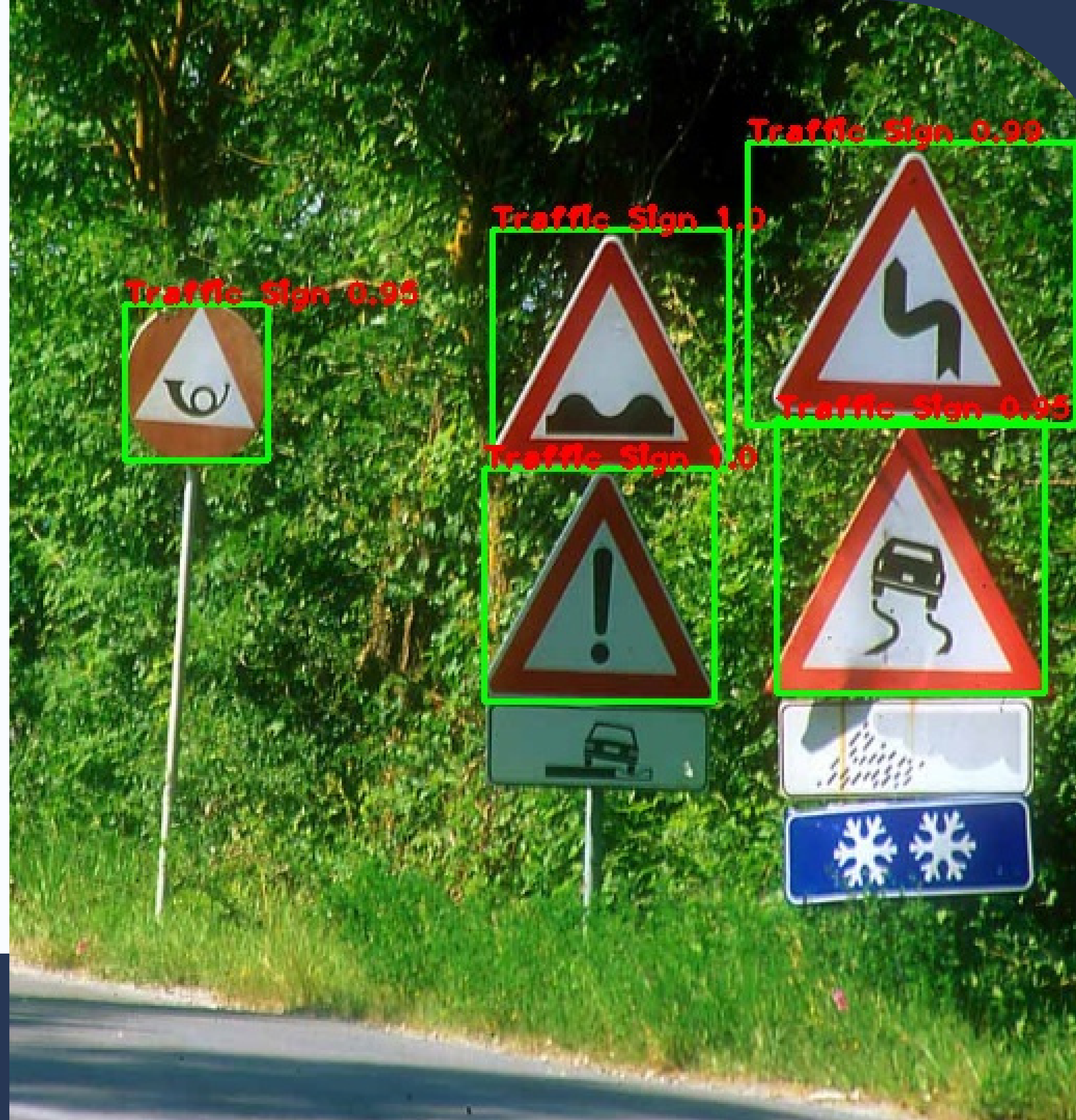


COGNITIVO CONSULTING

STREET SIGN DETECTION

Arpita Kumari



INTRODUCTION

Traffic signs are an integral part of our road infrastructure.

Naturally, autonomous vehicles must also abide by road legislation and therefore recognize and understand traffic signs.

Keeping this in mind, this project is created which is aimed at detecting the traffic signs accurately in the streets using Computer Vision technology. It will be trained on some images using a pre trained model framework. The custom trained weights are then used to detect street signs on images.

STEP 1

SELECTING A OBJECT DETECTION MODEL ARCHITECTURE

Object detection is a computer vision technique whose purpose is to detect and identify various objects like car, table, buildings, human beings, etc. from either images or video stream. This technique locates the presence of an object in an image and draws a bounding box around that object. This usually involves two processes; classifying and object's type, and then drawing a box around that object.

SOME OF THE COMMON OBJECT DETECTION MODEL ARCHITECTURE

- R-CNN
- MASK-RCNN
- FAST RCNN
- FASTER RCNN
- SSD
- YOLO

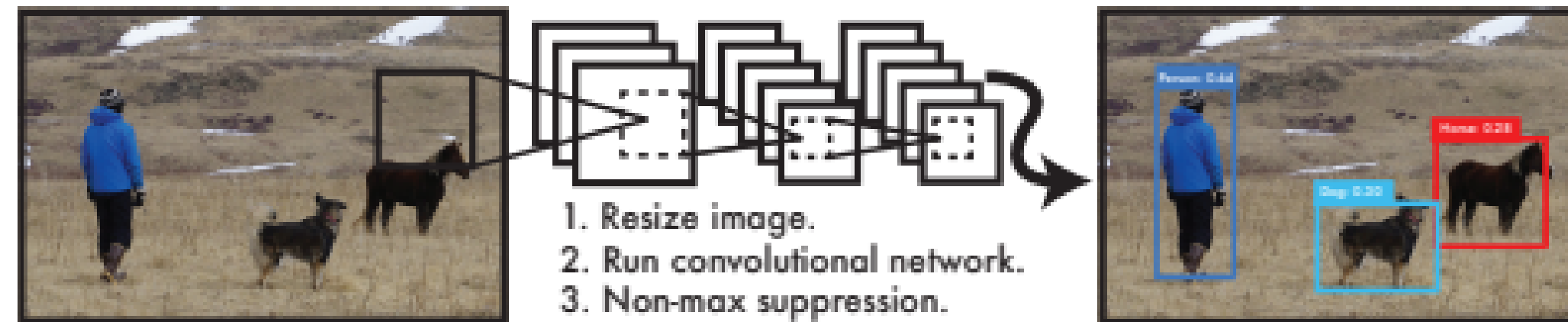


Figure 1: The YOLO Detection System. Processing images with YOLO is simple and straightforward. Our system (1) resizes the input image to 448×448 , (2) runs a single convolutional network on the image, and (3) thresholds the resulting detections by the model's confidence.

YOLO ALGORITHM IN A NUTSHELL

Out of all the common object detection algorithm, I chose YOLO(You Only Look Once) algorithm because of its speed of the detection (45 frames per second). Apart from that YOLO algorithm works best in real-time scenarios.

USEFUL LINKS FOR STUDYING YOLO ALGORITHM

- [Understanding YOLO](#)
- [Understanding YOLO\(.part 2\)](#)
- [Introduction to YOLOv4](#)
- [Introduction to YOLOv4\(.part 2\)](#)
- [YouTube Tutorial for YOLOv4](#)



STEP 2 :

DATASET COLLECTION & PREPARATION

For this project, I used a part of the German Traffic Sign Recognition Benchmark Dataset which is available for free. That means my model works pretty well on German Street Sign. My dataset contains near about 900 images with properly annotated text files. The annotated files are a necessary condition for YOLO model architecture.

The format of the annotated text file is,
<object-class> <x> <y> <width> <height>

Since here I am training the model just on a single class i.e., Traffic Sign, the object-class will be 0. This datasets are then saved in google drive so that we can use it for training purpose in google colab.

STEP 3 :

SETTING UP THE ENVIRONMENT

YOLOv4 model architecture works 100 times faster when trained on GPU enabled systems. Google colab provides free GPU in the cloud and that makes the training process less tiresome.

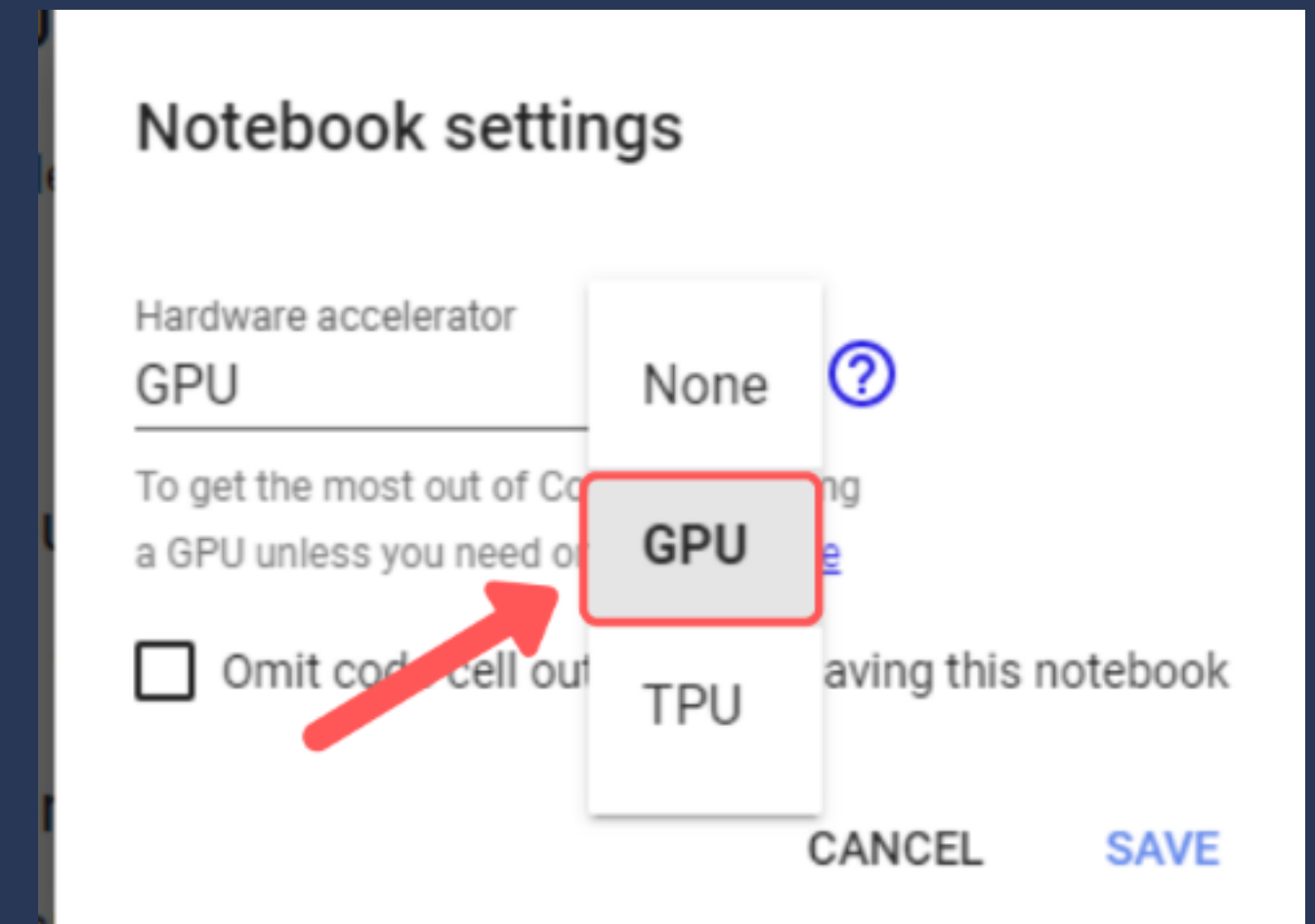
To do so we need to link our google colab with our cloud storage i.e., google drive.

[Tutorial for getting started with Google Colab](#)



NECESSARY STEPS TO SETUP THE ENVIRONMENT

- Click edit in the top left of the notebook and select notebook settings from the dropdown.
- Under 'Hardware Accelerator' select GPU and then hit Save.
- The next step is to mount the google drive with our notebook.



The environment is ready for use !

```
[ ] # mounting the google drive
%cd ..
from google.colab import drive
drive.mount('/content/gdrive')
```


STEP 4 :

TRAINING THE MODEL

The first step in training the model is to clone the open source darknet repository on github. It contains all executable files required for training the file.

The github repository link is : [Darknet](#)

Our dataset is divided into training set and test set. I have used 10% of my images as my test set. The reason for such a selection was my less number of images being used for training.

The next thing to do is to prepare 4 files, required for the training.

- train.txt - It holds the relative path of our training set.
- test.txt - It holds the relative path of our test set.
- obj.names - It contains the class name, in my case there was only 1 class name, i.e., Traffic Sign.
- obj.data - It contains multiple information like, number of classes, path of train.txt, path of test.txt, path of obj.names, path of our backup folder.

Our backup folder will save the weights file generated when the training gets finished.

CFG FILE :

We need to edit our cfg file as per our object detector model.

I set my batches = 64 and subdivisions = 16. I set my max_batches = 6000, steps = 4800, 5400.

I changed the classes = 1 in the three YOLO layers and filters = 18 in the three convolutional layers before the YOLO layers.

The formula for calculating the number of filter is -

$$\text{no. of filters} = (\text{no. of classes} + 5) * 3$$

(So in my case no of classes is 1, hence the number of filters is 18)

The formula for calculating the max_batches is -

max_batches = (# of classes) * 2000 (but no less than 6000 so if you are training for 1, 2, or 3 classes it will be 6000, however detector for 5 classes would have max_batches=10000)

The formula for calculating the steps is -

steps = (80% of max_batches), (90% of max_batches) (so if your max_batches = 10000, then steps = 8000, 9000)

After editing the cfg file, the next step is to download the pre-trained weights for the convolution layers. The link to download the weights is - [YOLOv4 Weight](#)
Using these weights make the training process faster.

Finally, we will train our model. The command to do so is

```
!./darknet detector train <path to obj.data> <path to custom config> yolov4.conv.137 -dont_show -map
```

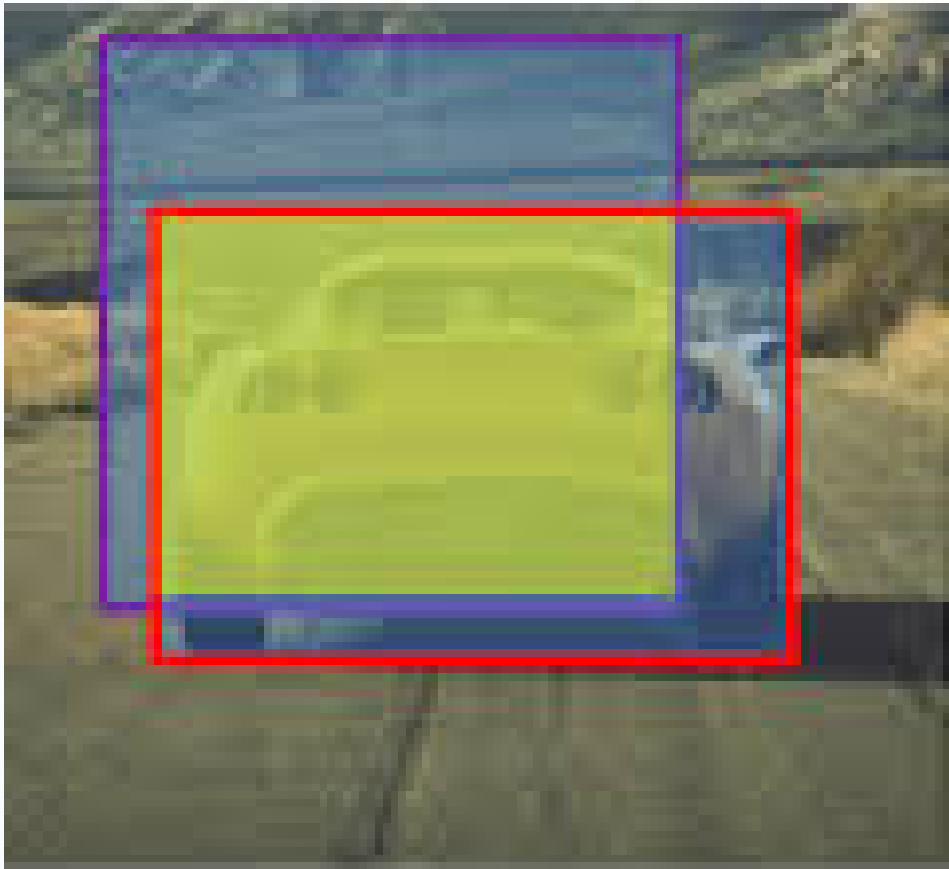
The purpose of -dont_show and -map flag is :

{-dont_show flag stops chart from popping up since Colab Notebook can't open images on the spot.
-map flag overlays mean average precision on chart to see how accuracy of your model is.}

For this model, 2 accuracy metrics are used. One is mAP(Mean Average Precision) and the other one is IoU(Intersection over Union). The command to check mAP is :

```
!./darknet detector map <path to obj.data> <path to custom config> <path to custom trained weights>
```

It will run for 6000 iterations and the final weights will be saved in our backup folder in our google drive.
The accuracy metric we have used is the IoU(Intersection over Union).



Intersection over union (IoU)

$$= \frac{\text{size of } \text{[yellow hatched box]}}{\text{size of } \text{[blue hatched box]}}$$

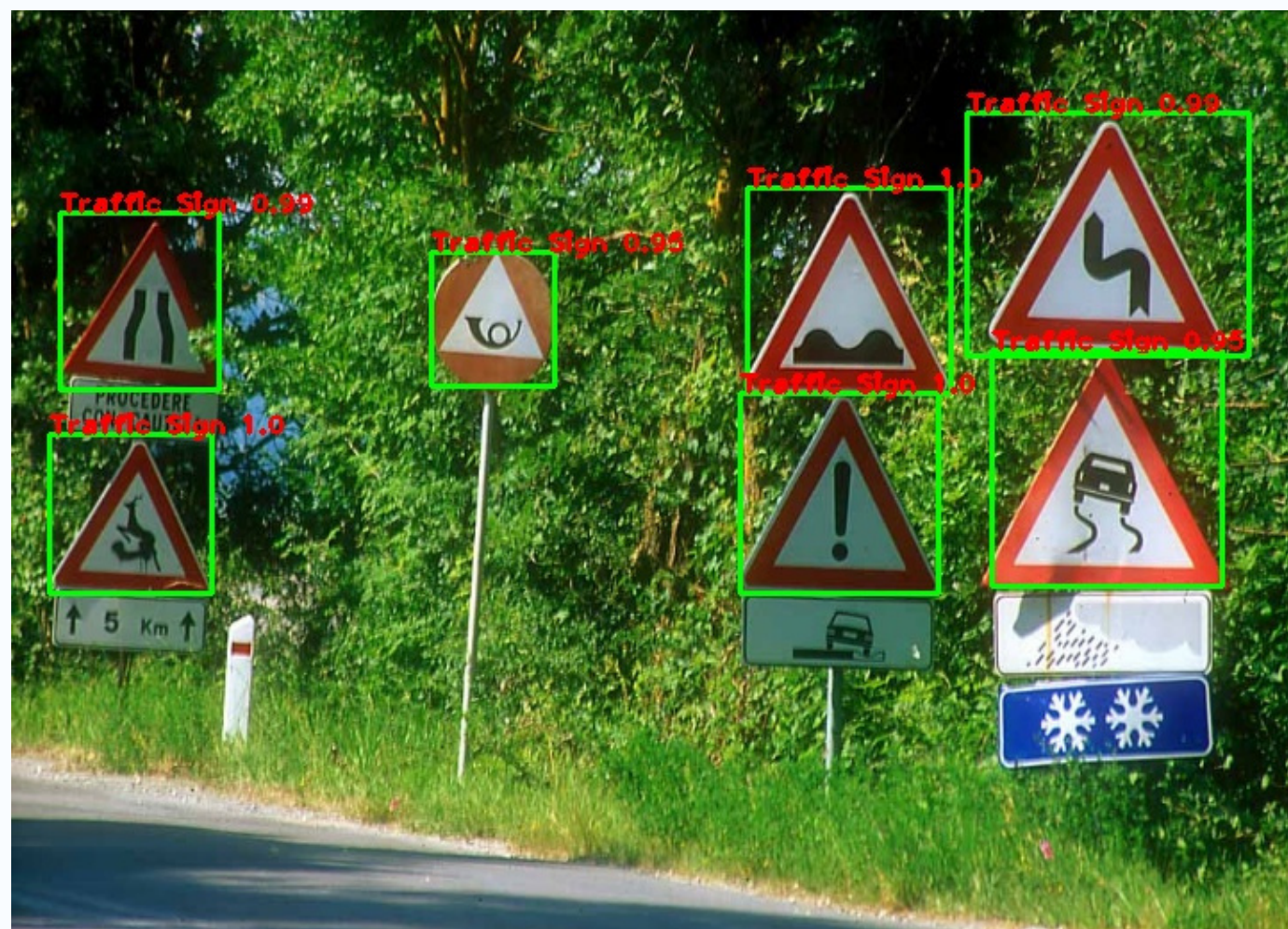
“Correct” if $\text{IoU} \geq 0,5$

After the training process gets over, the weights will be saved in backup folder.
Now we can test our training !

STEP 5 : TESTING THE MODEL

The weights achieved from the training process can now be used to test the model made. Now a simple python code will do the job.

Some of the predictions made by my model are,

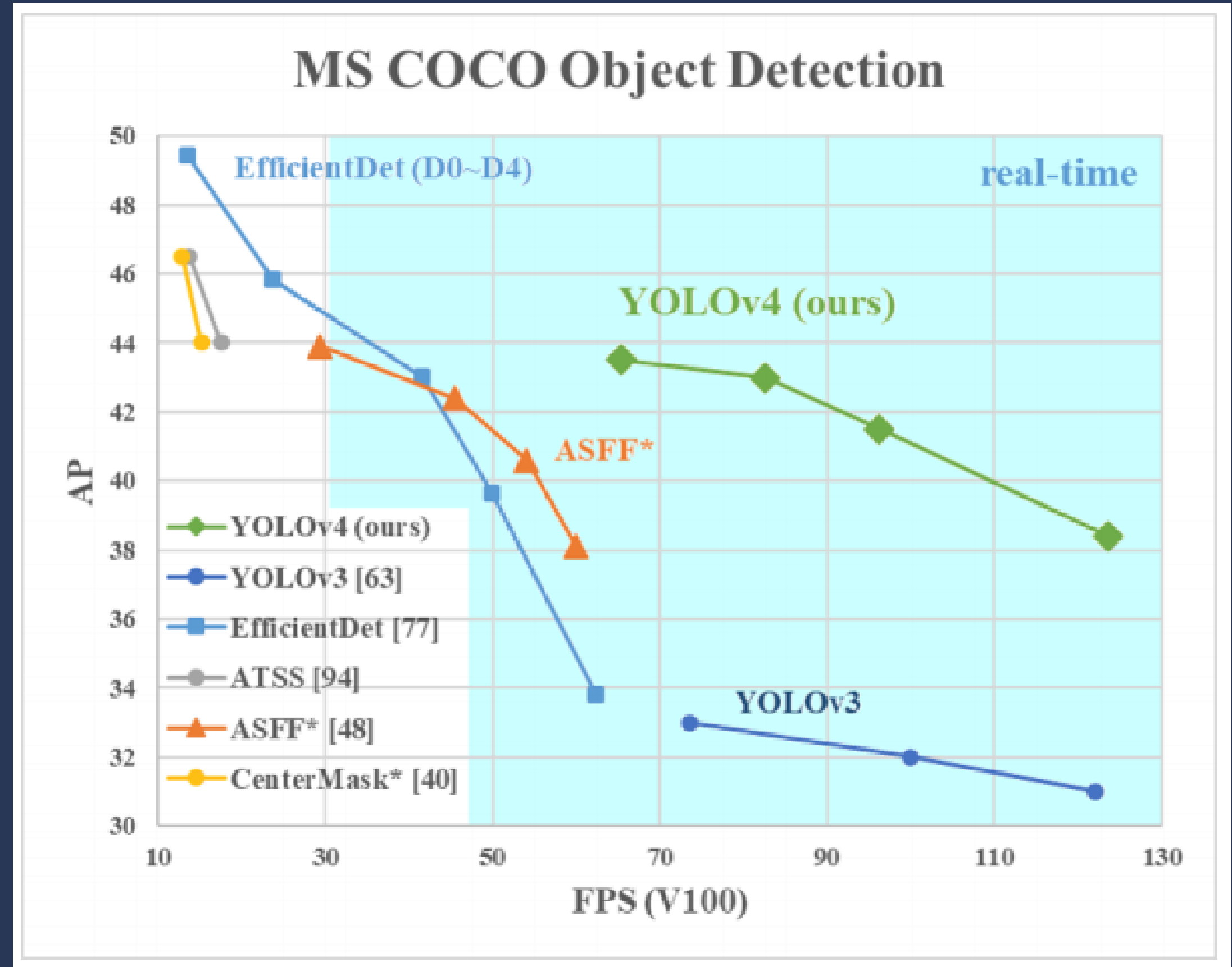


DISCUSSION

WHY YOLOV4 ?

YOLOv4 beats every other versions of YOLO in terms of FPS and Average Precision.

The high speed detection of YOLOv4 make it a convenient choice for real time scenarios.



CONCLUSION

The custom street sign detector model trained on nearly 700 images produces results that is 98% accurate. The test set of the model weights is only done on images but it can also be done on videos both real-time and downloaded.

There are still scopes of improvement in this model. If the models gets more dataset it may get trained better. One more scope is adjusting the ratio of training set to test set. Currently, the ratio of training set to test set is 10:1, i.e., 10%. If we increase the test set to 20% to 25%, then we may get better result.
