University of Mumbai

Traffic Sign Detection using Deep Learning

Submitted in partial fulfillment of requirements

For the degree of

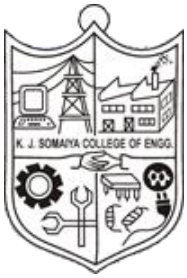
Bachelor of Technology

by

Shikta Das
Roll No: 1813011
Navneet Parab
Roll No: 1813035
Fenny Keniya
Roll No: 1813051
Aditya Shenoy
Roll No: 1813057

Guide

Prof. Maruti Zalte



Department of Electronics and Telecommunication Engineering
K. J. Somaiya College of Engineering, Mumbai-77
(Autonomous College Affiliated to University of Mumbai)

Batch 2018 -2022

University of Mumbai

Traffic Sign Detection using Deep Learning

Submitted in partial fulfillment of requirements

For the degree of

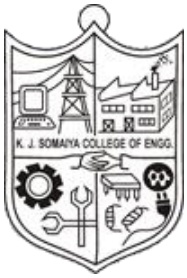
Bachelor of Technology

by

Shikta Das
Roll No: 1813011
Navneet Parab
Roll No: 1813035
Fenny Keniya
Roll No: 1813051
Aditya Shenoy
Roll No: 1813057

Guide

Mr. Maruti Zalte



Department of Electronics and Telecommunication Engineering
K. J. Somaiya College of Engineering, Mumbai-77
(Autonomous College Affiliated to University of Mumbai)

Batch 2018 -2022

K. J. Somaiya College of Engineering, Mumbai-77

(Autonomous College Affiliated to University of Mumbai)

Certificate

This is to certify that the dissertation report entitled **Traffic Sign Detection using Deep Learning** is a bona fide record of the dissertation work done by Shikta Das, Navneet Parab, Fenny Keniya and Aditya Shenoy in the year 2021-22 under the guidance of Mr. Maruti Zalte of Department of Electronics and Telecommunication Engineering in partial fulfillment of the requirement for the Bachelor of Technology degree in Electronics and Telecommunication Engineering of University of Mumbai.

_____ _____

Guide Head of the Department

_____

Principal

Date:

Place: Mumbai-77

K. J. Somaiya College of Engineering, Mumbai-77

(Autonomous College Affiliated to University of Mumbai)

Certificate of Approval of Examiners

We certify that this dissertation report entitled **Traffic Sign Detection using Deep Learning** is a bona fide record of project work done by Shikta Das, Navneet Parab, Fenny Keniya and Aditya Shenoy.

This project is approved for the award of a Bachelor of Technology Degree in Electronics and Telecommunication Engineering of University of Mumbai.

_____

Internal Examiner

_____

External Examiner

Date:

Place: Mumbai-77

K. J. Somaiya College of Engineering, Mumbai-77

(Autonomous College Affiliated to University of Mumbai)

DECLARATION

We declare that this written thesis submission represents the work done based on our and/or others' ideas with adequately cited and referenced the original source. We also declare that we have adhered to all principles of intellectual property, academic honesty and integrity as we have not misinterpreted or fabricated or falsified any idea/data/fact/source/original work/ matter in my submission.

We understand that any violation of the above will be cause for disciplinary action by the college and may evoke the penal action from the sources which have not been properly cited or from whom proper permission is not sought.

Signature of the Student **Roll No.**	**Signature of the Student** **Roll No.**
Signature of the Student **Roll No.**	**Signature of the Student** **Roll No.**

Date:

Place: Mumbai-77

Abstract

With the rapid development of technology, automobiles have become a crucial asset in our day-to-day lives. As the drivers are focused on the road while driving a lot of times, they miss out on essential traffic signs on the road. This leads to a lot of avoidable road accidents which is a threat to the lives of the drivers, passengers, and pedestrians. Having a way to alert the driver without shifting their attention from the road can be a great solution to improve road safety. A Traffic Sign Detection and Recognition system will detect and recognize any traffic signs on the road for the driver and alert him about them. This system allows the driver to be at ease while driving especially on risky or new roads.

When photos of traffic signs are collected in a real setting, they are modest in comparison to other items. As a result, it's more difficult to be precisely recognized and identified. Due to its great accuracy and speed of execution, the convolutional neural network (CNN) has recently made significant progress in object detection. YOLO (You Only Look Once) is an object detection method that employs the CNN core module to recognise traffic signs in a real-world setting. The objective of this thesis work is to implement Object Recognition techniques with the YOLOv3 framework, for detecting four categories of traffic signs from our custom dataset in a real environment and implementing our model on Raspberry Pi hardware.

***Key words*:** Traffic sign, YOLOv3, object detection, Deep Learning, Traffic Signs Dataset, Raspberry Pi, PiCam

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This chapter presents the introduction of our project, brief background, and our motivation to choose this particular topic as our project. It also gives an entire overview of what the scope of our project covers.

1.1 Background

Road accidents kill over one million people every year. It is the second highest source of death rate in India. Traffic signs are used to inform driver about the condition of the road, street turns, allowing or stopping of vehicle and whether one can use certain car attributes at certain time or place.

If the drive can correctly understand the traffic sign on the road a chance of an accident can be significantly reduced. Our goal is to create a Traffic Sign Detection system where we utilize Deep Learning concepts that incorporate detection and classification.

If we can detect and correctly recognise the traffic signs on the road ahead without manual aid we are performing traffic sign classification. Road signs such as speed limit signs, go slow signs, stop signs, and so on, are tough to understand from far. Being able to recognise traffic signs automatically allows us to install "driver alert" systems in automobiles that need to understand the road around them to assist and protect drivers. Computer vision and deep learning can help with a variety of issues, including traffic sign identification.

1.2 Motivation

Traffic sign detection and recognition is a future moving forward in the computer vision and a significant study in the field of the Advanced Driver Assistance System. There are basically divided in 2 methodologies, detection and recognition. The exactness of detection proportionally influences the output recognition. Road signs encompass necessary information regarding latest traffic conditions, describe road rights, , prompt dangerous

messages, road safety, and they prohibit and permit behaviors and driving routes, to name a few. Drivers get an enormous help when traffic signs are alerted to them as it can aid them focus on driving, make fewer wrong turns and also decrease rash driving due to unnecessary delay

1.3 Scope of the project

Every traffic signs in India have common elements like size, shape and color. Our sings are usually in the colors red, blue or yellow. They also have white and black usually used as background or to write text. There are common behavior in shapes as well. All signs in India are circles, triangles and rectangles. Having such interlinked characteristics can help gather common area of interest which aids in the detection and classification of signs on the road.

Traffic signs are bare on the road without any protection hence exposed in nature. It is often attacked by external factors, such as rain, sun, wind and dust [1]. Traffic-sign recognition is a essential for paving the way towards a better future for transportation as a whole.

1.4 Brief Description of project

In this project we are using a custom-curated data set with four different classes of road traffic signs namely no stopping sign, one-way sign, go slow sign, and speed limit. We will optimize this dataset with YOLO standards. A comparison between different YOLO versions will be conducted to understand and find the best algorithm for our project. We will construct an object detection code for reading traffic signs. Here training of the dataset and tuning of parameters will be done. We will analyze and classify various types of traffic signs.

In the next stage of our project, we will design a system that alerts the driver in real-time about the various traffic signs on the road. We will integrate OpenCV Module for hardware implementation. We will use Raspberry Pi for the hardware implementation of our program. The real-time traffic detection system will also consist of Pi Camera where the real-time object detection will be tested.

1.5 Organization of the report

This paper is organized into five major parts. Chapter 1 lays the foundation of our project and consists of an introduction, background, motivation behind our project and the scope of our project. Chapter 2 consists of the literature survey in which we used a deep understanding of our topic and the technologies to be used in our project. We surveyed four IEEE papers that were about object detection, the Yolo algorithm and traffic sign detection. Chapter 3 contains details about our project design and the general flow of our project. It also contains our problem statement, objectives of the thesis and theoretical explanations about object detection, YOLOv3 and our hardware Raspberry Pi. Chapter 4 presents the stages of implementation and analysis of our project which include obtaining the dataset, labelling process, comparing between YOLO versions, model training, model testing and hardware implementation. Chapter 5 presents the conclusions drawn by us based on our project and the future scope and further work of our project including discussions of our project applications.

Chapter 2

Literature Survey

This chapter presents a comprehensive and thorough literature survey of all relevant research papers and studies needed as a prerequisite before implementing the project. It consists of the survey of four IEEE research papers.

2.1 Traffic Sign Detection and Recognition with Convolutional Neural Network

A traffic sign detection and recognition system (TSDRS) aids in improving the current transportation systems. If traffic sings are detected with the help of automation then the driver can drive at ease with focus on the road. This is a huge blessing as it can significantly reduce road collisions. Here we present a system that can recognise traffic sign using deep learning algorithm. Main objective here is the detection of circle in a sign and classification of all traffic signs based on the detection of this circle shape.

Complicated and robust neural network fabric is used. This model can easily read characteristics from small objects. Hough's logic is used to mathematically locate the circle shape within our model. It identifies and processes the images before detection, which helps to boosts how accurate our predictions are and decrease run-time. Traffic-sign detection and recognition is worked through our model in multiple steps like Pre-processing, Detection and Classification.

Firstly, an image is processed using image enhancement techniques to spotlight essential information. After applying transformations we green light the detecting and identifying appropriate areas. Lastly, classification of the detected traffic begins using our proposed model. CNN is widely used to understand a variety of computer vision tasks. High accuracy and low losses are attained. TensorFlow is utilized for the implementation of CNN. About 98% accuracy is achieved using the custom dataset.

CNN shows great efficiency but cannot work in real-time. This is because it first finds the area of interest and then objects detection occurs. YOLO is super-fast as both activities are done simultaneously. YOLO can run in real-time which provides real-time detection and

helps the driver. The future scope is to develop object detection for traffic signs on YOLO[2].

2.2 Comparative study of various image classification deep learning algorithms

As the world moves towards automation and artificial intelligence, proper detection and classification of visual media become imperative. From driverless cars to hi-tech surveillance systems, there is a need for accurate and efficient image processing and classification. While its application may look commonplace, the task of image classification is not that easy.

There is a myriad of different algorithms which can be implemented for one to choose from, based on the computational budget, the degree of accuracy required and the use case. This research paper compares the performance of three of the most popular image classification deep learning algorithms as an attempt to find which is the fastest and most efficient one.

We provide a brief introduction for the origin of these algorithms, the paper explains the in-depth working of each, with the complex mathematical concepts that go behind every image classification algorithm. Once we are familiar with the theory behind them, the authors then explain the experimental setup that they used for the comparative analysis. The common dataset used for all three algorithms was the MICROSOFT COCO dataset, which is the benchmark dataset used for exactly that purpose. The authors use the Average Precision and F1 score as the metric on which they base their conclusions, after implementing the three models on the data.

The researchers found that SSD performed quite poorly as compared to the other two in classifying small objects and it also required a large amount of training data to provide any significant result. Faster R-CNN was slower than YOLO without and while it was relatively accurate, time complexity is a huge tradeoff since it required multiple passes over the same image.

YOLO was found to be the best, although it was optimized, since it gave the least false detections. YOLO is a clear winner among the three and between FRCNN and SSD, the

former is more accurate while the latter is faster. Looking at this, it was easy to decide that we wanted to use the YOLO algorithm for our project [3].

2.3 Real-Time Object Detection with YOLO

This research paper discussed the overview of the YOLO algorithm and it is working in detail for real-time object detector. Classification and Regression were the two categories discussed . It explains the difference between traditional Convolutional Neural Networks and YOLO algorithms which differ in their use of Classification and Regression. The work was explained using a sample image divided into 3x3 grids. If an object was found, the objectness was valued as one as opposed to zero if no proper object was found. These grids were used to mark the bounding box and assign labels. These labels consisted of information about the object being detected, its position, and class.

Methods such as Intersection over Union and Non-Max Suppression were used to accomplish the same. Calculation of Intersection over Union is done by Area of Intersection/Area of Union. IoU values which have a higher value than the threshold value (0.5) are considered precise prediction in nature. Non-Max Suppression was calculated by using high probability boxes and boxes whose IoU values were suppressed. We repeated this task until a box was selected. This is considered as the bounding box for that desired object. In the case of multiclass detection in the same grid, Anchor boxes were used. For the training of the entire model, loss functions such as Classification, Localization and Confidence were described in detail [4].

2.4 Detection of Malaysian Traffic Signs via Modified YOLOv3 Algorithm

This research paper presents a complete overview of the experimentation of a deep learning model through a modified YOLO version3 algorithm that has the inclusion of Spatial Pyramid Pooling (SPP) before its final convolution layer to detect the road signs in a live environment for the roads in the country of Malaysia. The dataset used was 2000 daytime images of ten different traffic signs of Malaysia. The implementation overview included the

methodology, conclusions, and future scope for traffic sign detection using YOLOv3. YOLO used an object detector which is modelled according to convolutional neural network.

The paper explains the working of the CNN and YOLO framework and runs us through the four major processes involved in the entire object detection process. Further, the paper compares YOLOv3 to YOLO and points out why YOLOv3 is a better fit as well as more accurate for our use case. It helps us understand the working of the SPP layer and how it improves the overall accuracy. The paper details the sources and the distribution of the sources for the training and testing dataset used for the implementation and shares with us the metrics used for evaluation and presents them as results.

Results of this research paper have shown to be promising. The mAP value for our proposed CNN model is that our model detected the traffic sign at 82% accuracy [5].

Chapter 3

Project Design

This chapter presents the core problem statement of the project and the flow that was followed while implementing it. Important concepts and software's that were used are also briefly explained.

3.1 Problem Statement

Implement a real-time traffic sign detection system using the YOLOv3 deep learning algorithm to integrate it with a Raspberry Pi camera module.

3.2 Block Diagram

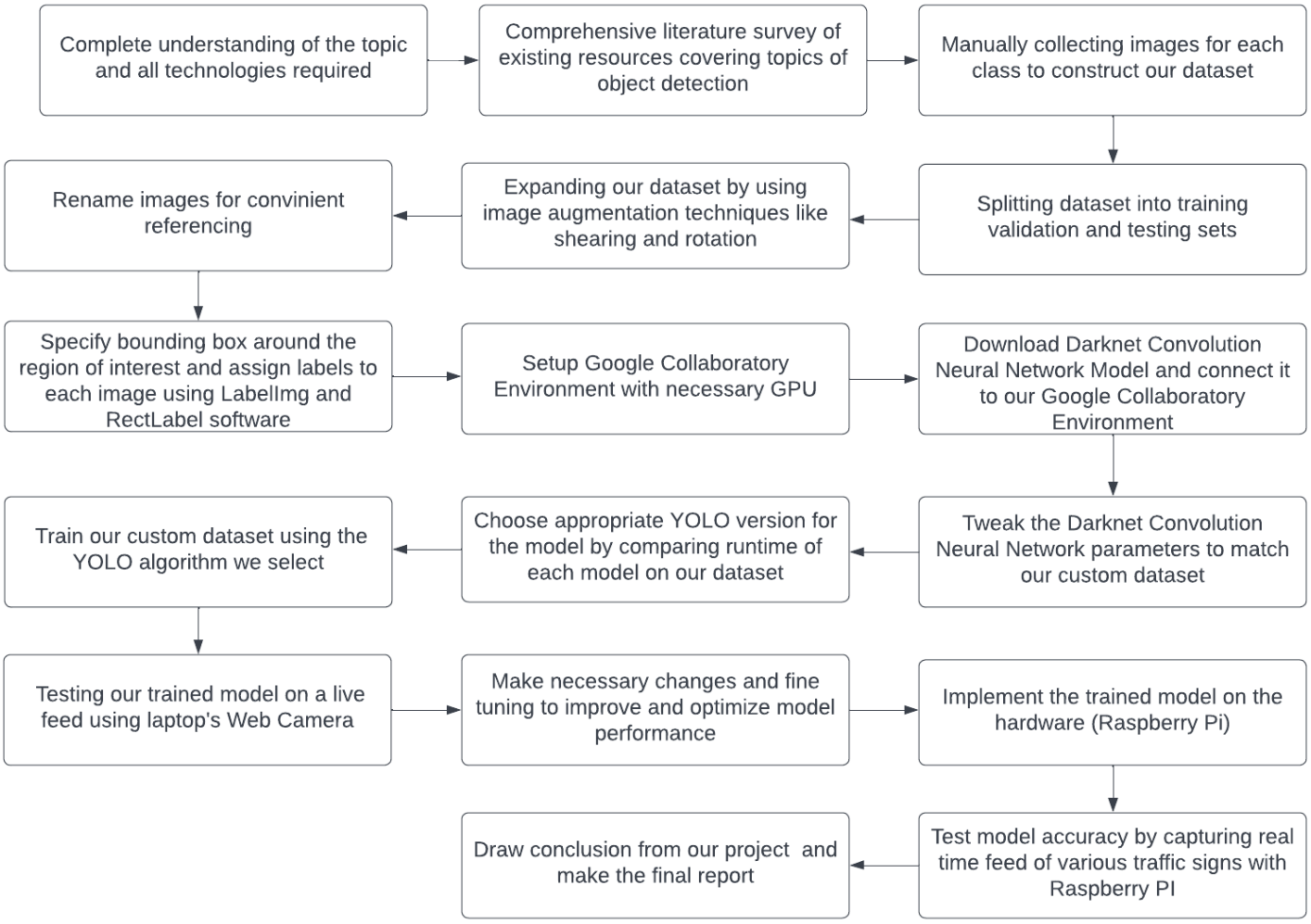


Fig. 3.1 Block diagram of our project work flow

3.3 Objectives

- Complete understanding of the topic and all technologies required including significant research and a comprehensive literature survey of existing resources covering relevant topics related to the project.
- Compilation of custom multiclass dataset of traffic signs. Manually click photos of available signs for each class.
- Standardization and formatting of all images as well as image augmentation techniques. Use labelImg and RectLabel Lite for setting bounding boxes and labelling each image.
- Setting up the Google Colaboratory environment for GPU intensive training.
- Downloading and connecting the DarkNet CNN to our dataset and making necessary changes in the input parameters.
- Choose appropriate YOLO version for the model.
- Train our custom dataset using the YOLOv3 algorithm.
- Hardware implementation of the model on Raspberry Pi.
- Evaluate performance and accuracy of the model.
- Conclude the project with final report.

3.4 Object Detection

Object detection is a procedure in computer vision that is responsible for the identification and localization of objects with a video or image. In a live feed, we can locate our desired objects by creating bounding boxes around them, this is how the object detection works. Object detection is usually misinterpreted with image recognition[6].

The various types of computer vision problems are as follows:

1. **Image classification:** The most renowned problem in computer vision probably. It includes the problem of classifying a particular image into one of the different classes.

2. **Localization:** It finds the location of a single object within an image very similar to classification. Classification and localization can be combined to first locate an object and then classify it into one of many possible classes.

3. **Instance segmentation:** This is used to find the pixel-by-pixel mask of each of the detected objects and not only find the objects inside the image.

4. **Object detection:** We use object detection when we want to locate and classify multiple objects in an image or video at the same instance and not just a single object. Unlike the classification problem, the output of object detection can be varying in length due to the variability of the number of objects detected in each image[1].

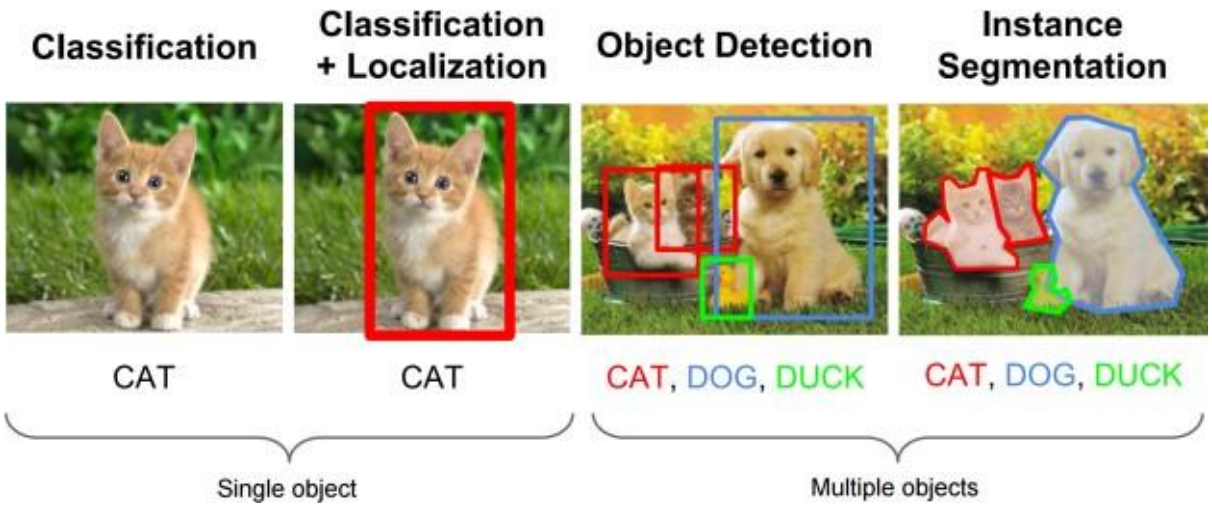


Fig. 3.2 Comparison between the various computer vision problems

Object Detection using deep learning techniques includes:

1. Convolutional Neural Networks (CNN): CNNs are very good for images since they are a kind of neural network that are very fast and accurate at detecting patterns in multidimensional spaces.

2. Region-based CNN (R-CNN): They supposedly improve the object detection problem by almost 50%.

   They recommend a three-step process-

1. Use a region proposal strategy like selective searching to extract all the possible objects.

2. Extract features from every region using a convolution neural net

3. Categorize every region with support vector machines

1. Fast R-CNN: In 2015, an upgrade to the R-CNN was proposed with a new architecture by the main author of the R-CNN paper. In a single model, Fast R-CNN can region selection and perform feature extraction as well as solve some of the problems of R-CNN.

2. Faster R-CNN: By adding the region extraction mechanism into the object detection network a huge part of the object detection problem was solved by Faster R-CNN. For an input image, it can give bounding boxes around the objects in the image and the list of the classes the objects are classified into as the output.

3. YOLO: It offers improved speed and accuracy for object detection with deep learning. YOLO has the integration of the entire object detection and classification in a single neural network. It performs all the steps in a single pass through a single powerful network instead of extracting characteristics and regions separately. Since it can perform object detection at video streaming frame rates, it is apt for an application that requires real-time detection and recognition [7].

Object detection is an essential technology much needed for advanced driver assistance systems. These systems can help the cars to detect driving lanes, detect pedestrians and detect traffic signs to improve vehicle safety and avoid road accidents.

3.5 You Only Look Once Version 1 (YOLOv1)

YOLOv1 is a simple and quick object detector that is well suited to applications that require real-time predictions. It considers detection to be a regression problem rather than a classification one. Rather than filtering out areas with a high likelihood of being an item and giving them to a network that finds boxes, a single monolithic network must handle feature extraction, box regression, and classification. Yolo has less background mistakes because it is applied to the entire image. The image must be 448x448 pixels in size.

Batch normalization and leaky ReLU activation are used in the Yolo-V1 architecture. Yolo divides the subject into NxN grids to deal with. Each grid represents a classifier that generates bounding boxes around observable items. when an image is sent through YOLOv1 It returns the x,y coordinates of the object's centroid, as well as the object's width and height, together with a class probability. YOLOv1 makes use of the darknet architecture, which has been tweaked to include convolution and two fully connected layers [8].

There are several drawbacks to this architecture. It has a low recall and produces a lot of localization blunders. Small items are difficult to notice. We utilize a fully connected layer to regress the bounding box, thus it has strong bounding box limitations.

3.6 You Only Look Once Version 2 (YOLOv2)

Improvements have been made in version 2, bringing the mean average precision of this model to the same level as Faster-RCNN and SSD. The fully-connected layer is removed in the end to achieve this accuracy. As a result, the architecture is resolution-independent, meaning the network parameters can be adjusted to meet any input resolution.

V2 employs darknet 19, which consists of 19 convolution layers, 5 max pulling layers, and a SoftMax layer for categorization. Darknet achieves great accuracy by teaching the model to generalize and detect objects via multiscale training. As the MAP value grows and localization errors reduce, the fine grain characteristics aid enhance precision for recognizing smaller objects.[8]

3.7 You Only Look Once Version 4 (YOLOV4)

Previous YOLO versions utilize many GPUs for training with a huge mini-batch size, and doing this with one GPU makes the runtime for training impractical. YOLO v4 addresses the issue by creating an object detector that trains on a single GPU with a decreased mini-batch size. This helps in training a super-fast and accurate object detector with a single 1080 Ti or 2080 Ti GPU.

YOLOVv4 uses a modern method of data augmentation called “Mosaic” that transforms four images of the training dataset into one. This reduces the requirement of choosing a large mini-batch size for training. They also use Self-Adversarial Training, which works in 2 forward-backward stages. In the 1st stage, the neural network changes the original image instead of the weights. This lets the neural network execute an adversarial attack on itself, altering the original image to create the deception that there is no desired object on the image.

YOLO v4 achieves advanced results (43.5% AP) for real-time object detection and can run at a speed of 65 FPS on a V100 GPU. This model is good if one wants less accuracy but much higher FPS. [8]

3.8 You Only Look Once Version 3 (YOLOv3)

Object detection is one of the most conventional problems in computer vision: Recognize what the objects are inside a given image and also where they are in the image. Detection has more complexity than classification. It recognizes objects but doesn't tell the location of the object in the image and it cannot read from images that have multiple objects.

You Only Look Once, Version 3 is a real-time object detection model that recognizes desired objects in videos, live feeds, or images. YOLO uses a variety of characteristics for learning using a deep convolutional neural network to identify an object. YOLO Versions 1,2 and 3 were designed by Joseph Redmon and Ali Farhadi[11].

The 1st version of YOLO was made in 2016, and version 3, which we have used in our project, was born two years later in 2018. YOLOv3 is an upgraded version of YOLO and YOLOv2. YOLO is applied using the Keras Deep Learning modules. Object classification systems are used by Artificial Intelligence programs to observe objects in a class as subjects of desire. They sort objects from images into groups where objects with similar features are placed in one class, while others are ignored unless planned to do otherwise [9].

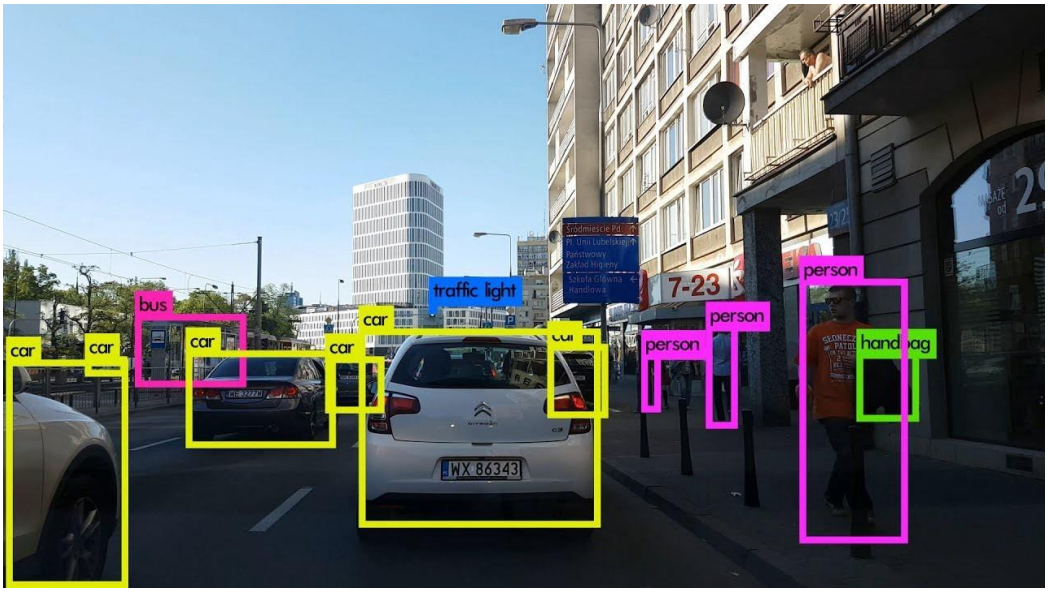


Fig. 3.3 Image classification through YOLOv3

How does YOLOv3 work?

YOLO is a Convolutional Neural Network for execution of object detection in real-time. CNN's are classification-based systems that can analyse input images as structured vector of data and detects features with similarity. YOLO has the benefit of being extremely fast compared to other networks and still maintains high accuracy. It allows the algorithm to visualize the whole image at test time, so its predictions are well-versed by the entire context in the image. YOLO and other convolutional neural network algorithms "score" regions based on their resemblances to established classes. High-scoring regions are considered positive detections of desired class they most closely detect with. For example, in a live feed of a road, YOLO can be used to detect various traffic signs depending on the regions of video that scores highly in comparison to selected classes of these traffic signs [9].

How is YOLOv3 better than its older versions?

YOLO v2 used a deep learning architecture darknet-19, an originally 19-layer network supplemented with 11 more layers for object detection. With a 30-layer architecture, YOLO v2 shows low accuracy with small object detections. This happens due to the loss of fine-grained characteristics as the layers downsampled the input. To improve this, YOLO v2 identity mapping, concatenating feature maps from a previous layer to store low-level features. However, YOLO v2's architecture was still deficient of some essential elements that are now common in every advanced model. YOLO v3 incorporates all of these. YOLO v3 uses a modified style of Darknet, which has 53-layer neural network trained on Imagenet. For detection, 53 more layers are stacked onto it, providing a 106-layer fully convolutional architecture for YOLO v3. This causes some slowness with YOLO v3 compared to YOLO v2 [9].

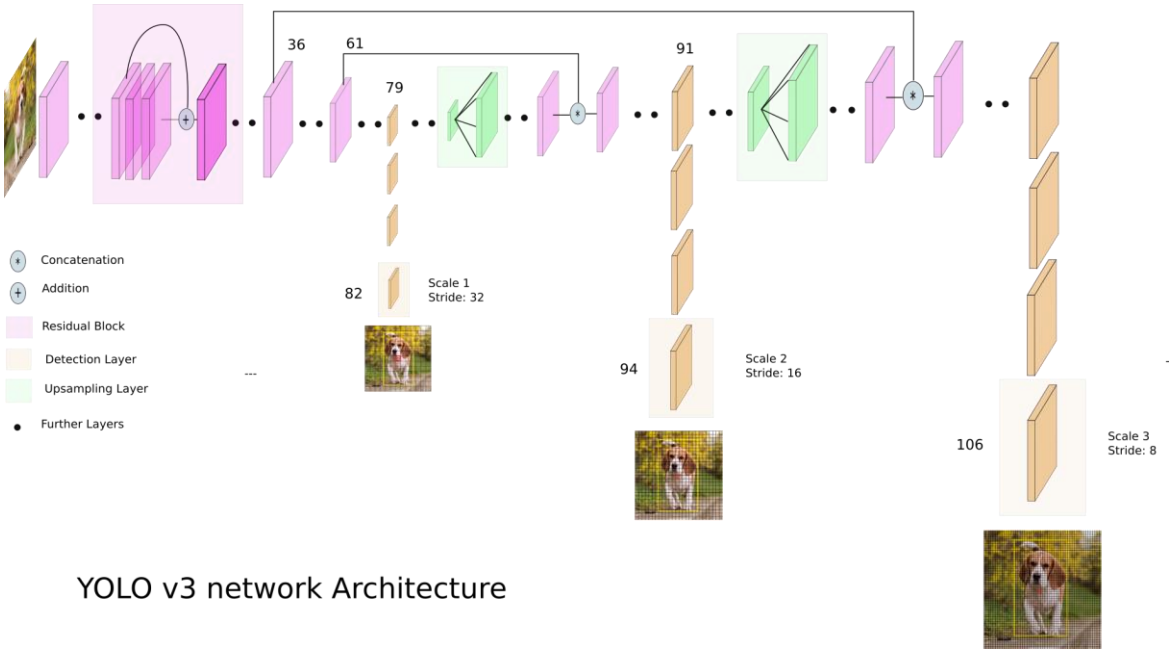


Fig. 3.4 YOLO v3 Architechture

The following are additional features of YOLO version 3:

1. **Speed:** YOLOv3 is fast and efficient in terms of mean average precision (mAP) and intersection over union (IOU) values. It runs faster than rest of the detection algorithms with decent performance. One can easily trade-off between speed and accuracy simply by altering the model's size, and no retraining parameters are required.

2. **Precision for Detecting Smaller Objects:** Detections at different layers helps address the issue of detecting small objects, a frequent complaint with YOLO v2. The upsampled layers added with the layers before help reserve the powdered features which help in detecting smaller objects and boosting accuracy. The 13 x 13 layer is responsible for detecting bigger objects, and the 52 x 52 layer detects the smaller objects, with the 26 x 26 layer detecting all the objects within the range.

3. **Specificity of Classes:** YOLOv3 uses autonomous logistic classifiers and binary cross-entropy loss for the predictions of classes during training of model. This make it possible to use datasets with higher complexity. YOLO v3 uses a multilabel process where classes can be more specific and be multiple in nature for individual bounding boxes.

4. **Detection at three scales:** The newer architecture boasts residual skip connections, and upsampling. The most prominent characteristics of version3 is that it makes detections at three diverse scales. YOLO is a fully convolutional network and its output is generated by applying a 1 x 1 kernel on a feature map. In YOLO v3, the detection happens by applying 1 x 1 detection kernels on feature maps of three diverse sizes at three different location within the network.

5. **More bounding boxes per image:** For an input image of the identical size, YOLO v3 predicts higher number of bounding boxes than YOLO v2. At its instinctive resolution of 416 x 416, YOLO v2 detects13 x 13 x 5 = 845 boxes. At each grid cell, 5 boxes were predicted using 5 anchors. On the other hand, YOLO v3 detects boxes at 3 diverse scales. For the same image of 416 x 416, the predicted boxes is 10,647. This means that YOLO v3 predicts 10 times the number of boxes detected by YOLO v2. At every scale, each grid can predict 3 boxes using 3 anchors. Since there are three diverse scales, the number of anchor boxes used in this is 9, 3 for each scale.

3.9 Raspberry Pi

The Raspberry Pi is a cheap, small-sized computer that plugs into a computer display or TV, and uses a general keyboard and mouse. Raspberry Pi is generally used for Image/Video Processing, IoT-based applications, and Robotics applications.. Raspberry Pi is slower than a laptop or

desktop but is still a computer that can provide all the expected features or abilities, at low power consumption. Raspberry Pi is a single-board computer, because all components that make a computer like the microprocessor, memory, wireless radios, and ports are all on one circuit board.

The Pi is a Linux computer, so technically it can do everything a Linux computer can do, such as running email and Web servers, acting as network storage, or being used for object detection.

Unlike computers with a built-in hard drive or SSD, the Pi's OS is installed onto a microSD card, which consists of all our files since the board doesn't consist of any built-in storage. This structure makes it easy for us to expand the storage and switch between different operating systems by swapping out microSD cards[10].

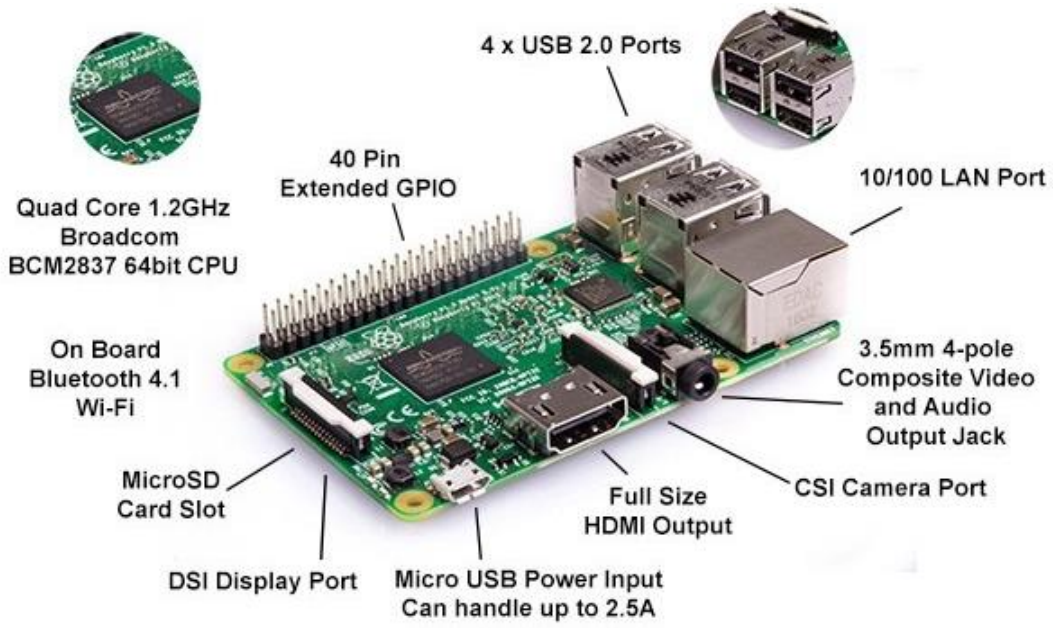


Fig. 3.5 Features Raspberry Pi version 4

Chapter 4

Implementation and Experimentation

This chapter presents the stages of implementation and analysis of our entire project which include obtaining the dataset, labelling process, comparing between YOLO versions, model training, model testing and hardware implementation.

4.1 Obtaining the Dataset

The first step of any object detection model is collecting images and performing annotation. For our traffic sign detection project using deep learning, we chose to make our dataset of the traffic signs in and around Mumbai City due to the lack of any decent already available dataset of the Mumbai roads. We decided to aim at four of the most common traffic signs on the Mumbai roads for moving vehicles.

The four traffic signs selected are as follows:

1. Speed Limit Sign
2. Go Slow Sign
3. One Way Sign
4. No Stop Sign

To expand our dataset further, we applied image augmentation techniques of rotation (between -15 degrees and +15 degrees) and shearing. Finally, our dataset consists of 4528 images across four classes of traffic signs.

Further, we split the entire dataset into a training, validation, and testing set as follows:

Training set- 3963 images

Validation set- 376 images

Testing set- 189 images

All the photos in the dataset are manually clicked by us using four different cameras. The specifications of the four cameras are as follows:

1. iPhone 12, 2 cameras-12 MP, 12MP

2. Samsung A52, 4 cameras- 64MP, 12 MP, 5 MP, 5 MP
3. iPhone 13, 2 cameras-12 MP, 12MP
4. iPhone 13, 2 cameras- 12 MP, 12 MP

It was important for us to manually get images clicked of the traffic signs to include variability in our dataset and ensure the robustness of our model. The different variabilities are included by:

- Taking pictures with different lighting (day and night)
- Taking pictures from different distances and angles
- Including signs that are broken, bent and rusted
- Taking pictures of the traffic sign with multiple signs in the frame
- Including different types of traffic signs in each class
- Taking pictures from moving and stationary vehicles

Fig. 4.1 Examples of one way signs in our dataset

Fig. 4.2 Examples of speed limit signs in our dataset

Fig. 4.3 Examples of Go Slow in our dataset

Fig. 4.4 Examples of No Stop in our dataset

4.2 Labelling Process

For the YOLO algorithm to train upon the training data, labelling of the image was necessary. Labelling of the image is meant for the creation of tags that would contain the information about the object like the class type and the coordinates of the bounding boxes. This information is stored in a'.txt' file with that name of the image as the title of this text file.

	<object-class>	<x>	<y>	<width>	<height>
For eg. 3	0.565833	0.446250	0.595000	0.787500	

Fig. 4.5 Example of Txt file with labelled values

Here, the first value 3 refers to the object class while 0.565833 and 0.446250 refer to the location of the center of the bounding box on the x and y axis 0.787500 and . 0.595000 are the height and width of the bounding box which we manually select.

If image contains more than one instance of the objects (same or different), multiple instances of object-class, coordinates and dimensions can be stored in the '.txt' file by using a new line. The YOLO algorithm can identify various classes in a single image with the help of this single .txt file. For the above task, we made use of software such as Labellmg for Windows systems and RectLabel for macOS systems. This software requires the input data to be in .jpeg format only. Since our data was obtained through various sources of phones and cameras we had to convert the file formats from .RAW, .HEIC, etc. to .jpeg.

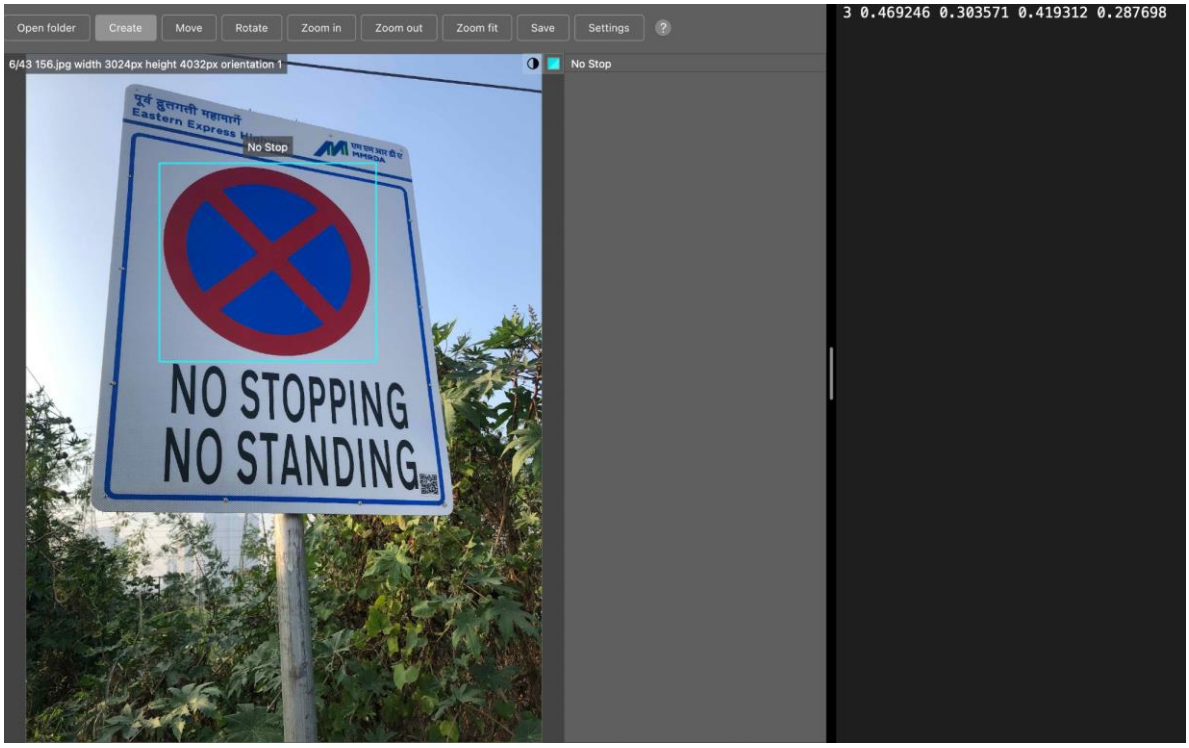


Fig. 4.6 RectLabel Lite software for creating bounding boxes and labelling the images

4.3 Experimentation between the various versions of YOLO

We used a part of our dataset to run all the four versions of YOLO and evaluated them based on their features like time taken to train and IOU values and chose to use YOLOv3 for our model training because it provided the best trade-off.

Features	YOLOv1	YOLOv2	YOLOv3	YOLOv4
Time	7.69 hours	12.3 hours	15.7 hours	22 hours
IOU value	0.59	0.62	0.75	0.78

Fig 4.1 Properties of various YOLO versions

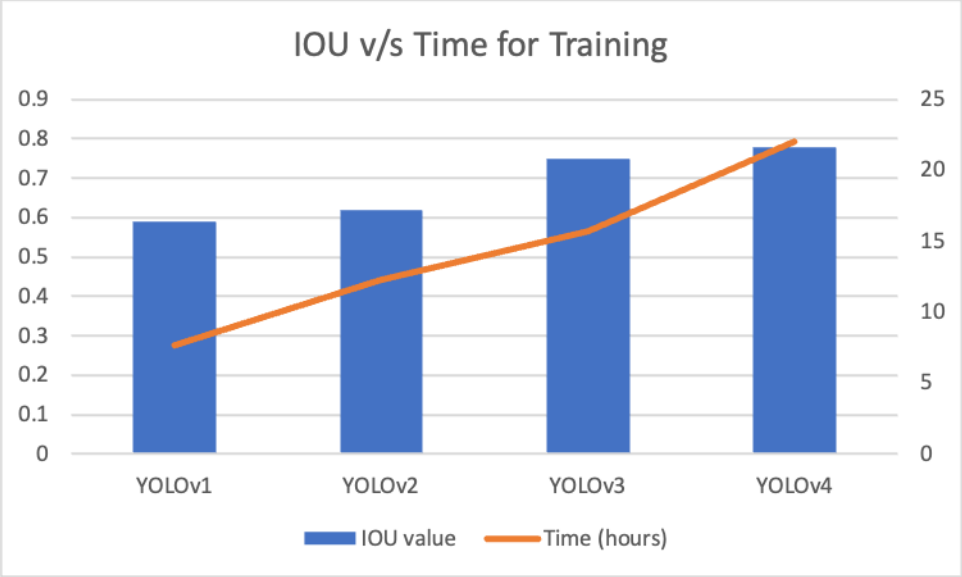


Fig 4.2 Analysis to select desired model

```
 100 conv    256       3 x 3/ 1     52 x  52 x 128 ->   52 x  52 x 256 1.595 BF
 101 conv    128       1 x 1/ 1     52 x  52 x 256 ->   52 x  52 x 128 0.177 BF
 102 conv    256       3 x 3/ 1     52 x  52 x 128 ->   52 x  52 x 256 1.595 BF
 103 conv    128       1 x 1/ 1     52 x  52 x 256 ->   52 x  52 x 128 0.177 BF
 104 conv    256       3 x 3/ 1     52 x  52 x 128 ->   52 x  52 x 256 1.595 BF
 105 conv     18       1 x 1/ 1     52 x  52 x 256 ->   52 x  52 x  18 0.025 BF
 106 yolo
[yolo] params: iou loss: mse (2), iou_norm: 0.59, obj_norm: 1.00, cls_norm: 1.00, delta_norm: 1.00, scale_x_y:
1.00
Time elapsed = 7.69420 hours
Total BFLOPS 65.304
avg_outputs = 516723
Allocate additional workspace_size = 52.43 MB
Done! Loaded 75 layers from weights-file
Learning Rate: 0.001, Momentum: 0.9, Decay: 0.0005
```

Fig 4.7 YOLOv1 output

```
   CUDA-version: 11010 (11020), cuDNN: 7.6.5, GPU count: 1
 OpenCV version: 3.2.0
Yolov2.3.2_training
 0 : compute_capability = 750, cudnn_half = 0, GPU: Tesla T4
net.optimized_memory = 0
mini_batch = 4, batch = 128, time_steps = 1, train = 1 12.3809 hours left
   layer   filters  size/strd(dil)      input                output
   0 Create CUDA-stream - 0
 Create cudnn-handle 0
conv     32       3 x 3/ 1    832 x 832 x   3 ->  832 x 832 x  32 0.489 BF
   1 conv     64       3 x 3/ 2    416 x 416 x  32 ->  208 x 208 x  64 1.595 BF
   2 conv     32       1 x 1/ 1    208 x 208 x  64 ->  208 x 208 x  32 0.177 BF
   3 conv     64       3 x 3/ 1    208 x 208 x  32 ->  208 x 208 x  64 1.595 BF
   4 Shortcut Layer: 1,  wt = 0, wn = 0, outputs: 416 x 416 x  64 0.008 BF
   5 conv    128       3 x 3/ 2    208 x 208 x  64 ->  104 x 104 x 128 1.595 BF
   6 conv     64       1 x 1/ 1    104 x 104 x 128 ->  104 x 104 x  64 0.177 BF
   7 conv    128       3 x 3/ 1    104 x 104 x  64 ->  104 x 104 x 128 1.595 BF
   8 Shortcut Layer: 5,  wt = 0, wn = 0, outputs: 208 x 208 x 128 0.001 BF
   9 conv     64       1 x 1/ 1    104 x 104 x 128 ->  104 x 104 x  64 0.177 BF
  10 conv    128       3 x 3/ 1    104 x 104 x  64 ->  104 x 104 x 128 1.595 BF
  11 Shortcut Layer: 8,  wt = 0, wn = 0, outputs: 52 x 52 x 128 0.001 BF
  12 conv    256       3 x 3/ 2    104 x 104 x 128 ->   52 x  52 x 256 1.595 BF
  13 conv    128       1 x 1/ 1     52 x  52 x 256 ->   52 x  52 x 128 0.177 BF
```

Fig 4.8 YOLOv2 output

```
CUDA-version: 11010 (11020), cuDNN: 7.6.5, GPU count: 1
 OpenCV version: 3.2.0
yolov3_training
 0 : compute_capability = 750, cudnn_half = 0, GPU: Tesla T4
net.optimized_memory = 0
mini_batch = 4, batch = 64, time_steps = 1, train = 1 15.72360 hours left
   layer   filters  size/strd(dil)      input                output
   0 Create CUDA-stream - 0
 Create cudnn-handle 0
conv     32       3 x 3/ 1    416 x 416 x   3 ->  416 x 416 x  32 0.299 BF
   1 conv     64       3 x 3/ 2    416 x 416 x  32 ->  208 x 208 x  64 1.595 BF
   2 conv     32       1 x 1/ 1    208 x 208 x  64 ->  208 x 208 x  32 0.177 BF
   3 conv     64       3 x 3/ 1    208 x 208 x  32 ->  208 x 208 x  64 1.595 BF
   4 Shortcut Layer: 1,  wt = 0, wn = 0, outputs: 208 x 208 x  64 0.003 BF
   5 conv    128       3 x 3/ 2    208 x 208 x  64 ->  104 x 104 x 128 1.595 BF
   6 conv     64       1 x 1/ 1    104 x 104 x 128 ->  104 x 104 x  64 0.177 BF
   7 conv    128       3 x 3/ 1    104 x 104 x  64 ->  104 x 104 x 128 1.595 BF
   8 Shortcut Layer: 5,  wt = 0, wn = 0, outputs: 104 x 104 x 128 0.001 BF
   9 conv     64       1 x 1/ 1    104 x 104 x 128 ->  104 x 104 x  64 0.177 BF
  10 conv    128       3 x 3/ 1    104 x 104 x  64 ->  104 x 104 x 128 1.595 BF
  11 Shortcut Layer: 8,  wt = 0, wn = 0, outputs: 104 x 104 x 128 0.001 BF
  12 conv    256       3 x 3/ 2    104 x 104 x 128 ->   52 x  52 x 256 1.595 BF
  13 conv    128       1 x 1/ 1     52 x  52 x 256 ->   52 x  52 x 128 0.177 BF
  14 conv    256       3 x 3/ 1     52 x  52 x 128 ->   52 x  52 x 256 1.595 BF
  15 Shortcut Layer: 12,  wt = 0, wn = 0, outputs:  52 x  52 x 256 0.001 BF
  16 conv    128       1 x 1/ 1     52 x  52 x 256 ->   52 x  52 x 128 0.177 BF
  17 conv    256       3 x 3/ 1     52 x  52 x 128 ->   52 x  52 x 256 1.595 BF
  18 Shortcut Layer: 15,  wt = 0, wn = 0, outputs:  52 x  52 x 256 0.001 BF
```

Fig 4.9 YOLOv3 output

```
CUDA-version: 11010 (11020), cuDNN: 7.6.5, GPU count: 1
 OpenCV version: 3.2.0
yolov4_training
 0 : compute_capability = 370, cudnn_half = 0, GPU: Tesla K80
net.optimized_memory = 0
mini_batch = 8, batch = 64, time_steps = 1, train = 1 22.00416 hours left
   layer   filters  size/strd(dil)      input                output
   0 Create CUDA-stream - 0
 Create cudnn-handle 0
conv     32       3 x 3/ 1    608 x 608 x   3 ->  608 x 608 x  32 0.639 BF
   1 conv     64       3 x 3/ 2    608 x 608 x  32 ->  304 x 304 x  64 3.407 BF
   2 conv     64       1 x 1/ 1    304 x 304 x  64 ->  304 x 304 x  64 0.757 BF
   3 route  1                                     ->  304 x 304 x  64
   4 conv     64       1 x 1/ 1    304 x 304 x  64 ->  304 x 304 x  64 0.757 BF
   5 conv     32       1 x 1/ 1    304 x 304 x  64 ->  304 x 304 x  32 0.379 BF
   6 conv     64       3 x 3/ 1    304 x 304 x  32 ->  304 x 304 x  64 3.407 BF
   7 Shortcut Layer: 4,  wt = 0, wn = 0, outputs: 304 x 304 x  64 0.006 BF
   8 conv     64       1 x 1/ 1    304 x 304 x  64 ->  304 x 304 x  64 0.757 BF
   9 route  8 2                                   ->  304 x 304 x 128
  10 conv     64       1 x 1/ 1    304 x 304 x 128 ->  304 x 304 x  64 1.514 BF
  11 conv    128       3 x 3/ 2    304 x 304 x  64 ->  152 x 152 x 128 3.407 BF
  12 conv     64       1 x 1/ 1    152 x 152 x 128 ->  152 x 152 x  64 0.379 BF
  13 route  11                                    ->  152 x 152 x 128
  14 conv     64       1 x 1/ 1    152 x 152 x 128 ->  152 x 152 x  64 0.379 BF
```

Fig 4.10 YOLOv4 output

4.4 Model Training

What is model training?

Model training is a process wherein our Machine Learning algorithm looks at the pre-labelled dataset and learns how to make accurate predictions with minimal error. Since a dataset with marked output is used it can easily aim to improve the weights of our model. This is done by linking the input data to output where similarities are shared. The output is generally the object desired by our model that captures these functions.

These are steps we followed for model training:

1) We cloned our darknet repository on our google drive

2) We make changes to the existing make file with regards to the GPU, OpenCV, and Kuda variables. We changed the variables from 0 to 1.

3) We modified the YOLOv3 architecture by changing the variables like batch size, subdivision, highest batch sizes, classes and the filters through which the output will have only four classes.

4) We used pre-trained weights for transfer learning since that would reduce our computational resources and time as well.

5) We unzipped the uploaded data and created a .data file for every class selected.

6) Then we trained the YOLO model by importing the glob module. The globe module converts our images into lists and we will be using the glob to train the model.

7) Once the final training has started it displays the accuracy, and loss which keeps changing as consecutive epochs are run and the remaining time.

What is model validation?

Model validation refers to the process of verifying the accuracy of our model on a completely new dataset which the model has not seen. This step gives and extra layer of surety, where we can analysis our misaligned values through various metrics.

The screenshots below display our training model:

1- Mount the google drive

- On the google drive, create a main folder for YOLO custom detection
- Within that folder, create subfolders for backup and custom weight
- Upload the dataset zip file within the main folder

```
[ ] #import google drive
    from google.colab import drive
```

```
#mount google drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

```
[ ] #check if the dataset exisit
    !ls '/content/drive/My Drive/yolo_custom_model_Training'
```

```
images.zip
```

2- Clone the Darknet

```
[ ] !git clone 'https://github.com/AlexeyAB/darknet' '/content/drive/My Drive/yolo_custom_model_Training/darknet'
```

```
Cloning into '/content/drive/My Drive/yolo_custom_model_Training/darknet'...
remote: Enumerating objects: 15412, done.
remote: Total 15412 (delta 0), reused 0 (delta 0), pack-reused 15412
Receiving objects: 100% (15412/15412), 14.02 MiB | 5.29 MiB/s, done.
Resolving deltas: 100% (10356/10356), done.
Checking out files: 100% (2050/2050), done.
```

3- Compile Darknet after enabling GPU, CUDA and OpenCV

```
[ ] # change makefile to have GPU and OPENCV enabled
    %cd '/content/drive/My Drive/yolo_custom_model_Training/darknet'
    !sed -i 's/OPENCV=0/OPENCV=1/' Makefile
    !sed -i 's/GPU=0/GPU=1/' Makefile
    !sed -i 's/CUDNN=0/CUDNN=1/' Makefile
    !make
```

```
[ ] %cd '/content/drive/My Drive/yolo_custom_model_Training/darknet'

    /content/drive/My Drive/yolo_custom_model_Training/darknet

[ ] !ls

    3rdparty        DarknetConfig.cmake.in  json_mjpeg_streams.sh  scripts
    backup          darknet_images.py       LICENSE                src
    build           darknet.py              Makefile               vcpkg.json
    build.ps1       darknet_video.py        net_cam_v3.sh          video_yolov3.sh
    cfg             data                    net_cam_v4.sh          video_yolov4.sh
    cmake           image_yolov3.sh         obj
    CMakeLists.txt  image_yolov4.sh         README.md
    darknet         include                 results
```

4- Create a copy of yolov4 (or any other version of yolo you want to train)

- Need to run this once

```
[ ] !cp cfg/yolov3.cfg cfg/yolov3_training.cfg
```

5- Create a copy of yolov4 (or any other version of yolo you want to train)

- Need to run this once

```
#Modify YOLO archicture
!sed -i 's/batch=1/batch=64/' cfg/yolov3_training.cfg
!sed -i 's/subdivisions=1/subdivisions=16/' cfg/yolov3_training.cfg
!sed -i 's/max_batches = 500200/max_batches = 2000/' cfg/yolov3_training.cfg
!sed -i '610 s@classes=80@classes=1@' cfg/yolov3_training.cfg
!sed -i '696 s@classes=80@classes=1@' cfg/yolov3_training.cfg
!sed -i '783 s@classes=80@classes=1@' cfg/yolov3_training.cfg
!sed -i '603 s@filters=255@filters=18@' cfg/yolov3_training.cfg
```

6- Unzip the uploaded data

```
!unzip '/content/drive/My Drive/yolo_custom_model_Training/images.zip' -d 'data/obj'

Archive:  /content/drive/My Drive/yolo_custom_model_Training/images.zip
checkdir:  cannot create extraction directory: data/obj
           File exists
```

7- Create .data file

```
/obj.names'
 = data/train.txt\nvalid  = data/test.txt\nnames = data/obj.names\nbackup = /content/drive/My Drive/yolo_custom_model_Training/b

[ ] !ls 'data/obj (1)'

    dataset
```

8- Train YOLO

```
[ ] import glob
    images_list = glob.glob("data/obj (1)/dataset/*.jpg")
    print(images_list)

    (1)/dataset/170.jpg', 'data/obj (1)/dataset/174.jpg', 'data/obj (1)/dataset/160.jpg', 'data/obj (1)/dataset/49.jpg', 'data/obj (1
```

8- Train YOLO

```
[ ] import glob
    images_list = glob.glob("data/obj (1)/dataset/*.jpg")
    print(images_list)

    (1)/dataset/170.jpg', 'data/obj (1)/dataset/174.jpg', 'data/obj (1)/dataset/160.jpg', 'data/obj (1)/dataset/49.jpg', 'data/obj (1)/dataset/161.jpg', 'data/ot

    #Create training.txt file
    file = open("data/train.txt", "w")
    file.write("\n".join(images_list))
    file.close()

[ ] !./darknet detector train data/obj.data cfg/yolov3_training.cfg darknet53.conv.74 -dont_show
     total_bbox = 4681, rewritten_bbox = 0.000000 %
    v3 (mse loss, Normalizer: (iou: 0.75, obj: 1.00, cls: 1.00) Region 82 Avg (IOU: 0.374910), count: 2, class_loss = 190.082169, iou_loss = 0.856583, total_los
    v3 (mse loss, Normalizer: (iou: 0.75, obj: 1.00, cls: 1.00) Region 94 Avg (IOU: 0.542572), count: 1, class_loss = 977.052979, iou_loss = 0.321594, total_los
    v3 (mse loss, Normalizer: (iou: 0.75, obj: 1.00, cls: 1.00) Region 106 Avg (IOU: 0.021367), count: 1, class_loss = 2277.900146, iou_loss = 4.470459, total_l
     total_bbox = 4685, rewritten_bbox = 0.000000 %
    v3 (mse loss, Normalizer: (iou: 0.75, obj: 1.00, cls: 1.00) Region 82 Avg (IOU: 0.525542), count: 2, class_loss = 191.296783, iou_loss = 0.537170, total_los
    v3 (mse loss, Normalizer: (iou: 0.75, obj: 1.00, cls: 1.00) Region 94 Avg (IOU: 0.146253), count: 1, class_loss = 976.882507, iou_loss = 2.148438, total_los
    v3 (mse loss, Normalizer: (iou: 0.75, obj: 1.00, cls: 1.00) Region 106 Avg (IOU: 0.233109), count: 1, class_loss = 2303.714355, iou_loss = 1.125732, total_l
     total_bbox = 4689, rewritten_bbox = 0.000000 %
    v3 (mse loss, Normalizer: (iou: 0.75, obj: 1.00, cls: 1.00) Region 82 Avg (IOU: 0.321660), count: 1, class_loss = 190.922485, iou_loss = 0.482544, total_los
    v3 (mse loss, Normalizer: (iou: 0.75, obj: 1.00, cls: 1.00) Region 94 Avg (IOU: 0.289607), count: 2, class_loss = 974.544983, iou_loss = 1.863647, total_los
    v3 (mse loss, Normalizer: (iou: 0.75, obj: 1.00, cls: 1.00) Region 106 Avg (IOU: 0.074548), count: 1, class_loss = 2315.402832, iou_loss = 2.652100, total_l
     total_bbox = 4693, rewritten_bbox = 0.000000 %
    v3 (mse loss, Normalizer: (iou: 0.75, obj: 1.00, cls: 1.00) Region 82 Avg (IOU: 0.062615), count: 1, class_loss = 190.187592, iou_loss = 2.235123, total_los
    v3 (mse loss, Normalizer: (iou: 0.75, obj: 1.00, cls: 1.00) Region 94 Avg (IOU: 0.261358), count: 1, class_loss = 976.396423, iou_loss = 1.065979, total_los
    v3 (mse loss, Normalizer: (iou: 0.75, obj: 1.00, cls: 1.00) Region 106 Avg (IOU: 0.134583), count: 1, class_loss = 2293.781738, iou_loss = 1.502197, total_l
     total_bbox = 4696, rewritten_bbox = 0.000000 %
    v3 (mse loss, Normalizer: (iou: 0.75, obj: 1.00, cls: 1.00) Region 82 Avg (IOU: 0.194134), count: 1, class_loss = 190.282852, iou_loss = 0.900909, total_los
    v3 (mse loss, Normalizer: (iou: 0.75, obj: 1.00, cls: 1.00) Region 94 Avg (IOU: 0.338841), count: 3, class_loss = 976.012695, iou_loss = 1.462708, total_los
    v3 (mse loss, Normalizer: (iou: 0.75, obj: 1.00, cls: 1.00) Region 106 Avg (IOU: 0.000000), count: 1, class_loss = 2291.547119, iou_loss = 0.000000, total_l
```

Fig 4.11 model training

The following graphs display the performance of our model's training and validation dataset.

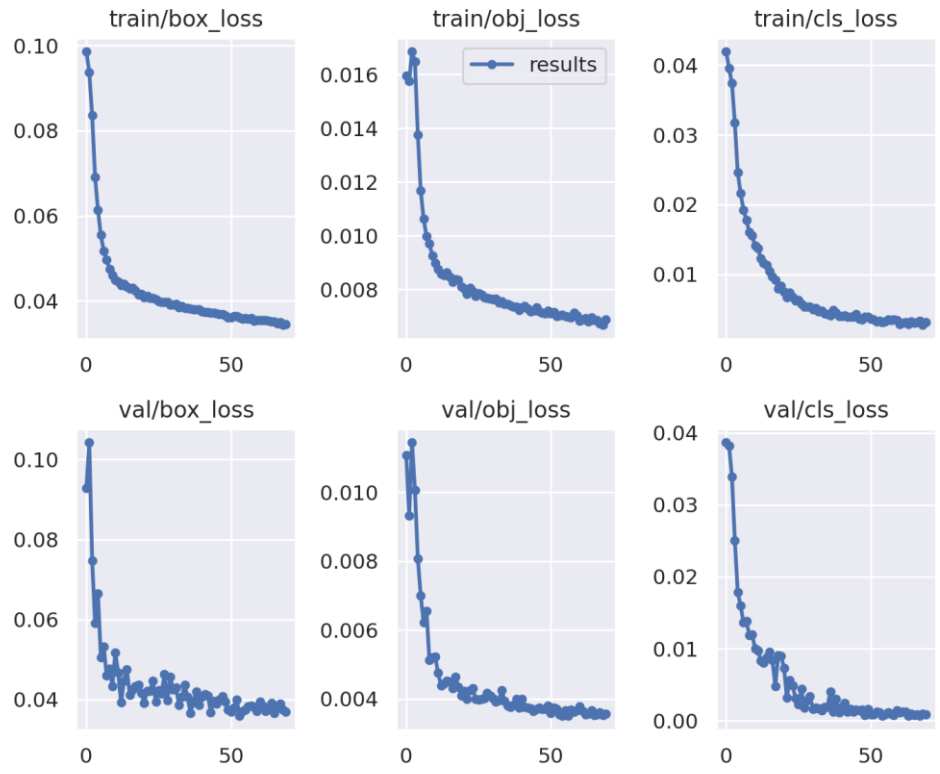


Fig 4.12 Loss metrics in YOLOv3

4.5 Model Testing

After training and validating our model we tested our model by using the test dataset that contains unknown weights.

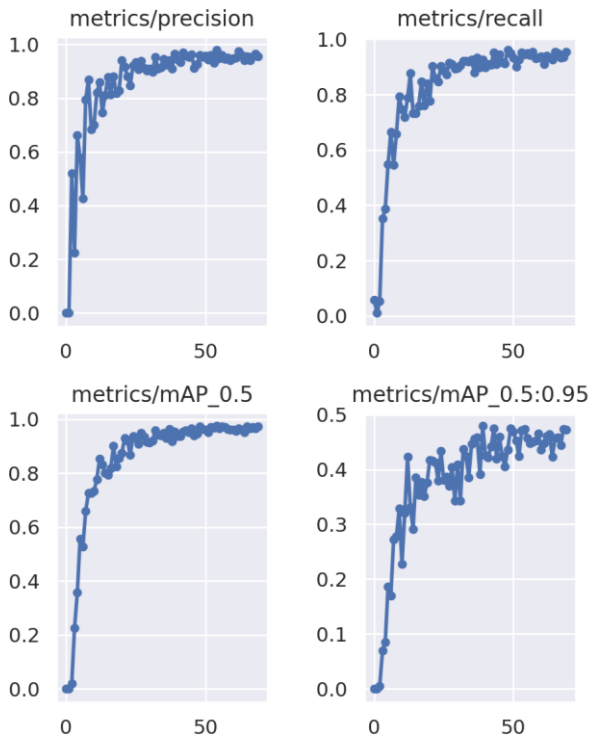


Fig 4.13 Analytic metrics during our testing phase

Metrics	Values
mAP	94.3%
Precision	95.9%
Recall	92.6%

Fig 4.3 Evaluation Metric Table

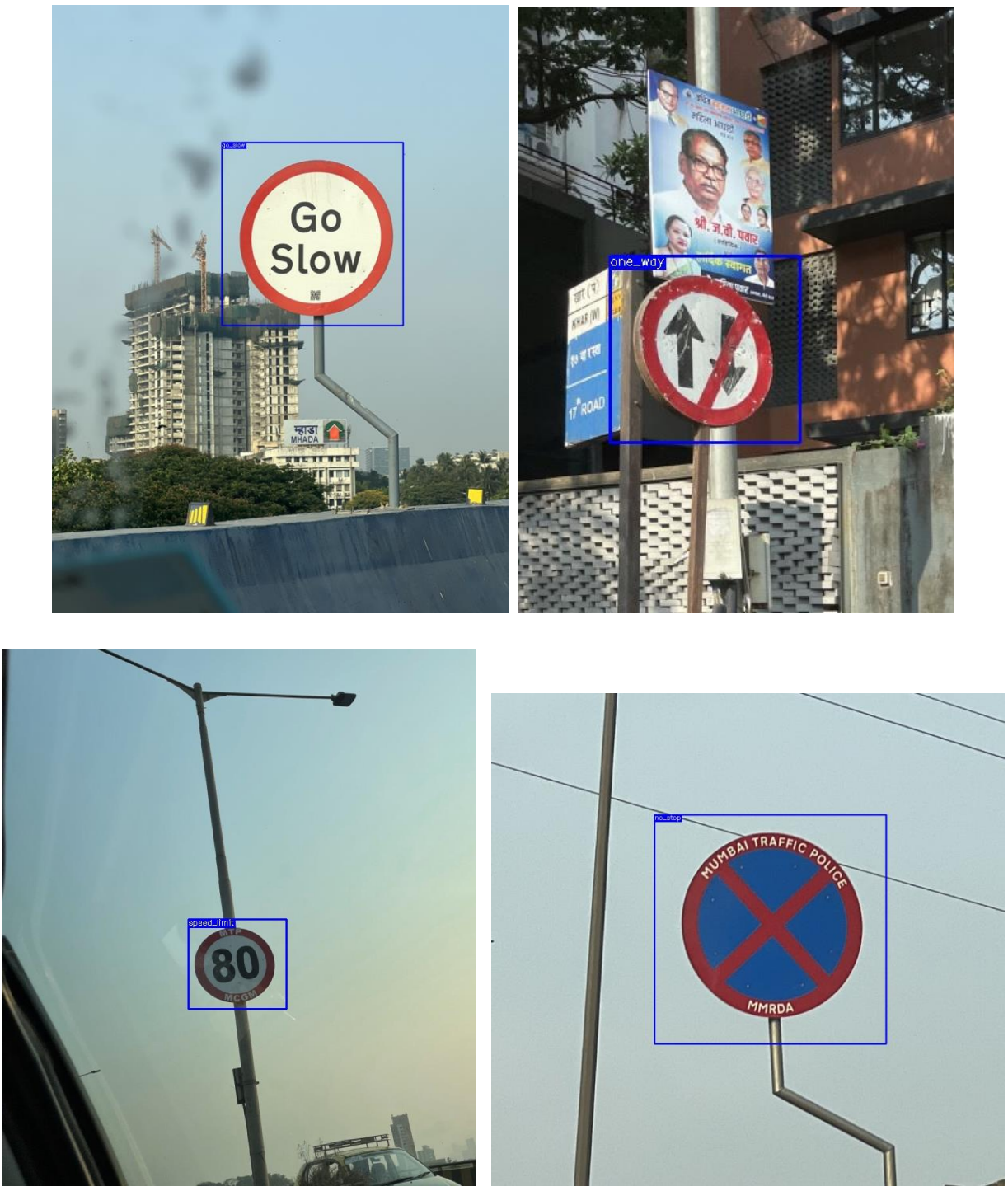


Fig 4.14 Recognition in testing phase

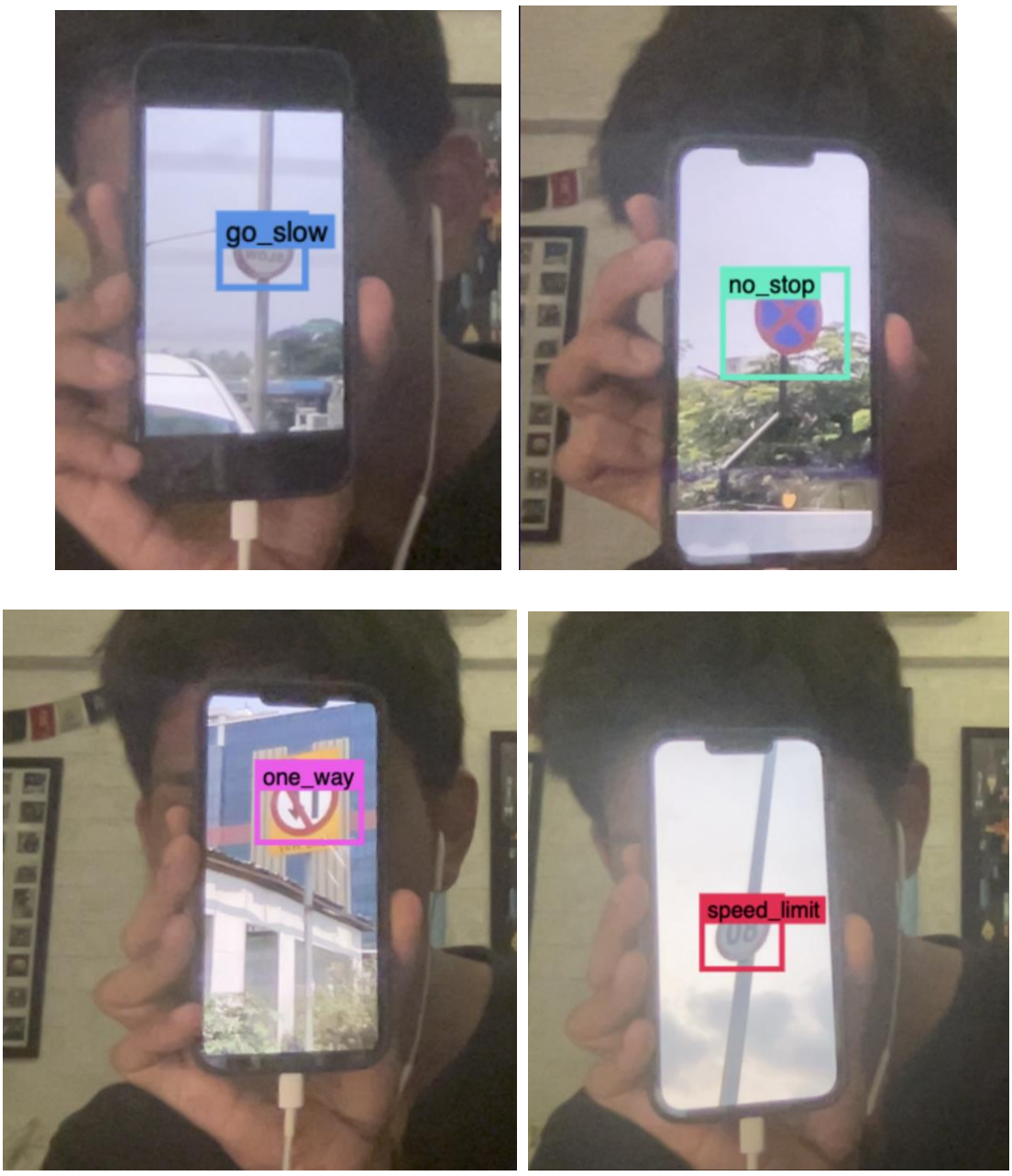


Fig 4.15 Screenshots of Live feed

4.6 Hardware implementation

After having tested our model we were able to recognise our respective traffic signs with a live feed, our next step was to add a hardware interface to our model.

Herein we used Raspberry Pi and an/a USB Camera/Pi Camera. Our hardware captures the live feed and helps us in the process of image detection and recognition.

We started with the OS installation on our Raspberry Pi and made sure our Camera was compatible with our version of Raspberry Pi which is version 4 and the OS version.

We will be using our trained models '.h5' file and using OpenCV we will read the live feed, convert it to a readable format, run it through our object detection model and display the bounding boxes over the appropriate traffic sign accurately.

Chapter 5

Conclusions and Further Work

This chapter presents a conclusion derived by us of the entire project and discusses the future scope and applications of our project.

5.1 Conclusions

We were able to conduct an inclusive study on the various kinds of Object Detection algorithms and the technologies around them for their real-time applications. Through our research, analysis, and experimentation, the YOLOv3 (You Only Look Once version 3) algorithm was the most suitable for our implementation of Traffic Sign Detection and Recognition since it can perform the learning part of the deep learning model at a faster rate with lesser computational resources as compared to its other counterparts.

For model training, we successfully created an original custom dataset consisting of various images of traffic signs around Mumbai. We transformed and prepared our dataset and trained our model. We fine-tuned the model to improve its accuracy after testing it on a live feed of images through our laptop camera. Finally, we implemented our code and model on Raspberry Pi for hardware implementation using a Pi Camera. Our model can detect and classify traffic signs from all the four categories selected by us and also detected multiple signs in a frame.

Our model during the test phase gave a precision of 95.9%, recall of 92.6%, and a mean average precision value of about 94.3%. Our model was also correctly able to detect and classify traffic signs through live demonstration further confirming our model's accuracy.

5.2 Scope for Further Work

- Our model is accurately detecting four classes of traffic signs. Our first scope of further work can be an inclusion of more traffic sign classes that exist on the roads of Mumbai.
- We can improve the accuracy and robustness of our model by further increasing the size of our dataset. This will help us give more precise weight values for the neural network in our model.
- We can improve the mAP values by fine-tuning our biases and weight values in this algorithm.
- We can make an alert system that can notify the driver about the upcoming traffic sign which it can recognize through our model.
- A traffic sign detection and recognition system can be deployed in self-driving cars.

Bibliography

[1] Computer Vision. (2017). *Communications In Computer And Information Science*. doi: 10.1007/978-981-10-7305-2

[2] Y. Sun, P. Ge and D. Liu, "Traffic Sign Detection and Recognition Based on Convolutional Neural Network," *2019 Chinese Automation Congress (CAC)*, 2019, pp. 2851-2854, doi: 10.1109/CAC48633.2019.8997240.

[3] Srivastava, S., Divekar, A.V., Anilkumar, C. *et al.* "Comparative analysis of deep learning image detection algorithms." *J Big Data* **8,** 66 (2021). https://doi.org/10.1186/s40537-021-00434-w

[4] Geethapriya. S, N. Duraimurugan, S.P. Chokkalingam et al. Real-Time Object Detection with Yolo (IJEAT) ISSN: 2249 – 8958, Volume-8, Issue-3S, February 2019

[5] W. -N. Mohd-Isa, M. -S. Abdullah, M. Sarzil, J. Abdullah, A. Ali and N. Hashim, "Detection of Malaysian Traffic Signs via Modified YOLOv3 Algorithm," *2020 International Conference on Data Analytics for Business and Industry: Way Towards a Sustainable Economy (ICDABI)*, 2020, pp. 1-5, doi: 10.1109/ICDABI51230.2020.9325690.

[6] Object Detection Guide | Fritz AI. (2022). Retrieved 5 May 2022, from https://www.fritz.ai/object-detection/

[7] (2022). Retrieved 5 May 2022, from https://hobby.4bwa.com/2021/06/28/heres-how-deep-learning-helps-computers-detect-objects/?cv=1

[8] Computer Vision. (2017). *Communications In Computer And Information Science*. doi: 10.1007/978-981-10-7305-2

[9] YOLOv3: Real-Time Object Detection Algorithm (What's New?) - viso.ai. (2022). Retrieved 5 May 2022, from https://viso.ai/deep-learning/yolov3-overview/#:~:text=YOLOv3%20(You%20Only%20Look%20Once%2C%20Version%203)%20is%20

[10] UNIVERSITI TEKNIKAL MALAYSIA MELAKA INSTITUTIONAL REPOSITORY from http://eprints.utem.edu.my/id/eprint/17571/

[11] Tejas Jnanesh Ghalsasi “A Survey on Real Time Object Detection using Voice Activated Smart IoT” e-ISSN: 2395-0056, p-ISSN: 2395-0072

Acknowledgments

We hereby take this opportunity to thank our college KJ Somaiya College of Engineering to provide us with the opportunity to work on this project. We are thankful of our Guide Mr. Maruti Zalte for giving guidance throughout the project. Your insights have helped us immensely during the tenure of our project. We would also like to thank Mrs Akshata Prabhu who provided her guidance during the starting phases of our project.

We would like to show gratitude towards the entire faculty and lab assistance for always helping us and making this project a success.

Thank you