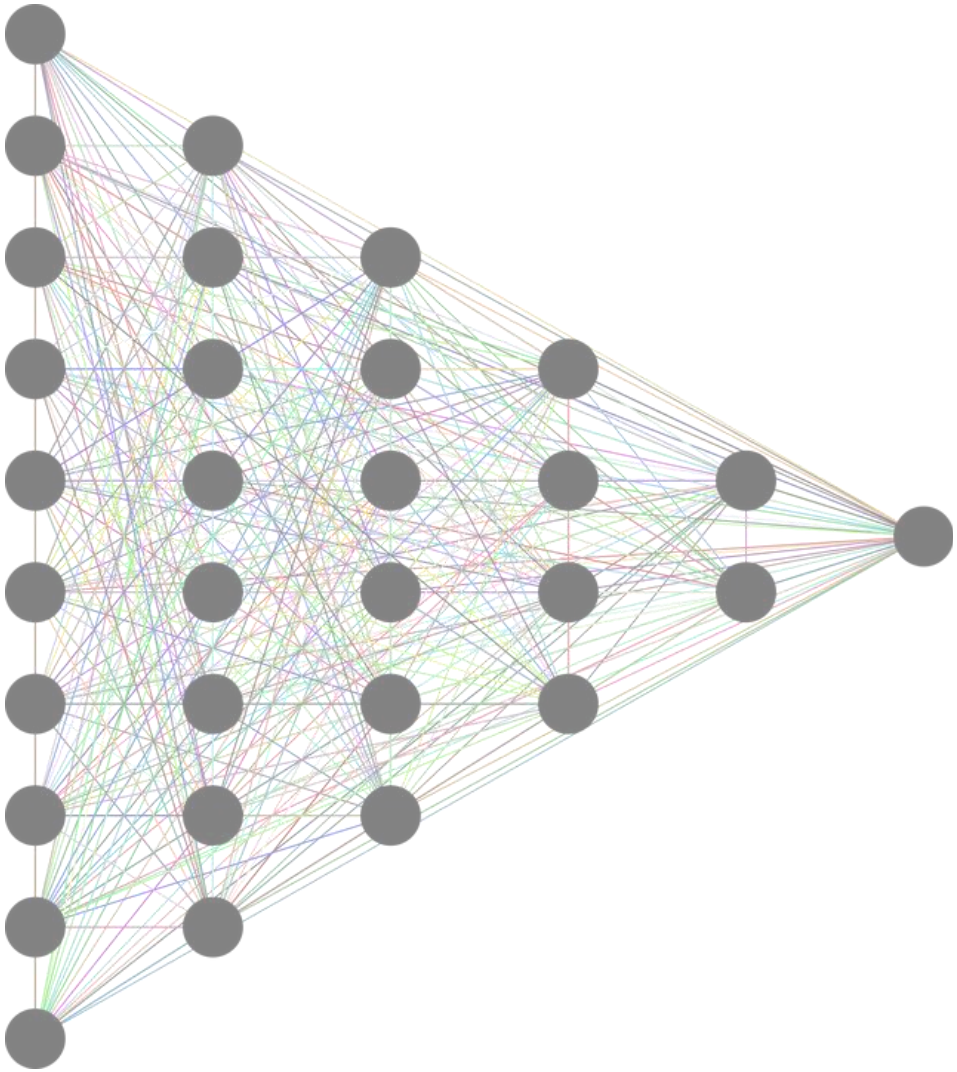
Deep Learning

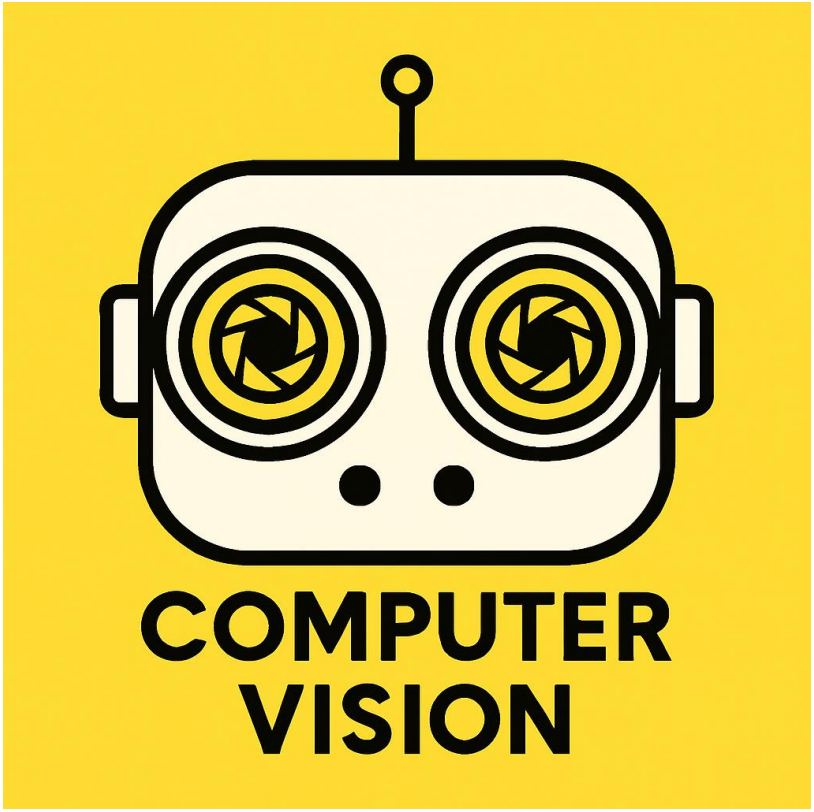
2025. 5. 28

박 준 오

2. Deep Learning



2. Deep Learning



2. Deep Learning

1. CNN (Convolutional Neural Network)

– 기본 기반 기술

- 등장 배경:이미지 분류 문제를 해결하기 위한 기본 구조
- 고정된 위치에 있는 객체를 분류만 할 수 있음

– 기술적 특이점:

정확한 위치(localization) 불가능

전체 이미지를 보고 "고양이다"는 말은 가능하지만, "어디에 고양이가 있다"는 건 못함

기본 이미지 분류에만 사용됨

– 객체 탐지의 시작점, 하지만 "어디 있는지"는 모름

2. Deep Learning

2. R-CNN (Regions with CNN features, 2014)

– 등장 배경:

- "분류"만 하던 CNN에 위치 정보(객체 박스)를 붙이고 싶다
- Selective Search를 이용해 후보영역(region proposals)을 만든 뒤, 각 영역마다 CNN을 적용

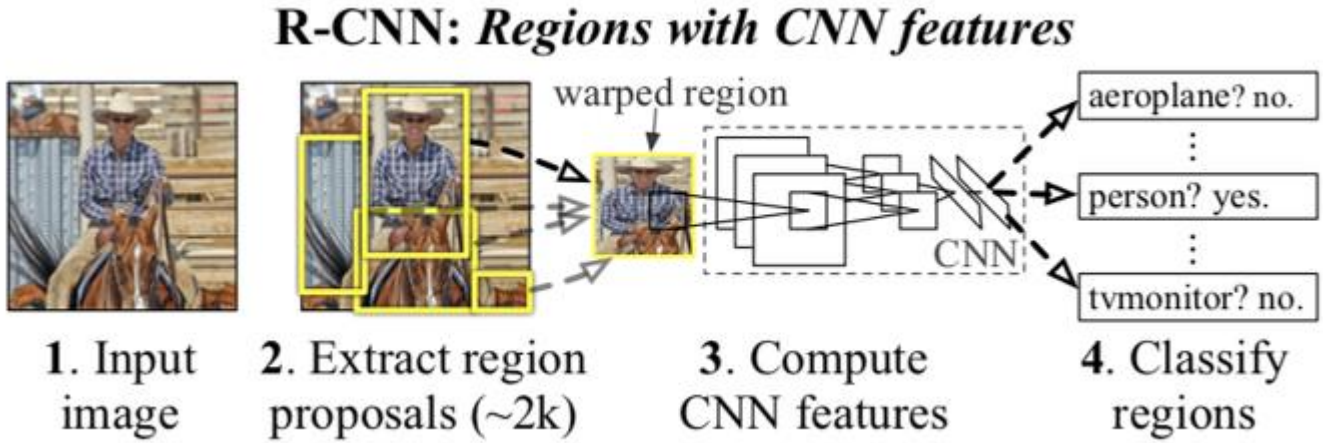
– 기술적 특이점:

- 두 단계 방식 (Two-stage Detection)
- 후보 박스 생성 (Selective Search)
- CNN으로 각 후보 영역 분류
- 정확도는 높지만 속도가 엄청 느림 (한 이미지당 수천 번 CNN 실행)

– CNN으로 객체 위치까지 찾기 시작했지만 너무 느렸음

https://velog.io/@cha-suyeon/%EB%94%A5%EB%9F%AC%EB%8B%9D-Object-Detection-Architecture-1-stage-detector%EC%99%80-2-stage-detector

2. Deep Learning (객체탐지) two stage detector



R-CNN은 object가 있을 법한 후보 영역을 뽑아내는 "Region proposal" 알고리즘과 후보 영역을 분류하는 CNN을 사용,
예시)약 2,000개에 달하는 후보 이미지 각각에 대해서 convolution 연산을 수행

Proposal을 만들어내는 데에는 Selective search라는 비 신경망 알고리즘이 사용됩니다. 이후에 후보 영역의 Classification과 Bounding Box의 regression을 위해 신경망을 사용

2. Deep Learning

3. Fast R-CNN (2015)

– 등장 배경:

- R-CNN의 느린 속도와 중복된 CNN 연산 문제 해결 목적

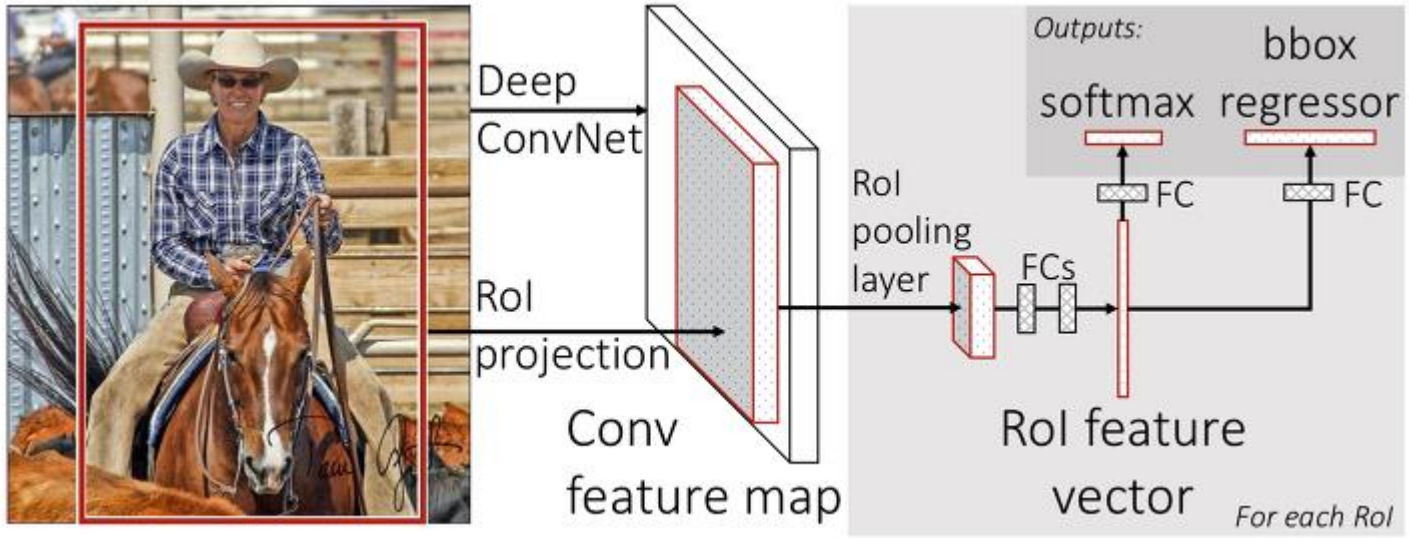
– 기술적 특이점:

- 전체 이미지에 CNN을 한 번만 적용
- 그 후, ROI(Region of Interest) Pooling을 통해 후보영역 추출
- 단일 CNN feature map을 재활용 → 속도 향상
- 학습도 end-to-end로 가능

–이미지 한 번만 보고, 후보만 쪼개서 분류하자

2. Deep Learning (객체탐지) two stage detector

Fast R-CNN

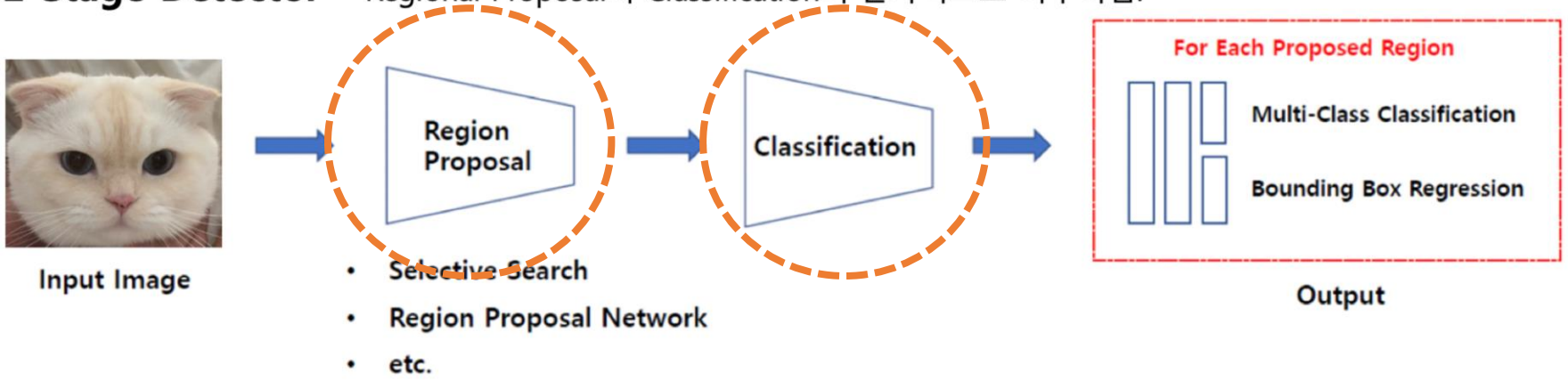


R-CNN의 경우, region proposal을 selective search로 수행한 뒤 많은 수의 후보 이미지 각각에 대해서 convolution 연산을 수행. 이 경우 한 이미지에서 feature을 반복해서 추출하기 때문에 비효율적이고 느리다는 단점

Fast R-CNN에서는 입력 이미지를 한 번만 CNN에 통과시키고, 객체 탐지를 위한 후보 영역 추출과 특징 추출을 분리하여 수행. 이로써 앵커 박스와 특징 맵의 일부 영역을 공유하므로, RCNN보다 효율적인 속도와 좋은 탐지 성능을 제공

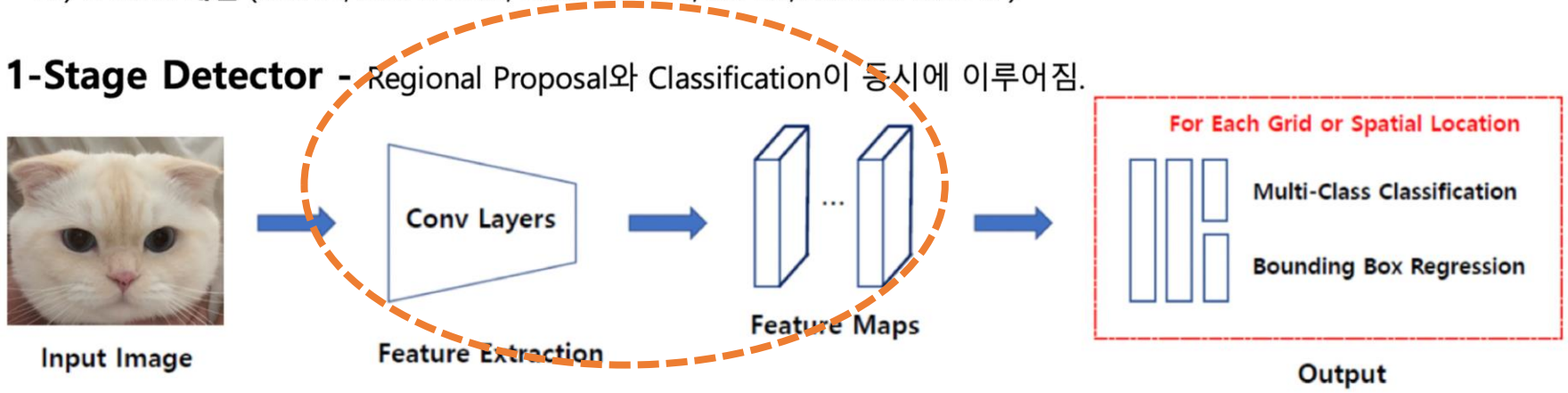
2. Deep Learning

2-Stage Detector - Regional Proposal와 Classification이 순차적으로 이루어짐.



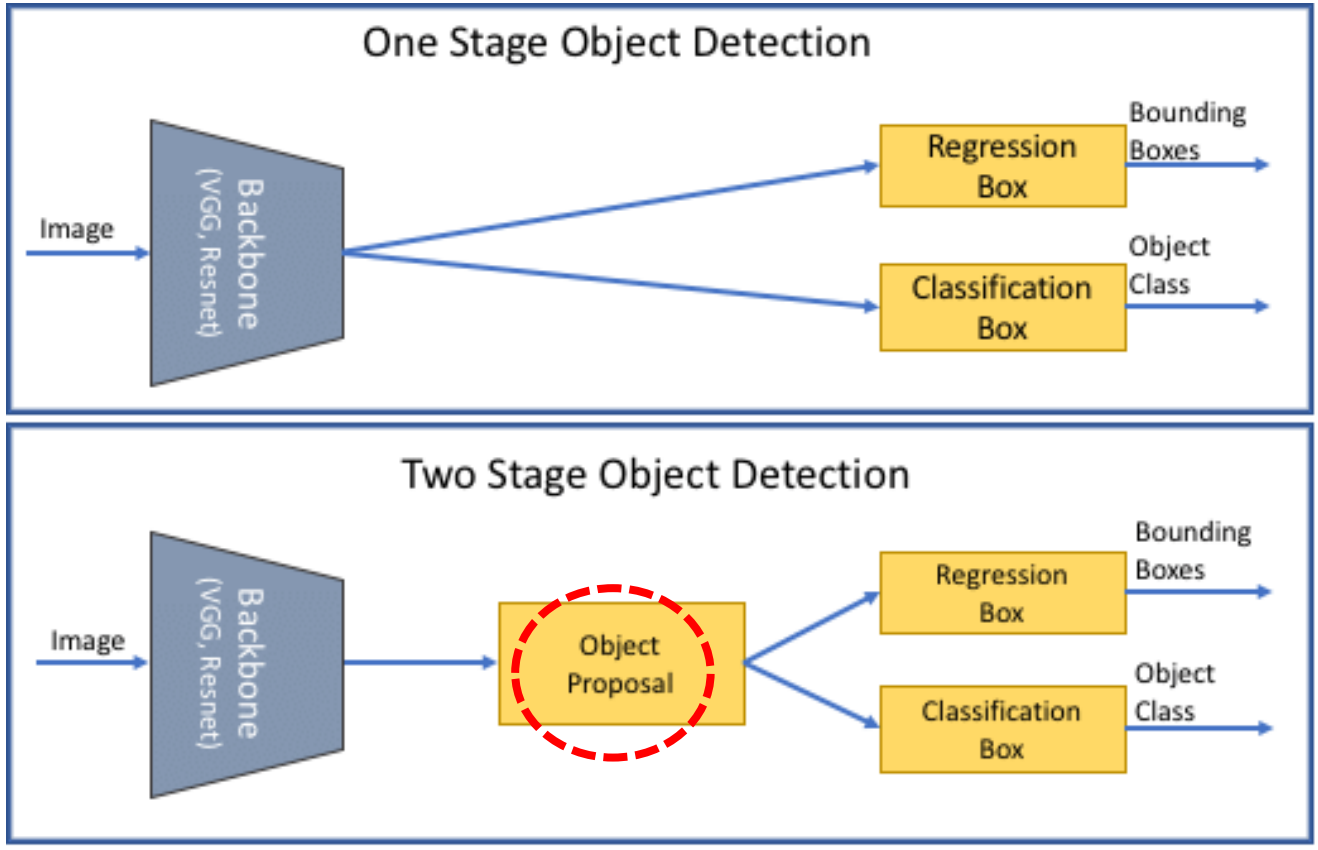
Ex) **R-CNN 계열** (R-CNN, Fast R-CNN, Faster R-CNN, R-FCN, Mask R-CNN ...)

1-Stage Detector - Regional Proposal와 Classification이 동시에 이루어짐.



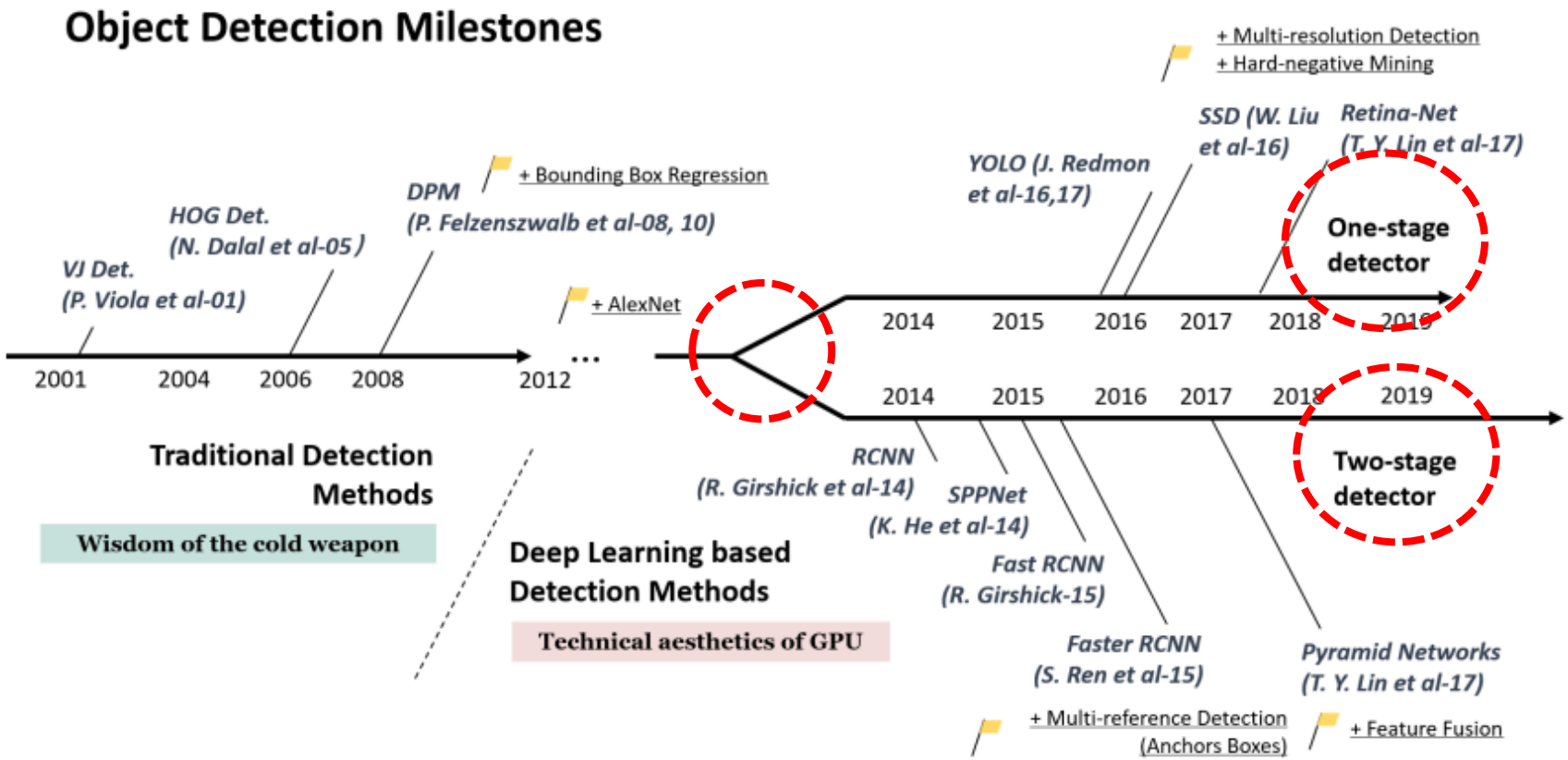
Ex) **YOLO 계열** (YOLO v1, v2, v3) , **SSD 계열** (SSD, DSSD, DSOD, RetinaNet, RefineDet ...)

2. Deep Learning (객체탐지)



One stage Detector : 객체의 검출과 분류, 그리고 바운딩 박스 regression을 한 번에
Two stage Detector : object 위치 후보(Region proposals) 들을 뽑아내는 단계,
객체 검출, 이후 object가 있는지를 Classification과
정확한 바운딩 박스를 구하는 Regression을 수행하는 단계.

2. Deep Learning (객체탐지)



One stage Detector : 객체의 검출과 분류, 그리고 바운딩 박스 regression을 한 번에

Two stage Detector : object 위치 후보(Region proposals) 들을 뽑아내는 단계,
객체 검출, 이후 object가 있는지를 Classification과
정확한 바운딩 박스를 구하는 Regression을 수행하는 단계.

2. Deep Learning

4. Faster R-CNN (2015)

– 등장 배경:

- Fast R-CNN은 여전히 Selective Search에 의존 (느림)
- 그래서 후보영역 생성조차도 신경망으로 대체하자

– 기술적 특이점:

- Region Proposal Network (RPN) 도입
- RPN이 직접 anchor 기반으로 후보 영역을 예측
- 완전한 end-to-end 학습 가능
- 정확도와 속도 모두 뛰어남

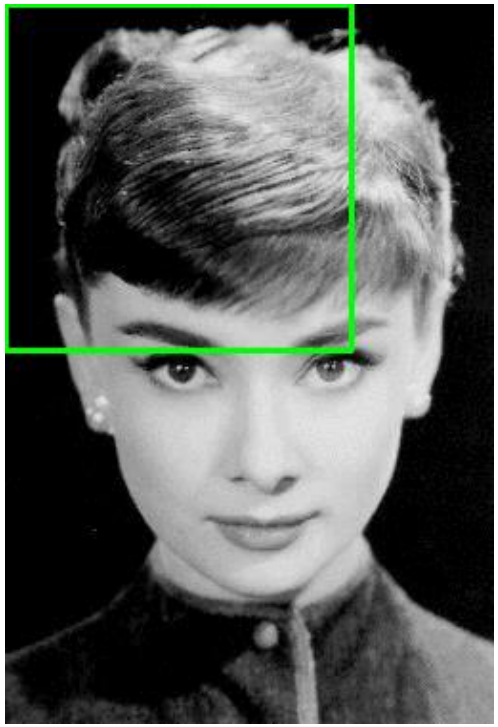
– "후보 영역도 CNN으로 만들자" → 진짜 완성된 객체 탐지 모델

2. Deep Learning

– **Region Proposal Network (RPN) 이란?**

CNN feature map 위에 슬라이딩 윈도우를 돌리면서
Anchor Box라는 다양한 크기/비율의 박스를 겹쳐놓고

각각의 박스가 물체가 있는지 없는지, 좌표는 어디인지 예측
즉, 후보영역 뽑기마저 CNN으로 직접 처리!



2. Deep Learning

5. YOLO (You Only Look Once, 2016)

– 등장 배경:

- R-CNN 계열은 정확하지만 실시간에 부적합 (2-stage 구조)
- 객체 탐지를 한 번에 끝내고 싶다

– 기술적 특이점:

- 이미지를 grid로 분할 후, 각 셀에서 객체 존재 여부 + 위치 + 클래스 예측
- 단일 CNN만으로 전체 처리
- 실시간 처리 가능속도는 훌륭하지만, 초기 버전(YOLOv1)은 작은 객체 탐지에 약했음

–딱 한 번 본다

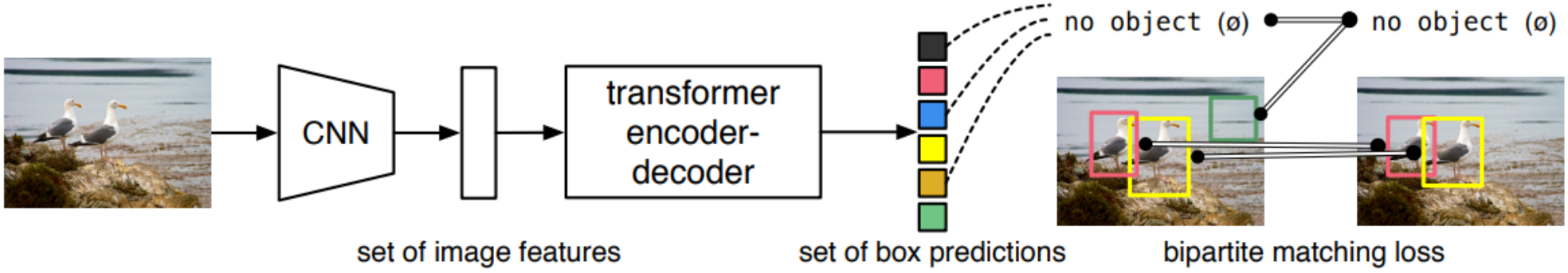
– 빠르고 직관적인 객체 탐지기

2. Deep Learning

6. DETR (DEtection TRansformer)

2020년 Facebook AI 발표, "Transformer를 이용해 객체 탐지를 End-to-End로 처리하는 모델“

등장 배경
기존 객체 탐지 알고리즘들은 거의 다 CNN + 복잡한 후처리(Anchor, NMS) 구조였음
anchor 설정, region proposal, NMS 등의 "핸드메이드 규칙"이 너무 많았음
"이걸 다 없애고, 순수하게 딥러닝(Transformer)으로만 처리할 수 없을까?”



2. Deep Learning

– DETR (DEtection TRansformer)

기술적 특이점

1. Object Detection을 "Sequence Prediction"으로 봄
기존 탐지: “여기 뭔가 있어!” 박스를 예측
DETR: “이 이미지는 N개의 객체가 있고, 각각의 class와 위치는 이거야” 라고 문장처럼 예측

2. Transformer 구조 사용 (Encoder-Decoder)
Encoder: 이미지 → CNN으로 feature map 추출 후 positional encoding 추가
Decoder: 객체를 나타내는 learnable queries (예: 100개) 를 넣어 Transformer가 어떤 객체들이 있는지 예측

3. No Anchors, No NMS, No Region Proposal, NMS 전부 제거됨!
대신 예측한 box들과 실제 box를 Hungarian Matching 알고리즘으로 매칭해서 학습
이 덕분에 완전히 end-to-end 학습 가능함

2. Deep Learning

7. Vision Transformer (ViT) : 이미지 분류(Image Classification)

자연어 처리(NLP)에서 성공을 거둔 Transformer 구조를 컴퓨터 비전 분야에 적용하여,
CNN 없이도 이미지 분류에서 높은 성능을 달성하고자 함

구조 및 특징:
입력 처리: 이미지를 고정 크기의 패치로 분할하고, 각 패치를 시퀀스로 변환
포지셔널 인코딩: 패치의 순서 정보를 보존하기 위해 위치 정보를 추가
Transformer 인코더: Self-Attention을 통해 패치 간의 관계를 학습
분류 헤드: 특별한 [CLS] 토큰을 통해 전체 이미지의 표현을 얻고, 이를 분류기로 사용

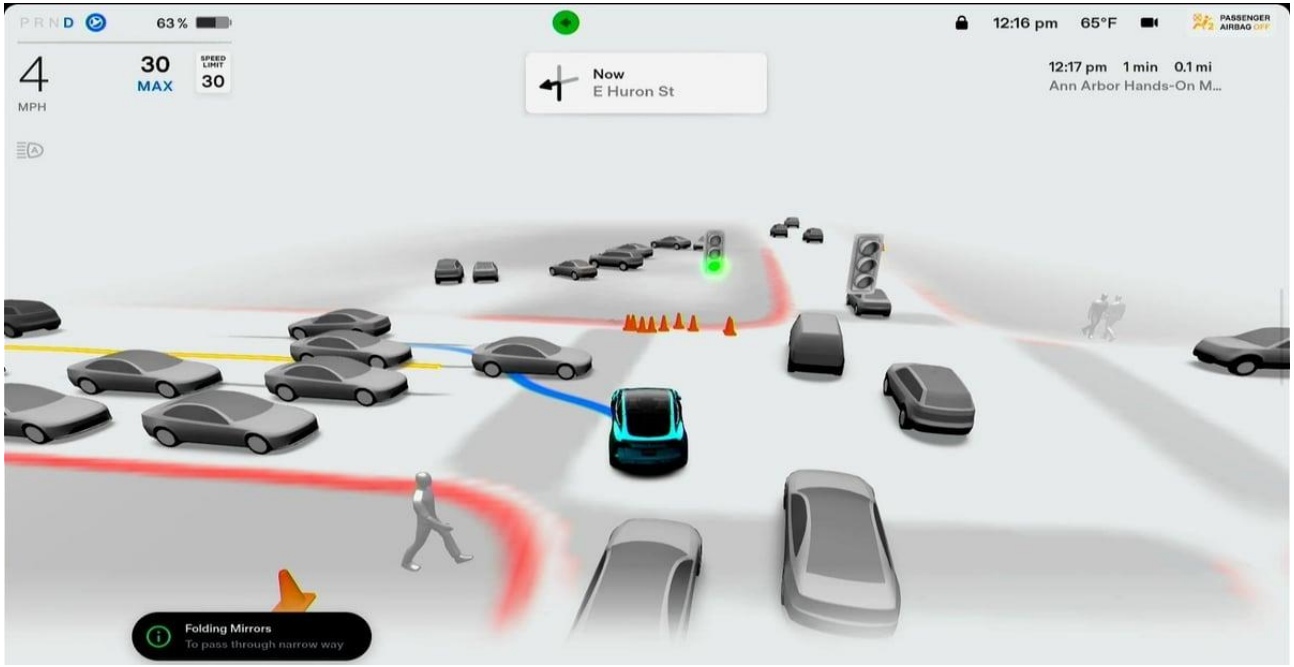
장점:
전역적인 정보 처리 능력: Self-Attention을 통해 이미지 전체의 문맥을 고려한 특징 추출 가능
확장성: 대규모 데이터셋에서의 학습을 통해 높은 성능 달성

단점:
데이터 효율성 부족: CNN에 비해 적은 데이터로는 성능이 저하될 수 있음
로컬 정보 처리의 한계: CNN이 갖는 지역적인 특성을 활용하지 못함

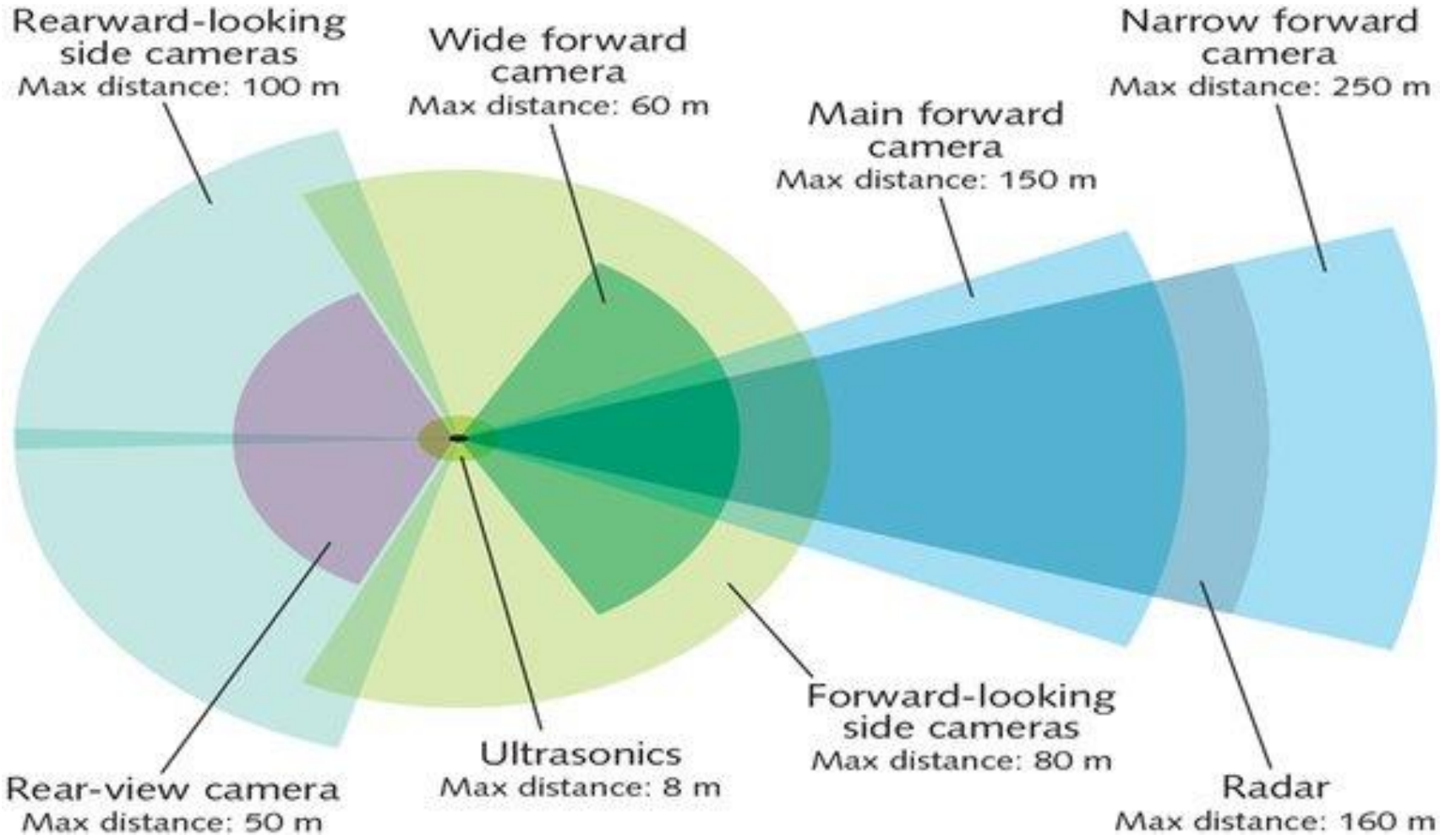
2. Deep Learning

2. Deep Learning

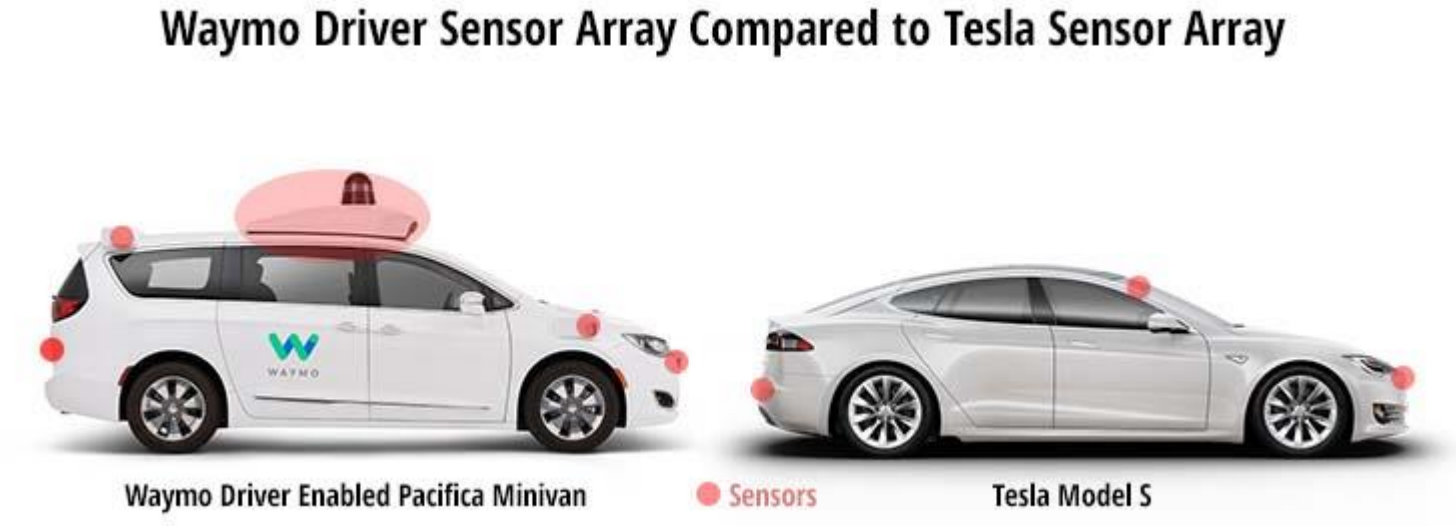
Full Self Driving



2. Deep Learning



2. Deep Learning



2. Deep Learning

Vision기반의 자율주행의 한계

Lida/Rader sensing(물리적 감지) vs Vision Sensing(신경망을 통한 감지)

점유네트워크로 개선한 비젼 기반 자율 주행 문제점

- 조감도(Bird eye view) : 2D 조감도를 3D 조감도로 업데이트

- 고정 직사각형(fixed recangle) : voxel(volume+pixel)로 나눈 이미지로 디테일 확장
  불규칙한 모양과 돌출된 부분도 감지

- 객체 감지(Object detection) : 학습된 객체만 인식하는 한계를 극복
  객체의 인식보다 물체의 존재 여부에 집중

움직이는 객체와 않움직이는 객체를 구분하여 메모리 효율성도 향상

2. Deep Learning

Occupancy Network : 점유 그리드 매핑, 탐지보다는 점유여부 체크

차량 주변 환경을 3D 격자 형태로 표현하고, 각 격자에 물체가 존재하는지 여부를 예측하는 딥러닝 모델, 실시간으로 차량 주변의 장애물과 공간을 인식, 멀티뷰 사용가능

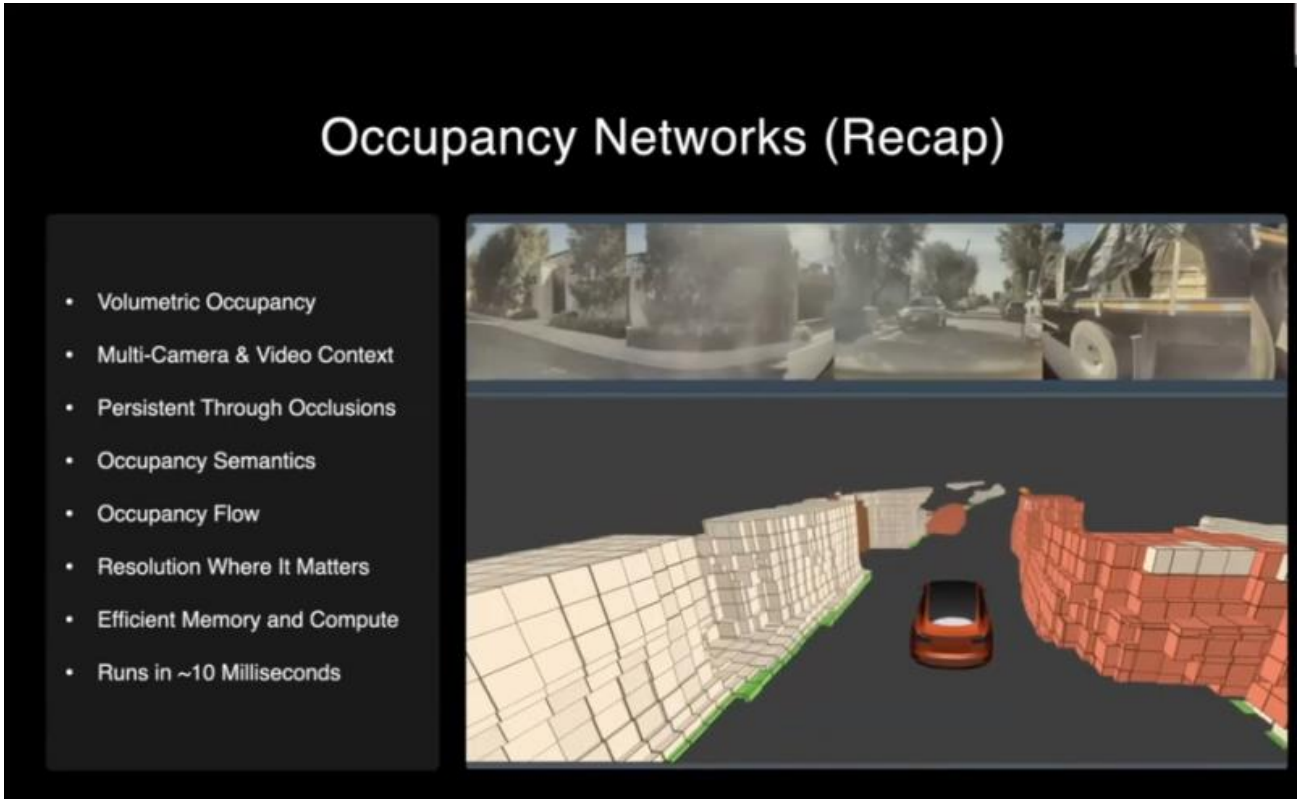
Occupancy Network의 작동 방식

1.데이터(라이다, 레이더, 카메라 등)를 입력
2.데이터를 기반으로 차량 주변 환경을 3D 격자로 나눕니다.
3.각 격자에 대해 물체의 존재 여부를 0(빈 공간) 또는 1(물체 존재)로 예측
4.예측된 3D 격자 데이터를 활용하여 주행 경로를 계획하고 장애물을 회피

예를 들어:차량 전방에 다른 차량과 보행자가 있고, 우측에는 가로수

이 장면을 3D 격자로 표현하고, 각 격자에서 물체 존재 여부를 예측
1(물체 존재)로 예측되고, 그 외 격자는 0(빈 공간)으로 예측
자율주행 시스템은 전방 차량과 보행자, 가로수를 인식하고 이를 회피 경로를 계획
실시간으로 복잡한 환경을 정확히 인식,
자율주행/로봇 내비게이션/증강현실 등 다양한 분야에서 활용

2. Deep Learning



Occupancy Flow : 2D 이미지를 3D로 변환하여 움직이는 물체와 움직이지 않는 물체의 구분
(vs. 다른 차량 등을 통해 작성된 3D지도: NeRFs),
실시간 비교 분석 > 주행 차량과의 주행관계를 인식

Occupancy Volume : 물체가 차지하는 부피를 산출

2. Deep Learning

날씨가 흐리거나 어두운 밤 등의 영향으로 입력 이미지가 좋지 않은 경우 :
차량의 여러 이미지, 여러 차량의 동일 이미지를 고려하여
평균값을 산출/예측하여 활용

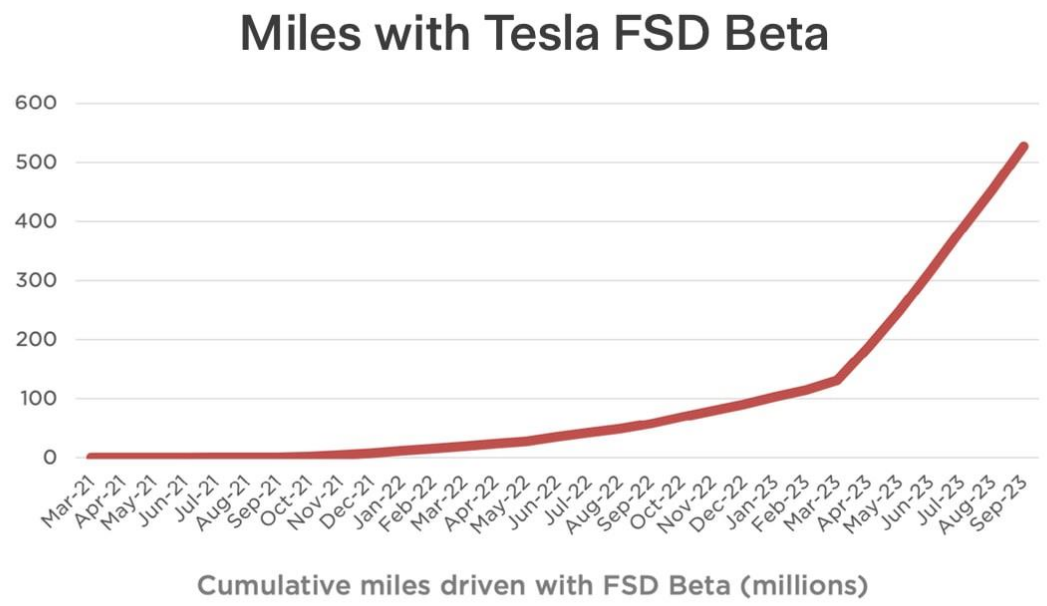
재구성한 이미지 지도 vs 점유네트워크로 생성된 실제상황(현실 이미지)

차이를 비교/분석하여 FSD(Autopilot)에 활용

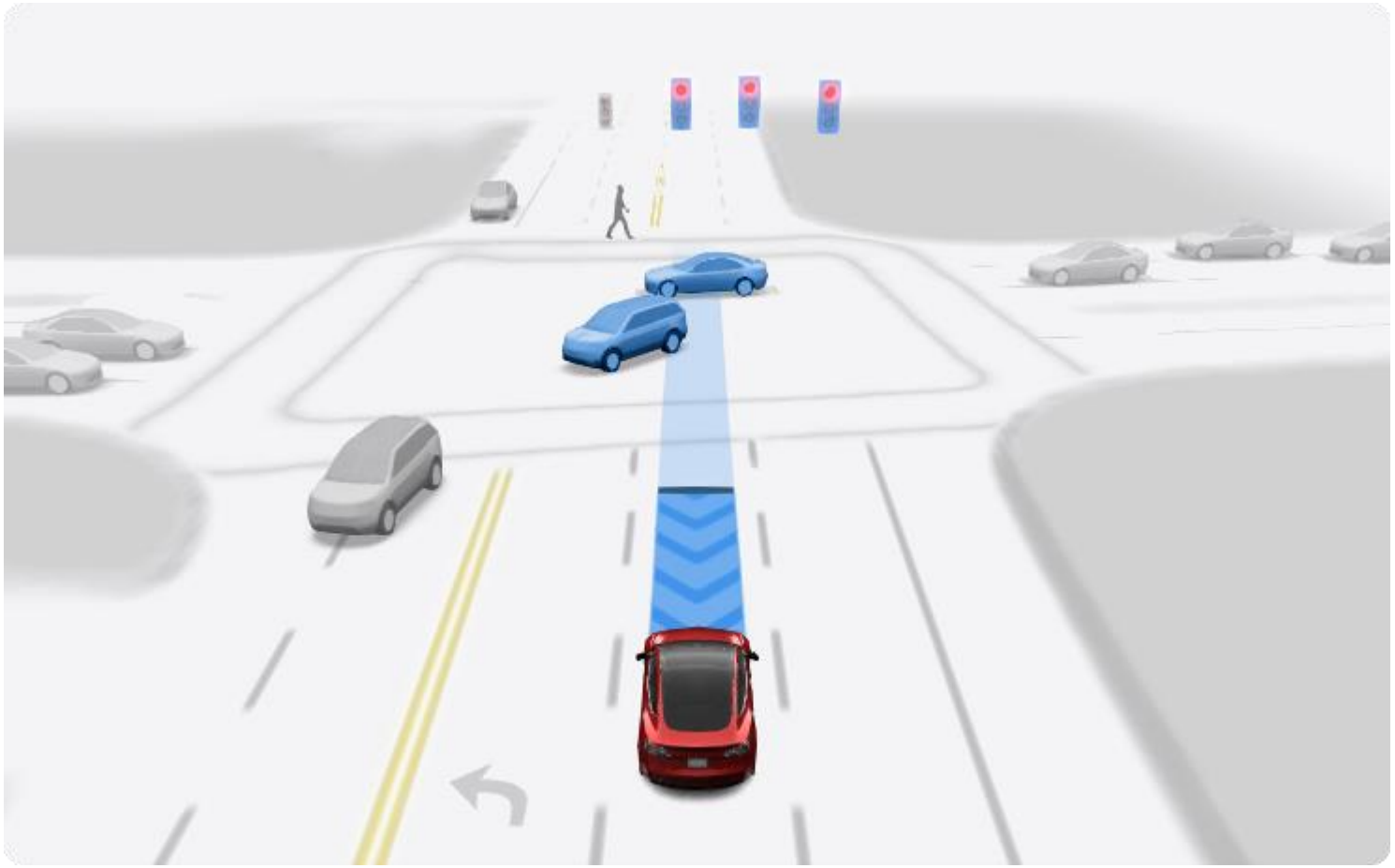
2. Deep Learning

점유 흐름은 부피를 차지하는 물체가 움직이는지 여부와 방향, 속도를 알려줍니다.
앞차가 정지해 있는지, 갑자기 멈췄는지 등을 감지하여 반응할 수 있습니다

점유 네트워크 출력(실제 상황)과 재구성 장면을 비교하여 지속적으로 학습시킵니다.
차량 내에서 실시간으로 작동하며 끊임없이 주변 환경을 확인하고 검증합니다.



2. Deep Learning



2. Deep Learning

– Occupancy Network

"공간 안에 어떤 점이 물체 내부냐 외부냐"를 신경망이 판단하게 하는 방식

핵심 개념

- 입력: 어떤 3D 공간의 좌표 (x,y,z)(x,y,z)
- 출력: 해당 좌표가 물체 안인지 밖인지 (occupancy: 0 또는 1)
- 이걸 하나의 MLP(다층 퍼셉트론)가 학습해서 공간 전체를 묘사함
- 학습 후에는 임의의 해상도로 표면을 추출(Marching Cubes 등) 가능

2. Deep Learning

Occupancy Networks란?

Occupancy Networks는 전통적인 객체 감지 방식을 넘어서는 새로운 접근 방식

- **3D 공간 분할**: 공간을 작은 3D 격자(보통 voxel)로 나누고, 각 voxel이 점유되었는지를 예측.
- **형상 기반 인식**: 객체의 존재 여부를 판단하는 데 초점며, 객체의 종류나 클래스에 의존 않음.
- **동적 객체 인식**: 정적 및 동적 객체를 구분 가능.
- **고속 처리**: 100 FPS 이상의 속도로 실행되어 실시간 처리에 적합.

2. Deep Learning

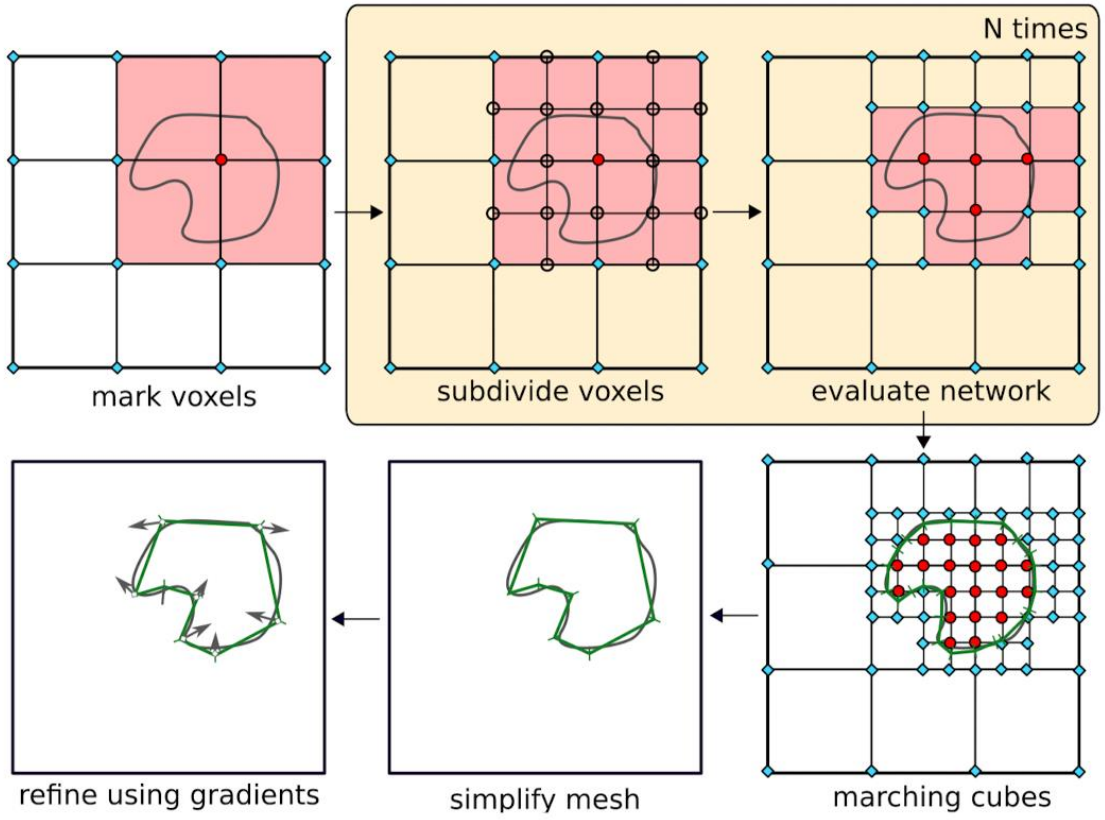


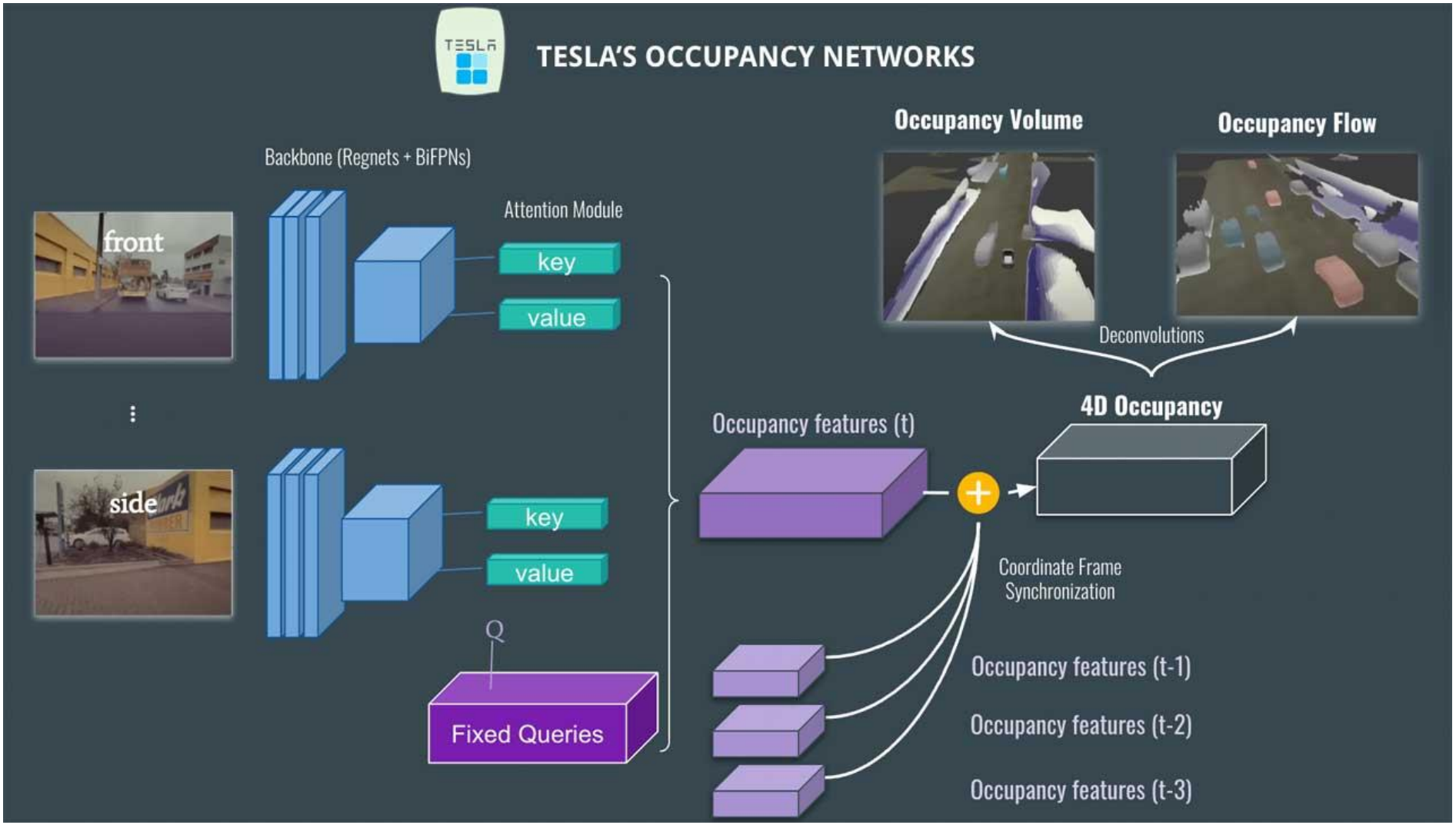
Figure 2: **Multiresolution IsoSurface Extraction:** We first

2. Deep Learning

Tesla의 Occupancy Network 아키텍처 개요

- 입력: 8개의 카메라 영상
- 특징 추출: RegNet 및 BiFPN을 사용하여 이미지에서 특징을 추출
- Attention 모듈: 위치 정보를 포함한 특징을 사용, Occupancy Feature Volume을 생성
- 시간적 융합: 이전 프레임들과의 정보를 융합하여 4D Occupancy Grid를 생성
- 출력: Occupancy Volume과 Occupancy Flow를 생성

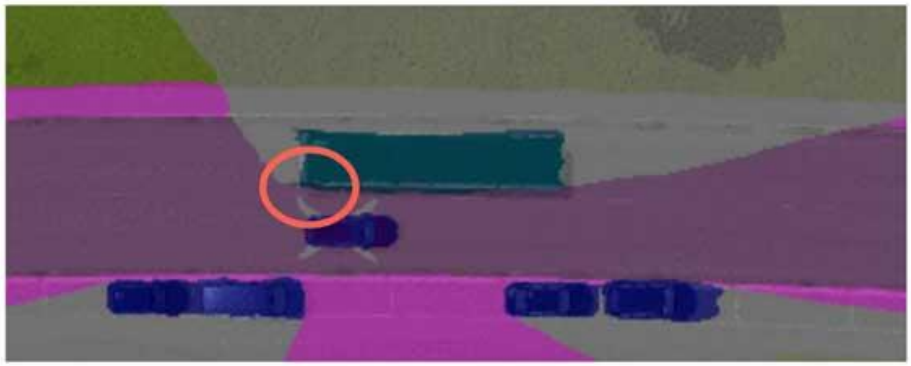
2. Deep Learning

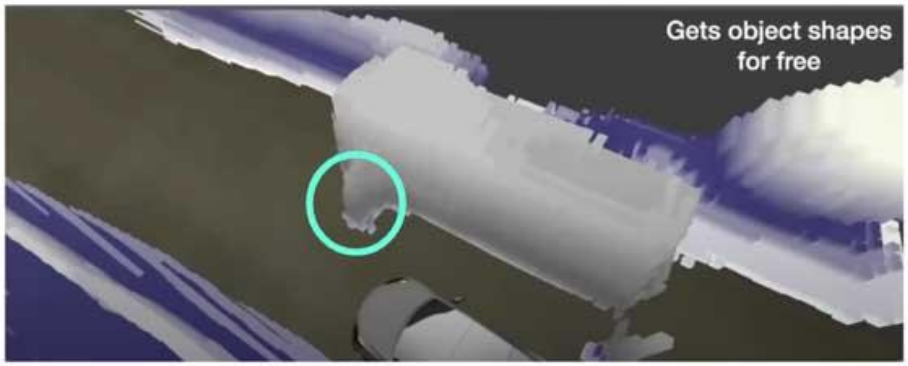


2. Deep Learning

Before

After





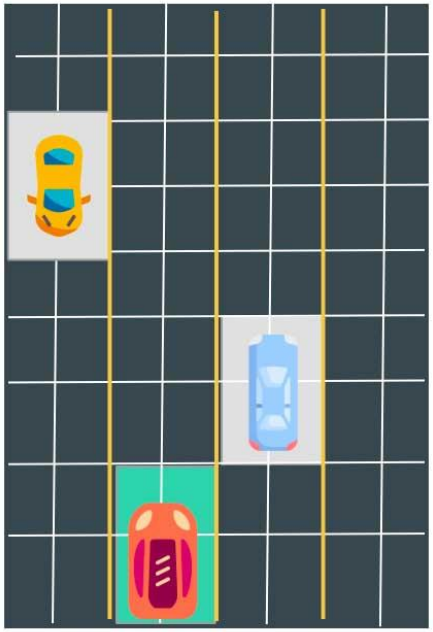
We use a **fixed size rectangle**, no matter the real shape of the truck

we **note the volume as occupied,** and therefore don't risk hitting that.

https://www.thinkautonomous.ai/blog/occupancy-networks/

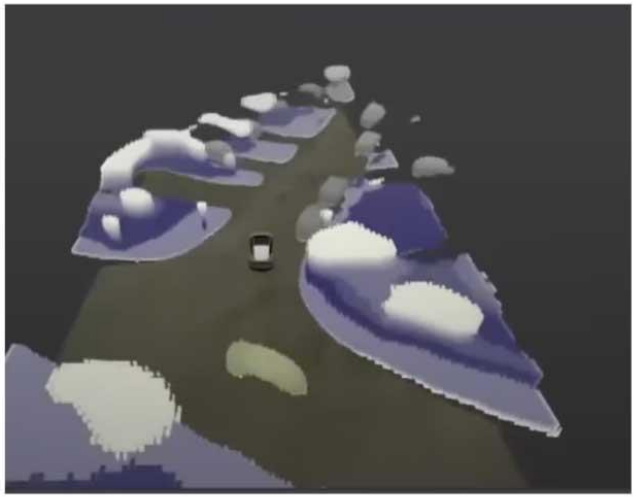
2. Deep Learning

Occupancy Grid / Bird-Eye View



2D

Occupancy Volume



3D

2. Deep Learning

RegNet (Regularized Network)

- **등장 배경**

  ResNet, DenseNet 등은 성능은 좋지만 구조가 사람 손으로 설계된 비정형적 구조

  Facebook AI는 실험적으로 수천 개의 네트워크를 생성해봤더니 성능 좋은 네트워크들은 일정한 규칙에 따라 폭과 깊이가 증가하는 구조를 가짐
  수학적으로 간단한 함수로 커지는 네트워크(RegNet) 설계

- **구조적 특징**

  Block의 채널 수 (width), 깊이 (depth) 를 함수로 정의
  하드웨어 최적화 :
  연산량(Flops) 대비 정확도 우수, 범용성 백본(backbone)으로 쓰기에 안정적

  예시: RegNetY-16GF: COCO나 ImageNet에서 우수한 성능을 내는 대표 모델
  Tesla도 Occupancy Network에서 이 구조를 backbone으로 사용

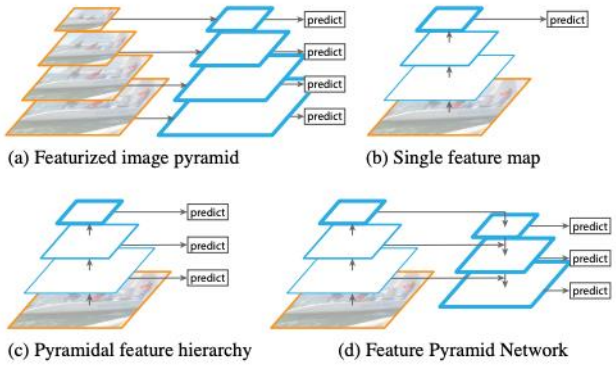
2. Deep Learning

BiFPN (Bidirectional Feature Pyramid Network)

- **등장 배경**
  기존 FPN (Feature Pyramid Network)은 top-down (큰→작은) 방향만 정보 전달하지만,
  객체는 큰 것과 작은 것이 같이 있기 때문에 작은 해상도 정보도 위로 전달하는
  구조가 필요했음

- **BiFPN의 특징**
  양방향 흐름: Top-down + Bottom-up
  Skip 연결: 중복된 feature들끼리 병합

- **가중치 학습:**
  여러 scale에서 들어오는 feature를 단순히 더하지 않고,
  각각의 importance를 학습하여 더함 (Weighted sum)



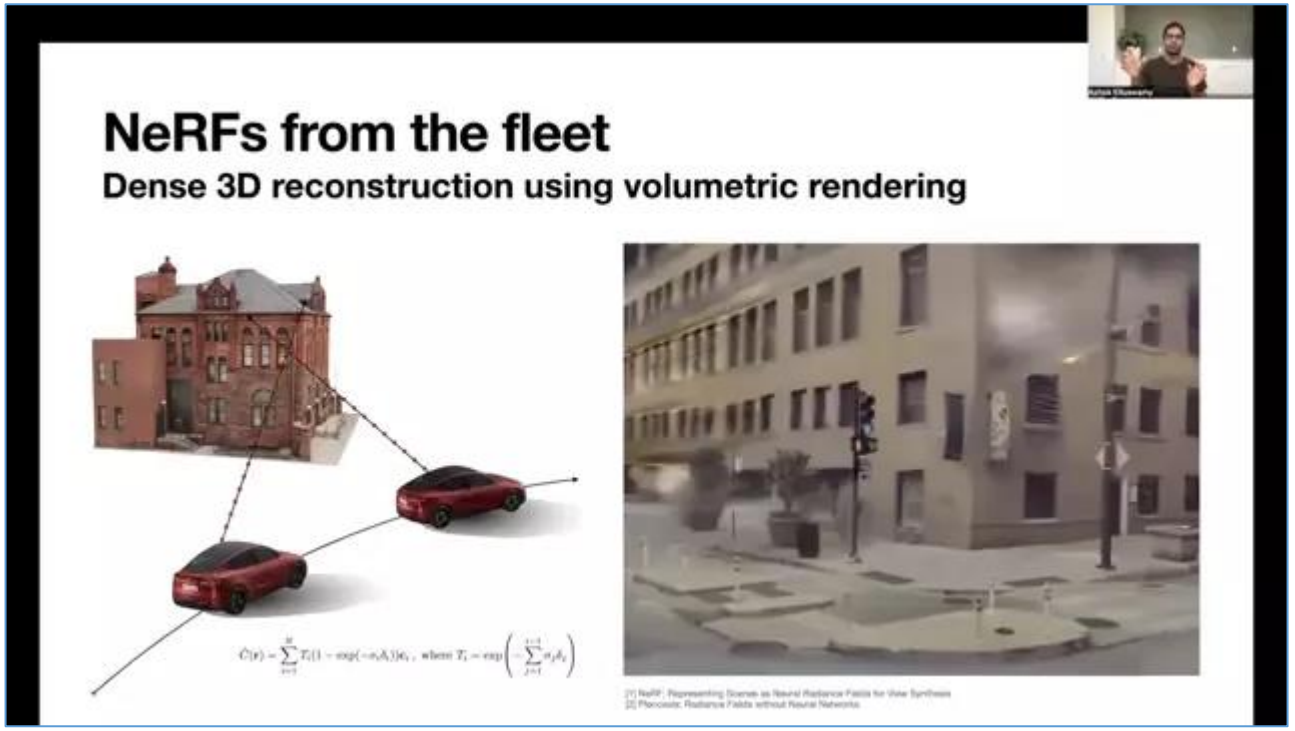
2. Deep Learning

NeRF (Neural Radiance Fields)
3D 공간의 한 점에서의 색(RGB)과 밀도(α)를 예측해, 2D 이미지로 렌더링 가능하게

- **등장 배경**
  3D 재구성에서 photorealistic한 이미지 렌더링이 어려움
  "이미지를 쏴서 다시 이미지를 만들자!"라는 접근으로 시작됨 (볼륨 렌더링 기반)

- **핵심 개념**
  입력: 3D 좌표 (x,y,z) + viewing direction (θ,φ)
  출력: 그 점에서 보는 RGB 색상과 밀도 값
  이를 통해 volume rendering 알고리즘을 이용해 이미지를 재구성

2. Deep Learning



https://blog-ko.superb-ai.com/nerf-view-synthesis-for-representing-scenes/

2. Deep Learning

https://waabi.ai/implicito/

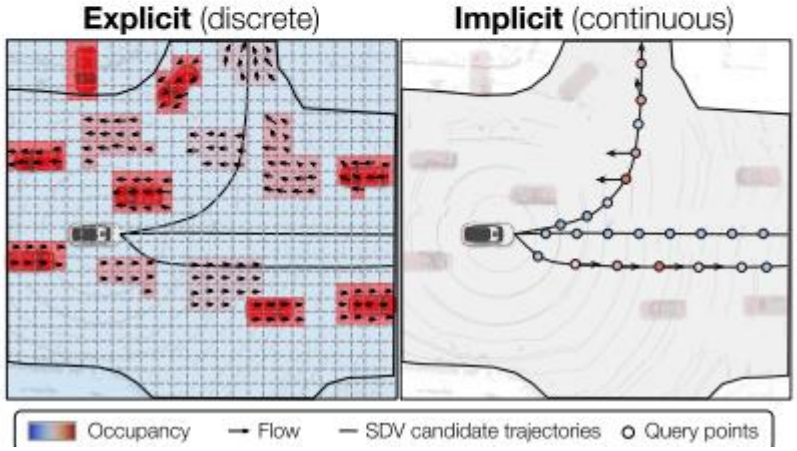
2. Deep Learning

Implicit Occupancy Flow Fields for Perception and Prediction in Self-Driving

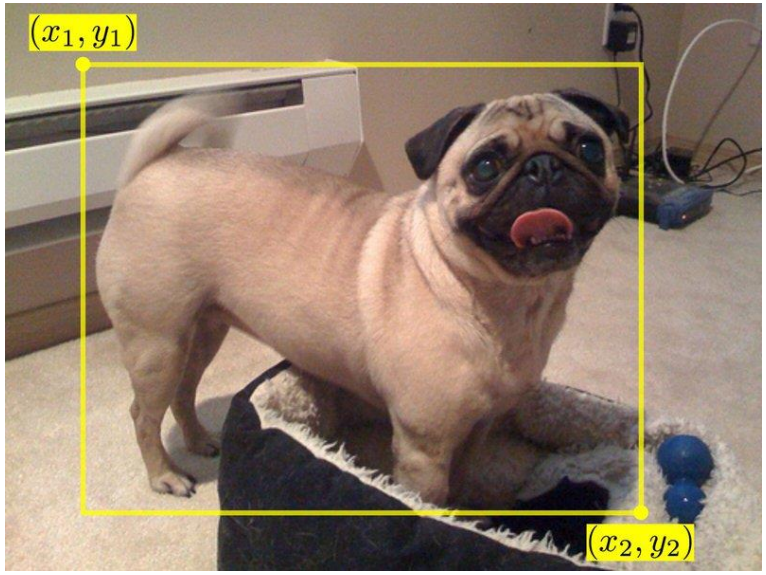
Ben Agro*, **Quinlan Sykora***, **Sergio Casas***, **Raquel Urtasun**
Waabi, University of Toronto
{bagro, qsykora, sergio, urtasun}@waabi.ai

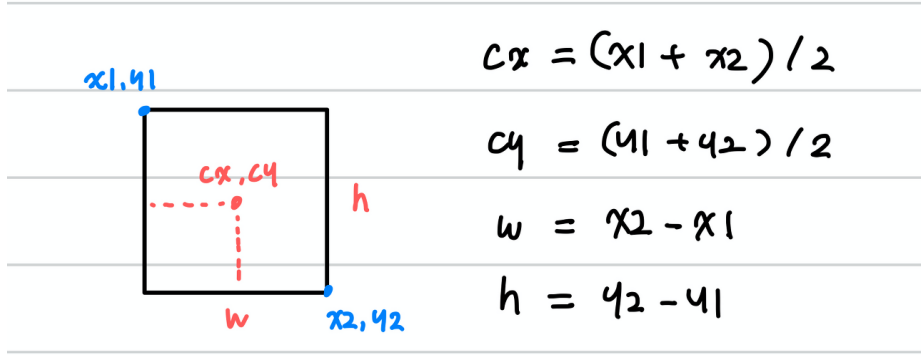
Abstract

A self-driving vehicle (SDV) must be able to perceive its surroundings and predict the future behavior of other traffic participants. Existing works either perform object detection followed by trajectory forecasting of the detected objects, or predict dense occupancy and flow grids for the whole scene. The former poses a safety concern as the number of detections needs to be kept low for efficiency reasons, sacrificing object recall. The latter is computationally expensive due to the high-dimensionality of the out-



2. Deep Learning(객체탐지)





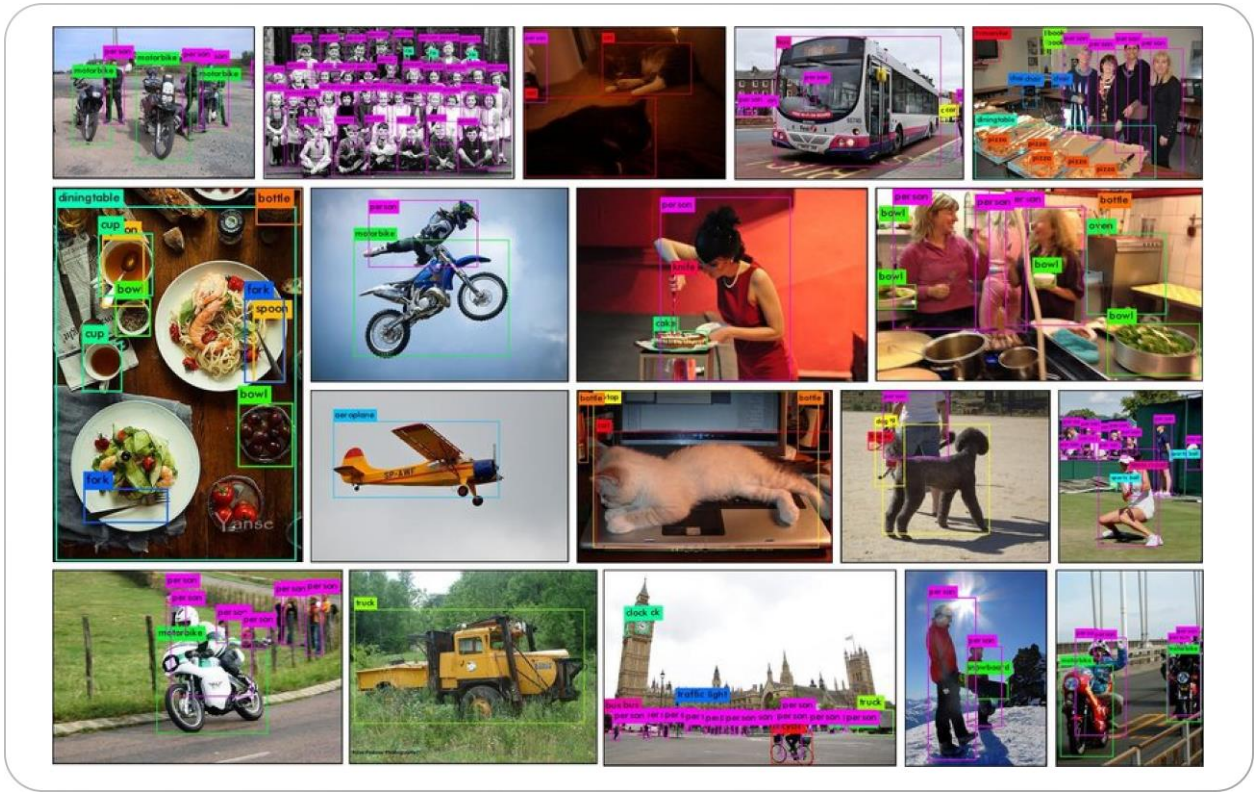
Bounding Box

Bounding Box는 바운딩 박스, Bbox

corner coordinates : bounding box를 좌상점과 우하점으로 나타낸다 ,
center coordinates : bounding box를 중심점과 width, height 로 나타낸다

- 코너 좌표 방식: (10, 20), (50, 70)
- 중심 좌표 방식: (30, 45), 너비 40, 높이 50

2. Deep Learning(객체탐지)



COCO dataset(common objects in context)

약 80개의 객체 카테고리와 약 330,000개의 이미지로 구성.
바운딩 박스 정보와 객체 분할을 위한 픽셀-레벨 마스크 정보가 포함.
객체 탐지 및 분할 알고리즘의 성능 평가를 위해 많이 사용.(YOLO의 학습데이터!!)
평균 정밀도(AP), 평균 재현율(AR) 등의 지표를 통해 알고리즘의 정확도와 성능을 평가

2. Deep Learning(객체탐지)

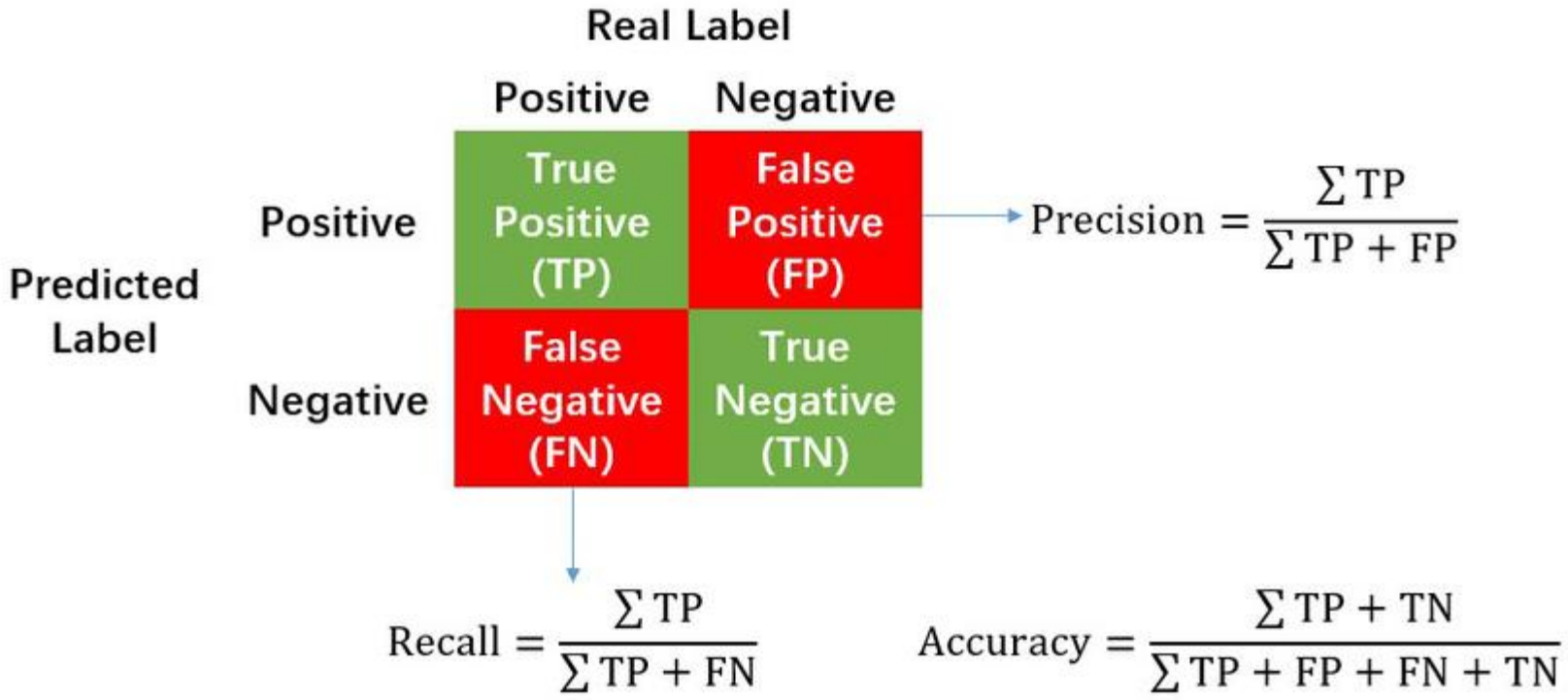
Type	Model Name	mAP val
Two-Stage	FRCNN - R50 FPN	40.2
Two-Stage	DetectorRS - R50 HTC	49.1
Single stage w anchors	SSD - VGG16	29.4
Single stage w anchors	EfficientDet - D4	48.3
Single stage w anchors	YOLOv5 - XL	49.2
Single stage w/o anchors	CornetNet - Hourglass 104	41.8
Single stage w/o anchors	CenterNet - DLA 34	37.4

$$mAP = \frac{1}{k}\sum_{i}^{k} AP_i$$

mAP(means of Average Precision)

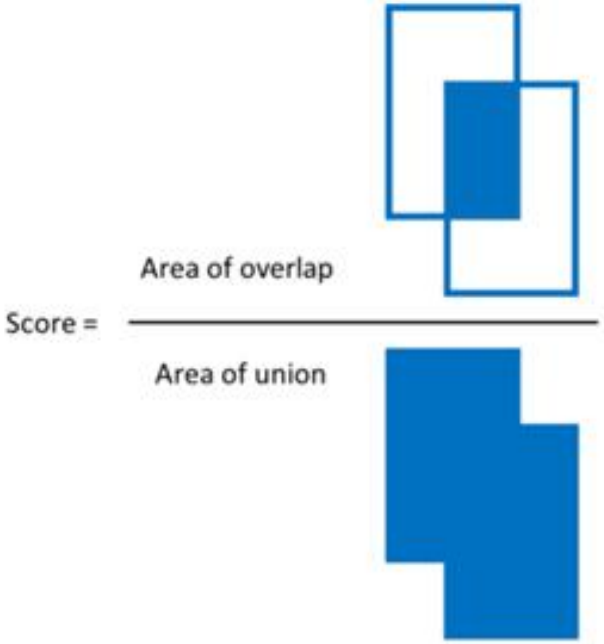
Data set내의 전체 클래스에 대해 각 클래스에 대한 AP(Average Precision)을 계산하고 그 숫자의 평균(means)
단일 클래스의 모델성능정보 대비 전체 Data set의 모델 성능정보 기준 !!

2. Deep Learning(객체탐지)



mAP(means of Average Precision)

2. Deep Learning(객체탐지)



IoU(Intersection over Union)

정답영역(Ground Truth bounding box)과 예측영역(predicted bounding box), 박스의 차이를 상대적으로 평가하기 위한 방법 , 교차하는 영역을 합친 영역으로 나눈 값
정답 영역과 예측 영역이 겹쳐진 부분이 클 수록 IoU의 값이 커진다

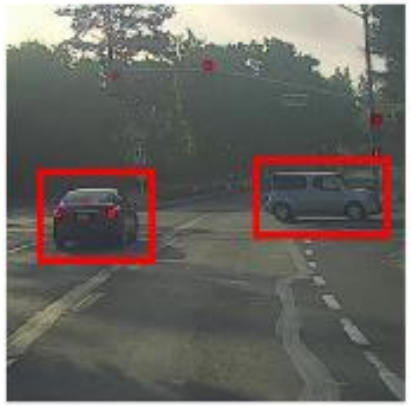
https://velog.io/@cha-suyeon/%EB%94%A5%EB%9F%AC%EB%8B%9D-Object-Detection-Localization-%EA%B0%9C%EB%85%90-%EC%A0%95%EB%A6%AC

2. Deep Learning (객체탐지)

Image classification

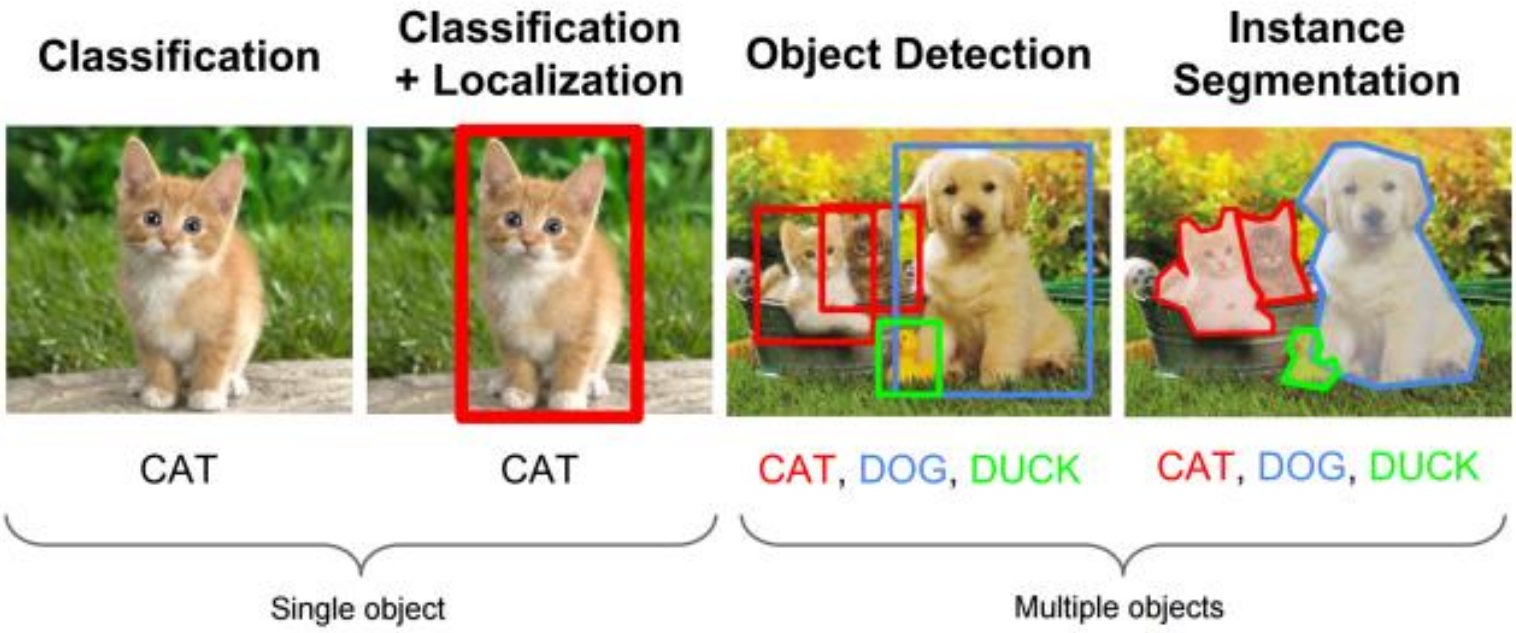
Classification with localization

Detection



단순히 이미지를 분류하는 것이 **classification**,
car라는 **object의 위치**를 알아내는 것이 Classification with **localization**,
localization을 통해 **여러 object**를 인식하는 것을 **Detection**

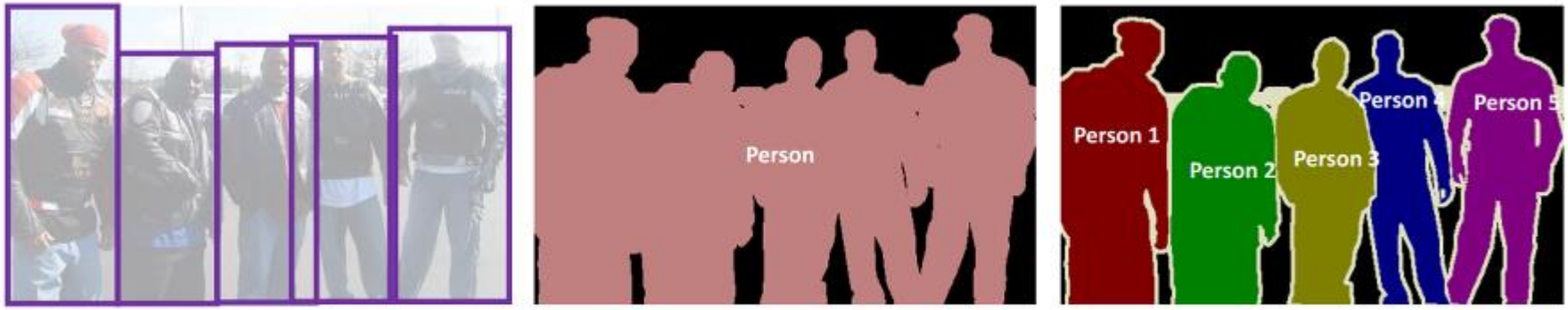
2. Deep Learning (객체탐지)



Object Detection이란
물체를 분류하는 Classification,
물체가 박스를 통해 (Bounding box) 위치 정보 찾는Localization

Object Detection = Classification + Localization
Object Detection = Multi-Labeled Classification+ Bounding Box Regression(Localization)

2. Deep Learning (객체탐지)



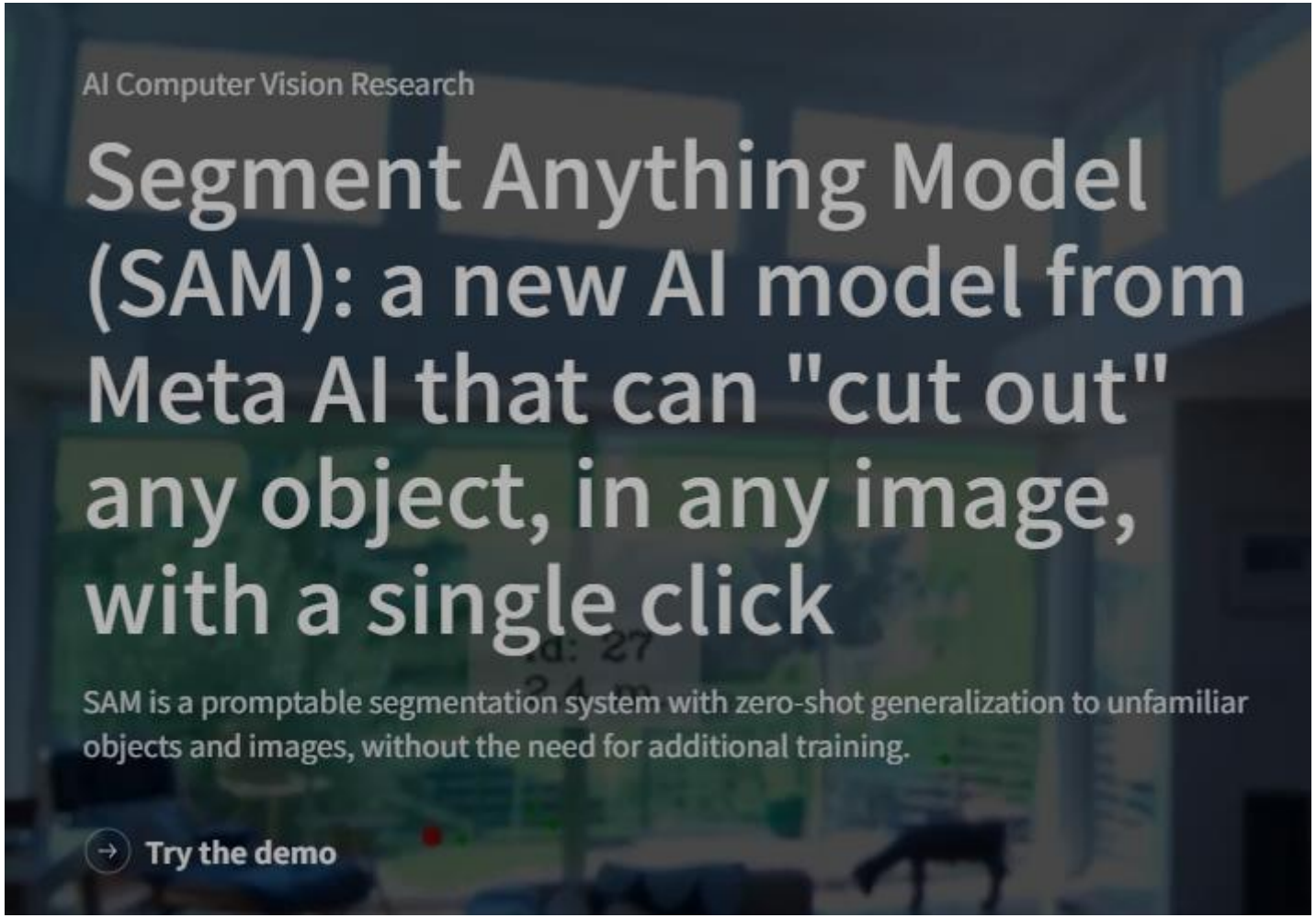
Object Detection　　Semantic Segmentation　　**Instance Segmentation**

Image Segmentation :이미지를 픽셀 단위의 다양한 segments로 분할 task.
이미지의 모든 픽셀에 라벨을 할당하는 task.

Segmentation :
- 동일한 클래스에 해당하는 픽셀을 같은 색으로 칠하는 Semantic Segmentation.
- 동일한 클래스/다른 사물의 픽셀이면 다른 색으로 칠하는 Instance Segmentation

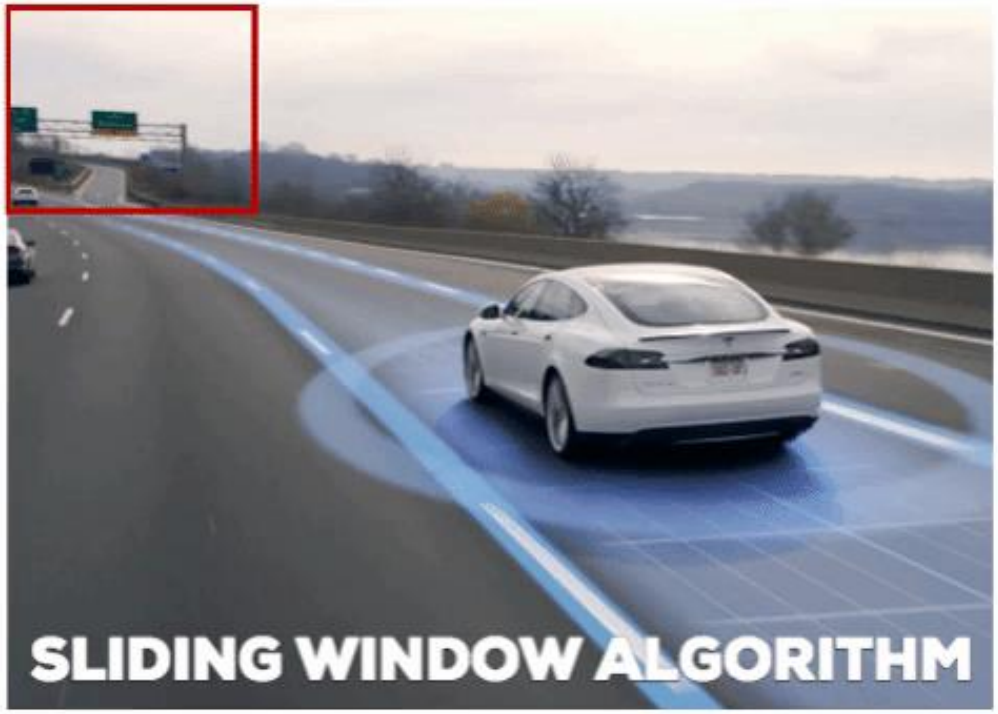
https://segment-anything.com/

2. Deep Learning (객체탐지)



https://pyimagesearch.com/2015/03/23/sliding-windows-for-object-detection-with-python-and-opencv/

2. Deep Learning (객체탐지)

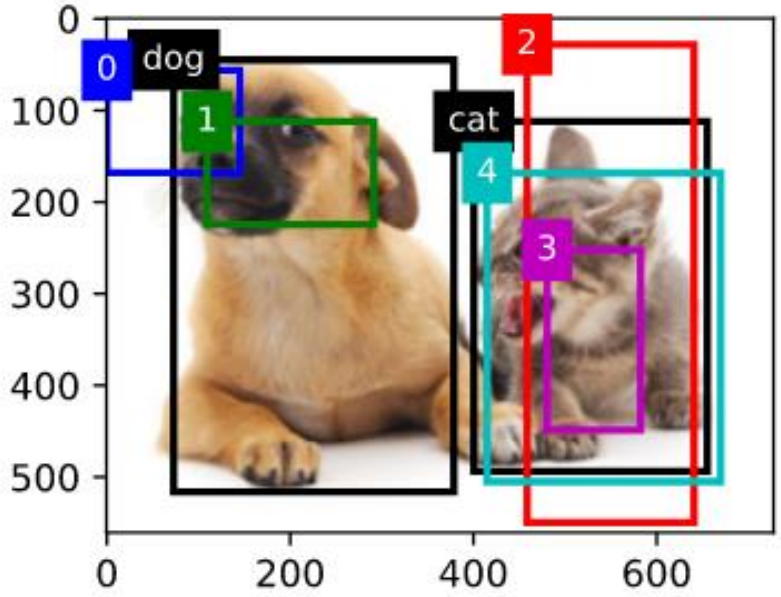


슬라이딩 윈도우(Sliding window)

Localization network의 입력으로 만들기 위해 원본 이미지에서 잘라내는 크기를 window 크기로 영역을 이동시키며(sliding) 이미지 탐색을 수행하는 방식
이미지 내의 오브젝트가 어떤 크기로 어떤 종횡비를 갖는지 알 수 없기 때문에 매우 많은 종류의 Windows를 생성해야 하기 때문에 매우 비효율적인 방법일 수 있다.

https://velog.io/@cha-suyeon/%EB%94%A5%EB%9F%AC%EB%8B%9D-Object-Detection-Anchor-Boxes-NMSNon-Max-Suppression/

2. Deep Learning (객체탐지)



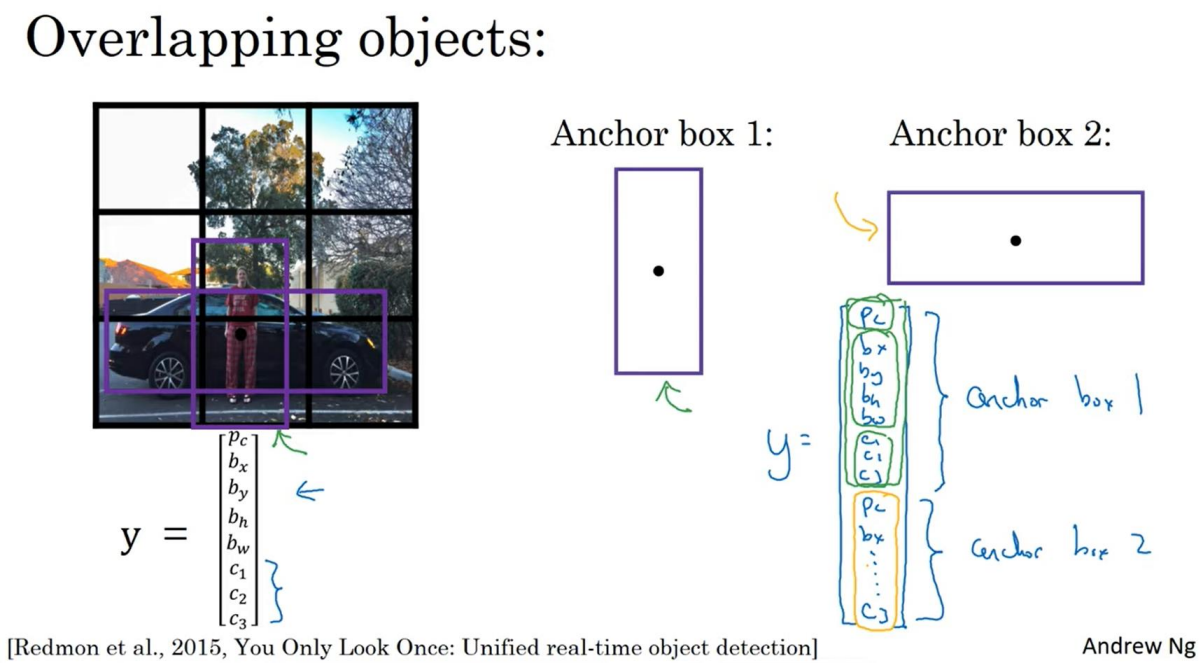
앵커 박스(Anchor Box)는 사전에 정의된 크기와 종횡비를 가지는 사각형. 일반적으로 이미지 내에서 객체가 나타날 것으로 예상되는 여러 위치에 앵커 박스들을 미리 배치.

바운딩 박스는 실제로 탐지된 객체의 경계를 나타내는 사각형. 객체의 경계선을 둘러싸는 최소한의 크기로 설정되며, 바운딩 박스의 좌표와 크기 정보를 통해 객체의 위치와 형태를 추론할 수 있다.

객체 탐지는 앵커 박스를 기반으로 예측한 객체의 위치 및 크기를 바운딩 박스로 표현

https://velog.io/@cha-suyeon/%EB%94%A5%EB%9F%AC%EB%8B%9D-Object-Detection-Anchor-Boxes-NMSNon-Max-Suppression

2. Deep Learning (객체탐지)



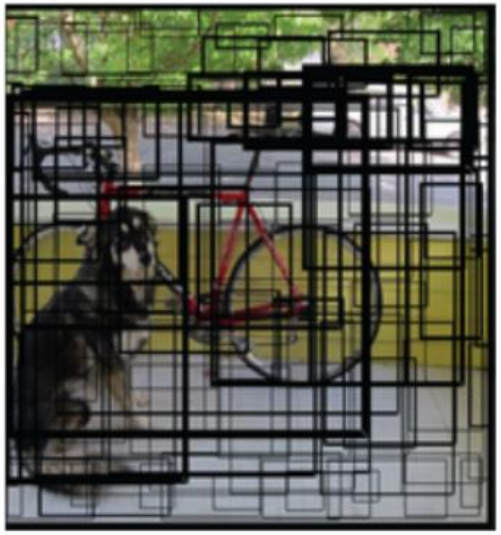
Anchor box 정리

•y의 label을 보면 Anchor box가 2개가 됨에 따라서 output dimension이 두 배가 되었다. 그리고 각각은 정해진 Anchor box에 매칭된 object를 책임지게 됩니다.

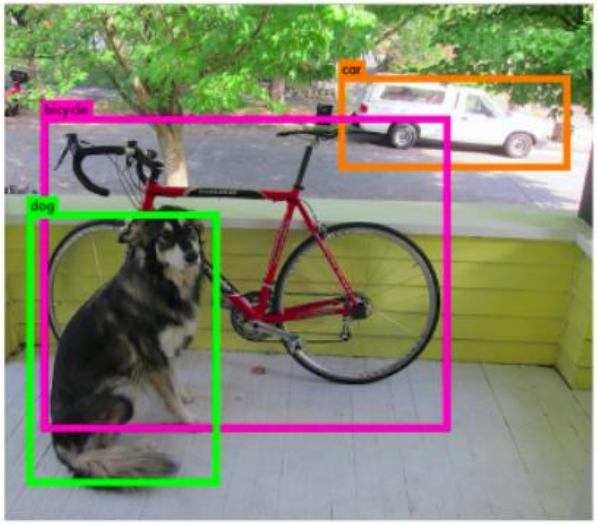
• grid cell에서 Anchor box에 대한 object 할당은 IoU로 할 수 있다. 인식 범위 내에 object가 있고 두 개의 Anchor box 경우 IoU가 더 높은 Anchor box에 object를 할당.

https://velog.io/@cha-suyeon/%EB%94%A5%EB%9F%AC%EB%8B%9D-Object-Detection-Anchor-Boxes-NMSNon-Max-Suppression

2. Deep Learning (객체탐지)



Multiple Bounding Boxes



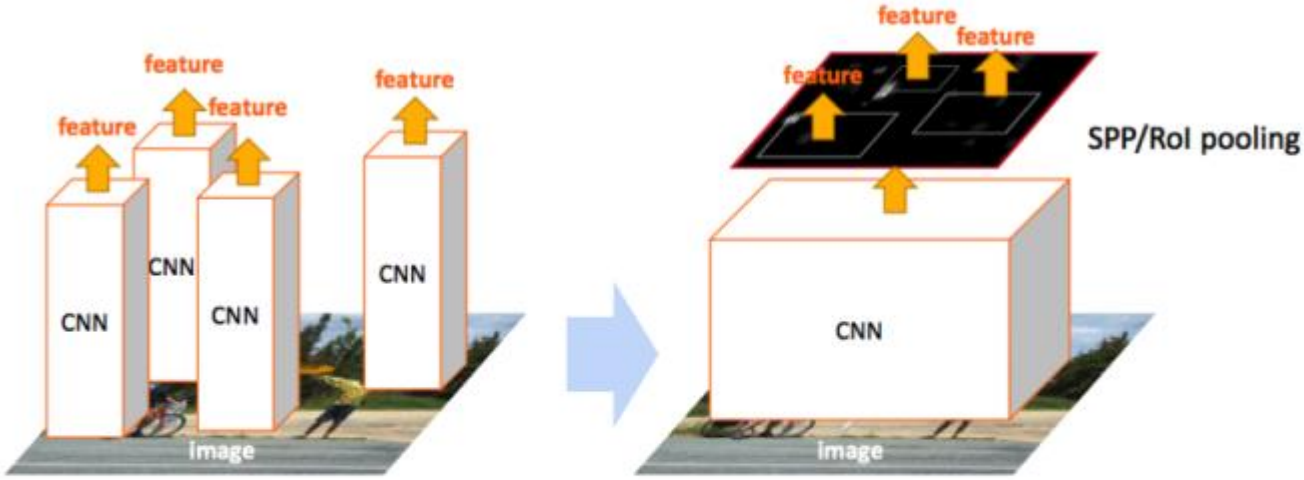
Final Bounding Boxes

NMS(Non-Max Suppression), 비-최대 억제

object detection 알고리즘은 탐지된 객체에 여러 개의 bounding boxes를 생성. 이 중 하나의 bounding box만을 선택해야 하는데 이때 적용하는 기법

NMS는 겹친 박스들이 있을 경우 가장 확률이 높은 박스를 기준으로 기준이 되는 IoU 이상인 것들을 없앤다.

2. Deep Learning (객체탐지)



R-CNN

- Extract image regions
- 1 CNN per region (2000 CNNs)
- Classify region-based features
- Complexity: ~224 × 224 × 2000

SPP-net & Fast R-CNN (the same forward pipeline)

- 1 CNN on the entire image
- Extract features from feature map regions
- Classify region-based features
- Complexity: ~600 × 1000 × 1
- ~160x faster than R-CNN

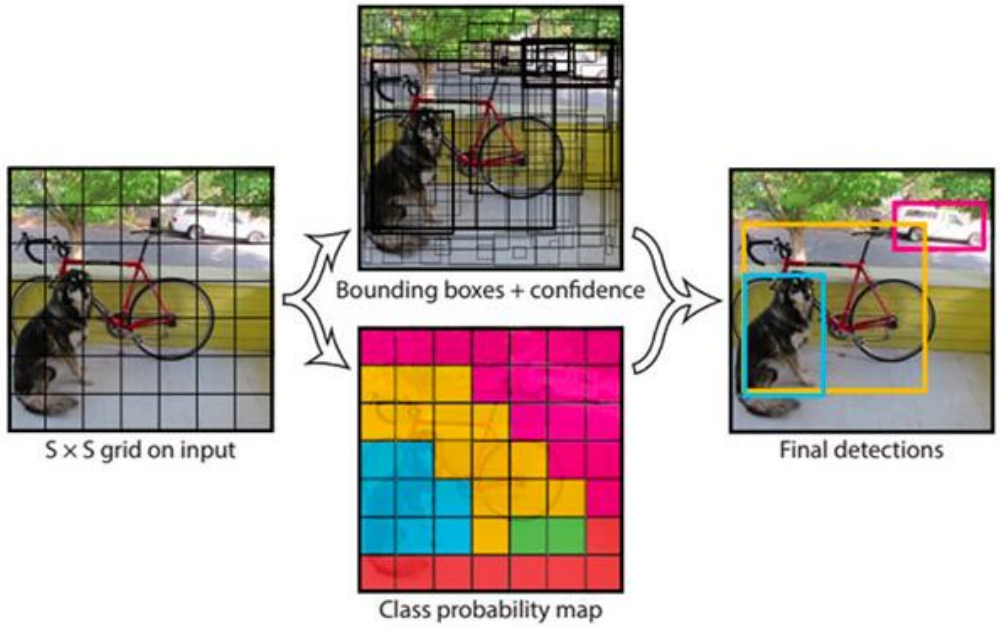
ROI(Region of Interest) pooling

주로 Fast R-CNN에서 적용. 추출된 특징 맵에서 앵커 박스에 해당하는 부분을 잘라내고 고정된 크기의 특징 맵으로 변환하는 과정을 수행.

예를 들어, 자동차 탐지를 수행하는 Fast R-CNN에서 추출된 특징 맵과 앵커 박스가 주어졌을 때, ROI pooling은 앵커 박스를 일정한 크기로 분할하여 각 분할 영역에서 최댓값을 선택하고, 선택된 값들을 모아 고정된 크기의 특징 벡터로 변환

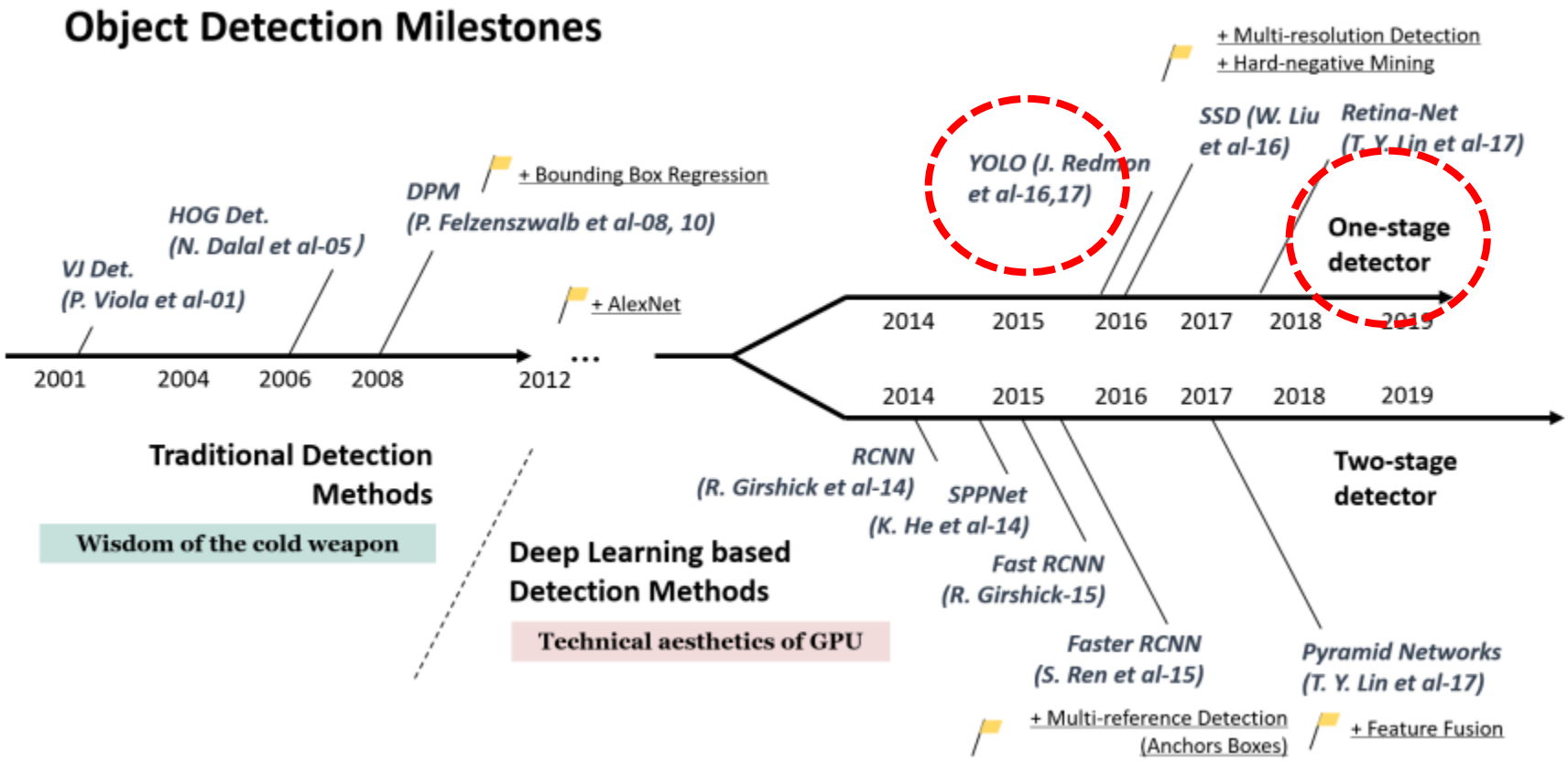
2. Deep Learning (객체탐지) one stage detector

YOLO (You Only Look Once)

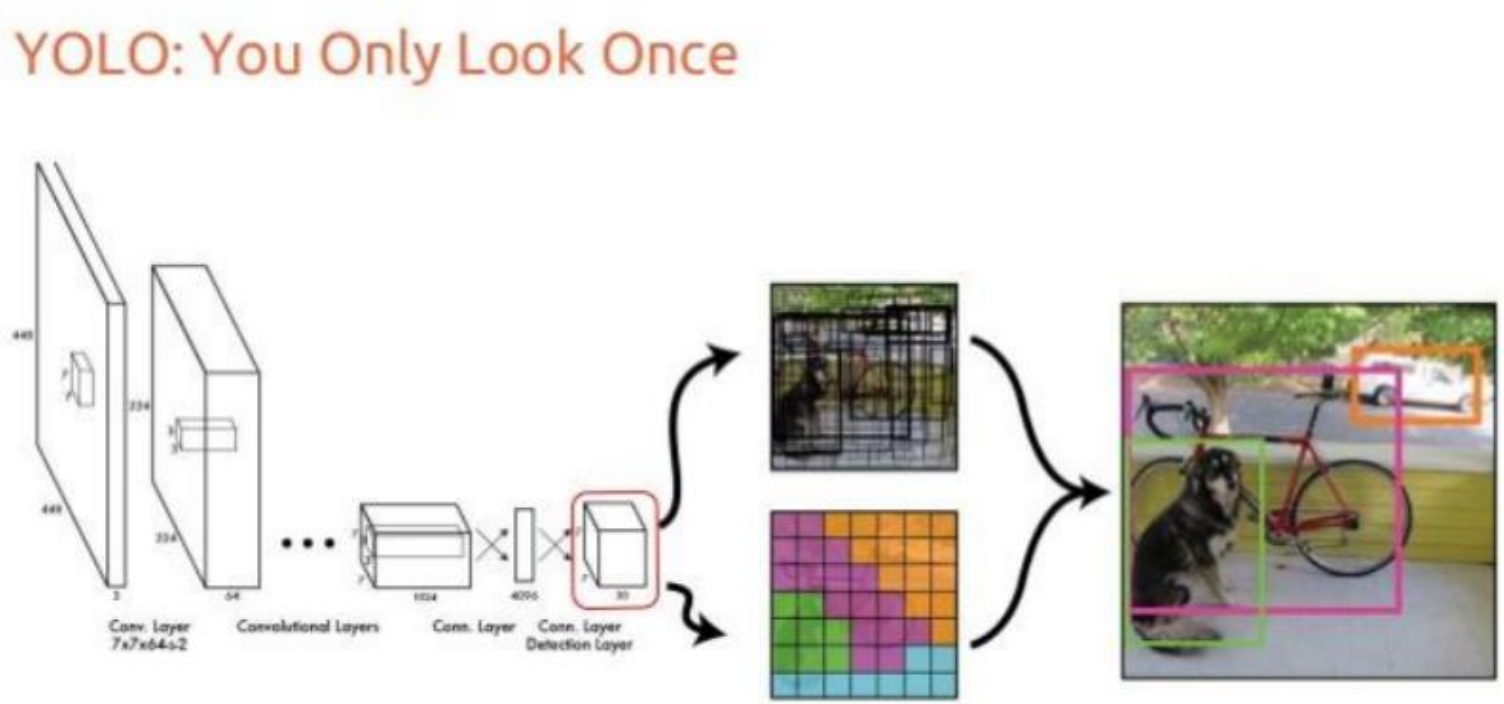


YOLO는
이미지를 grid로 나누고, Sliding window 기법을 Convolution 연산으로 대체해 Fully Convolutional Network 연산을 통해 grid cell별로 Bbox를 얻어낸 뒤, Bbox들에 대해 NMS를 한 방식

2. Deep Learning (객체탐지)



2. Deep Learning (객체탐지)

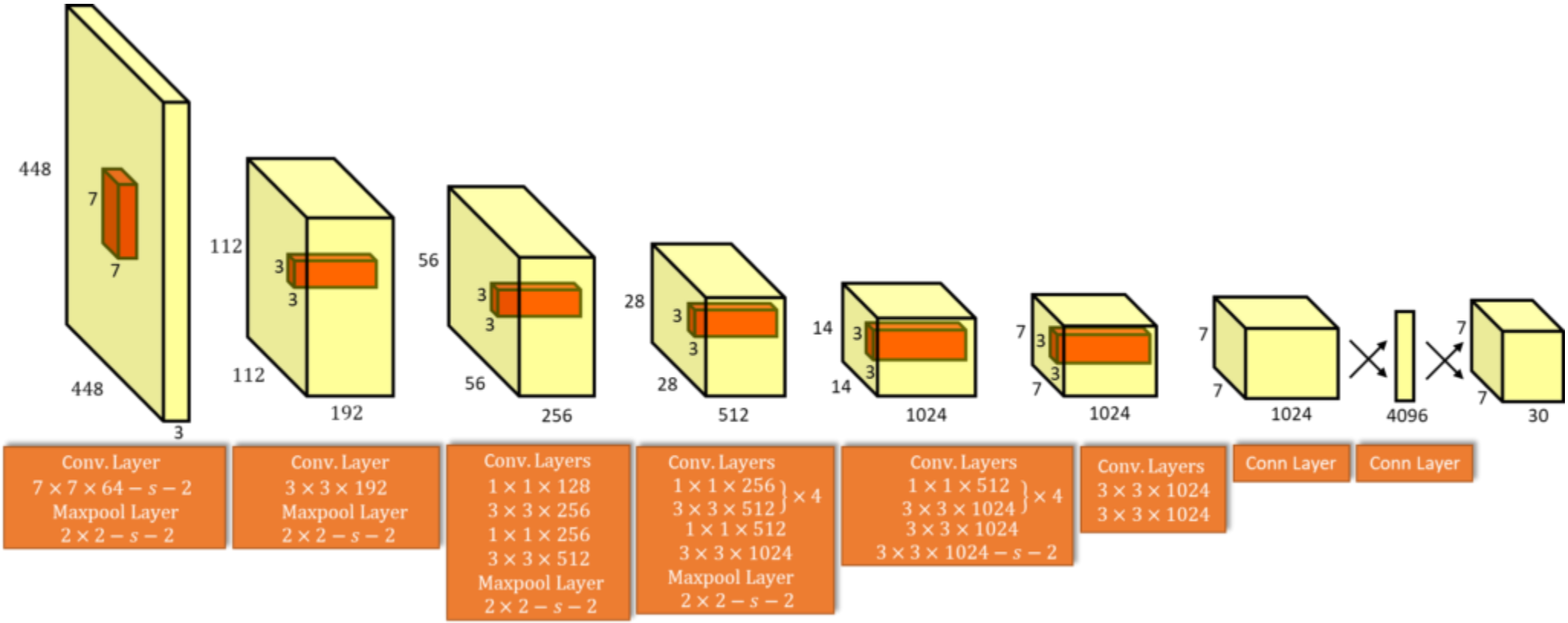


Bounding box 조정과 Classification을 동일 신경망 구조를 통해 동시에 실행하는 통합 인식(Unified Detection)을 구현

S*S개의 Grid 구현 > Grid내 B개의 Bounding Box 예측 > Bbox내의 물체가 존재할 확률 계산(Confidence) > Grid마다 특정물체의 클래스 확률 계산

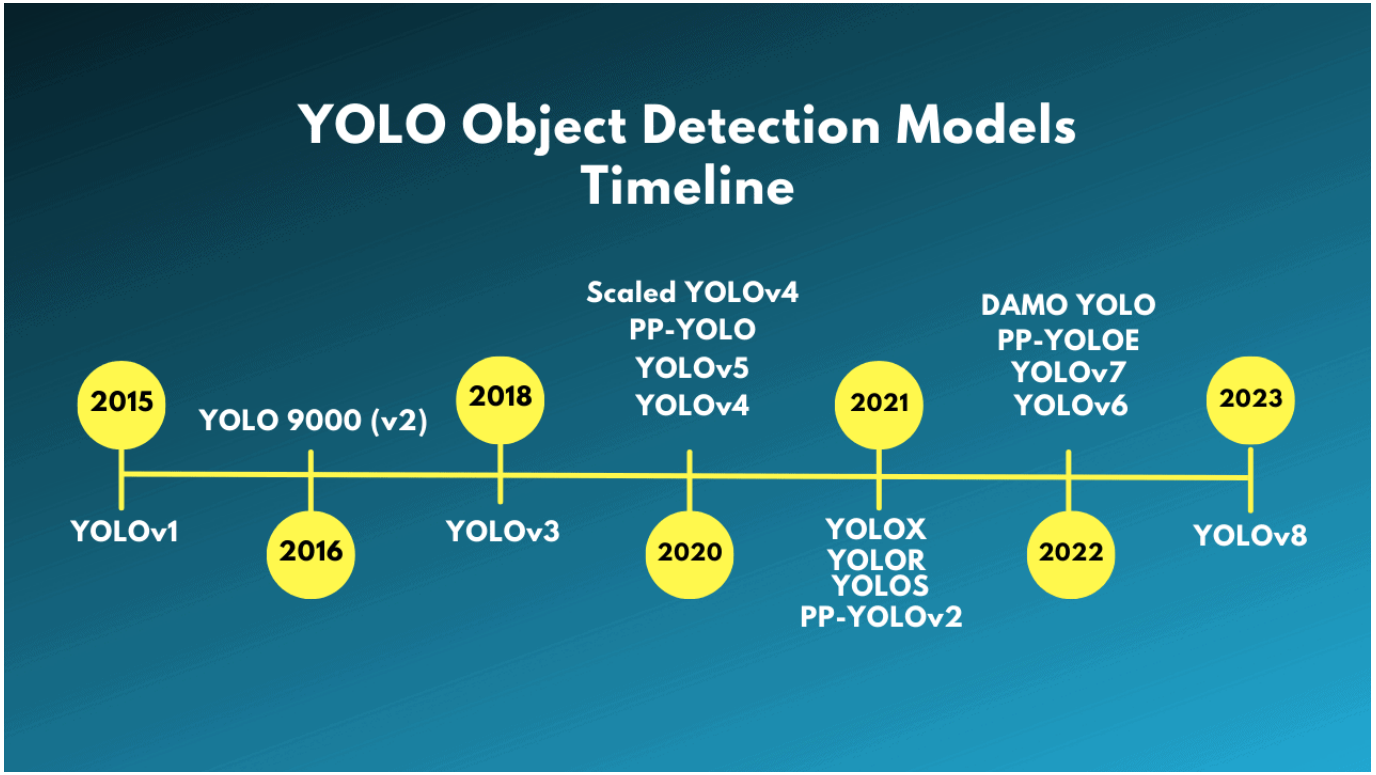
https://datahacker.rs/how-to-peform-yolo-object-detection-using-keras/

2. Deep Learning (객체탐지)



https://openmmlab.medium.com/dive-into-yolov8-how-does-this-state-of-the-art-model-work-10f18f74bab1

2. Deep Learning (객체탐지)



	YOLOv5	YOLOv6	YOLOv7	YOLOv8
Inference Time	19ms	22ms	19ms	16.1 ms

2. Deep Learning(YOLO)

mAP (mean Average Precision) @0.5
모델의 **정확도(Precision)**와 **재현율(Recall)**의 조합을 종합적으로 측정하는 지표
"모델이 객체를 얼마나 잘 탐지하고, 정확히 위치시켰는가?".

"IoU ≥ 0.5일 때" 정확도(Precision)의 평균 값을 의미
예측 박스와 정답 박스가 50% 이상 겹치면 성공(True Positive), 모든 클래스의 평균 정확도.

모델	mAP@0.5	추론 속도(ms)	주요 특징
YOLOv5	93.4%	7.1	빠른 추론 속도와 높은 정밀도
YOLOv8	94.5%	15.9	높은 리콜과 다양한 비전 작업 지원
YOLOv11	95.3%	7.7	정확도와 속도의 균형, 다양한 환경에 적합
YOLOv12	96.1%	6.5	향상된 정확도와 추론 속도, 고정밀 객체 탐지에 적합

2. Deep Learning(YOLO)

1. 학습 출력 관련 주요 용어

Bounding Box (바운딩 박스): 객체를 감싸는 사각형 영역
바운딩 박스는 (x, y) 좌표와 (width, height)로 정의.

Confidence Score (신뢰도 점수): 모델이 특정 객체를 탐지했을 때 해당 객체가 있을 확률

Class Probability (클래스 확률): 바운딩 박스 안에서 탐지된 객체가 특정 클래스에 속할 확률

IoU (Intersection over Union): 예측한 바운딩 박스와 실제 정답 바운딩 박스의 겹치는 부분의 비율. IoU는 (겹치는 영역의 면적) / (두 박스의 합집합 면적)으로 계산.

IoU Threshold (IoU 임계값): IoU가 일정 임계값 이상인 경우, 객체 탐지가 정확하다고 간주
예를 들어, IoU가 0.5 이상이면 예측된 객체가 실제 객체와 일치한다고 판단

AP (Average Precision): 모델의 정확도를 평가하는 지표로, Precision-Recall Curve(정밀도-재현율 곡선)를 기반으로 계산됩니다. AP는 여러 IoU 임계값에 대해 계산된 평균값.

mAP (mean Average Precision): AP를 여러 클래스에 대해 평균화한 값

2. Deep Learning(YOLO)

2. 모델 평가 관련 주요 용어

- Precision (정밀도): 모델이 예측한 객체 중에서 실제로 객체가 맞는 비율.
  Precision = TP / (TP + FP)TP(True Positive): 실제 객체를 정확히 탐지한 경우

- FP(False Positive): 객체가 없는데 객체가 있다고 잘못 예측한 경우

- Recall (재현율): 실제 객체 중에서 모델이 정확히 탐지한 비율.
  Recall = TP / (TP + FN)

- FN(False Negative): 실제 객체가 있는데 모델이 이를 탐지하지 못한 경우

- F1-Score: Precision과 Recall의 조화 평균으로, 두 지표의 균형을 평가
  두 값이 매우 다르면 F1-Score가 낮습니다.
  F1 = 2 * (Precision * Recall) / (Precision + Recall)

2. Deep Learning(YOLO)

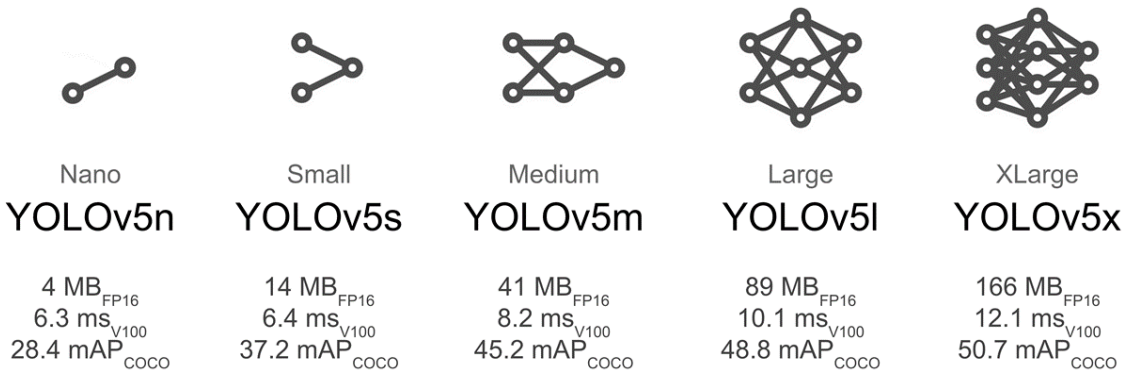
YOLO에서 중요하게 여기는 평가 기준

- **inference Speed (추론 속도):**
  모델이 실제 환경에서 얼마나 빠르게 동작하는지를 평가하는 지표

- **mAP@0.5 (mean Average Precision at IoU threshold 0.5):**
  일반적으로 객체 탐지 모델의 성능을 평가하는 주요 지표로,
  IoU 임계값을 0.5로 설정하고 평균 정밀도를 계산.

2. Deep Learning(YOLO 5)

2. Deep Learning(YOLO 5)



FP16 : 부동소수점 16비트 연산(Floating Point 16-bit)

6.3ms, V100 :YOLO 모델이 약 6.3밀리초의 추론 속도,
NVIDIA V100 GPU 하드웨어를 사용 기준

28.4mAP : 모델이 COCO 데이터셋에 대해 28.4%의 평균 정밀도

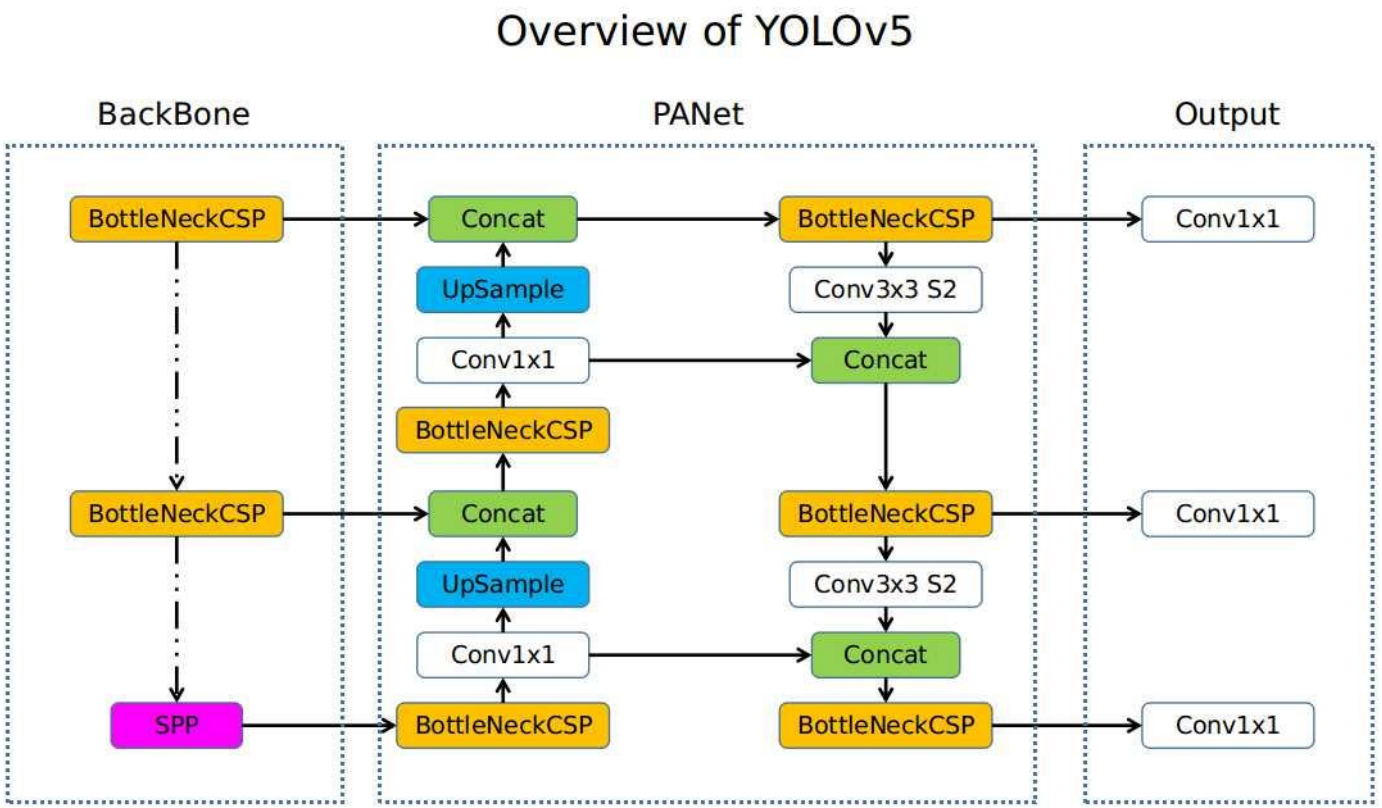
2. Deep Learning(YOLO 5)

Small Models : YOLOv5s with smaller configuration
속도가 빠르고 경량화된 모델로, 작은 객체를 탐지하는 데 유용.
작은 모델은 리소스 제약이 있는 환경이나 실시간 처리가 필요한 경우에 유용.

Medium Models: YOLOv5m
중간 크기의 입력 이미지에 대한 객체 탐지 작업에 적합하며,
정확도와 속도 사이의 균형을 유지.

Large Models :YOLOv5l, YOLOv5x
입력 이미지의 크기에 상관없이 다양한 크기의 객체를 정확하게 탐지할 수 있으며,
더 많은 컴퓨팅 리소스가 필요.
정확도를 우선시하는 복잡한 객체 탐지 작업에 적합

2. Deep Learning(YOLO 5)



2. Deep Learning(YOLO 5)

1. BackBone (특징 추출부)

- **BottleNeckCSP**

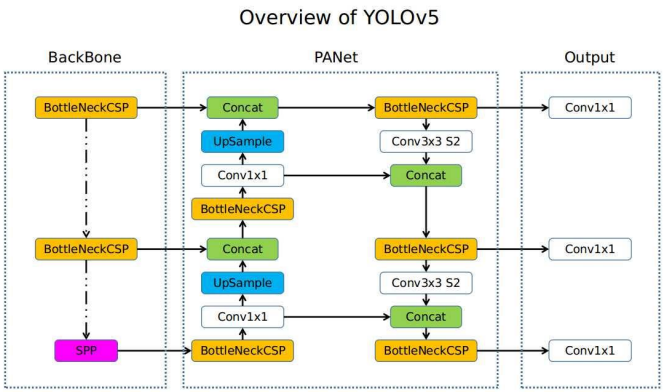
"Bottleneck": CNN 블록 구조로, 연산량을 줄이면서도 특징을 잘 추출하게 해줌
"CSP" (Cross Stage Partial): 특징 정보 일부만 전달, 계산량은 줄이고 성능은 유지

- **SPP (Spatial Pyramid Pooling)**

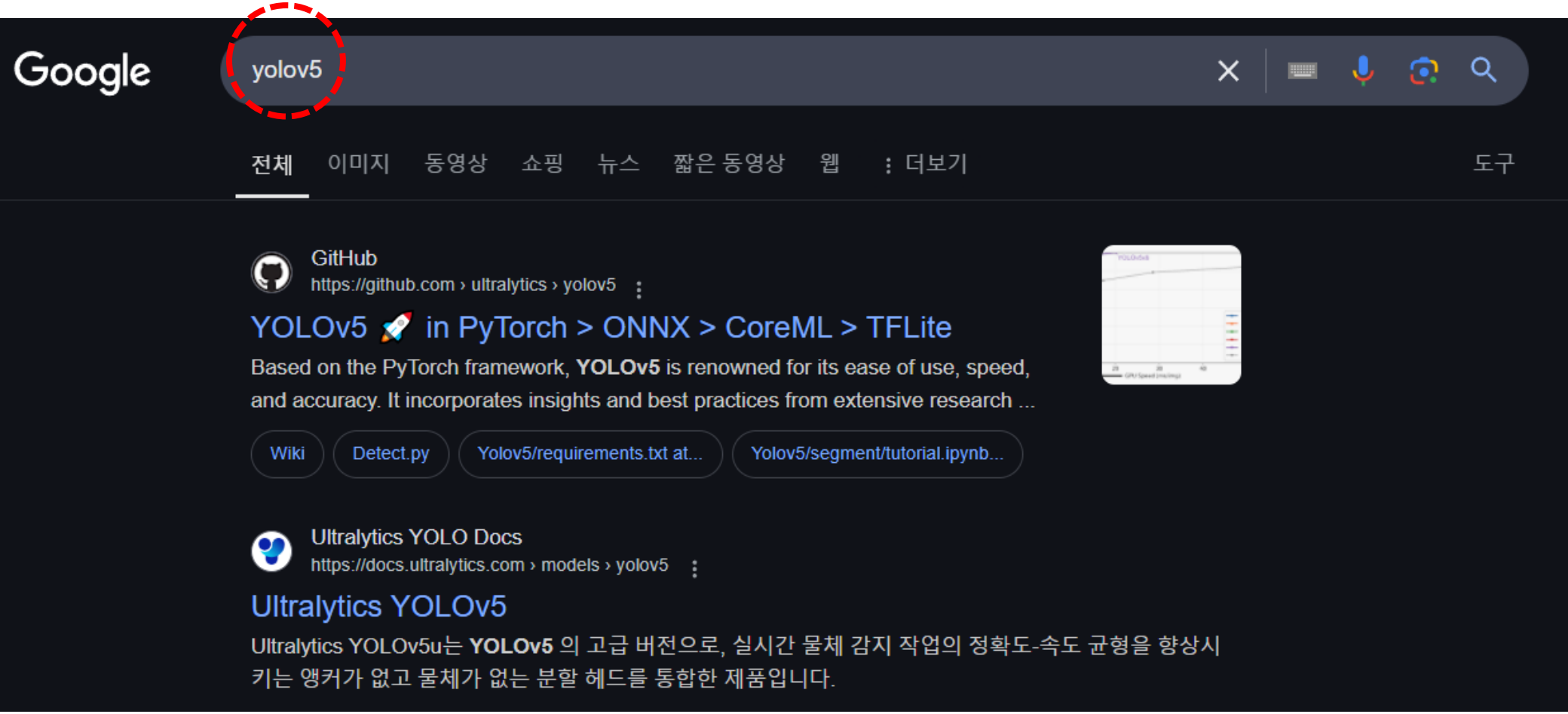
여러 크기의 커널로 풀링하여 다양한 스케일의 정보를 확보

2. PANet (Path Aggregation Network, 특징 통합부)

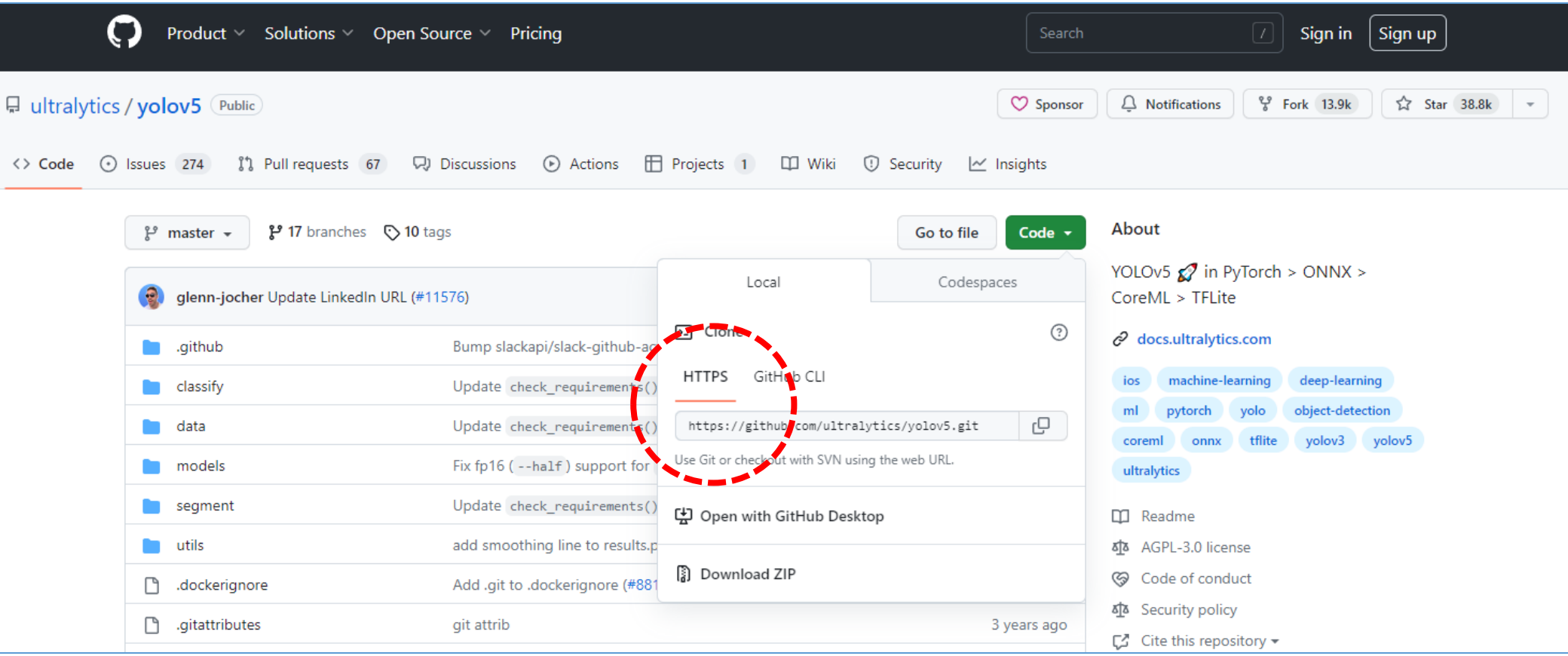
3. Output (출력부)



2. Deep Learning(YOLO 5)



2. Deep Learning(YOLO 5)



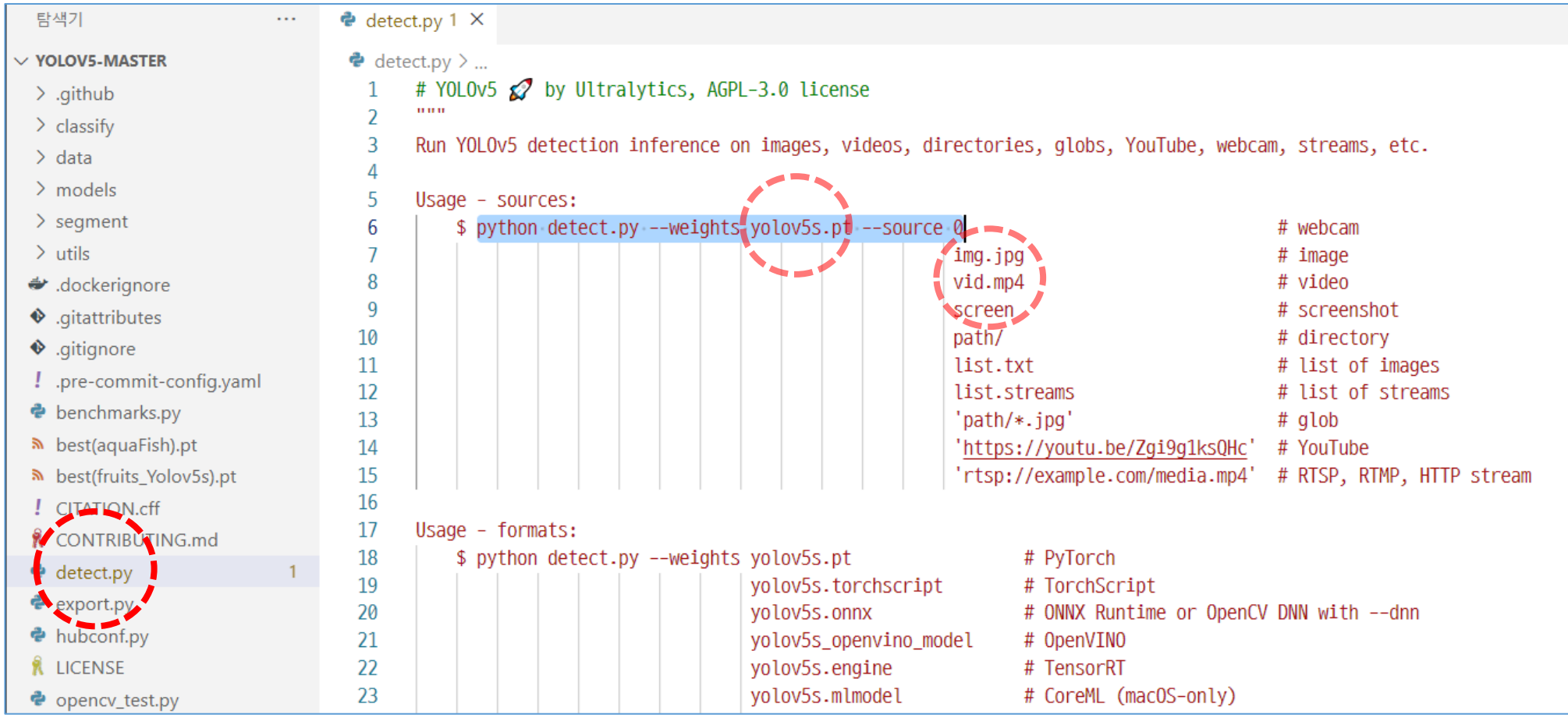
YOLO의 깃허브 사이트에서 주어진 URL을 복사하여 설치한다.

2. Deep Learning(YOLO 5) requirements

Requirements : yolo5 학습용 실행 환경 구성 !!

YOLO 실행에 필요한 환경으로 반드시 실행해서 환경설정을 해줘야 한다.

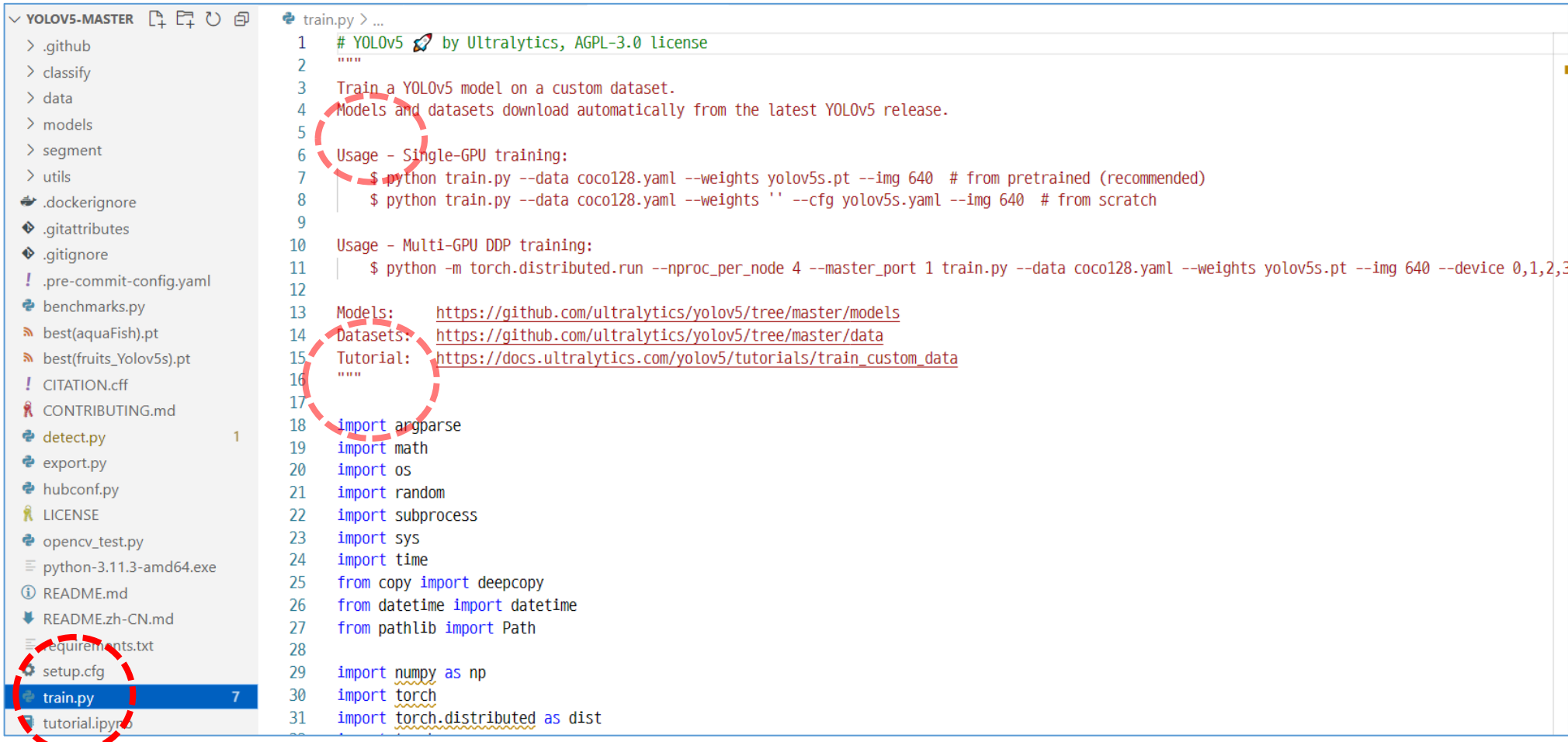
2. Deep Learning(YOLO 5) detect.py



Detect.py : 학습된 모델로 탐지를 수행하는 코드 !!

객체 탐지시 사용해야 하는 명령어 예제를 참조하여 해당 옵션을 지정한다.

2. Deep Learning(YOLO 5) train.py

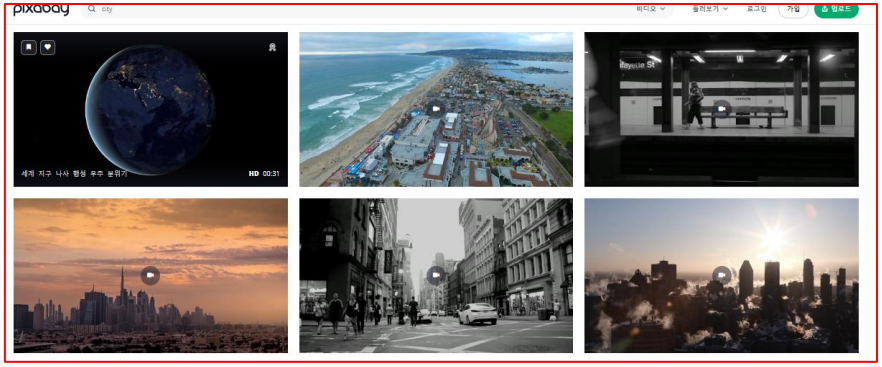


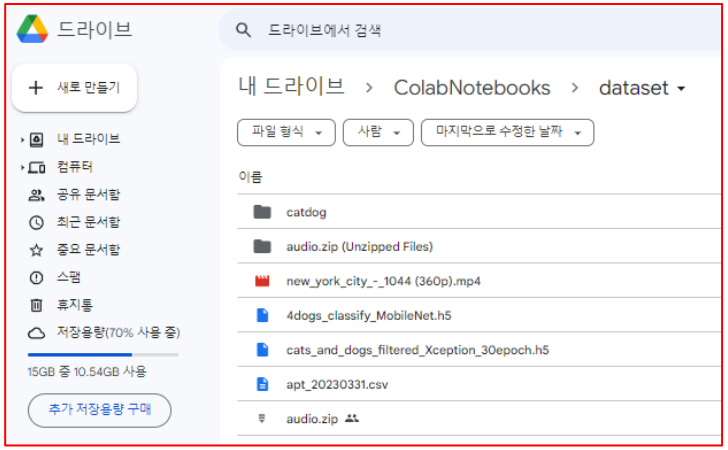
train.py : 지정된 옵션으로 학습을 수행하고 학습 모델을 구축/저장 !!

https://docs.ultralytics.com/yolov5/tutorials/train_custom_data/#1-create-dataset

2. Deep Learning(YOLO 5) 테스트 준비사항

1. 깃허브에서 YOLO5 필수 설치 파일 복사 준비
2. 객체 탐지를 시도할 동영상 등 준비
3. 구글드라이브에 업로드하여 위치 확인
4. 코랩의 환경설정





https://docs.ultralytics.com/yolov5/tutorials/train_custom_data/#1-create-dataset

2. Deep Learning(YOLO 5) pretrained model

```
 1 !pwd
```

```
/content
```

```
 1 !git clone https://github.com/ultralytics/yolov5.git
```

```
Cloning into 'yolov5'...
remote: Enumerating objects: 15705, done.
remote: Counting objects: 100% (33/33), done.
remote: Compressing objects: 100% (30/30), done.
remote: Total 15705 (delta 9), reused 15 (delta 3), pack-reused 15672
Receiving objects: 100% (15705/15705), 14.51 MiB | 30.01 MiB/s, done.
Resolving deltas: 100% (10754/10754), done.
```

```
 %cd yolov5
```

```
/content/yolov5
```

!pwd : print working directory
현재 작업 디렉토리를 출력하는 명령어. 현재 실행 중인 주피터 노트북이나 코드의 위치.
코랩의 기본 작업 디렉토리는 /content로 설정되어 있다.
cd와 같은 명령어를 사용하여 다른 디렉토리로 이동할 수 있다.

%cd : change directory
작업 디렉토리를 변경하는 명령어

https://github.com/ultralytics/yolov5

2. Deep Learning(YOLO 5) pretrained model

https://github.com/ultralytics/yolov5

https://github.com/ultralytics/yolov5.git

```
[6]  1 !git clone https://github.com/ultralytics/yolov5.git

     Cloning into 'yolov5'...
     remote: Enumerating objects: 16008, done.
     remote: Total 16008 (delta 0), reused 0 (delta 0), pack-reused 16008
     Receiving objects: 100% (16008/16008), 14.51 MiB | 27.98 MiB/s, done.
     Resolving deltas: 100% (11012/11012), done.

[7]  1 %cd yolov5

     /content/yolov5/yolov5

[10] 1 !pip install -r requirements.txt

     Collecting gitpython>=3.1.30 (from -r requirements.txt (line 5))
       Downloading GitPython-3.1.31-py3-none-any.whl (184 kB)
```

Yolo의 깃허브 사이트로 이동하여 테스트에 필요한 환경을 설정한다.

2. Deep Learning(YOLO 5) pretrained model

!pip install -r requirements.txt
pip를 사용하여 Python 패키지를 설치 . -r 옵션은 requirements 파일을 사용하여 여러 패키지를 한 번에 설치.
requirements.txt 파일에 명시된 각 패키지와 버전이 자동으로 설치됩니다. 설치된 패키지는 현재 환경에 추가되어 해당 프로젝트에서 사용할 수 있게 됩니다.

!git clone https://github.com/ultralytics/yolov5.git
Git을 사용하여 특정 GitHub 저장소를 복사
이 명령을 실행하면 yolov5라는 이름의 디렉토리가 생성되고, 해당 디렉토리에는 클론된 저장소의 모든 파일과 폴더가 포함.

https://github.com/ultralytics/yolov5
https://pixabay.com/

2. Deep Learning(YOLO 5) pretrained model

```
1 !python detect.py --weights yolov5s.pt --source /content/drive/MyDrive/ColabNotebooks/dataset/new_york_city.mp4

video 1/1 (30/349) /content/drive/MyDrive/ColabNotebooks/dataset/new_york_city.mp4: 384x640 17 persons, 4 cars, 1 handbag,
video 1/1 (31/349) /content/drive/MyDrive/ColabNotebooks/dataset/new_york_city.mp4: 384x640 16 persons, 4 cars, 1 truck, 1
video 1/1 (32/349) /content/drive/MyDrive/ColabNotebooks/dataset/new_york_city.mp4: 384x640 15 persons, 4 cars, 1 truck, 1
video 1/1 (33/349) /content/drive/MyDrive/ColabNotebooks/dataset/new_york_city.mp4: 384x640 17 persons, 4 cars, 1 handbag,
```

Pixabay 등의 사이트에서 이미지를 다운받아 구글드라이브에 업로드

주어진 코드로 해당이미지의 객체 인식을 테스트 한다.

이미지는 프레임별로 객체 인식을 진행하여 특정위치에 저장된다.

Detect.py – weights yolov5s.pt를 실행시키면 yolov5s.pt가 생성된다.

```
!python detect.py --weights yolov5s.pt --source /content/new_york_city_360p.mp4
```

https://docs.ultralytics.com/yolov5/tutorials/train_custom_data/#1-create-dataset

2. Deep Learning(YOLO 5) pretrained model

!python detect.py --weights yolov5s.pt --source /content/new_york_city_360p.mp4

YOLOv5의 detect.py 스크립트를 사용하여 동영상을 검출.
해당 명령어를 실행하려면 yolov5s.pt와 new_york_city_360p.mp4 파일이 존재해야 .

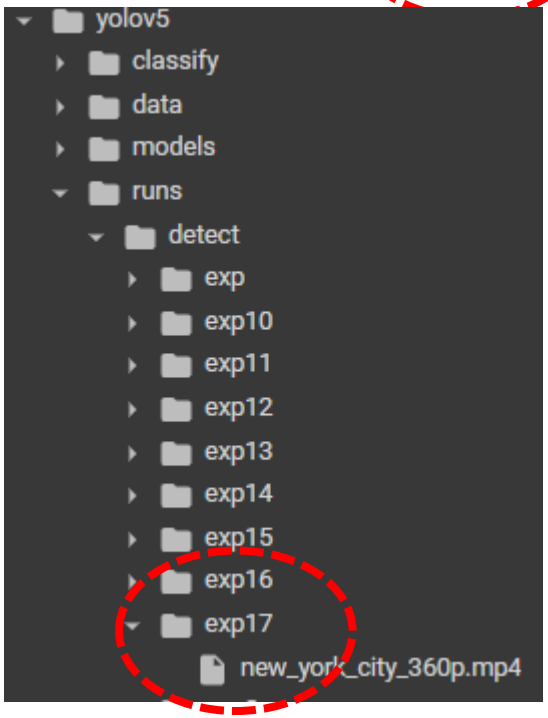
--weights yolov5s.pt: 가중치 파일을 지정. yolov5s.pt는 사전 훈련된 가중치 파일.

--source /content/new_york_city_360p.mp4: 입력 동영상 파일의 경로를 지정.

https://docs.ultralytics.com/yolov5/tutorials/train_custom_data/#1-create-dataset

2. Deep Learning(YOLO 5) pretrained model

```
video 1/1 (346/349) /content/new_york_city_360p.mp4: 384x640 12 persons, 6 cars, 1 skateboard, 5.6ms
video 1/1 (347/349) /content/new_york_city_360p.mp4: 384x640 14 persons, 6 cars, 1 skateboard, 5.6ms
video 1/1 (348/349) /content/new_york_city_360p.mp4: 384x640 15 persons, 7 cars, 5.6ms
video 1/1 (349/349) /content/new_york_city_360p.mp4: 384x640 13 persons, 6 cars, 5.6ms
Speed: 0.4ms pre-process, 6.7ms inference, 1.0ms NMS per image at shape (1, 3, 640, 640)
Results saved to runs/detect/exp17
```



사전학습된 모델(yolov5s.pt)을 사용하여 동영상 내의 사물을 탐지하면 결과를 저장해 준다

https://docs.ultralytics.com/yolov5/tutorials/train_custom_data/#1-create-dataset

2. Deep Learning(YOLO 5)

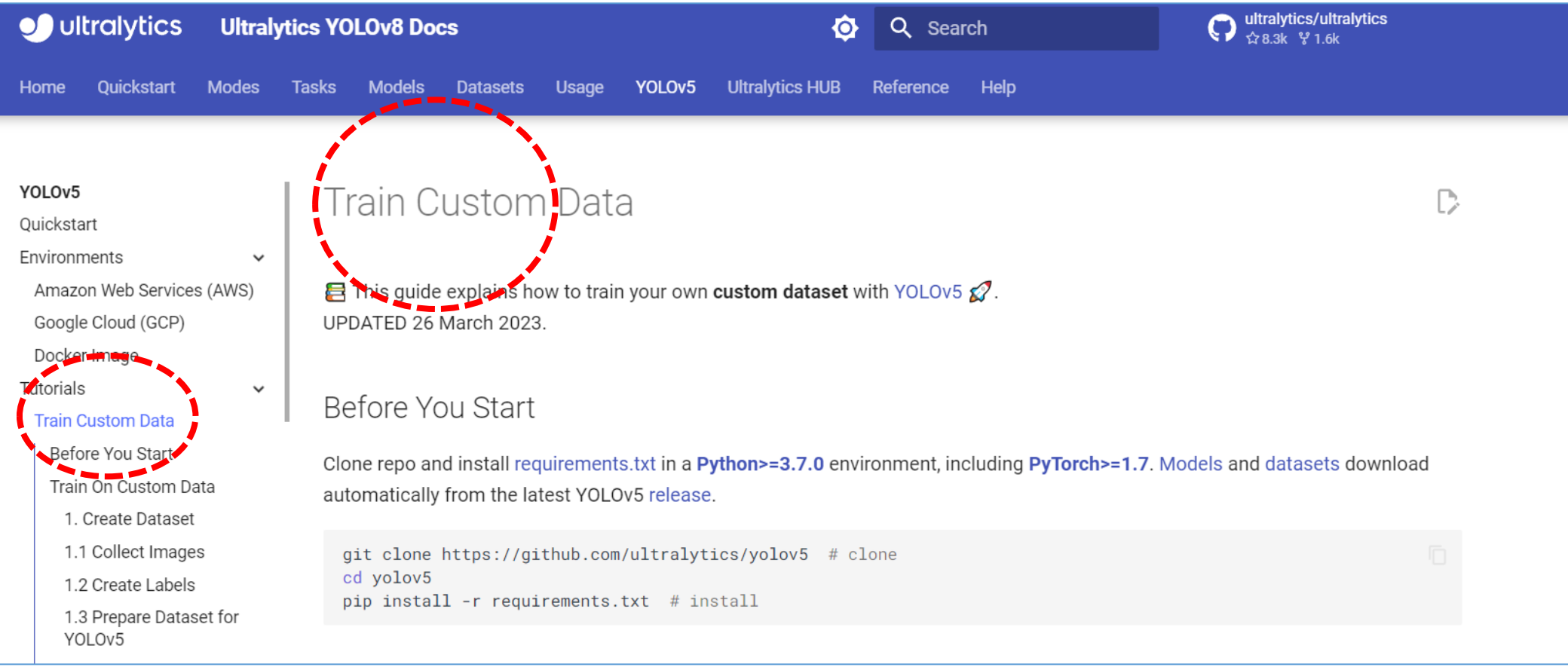
YOLO 5

custom data

작성 방법

https://docs.ultralytics.com/yolov5/tutorials/train_custom_data/#1-create-dataset

2. Deep Learning(YOLO 5)custom data



커스텀 데이터 학습 및 예측

2. Deep Learning(YOLO 5)custom data

Use Roboflow to create your dataset in YOLO format

Or manually prepare your dataset

1.1 Create dataset.yaml

COCO128 is an example small tutorial dataset composed of the first 128 images in COCO train2017. These same 128 images are used for both training and validation to verify our training pipeline is capable of overfitting. data/coco128.yaml, shown below, is the dataset config file that defines 1) the dataset root directory `path` and relative paths to `train` / `val` / `test` image directories (or *.txt files with image paths) and 2) a class `names` dictionary:

```yaml
# Train/val/test sets as 1) dir: path/to/imgs, 2) file: path/to/imgs.txt, or 3) list: [path/to/imgs1, path/to/imgs2, ..]
path: ../datasets/coco128  # dataset root dir
train: images/train2017  # train images (relative to 'path') 128 images
val: images/train2017  # val images (relative to 'path') 128 images
test:  # test images (optional)

# Classes (80 COCO classes)
names:
  0: person
  1: bicycle
  2: car
  ...
  77: teddy bear
  78: hair drier
  79: toothbrush
```

1. **Roboflow에서 데이터 셋을 직접 생성하거나**
2. **COCO128(80개 클라스) 등의 샘플 데이터 셋을 사용 가능**
   (yaml화일에 데이터셋의 경로를 포함하고 있음에 유의 !!)

2. Deep Learning(YOLO 5)custom data

1.2 Create Labels

After using an annotation tool to label your images, export your labels to **YOLO format**, with one `*.txt` file per image (if no objects in image, no `*.txt` file is required). The `*.txt` file specifications are:

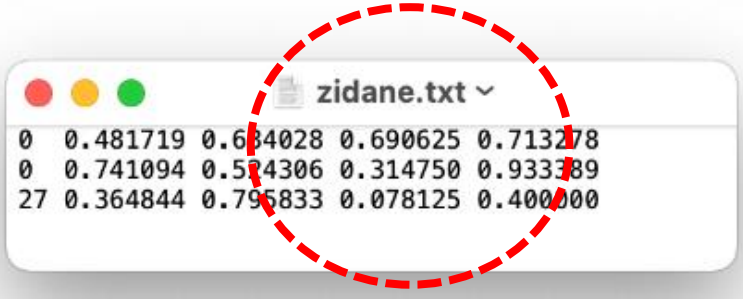
- One row per object
- Each row is `class x_center y_center width height` format.
- Box coordinates must be in **normalized xywh** format (from 0 - 1). If your boxes are in pixels, divide `x_center` and `width` by image width, and `y_center` and `height` by image height.
- Class numbers are zero-indexed (start from 0).

준비된 이미지 데이터의 라벨 데이터(*.txt) 는 다음의 YOLO format으로 준비

- 객체별 한줄
- 각 줄별로 x_center, y_center, width, height
- nomarlized xywh format(from 0 – 1)
(x축은 이미지 폭으로 나누고 y 축은 이미지 높이로 나눈다)
- 클래스의 번호는 0부터 index

2. Deep Learning(YOLO 5)custom data

The label file corresponding to the above image contains 2 persons (class `0`) and a tie (class `27`):



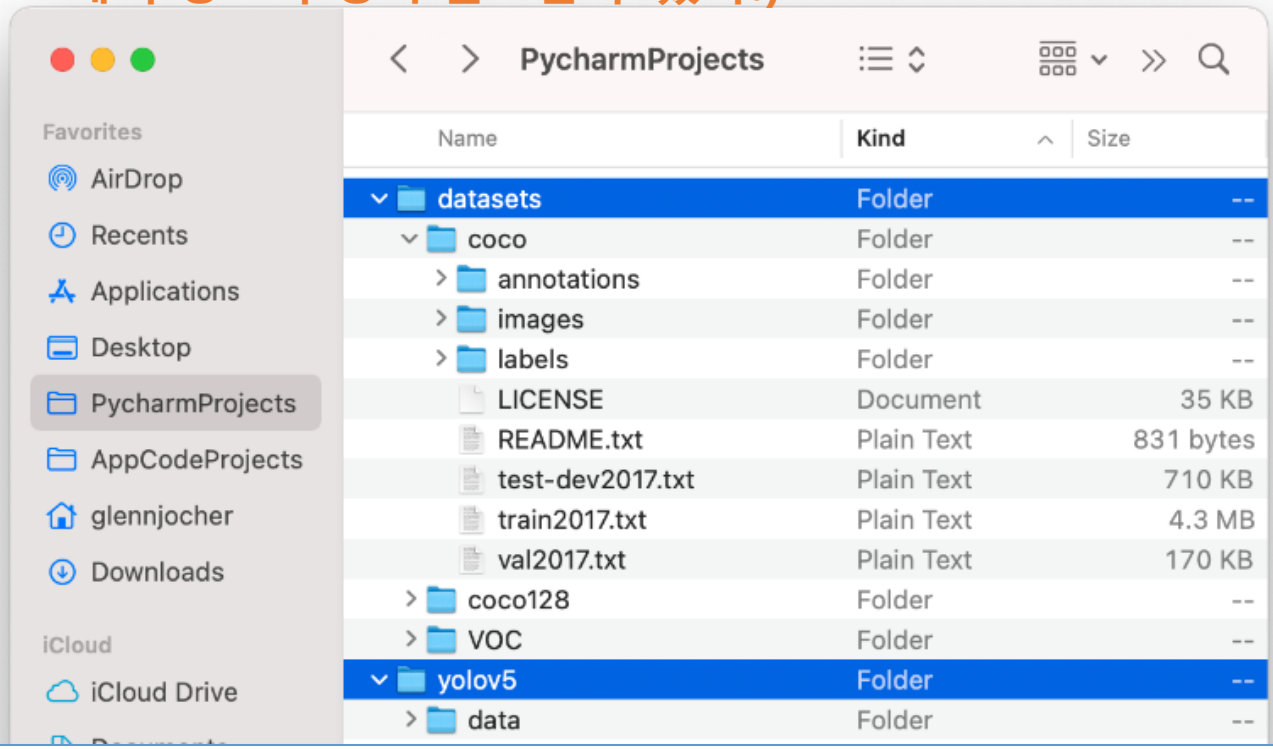
2. Deep Learning(YOLO 5)custom data

1.3 Organize Directories

Organize your train and val images and labels according to the example below. YOLOv5 assumes `/coco128` is inside a `/datasets` directory **next to** the `/yolov5` directory. **YOLOv5 locates labels automatically for each image** by replacing the last instance of `/images/` in each image path with `/labels/`. For example:

```
../datasets/coco128/images/im0.jpg  # image
../datasets/coco128/labels/im0.txt  # label
```

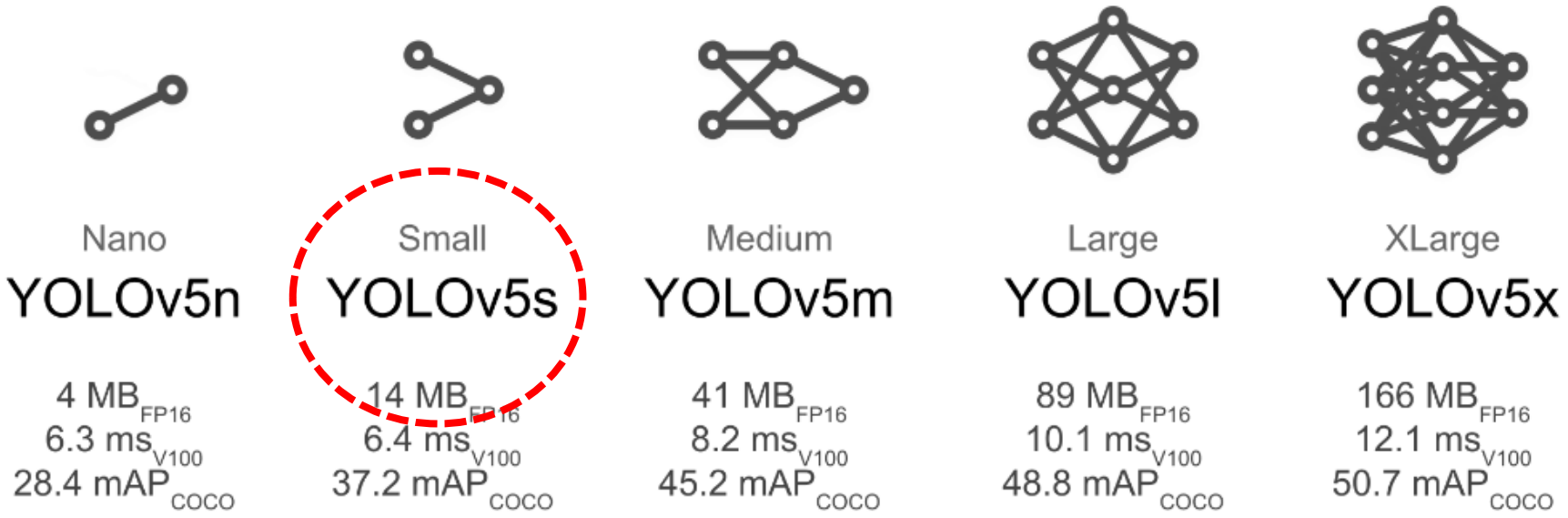
YOLO5의 디렉토리와 같은 레벨의 디렉토리에 데이터 셋을 위치 시킨다.
(*.yaml 에서 경로 수정이 필요할 수 있다.)



2. Deep Learning(YOLO 5)custom data

2. Select a Model

Select a pretrained model to start training from. Here we select YOLOv5s, the second-smallest and fastest model available. See our README table for a full comparison of all models.



학습을 위해 사용할 사전학습된 모델을 선택(예: YOLOv5s.pt)

2. Deep Learning(YOLO 5)custom data

Pretrained Checkpoints

Model	size (pixels)	mAP^{val} 50-95	mAP^{val} 50	Speed CPU b1 (ms)	Speed V100 b1 (ms)	Speed V100 b32 (ms)	params (M)	FLOPs @640 (B)
YOLOv5n	640	28.0	45.7	**45**	**6.3**	**0.6**	**1.9**	**4.5**
YOLOv5s	640	37.4	56.8	98	6.4	0.9	7.2	16.5
YOLOv5m	640	45.4	64.1	224	8.2	1.7	21.2	49.0
YOLOv5l	640	49.0	67.3	430	10.1	2.7	46.5	109.1
YOLOv5x	640	50.7	68.9	766	12.1	4.8	86.7	205.7
YOLOv5n6	1280	36.0	54.4	153	8.1	2.1	3.2	4.6
YOLOv5s6	1280	44.8	63.7	385	8.2	3.6	12.6	16.8
YOLOv5m6	1280	51.3	69.3	887	11.1	6.8	35.7	50.0
YOLOv5l6	1280	53.7	71.3	1784	15.8	10.5	76.8	111.4
YOLOv5x6 + TTA	1280 1536	55.0 **55.8**	72.7 **72.7**	3136 -	26.2 -	19.4 -	140.7 -	209.8 -

2. Deep Learning(YOLO 5)custom data

3. Train

Train a YOLOv5s model on COCO128 by specifying dataset, batch-size, image size and either pretrained `--weights yolov5s.pt` (recommended), or randomly initialized `--weights '' --cfg yolov5s.yaml` (not recommended). Pretrained weights are auto-downloaded from the latest YOLOv5 release.

```
python train.py --img 640 --epochs 3 --data coco128.yaml --weights yolov5s.pt
```

데이터 셋, 배치 사이즈, 이미지 사이즈 등을 정한 후
준비된 데이터에 대해 선택된 모델로 학습을 시작한다.

```
!python train.py --img 416 --batch 16 --epochs 50 --data /content/data.yaml --cfg ./models/yolov5s.yaml --weights yolov5s.pt --name aquafish_yolov5s_results
```

2. Deep Learning(YOLO 5)custom data

```
!python train.py --img 416 --batch 16 --epochs 50 --data
/content/data.yaml --cfg ./models/yolov5s.yaml --weights yolov5s.pt --
name aquafish_yolov5s_results
```

1.!python train.py: Python 스크립트 **train.py**를 실행. YOLOv5 모델의 훈련을 담당.
2.--img 416: 입력 이미지의 크기를 416x416 픽셀로 설정.
YOLOv5 모델은 정사각형 이미지를 사용하며, 입력 이미지의 크기를 지정.
3.--batch 16: 배치 크기를 16으로 설정.
배치 크기는 한 번에 처리되는 이미지의 개수를 의미, GPU 메모리에 맞게 조정.
4.--epochs 50: 에포크는 전체 데이터셋에 대해 한 번의 전방향 패스와 역전파를 수행하는 단위.
5.--data /content/data.yaml: 데이터 구성을 정의하는 YAML 파일의 경로를 지정.
데이터셋의 경로, 클래스 정보, 훈련 및 검증 데이터의 비율 등이 포함.
6.--cfg ./models/yolov5s.yaml: 모델 구성 파일의 경로를 지정.
YOLOv5 모델의 아키텍처와 설정을 정의하는 YAML 파일.
cfg는 "configuration", 모델의 아키텍처와 설정을 포함하는 YAML 파일을 의미
7.--weights yolov5s.pt: 사전 훈련된 가중치 파일의 경로를 지정.
8.--name aquafish_yolov5s_results: 훈련 결과를 저장할 디렉토리 이름을 지정.

2. Deep Learning(YOLO 5)custom data

4. Visualize

Comet Logging and Visualization 🌟 NEW

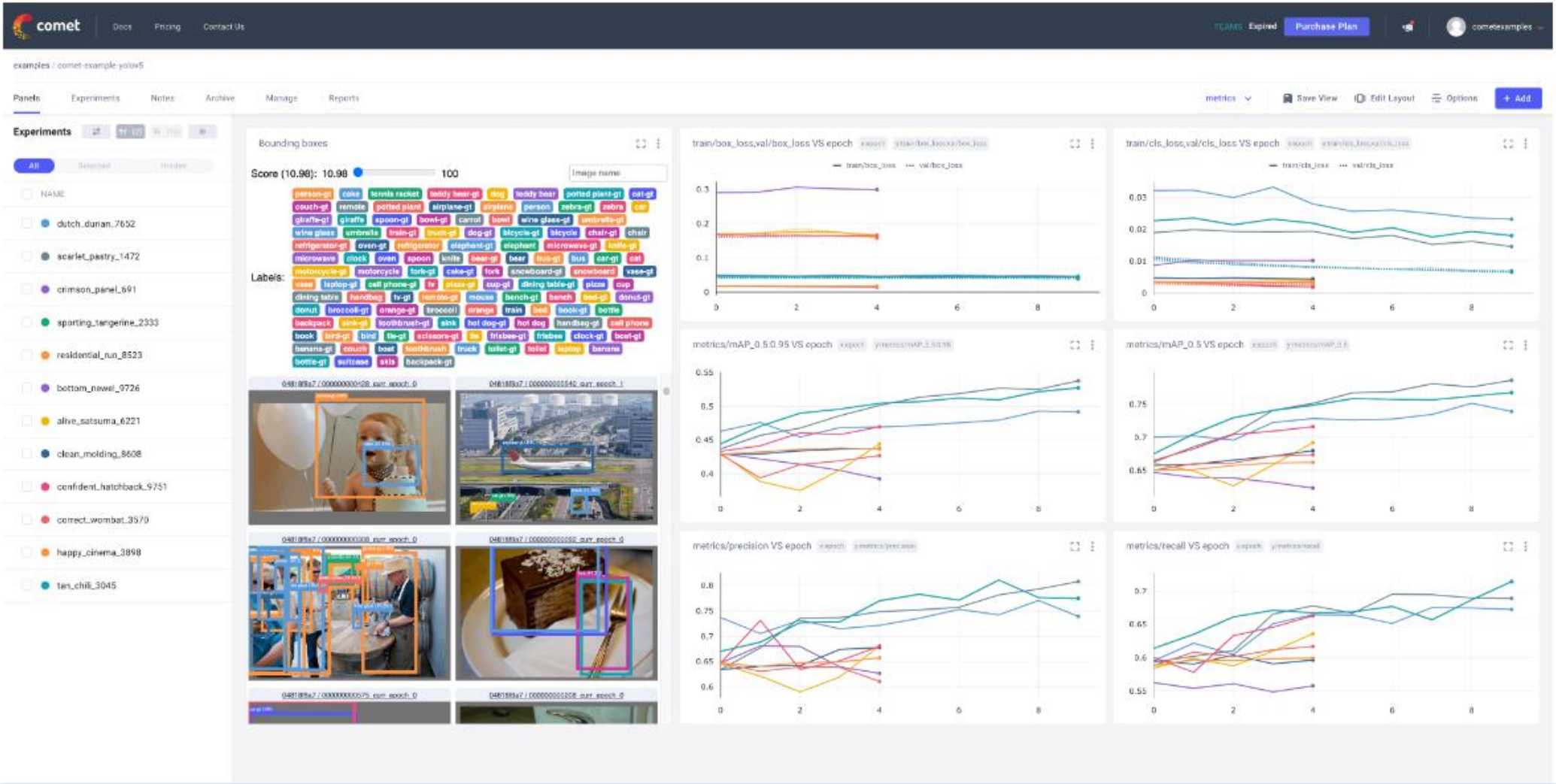
Comet is now fully integrated with YOLOv5. Track and visualize model metrics in real time, save your hyperparameters, datasets, and model checkpoints, and visualize your model predictions with Comet Custom Panels! Comet makes sure you never lose track of your work and makes it easy to share results and collaborate across teams of all sizes!

Getting started is easy:

```
pip install comet_ml  # 1. install
export COMET_API_KEY=<Your API Key>  # 2. paste API key
python train.py --img 640 --epochs 3 --data coco128.yaml --weights yolov5s.pt  # 3. train
```

학습된 데이터의 예측결과 등을 분석 및 시각화(Comet Custom Panes)

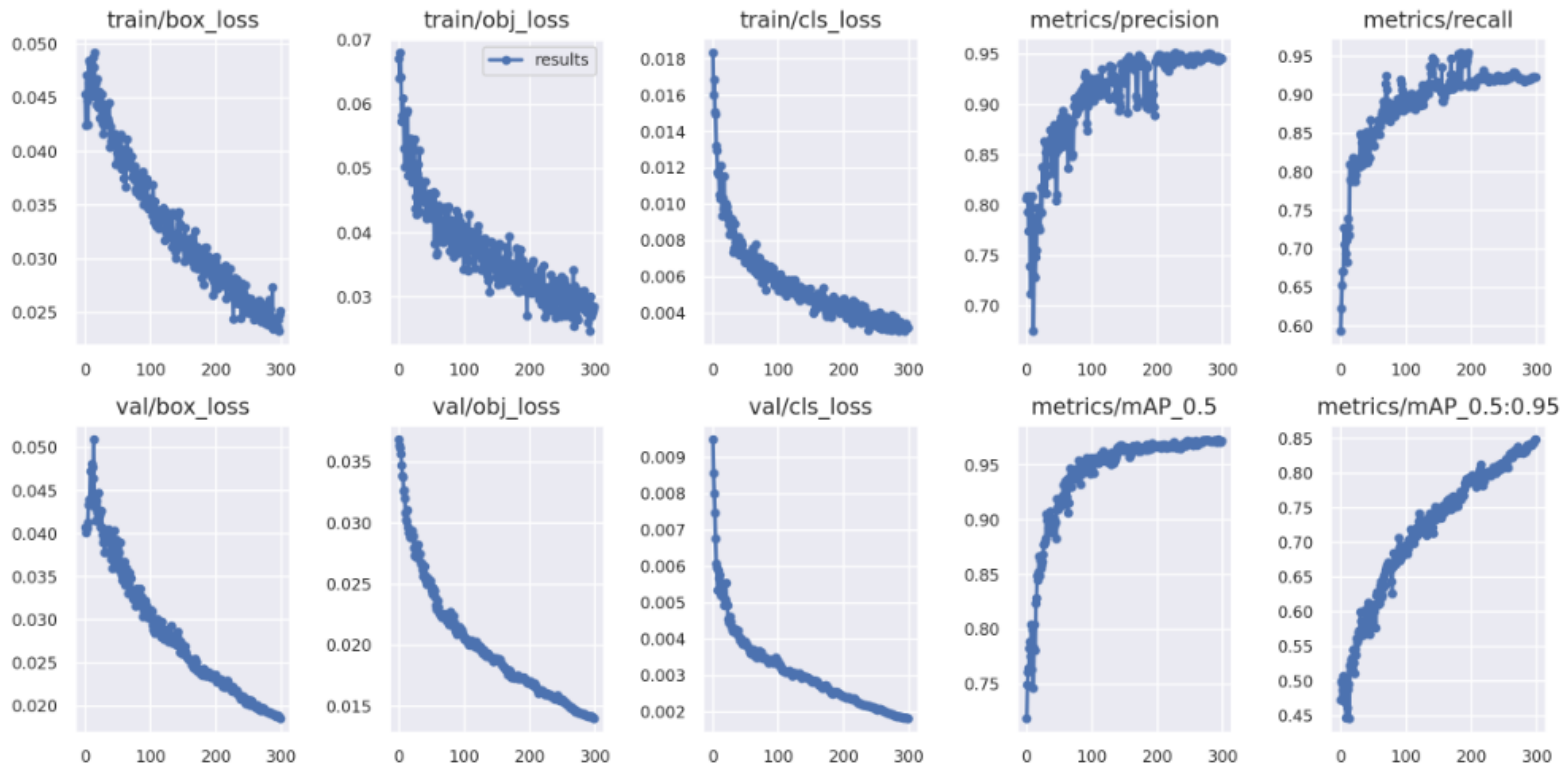
2. Deep Learning(YOLO 5)custom data



2. Deep Learning(YOLO 5)custom data

Results file `results.csv` is updated after each epoch, and then plotted as `results.png` (below) after training completes. You can also plot any `results.csv` file manually:

```
from utils.plots import plot_results
plot_results('path/to/results.csv')  # plot 'results.csv' as 'results.png'
```



2. Deep Learning(YOLO 5)custom data

Next Steps

Once your model is trained you can use your best checkpoint `best.pt` to:

- Run CLI or Python inference on new images and videos
- Validate accuracy on train, val and test splits
- Export to TensorFlow, Keras, ONNX, TFlite, TF.js, CoreML and TensorRT formats
- Evolve hyperparameters to improve performance
- Improve your model by sampling real-world images and adding them to your dataset

2. Deep Learning(YOLO 5)custom data

Custom data

실습 예제

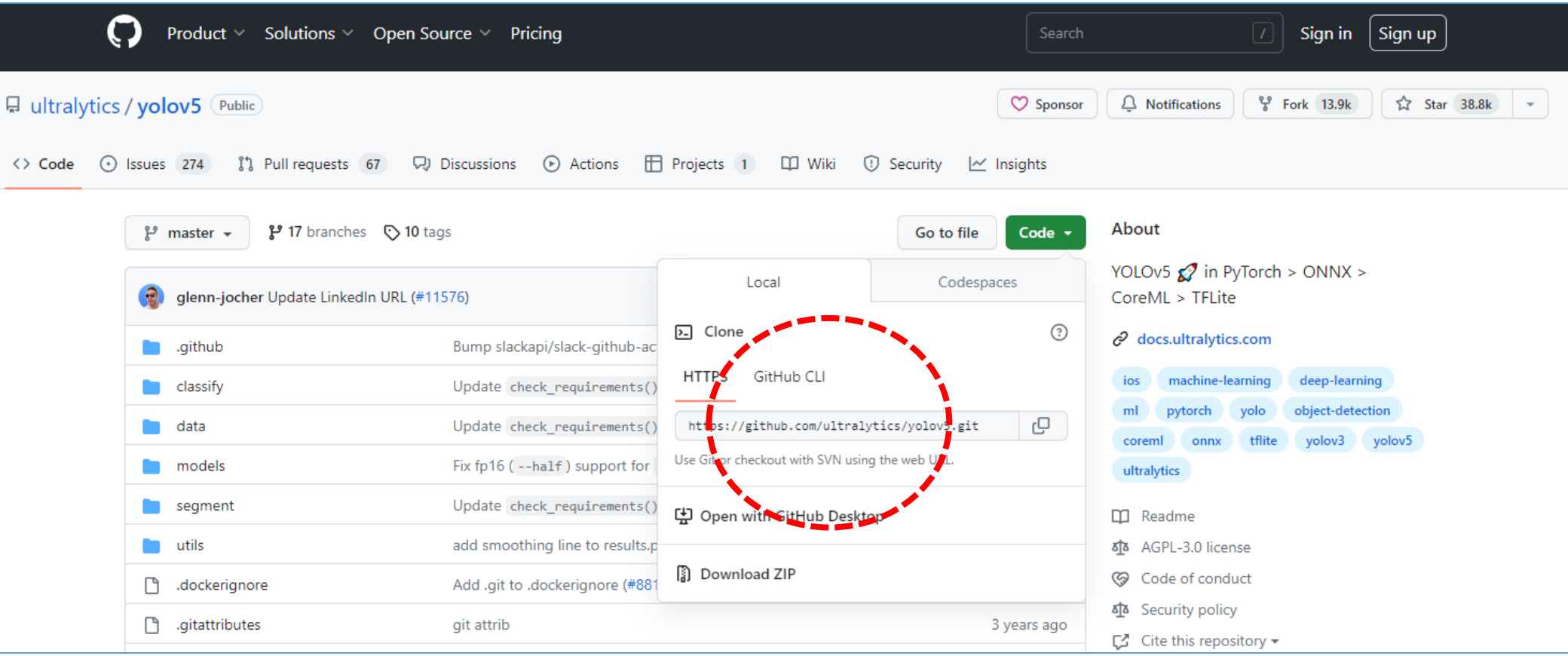
2. Deep Learning(YOLO 5)custom data

Synthetic Fruits Data set

(YOLO 5)

https://colab.research.google.com/drive/1VLad4gz__PAKl-8_BzhJw-0Ju71KqBWZ?hl=ko#scrollTo=B0nduCT2_DR4&uniqifier=1

2. Deep Learning(YOLO 5)custom data



https://colab.research.google.com/drive/1VLad4gz__PAKl-8_BzhJw-0Ju71KqBWZ?hl=ko#scrollTo=B0nduCT2_DR4&uniqifier=1

2. Deep Learning(YOLO 5)custom data

```
[6]  1 !pwd
```

```
!pwd
```

```
/content
```

```
!git clone https://github.com/ultralytics/yolov5.git
```

```
[7]  1 !git clone https://github.com/ultralytics/yolov5.git
```

```
Cloning into 'yolov5'...
remote: Enumerating objects: 15705, done.
remote: Counting objects: 100% (33/33), done.
remote: Compressing objects: 100% (27/27), done.
remote: Total 15705 (delta 9), reused 23 (delta 6), pack-reused 15672
Receiving objects: 100% (15705/15705), 14.44 MiB | 30.06 MiB/s, done.
Resolving deltas: 100% (10755/10755), done.
```

```
1 %cd yolov5
```

```
%cd yolov5
```

```
/content/yolov5
```

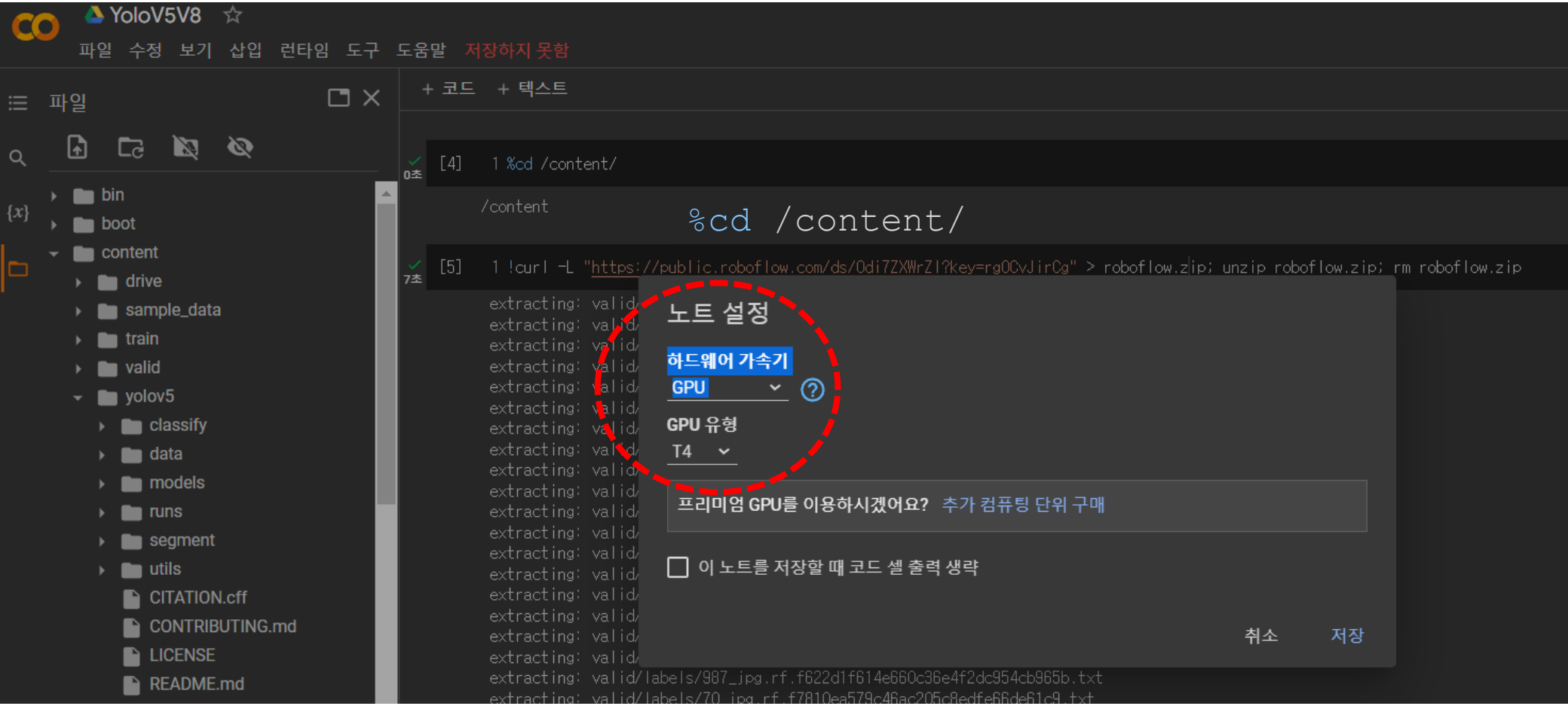
```
!pip install -r requirements.txt
```

```
[9]  1 !pip install -r requirements.txt
```

현재 디렉토리 위치와 환경설정 코드 재확인 !

https://www.youtube.com/watch?v=U-wkRQ8U3GE

2. Deep Learning(YOLO 5)custom data

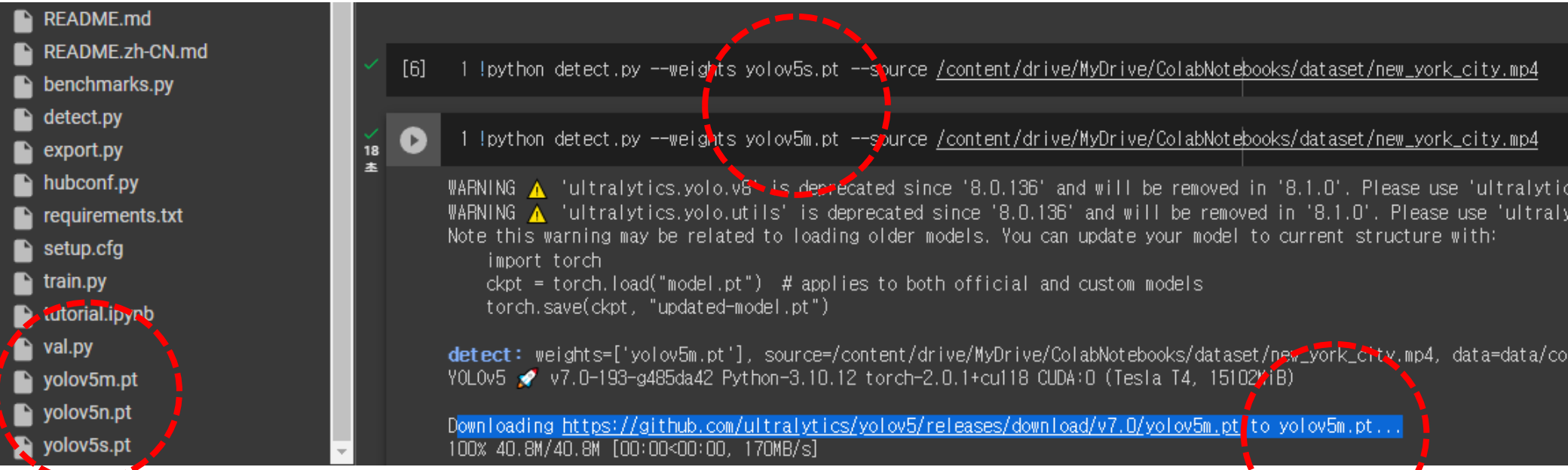


!curl -L "https://public.roboflow.com/ds/0di7ZXWrZI?key=ㅌㅌㅌㅌㅌㅌ" > roboflow.zip; unzip roboflow.zip; rm roboflow.zip

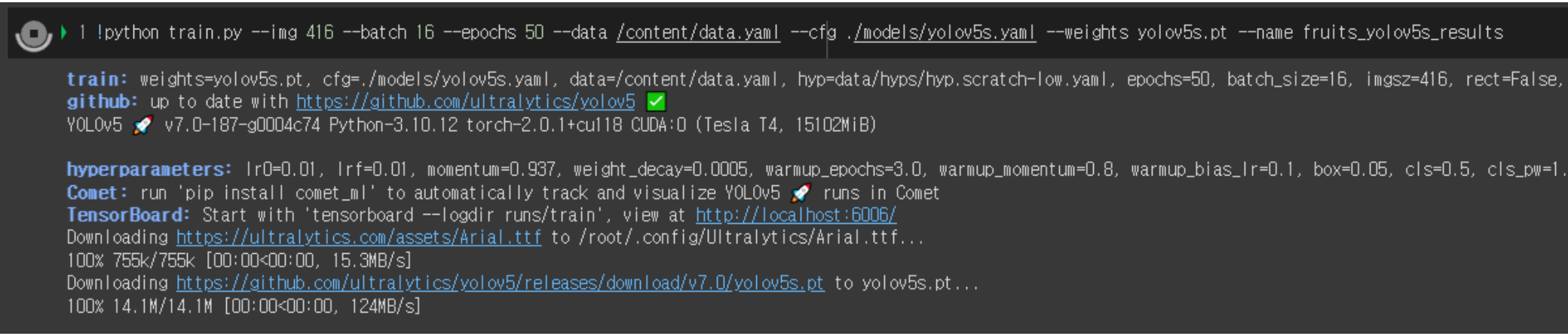
코랩의 현재 위치한 디렉토리를 확인 / GPU 설정 확인!!
Public Roboflow사이트에서 데이터 코드를 복사하여설치해 준다.

2. Deep Learning(YOLO 5)custom data

가중치 파일(yolov5s.pt 등)이 생성이 않되었을 때는 detect.py를 실행해 본다. !!
(train.py를 실행시에도 탐지 후 다운로드를 받아야 정상이나 실행이 않되는 경우 yaml 파일 등의 경로를 현재위치를 근거로 재설정한다. /content/yolo5에 있는 경우 그 이후의 경로만 지정해 본다.!!, 가중치 파일은 자동인식하여 깃허브사이트에서 복사해 오는 과정을 거친다.)



2. Deep Learning(YOLO 5)custom data




```
!python train.py --img 416 --batch 16 --epochs 50 --data /content/data.yaml
--cfg ./models/yolov5s.yaml --weights yolov5s.pt --name fruit_yolov5s_results
```

명령어의 실행시 코드의 위치와 데이터의 위치에 주목한다.

현재 디렉토리를 /content/yolov5로 설정 후 명령어 코드 위치 확인!!
데이터셋의 위치 확인

2. Deep Learning(YOLO 5)custom data

```
Validating runs/train/fruits_yolov5s_results/weights/best.pt...
Fusing layers...
YOLOv5s summary: 157 layers, 7180036 parameters, 0 gradients, 16.3 GFLOPs
                 Class     Images  Instances          P          R      mAP50   mAP50-95: 100% 32/32 [00:15<00:00,  2.04it/s]
                   all       1000       2803      0.945      0.898      0.944      0.804
                 Apple       1000         42      0.626      0.643      0.704      0.631
               Apricot       1000         43          1      0.934      0.995      0.886
               Avocado       1000         54      0.993      0.981      0.985       0.85
                Banana       1000         54      0.953      0.648      0.783      0.545
              Beetroot       1000         43      0.931      0.953      0.982       0.77
             Blueberry       1000         34      0.956      0.941      0.973      0.924
                Cactus       1000         54      0.907      0.944      0.957       0.84
            Cantaloupe       1000         49      0.979      0.972      0.975      0.896
              Carambula       1000         34          1      0.951      0.992      0.843
            Cauliflower       1000         54       0.94      0.981      0.992      0.803
                Cherry       1000         46      0.861      0.761      0.894       0.82
              Chestnut       1000         37      0.914      0.838      0.946      0.777
            Clementine       1000         47      0.978      0.958      0.993       0.86
                 Cocos       1000         34      0.898      0.912      0.953      0.829
                 Dates       1000         44      0.905      0.955      0.973      0.821
              Eggplant       1000         53      0.925      0.698       0.83      0.661
                Ginger       1000         53          1      0.879      0.975      0.752
             Granadilla       1000         46      0.973      0.978      0.959      0.585
                 Grape       1000         42          1       0.83      0.965      0.883
            Strawberry       1000         44      0.952      0.932       0.94      0.734
               Tamarillo       1000         34      0.991      0.971       0.98      0.749
               Tangelo       1000         39       0.96      0.974      0.992      0.887
                Tomato       1000         43      0.945       0.93      0.967      0.894
                Walnut       1000         51       0.93          1      0.995       0.92
Results saved to runs/train/fruits_yolov5s_results
```

학습 후 특정위치에 저장(runs/train/...)

2. Deep Learning(YOLO 5)

```
50 epochs completed in 1.572 hours.
Optimizer stripped from runs/train/fruit_yolov5s_results/weights/last.pt, 14.6MB
Optimizer stripped from runs/train/fruit_yolov5s_results/weights/best.pt, 14.6MB

Validating runs/train/fruit_yolov5s_results/weights/best.pt...
Fusing layers...
YOLOv5s summary: 157 layers, 7180036 parameters, 0 gradients, 16.3 GFLOPs
                 Class     Images  Instances          P          R      mAP50   mAP50-95: 100% 32/32 [00:14<00:00,  2.16it/s]
                   all       1000       2803      0.945      0.898      0.944      0.804
                 Apple       1000         42      0.626      0.643      0.704      0.631
               Apricot       1000         43          1      0.934      0.995      0.886
               Avocado       1000         54      0.993      0.981      0.985       0.85
                Banana       1000         54      0.953      0.648      0.783      0.545
              Beetroot       1000         43      0.931      0.953      0.982       0.77
             Blueberry       1000         34      0.956      0.941      0.973      0.924
                Cactus       1000         54      0.907      0.944      0.957       0.84
            Cantaloupe       1000         49      0.979      0.972      0.975      0.896
             Carambula       1000         34          1      0.951      0.992      0.843
           Cauliflower       1000         54       0.94      0.981      0.992      0.803
                Cherry       1000         46      0.861      0.761      0.894       0.82
              Chestnut       1000         37      0.914      0.838      0.946      0.777
            Clementine       1000         47      0.978      0.958      0.993       0.86
```

50 epoch에 1.5시간의 학습 시간이 소요
7,180,036 파라미터를 처리한 결과 클래스별 검증결과를 제시

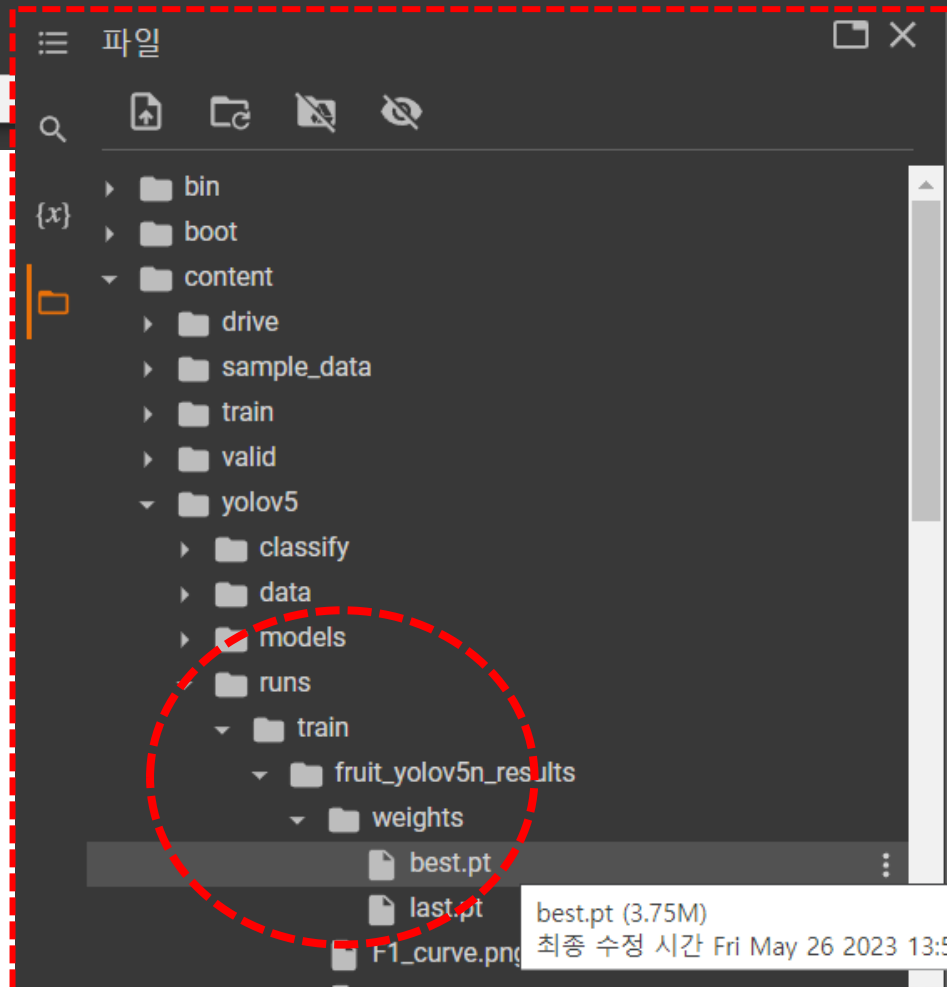
2. Deep Learning(YOLO 5)custom data

Tangelo	1000	39	0.96	0.974	0.992	0.887
Tomato	1000	43	0.945	0.93	0.967	0.894
Walnut	1000	51	0.93	1	0.995	0.92

Results saved to **runs/train/fruit_yolov5s_results**

학습을 완료 후
yolov5>runs>train>fruit_yolov5_results>weights 위치에

best.pt와 last.pt로 저장되어 있다



2. Deep Learning(YOLO 5)

```
1 !python detect.py --weights /content/yolov5/runs/train/fruits_yolov5s_results/weights/best.pt --img 416 --conf 0.5 --source /content/fruits.jpg

detect: weights=['/content/yolov5/runs/train/fruits_yolov5s_results/weights/best.pt'], source=/content/fruits.jpg, data=data/coco128.yaml, imgsz=[416, 416]
YOLOv5 🚀 v7.0-172-gc3c1304 Python-3.10.11 torch-2.0.1+cu118 CUDA:0 (Tesla T4, 15102MiB)

Fusing layers...
YOLOv5s summary: 157 layers, 7180036 parameters, 0 gradients, 16.3 GFLOPs
image 1/1 /content/fruits.jpg: 288x416 (no detections), 41.1ms
Speed: 0.4ms pre-process, 41.1ms inference, 0.8ms NMS per image at shape (1, 3, 416, 416)
Results saved to runs/detect/exp7
```

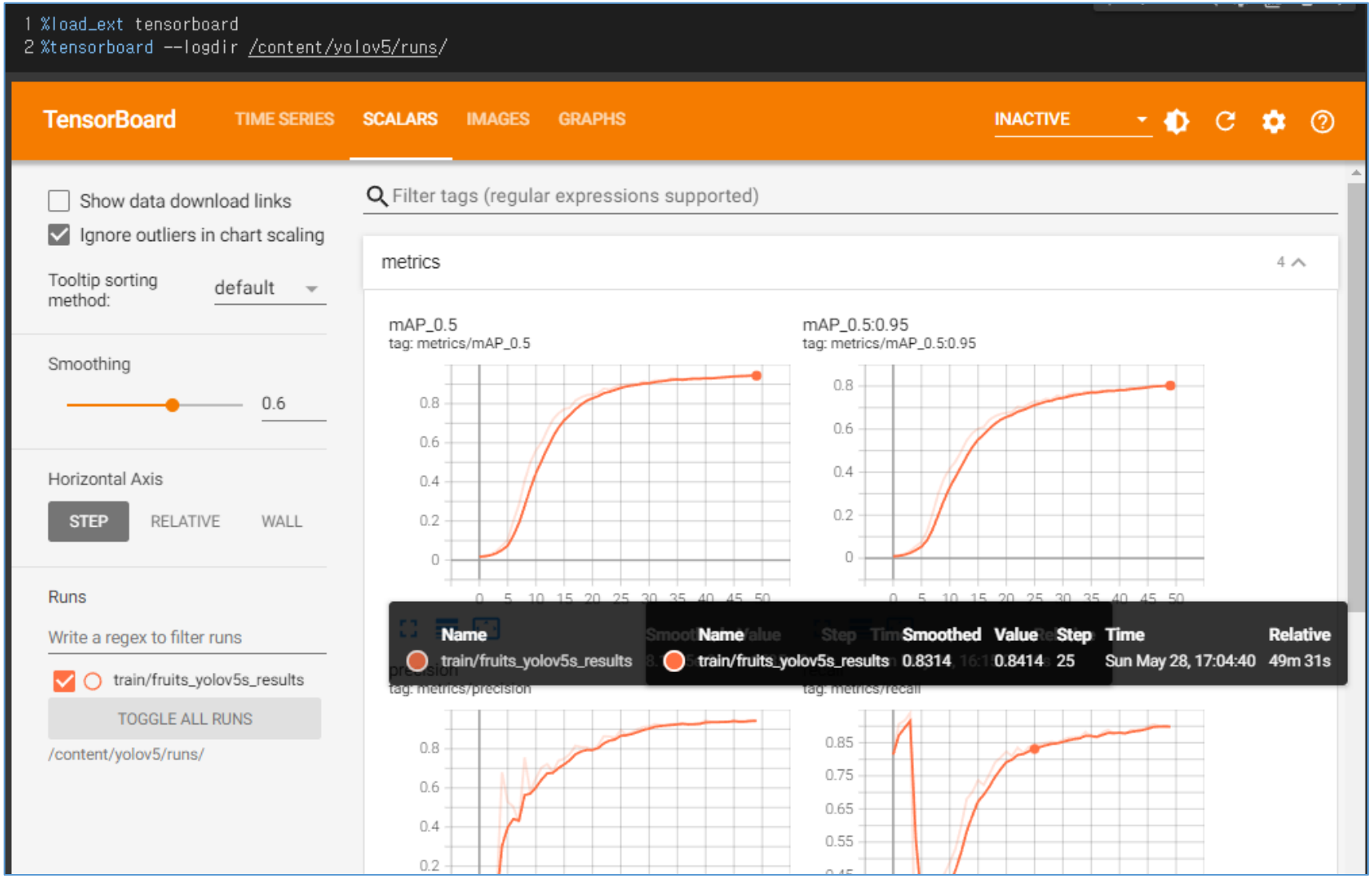
detect.py: YOLOv5의 객체 탐지 스크립트 파일.

--weights /content/yolov5/runs/train/fruits_yolov5s_results/weights/best.pt: best.pt 파일은 가장 좋은 성능을 보이는 가중치.
--img 416: 입력 이미지의 크기를 416x416으로 설정
--conf 0.5: 신뢰도(confidence) 임계값보다 낮은 신뢰도를 가진 객체는 탐지 결과에서 제외.
--source /content/fruits.jpg: 객체 탐지를 수행할 이미지 파일의 경로를 지정.

위의 명령어는 학습된 모델 가중치인 best.pt를 사용하여
416x416 크기의 이미지 fruits.jpg에서 객체 탐지를 수행. 탐지 결과 중
신뢰도가 0.5보다 높은 객체들을 반환

```
%load_ext tensorboard
%tensorboard --logdir /content/yolov5/runs/
```

2. Deep Learning(YOLO 5)



2. Deep Learning(YOLO 5) 샘플 테스트

입력 이미지가 지정한 이미지보다 큰 경우(1280*853)
리사이징되어 입력되고 예측의 결과가 달라질 수 있다

2. Deep Learning(YOLO 5)

이미지 사이즈를 **416으로** 지정한 경우 탐지된 객체가 없다

```
1 !python detect.py --weights /content/yolov5/runs/train/fruits_yolov5s_results/weights/best.pt --img 416 --conf 0.5 --source /content/fruits.jpg

detect: weights=['/content/yolov5/runs/train/fruits_yolov5s_results/weights/best.pt'], source=/content/fruits.jpg, data=data/coco128.yaml, imgsz=[416, 416]
YOLOv5 🚀 v7.0-172-gc3c1304 Python-3.10.11 torch-2.0.1+cu118 CUDA:0 (Tesla T4, 15102MiB)

Fusing layers...
YOLOv5s summary: 157 layers, 7180036 parameters, 0 gradients, 16.3 GFLOPs
image 1/1 /content/fruits.jpg: 288x416 (no detections), 41.1ms
Speed: 0.4ms pre-process, 41.1ms inference, 0.8ms NMS per image at shape (1, 3, 416, 416)
Results saved to runs/detect/exp7
```

2. Deep Learning(YOLO 5)

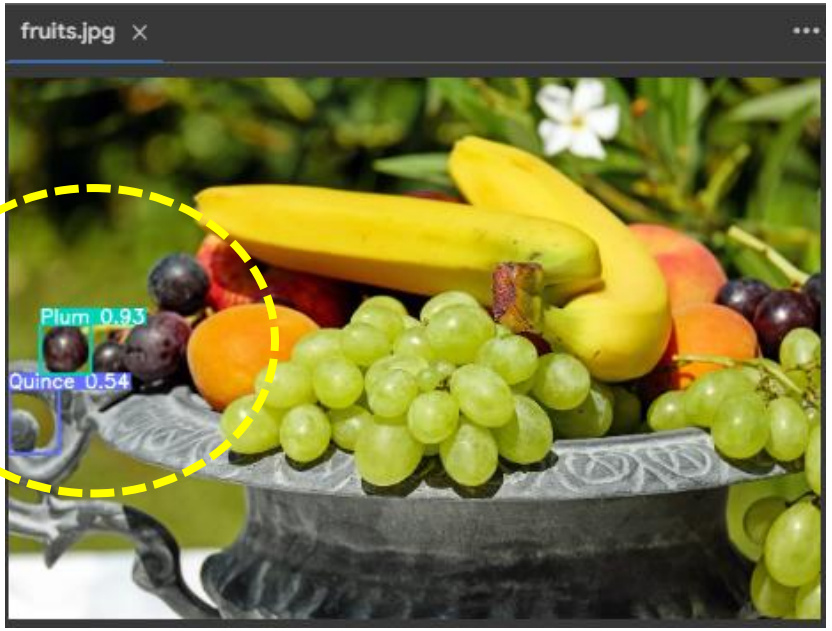
이미지 사이즈를 **850으로** 지정한 경우 탐지된 객체가 일부 있다

```
1 !python detect.py --weights /content/yolov5/runs/train/fruits_yolov5s_results/weights/best.pt --img 850 --conf 0.5 --source /content/fruits.jpg

detect: weights=['/content/yolov5/runs/train/fruits_yolov5s_results/weights/best.pt'], source=/content/fruits.jpg, data=data/coco128.yaml, imgsz=[8
YOLOv5 🚀 v7.0-172-gc3c1304 Python-3.10.11 torch-2.0.1+cu118 CUDA:0 (Tesla T4, 15102MiB)

Fusing layers...
YOLOv5s summary: 157 layers, 7180036 parameters, 0 gradients, 16.3 GFLOPs
WARNING ⚠️ --img-size [850, 850] must be multiple of max stride 32, updating to [864, 864]
image 1/1 /content/fruits.jpg: 576x864 1 Grape, 1 Mangostan, 1 Plum, 63.9ms
Speed: 0.7ms pre-process, 63.9ms inference, 1.8ms NMS per image at shape (1, 3, 864, 864)
Results saved to runs/detect/exp5
```

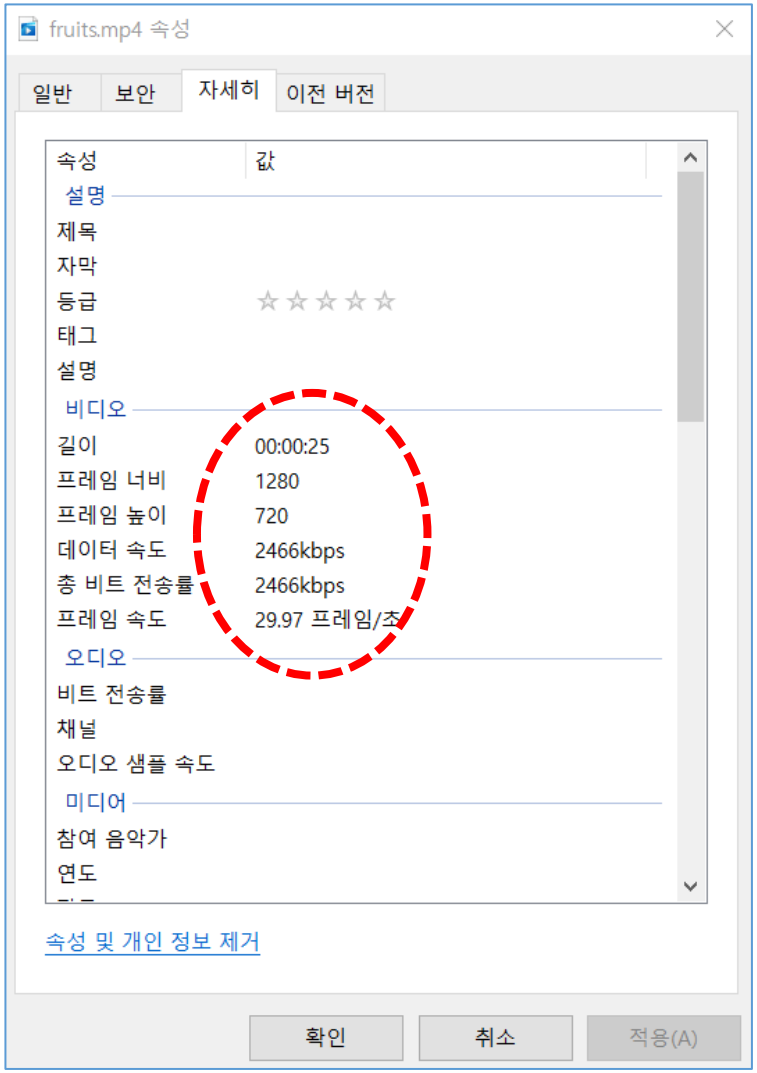
2. Deep Learning(YOLO 5)

이미지 사이즈를 **1,280으로** 지정한 경우 탐지된 객체가 일부 있다

```
1 !python detect.py --weights /content/yolov5/runs/train/fruits_yolov5s_results/weights/best.pt --img 1280 --conf 0.5 --source /content/fruits.jpg

detect: weights=['/content/yolov5/runs/train/fruits_yolov5s_results/weights/best.pt'], source=/content/fruits.jpg, data=data/coco128.yaml, imgsz=[1280, 1280]
YOLOv5 🚀 v7.0-172-gc3c1304 Python-3.10.11 torch-2.0.1+cu118 CUDA:0 (Tesla T4, 15102MiB)

Fusing layers...
YOLOv5s summary: 157 layers, 7180036 parameters, 0 gradients, 16.3 GFLOPs
image 1/1 /content/fruits.jpg: 864x1280 1 Plum, 1 Quince, 42.4ms
Speed: 1.1ms pre-process, 42.4ms inference, 1.5ms NMS per image at shape (1, 3, 1280, 1280)
Results saved to runs/detect/exp6
```



2. Deep Learning(YOLO 5)



입력 동영상의 사이즈가 **지정한 사이즈보다 큰 경우(1280*720) 리사이징되어 입력되고 예측의 결과가 달라질 수 있다**

2. Deep Learning(YOLO 5)

동영상 사이즈를 **416으로** 지정한 경우 탐지된 객체가 일부 있다

```
1 !python detect.py --weights /content/yolov5/runs/train/fruits_yolov5s_results/weights/best.pt --img 416 --conf 0.5 --source /content/fruits.mp4
video 1/1 (703/756) /content/fruits.mp4: 256x416 1 Plum, 6.6ms
video 1/1 (704/756) /content/fruits.mp4: 256x416 1 Plum, 7.1ms
video 1/1 (705/756) /content/fruits.mp4: 256x416 1 Plum, 6.3ms
video 1/1 (706/756) /content/fruits.mp4: 256x416 1 Plum, 6.3ms
video 1/1 (707/756) /content/fruits.mp4: 256x416 1 Plum, 8.4ms
video 1/1 (708/756) /content/fruits.mp4: 256x416 1 Plum, 7.4ms
video 1/1 (709/756) /content/fruits.mp4: 256x416 1 Plum, 6.5ms
video 1/1 (710/756) /content/fruits.mp4: 256x416 1 Plum, 6.6ms
video 1/1 (711/756) /content/fruits.mp4: 256x416 1 Plum, 7.4ms
video 1/1 (712/756) /content/fruits.mp4: 256x416 1 Plum, 7.3ms
```

2. Deep Learning(YOLO 5)

동영상 사이즈를 **720으로** 지정한 경우 탐지된 객체가 일부 있다

```
1 !python detect.py --weights /content/yolov5/runs/train/fruits_yolov5s_results/weights/best.pt --img 720 --conf 0.5 --source /content/fruits.mp4

detect: weights=['/content/yolov5/runs/train/fruits_yolov5s_results/weights/best.pt'], source=/content/fruits.mp4, data=data/coco128.yaml, imgsz=[7
YOLOv5 🚀 v7.0-172-gc3c1304 Python-3.10.11 torch-2.0.1+cu118 CUDA:0 (Tesla T4, 15102MiB)

Fusing layers...
YOLOv5s summary: 157 layers, 7180036 parameters, 0 gradients, 16.3 GFLOPs
WARNING ⚠️ --img-size [720, 720] must be multiple of max stride 32, updating to [736, 736]
video 1/1 (1/756) /content/fruits.mp4: 416x736 2 Plums, 54.3ms
video 1/1 (2/756) /content/fruits.mp4: 416x736 1 Plum, 9.5ms
video 1/1 (3/756) /content/fruits.mp4: 416x736 2 Plums, 9.5ms
video 1/1 (4/756) /content/fruits.mp4: 416x736 2 Plums, 9.5ms
video 1/1 (5/756) /content/fruits.mp4: 416x736 2 Plums, 9.5ms
video 1/1 (6/756) /content/fruits.mp4: 416x736 1 Plum, 9.5ms
video 1/1 (7/756) /content/fruits.mp4: 416x736 2 Plums, 9.5ms
video 1/1 (8/756) /content/fruits.mp4: 416x736 2 Plums, 9.5ms
```

2. Deep Learning(YOLO 5)

동영상 사이즈를 **360으로** 지정한 경우 탐지된 객체가 일부 있다

```
1 !python detect.py --weights /content/yolov5/runs/train/fruits_yolov5s_results/weights/best.pt --img 360 --conf 0.5 --source /content/fruits2_360.mp4

video 1/1 (697/751) /content/fruits2_360.mp4: 224x384 1 Beetroot, 1 Dates, 4 Pomegranates, 5.2ms
video 1/1 (698/751) /content/fruits2_360.mp4: 224x384 2 Kakis, 2 Pomegranates, 1 Tomato, 5.4ms
video 1/1 (699/751) /content/fruits2_360.mp4: 224x384 1 Pomegranate, 5.2ms
video 1/1 (700/751) /content/fruits2_360.mp4: 224x384 1 Kaki, 1 Onion, 3 Pomegranates, 8.2ms
video 1/1 (701/751) /content/fruits2_360.mp4: 224x384 2 Pomegranates, 5.3ms
video 1/1 (702/751) /content/fruits2_360.mp4: 224x384 1 Cherry, 1 Grape, 1 Kohlrabi, 4 Pomegranates, 5.5ms
video 1/1 (703/751) /content/fruits2_360.mp4: 224x384 1 Grape, 2 Kakis, 2 Pomegranates, 7.2ms
video 1/1 (704/751) /content/fruits2_360.mp4: 224x384 2 Kakis, 3 Pomegranates, 5.6ms
video 1/1 (705/751) /content/fruits2_360.mp4: 224x384 5 Kakis, 3 Pomegranates, 5.5ms
video 1/1 (706/751) /content/fruits2_360.mp4: 224x384 1 Clementine, 6 Kakis, 4 Pomegranates, 5.6ms
video 1/1 (707/751) /content/fruits2_360.mp4: 224x384 6 Kakis, 1 Pepper, 1 Pomegranate, 1 Tomato, 5.9ms
video 1/1 (708/751) /content/fruits2_360.mp4: 224x384 3 Kakis, 5.7ms
video 1/1 (709/751) /content/fruits2_360.mp4: 224x384 3 Kakis, 1 Pomegranate, 5.4ms
video 1/1 (710/751) /content/fruits2_360.mp4: 224x384 1 Grape, 3 Kakis, 1 Mangostan, 5.7ms
```

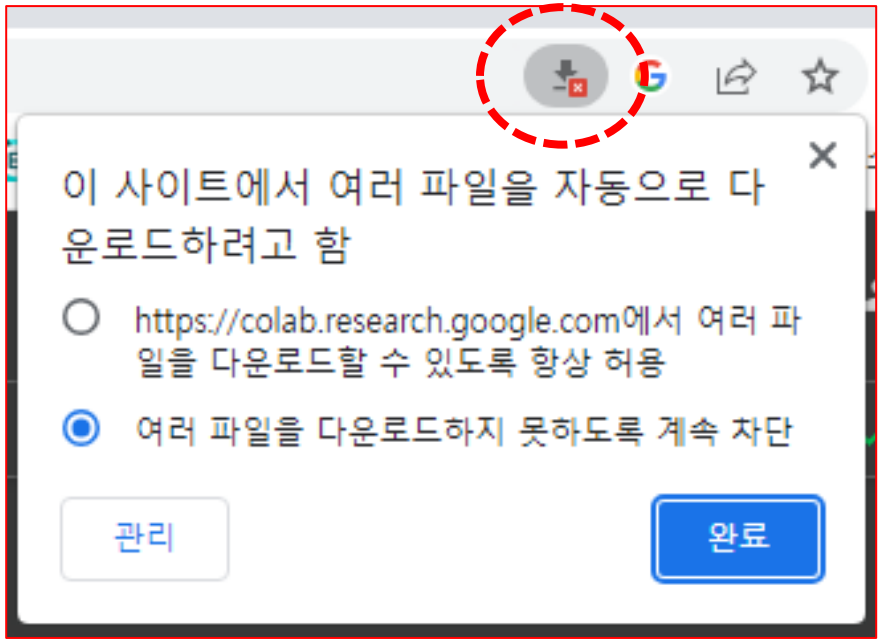
2. Deep Learning(YOLO 5)

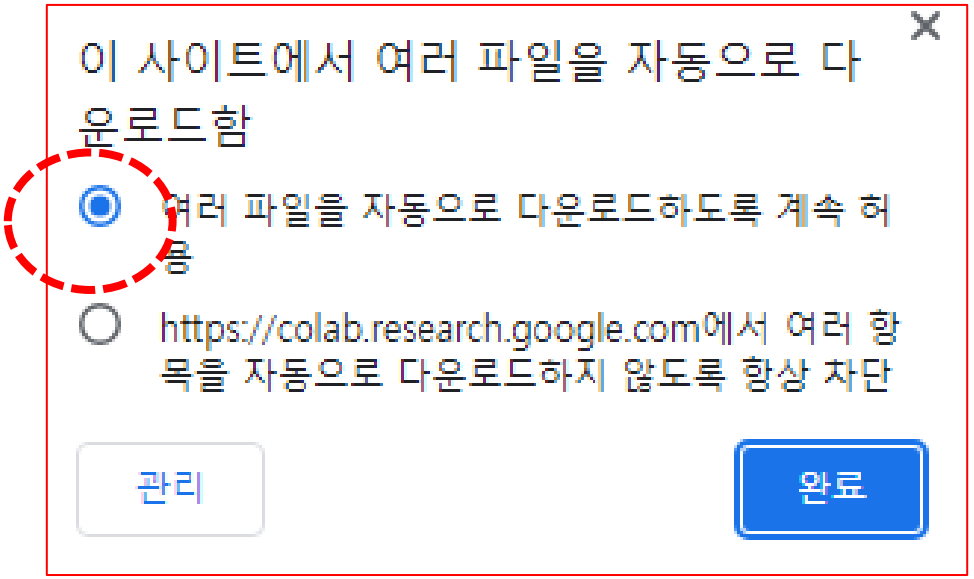
학습된 가중치 파일인 sys_fruit_best.pt를 별도 지정하여 예측 시행

```
!tect.py --weights /content/drive/MyDrive/ColabNotebooks/dataset/syn_fruit_best.pt --img 360 --conf 0.5 --source /content/drive/MyDrive/ColabNotebooks/dataset/fruits/fruits2_360.mp4

video 1/1 (697/751) /content/drive/MyDrive/ColabNotebooks/dataset/fruits/fruits2_360.mp4: 224x384 1 Beetroot, 1 Dates, 4 Pomegranates, 7.9ms
video 1/1 (698/751) /content/drive/MyDrive/ColabNotebooks/dataset/fruits/fruits2_360.mp4: 224x384 2 Kakis, 2 Pomegranates, 1 Tomato, 7.8ms
video 1/1 (699/751) /content/drive/MyDrive/ColabNotebooks/dataset/fruits/fruits2_360.mp4: 224x384 1 Pomegranate, 7.5ms
video 1/1 (700/751) /content/drive/MyDrive/ColabNotebooks/dataset/fruits/fruits2_360.mp4: 224x384 1 Kaki, 1 Onion, 3 Pomegranates, 9.3ms
video 1/1 (701/751) /content/drive/MyDrive/ColabNotebooks/dataset/fruits/fruits2_360.mp4: 224x384 2 Pomegranates, 12.2ms
video 1/1 (702/751) /content/drive/MyDrive/ColabNotebooks/dataset/fruits/fruits2_360.mp4: 224x384 1 Cherry, 1 Grape, 1 Kohlrabi, 4 Pomegranates, 8.2ms
video 1/1 (703/751) /content/drive/MyDrive/ColabNotebooks/dataset/fruits/fruits2_360.mp4: 224x384 1 Grape, 2 Kakis, 2 Pomegranates, 7.8ms
video 1/1 (704/751) /content/drive/MyDrive/ColabNotebooks/dataset/fruits/fruits2_360.mp4: 224x384 2 Kakis, 3 Pomegranates, 7.6ms
video 1/1 (705/751) /content/drive/MyDrive/ColabNotebooks/dataset/fruits/fruits2_360.mp4: 224x384 5 Kakis, 3 Pomegranates, 8.0ms
video 1/1 (706/751) /content/drive/MyDrive/ColabNotebooks/dataset/fruits/fruits2_360.mp4: 224x384 1 Clementine, 6 Kakis, 4 Pomegranates, 7.7ms
video 1/1 (707/751) /content/drive/MyDrive/ColabNotebooks/dataset/fruits/fruits2_360.mp4: 224x384 6 Kakis, 1 Pepper, 1 Pomegranate, 1 Tomato, 7.9ms
video 1/1 (708/751) /content/drive/MyDrive/ColabNotebooks/dataset/fruits/fruits2_360.mp4: 224x384 3 Kakis, 12.2ms
video 1/1 (709/751) /content/drive/MyDrive/ColabNotebooks/dataset/fruits/fruits2_360.mp4: 224x384 3 Kakis, 1 Pomegranate, 8.3ms
video 1/1 (710/751) /content/drive/MyDrive/ColabNotebooks/dataset/fruits/fruits2_360.mp4: 224x384 1 Grape, 3 Kakis, 1 Mangostan, 7.9ms
video 1/1 (711/751) /content/drive/MyDrive/ColabNotebooks/dataset/fruits/fruits2_360.mp4: 224x384 3 Kakis, 7.9ms
video 1/1 (712/751) /content/drive/MyDrive/ColabNotebooks/dataset/fruits/fruits2_360.mp4: 224x384 2 Kakis, 1 Strawberry, 1 Tomato, 7.1ms
video 1/1 (713/751) /content/drive/MyDrive/ColabNotebooks/dataset/fruits/fruits2_360.mp4: 224x384 2 Kakis, 1 Pomegranate, 1 Tomato, 13.8ms
video 1/1 (714/751) /content/drive/MyDrive/ColabNotebooks/dataset/fruits/fruits2_360.mp4: 224x384 1 Clementine, 1 Kaki, 1 Pomegranate, 8.4ms
```

2. Deep Learning(YOLO 5)

코랩에서 다운로드가 않되는 경우 주소창의 우측 상단에 옵션 설정 변경





2. Deep Learning(YOLO 5)

동영상 사이즈를 360으로 지정한 경우 탐지된 객체가 일부 있다

```
!python detect.py --weights /content/yolov5/runs/train/fruits_yolov5s_results/weights/best.pt --img 360 --conf 0.5 --source /content/strawberry_360p.mp4
video 1/1 (162/216) /content/strawberry_360p.mp4: 224x384 2 Strawberrys, 4.7ms
video 1/1 (163/216) /content/strawberry_360p.mp4: 224x384 2 Strawberrys, 4.7ms
video 1/1 (164/216) /content/strawberry_360p.mp4: 224x384 2 Strawberrys, 4.7ms
video 1/1 (165/216) /content/strawberry_360p.mp4: 224x384 2 Strawberrys, 5.0ms
video 1/1 (166/216) /content/strawberry_360p.mp4: 224x384 2 Strawberrys, 4.7ms
video 1/1 (167/216) /content/strawberry_360p.mp4: 224x384 2 Strawberrys, 5.0ms
video 1/1 (168/216) /content/strawberry_360p.mp4: 224x384 2 Strawberrys, 4.7ms
video 1/1 (169/216) /content/strawberry_360p.mp4: 224x384 2 Strawberrys, 5.0ms
video 1/1 (170/216) /content/strawberry_360p.mp4: 224x384 2 Strawberrys, 5.1ms
video 1/1 (171/216) /content/strawberry_360p.mp4: 224x384 2 Strawberrys, 4.8ms
video 1/1 (172/216) /content/strawberry_360p.mp4: 224x384 2 Strawberrys, 4.7ms
video 1/1 (173/216) /content/strawberry_360p.mp4: 224x384 2 Strawberrys, 5.2ms
video 1/1 (174/216) /content/strawberry_360p.mp4: 224x384 3 Strawberrys, 4.8ms
```

2. Deep Learning(YOLO 5)Youtube

유튜브 영상 : 384로 시도시 지정한 경우 탐지된 객체가 일부 있다

```
1 !python detect.py --weights /content/yolov5/runs/train/fruits_yolov5s_results/weights/best.pt --img 384 --conf 0.5 --source  https://www.youtube.com/watch?v=zqldBJq6Kvs

detect: weights=['/content/yolov5/runs/train/fruits_yolov5s_results/weights/best.pt'], source=https://www.youtube.com/watch?v=zqldBJq6Kvs, data=data/coco128.yaml, imgsz=[384, 384],
YOLOv5 🚀 v7.0-172-gc3c1304 Python-3.10.11 torch-2.0.1+cu118 CUDA:0 (Tesla T4, 15102MiB)

Fusing layers...
YOLOv5s summary: 157 layers, 7180036 parameters, 0 gradients, 16.3 GFLOPs
WARNING ⚠ Environment does not support cv2.imshow() or PIL Image.show()

1/1: https://www.youtube.com/watch?v=zqldBJq6Kvs...  Success (735 frames 1280x720 at 29.97 FPS)

0: 224x384 (no detections), 308.7ms
0: 224x384 (no detections), 14.5ms
0: 224x384 (no detections), 15.3ms
0: 224x384 (no detections), 9.1ms
0: 224x384 (no detections), 7.2ms
```

```
0: 224x384 2 Mandarines, 5.0ms
0: 224x384 2 Mandarines, 5.1ms
0: 224x384 2 Mandarines, 5.8ms
0: 224x384 2 Mandarines, 5.1ms
0: 224x384 2 Mandarines, 5.1ms
0: 224x384 1 Cantaloupe, 1 Mandarine, 6.1ms
0: 224x384 1 Cantaloupe, 1 Mandarine, 5.3ms
0: 224x384 1 Cantaloupe, 1 Mandarine, 5.3ms
0: 224x384 1 Cantaloupe, 1 Mandarine, 5.3ms
0: 224x384 1 Cantaloupe, 1 Mandarine, 5.4ms
0: 224x384 1 Cantaloupe, 1 Mandarine, 5.2ms
0: 224x384 1 Cantaloupe, 1 Mandarine, 5.0ms
0: 224x384 1 Cantaloupe, 1 Mandarine, 5.4ms
0: 224x384 1 Cantaloupe, 1 Mandarine, 5.3ms
0: 224x384 1 Cantaloupe, 1 Mandarine, 5.2ms
0: 224x384 1 Cantaloupe, 1 Mandarine, 9.3ms
```

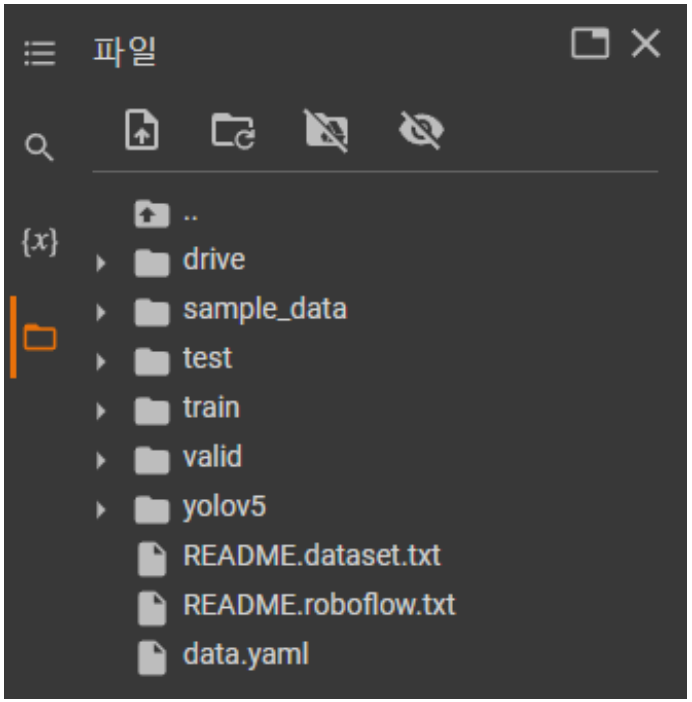
2. Deep Learning(YOLO 5)

Aquarium Data Set (YOLO 5)

2. Deep Learning(YOLO 5)

```python
import shutil
import os

# 삭제할 폴더 경로 지정
folder_path = '/content/test'

# 폴더가 존재하는지 확인 후 삭제
if os.path.exists(folder_path):
    shutil.rmtree(folder_path)
    print(f"폴더 '{folder_path}'가 삭제되었습니다.")
else:
    print(f"폴더 '{folder_path}'가 존재하지 않습니다.")

```

```
폴더 '/content/test'가 삭제되었습니다.
```



연속하여 데이터 분석을 시행할 경우
신규 데이터셋이 설치않되는 경우가 있다 >> 위의 코드를 실행시켜 해당 폴더를 삭제한다(test, train, valid), yaml 등의 파일은 replace시켜야 한다.

2. Deep Learning(YOLO 5)

```
1 %cd /content/
```

```
/content
```

```
1 !curl -L "https://public.roboflow.com/ds/8pM0hfAlQj?key=Dr5zacflxn" > roboflow.zip; unzip roboflow.zip; rm roboflow.zip
```

```
extracting: valid/labels/IMG_2645_jpeg_jpg.rf.95466fb2ccc8118fbd346f305b7d4319.txt
extracting: valid/labels/IMG_3120_jpeg_jpg.rf.05e302318ebf9502b3467828b1f1e45a.txt
extracting: valid/labels/IMG_3124_jpeg_jpg.rf.539a07bdc20ae24c50d185c747476b94.txt
extracting: valid/labels/IMG_3126_jpeg_jpg.rf.f9991ffcba599a478eb6cf04c69cdd73.txt
```

현재 디렉토리를 확인 후

Public roboflow에서 aquarium 데이터의 링크를 실행하여 데이터를 설치한다.

2. Deep Learning(YOLO 5)

roboflow

Universe | Public Datasets | Model Zoo | Blog | Docs

Explore these datasets, models, and more on Roboflow Universe. →

66+ MILLION IMAGES | 90,000+ DATASETS | 7,000+ PRE-TRAINED MODELS

Dataset Summary

Dataset Health Check

DOWNLOADS

raw-1024 638

Try Pre-Trained Model

Aquarium Dataset

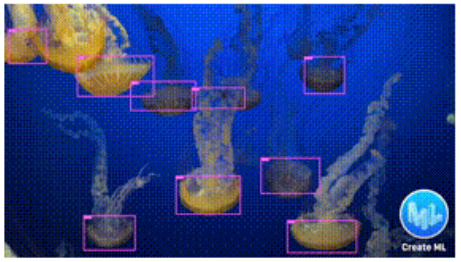
Shared By
Roboflow
November 2020

License
CC BY 4.0
More Info

Annotations
creatures
Object Detection

Downloads

raw-1024 638 Images



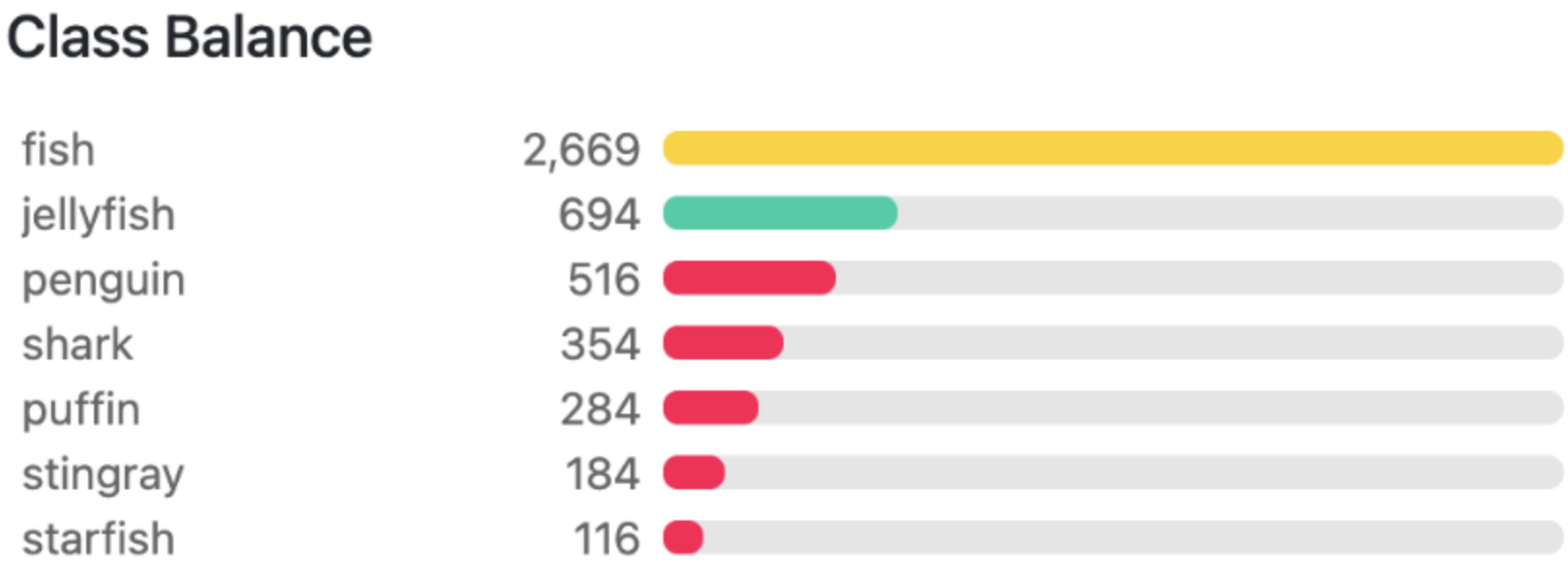
Dataset Details

This dataset consists of 638 images collected by Roboflow from two aquariums in the United States: The Henry Doorly Zoo in Omaha (October 16, 2020) and the National Aquarium in Baltimore (November 14, 2020). The images were labeled for object detection by the Roboflow team (with some help from SageMaker Ground Truth). Images and annotations are released under a Creative Commons By-Attribution license. You are free to use them for any purposes personal, commercial, or academic provided you give acknowledgement of their source.

2. Deep Learning(YOLO 5)

Class Breakdown

The following classes are labeled: fish, jellyfish, penguins, sharks, puffins, stingrays, and starfish. Most images contain multiple bounding boxes.



2. Deep Learning(YOLO 5)

```
 1 !pwd
```

```
/content
```

```
 1 !git clone https://github.com/ultralytics/yolov5.git
```

```
fatal: destination path 'yolov5' already exists and is not an empty directory.
```

```
 1 %cd yolov5
```

```
/content/yolov5
```

```
 1 !pip install -r requirements.txt
```

```
Requirement already satisfied: gitpython>=3.1.30 in /usr/local/lib/python3.10/dist-packages (from -r requirements.txt (line 5)) (3.1.
Requirement already satisfied: matplotlib>=3.3 in /usr/local/lib/python3.10/dist-packages (from -r requirements.txt (line 6)) (3.7.1)
Requirement already satisfied: numpy>=1.18.5 in /usr/local/lib/python3.10/dist-packages (from -r requirements.txt (line 7)) (1.22.4)
```

YOLO5의 실행환경에 맞는 설정과 설치를 확인한다.

2. Deep Learning(YOLO 5)

```
1 !python train.py --img 416 --batch 16 --epochs 50 --data /content/data.yaml --cfg ./models/yolov5s.yaml --weights yolov5s.pt --name aquafish_yolov5s_results

train: weights=yolov5s.pt, cfg=./models/yolov5s.yaml, data=/content/data.yaml, hyp=data/hyps/hyp.scratch-low.yaml, epochs=50, batch_size=16, imgsz=416, rect=Fal
github: up to date with https://github.com/ultralytics/yolov5 ✅
YOLOv5 🚀 v7.0-172-gc3c1304 Python-3.10.11 torch-2.0.1+cu118 CUDA:0 (Tesla T4, 15102MiB)

hyperparameters: lr0=0.01, lrf=0.01, momentum=0.937, weight_decay=0.0005, warmup_epochs=3.0, warmup_momentum=0.8, warmup_bias_lr=0.1, box=0.05, cls=0.5, cls_p
ClearML: run 'pip install clearml' to automatically track, visualize and remotely train YOLOv5 🚀 in ClearML
Comet: run 'pip install comet_ml' to automatically track and visualize YOLOv5 🚀 runs in Comet
TensorBoard: Start with 'tensorboard --logdir runs/train', view at http://localhost:6006/
Overriding model.yaml nc=80 with nc=7

                 from  n    params  module                                  arguments
  0                -1  1      3520  models.common.Conv                      [3, 32, 6, 2, 2]
  1                -1  1     18560  models.common.Conv                      [32, 64, 3, 2]
  2                -1  1     18816  models.common.C3                        [64, 64, 1]
  3                -1  1     73984  models.common.Conv                      [64, 128, 3, 2]
  4                -1  2    115712  models.common.C3                        [128, 128, 2]
  5                -1  1    295424  models.common.Conv                      [128, 256, 3, 2]
  6                -1  3    625152  models.common.C3                        [256, 256, 3]
  7                -1  1   1180672  models.common.Conv                      [256, 512, 3, 2]
  8                -1  1   1182720  models.common.C3                        [512, 512, 1]
  9                -1  1    656896  models.common.SPPF                      [512, 512, 5]
 10                -1  1    131584  models.common.Conv                      [512, 256, 1, 1]
```

!python train.py --img 416 --batch 16 --epochs 50 --data /content/data.yaml --cfg ./models/yolov5s.yaml --weights yolov5s.pt --name aquafish_yolov5s_results

아쿠아리움 데이터로 학습을 진행 !!

2. Deep Learning(YOLO 5)

```
50 epochs completed in 0.377 hours.
Optimizer stripped from runs/train/aquafish_yolov5s_results/weights/last.pt, 14.3MB
Optimizer stripped from runs/train/aquafish_yolov5s_results/weights/best.pt, 14.3MB

Validating runs/train/aquafish_yolov5s_results/weights/best.pt...
Fusing layers...
YOLOv5s summary: 157 layers, 7029004 parameters, 0 gradients, 15.8 GFLOPs
                 Class     Images  Instances          P          R      mAP50  mAP50-95: 100% 4/4 [00:06<00:00,  1.59s/it]
                   all        127        909       0.77      0.694      0.747      0.404
                  fish        127        459      0.818      0.734      0.816      0.421
             jellyfish        127        155      0.784       0.91      0.918      0.511
               penguin        127        104      0.683      0.692      0.736       0.31
                puffin        127         74      0.678      0.455      0.487      0.195
                 shark        127         57      0.826      0.675      0.767      0.398
              starfish        127         27       0.83      0.723      0.775      0.527
              stingray        127         33      0.773      0.667       0.73      0.467
Results saved to runs/train/aquafish_yolov5s_results
```

50 epochs를 0.37 시간동안 진행 후 예측 검증된 데이터 결과

2. Deep Learning(YOLO 5)

테스트할 데이터 샘플 준비(픽사베이 등)

2. Deep Learning(YOLO 5)

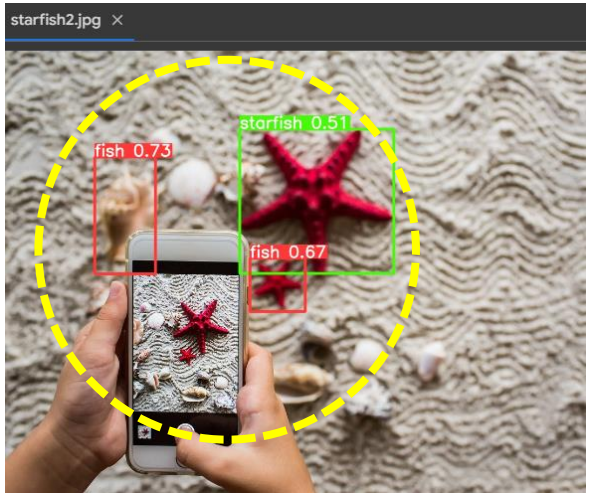
```
!python detect.py --weights /content/yolov5/runs/train/fruit_yolov5s_results/weights/best.pt --conf 0.5 --source /content/starfish2.jpg

detect: weights=['/content/yolov5/runs/train/fruit_yolov5s_results/weights/best.pt'], source=/content/starfish2.jpg, data=data/coco128.yam
YOLOv5 🚀 v7.0-172-gc3c1304 Python-3.10.11 torch-2.0.1+cu118 CUDA:0 (Tesla T4, 15102MiB)

Fusing layers...
YOLOv5s summary: 157 layers, 7029004 parameters, 0 gradients, 15.8 GFLOPs
image 1/1 /content/starfish2.jpg: 448x640 2 fishs, 1 starfish, 46.8ms
Speed: 0.6ms pre-process, 46.8ms inference, 1.6ms NMS per image at shape (1, 3, 640, 640)
Results saved to runs/detect/exp4
```

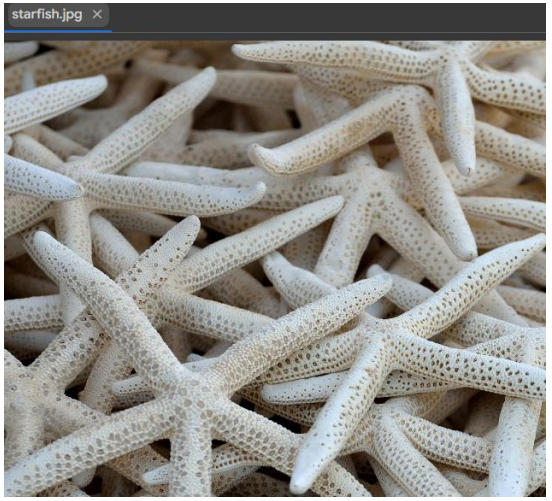
```
!python detect.py --weights /content/yolov5/runs/train/fruit_yolov5s_results/weights/best.pt --conf 0.5 --source /content/starfish.jpg

detect: weights=['/content/yolov5/runs/train/fruit_yolov5s_results/weights/best.pt'], source=/content/starfish.jpg, data=data/coco128.yaml
YOLOv5 🚀 v7.0-172-gc3c1304 Python-3.10.11 torch-2.0.1+cu118 CUDA:0 (Tesla T4, 15102MiB)

Fusing layers...
YOLOv5s summary: 157 layers, 7029004 parameters, 0 gradients, 15.8 GFLOPs
image 1/1 /content/starfish.jpg: 608x640 (no detections), 45.8ms
Speed: 0.6ms pre-process, 45.8ms inference, 0.4ms NMS per image at shape (1, 3, 640, 640)
Results saved to runs/detect/exp7
```

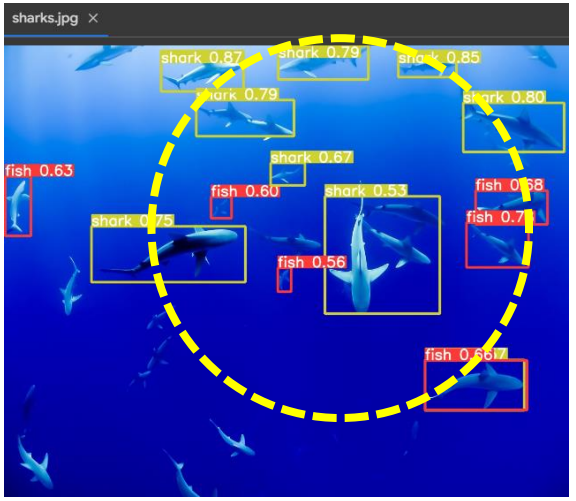
```
!python detect.py --weights /content/yolov5/runs/train/fruit_yolov5s_results/weights/best.pt --conf 0.5 --source /content/sharks.jpg

detect: weights=['/content/yolov5/runs/train/fruit_yolov5s_results/weights/best.pt'], source=/content/sharks.jpg, data=data/coco128.yaml,
YOLOv5 🚀 v7.0-172-gc3c1304 Python-3.10.11 torch-2.0.1+cu118 CUDA:0 (Tesla T4, 15102MiB)

Fusing layers...
YOLOv5s summary: 157 layers, 7029004 parameters, 0 gradients, 15.8 GFLOPs
image 1/1 /content/sharks.jpg: 480x640 6 fishs, 9 sharks, 69.3ms
Speed: 0.6ms pre-process, 69.3ms inference, 2.2ms NMS per image at shape (1, 3, 640, 640)
Results saved to runs/detect/exp8
```

2. Deep Learning(YOLO 5)

탐지된 객체가 일부 있거나 탐지되지 않은 객체도 있다







```
!python detect.py --weights /content/yolov5/best_aquaFish.pt --
img 416 --conf 0.5 --source /content/rays.mp4
```

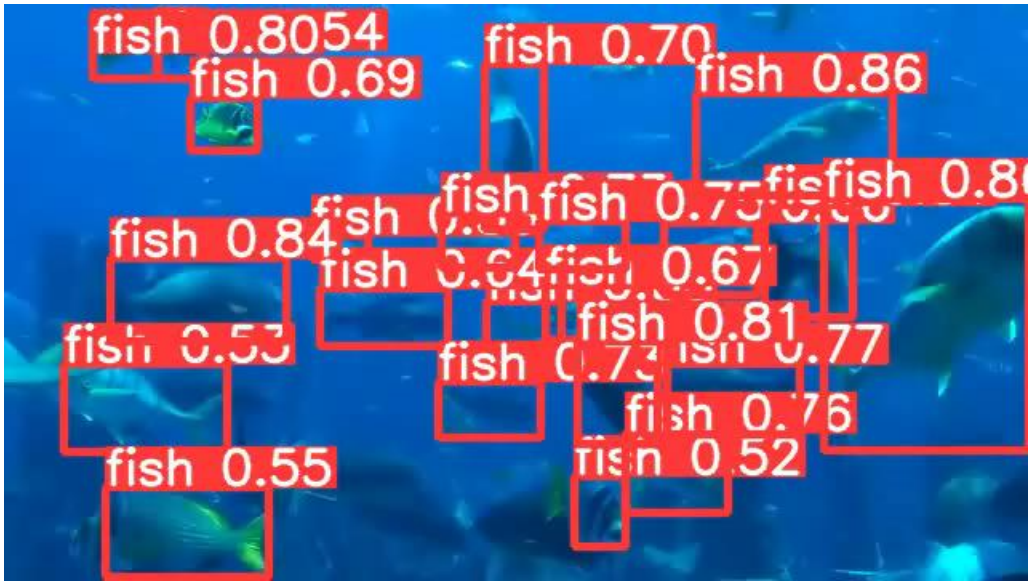
2. Deep Learning (YOLO 5) 동영상

```
 1 !python detect.py --weights /content/yolov5/best_aquaFish.pt --img 416 --conf 0.5 --source /content/rays.mp4

detect: weights=['/content/yolov5/best_aquaFish.pt'], source=/content/rays.mp4, data=data/coco128.yaml, imgsz=[416, 416]
YOLOv5 🚀 v7.0-172-gc3c1304 Python-3.10.11 torch-2.0.1+cu118 CUDA:0 (Tesla T4, 15102MiB)

Fusing layers...
YOLOv5s summary: 157 layers, 7029004 parameters, 0 gradients, 15.8 GFLOPs
video 1/1 (1/512) /content/rays.mp4: 256x416 22 fishs, 60.8ms
video 1/1 (2/512) /content/rays.mp4: 256x416 19 fishs, 8.0ms
video 1/1 (3/512) /content/rays.mp4: 256x416 19 fishs, 7.6ms
video 1/1 (4/512) /content/rays.mp4: 256x416 17 fishs, 9.0ms
video 1/1 (5/512) /content/rays.mp4: 256x416 15 fishs, 6.9ms
video 1/1 (6/512) /content/rays.mp4: 256x416 13 fishs, 7.0ms
video 1/1 (7/512) /content/rays.mp4: 256x416 16 fishs, 7.5ms
video 1/1 (8/512) /content/rays.mp4: 256x416 17 fishs, 6.9ms
video 1/1 (9/512) /content/rays.mp4: 256x416 16 fishs, 8.8ms
video 1/1 (10/512) /content/rays.mp4: 256x416 13 fishs, 7.2ms
video 1/1 (11/512) /content/rays.mp4: 256x416 14 fishs, 6.9ms
video 1/1 (12/512) /content/rays.mp4: 256x416 15 fishs, 8.8ms
video 1/1 (13/512) /content/rays.mp4: 256x416 13 fishs, 8.9ms
video 1/1 (14/512) /content/rays.mp4: 256x416 17 fishs, 9.3ms
video 1/1 (15/512) /content/rays.mp4: 256x416 15 fishs, 9.0ms
video 1/1 (16/512) /content/rays.mp4: 256x416 18 fishs, 9.1ms
video 1/1 (17/512) /content/rays.mp4: 256x416 17 fishs, 9.2ms
```

```
video 1/1 (507/512) /content/rays.mp4: 256x416 10 fishs, 2 sharks, 5.0ms
video 1/1 (508/512) /content/rays.mp4: 256x416 14 fishs, 2 sharks, 4.9ms
video 1/1 (509/512) /content/rays.mp4: 256x416 12 fishs, 1 shark, 4.9ms
video 1/1 (510/512) /content/rays.mp4: 256x416 11 fishs, 1 shark, 5.1ms
video 1/1 (511/512) /content/rays.mp4: 256x416 15 fishs, 2 sharks, 4.8ms
video 1/1 (512/512) /content/rays.mp4: 256x416 13 fishs, 2 sharks, 4.8ms
Speed: 0.2ms pre-process, 5.6ms inference, 0.9ms NMS per image at shape (1, 3, 416, 416)
Results saved to runs/detect/exp
```

2. Deep Learning (YOLO 5) 동영상

동영상에 대해 416으로 지정한 경우 탐지된 객체가 일부 있다



2. Deep Learning (YOLO 5) 동영상

YOLO의 객체 인식이 잘 안되는 경우에는 몇 가지 일반적인 상황

1.작은 객체: 모델이 작은 객체의 세부 정보를 충분히 인식하기 어렵게 만들 수 있다.
2.멀리 떨어진 객체: 이미지에 더 많은 노이즈와 함께 희미하게 나타날 수 있다.
3.비슷한 객체: 같은 종류의 과일/동물들이 많이 있는 경우 구분하기 어렵게 될 수 있다.
4.부적절한 학습 데이터: 모델이 실내 이미지로 학습되었지만 테스트 이미지가 야외에서 촬영된 경우, 모델의 성능이 저하될 수 있다.
5.모델 크기와 복잡도: 작은 모델은 작은 객체를 잘 인식하지만, 대신 세부 정보를 잃을 수 있다. 반면 큰 모델은 더 많은 세부 정보를 인식할 수 있지만, 계산 비용이 증가

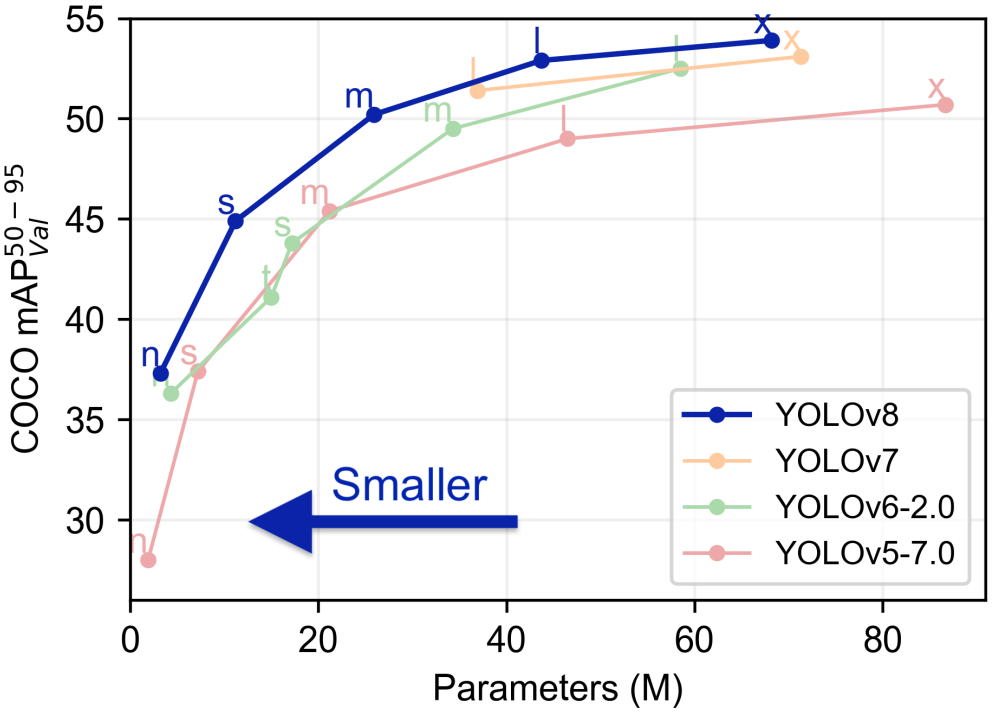
다양한 환경에서 촬영된 이미지들은 **조명 조건, 색감, 명암 대비 등의 차이**가 있을 수 있으며, 이는 모델의 성능에 영향을 미칠 수 있다.

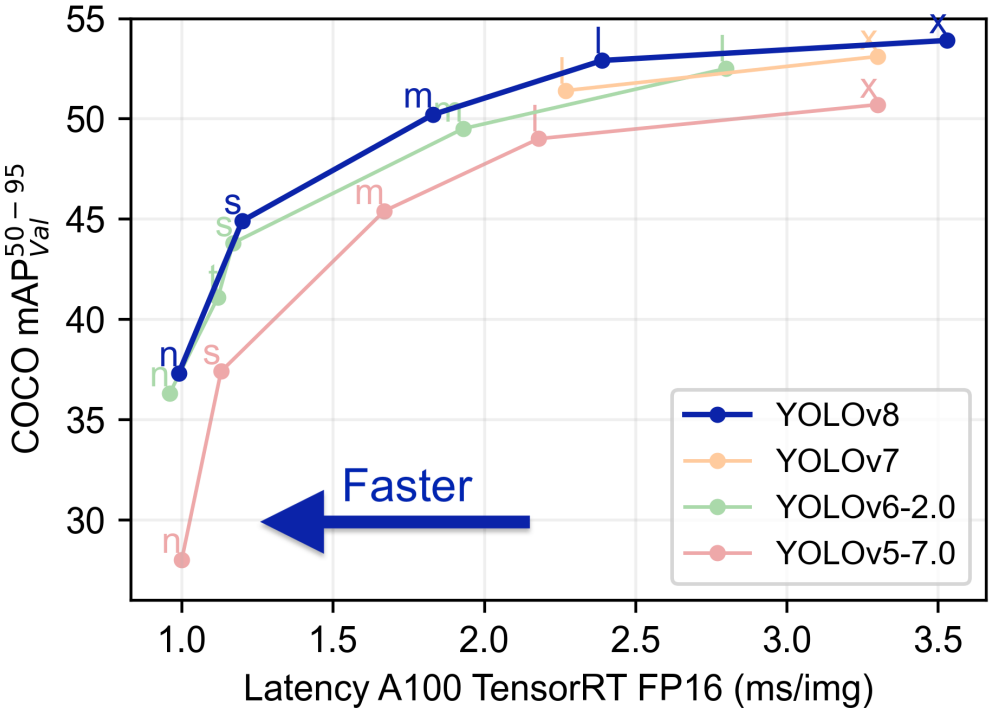
객체 인식의 성능을 향상시키기 위해서는 **다양한 조건**에서 학습된 데이터셋을 사용하고, **적절한 데이터 증강 기법**을 적용하여 모델의 **일반화 능력을 향상**시키는 것이 중요.
또한 특정 환경에서 모델의 성능을 검증하고 필요한 경우 추가적인 조치를 취하는 것도 중요.

2. Deep Learning(YOLO 8)

GitHub - ultralytics/ultralytics: NEW - YOLOv8 🚀 in PyTorch > ONNX > CoreML > TFLite

2. Deep Learning(YOLO 8)





YOLO8 : YOLO5보다 작은 모델 규모(파라미터)에 더 빠른 객체 탐지 성능

https://colab.research.google.com/github/ultralytics/ultralytics/blob/main/examples/tutorial.ipynb

2. Deep Learning(YOLO 8)

▼ Install

Pip install the ultralytics package including all requirements in a **Python>=3.7** environment with **PyTorch>=1.7**.

```
pip install ultralytics
```

▼ Usage

CLI

YOLOv8 may be used directly in the Command Line Interface (CLI) with a `yolo` command:

```
yolo predict model=yolov8n.pt source='https://ultralytics.com/images/bus.jpg'
```

`yolo` can be used for a variety of tasks and modes and accepts additional arguments, i.e. `imgsz=640`. See the YOLOv8 CLI Docs for examples.

Ultralytics 패키지를 설치 후 모델을 지정하여 예측 시행

2. Deep Learning(YOLO 8)

Python

YOLOv8 may also be used directly in a Python environment, and accepts the same arguments as in the CLI example above:

```python
from ultralytics import YOLO

# Load a model
model = YOLO("yolov8n.yaml")  # build a new model from scratch
model = YOLO("yolov8n.pt")  # load a pretrained model (recommended for training)

# Use the model
model.train(data="coco128.yaml", epochs=3)  # train the model
metrics = model.val()  # evaluate model performance on the validation set
results = model("https://ultralytics.com/images/bus.jpg")  # predict on an image
path = model.export(format="onnx")  # export the model to ONNX format
```

2. Deep Learning(YOLO 8)

▼ Detection

See Detection Docs for usage examples with these models.

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

▼ Segmentation

See Segmentation Docs for usage examples with these models.

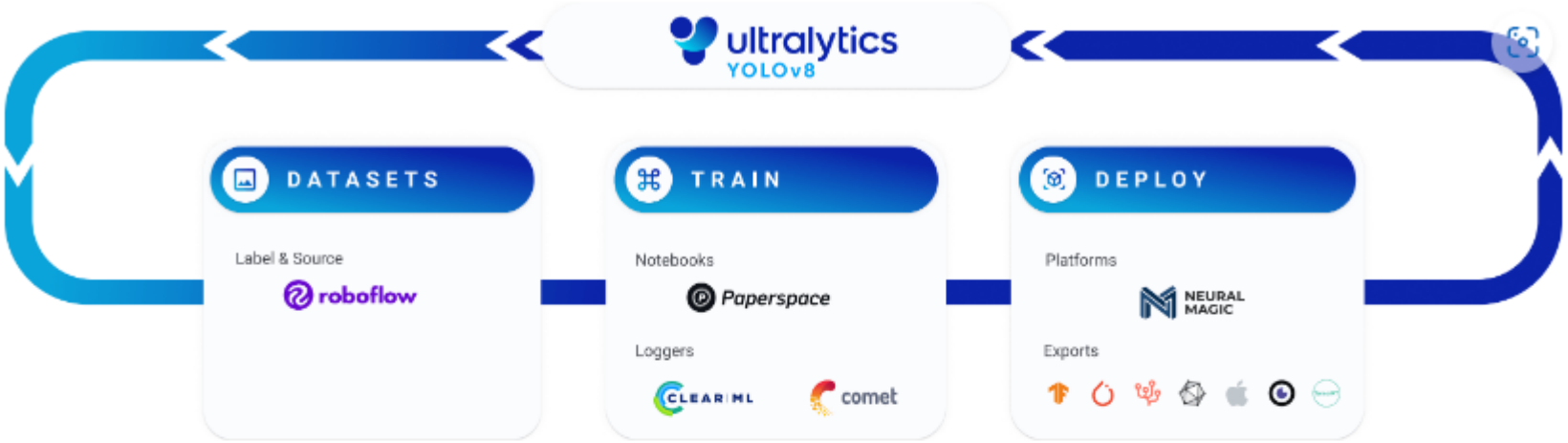
Model	size (pixels)	mAP^{box} 50-95	mAP^{mask} 50-95	Speed CPU ONNX (ms)	Speed A100 TensorRT (ms)	params (M)	FLOPs (B)
YOLOv8n-seg	640	36.7	30.5	96.1	1.21	3.4	12.6
YOLOv8s-seg	640	44.6	36.8	155.7	1.47	11.8	42.6
YOLOv8m-seg	640	49.9	40.8	317.0	2.18	27.3	110.2
YOLOv8l-seg	640	52.3	42.6	572.4	2.79	46.0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

▼ Classification

See Classification Docs for usage examples with these models.

Model	size (pixels)	acc top1	acc top5	Speed CPU ONNX (ms)	Speed A100 TensorRT (ms)	params (M)	FLOPs (B) at 640
YOLOv8n-cls	224	66.6	87.0	12.9	0.31	2.7	4.3
YOLOv8s-cls	224	72.3	91.1	23.4	0.35	6.4	13.5
YOLOv8m-cls	224	76.4	93.2	85.4	0.62	17.0	42.7
YOLOv8l-cls	224	78.0	94.1	163.0	0.87	37.5	99.7
YOLOv8x-cls	224	78.4	94.3	232.0	1.01	57.4	154.8

2. Deep Learning(YOLO 8)



Roboflow	ClearML ⭐ NEW	Comet ⭐ NEW	Neural Magic ⭐ NEW
Label and export your custom datasets directly to YOLOv8 for training with Roboflow	Automatically track, visualize and even remotely train YOLOv8 using ClearML (open-source!)	Free forever, Comet lets you save YOLOv8 models, resume training, and interactively visualize and debug predictions	Run YOLOv8 inference up to 6x faster with Neural Magic DeepSparse

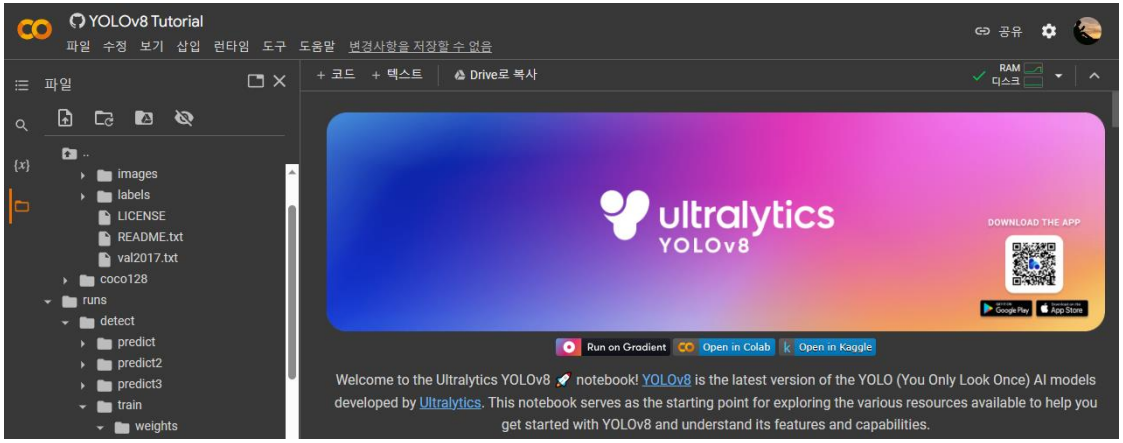
2. Deep Learning(YOLO 8)

Ultralytics HUB

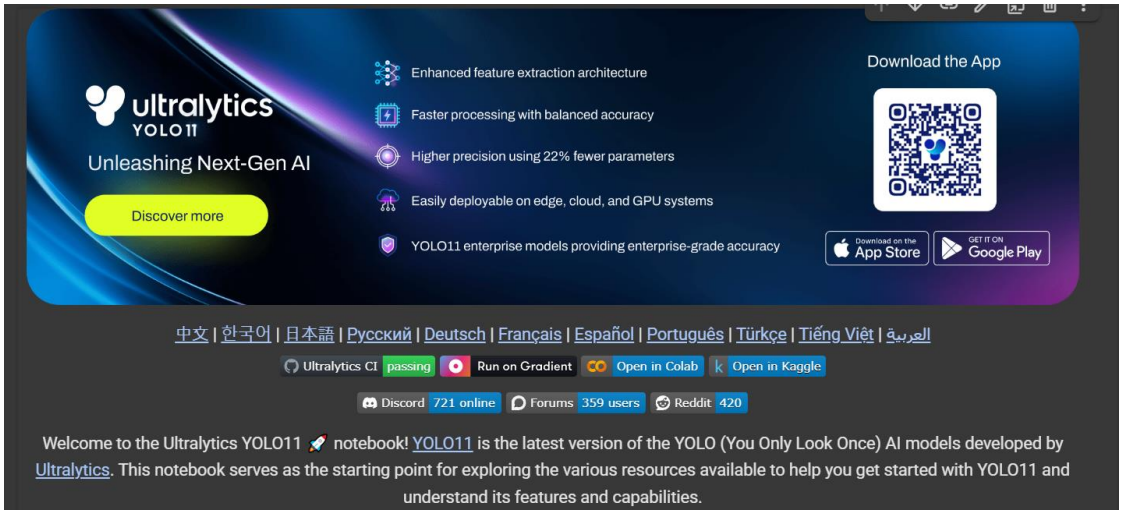
Experience seamless AI with Ultralytics HUB ⭐, the all-in-one solution for data visualization, YOLOv5 and YOLOv8 🚀 model training and deployment, without any coding. Transform images into actionable insights and bring your AI visions to life with ease using our cutting-edge platform and user-friendly Ultralytics App. Start your journey for **Free** now!

https://colab.research.google.com/github/ultralytics/ultralytics/blob/main/examples/tutorial.ipynb

2. Deep Learning(YOLO8 >11)

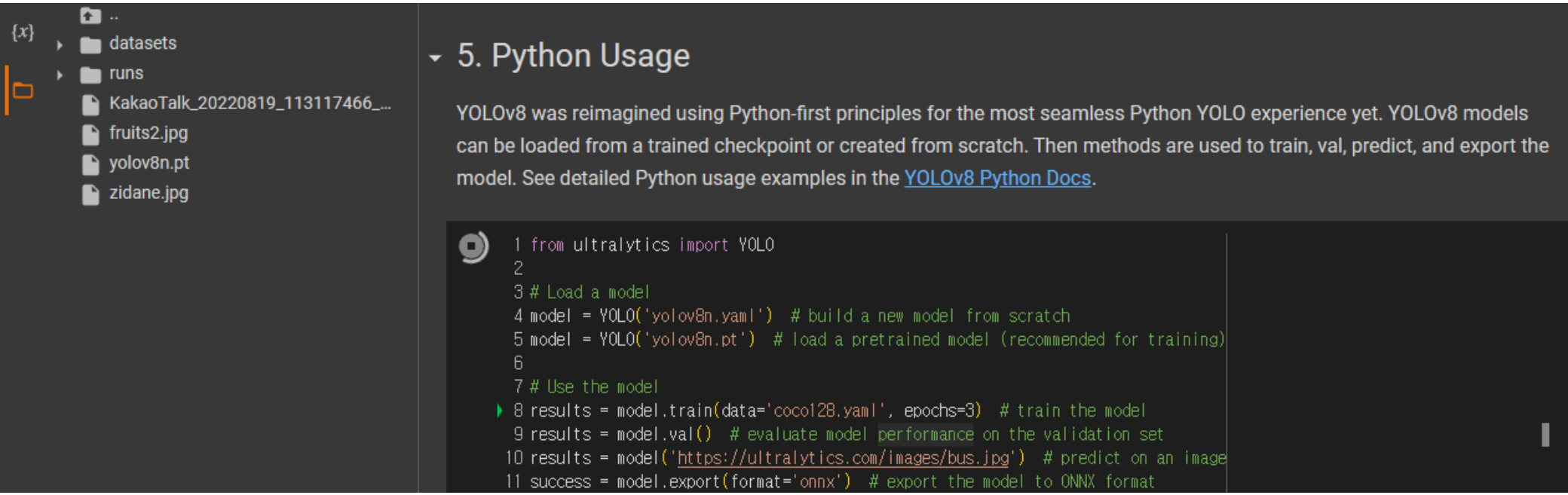


YOLO의 튜토리얼을 통한 객체 탐지 체험



https://colab.research.google.com/github/ultralytics/ultralytics/blob/main/examples/tutorial.ipynb

2. Deep Learning(YOLO 8)



YOLO 모델의 pretrained model 지정과 학습방법

https://docs.ultralytics.com/ko/modes/train/#tensorboard

2. Deep Learning(YOLO)

Arguments

YOLO 모델의 다양한 하이퍼 파라미터 사용법

Training settings for YOLO models refer to the various hyperparameters and configurations used to train the model on a dataset. These settings can affect the model's performance, speed, and accuracy. Some common YOLO training settings include the batch size, learning rate, momentum, and weight decay. Other factors that may affect the training process include the choice of optimizer, the choice of loss function, and the size and composition of the training dataset. It is important to carefully tune and experiment with these settings to achieve the best possible performance for a given task.

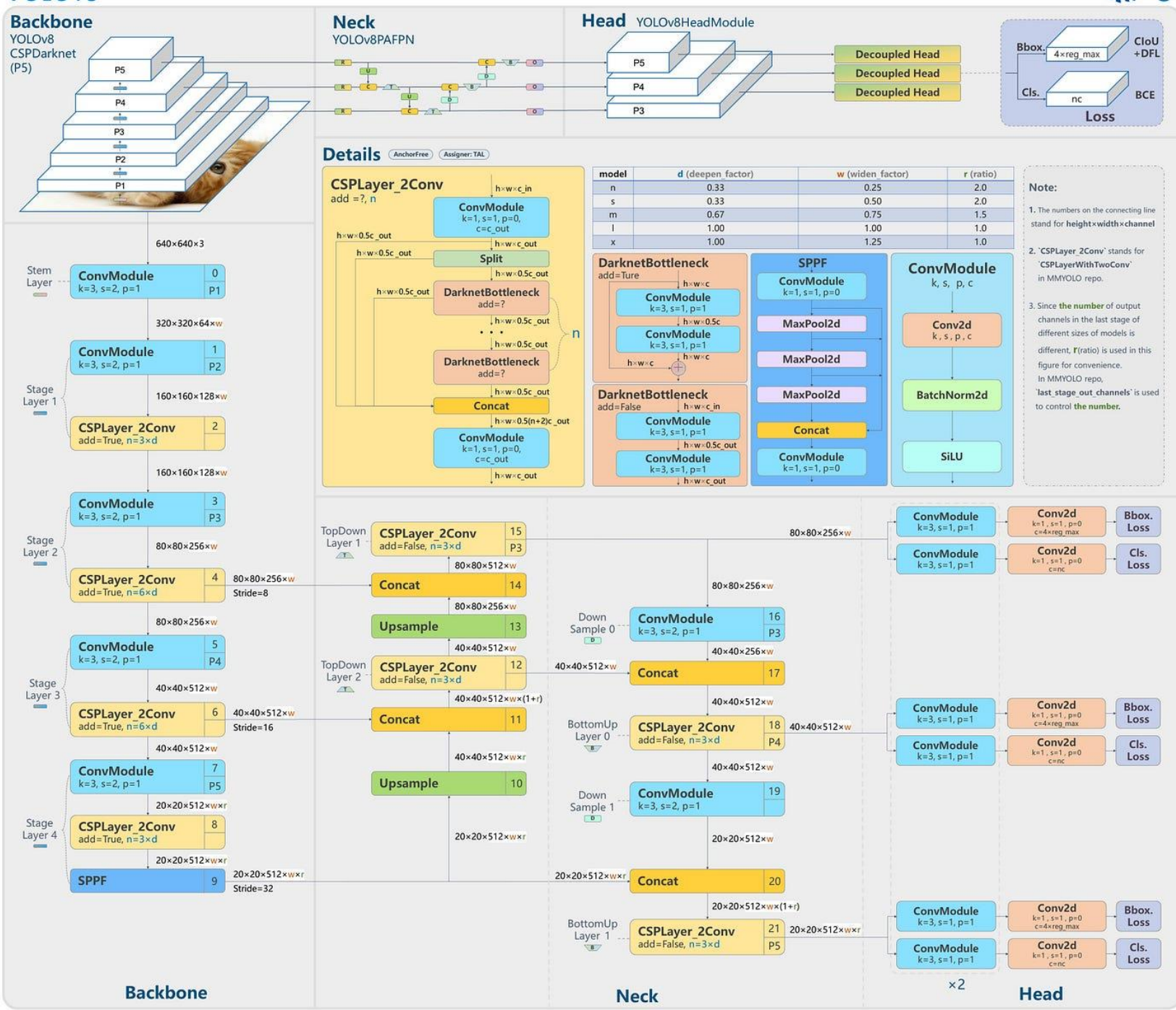
Key	Value	Description
`model`	`None`	path to model file, i.e. yolov8n.pt, yolov8n.yaml
`data`	`None`	path to data file, i.e. coco128.yaml
`epochs`	`100`	number of epochs to train for
`patience`	`50`	epochs to wait for no observable improvement for early stopping of training
`batch`	`16`	number of images per batch (-1 for AutoBatch)
`imgsz`	`640`	size of input images as integer or w,h
`save`	`True`	save train checkpoints and predict results
`save_period`	`-1`	Save checkpoint every x epochs (disabled if < 1)
`cache`	`False`	True/ram, disk or False. Use cache for data loading

Train - Ultralytics YOLOv8 Docs

2. Deep Learning(YOLO 8)

device	None	device to run on, i.e. cuda device=0 or device=0,1,2,3 or device=cpu
workers	8	number of worker threads for data loading (per RANK if DDP)
project	None	project name
name	None	experiment name
exist_ok	False	whether to overwrite existing experiment
pretrained	False	whether to use a pretrained model
optimizer	'SGD'	optimizer to use, choices=['SGD', 'Adam', 'AdamW', 'RMSProp']
verbose	False	whether to print verbose output
seed	0	random seed for reproducibility
deterministic	True	whether to enable deterministic mode
single_cls	False	train multi-class data as single-class
rect	False	rectangular training with each batch collated for minimum padding
cos_lr	False	use cosine learning rate scheduler
close_mosaic	0	(int) disable mosaic augmentation for final epochs
resume	False	resume training from last checkpoint
amp	True	Automatic Mixed Precision (AMP) training, choices=[True, False]
fraction	1.0	dataset fraction to train on (default is 1.0, all images in train set)

YOLOv8


Backbone
YOLOv8
CSPDarknet
(P5)
P5
P4
P3
P2
P1
Neck
YOLOv8PAFPN
Head
YOLOv8HeadModule
P5
P4
P3
Decoupled Head
Decoupled Head
Decoupled Head
Bbox.
4×reg_max
CIoU
+DFL
Cls.
nc
BCE
Loss
Details
AnchorFree
Assigner: TAL
CSPLayer_2Conv
add =?, n
h×w×c_in
ConvModule
k=1, s=1, p=0,
c=c_out
h×w×0.5c_out
h×w×c_out
Split
h×w×0.5c_out
DarknetBottleneck
add=?
h×w×0.5c_out
h×w×0.5c_out
DarknetBottleneck
add=?
h×w×0.5c_out
Concat
h×w×0.5(n+2)c_out
ConvModule
k=1, s=1, p=0,
c=c_out
h×w×c_out
n
model
d (deepen_factor)
w (widen_factor)
r (ratio)
n
0.33
0.25
2.0
s
0.33
0.50
2.0
m
0.67
0.75
1.5
l
1.00
1.00
1.0
x
1.00
1.25
1.0
DarknetBottleneck
add=Ture
h×w×c
ConvModule
k=3, s=1, p=1
h×w×0.5c
ConvModule
k=3, s=1, p=1
h×w×c
h×w×c
DarknetBottleneck
add=False
h×w×c_in
ConvModule
k=3, s=1, p=1
h×w×0.5c_out
ConvModule
k=3, s=1, p=1
h×w×c_out
SPPF
ConvModule
k=1, s=1, p=0
MaxPool2d
MaxPool2d
MaxPool2d
Concat
ConvModule
k=1, s=1, p=0
ConvModule
k, s, p, c
Conv2d
k , s , p , c
BatchNorm2d
SiLU
Note:
1. The numbers on the connecting line stand for height×width×channel
2. `CSPLayer_2Conv` stands for `CSPLayerWithTwoConv` in MMYOLO repo.
3. Since the number of output channels in the last stage of different sizes of models is different, r(ratio) is used in this figure for convenience. In MMYOLO repo, `last_stage_out_channels` is used to control the number.
640×640×3
Stem Layer
ConvModule
k=3, s=2, p=1
0
P1
320×320×64×w
ConvModule
k=3, s=2, p=1
1
P2
Stage Layer 1
160×160×128×w
CSPLayer_2Conv
add=True, n=3×d
2
160×160×128×w
ConvModule
k=3, s=2, p=1
3
P3
Stage Layer 2
80×80×256×w
CSPLayer_2Conv
add=True, n=6×d
4
80×80×256×w
Stride=8
80×80×256×w
ConvModule
k=3, s=2, p=1
5
P4
Stage Layer 3
40×40×512×w
CSPLayer_2Conv
add=True, n=6×d
6
40×40×512×w
Stride=16
40×40×512×w
ConvModule
k=3, s=2, p=1
7
P5
Stage Layer 4
20×20×512×w×r
CSPLayer_2Conv
add=True, n=3×d
8
20×20×512×w×r
SPPF
9
20×20×512×w×r
Stride=32
Backbone
TopDown Layer 1
CSPLayer_2Conv
add=False, n=3×d
15
P3
80×80×512×w
Concat
14
80×80×256×w
Upsample
13
40×40×512×w
TopDown Layer 2
CSPLayer_2Conv
add=False, n=3×d
12
40×40×512×w×(1+r)
Concat
11
40×40×512×w×r
Upsample
10
20×20×512×w×r
80×80×256×w
Down Sample 0
ConvModule
k=3, s=2, p=1
16
P3
40×40×256×w
40×40×512×w
Concat
17
40×40×512×w
BottomUp Layer 0
CSPLayer_2Conv
add=False, n=3×d
18
P4
40×40×512×w
Down Sample 1
ConvModule
k=3, s=2, p=1
19
20×20×512×w
20×20×512×w×r
Concat
20
20×20×512×w×(1+r)
BottomUp Layer 1
CSPLayer_2Conv
add=False, n=3×d
21
P5
20×20×512×w×r
Neck
80×80×256×w
ConvModule
k=3, s=1, p=1
Conv2d
k=1 , s=1 , p=0
c=4×reg_max
Bbox. Loss
ConvModule
k=3, s=1, p=1
Conv2d
k=1 , s=1 , p=0
c=nc
Cls. Loss
40×40×512×w
ConvModule
k=3, s=1, p=1
Conv2d
k=1 , s=1 , p=0
c=4×reg_max
Bbox. Loss
ConvModule
k=3, s=1, p=1
Conv2d
k=1 , s=1 , p=0
c=nc
Cls. Loss
20×20×512×w×r
ConvModule
k=3, s=1, p=1
Conv2d
k=1 , s=1 , p=0
c=4×reg_max
Bbox. Loss
ConvModule
k=3, s=1, p=1
Conv2d
k=1 , s=1 , p=0
c=nc
Cls. Loss
×2
Head

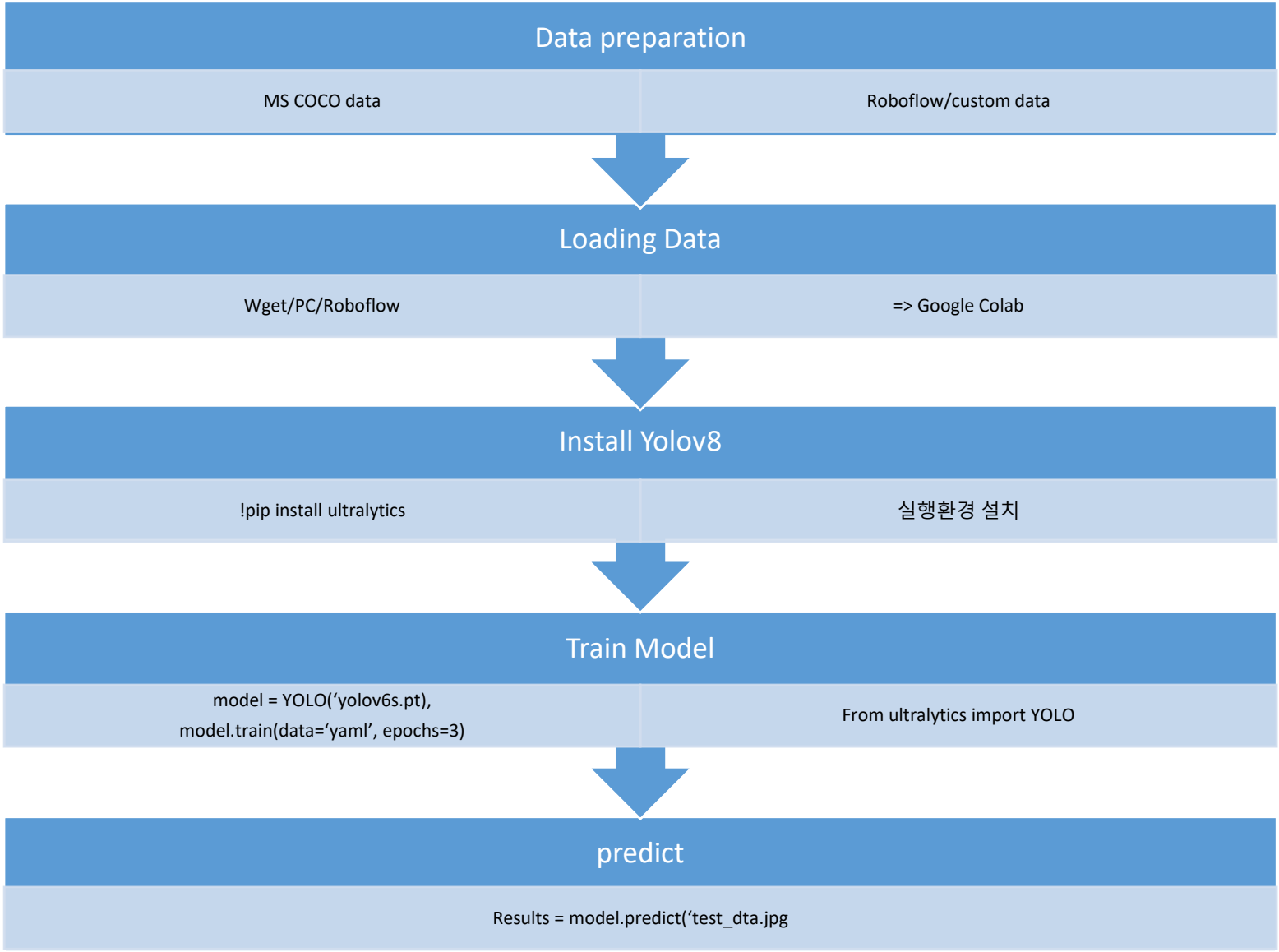
2. Deep Learning(YOLO 8)

Backbone: responsible for extracting high-level features from the input data. It usually consists of a series of convolutional layers and pooling layers. The backbone's purpose is to capture relevant patterns and spatial information in the input data. **=> CNN 특징 추출**

Neck: the intermediate or transition layers. It typically consists of additional convolutional layers or other operations that further process the features extracted by the backbone. The purpose of the neck is to refine and transform the features into a format that is suitable for the specific task the model is designed for. **=>특징의 적절한 변환**

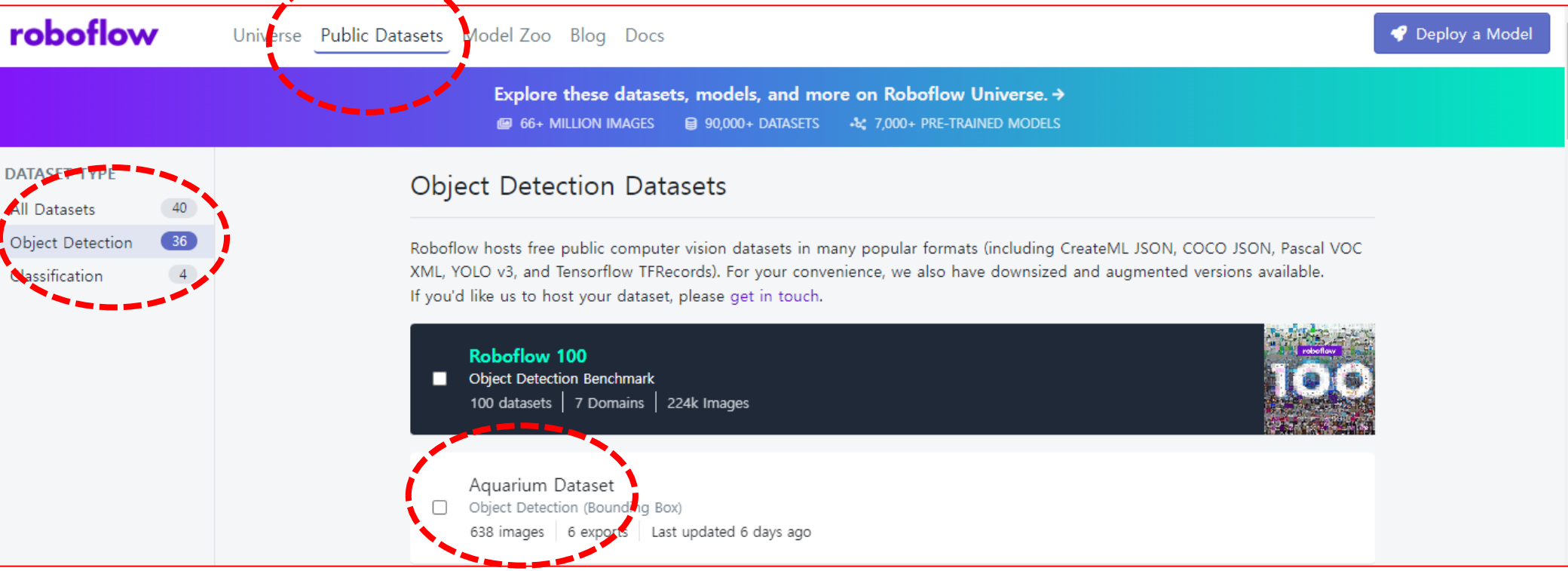
Head: responsible for producing the desired output or predictions. It usually includes one or more fully connected layers or convolutional layers followed by a classifier or regression layer, depending on the task. **=> 예측결과의 분류 및 생성**

2. Deep Learning(YOLO 8)



https://public.roboflow.com/

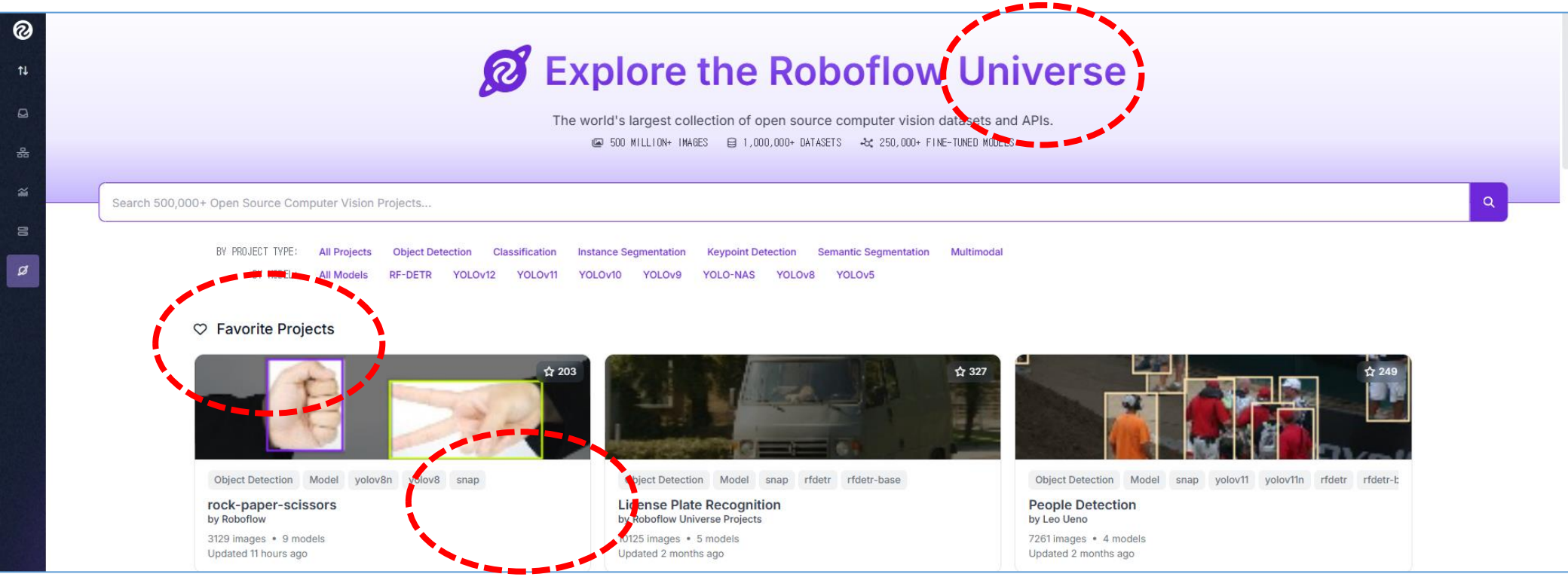
2. Deep Learning(YOLO 8)custom data



Roboflow사이트에서 데이터 코드의 선택 및 설치 방법 !!.

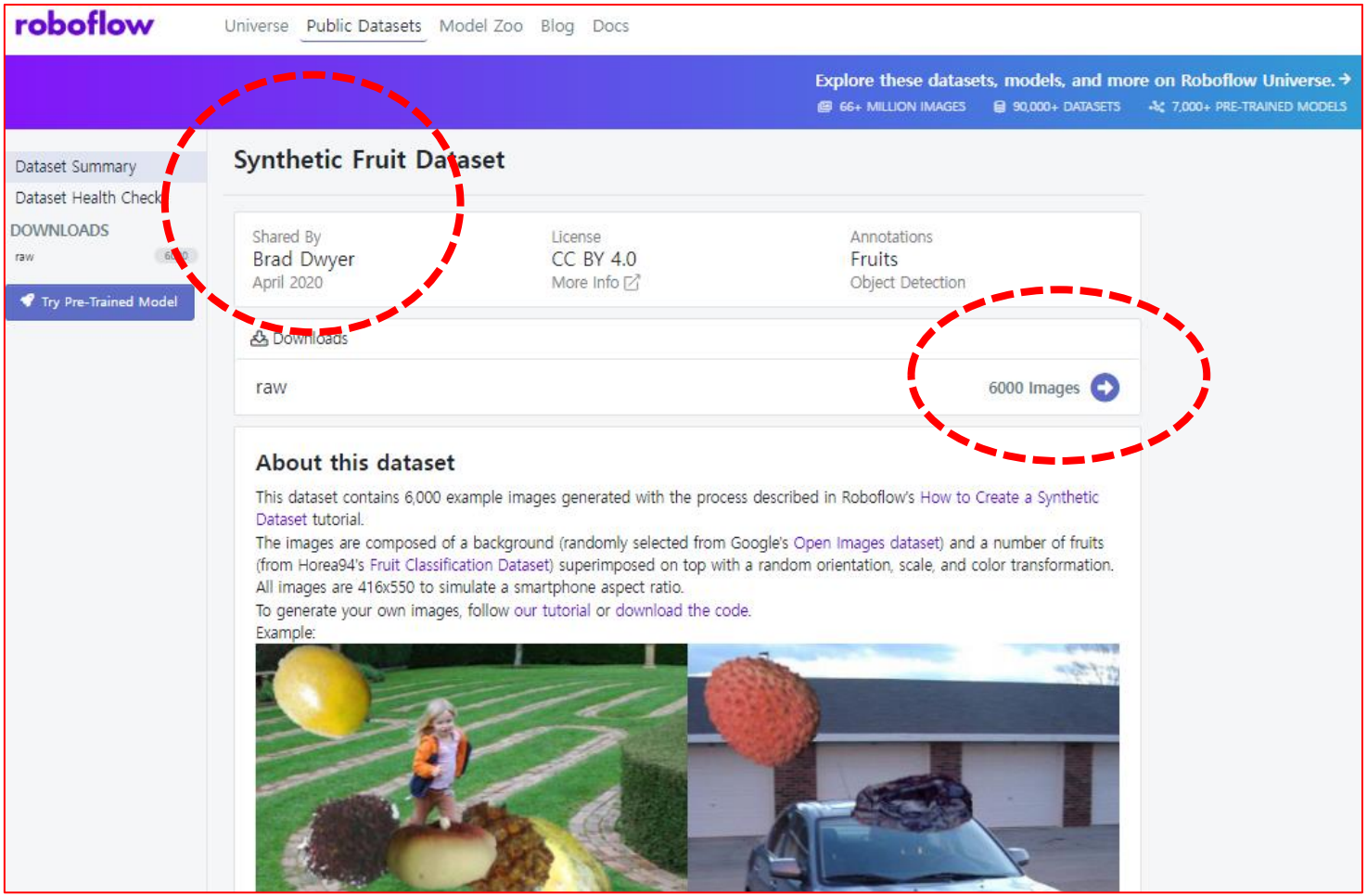
https://universe.roboflow.com/

2. Deep Learning(YOLO 8)custom data



https://public.roboflow.com/object-detection/synthetic-fruit

2. Deep Learning(YOLO 8)custom data



과일 합성 데이터 :
416 * 550 사이즈, 6000개의 샘플 이미지, 임의의 배경에 과일을 합성한 사진 데이터

https://public.roboflow.com/object-detection/synthetic-fruit

2. Deep Learning(YOLO 8)custom data

About this dataset

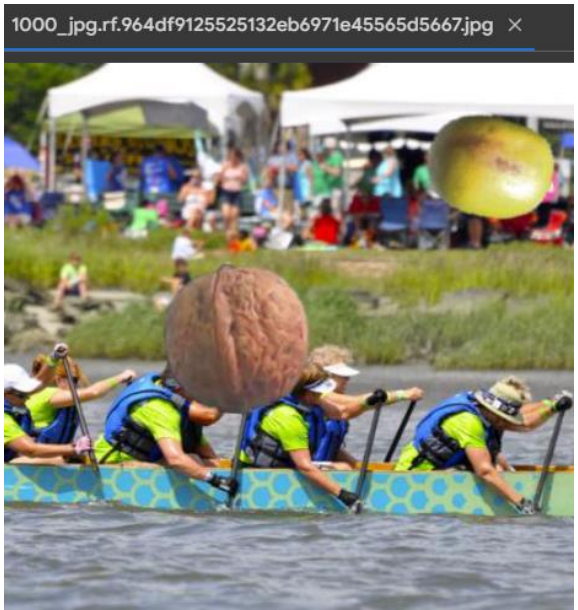
This dataset contains 6,000 example images generated with the process described in Roboflow's How to Create a Synthetic Dataset tutorial.

The images are composed of a background (randomly selected from Google's Open Images dataset) and a number of fruits (from Horea94's Fruit Classification Dataset) superimposed on top with a random orientation, scale, and color transformation.

All images are 416x550 to simulate a smartphone aspect ratio.

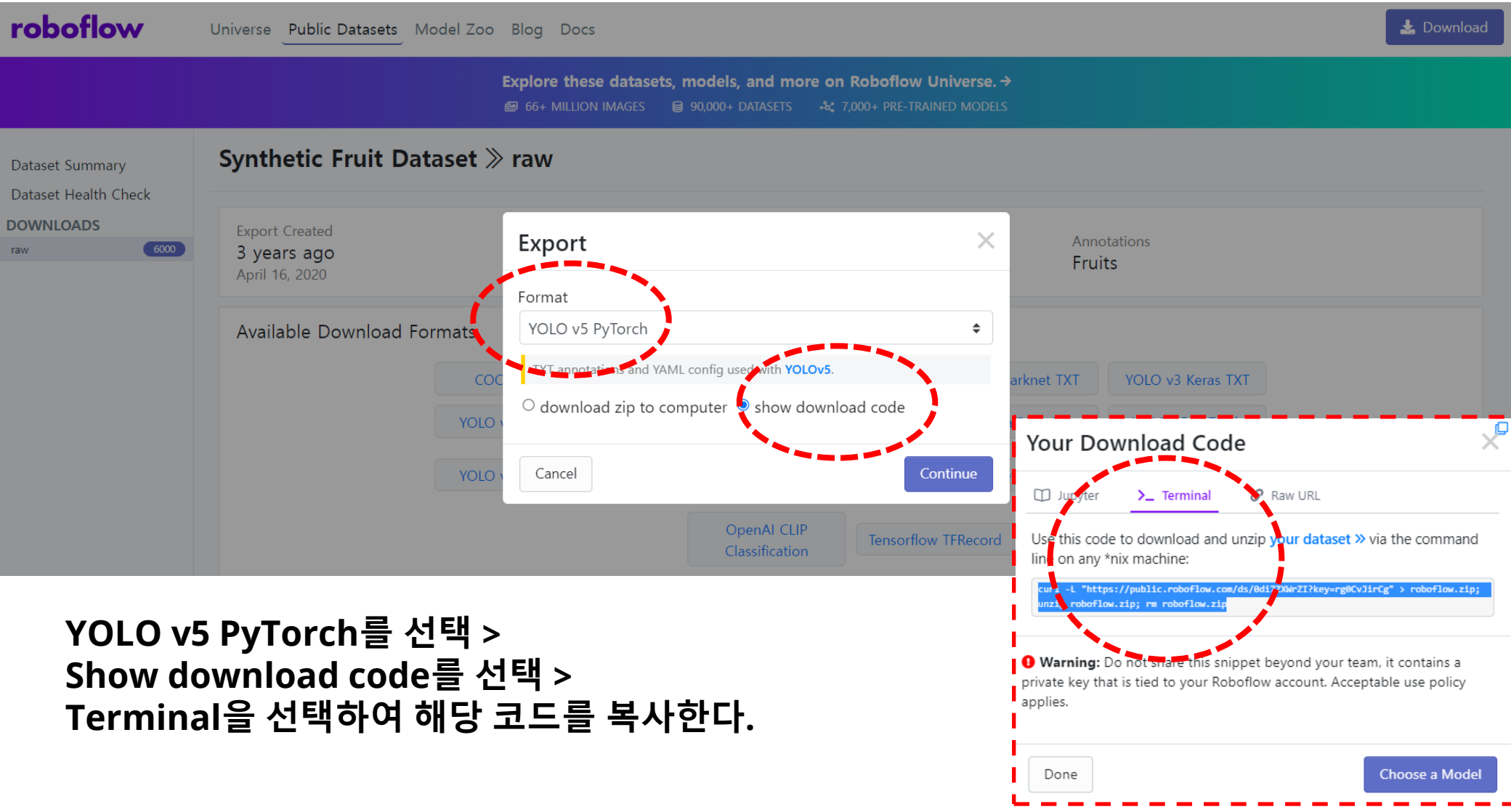
To generate your own images, follow our tutorial or download the code.

Example:



2. Deep Learning(YOLO 8)custom data

```
!curl -L "https://public.roboflow.com/ds/0di7ZXWrZI?key=ㅌㅌㅌㅌ" > roboflow.zip; unzip roboflow.zip; rm roboflow.zip
```



YOLO v5 PyTorch를 선택 >
Show download code를 선택 >
Terminal을 선택하여 해당 코드를 복사한다.

https://public.roboflow.com/object-detection/synthetic-fruit

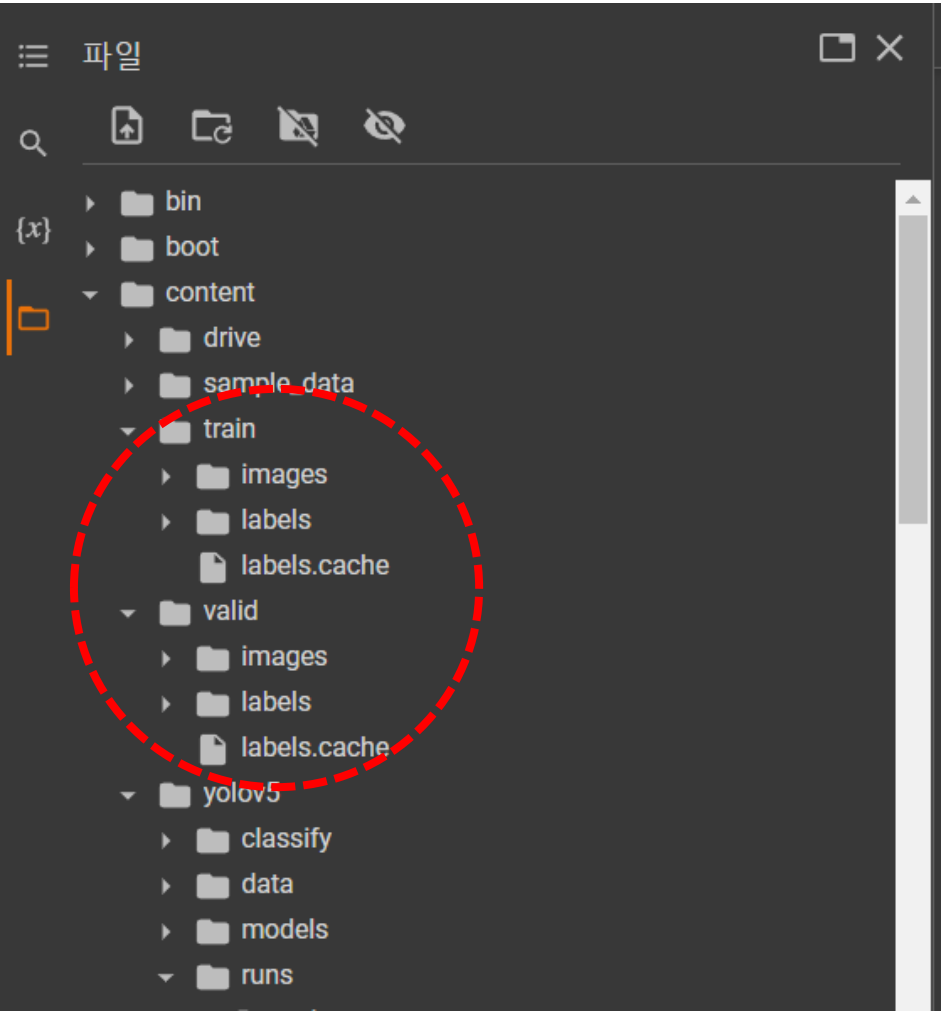
2. Deep Learning(YOLO 8)custom data

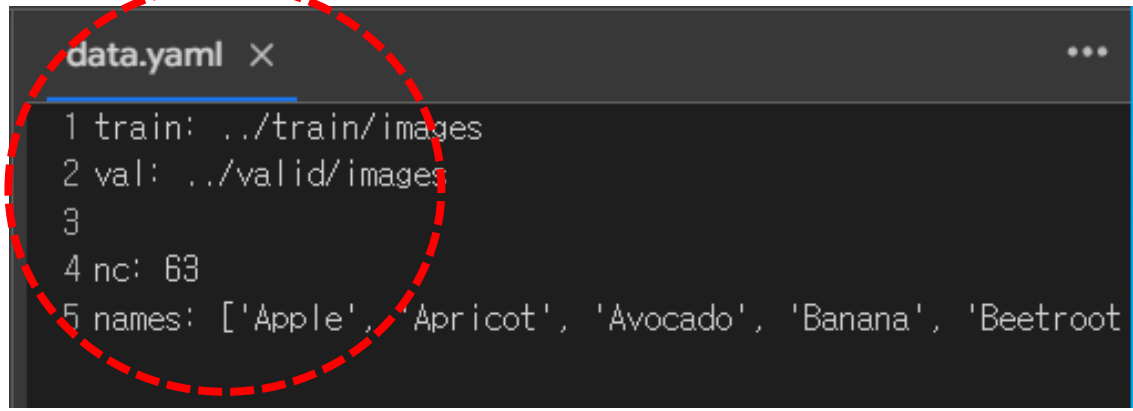
학습용 데이터 셋을 준비하여 압축을 풀어준다

curl -L "https://public.roboflow.com/ds/0di7ZXWrZI?key=rg0CvJirCg" > roboflow.zip; unzip roboflow.zip; rm roboflow.zip

```
[ ]   1 !curl -L "https://public.roboflow.com/ds/0di7ZXWrZI?key=rg0CvJirCg" > roboflow.zip; unzip roboflow.zip; rm roboflow.zip

  스트리밍 출력 내용이 길어서 마지막 5000줄이 삭제되었습니다.
   extracting: train/labels/2126_jpg.rf.31b8e6dc4420419efd473a6107107b3d.txt
   extracting: train/labels/953_jpg.rf.3237a6da506b0e14519398ac5154df88.txt
   extracting: train/labels/536_jpg.rf.32447c4f1a00e8b7f5704ccb90e9bf2c.txt
```

2. Deep Learning(YOLO 8)





데이터 URL을 설치하면(/content/에 설치)

train, valid 폴더와 data.yaml 파일이 생성된다

Yaml 파일에는 데이터에 대한 정보가 포함된다
(위치, 클래스 갯수, 클래스 이름 등)

https://public.roboflow.com/object-detection/aquarium/2

2. Deep Learning(YOLO 8)

YOLO 8

2. Deep Learning(YOLO 8)

```
!pip install ultralytics
```

```
Collecting ultralytics
  Downloading ultralytics-8.3.146-py3-none-any.whl.metadata (37 kB)
Requirement already satisfied: numpy>=1.23.0 in /usr/local/lib/python3.11/dist-packages (from ultralytics) (2.0.2)
Requirement already satisfied: matplotlib>=3.3.0 in /usr/local/lib/python3.11/dist-packages (from ultralytics) (3.10.0)
Requirement already satisfied: opencv-python>=4.6.0 in /usr/local/lib/python3.11/dist-packages (from ultralytics) (4.11.0.86)
Requirement already satisfied: pillow>=7.1.2 in /usr/local/lib/python3.11/dist-packages (from ultralytics) (11.2.1)
Requirement already satisfied: pyyaml>=5.3.1 in /usr/local/lib/python3.11/dist-packages (from ultralytics) (6.0.2)
```

2. Deep Learning(YOLO 8)

```
from ultralytics import YOLO

# [1] 저장된 모델 불러오기 (학습된 모델이나 기본 yolov8n.pt 사용)
model = YOLO('yolov8n.pt')  # or 'runs/detect/train3/weights/best.pt'

# [2] 예측할 미디어 경로 설정
image_path = '/content/man_car_cat.jpg'          # 이미지 파일

# [3] 이미지 예측
results_image = model.predict(source=image_path, save=True, show=True)
print("이미지 예측 결과:", results_image)
```

```
orig_shape: (960, 720)
path: '/content/man_car_cat.jpg'
probs: None
save_dir: 'runs/detect/predict3'
speed: {'preprocess': 8.558590999882654, 'inference': 76.84499599986339, 'postprocess': 313.3790879999965}]
```

2. Deep Learning(YOLO 8)

```
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',
```

2. Deep Learning(YOLO 8)

```
!curl -L "https://public.roboflow.com/ds/c7SehbkfK4?key=5BCTbB5LTm" > roboflow.zip; unzip roboflow.zip; rm roboflow.zip
```

```
  extracting: valid/labels/IMG_8582_MOV-4_jpg.rf.3836024d5cae38ca6ba3512ee98ea3a3.txt
  extracting: valid/labels/IMG_8590_MOV-0_jpg.rf.0ffbdc8627360449c3b1d94adab945c4.txt
  extracting: valid/labels/IMG_8591_MOV-1_jpg.rf.a93070247a5bd3fae05926c0514b1b4c.txt
  extracting: valid/labels/IMG_8593_MOV-1_jpg.rf.9aa3076a6322416b5ce1c892d97a102a.txt
  extracting: valid/labels/IMG_8595_MOV-1_jpg.rf.7ec06740adf2a710a14c479f9bd8db5b.txt
  extracting: valid/labels/IMG_8599_MOV-1_jpg.rf.8b8f5d9d4eacd671546a3025e6f52a0e.txt
```

2. Deep Learning(YOLO 8)

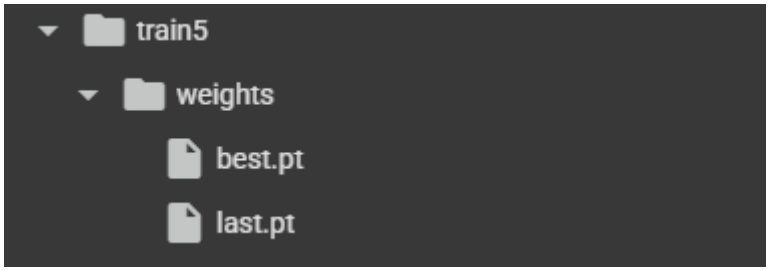
```
!cat data.yaml
```

```
train: ../train/images
val: ../valid/images
test: ../test/images

nc: 7
names: ['fish', 'jellyfish', 'penguin', 'puffin', 'shark', 'starfish', 'stingray']

roboflow:
  workspace: brad-dwyer
  project: aquarium-combined
  version: 2
  license: CC BY 4.0
  url: https://universe.roboflow.com/brad-dwyer/aquarium-combined/dataset/2
```

2. Deep Learning(YOLO 8)

```
model.train(data='/content/data.yaml', epochs=10, batch=32, imgsz=640, lr0=0.001, augment=True, patience=10, save=True)
```

```
fitness: np.float64(0.3881757112000368)
keys: ['metrics/precision(B)', 'metrics/recall(B)', 'metrics/mAP50(B)', 'metrics/mAP50-95(B)']
maps: array([    0.38387,     0.54003,     0.26967,     0.19332,     0.30051,     0.39508,     0.46326])
names: {0: 'fish', 1: 'jellyfish', 2: 'penguin', 3: 'puffin', 4: 'shark', 5: 'starfish', 6: 'stingray'}
plot: True
results_dict: {'metrics/precision(B)': np.float64(0.6724563040086318), 'metrics/recall(B)': np.float64(0.5633853324948074), 'metrics/mAP50(B)': np.float64(0.6086682946379455),
'metrics/mAP50-95(B)': np.float64(0.3636765352624914), 'fitness': np.float64(0.3881757112000368)}
save_dir: PosixPath('runs/detect/train5')
speed: {'preprocess': 0.2139230157515308, 'inference': 7.410926456683715, 'loss': 0.0001843228296713176, 'postprocess': 7.4196420157566925}
task: 'detect'
```



2. Deep Learning(YOLO 8)

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

```
<class 'dict'> 7
{0: 'fish', 1: 'jellyfish', 2: 'penguin', 3: 'puffin', 4: 'shark', 5: 'starfish', 6: 'stingray'}
```

2. Deep Learning(YOLO 8)

```
from ultralytics import YOLO

# [1] best.pt 모델 로드
model = YOLO('runs/detect/train/weights/best.pt')  # 경로를 본인의 best.pt로 수정!

# [2] 예측할 이미지 경로
image_path = '/content/jellyfish.jpg'  # 예측할 이미지 경로

# [3] 예측 수행
results = model.predict(
    source=image_path,  # 이미지 파일 경로 (또는 폴더 경로, 비디오 경로, 0: 웹캠)
    save=True,          # 예측 결과 이미지 저장
    show=True,          # 팝업 창에 결과 출력
    conf=0.5,           # 탐지 신뢰도 threshold
    device=0            # GPU: 0 / CPU: 'cpu'
)

# [4] 결과 확인
for result in results:
    print(result.boxes)  # 감지된 객체의 경계 상자 정보
```

```
image 1/1 /content/jellyfish.jpg: 448x640 (no detections), 14.5ms
Speed: 6.2ms preprocess, 14.5ms inference, 1.2ms postprocess per image at shape (1, 3, 448, 640)
Results saved to runs/detect/predict4
ultralytics.engine.results.Boxes object with attributes:

cls: tensor([], device='cuda:0')
conf: tensor([], device='cuda:0')
data: tensor([], device='cuda:0', size=(0, 6))
id: None
is_track: False
orig_shape: (854, 1280)
shape: torch.Size([0, 6])
xywh: tensor([], device='cuda:0', size=(0, 4))
xywhn: tensor([], device='cuda:0', size=(0, 4))
xyxy: tensor([], device='cuda:0', size=(0, 4))
xyxyn: tensor([], device='cuda:0', size=(0, 4))
```

웹캠이 설치된 노트북에서 별도의 vscode실행:ultalytics, opencv-python 설치

2. Deep Learning(YOLO 8) 실시간 객체 탐지

```
import cv2
from ultralytics import YOLO  # YOLOv8 라이브러리

# 1. YOLO 모델 불러오기 (사전학습된 모델 사용)
model = YOLO('yolov8n.pt')  # 경량화 모델로 빠르게 확인 가능

# 2. 웹캠 열기 (기본 카메라 index=0)
cap = cv2.VideoCapture(0)

# 3. 반복적으로 프레임 읽고 객체 감지
while True:
    ret, frame = cap.read()
    if not ret:
        break

    # 4. YOLO 예측
    results = model(frame, show=False, stream=True)  # stream=True: 실시간 감지

    # 5. 결과 프레임 출력
    for result in results:
        annotated_frame = result.plot()  # 예측결과로 이미지 그리기
        cv2.imshow('YOLOv8 Real-time Detection', annotated_frame)

    # 6. 종료 조건 (q 키를 누르면 종료)
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

# 7. 자원 해제
cap.release()
cv2.destroyAllWindows()
```

https://inpa.tistory.com/entry/YAML-%F0%9F%93%9A-yaml-%EA%B0%9C%EB%85%90-%EB%AC%B8%EB%B2%95-%EC%9D%B4%ED%95%B4%ED%95%98%EA%B8%B0-%F0%9F%92%AF-%EC%B4%9D%EC%A0%95%EB%A6%AC

2. Deep Learning (YOLO 8)

YAML :

xml과 json 포맷과 같이 타 시스템 간에 데이터를 주고받을 때 약속된 포맷(규칙)이 정의되어있는 또 하나의 파일 형식

yaml은 주로 Doker Compose, Kubernetes, Flutter, Spring boot 프로젝트에서 설정 파일을 정의할때 자주 애용된다.

XML

```xml
<Servers>
  <Server>
    <name>Server1</name>
    <owner>Prajwal</owner>
    <status>active</status>
  </Server>
  <Server>
    <name>Server2</name>
    <owner>John</owner>
    <status>inactive</status>
  </Server>
</Servers>
```

JSON

```json
{
  "Servers": [
    {
      "name": "Server1",
      "owner": "Prajwal",
      "status": "active"
    },
    {
      "name": "Server2",
      "owner": "John",
      "status": "inactive"
    }
  ]
}
```

YAML

```yaml
Servers:
  - name: Server1
    owner: Prajwal
    status: active
  - name: Server2
    owner: John
    status: inactive
```

2. Deep Learning (YOLO 8)

YAML :

객체 탐지(Object Detection)와 같은 작업에 사용되는 설정 파일.
모델 아키텍처, 데이터 경로, 하이퍼파라미터, 학습 및 평가 설정 등 설정 값을 포함

- 데이터 경로: 학습 및 검증 데이터셋의 경로와 형식을 지정.
- 모델 아키텍처: 사용할 모델의 구조와 레이어를 정의.
- 하이퍼파라미터: 학습률, 배치 크기, 가중치 감소 등 관련된 하이퍼파라미터를 설정.
- 데이터 전처리: 이미지 크기 조정, 데이터 증강 등의 전처리 단계를 정의.
- 학습 설정: 학습 반복 횟수(epoch), 옵티마이저 종류, 학습률 스케줄링 등 지정.
- 검증 설정: 평가 지표, 검증 주기 등 검증에 관련된 설정을 지정.
- 클래스 정보: 탐지할 객체 클래스의 목록을 정의.

2. Deep Learning (YOLO 8)

객체 탐지에서 YAML 파일을 사용하는 이유

1.구성의 통합과 일관성: YAML 파일을 사용하면 객체 탐지 시스템의 구성 요소를 하나의 파일에 통합. 모델 아키텍처, 데이터 경로, 하이퍼파라미터, 학습 및 평가 설정 등과 같은 여러 가지 요소를 하나의 장소에서 관리.

2.유연성과 재사용성: YAML 파일은 수정 가능한 설정 값을 포함하므로 사용자는 해당 값을 변경하여 다양한 실험을 수행. 모델의 하이퍼파라미터, 데이터 경로, 전처리 단계 등.

3.가독성과 관리 용이성: YAML 파일은 사람이 읽고 이해하기 쉬운 형식. 들여쓰기를 사용하여 계층적인 구조를 표현하며, 키-값 쌍의 형태로 데이터를 표현.

4.설정의 분리와 추상화: YAML 파일은 설정 값을 분리하고 추상화할 수 있는 기능을 제공. 예를 들어, 데이터 경로, 모델 아키텍처, 학습 설정 등을 YAML 파일로 정의하면 해당 값들을 추상화하여 여러 실험 또는 프로젝트에서 재사용할 수 있다.

https://public.roboflow.com/object-detection/aquarium/2

2. Deep Learning(YOLO 8)

roboflow Universe Public Datasets Model Zoo Blog Docs

Explore these datasets, models, and more on Roboflow Universe. →
66+ MILLION IMAGES 90,000+ DATASETS 7,000+ PRE-TRAINED MODELS

Dataset Summary
Dataset Health Check
DOWNLOADS
raw-1024 638
Try Pre-Trained Model

Aquarium Dataset

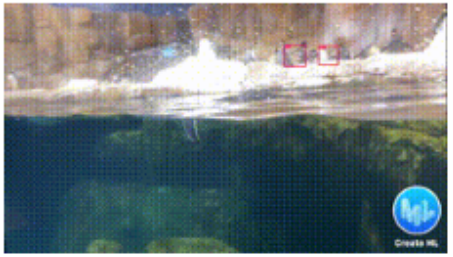
Shared By
Roboflow
November 2020

License
CC BY 4.0
More Info

Annotations
creatures
Object Detection

Downloads

raw-1024 638 Images



Dataset Details

This dataset consists of 638 images collected by Roboflow from two aquariums in the United States: The Henry Doorly Zoo in Omaha (October 16, 2020) and the National Aquarium in Baltimore (November 14, 2020). The images were labeled for object detection by the Roboflow team (with some help from SageMaker Ground Truth). Images and annotations are released under a Creative Commons By-Attribution license. You are free to use them for any purposes personal, commercial, or academic provided you give acknowledgement of their source.

https://public.roboflow.com/object-detection/aquarium/2

2. Deep Learning(YOLO 8)

Dataset Summary

Dataset Analytics

DOWNLOADS

raw-1024 638

Try Pre-Trained Model

Aquarium Dataset

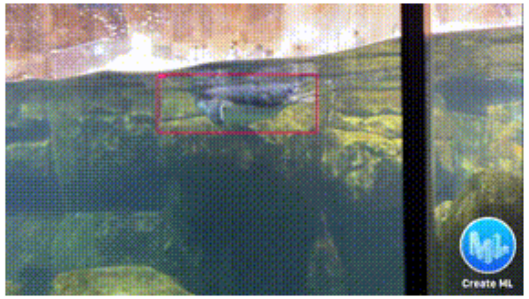
Shared By
Roboflow
November 2020

License
CC BY 4.0
More Info

Annotations
creatures
Object Detection

Downloads

raw-1024 638 Images



Dataset Details

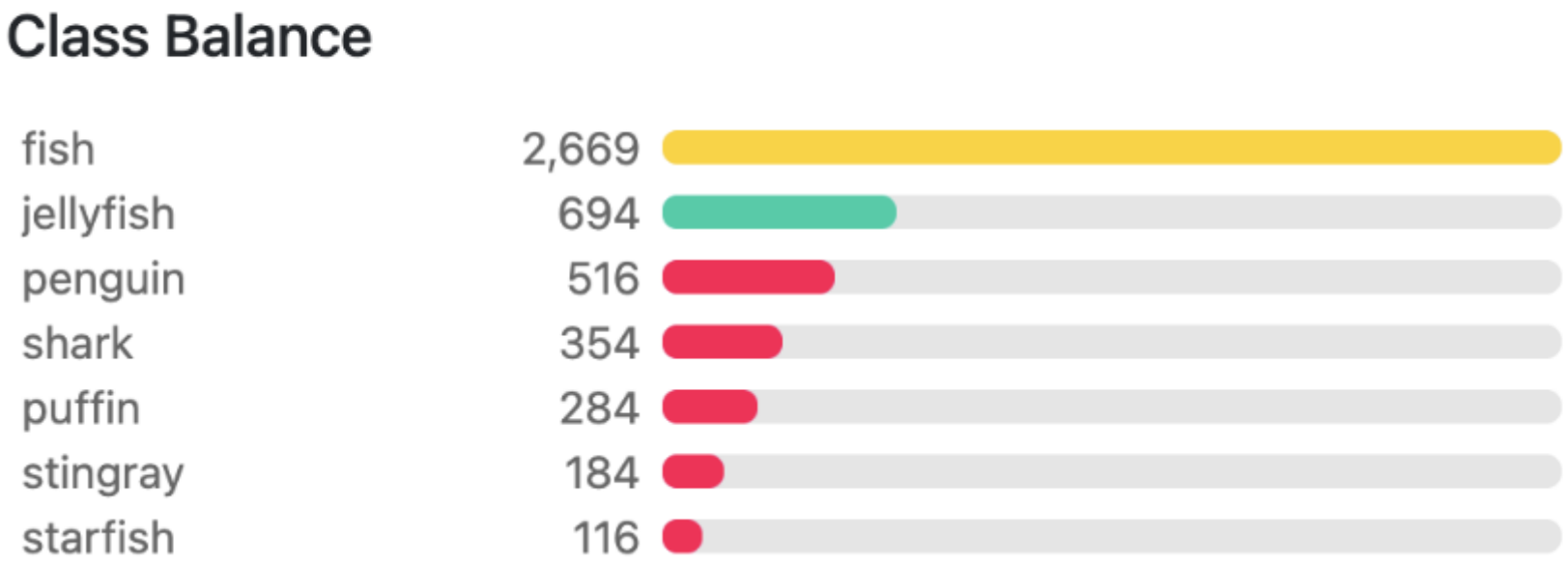
This dataset consists of 638 images collected by Roboflow from two aquariums in the United States: The Henry Doorly Zoo in Omaha (October 16, 2020) and the National Aquarium in Baltimore (November 14, 2020). The images were labeled for object detection by the Roboflow team (with some help from SageMaker Ground Truth). Images and annotations are released under a Creative Commons By-Attribution license. You are free to use them for any purposes personal, commercial, or academic provided you give acknowledgement of their source.

https://public.roboflow.com/object-detection/aquarium/2

2. Deep Learning(YOLO 8)

Class Breakdown

The following classes are labeled: fish, jellyfish, penguins, sharks, puffins, stingrays, and starfish. Most images contain multiple bounding boxes.

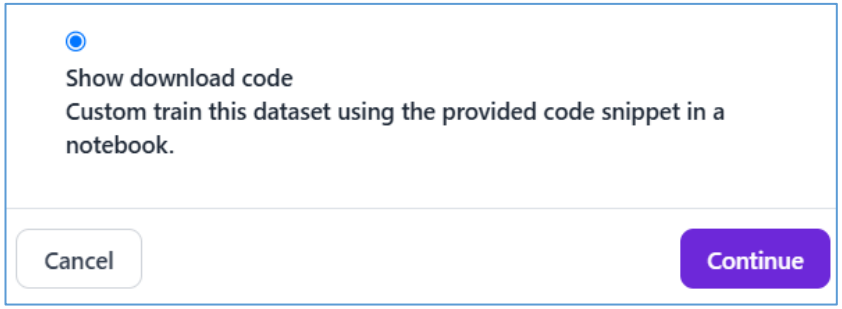


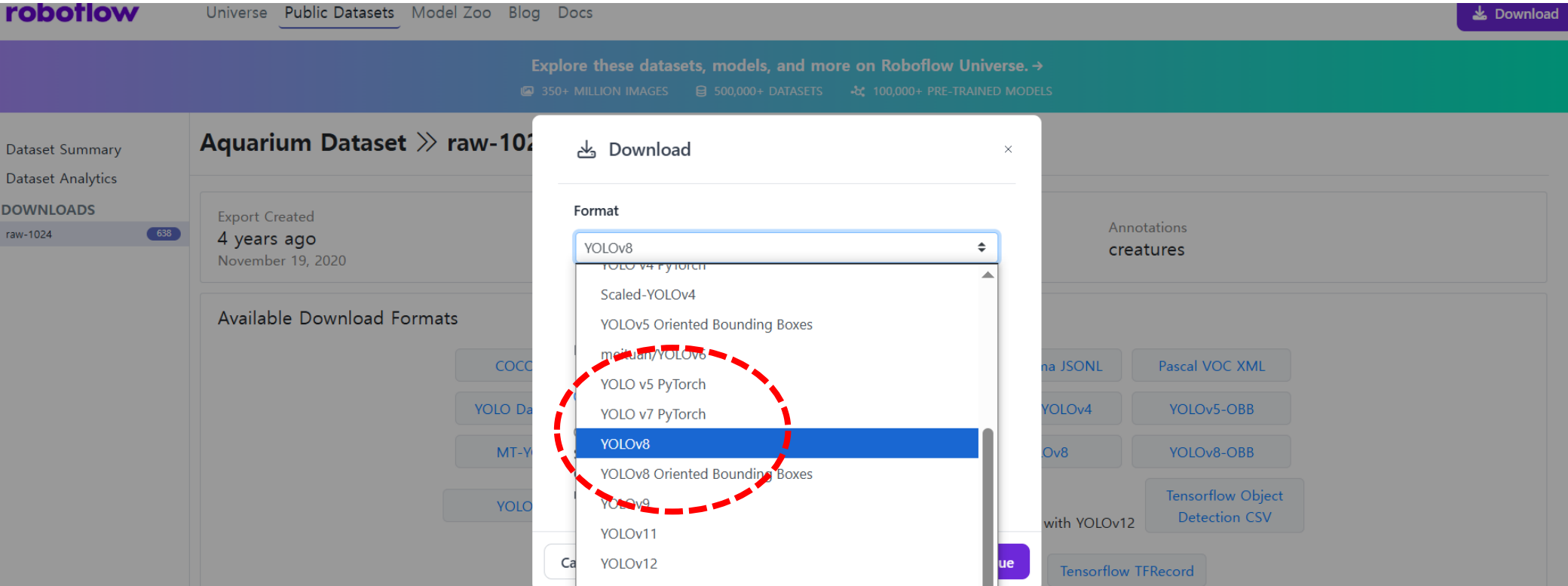
2. Deep Learning(YOLO 5)

```
train: ../train/images
val: ../valid/images
test: ../test/images

nc: 7
names: ['fish', 'jellyfish', 'penguin', 'puffin', 'shark', 'starfish', 'stingray']

roboflow:
  workspace: brad-dwyer
  project: aquarium-combined
  version: 2
  license: CC BY 4.0
  url: https://universe.roboflow.com/brad-dwyer/aquarium-combined/dataset/2
```

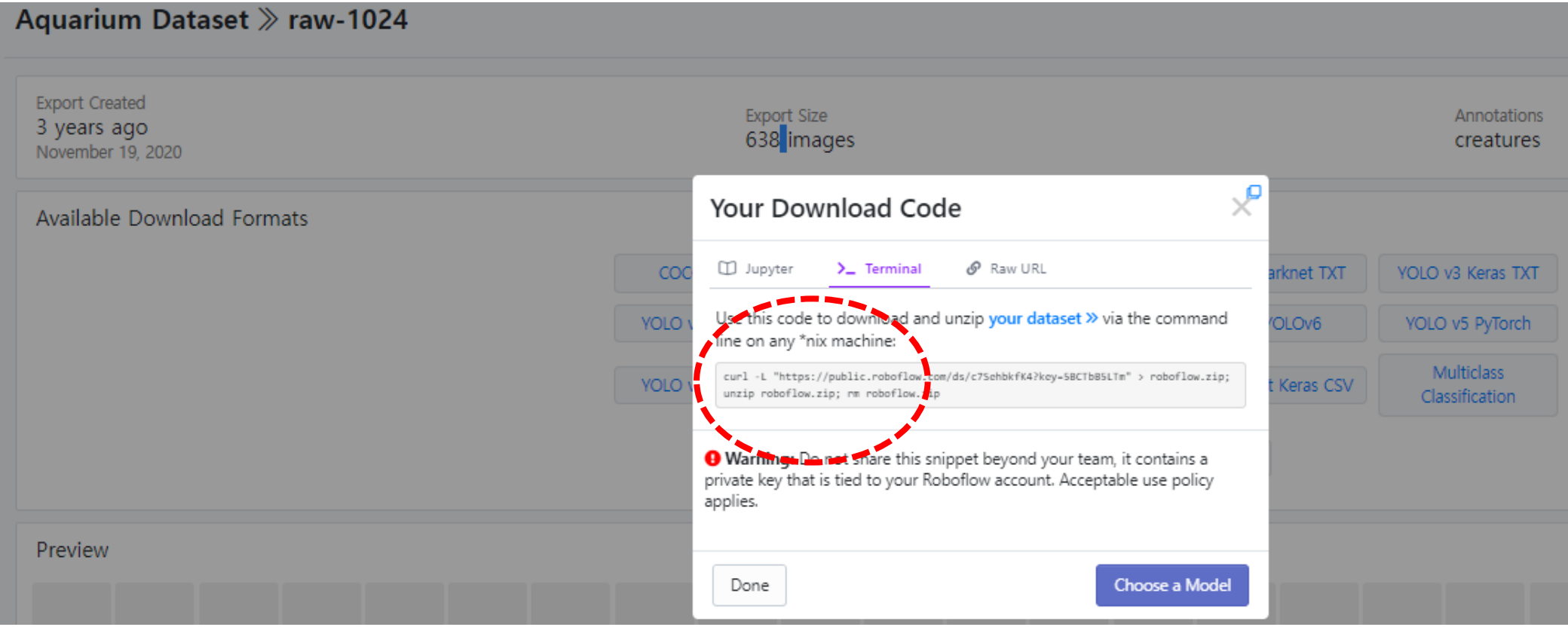
https://public.roboflow.com/object-detection/aquarium/2

2. Deep Learning(YOLO 8)





2. Deep Learning(YOLO 8)



아쿠아 데이터셋을 YOLO8로 설정 후 다운로드 코드 복사!!

2. Deep Learning(YOLO 8)

```
[1]  1 %cd /content/

/content
```

```
[2]  1 !pwd

/content
```

yolov8은 validation자료를 별도 구성한다

```
[3]  1 !curl -L "https://public.roboflow.com/ds/c7SehbkfK4?key=5BCTbB5LTm" > roboflow.zip; unzip roboflow.zip; rm roboflow.zip
  extracting: valid/labels/IMG_2333_jpeg_jpg.rf.0dfdf7c17f438c4a2eacebdfef82bdeb.txt
  extracting: valid/labels/IMG_2334_jpeg_jpg.rf.b545bdad952bf47fbadb8ee504e52c36.txt
  extracting: valid/labels/IMG_2337_jpeg_jpg.rf.8c0fdb28fa8bfd6adf906153bb4c90a2.txt
  extracting: valid/labels/IMG_2339_jpeg_jpg.rf.f31a698f6d75d00ce27b24801b6d88dd.txt
  extracting: valid/labels/IMG_2342_jpeg_jpg.rf.f36e481b4e01c2e76e0b27e494682873.txt
  extracting: valid/labels/IMG_2345_jpeg_jpg.rf.1c32346981ba9d501078eb82f2c63555.txt
  extracting: valid/labels/IMG_2348_jpeg_jpg.rf.79943ee250febaea3c2faaca7f175a52.txt
  extracting: valid/labels/IMG_2353_jpeg_jpg.rf.6f07383174c9ac1c07641f16ee637f33.txt
```

아쿠아 데이터셋을 YOLO8로 다운로드하여 설치 후 ultralyics의 설치 !!

2. Deep Learning(YOLO 8)

```
[4]  1 !pip install ultralytics

Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Collecting ultralytics
  Downloading ultralytics-8.0.110-py3-none-any.whl (592 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 592.7/592.7 kB 14.8 MB/s eta 0:00
Requirement already satisfied: matplotlib>=3.2.2 in /usr/local/lib/python3.10/dist-packages (from ultralytics) (3.7
Requirement already satisfied: opencv-python>=4.6.0 in /usr/local/lib/python3.10/dist-packages (from ultralytics) (4
Requirement already satisfied: Pillow>=7.1.2 in /usr/local/lib/python3.10/dist-packages (from ultralytics) (8.4.0)
Requirement already satisfied: PyYAML>=5.3.1 in /usr/local/lib/python3.10/dist-packages (from ultralytics) (6.0)
Requirement already satisfied: requests>=2.23.0 in /usr/local/lib/python3.10/dist-packages (from ultralytics) (2.27
Requirement already satisfied: scipy>=1.4.1 in /usr/local/lib/python3.10/dist-packages (from ultralytics) (1.10.1)
Requirement already satisfied: torch>=1.7.0 in /usr/local/lib/python3.10/dist-packages (from ultralytics) (2.0.1+cu
Requirement already satisfied: torchvision>=0.8.1 in /usr/local/lib/python3.10/dist-packages (from ultralytics) (0.
Requirement already satisfied: tqdm>=4.64.0 in /usr/local/lib/python3.10/dist-packages (from ultralytics) (4.65.0)
Requirement already satisfied: pandas>=1.1.4 in /usr/local/lib/python3.10/dist-packages (from ultralytics) (1.5.3)
```

YOLO8 : ultralyics의 설치 !!

2. Deep Learning(YOLO 8)

```
[5]  1 from ultralytics import YOLO
     2 model = YOLO('yolov8n.pt')
     3
     4 print(type(model.names), len(model.names))
     5 print(model.names)
```

```
Downloading https://github.com/ultralytics/assets/releases/download/v0.0.0/yolov8n.pt to yolov8n.pt...
100%|██████████| 6.23M/6.23M [00:00<00:00, 115MB/s]<class 'dict'> 80
{0: 'person', 1: 'bicycle', 2: 'car', 3: 'motorcycle', 4: 'airplane', 5: 'bus', 6: 'train', 7: 'truck', 8: 'boat', 9: 'traffic light',
```

```
[6]  1 !cat data.yaml
```

```
train: ../train/images
val: ../valid/images
test: ../test/images

nc: 7
names: ['fish', 'jellyfish', 'penguin', 'puffin', 'shark', 'starfish', 'stingray']
```

YOLO8 : YOLO 모델을 선택하고 yaml화일 지정 !
pretrained model인 **yolov8n.pt**는 80개의 객체에 대해 사전학습된 모델

Data.yaml은 다운로드한 aqua data set의 정보를 보유
(**!cat** 명령어는 일반적으로 **파일의 내용을 출력**)

2. Deep Learning(YOLO 8)

```
1 model.train(data ='/content/data.yaml' epochs=10, batch=32, imgsz=416)
```

```
Ultralytics YOLOv8.0.110 🚀 Python-3.10.11 torch-2.0.1+cu118 CUDA:0 (Tesla T4, 15102MiB)
yolo/engine/trainer: task=detect, mode=train, model=yolov8n.pt, data=/content/data.yaml, epochs=10, patience=50, batch=32, imgsz=416,
Downloading https://ultralytics.com/assets/Arial.ttf to /root/.config/Ultralytics/Arial.ttf...
100%|██████████| 755k/755k [00:00<00:00, 22.3MB/s]
Overriding model.yaml nc=80 with nc=7

                   from  n    params  module                                       arguments
  0                  -1  1       464  ultralytics.nn.modules.conv.Conv             [3, 16, 3, 2]
  1                  -1  1      4672  ultralytics.nn.modules.conv.Conv             [16, 32, 3, 2]
  2                  -1  1      7360  ultralytics.nn.modules.block.C2f             [32, 32, 1, True]
  3                  -1  1     18560  ultralytics.nn.modules.conv.Conv             [32, 64, 3, 2]
  4                  -1  2     49664  ultralytics.nn.modules.block.C2f             [64, 64, 2, True]
  5                  -1  1     73984  ultralytics.nn.modules.conv.Conv             [64, 128, 3, 2]
  6                  -1  2    197632  ultralytics.nn.modules.block.C2f             [128, 128, 2, True]
  7                  -1  1    295424  ultralytics.nn.modules.conv.Conv             [128, 256, 3, 2]
  8                  -1  1    460288  ultralytics.nn.modules.block.C2f             [256, 256, 1, True]
  9                  -1  1    164608  ultralytics.nn.modules.block.SPPF            [256, 256, 5]
 10                  -1  1         0  torch.nn.modules.upsampling.Upsample         [None, 2, 'nearest']
```

YOLO8 : **지정된 aqua data의 yaml파일과 지정된 파라미터 설정으로 학습을 진행**

2. Deep Learning(YOLO 8)

```
model.train(data ='/content/data.yaml' epochs=10, batch=32, imgsz=416)
```

```
Ultralytics YOLOv8.0.110 🚀 Python-3.10.11 torch-2.0.1+cu118 CUDA:0 (Tesla T4, 15102MiB)
yolo/engine/trainer: task=detect, mode=train, model=yolov8n.pt, data=/content/data.yaml, epochs=10, patience=50, batch=32, imgsz=416,
Downloading https://ultralytics.com/assets/Arial.ttf to /root/.config/Ultralytics/Arial.ttf...
100%|██████████| 755k/755k [00:00<00:00, 22.3MB/s]
Overriding model.yaml nc=80 with nc=7
```

```
Validating runs/detect/train/weights/best.pt...
Ultralytics YOLOv8.0.110 🚀 Python-3.10.11 torch-2.0.1+cu118 CUDA:0 (Tesla T4, 15102MiB)
Model summary (fused): 168 layers, 3007013 parameters, 0 gradients
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 2/2 [00:02<00:00,  1.48s/it]
                   all        127        909      0.474      0.419      0.409      0.218
                  fish        127        459      0.674      0.437      0.539       0.25
             jellyfish        127        155      0.676      0.832        0.8      0.417
               penguin        127        104       0.44       0.26      0.304      0.125
                puffin        127         74      0.256      0.107     0.0705     0.0365
                 shark        127         57      0.316      0.298      0.256       0.15
              starfish        127         27      0.445      0.481      0.397      0.233
              stingray        127         33      0.507      0.515      0.498      0.313
Speed: 1.1ms preprocess, 1.1ms inference, 0.0ms loss, 3.4ms postprocess per image
Results saved to runs/detect/train
```

YOLO8 : **지정된 aqua data의 yaml파일과 지정된 파라미터 설정으로 학습을 진행**

2. Deep Learning(YOLO 8)

```
10 epochs completed in 0.019 hours.
Optimizer stripped from runs/detect/train/weights/last.pt, 22.5MB
Optimizer stripped from runs/detect/train/weights/best.pt, 22.5MB

Validating runs/detect/train/weights/best.pt...
Ultralytics 8.3.107 🚀 Python-3.11.12 torch-2.6.0+cu124 CUDA:0 (Tesla T4, 15095MiB)
Model summary (fused): 72 layers, 11,128,293 parameters, 0 gradients, 28.5 GFLOPs
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████|
                   all        127        909      0.686      0.623      0.672      0.396
                  fish         63        459      0.741      0.667       0.72      0.383
             jellyfish          9        155      0.789      0.897      0.915      0.501
               penguin         17        104       0.59      0.683      0.597      0.275
                puffin         15         74      0.618      0.285      0.429      0.188
                 shark         28         57      0.644      0.614      0.663      0.398
              starfish         17         27      0.852      0.638      0.702      0.518
              stingray         23         33       0.57      0.576       0.68      0.512
Speed: 0.1ms preprocess, 2.2ms inference, 0.0ms loss, 3.7ms postprocess per image
Results saved to runs/detect/train
```

2. Deep Learning(YOLO 8)

1. 모델 요약
층 수: 72층.
파라미터 수: 모델에는 11,128,293개의 파라미터
GFLOPs: 28.5 GFLOPs (Giga Floating Point Operations).
모델의 계산 복잡도를 나타내며, GFLOPs 수가 클수록 더 많은 계산 자원이 필요

2. 검증 결과
모델은 127개의 이미지에서 909개의 객체를 검증
평가 지표로는 정밀도(Precision), 재현율(Recall), mAP (mean Average Precision)을 사용

정밀도(Precision): 0.686모델이 탐지한 객체 중 **실제로 정확한 객체**인 비율
재현율(Recall): 0.623실제 객체 중 모델이 **탐지한 객체**의 비율
mAP50 (IoU 0.5 기준 mAP): 0.672
IoU(Intersection over Union)가 0.5 이상일 때의 평균 정밀도
mAP50-95 (IoU 0.5부터 0.95까지 평균 mAP): 0.396다양한 IoU 기준(0.5부터 0.95까지)에서의 평균 정밀도. **이 값이 낮다는 것은 모델의 성능이 일정하지 않다**는 의미.

2. Deep Learning(YOLO 8)

Fish (물고기) :
정밀도: 0.741재현율: 0.667mAP50: 0.72mAP50-95: 0.383
물고기 탐지는 좋은 편으로, 높은 정밀도와 재현율

Jellyfish (해파리)
정밀도: 0.789재현율: 0.897mAP50: 0.915mAP50-95: 0.501해파리는 매우 잘 탐지되고 있으며, 높은 정밀도와 재현율, 그리고 mAP50-95가 매우 우수하여 모델 성능이 뛰어남.

Penguin (펭귄)
정밀도: 0.59재현율: 0.683mAP50: 0.597mAP50-95: 0.275펭귄 탐지는 상대적으로 약한 편

모델의 전반적인 성능은 해파리와 불가사리, 물고기 클래스에서는 상당히 좋은 성능을 보이고 있으며, 특히 해파리의 경우 매우 높은 정확도를 기록
펭귄과 퍼핀 클래스에서는 성능이 부족한 것으로 나타났으며, 일부 클래스에서는 mAP50-95가 낮아 성능이 일관되지 않다는 특징

2. Deep Learning(YOLO 8)

모델을 개선하기 위한 접근법

- **데이터 부족 클래스**

개선펭귄과 퍼핀 클래스의 성능이 낮은 이유는 데이터가 부족하거나 모델이 학습하기 어려운 특성을 가질 가능성, 데이터셋을 확장하거나 데이터 증강 기법을 활용

- **정밀도와 재현율 균형 맞추기**

일부 클래스에서 정밀도가 높지만 재현율이 낮은 경우, 더 많은 객체를 탐지할 수 있도록 모델을 개선할 수 있습니다.

- **IoU 기준 최적화**

mAP50-95가 낮은 클래스에 대해서는 IoU 기준을 최적화하여 보다 정확한 경계 상자 예측을 할 수 있도록 하여 모델의 탐지 성능을 향상

- **모델 튜닝 및 하이퍼파라미터 조정**

학습률이나 배치 크기 조정을 통해 모델 성능을 개선

2. Deep Learning(YOLO 8)

```
model.train(data='/content/data.yaml', epochs=50, batch=64, imgsz=640, lr0=0.001, augment=True, patience=10)
```

- epochs=50: 에폭을 50으로 설정하여 더 긴 학습을 진행.
- batch=64: 배치 크기를 64로 늘려서 처리 속도를 향상.
- imgsz=640: 이미지를 640x640 크기로 변경하여 모델이 더 많은 정보를 학습하도록
- lr0=0.001: 학습률을 0.001로 설정하여 모델이 안정적으로 학습.
- augment=True: 데이터 증강을 활성화하여 모델이 다양한 변형된 데이터를 학습.
- patience=10: 성능 개선이 없으면 10 에폭 후에 학습을 종료.

2. Deep Learning(YOLO 8)

model.train(data='/content/data.yaml', epochs=50, batch=64, imgsz=640, lr0=0.001, augment=True, patience=10)

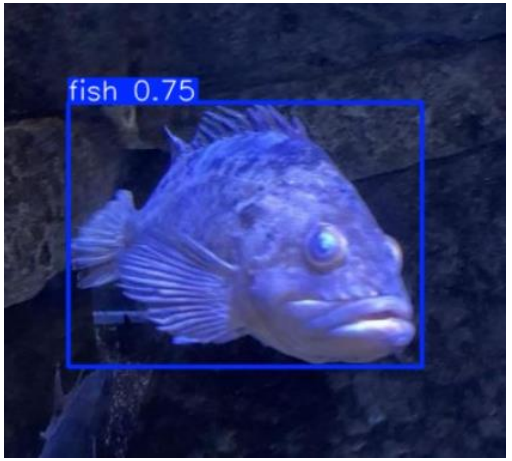
```
Validating runs/detect/train5/weights/best.pt...
Ultralytics 8.3.107 🚀 Python-3.11.12 torch-2.6.0+cu124 CUDA:0 (Tesla T4, 15095MiB)
Model summary (fused): 72 layers, 11,128,293 parameters, 0 gradients, 28.5 GFLOPs
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████|
                   all        127        909      0.827      0.712      0.785      0.504
                  fish         63        459      0.842      0.741      0.809      0.486
             jellyfish          9        155      0.892      0.923      0.962      0.584
               penguin         17        104      0.717      0.658      0.746      0.352
                puffin         15         74      0.711        0.5      0.597       0.32
                 shark         28         57      0.789      0.667      0.722      0.482
              starfish         17         27      0.989      0.741      0.809      0.645
              stingray         23         33      0.849      0.758      0.851      0.658
Speed: 0.2ms preprocess, 12.5ms inference, 0.0ms loss, 1.0ms postprocess per image
Results saved to runs/detect/train5
ultralytics.utils.metrics.DetMetrics object with attributes:
```

verbose=False, 불필요한 자료의 출력을 방지 !!

2. Deep Learning(YOLO 8)

```
[8]   1 results = model.predict(source = '/content/fish3.JPG', save = True)

image 1/1 /content/fish3.JPG: 416x384 22 jellyfishs, 58.0ms
Speed: 2.1ms preprocess, 58.0ms inference, 2.2ms postprocess per image at shape (1, 3, 416, 416)
Results saved to runs/detect/predict
```

YOLO8 : 테스트 데이터를 업로드하여 객체 탐지를 시도(22개의 객체를 탐지)



https://colab.research.google.com/drive/1ebVqbaYHVejZsA_bcZoNJCZkUOssQLPs?usp=sharing

2. Deep Learning(YOLO 8)

```
1 results = model.predict(source = '/content/seaFish.mp4' save = True)
```

```
video 1/1 (465/467) /content/seaFish.mp4: 256x416 1 fish, 1 jellyfish, 5.6ms
video 1/1 (466/467) /content/seaFish.mp4: 256x416 1 fish, 1 jellyfish, 6.3ms
video 1/1 (467/467) /content/seaFish.mp4: 256x416 1 fish, 5.8ms
Speed: 1.3ms preprocess, 7.8ms inference, 1.6ms postprocess per image at shape (1, 3, 416, 416)
Results saved to runs/detect/predict
```

YOLO8 : 동영상에서 객체 탐지를 시도

2. Deep Learning(YOLO 8)

YOLO5 vs YOLO8

구 분	YOLO 5	YOLO 8	비 고
모델 지정	yolo5.git의 cloning git clone https://github.com/ultralytics/yolov5.git %cd yolov5 Requirements !pip install -r requirements.txt	Install ultralytics model = YOLO('yolov8n.pt')	
학습	!python train.py … !python train.py --img 416 --batch 16 --epochs 50 --data /content/data.yaml --cfg ./models/yolov5s.yaml --weights yolov5s.pt --name fruits_yolov5s_results	model.train(data = ' ', epochs, batch, imgsz)	
예측	!python detect.py … !python detect.py --weights /content/best.pt --img 416 --conf 0.5 --source /grape.jpg	model.predict(source=' ', save = True)	
결과 저장	별도의 yolo5 폴더 하위에 학습모델 : Runs/train/weights/… 예측결과 : Runs/predict/…	학습모델 : Runs/train/weights/… 예측결과 : Runs/predict/…	

2. Deep Learning(YOLO 8)

실습) 가중치 파일 선택 vs 이미지 사이즈 의 인식 정확도 비교

가중치 모델	224 pxl	416 pxl	640 pxl	mp4	youtube
YOLOv8n					
YOLOv8s					
YOLOv8m					
YOLOv8l					
YOLOv8xl					

2. Deep Learning(YOLO 8) summary

- 외부 데이터의 입력(사진/동영상)의 인식률이 떨어졌다.

- YOLOv8n.pt Yolov8s.pt의 정밀도와 정확도가 차이가
  났으나 인식결과는 Yolov8s가 조금 더 개선되었다

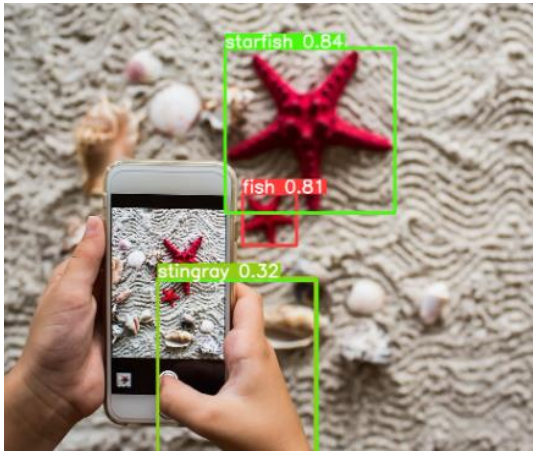
- epoches가 낮으면 정밀도 정확도의 수치가 낮았다.
  => 학습된 이미지의 화질/사이즈와 입력된 이미지의
  동일성이 유지될 수록 예측결과가 개선되면 과적합이 일부 발생했다.

2. Deep Learning(YOLO 8)

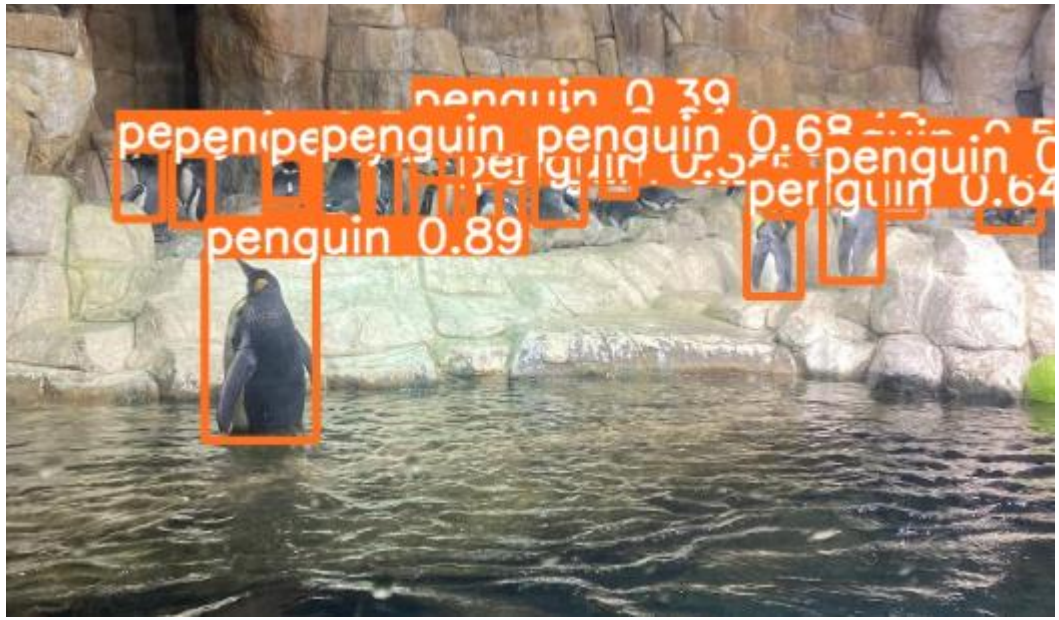
실습) 1.각자 선택한 CUSTOM DATA로 학습 후 예측을 시도해 봅시다.

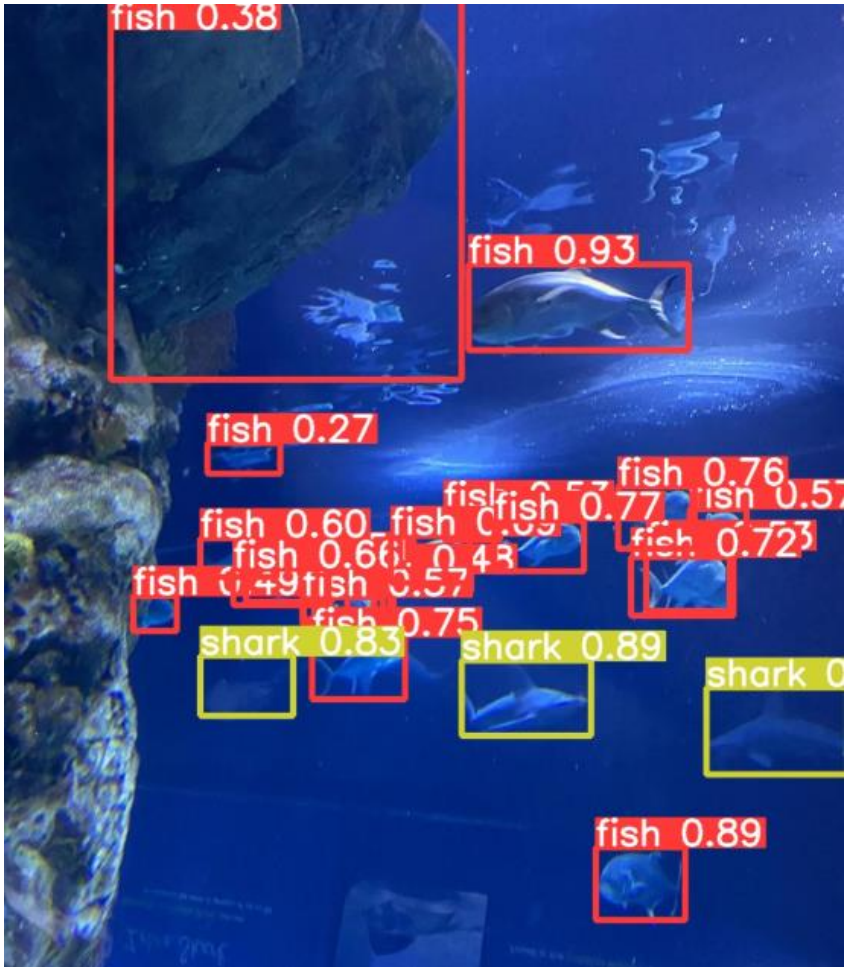
2. yolov8m.pt 등 다른 모델을 시도해 봅시다

3. 시도한 내용을 정리하여 비교분석 후 공유해 봅시다.

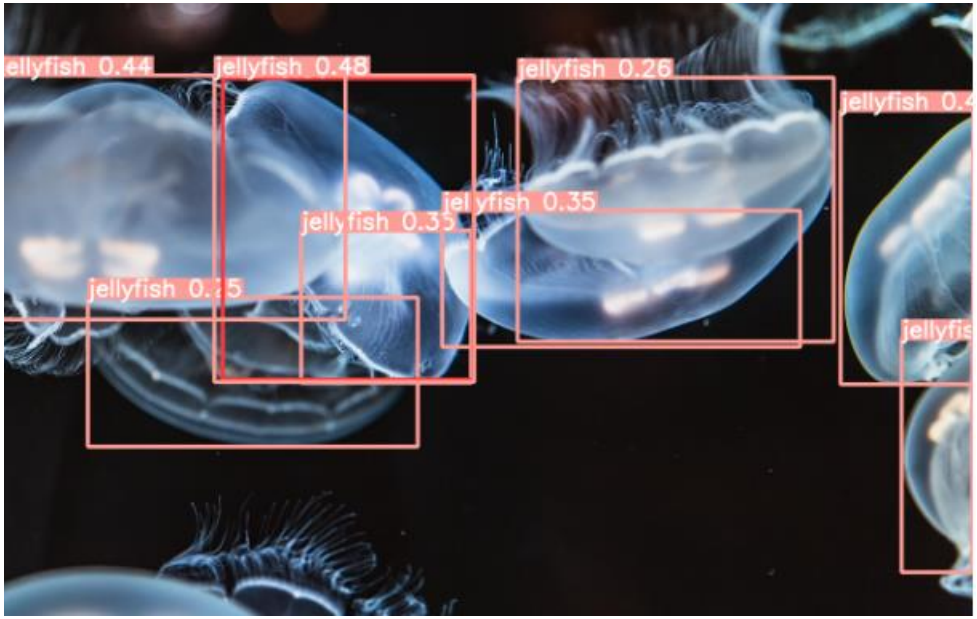


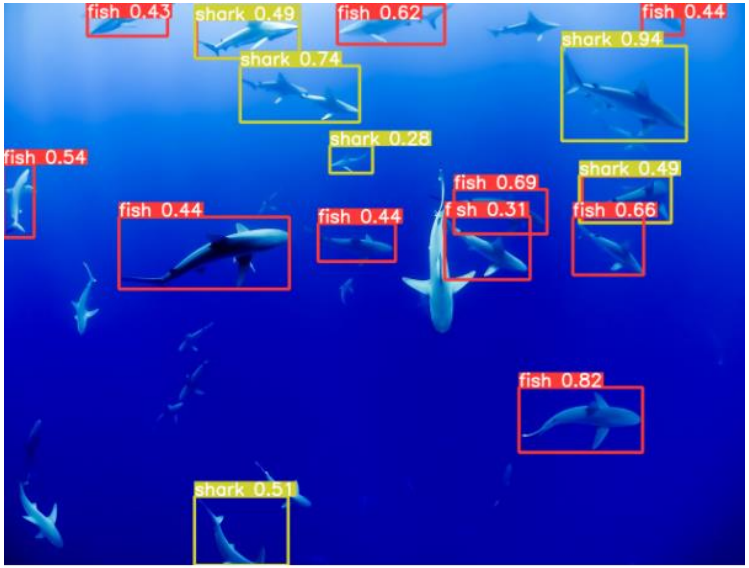
2. Deep Learning(YOLO 8)



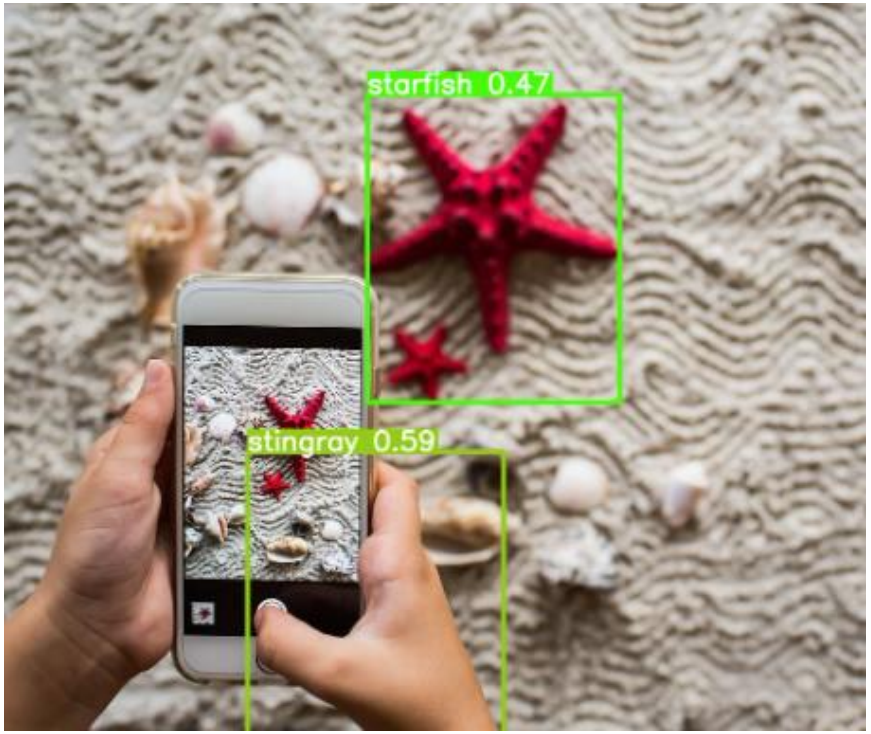


2. Deep Learning(YOLO 8)





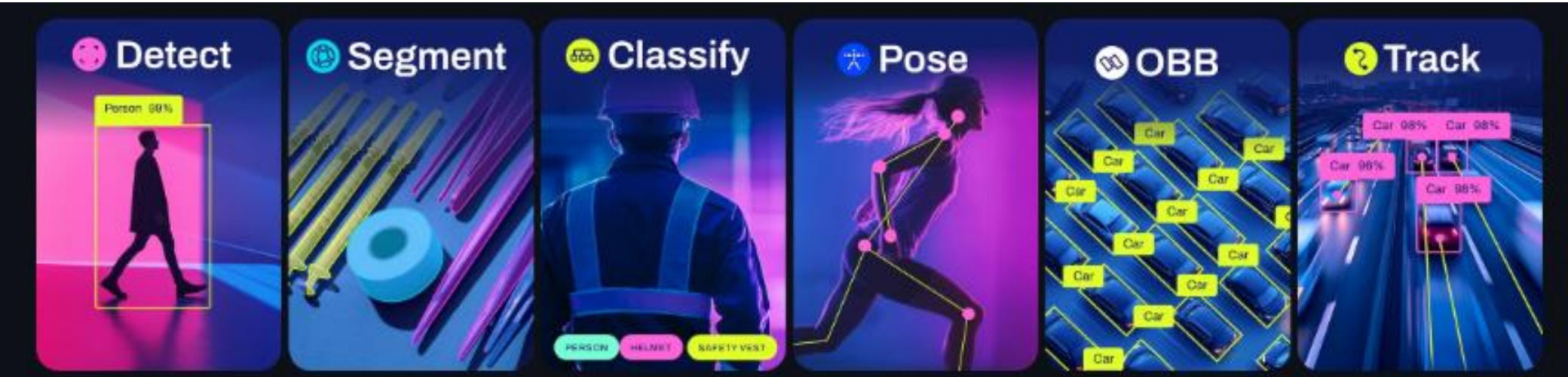
2. Deep Learning(YOLO 8)



2. Deep Learning(YOLO 11)

YOLO 11

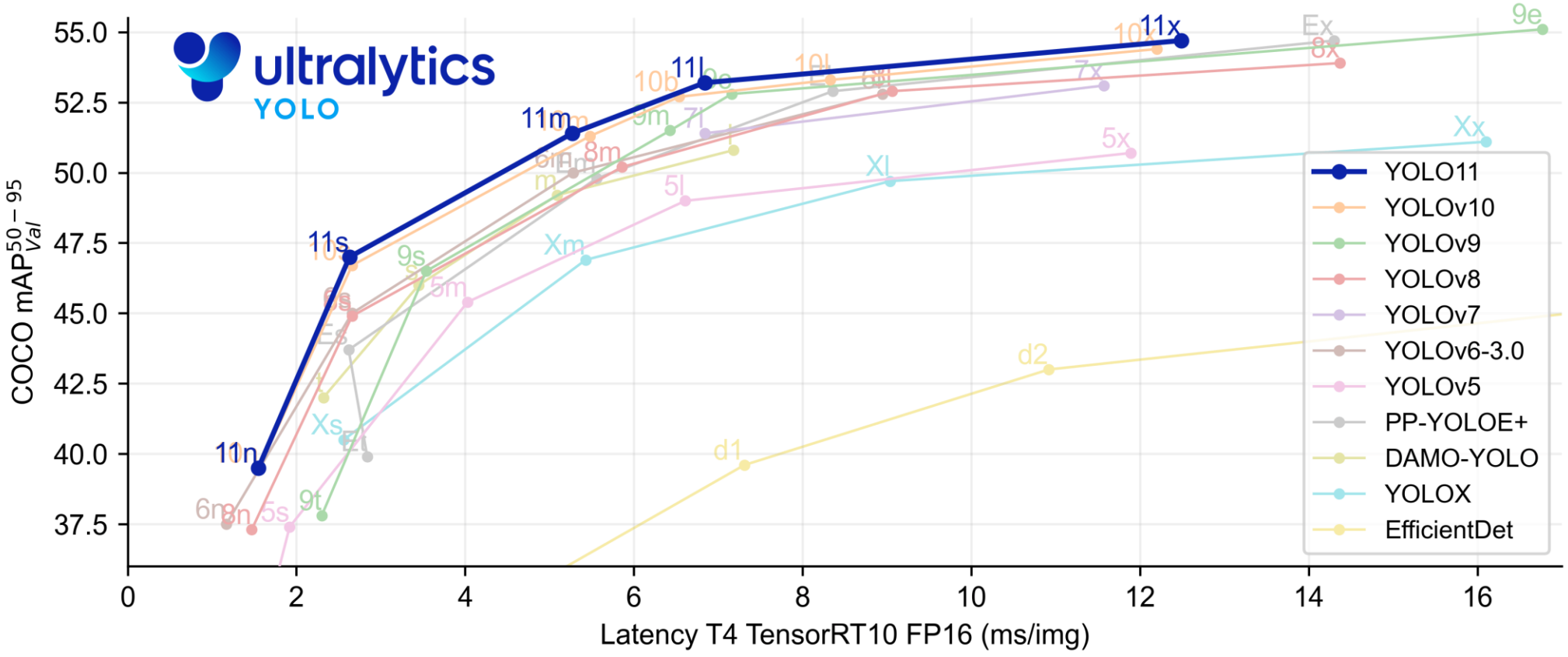
2. Deep Learning(YOLO 11)



▼ Detection (COCO)

Explore the Detection Docs for usage examples. These models are trained on the COCO dataset, featuring 80 object classes.

2. Deep Learning(YOLO 11)



2. Deep Learning(YOLO 11)

모델	파일 이름	작업	추론	유효성 검사	교육	내보내기

YOLO11	`yolo11n.pt`
	`yolo11s.pt`
	`yolo11m.pt`
	`yolo11l.pt`
	`yolo11x.pt`
		탐지	✅	✅	✅	✅

YOLO11-seg	`yolo11n-seg.pt`
	`yolo11s-seg.pt`
	`yolo11m-seg.pt`
	`yolo11l-seg.pt`
	`yolo11x-seg.pt`
		인스턴스 세분화	✅	✅	✅	✅

YOLO11-pose	`yolo11n-pose.pt`
	`yolo11s-pose.pt`
	`yolo11m-pose.pt`
	`yolo11l-pose.pt`
	`yolo11x-pose.pt`
		포즈/키포인트	✅	✅	✅	✅

YOLO11-obb	`yolo11n-obb.pt`
	`yolo11s-obb.pt`
	`yolo11m-obb.pt`
	`yolo11l-obb.pt`
	`yolo11x-obb.pt`
		방향 탐지	✅	✅	✅	✅

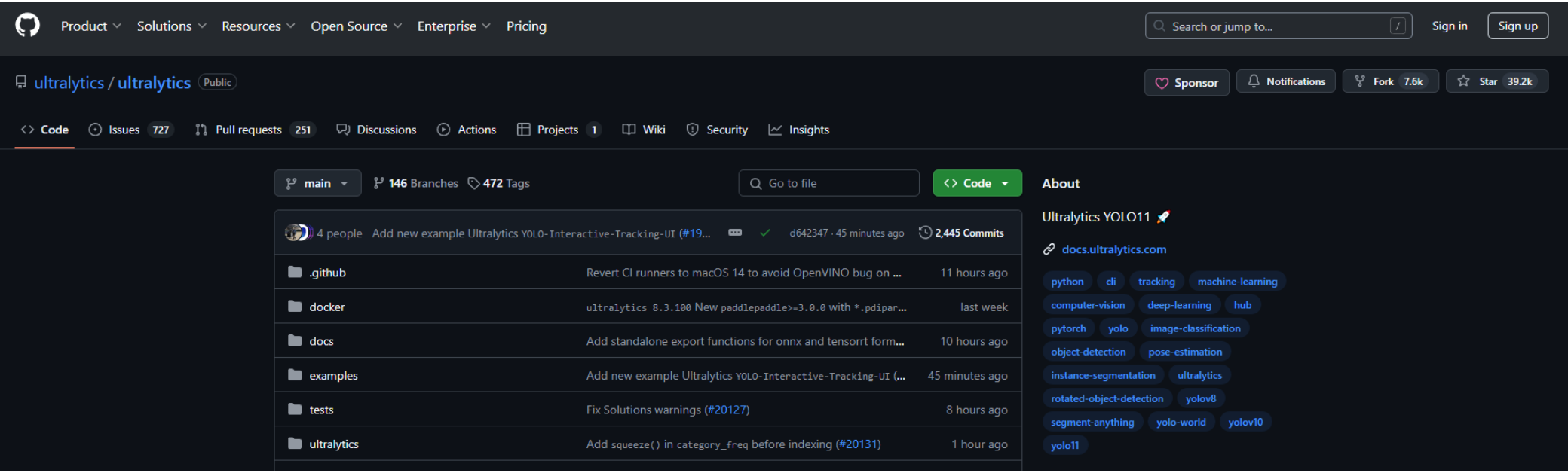
YOLO11-cls	`yolo11n-cls.pt`
	`yolo11s-cls.pt`
	`yolo11m-cls.pt`
	`yolo11l-cls.pt`
	`yolo11x-cls.pt`
		분류	✅	✅	✅	✅

2. Deep Learning(YOLO 11)

Model	size (pixels)	mAP^{val} 50-95	Speed CPU ONNX (ms)	Speed T4 TensorRT10 (ms)	params (M)	FLOPs (B)
YOLO11n	640	39.5	56.1 ± 0.8	1.5 ± 0.0	2.6	6.5
YOLO11s	640	47.0	90.0 ± 1.2	2.5 ± 0.0	9.4	21.5
YOLO11m	640	51.5	183.2 ± 2.0	4.7 ± 0.1	20.1	68.0
YOLO11l	640	53.4	238.6 ± 1.4	6.2 ± 0.1	25.3	86.9
YOLO11x	640	54.7	462.8 ± 6.7	11.3 ± 0.2	56.9	194.9

https://github.com/ultralytics/ultralytics

2. Deep Learning(YOLO 11)



https://public.roboflow.com/object-detection/shellfish-openimages

2. Deep Learning(YOLO 11)

roboflow Universe Public Datasets Model Zoo Blog Docs Deploy a Model

Explore these datasets, models, and more on Roboflow Universe. →

350+ MILLION IMAGES 500,000+ DATASETS 100,000+ PRE-TRAINED MODELS

DATASET TYPE

All Datasets 61
Object Detection 37
Classification 11

Shellfish-OpenImages Dataset
Object Detection (Bounding Box)
581 images | 3 exports | Last updated 3 years ago

Oxford Pets Dataset
Object Detection (Bounding Box)
3680 images | 3 exports | Last updated 3 years ago

Images
581
0 missing annotations
0 null examples

Annotations
1,215
2.1 per image (average)
across 3 classes

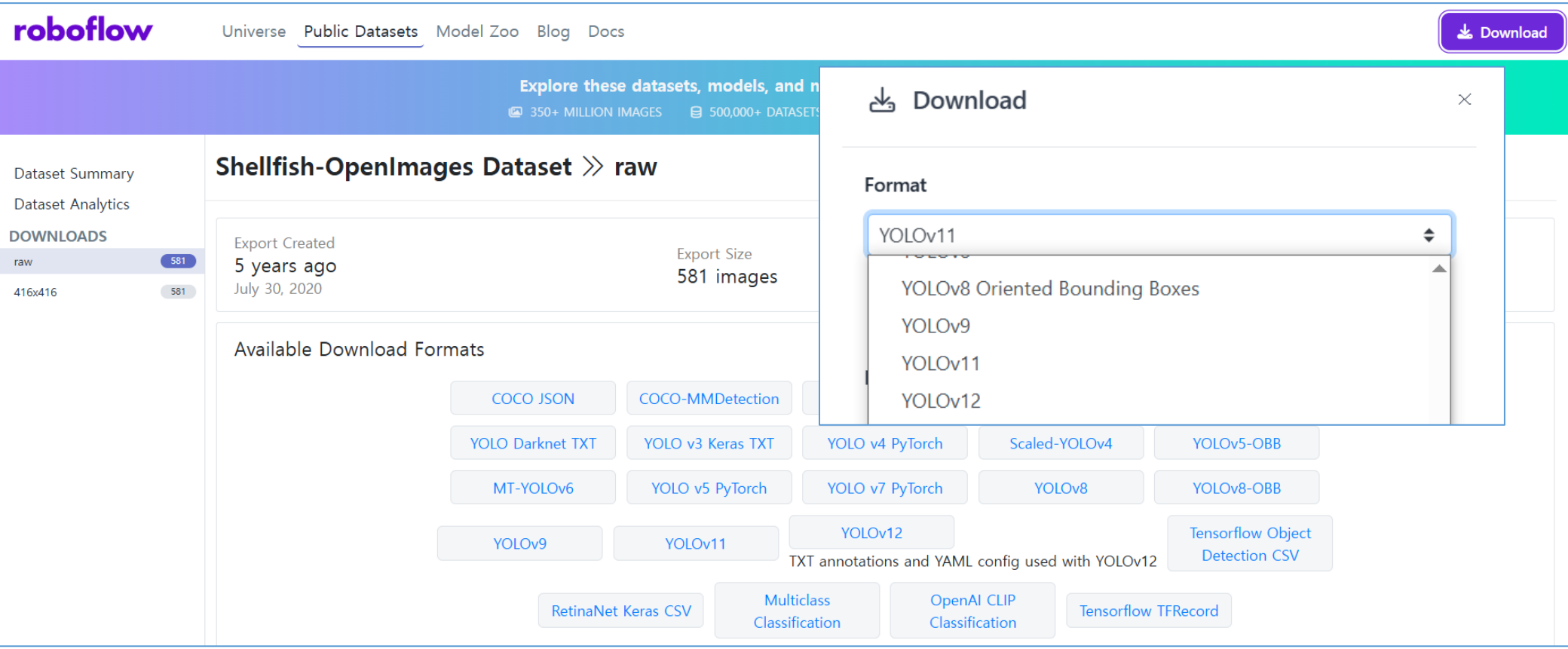
Average Image Size
0.78 mp
from 0.52 mp
to 5.04 mp

Median Image Ratio
1024x768
wide

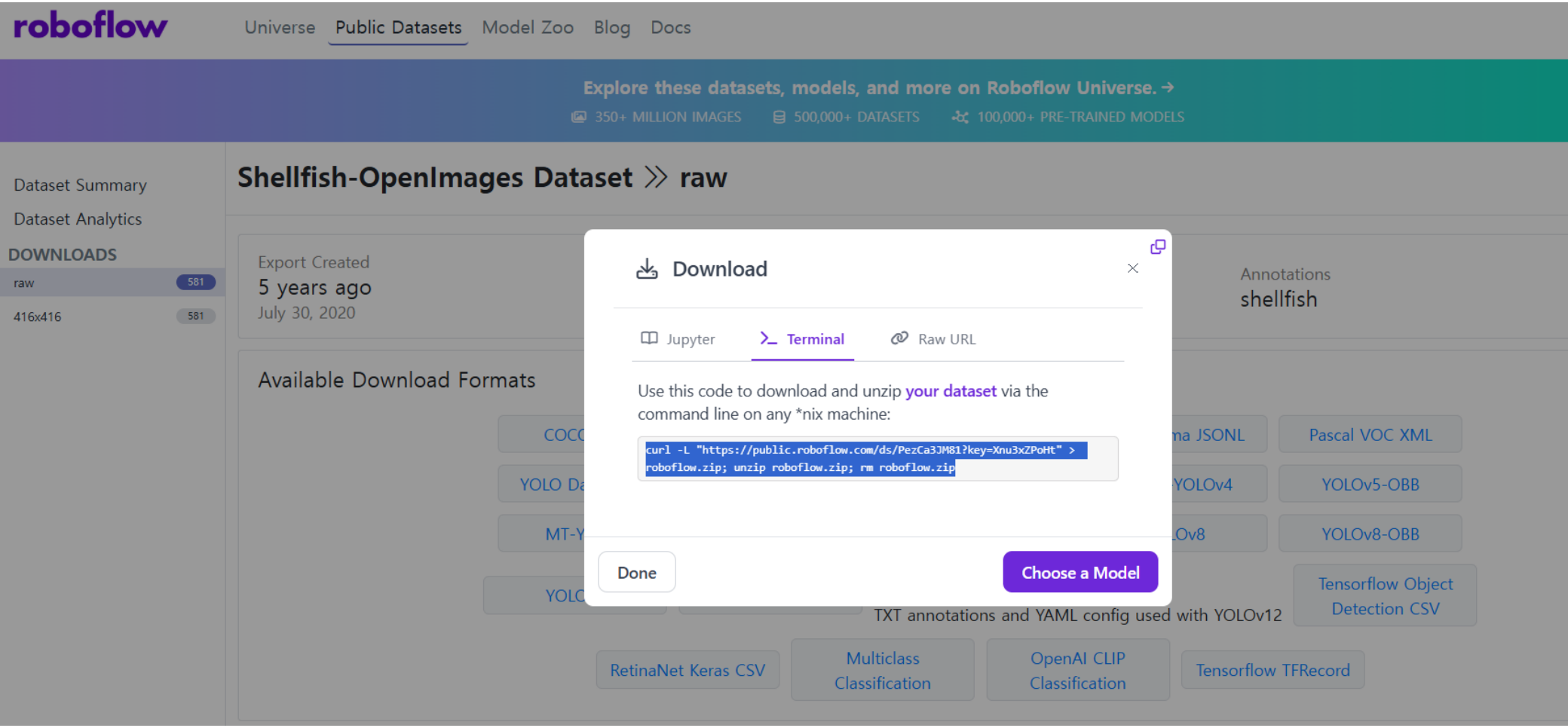
Class Balance

Shrimp	710	
Lobster	269	under represented
Crab	236	under represented

2. Deep Learning(YOLO 11)

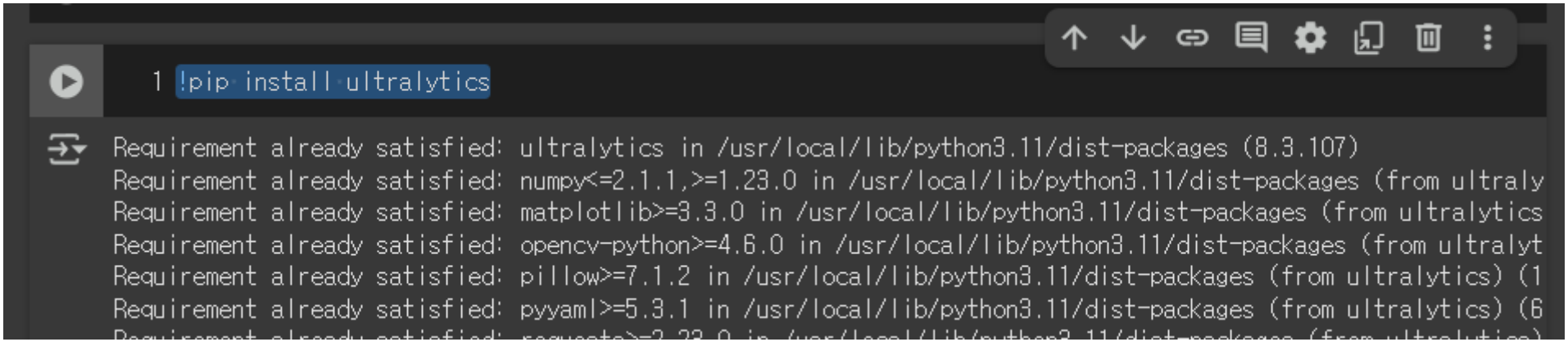


2. Deep Learning(YOLO 11)



2. Deep Learning(YOLO 11)

```
!pip install ultralytics
```



2. Deep Learning(YOLO 11)

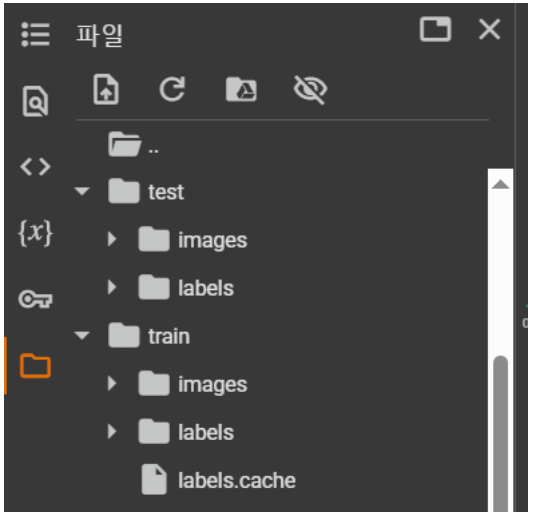
```
from ultralytics import YOLO
model = YOLO('yolo11n.pt')

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

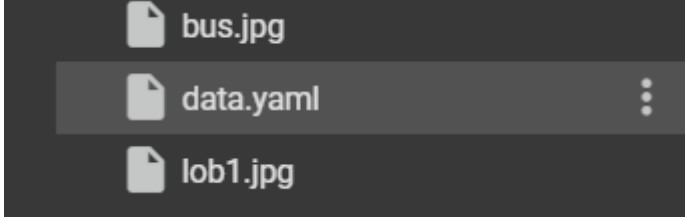
```
[28]  1 from ultralytics import YOLO
      2 model = YOLO('yolo11n.pt')
      3
      4 print(type(model.names), len(model.names))
      5 print(model.names)

      <class 'dict'> 80
      {0: 'person', 1: 'bicycle', 2: 'car', 3: 'motorcycle', 4: 'airplane', 5: 'bus', 6: 'train', 7: 'truck', 8:
```

2. Deep Learning(YOLO 11)

!curl -L "https://public.roboflow.com/ds/PezCa3JM81?key=Xnu3xZPoHt" > roboflow.zip; unzip roboflow.zip; rm roboflow.zip

```
1 !curl -L "https://public.roboflow.com/ds/PezCa3JM81?key=Xnu3xZPoHt" > roboflow.zip; unzip roboflow.zip; rm roboflow.zip

extracting: train/images/ccd6ed827632e845_jpg.rf.149de769969e70cb713e9b1881d604ba.jpg
extracting: train/images/cec1f95192614824_jpg.rf.49973cb5e84906575dfb5a3ebc8e4239.jpg
extracting: train/images/cf4899ff91bc7e52_jpg.rf.2215356223a31fd3cf96d8ec5c7e6b03.jpg
extracting: train/images/cfd8ba188d53a049_jpg.rf.7e2bcd53883b9b72efdecc46881895a0.jpg
extracting: train/images/d20b3661a1af49cd_jpg.rf.608081debf24590d376fe1afb2b057d5.jpg
extracting: train/images/d23846015c65244e_jpg.rf.1e1d53f25ef69e8244eb1d306db6f487.jpg
```





2. Deep Learning(YOLO 11)

!cat data.yaml

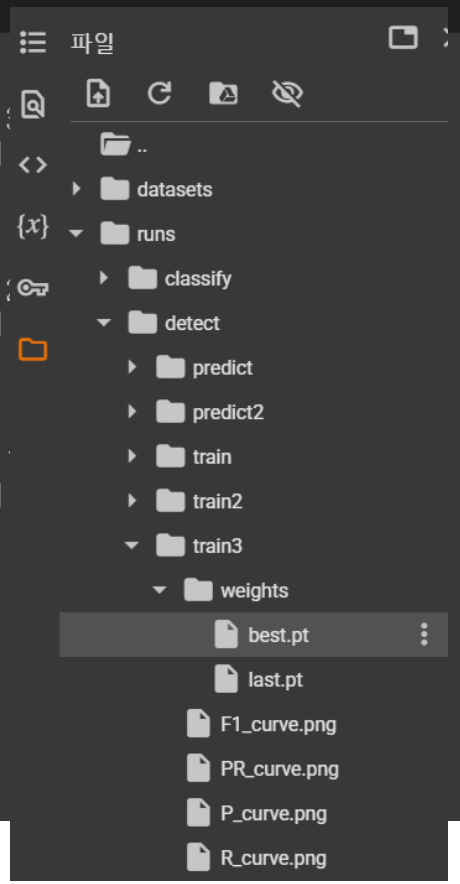
```
1 !cat data.yaml
```

```
train: ../train/images
val: ../valid/images
test: ../test/images

nc: 3
names: ['Crab', 'Lobster', 'Shrimp']

roboflow:
  workspace: jacob-solawetz
  project: shellfish-openimages
  version: 3
  license: CC BY 4.0
  url: https://universe.roboflow.com/jacob-solawetz/shellfish-openimages/dataset/3
```

2. Deep Learning(YOLO 11)

model.train(data='/content/data.yaml', epochs=3, batch=32, imgsz=416)

```
1 model.train(data='/content/data.yaml', epochs=3, batch=32, imgsz=416)

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
        1/3         0G      1.251      3.319      1.413        114        416: 100%|| 13/13 [02:
                  Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
        2/3         0G      1.276      3.014      1.403         88        416: 100%|| 13/13 [02:
                  Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
        3/3         0G      1.229      2.567      1.381         82        416: 100%|| 13/13 [02:
                  Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|

3 epochs completed in 0.134 hours.
Optimizer stripped from runs/detect/train3/weights/last.pt, 5.4MB
Optimizer stripped from runs/detect/train3/weights/best.pt, 5.4MB

Validating runs/detect/train3/weights/best.pt...
Ultralytics 8.3.107 🚀 Python-3.11.12 torch-2.6.0+cu124 CPU (Intel Xeon 2.20GHz)
YOLO11n summary (fused): 100 layers, 2,582,737 parameters, 0 gradients, 6.3 GFLOPs
```



2. Deep Learning(YOLO 11)

```
results = model.predict(source='/content/lob1.jpg')
```

```
image 1/1 /content/lob1.jpg: 288x416 (no detections), 105.8ms
Speed: 3.8ms preprocess, 105.8ms inference, 1.0ms postprocess per image at shape (1, 3, 288, 416)
```

2. Deep Learning(YOLO 11)

실습) YOLO 11을 사용해 보고 버젼별/모델별 성능평가를 공유해 봅시다

1. YOLOv3: The third iteration of the YOLO model family, originally by Joseph Redmon, known for its efficient real-time object detection capabilities.
2. YOLOv4: A darknet-native update to YOLOv3, released by Alexey Bochkovskiy in 2020.
3. YOLOv5: An improved version of the YOLO architecture by Ultralytics, offering better performance and speed trade-offs compared to previous versions.
4. YOLOv6: Released by Meituan in 2022, and in use in many of the company's autonomous delivery robots.
5. YOLOv7: Updated YOLO models released in 2022 by the authors of YOLOv4.
6. YOLOv8: A versatile model featuring enhanced capabilities such as instance segmentation, pose/keypoints estimation, and classification.
7. YOLOv9: An experimental model trained on the Ultralytics YOLOv5 codebase implementing Programmable Gradient Information (PGI).
8. YOLOv10: By Tsinghua University, featuring NMS-free training and efficiency-accuracy driven architecture, delivering state-of-the-art performance and latency.
9. YOLO11 🚀 **NEW**: Ultralytics' latest YOLO models delivering state-of-the-art (SOTA) performance across multiple tasks including detection, segmentation, pose estimation, tracking, and classification.

2. Deep Learning(YOLO 11)

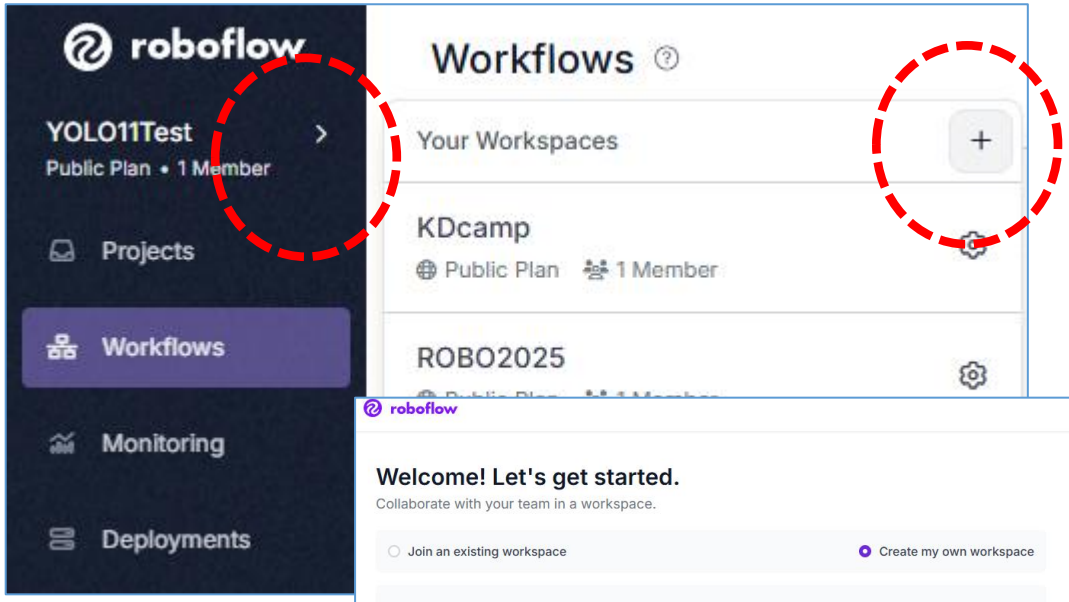
Custom Data Set 직접 만들기

2. Deep Learning(YOLO 11)

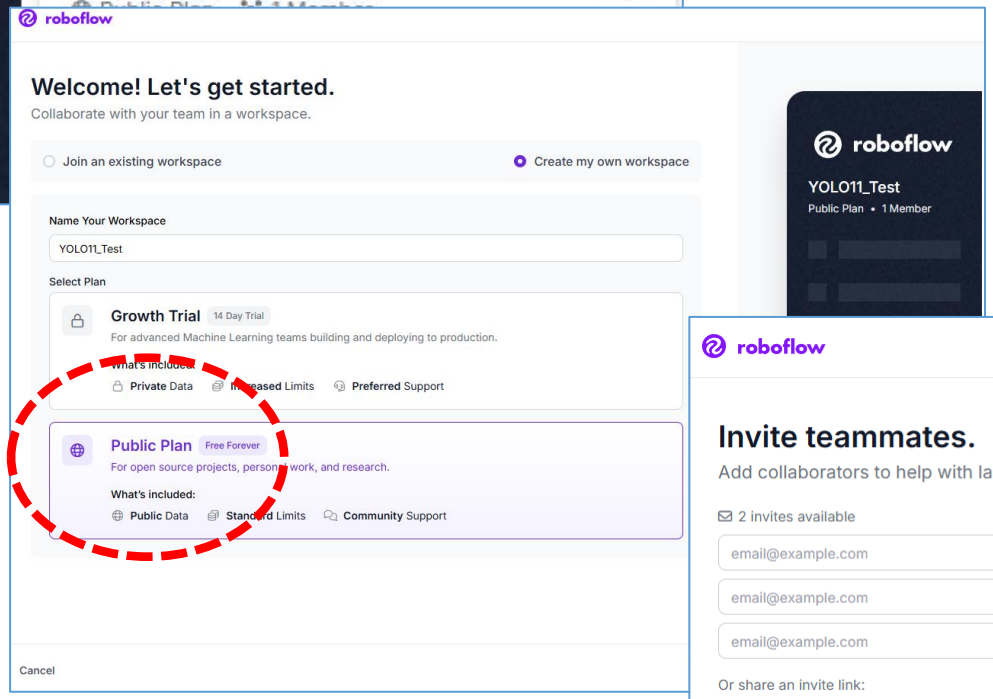
Roboflow with Custom data set

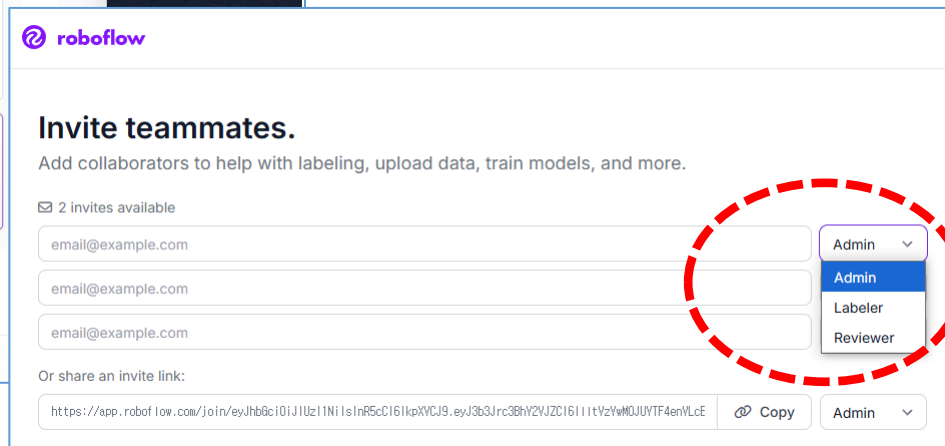
https://app.roboflow.com/new-workspace-zzixh

2. Deep Learning(YOLO 11)

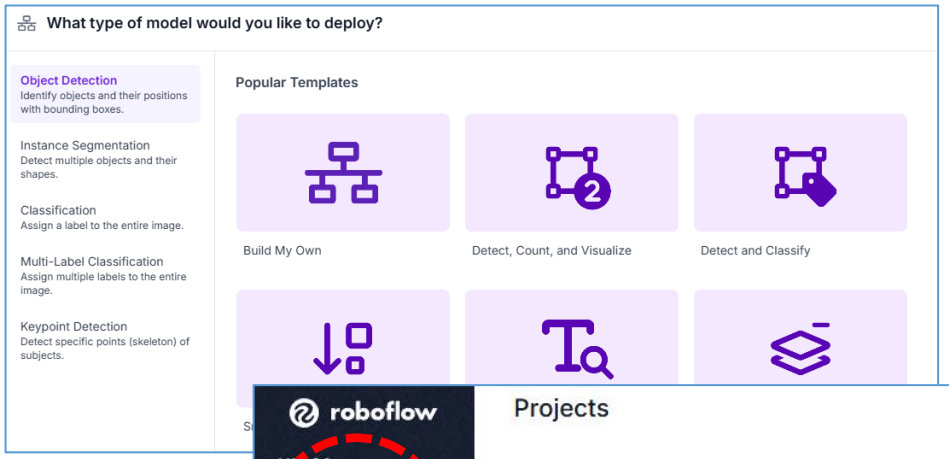


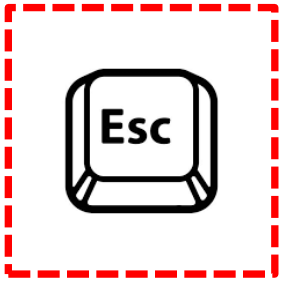
워크스페이스를 먼저 만든다 !!

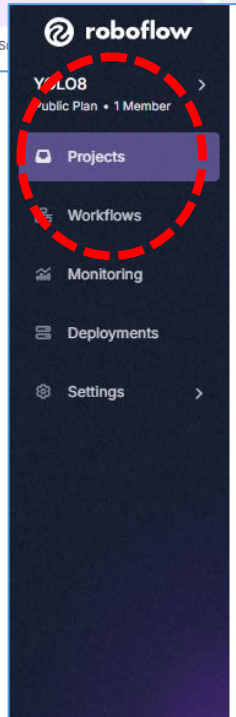




2. Deep Learning(YOLO 11)







Projects

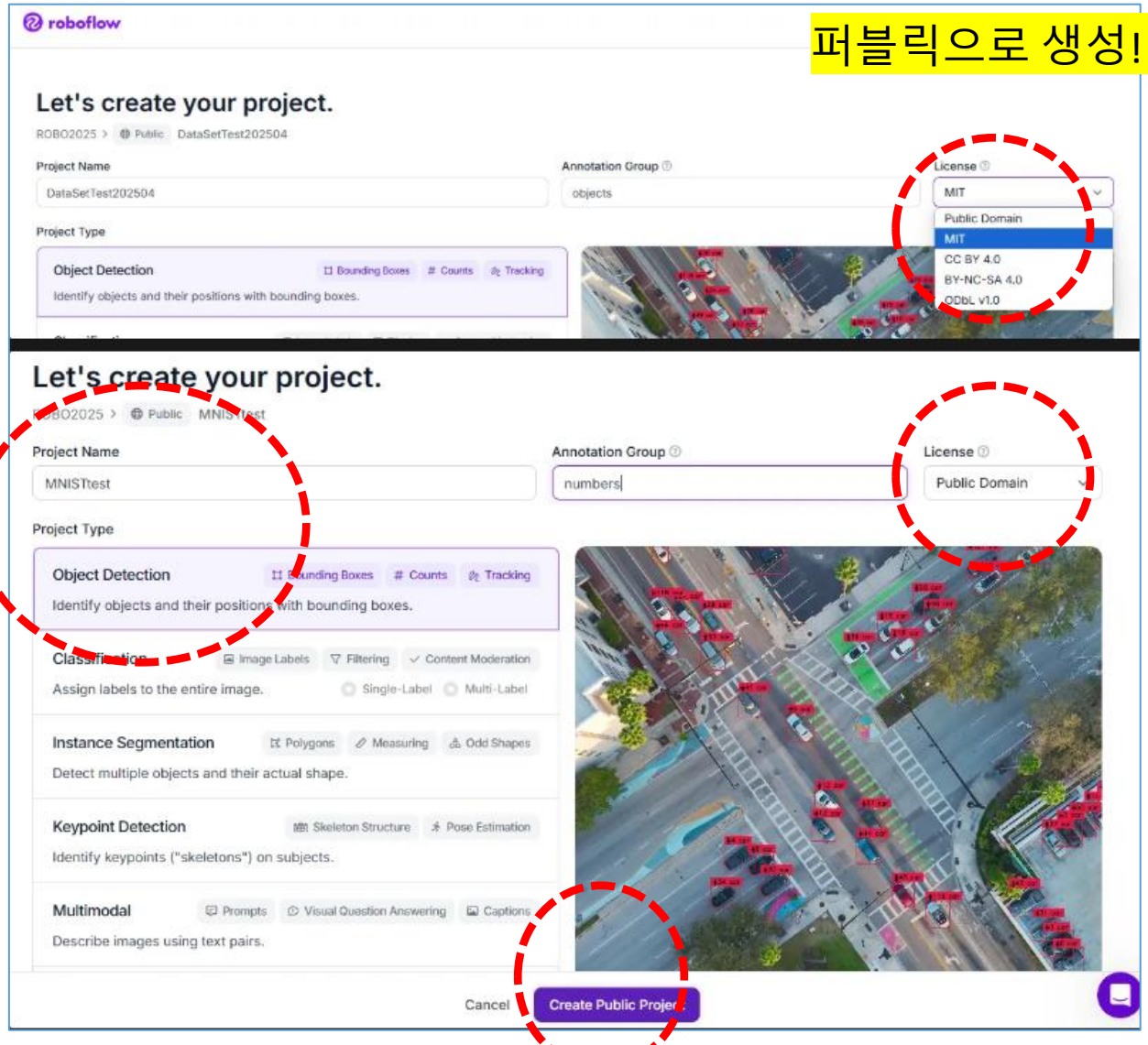
프로젝트를 만든다 !!

There are no projects in this workspace.

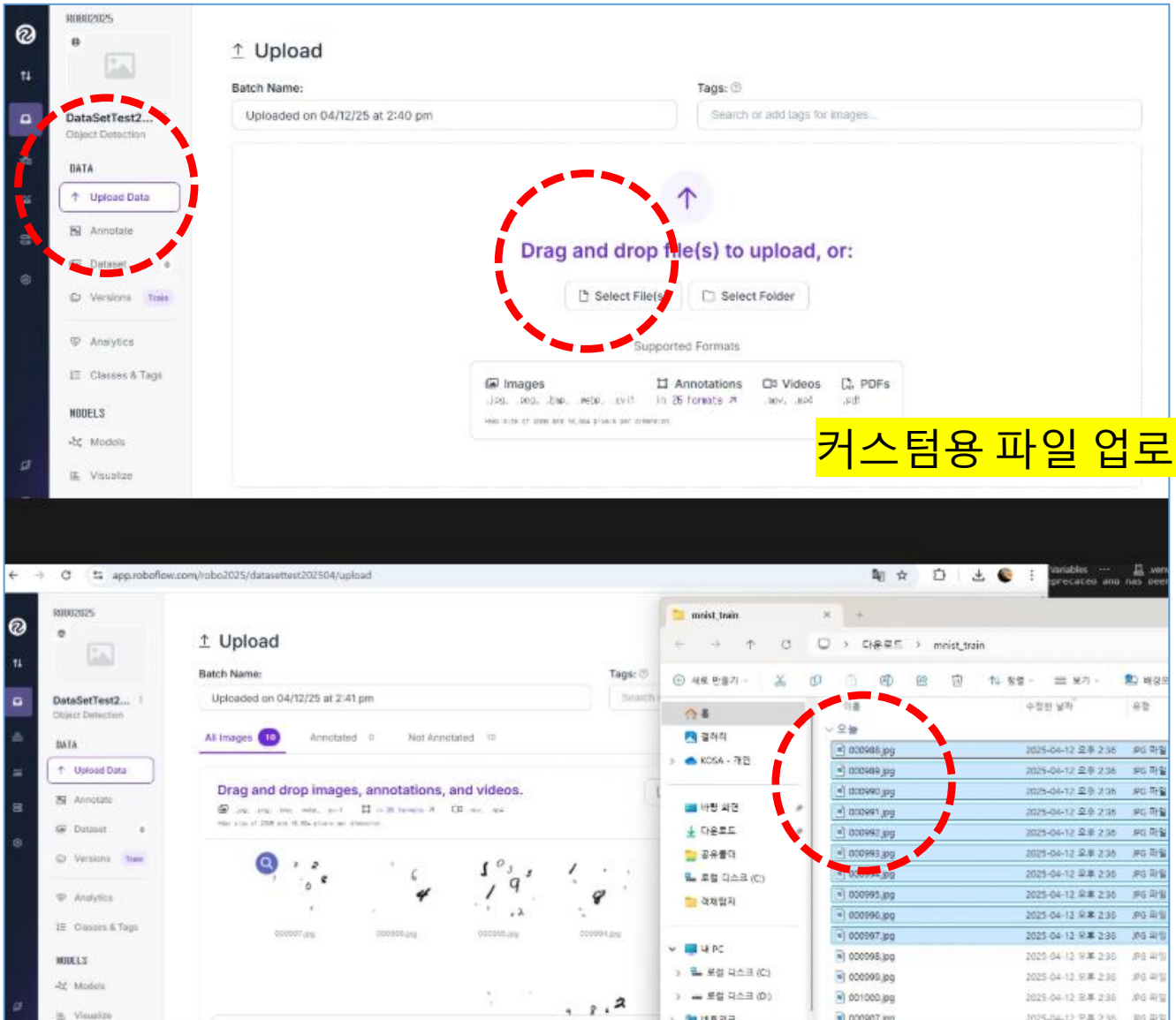
Create a project and upload images to start labeling, training, and deploying your computer vision model.

+ New Project View a Tutorial

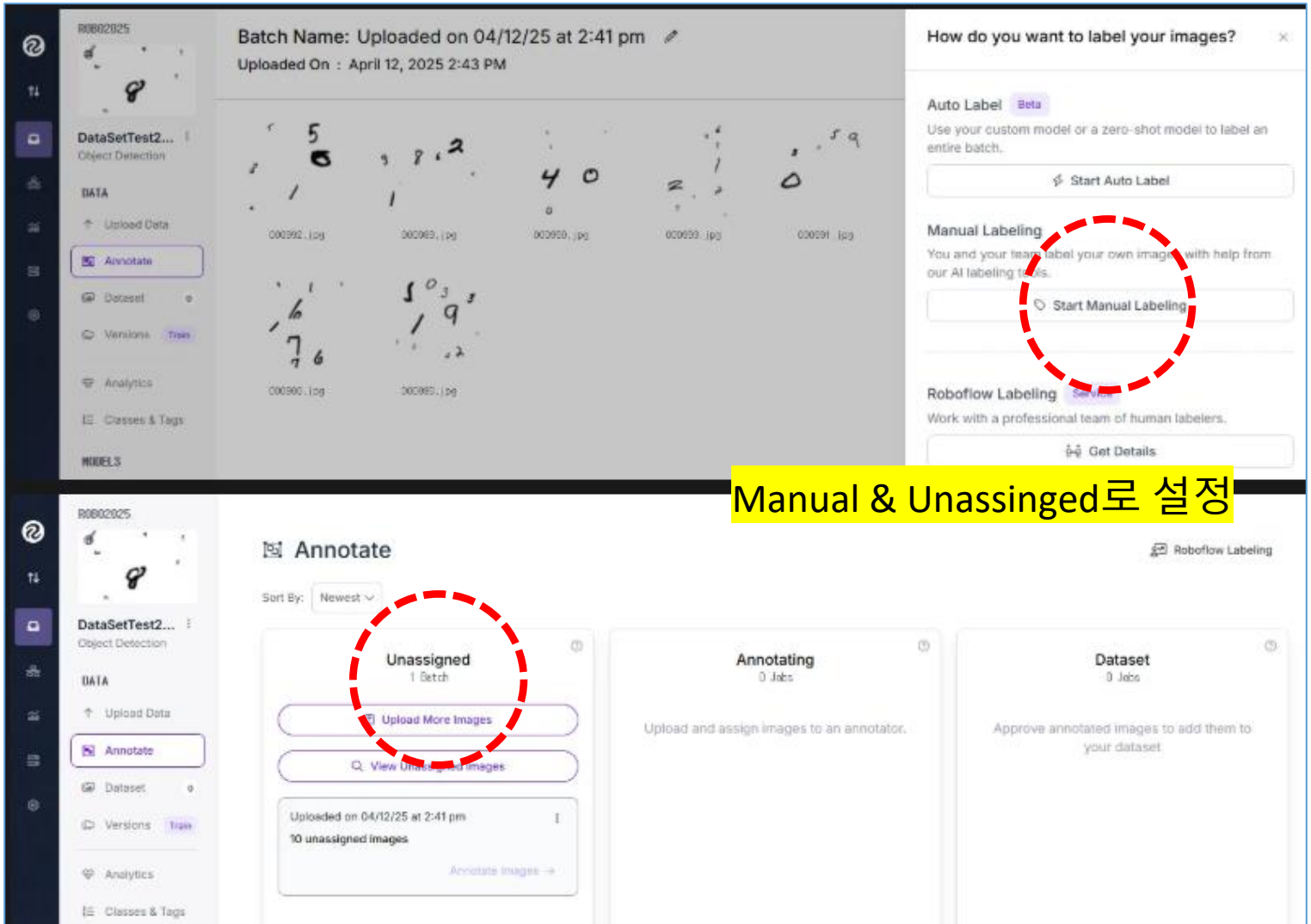
2. Deep Learning(YOLO 11)



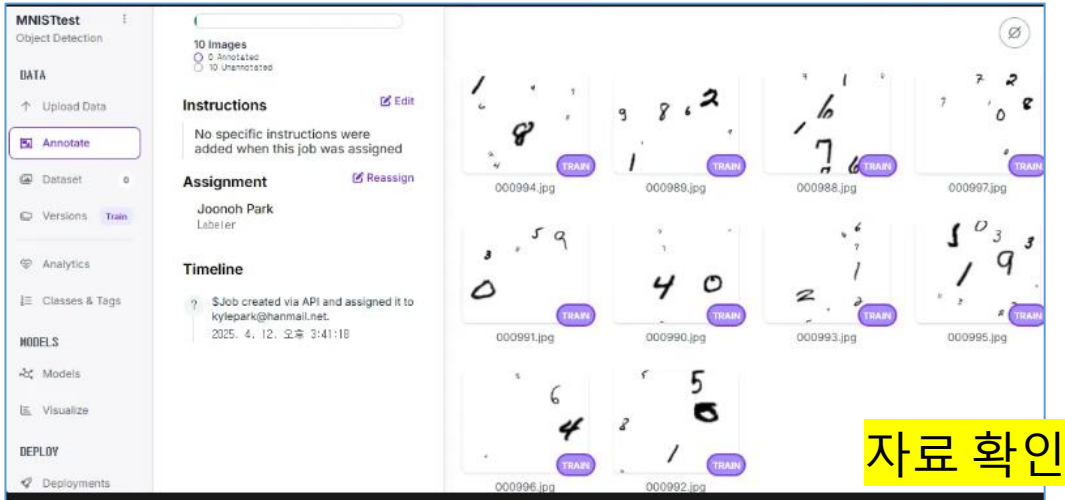
2. Deep Learning(YOLO 11)



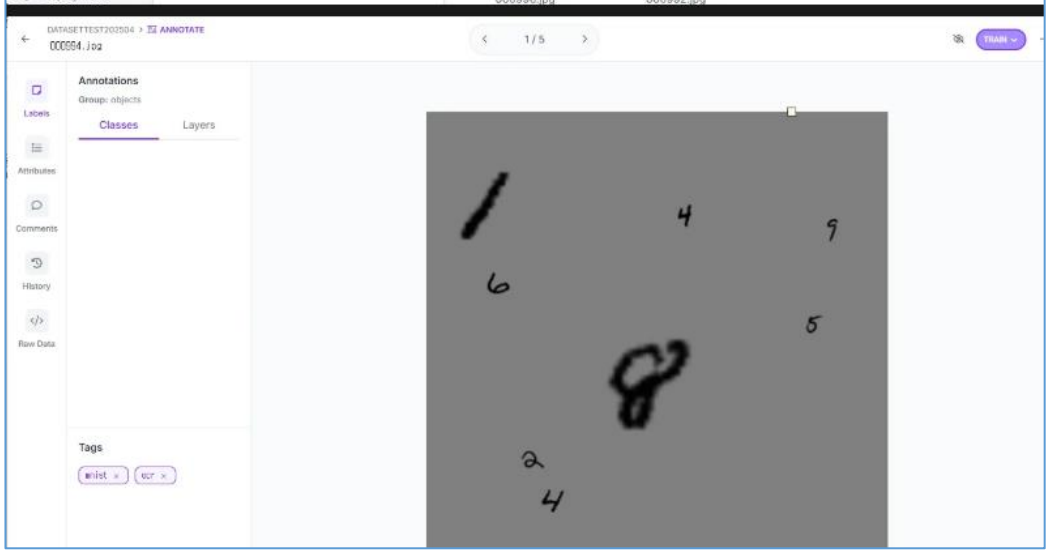
2. Deep Learning(YOLO 11)



2. Deep Learning(YOLO 11)

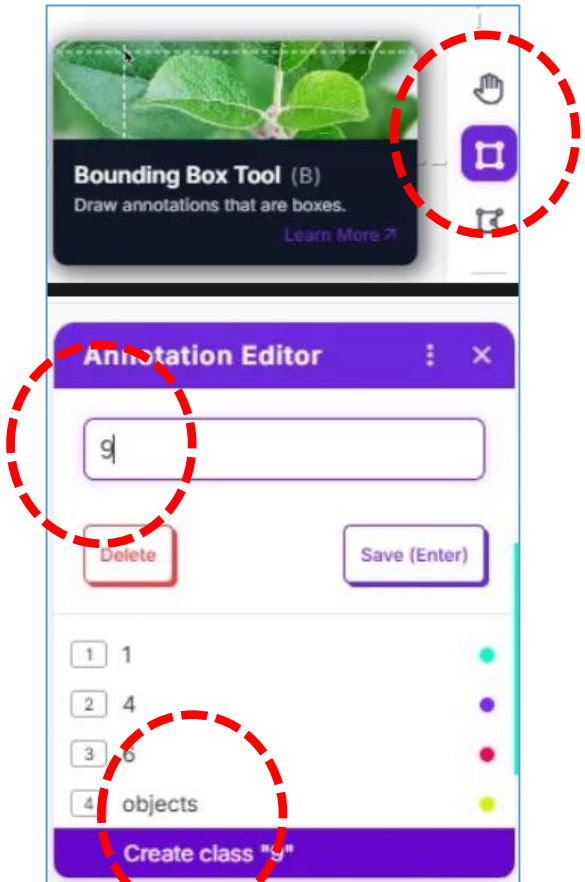


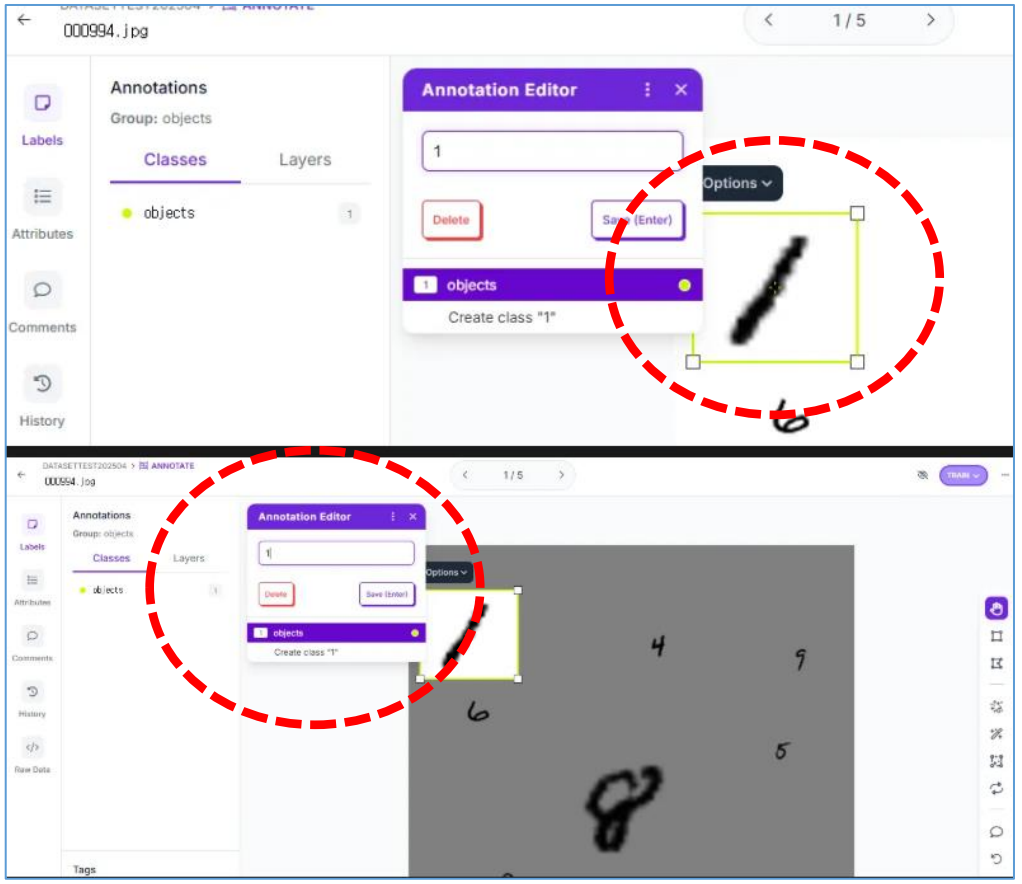
자료 확인!!



2. Deep Learning(YOLO 11)

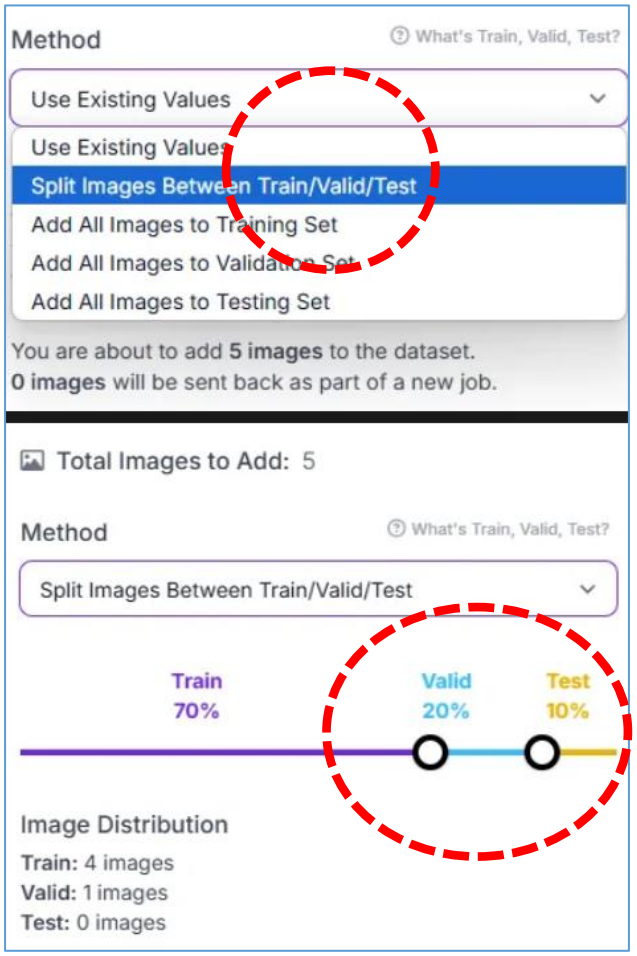
사각형 선택 후 클래스(라벨)을
선택하고 클래스별로 입력!!





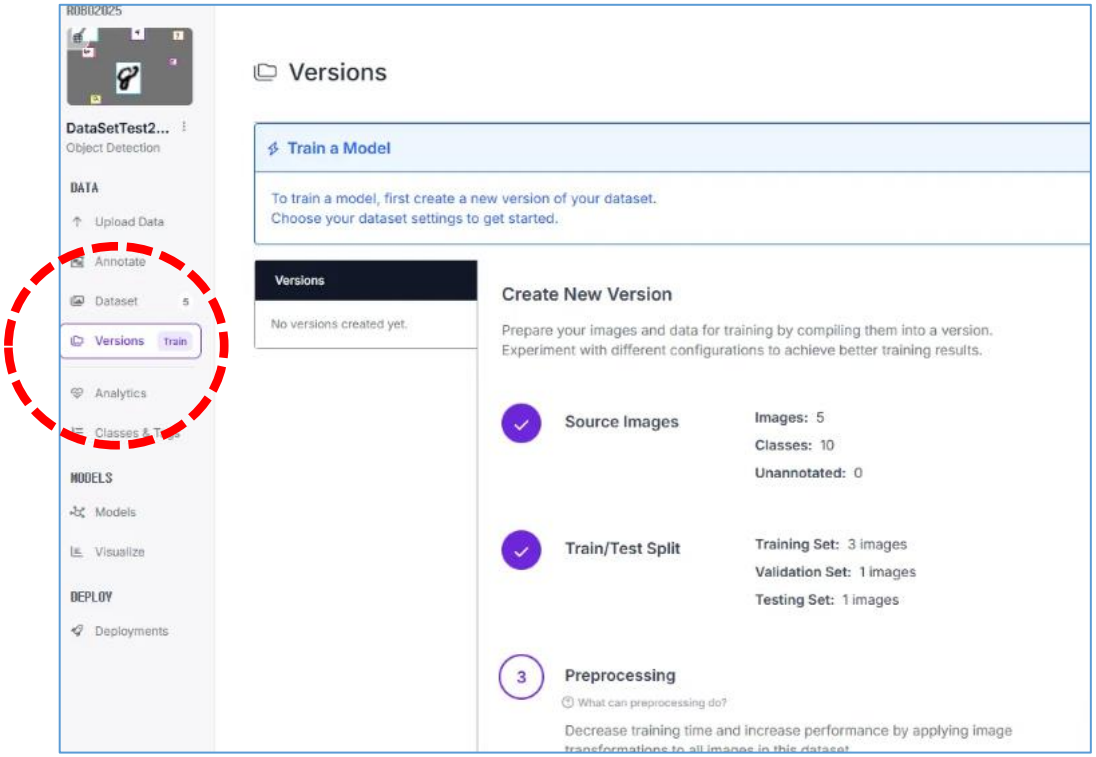
2. Deep Learning(YOLO 11)

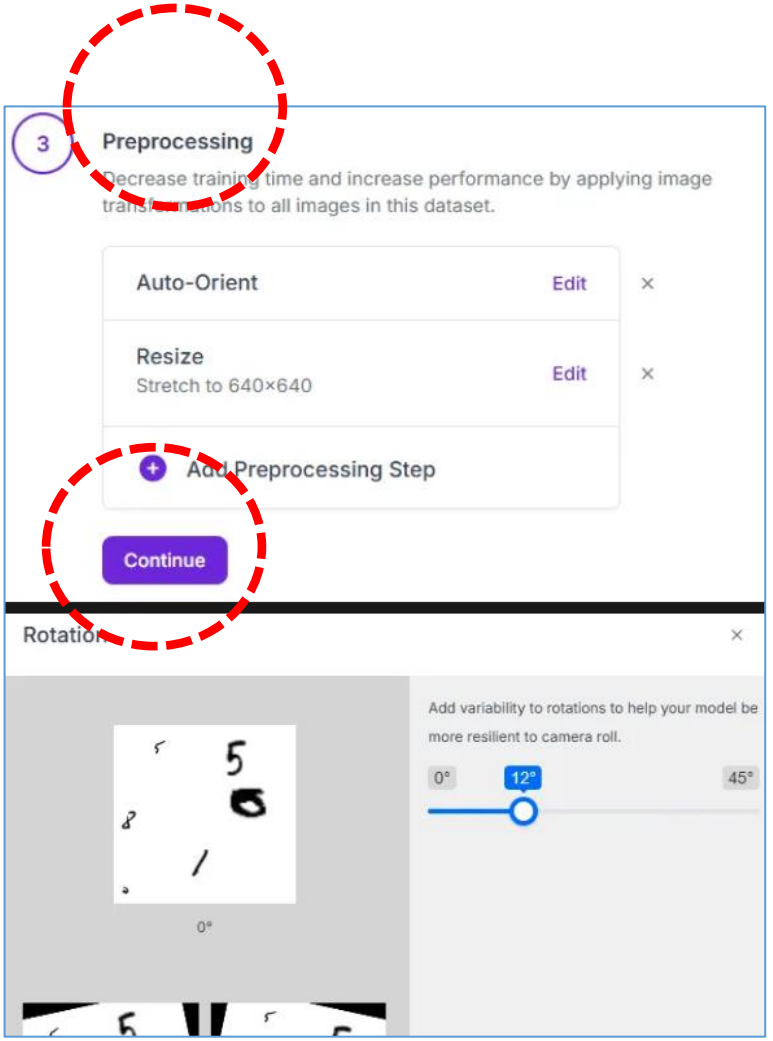
클래스별 오류 확인 후 추가 또는 train/test 분리!!



2. Deep Learning(YOLO 11)

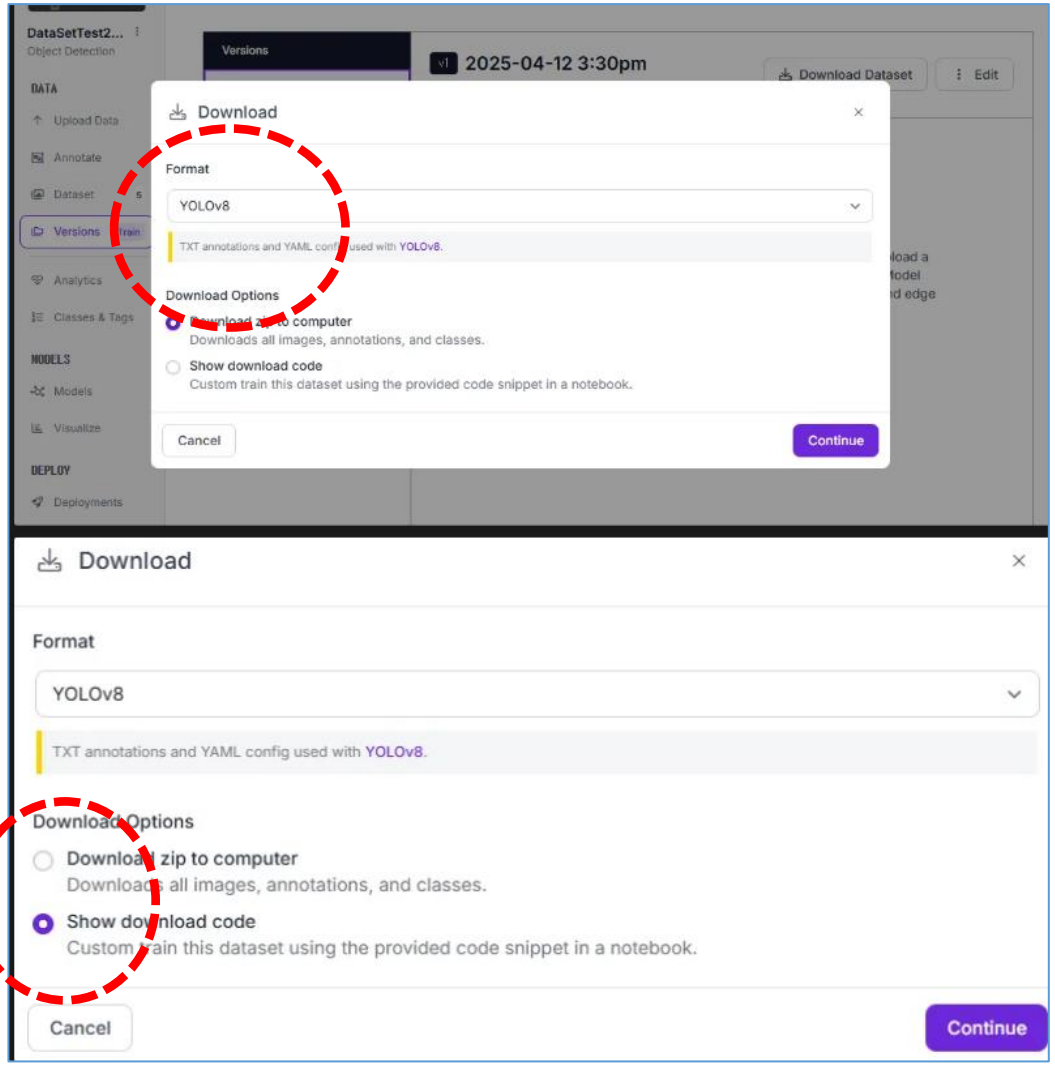
데이트 증강 선택(2개 무료)





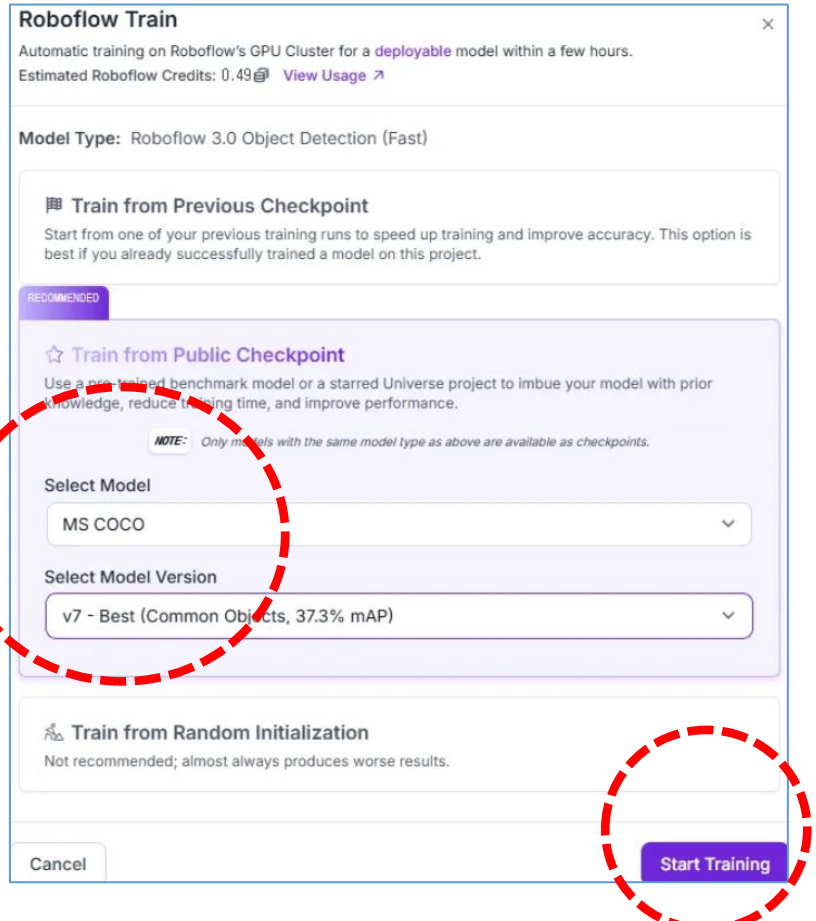
2. Deep Learning(YOLO 11)

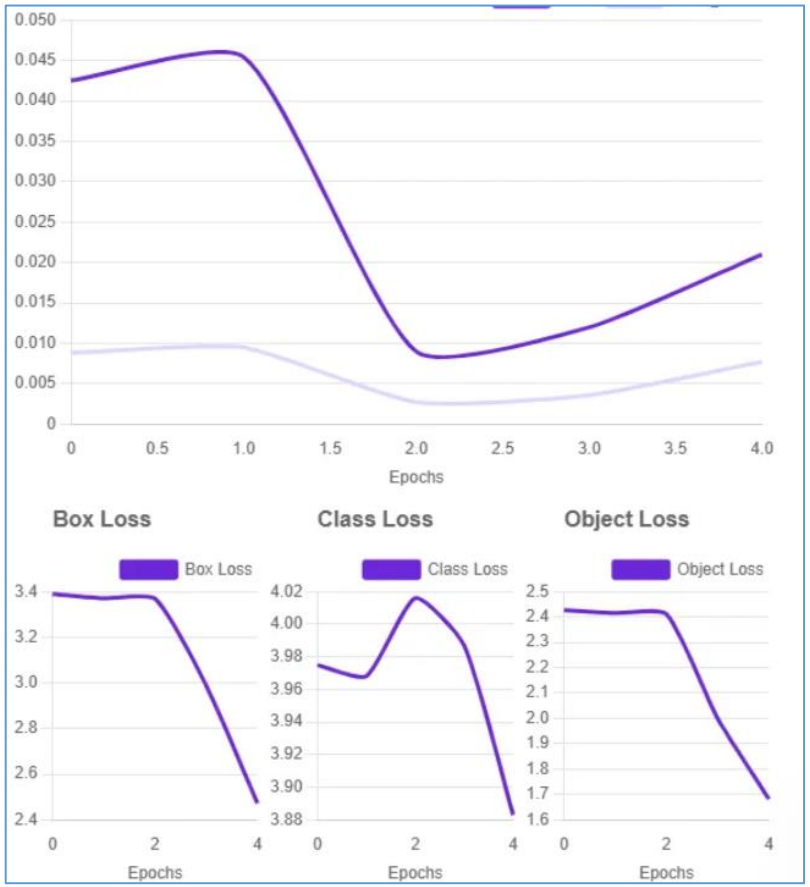
코드 출력 후 다운로드 !!



2. Deep Learning(YOLO 11)

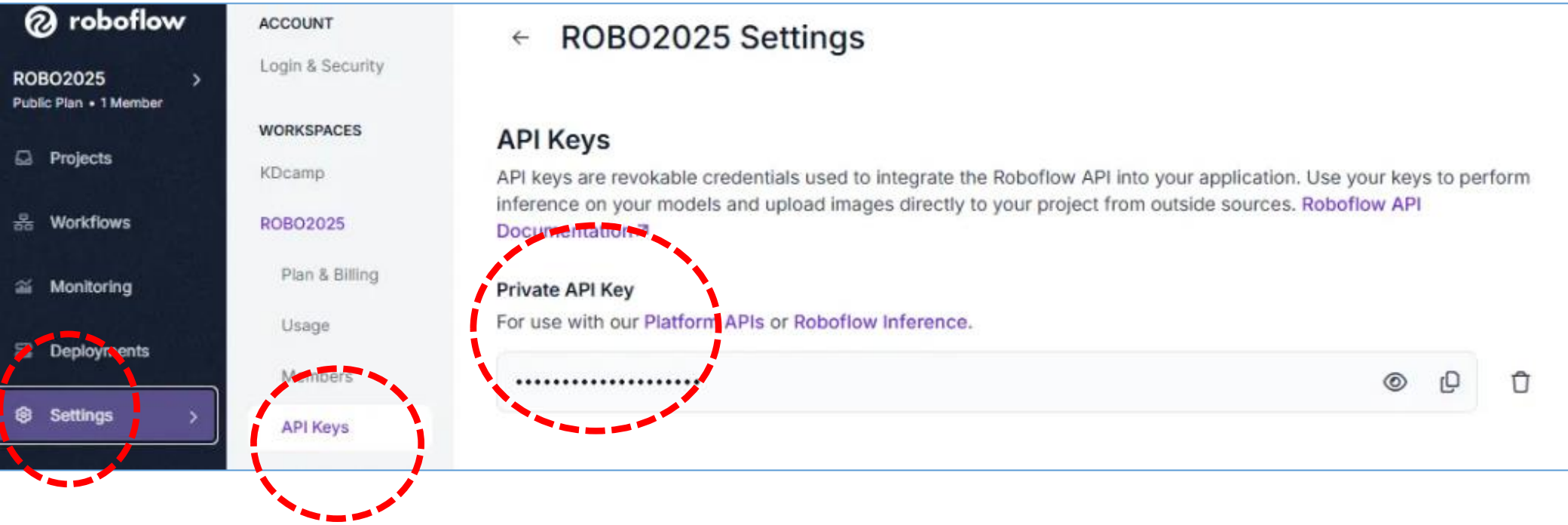
내장된 모델로 테스트 훈련 가능 !!



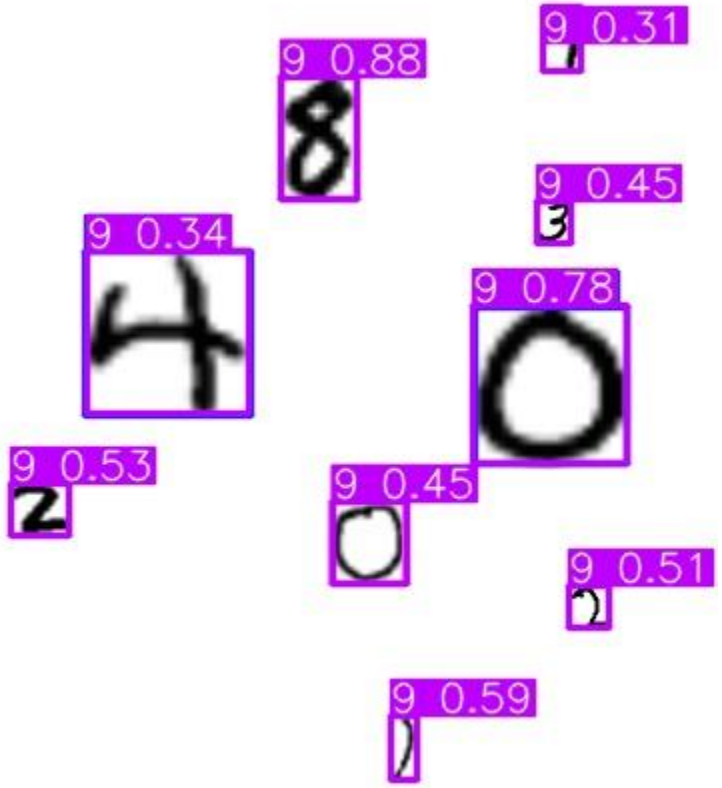


2. Deep Learning(YOLO 11)

워크스페이스/프로젝트 단위로 API key 발급



2. Deep Learning(YOLO 11)



2. Deep Learning(YOLO 11)

YOLO data 분석 리포트 작성

2. Deep Learning(YOLO 11)

YOLO 병충해 감지 모델 학습 결과 분석 보고서

1. 서론

본 보고서는 YOLO (You Only Look Once) 객체 탐지 모델을 사용하여 식물 병충해를 감지하는 사용자 정의 데이터셋에 대한 학습 결과를 분석합니다. 학습의 목표는 이미지 내에서 병해(disease)와 해충(pest)을 정확하게 식별하고 위치를 파악하는 것입니다.

2. 데이터셋

본 학습에는 사용자 정의 병충해 이미지 데이터셋이 사용되었습니다. 데이터셋은 다음과 같은 두 가지 클래스로 구성됩니다.

- **pest**: 식물에 해를 끼치는 해충
- **disease**: 식물에 나타나는 질병의 징후

데이터셋의 크기 및 구성에 대한 자세한 내용은 생략합니다.

2. Deep Learning(YOLO 11)

3. 학습 설정

- **모델:** YOLOv5 (가정)
- **이미지 크기:** 640x640
- **Epoch 수:** 100
- **데이터 증강:** 랜덤 플립, 회전, 스케일링 등 적용
- **최적화 함수:** Adam
- **학습 환경:** [학습 환경에 대한 정보가 있다면 기재]

4. 검증 결과 분석

학습 완료 후 검증 데이터셋을 사용하여 모델의 성능을 평가했습니다. 주요 지표는 다음과 같습니다.

클래스	정밀도 (Precision)	재현율 (Recall)	mAP@0.50	mAP@0.50:0.95
pest	0.85	0.78	0.82	0.65
disease	0.75	0.82	0.79	0.58
All	**0.80**	**0.80**	**0.81**	**0.62**

2. Deep Learning(YOLO 11)

- **정밀도 (Precision):** 모델이 'pest' 또는 'disease'로 예측한 결과 중 실제로 해당 클래스인 비율입니다. 'pest' 클래스의 정밀도는 85%, 'disease' 클래스의 정밀도는 75%로 나타났습니다. 이는 모델이 해충을 비교적 정확하게 식별하지만, 질병의 경우 잘못된 긍정 예측이 더 있을 수 있음을 의미합니다.
- **재현율 (Recall):** 실제 'pest' 또는 'disease' 객체 중에서 모델이 얼마나 많이 탐지했는지를 나타냅니다. 'pest' 클래스의 재현율은 78%, 'disease' 클래스의 재현율은 82%입니다. 이는 모델이 질병 징후를 해충보다 더 잘 찾아내는 경향이 있음을 보여줍니다.
- **mAP@0.50 (Mean Average Precision at IoU=0.50):** IoU 임계값을 0.5로 설정했을 때의 평균 정밀도입니다. 전반적으로 81%의 mAP@0.50 성능을 보이며, 이는 모델이 객체의 위치를 비교적 정확하게 예측함을 나타냅니다.
- **mAP@0.50:0.95 (Mean Average Precision averaged over IoU thresholds from 0.50 to 0.95):** 다양한 IoU 임계값에서 모델의 성능을 종합적으로 평가하는 지표입니다. 62%의 mAP@0.50:0.95 성능은 모델이 객체의 위치를 매우 정확하게 예측하는 능력은 다소 부족할 수 있음을 시사합니다.

2. Deep Learning(YOLO 11)

5. 정성적 분석 (선택 사항)

[실제 이미지와 모델 예측 결과를 시각적으로 비교 분석한 내용을 추가할 수 있습니다. 예를 들어, 특정 상황에서 모델이 어려움을 겪는 경우 (작은 객체, 겹쳐 있는 객체, 유사한 외형 등)를 언급할 수 있습니다.]

6. 결과 논의

전반적으로 모델은 병충해 감지 작업에서 준수한 성능을 보입니다. 특히 정밀도와 재현율이 모두 80% 수준으로 균형 잡힌 성능을 나타내고 있습니다. 하지만 mAP@0.50:0.95 값이 상대적으로 낮은 것은 모델이 객체의 위치를 매우 정확하게 예측하는 데 어려움을 겪을 수 있음을 의미합니다. 이는 특히 작은 크기의 병충해나 복잡한 환경에서 발생할 수 있습니다.

클래스별 성능을 보면 'pest' 클래스의 정밀도가 'disease' 클래스보다 약간 높은 반면, 재현율은 'disease' 클래스가 더 높습니다. 이는 데이터셋의 특성이나 클래스 간의 시각적 유사성 등 다양한 요인에 의해 발생할 수 있습니다.

2. Deep Learning(YOLO 11)

7. 개선 방안

모델의 성능을 더욱 향상시키기 위해 다음과 같은 방법을 고려해 볼 수 있습니다.

- **데이터셋 확장 및 다양화:** 더 많은 양의 다양한 병충해 이미지 데이터를 확보하고, 다양한 환경 조건 (조명, 각도, 배경 등)을 포함하도록 데이터셋을 확장합니다.
- **데이터 증강 강화:** 더 강력한 데이터 증강 기법을 적용하여 모델의 일반화 능력을 향상시킵니다.
- **모델 구조 변경 또는 하이퍼파라미터 튜닝:** YOLOv5 외에 다른 YOLO 버전 (예: YOLOv7, YOLOv8)을 시도하거나, 현재 모델의 학습률, 배치 크기 등 하이퍼파라미터를 최적화합니다.
- **특정 클래스에 대한 추가 학습:** 성능이 낮은 특정 클래스 ('disease' 등)에 대한 데이터를 더 많이 학습시키거나, 클래스별 가중치를 조정합니다.
- **앙상블 기법:** 여러 모델의 예측 결과를 결합하여 최종 예측의 정확도를 높이는 앙상블 기법을 적용해 볼 수 있습니다.

2. Deep Learning(YOLO 11)

8. 결론

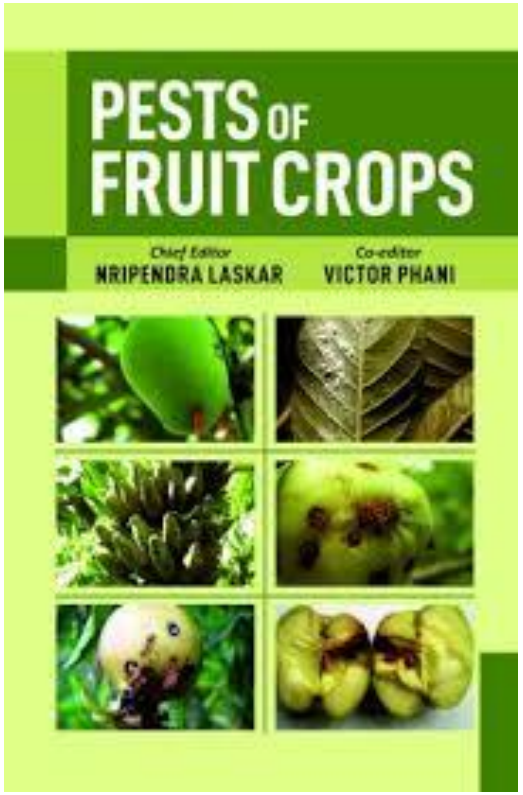
본 보고서는 YOLO 모델을 사용한 병충해 감지 학습 결과를 분석했습니다. 모델은 전반적으로 준수한 성능을 보였지만, 특히 객체의 정확한 위치 예측 능력과 특정 클래스에 대한 성능 향상이 필요합니다. 제시된 개선 방안들을 통해 모델의 성능을 더욱 발전시킬 수 있을 것으로 기대됩니다.

참고: 위 보고서는 실제 학습 결과가 아닌 예시를 기반으로 작성되었습니다. 실제 분석 보고서에는 사용된 데이터셋의 상세 정보, 학습 과정에 대한 더 자세한 설명, 시각화 결과 (예측 박스 시각화, Precision-Recall 곡선 등) 등이 포함될 수 있습니다.

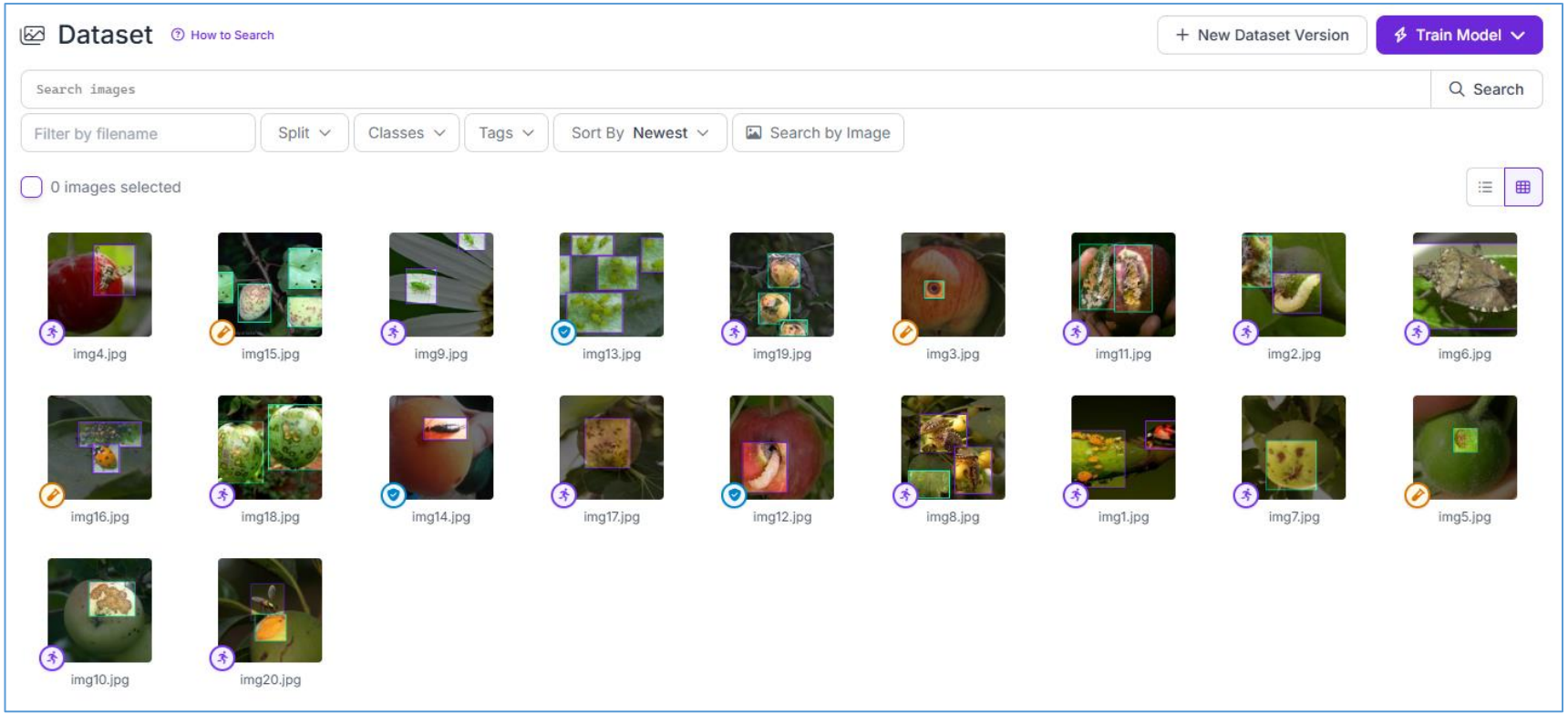
2. Deep Learning(YOLO 11)

실습) YOLO 커스텀 데이터로 학습/예측에 대해 리포트를 작성 공유하세요

2. Deep Learning(YOLO 11)

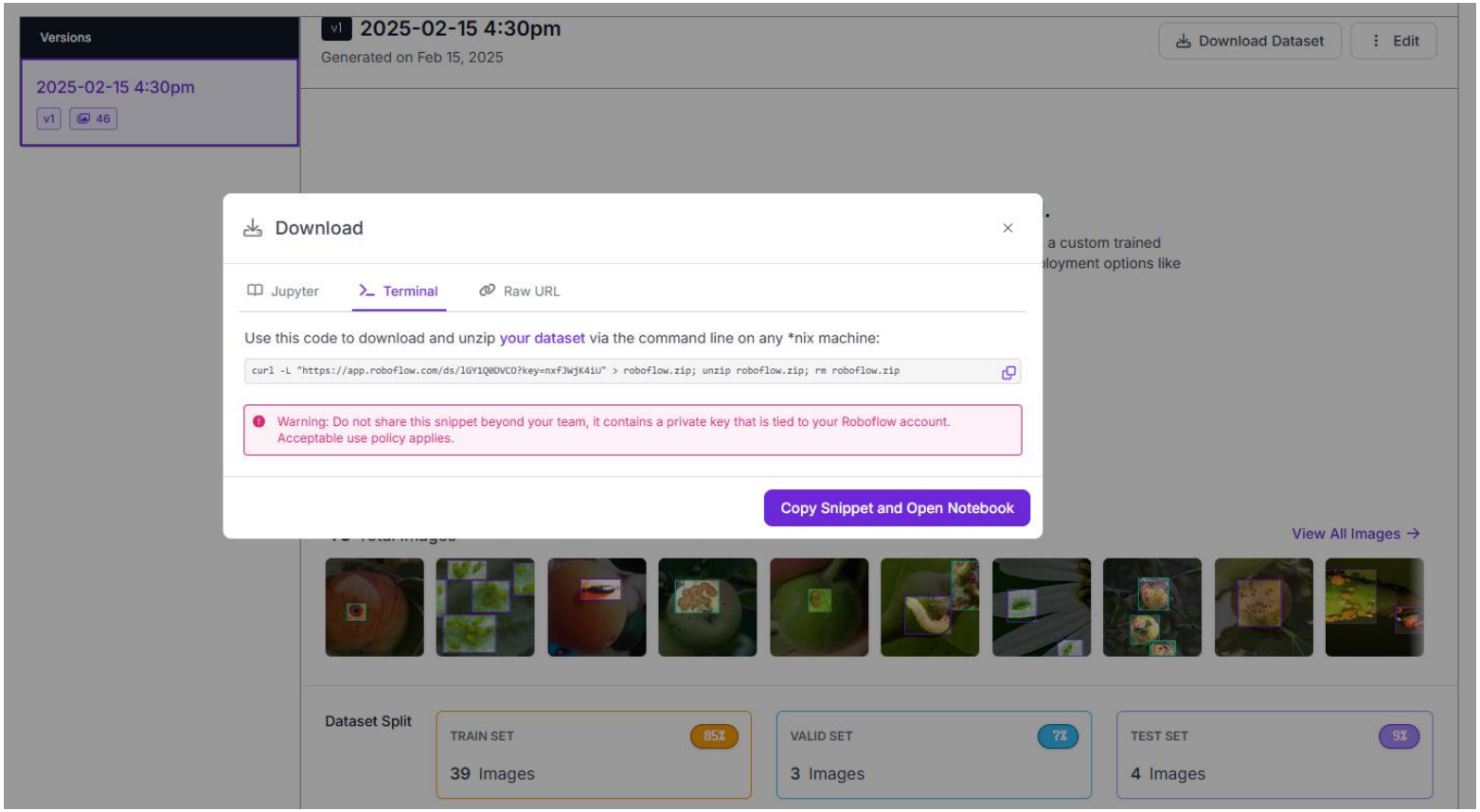


2. Deep Learning(YOLO 11)

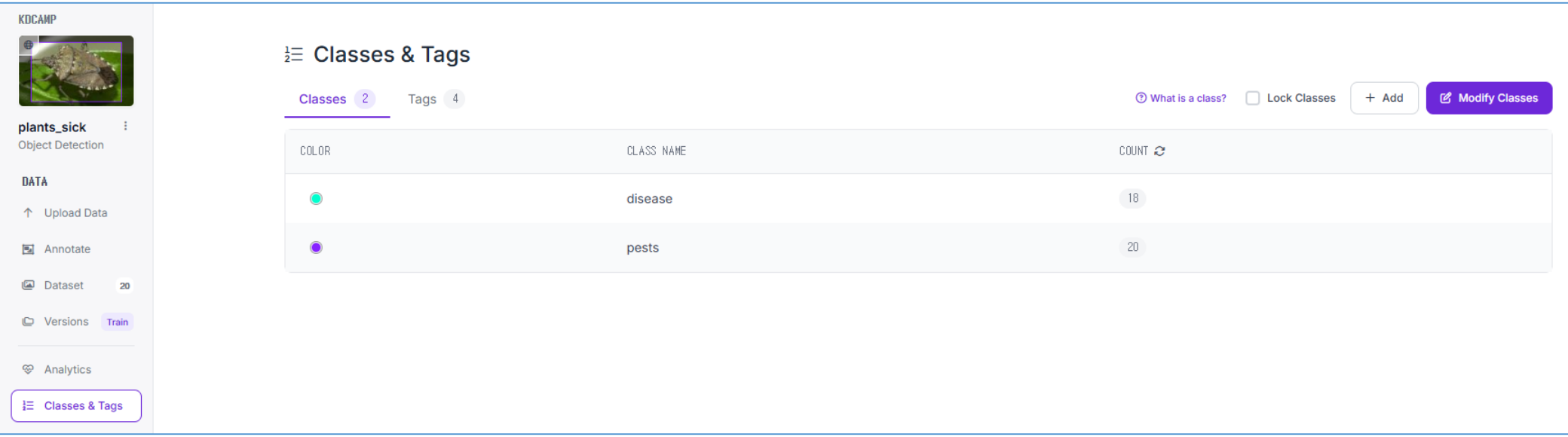


curl -L "https://app.roboflow.com/ds/lGY1Q0DVCO?key=nxfJWjK4iU" > roboflow.zip; unzip roboflow.zip; rm roboflow.zip

2. Deep Learning(YOLO 11)



2. Deep Learning(YOLO 11)



2. Deep Learning(YOLO 11)

PlantSick

```
[5]    1 !curl -L "https://app.roboflow.com/ds/IGY1QODVCO?key=nxfJWjK4iU" > roboflow.zip; unzip roboflow.zip; rm roboflow.zip
```

```
extracting: train/images/img4_jpg.rf.f3ace18704ec1bfc2c76bd7887c22a17.jpg
extracting: train/images/img6_jpg.rf.2a5ae8c905b40b9b443025841c92db68.jpg
extracting: train/images/img6_jpg.rf.48a3e65dc277eb7f5c4aad629734b0ec.jpg
extracting: train/images/img6_jpg.rf.d76b46ae7a5bb5cc29e65d8f6902c00c.jpg
extracting: train/images/img7_jpg.rf.5d8ef144066378b94106e28b1a5c2810.jpg
```

2. Deep Learning(YOLO 11)

```
1 !cat data.yaml
```

```
train: ../train/images
val: ../valid/images
test: ../test/images

nc: 2
names: ['disease', 'pests']

roboflow:
  workspace: kdcamp-65wrk
  project: plants_sick
  version: 1
  license: CC BY 4.0
  url: https://universe.roboflow.com/kdcamp-65wrk/plants_sick/dataset/1
```

2. Deep Learning(YOLO 11)

```
1 pip install ultralytics
```

```
Downloading nvidia_cuda_cupti_cu12-12.4.127-py3-none-manylinux2014_x86_64.whl (13.8 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 13.8/13.8 MB 81.8 MB/s eta 0:00:00
Downloading nvidia_cuda_nvrtc_cu12-12.4.127-py3-none-manylinux2014_x86_64.whl (24.6 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 24.6/24.6 MB 68.2 MB/s eta 0:00:00
Downloading nvidia_cuda_runtime_cu12-12.4.127-py3-none-manylinux2014_x86_64.whl (883 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 883.7/883.7 kB 46.7 MB/s eta 0:00:00
Downloading nvidia_cudnn_cu12-9.1.0.70-py3-none-manylinux2014_x86_64.whl (664.8 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 664.8/664.8 MB 2.1 MB/s eta 0:00:00
```

```
1 from ultralytics import YOLO
2 model = YOLO('yolov8s.pt')
3
4 print(type(model.names), len(model.names))
5 print(model.names)
```

```
Creating new Ultralytics Settings v0.0.6 file ✅
View Ultralytics Settings with 'yolo settings' or at '/root/.config/Ultralytics/settings.json'
Update Settings with 'yolo settings key=value', i.e. 'yolo settings runs_dir=path/to/dir'. For help see https://docs.ultralytics.com/quickstart/#ultralytics-settings.
Downloading https://github.com/ultralytics/assets/releases/download/v8.3.0/yolov8s.pt to 'yolov8s.pt'...
100%|██████████| 21.5M/21.5M [00:00<00:00, 115MB/s]
<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
```

2. Deep Learning(YOLO 11)

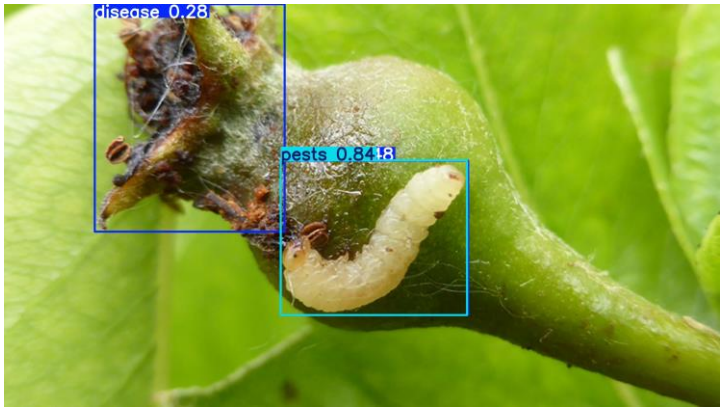
```
model.train(data='/content/data.yaml', epochs=10, batch=32, imgsz=416, verbose=False)
```

```
      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
      10/10      3.61G      1.128      1.492      1.331         13        416: 100%|██████████| 2/2 [00:00<00:00,  5.02it/s]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 1/1 [00:00<00:00, 23.27it/s]


10 epochs completed in 0.005 hours.
Optimizer stripped from runs/detect/train/weights/last.pt, 22.5MB
Optimizer stripped from runs/detect/train/weights/best.pt, 22.5MB

Validating runs/detect/train/weights/best.pt...
Ultralytics 8.3.108 🚀 Python-3.11.12 torch-2.6.0+cu124 CUDA:0 (Tesla T4, 15095MiB)
Model summary (fused): 72 layers, 11,126,358 parameters, 0 gradients, 28.4 GFLOPs
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 1/1 [00:00<00:00, 35.12it/s]
                   all          3          7      0.467      0.286      0.303      0.101
Speed: 0.1ms preprocess, 5.1ms inference, 0.0ms loss, 1.1ms postprocess per image
Results saved to runs/detect/train
ultralytics.utils.metrics.DetMetrics object with attributes:
```

2. Deep Learning(YOLO 11)

```
results=model.predict(source='/content/bug1.jpg',save=True)
```

```
Results saved to runs/detect/train2
```



2. Deep Learning(YOLO 11)

실습) 쥬피터 노트북과 GPU로 학습해 보기