

# LAB9 tutorial for Machine Learning

---

## Object Detection and Tracking

---

The document description are designed by Jla Yanhong in 2022. Nov. 6th

### 1. Objective

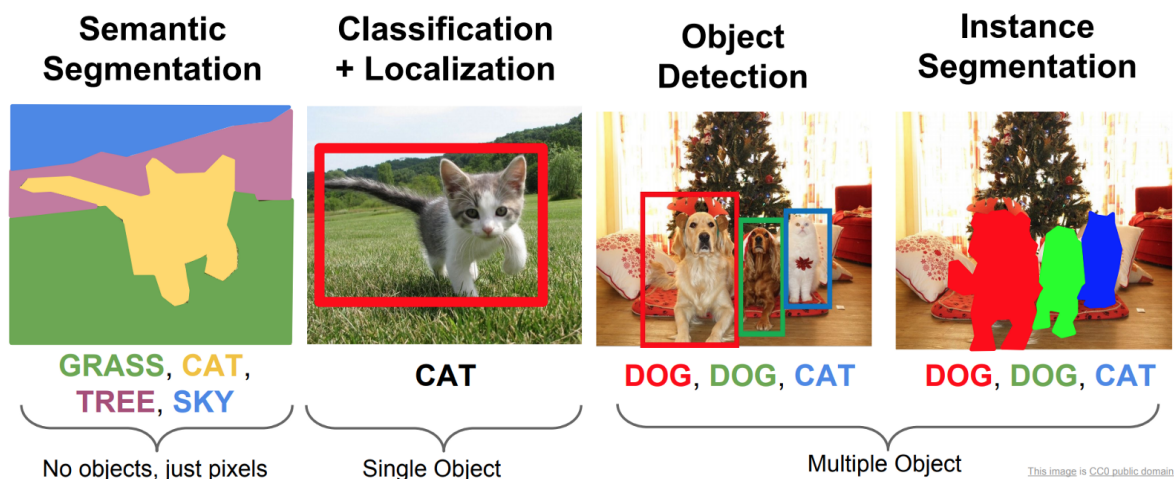
---

- Understanding Object Detection
- Use YOLOv5 for object detection on various images and videos
- Use YOLOv5+DeepSort for object detection and tracking
- Prepare the opening presentation of final project.

### 2. Preface

---

There are a lot many interesting problems in the Image domain.



In this lab, we focus on Object Detection.

Object Detection is a Computer Vision task to detect and localize objects in images and video, it is one of the fundamental problems of computer vision.



Object Detection is used almost everywhere these days. The use cases are endless, be it Tracking objects, Video surveillance, Pedestrian detection, Anomaly detection, People Counting, Self-driving cars or Face detection, the list goes on.

State-of-the-art object detection methods can be categorized into two main types: One-stage vs. two-stage object detectors.

- The two-stage detectors involves:
  - (1) Object region proposal with conventional Computer Vision methods or deep networks, followed by
  - (2) Object classification based on features extracted from the proposed region with bounding-box regression.
- One-stage detectors
 

One-stage detectors predict bounding boxes over the images without the region proposal step. This process consumes less time and can therefore be used in real-time applications.

After 2014 – Two-stage object algorithms

- RCNN and SPPNet (2014)
- Fast RCNN and Faster RCNN (2015)
- Mask R-CNN (2017)
- Pyramid Networks/FPN (2017)
- G-RCNN (2021)

Most important one-stage object detection algorithms

- YOLO (2016)
- SSD (2016)
- RetinaNet (2017)
- YOLOv3 (2018)
- YOLOv4 (2020)
- YOLOv5 (2020)
- YOLOR (2021)
- YOLOv7 (2022)



Two-stage methods achieve the highest detection accuracy but are typically slower. Because of the many inference steps per image, the performance (frames per second) is not as good as one-stage detectors.

One - step detector is used more often in real - time object detection in engineering applications.

In this lab, we will go through the tutorial of YOLOv5 for object detection.

We will understand what is YOLOv5 and show you how to use YOLOv5 for object detection on various images and videos.

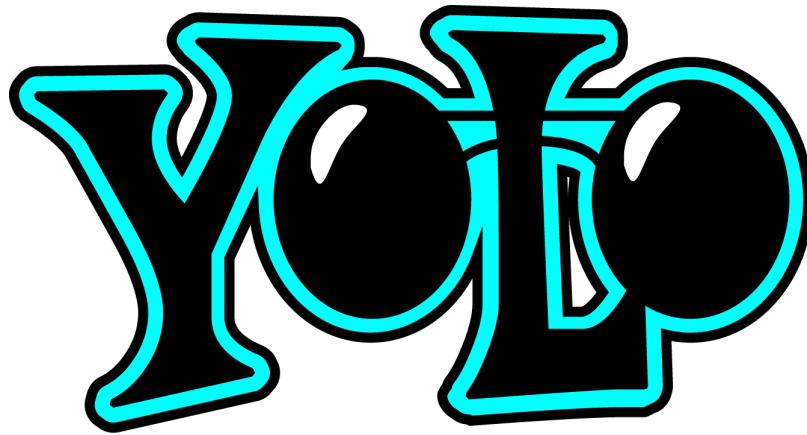
*(But please note that the inclusion of YOLOv5 in the YOLO family is a matter of debate in the community, and neither its paper has been released officially for peer review. So its architectural and performance details mentioned here, as collected from the various sources have to be taken with a pinch of salt.)*

## 3. YOLOv5: Real-Time Object Detection

---

### 3.1 History and Controversy

YOLO stands for You Look Only Once and it is one of the finest family of object detection models with state-of-the-art performances.



Its first model was released in 2016 by Joseph Redmon who went on to publish YOLOv2 (2017) and YOLOv3 (2018). In 2020 Joseph Redmon stepped out from the project citing ethical issues in the computer vision field and his work was further improved by Alexey Bochkovskiy who produced YOLOv4 in 2020.

YOLOv5 is the next controversial member of the YOLO family released in 2020 by the company Ultralytics just a few days after YOLOv4.

### 3.2 Where is YOLOv5 Paper?

YOLOv5 is controversial due to the fact that no paper has been published yet (till the time of writing this) by its author Glenn Jocher for the community to peer review its benchmark. Neither it is seen to have implemented any novel techniques to claim itself as the next version of YOLO. Instead, it is considered as the PyTorch extension of YOLOv3 and a marketing strategy by Ultralytics to ride on the popularity of the YOLO family of object detection models.

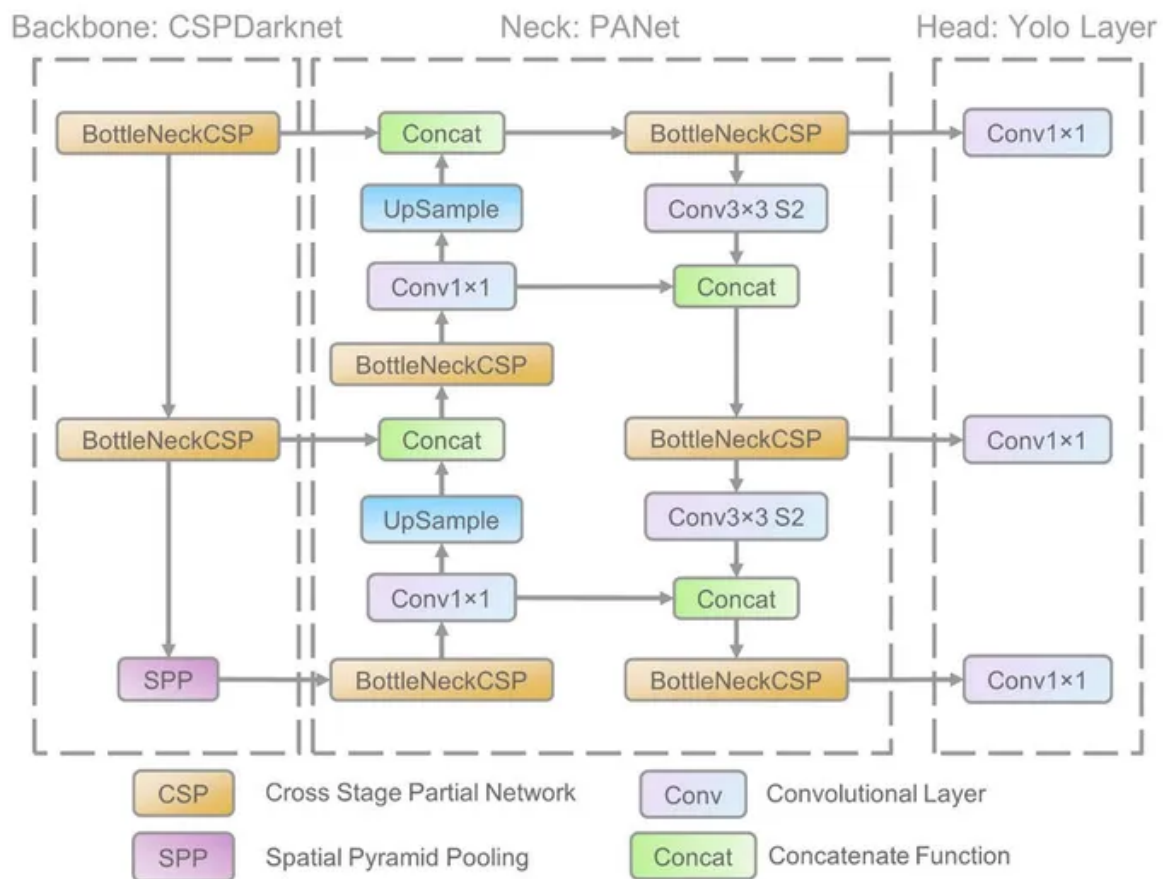
But one should note that when YOLOv3 was created, Glenn Jocher (creator of YOLOv5) contributed to it by providing the implementation of mosaic data augmentation and genetic algorithm.

### 3.3 Is YOLOv5 Good or Bad?

Certainly, the controversy behind YOLOv5 is just due to its choice of name, but it does not take away the fact that this is after all a great YOLO object detection model ported on PyTorch.

Probably if you are just a developer, you would not even care about the controversy and may enjoy working with YOLOv5 due to its ease of use. (As we will see in the examples of this tutorial)

### 3.4 YOLOv5 Architecture



(Source)





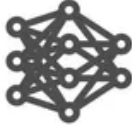
The YOLO family of models consists of three main architectural blocks i) Backbone, ii) Neck and iii) Head.

1. **YOLOv5 Backbone:** It employs CSPDarknet as the backbone for feature extraction from images consisting of cross-stage partial networks.
2. **YOLOv5 Neck:** It uses PANet to generate a feature pyramids network to perform aggregation on the features and pass it to Head for prediction.
3. **YOLOv5 Head:** Layers that generate predictions from the anchor boxes for object detection.

Apart from this YOLOv5 uses the below choices for training –

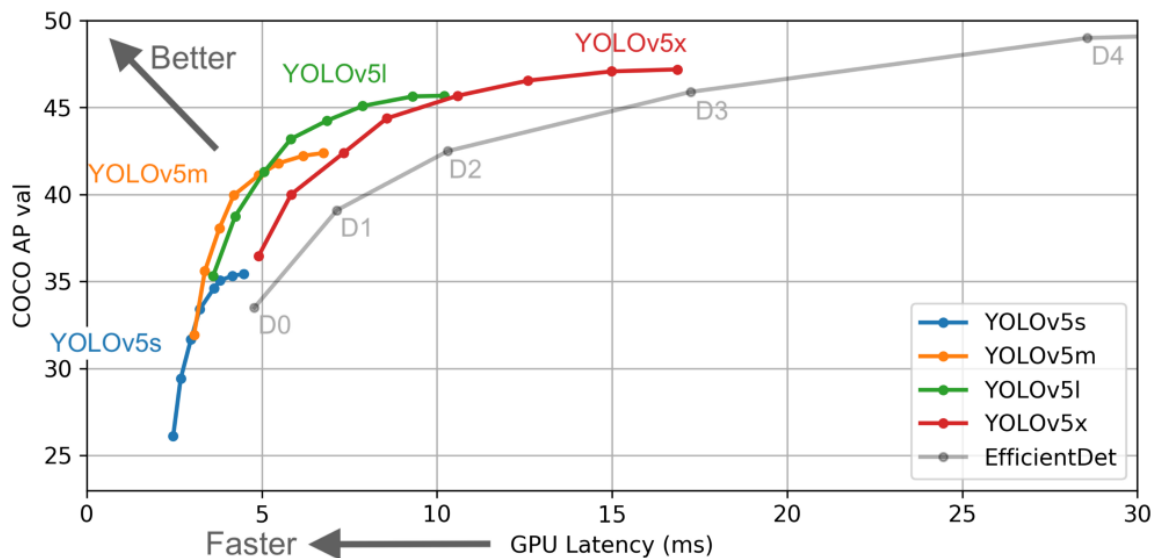
1. **Activation and Optimization:** YOLOv5 uses leaky ReLU and sigmoid activation, and SGD and ADAM as optimizer options.
2. **Loss Function:** It uses Binary cross-entropy with logits loss.

### 3.5 Different Types of YOLOv5

				
Nano YOLOv5n	Small YOLOv5s	Medium YOLOv5m	Large YOLOv5l	XLarge YOLOv5x
4 MB <sub>FP16</sub> 6.3 ms <sub>V100</sub> 28.4 mAP <sub>COCO</sub>	14 MB <sub>FP16</sub> 6.4 ms <sub>V100</sub> 37.2 mAP <sub>COCO</sub>	41 MB <sub>FP16</sub> 8.2 ms <sub>V100</sub> 45.2 mAP <sub>COCO</sub>	89 MB <sub>FP16</sub> 10.1 ms <sub>V100</sub> 48.8 mAP <sub>COCO</sub>	166 MB <sub>FP16</sub> 12.1 ms <sub>V100</sub> 50.7 mAP <sub>COCO</sub>

#### [YOLOv5 Model Comparison](#)

YOLOv5 has multiple varieties of pre-trained models as we can see above. The difference between them is the trade-off between the size of the model and inference time. The lightweight model version YOLOv5s is just 14MB but not very accurate. On the other side of the spectrum, we have YOLOv5x whose size is 168MB but is the most accurate version of its family.



## 4. YOLOv5 Tutorial for Object Detection

In this section, we will see hands-on examples of using YOLOv5 for object detection of both images and videos.

## 4.1 Cloning the YOLOv5 Repository

Clone the YOLOv5 repository made and maintained by Ultralytics.

YOLOv5 github: <https://github.com/ultralytics/yolov5.git>

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

## 4.2 Installing Requirements

```
cd yolov5  
pip install -r requirements.txt # install
```

## 4.3 Train On Custom Data

### 4.4 Create Dataset

YOLOv5 models must be trained on labelled data in order to learn classes of objects in that data. There are two options for creating your dataset before you start training:

#### 4.4.1 Use [Roboflow](#) to label, prepare, and host your custom data automatically in YOLO format

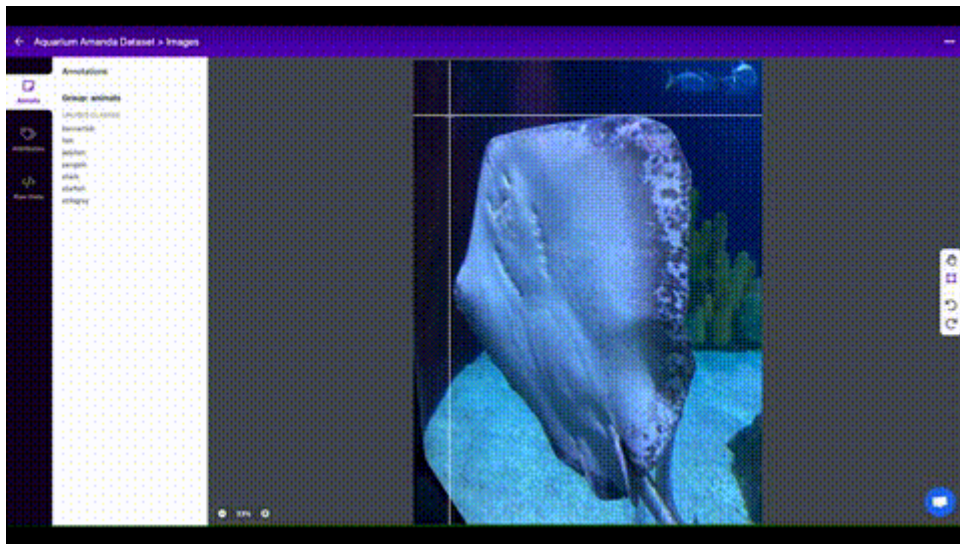
##### 4.4.1.1 Collect Images

Your model will learn by example. Training on images similar to the ones it will see in the wild is of the utmost importance. Ideally, you will collect a wide variety of images from the same configuration (camera, angle, lighting, etc) as you will ultimately deploy your project.

If this is not possible, you can start from [a public dataset](#) to train your initial model and then [sample images from the wild during inference](#) to improve your dataset and model iteratively.

##### 4.4.1.2 Create Labels

Once you have collected images, you will need to annotate the objects of interest to create a ground truth for your model to learn from.



[Roboflow Annotate](#) is a simple web-based tool for managing and labeling your images with your team and exporting them in [YOLOv5's annotation format](#).

#### 4.4.1.3 Prepare Dataset for YOLOv5

##### 4.4.1.3.1 Prepare, Export, and Host Your Dataset with Roboflow

Whether you [label your images with Roboflow](#) or not, you can use it to convert your dataset into YOLO format, create a YOLOv5 YAML configuration file, and host it for importing into your training script.

[Create a free Roboflow account](#) and upload your dataset to a `Public` workspace, label any unannotated images, then generate and export a version of your dataset in `YOLOv5 Pytorch` format.

Note: YOLOv5 does online augmentation during training, so we do not recommend applying any augmentation steps in Roboflow for training with YOLOv5. But we recommend applying the following preprocessing steps:



3

### Preprocessing

Decrease training time and increase performance by applying image transformations to all images in this dataset.

Auto-Orient	Edit	×
Resize Stretch to 640×640	Edit	×
<div><div>+</div>Add Preprocessing Step</div>		

Continue

- **Auto-Orient** - to strip EXIF orientation from your images.
- **Resize (Stretch)** - to the square input size of your model (640x640 is the YOLOv5 default).

Generating a version will give you a point in time snapshot of your dataset so you can always go back and compare your future model training runs against it, even if you add more images or change its configuration later.

## Export

Format

YOLO v5 PyTorch

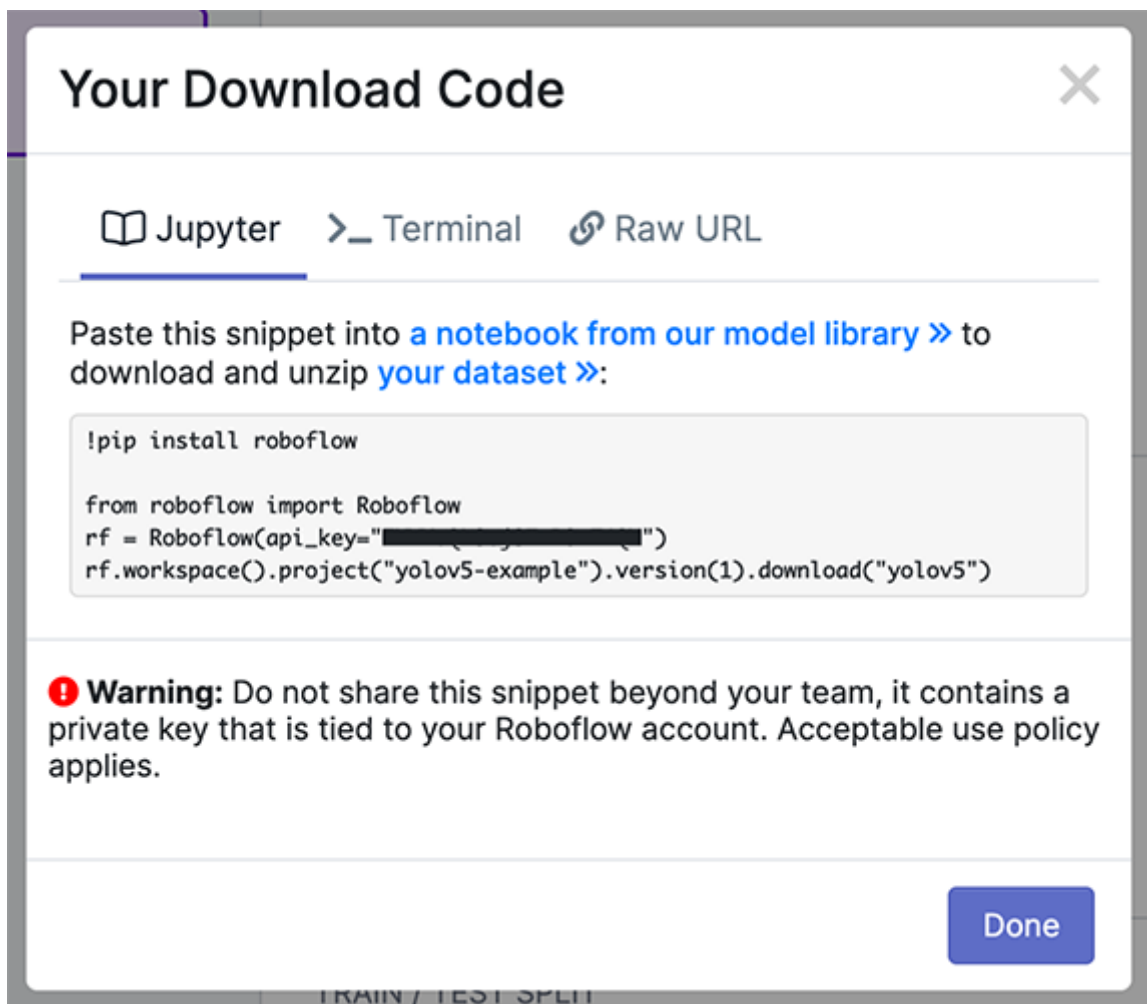
TXT annotations and YAML config used with YOLOv5.

☐ download zip to computer ☒ show download code

Cancel

Continue

Export in `YOLOv5 Pytorch` format, then copy the snippet into your training script or notebook to download your dataset.



Now continue with `2. Select a Model`.

## 4.4.2 Or manually prepare your dataset

### 4.4.2.1 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:

```
# 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

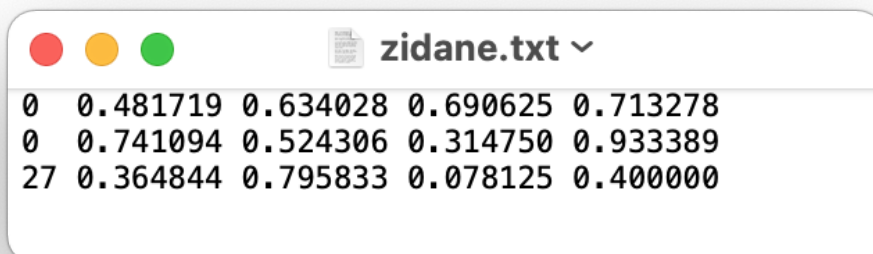
#### 4.4.2.2 Create Labels

After using a tool like [Roboflow Annotate](#) 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).



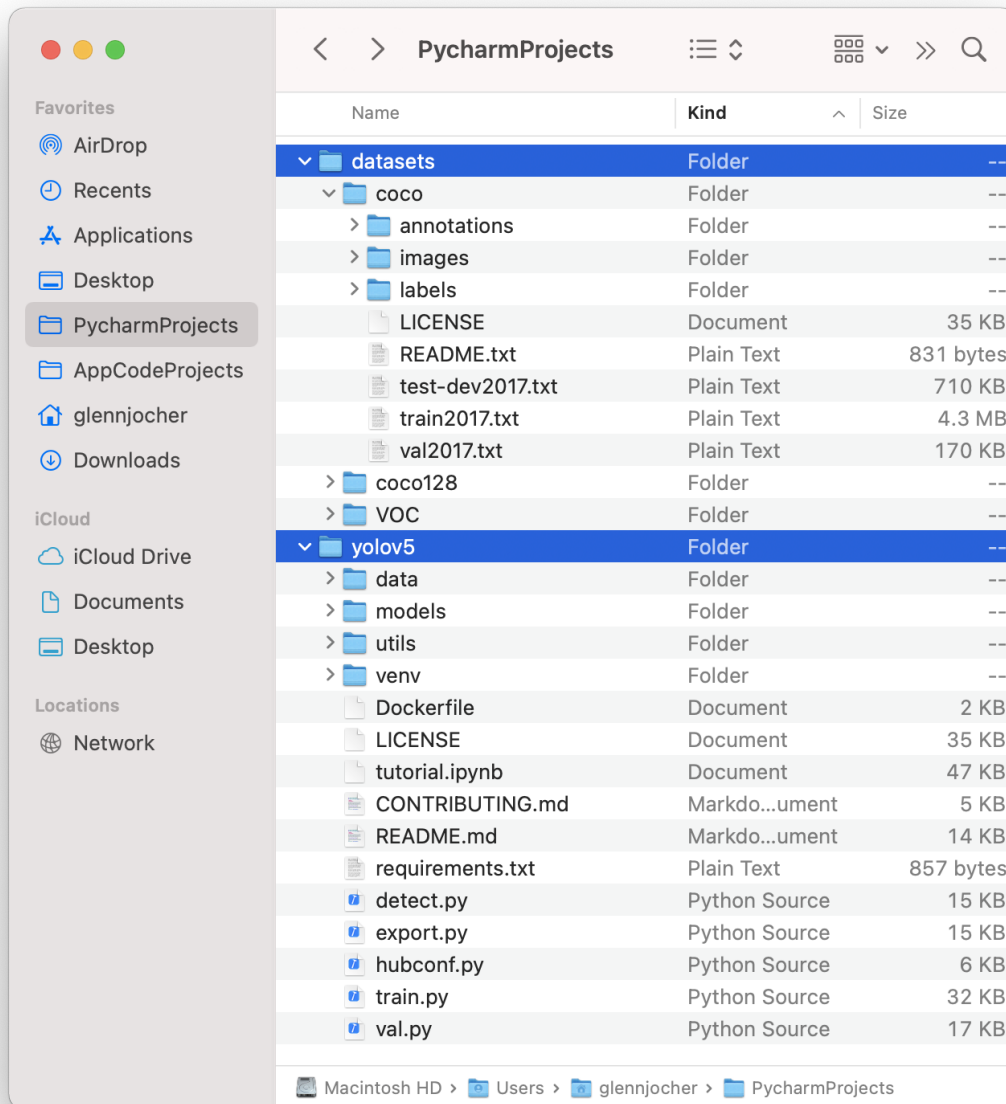
The label file corresponding to the above image contains 2 persons (class 0) and a tie (class 27):



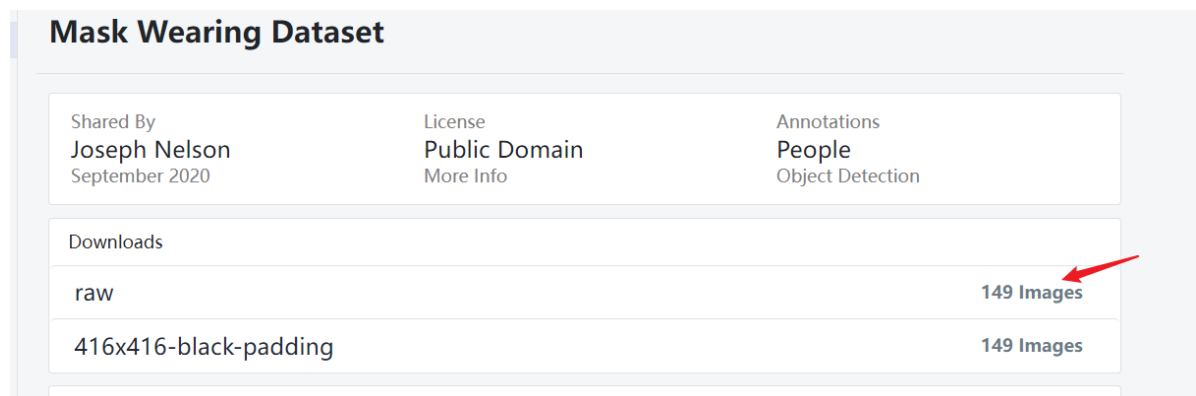
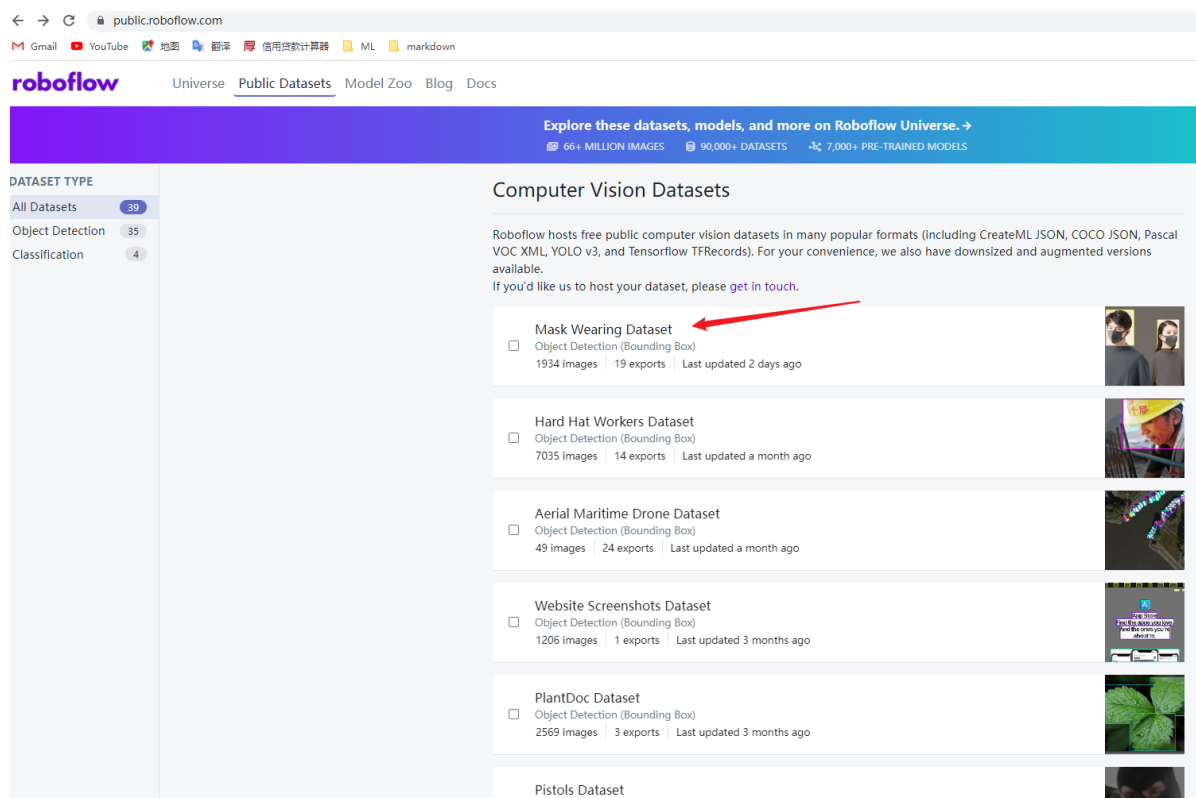
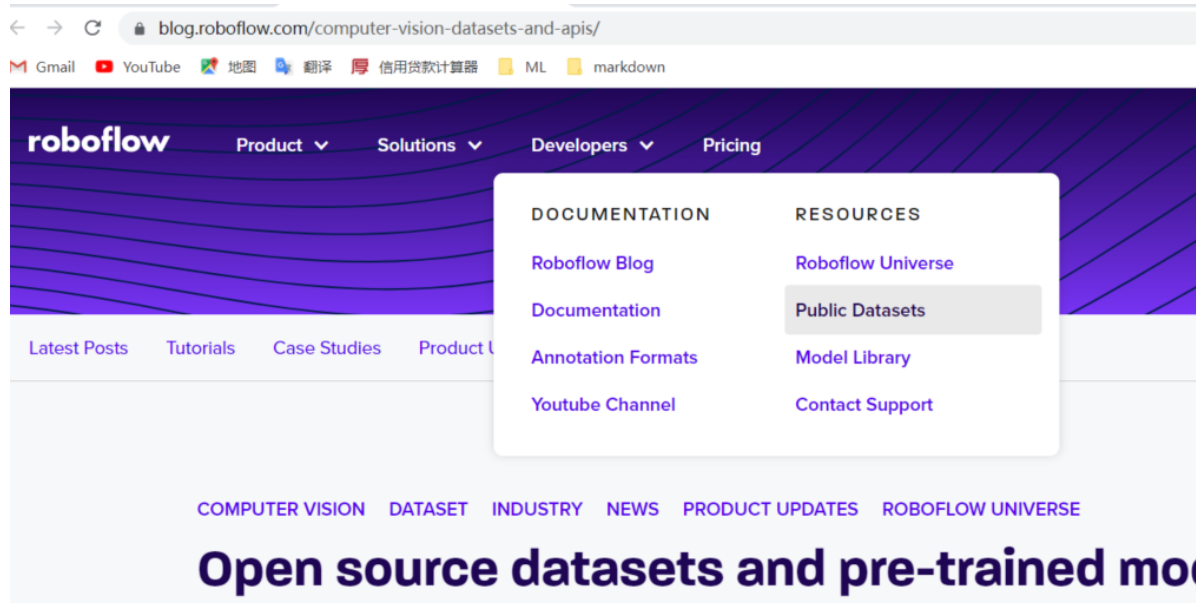
#### 4.4.2.3 Organize Directories

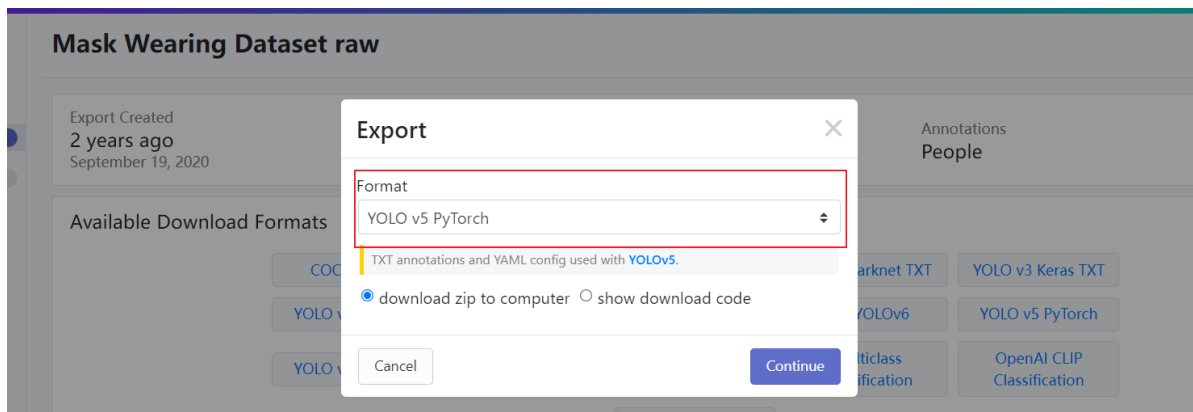
Organize your train and val images and labels according to the example below. YOLOv5 assumes `/coco128` is inside a `/datasets` directory **next to** the `/yolov5` directory. **YOLOv5 locates labels automatically for each image** by replacing the last instance of `/images/` in each image path with `/labels/`.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

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



Here we download a [mask wearing dataset](#) directly from roboflow's public dataset.





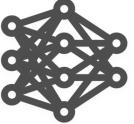




名称	修改日期	类型	大小
test	2022/11/5 15:24	文件夹	
train	2022/11/5 15:50	文件夹	
valid	2022/11/5 16:05	文件夹	
data.yaml	2022/11/5 15:50	Yaml 源文件	1 KB
README.dataset.txt	2020/9/19 11:48	文本文档	2 KB
README.roboflow.txt	2020/9/19 11:48	文本文档	1 KB

## 4.5 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.

				
Nano YOLOv5n	Small YOLOv5s	Medium YOLOv5m	Large YOLOv5l	XLarge YOLOv5x
4 MB <sub>FP16</sub> 6.3 ms <sub>V100</sub> 28.4 mAP <sub>COCO</sub>	14 MB <sub>FP16</sub> 6.4 ms <sub>V100</sub> 37.2 mAP <sub>COCO</sub>	41 MB <sub>FP16</sub> 8.2 ms <sub>V100</sub> 45.2 mAP <sub>COCO</sub>	89 MB <sub>FP16</sub> 10.1 ms <sub>V100</sub> 48.8 mAP <sub>COCO</sub>	166 MB <sub>FP16</sub> 12.1 ms <sub>V100</sub> 50.7 mAP <sub>COCO</sub>

## 4.6 Train

Train a YOLOv5s model on `mask_wearing_dataset` by specifying dataset, batch-size, image size and either pretrained `--weights best.pt`, or randomly initialized `--weights '' --cfg yolov5s.yaml`. Pretrained weights are auto-downloaded from the [latest YOLOv5 release](#).

```
# Train YOLOv5s on mask wearing dataset for 3 epochs
$ python train.py --img 640 --batch 16 --epochs 30 --data
D:/yolo/datasets/MaskDataSet/data.yaml --weights '' --cfg yolov5s.yaml
```



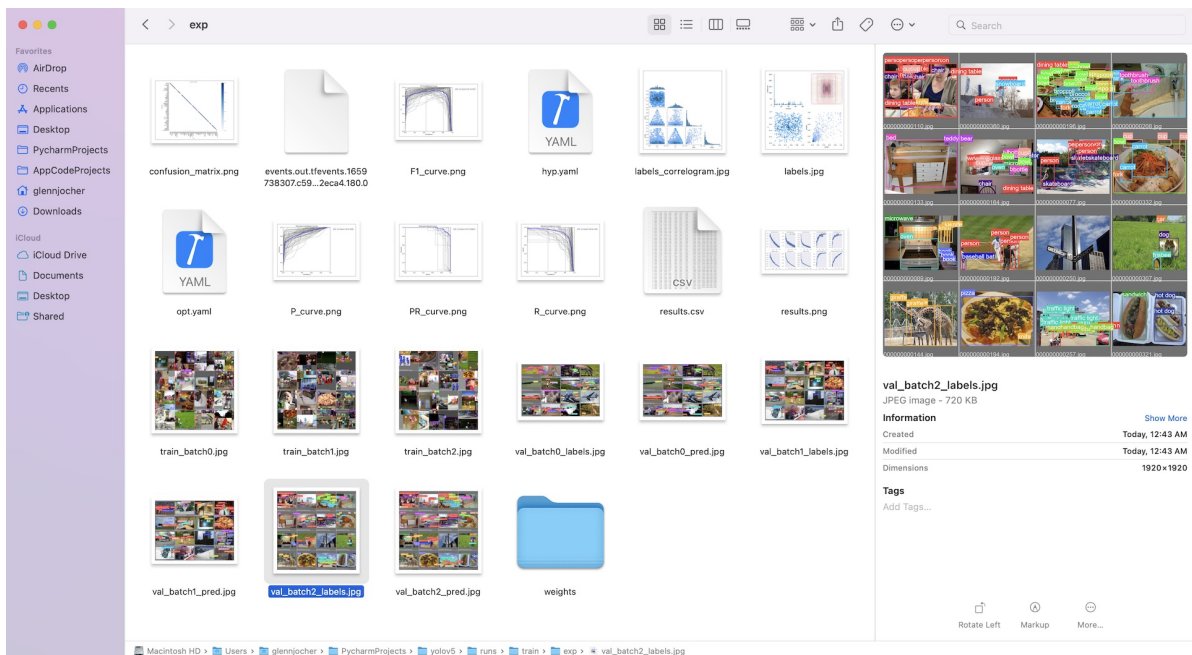
```
$ python train.py --img 640 --batch 16 --epochs 30 --data
D:/yolo/datasets/MaskDataSet/data.yaml --weights
runs\train\exp11\weights\best.pt
```

All training results are saved to `runs/train/` with incrementing run directories, i.e. `runs/train/exp2`, `runs/train/exp3` etc

## 4.7 Visualize (Local Logging)

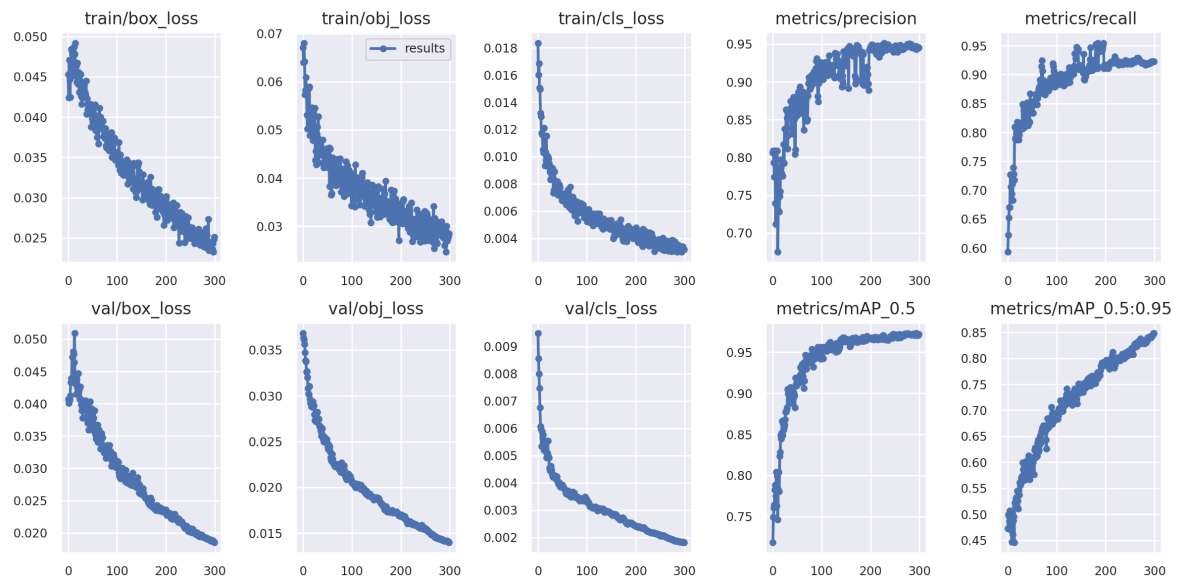
Training results are automatically logged with [Tensorboard](#) and [CSV](#) loggers to `runs/train`, with a new experiment directory created for each new training as `runs/train/exp2`, `runs/train/exp3`, etc.

This directory contains train and val statistics, mosaics, labels, predictions and augmented mosaics, as well as metrics and charts including precision-recall (PR) curves and confusion matrices.



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'
```



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

- Run [Python](#) inference on new images and videos
- [Validate](#) accuracy on train, val and test splits

## 4.8 validation with val.py

Validate a trained YOLOv5 detection model on a detection dataset

```
$ python val.py --weights runs\train\exp11\weights\best.pt --data
D:/yolo/datasets/MaskDataSet/data.yaml --img 640
```

## 4.9 Inference with detect.py

Run YOLOv5 detection inference on images, videos, directories, globs, YouTube, webcam, streams, etc.

`detect.py` runs inference on a variety of sources, downloading [models](#) automatically from the latest YOLOv5 [release](#) and saving results to `runs/detect`.



```
$ python detect.py --weights runs\train\exp11\weights\best.pt --source 0
                        # webcam                                img.jpg
                        # image                                vid.mp4
                        # video                                path/
                        # directory
'path/*.jpg'            # glob
'https://youtu.be/Zgi9g1ksQHc' # YouTube
'rtsp://example.com/media.mp4' #RTSP, RTMP, HTTP stream
```

## 5. Yolov5 + Deep Sort Tutorial for Object Detection and Tracking

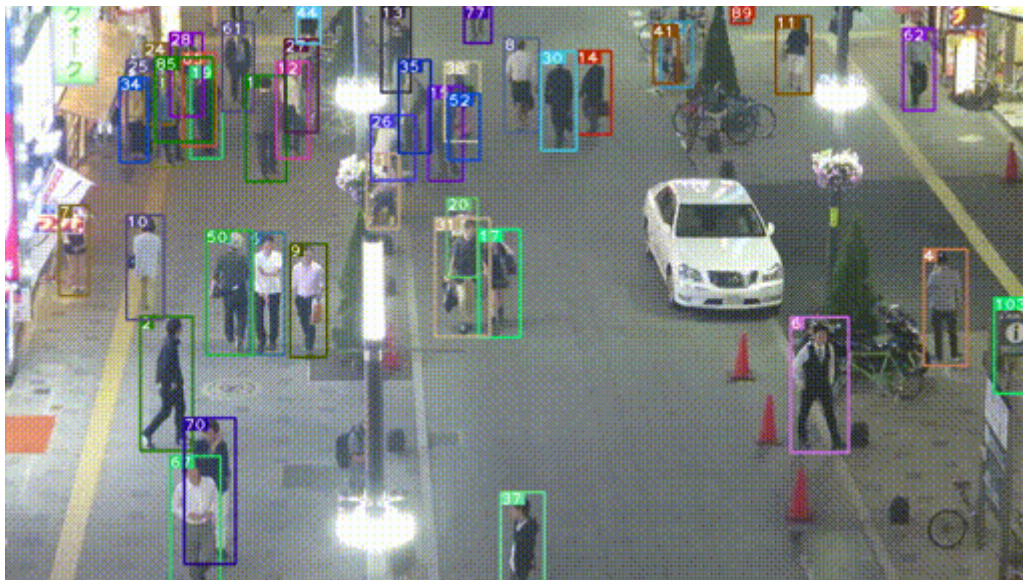
---

### 6. Introduction

---

This repository contains a two-stage-tracker. The detections generated by [YOLOv5](#), a family of object detection architectures and models pretrained on the COCO dataset, are passed to a [Deep Sort algorithm](#) which tracks the objects. It can track any object that your Yolov5 model was trained to detect.





## 7. Before running the tracker

---

### 7.1 Cloning the YOLOv5-Deepsort Repository

yolov5-deepsort github: [https://github.com/HowieMa/DeepSORT\\_YOLOv5\\_Pytorch](https://github.com/HowieMa/DeepSORT_YOLOv5_Pytorch)

```
git clone https://github.com/HowieMa/DeepSORT_YOLOv5_Pytorch.git # clone
```

Make sure that you fulfill all the requirements: Python 3.8 or later with all [requirements.txt](#) dependencies installed, including torch>=1.7. To install, run:

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

Download the yolov5 weight. I already put the `yolo5s.pt` inside. If you need other models, please go to [official site of yolov5](#). and place the downloaded `.pt` file under `yolo5/weights/`. And I also already downloaded the deepsort weights. You can also download it from [here](#), and place `ckpt.t7` file under `deep_sort/deep/checkpoint/`

## 8. Running tracker

---

```
# on video file
python main.py --input_path [VIDEO_FILE_NAME] --display

# on webcam
python main.py --cam 0 --display
```

## 9. Other Tutorials

---

- [Yolov5 training on Custom Data \(link to external repository\)](#)
- [DeepSort deep descriptor training \(link to external repository\)](#)
- [Yolov5 deep\\_sort pytorch evaluation](#)
- [Yolov5\\_DeepSort\\_Pytorch](#)
- [yolov5](#)
- [deep\\_sort\\_pytorch](#)
- [deep\\_sort](#)

## 10. Reference

---

- [1] Joseph Redmon, et al. "You only look once: Unified, real-time object detection." CVPR 2016.
- [2] Joseph Redmon and Ali Farhadi. "YOLO9000: Better, Faster, Stronger." CVPR 2017.
- [3] Joseph Redmon, Ali Farhadi. "YOLOv3: An incremental improvement."
- [4] Lilian Weng. Object Detection Part 4: Fast Detection Models Dec 27, 2018