

AI+X 고급 (4주차)

# AI Development Strategy and Data Collection

동기부여 특허  
출원서 초안 작성 中

# 단계별 순화 특허 해외특허 추진 中 (미국)

## 학교의 심사 통과!

특허심의위원회 심의 결과 해외출원에 문제가 없을 것으로 보이며, 출원 희망 기간 내 신속히 출원할 수 있도록 협력하는 것이 바람직하다고 판단되어, 해외출원 진행하시면 될 것 같습니다.



Silicon Valley

# SCI 논문

## 11월內 Submit 목표

# 이번학기 과제

# The Plan (1/2)

9/8 Intro

9/15 6 Ideas Presentation (15% 결과: Pass)  
(KT 교수님 지도)

9/22 Theme 확정, 데이터 수집 검토

※ 데이터 수집계획(案) (어떤 데이터를 어떻게?) 9/29(금)까지 제출 (제출 완료)

9/29 추석

# The Plan (2/2)

10/6 데이터 수집 및 전처리 착수, 모델링 착수

10/13 모델링 지속, 중간고사 발표 가이드

10/20 데이터 및 모델링 점검 (KT 교수님 지도)

10/27 모델링 개선 및 Back/Front 개발 착수

11/3 **중간고사 발표** (KT 교수님 지도) (20%)



피드백

# Project Scope

1. 컨셉 구체화 및 기능 및 서비스 정의
2. 데이터 수집 및 전처리 로드가 큼  $\pi$
3. AI 모델링
4. Backend/Frontend 개발

주어진 것은 한 학기

# Scoping

1. PM은 **킥보드로 한정**
2. 불법주차 위치가 실제로는 다양하지만  
→ 1~2개로 선정

# 고민들 (모델링)

- 세그멘테이션과 바운딩 박스를 Yolo 하나의 모델로 동시에 처리할 수 없다.
  - 도로나 횡단보도같은 영역은 세그멘테이션으로 분류, PM과 같은 객체는 바운딩 박스로 검출 할 수 있으면 좋을것 같다.
  - 도로같은 영역은 FCN, U-net으로 많이 한다고....
  - 객체탐지는 Yolov8
- 2개의 모델을 구축해 합치기...가 제일 베스트
- 불법주차에 해당하는 거리를 계산하는 문제는 별도 (킵보드의 비율을 이용해서 계산?) → 공부 필요

# 개발순서

1. YOLO 통한 키포드 Object Detection 개발 (Model 1)

2. 합법/불법 분류 모델 (Model 2)

- **불법위치**를 인식하는게 제일 중요!
- 여러가지 불법주차 위치에서의 사진을 찍어 학습하기엔 시간이 많이 걸리므로,  
Roboflow  
**1~2개 불법위치만 선정**하고 사진 모우고 증강 및 학습
- 핵심은 1~2개가 빈번하고 자주 “불법 ” 하는 곳이어야함  
*줄인다고 아무거나 선정하면 안됨*

# 개발순서

## 2. 합법/불법 분류 모델 (Model 2)

- 불법위치를 인식하는게 제일 중요!
- 여러가지 불법주차 위치에서의 사진을 찍어 학습하기엔 시간이 많이 걸리므로,  
1~2개 불법위치만 선정하고 사진 모우고 증강 및 학습
- 핵심은 1~2개가 **빈번하고 자주 “불법 ” 하는 곳이어야함**
  - . 버스정류장이나 지하철입구 등 사람들이 많이 이용하는데 위치한 키포드를 찾아 사진 찍고 이미지 학습, 이미지 불법/합법 분류
  - . AND/OR 위도경도로 지하철위치, 버스정류장위치 찾아 불법 여부 확인
  - . 키포드와 불법위치가 포함된 사진을 모으는게 핵심
- CF) 키포드와 불법위치 포함된 사진에서 키포드와 불법위치 둘 다 객체탐지 되면 좋은데 쉽지 않음

# 개발순서

3. 사진을 업로드 →
  1. �보드 Detection 모델
  2. 합법/불법 분류 모델
4. �보드 신고사이트 자동 연동 (서비스 기획 중)

# 고민들 (Data)

2. 데이터 수집 관련해서는 다음의 고민이 있습니다.

roboflow에 scooter라고 검색하면 많이 나옴 → PM의 데이터는 문제 없을 듯

- 지하철 역 출입구, 교통섬을 제외하면 대부분 오픈 데이터 셋이 존재
- 라벨링이 되어 있는지는 불명확



# 피드백

1. 오픈소스에서 제공하는 데이터를 최대한 활용
2. 없으면 Custom으로 만들어야하는데  
이미지 사진 확보하는데 **많은 시간 소요**
3. 필요에 따라 Labeling, Bounding Box도 필요

# YOLO (You Only Look Once)

1 Stage Detector

YOLO

## You Only Look Once: Unified, Real-Time Object Detection

Joseph Redmon\*, Santosh Divvala<sup>†</sup>, Ross Girshick<sup>‡</sup>, Ali Farhadi<sup>†</sup>

University of Washington\*, Allen Institute for AI<sup>†</sup>, Facebook AI Research<sup>‡</sup>

<http://pjreddie.com/yolo/>

### Abstract

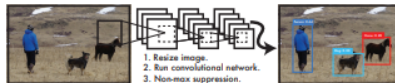
We present YOLO, a new approach to object detection. Prior work on object detection repurposes classifiers to perform detection. Instead, we frame object detection as a regression problem to spatially separated bounding boxes and associated class probabilities. A single neural network predicts bounding boxes and class probabilities directly from full images in one evaluation. Since the whole detection pipeline is a single network, it can be optimized end-to-end directly on detection performance.

Our unified architecture is extremely fast. Our base YOLO model processes images in real-time at 45 frames per second. A smaller version of the network, Fast YOLO, processes an astounding 155 frames per second while still achieving double the mAP of other real-time detectors. Compared to state-of-the-art detection systems, YOLO makes more localization errors but is less likely to predict false positives on background. Finally, YOLO learns very general representations of objects. It outperforms other detection methods, including DPM and R-CNN, when generalizing from natural images to other domains like artwork.

### 1. Introduction

Humans glance at an image and instantly know what objects are in the image, where they are, and how they interact. The human visual system is fast and accurate, allowing us to perform complex tasks like driving with little conscious thought. Fast, accurate algorithms for object detection would allow computers to drive cars without specialized sensors, enable assistive devices to convey real-time scene information to human users, and unlock the potential for general purpose, responsive robotic systems.

Current detection systems repurpose classifiers to perform detection. To detect an object, these systems take a classifier for that object and evaluate it at various locations



**Figure 1: The YOLO Detection System.** Processing images with YOLO is simple and straightforward. Our system (1) resizes the input image to  $448 \times 448$ , (2) runs a single convolutional network on the image, and (3) thresholds the resulting detections by the model’s confidence.

methods to first generate potential bounding boxes in an image and then run a classifier on these proposed boxes. After classification, post-processing is used to refine the bounding boxes, eliminate duplicate detections, and rescore the boxes based on other objects in the scene [13]. These complex pipelines are slow and hard to optimize because each individual component must be trained separately.

We reframe object detection as a single regression problem, straight from image pixels to bounding box coordinates and class probabilities. Using our system, you only look once (YOLO) at an image to predict what objects are present and where they are.

YOLO is refreshingly simple: see Figure 1. A single convolutional network simultaneously predicts multiple bounding boxes and class probabilities for those boxes. YOLO trains on full images and directly optimizes detection performance. This unified model has several benefits over traditional methods of object detection.

First, YOLO is extremely fast. Since we frame detection as a regression problem we don’t need a complex pipeline. We simply run our neural network on a new image at test time to predict detections. Our base network runs at 45 frames per second with no batch processing on a Titan X GPU and a fast version runs at more than 150 fps. This means we can process streaming video in real-time with less than 25 milliseconds of latency. Furthermore, YOLO



## CVPR 2016 open access

These CVPR 2016 papers are the Open Access versions, provided by the [Computer Vision Foundation](#).

Except for the watermark, they are identical to the accepted versions; the final published version of the proceedings is available on IEEE Xplore.

*This material is presented to ensure timely dissemination of scholarly and technical work. Copyright and all rights therein are retained by authors or by other copyright holders. All persons copying this information are expected to adhere to the terms and constraints invoked by each author's copyright.*

# You Only Look Once: Unified, Real-Time Object Detection

**Joseph Redmon, Santosh Divvala, Ross Girshick, Ali Farhadi**; Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016, pp. 779-788

## Abstract

We present YOLO, a new approach to object detection. Prior work on object detection repurposes classifiers to perform detection. Instead, we frame object detection as a regression problem to spatially separated bounding boxes and associated class probabilities. A single neural network predicts bounding boxes and class probabilities directly from full images in one evaluation. Since the whole detection pipeline is a single network, it can be optimized end-to-end directly on detection performance. Our unified architecture is extremely fast. Our base YOLO model processes images in real-time at 45 frames per second. A smaller version of the network, Fast YOLO, processes an astounding 155 frames per second while still achieving double the mAP of other real-time detectors. Compared to state-of-the-art detection systems, YOLO makes more localization errors but is less likely to predict false positives on background. Finally, YOLO learns very general representations of objects. It outperforms other detection methods, including DPM and R-CNN, when generalizing from natural images to other domains like artwork.

## Related Material

- 이전까지는 Faster R-CNN Architecture 사용 (7 FPS, 실시간 X)
- '15년 YOLO는 45 FPS 성능으로

### Object Detection (객체 검출) 분야 혁신을 이룸

- YOLO v5부터 PyTorch로 개발이 되었고, (이제는 상대적으로 쉬운 구조)

현재 v8은

(1) 파이썬 인터페이스로 개발 가능한 v8 Architecture로 발전

(2) 성능 우위 (SOTA)

(3) 1-Stage Detector (“이미지 전체를 한 번만 본다!”)

CF) 기존 CNN처럼 이미지를 여러장으로 분할해 해석 X

<https://github.com/ultralytics>

모든 소스 공개!

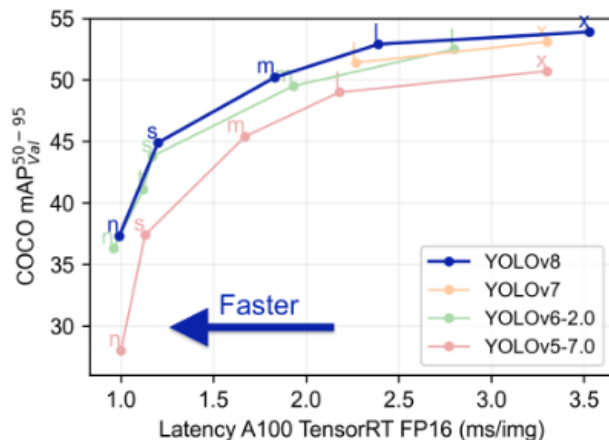
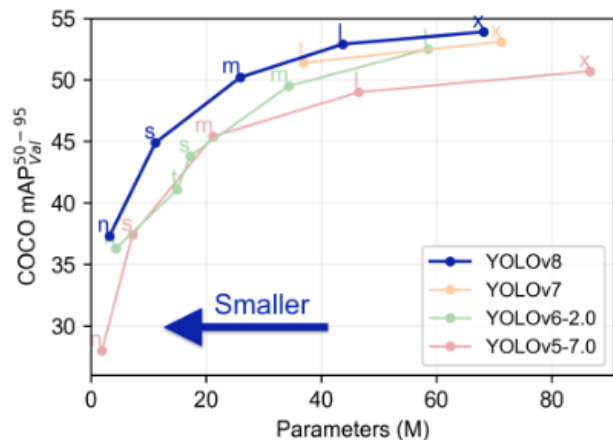
# <https://github.com/ultralytics>

[Ultralytics YOLOv8](#) is a cutting-edge, state-of-the-art (SOTA) model that builds upon the success of previous YOLO versions and introduces new features and improvements to further boost performance and flexibility. YOLOv8 is designed to be fast, accurate, and easy to use, making it an excellent choice for a wide range of object detection and tracking, instance segmentation, image classification and pose estimation tasks.

이것을 할 수 있다!

We hope that the resources here will help you get the most out of YOLOv8. Please browse the YOLOv8 [Docs](#) for details, raise an issue on [GitHub](#) for support, and join our [Discord](#) community for questions and discussions!

To request an Enterprise License please complete the form at [Ultralytics Licensing](#).



패러미터 수는 적어지고 속도는 증가함!



Is this a dog?



Image Classification

What is there in image  
and where?



Object Detection

Which pixels belong to  
which object?



Image Segmentation

# Visual Tracking

영상에서 시간에 따라 움직이는 객체 위치 찾는 과정



# Pose Estimation

영상에서 인물을 탐지(Detect)해서 인체 부위의 위치를 식별하고 부위를 연결하는 선 찾아서 자세 추정. 게임, AR, 스포츠, 헬스케어에서 활용



- YOLO V8

. Object Detection + 이미지, 동영상의 Image Segmentation을 동일 API로 구현 가능해짐

# 동일한 API의 사용!

## Object Detection

```
from ultralytics import YOLO

# Load a pre-trained model
model = YOLO("yolov8n.pt")

# Train the model
model.train(data="data.yaml", epochs=100, imgsz=640)

# Predict with the model
results = model.predict(source="test.jpg")
```

## Image Segmentation

```
from ultralytics import YOLO

# Load a pre-trained model
model = YOLO("yolov8n-seg.pt")

# Train the model
model.train(data="data-seg.yaml", epochs=100, imgsz=640)

# Predict with the model
results = model.predict(source="test.jpg")
```

# UltraLytics 패키지

옛날에!



- DarkNet Framework 기반 YOLO v3을 PyTorch로 변환
- YOLO v5 개발

# UltraLytics 패키지

<https://github.com/ultralytics>

## Pinned

 **ultralytics** Public

NEW - YOLOv8 🚀 in PyTorch > ONNX > OpenVINO > CoreML > TFLite

Python 13.5k 2.6k

 **yolov5** Public

YOLOv5 🚀 in PyTorch > ONNX > CoreML > TFLite

Python 42k 14.7k

 **yolov3** Public


YOLOv3 in PyTorch > ONNX > CoreML > TFLite

Python 9.7k 3.4k

 **xview-yolov3** Public


xView 2018 Object Detection Challenge: YOLOv3 Training and Inference.

Python 228 58

 **hub** Public

Ultralytics HUB tutorials and support

Jupyter Notebook 76 8

 **JSON2YOLO** Public

Convert JSON annotations into YOLO format.

Python 523 169

# UltraLytics Quickstart

<https://docs.ultralitics.com/quickstart/>



# General Development Process

## 1. Data Preparation

(1) YOLO v8의 PT(MS COCO DB 기학습)만으로 예측할꺼면 Image만 준비

(2) Custom Dataset으로 FT하는 경우 Image 및 Annotation 필요

a. Roboflow의 Training Dataset 활용

b. Labeling Tool 이용해 Labeling시킨 Image / Annotation을 통한

Training Dataset 구축!

# General Development Process

## 2. Loading Data and Preparation

(1) Google Colab으로 데이터셋 로딩

(2) `pip install ultralytics`

# General Development Process

## 3. Train Model

Train model

```
from ultralytics import YOLO
```

```
model = YOLO('yolov8n.pt')
```

```
model.train(data='data.yaml', epochs=10)
```

```
# load a pretrained detection model
```

```
# train the model
```

```
from ultralytics import YOLO
```

```
model = YOLO('yolov8n-seg.pt')
```

```
model.train(data='data-seg.yaml', epochs=10)
```

```
# load a pretrained segmentation model
```

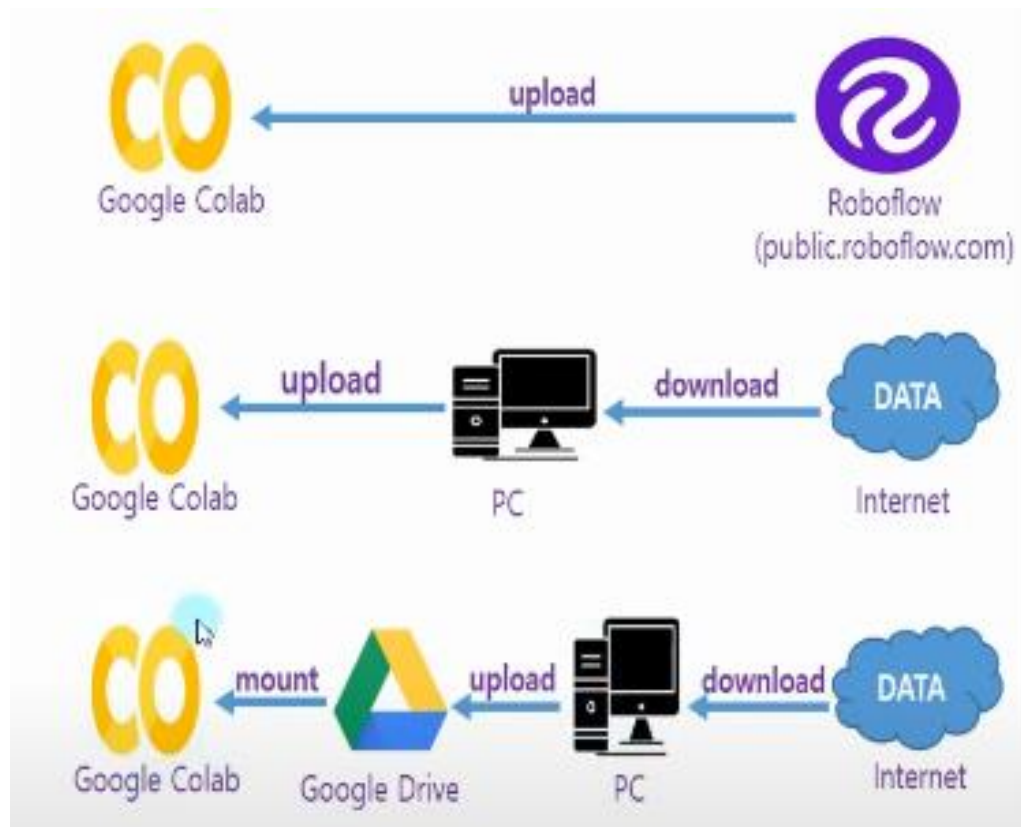
```
# train the model
```

# General Development Process

## 4. Prediction

```
Results = model.predict('my_test_data.jpg')    # predict on a test image
```

## 개발 환경 세팅 (데이터 연동)



# Roboflow

<https://roboflow.com/>

**roboflow**

Product ▾ Solutions ▾ Resources ▾ Pricing Docs


Talk to Sales

Sign in →

## Everything you need to build and deploy computer vision models

Used by over 250,000 engineers to create datasets, train models, and deploy to production.

Get Started →




Upload

Webcam

Try on Mobile

Microsoft COCO ▾

 [github](#) [homepage-demo](#)

# Roboflow

<https://roboflow.com/>

- CV 기술을 이용해 다양한 App 개발을 지원하는 서비스
- 데이터 셋 생성, 전처리 작업, 증강 작업을 한 큐에!
- 다양한 무료 Dataset 제공
- Custom Dataset 구축을 위한 Bounding Box 툴!

# Choose a plan for you or your team

## FOR COMMUNITY

### Public

Share your personal projects, class assignments, and experiments to unlock all our premium features

From \$0.00

per month, pay as you go for training and inference costs

 Public Projects

Select Public

No credit card required

## FOR BUSINESS

### Sandbox

Experiment with your business' images and data in a private environment until ready to deploy to production

Free

For evaluation purposes only

 Private Projects

Select Sandbox

No credit card required

## FOR BUSINESS

### Growth

Deploy private models into production and continually improve performance through active learning

Starts at \$1,000

per month, customized to your needs

 Private Projects

Talk to Sales

Or call (415) 938-4001

## FOR BUSINESS

### Enterprise

All the features of Growth, plus custom contracts, audit logs security, and dedicated support

Custom pricing

Based on your needs

 Private Projects

Talk to Sales

Or call (415) 938-4001



## YOLO에 맞는 데이터 형태로 Export 가능

333 images

Version 2 Generated Mar 24, 2021

Export

More ⋮

---

TRAINING OPTIONS

Use Roboflow Train

Let us train your model and get results within 24 hours along with a hosted API endpoint for making predictions. [Learn More »](#)

Start Training

Available Credits: 0

Train Outside Roboflow

Export your data to use a model from [our model library »](#) with Google Colab or your own machine.

Format

YOLO v5 PyTorch

Export

- 다운 가능한 링크 코드가 생성, YOLO 학습전 실행, 폴더 및 **data.yaml** 생성됨!

## WORKSPACES

Hwanython

1

New Workspace

1

New Workspace

0

+ Add Workspace

## RESOURCES

Getting Started

Tutorials

Public Datasets

Model Library

Help &amp; Support

Welcome to Roboflow, let's get started.

**Check Out a Dataset Health Check**

Get insights into your data quality and potential issues.

View

2 minutes



Hwanython



H

Invite



Create New Project

**Hard Hat Sample**

Private



Modified 16 hours ago

...

## Computer Vision Datasets

Roboflow hosts free public computer vision datasets in many popular formats (including CreateML JSON, COCO JSON, Pascal VOC XML, YOLO v3, and Tensorflow TFRecords). For your convenience, we also have downsized and augmented versions available.

If you'd like us to host your dataset, please [get in touch](#).

### Anki Vector Robot Dataset Dataset

☐ Object Detection (Bounding Box)

1193 images   8 exports   Last updated a day ago



### EgoHands Dataset

☐ Object Detection (Bounding Box)

4800 images   5 exports   Last updated 3 months ago



### Microsoft COCO 2017 Dataset

☐ Object Detection (Bounding Box)

121448 images   9 exports   Last updated 4 months ago



### North American Mushrooms Dataset

☐ Object Detection (Bounding Box)

51 images   3 exports   Last updated 5 months ago



### Cottontail-Rabbits Dataset

☐ Object Detection (Bounding Box)

95 images   4 exports   Last updated 5 months ago



**roboflow**

UniversePublic DatasetsModel ZooBlogDocs

DATASET TYPE

All Datasets40

Object Detection36

Classification4

## Computer Vision Datasets

Roboflow hosts free public computer vision datasets in many popular formats (including CreateML JSON, COCO JSON, Pascal VOC XML, YOLO v3, and Tensorflow TFRrecords). For your convenience, we also have downzipped and augmented versions available.

If you'd like us to host your dataset, please [get in touch](#).

**Roboflow 100**

Object Detection Benchmark

100 datasets | 7 Domains | 224k Images

☐ **Microsoft COCO 2017 Dataset**

Object Detection (Bounding Box)

120362 images | 10 exports | Last updated 4 days ago

☐ **Pascal VOC 2012 Dataset**

Object Detection (Bounding Box)

17112 images | 9 exports | Last updated 9 days ago

☐ **Aquarium Dataset**

Object Detection (Bounding Box)

638 images | 5 exports | Last updated 19 days ago

**roboflow**

UniversePublic DatasetsModel ZooBlogDocs

Dataset Summary

Dataset Health Check

DOWNLOADS

raw-1024638

Try Pre-Trained Model

## Aquarium Dataset

Shared By

**Roboflow**

November 2020

License

**CC BY 4.0**

More Info

Annotations

**creatures**

Object Detection

Downloads

raw-1024638 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.

Projects Using this Dataset:

No Code Mobile

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

66+ MILLION IMAGES

90,000+ DATASETS

7,000+ PRE-TRAINED MODELS

Dataset Health Check

## DOWNLOADS

raw-1024

638

Export Created  
2 years ago  
November 19, 2020Export Size  
638 images

## Available Download Formats

## Export

Format

YOLOv8

TXT annotations and YAML config used with YOLOv8.

☐ download zip to computer ☒ show download code

Cancel

Continue

YOLOv8

JSON

COCO

CreateML

XML

Pascal VOC

TXT

YOLO Darknet

YOLO v3 Keras

YOLO v4 PyTorch

Scaled-YOLOv4

YOLOv5 Oriented Bounding Boxes

meituan/YOLOv6

YOLO v5 PyTorch

YOLO v7 PyTorch

YOLOv8

CSV

Tensorflow Object Detection

RetinaNet Keras

Multi-Label Classification

Other

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

## Aquarium Dataset » raw-1024

Export Created

2 years ago

November 19, 2020

Available Download Formats

COC

YOLO v

YOLO v

### Your Download Code

Jupyter

Terminal

Raw URL

The direct link to download **your zip file** » is:

<https://public.roboflow.com/ds/FAggOgdewl?key=1uz2izrHYH>

**Warning:** Do not share this snippet beyond your team, it contains a private key that is tied to your Roboflow account. Acceptable use policy applies.

Done

Choose a Model

Annotations

creatures

arknet TXT

YOLO v3 Keras TXT

YOLOv6

YOLO v5 PyTorch

Net Keras

SV

Multiclass

Classification

- 코랩에서 wget을 통해 데이터 가져옴

## Augmentation Options



Augmentations create new training examples for your model to learn from.

### IMAGE LEVEL AUGMENTATIONS



Flip



90° Rotate



Crop



Rotation



Shear



Grayscale



Hue



Saturation



Brightness



Exposure



Blur



Noise

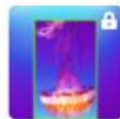


Cutout



Mosaic

### BOUNDING BOX LEVEL AUGMENTATIONS



Flip



90° Rotate



Crop



Rotation



Shear



Brightness



Exposure



Blur



Noise

# YOLO 활용 코드 예시 (기존 PT 모델 활용)



## ➤ Data Preparation and Load Data

```
import os
import zipfile

with zipfile.ZipFile('/content/test_image_dir.zip') as target_file:
    target_file.extractall('/content/test_image_dir')
```

```
print('test images = ', os.listdir('/content/test_image_dir'))
```

```
test_images = ['test3.jpg', 'test2.jpg', 'test4.jpg', 'test1.jpg']
```



사전학습되어 있는 YOLO 모델의 prediction 기능만을 이용해서 예측할 것이므로, 이러한 예측에 사용되는 총 4장의 test image(test1.jpg~test4.jpg)를 Colab에 업로드 함 ('/content/test\_image\_dir')

이미지 출처: MS COCO Dataset

test1.jpg



test2.jpg



test3.jpg



test4.jpg



## ➤ Install YOLOv8

```
!pip install ultralytics
```

 ← YOLOv8와 YOLOv8 실행에 필요한 라이브러리 설치

```
import ultralytics
```

```
ultralytics.checks()
```

Ultralytics YOLOv8.0.49 🚀 Python-3.8.10 torch-1.13.1+cu116 CPU  
Setup complete ✅ (2 CPUs, 12.7 GB RAM, 25.8/107.7 GB disk)

## ➤ Load a pre-trained model

```
from ultralytics import YOLO
```

```
model = YOLO('yolov8n.pt')
```

 ← MS COCO Dataset 사전학습된 yolov8n 모델을 로드함.  
yolov8n 외에도 yolov8s, yolov8m, yolov8l, yolov8x 등이 있음

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, 15.2MB/s]

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

```
print(model.names)
```

 ← MS COCO Dataset 에 정의되어 있는 클래스 개수와 종류는 model.names 를 통해서 확인할수 있음 (총 80개, 0~79)

```
<class 'dict'> 80
```

```
{0: 'person', 1: 'bicycle', 2: 'car', 3: 'motorcycle', 4: 'airplane', 5: 'bus', 6: 'train', 7: 'truck', 8: 'boat', 9: 'traffic light', 10: 'fire hydrant', 11: 'stop sign', 12: 'parking meter', 13: 'bench', 14: 'parking meter', 15: 'stop sign', 16: 'parking meter', 17: 'stop sign', 18: 'parking meter', 19: 'stop sign', 20: 'parking meter', 21: 'stop sign', 22: 'parking meter', 23: 'stop sign', 24: 'parking meter', 25: 'stop sign', 26: 'parking meter', 27: 'stop sign', 28: 'parking meter', 29: 'stop sign', 30: 'parking meter', 31: 'stop sign', 32: 'parking meter', 33: 'stop sign', 34: 'parking meter', 35: 'stop sign', 36: 'parking meter', 37: 'stop sign', 38: 'parking meter', 39: 'stop sign', 40: 'parking meter', 41: 'stop sign', 42: 'parking meter', 43: 'stop sign', 44: 'parking meter', 45: 'stop sign', 46: 'parking meter', 47: 'stop sign', 48: 'parking meter', 49: 'stop sign', 50: 'parking meter', 51: 'stop sign', 52: 'parking meter', 53: 'stop sign', 54: 'parking meter', 55: 'stop sign', 56: 'parking meter', 57: 'stop sign', 58: 'parking meter', 59: 'stop sign', 60: 'parking meter', 61: 'stop sign', 62: 'parking meter', 63: 'stop sign', 64: 'parking meter', 65: 'stop sign', 66: 'parking meter', 67: 'stop sign', 68: 'parking meter', 69: 'stop sign', 70: 'parking meter', 71: 'stop sign', 72: 'parking meter', 73: 'stop sign', 74: 'parking meter', 75: 'stop sign', 76: 'parking meter', 77: 'stop sign', 78: 'parking meter', 79: 'stop sign'}
```

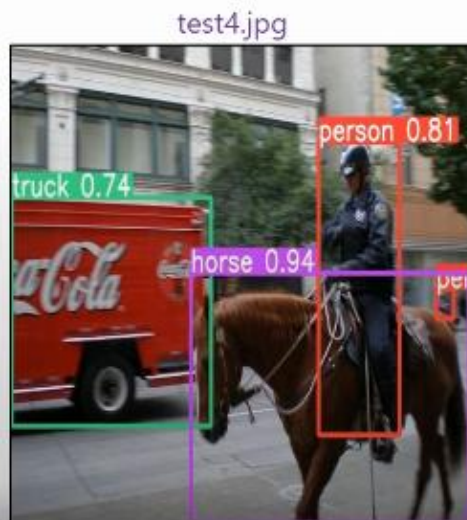
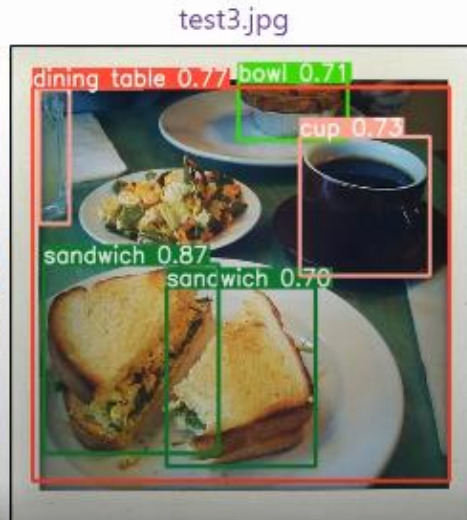
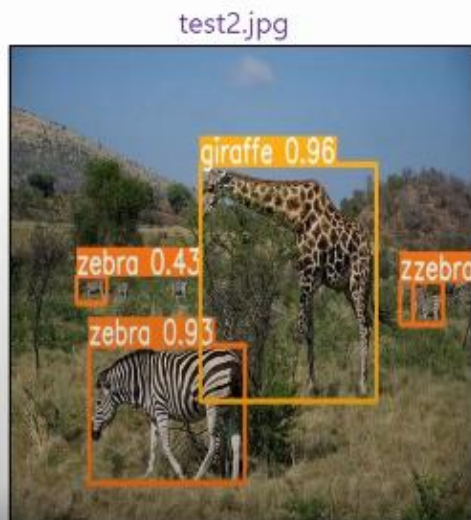
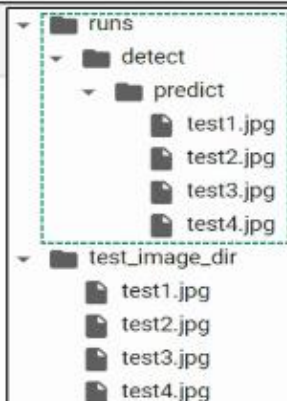
## Prediction

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

Ultralytics YOLOv8.0.49 Python-3.8.10 torch-1.13.1+cu116 CPU  
YOLOv8n summary (fused): 168 layers, 3151904 parameters, 0 gradients, 8.7 GFL0Ps

주어진 이미지에 포함된 객체의 종류와 개수를 나타냄

image 1/4 /content/test\_image\_dir/test1.jpg: 544x640 1 laptop, 1 mouse, 2 keyboards, 1 cell phone, 324.6ms  
image 2/4 /content/test\_image\_dir/test2.jpg: 448x640 4 zebras, 1 giraffe, 179.4ms  
image 3/4 /content/test\_image\_dir/test3.jpg: 640x640 2 cups, 1 bowl, 2 sandwiches, 1 dining table, 264.2ms  
image 4/4 /content/test\_image\_dir/test4.jpg: 480x640 2 persons, 1 truck, 1 horse, 188.2ms  
Speed: 2.5ms preprocess, 239.1ms inference, 7.0ms postprocess per image at shape (1, 3, 640, 640)  
Results saved to runs/detect/predict





```
import numpy as np
```

```
for result in results: ← results = model.predict(...)
```

```
uniq, cnt = np.unique(result.bboxes.cls.cpu().numpy(), return_counts=True) # Torch.Tensor -> numpy  
uniq_cnt_dict = dict(zip(uniq, cnt))
```

```
print('\n{class num:counts} =', uniq_cnt_dict, '\n')
```

객체의 종류를 나타내는 고유 값이 result.bboxes.cls 에 저장되어 있으며, 현재는 YOLOv8 모델을 그대로 사용했기 때문에 MS COCO Dataset 에서 정의한 0~79 값이 디폴트로 사용되고 있음

```
for c in result.bboxes.cls:  
    print('class num =', int(c), ', class_name =', model.names[int(c)])
```

```
{class num:counts} = {63.0: 1, 64.0: 1, 66.0: 2, 67.0: 1}
```

```
class num = 64 , class_name = mouse  
class num = 63 , class_name = laptop  
class num = 66 , class_name = keyboard  
class num = 67 , class_name = cell phone  
class num = 66 , class_name = keyboard
```

test1.jpg

```
{class num:counts} = {22.0: 4, 23.0: 1}
```

```
class num = 23 , class_name = giraffe  
class num = 22 , class_name = zebra  
class num = 22 , class_name = zebra  
class num = 22 , class_name = zebra  
class num = 22 , class_name = zebra
```

test2.jpg

```
{class num:counts} = {41.0: 2, 45.0: 1, 48.0: 2, 60.0: 1}
```

```
class num = 48 , class_name = sandwich  
class num = 60 , class_name = dining table  
class num = 41 , class_name = cup  
class num = 41 , class_name = cup  
class num = 45 , class_name = bowl  
class num = 48 , class_name = sandwich
```

test3.jpg

```
{class num:counts} = {0.0: 2, 7.0: 1, 17.0: 1}
```

```
class num = 17 , class_name = horse  
class num = 0 , class_name = person  
class num = 7 , class_name = truck  
class num = 0 , class_name = person
```

test4.jpg

# YOLO 활용 코드 예시

## (Custom Data 활용)

```
!wget -O Aquarium_Data.zip https://public.roboflow.com/ds/FAgq0gdewl?key=1uz2izrHYH
```

```
import zipfile

with zipfile.ZipFile('/content/Aquarium_Data.zip') as target_file:

    target_file.extractall('/content/Aquarium_Data/') ----->
```

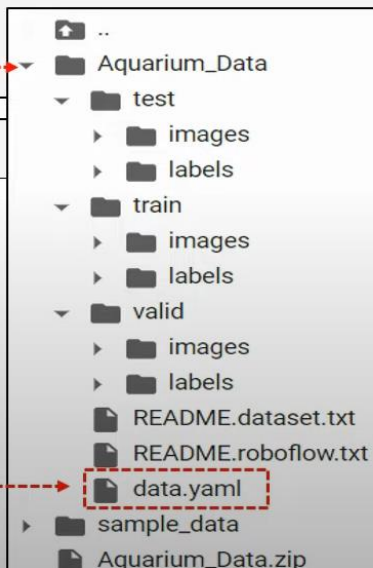
```
!cat /content/Aquarium_Data/data.yaml ----->
```

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

실제 커스텀 데이터가 저장되어 있는 train, valid  
디렉토리 경로로 반드시 변경해줘야 함

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

```
roboflow:
  workspace: brad-dwyer
  project: aquarium-combined
  version: 2
  license: CC BY 4.0
```



Make YAML file	YOLOv8으로 Custom Data를 학습하기 위해서는 YAML 파일이 반드시 필요한데, 이러한 YAML 파일에는 다음과 같은 정보가 저장되어 있어야 함. [1] 이미지와 정답이 저장되어 있는 디렉토리 정보 [2] 인식(Detection)하고 싶은 클래스 종류와 대응되는 각각의 이름
<p style="text-align: center;">YAML 예시</p> <pre> train: ../train/images val: ../valid/images test: ../test/images  nc: 7 names: ['fish', 'jellyfish', 'penguin', 'puffin', 'shark', 'starfish', 'stingray'] </pre>	

그런데 Roboflow에서  
만들어줌

Install YOLOv8	<code>pip install ultralytics</code> # yolov8 실행에 필요한 라이브러리 설치 및 dependency 체크
----------------	--

Train model	<pre> from ultralytics import YOLO  model = YOLO('yolov8n.pt')          # 사전학습모델 yolov8n.pt 로드 model.train(data='mydata.yaml', epochs=10)  # mydata.yaml 참조하여 학습(파인튜닝) </pre>
-------------	---

Prediction	<code>results = model.predict(source='/content/test/')</code> # predict on test images
------------	--

# YAML

Yet Another Markup Language → YAML Ain't Markup Language

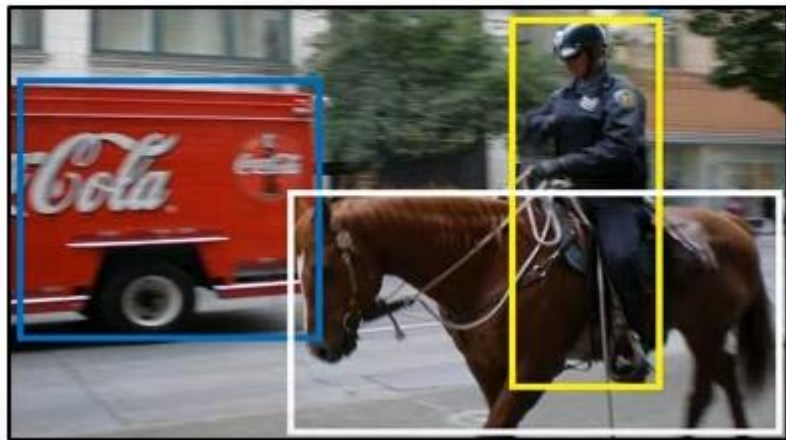
- 문법이 파이썬스러움
- 띄어쓰기로 데이터 구분 cf) JSON (중괄호)
- JSON: 웹에서 데이터 통신, YAML: 복잡한 객체 구조를 표현할 때

```
- name: create users
  hosts: all
  tasks:
    - user:
      name: "{{ item.name }}"
      state: present
      groups: "{{ item.groups }}"
    with_items:
      - { name: 'linda', groups: 'wheel' }
      - { name: 'lisa', groups: 'root' }
```



## Custom Data 예시

이미지 데이터  
image\_data.jpg



정답 데이터  
image\_data.txt

0	0.750000	0.538542	0.178125	0.681250
1	0.686719	0.755208	0.607812	0.477083
2	0.211719	0.552083	0.417187	0.470833

person=0 horse=1 truck=2  
class 종류

- 이미지 데이터와 정답 데이터는 파일 이름이 동일
- YOLO v8 기준, 정답 파일(Annotation File)의 확장자는 반드시 .txt

# YAML 파일의 생성

!pip install PyYAML → 파이썬에서 YAML 파일을 사용하기 위해서 PyYAML 라이브러리 설치

```
import yaml

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

→ YOLOv8 학습과 검증에 사용되는 train, valid data 가 저장되어 있는 디렉토리 경로

→ Detection 하고싶은 클래스 개수 (7개)와 클래스에 대응되는 클래스 이름(names)

```
with open('/content/Aquarium_Data/Aquarium_Data.yaml', 'w') as f:
    yaml.dump(data, f)
```

→ 데이터 경로와 클래스 정보를 저장하고 있는 딕셔너리 객체 data를 YOLOv8 학습에 필요한 Aquarium\_Data.yaml 저장

```
with open('/content/Aquarium_Data/Aquarium_Data.yaml', 'r') as f:
    aquarium_yaml = yaml.safe_load(f)
    display(aquarium_yaml)
```

→ Aquarium\_Data.yaml 읽어서 화면에 출력

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

## ➤ Install YOLOv8

```
!pip install ultralytics
```

```
import ultralytics
```

```
ultralytics.checks()
```

Ultralytics YOLOv8.0.54 🚀 Python=3.9.16 torch=1.13.1+cu116 CUDA:0 (Tesla T4, 15102MiB)  
Setup complete ✅ (2 CPUs, 12.7 GB RAM, 25.6/78.2 GB disk)

## ➤ Load a pre-trained model

```
from ultralytics import YOLO
```

```
model = YOLO('yolov8n.pt') → MS COCO Dataset 사전학습된 yolov8n 모델을 로드함. yolov8n 외에도 yolov8s, yolov8m, yolov8l, yolov8x 등이 있음
```

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, 15.2MB/s]

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

```
print(model.names)
```

→ YOLOv8 은 MS COCO 데이터로 사전학습되어 있기 때문에, MS COCO Dataset 에 정의되어 있는 클래스 개수와 종류는 `model.names` 를 통해서 확인할수 있음 (총 80개, 0~79)

```
<class 'dict'> 80
```

```
{0: 'person', 1: 'bicycle', 2: 'car', 3: 'motorcycle', 4: 'airplane', 5: 'bus', 6: 'train', 7: 'truck', 8: 'boat', 9: 'traffic light', 10: 'fire hydrant', 11: 'stop sign', 12: 'parking meter', 13: 'bench', 14: 'backpack', 15: 'umbrella', 16: 'handbag', 17: 'suitcase', 18: 'frisbee', 19: 'skis', 20: 'snowboard', 21: 'surfboard', 22: 'tennis racket', 23: 'baseball bat', 24: 'baseball glove', 25: 'softball', 26: 'baseball', 27: 'tennis ball', 28: 'bottle', 29: 'wine glass', 30: 'cup', 31: 'fork', 32: 'knife', 33: 'spoon', 34: 'bowl', 35: 'plate', 36: 'chair', 37: 'couch', 38: 'potted plant', 39: 'bed', 40: 'dining table', 41: 'desk', 42: 'toilet', 43: 'sink', 44: 'bathtub', 45: 'showerhead', 46: 'fountain', 47: 'fireplace', 48: 'stove', 49: 'refrigerator', 50: 'oven', 51: 'microwave', 52: 'toaster', 53: 'toaster oven', 54: 'coffee maker', 55: 'blender', 56: 'juicer', 57: 'dishwasher', 58: 'washing machine', 59: 'dryer', 60: 'iron', 61: 'vacuum', 62: 'hair dryer', 63: 'curling iron', 64: 'straightener', 65: 'hairbrush', 66: 'toothbrush', 67: 'toilet paper', 68: 'paper towel', 69: 'napkin', 70: 'paper cup', 71: 'paper plate', 72: 'paper bowl', 73: 'paper bag', 74: 'paper bag', 75: 'paper bag', 76: 'paper bag', 77: 'paper bag', 78: 'paper bag', 79: 'paper bag'}
```

```
model.train(data='/content/Aquarium_Data/Aquarium_Data.yaml', epochs=100, patience=30, batch=32, imgsz=416)
```

100 epochs completed in 0.708 hours

Optimizer stripped from runs/detect/train/weights/last.pt, 6.2MB

Optimizer stripped from runs/detect/train/weights/best.pt, 6.2MB

train, valid 데이터가 저장되어 있는 디렉토리 경로와 클래스 정보등이 설정된 **Aquarium\_Data.yaml**

Validating runs/detect/train/weights/best.pt...

Ultralytics YOLOv8.0.53 Python=3.9.16 torch=1.13.1+cu116 CUDA:0 (Tesla T4, 15102MiB)

Model summary (fused): 168 layers, 3007013 parameters, 0 gradients, 8.1 GFLOPs

Class	Images	Instances	Box(P	R	mAP50	mAP50-95): 100%
all	127	909	0.784	0.668	0.701	0.423
fish	127	459	0.779	0.7	0.771	0.408
jellyfish	127	155	0.876	0.884	0.92	0.503
penguin	127	104	0.623	0.635	0.656	0.294
puffin	127	74	0.67	0.549	0.491	0.235
shark	127	57	0.791	0.614	0.638	0.441
starfish	127	27	0.947	0.658	0.724	0.602
stingray	127	33	0.803	0.636	0.708	0.481

Speed: 0.6ms preprocess, 1.2ms inference, 0.0ms loss, 1.8ms postprocess per image

Results saved to **runs/detect/train** → 커스텀 데이터로 학습(파인튜닝)된 모델, 즉 **best.pt**, **last.pt** 가 저장된 디렉토리

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

```
print(model.names)
```

우리는 **Aquarium\_Data.yaml** 에 기술되어 있는 커스텀 데이터로 학습되었기 때문에, 학습을 마친 후에 **model.names** 값을 보면, 사전학습된 MS COCO 데이터의 80개가 아닌 우리가 YAML 파일에서 설정한 7개의 클래스와 이름으로 바뀌어 있는 것을 알 수 있음

```
<class 'dict'> 7
```

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



```
results = model.predict(source='/content/Aquarium_Data/test/images/', save=True)
```

→ 예측하고 싶은 테스트 데이터가 저장되어 있는 디렉토리

image 1/63 /content/Aquarium\_Data/test/images/IMG\_2289\_jpeg.jpg.rf.fe2a7a149e7b11f2313f5a7b30386e85.jpg: 416x320 1 puffin, 12.7ms

image 2/63 /content/Aquarium\_Data/test/images/IMG\_2301\_jpeg.jpg.rf.2c19ae5efbd1f8611b5578125f001695.jpg: 416x320 15 penguins, 11.4ms

.....

image 63/63 /content/Aquarium\_Data/test/images/IMG\_8599\_MOV-3.jpg.rf.412ebb16ea80e964b4464c50e757df0e.jpg: 416x256 4 jellyfishs, 12.9ms

Speed: 0.3ms preprocess, 11.7ms inference, 1.6ms postprocess per image at shape (1, 3, 416, 416)

Results saved to **runs/detect/predict** → 예측 이미지 저장된 디렉토리

