REAL-TIME TRAFFIC VEHICLE DETECTION IN BANGLADESH USING YOLO

by

Mahmudul Hasan Udoy (160021081)
Andalib Rahman (160021101)
MD Abu Sayeed Zamee Chowdhury (160021123)

A Thesis Submitted to the Academic Faculty in Partial Fulfillment of the Requirements for the Degree of

BACHELOR OF SCIENCE IN ELECTRICAL AND ELECTRONIC ENGINEERING

Department of Electrical and Electronic Engineering
Islamic University of Technology (IUT)
Gazipur, Bangladesh

February 2021

REAL-TIME TRAFFIC VEHICLE DETECTION IN BANGLADESH USING YOLO

Approved by:

[illegible]

Dr. Golam Sarowar
Supervisor and Professor,
Department of Electrical and Electronic Engineering,
Islamic University of Technology (IUT),
Boardbazar, Gazipur-1704.

Date: ..18/03/2021

Table of Contents

List of Figures V
List of Tables VII
List of Acronyms VIII
Acknowledgements IX
Abstract X
Chapter 1: Introduction 1
1.1 VEHICLE DETECTION 1
1.2 OBJECT DETECTION VS. IMAGE CLASSIFICATION 1
1.3 REAL-TIME OBJECT DETECTION 2
Chapter 2: Literature Review 4
2.1 LITERATURE REVIEWS 4
2.2 RESEARCH OBJECTIVE 5
Chapter 3: How YOLO Works 7
3.1 FUNCTION OF THE YOLO FRAMEWORK 7
3.2 ENCODING BOUNDING BOXES 11
3.3 INTERSECTION OVER UNION AND NON-MAX SUPPRESSION 13
3.4 ANCHOR BOXES 16
Chapter 4: Research Methodology 20
4.1 ARCHITECTURE 20
4.2 BAG OF FREEBIES 22
4.3 BAG OF SPECIALS 23
4.4 DATASET 24
4.5 YOLO TRAINING PARAMETERS 26
4.5.1 YOLOv3 26
4.5.2 YOLOv4-tiny 27
4.5.3 YOLOv5 27
4.6 DARKNET REQUIREMENTS AND INSTALLATION 27
4.6.1 Requirements 28
4.6.2 Installation 28
4.7 PLATFORMS AND SYSTEM SPECIFICATIONS 29
4.7.1 Workstation 29
4.7.2 Cloud 29
4.8 TRAINING 30
4.8.1 YOLOv3 30
4.8.2 YOLOv4-tiny 30
4.8.3 YOLOv5 30
4.7 RESEARCH TIMELINE 34
4.9.1 Phase 1 34
4.9.2 Phase 2 35
4.9.3 Phase 3 35

4.9.4 Phase 4 35
Chapter 5: Results........ 37
5.1 INFERENCE ON IMAGES 37
5.2 INFERENCE ON VIDEOS 43
5.3 COMPARISON OF RESULTS 44
5.4 COMPARISON ANALYSIS 47
Chapter 6: Conclusion........ 48
6.1 SIGNIFICANCE OF THE PROJECT 48
6.2 FUTURE SCOPES 49
References........ 50

List of Figures

Figure 1.1: Classification vs Object Detection 2

Figure 1.2: R-CNN Architecture: Extract Regions, then Classify 3

Figure 1.3: YOLO Architecture: A Single CNN that Detects Objects 3

Figure 3.1: Input Image to YOLO 8

Figure 3.2: Bounding Boxes Drawn on the Input Image 8

Figure 3.3: Grid with no Object 9

Figure 3.4: Grid Containing a Car 10

Figure 3.5: YOLO Model for a 100x100 RGB Image 11

Figure 3.6: Bounding Box around a Car 11

Figure 3.7: Setting the Midpoint of a Bounding Box 12

Figure 3.8: Determining bx and by 12

Figure 3.9: Predicted Box and Ground Truth Box 13

Figure 3.10: Area of the intersection and Area of the union 14

Figure 3.11: Multiple Boxes for the Same Object 15

Figure 3.12: Selecting the Box with the Highest Score 15

Figure 3.13: Selecting the Next Box with the Highest Score and Taking IoU 16

Figure 3.14: Result after NMS 16

Figure 3.15: Drawing Grids on the Input Image 17

Figure 3.16: Two Objects in the Same Grid 17

Figure 3.17: Assigning Two Anchor Boxes 18

Figure 4.1: Architecture for YOLO 20

Figure 4.2: Different Classes of Vehicles of Dhaka City (1) 25

Figure 4.3: Different Classes of Vehicles of Dhaka City (2) 25

Figure 4.4: Dataset Health from Roboflow 26

Figure 4.5: Summary of Dataset and workstation 29

Figure 4.6: Loss vs Iterations for YOLOv3 Training 31

Figure 4.7: Loss vs Iterations for YOLOv4-tiny Training 32

Figure 4.8: Confusion Matrix for YOLOv5 33

Figure 4.9: Performance metrics vs Epochs for YOLOv5 33

Figure 4.10: Timeline of Research Progression 34

Figure 5.1: Yolov3 on Image 1 38

Figure 5.2: Yolov3 on Image 2 38

Figure 5.3: Yolov3 on Image 3 39

Figure 5.4: Yolov4-tiny on Image 1 40

Figure 5.5: Yolov4-tiny on Image 2 40

Figure 5.6: Yolov4-tiny on Image 3 41

Figure 5.7: Yolov5 on Image 1 42

Figure 5.8: Yolov5 on Image 2 42

Figure 5.9: Yolov5 on Image 3 43

Figure 5.10: Class detection performance between different YOLO frameworks 46

Figure 5.11: Speed of detection between different YOLO frameworks 46

List of Tables

Table 3.1: Output Vector 9

Table 3.2: Output Vector for no Detection 10

Table 3.3: Output Vector for Detection 10

Table 3.4: Output Vector for a Successful Detection 13

Table 3.5: Output Vector without Anchor Box.......... 18

Table 3.6: Output Vector with Two Anchor Boxes 18

Table 5.1: Comparison between the YOLO versions on Image 1 44

Table 5.2: Comparison between the YOLO versions on Image 3 44

Table 5.3: Comparison between the YOLO versions on Image 2 45

List of Acronyms

YOLO	You Only Look Once
IP	Internet Protocol
CNN	Convolutional Neural Network
R-CNN	Region Based Convolutional Neural Networks
FPS	Frames Per Second
GPU	Graphics Processing Unit
SVM	Support Vector Machine
mAP	mean Average Precision
UAV	Unmanned Aerial Vehicle
PC	Personal Computer
IoU	Intersection over Union
CPU	Central Processing Unit
VGG	Visual Geometry Group
FPN	Feature Pyramid Network
MSE	Mean Square Error
SPM	Spatial Pyramid Matching
SE	Squeeze and Excitation
SAM	Spatial Attention Module
ReLU	Rectified Linear Unit
NMS	Non-Maximum Suppression
CUDA	Compute Unified Device Architecture
OpenCV	Open Source Computer Vision
cuDNN	CUDA Deep Neural Network
CC	Communicating Cores
MSVC	Microsoft Visual C++
CMD	Command Prompt
RAM	Read Only Memory
SUV	Sport Utility Vehicles
MOOC	Massive Open Online Course
AI	Artificial Intelligence
QR	Quick Response

Acknowledgements

First and foremost, praise and glory to Allah Subhanahu wa Ta'ala, the Almighty, for without His endless stream of blessings this study would not have been possible.

It is a genuine pleasure to express our admiration and gratitude to our beloved Professor Dr. Golam Sarowar sir for giving us the opportunity to do this research. We are extremely grateful for his precious guidance and helpful advices throughout the whole work.

We would like to thank the other members of the thesis committee for facilitating this accomplishment and listening to our needs. We also wish to express our deep appreciation to our family members for their abundant love and support, and for their prayers that always kept us strongly motivated.

Abstract

In this thesis, the implementation of machine learning-based object detection algorithms in computer vision is discussed. The key focus is the detection and identification of different types of vehicles found in Bangladesh in real-time. Vehicle detection is one of the fundamental requirements for applications like traffic surveillance and autonomous cars. It is more challenging in Bangladesh because of the irregular traffic, numerous types of vehicles, and the lack of a healthy dataset. Different neural networks were trained using YOLOv3, YOLOv4-tiny, and YOLOv5 on the "Dhaka Traffic Detection Challenge Dataset" for the classification. Both YOLOv3 and v4-tiny were run on the Darknet framework, trained on a moderately powered computer. YOLOv5 ran on PyTorch and the model was trained on Google Colaboratory, a cloud-based platform. Codes were written in Python 3.6. Roboflow, an online-based computer vision application, was used to organize the dataset for training. For real-time detection, an IP camera was used. The camera is capable of streaming video wirelessly without significant lag. The results of different models are compared. YOLOv5 performed the best among the models and produced the most promising results. The model took not more than 50 milliseconds to process each frame of the video feed on our moderately powered workstation. Which is good enough to detect vehicles in real-time. It was also able to detect more vehicles accurately that the other two models. This research has great potential and can be considered a step forward towards smart traffic systems and autonomous vehicles in Bangladesh.

Chapter 1

Introduction

1.1 Vehicle Detection

With the recent developments in self-driving cars and smart traffic management systems, the importance of vehicle detection is ever rising. Vehicle detection is where we make the autonomous systems see and recognize traffic vehicles in order to take necessary decisions regarding vehicle management, allocation of resources, traffic control etc. based on the information obtained. Different methods have been used in this field such as using magnetometers [1], ultrasonic sensors [2], radar sensors [3], infrared optical sensors [4], microphone arrays [5] and such. But they can be very difficult to implement for applications of real time traffic detection and self-driving cars. The task is even more difficult for Bangladesh because of some of the unique vehicles and challenging traffic environment we have here. For example, it can be quite daunting to make any autonomous system understand the complex geometry of a Rickshaw and distinguish from that of a bicycle in a congested traffic environment in real time using conventional methods. To make the work easy and viable, in our research, we took the aid of machine learning based computer vision to train a neural network to detect and identify 21 classes of typical vehicles found in Bangladesh. To make the systems see vehicles in real time we chose the YOLO [6] architecture built on Darknet [7] and PyTorch [8] frameworks.

1.2 Object Detection vs. Image Classification

Object detection and image classification are two of the classical problems of computer vision and image processing. There is a difference between classification and object detection. In classification, we predict just the name of the object, usually using a CNN. But in object detection we also draw bounding boxes around the object to locate them within the image as demonstrated in Figure 1.1 [9]. This is more difficult because of multiple reasons. There could be multiple objects within the same image, so the output layer length cannot be a constant anymore.

A crude solution this problem would be to select different regions of interest from the image at first, and then use a CNN to classify the object within that region like R-CNN shown in Figure 1.2. But the problem is that the objects can appear in different locations and have different aspect ratios. Therefore, the algorithms needs to select a huge number of regions, requiring a lot of computational power.

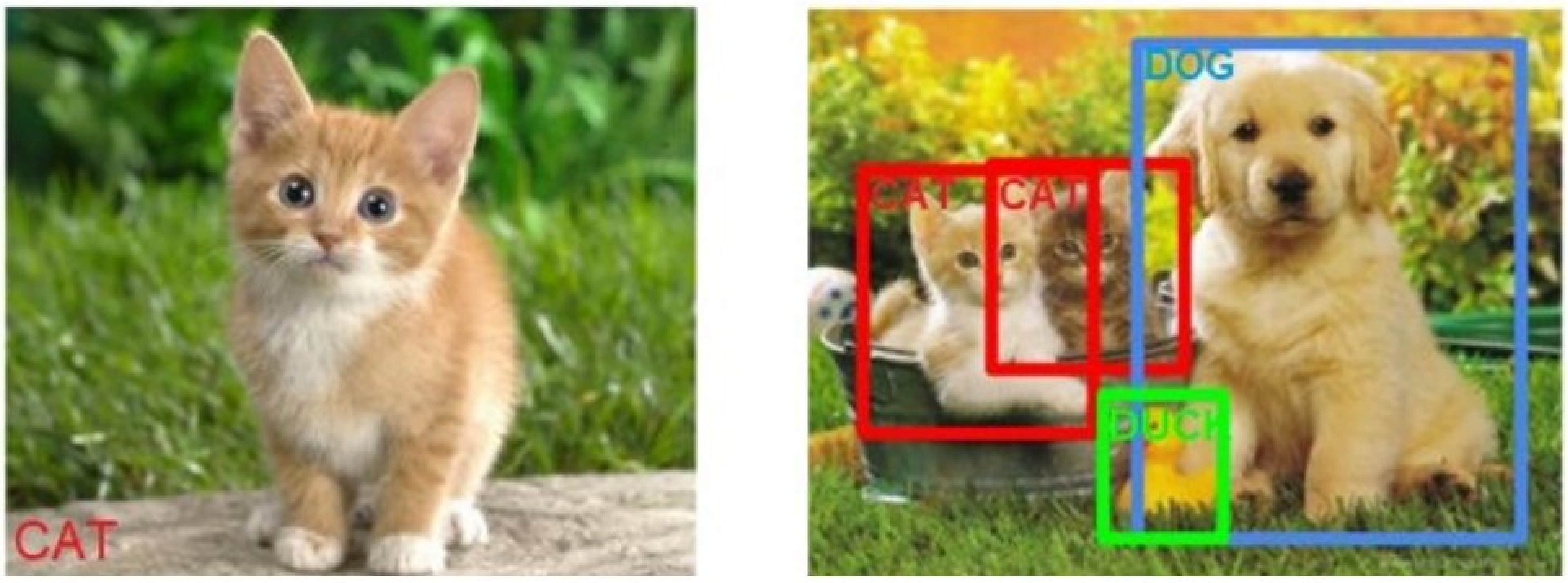


Figure 1.1: Classification vs Object Detection

1.3 Real-Time Object Detection

For our problem of vehicle detections, we need our system to perform object detection fast enough to be implemented in real time applications. The input to our detection system will not just be a single image but a live video feed from a camera which is actually many images captured within a small period of time. So the detection algorithm has to perform processing of a lot of images very fast. This possesses a big challenge and requires an efficient architecture along with machines of adequate computational power. Before YOLO, many machine learning based object detection architectures such as R-CNN [10], Fast R-CNN [11] and Faster R-CNN [12] tried to solve the issue of real-time processing and made improvements in their consecutive versions.

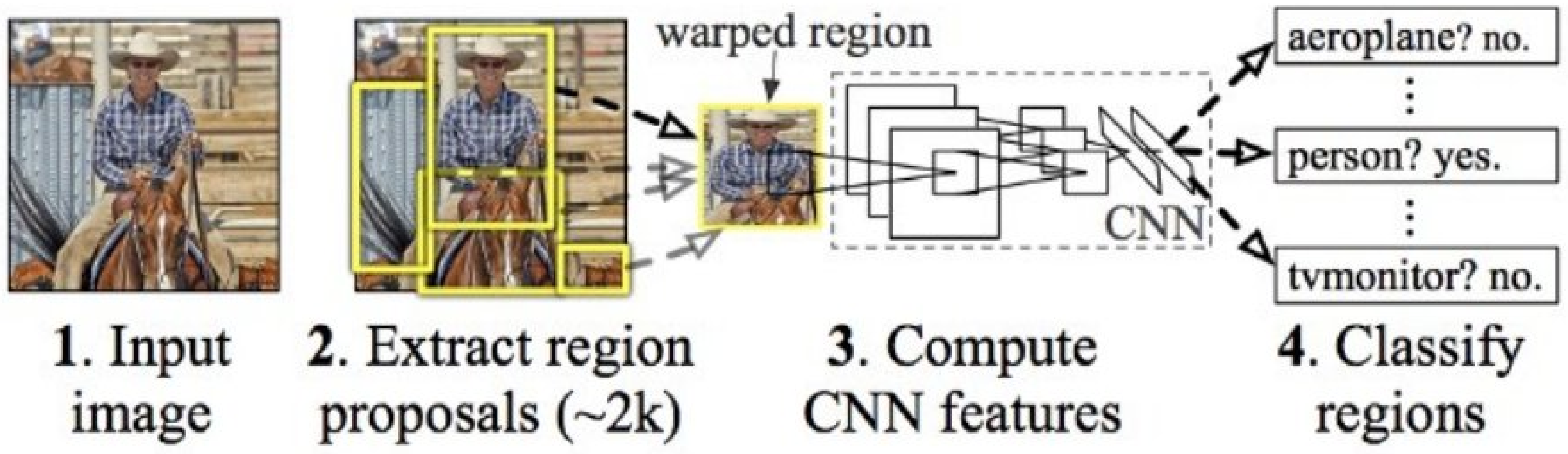


Figure 1.2: R-CNN Architecture: Extract Regions, then Classify

But among all of them YOLO is the fastest in both training time and inference time. That is why for our research of detecting vehicles in real-time, we chose YOLO. YOLO is orders of magnitude faster (45 FPS) than other object detection algorithms [13]. Because YOLO used one single convolutional network that predicts the bounding boxes and the class probabilities for these boxes as demonstrated in Figure 1.3.

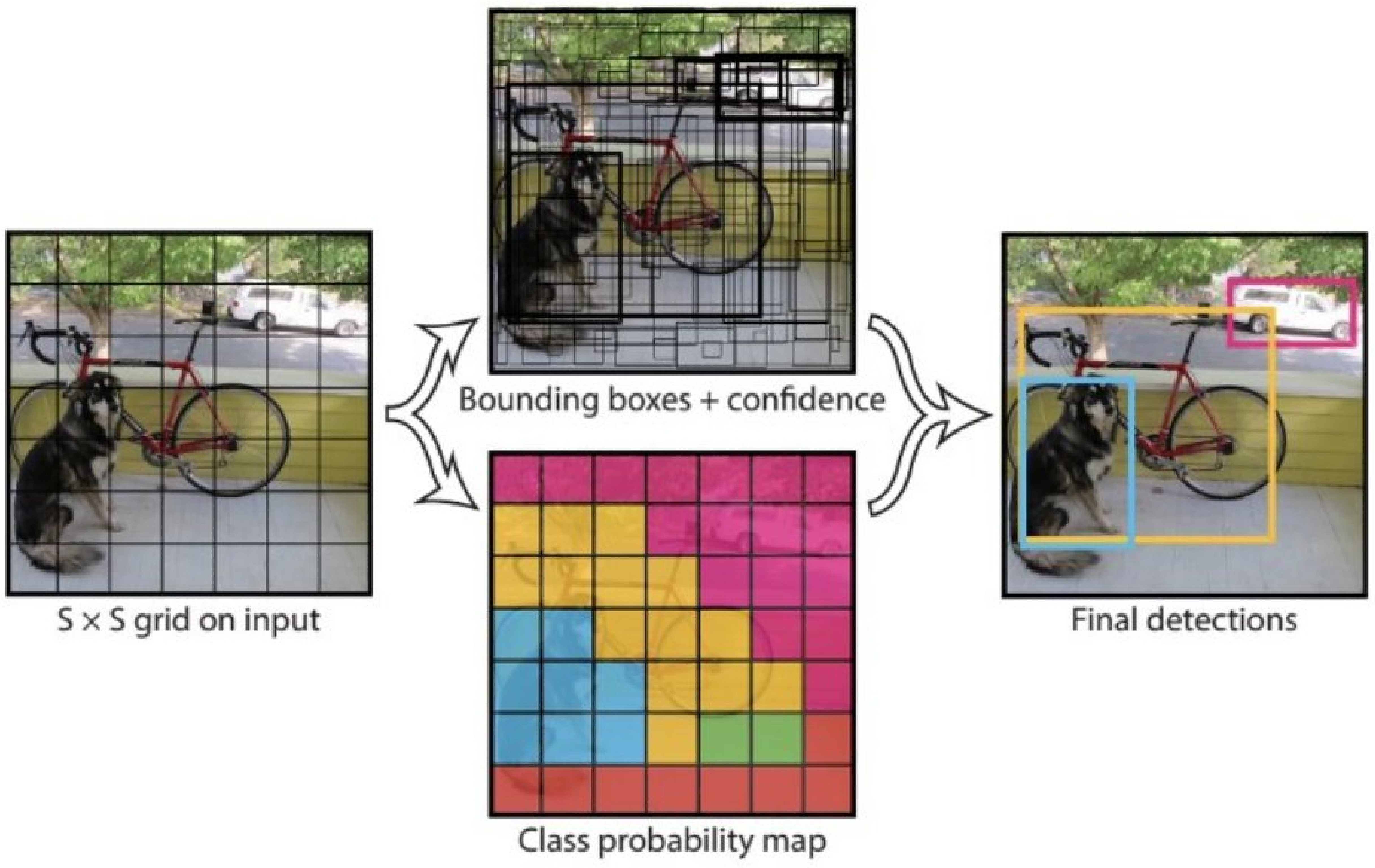


Figure 1.3: YOLO Architecture: A Single CNN that Detects Objects

Chapter 2

2.1 Literature Review

With the advances of deep learning and convolutional neural network, in the past decade object detection has reached a significant progress. Computer vision have witnessed an impressive improvement for higher accuracy, instance segmentation and object recognition. Also with the evolution of Graphic Processing Unit (GPU), real-time object detection becomes for faster and realizable. Here we will investigate some great works based on object detection.

1. A comparison was proposed by Bilel at el. [14] for vehicle detection between Faster R-CNN and YOLOv3. It was based on precision, F1 score, recall, quality and processing time. The study shows that YOLO out performs R_CNN for most of the pictures.

2. Hou-Ning at el. [15] suggest an online network architecture from a series of picture to track and detect vehicles. Still image of different angle is used for this purpose which extend the working load for detecting vehicles.

3. A bidirectional cooperation between recognition and tracking is analysed by Foresti at el. [16], these assigns a semantic levels, establishing identity and pose correspondence between objects detected at various time instants.

4. Zehang Sun at el. [17] explore multi-scale driven hypothesis generation and appearance based hypothesis verification to present an in-vehicle real-time monocular precrash vehicle detection system. Haar Wavelet decomposition for feature extraction and Support Vector Machines (SVMs) for classification were used here for appearance-based hypothesis verification.

5. In another research paper [18], they used same methodology to detect vehicle based on on-road video where the camera was mounted on the vehicle itself.

6. Based on YOLOv2 Jun Sang at el. [19] proposed their research for vehicle detection where k-means++ clustering algorithm was used to cluster the vehicle bounding boxes with 6 anchor boxes of different sizes. Their results shows that the mean Average Precision (mAP) could reach 94.78%.

7. Vehicle detection is performed image from UAV, based on optimal dense YOLO method by Zhi Xu at el. [20]. This research greatly beneficial for detection of small targets and designed for the characteristics of vehicle targets.

8. Real-time detection is proposed by Shaobin Chen at el. [21] for embedded system and Yolo v3-live was used as the algorithm which reduced the complexity of computing of the embedded operating devices.

9. Xiangwu Ding at el. [22] applied Yolo v3 to parking spaces and to detect vehicle in parking lots. Four different scale feature maps for object detection is used to extract deep vehicle parking space features which reduced the missed detection rate.

2.2 Research Objective

The main objective of our research work is to familiarize Bangladeshi vehicles with autonomous systems related to traffic management and self-driving cars. Some of the unique Bangladeshi vehicles are not recognized by the already existing autonomous cars and traffic management system. Bangladesh needs to have its own vehicle detection system where all of the vehicles of unique characteristics are recognized. The objectives of our thesis so far are:

1. Detecting Bangladeshi Vehicles in Challenging Environments: Our primary goal is to develop a system which can see some of the unique vehicles of Bangladesh.

2. Using Computer Vision for Vehicle Detection: We wanted a machine learning based computer vision algorithm, YOLO, for detecting various classes of vehicles.

3. Real-Time Detection: Detecting road vehicles in real-time for self-driving cars is a prerequisite. We wanted to make a real-time detection system.

4. Using Models That Are Trainable in Cloud and Local Computer: As we are resorting to a machine learning based detection architecture, we wanted to train our model in both cloud platforms and in our local PCs. Cloud platforms such as Google Colaboratory [23] offered better computational power than the hardware that were available to us. It reduced the training time to a significant extent.

5. Comparison between Different Versions of YOLO: We ran training on the dataset in different versions of YOLO: YOLOv3 [24] [25], YOLOv4-tiny [26] [27] and YOLOv5 [28] and the goal was to observe and evaluate their performance in both quantitative and qualitative parameters.

Chapter 3

How YOLO Works

Object detection being one of the classical problems in computer vision where algorithms are applied to recognize what and where, specifically what objects are inside a given image, means the detection of objects and also where they are in the image.

YOLO was introduced on the computer vision scene with the seminal 2015 paper by Joseph Redmon et al. “You Only Look Once: Unified, Real-Time Object Detection” [29] and immediately got a lot of popularity within fellow computer vision researchers.

YOLO is one of the useful and popular computer vision algorithms because it achieves high accuracy while also being able to run in real-time. The algorithm “only looks once” at the image in the sense that it requires only one forward propagation pass through the neural network to make predictions instead of taking huge numbers of region of interests. After non-max suppression (which makes sure the object detection algorithm only detects each object once), it then outputs recognized objects together with the bounding boxes. Benefits of YOLO over other image classifiers:

- YOLO is extremely fast.
- YOLO sees the entire image during training and test time instead of region of interest, so it implicitly encodes contextual information about the classes as well as their appearance.
- YOLO can be used for generalizable representations of objects so that when trained on natural images and tested on artwork, the algorithm outperforms other top detection methods.

3.1 Function of the YOLO Framework

In this section we will describe the step-by-step function of YOLO.

1. YOLO first takes an input image (Figure 3.1):

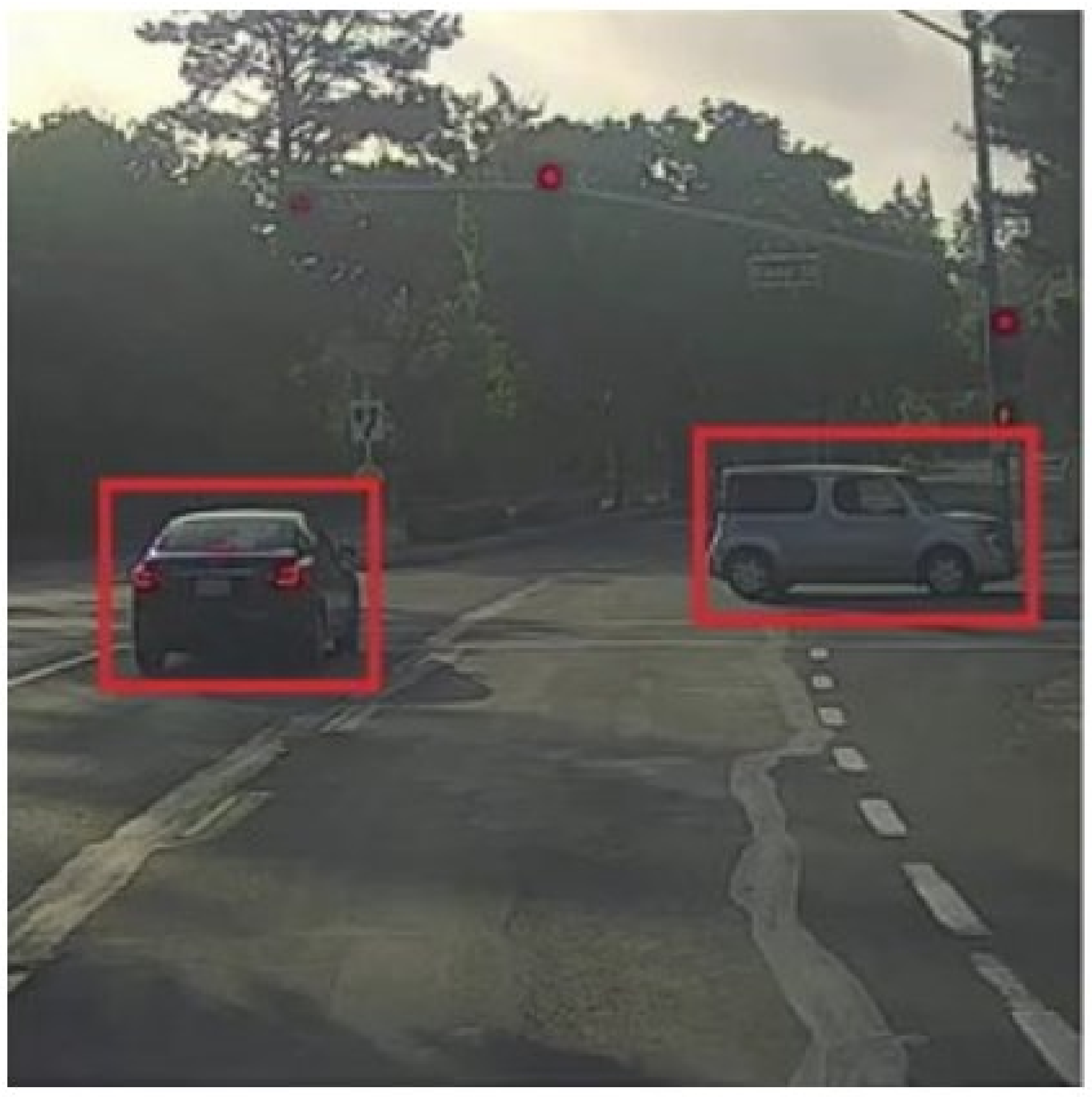


Figure 3.1: Input Image to YOLO

2. The algorithm then splits the image into cells, typically 19x19 grid instead of searching for interested regions in the input image that could contain an object. For our understanding lets divide the image into 3x3 grid (Figure 3.2).

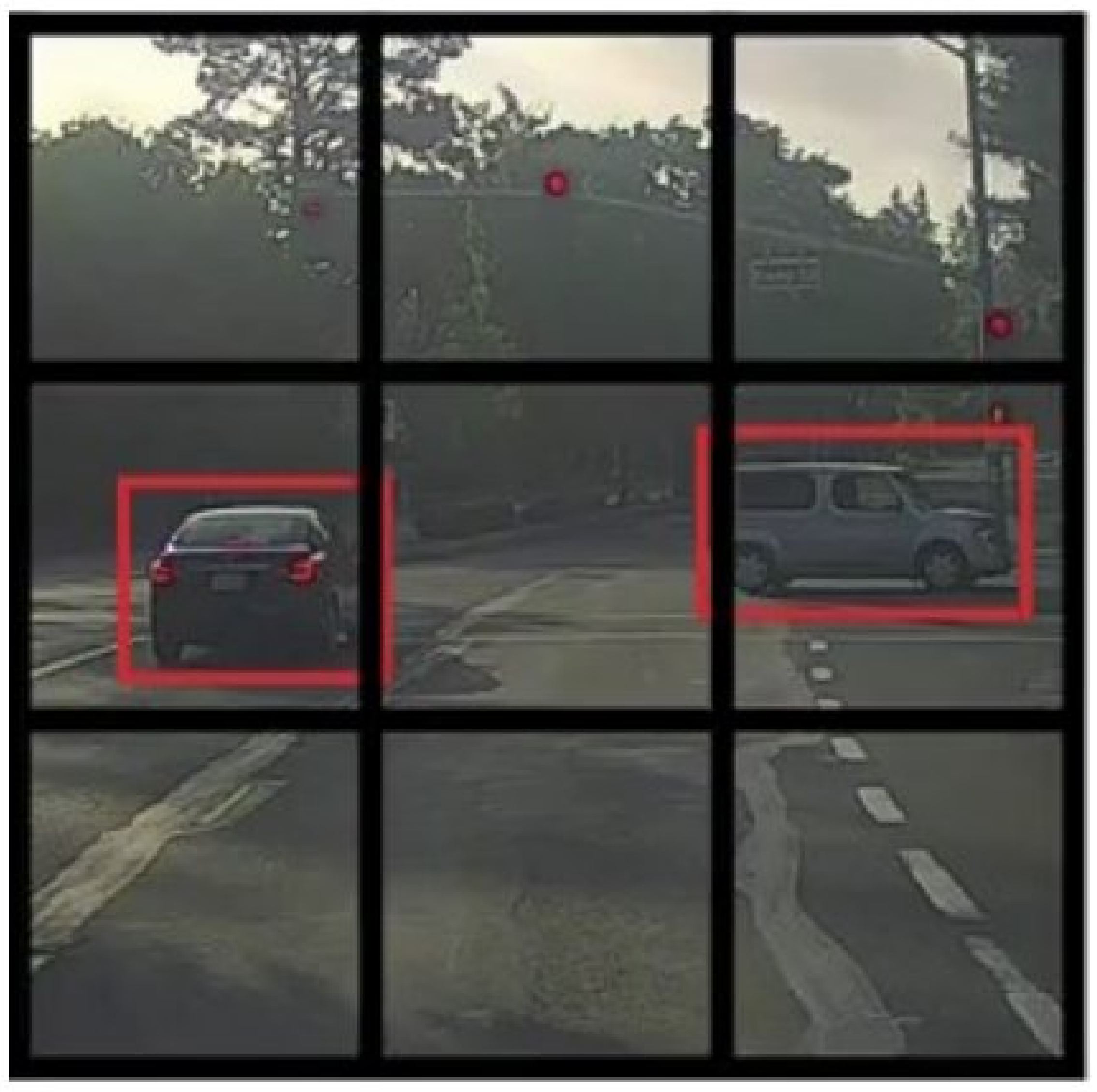


Figure 3.2: Bounding Boxes Drawn on the Input Image

3. Image classification and localization are applied on each grid. YOLO then predicts the bounding boxes and their corresponding class probabilities for objects if there is any object. The equation for certain class probability is,

$$\mathrm{score}_{c,i} = p_c * c_i$$

Then we pass the labelled data to the model in order to train it. Suppose we have divided the image into a grid of size 3 X 3 and there are a total of 3 classes which we want the objects to be classified into. Let's assume classes are Pedestrian, Car, and Motorcycle respectively. So, for each grid cell, the label y will be an eight dimensional vector shown in Table 3.1:

Table 3.1: Output Vector

y =	pc
	bx
	by
	bh
	bw
	c1
	c2
	c3

Here,

- pc defines whether an object is present in the grid or not in probability.
- bx, by, bh, bw specify the bounding box if there is an object.
- c1, c2, c3 represent the classes. So, if the object is a car, c2 will be 1 and c1 & c3 will be 0, and so on.

Let's select the first grid from the image in Figure 3.2 (Figure 3.3):

Figure 3.3: Grid with no Object

Since there is no object in this grid, pc will be zero and the y label for this grid will be that of Table 3.2.

Table 3.2: Output Vector for no Detection

y =	0
	?
	?
	?
	?
	?
	?
	?

In Table 3.2, '?' means that it doesn't matter what bx, by, bh, bw, c1, c2, and c3 contain as there is no object in the grid. Let's take another grid in which we have a car (c2 = 1) as shown in Figure 3.4.

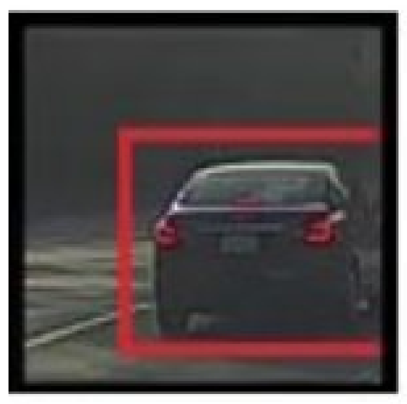


Figure 3.4: Grid Containing a Car

Let us first describe how YOLO decides whether there actually is an object in the grid. In the above image, there are two objects (two cars), so YOLO will take the mid-point of these two objects and these objects will be assigned to the grid which contains the mid-point of these objects. The y label for the center left grid with the car will be like shown in Table 3.3.

Table 3.3: Output Vector for Detection

y =	1
	bx
	by
	bh
	bw
	0
	1
	0

Since there is an object in this grid, pc will be equal to 1. bx, by, bh, bw will be calculated relative to the particular grid cell we are dealing with. Since car is the second class, c2 = 1 and c1 and c3 = 0. So, for each of the 9 grids, we will have an eight dimensional output vector. This output will have a shape of 3 X 3 X 8.

So now we have an input image and its corresponding target vector. Using the above example (input image – 100 X 100 X 3, output – 3 X 3 X 8), our model will be trained as shown in Figure 3.5.

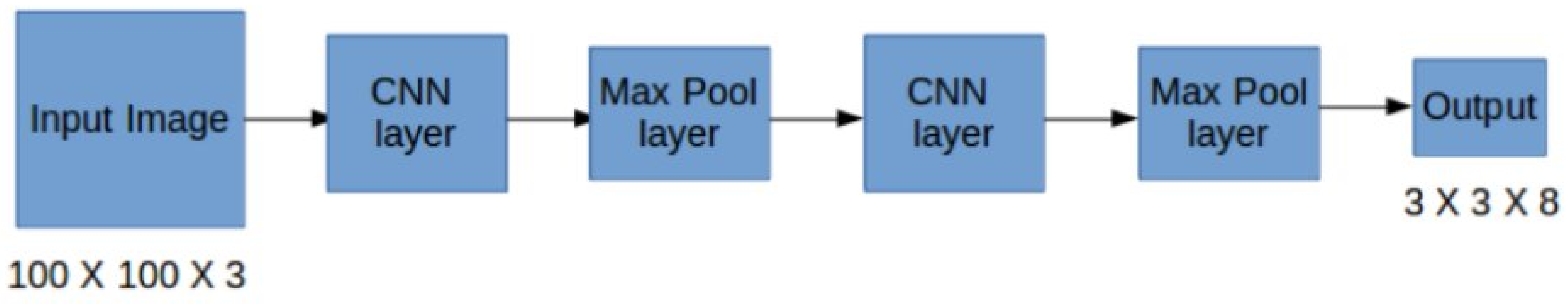


Figure 3.5: YOLO Model for a 100x100 RGB Image

Practically we use higher order grid (19x19 grid) to train the model. Even if an object spans out to more than one grid, it will only be assigned to a single grid in which its mid-point is located. We can reduce the chances of multiple objects appearing in the same grid cell by increasing the number of grids.

3.2 Encoding Bounding Boxes

bx, by, bh, and bw are calculated relative to the grid cell we are dealing with. Let us consider the center-right grid which contains a car as shown in Figure 3.6.

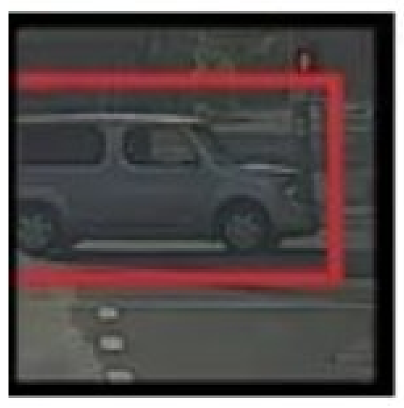


Figure 3.6: Bounding Box around a Car

bx, by, bh, and bw will be calculated relative to this grid only. We can write the y label as,

- pc = 1 since there is an object in this grid

- c2 = 1 since it is a car

To decide bx, by, bh and bw, YOLO assigned the coordinates to all the grids as indicated in Figure 3.7.

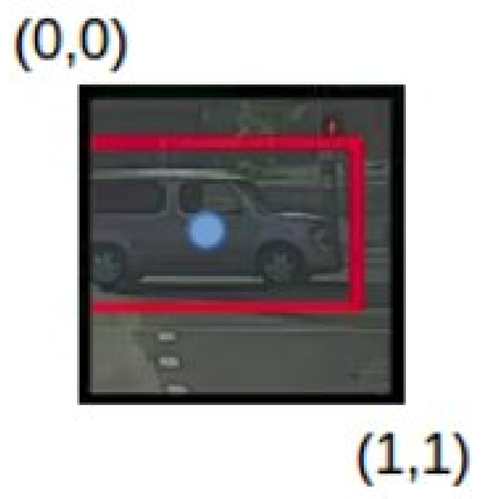


Figure 3.7: Setting the Midpoint of a Bounding Box

bx, by are the x and y coordinates of the midpoint of the object with respect to this grid. In this case, it will be (around) bx = 0.4 and by = 0.3 as shown in Figure 3.8.

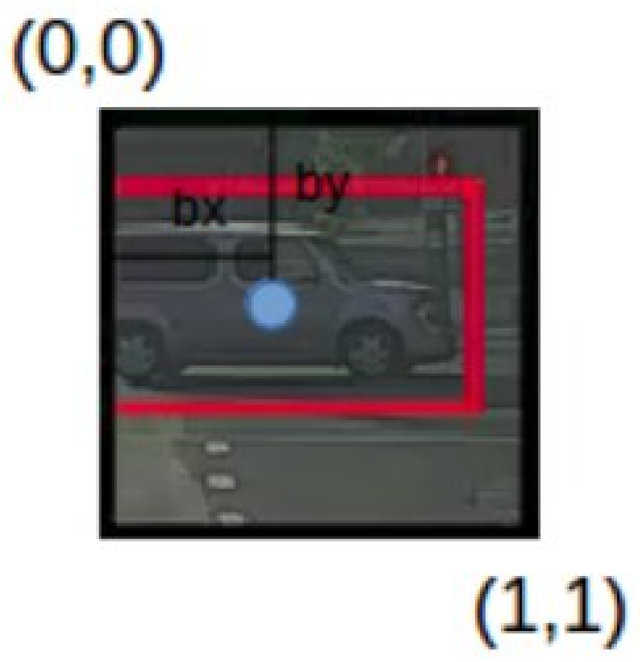


Figure 3.8: Determining bx and by

bh is the ratio of the height of the bounding box (red box in the above example) to the height of the corresponding grid cell, which in our case is around 0.9. So, bh = 0.9. bw is the ratio of the width of the bounding box to the width of the grid cell. So, bw = 0.5 (approximately). The y label for this grid will be like shown in Table 3.4.

Table 3.4: Output Vector for a Successful Detection

y =	1
	0.4
	0.3
	0.9
	0.5
	0
	1
	0

As the midpoint will always lie within the grid, bx and by will always range between 0 and 1. Whereas bh and bw can be more than 1 in case the dimensions of the bounding box are more than the dimension of the grid.

3.3 Intersection over Union and Non-Max Suppression

Intersection over Union:

For a single object in an image there can be multiple bounding boxes. These bounding boxes are produced from the grids next to the grid that contains the mid-point of the object. We need to know which bounding box is giving good outcome or predicting it correctly and eliminate the other bounding boxes. To eliminate the unnecessary bounding boxes, comes the idea Intersection over Union. Here intersection over union of the actual bounding box and predicted bounding boxes are calculated. For example consider the image in Figure 3.9 containing the actual and predicted boxes for a car:

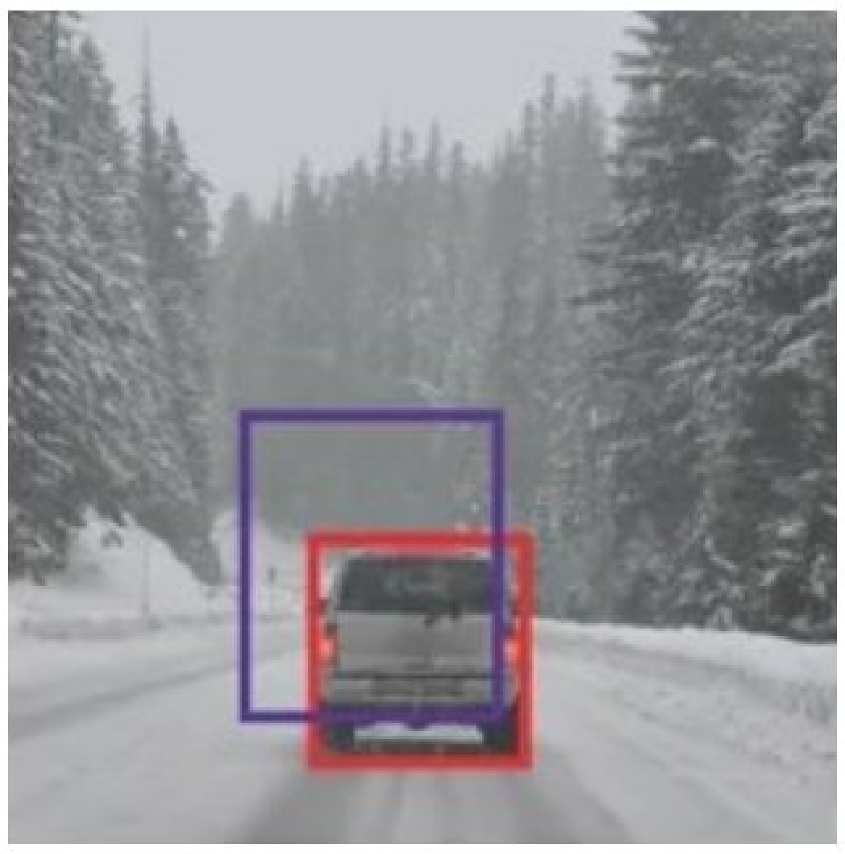


Figure 3.9: Predicted Box and Ground Truth Box

Here in Figure 3.10 the blue box is predicted and the red one is actual bounding or grand truth box. IoU is calculated for this boxes to decide the whether the prediction is good or not.

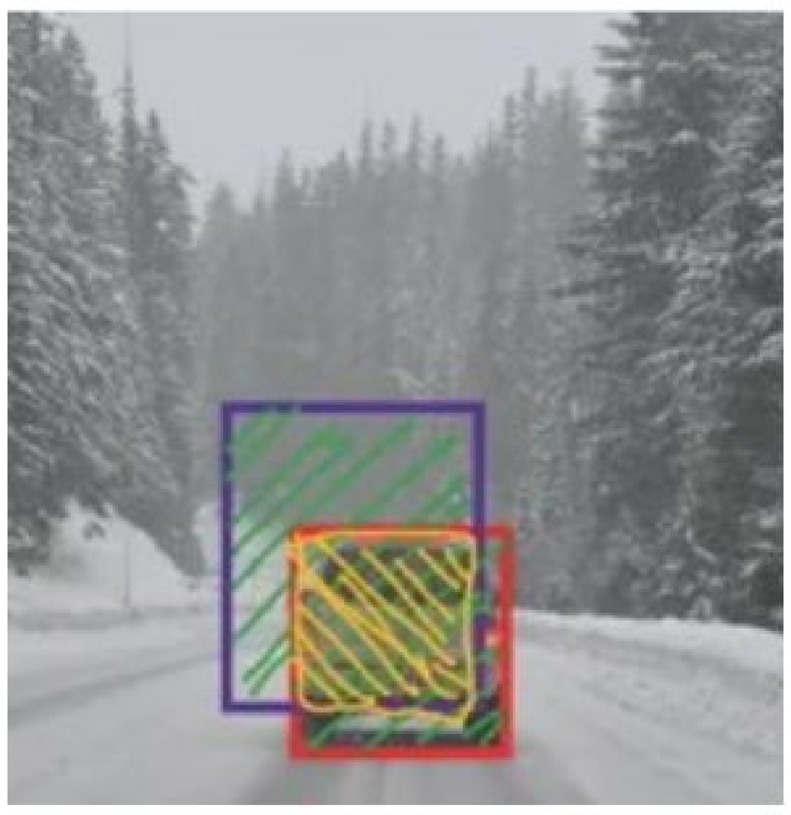


Figure 3.10: Area of the intersection and Area of the union

IoU = Area of the intersection / Area of the union

For the image in Figure 3.10,

IoU = Area of the yellow box / Area of green box

The value of IoU is preset to predict the bounding box. If the IoU is greater than threshold value, we can say that the prediction is good enough. Intuitively, the more increased threshold value, the better the prediction become.

Apart from IoU, there are also some technique which improve the performance of YOLO significantly, Non-max suppression is one of these.

Non-Max Suppression:

The most common problem for object detection is, a single object can be detected multiple times rather than once. Hence there can be more than one bounding boxes. The image below will clarify it,

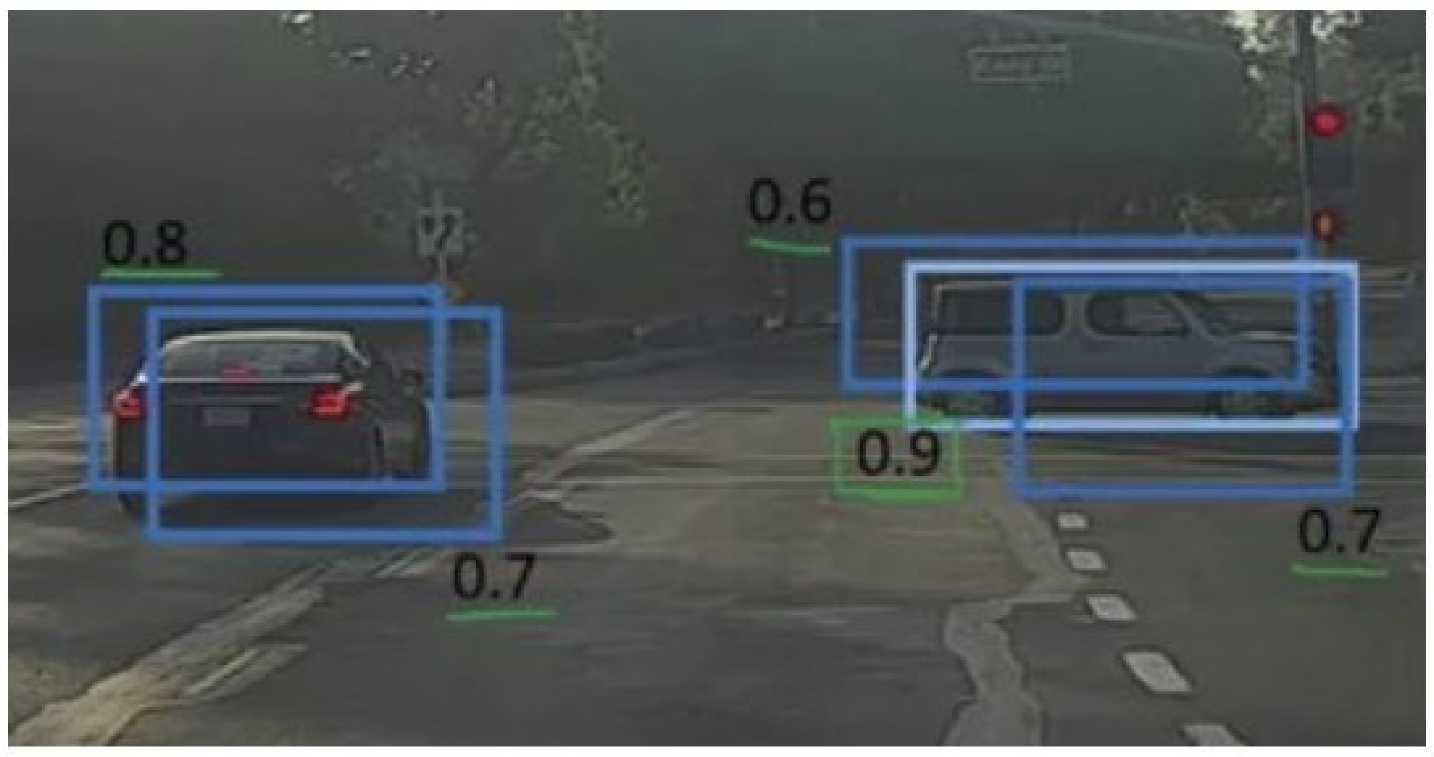


Figure 3.11: Multiple Boxes for the Same Object

Here an approach is taken to discard the additional bounding boxes for a single car like shown in Figure 3.11. Let's see how it works,

1. At first largest probability associated with the detection is taken care of. For above image 0.9 probability will be selected first.

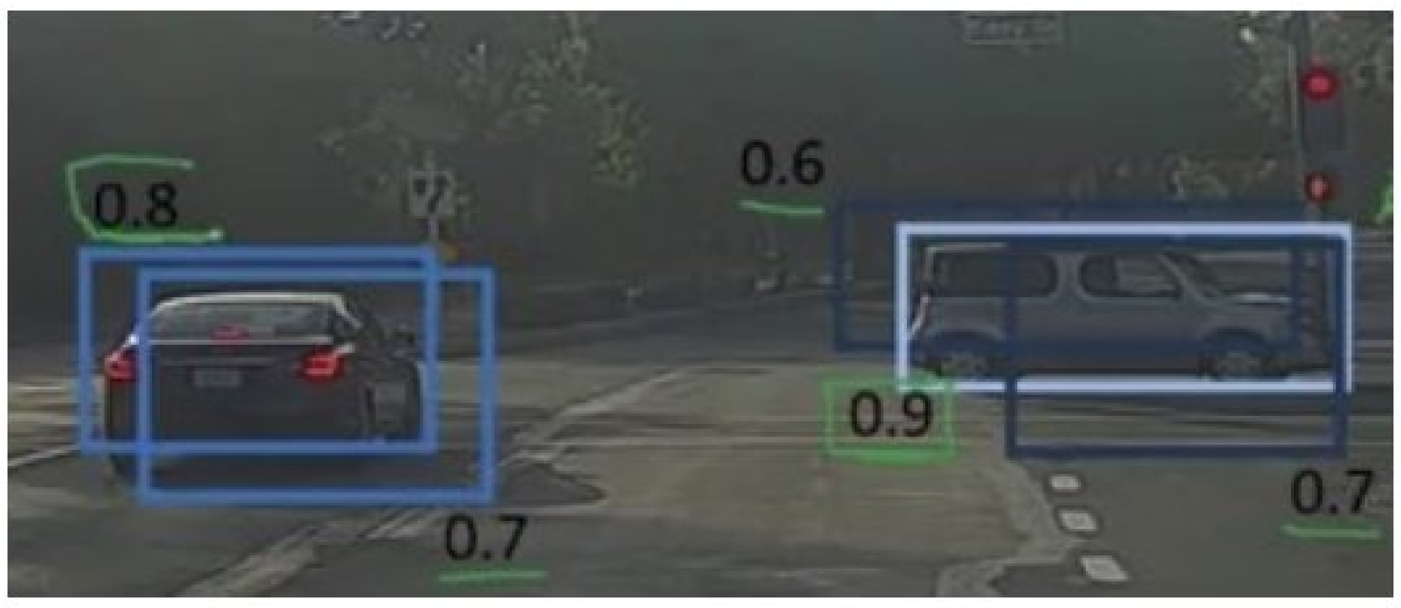


Figure 3.12: Selecting the Box with the Highest Score

2. Now all other boxes in the image are compared with the current box. The boxes having high IoU with the current box is suppressed. So, as shown in Figure 3.12, 0.6 and 0.7 probabilities will be eliminated.
3. After suppression, algorithm then select the next box with highest probability. Which is 0.8 in our image.

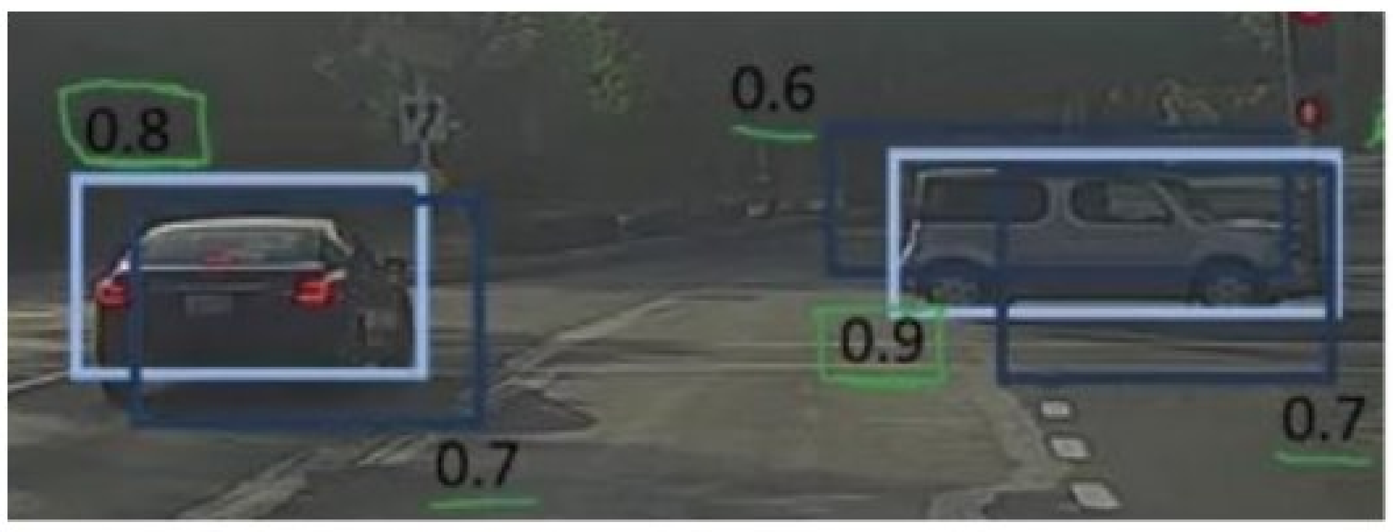


Figure 3.6: Selecting the Next Box with the Highest Score and Taking IoU

4. Again, IoU is compared with the remaining boxes and compress the boxes with high IoU (Figure 3.13).
5. These steps are further repeated until all the boxes have either selected or compressed and we get the final bounding boxes (Figure 3.14).

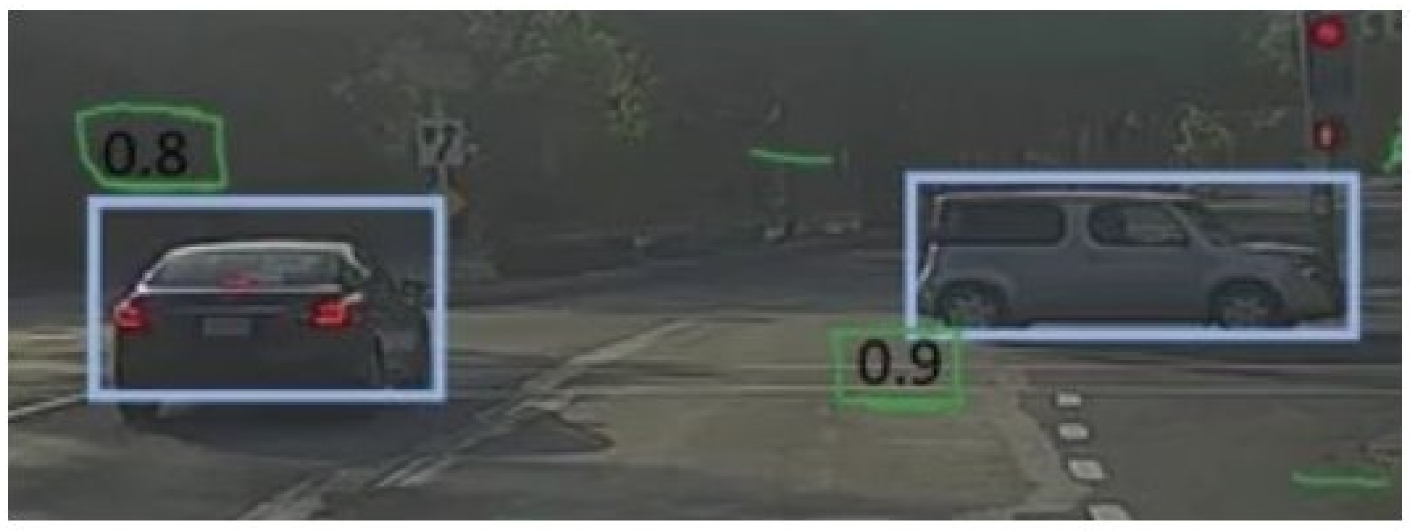


Figure 3.7: Result after NMS

3.4 Anchor Boxes

Practically there can be more than one object in a single grid. Till now we have seen a grid contain one object and its corresponding y-level output. The concept of anchor boxes comes when a single grid contains multiple objects. Consider the image in Figure 3.15 which is divided into 3x3 grid.

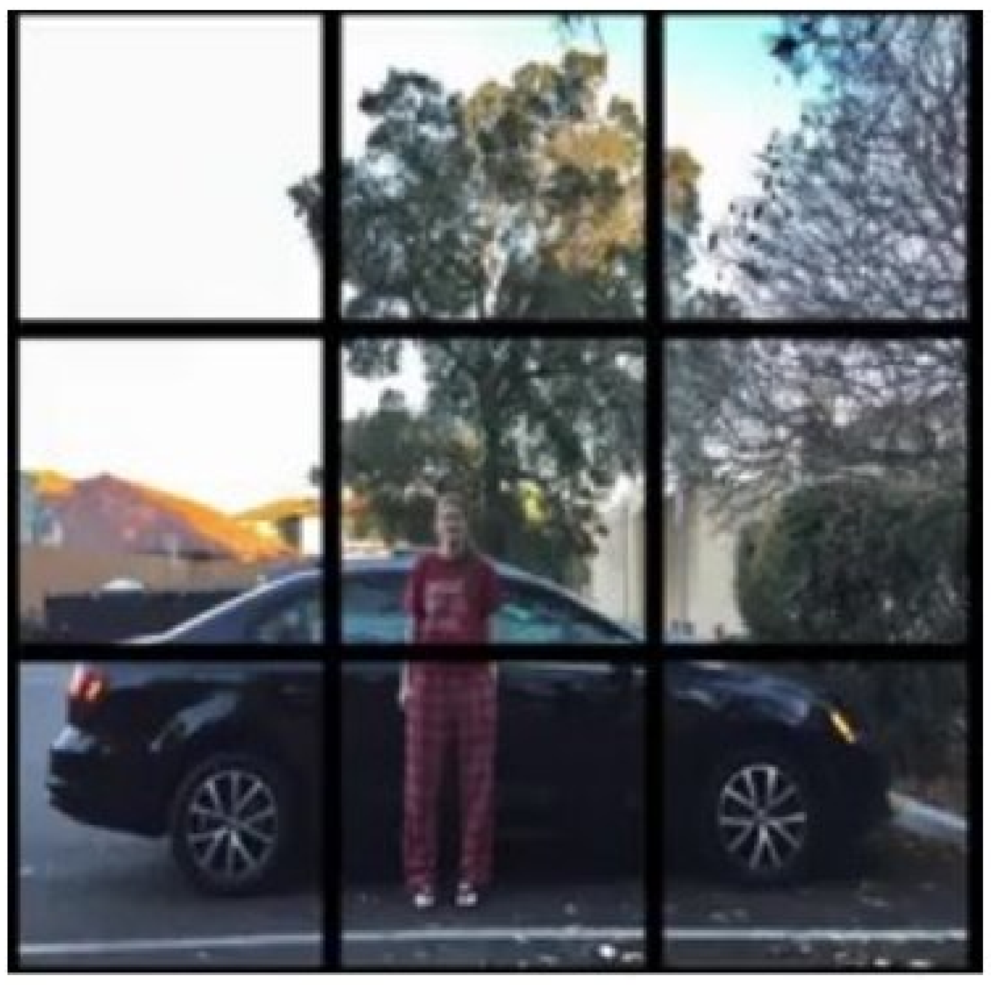


Figure 3.15: Drawing Grids on the Input Image

As we can remember, to assign an object to a grid by taking the midpoint of the object and based on its location. For the above image midpoint of the two object lies in the same grid. The bounding boxes of these object will be looked like as shown in Figure 3.16.

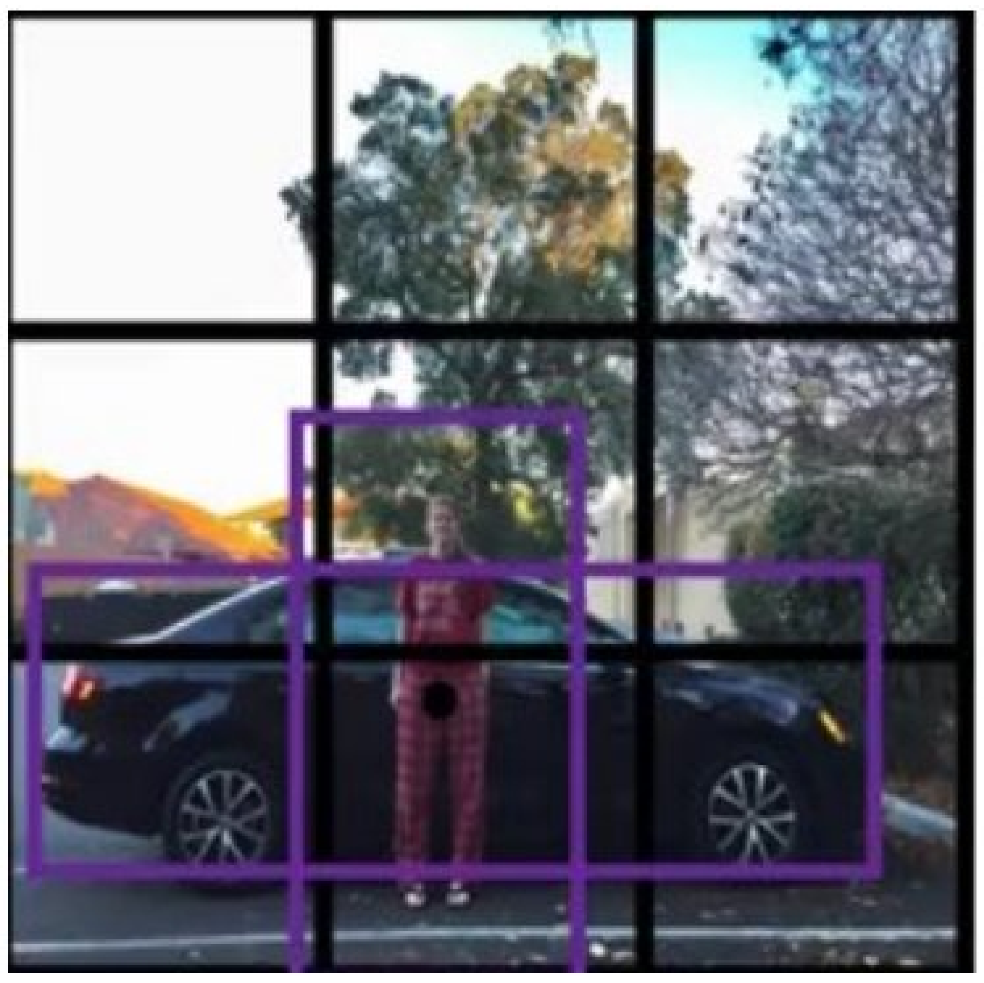


Figure 3.16: Two Objects in the Same Grid

According to our previous concept we will be only getting one of the two boxes, either for the car or the person. But using the idea of anchor we might get the information for both the boxes. To do this, first we pre-define two different shapes called anchor boxes or anchor boxes shapes. Now, for each grid we will get two outputs instead of one output. Anchor box number increase as the number of class increase (Figure 3.17).

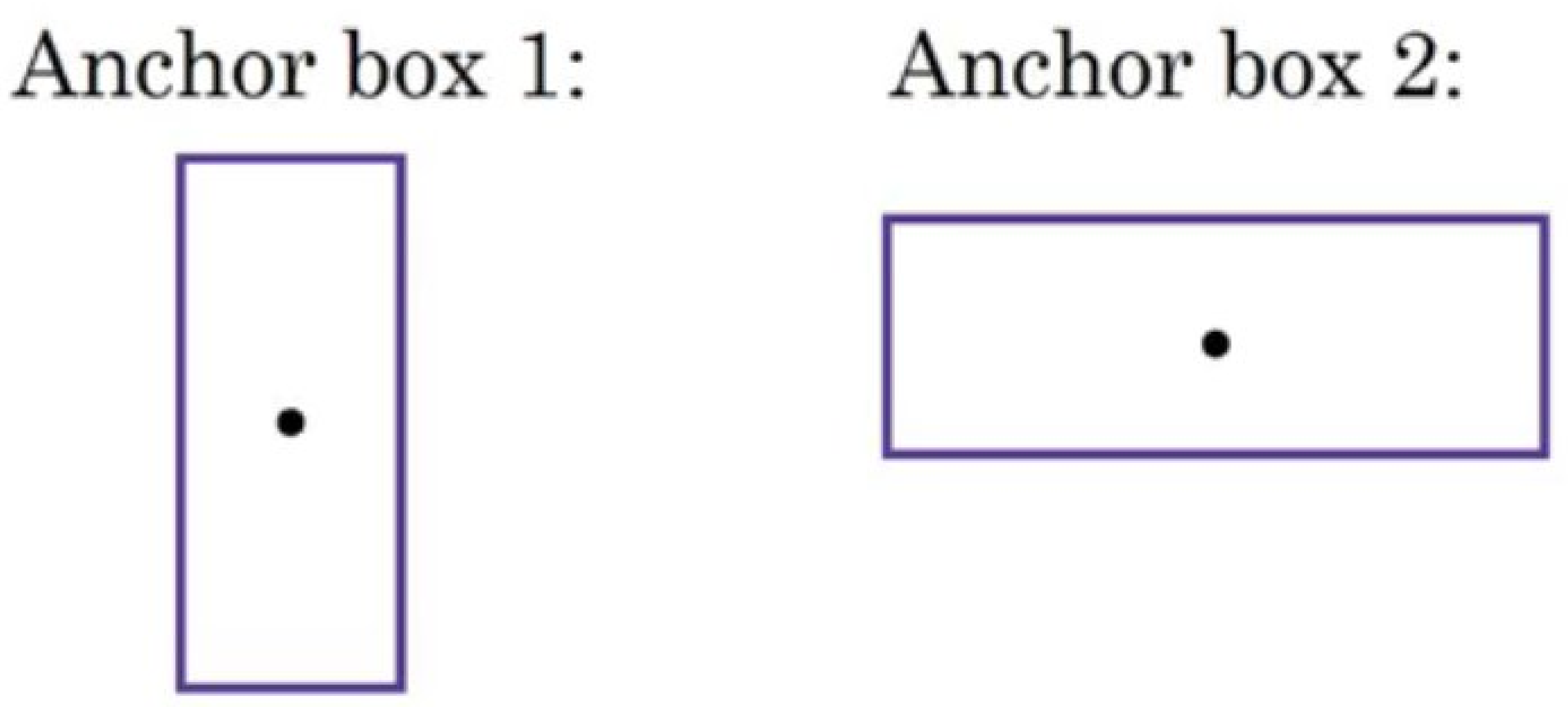


Figure 3.8: Assigning Two Anchor Boxes

The y-level without anchor box will be like as shown in Table 3.5.

Table 3.5: Output Vector without Anchor Box

y =	pc
	bx
	by
	bh
	bw
	c1
	c2
	c3

The y-level for 2 anchor boxes will be as shown in Table 3.6.

Table 3.6: Output Vector with Two Anchor Boxes

y =	pc
	bx
	by
	bh
	bw
	c1
	c2
	c3
	pc
	bx
	by
	bh
	bw
	c1
	c2
	c3

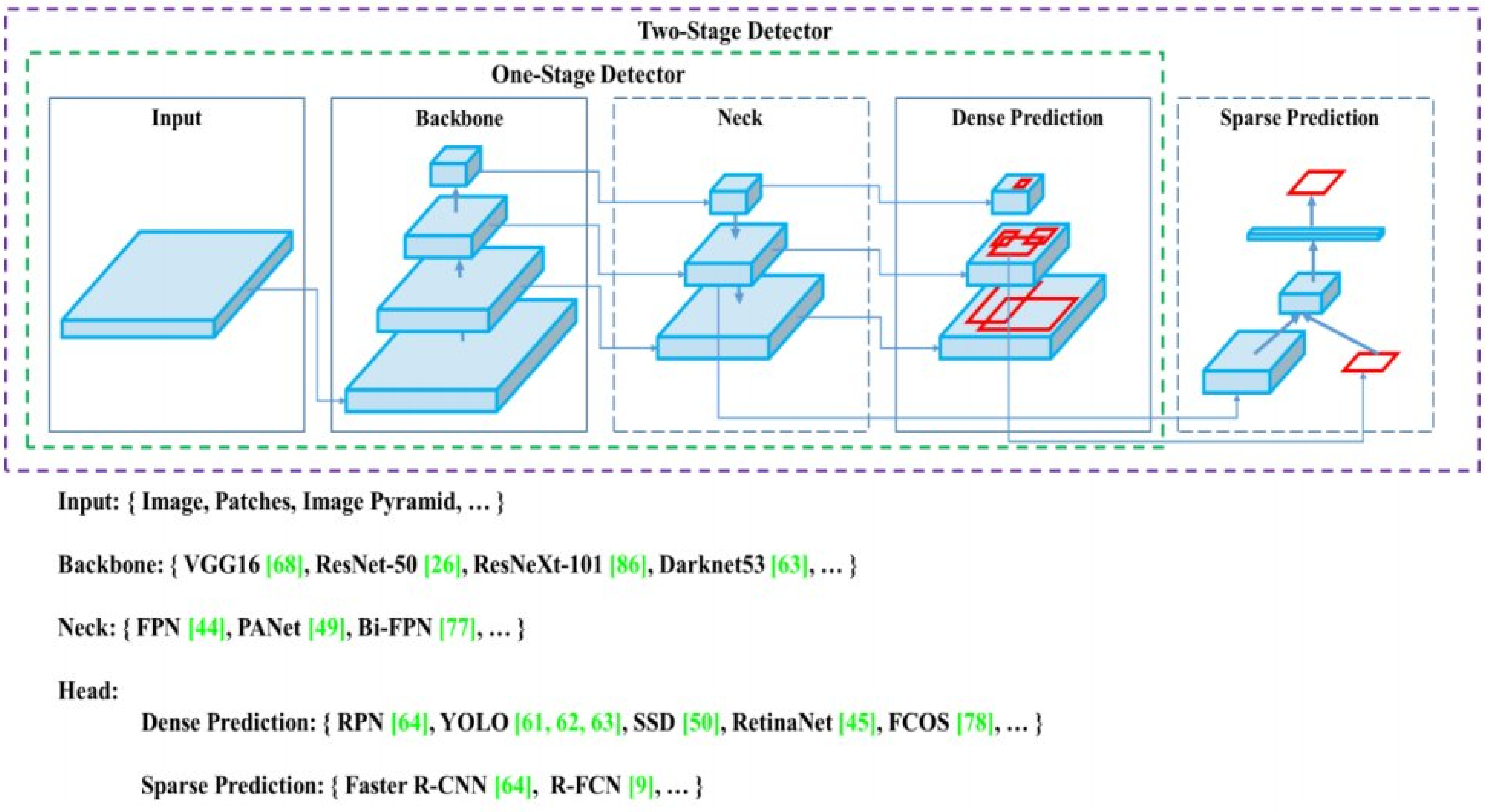
The first 8 rows are for anchor box 1 and the rest are for anchor box 2. The objects are assigned to anchor boxes based on the similarity of bounding boxes and the shape of the bounding boxes. Here the shape of the anchor box 1 is similar to the bounding box of the person. So, it is assigned to the bounding box 1 and the car will be assigned to the anchor box 2. The size of the y-level output will be 3x3x16 (using 3x3 grid and 3 classes) instead of 3x3x8. So, the higher the number of anchor the more object we can detect.

Chapter 4

Research Methodology

4.1 Architecture

Modern object detectors generally have two parts- the backbone and head. The backbone uses the pre-trained weights and the head draws bounding boxes around the objects detected and also assigns a class probability to each box. Figure 4.1 shows YOLO architecture.



Input: { Image, Patches, Image Pyramid, … }

Backbone: { VGG16 [68], ResNet-50 [26], ResNeXt-101 [86], Darknet53 [63], … }

Neck: { FPN [44], PANet [49], Bi-FPN [77], … }

Head:

Dense Prediction: { RPN [64], YOLO [61, 62, 63], SSD [50], RetinaNet [45], FCOS [78], … }

Sparse Prediction: { Faster R-CNN [64], R-FCN [9], … }

Figure 4.1: Architecture for YOLO

- Input: The input to any image detector is the image or the batch of images that are undergoing the object detection.

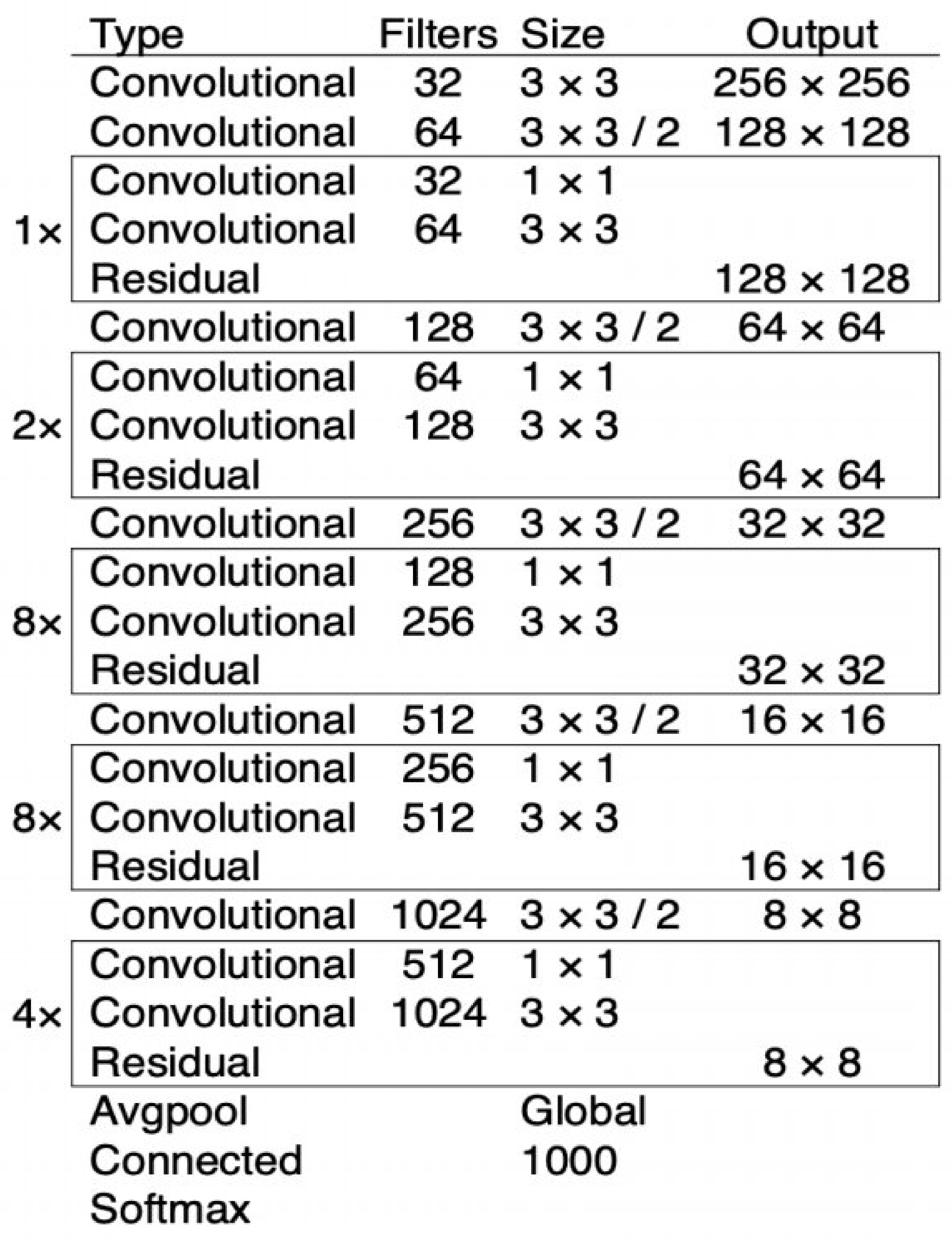
- Backbone: The backbone layer in addition to using the weights is involved in feature extraction. Depending on whether the backbone layer is run on GPU or CPU we can select either VGG [30], ResNet-101 [31], Darknet53, MobileNets [32] etc. For our research work we chose Darknet53 because it has 53 convolutional layers and has the best performance as compared to the rest [33]. Table 4.1 contains a detailed breakdown of the Darknet53 layer.

- Neck: The layer between the backbone and the head used to collect feature maps from different stages in the detector. The neck consists of multiple bottom-up paths and bottom-down paths. There are many layers equipped with this mechanism most notably, FPN [34], PAN [35], Bi-FPN [36] etc. These path aggregation blocks facilitate the process of feature detection and speed up the mean accuracy of precision. For our model we chose PANet [37] due to its robustness and ability to extract most features at the shortest time.

- Head: Object detectors usually have two type of prediction layers for the outputs. For one stage detectors only a Dense Prediction layer is used. For our purpose, we used the YOLOv3 as the head for YOLOv4-tiny. Anchor based heads can be used for the dense layer.

In addition to the Dense Prediction layer, two stage detectors use a Sparse Prediction layer as the final output layer. These are primarily used for segmentation models use as R-CNN, Mask R-CNN [38] and other anchor free models.

Table 4.1: Darknet53 Convolutional Layer Breakdown



	Type	Filters	Size	Output
	Convolutional	32	3 × 3	256 × 256
	Convolutional	64	3 × 3 / 2	128 × 128
	Convolutional	32	1 × 1	
1×	Convolutional	64	3 × 3	
	Residual			128 × 128
	Convolutional	128	3 × 3 / 2	64 × 64
	Convolutional	64	1 × 1	
2×	Convolutional	128	3 × 3	
	Residual			64 × 64
	Convolutional	256	3 × 3 / 2	32 × 32
	Convolutional	128	1 × 1	
8×	Convolutional	256	3 × 3	
	Residual			32 × 32
	Convolutional	512	3 × 3 / 2	16 × 16
	Convolutional	256	1 × 1	
8×	Convolutional	512	3 × 3	
	Residual			16 × 16
	Convolutional	1024	3 × 3 / 2	8 × 8
	Convolutional	512	1 × 1	
4×	Convolutional	1024	3 × 3	
	Residual			8 × 8
	Avgpool		Global	
	Connected		1000	
	Softmax			

4.2 Bag of Freebies

Most conventional object detectors are trained offline using GPUs and other computational resources. Thus, researchers constantly try to increase the accuracy of the detector without increasing the cost of inference. The strategy used to best obtain this tradeoff is called "Bag of Freebies". Techniques used:

1. Data Augmentation: Increases the variability of images that are inputted into the detector. This allows for greater robustness and takes into consideration the environmental factors along with photometric and geometric distortions. Pixel-wise adjustments such as CutOut, CutMix, MixUp, DropOut, DropConnect etc. are used that retain all the pixel information in the adjusted area. For our models, we used MixUp

which uses two images for multiplication and super imposition with distinct coefficient ratios and then adjusts the labels according to the superimposed labels.

2. Focal loss reduction: Used to mitigate the data imbalance between different classes within the dataset. This expresses the relationship of association between the different classes with one-hot hard representation.

3. Label smoothing: Converts hard labels to soft labels that are used in training to make the model more robust. The label refinement network introduces the idea of knowledge distillation.

4. Bounding Box (BBox) regression: Traditionally, MSE used as regression model to find the coordinates of the center, width and height of the bounding box. The estimated coordinates of each point of the bounding box are treated as an independent variable which compromises the integrity of the object. Thus, Intersection over Union (IoU) is used which takes into account the predicted bounding box area and ground truth.

4.3 Bag of Specials

The plugin modules and post-processing methods that significantly increase the accuracy of detection but at a small inference cost are called “bag of specials”. They enhance certain attributes of the model that allows better reads and more accurate predictions. The modules are:

1. Receptive Field: These modules increase the area on the image over which the model performs detection. Spatial Pyramid Matching (SPM) splits the feature maps into square blocks to extract bag-of-word features. It uses max pooling and outputs one dimensional feature vector.

2. Attention module: Has channel-wise attention and point-wise attention. The Squeeze and Excitation (SE) and Spatial Attention Module (SAM) improve both the accuracy and inference times.

3. Feature Integration: Skip connection or Hyper-column integrates low level physical features to high level semantic features.

4. Activation Function: Activation functions allow the gradient to be propagated more efficiently without extra computational cost. We used ReLU activation function.

5. Post-processing method: Used to filter out and eliminate bounding boxes with low confidence scores and retain those that meet the threshold required. NMS is used to optimize the process and meet the requirement criterion.

4.4 Dataset

The dataset for training was provided by Dhaka-AI [39], named "Dhaka Traffic Detection Challenge Dataset" [40] [41]. The Dataset is composed of the most common 21 different classes of vehicles of Dhaka city. There are total 3003 images in the dataset and 24,368 annotations. List of the class names and their number of appearance in the dataset:

• Ambulance	: 70	• Minibus	: 95	• Scooter	: 38
• Auto-rickshaw	: 43	• Minivan	: 935	• SUV	: 860
• Bicycle	: 459	• Motorbike	: 2284	• Taxi	: 60
• Bus	: 3340	• Pickup	: 1225	• CNG	: 2990
• Car	: 5476	• Army vehicle	: 43	• Truck	: 1492
• Garbage van	: 3	• Police car	: 32	• Van	: 756
• Human hauler	: 169	• Rickshaw	: 3549	• Wheelbarrow	: 120

The dataset is an imbalanced dataset. Car, rickshaw, bus and CNG are overrepresented where except only motorbike, truck, pickup and minivan, the rest of the vehicles are underrepresented. Figure 4.4 shows the class balance report generated by Roboflow [42]. Figure 4.2 and 4.3 show a typical image for each of the classes of vehicles.

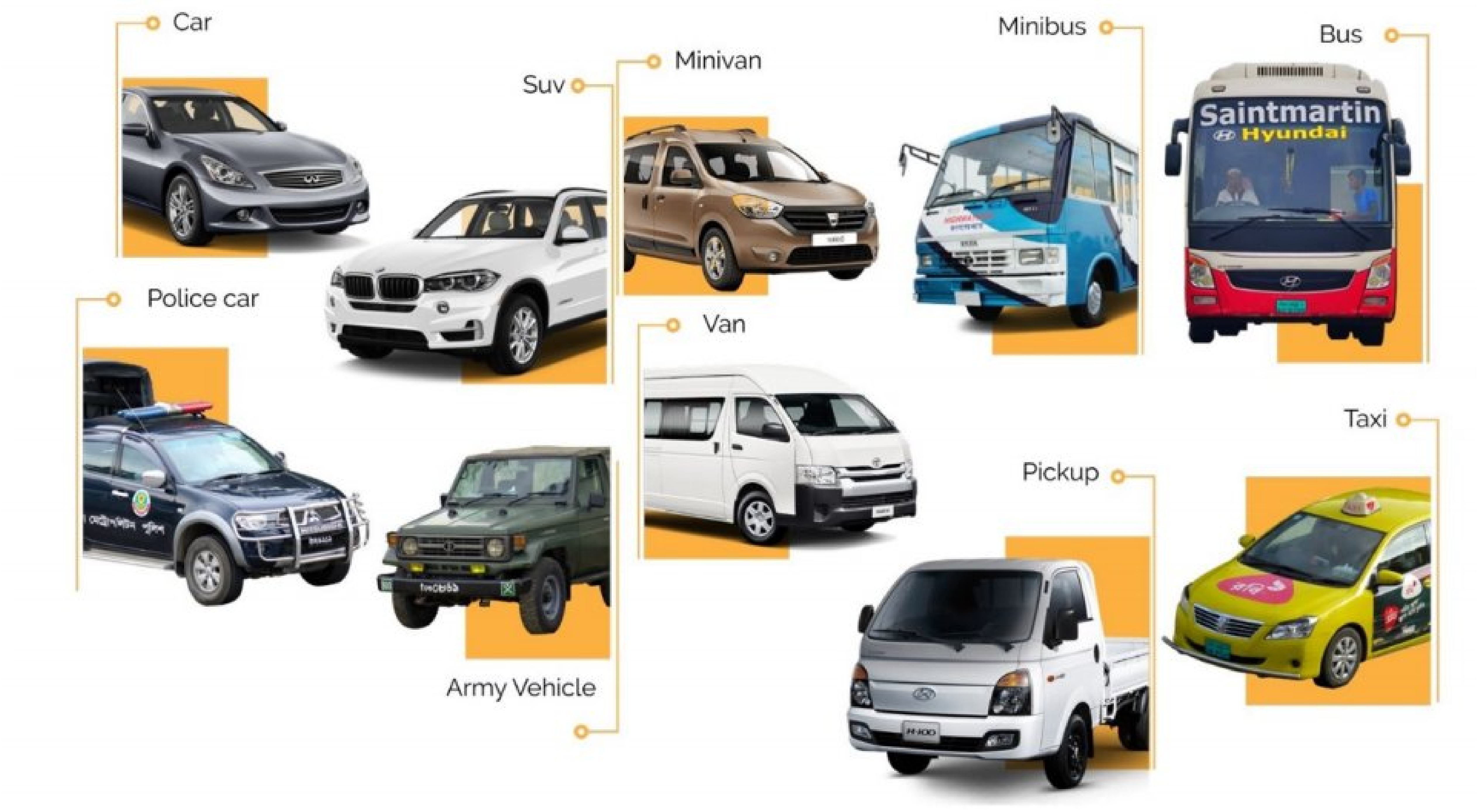


Figure 4.2: Different Classes of Vehicles of Dhaka City (1)

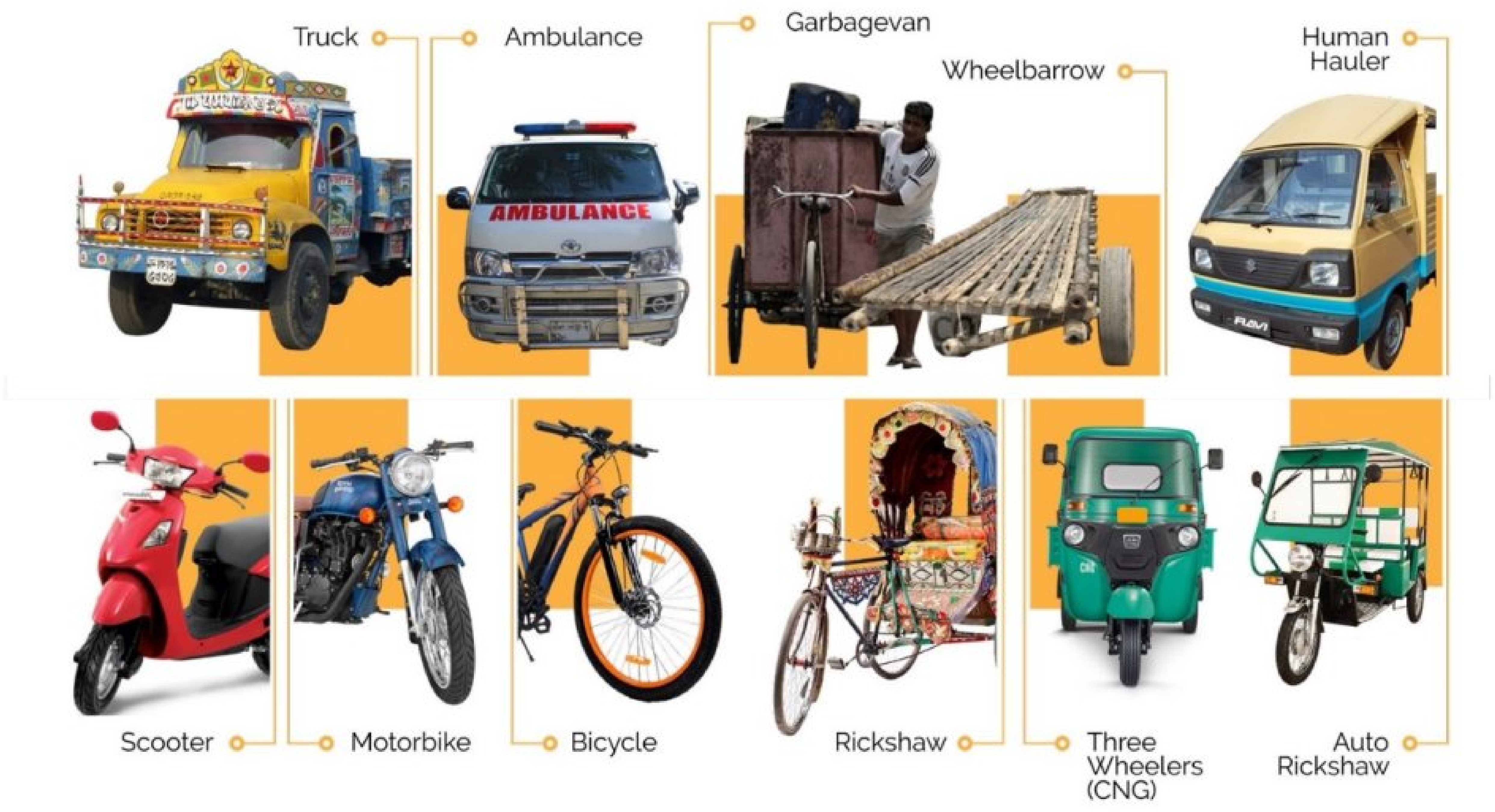


Figure 4.3: Different Classes of Vehicles of Dhaka City (2)

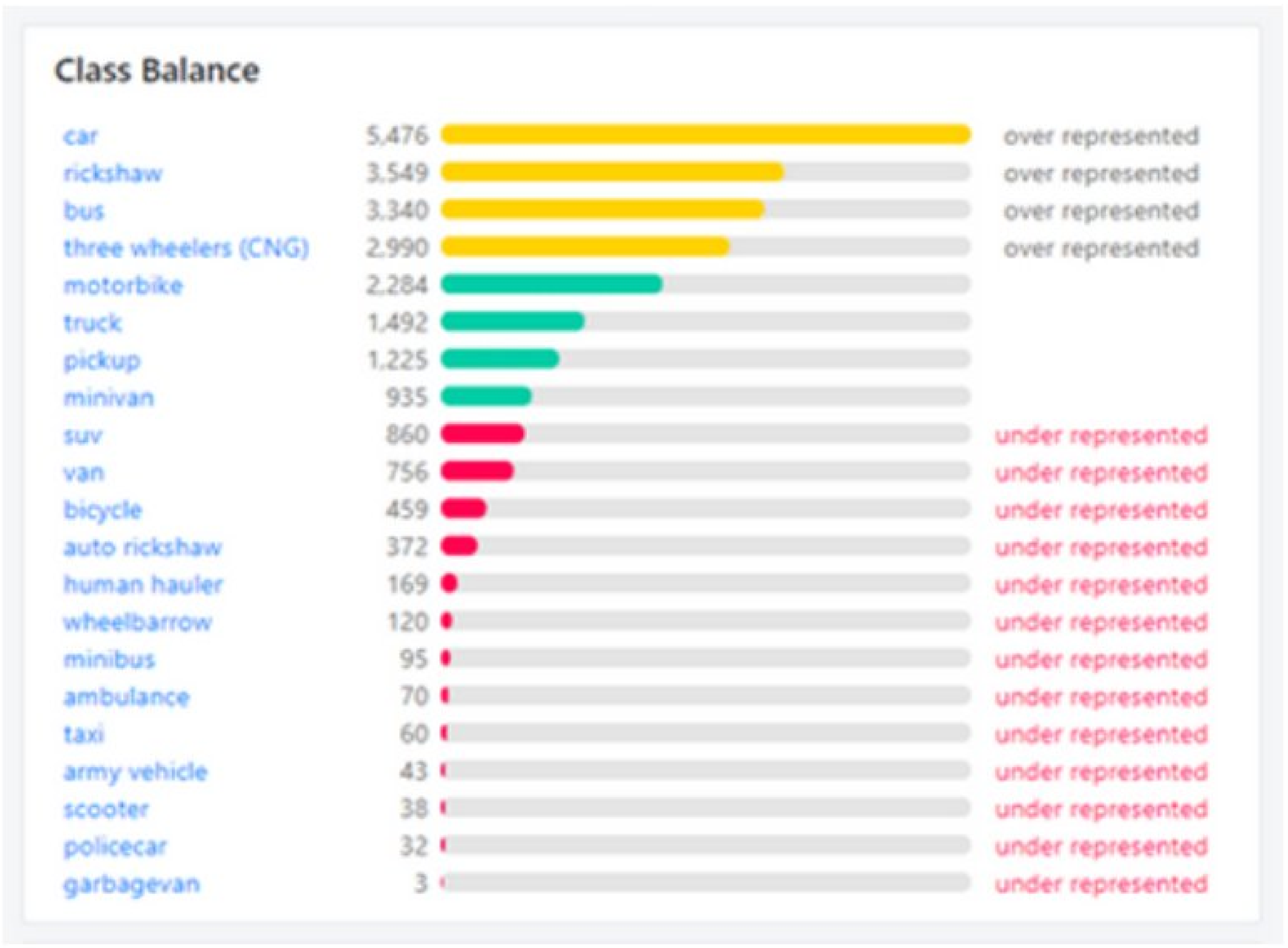


Figure 4.4: Dataset Health from Roboflow

4.5 YOLO Training Parameters

4.5.1 YOLOv3

1. Batch size, subdivisions and steps:
   batch = 64, subdivisions = 64, max_batches = 500200, policy = steps, steps = 400000,450000
2. Image dimensions in pixels:
   width = 416, height = 416, channels = 3
3. Hyperparameters:
   momentum = 0.9, decay = 0.0005, angle = 0, learning_rate = 0.001, burn_in = 1000
4. Image augmentation:
   scales = .1,.1, saturation = 1.5, exposure = 1.5, hue =.1
5. YOLO specific parameters:
   classes = 21, convolutional layer filters = 78
   num = 9, jitter = .3
   ignore_thresh = .7, truth_thresh = 1, random =

4.5.2 *YOLOv4-tiny*

1. Batch size, subdivisions and steps:
   batch = 64, subdivisions = 64, max_batches = 42000, policy = steps, steps = 33600, 37800
2. Image dimensions in pixels:
   width = 416, height = 416, channels = 3
3. Hyperparameters:
   Momentum = 0.9, decay = 0.0005, angle = 0, learning_rate = 0.00261, burn_in =1000
4. Image augmentation:
   scales =.1,.1, saturation = 1.5, exposure = 1.5, hue =.1
5. YOLO specific parameters:
   classes = 21, convolutional layer filters = 78
   num = 6, jitter = .3, scale_x_y = 1.05, cls_normalizer = 1.0
   iou_normalizer = 0.07, iou_loss = ciou, ignore_thresh = .7, truth_thresh = 1, random =1, resize =1.5, nms_kind = greedynms, beta_nms = 0.6

4.5.3 *YOLOv5*

1. Batch size, subdivisions and steps:
   batch = 16, epochs = 600
2. Image dimensions in pixels:
   width = 640, height = 640, channels = 3
3. Image augmentation:
   Random, horizontal, flip
4. YOLO specific parameters:
   classes = 21

4.6 Darknet Requirements and Installation

We ran YOLOv3 and v4-tiny on the Darknet53 framework as discussed before. We trained these two models on a moderately powered workstation. The specifications will be given later.

4.6.1 Requirements

1. Windows or Linux
2. CMake >= 3.12
3. CUDA >= 10.0
4. OpenCV >= 2.4
5. cuDNN >= 7.0
6. GPU with CC >= 3.0
7. Windows MSVC 2017/2019
8. Microsoft Visual Studio
9. Python >=3.6
10. IP Webcam4.6.2

4.6.2 Installation

1. Install Python and add to Path. Use Python 3.6 or 3.7.
2. Install numpy [43] package using pip and cmd.
3. Install Git [44].
4. Install Cmake [45].
5. Download and install CUDA [46] drivers on your NVIDIA GPU enabled machine
6. Download cuDNN [47] from Nvidia website depending on your GPU version compatibility.
7. Follow the documentation on proper installation given on the Nvidia website
8. Add cuDNN to the necessary paths and reboot system.
9. Download OpenCV [48] 4.1.2 from the github repo using the git command in the cmd
10. Install OpenCV using Cmake and the necessary CUDA enabled parameters.
11. Download the Darknet repo by AlexeyAB's Github account [49] and extract the contents to your desired location.
12. Using Microsoft Visual Studio [50] build the darknet.sln file and debug accordingly.
13. Then add the darknet folder location to the system variables and add to path
14. We used a smartphone camera as an IP webcam. Install the IP webcam app on your smartphone.
15. Ensure the smartphone and workstation are both connected to the same network
16. Using the IP address on the webcam and the darknet cmd, run the commands to enable access to the phone's camera using the workstation.
17. Thus, we can use the phone's camera to act as real time input and the workstation outputs the real time detection.

4.7 Platforms and System Specifications

4.7.1 Workstation

YOLOv3 and v4-tiny models were trained on Acer Predator Helios 300 laptop. The specifications do indicate that it is a quite powerful system, which are following:

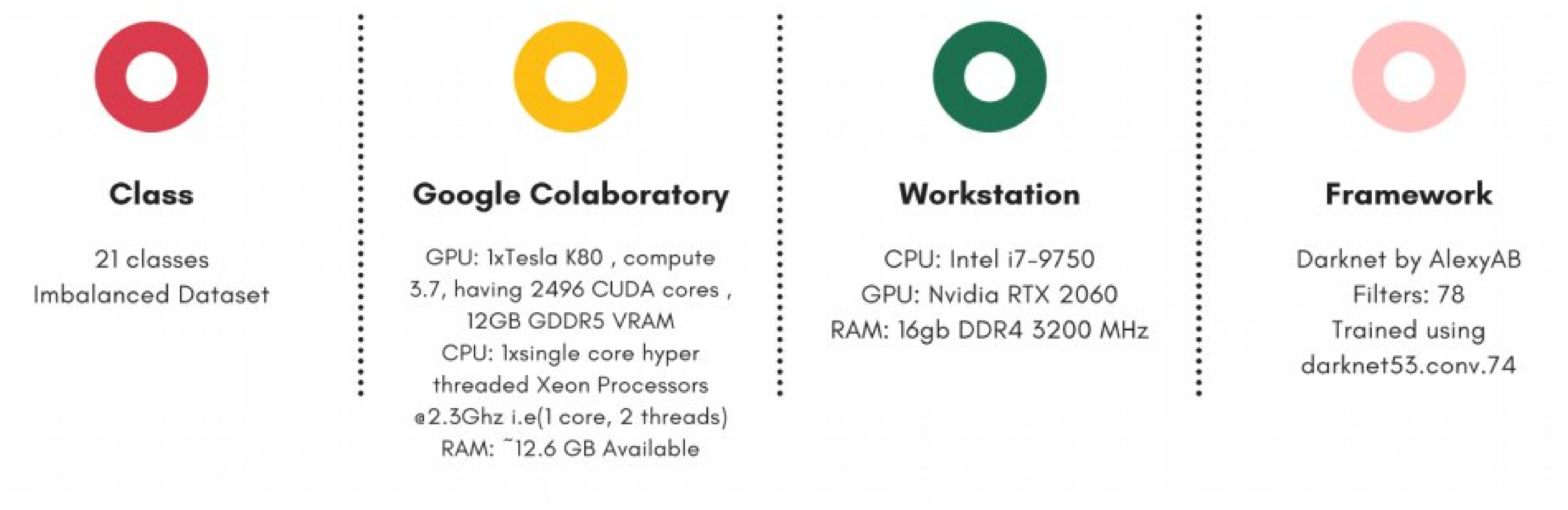
- CPU: Intel i7-9750H (6 cores, 12 Threads @2.60 GHz and boosted @4.50 GHz)
- GPU: NVIDIA RTX 2060 with 1,920 CUDA cores, 6 GB GDDR5 VRAM
- RAM: 16 GB DDR4 3200 MHz

4.7.2 Cloud

Our YOLOv5 Model was trained on Google Colaboratory, a cloud platform which can be used for machine learning applications. Google Colaboratory lets us import an image dataset and train any model on them. Even the free version of Colab has more powerful and impressive specifications than our workstation.

- CPU: 1xsingle core hyper threaded Xeon Processors @2.3Ghz (1 core, 2 threads)
- GPU: 1xTesla K80, compute 3.7, having 2496 CUDA cores, 12GB GDDR5 VRAM
- RAM: ~12.6 GB

Figure 4.5 shows a summary of the dataset and the training platforms.



Class
21 classes
Imbalanced Dataset

Google Colaboratory
GPU: 1xTesla K80 , compute 3.7, having 2496 CUDA cores , 12GB GDDR5 VRAM
CPU: 1xsingle core hyper threaded Xeon Processors @2.3Ghz i.e(1 core, 2 threads)
RAM: ~12.6 GB Available

Workstation
CPU: Intel i7-9750
GPU: Nvidia RTX 2060
RAM: 16gb DDR4 3200 MHz

Framework
Darknet by AlexyAB
Filters: 78
Trained using
darknet53.conv.74

Figure 4.5: Summary of Dataset and workstation

4.8 Training

In this section we'll discuss training details for our different models in different platforms.

4.8.1 YOLOv3

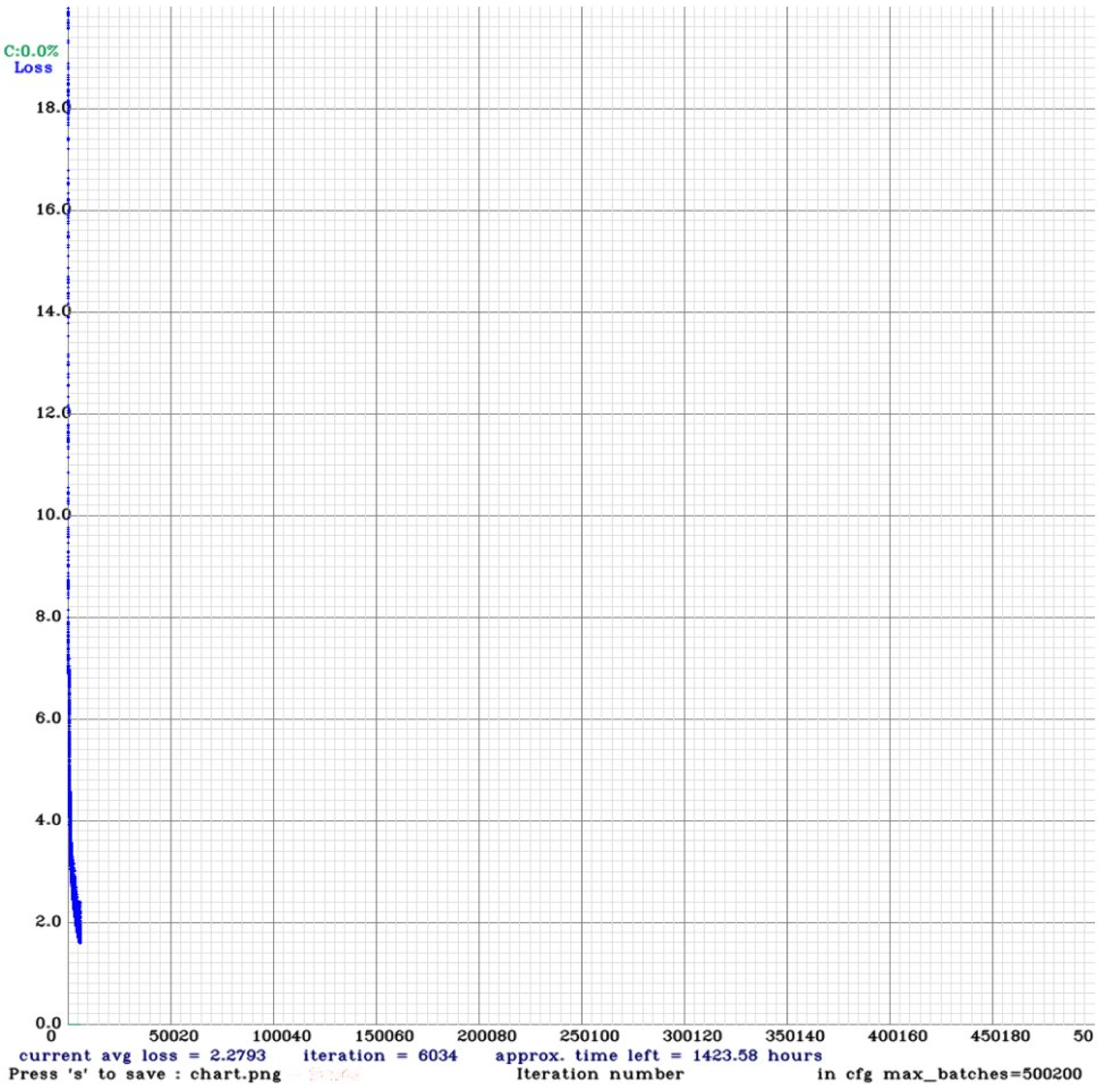
The YOLOv3 model was trained on our dataset for 6034 iterations for an approximate training time of 22 hours. The model was trained on our workstation and on a single GPU system. The average loss of the model was about 2.2793. Figure 4.6 shows loss vs. iterations for YOLOv3 training.

4.8.2 YOLOv4-tiny

The YOLOv4-tiny model was trained on our dataset for 126000 iterations for an approximate training time of 36 hours. The model was trained on our workstation and on a single GPU system. The average loss of the model was about 1.7429. Figure 4.7 shows Figure shows loss vs. iterations for YOLOv4-tiny training.

4.8.3 YOLOv5

The YOLOv5 model was trained on our dataset for 600 epochs for an approximate training time of 10 hours. This model was trained on Google Colaboratory, which has a more powerful GPU than our workstation. This is why training time was the least for YOLOv5 and also its own architecture is the fastest among all the previous versions of YOLO. The dataset was imported to Colab using Roboflow. Roboflow also provided us with enough instruction on how to train our desired model. Figure 4.9 shows different performance matrices vs. epoch for the training. Figure 4.8 shows the confusion matrix for YOLOv5 for our dataset.



current avg loss = 2.2793 iteration = 6034 approx. time left = 1423.58 hours
Press 's' to save : chart.png in cfg max_batches=500200

Figure 4.6: Loss vs Iterations for YOLOv3 Training

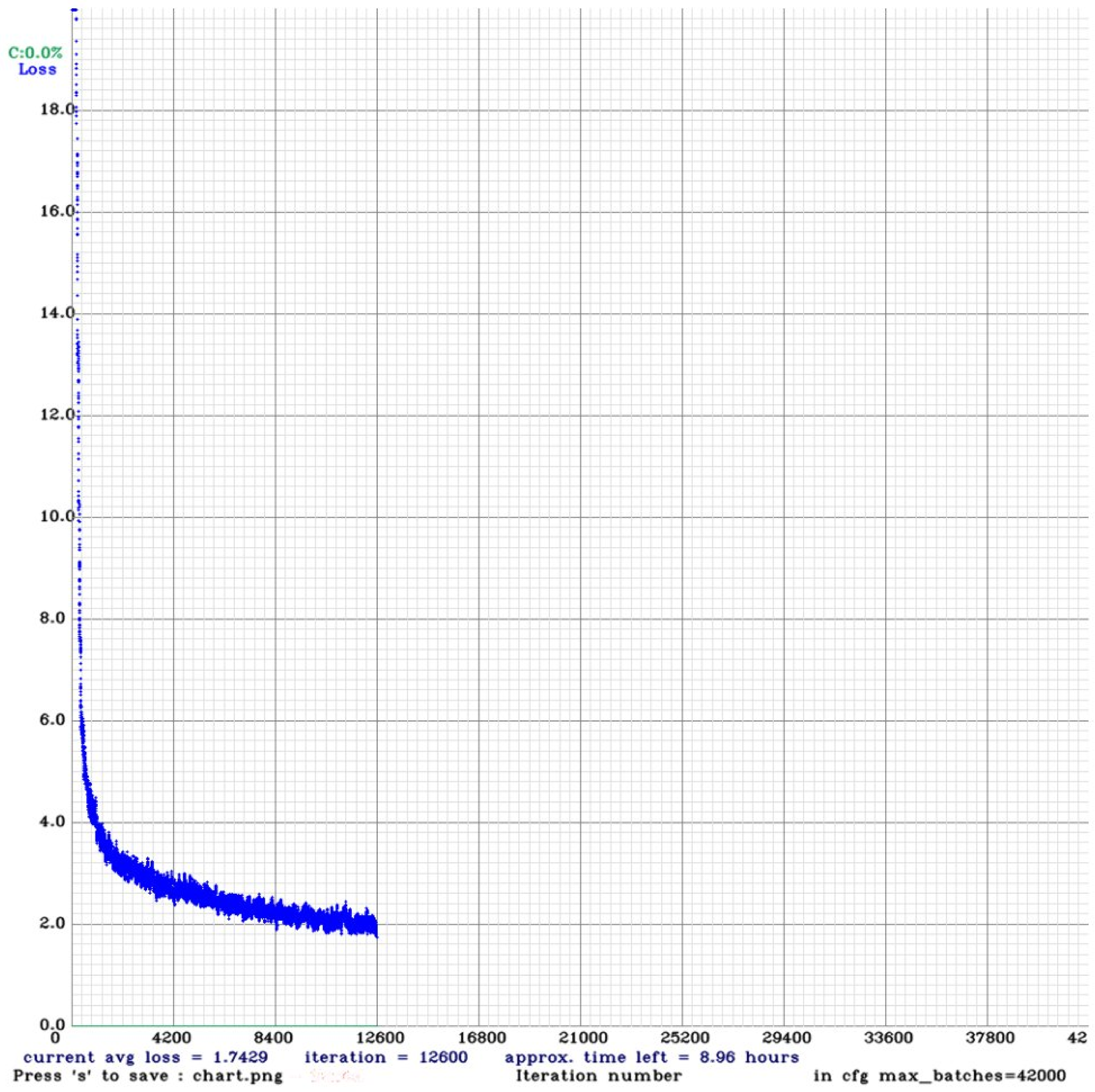


Figure 4.7: Loss vs Iterations for YOLOv4-tiny Training

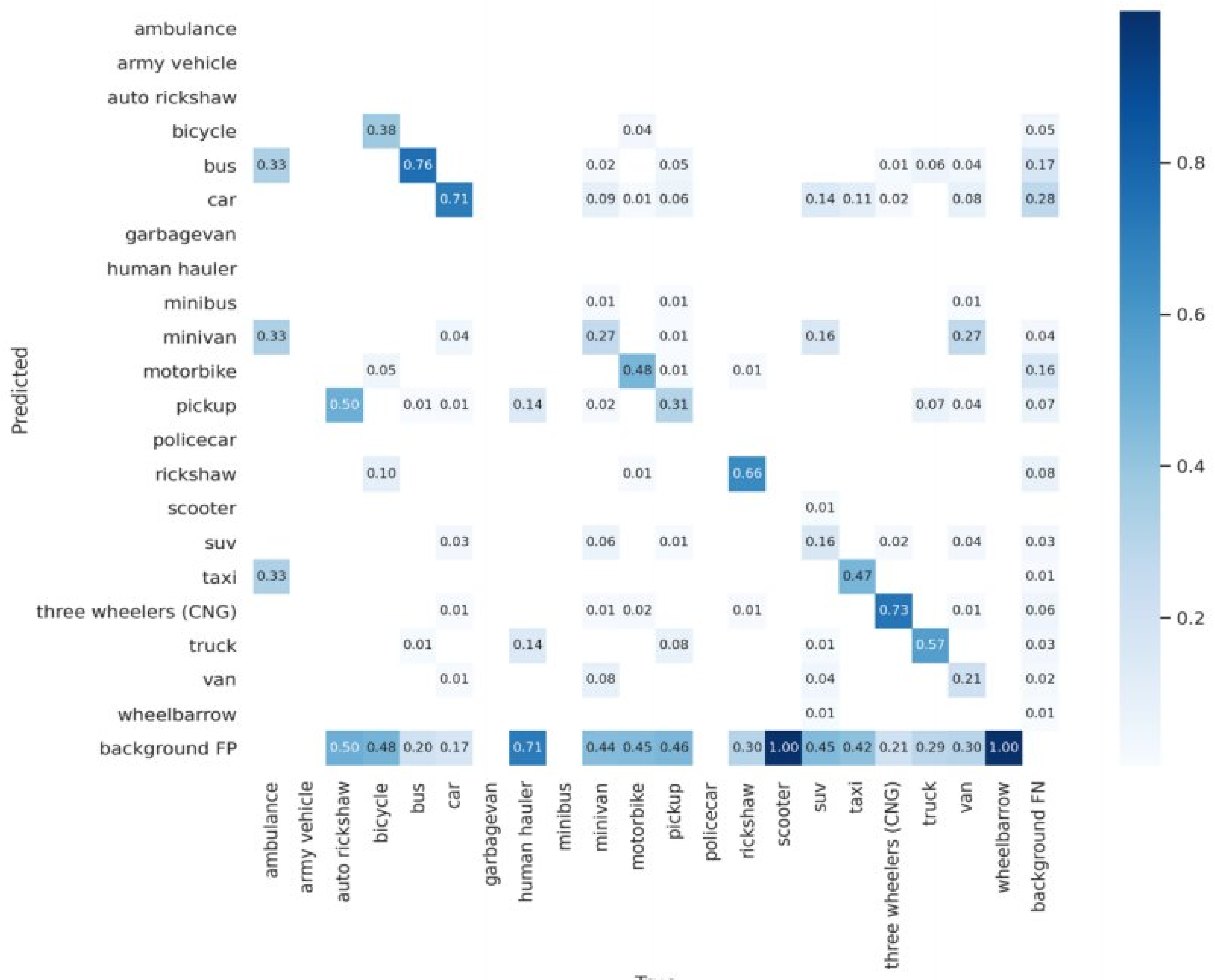


Figure 4.8: Confusion Matrix for YOLOv5

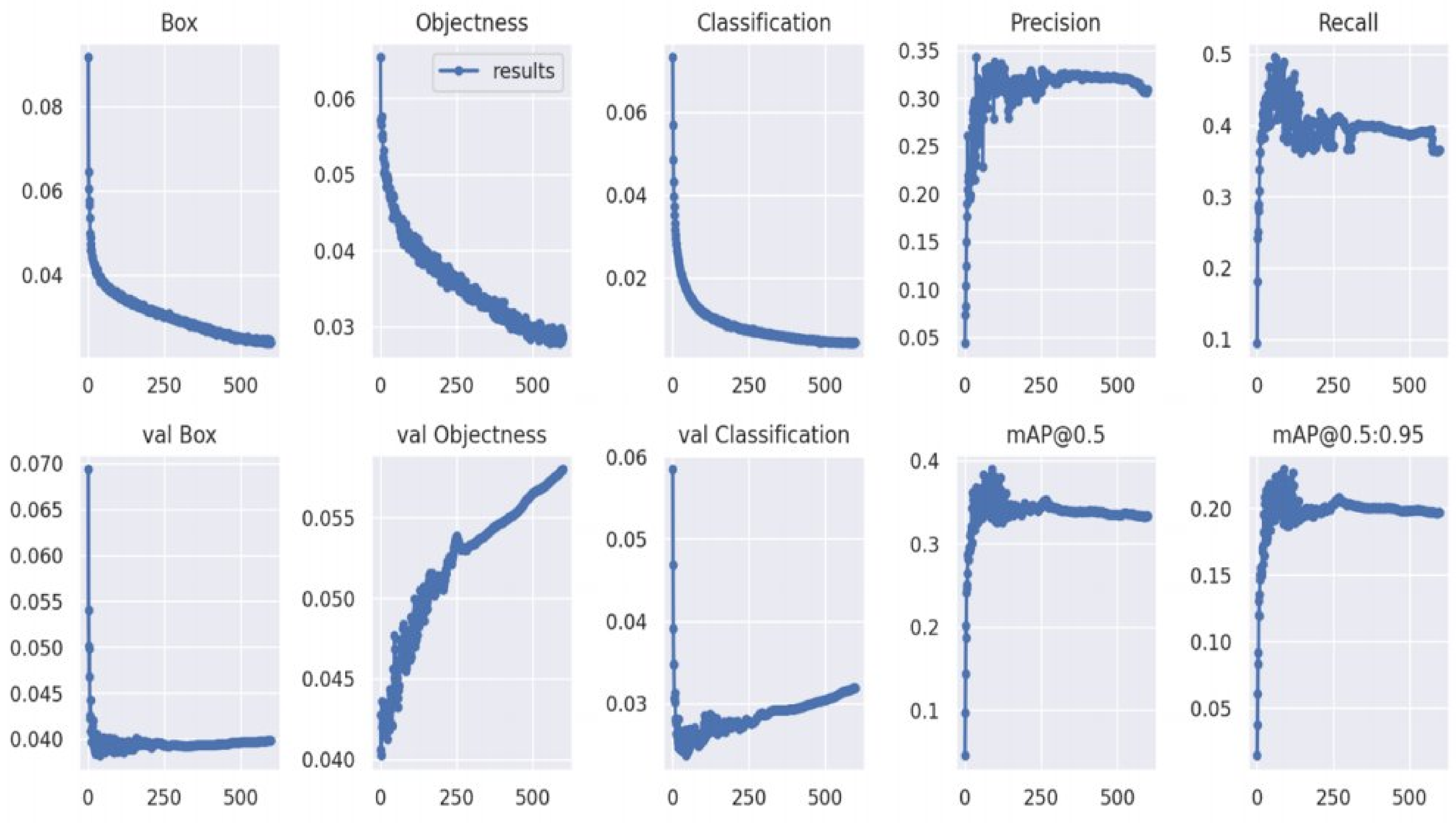


Figure 4.9: Performance metrics vs Epochs for YOLOv5

4.7 Research Timeline

We meticulously divided our entire research work into 4 phases to better facilitate our progress and ensure we had a rigid timeline to follow that would help us stay in track. Instead of jumping to the finish line and worrying about the end result, we decided to take a more structured approach to our learning and work. For each phase, we set certain goals that we wanted to achieve and were flexible on the time in order for us to accomplish our goals thoroughly instead of doing anything half done.

We had one common goal in mind: To learn and explore. Given how machine learning and deep learning were unknowns to us, we wanted to have a strong fundamental basics on whatever we did. We focused on the theoretical aspects of learning and implement what we learned as we went into this quest. We did projects of various kinds to better facilitate our learning and eventually lead to our dissertation work. A summarized timeline of our research is shown in Figure 4.10.

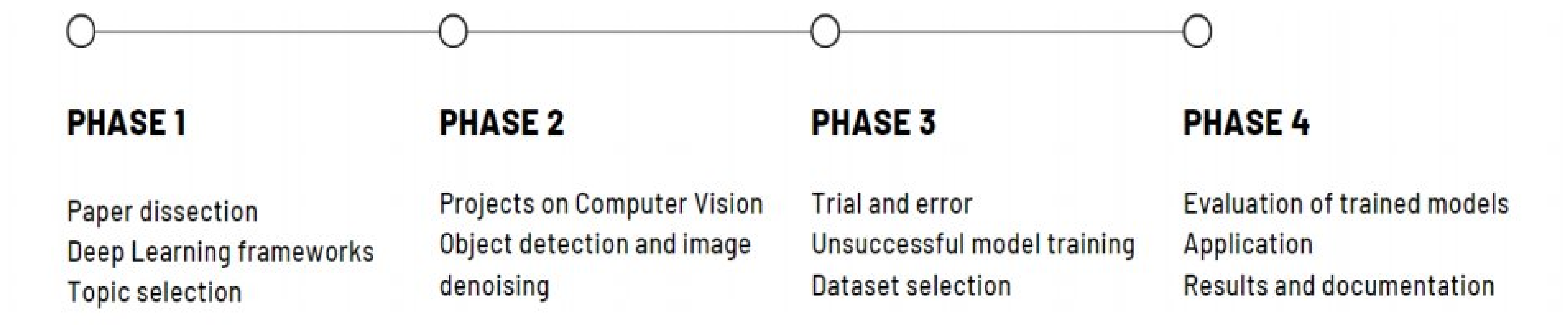

PHASE 1
Paper dissection
Deep Learning frameworks
Topic selection
PHASE 2
Projects on Computer Vision
Object detection and image denoising
PHASE 3
Trial and error
Unsuccessful model training
Dataset selection
PHASE 4
Evaluation of trained models
Application
Results and documentation

Figure 4.10: Timeline of Research Progression

4.9.1 Phase 1

The first phase took the longest. It started after the end of our 3rd year final exams. We started doing online courses, watching YouTube videos and other MOOCs to learn about Python programming, machine learning and subsequently deep learning. We also started reading research papers on the various techniques and methods used in machine and deep learning. The entire phase took about six months and gave us enough insight to find a topic for our dissertation. We chose Computer Vision because the idea that machines could see and recognize objects like humans was fascinating to us. We realized this is what we wanted to explore and build on.

4.9.2 Phase 2

The second phase was much more advanced and challenging. This was the phase we had dived deep into Computer Vision- the theory and the execution. We learned different techniques used for Computer Vision, the classical and the new. We explored through Github [51] and Stack Overflow [52] to find projects that piqued our interest. We did many small projects such as real-time attendance system and image denoising to strengthen our grip on this area of research and try to find inspiration for our own thesis. This phase took about 3 months.

4.9.3 Phase 3

During this phase we spent hours trying to find a consensus on what subtopic and dataset we should be using for our thesis work. We scoured through Kaggle [53] and UCI Datasets to find a topic that we enjoyed doing. Whilst there were many intriguing topics, we always felt we wanted to work on something related to our own surroundings. We noticed a lack of Bangladesh specific datasets. This prompted the idea to create our own dataset and contribute to the Deep Learning community.

Luckily for us, we stumbled across the competition "Dhaka-AI" which was an AI based competition organized in Bangladesh using Dhaka specific dataset. Thus, we used the Dhaka traffic dataset to create a real time vehicle detection system using deep learning. We used different techniques and spent a lot of time on trial and error. Finally, we landed on YOLO framework as the basis of our thesis work given its superior performance and ease of customizability. This phase took about 4 months.

4.9.4 Phase 4

Given, we had selected our topic, dataset and mode of action, all that was left for us, was to put our learning into action and bring our thesis work to life. We spent the next 2 months on making our models more efficient and trained them to obtain the best possible results. As this was the final stretch, we documented all our findings and were preparing for our defense. We tweaked our models and tried various techniques to get even better results. We wanted to ensure no stones were left unturned and that we were able to proudly and successfully defend our thesis in front of the thesis committee.

By the grace of Almighty Allah, the guidance and support of our supervisor, Professor Dr. Golam Sarowar sir and the love and support of our parents, we were able to overcome all obstacles and present or thesis.

Chapter 5

Results

After successful training sessions of our different YOLO models, we fed the models both images and videos outside of the dataset to see how they performed in detecting vehicles, and how fast they were doing it. We also made a crude comparison between different versions of yellow on their performance. The following sections demonstrate the results:

5.1 Inference on Images

We randomly selected three images outside of the dataset on which our models were not trained on. Then we ran inference on them using our YOLO models.

- YOLOv3 : Figure 5.1, 5.2 and 5.3 show inference on image 1, 2 and 3.
- YOLOv4-tiny : Figure 5.4, 5.5 and 5.6 show inference on image 1, 2 and 3.
- YOLOv3 : Figure 5.7, 5.8 and 5.9 show inference on image 1, 2 and 3.

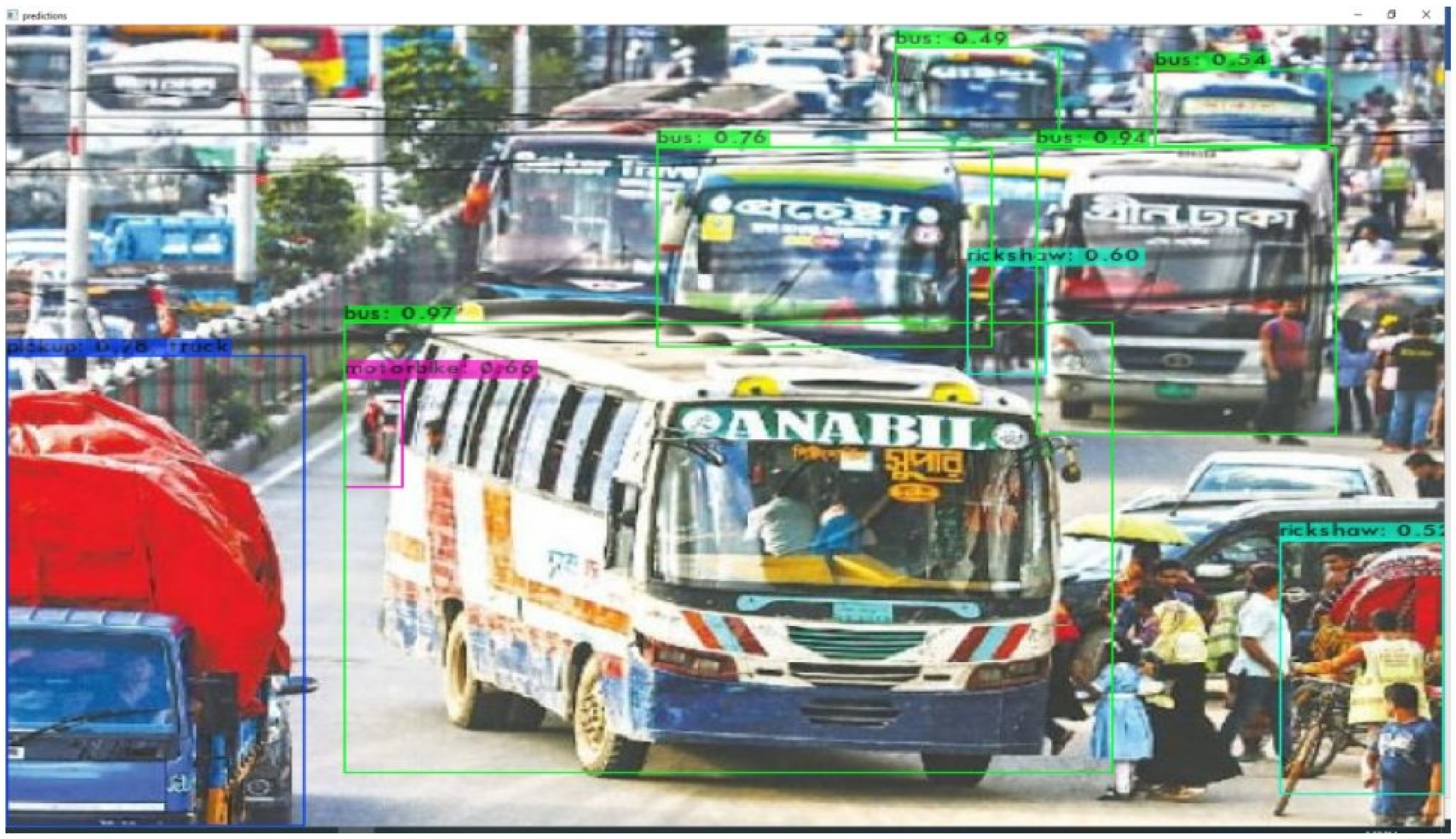


Figure 5.1: Yolov3 on Image 1

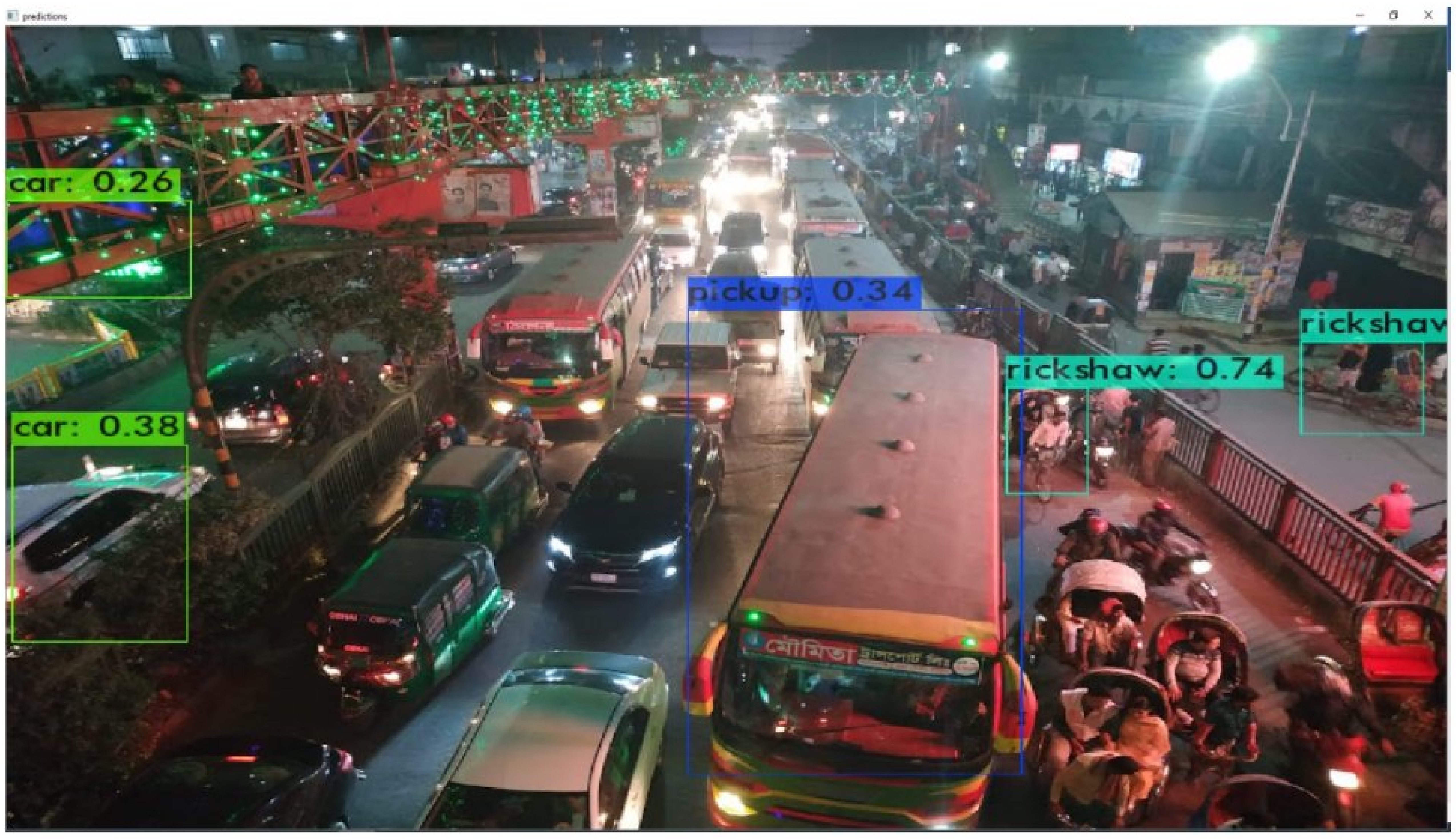


Figure 5.2: Yolov3 on Image 2

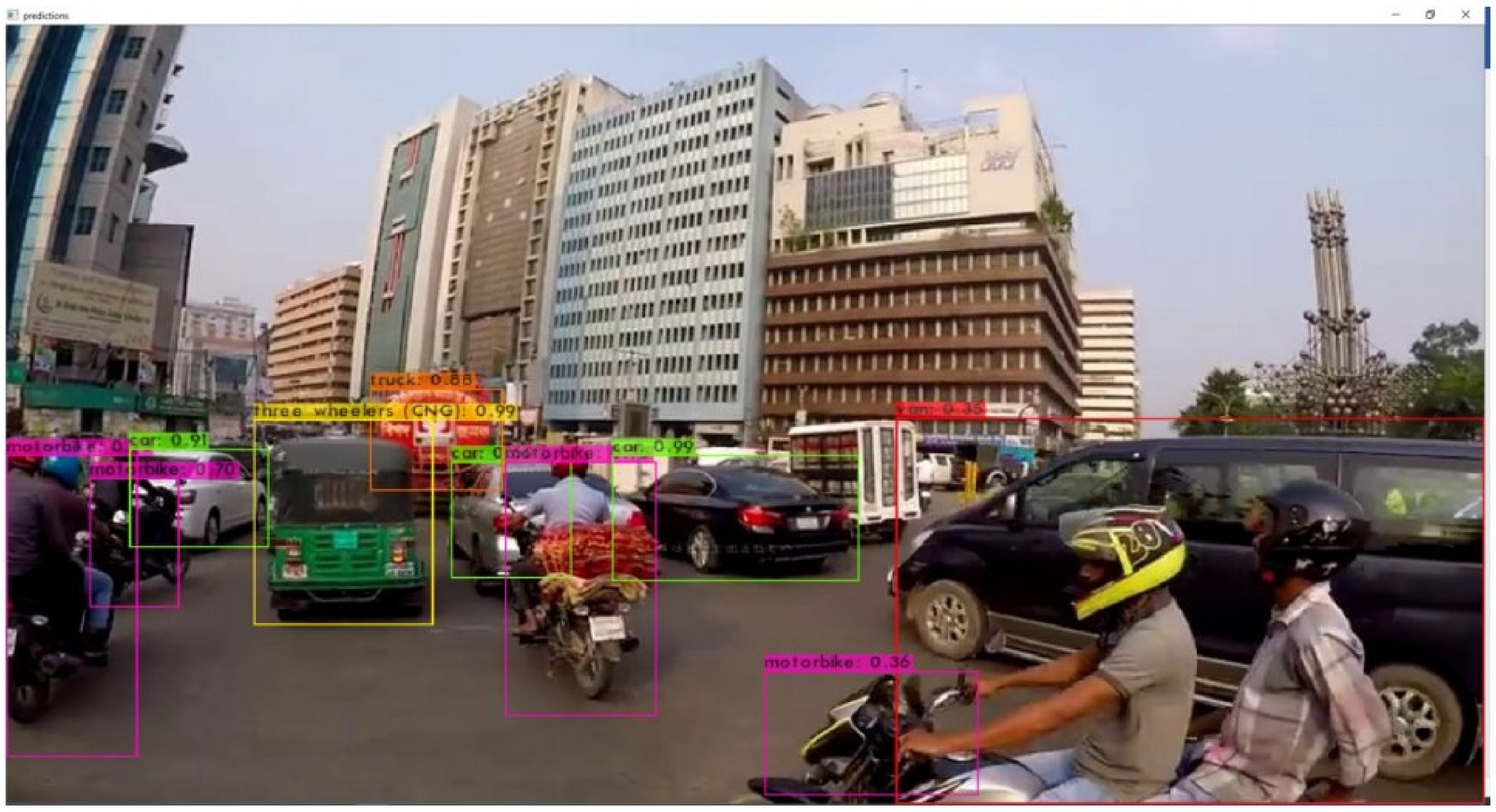


Figure 5.3: Yolov3 on Image 3

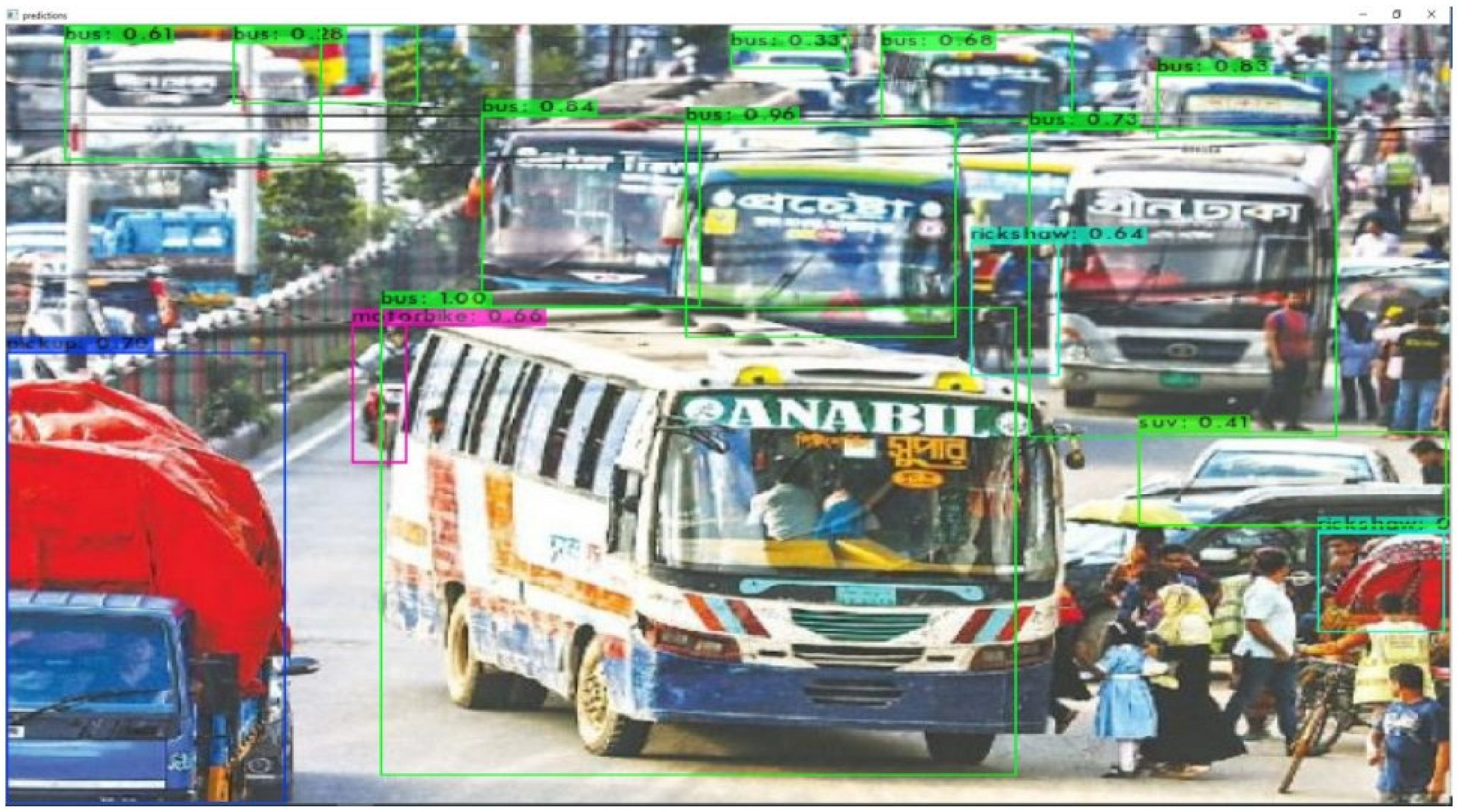


Figure 5.4: Yolov4-tiny on Image 1

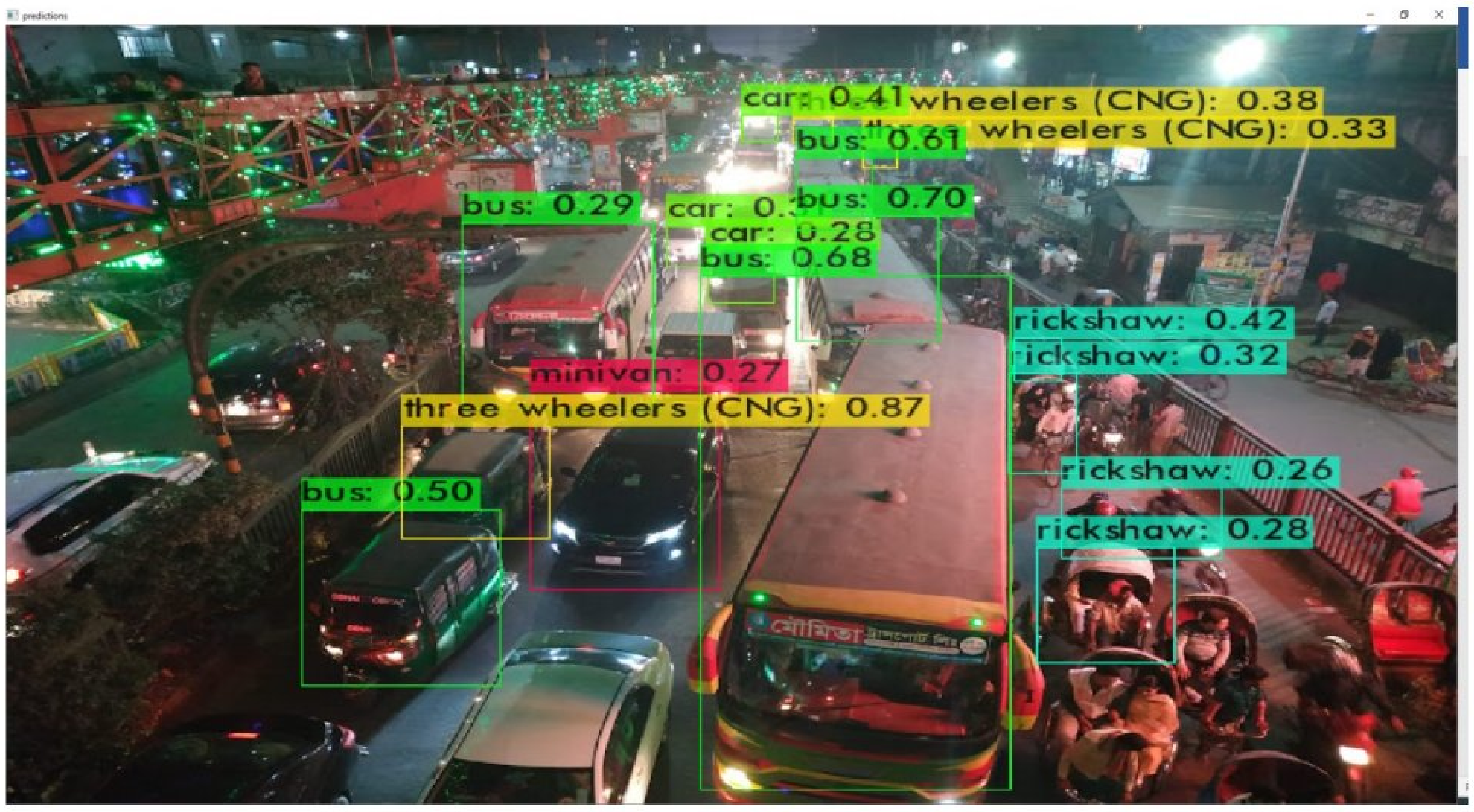


Figure 5.5: Yolov4-tiny on Image 2

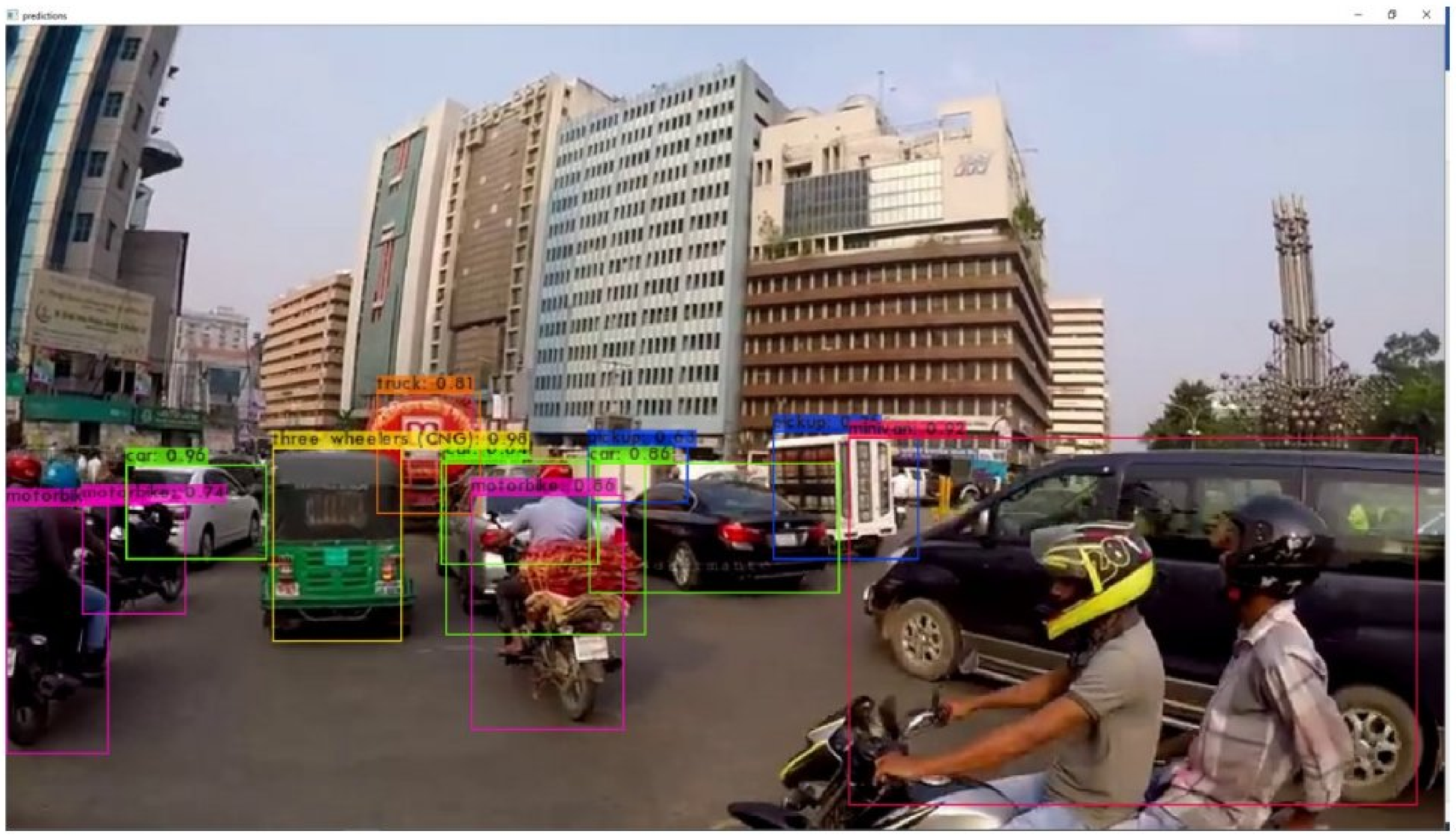


Figure 5.6: Yolov4-tiny on Image 3

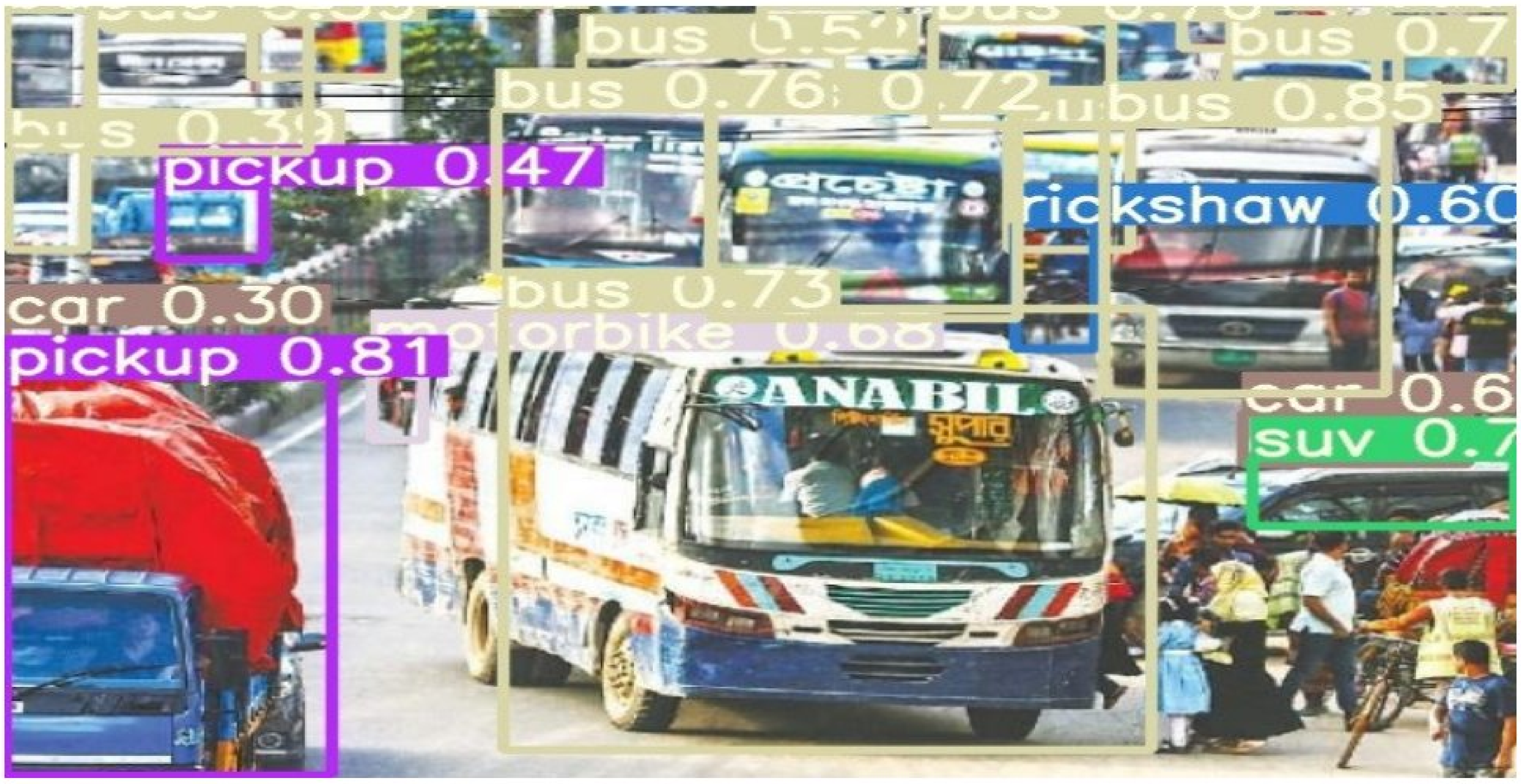


Figure 5.7: Yolov5 on Image 1

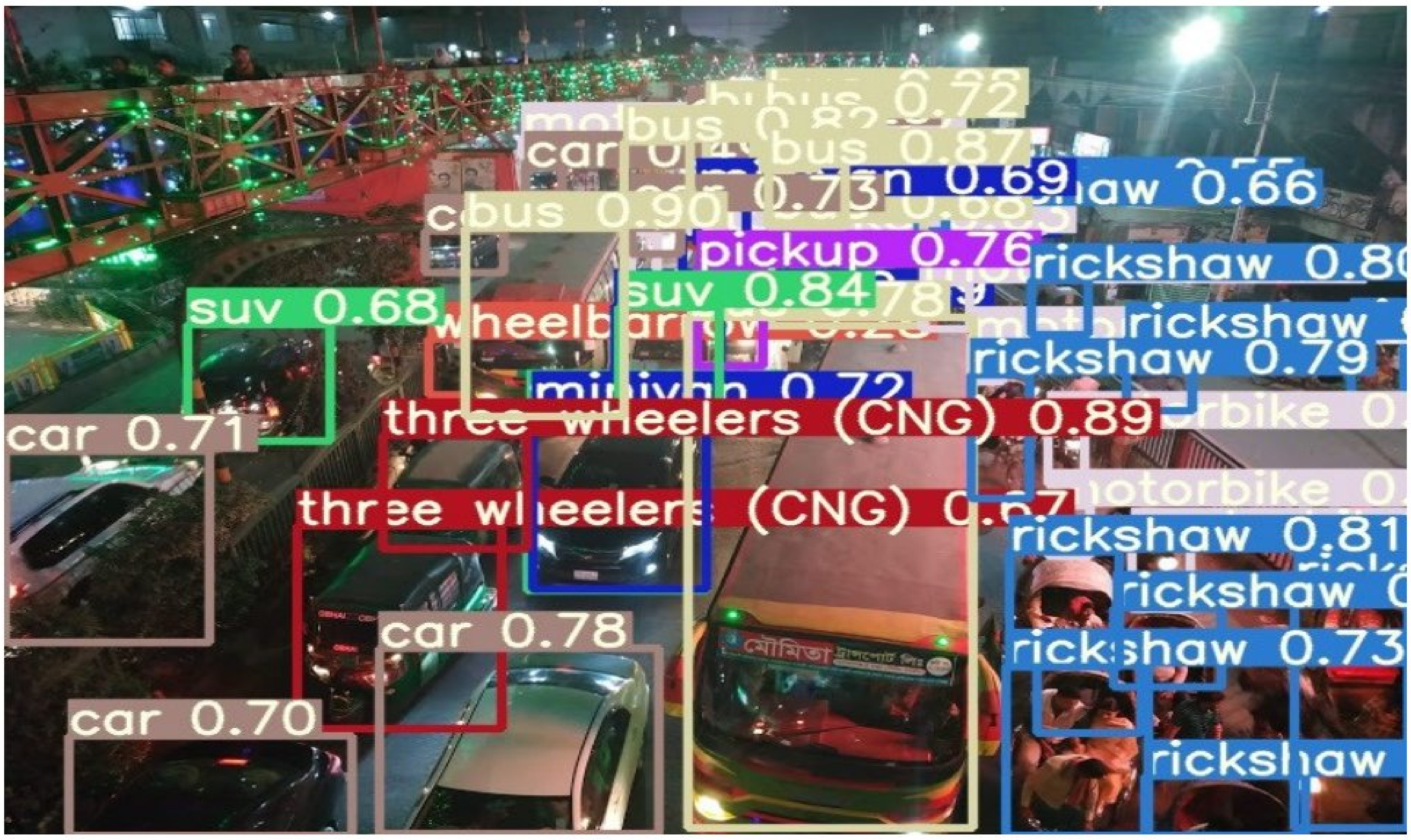


Figure 5.8: Yolov5 on Image 2

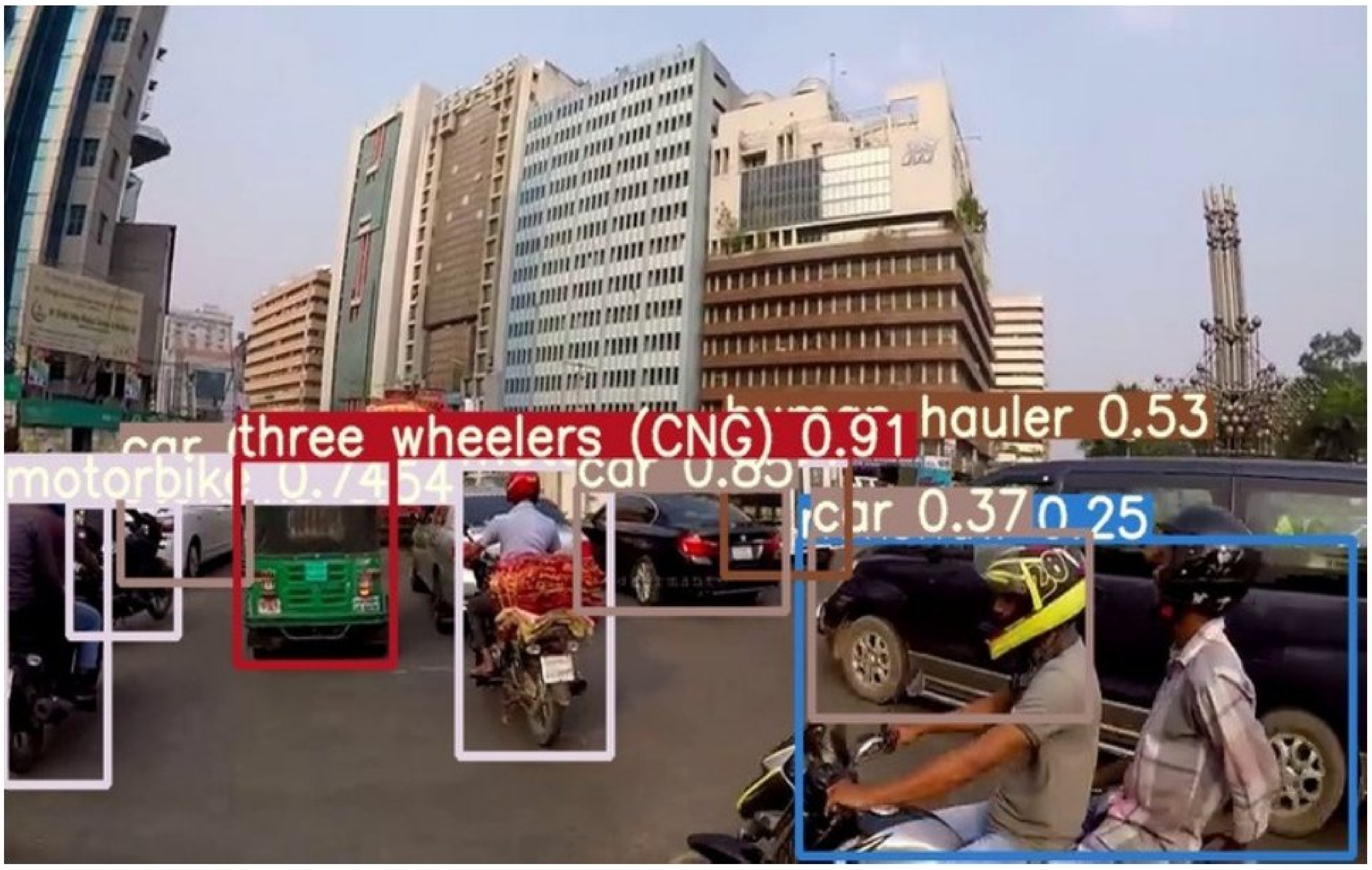


Figure 5.9: Yolov5 on Image 3

5.2 Inference on Videos

We also streamed video to our model from a smartphone camera used as a car's dash cam and ran inference to observe how YOLO performed. Some of the clips were recorded and uploaded on YouTube as unlisted videos. The links are given below with QR codes:

YOLOv3	YOLOv4-tiny	YOLOv5
https://youtu.be/ZAAo5-T-PAY	https://youtu.be/K4hnuzYWOIA	https://youtu.be/zBo9V2kTeVo

5.3 Comparison of Results

The following Tables (5.1, 5.2 and 5.3) and Figures (5.10 and 5.11) show the comparison between different YOLO versions based on the three Images.

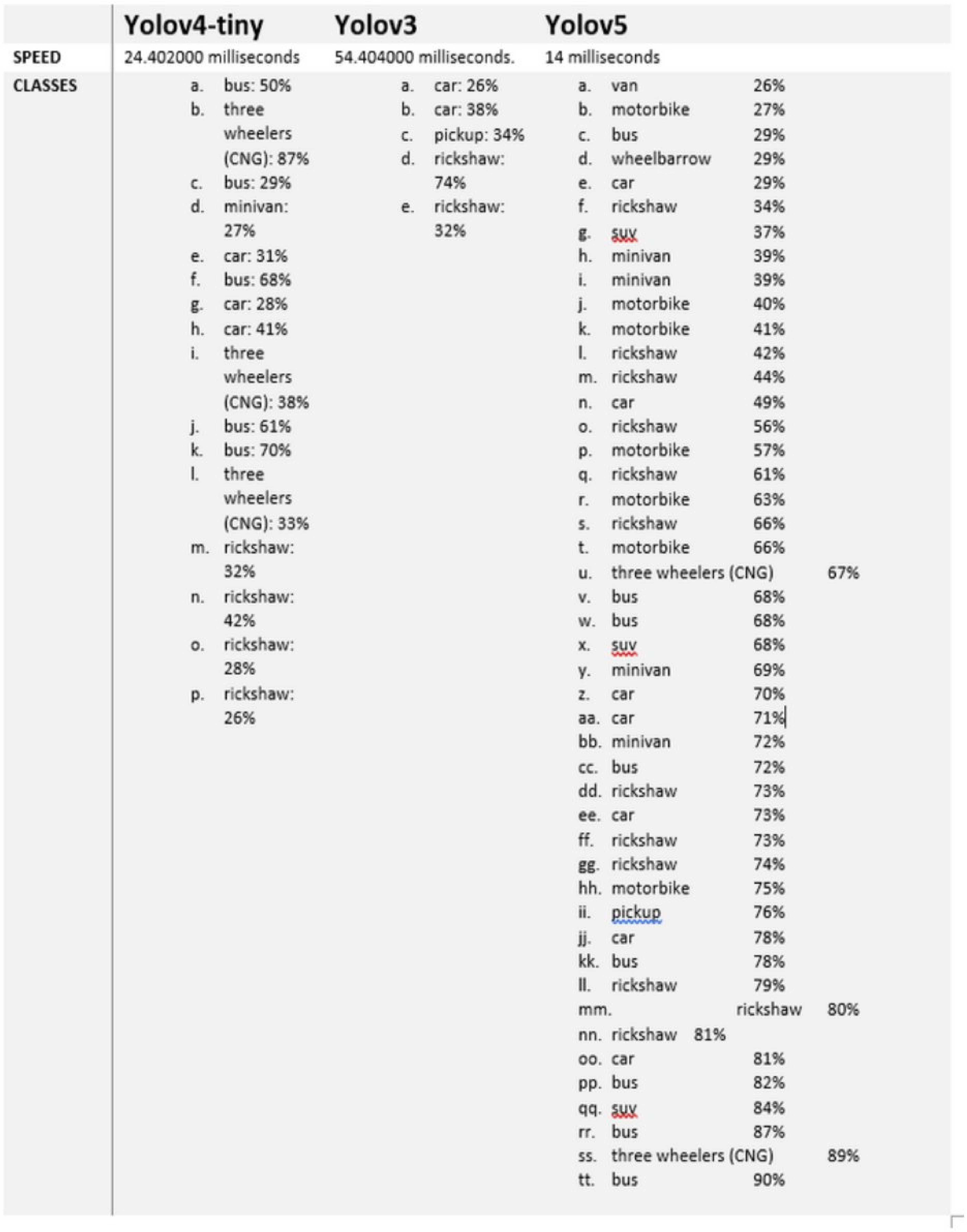
Table 5.1: Comparison between the YOLO versions on Image 1

	YOLOV4-TINY	YOLOV3	YOLOV5
SPEED	476.352000 milliseconds	546.863000 milliseconds	41 milliseconds
CLASSES	a. pickup: 70%	a. pickup: 78%	a. bus 26%
	b. bus: 61%	b. truck: 56%	b. bus 28%
	c. bus: 28%	c. motorbike: 66%	c. car 30%
	d. motorbike: 66%	d. bus: 97%	d. bus 33%
	e. bus: 100%	e. bus: 76%	e. bus 39%
	f. bus: 84%	f. bus: 49%	f. bus 40%
	g. bus: 96%	g. rickshaw: 60%	g. pickup 47%
	h. bus: 33%	h. bus: 94%	h. bus 48%
	i. bus: 68%	i. bus: 54%	i. bus 49%
	j. rickshaw: 64%	j. rickshaw: 52%	j. bus 52%
	k. bus: 73%		k. bus 57%
	l. suv: 41%		l. bus 59%
	m. bus: 83%		m. rickshaw 60%
	n. rickshaw: 37%		n. bus 62%
			o. car 63%
			p. motorbike 68%
			q. suv 70%
			r. bus 72%
			s. bus 73%
			t. bus 74%
			u. bus 76%
			v. bus 76%
			w. pickup 81%
			x. bus 85%

Table 5.2: Comparison between the YOLO versions on Image 3

	YOLOV4-TINY	YOLOV3	YOLOV5
SPEED	24.139000 milliseconds.	55.483000 milliseconds.	14 milliseconds
CLASSES	a. motorbike: 57%	a. motorbike: 73%	a. rickshaw25%
	b. motorbike: 74%	b. motorbike: 70%	b. car 37%
	c. car: 96%	c. car: 91%	c. human hauler 53%
	d. three wheelers (CNG): 98%	d. three wheelers (CNG): 99%	d. motorbike 54%
	e. truck: 81%	e. truck: 88%	e. car 66%
	f. car: 35%	f. car: 60%	f. motorbike 74%
	g. car: 64%	g. motorbike: 75%	g. motorbike 84%
	h. motorbike: 86%	h. car: 99%	h. car 85%
	i. pickup: 63%	i. motorbike: 36%	i. three wheelers (CNG) 91%
	j. car: 86%	j. van: 35%	
	k. pickup: 26%		
	l. minivan: 92%		

Table 5.3: Comparison between the YOLO versions on Image 2



	Yolov4-tiny	Yolov3	Yolov5
SPEED	24.402000 milliseconds	54.404000 milliseconds.	14 milliseconds
CLASSES	a. bus: 50%	a. car: 26%	a. van 26%
	b. three wheelers (CNG): 87%	b. car: 38%	b. motorbike 27%
	c. bus: 29%	c. pickup: 34%	c. bus 29%
	d. minivan: 27%	d. rickshaw: 74%	d. wheelbarrow 29%
	e. car: 31%	e. rickshaw: 32%	e. car 29%
	f. bus: 68%		f. rickshaw 34%
	g. car: 28%		g. suv 37%
	h. car: 41%		h. minivan 39%
	i. three wheelers (CNG): 38%		i. minivan 39%
	j. bus: 61%		j. motorbike 40%
	k. bus: 70%		k. motorbike 41%
	l. three wheelers (CNG): 33%		l. rickshaw 42%
	m. rickshaw: 32%		m. rickshaw 44%
	n. rickshaw: 42%		n. car 49%
	o. rickshaw: 28%		o. rickshaw 56%
	p. rickshaw: 26%		p. motorbike 57%
			q. rickshaw 61%
			r. motorbike 63%
			s. rickshaw 66%
			t. motorbike 66%
			u. three wheelers (CNG) 67%
			v. bus 68%
			w. bus 68%
			x. suv 68%
			y. minivan 69%
			z. car 70%
			aa. car 71%
			bb. minivan 72%
			cc. bus 72%
			dd. rickshaw 73%
			ee. car 73%
			ff. rickshaw 73%
			gg. rickshaw 74%
			hh. motorbike 75%
			ii. pickup 76%
			jj. car 78%
			kk. bus 78%
			ll. rickshaw 79%
			mm. rickshaw 80%
			nn. rickshaw 81%
			oo. car 81%
			pp. bus 82%
			qq. suv 84%
			rr. bus 87%
			ss. three wheelers (CNG) 89%
			tt. bus 90%

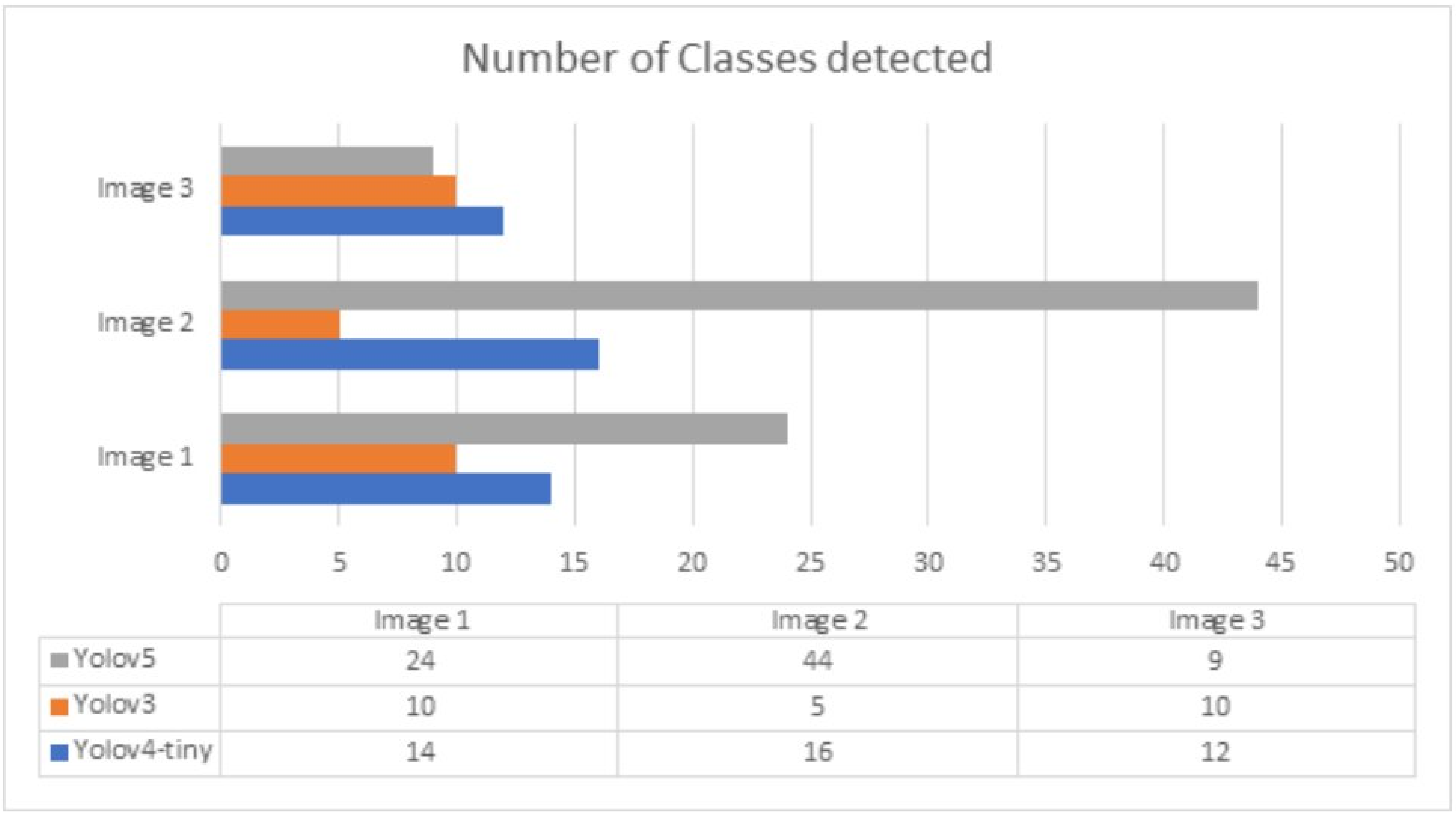


	Image 1	Image 2	Image 3
■ Yolov5	24	44	9
■ Yolov3	10	5	10
■ Yolov4-tiny	14	16	12

Figure 5.10: Class detection performance between different YOLO frameworks

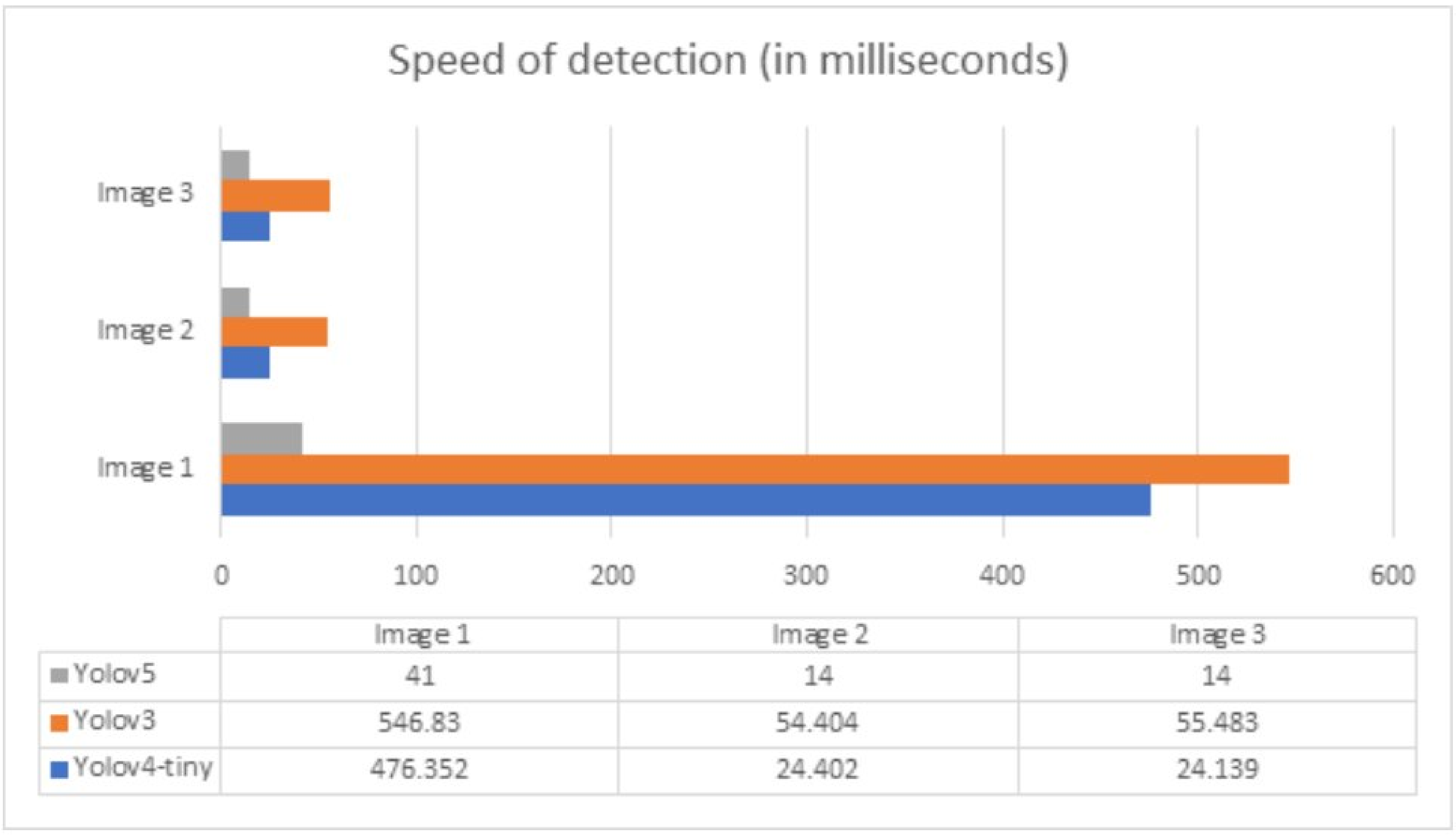


	Image 1	Image 2	Image 3
■ Yolov5	41	14	14
■ Yolov3	546.83	54.404	55.483
■ Yolov4-tiny	476.352	24.402	24.139

Figure 5.11: Speed of detection between different YOLO frameworks

5.4 Comparison Analysis

The figures above have been obtained using the different YOLO frameworks on our workstation and Google Colaboratory. We trained our models using the Dhaka-AI dataset. Using the trained weights, we evaluated our model for the different YOLO frameworks and documented the results obtained. We selected three images at random and then ran our YOLO frameworks on each and documented the results obtained. The images gave us bounding boxes around each object detected and named the class of the object. The probability of each bounding box was also mentioned and recorded for each class on each image.

The results clearly show the YOLOv5 framework to be much superior as compared to the YOLOv4-tiny and YOLOv3. The qualitative hypothesis was also true for the real time videos taken using the smartphone camera as the input. The videos can be accessed by scanning the QR codes or using YouTube links

For the quantitative analysis, we focused on the speed of the detection process and the number of classes each framework could detect accurately. The results were synonymous with our qualitative hypothesis and shows a numerical representation of the results obtained. These results were also visualized using bar charts so clearly show how the YOLO frameworks performed in comparison to each other. YOLOv5 lead the pack with YOLOv4-tiny taking the middle position and YOLOv3 coming in behind the other two.

The results we obtained were as expected from the paper "YOLOv4: Optimal Speed and Accuracy of Object Detection by Alexey Bochkovskiy, Chien-Yao Wang, Hong-Yuan Mark Liao". They used the MS COCO [54] dataset and obtained similar performance metrics as indicated in figure. The team at Ultralytics using YOLOv5 improved on this paper and incorporated YOLOv5 into their findings.

Chapter 6

Conclusion

6.1 Significance of the project

We believe our research to be a significant one in various aspects. In terms of scope, this project opens the possibility of implementing an intelligent traffic surveillance system in Bangladesh. Also autonomous cars will be able to recognize some of the unique vehicles which are still unknown to them that are there in Bangladesh.

In terms of resources, we trained our models and executed them in a moderately powered workstation and also in Google Colaboratory cloud, all of which are available for everyone to use.

In terms of time, the inferences were done pretty quickly, enabling us to produce real time results. But with a better GPU we could train our model fasters and also produce faster results.

In terms of scopes:

- Traffic management.
- Self-driving vehicles.
- Applicable for unique vehicles and congested traffic in Bangladesh.

In terms of resources:

- Trained and executed on Google Colab and a moderately powered workstation.
- A better dataset can help increase accuracy.

In terms of time

- Able to produce results in a matter of seconds.
- Possibility for detection and classification in real time.
- Stronger GPUs can significantly reduce training time.

6.2 Future Scopes

These section discusses the possible scopes of our project in future. We plan to improve the existing dataset to get more accurate results and add more classes like pedestrians and traffic signs. We also plan to use premium cloud training facilities, so that we can get unrestricted training time, which will improve our results more. As we've mentioned earlier, our research is a derivative work, but it is a fundamental block for traffic management and surveillance, and also in autonomous vehicles. Intelligent Parking systems can also be a possible application of our work.

- Real-time traffic management systems
- Autonomous vehicles
- Measurement of distance between vehicles
- Cloud-based detection and classification
- Automated parking systems
- Surveillance and monitoring

References

[1] Christou, Carol T., and Garry M. Jacyna. "Vehicle detection and localization using unattended ground magnetometer sensors." 2010 13th International Conference on Information Fusion. IEEE, 2010.

[2] Stiawan, Roni, et al. "An ultrasonic sensor system for vehicle detection application." Journal of Physics: Conference Series. Vol. 1204. No. 1. IOP Publishing, 2019.

[3] Park, Sang Jin, et al. "A novel signal processing technique for vehicle detection radar." IEEE MTT-S International Microwave Symposium Digest, 2003. Vol. 1. IEEE, 2003.

[4] Sun, Zehang, George Bebis, and Ronald Miller. "On-road vehicle detection using optical sensors: A review." Proceedings. The 7th International IEEE Conference on Intelligent Transportation Systems (IEEE Cat. No. 04TH8749). IEEE, 2004.

[5] Severdaks, Aivars, and Martins Liepins. "Vehicle counting and motion direction detection using microphone array." Elektronika ir Elektrotechnika 19.8 (2013): 89-92.

[6] Redmon, Joseph, et al. "You only look once: Unified, real-time object detection." Proceedings of the IEEE conference on computer vision and pattern recognition. 2016.

[7] https://github.com/pjreddie/darknet

[8] https://github.com/pytorch/pytorch

[9] https://en.wikipedia.org/wiki/Object_detection

[10] Girshick, Ross, et al. "Rich feature hierarchies for accurate object detection and semantic segmentation." Proceedings of the IEEE conference on computer vision and pattern recognition. 2014.

[11] Girshick, Ross. "Fast r-cnn." Proceedings of the IEEE international conference on computer vision. 2015.

[12] Ren, Shaoqing, et al. "Faster r-cnn: Towards real-time object detection with region proposal networks." arXiv preprint arXiv:1506.01497 (2015).

[13] https://towardsdatascience.com/r-cnn-fast-r-cnn-faster-r-cnn-yolo-object-detection-algorithms-36d53571365e

[14] Benjdira, Bilel, et al. "Car detection using unmanned aerial vehicles: Comparison between faster r-cnn and yolov3." 2019 1st International Conference on Unmanned Vehicle Systems-Oman (UVS). IEEE, 2019.

[15] Hu, Hou-Ning, et al. "Joint monocular 3D vehicle detection and tracking." Proceedings of the IEEE/CVF International Conference on Computer Vision. 2019.

[16] Foresti, Gian Luca, Vittorio Murino, and Carlo Regazzoni. "Vehicle recognition and tracking from road image sequences." IEEE Transactions on Vehicular Technology 48.1 (1999): 301-318.

[17] Sun, Zehang, et al. "A real-time precrash vehicle detection system." Sixth IEEE Workshop on Applications of Computer Vision, 2002.(WACV 2002). Proceedings.. IEEE, 2002.

[18] Sun, Zehang, George Bebis, and Ronald Miller. "On-road vehicle detection: A review." IEEE transactions on pattern analysis and machine intelligence 28.5 (2006): 694-711.

[19] Sang, Jun, et al. "An improved YOLOv2 for vehicle detection." Sensors 18.12 (2018): 4272.

[20] Xu, Zhi, et al. "Vehicle detection under uav based on optimal dense yolo method." 2018 5th International Conference on Systems and Informatics (ICSAI). IEEE, 2018.

[21] Chen, Shaobin, and Wei Lin. "Embedded system real-time vehicle detection based on improved YOLO network." 2019 IEEE 3rd Advanced Information Management, Communicates, Electronic and Automation Control Conference (IMCEC). IEEE, 2019.

[22] Ding, Xiangwu, and Ruidi Yang. "Vehicle and parking space detection based on improved yolo network model." Journal of Physics: Conference Series. Vol. 1325. No. 1. IOP Publishing, 2019.

[23] https://colab.research.google.com/

[24] Redmon, Joseph, and Ali Farhadi. "Yolov3: An incremental improvement." arXiv preprint arXiv:1804.02767 (2018).

[25] https://github.com/ultralytics/yolov3

[26] Bochkovskiy, Alexey, Chien-Yao Wang, and Hong-Yuan Mark Liao. "Yolov4: Optimal speed and accuracy of object detection." arXiv preprint arXiv:2004.10934 (2020).

[27] https://github.com/RenLuXi/tensorflow-yolov4-tiny

[28] https://github.com/ultralytics/yolov5

[29] https://arxiv.org/abs/1506.02640

[30] Simonyan, Karen, and Andrew Zisserman. "Very deep convolutional networks for large-scale image recognition." arXiv preprint arXiv:1409.1556 (2014).

[31] He, Kaiming, et al. "Deep residual learning for image recognition." Proceedings of the IEEE conference on computer vision and pattern recognition. 2016.

[32] Howard, Andrew G., et al. "Mobilenets: Efficient convolutional neural networks for mobile vision applications." arXiv preprint arXiv:1704.04861 (2017).

[33] Yolo v4 Object Detection - How it Works & Why it's So Amazing! https://youtu.be/_JzOFWx1vZg

[34] Lin, Tsung-Yi, et al. "Feature pyramid networks for object detection." Proceedings of the IEEE conference on computer vision and pattern recognition. 2017.

[35] Ma, Zheng, Ming Li, and Yuguang Wang. "PAN: Path integral based convolution for deep graph neural networks." arXiv preprint arXiv:1904.10996 (2019).

[36] Zhu, Lei, et al. "Bidirectional feature pyramid network with recurrent attention residual modules for shadow detection." Proceedings of the European Conference on Computer Vision (ECCV). 2018.

[37] Wang, Kaixin, et al. "Panet: Few-shot image semantic segmentation with prototype alignment." Proceedings of the IEEE/CVF International Conference on Computer Vision. 2019.

[38] He, Kaiming, et al. "Mask r-cnn." Proceedings of the IEEE international conference on computer vision. 2017.

[39] https://dhaka-ai.com/

[40] Shihavuddin, ASM; Mohammad Rifat Ahmmad Rashid, 2020, "DhakaAI", https://doi.org/10.7910/DVN/POREXF, Harvard Dataverse, V1

[41] https://www.kaggle.com/rifat963/dhakaai-dhaka-based-traffic-detection-dataset

[42] https://roboflow.com/

[43] https://github.com/numpy/numpy

[44] https://github.com/git/git

[45] https://github.com/Kitware/CMake

[46] https://developer.nvidia.com/cuda-zone

[47] https://developer.nvidia.com/cudnn

[48] https://github.com/opencv/opencv

[49] https://github.com/AlexeyAB

[50] https://visualstudio.microsoft.com/

[51] https://github.com/team

[52] https://stackoverflow.com/

[53] https://www.kaggle.com/

[54] Lin, Tsung-Yi, et al. "Microsoft coco: Common objects in context." European conference on computer vision. Springer, Cham, 2014.