699 gradients, 6.5 GFLOPs

Transferred 79/499 items from pretrained weights
TensorBoard: Start with 'tensorboard --logdir runs/detect/train', view at http://localhost:6006/
Freezing layer 'model.23.dfl.conv.weight'
AMP: running Automatic Mixed Precision (AMP) checks with YOLO11n...
AMP: checks passed ✅
train: Scanning /content/dataset/train/labels... 700 images, 0 backgrounds, 0 corrupt: 100%|██████████| 700/700 [00:00<00:00, 1081.91it/s]tr

train: New cache created: /content/dataset/train/labels.cache
albumentations: Blur(p=0.01, blur_limit=(3, 7)), MedianBlur(p=0.01, blur_limit=(3, 7)), ToGray(p=0.01, num_output_channels=3, method='weigh
/usr/local/lib/python3.10/dist-packages/albumentations/__init__.py:13: UserWarning: A new version of Albumentations is available: 1.4.18 (yo
  check_for_updates()
val: Scanning /content/dataset/valid/labels... 200 images, 0 backgrounds, 0 corrupt: 100%|██████████| 200/200 [00:00<00:00, 320.85it/s]val:

Plotting labels to runs/detect/train/labels.jpg...
optimizer: 'optimizer=auto' found, ignoring 'lr0=0.01' and 'momentum=0.937' and determining best 'optimizer', 'lr0' and 'momentum' automatic
optimizer: AdamW(lr=0.000345, momentum=0.9) with parameter groups 81 weight(decay=0.0), 88 weight(decay=0.0005), 87 bias(decay=0.0)
TensorBoard: model graph visualization added ✅
Image sizes 416 train, 416 val
Using 2 dataloader workers
Logging results to runs/detect/train
Starting training for 100 epochs...

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
      1/100      2.62G      3.354      4.745      2.828       1033        416: 100%|██████████| 22/22 [00:33<00:00,  1.52s/it]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 4/4 [00:02<00:00,  1.41it/s]

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
      2/100      2.65G      3.046      4.649       2.61         899        416: 100%|██████████| 22/22 [00:28<00:00,  1.30s/it]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 4/4 [00:02<00:00,  1.36it/s]

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
      3/100      2.73G      2.681      4.457      2.363        1027        416: 100%|██████████| 22/22 [00:25<00:00,  1.17s/it]
```

data.yaml에 기술되어 있는 커스텀 데이터로 학습했기 때문에 학습을 마친 후에
model.names 값을 보면. pretrained된 ms coco 데이터 클래스가 아닌 내가 설정한 25
개의 클래스의 이름 확인가능

```
print(type(model.names), len(model.names))
print(model.names)
```

```
<class 'dict'> 80
{0: 'person', 1: 'bicycle', 2: 'car', 3: 'motorcycle', 4: 'airplane', 5: 'bus', 6: 'train', 7: 'truck', 8: 'boat', 9: 'traffic light', 10: 'fire hydrant', 11: 'stop sign', 12: 'parking meter', 13: 't
```

5. 예측

내가 학습시킨 YOLOv11 이용해서 테스트 이미지 예측

```
results=model.predict(source='/content/dataset/valid/images/35_1_heading_0_jpg.rf.225c51b5593df6ea66d9bae66f46b699.jpg', save=True)

image 1/1 /content/dataset/valid/images/35_1_heading_0_jpg.rf.225c51b5593df6ea66d9bae66f46b699.jpg: 640x640 3 buildings, 5 cars, 1 cctv, 4 curbs, 1 guardrail, 1 road, 6 safety-signs, 1 sidewalk, 11.8ms
Speed: 2.1ms preprocess, 11.8ms inference, 1.5ms postprocess per image at shape (1, 3, 640, 640)
Results saved to runs/detect/predict2
```

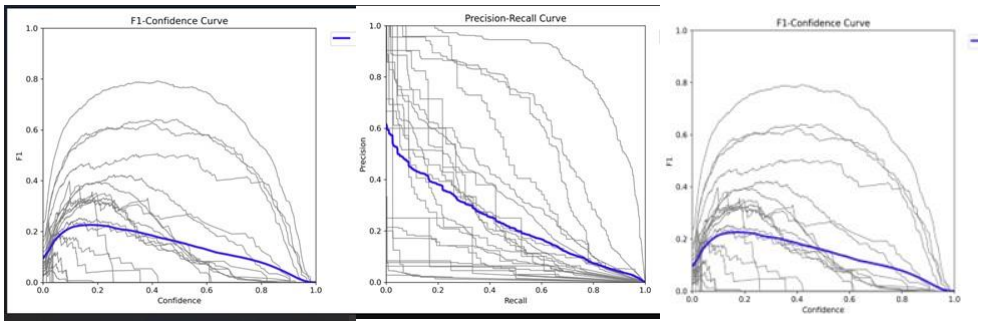
본 연구에서는 Google Maps API를 이용하여 금천구 전역에서 30m 간격으로 거리뷰 이미지를 수집하였다. 이후, DeepLabv3+ 딥러닝 모델을 이용하여 해당 이미지에서 보행자 영역을 추출하였고, 검출률과 정밀도 측면에서 성능을 평가하였다. 그 결과, 검출률은 %, 정밀도는 %로 나타났다.

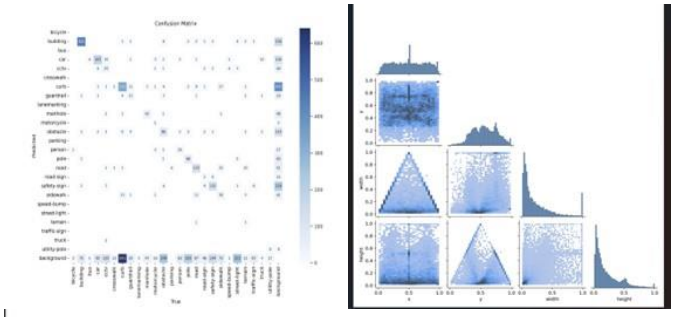
이러한 결과를 바탕으로, 향후에는 보다 많은 지역에서 대규모 데이터를 수집하고, 다양한 종류의 객체를 탐지하고 분할할 수 있는 기술을 개발할 예정이다. 또한, 실제 환경에서 적용 가능한 시스템을 구축하여 보행자 안전을 향상시키는 데 기여할 것이다.

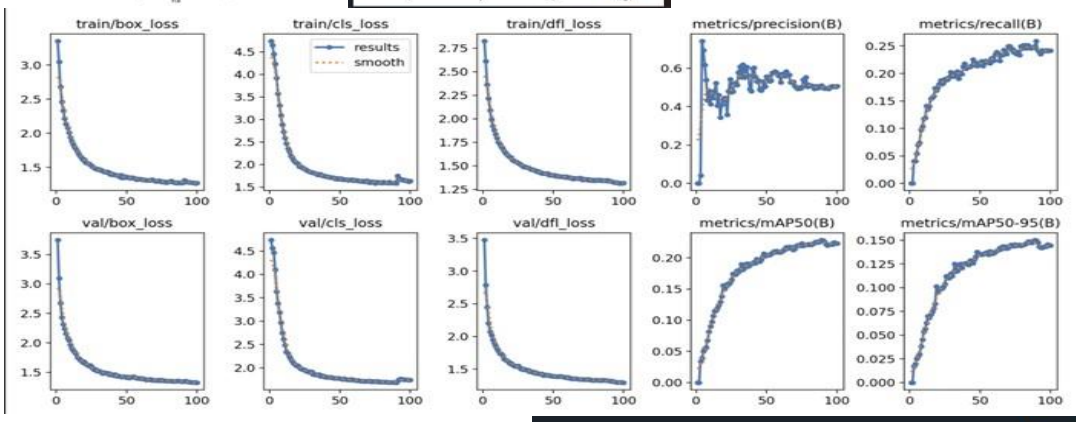
4. 연구결과

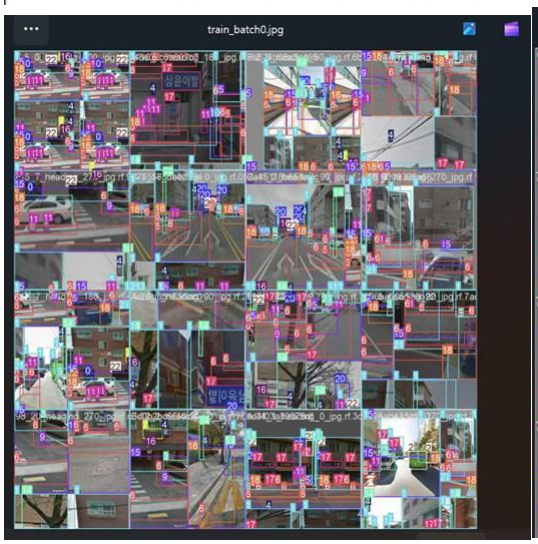
Recall , precision , f1 confidence curve ~~, confusion matrix, metrix/map50 , train batch, val batch ~

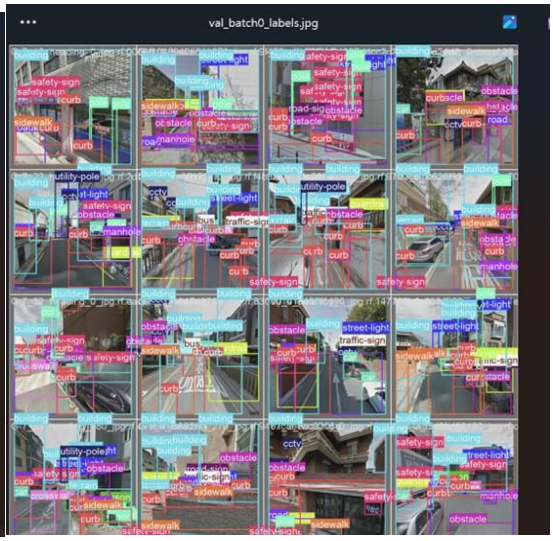
여러 성능평가 지표와 각 설명 추가하기. + 개선방향이나 ~~ 등











본 프로젝트를 통해 수집된 금천구 데이터는 보행자와 차량 간의 충돌을 줄이기 위한 도시 계획에 활용될 수 있습니다. 예를 들어, 좁은 골목에서 보행자와 차량 간의 안전 거리를 확보하거나, 위험 지역을 사전에 탐지하여 교통 관리 시스템에 반영하는 등의 실질적인 안전 대책 마련에 기여할 수 있습니다. 또한, 자율 주행 기술에 있어서도 중요한 역할을 합니다. 골목길과 같이 고위험 지역에서 보행자를 실시간으로 탐지하고 도로 및 보행로를 구분하여 자율 주행 차량이 안전하게 주행할 수 있도록 지원하는 데이터를 제공할 수 있습니다.

본 연구에서는 보행자 안전을 위한 실시간 객체 탐지 및 분할 기술 개발을 위해 딥러닝 모델을 활용하였다. 다양한 딥러닝 모델 중에서 YOLOv11와 Faster R-CNN 모델을 선택

하여 실험을 진행하였으며, 다음과 같은 결과를 얻었다.

먼저, YOLOv11 모델의 경우 검출 정확도는 mAP 기준으로 %로 나타났으며, 객체 분할 정확도는 Dice Coefficient 기준으로 ~로 나타났다. 이는 기존의 다른 딥러닝 모델들과 비교했을 때 높은 수준의 성능이다. 또한, 초당 ~프레임 이상의 속도로 객체를 탐지하고 분할할 수 있어 실시간 처리가 가능하다.

Faster R-CNN 모델의 경우 검출 정확도는 mAP 기준으로 ~%로 나타났으며, 객체 분할 정확도는 Dice Coefficient 기준으로 ~로 나타났다. 역시나 빠른 속도로 객체를 탐지하고 분할할 수 있으며, 특히 객체의 경계를 정확하게 구분할 수 있다는 장점이 있다.

실제 도로 환경에서의 적용 가능성을 검토하기 위해 서울 시내 금천구의 도로 영상을 수집하여 실험을 진행할 것이다. ~~. 두 모델 모두 실제 도로 환경에서도 높은 수준의 성능을 유지할 수 있다. 다만, 날씨나 조명 등의 환경 변화에 따라 성능이 다소 저하될 수 있으므로 추가적인 연구가 필요하다

결론적으로, 본 연구에서 개발한 딥러닝 모델은 보행자 안전을 위한 실시간 객체 탐지 및 분할 기술에 매우 유용하게 활용될 수 있을 것으로 기대된다. 향후에는 보다 다양한 환경에서 테스트를 진행하고, 이를 바탕으로 더욱 개선된 모델을 개발할 계획이다.

향후 연구 방향으로는 더 많은 데이터를 수집하여 모델의 학습을 강화하고, 다양한 환경에서 실험을 수행하여 일반화 성능을 더욱 향상시키는 것이 필요하다. 또한, 제안된 기술을 실제 교통 안전 시스템에 적용하여 효과를 검증하는 연구도 필요하다. 마지막으로, 인공지능 기술을 활용하여 보행자 안전을 증진시킬 수 있는 다양한 방안을 모색하는 것이 중요하다.

보행자 안전을 위해 본 연구에서 제안한 실시간 객체 탐지 및 분할 기술은 교통 안전 시스템에 적용되어 보행자 사고 예방에 큰 도움이 될 것으로 기대된다. 따라서, 이러한 기술을 적극적으로 활용하여 보행자 안전을 증진시키는 노력이 필요하다.

클래스명(25개)에 대한 설명

0: bicycle(자전거)

1: building(건물) 사람이 거주하거나 사용하는 구조물로, 도시나 주거지에서 흔히 볼 수 있음. ex)고층 아파트, 상업용 건물.

2: bus(버스)

3. car(승용차)

4. cctv(cctv카메라) 공공장소에서 보안 및 감시 목적으로 설치된 카메라. ex) 차로에 설치된 CCTV, 건물 입구에 설치된 보안 카메라.

5. crosswalk(횡단보도) 보행자가 도로를 건널 수 있는 곳에 표시된 안전 구역. ex) 도로에 그려진 흰색 줄무늬 횡단보도

6. curb(연석) 도로와 인도를 구분하는 경계석. 인도의 경계에 있는 돌로 된 연석.

7. guardrail(가드레일) 도로나 고속도로에서 차량의 이탈을 막기 위한 안전장치. 고속속도로 옆의 철제 가드레일.

8. lanemarking(차선) 도로에서 차선을 구분하기 위해 그려진 선 ex) 중앙선, 차선 변경 구역에 그려진 실선과 점선

9. manhole(맨홀) 하수도나 지하 설비 점검을 위해 설치된 뚜껑이 있는 구멍

10. motorcycle(오토바이)

11. obstacle(장애물) 도로나 인도에 있어 보행자나 차량의 이동을 방해하는 물체 ex) 도로에 떨어진 나뭇가지, 공사로 인한 임시 장벽.

12. parking(주차구역) 차량이 주차할 수 있는 공간. ex) 노상 주차 구역, 주차장에 표시된 주차 칸.

13. person(사람) 보행자, 운전자를 포함한 모든 사람.

14. pole(기둥) 전기, 조명, 교통 표지 등을 지탱하는 긴 기둥 ex) 가로등 기둥, 전신주.

15. road(도로) 차량과 보행자가 다니는 통행로. ex) 아스팔트로 포장된 도로, 비포장 시골길.

16. road-sign(도로 표지판) 교통 규칙이나 정보를 제공하는 표지판 ex) 일방통행, 제한 속도 표지판, 우회전 금지 표지판

17. safety-sign(안전 표지판) 보행자나 운전자에게 안전을 경고하거나 주의사항을 알리는 표지판. ex) 공사 중 경고 표지판, 미끄러운 도로 경고 표지판.

18. sidewalk(인도) 보행자가 다닐 수 있는 도로 옆의 길. ex) 나무가 심어진 인도, 자전거와 함께 이용하는 보행자 도로.

19. speed-bump(과속방지턱) 차량의 속도를 줄이기 위해 도로에 설치된 도로 경사 ex)

주차장 입구의 과속 방지턱

20. street-light(가로등) 밤에 도로나 거리를 비추는 조명 장치 ex) 도심의 가로등, 공원 내 설치된 가로등

21. terrain(지형) 도로나 보행로를 포함한 지형적 특성. ex) 평지, 경사진 도로

22. traffic-sign(교통 표지판) 교통 흐름을 안내하거나 통제하는 도로 표지판 ex) 횡단보도 표지판, 우회전 금지 표지판

23. truck(트럭) 물건을 운반하는 대형 차량 ex) 화물 운반용 트럭, 이삿짐 차량.

24. utility-pole(전신주) 전선이나 통신선을 지지하는 높은 기둥. ex) 전선을 지탱하는 전신주.

프로젝트 개요서(요약)

프로젝트명	# 프로젝트명 작성
팀 구성원	김선권
수행기간	# 2024년9월2일~ 0월0일
프로젝트 분야	# cv ~
추진배경 및 필요성	보행자 안전은 지속적으로 중요성이 강조되는 문제로, 특히 도심의 좁은 골목길과 주거 밀집 지역에서 보행자와 차량 간의 사고 위험이 높습니다. 금천구와 같은 사고 빈도가 높은 지역에서 교통 사고를 예방하기 위한 실시간 모니터링 시스템의 필요성이 증대되고 있습니다. 자율 주행 기술의 발전과 함께 교통 관리와 도시 계획에서 보행자의 안전성을 강화하기 위한 기술적 기반 마련이 필수적입니다. 본 프로젝트는 이러한 배경을 바탕으로 보행자 사고 예방을 위해 실시간 객체 탐지와 분할 기술을 적용하는 시스템을 개발합니다.
프로젝트 소개	본 프로젝트는 금천구의 거리뷰 이미지를 활용하여 실시간 객체 탐지 및 이미지 분할 모델을 개발하는 것을 목표로 합니다. Google Maps API를 통해 수집한 이미지를 바탕으로YOLOv11과DeepLabv3+ 같은 최신 인공지능 기술을 사용하여 보행자, 차량, 신호등 등의 객체를 탐지하고 분석합니다. 이러한 분석 데이터를 바탕으로 자율 주행 시스템 및 도시 안전 관리에 적용할 수 있는 실시간 안전 관리 시스템을 구축합니다.
프로젝트 특장점	**실시간 객체 탐지**: YOLOv8 모델을 통해 보행자, 차량, 신호등 등 다양한 객체를 실시간으로 탐지할 수 있습니다. **정확한 이미지 분할**: DeepLabv3+ 모델을 사용하여 도로와 보행로 등 중요한 인프라를 픽셀 단위로 분할하여 분석합니다. **금천구 지역 맞춤형 데이터 분석**: 금천구의 실제 거리 데이터를 활용하여 구체적인 사고 위험 구역을 분석하고 대응 방안을 마련합니다. **자율 주행 및 도시 계획에 적용 가능**: 수집된 데이터를 기반으로 자율 주행 시스템을 개선하거나, 도시 계획에서 보행자 안전을 강화하는 데 사용할 수 있습니다.
주요 기능	**Google Maps API 기반 데이터 수집**: 금천구 전역에서50m 간격으로 거리뷰 이미지를 수집합니다. **YOLOv11 기반 실시간 객체 탐지**: 수집한 이미지에서 보행자, 차량, 신호등 등의 객체를 탐지합니다. **DeepLabv3+ 기반 이미지 분할**: 도로와 보행로 등을 정확하게 분할하여 분석합니다. **실시간 모니터링 시스템**: 객체 탐지 및 이미지 분할 결과를 실시간으로 제공하여 보행자 안전을 모니터링합니다. 간단한 앱개발이나 웹페이지?
기대효과 및 활용 분야	**보행자 안전 강화**: 고위험 지역에서의 보행자 사고를 사전에 예방할 수 있습니다. **자율 주행 기술 고도화**: 객체 탐지 및 분할 데이터를 통해 자율 주행 차량의 안전성을 높일 수 있습니다. **도시 계획 및 정책 지원**: 데이터 기반으로 교통 관리와 도시 설계에서 보행자 안전을 고려한 인프라 개선 방안을 제시할 수 있습니다. **스마트 시티 구현**: 실시간 객체 탐지 데이터를 활용하여 스마트 시티 구축에 필요한 교통 관리 및 안전 인프라를 지원할 수 있습니다.