# VEHICLE DETECTION USING YOLO V7

# APPENDIX

# Introduction

Our teams project is vehicle detection using yolo v7 before going to yolo first of all let us see what is object detection .object detection is advanced form of image classification where the neural network predicts the objects in an image and point out them in the form of bounding boxes.

Object detection hence refers to the detection and localization of objects in an image that belong to a predefined set of classes.Tasks like detection, recognition, or localization are widely used in real-world scenarios, making object detection a very important subdomain of computer vision .

There are mainly two stages of object detection

Two-stage object detection mainly refers to the use of algorithms that divides the object detection problem statement into the following two-stages:

- Detecting possible object regions.
- Classifying the image in those regions into object classes There are mainly two steps which are Fast-RCNN and Faster-RCNN typically use a Region Proposal Network that gives regions of interest that might contain object.
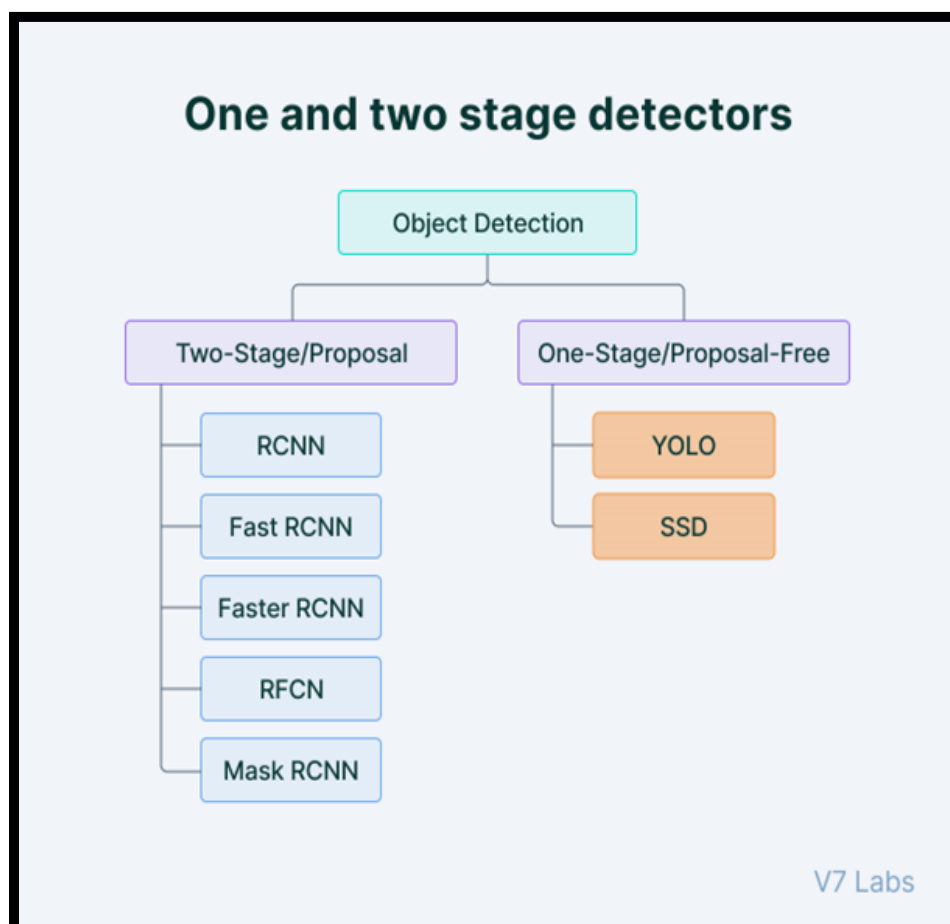
# YOLO - You Only Look Once

Now let us see what yolo is. When compared with approaches taken by object detection algorithms before YOLO, which repurpose classifiers to perform detection, YOLO proposes the use of an end-to-end neural network which makes predictions of bounding boxes and class probabilities all at once. YOLO achieves state-of-the-art results that far exceed other real-time object detection algorithms.

## YOLO compared to other detectors:

In addition to improved prediction accuracy and better intersection than bounding box union, YOLO has an inherent speed advantage. YOLO is a much faster algorithm than its counterparts, running at up to 45 FPS.:Lets study the Deep Architechture of Yolo v7.

## Intersection Over Union (IOU):

IOU is a common metric for measuring localization accuracy and calculating localization error for object detection models.

Compute IOU:
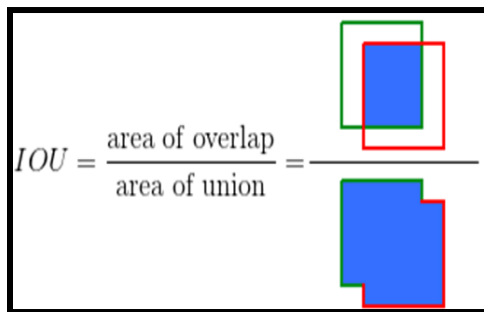


First, get the intersection of the bounding box of a given prediction with the ground truth bounding box of the same range.Computes the total area covered by two bounding boxes, also called union.Dividing the intersection by the sum gives the ratio of the overlap to the total area and gives an accurate estimate of how close the bounding box is to the original prediction.

# How does YOLO work?

The YOLO algorithm works by dividing the image into $N$ grids, each having an equal dimensional region of SxS. Each of these $N$ grids is responsible for the detection and localization of the object it contains.

Correspondingly, these grids predict B bounding box coordinates relative to their cell coordinates, along with the object label and probability of the object being present in the cell.

This process greatly lowers the computation as both detection and recognition are handled by cells from the image, but—

It brings forth a lot of duplicate predictions due to multiple cells predicting the same object with different bounding box predictions.

YOLO makes use of Non Maximal Suppression to deal with this issue.
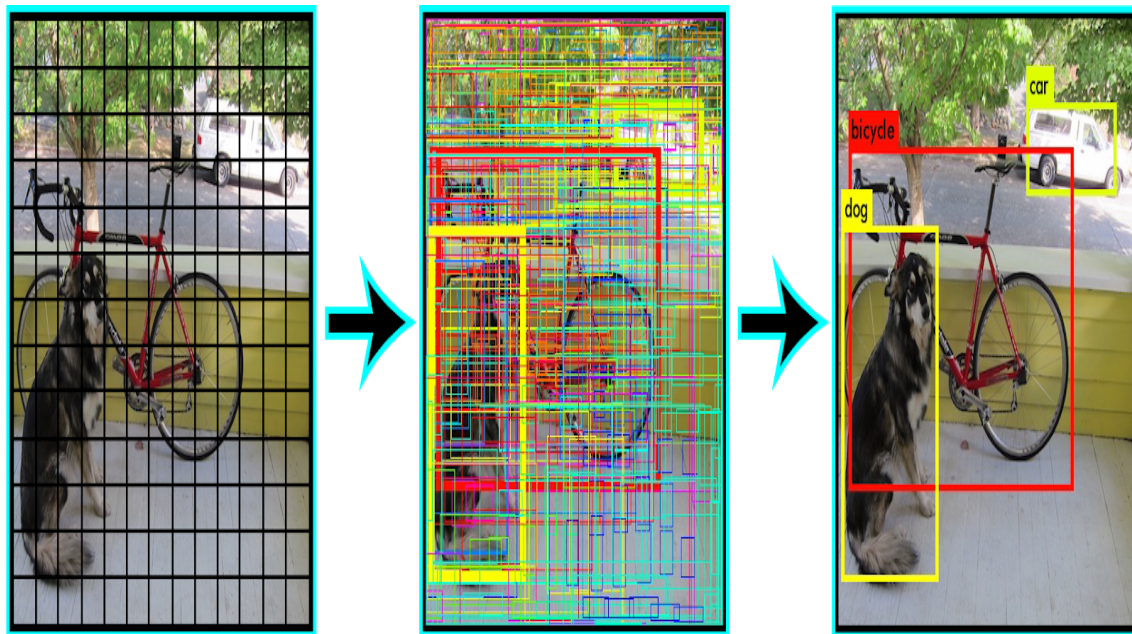


In Non Maximal Suppression, YOLO suppresses all bounding boxes that have lower probability scores.

YOLO achieves this by first looking at the probability scores associated with each decision and taking the largest one. Following this, it suppresses the bounding boxes having the largest Intersection over Union with the current high probability bounding box.This step is repeated till the final bounding boxes are obtained.

# YOLO ARCHITECTURE

Inspired by the GoogleNet architecture, YOLO's architecture has a total of 24 convolutional layers with 2 fully connected layers at the end.
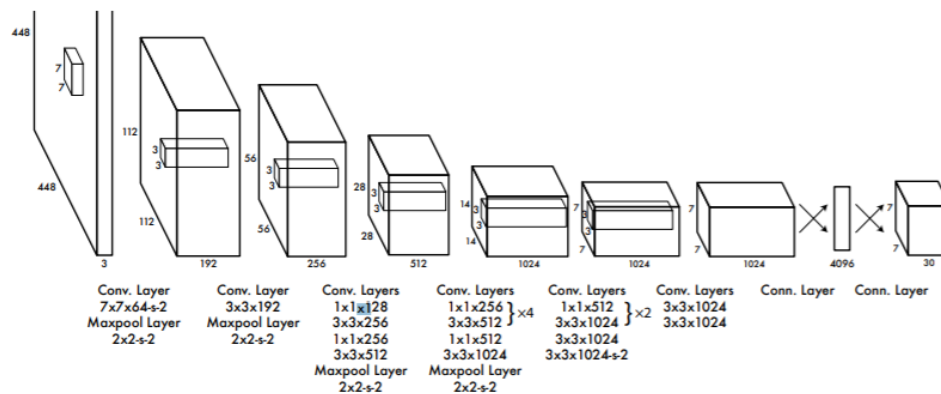


**Figure 3: The Architecture.** Our detection network has 24 convolutional layers followed by 2 fully connected layers. Alternating $1 \times 1$ convolutional layers reduce the features space from preceding layers. We pretrain the convolutional layers on the ImageNet classification task at half the resolution ($224 \times 224$ input image) and then double the resolution for detection.

# What is YOLOv7?

The YOLO v7 model was authored by WongKinYiu and Alexey Bochkovskiy (AlexeyAB).

The YOLO (You Only Look Once) v7 model is the latest in the family of YOLO models. YOLO models are single stage object detectors. In a YOLO model, image frames are featurized through a backbone. These features are combined and mixed in the neck, and then they are passed along to the head of the network YOLO predicts the locations and classes of objects around which bounding boxes should be drawn.

YOLO conducts a post-processing via non-maximum supression (NMS) to arrive at its final prediction.
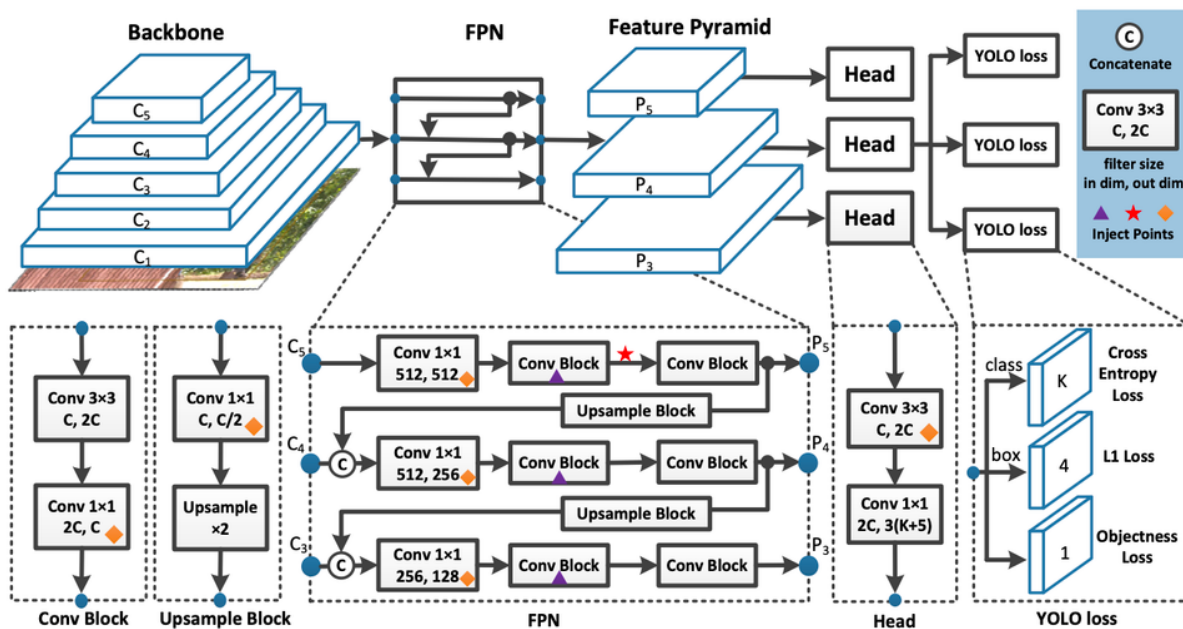


YOLO network architecture as depicted in PP-YOLO

In the beginning, [YOLO models](#) were used widely by the computer vision and machine learning communities for modeling object detection because they were small, nimble, and trainable on a single GPU. This is the opposite of the giant transformer architectures coming out of the leading labs in big tech which, while effective, are more difficult to run on consumer hardware.
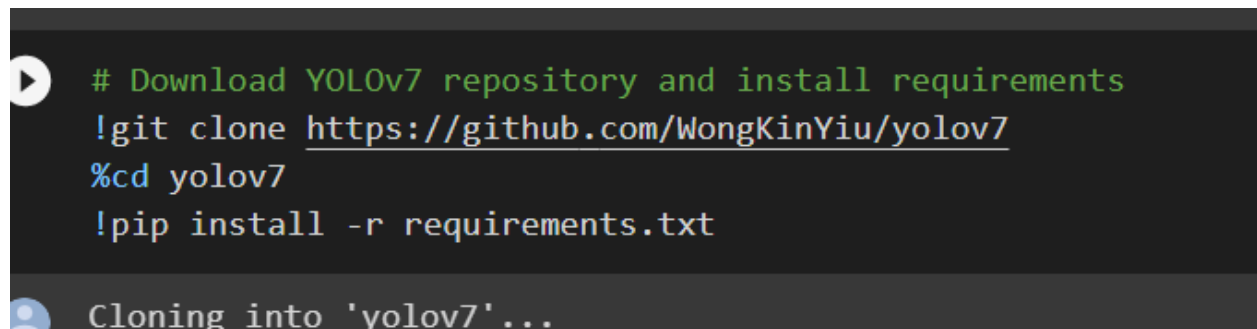
Since their introduction in 2015, YOLO models have continued to proliferate in the industry. The small architecture allows new ML engineers to get up to speed quickly when learning about YOLO and the realtime inference speed allows practitioners to allocate minimal hardware compute to power their applications.



YOLOv7 detecting horses with bounding box predictions

# Vehicle Detection by our Group

## Installing Dependencies



```
# Download YOLOv7 repository and install requirements
!git clone https://github.com/WongKinYiu/yolov7
%cd yolov7
!pip install -r requirements.txt

Cloning into 'yolov7'...
```

We are cloning official git repo of YOLO v7 by author WangKinyiu by above code. And alter the runtime of colab by having GPU accelerator so that the epochs will run faster.

## Getting Dataset Ready

I started after brainstorming myself with YOLO v7 models and analyzing some research papers and official blogs from roboflow and visiting the official git repository of author Wongkinyiu.

As all versions YOLO are implemented by adding new layers or updating weights of previous versions. (in simple reality the complexity differs largely).

Coming to the project as i was unable to download the UA-DETRAC dataset as it was throwing some error.And i actually got the dataset and the dataset should be split into yolo v7 format to run yolov7 model on that i.e in below hierarchy.

- —yolov7

- ○ Dataset
    - ■ Test
        - ● Images
        - ● labels
    - ■ Train
        - ● Images
        - ● labels
    - ■ Valid
        - ● images
    - ■ Roboflow text file

We should have the hierarchy in the above format. We can get UA-DETRAC using os and label libraries in python.Basically you have to take images and label them with particular objects.And the file in the Colab notebook environment space.BY clicking upload button as shown in figure.
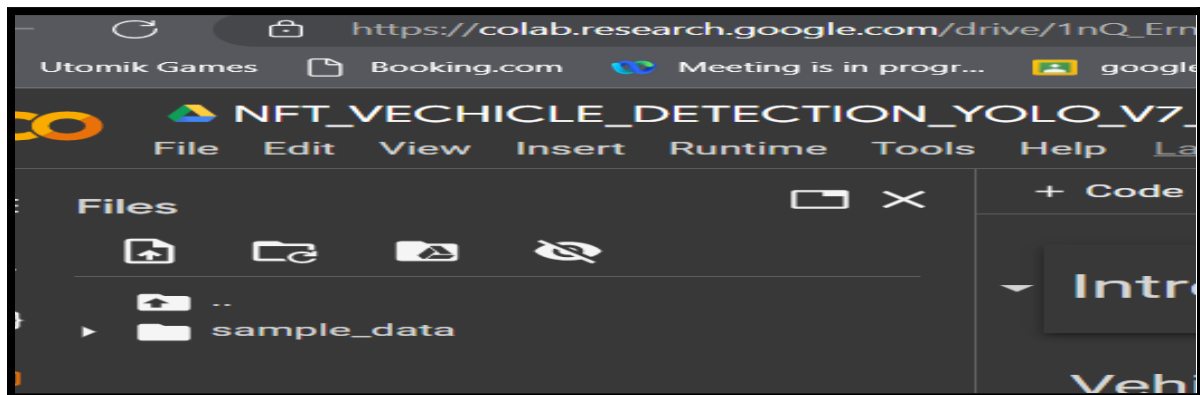


# Best Method for Datset Collection for yolo v7

Unlike the traditional method of uploading data and losing it in runtime you can use Roboflow apis to manage the data.
In the project we are going to use API method.IN roboflow you can use inbuilt dataset apis.The vehicle-type is a custom dataset that was created and by gathering images and was converted into Api.Method to do so is to visit Roboflow Website and creating Dataset and Download Dataset in Necessary Format .In this case its Yolo v7 Pytorch Format.Import the generated Code and Run it you can see a Dataset folder created in the yolo v7 folder of Hierarchy with train valid and test folders .(If not there you use valid folder images to train and test the model.)

**Code the corresponds to get dataset**

```
!pip install roboflow

from roboflow import Roboflow
rf = Roboflow(api_key="qyqO11THVXDf6E7Soqpq")
project = rf.workspace("nftdataset").project("nftdataset")
dataset = project.version(1).download("yolov7")
```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/

# Training YOLO v7 Model

To train yolo v7 model you can use the below part code

```
# run this cell to begin training

%cd /content/yolov7

!python train.py --batch 16 --epochs 5 --data /content/yolov7/vehicle-type.3-4/data.yaml --weights 'yolov7_training.pt' --device 0
```

`--batch-size`, total batch size for all GPUs (default value: `16`) GPU is typically a hardware accelerator and it should be preferable set in GOOgle colab by changing the run time.

`--epochs`, the number of times to cycle through your training data (default value: `300`) More no of epochs produce better accuracy rates.

`--weights`, initial weights path (default value: `'yolo7.pt')`

`--device`, cuda device, i.e. 0 or 0,1,2,3 or cpu (default value: `' '`)

# Testing the YOLO v7 Model

```
[ ]  # Run evaluation
     !python detect.py --weights runs/train/exp/weights/best.pt --conf 0.1 --source /content/yolov7/nftdataset-1/test/images
```

Yolo V7 Here the yolo v7 model is used to test on given data set.(although i mentioned it for train data it can be altered and at the evaluation the data given will used for testing only for display purposes, train is used again.)

To Display the Test Results use below code

## To Display the Outcomes of the Trained model

```
#display inference on ALL test images

import glob
from IPython.display import Image, display

i = 0
limit = 10 # max images to print
for imageName in glob.glob('/content/yolov7/runs/detect/exp/*.jpg'): #assuming JPG
    if i < limit:
      display(Image(filename=imageName))
      print("\n")
    i = i + 1
```

# What if Outcomes are not Efficient and Accurate:

● Use a Dataset which is cleanly labeled and has variety of pictures.
● Use Epochs Which are sufficient enough for the model to get good weights from given dataset so that it won't overfit or under fit on the YOLO v7 Model.(Preferably around 300 the default value)
● Training part plays major role in the accuracy of the model make sure everything is perfect there.

# Conclusion

YOLO provided a super fast and accurate object detection algorithm that revolutionized computer vision research related to object detection.

With over 5 versions (3 official) and cited more than 16 thousand times, YOLO has evolved tremendously ever since it was first proposed in 2015.

YOLO has large-scale applicability with thousands of use cases, particularly for autonomous driving, vehicle detection, and intelligent video analytics.