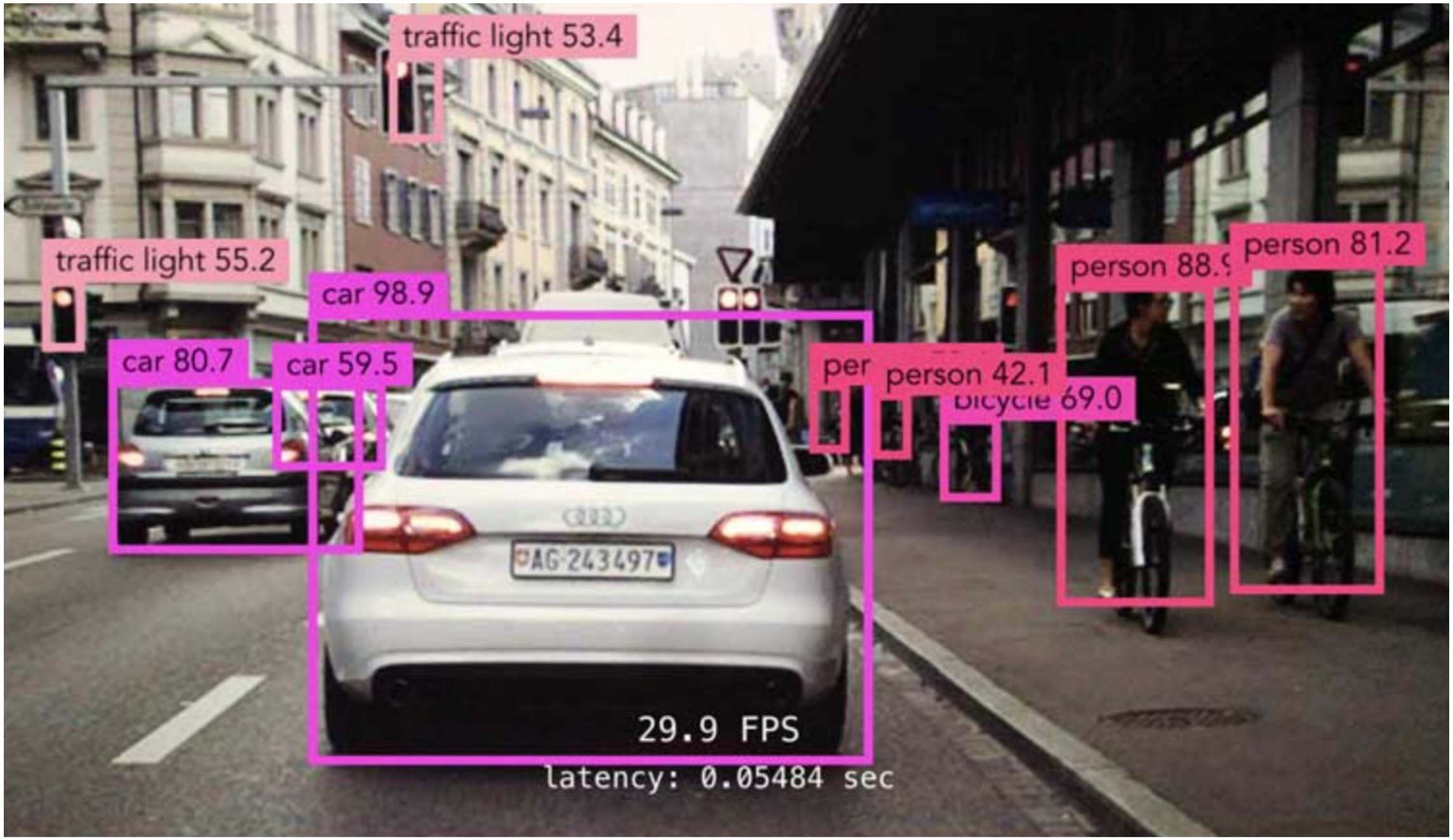
객체 검출

Detect, Segment, Pose

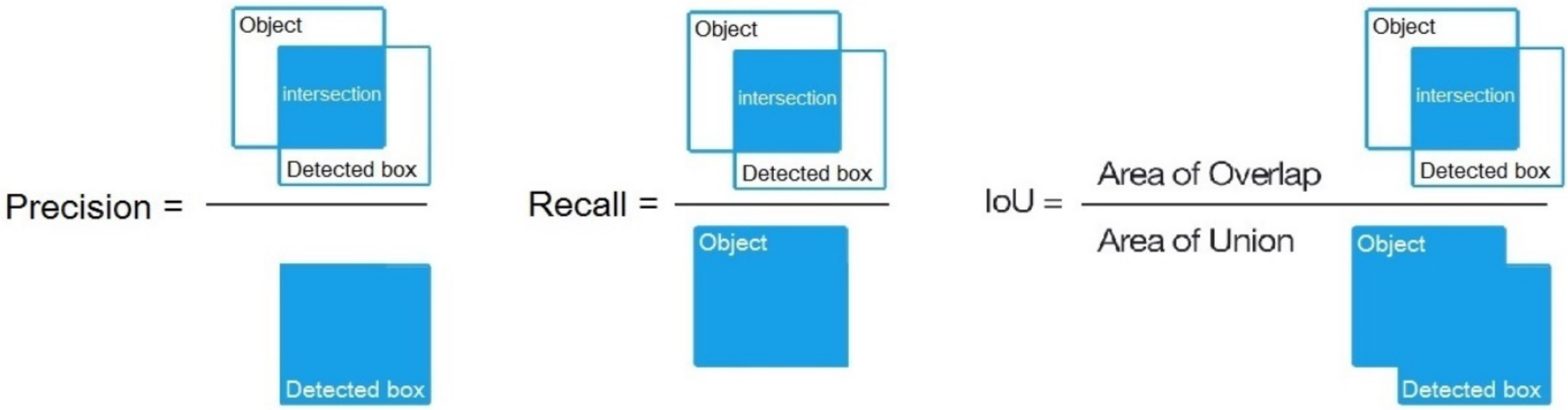
Object Detection

- Confidence score



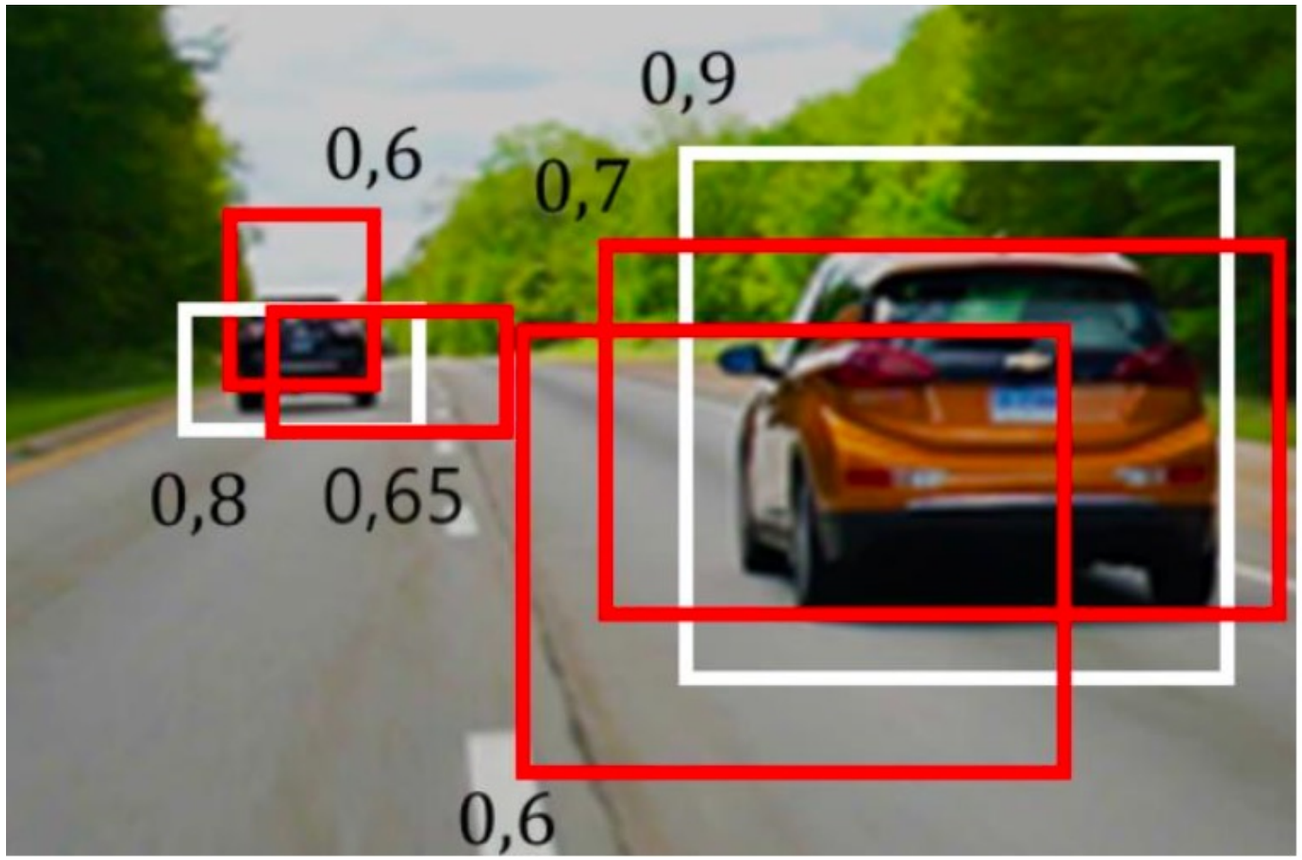
성능평가 - IOU

- Intersection over Union
- 참고: 정밀도(precision), 리콜(recall)



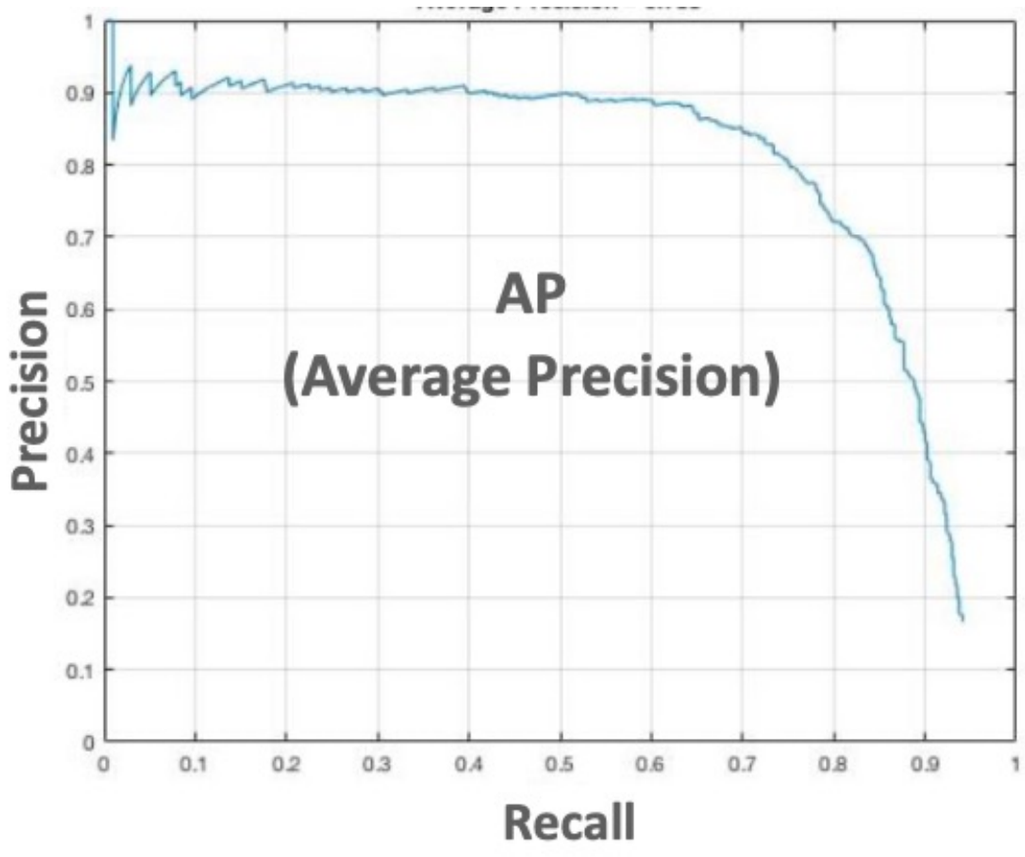
중복 객체 제거

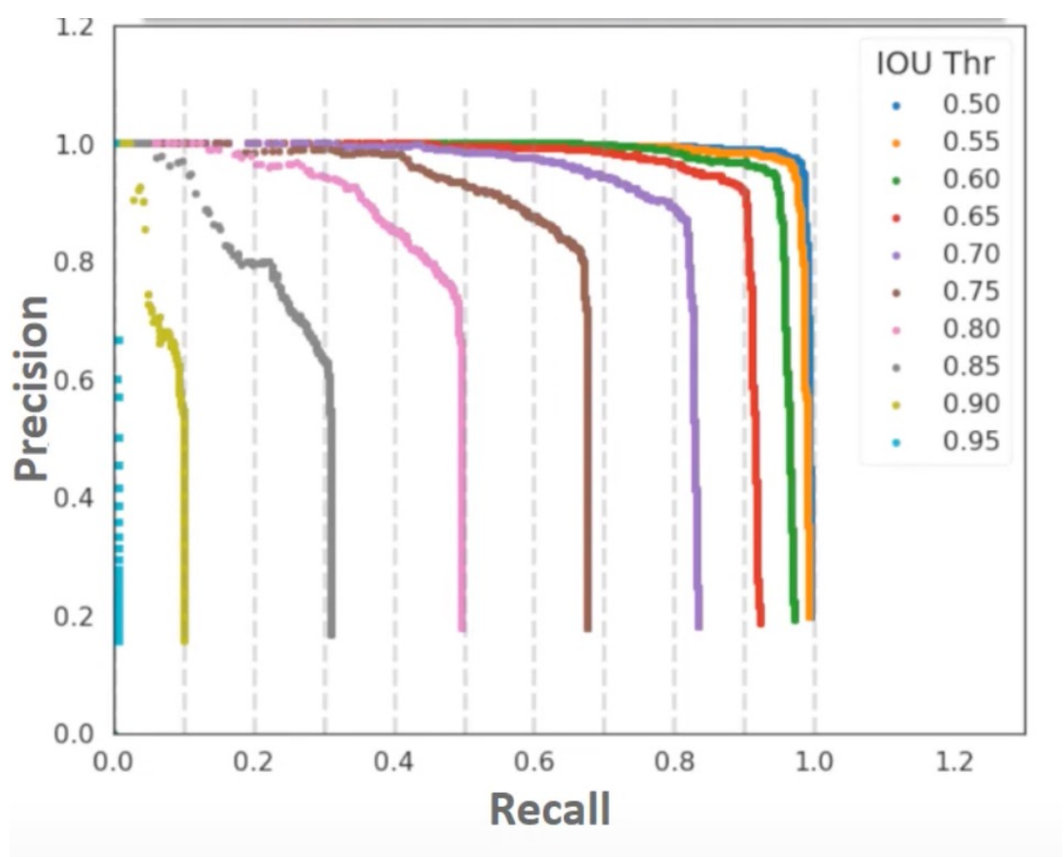
- NMS(Non Max Suppression)
  - 특정 Confidence threshold 이하 bounding box를 먼저 제거
  - 높은 confidence score를 가진 box와 겹치는 다른 box를 모두 조사하여 IOU가 특정 threshold 이상인 box를 모두 제거(1등만 남김)



성능평가 - mAP

- mAp(mean Average Precision)
  - 여러 클래스들의 AP를 평균한 값 블로그
- PASCAL VOC (Visual Object Classes Challenges)
  - 리콜값이 0, 0.1, 0.2, … 1.0까지 11개의 mAP를 구함
- Coco
  - IOU 값이 0.5, 0.55, 0.6, …, 0.95 경우의 mAP를 구함





YOLO v8

- 객체 검출과 객체 세분화를 쉽게 구현
- 사용 절차
  - 데이터 준비: 이미지와 레이블링(annotation)
  - 데이터셋 다운로드: Roboflow 등 사용
  - yaml 파일 생성 (데이터 저장 폴더 정보 등을 설정하는 파일)
  - 모델 설치: ultralytics에서 YOLO 설치
  - model = YOLO("….pt") # OD 또는 seg 선택
  - model.train(data = "…yaml")
  - model.predict(source='…jpg')
    - runs/detect/predict 폴더에 생성

YOLOv8 detect 성능

Model	size (pixels)	mAP^{val} 50-95	Speed CPU ONNX (ms)	Speed A100 TensorRT (ms)	params (M)	FLOPs (B)
YOLOv8n	640	37.3	80.4	0.99	3.2	8.7
YOLOv8s	640	44.9	128.4	1.20	11.2	28.6
YOLOv8m	640	50.2	234.7	1.83	25.9	78.9
YOLOv8l	640	52.9	375.2	2.39	43.7	165.2
YOLOv8x	640	53.9	479.1	3.53	68.2	257.8

- **mAP^{val}** values are for single-model single-scale on COCO val2017 dataset.
  Reproduce by `yolo val detect data=coco.yaml device=0`
- **Speed** averaged over COCO val images using an Amazon EC2 P4d instance.
  Reproduce by `yolo val detect data=coco128.yaml batch=1 device=0|cpu`

Segment

Model	size (pixels)	mAP^{box} 50-95	mAP^{mask} 50-95	Speed CPU ONNX (ms)	Speed A100 TensorRT (ms)	params (M)	FLOPs (B)
YOLOv8n-seg	640	36.7	30.5	96.1	1.21	3.4	12.6
YOLOv8s-seg	640	44.6	36.8	155.7	1.47	11.8	42.6
YOLOv8m-seg	640	49.9	40.8	317.0	2.18	27.3	110.2
YOLOv8l-seg	640	52.3	42.6	572.4	2.79	46.0	220.5
YOLOv8x-seg	640	53.4	43.4	712.1	4.02	71.8	344.1

- **mAP^{val}** values are for single-model single-scale on COCO val2017 dataset.
  Reproduce by `yolo val segment data=coco.yaml device=0`
- **Speed** averaged over COCO val images using an Amazon EC2 P4d instance.
  Reproduce by `yolo val segment data=coco128-seg.yaml batch=1 device=0|cpu`

이미지와 레이블 데이터셋

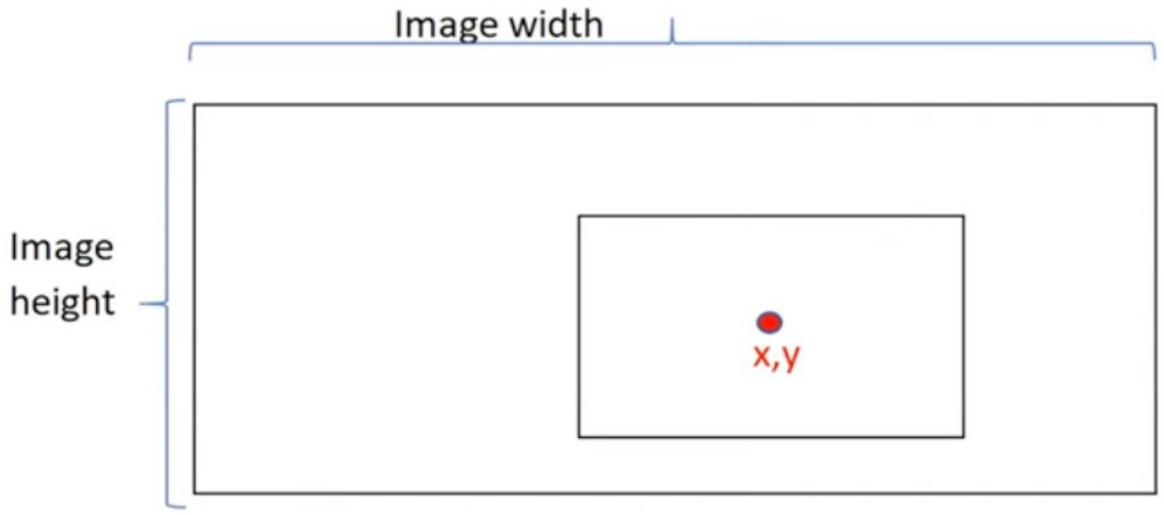
- 이미지와 annotation 파일명(txt)이 이미지 별로 동일해야 한다
  - imge7.jpg → image7.txt (아래는 3개의 객체가 들어 있다)
  - image7.txt 의 첫번째 행 bbox(bounding box) : (0.48, 0.63, 0.69, 0.71)

Dataset

- COCO(Common Objects in Context):
  - 가장 널리 사용되는 데이터 세트
  - 200,000개 이상의 이미지가 포함
  - 사람, 자동차, 개 등 80개의 클래스를 포함
  - Bounding Box, 인스턴스 분할 마스크, keypoints 등 제공

- Object segmentation
- Recognition in context
- Superpixel stuff segmentation
- 330K images (>200K labeled)
- 1.5 million object instances
- 80 object categories
- 91 stuff categories
- 5 captions per image
- 250,000 people with keypoints

YOLO와 COCO format

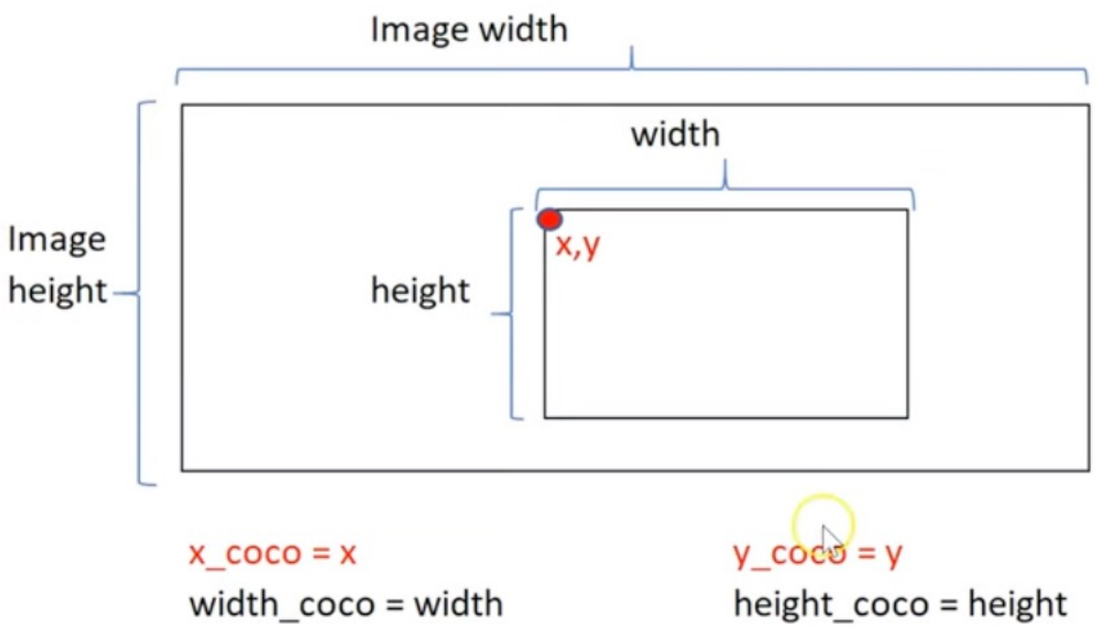


x_yolo = x/Image width y_yolo = y/Image height

width_yolo = width/Image width height_yolo = height/Image height

YOLO format bounding box = <x_yolo, y_yolo, width_yolo, height_yolo>

```
1 0.711719 0.302083 0.418750 0.334722
1 0.730469 0.807639 0.431250 0.315278
1 0.262109 0.715278 0.477344 0.238889
1 0.237500 0.254861 0.332812 0.262500
```



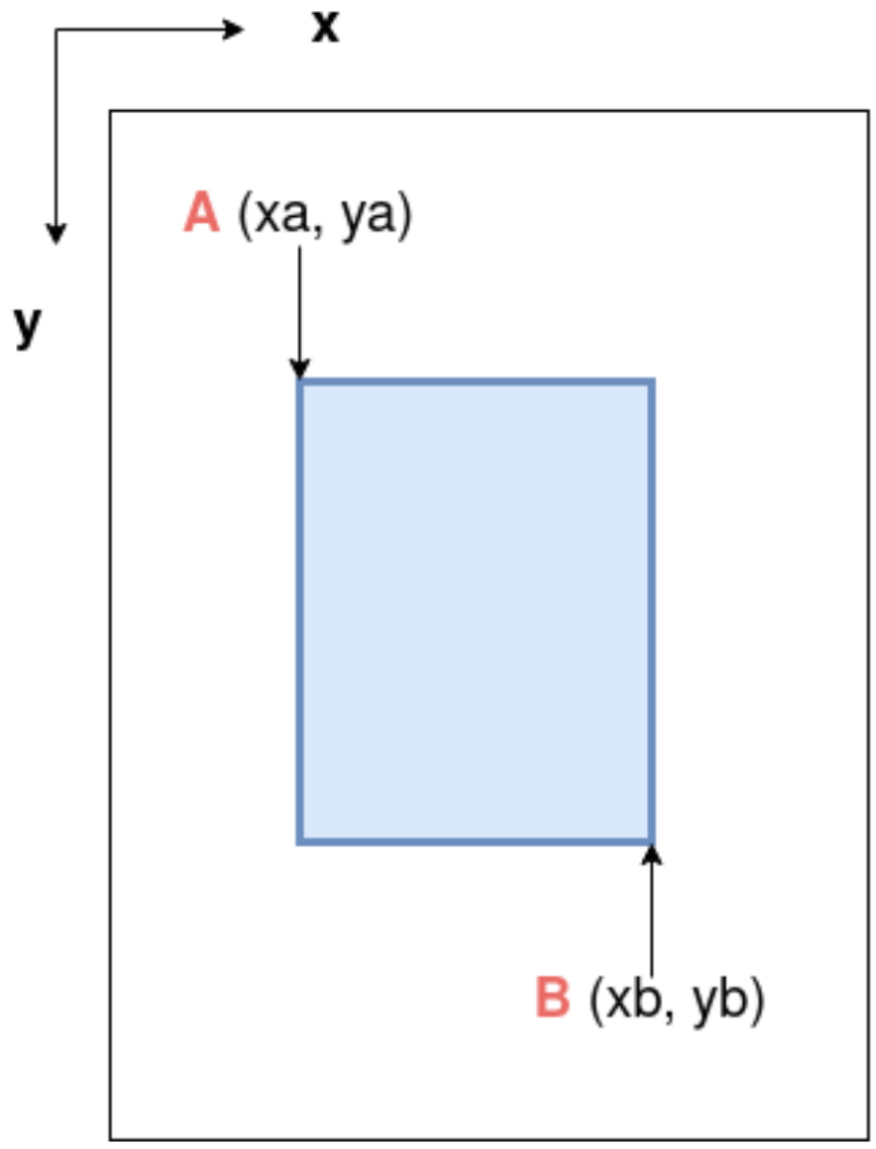
x_coco = x y_coco = y

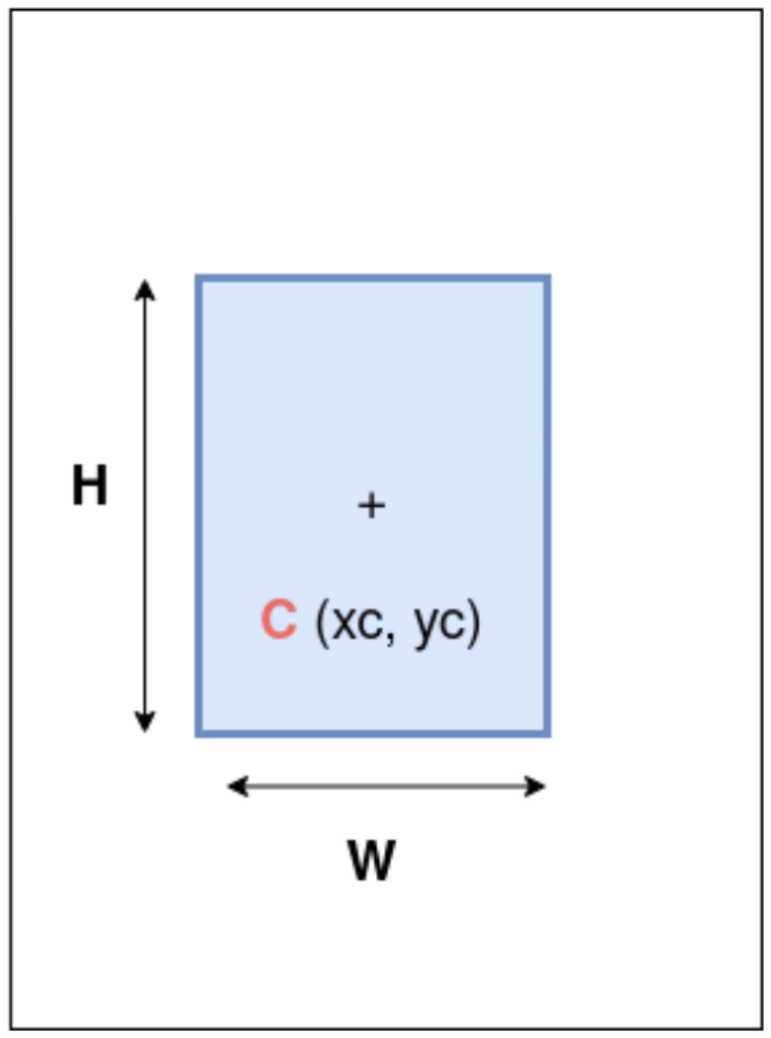
width_coco = width height_coco = height

COCO format bounding box = <x_coco, y_coco, width_coco, height_coco


```
"annotations": [
  {
    "id": 1,
    "image_id": 1,
    "category_id": 2,
    "iscrowd": 0,
    "area": 35041.209375000006,
    "bbox": [
      251,
      67,
      209.375,
      167.36100000000002
    ],
    "segmentation": [
      [
        251,
        67,
        460.375,
        67,
        460.375,
        234.36100000000002,
        251,
        234.36100000000002
      ]
    ]
  },
```

xyxy, xywh 포맷





데이터셋 준비

- Roboflow 이용
- 개인 데이터셋 업로드 후 레이블링
- 제공되는 공개 데이터셋 사용

Labelimg

- 개인 PC에서 레이블링 작업을 하는 도구
- 블로그
  - https://yeko90.tistory.com/entry/free-image-label-tool-labelimg
- git clone https://github.com/HumanSignal/labelImg.git
  - (맥에서 설치 필요) xcode-select –install
- 실행
  - python labelimg.py
  - 저장 포맷을 yolo로 설정해야 한다 (txt로 저장된다)
  - 레이블링 표현에는 xml, json, txt 등의 포맷이 있다

yaml 파일

- 객체 이름, 클래스수 (nc), 데이터가 있는 폴더명을 지정
- PyYAML을 사용하여 파이썬으로 설정
  - 딕셔너리로 설정할 내용을 지정

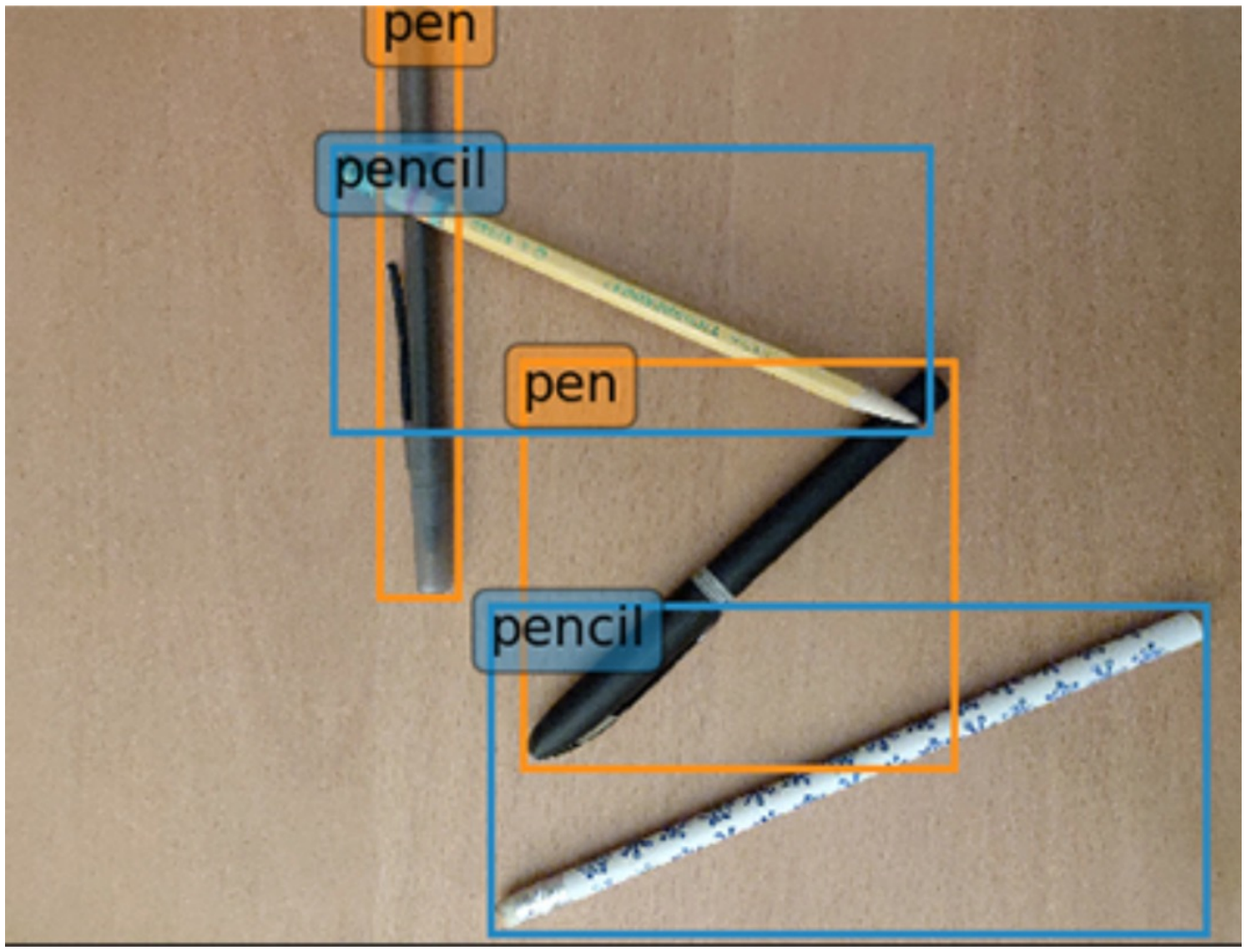
```
{'names': ['fish',
  'jellyfish',
  'penguin',
  'puffin',
  'shark',
  'starfish',
  'stingray'],
'nc': 7,
'test': '/content/Aquarium_Data/test/images',
'train': '/content/Aquarium_Data/train/images/',
'val': '/content/Aquarium_Data/valid/images/'}
```

predict 옵션

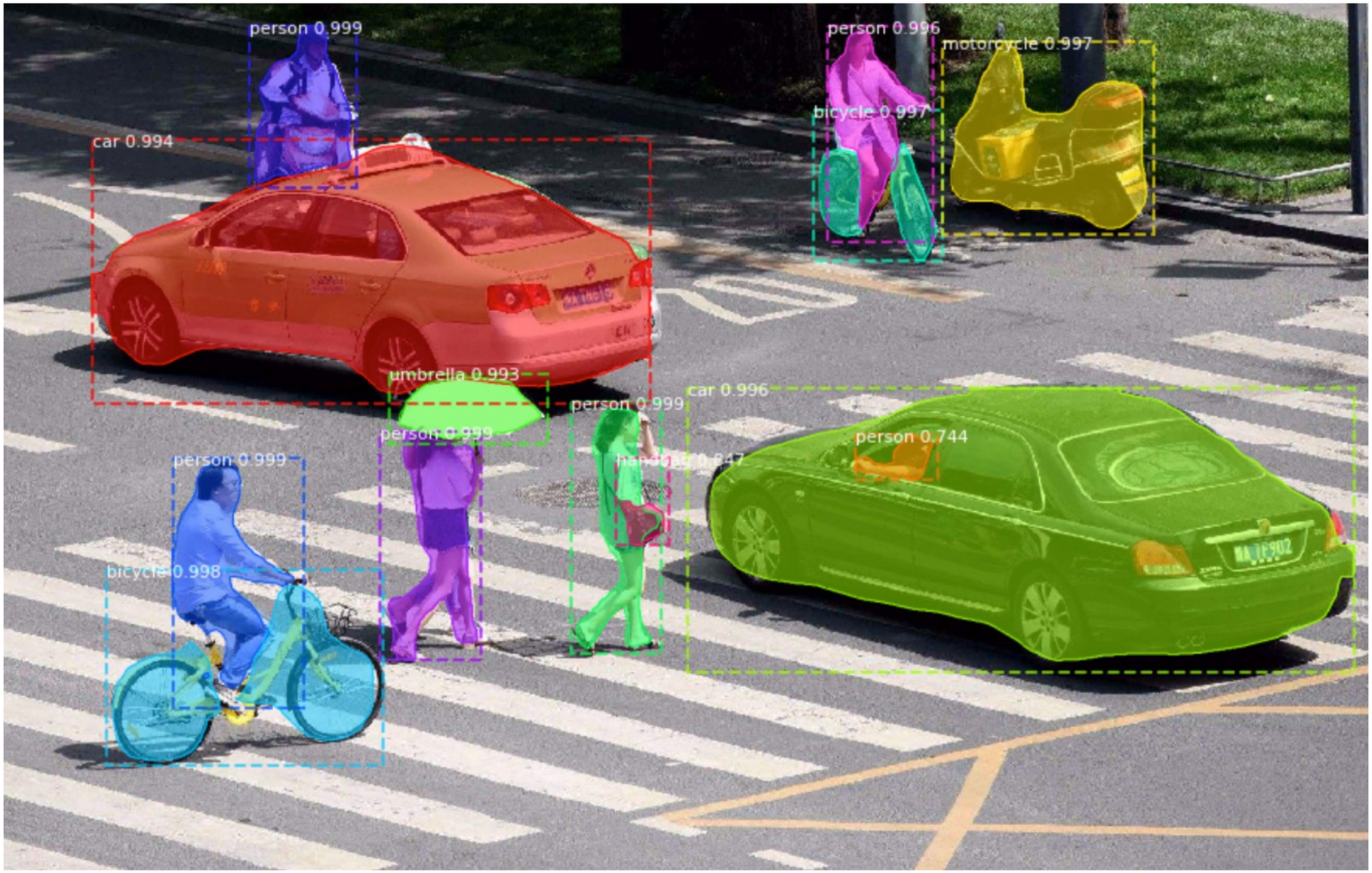
Name	Type	Default	Description
`source`	`str`	`'ultralytics/assets'`	source directory for images or videos
`conf`	`float`	`0.25`	object confidence threshold for detection
`iou`	`float`	`0.7`	intersection over union (IoU) threshold for NMS
`imgsz`	`int or tuple`	`640`	image size as scalar or (h, w) list, i.e. (640, 480)
`half`	`bool`	`False`	use half precision (FP16)
`device`	`None or str`	`None`	device to run on, i.e. cuda device=0/1/2/3 or device=cpu
`show`	`bool`	`False`	show results if possible
`save`	`bool`	`False`	save images with results
`save_txt`	`bool`	`False`	save results as .txt file
`save_conf`	`bool`	`False`	save results with confidence scores
`save_crop`	`bool`	`False`	save cropped images with results

Labeling

- 모든 객체를 레이블링
- 꽉 차게
- 클래스는 가능한 세분한 후 나중에 합친다
- 이미지 크기와 픽셀 고려



Object Segmentation



Object Segmentation