

### 과제3: N-Seoul Tower 객체 검출

전석남

**pip install -r requirements.txt: 약 16~20분 / 추론: 약 10~20초 소요 예상.**

#### 0. 과제 목표

임의의 이미지가 입력되었을 때 해당 이미지에서 N Seoul Tower의 존재 여부를 판별하고, 만약 N Seoul Tower가 있다면 위치(좌표)까지 출력하는 프로그램 제작.

#### 명령어 형식

```
$ python3 main.py --input [이미지 경로] --task presence
```

→ N-Tower가 존재한다면 true, 없다면 false 출력

```
$ python3 main.py --input [이미지 경로] --task bbox
```

→ N-Tower가 존재한다면 해당 위치를 (x, y, width, height) 형태로 출력, 없다면 none 출력

#### 1. 알고리즘 개요(검출/분류 핵심 아이디어)

본 과제에서는 N Seoul Tower의 존재 여부를 판단(Classification)하고 위치를 특정(Localization)하기 위해, 최신 객체 검출 모델인 YOLOv8 (You Only Look Once version 8)을 기반으로 전이 학습(Transfer Learning)을 수행하였다. 주요 핵심 알고리즘 및 전략은 다음과 같다.

##### 1) YOLOv8 기반의 One-stage Object Detection

입력 이미지를 그리드(Grid)로 나누고 객체의 위치(Bounding Box)와 클래스 확률(Class Probability)을 한 번의 신경망 연산으로 동시에 예측하는 **One-stage Detector** 구조를 채택하였다. 이는 Two-stage 방식(R-CNN 계열)에 비해 추론 속도가 매우 빠르며, 실시간 검출에 유리하다. 본 프로젝트에서는 제한된 컴퓨팅 자원과 빠른 추론을 고려하여 경량화 모델인 yolov8n (nano) 모델을 사용하였다.

##### 2) 전이 학습 (Transfer Learning) 및 Fine-tuning

방대한 데이터셋(COCO 등)으로 사전 학습된(Pre-trained) 가중치를 초기값으로 사용하였다. 사전 학습된 모델은 이미지의 저수준 특징(Edge, Texture 등)을 이미 학습하고 있으므로, 적은 양의 N Seoul Tower 데이터만으로도 높은 성능을 낼 수 있다. 이를 기반으로 N Seoul Tower 데이터셋에 맞춰 모델의 Head 부분을 Fine-tuning 하여 단일 클래스(n-tower)에 최적화된 모델을 구축하였다.

##### 3) Hard Negative Mining (배경 이미지 학습)

N Seoul Tower 데이터만을 학습한 초기 모델이 형태가 유사한 다른 타워(동방명주, 도쿄타워 등)를 N Seoul Tower 로 오인식(False Positive)하는 문제가 발생했었다. 이를 해결하기 위해 **Negative Sampling** 전략을 적용하였다.

- **방법:** N Seoul Tower 가 포함되지 않은 유사 타워 이미지나 풍경 이미지를 학습 데이터에 포함하되, 라벨링(Annotation)을 수행하지 않았다(Null Label).
- **효과:** YOLO 모델은 라벨이 없는 이미지를 '배경(Background)'으로 인식하여 학습한다. 이를 통해 모델은 "비슷하게 생겼지만 정답이 아닌 특징"을 학습하게 되어, 결과적으로 오탐률을 낮추고 정밀도(Precision)를 향상시켰다.

#### 4) Confidence Thresholding 을 통한 최종 판단

학습된 모델을 사용하여 추론(predict)을 수행할 때, **Confidence Score(확신도)** 임계값(conf)을 **0.5** 로 설정하였다.

- **코드 로직:** 모델이 예측한 Bounding Box 중 확신도가 0.5 이상인 객체가 존재하면 True 및 좌표를 반환하고, 기준을 넘는 객체가 없으면 False 혹은 None 을 출력하도록 구현하였다.
- **선정 이유:** 0.5 의 임계값은 원거리의 작은 객체에 대한 검출률(Recall)과 오탐 방지(Precision) 사이의 균형을 맞추기 위해 실험적으로 결정된 값이다. 초기에는 0.7 로 높은 값을 할당하였는데, 그 결과 먼 거리에서 찍힌 N-Seoul Tower 는 탐지되지 않는 문제가 있었다. 따라서 0.5 로 값을 낮추어 실험해 보았고, 좋은 검출률을 보여 본 과제에서는  $conf = 0.5$  를 사용하였다.

## 2. 파라미터 설명

본 과제 수행을 위해 모델 학습 및 추론 단계에서 설정한 주요 하이퍼파라미터와 그 설정 근거는 다음과 같다.

### 2.1. 학습 파라미터 (Training Parameters)

- **Model Architecture: yolov8n.pt (Nano)**
  - **설명:** YOLOv8 모델군 중 가장 파라미터 수가 적고 가벼운 Nano 버전을 사용하였다.
  - **선정 근거:** 과제 수행 환경(Google Colab T4 GPU)의 메모리 제약을 고려하고, 빠른 학습과 추론 속도를 확보하기 위함이다. N Seoul Tower 와 같은 단일 클래스 객체 검출에는 Nano 모델의 복잡도로도 충분한 성능을 낼 수 있다고 판단하였다.
- **Epochs: 50**
  - **설명:** 전체 데이터셋에 대해 학습을 반복하는 횟수이다.

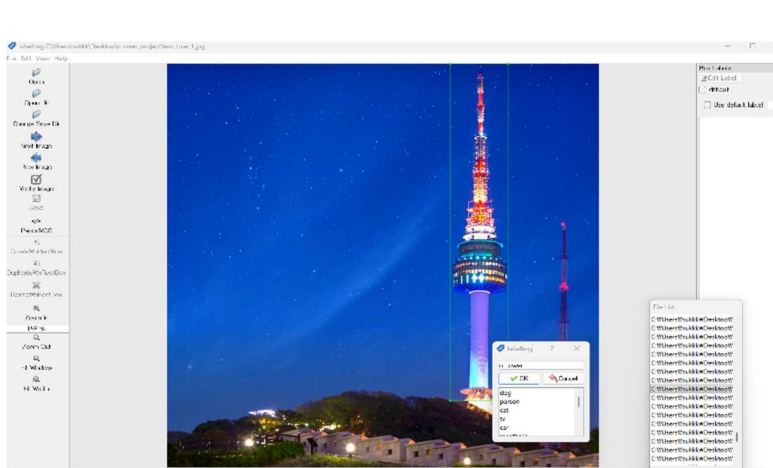
- **선택 근거:** 데이터셋의 크기(약 120 장)를 고려했을 때, 50 회 정도의 반복이면 Loss(손실) 값이 충분히 수렴하고 모델이 특징을 학습하기에 적절하다고 판단하였다. 과도한 Epoch 설정으로 인한 과적합(Overfitting)을 방지하기 위해 50 으로 제한하였다.
- **Image Size (imgsz): 640**
  - **설명:** 모델 입력으로 들어가는 이미지의 해상도이다 (640x640).
  - **선택 근거:** YOLO 모델의 표준 입력 크기이다. 원본 이미지의 비율을 유지하면서 리사이징(Letterbox resizing)하여, 타워의 전체적인 형상과 세부 특징을 잃지 않으면서 연산 효율성을 극대화하려고 하였다.
- **Batch Size: 16**
  - **설명:** 한 번의 가중치 업데이트를 위해 학습하는 데이터의 묶음 단위이다.
  - **선택 근거:** GPU 메모리 용량을 초과하지 않으면서 안정적인 학습(Gradient Descent)이 가능한 값으로 설정하였다.

## 2.2. 추론 파라미터 (Inference Parameters)

- **Confidence Threshold (conf): 0.5**
  - **설명:** 객체라고 판단하기 위한 최소한의 확신도(Probability) 기준값이다. 모델이 예측한 Bounding Box 중 신뢰도 점수가 0.5(50%) 이상인 것만 최종 결과로 출력한다.
  - **선택 근거:**
    - **Precision/Recall Trade-off:** 임계값을 너무 높게(예: 0.7) 설정할 경우, 흐리거나 멀리 있는 N Seoul Tower 를 검출하지 못하는 미탐(False Negative) 문제가 발생하였다.
    - **오탐 방지:** 반대로 너무 낮게 설정할 경우 유사한 형태의 다른 구조물을 오인식할 위험이 있다.
    - **결론:** 실험 결과 0.5 가 원거리 객체 검출률을 높이면서도 오탐을 효과적으로 제어할 수 있는 최적의 균형점이라 판단하여 최종 적용하였다.

## 3. YOLOv8n 모델 학습 과정

### 1) Dataset 수집 후 labellmg 프로그램으로 이미지 전처리



## 2) Colab 을 활용한 YOLO 학습

```

from ultralytics import YOLO

# 모델 로드
model = YOLO('yolov8n.pt')

# 학습 시작
# data 인자에 방금 만든 yaml 파일 경로를 넣어줌.
model.train(
    data='/content/dataset/data.yaml',
    epochs=50,
    imgs=640,
    project='/content/drive/MyDrive/ntower_result', # 결과물을 구글 드라이브에 바로 저장 (날라람 방지)
    name='train_result'
)

...
20      [-1, 9] 1      0      ultralytics.nn.modules.conv.Concat      [1]
21      -1      1      493056      ultralytics.nn.modules.block.C2f      [384, 256, 1]
22      [15, 18, 21] 1      751507      ultralytics.nn.modules.head.Detect      [1, [64, 128, 256]]
Model summary: 129 layers, 3,011,043 parameters, 3,011,027 gradients, 8.2 GFLOPs

Transferred 319/355 items from pretrained weights
Freezing layer 'model_22.dfl.conv.weight'
AMP: running Automatic Mixed Precision (AMP) checks...
Downloading https://github.com/ultralytics/assets/releases/download/v8.3.0/yolo10n.pt to 'yolo10n.pt': 100% ————— 5.4MB 84.2MB/s 0.1s
AMP: checks passed
train: Fast image access (ping: 0.0±0.0 ms, read: 1844.1±2073.3 MB/s, size: 419.6 KB)
train: Scanning /content/dataset/ntower_project/train/labels... 72 images, 29 backgrounds, 0 corrupt: 100% ————— 101/101 1.1K/s 0.1s
train: New cache created: /content/dataset/ntower_project/train/labels.cache
augmentations: Blur(p=0.01, blur_limit=(3, 7)), MedianBlur(p=0.01, blur_limit=(3, 7)), ToGray(p=0.01, method='weighted_average', num_output_channels=3), CLAHE(p=0.01, clip_limit=(1.0
val: Fast image access (ping: 0.0±0.0 ms, read: 922.6±539.9 MB/s, size: 72.6 KB)
val: Scanning /content/dataset/ntower_project/valid/labels... 16 images, 9 backgrounds, 0 corrupt: 100% ————— 25/25 72.6K/s 0.3s
val: New cache created: /content/dataset/ntower_project/valid/labels.cache
Plotting labels to /content/drive/MyDrive/ntower_result/train_result/labels.jpg...
optimizer: 'optimizerauto' found, ignoring 'lr0=0.01' and 'momentum=0.937' and determining best 'optimizer', 'lr0' and 'momentum' automatically...
optimizer: Adam(lr=0.002, momentum=0.9) with parameter groups 57 weight(decay=0.0), 64 weight(decay=0.0005), 63 bias(decay=0.0)

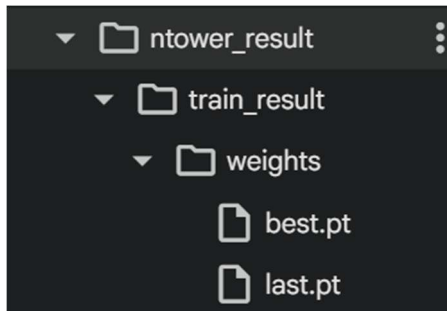
```

(사용한 이미지 데이터: N Seoul Tower 100 장, Not N Seoul Tower 40 장)

## 3) YOLO 학습 완료

Epoch	GPU_mem	box_loss	cls_loss	dfl_loss	Instances	Size
47/50	3.12G	0.8625	1.039	0.9631	5	640: 100% ————— 7/7 4.8it/s 1.5s
	Class	Images	Instances	Box(P)	R	mAP50 mAP50-95): 100% ————— 1/1 3.8it/s 0.3s
	all	25	16	0.99	1	0.995 0.758
Epoch	GPU_mem	box_loss	cls_loss	dfl_loss	Instances	Size
48/50	3.15G	0.8063	0.9859	0.9534	3	640: 100% ————— 7/7 4.8it/s 1.4s
	Class	Images	Instances	Box(P)	R	mAP50 mAP50-95): 100% ————— 1/1 6.5it/s 0.2s
	all	25	16	0.991	1	0.995 0.745
Epoch	GPU_mem	box_loss	cls_loss	dfl_loss	Instances	Size
49/50	3.16G	0.7351	0.9816	0.9226	3	640: 100% ————— 7/7 4.9it/s 1.4s
	Class	Images	Instances	Box(P)	R	mAP50 mAP50-95): 100% ————— 1/1 5.1it/s 0.2s
	all	25	16	0.992	1	0.995 0.724
Epoch	GPU_mem	box_loss	cls_loss	dfl_loss	Instances	Size
50/50	3.18G	0.7741	0.9558	0.9467	5	640: 100% ————— 7/7 4.6it/s 1.5s
	Class	Images	Instances	Box(P)	R	mAP50 mAP50-95): 100% ————— 1/1 4.5it/s 0.2s
	all	25	16	0.992	1	0.995 0.712

#### 4) 가중치 파일 추출(best.pt)



#### 5) main.py 에서 bset.pt 가중치로 동작하는 YOLO 모델로 추론 실행

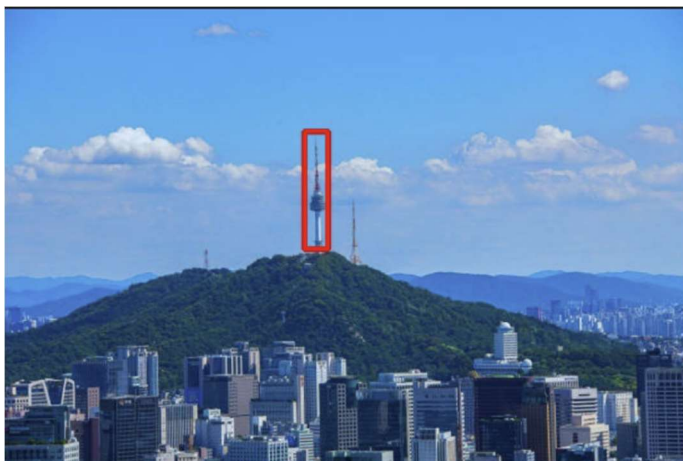
```
def main():
    args = parse_args()

    # 1. 모델 로드
    try:
        model = YOLO('best.pt')
    except Exception as e:
        print(f"Error loading model: {e}")
        return 1

    # 2. 이미지 추론
    try:
        results = model.predict(source=args.input, save=False, verbose=False, conf=0.5)
    except Exception as e:
        print(f"Error prediction: {e}")
        return 1
```

#### 4. 프로그램 실행 결과

##### 4.1. 입력 이미지에 N Seoul Tower 가 있는 경우



```
(.venv) jeon@BOOK-VK6EDB557F:/mnt/c/Users/sukkk/Desktop/CV_assign_3/n-tower-jeonseoknam$ python3 main.py --input test_true_8.jpg --task presence true
```

(좌표가 올바른지 디버깅을 하기 위해 임시로 --vis\_output [출력 이미지 이름] 형식의 코드를 작성하였다.)

(명령어 실행시 --vis\_output 옵션을 빼고 실행하면 과제 명세대로 좌표만 출력.)

```
(.venv) jeon@BOOK-VK6EDB557F:/mnt/c/Users/sukkk/Desktop/CV_assign_3/n-tower-jeonseoknam$ python3 main.py --input test_true_8.jpg --task bbox --vis_output result.jpg 267,108,21,104
```

입력 이미지에 N-Tower가 있으므로 --task presence 옵션시 true 출력, --task bbox 옵션시 타워의 좌표 출력

## 4.2. 입력 이미지에 N Seoul Tower 가 없는 경우



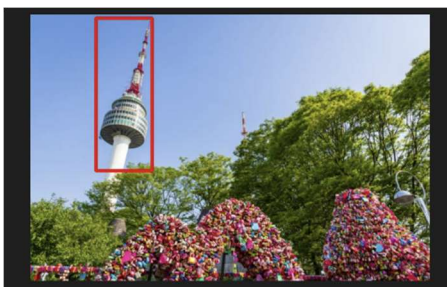
```
(.venv) jeon@BOOK-VK6EDB557F:/mnt/c/Users/sukkk/Desktop/CV_assign_3/n-tower-jeonseoknam$ python3 main.py --input test_false_1.jpg --task presence false
(.venv) jeon@BOOK-VK6EDB557F:/mnt/c/Users/sukkk/Desktop/CV_assign_3/n-tower-jeonseoknam$ python3 main.py --input test_false_1.jpg --task bbox none
```

명세와 동일하게 --task presence 옵션 사용시 N-Tower가 없으므로 false 출력, --task bbox 옵션 사용시 none 출력.

## 5. 한계/오류 사례 및 개선 방안

본 프로젝트를 통해 N Seoul Tower 를 효과적으로 검출하는 모델을 구현하였으나, 구조적인 한계와 환경적 요인으로 인한 몇 가지 개선점이 식별되었다.

### 1) Axis-Aligned Bounding Box (AABB)의 구조적 한계



- **현상:** N Seoul Tower 와 같이 세로로 길거나 대각선으로 기울어진 객체를 검출할 때, Bounding Box 가 객체에 타이트하게 밀착되지 않고 주변의 배경(하늘, 나무 등)을 과도하게 포함하는 현상이 발생할 수 있다.
- **원인:** YOLO 를 포함한 일반적인 객체 검출 모델은 이미지의 x 축, y 축에 평행한 직사각형(**Axis-Aligned Bounding Box**) 형태로만 위치를 예측한다. 타워가 비스듬하게 촬영되거나 얇은 구조를 가질 경우, 객체의 가장 끝점들을 포함하기 위해 필연적으로 빈 공간(여백)이 박스 안에 포함되게 된다. 이는 단순히 학습 부족의 문제가 아니라, '사각형 박스'로만 세상을 표현하는 Detection Task 의 근본적인 한계이다.
- **개선 방안:**
  - **Instance Segmentation 도입:** 객체를 박스(Box)가 아닌 픽셀(Pixel) 단위로 검출하는 YOLOv8-seg 모델을 사용하여 배경을 제외한 타워 영역만 정밀하게 추출한다.
  - **Oriented Bounding Box (OBB) 적용:** 직사각형을 회전시킬 수 있는 OBB 모델을 사용하여, 기울어진 타워의 각도에 맞춰 박스를 회전시킴으로써 배경 노이즈를 최소화한다.
  - 하지만 과제 명세에는 bounding box 로 출력할 것을 명시하고 있기 때문에, 위 개선 방안을 적용하여 프로그램을 구성하지는 않았다.

## 2) 조도 변화 및 기상 악화 시 검출 성능 저하 가능성

- **현상:** 학습 데이터의 대부분이 주간의 맑은 날씨 환경으로 구성되어 있어, 야간, 안개, 비, 역광 등 조도 변화가 심한 환경에서는 검출률(Recall)이 저하될 가능성이 있다.
- **원인:** 딥러닝 모델은 학습 데이터의 분포(Distribution)에 편향되는 경향이 있다. N Seoul Tower 의 야경이나 흐린 날의 특징 데이터가 부족할 경우 모델은 이를 새로운 객체로 오인하거나 배경으로 처리할 수 있다.
- **개선 방안:**
  - **데이터 증강(Augmentation) 강화:** 학습 시 Brightness(밝기), Contrast(대비), Blur(블러), Noise(노이즈) 등의 증강 기법을 무작위로 적용하여 악천후 환경에 대한 모델의 강건성(Robustness)을 확보한다.
  - **다양한 환경 데이터 수집:** 야간 조명이나 안개 낀 날의 이미지를 추가로 수집하여 Fine-tuning 을 수행한다. 본 과제에서는 이 방법을 사용하였다.

## 3) 원거리 소형 객체에 대한 정보 손실

- **현상:** N Seoul Tower 가 이미지 내에서 매우 작게(멀리) 촬영된 경우, conf 임계값을 낮춰야만 검출되거나 아예 검출되지 않는 사례가 발생한다.
- **원인:** CNN(Convolutional Neural Network) 구조상 레이어를 통과할수록 이미지가 압축(Down-sampling)되는데, 이 과정에서 작은 객체의 픽셀 정보는 소실되거나 배경 특징에 묻히기 쉽다.
- **개선 방안:**

- **고해상도 입력 사용:** 학습 및 추론 시 이미지 입력 크기(imgsz)를 640 에서 1280 등으로 상향하여 작은 객체의 픽셀 정보를 보존한다.
- **SAHI (Slicing Aided Hyper Inference) 기법 적용:** 큰 이미지를 여러 조각(Slice)으로 잘라서 각각 추론한 뒤 합치는 기법을 도입하여, 원거리 객체를 마치 가까이 있는 것처럼 확대하여 검출하는 효과를 낸다.

## 6. 결론

본 프로젝트에서는 최신 딥러닝 객체 검출 모델인 YOLOv8(Nano)을 활용하여 N Seoul Tower 의 존재 여부를 판별하고 그 위치를 시각화하는 프로그램을 구현하였다.

### 1) 프로젝트 결과 요약

제한된 데이터셋 환경에서도 전이 학습(Transfer Learning)과 Fine-tuning 기법을 적용하여 모델을 빠르게 최적화하였다. 특히, 동방명주나 도교타워와 같이 형태적으로 유사한 객체를 오인식하는 문제를 해결하기 위해, 라벨링이 없는 배경 이미지를 학습에 포함시키는 Negative Mining 전략을 수행하여 모델의 변별력을 크게 강화하였다. 또한, 원거리의 소형 객체 검출률(Recall)을 확보하기 위해 Confidence Threshold 를 실험적으로 조정(0.5)하여 최적의 성능 균형점을 도출하였다.

### 2) 기술적 통찰 및 의의

과제 수행 과정을 통해 Object Detection(객체 검출)의 핵심 원리와 한계를 명확히 이해할 수 있었다. 특히 Bounding Box 방식(AABB)이 가지는 구조적 한계로 인해, 타워와 같이 세로로 긴 객체를 검출할 때 배경 노이즈가 포함되는 현상을 확인하였다. 이는 단순한 모델 성능의 문제가 아니라 Task 정의(Detection vs Segmentation)에 따른 차이임을 인지하였으며, 향후 Instance Segmentation 이나 OBB(Oriented Bounding Box) 기술 도입을 통해 개선될 수 있을 것이라 생각한다.

### 3) 최종 결론

결론적으로 본 시스템은 주어진 과제 요구사항인 '존재 여부 판단(presence)'과 '위치 좌표 출력(bbox)'을 충실히 수행하며, 다양한 환경의 이미지에서도 준수한 검출 성능을 보였다. 특히 박스의 정확도와 검출율을 종합적으로 보여주는 mAP50 기준으로 약 0.995 의 성능을 기록하였다. 이번 과제를 통해서 딥러닝 같은 AI 모델까지 활용하여 컴퓨터 비전 Task 를 해결하는 것이, 수업을 통해 공부했던 이론, 규칙 기반만을 사용하여 Task 를 해결하는 것보다 더 좋은 성능을 내기에 유리하다는 것을 느꼈다. 더불어 데이터 중심(Data-centric)의 AI 개발 방법론과 하이퍼 파라미터 튜닝의 중요성을 체득하는 계기가 되었다. 그렇지만, AI 모델을 잘 활용하거나 좋은 모델을 만들기 위해서는 컴퓨터 비전의 기본과도 같은 rule-based 수학적 이론들을 깊게 이해하는 것 또한 중요하다고 생각하며 과제를 마무리 하였다.