Towards Solving Perception based Autonomous Driving Assistant System

by

Md. Arafat Hossain
17201079
Md. Sazidur Rahman
17201089
Md. Jisan Bin Islam
17201090
Sazzad Alam Bhuiyan
17201092

A thesis submitted to the Department of Computer Science and Engineering
in partial fulfillment of the requirements for the degree of
B.Sc. in Computer Science

Department of Computer Science and Engineering
Brac University
September 2021

Declaration

It is hereby declared that

1. The thesis submitted is our own original work while completing degree at Brac University.
2. The thesis does not contain material previously published or written by a third party, except where this is appropriately cited through full and accurate referencing.
3. The thesis does not contain material which has been accepted, or submitted, for any other degree or diploma at a university or other institution.
4. We have acknowledged all main sources of help.

Student's Full Name & Signature:

[illegible]	[illegible]
Md. Arafat Hossain 17201079	Md. Sazidur Rahman 17201089
Jisan	[illegible]
Md. Jisan Bin Islam 17201090	Sazzad Alam Bhuiyan 17201092

Approval

The thesis titled ”Towards Solving Perception based Autonomous Driving Assistant System” submitted by

1. Md. Arafat Hossain(17201079)
2. Md. Sazidur Rahman(17201089)
3. Md. Jisan Bin Islam(17201090)
4. Sazzad Alam Bhuiyan(17201092)

Of Summer, 2021 has been accepted as satisfactory in partial fulfillment of the requirement for the degree of B.Sc. in Computer Science on September 26, 2021.

Examining Committee:

Supervisor:
(Member)

Md. Khalilur Rhaman

Dr. Md. Khalilur Rahman
Designation
Department of Computer Science and Engineering
Brac University

Head of Department:
(Chair)

Sadia Hamid Kazi, PhD
Chairperson and Associate Professor
Department of Computer Science and Engineering
Brac University

Abstract

This thesis scrutinizes the problem of perception in the self-driving car system. Self-driving car is the face of the future and the decade's research focus. Tech giants like Google, Uber, Tesla, Commai, Intel MobilEye etc. are now immensely investing in this particular technology. In our work, we mainly address the perception problem of autonomous vehicle and try to solve it with only cameras and comparatively lower computational cost. Firstly, to detect the lane we propose QLD (Quick Lane Detection) model on CULane dataset which gives significantly improved results in the roads of countries like Bangladesh than other existing methods. Secondly, for object detection we propose our own dataset BDCO or Bangladeshi Common Objects, and merge it with MS COCO dataset to make it suitable for Bangladeshi roads. We train BDCO dataset in a CNN based object detection model (CbOD) which also gives very promising results in local roads. Finally, we cascade QLD and CbOD with our decision-making system which outputs the warnings based on the analysis of the inputs from cameras in the vehicle. Our hands-on evaluations show that, our cascaded network Bangladeshi Driving Assistant (BD-DA) attains performance competitive to the state-of-the-art systems on a indistinguishable benchmark.

Keywords: Machine Learning; Object detection; Prediction; Max pooling; Convolutional Neural Network

Acknowledgement

First of all, we are grateful to Almighty Allah for His blessings which helped us to complete our thesis without any interruptions.

Moreover, we would like to give our foremost gratitude and respect to our advisor Dr. Md. Khalilur Rahman Sir without whom we would not make any progress. He was by our side like a shadow whenever we hit a wall throughout our project.

And finally, special thanks to our parents. Without their constant help, support and prayers, we could not have come this far.

Contents

Declaration **i**

Approval **ii**

Abstract **iii**

Acknowledgment **iv**

Table of Contents **iv**

Nomenclature **1**

1 Introduction **2**
- 1.1 Problem Definition 4
  - 1.1.1 Scope of the Problem 4
    - 1.1.1.1 Position of the Camera 5
    - 1.1.1.2 Quality of the Camera 5
- 1.2 Method Outline 5
  - 1.2.1 Quick Lane Detection(QLD) 6
  - 1.2.2 CNN based Object Detection(CbOD) 6
  - 1.2.3 Bangladeshi Driving Assistant(Cascaded Netowrk) 6
- 1.3 Thesis Organization 6

2 Previous Work **7**
- 2.1 Driving Assistance System 7
- 2.2 Existing Object Detection Algorithm 8
  - 2.2.1 Fast Region-Based Convolutional Network 8
  - 2.2.2 Spatial Pyramid Pooling (SPP-net) 8
  - 2.2.3 Histogram of Oriented Gradients (HOG) 9
  - 2.2.4 Single Shot Detector 9
  - 2.2.5 You Only Look Once(YOLO) 10
- 2.3 Existing lane detection algorithm 11
  - 2.3.1 FOLOLane 11
  - 2.3.2 CondLaneNet 11
  - 2.3.3 LDNet 12
  - 2.3.4 Ultra Fast Lane Detection 13
- 2.4 Cascaded Algorithms 13
- 2.5 Deep Network 14
  - 2.5.1 Neural network history 14

2.5.2 Convolutional Neural Network 15
2.5.3 Recurrent neural network 16
2.6 Simulation software 17
2.6.1 CARLA 17
2.6.2 MATLAB/ Simulink 18
2.6.3 PreScan 19

3 Individual Network **21**
3.1 Overview of the model 21
3.2 Data Preprocessing 21
3.2.1 Data Collection 21
3.2.2 Data Labeling 22
3.2.3 Splitting Dataset 23
3.2.4 Training 23
3.3 Object Detection Model CbOD 24
3.3.1 Camera Calibration 24
3.3.2 Object Detection Architecture 25
3.3.2.1 Object window prediction 25
3.3.2.2 Class Prediction 26
3.3.3 Object Detection Algorithm 27
3.4 Lane Detection QLD 30
3.4.1 Lane detection architecture 30
3.4.2 Lane Detection Algorithm 33
3.5 Distance Measurement 34
3.6 Safe Zone Detection 34
3.7 Decision Making 36
3.8 Steering Angle Prediction 37

4 BD-DA Cascaded Network **38**
4.1 Cascaded Block 38
4.2 Loss Function 40

5 Test Run on Simulator **42**

6 Results and Experiments **44**
6.1 Lane Detection(QLD) 48
6.2 Output 49

7 Conclusion and Future Work **51**

Bibliography **55**

Appendix A **56**

Appendix B **57**

Chapter 1

Introduction

Driverless cars are ones that are controlled entirely by digital technologies that have been invented by researchers and engineers over the past hundred years which require no human involvement. They can drive and navigate on the roads by getting the effects and information of the surroundings. The whole design is intended to take up less road space, reducing traffic congestion and the risk of accidents. Inventing self-driving cars was as impossible as the invention of normal automobiles. It all started hundreds of years before even inventing normal automobiles when Leonardo De Vinci sketched a rough plan about vehicles without any driver. In 1925, Karl Benz fulfilled the dream of inventing the first true automobile by which all the dreams were coming true [15]. As early as 1925 Francis Udina demonstrated a driver-less car called "The American Wonder"[37]. The self-driving car dream was alive and well in 1956 claiming that GM self-driving cars would arrive by 1976. This journey started out with some early advances in Europe in the 1980s by Pioneer company. They had been working on self-driving cars for over 40 years. During the 90s there were many self-driving car prototypes like Eureka Prometheus, Navlab1, Navlab2, and modified Humvee.

Since the 90s many things have happened and in 2003 Toyota introduced first autonomous parking technology with its intelligent parking assist that helps driver park parallelly. Google got the first ever autonomous vehicle testing licence by which they could test autonomous vehicles on the roads which was given by the Nevada department of Motor vehicles in 2012. Almost a year after, Google introduced their firefly car which was independent up to 40 kilo-meters per hour. Passengers can sit back and enjoy the autonomous vehicle system and can unwind it with just one button if they feel unsafe. By doing this, it was revealed what the autonomous vehicle future looks like.Tesla was not far behind. Around the end of 2015, they presented program called autopilot. By 2016, they had demonstrated how autopilot may self-stop and could be stopped by a button. This represented as the biggest achievement that had mostly autonomous features at the time, and so Tesla rapidly stacked up millions of kilo-meters driven with their autopilot. After Tesla introduced their semi-autonomous car, more companies like Comma ai, Zoox, Uber ATG, ArgoAI etc. became more interested in this platform. Computerized vehicle's potential to save lives and decrease accidents is established in one awful truth which is 94% of genuine crashes happen because of human mistake. Autonomous vehicles have the option to end human mistakes from the crash condition, which offers assistance to secure passengers and drivers as well as bicyclists and people on foot.[34] Autonomous vehicles seem to provide extra financial and extra societal benefits.

Levels of Autonomy

It requires extensive research and time to upgrade the automation level of the self-driving cars. As of now some companies like Tesla and Waymo achieved level 3 of automation after years of research and a million miles of testing. It is still unsure of when the companies will reach their goal to upgrade the automation level of 4 and 5. The table 1.1 below shows the levels and features of the automation system:

Levels	Name	Features	image
Level 1	**Driver Assistance**	The first level of automation requires human involvement to the fullest. The whole system is a single mechanized framework which helps the driver to control steering and acceleration which is also known as cruise control. This level of autonomous system only holds the vehicle in a safe distance while the driver control all the other driving components.	
Level 2	**Partial Driving Automation**	As it is level 2 of the autonomous driving system, it can perform advanced driver assistance which is better than level 1. Accelerating, decelerating and steering can be controlled at this level. Although human drivers still monitor all the functions performed by the system and it can be handled manually if the driver or passenger feel unsafe.	
Level 3	**Conditional Driving Automation**	This category of vehicles have all the surrounding environmental data. With the help of these data the vehicle can drive on the road with some restrictions. It can help human drivers by doing lane following, overtaking slow moving vehicle or perk the car itself, but Human attention is still required.	
Level 4	**High Driving Automation**	In level 4 automation the vehicle can drive on it's own without any help from humans. It can drive in complex situations and drive accurately by following the traffic rules. However a human can override the vehicle control manually if it shows any error.	
Level 5	**Full Driving Automation**	In level 5 automation the vehicle does not involve any human interaction on driving. It can drive in any kind of scenario and terrain. This type of vehicle does not need to have steering wheels or acceleration/brake pedals.	

Table 1.1: Levels of Autonomy

Challenges

A NHTSA study shows that engine vehicle crashes in 2010 fetched $242 billion in financial action, counting $57.6 billion in misplaced work environment efficiency, and $594 billion due to misfortune of life and diminished quality of life due to injuries [34]. In Bangladesh, the rate of road accidents are so high that more than 60 people die for every 10,000 vehicles on the road. According to government data, around

8 people face death everyday in car accidents[9]. From 1992 to 2016, 55834 people have lost their lives only from car accidents [11],[36],[38]. Even in recent years the rate of road accidents is increasing rapidly in our country. Bangladesh is a country where traffic is so dense and most of the times the roads are under construction. Making an autonomous driving assistance system under these circumstances is quite hard. Most of the drivers in our country are uneducated who do not follow any traffic rules which leads to reckless driving and road accidents. Taking these as concerns we came up with an idea to reduce road accidents and make autonomous driving systems available for the people of Bangladesh.

Contribution

- We have created a dataset suitable for Bangladesh.
- We introduce a efficient and faster way to detect lanes in Bangladesh.
- We propose a convolutional neural network based object detection that achieves improved detection accuracy.
- We have cascaded all the improved individual networks and implemented some important useful features to our final network which works as a driving assistance system.

1.1 Problem Definition

We are aiming to improve the autonomous driving system for countries like Bangladesh where traffic is more dense and tends to violate traffic laws more often. In Bangladesh most of the roads are under construction, traffic signs and signals are not managed properly. So other proven and state-of-the-art networks and models struggle to work properly in this condition[14]. Also there are some different types of traffic which are not added to any large-scale object detection dataset. Besides there are a lot of roads which do not have clear road lanes for autonomous driving systems to detect and drive through the road. Below we have discussed some desired characteristics of the solution:

- Giving real time warning to driver if a vehicle is in close distance.
- More robust and user friendly for drivers in Bangladesh.
- Accurate decision making by using only cameras.

1.1.1 Scope of the Problem

We have made some assumptions to reduce the scope of our problem. Firstly, our system takes video input and detects lane but if the lanes are not clearly visible the accuracy rate of lane detection reduces and in some cases it fails to detect any lanes. Secondly, we could not use more than 3 cameras because of resource limitations. The more camera we will add the more accurate decisions the system can give. We further discuss some other assumptions that are required to mention.

1.1.1.1 Position of the Camera

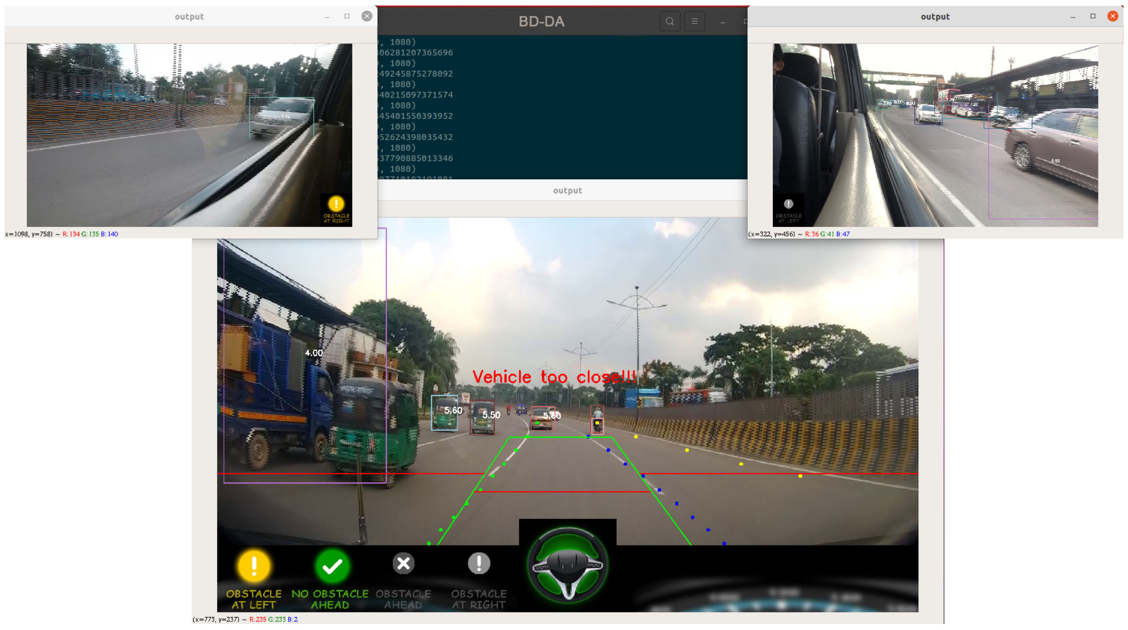
To take the full advantage of our system, the cameras need to be positioned in specific places. The main camera will be in front, the left camera will in the front left of the car and the right camera should be placed in the front right of the car. If the cameras are placed in different places other than these 3 places the system may not give the best result. In some cases the system might fail. For this we need to calibrate the system after setting the cameras in the car.

1.1.1.2 Quality of the Camera

The video quality of the cameras must be excellent. The cameras should produce at least 1280X720p resolution of high quality videos to make our system work. The cameras which produce 1920X1080p resolution high quality video will give the best result. Other than these, the system may work poorly or in some conditions may fail.

1.2 Method Outline

We want to create driving assistance which can work on Bangladeshi roads and assist the driver and passenger to drive more safely and accurately. In our work we want to show how our object detection and lane detection works against the other state-of-the-art algorithms in the context of Bangladeshi roads. The system works fine with a single front view camera input but if we use two more cameras with left and right view it increases the accuracy. In the following section we will discuss the following section briefly.



```
, 1080)
06281207365696
, 1080)
49245875278092
, 1080)
40215097371574
, 1080)
45401550393952
, 1080)
52624398035432
, 1080)
37790885013346
, 1080)
```

Figure 1.1: Output of the whole system

1.2.1 Quick Lane Detection(QLD)

It took the video frame from our input camera or cameras and detected the lane using our QLD algorithm. We divide the input video into two branches- main and auxiliary branches. The main branch detects the lanes in residual blocks and group classification and in the auxiliary branch it converts all the lane data into a lane which only shows the lane in a dark frame. Later it merges and forms a complete lane detection frame as an output frame.

1.2.2 CNN based Object Detection(CbOD)

It uses the output of QLD as an input for CbOD. In this the image frames are divided into three different resolutions- small, medium and large frame. Smaller frames are used to detect large objects and larger frames are used to capture the small objects. Then it up scales the image with the detected data as an output frame.

1.2.3 Bangladeshi Driving Assistant(Cascaded Netowrk)

Our network first detects the lane then it detects the object on the lane detection data. After getting the lane and object detection and data it applies some functions and calculations to guide the car's steering wheel, measure the distance of other objects from the host car and identify the safe zone. It also gives warning and some information on the display for the drivers and passengers.

1.3 Thesis Organization

Our thesis is organized in the following manner- Firstly in chapter 2 we talk and assess about the previous work done on the object and lane detection and how other researchers approached to solve different problems and overall literature. In chapter 3 we discuss our individual network for object detection(CbOD), lane detection(QLD), steering control, distance measurement and safe zone identification. After that in chapter 4 we show how we merge all the individual networks and form the whole network(BD-DA). In chapter 5 we test our whole network in the CARLA simulation environment. In chapter 6 we discuss the pros and cons about our network. We also show comparisons between similar state-of-the-art networks and ours. Finally in chapter 7 we emphasize our main contributions and talk over possible future improvements.

Chapter 2

Previous Work

Since Karl Benz produced the first automobile in 1885, the dream of autonomous driving has been on people's mind. A lot of experiments and researches have been put through to redeem progress on driving without any drivers on board. Achieving a fully autonomous system is still yet to come. Researchers around the world are trying for many years to accomplish this invention and put through a lot of work to make it happen.

2.1 Driving Assistance System

The idea of a driving assistance system was first introduced in the 1950's[35]. The early driving assistance system mainly worked on mechanical level. As the technology evolved, it became more software circuit based and later on artificial intelligence was introduced. In 2000's car companies started to implement adaptive cruise control with the help of equipment that are based on laser and radar. It became a new method where vehicles can connect to other hosts wireless to improve the efficiency which is called cooperative adaptive cruise control[8]. In the next 10 years when the image recognition technology became more advanced, the companies and researchers started to use camera and video input for driving assistance systems. In recent years, many companies started to use machine learning with multiple sensors that are combined with radar, sonar, and camera with the adaptive method to improve the driving assistance system[51]. There are some interesting approaches to make a driving assistance system by using only cameras and machine learning language. One of the most successful one is comma ai's autonomous driving assistance system[26]. They have implemented 3 cameras for 360° view and have used machine learning to implement the system in any modern car with cruise control. Using only a camera as an input device is more economical but it has some drawbacks too. Camera based driving assistance has some limitations against different weather conditions and it does not provide top notch performance for distance and obstacle measurement. To improve this issue, the combination of camera and radar works tremendously well. Radar and camera fusion systems eliminate each other's limitations and increase reliability and accuracy.

2.2 Existing Object Detection Algorithm

Object detection is such a complex and comprehensive improvement in recent discoveries that helps to localise any objects in computer vision and divide each of them into different classes based on their categories. Detecting objects and predicting their position is called object localization and dividing them based on their categories is known as object classification. There are various methods to detect objects that are being used widely because of their accuracy are mentioned below:

2.2.1 Fast Region-Based Convolutional Network

Ross Girshick [18] from microsoft research proposes a way to improve R-CNN for object detection. Fast R-CNN takes an image with some expected object proposal set for an input. The network produces a convolutional feature map which has max pooling and convolutional layers. Then from that feature map, it takes out the feature vector with fixed length according to the object proposal. Each feature vector

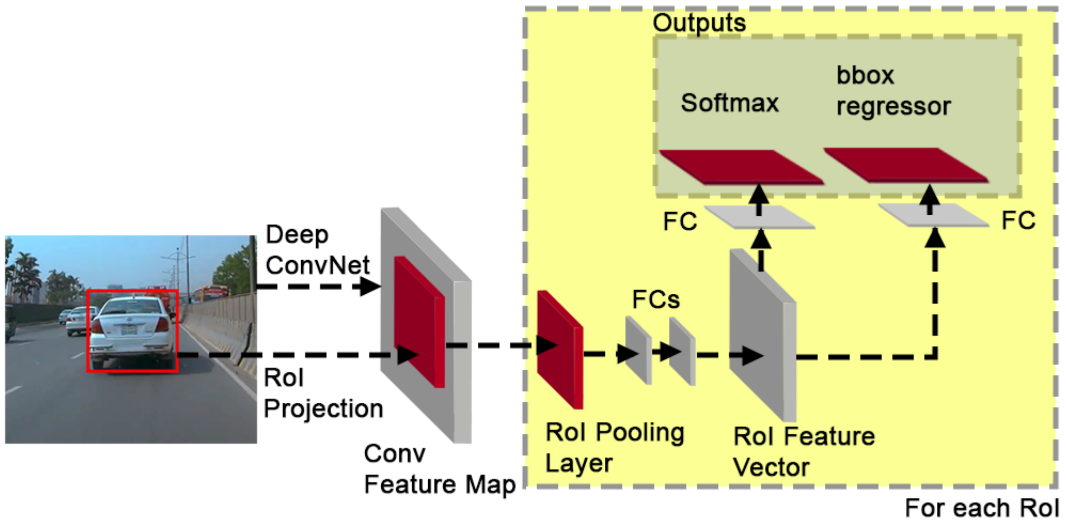


Figure 2.1: Fast R-CNN.

contains a softmax and a real valued branch. With these efficient tweaks Grishick's model has improved detection quality (mAP) than SPPnet and R-CNN. It also has a single stage training which updates all network layers and no separate disk space is required. In figure 2.1 is the architecture of Fast R-CNN, the image along with multiple RoIs(region of interest) are taken as an input for a fully convolutional network. Every region of interest(RoI) is pooled into a feature map and feature vector. Fast R-CNN network has two output- bounding box regression and softmax probabilities per RoI.

2.2.2 Spatial Pyramid Pooling (SPP-net)

Spatial pyramid pooling can be used for visual recognition as Kaiming He et al[16] have conducted various experiments on using spatial pyramid pooling to detect objects. SPP-net extracts the feature map from the whole image and applies spatial pyramid pooling on each candidate window of the feature maps to pool a fixed-length representation of that window. Figure 2.2 shows the architecture of SSP-Net. The entryflow takes the input image, after taking out the feature map it sends a sequence of CNNs by DU(downscaling unit) and UU(upscaling unit). As a result it outputs a refined supervised pose prediction from every prediction block. This method also

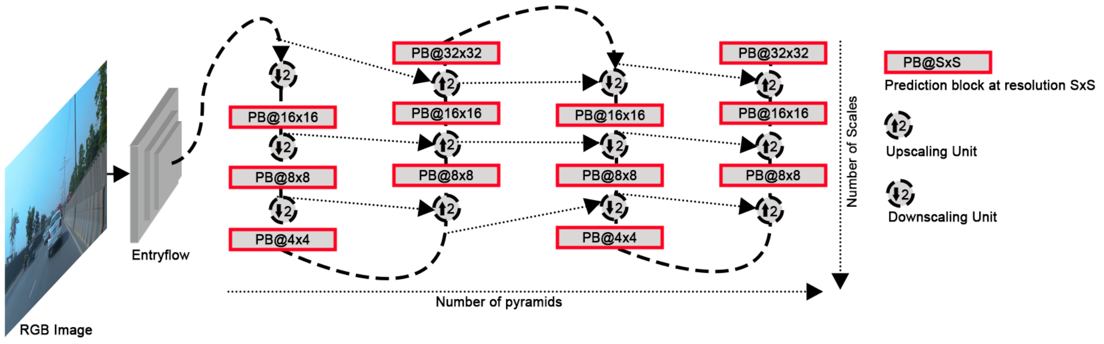


Figure 2.2: SPP-net.

extracts window-wise regions of the feature maps instead of extracting directly from image regions which is better than R-CNN. It avoids repetitive computation of the convolutional features. Despite having different sizes or scales, it can generate fixed-length representation of that specific window.

2.2.3 Histogram of Oriented Gradients (HOG)

HOG is an algorithm which loads an image as an input and returns feature vectors. It is an object detection algorithm for image processing and computer vision from Intel[30]. This method disectects the image into a cell and computes a gradient histogram in the cell. Each cell contains angular bins and gradient orientation. All

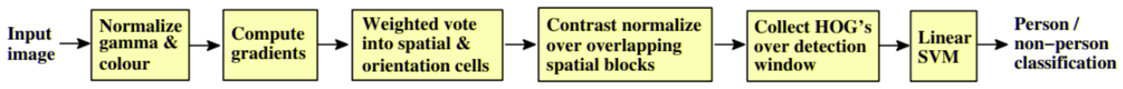

Input image
Normalize gamma & colour
Compute gradients
Weighted vote into spatial & orientation cells
Contrast normalize over overlapping spatial blocks
Collect HOG's over detection window
Linear SVM
Person / non−person classification

Figure 2.3: Histogram of Oriented Gradients.

the cells in an area with similar values build into a block which creates a block histogram to constitute a descriptor. In figure 2.3 It shows how HOG extracts feature maps and detects objects.

2.2.4 Single Shot Detector

Liu et al[21] introduced a method to optimize object detection within one neural network. This method has a convolutional filtered offset box and category predicting core which can predict different scales. It has a different aspect ratio for higher separated prediction.

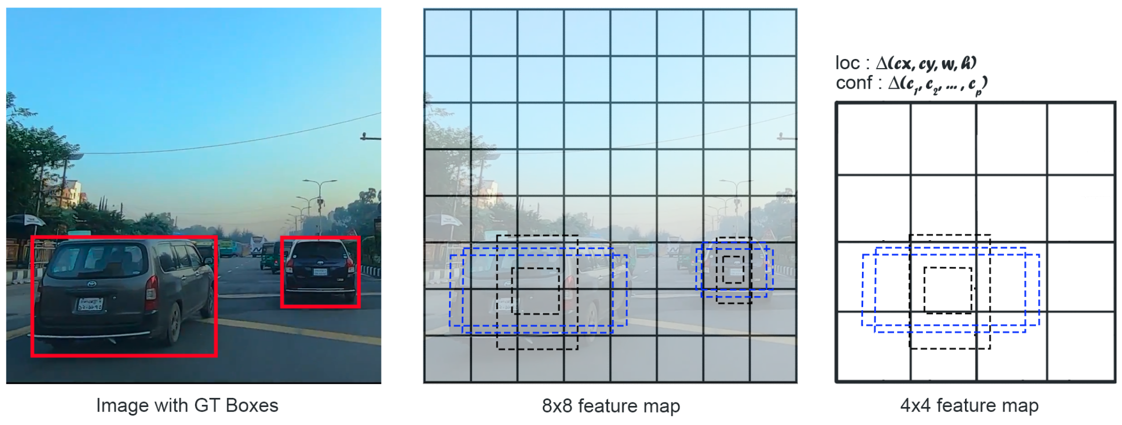
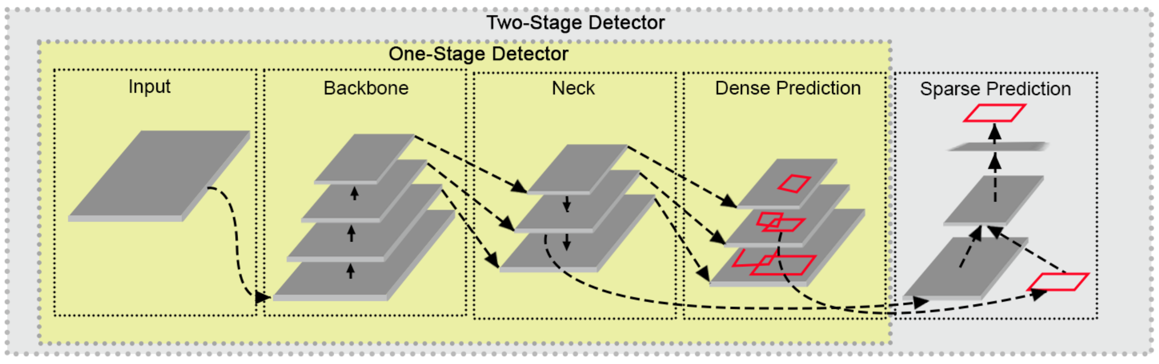

loc : $\Delta(cx, cy, w, h)$
conf : $\Delta(c_1, c_2, \dots, c_p)$
Image with GT Boxes
8x8 feature map
4x4 feature map

Figure 2.4: Single Shot Detector.

These design features yield a simple training with much higher accuracy in not only high resolution images but also low resolution images which improves the speed without reducing the accuracy.In figure 2.4 it shows how the SSD takes an input image and converts it into an 8x8 and 4x4 feature map by using convolutional fashion.

2.2.5 You Only Look Once(YOLO)

J. Redmon et al[22][25][32][44] introduced the real time, state-of-the-art object detection system. YOLO is very accurate and fast compared to other object detection systems. Their latest version, YOLOv4 uses CSPDarknet53 as its backbone which can outperform CenterNet, CornerNet, EfficientDet[44].



Input: { Image, Petches, Image Pyramid, ... }
Backbone: { VGG16, ResNet-50, ResNeXt-101, Darknet53, ... }
Neck: { FPN, PANet, Bi-FPN, ... }
Head:
  Dense Prediction: { RPN, YOLO, SSD, RetinaNet,FCOS, ... }
  Sparse prediction: { Faster R-CNN, R-FCN, ... }

Figure 2.5: YOLOv4 object detector.

It also uses BoF(Bag of Freebies) and BoS(Bag of Specials) to increase their accuracy along with a fast system. It also includes new features like Cross mini-Batch Normalization, Weighted Residual Connections, Self Adversarial Training, Cross Stage Partial, Mosaic data augmentation and many more.In figure 2.5 it shows the two stage object detector of the YOLOv4 algorithm. It takes an image input and passes it through a backbone, neck and head. Head contains dense prediction and sparse prediction.

2.3 Existing lane detection algorithm

In intelligent vehicle systems, lane detection is essential. The researchers around the globe have been using different algorithms to detect lanes in their system. In typical lane detection systems as input, the system uses pictures from a front-facing vision sensor mounted behind the windscreen and facing the road. The road model is then created using a variety of image processing techniques. There are various lane detection algorithms that has been used for many years because of their accuracy level and efficiency.

2.3.1 FOLOLane

Z. Qu et al[50] in their FOLOLane proposed a unique solution which works smarter than a lot of mainstream and complex lane detection. FOLOLane focuses on a specific area based on local curve estimation which is a local model geometry subtask.

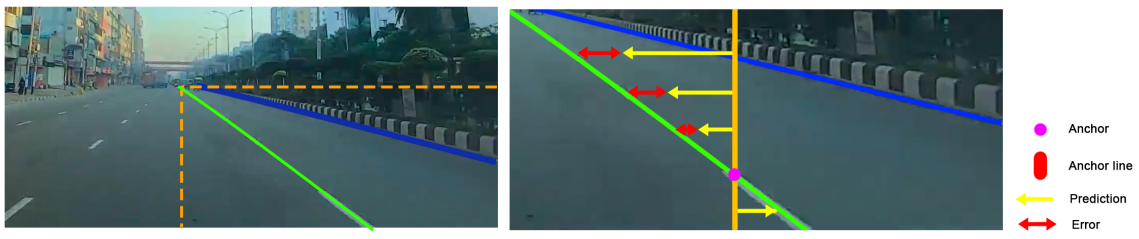


Figure 2.6: FoloLane.

It has two different algorithms which can decode different elements that are local information integration. This model can lead the system to gain a much higher accuracy in real time. The writers also showed that their method is outperforming the existing methods in every way. In figure 2.6 shows the influence of field of view(FOV) in lane detection on the basis of FOLOLane method.

2.3.2 CondLaneNet

L. Liu et al[49] presents a dynamic and instance-first method to detect lanes in a row-wise and conditional formation in their CondLaneNet paper. In a condensed lane scenario it uses complex topologies to fork down the lanes and detect it individually.

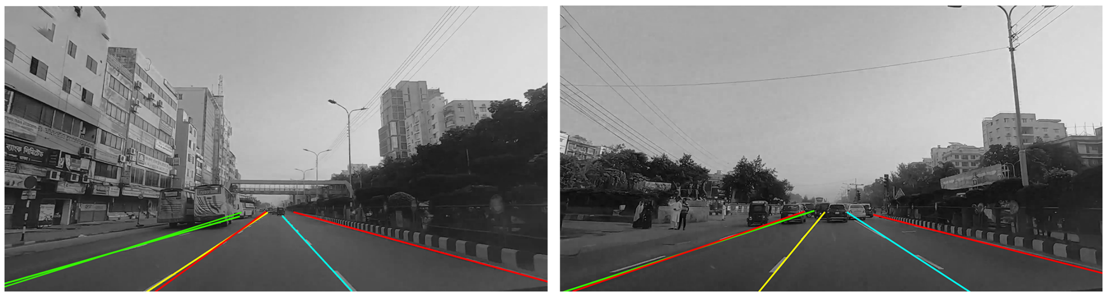


Figure 2.7: CondLaneNet.

By following these simple algorithms, CondLaneNet can achieve roughly 3.2-4.6% higher accuracy than all the popular lane detection algorithms. It can also detect objects in a much faster way by achieving great frames per seconds. In figure 2.7 it shows the lane line with a complex topology of CondLaneNet.

2.3.3 LDNet

F. Munir et al[45] brought forward a huge limitation of RGB cameras and solved it in their LDNet.RGB cameras are prone to sun glare, distinct illumination and motion blur. By using an encoder-decoder architecture it makes a RGB value input into B&W value and ASPP block prediction to predict a lane. In this process the decoder shows great improvement over state-of-the-art methods. In figure 2.8 shows the performance of the LDNet in lane detection after converting a RGB image into a black and white(B&W) image.

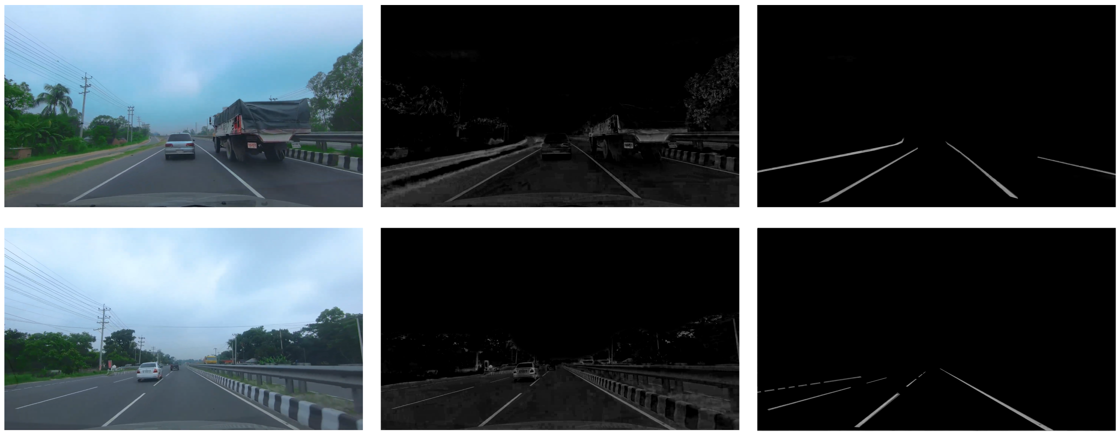


Figure 2.8: LDNet.

2.3.4 Ultra Fast Lane Detection

Z. Qin et al[46] introduced a method to optimize the system performance and fast rendering for lane detection. They select the estimated area for the lane and divide it into rows for fast iteration. It also uses global feature prediction to overcome the no-visual-input problem. Ultra fast lane detection has a structure loss to optimize the information of the lane. This method can achieve 4 times faster performance than state-of-the-art methods. In figure 2.9 it illustrates the lane detection with Ultra Fast Lane Detection with different colors.

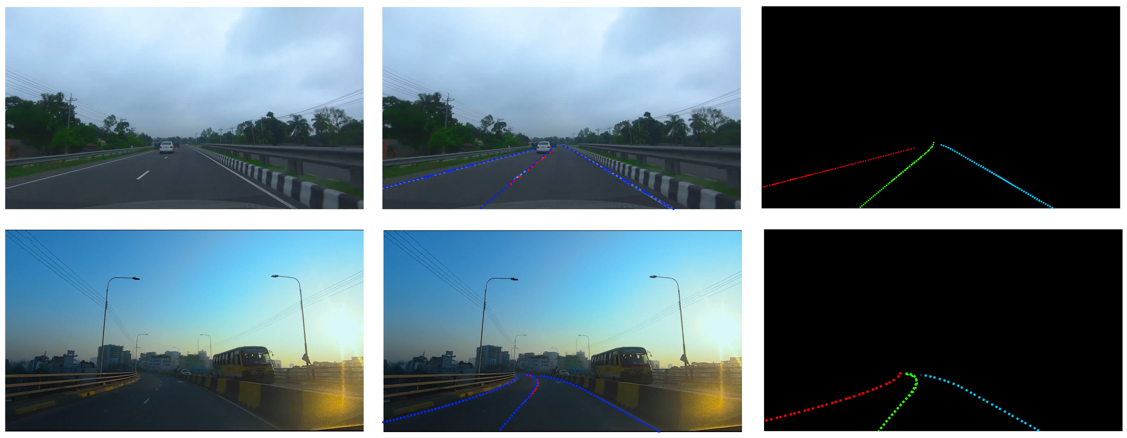


Figure 2.9: Ultrafast Lane Detection.

2.4 Cascaded Algorithms

There are a lot of variables to observe and calculate to run a car autonomously. Object detection, lane detection, steering and speed control and many different things come to the list. In this part, we are going to talk about some previous project about autonomous driving assistance where the researcher implemented multiple algorithms and ran their system. The Autonomous driving assistance system (ADAS) is implemented by many commercial car companies and many of them are available in many countries. One of the front runners in this race is Tesla Inc with their AUTOPILOT. They use camera-rader implementation and offer most of the adas features[48]. Ford automobiles also has a camera-radar system and offer adas features with their Co-Pilot 360. German car company Audi uses LiDAR along with camera and radar to operate their car with some of the ADAS features. There are many American, German and Chinese car companies that are focused on autonomous driving systems. Also tech giants such as Google and Apple are invested in the project of autonomous driving[41]. Along with the big budget tech and automobile companies many small start ups and projects are also focusing on autonomous driving. A. Shaout et al [12] talked about the embedded system which can be implemented as ADAS. D. Gonzalez et al[19] tested the previous method for motion plan and improved the consistency and efficiency of motion planning. B. Paden et al[24] also worked on motion planning and decision making to improve the driving assistance system. Then P. Scott et al[29] introduced their own algorithm for planning which leads to better performance. Also their control implementation and software leads them to run and test their vehicle in the real world with a better result. Furthermore, E. Yourtsever et al[29] implemented new design and architecture for their system and explored the sensor and hardware features to their fullest. They also introduced new ways for mapping, planning, localization and decision making. Along with many researchers, the project of G. Hotz and E. Santana's[22] CommaAI is one of the most successful one. They can use their camera module to control any modern car with cruise control that supports up to 100+ vehicles. They use open source software mainly developed by them which is called openpilot which is more likely to perform as good as tesla's Autopilot.

Figure 2.10: CommaAI.

2.5 Deep Network

Our system is based on recent works of deep neural networks. Our object detection uses the recent improvements on convolutional neural networks. In the following part we will briefly discuss the history of neural networks and research related to these different architectures and networks.

2.5.1 Neural network history

Human brains are so complex that it got the attention of many researchers and scientists who have been trying to replicate them for many decades. The first model that tried to copy the human brain was proposed by McCulloch and Pitts [4] in 1943. But it had major drawbacks because the technologies had not progressed that much. In 1958, Rossenblatt[6] proposed an algorithm which was based on the single-layer perceptron. But later in 1969, Minsky and Papert[1] argued about a shortcoming of the existing single-layer perceptron. They claimed that the perceptron algorithm had a major drawback because it could not handle Exclusive-Or(XOR) operations. They also mentioned the fact that it was impossible to perform computation for larger neural networks. For this reason, neural network research and artificial intelligence research got interrupted. But Paul Werbo's[2] came up with an invention that made the research keep going by solving the XOR computation problem. Rumelhart and McClelland[3] also ran experiments on simulated neural networks by using parallel processing. In the 90's, people became more interested in Support Vector Machines

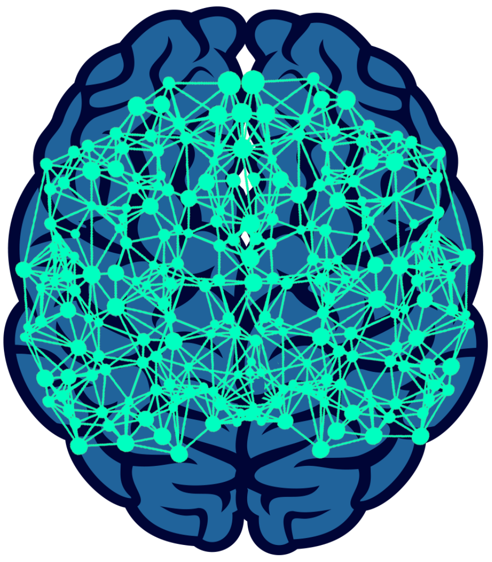


Figure 2.11: Brain neural network.

(SVM), random-forests etc. and for this reason the popularity of neural networks started to fade away. Schmidhuber[5] was the first to propose the idea of transfer learning. Stenkraus et al's[10] paper on fully connected networks was the first one to implement a neural network on Graphics Processing Unit (GPU). This created a lot of opportunities and interest for researchers around the scientific community. But later in 2012, when Alexnet[13] won the annual umagenet competition by using a deep neural network which was designed by Alex Krizhevsky. He was able to achieve only 16% error which is less than other existing methods. Since then, people from the computer vision community and other researchers have started to use deep neural networks.

2.5.2 Convolutional Neural Network

The convolutional neural network (CNN) has been one of the most important driving factors in computer vision and other artificial intelligence applications in recent years. A standard Convolutional Neural Network (CNN) [7] consists of many convolutional layers with padding and max-pooling layers between the convolutional layers, each with a variable size of kernel. It is a deep learning method that takes an input picture or other sequence of data and assigns significance to particular areas of the image using updatable weights, as well as extracting the required low and high level features for learning about the image. Because CNN is quicker and requires less preprocessing calculation, it is more effective than traditional hand-crafted features. The architecture of Convnet was inspired by our brain's neurons and visual cortex system. Each neuron or node is in charge of collecting certain characteristics and transmitting information to the neurons of the next layer over a limited area of the visual field known as the receptive field.

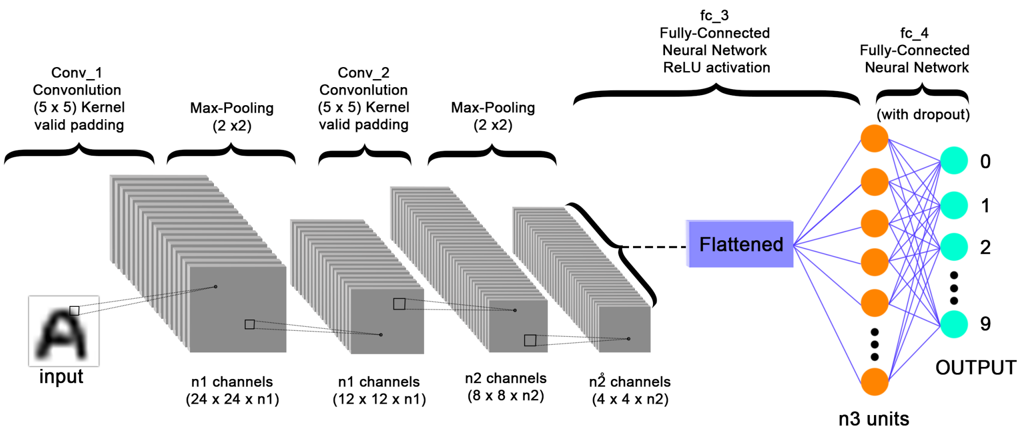


Figure 2.12: Convolutional neural network.

2.5.3 Recurrent neural network

For processing sequential data, the recurrent neural network (RNN) has proven to be highly successful. The term ”recurrent neural network” has been applied to two networks with identical design without distinction. One network is in charge of processing finite-length input data, while the other is in charge of processing infinite-length data, such as any temporal data, including electrical and acoustic data. A Recurrent Neural Network in its most basic form.[33] RNN is useful for capturing temporal data’s dynamic nature. Rumelhart’s work from 1986 influenced the development of early recurrent neural networks.

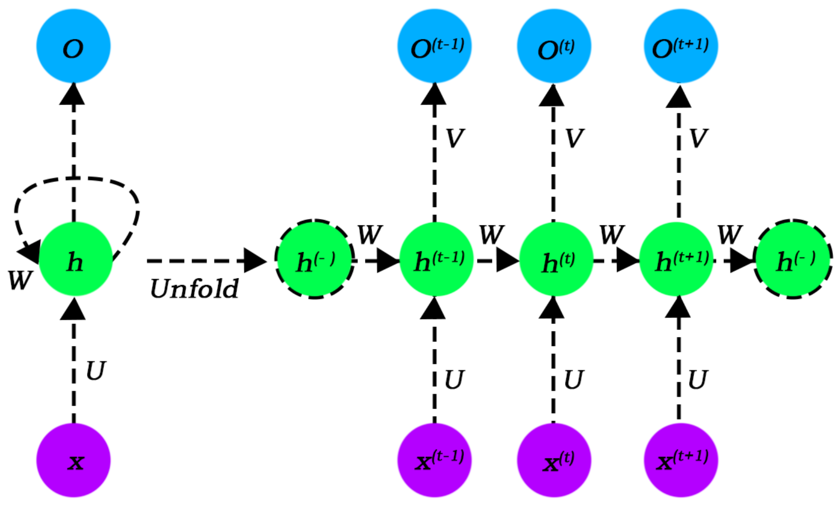


Figure 2.13: Recurrent neural network.

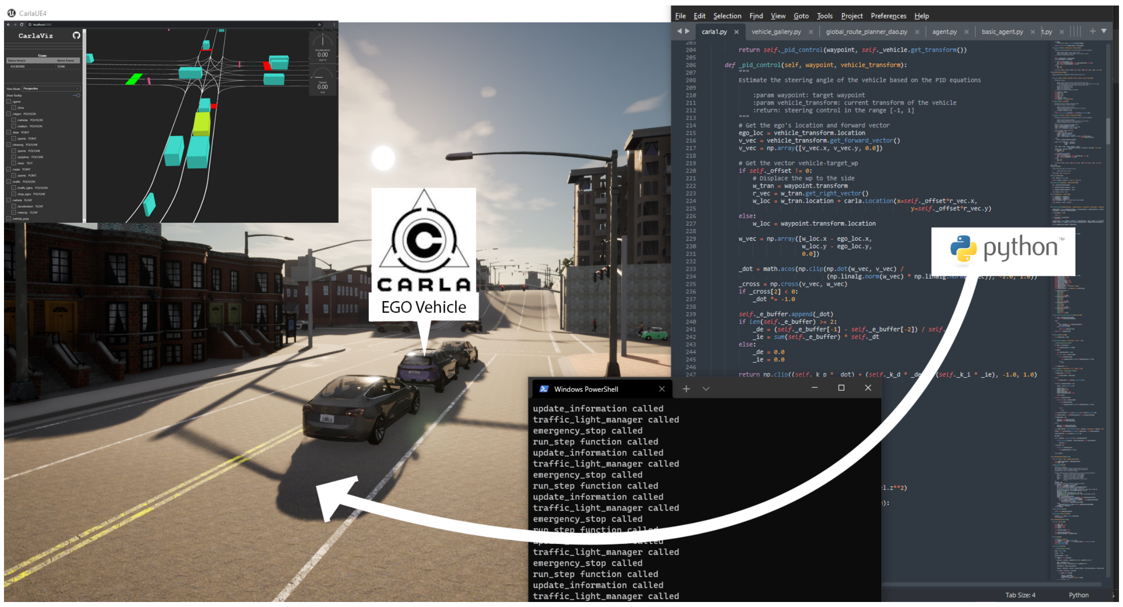
In 1982, John Hopfield presented a unique RNN version. Later in 1997 Hochreiter and Schmidhuber introduced LSTm which later became one of the most popular RNN's. RNN can access gradients and computation from previous layers to the current layer by using looping architecture which creates a new era in RNN on multiple domains.

2.6 Simulation software

The technology began to improve and researchers started to test those improvements. Some of the tests are far too dangerous to conduct in the real world. Thus the simulation introduced where all the variables are similar to the real world to conduct the simulation with safety. Driving an autonomous vehicle in the real world is also dangerous at its primitive stages. In this part we are going to discuss some simulation software where we can try our autonomous vehicle performance.

2.6.1 CARLA

CARLA is an open-source autonomous driving simulator. Unreal Engine, a open source engine is used to construct the simulator. It's a flexible, modular solution with a solid API for ADAS system training and certifications. CARLA is a simulation engine based on the Unreal Engine that leverages the OpenDRIVE standard to build roads and urban environments. The CARLA API may be customized by users, giving them control over the simulation. It is based on Python and C++ and is constantly evolving in tandem, which is an ecosystem of projects produced by the community around the primary platform. The client side is made up of client modules that control the logic of agents such as pedestrians, vehicles, bicycles, and motorbikes that appear in the scenario. The CARLA API is used to set up all of the



```python
        return self._pid_control(waypoint, self._vehicle.get_transform())

    def _pid_control(self, waypoint, vehicle_transform):
        """
        Estimate the steering angle of the vehicle based on the PID equations

            :param waypoint: target waypoint
            :param vehicle_transform: current transform of the vehicle
            :return: steering control in the range [-1, 1]
        """
        # Get the ego's location and forward vector
        ego_loc = vehicle_transform.location
        v_vec = vehicle_transform.get_forward_vector()
        v_vec = np.array([v_vec.x, v_vec.y, 0.0])

        # Get the vector vehicle-target_wp
        if self._offset != 0:
            # Displace the wp to the side
            w_tran = waypoint.transform
            r_vec = w_tran.get_right_vector()
            w_loc = w_tran.location + carla.Location(x=self._offset*r_vec.x,
                                                     y=self._offset*r_vec.y)
        else:
            w_loc = waypoint.transform.location

        w_vec = np.array([w_loc.x - ego_loc.x,
                          w_loc.y - ego_loc.y,
                          0.0])

        _dot = math.acos(np.clip(np.dot(w_vec, v_vec) /
                                 (np.linalg.norm(w_vec) * np.linalg.norm([illegible]ec)), -1.0, 1.0))
        _cross = np.cross(v_vec, w_vec)
        if _cross[2] < 0:
            _dot *= -1.0

        self._e_buffer.append(_dot)
        if len(self._e_buffer) >= 2:
            _de = (self._e_buffer[-1] - self._e_buffer[-2]) / self.[illegible]
            _ie = sum(self._e_buffer) * self._dt
        else:
            _de = 0.0
            _ie = 0.0

        return np.clip((self._k_p * _dot) + (self._k_d * _de[illegible](self._k_i * _ie), -1.0, 1.0)
```

```
update_information called
traffic_light_manager called
emergency_stop called
run_step function called
update_information called
traffic_light_manager called
emergency_stop called
run_step function called
update_information called
traffic_light_manager called
emergency_stop called
run_step function called
update_information called
traffic_light_manager called
emergency_stop called
run_step function called
update_information called
traffic_light_manager called
```

Figure 2.14: CARLA Simulator.

client modules. CARLA delivers open digital assets such as automobiles, buildings, and urban layouts. Furthermore, environmental conditions such as varying weather conditions and a changeable sensor suit specification are provided. CARLA uses RTrees to speed up searches and it has more accurate vehicle volumes and realistic fundamental physics. In addition, using the information provided by the OpenDRIVE file, the process of adding traffic lights and stop signs to the scene has been altered from manual to automatic. Based on the RSS library, CARLA presents a safety assurance module. The RSS module can evaluate different road segments using OpenDRIVE signals, which helps determine priority and safety at junctions.[27] Secondly, it employs the Object-Oriented Graphics Rendering Engine (OGRE), a rendering engine that enables the depiction of dynamic 3D objects and scenarios. Finally, the communication library allows various Gazebo pieces to communicate with one another. It's worth mentioning that Gazebo has a sizable community that allows you to share and use models created by others. It also comes with well-maintained documentation and several tutorials. Gazebo may also function as a standalone simulator. However, it is most often used in combination with ROS. Gazebo may be used to model almost any form of robot.[23]

2.6.2 MATLAB/ Simulink

The Automated Driving Toolbox is a collection of MATLAB/ Simulink-based tools for automated driving. It is used to come up with the design, simulation, and testing of advanced driver assistance systems (ADAS). It lets users put their observation, path planning, and vehicle control skills to the test. The ability to import HERE HD live map data and OpenDRIVE road networks into MATLAB for various design and testing reasons is a major advantage.

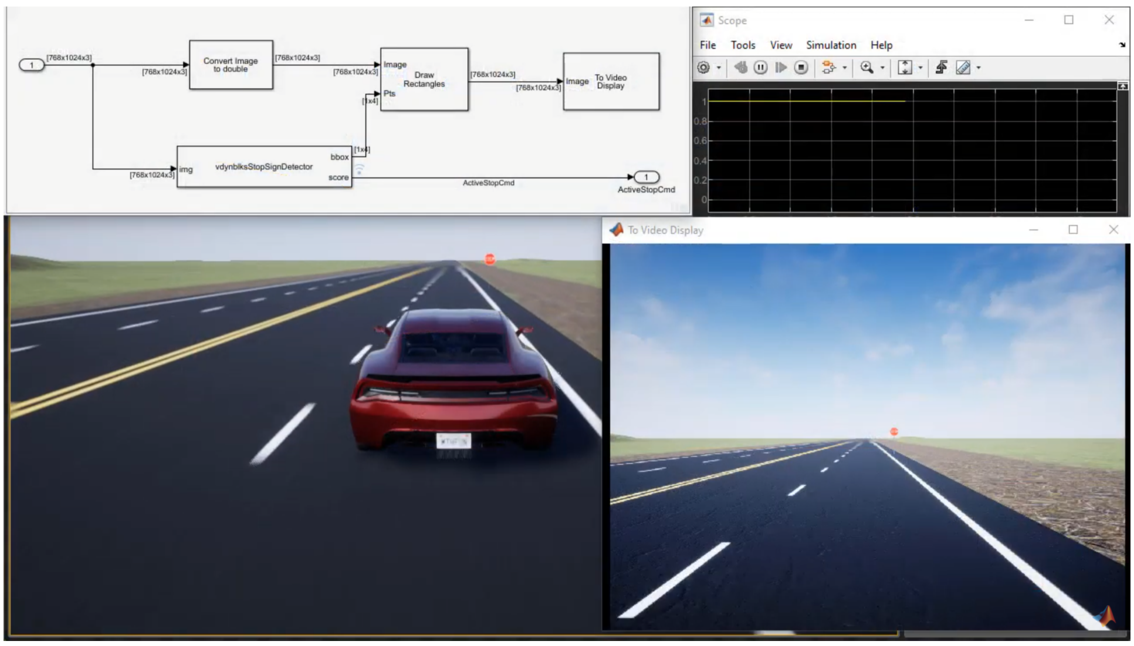


Figure 2.15: MATLAB/ Simulink.

There are options for users to create photo-realistic 3D settings and various sensors and also has a visualizer to see real time sensor detection of the system. The Ground Truth Labeler application to automate the labelling of objects in order to serve as a simulation and design based environment which helps to train or evaluate sensor performance. The toolbox supports Hardware In the Loop (HIL) testing as well as C/C++ code creation which helps to allow for faster prototyping.[47] Finally, MATLAB includes multiple examples for simulating ADAS features like Adaptive Cruise Control, Automatic Parking Assist, Automatic Emergency Braking, and others.

2.6.3 PreScan

PreScan is a simulation framework for ADAS and self-driving car development. Manufacturers may test their clever technology in a range of simulated traffic scenarios and realistic settings. This is where PreScan's automated traffic generator comes in handy. Customers may also design their own sensor suites, control logic, and collision detection features with PreScan 4. PreScan also allows for Hardware In the Loop (HIL) simulation, which is a common method of evaluating ECUs (ECU). PreScan excels in sensor input computations based on physics.

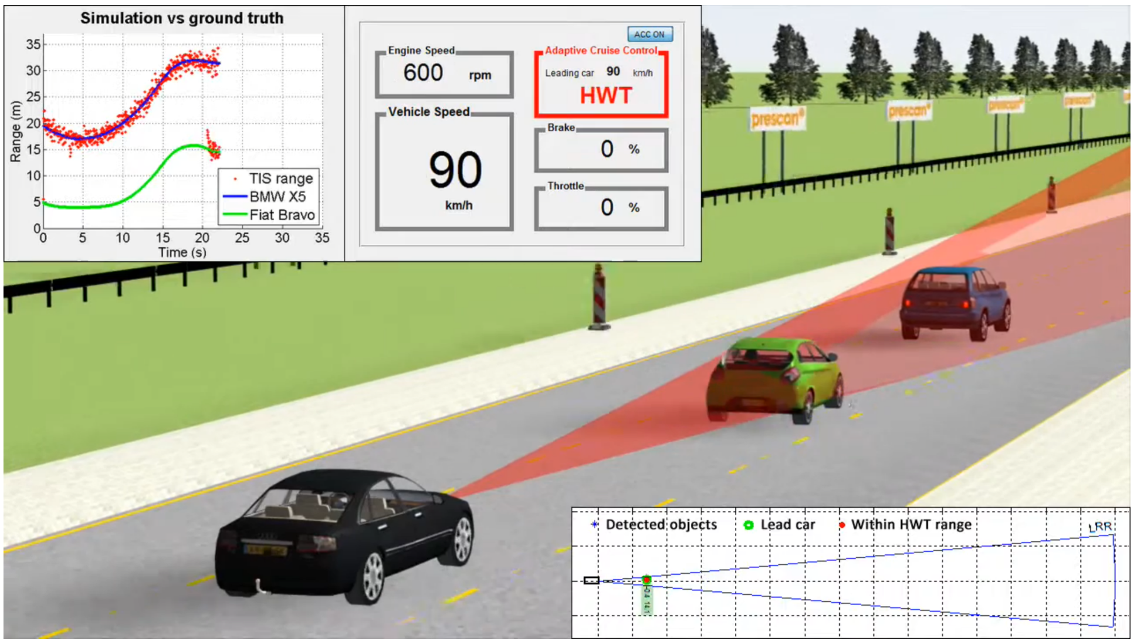
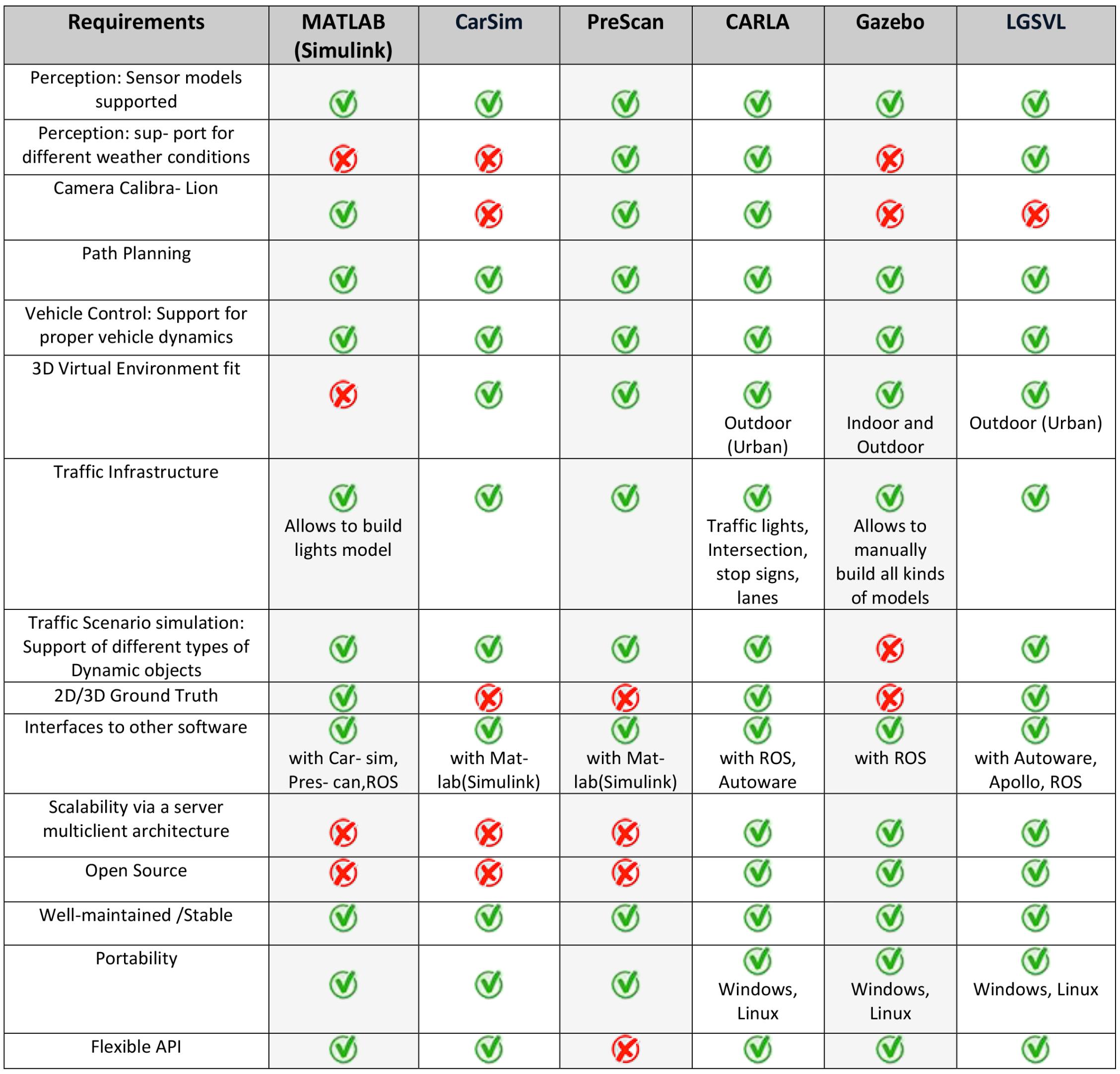


Figure 2.16: PreScan

The ECU receives the sensor signals and utilizes them to perform various algorithms. For driving reasons, the signals can also be transmitted to the loop or the camera HIL. It also enables the recording of real-time data as well as GPS vehicle data, which can subsequently be replayed. The Vehicle Hardware-in-the-Loop laboratory is another unique element of PreScan. VeHIL can provide detailed simulations for ADAS utilizing this mix of ego vehicle and mobile robots.[52]



Requirements	MATLAB (Simulink)	CarSim	PreScan	CARLA	Gazebo	LGSVL
Perception: Sensor models supported	✓	✓	✓	✓	✓	✓
Perception: sup- port for different weather conditions	✗	✗	✓	✓	✗	✓
Camera Calibra- Lion	✓	✗	✓	✓	✗	✗
Path Planning	✓	✓	✓	✓	✓	✓
Vehicle Control: Support for proper vehicle dynamics	✓	✓	✓	✓	✓	✓
3D Virtual Environment fit	✗	✓	✓	✓ Outdoor (Urban)	✓ Indoor and Outdoor	✓ Outdoor (Urban)
Traffic Infrastructure	✓ Allows to build lights model	✓	✓	✓ Traffic lights, Intersection, stop signs, lanes	✓ Allows to manually build all kinds of models	✓
Traffic Scenario simulation: Support of different types of Dynamic objects	✓	✓	✓	✓	✗	✓
2D/3D Ground Truth	✓	✗	✗	✓	✗	✓
Interfaces to other software	✓ with Car- sim, Pres- can,ROS	✓ with Mat- lab(Simulink)	✓ with Mat- lab(Simulink)	✓ with ROS, Autoware	✓ with ROS	✓ with Autoware, Apollo, ROS
Scalability via a server multiclient architecture	✗	✗	✗	✓	✓	✓
Open Source	✗	✗	✗	✓	✓	✓
Well-maintained /Stable	✓	✓	✓	✓	✓	✓
Portability	✓	✓	✓	✓ Windows, Linux	✓ Windows, Linux	✓ Windows, Linux
Flexible API	✓	✓	✗	✓	✓	✓

Figure 2.17: Comparison among various kinds of autonomous driving simulators.

Chapter 3

Individual Network

3.1 Overview of the model

Autonomous driving is now a common thing for any first world country. A third world country like Bangladesh is still seeing the dream of autonomous driving because of its unique environment.So we have tried to make a suitable autonomous driving system for our country. The whole model is based on cameras which use the input of video frames. First, we detect an object and it's position in the frame, then we are able to detect the lanes of the roads by using a lane detection algorithm. Our model can acknowledge the vehicle position prior to the lanes of the road which helps to measure the distance between our vehicle and other available vehicles on the road. From that result we were able to give suggestions of steering control and lane keeping. Moreover, we divided pixels of the video frame into different zones to provide awareness which helps the model to acknowledge all the safe zones possible and make decisions accordingly.

3.2 Data Preprocessing

3.2.1 Data Collection

The first step for data preprocessing is collecting data. As our main focus is on Bangladeshi roads, we needed a lot of pictures and videos of roads and vehicles in Bangladesh. There are not a lot of dataset based on these Bangladesh roads and vehicles and the existing ones are not enough for our system. So we decided to capture images and videos for our dataset.The first couple of months of our thesis, we solely focused on capturing data on Bangladeshi roads. We took footage of several road trips from Dhaka to Mawa, Dhaka to Narayanganj, Dhaka to Gazipur, Dhaka to Chandpur and some small footage inside Dhaka from our car deck.

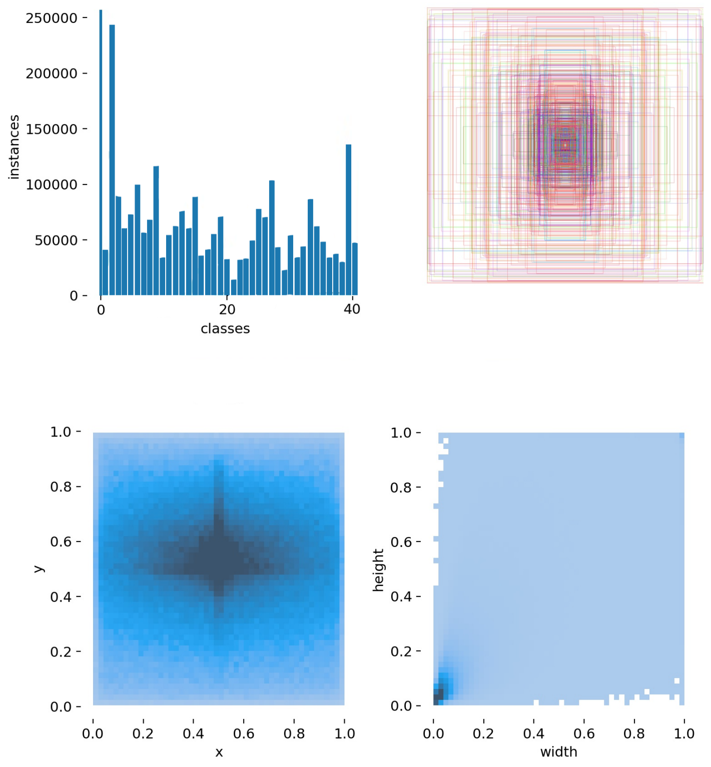
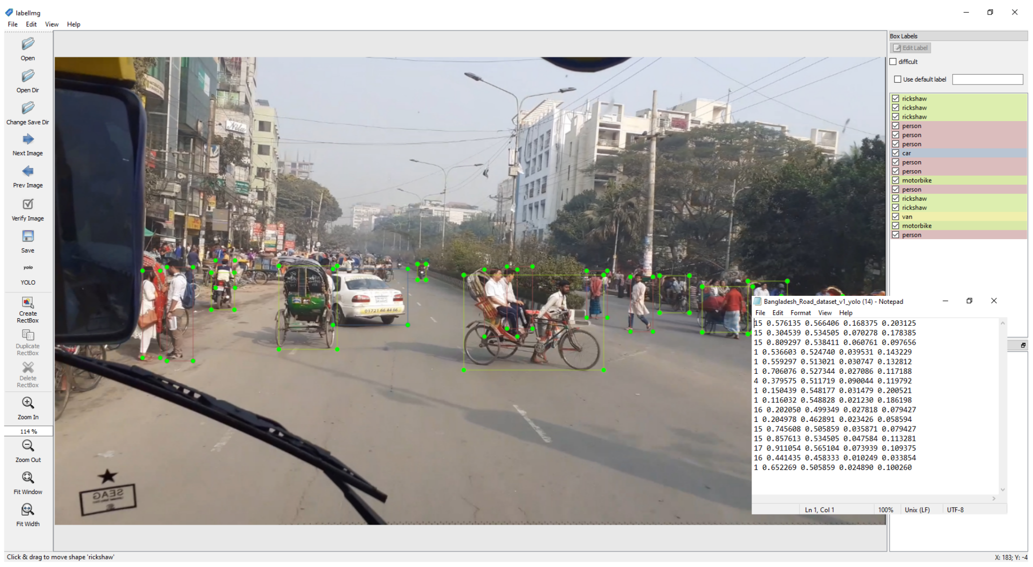


Figure 3.1: Training summary of object detection dataset.

We have about 34 hours of driving data of our own. We also collected some data from some other open source and our thesis instructor also gave us the footage of his road trip from Dhaka to Sirajganj. Apart from the driving data we also needed some specific type of unique vehicles from Bangladesh. So we captured around a couple thousand still images of rickshaw, cng, and local bus for our object detection dataset. We also take help from some Bangladeshi databases and Microsoft COCO(Common Objects in Context) database.

3.2.2 Data Labeling

For labelling the data we used an open source software named LebelImg. As MS COCO uses yolo format to label the images,[17] we also take the same approach for convenience and future dataset enlargement. We take the video we have taken from our road trip and take a frame every 10 sec. Including the still images we roughly had around thirty thousand images for us to label. Detecting the object is as important as the bounding box as we used the bounding box to predict the distance. So, we carefully set the bounding box for objects. We decided to keep 80 labels MS COCO used as it's important to detect any kind of object on the road. Figure 3.2 shows how we are creating our own dataset by labeling images of Bangladeshi roads.



☑ rickshaw
☑ rickshaw
☑ rickshaw
☑ person
☑ person
☑ person
☑ car
☑ person
☑ person
☑ motorbike
☑ person
☑ rickshaw
☑ rickshaw
☑ van
☑ motorbike
☑ person

```
15 0.576135 0.566406 0.168375 0.203125
15 0.304539 0.534505 0.070278 0.178385
15 0.809297 0.538411 0.060761 0.097656
1 0.536603 0.524740 0.039531 0.143229
1 0.559297 0.513021 0.030747 0.132812
1 0.706076 0.527344 0.027086 0.117188
4 0.379575 0.511719 0.090044 0.119792
1 0.150439 0.548177 0.031479 0.200521
1 0.116032 0.548828 0.021230 0.186198
16 0.202050 0.499349 0.027818 0.079427
1 0.204978 0.462891 0.023426 0.058594
15 0.745608 0.505859 0.035871 0.079427
15 0.857613 0.534505 0.047584 0.113281
17 0.911054 0.565104 0.073939 0.109375
16 0.441435 0.458333 0.010249 0.033854
1 0.652269 0.505859 0.024890 0.100260
```

Figure 3.2: Labeling dataset using LableImg.

3.2.3 Splitting Dataset

It's important to keep non trained data to check the performance of the algorithm and dataset. We divided the dataset into two sets- train data and test data. We split the dataset in a ratio of 80:20. 80% for the training data and 20% for test data. We train our system with this 80% data from our dataset and after that the 20% data will be used for test the performance of the training.

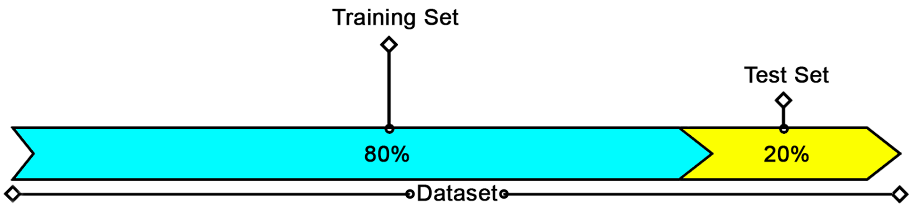
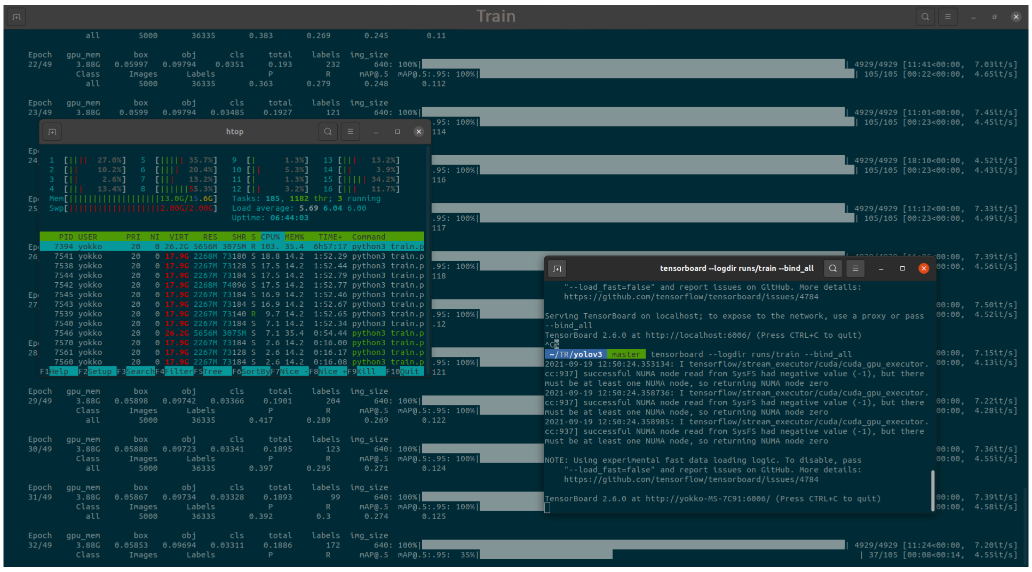


Figure 3.3: Dataset splitting.

3.2.4 Training

First we have to organize our directory to run the training. Images and labels file have to be in the same directory to automatically be able to direct the label data. Then we train our model with our dataset by specifying batch size, weight and image size. All the train data is then saved in the /runs/train folder. After days of training, we will get the trained data and some charts about train and test loss about boundary box, class and object. Figure 3.4 is showing that our object detection model is being trained on our dataset.



```
             all       5000      36335      0.383      0.269      0.245       0.11

     Epoch   gpu_mem       box       obj       cls     total    labels  img_size
     22/49     3.88G   0.05997   0.09794    0.0351     0.193       232       640: 100%| 4929/4929 [11:41<00:00,  7.03it/s]
               Class     Images     Labels          P          R     mAP@.5  mAP@.5:.95: 100%| 105/105 [00:22<00:00,  4.65it/s]
                 all       5000      36335      0.363      0.279      0.248      0.112

     Epoch   gpu_mem       box       obj       cls     total    labels  img_size
     23/49     3.88G    0.0599   0.09794   0.03485    0.1927       121       640: 100%| 4929/4929 [11:01<00:00,  7.45it/s]
                                                                          .95: 100%| 105/105 [00:23<00:00,  4.45it/s]
                                                                           114
     Ep
     24                                                                              | 4929/4929 [18:10<00:00,  4.52it/s]
                                                                          .95: 100%| 105/105 [00:23<00:00,  4.43it/s]
                                                                           116
     Ep
     25                                                                              | 4929/4929 [11:12<00:00,  7.33it/s]
                                                                          .95: 100%| 105/105 [00:23<00:00,  4.49it/s]
                                                                           117
     Ep
     26                                                                                                  00:00,  7.39it/s]
                                                                          .95: 100%|                     00:00,  4.56it/s]
                                                                           118
     Ep
     27                                                                                                  00:00,  7.50it/s]
                                                                          .95: 100%|                     00:00,  4.52it/s]
                                                                          .12
     Ep
     28                                                                                                  00:00,  7.15it/s]
                                                                          .95: 100%|                     00:00,  4.13it/s]
                                                                           121

     Epoch   gpu_mem       box       obj       cls     total    labels  img_size
     29/49     3.88G   0.05898   0.09742   0.03366    0.1901       204       640: 100%|                     00:00,  7.22it/s]
               Class     Images     Labels          P          R     mAP@.5  mAP@.5:.95: 100%|           00:00,  4.28it/s]
                 all       5000      36335      0.417      0.289      0.269      0.122

     Epoch   gpu_mem       box       obj       cls     total    labels  img_size
     30/49     3.88G   0.05888   0.09723   0.03341    0.1895       123       640: 100%|                     00:00,  7.36it/s]
               Class     Images     Labels          P          R     mAP@.5  mAP@.5:.95: 100%|           00:00,  4.55it/s]
                 all       5000      36335      0.397      0.295      0.271      0.124

     Epoch   gpu_mem       box       obj       cls     total    labels  img_size
     31/49     3.88G   0.05867   0.09734   0.03328    0.1893        99       640: 100%|                     00:00,  7.39it/s]
               Class     Images     Labels          P          R     mAP@.5  mAP@.5:.95: 100%|           00:00,  4.58it/s]
                 all       5000      36335      0.392        0.3      0.274      0.125

     Epoch   gpu_mem       box       obj       cls     total    labels  img_size
     32/49     3.88G   0.05853   0.09694   0.03311    0.1886       172       640: 100%| 4929/4929 [11:24<00:00,  7.20it/s]
               Class     Images     Labels          P          R     mAP@.5  mAP@.5:.95:  35%|  | 37/105 [00:08<00:14,  4.55it/s]
```

```
  1  [||||   27.0%]   5  [|||||  35.7%]   9  [|       1.3%]   13 [|||    13.2%]
  2  [||     10.2%]   6  [||||   20.4%]   10 [||      5.3%]   14 [||      3.9%]
  3  [||      2.6%]   7  [|||    13.2%]   11 [|       1.3%]   15 [||||| 34.2%]
  4  [|||    13.4%]   8  [||||||55.3%]    12 [||      3.2%]   16 [|||    11.7%]
  Mem[||||||||||||||||||13.0G/15.6G]    Tasks: 185, 1182 thr; 3 running
  Swp[||||||||||||||||||2.00G/2.00G]    Load average: 5.69 6.04 6.00
                                        Uptime: 06:44:03

    PID USER       PRI  NI  VIRT   RES   SHR S CPU% MEM%   TIME+  Command
   7394 yokko       20   0 26.2G 5656M 3075M R 103. 35.4  6h57:17 python3 train.p
   7541 yokko       20   0 17.9G 2268M 73180 S 18.8 14.2  1:52.29 python3 train.p
   7538 yokko       20   0 17.9G 2267M 73128 S 17.5 14.2  1:52.44 python3 train.p
   7544 yokko       20   0 17.9G 2267M 73184 S 17.5 14.2  1:52.79 python3 train.p
   7542 yokko       20   0 17.9G 2268M 74096 S 17.5 14.2  1:52.77 python3 train.p
   7545 yokko       20   0 17.9G 2267M 73184 S 16.9 14.2  1:52.46 python3 train.p
   7543 yokko       20   0 17.9G 2267M 73184 S 16.9 14.2  1:52.67 python3 train.p
   7539 yokko       20   0 17.9G 2267M 73140 R  9.7 14.2  1:52.65 python3 train.p
   7540 yokko       20   0 17.9G 2267M 73184 S  7.1 14.2  1:52.34 python3 train.p
   7546 yokko       20   0 26.2G 5656M 3075M S  7.1 35.4  0:54.44 python3 train.p
   7570 yokko       20   0 17.9G 2267M 73184 S  2.6 14.2  0:16.00 python3 train.p
   7561 yokko       20   0 17.9G 2267M 73128 S  2.6 14.2  0:16.17 python3 train.p
   7560 yokko       20   0 17.9G 2267M 73184 S  2.6 14.2  0:16.08 python3 train.p
F1Help  F2Setup  F3SearchF4FilterF5Tree   F6SortByF7Nice -F8Nice +F9Kill   F10Quit
```

```
    "--load_fast=false" and report issues on GitHub. More details:
    https://github.com/tensorflow/tensorboard/issues/4784

Serving TensorBoard on localhost; to expose to the network, use a proxy or pass
--bind_all
TensorBoard 2.6.0 at http://localhost:6006/ (Press CTRL+C to quit)
^C%
 ~/TR/yolov3  master  tensorboard --logdir runs/train --bind_all
2021-09-19 12:50:24.353134: I tensorflow/stream_executor/cuda/cuda_gpu_executor.
cc:937] successful NUMA node read from SysFS had negative value (-1), but there
must be at least one NUMA node, so returning NUMA node zero
2021-09-19 12:50:24.358736: I tensorflow/stream_executor/cuda/cuda_gpu_executor.
cc:937] successful NUMA node read from SysFS had negative value (-1), but there
must be at least one NUMA node, so returning NUMA node zero
2021-09-19 12:50:24.358985: I tensorflow/stream_executor/cuda/cuda_gpu_executor.
cc:937] successful NUMA node read from SysFS had negative value (-1), but there
must be at least one NUMA node, so returning NUMA node zero

NOTE: Using experimental fast data loading logic. To disable, pass
    "--load_fast=false" and report issues on GitHub. More details:
    https://github.com/tensorflow/tensorboard/issues/4784

TensorBoard 2.6.0 at http://yokko-MS-7C91:6006/ (Press CTRL+C to quit)
```

Figure 3.4: Dataset training.

3.3 Object Detection Model CbOD

Detecting instances of objects of a specific class inside an image is known as object detection. There are two primary categories of state-of-the-art approaches: one-stage methods and two-stage methods. One-stage techniques emphasizes on inference speed. And two-stage techniques prioritizes detection accuracy. The first barrier to cross to make an autonomous car is to detect all the objects on the road. All the self-driving vehicle researchers have used different techniques to detect objects on the road which includes using various dataset and implementing different algorithms in their system. That is why we have created our own method which is CNN based object detection (CbOD) system.

3.3.1 Camera Calibration

For object detection, the first thing we had to do is camera calibration. As the field of the viewing angle of cameras can differ from each other so it was necessary to adjust the camera angle with our algorithm. For this, we use a method called checkerboard calibration[36]. The camera we have used is "ThiEYE action camera" which has a 170° wide field of view. Then we took images of a checkerboard with this camera from different angles. We know the black and white box of a checkerboard is parallel and identical. But for a 170° wide angle camera these boxes are distorted. That is why we used a checkerboard calibration algorithm on these images (in figure 3.5) for better accuracy in object detection.

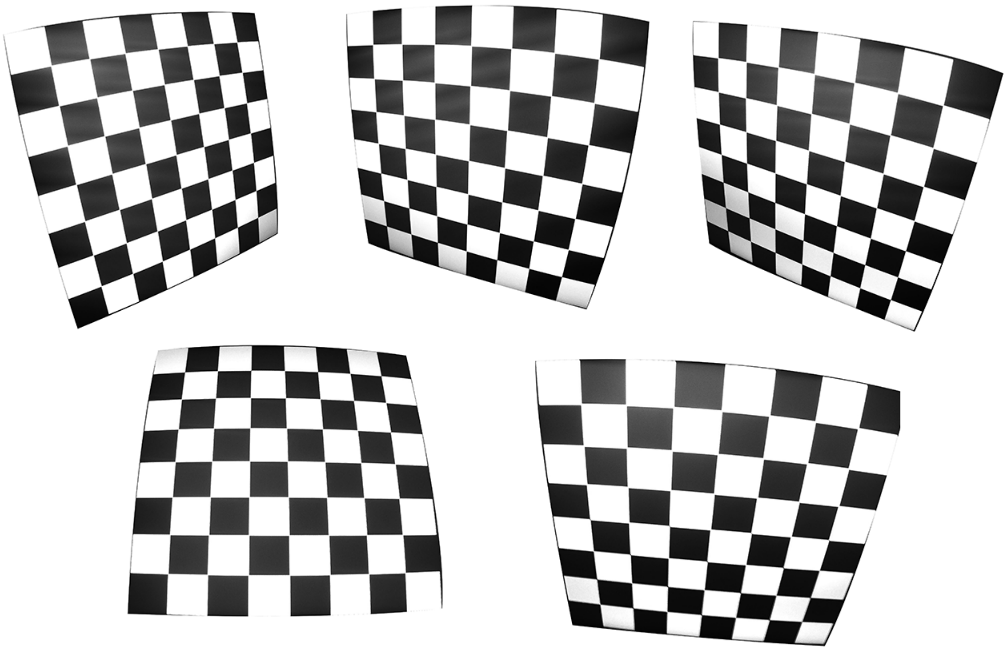


Figure 3.5: Chessboard images for checkerboard calibration.

3.3.2 Object Detection Architecture

There are many methods available to detect objects based on various environments. But in Bangladesh, the roads are packed with unnecessary objects and for this reason all the methods that are available are not suitable for this environment. To optimize an object detection system which is optimized and capable of running in Bangladesh we were challenged with two distinct questions, what the object is and where it is positioned within the frame. To answer this question, the real-time video data that is used as input is separated into different frames and using the object window prediction method, the position of different objects within the frame are detected through marking them with separate object windows answering the “Where” part of the question. Now, through the use of class prediction, the object within a window is compared to the pre-processing BDCO dataset to successfully identify the “What” part of the question. The two processes are given below.

3.3.2.1 Object window prediction

We took help from YOLO9000 to create our object window prediction. This window has a network which can predict 4 coordinates for each object window as a_x, a_y, a_w, a_h. Furthermore, this window has a height h and width w. We denoted the top left grid as b_x and b_y.

So, the prediction will be-

$$O_x = \delta(a_x) + c_x \tag{3.1}$$

$$O_y = \delta(a_y) + c_y \tag{3.2}$$

$$O_w = we^{a_w} \tag{3.3}$$

$$O_h = he^{a_h} \tag{3.4}$$

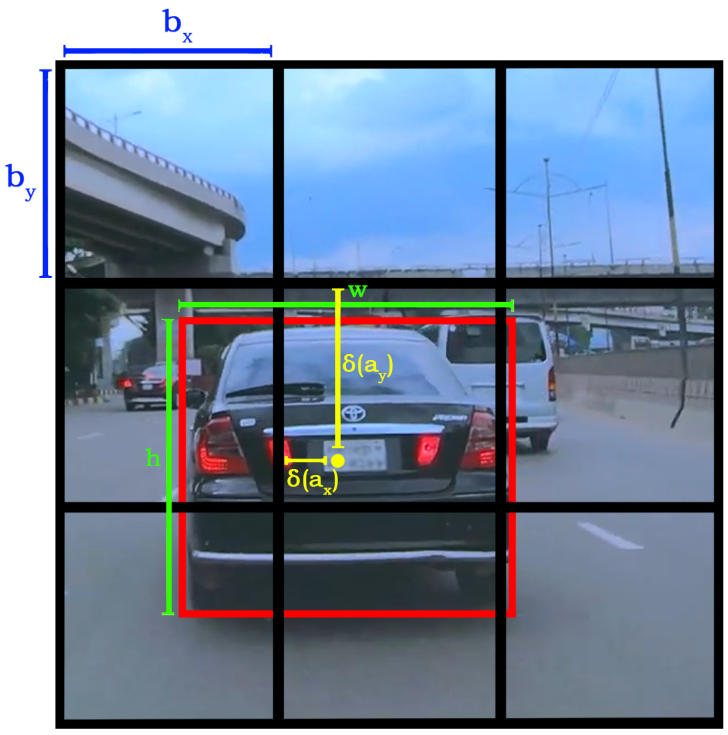


Figure 3.6: Object Window's Calculation.

We reduced the focus area of the window so that it can detect all the important objects on the road which makes it more optimized to drive autonomously despite having all the unnecessary objects on the road of Bangladesh.

3.3.2.2 Class Prediction

Another important task that we have overcome is to acknowledge the objects that are detected through the object window prediction. The objects are recognized by the help of the extended pre-trained dataset library(BDCO) that we have created and divided into various classes. All the major objects are put into individual classes and a single class is created for all the other minor objects that are seen on the road of Bangladesh. We were able to increase the accuracy level and optimize the overall model by reducing the classes.

3.3.3 Object Detection Algorithm

In our model, the object algorithm that we have made in a way that can detect multiple objects at the same time with a satisfactory level of accuracy. Our algorithm first compares with the BDCO dataset and predicts classes of the objects and identifies its location within the frame. Our model is applied with a single neural network and divides the frames into grid cells.

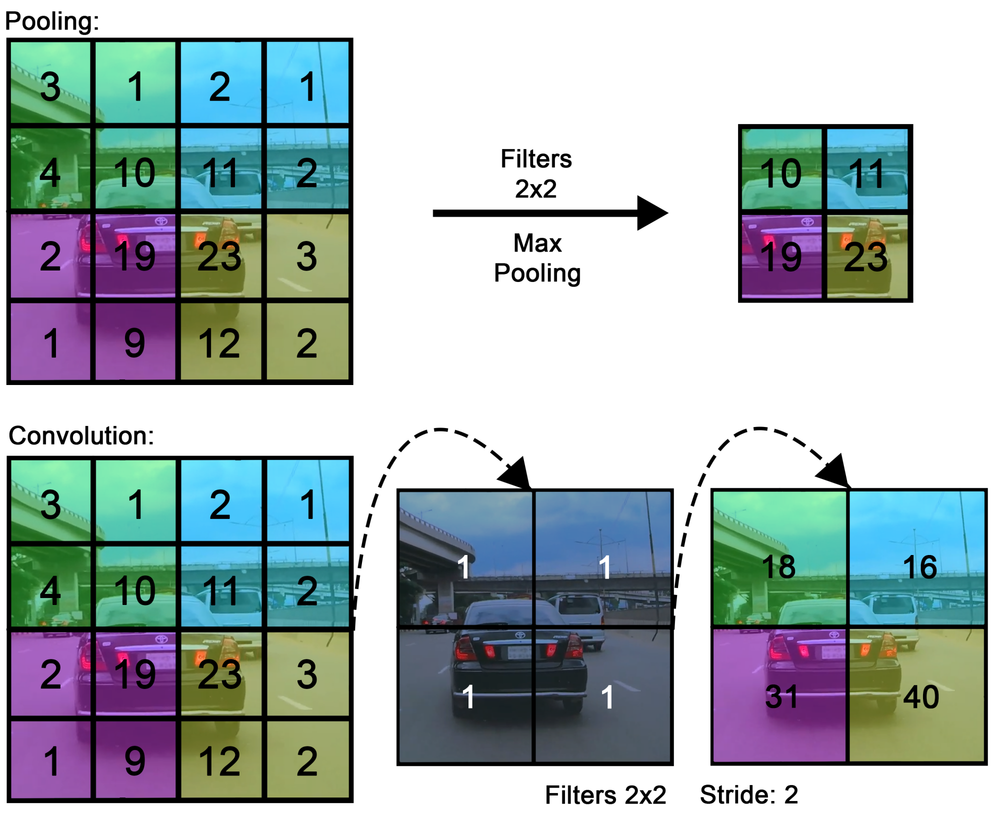


Figure 3.7: Pooling vs Convolutional network.

Furthermore, it gives probabilities of the existence of the objects on the grid cells. Instead of having pooling layers to down-sample feature maps, additional convolutional layers are used with stride 2. Due to this circumstance, prevention of loss of low-level features takes place which would have been excluded by the pooling layer. This helps in improving the ability of detecting small objects. An example of this can be observed where, pooling excludes the numbers from images, but these numbers are taken into account in convolution.

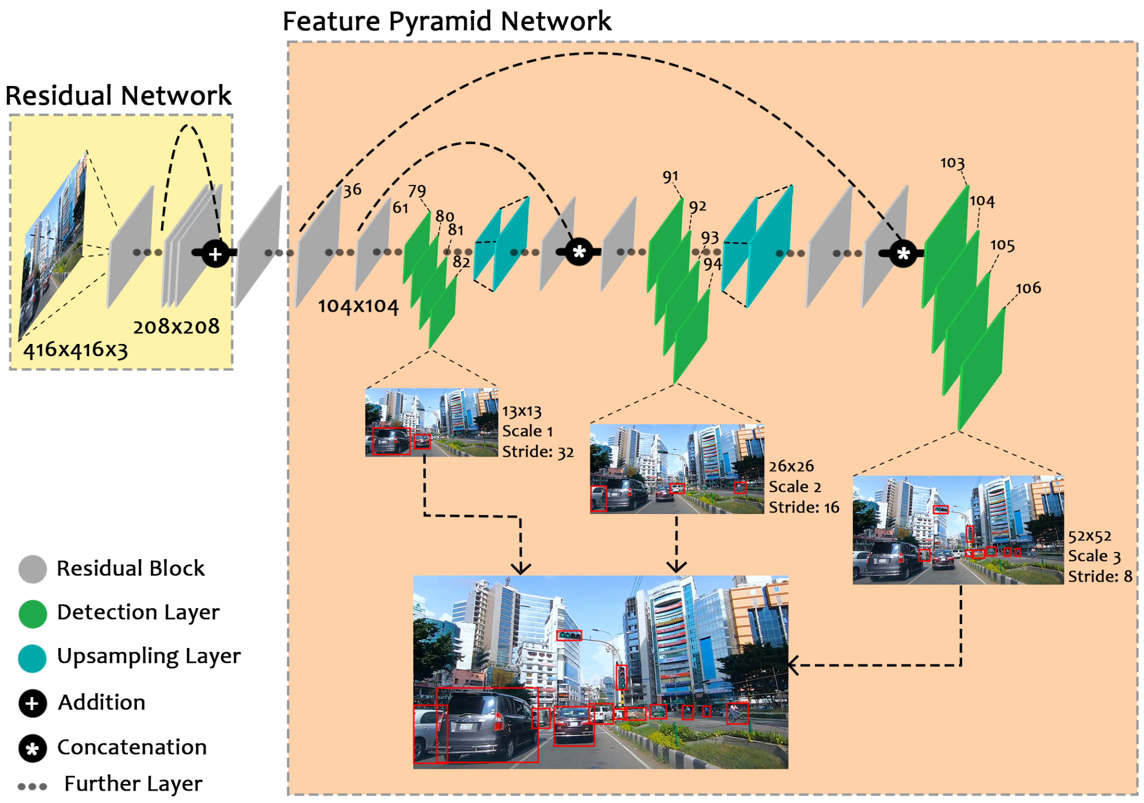
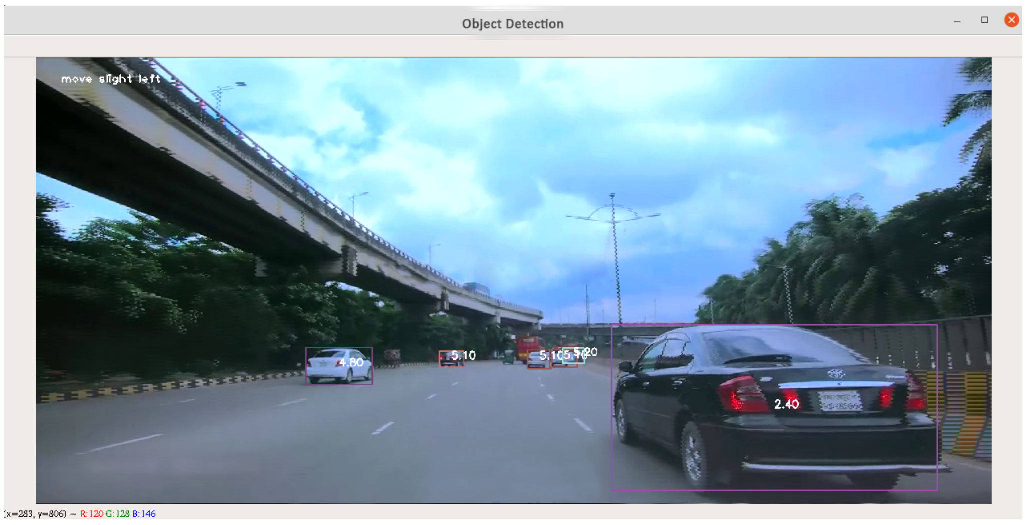
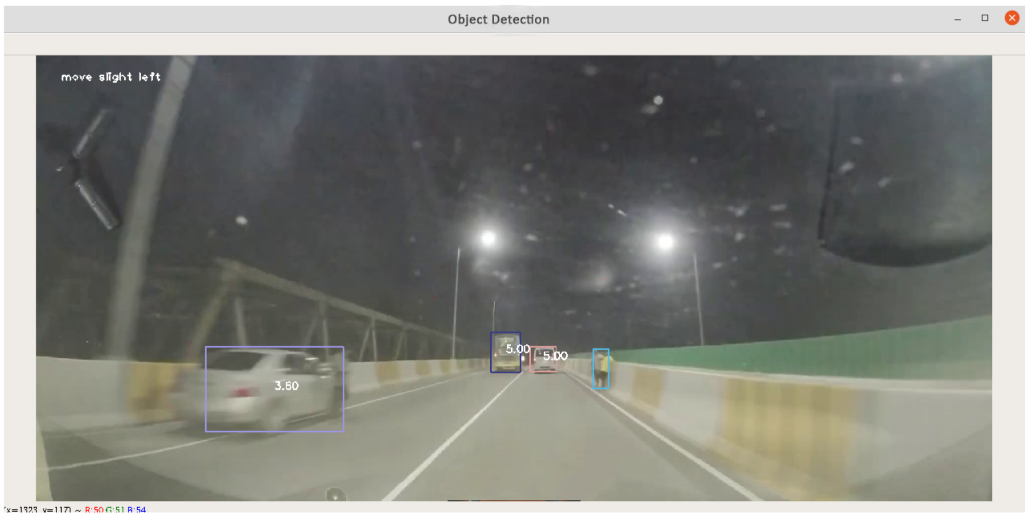


Figure 3.8: Architecture of CbOD.

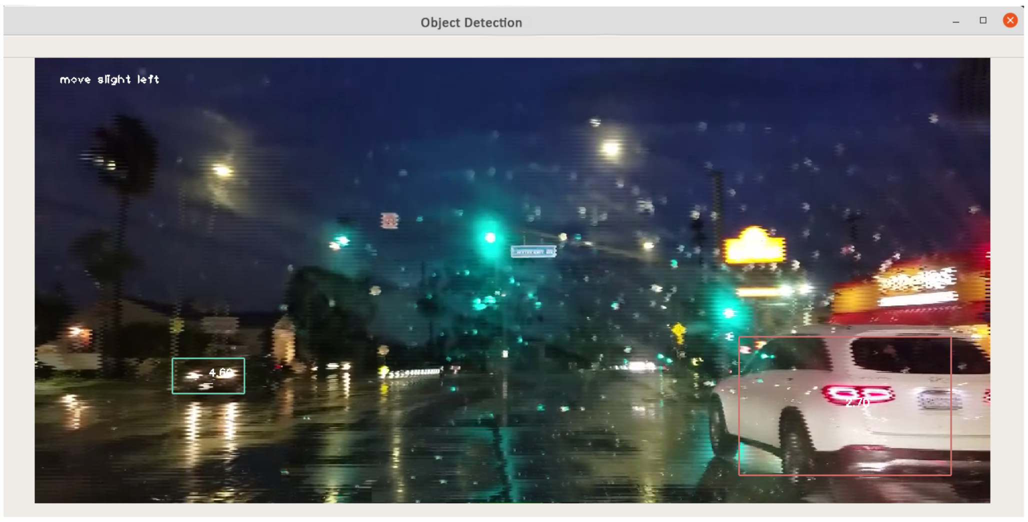
Our model has 53 CNN layers(darknet-53) and 53 other layers which in total is 106 layers. There are 3 detection layers which make it perform better. These detection layers are 82,94,106. All these layers can generate gradually small(13x13), medium(26x26), large(52x52) sizes of frames. Smaller frames are used to detect large objects and larger frames are used to capture the small objects. Small(13x13) sized frame is used to detect the largest object in the frame, while a medium(26x26) sized frame is used to detect objects comparatively smaller than before. Lastly, a large(52x52) sized frame is used to detect the smallest objects. This method helps the algorithms in detection of objects accurately while maintaining lightning fast speed(figure 3.8).Figure 3.9 shows object detection from video input in different weather conditions by our object detection model CbOD.



Normal Day



Normal Night



Rainy Night

Figure 3.9: Object Detection Output in Different Weather Conditions.

3.4 Lane Detection QLD

After detecting an object, an autonomous vehicle needs to follow a path to reach its goal. So the next step is to detect lanes. All the researchers and engineers who have worked on autonomous systems have used different methods of lane detection. Traditional video-based lane detection technology relies on image processing algorithms to collect lane line features, reduce image channels and then produce lane line accommodations after extracting the input images. In our model, we have come up with our own version of lane detection and we have named it QLD (Quick Lane Detection).

3.4.1 Lane detection architecture

There are so many methods that are available which can detect lanes based on various environments. Among all these methods, traditional segmentation is more precise. Traditional segmentation takes real time video frames and divides them into pixels. Those pixels are compared with the CULane dataset. If any pixel in the video frame contains a line, it marks that pixel as a lane. Though it is one of the accurate methods, it is not efficient as it has to check 1920x1080 pixels per frame for a FullHD (1080p) or 1280x720 pixels per frame for HD (720p) video data. It will cost more time and resources to process frame by frame. To optimize this issue we customized our own method. In our method, we segmented the lower half of the frames with 15 rows and 200 columns to make gridding cells. Then we compare these gridding cells with the CULane datasets to find lines. If any gridding cell contains a line, it will mark that cell as a lane in that video frame.

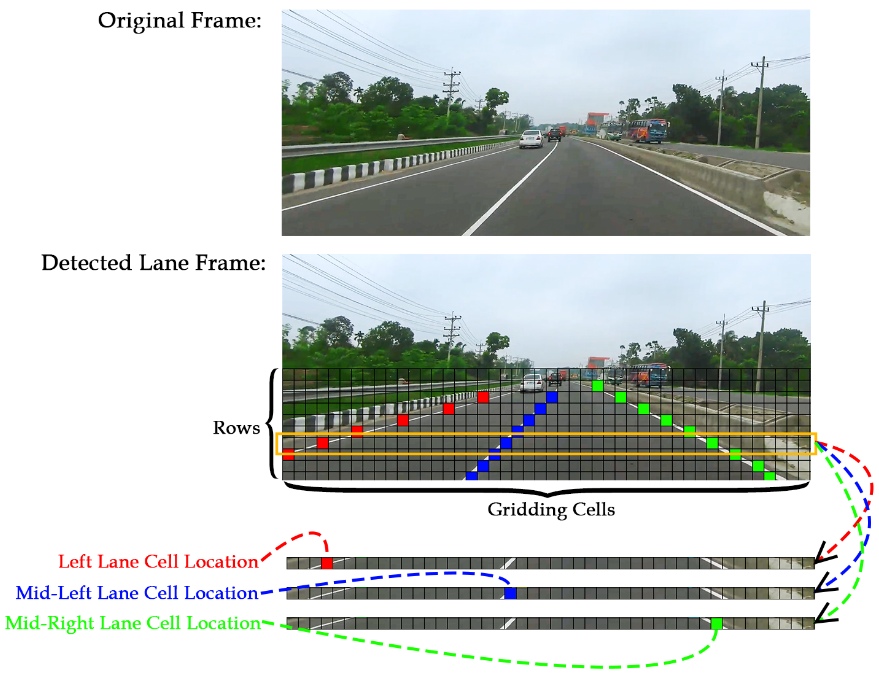
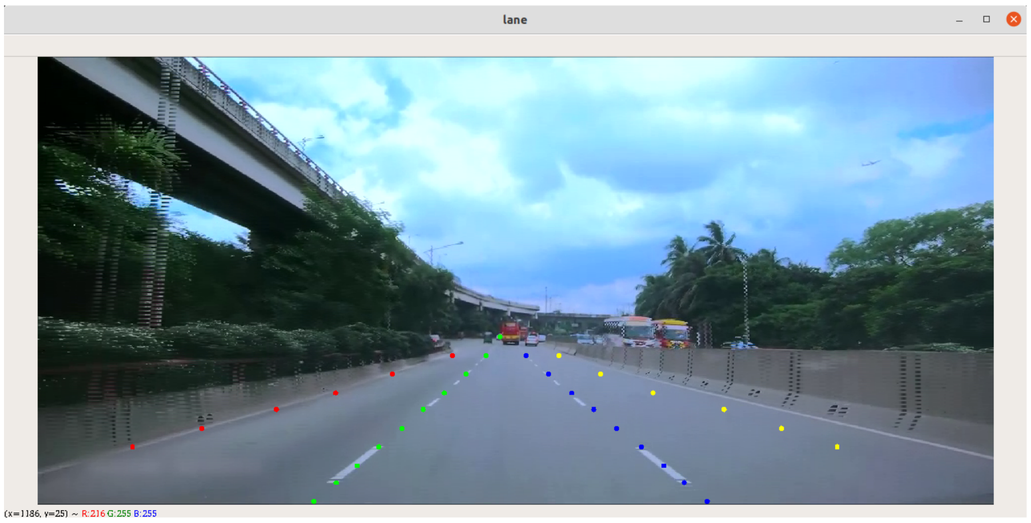
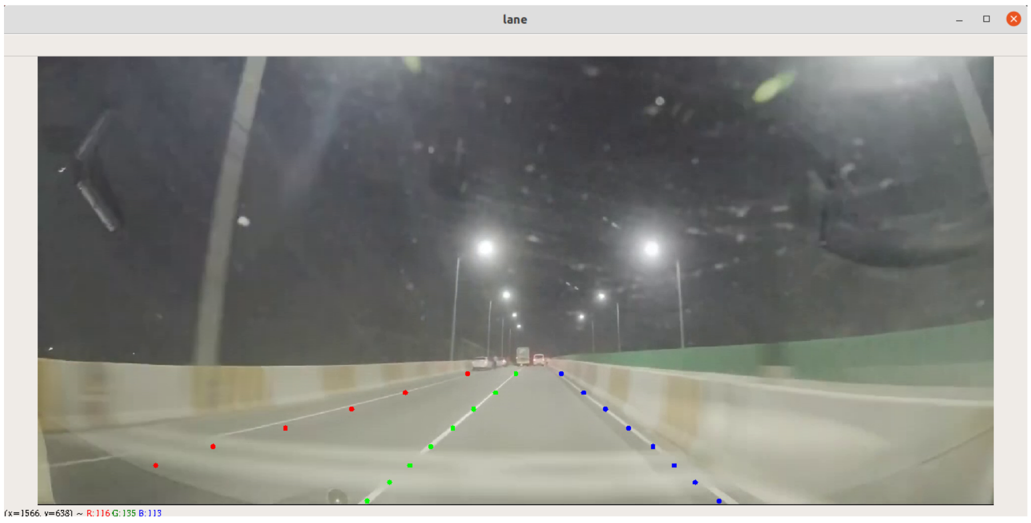


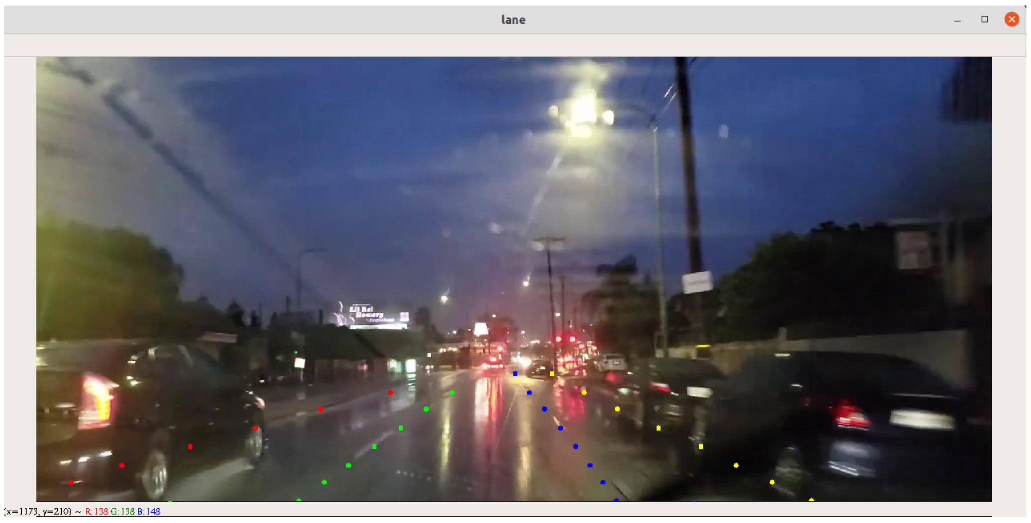
Figure 3.10: Segmentation of Lanes.



Normal Day



Normal Night



Rainy Night

Figure 3.11: Lane detection in different weather condition.

In figure 3.10 , we can clearly acknowledge that using gridding cells in our model can direct lanes accurately with less computational cycle. Figure 3.11 shows lane detection output from video input of different challenging weather conditions by our QLD. In our method, we segmented the lower half of the frames with 15 rows and 200 columns to make gridding cells. Then we compare these gridding cells with the CULane datasets to find lines. If any gridding cell contains a line, it will mark that cell as a lane in that video frame.

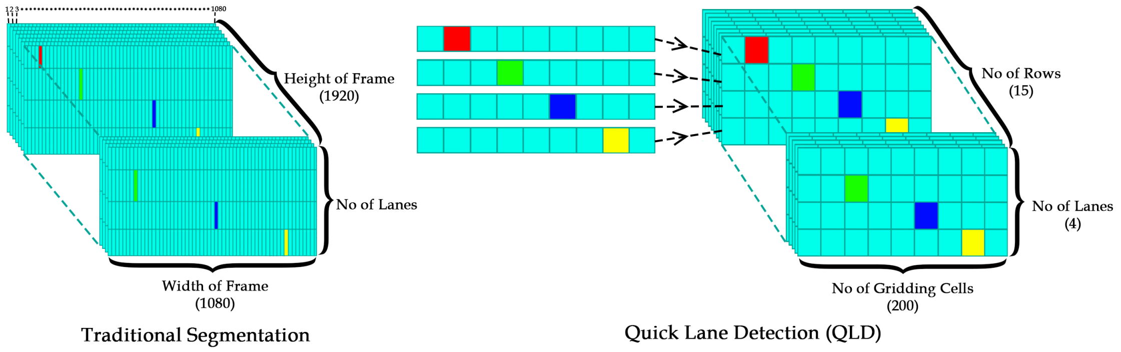


Figure 3.12: Comparison between traditional and ours lane detection algorithm.

Here, in figure 3.12 it is clearly shown that traditional segmentation needs to compute all the pixels, whereas our customized model needs to compute only 15 rows for each frame.

Computational Cost of Traditional Segmentation	**Computational Cost of Our Customized Model**
Height of Frame x Width of Frame x Number of Lanes	Number of Rows x Number of Gridding Cell x Number of Lanes

Table 3.1: Comparison between traditional and ours lane detection computational cost.

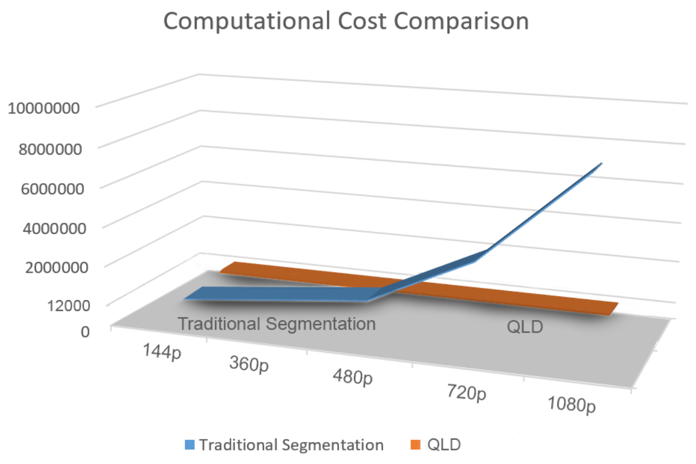


Figure 3.13: Computational cost comparison.

Here, for a low resolution video frame (176p x 144p) the computational cost : 176 x 144 x 4 = 101.2x10e3 And for our model it will be : 15 x 200 x 4 = 1.2x10e3 only. So it is around 80 times faster than traditional segmentation. As our model divides any resolution frame into 15 x 200 segments it will be constant for any kind of resolution. For a decent quality resolution video frame (480p x 360p) the computational cost : 6.9x10e5. Our approach will generate the same computational cost(15 x 200 x 4 = 1.2x10e3) as before which leads to around 575 times faster performance. The computational cost comparison graph in different resolutions are shown in figure 3.13.

3.4.2 Lane Detection Algorithm

To detect the lane much precisely we divided the expected portion of the frame into 15 rows and each row into 200 segments. After receiving the input frame the algorithm extracts the feature and tune it down the resolution a couple of times. From there the value goes to group classification where it handles the classification based prediction. The row anchors and row segmentation is also done in the group classification stage. During training the value from the residual block goes to auxiliary segmentation. From the residual block it detects the lanes and only takes the lane-detection information and converts it into one frame in auxiliary segmentation. This lane information is transferred to the main branch where it shows the lane detection along with the input images which are shown in figure 3.14.

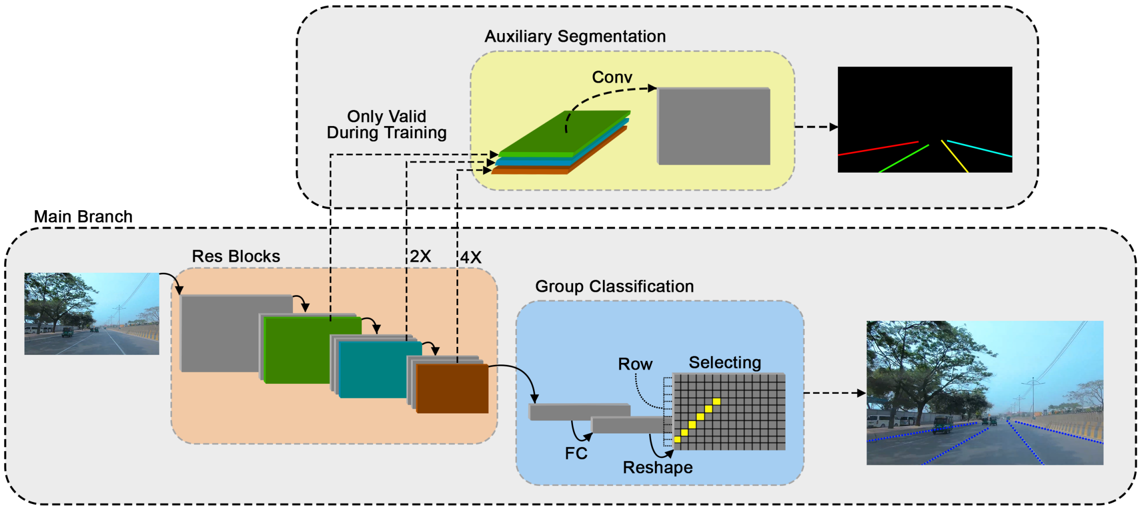


Figure 3.14: Algorithm of QLD.

3.5 Distance Measurement

The assistance system can measure the approximate distance of the detected objects. This helps to generate the warnings if the object is too close to the vehicle. If our object detection algorithm detects an object, it also calculates the distance from the vehicle. We used a formula to calculate the approximate distance.

$$D = (h - bbox[3] * r) * b \tag{3.5}$$

Here, h denotes the height of the input frame, bbox[3] refers to a corner point of the bounding box of the detected object. r and b refer to ratio and bias respectively. We need to calibrate these values to get the appropriate distance. If these values can be calibrated accurately, it is possible to get the actual distance from this formula. Our algorithm is so efficient and fast that if 50 objects are detected in a frame, all objects' distance will be calculated at the same time and will be shown in the output.

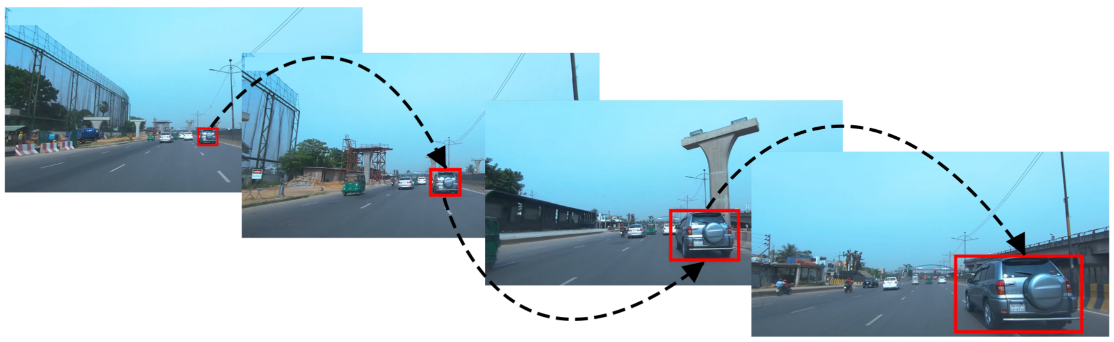


Figure 3.15: Distance Measurement of the Object.

In figure 3.15 we can see that as the object is approaching closer to our vehicle the object window is getting bigger. This can be calibrated by tweaking ratio and bias how bigger the object windows will be when they are closer in the formula to measure the appropriate distance.

3.6 Safe Zone Detection

An autonomous vehicle can not function with ease if the model can not identify all the safe zones for that vehicle. Our model uses video frames as input and we divided different pixels of the frame and assigned them with various zones such as left zone, right zone, green zone, red zone, danger zone.
Left zone: If any vehicle comes in the left zone, our model shows a warning of the existence of that vehicle.
Right zone: If any vehicle comes in the right zone, our model shows a warning of the existence of that vehicle.

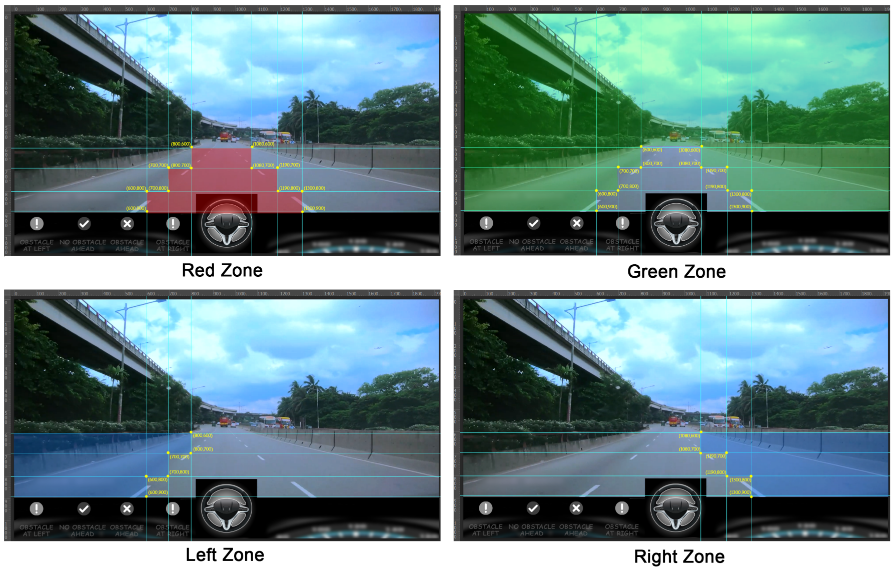


Figure 3.16: Different zone areas.

Green zone: Green zone works as a safe zone. If any object is detected in this zone, the model will suggest to drive forward with ease.
Red zone: This zone can detect objects if the objects are available in the 5m range in front and show warning accordingly.
Danger zone: This zone is part of all the zones except the green zone. If any objects are detected in any of these zones which are closer than 2m or less, the model shows a warning based on the position of the vehicle.
Figure 3.16 shows all the zones with coordinates that are taken from the front camera that we have used in our model.

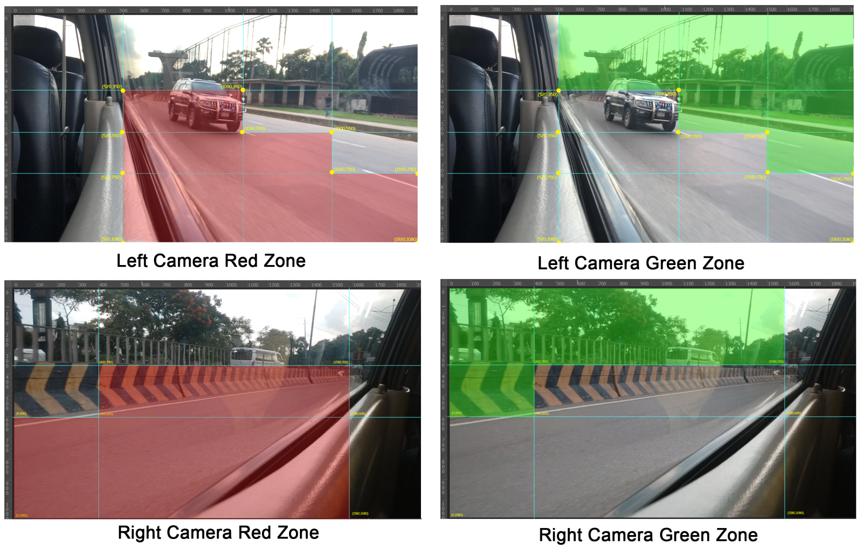


Figure 3.17: Left-Right zones.

As our model has usage of three cameras one in the front and other two cameras are situated in the right and left side of the vehicle. So it was necessary to assign zones in these cameras too and that is what we did which is shown in the figure 3.17. We implemented only red and green zones for these two cameras.

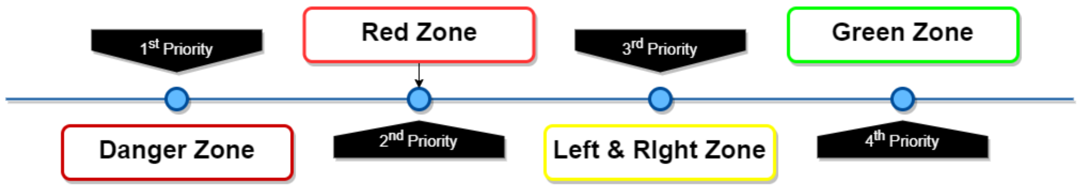


Figure 3.18: Zone presidency.

The precedence of the zones that our model follows are shown in the figure 3.18.

3.7 Decision Making

After detecting objects and dividing video frames into different zones, our aim was to make decisions based on the heterogeneous environment. To achieve this goal, we took several points from the object window which was found from CbOD. We took six points from that object window. We used the coordinates of the object window to get the points. Figure 3.19 shows how we have calculated some of the points based on the coordinates.

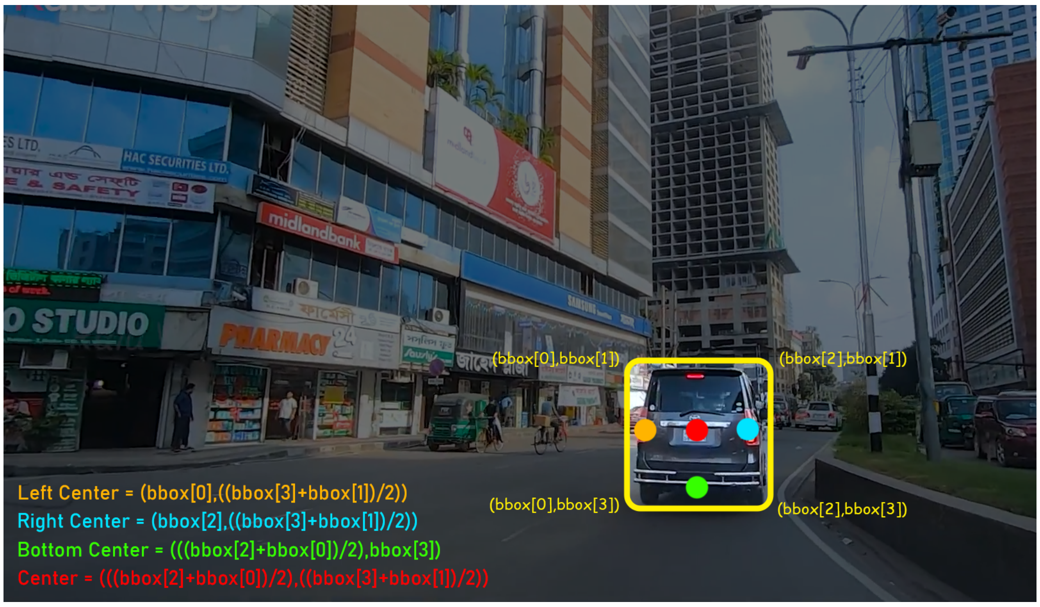

(bbox[0],bbox[1])
(bbox[2],bbox[1])
(bbox[0],bbox[3])
(bbox[2],bbox[3])
Left Center = (bbox[0],((bbox[3]+bbox[1])/2))
Right Center = (bbox[2],((bbox[3]+bbox[1])/2))
Bottom Center = (((bbox[2]+bbox[0])/2),bbox[3])
Center = (((bbox[2]+bbox[0])/2),((bbox[3]+bbox[1])/2))

Figure 3.19: Object Window Calculation.

After finding the corresponding points, we applied various conditions in a way so that our system can make decisions based on the corresponding predefined zones, if any points come across any of the zones. Figure 3.20 Shows how our model makes decisions by considering the object window and the predefined zones.

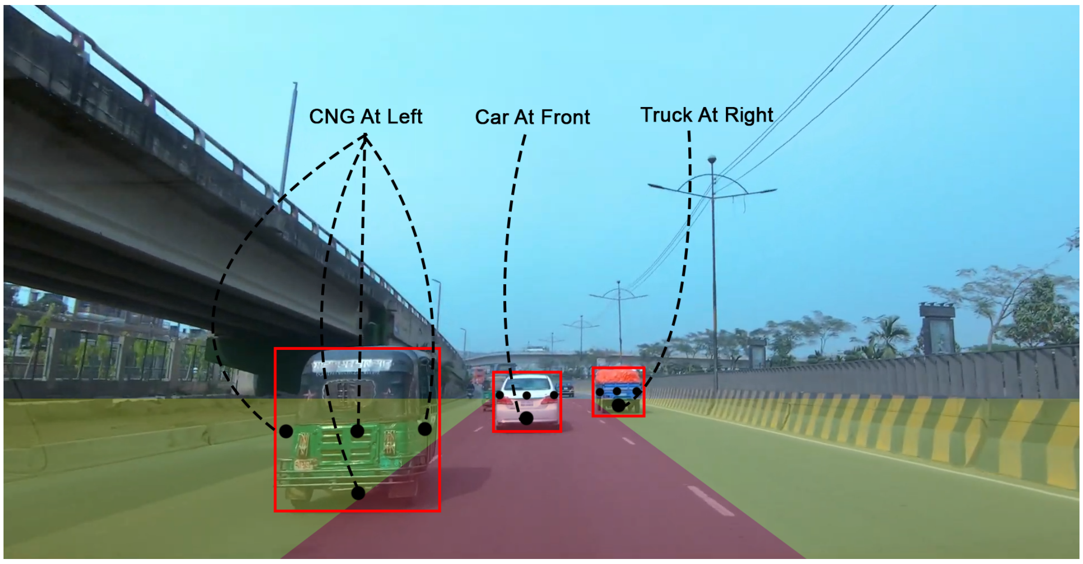


Figure 3.20: Decision Making.

3.8 Steering Angle Prediction

Our driving assistance system can predict steering angle of the vehicle based on lane detection. It follows the lane and gives an approximate steering angle to keep the vehicle in the lane. This feature is almost similar to lane keeping assistance and it helps drivers of the vehicle to keep the vehicle in the right lane and avoid accidents. We have used a formula to predict the steering angle which is used simultaneously with lane detection in each frame.

$$angle = \tan^{-1}(\frac{180}{\pi}) * (\frac{angle}{5}) \tag{3.6}$$

This steering control module helps our vehicle to maintain a safe path throughout the way. In our model,if the roads are free or busy ahead, the steering changes colour accordingly which helps to understand the condition of the roads so easily. Figure 3.21 shows how we get the steering angle according to lane detection from a video input.

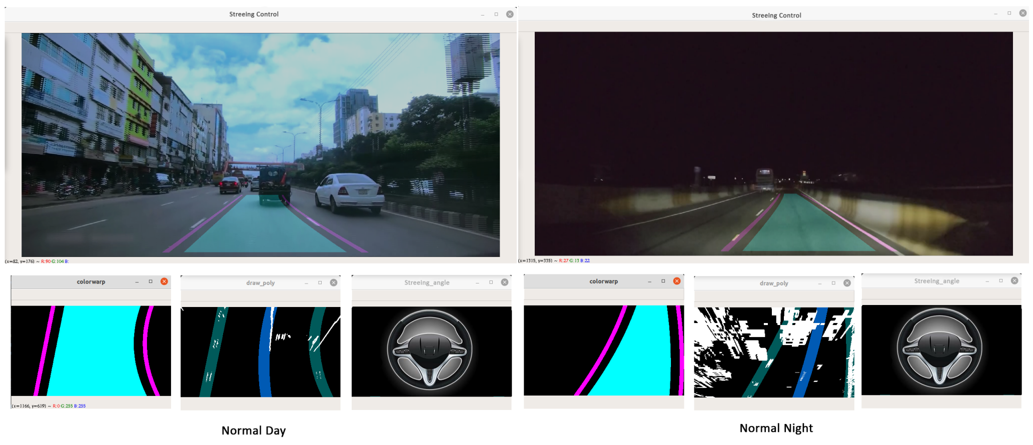


Figure 3.21: Steering angle prediction.

Chapter 4

BD-DA Cascaded Network

We separately implemented all the algorithms or our driving assistance. For further optimization, utilization and benefit we will merge and implement all of our algorithms in a single cascaded network. We will discuss it in detail in the following part.

4.1 Cascaded Block

After taking the input from the video frame it goes into the residual block of lane detection. There it divides into two different branches- main branch, auxiliary branch. These two branches include residual block, group classification and auxiliary segmentation. In this part it does all the process for lane detection including- feature extraction, classification based prediction, rows and row segmentation. After detecting the lane it creates an output frame which then goes into the object detection portion. In this part it uses the output frame of lane detection and uses it as an input frame for object detection. Here the system generates three different sizes of resolution from the input frame- small(13x13), medium(26x26), large(52x52). Larger frames are used to capture small objects and Smaller frames are used to detect large objects. After that it combines all the predicted object class and position into a single frame as an output frame.
The output frame of the object detection has all the detected lanes and objects with it. This output frame is fed into three different concurrent processes- distance measurement, safe zone identification and steering control. In the distance measurement part it detects the distance of other important objects by comparing the given size of the object from our database to the input size of the object. It can calculate the distance by applying the formula to predict the distance from the host vehicle. At the same time in the safe zone identification algorithm it detects if there is any object in the selected box located in the frame. There are multiple boxes labeled for the different levels of warning. In the steering control segment it takes the input frame and locates its position on the current lane. If there is no vehicle in front of the current lane then it suggests to center the vehicle in perspective of the current lane. In the turning points it also suggests to center the vehicle and assist to keep driving in the lane.

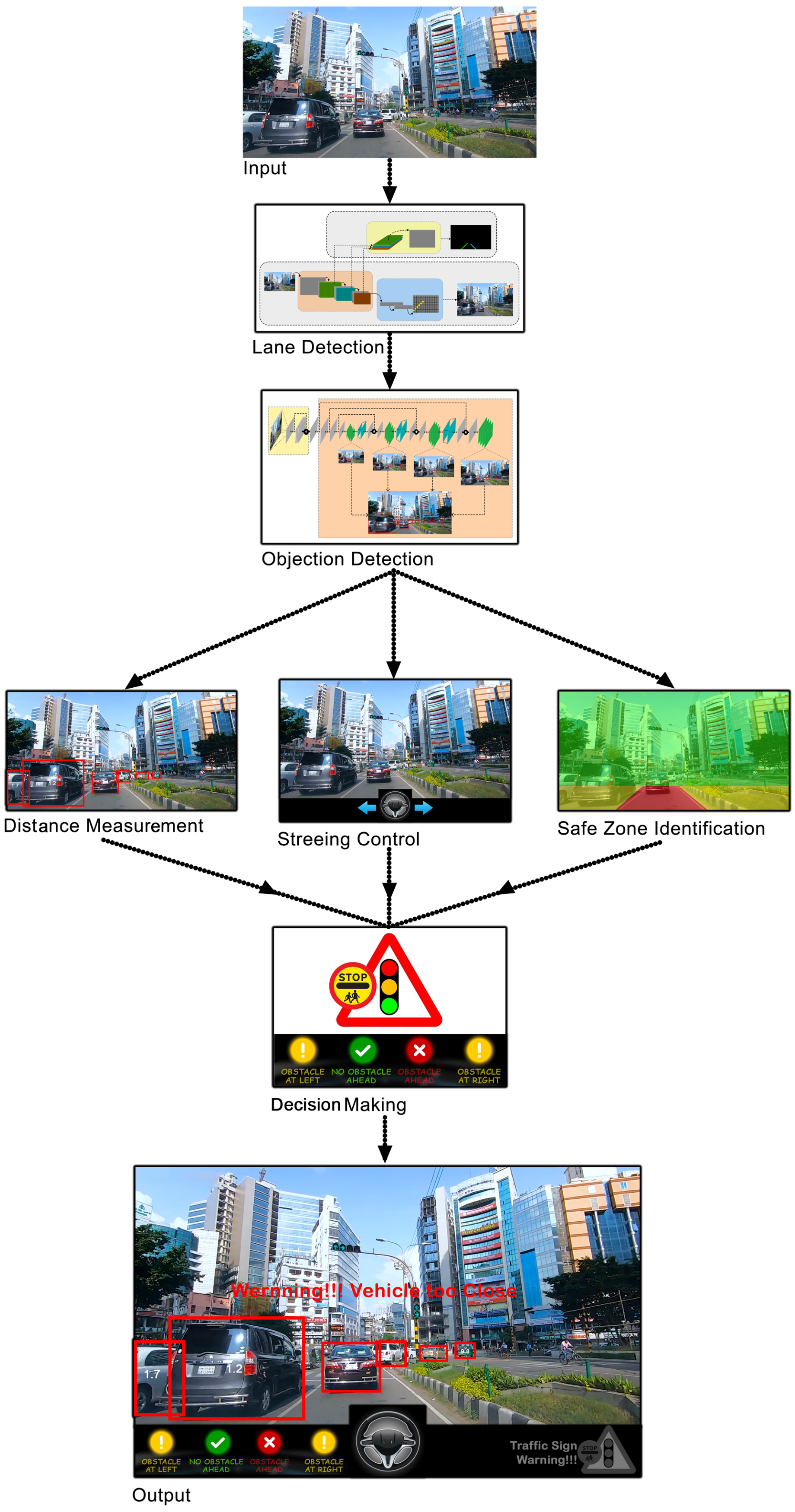


Figure 4.1: BD-DA Cascaded Network.

All three consecutive algorithms- distance measurement, safe zone identification and steering control creates a single output frame. This output frame includes all the data including lane detection object detection, distance measurement, safe zone identification and steering control. After that our system applies some small but important algorithm to display some info which is of great importance. For example - a warning if there is any vehicle in the left or right lane, a warning if there is any vehicle in the front lane and a green signal if there is no vehicle in close range. The system also works for three different camera inputs-front view, left view and right view continuously(figure 4.1).

4.2 Loss Function

As our model is mainly a prediction based model, there might be prediction error while training and validating the model. For the best result, we should try to reduce the loss as much as possible. That is why we tried to make our dataset rich so that there are less prediction error.
In our system there are mainly four parts of loss function. They are -

- BCELoss or Binary cross entropy loss. It adjusts the parameter of the center point of object window x and y.
- MSELoss or Mean Square entropy loss. This adjusts the parameters of anchor's width and height.
- Confidence Factor - BCELoss calculates the confidence factor of the model.
- Classification loss – BCELoss calculates classification loss of the model.

Among these, BCELoss plays significant part in calculating the overall loss of the model. The formula[53] to calculate BCELoss is-

$$loss = -target * log(prediction)(1.0target) * log(1.0 - prediction) \tag{4.1}$$

As we have merged MS-COCO with our own labeled dataset we already know our target predictions. After training and testing, when we got the original prediction we used this formula[43] to calculate the loss.

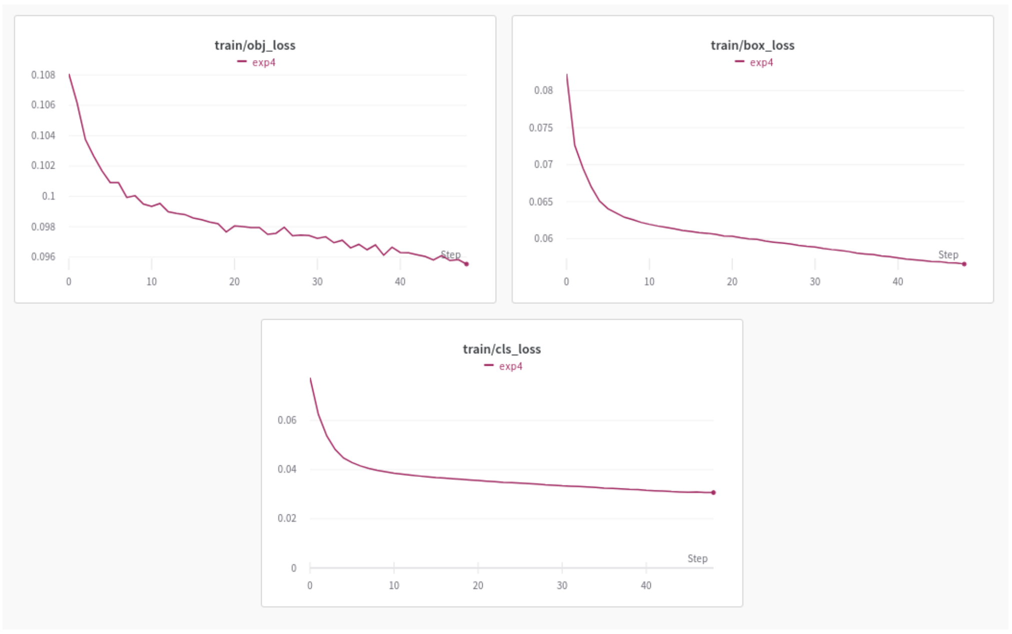


Figure 4.2: Loss while training.

We can see from the above figure 4.2 that while training the object detection model on our BDCO dataset 3 types of BCELoss is generated. As the training moves forward the object loss and the box loss reduce significantly and the classification loss reduced gradually.

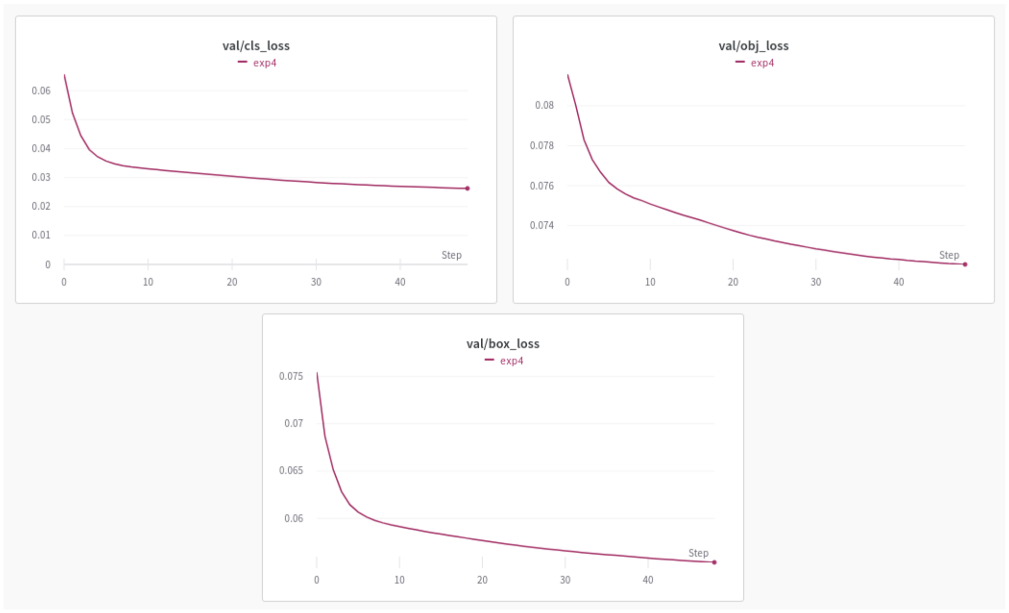


Figure 4.3: Loss while testing.

While testing the generated weights from the training, we can see in figure 4.3 that the object loss and the box loss reduces considerably and the classification loss reduces slowly.[54]

Chapter 5

Test Run on Simulator

Autonomous vehicles can manoeuvre on the road without any assistance or direct control from the driver or passenger. It can offer a great comfort and calmant journey for the passenger. As much as it has its significance it also has some caveats. As it is a new technology, people do not believe in it's potential and in some countries it has some restrictions about driving and testing an autonomous vehicle. Including this it also needs a hefty chunk of financial support to build a system around a vehicle to test it. So, we have an opportunity to try our own algorithm and try it in a simulation environment. As we found CARLA provides one of the best results among other simulation environments. We took OpenDrive's map and environment used in the CARLA simulator and used our algorithms and techniques through the CARLA API to find how it runs in that simulation.
In CARLA's default simulation setting it uses camera, radar, lidar, GNSS, IMU and many more sensors to collect data from the environment[27]. We used only one front RGB camera and two RGB cameras which are situated on the left and the right side. We used our own object detection algorithm to detect the objects.

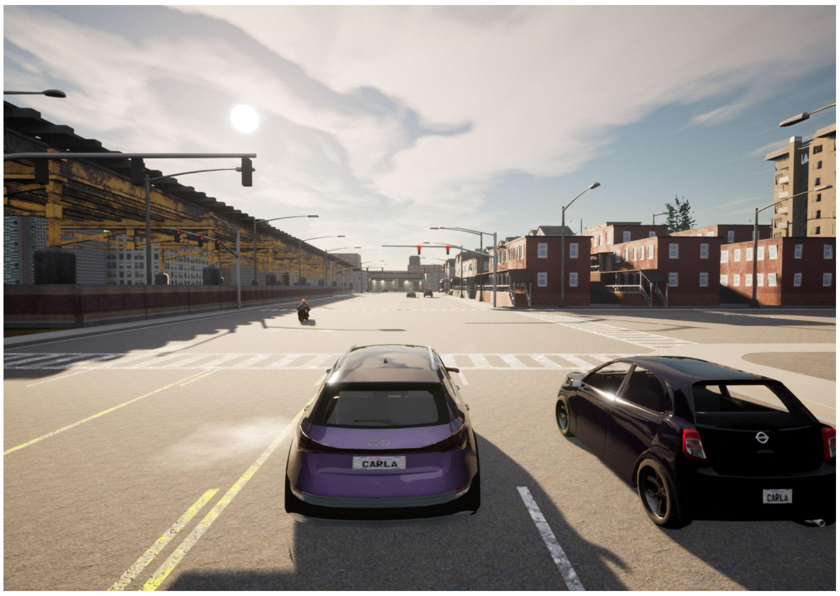


Figure 5.1: Stopping in red light.

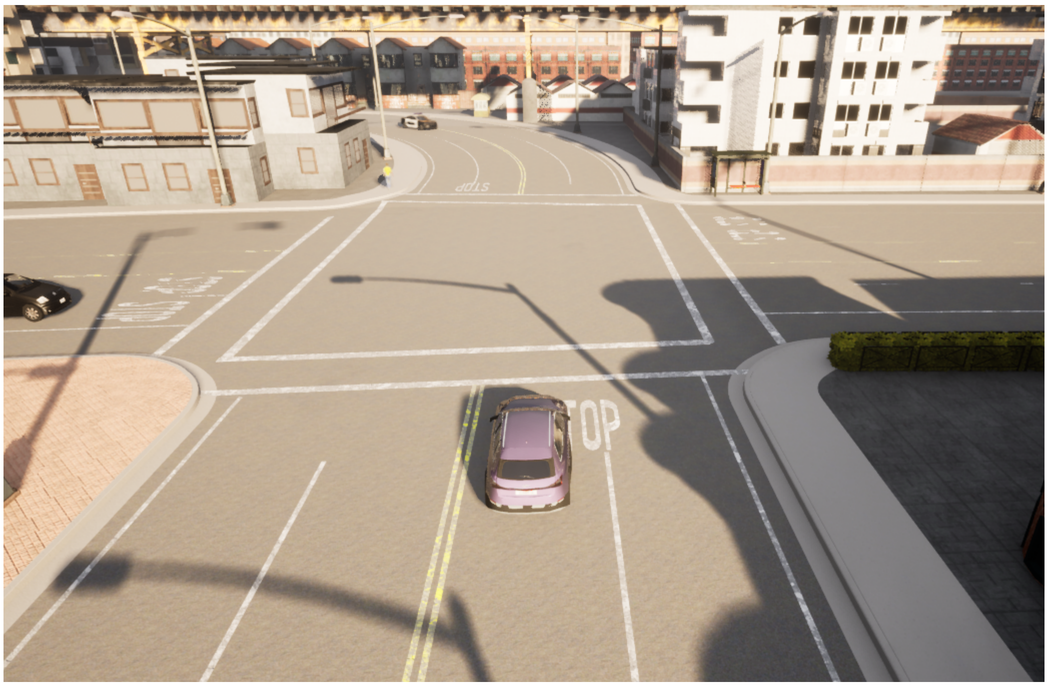


Figure 5.2: Stopping in stop sign.

It can detect other cars and stop accordingly without colliding. It can also detect traffic lights and road signs and interact with them accurately(figure 5.1, figure 5.2). For lane detection we also used our own algorithm and used the same three RGB cameras as an input. As the simulation environment has a clear and bright lane so our algorithm easily detects the lanes and drives accordingly. It follows the lane and goes through the lane on the road without taking any turn(figure 5.3).

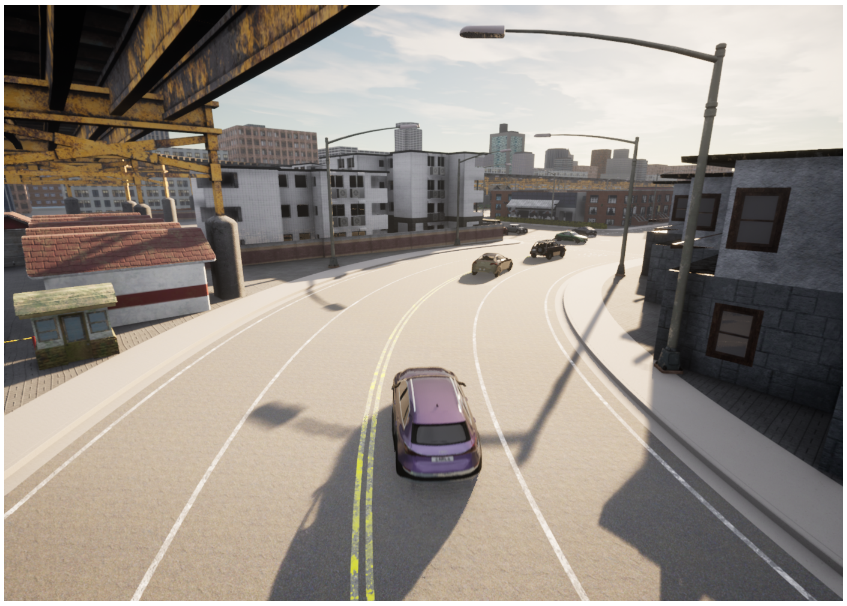


Figure 5.3: Following lane.

Chapter 6

Results and Experiments

In our autonomous assistance System, we used only 3 cameras as sensors. One at front and other two are placed at two sides of our vehicle. The cameras generate video frames as input by which we detect objects nearby and mark them with frames. It also identifies lanes and it draws lines corresponding to the lanes as output. It can also calculate the distance between other vehicles and provide suggestions to drive safely. It has also a unique feature named Steering control which gives suggestions of controlling the directions of the steering.
We programmed our model in such a way that it can perform in different conditions. So far we have tasted our system in various modes:

- Normal day mode
- Normal night mode
- Rainy day mode
- Rainy night mode

Figure 6.1 shows the accuracy test of our system in different weather conditions in Bangladesh. We accomplished the highest accuracy in normal weather condition daylight condition mode. Where the visibility of objects and lanes are the highest so our network performs the best in that category. In rainy weather condition night mode the visibility is the lowest so our network performs lowest in this category. It can be improved by using a camera outside of the car because the rain water running on the vehicle windshield causes less visibility. In normal weather conditions night mode vehicle detection accuracy is high as all other conditions but the result is less consistent.

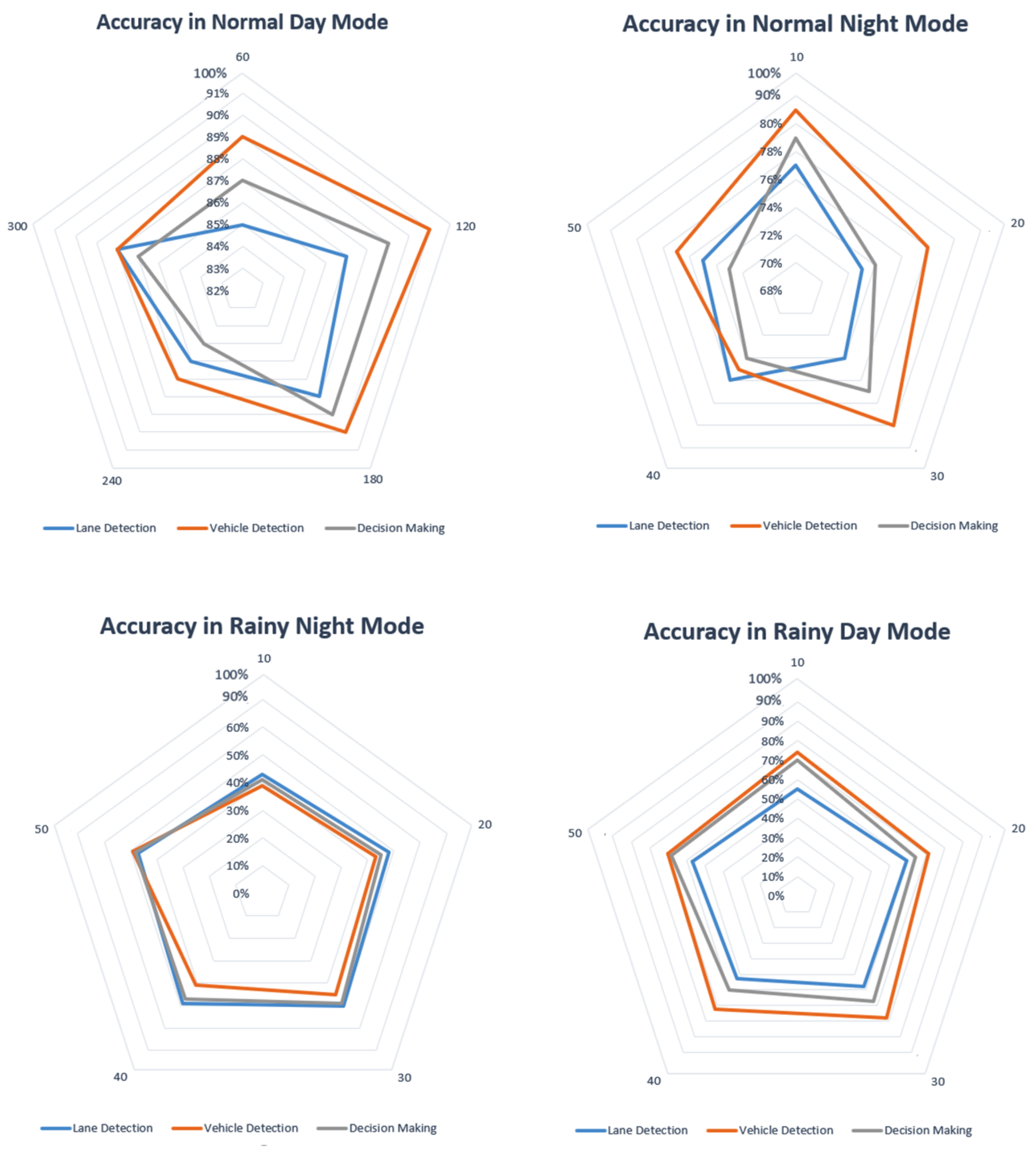


Figure 6.1: Accuracy in different weather conditions.

We have used Darknet-53[32] which is the backbone of our object detection mode. While comparing Darknet-53[32] with other methods, we have found that it is more robust than Darknet-19[25] and it also outperforms ResNet-101 and ResNet-152 in terms of productivity(table 6.1).

Backbone	**Top-1**	**Top-5**	**Billions of Operations**	**Billion Floating Points Operation per second**	**FPS**
Darknet-19[[25]]	74.1	91.8	7.29	1246	**171**
ResNet-101[[20]]	77.1	93.7	19.7	1039	53
ResNet-152[[20]]	**77.6**	**93.8**	**29.4**	1090	37
Darknet-53[[32]]	77.2	**93.8**	18.7	**1457**	78

Table 6.1: Comparison of Object detection models.

Here, we trained each network with similar configuration and checked at 256X256 single-crop validation accuracy. After analyzing, we can say that Darknet-53 achieves 1.5 times comparatively higher than ResNet-101. Though it performs identical to Resnet-152, still it is two times quicker. In addition, among other backbones it has the fastest floating point operations per second which makes the network use the GPU more efficiently which leads to the performance of this network getting boosted.

While detecting objects CNN based Object Detection(CbOD) takes data from our dataset (BDCO) and compares the objects from the input frame with the defined classes from the dataset. In the MS COCO dataset, there are 80 classes. But some classes are not important for driving such as fruits, cutlery, books etc. That is why we have merged those less important classes into one class. In Bangladeshi condition, there are some unusual objects like Rickshaw, Van, Autorickshaw etc. which are not classified in MS COCO dataset. So, we collected those unusual object's data, labeled them and merged them with MS COCO dataset so that it can perform better in Bangladeshi conditions. There were 5,81,896 images in MS COCO dataset and we added 30,257 images which is in total 6,12,153 images for training and testing. We named our newly formed dataset Bangladeshi Common Objects (BDCO) dataset. We examined the BDCO dataset with several object detection models. Figure 6.3 describes the inference accuracy versus time graph on the mAP at 0.5 IOU metric. Here, mAP represents mean average precision. The precision calculates how accurate the predictions are. It also calculates how many predictions of the model are correct. The mAP is the average of the average precision (AP) for all classes in the dataset. The formula to calculate mAP is,[42]

$$mAP = \frac{1}{N}\sum_{i=1}^{N} AP_i \tag{6.1}$$

To calculate the AP, first we need IOU(Intersection over union). The IOU is calculated by the ratio of the area of intersection and area of union of the predicted bounding box and ground truth bounding box[42]. In figure 6.2 we can see the graphical representation of IOU.

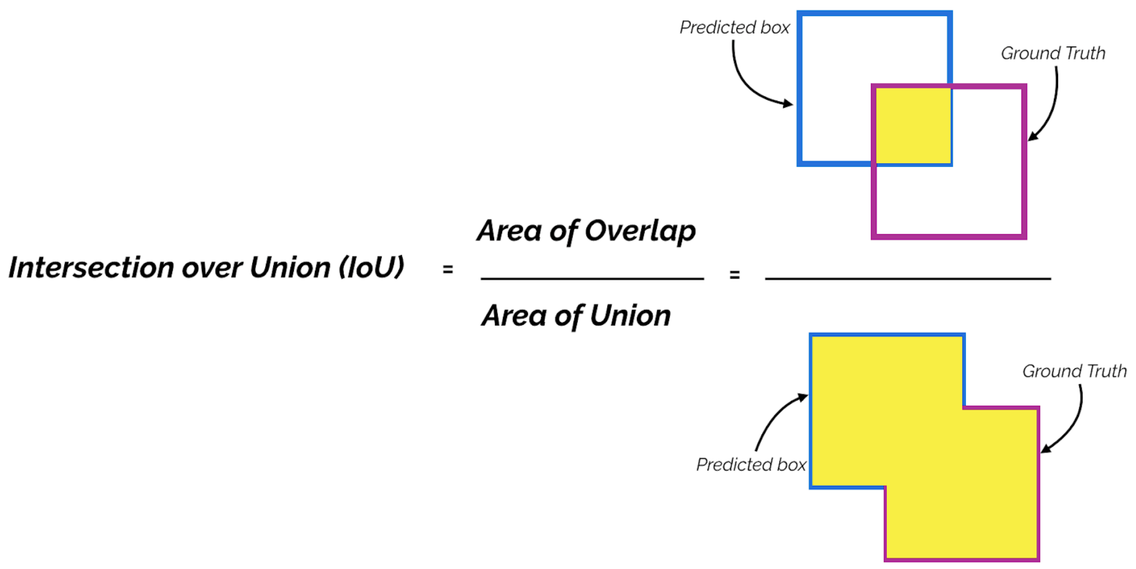
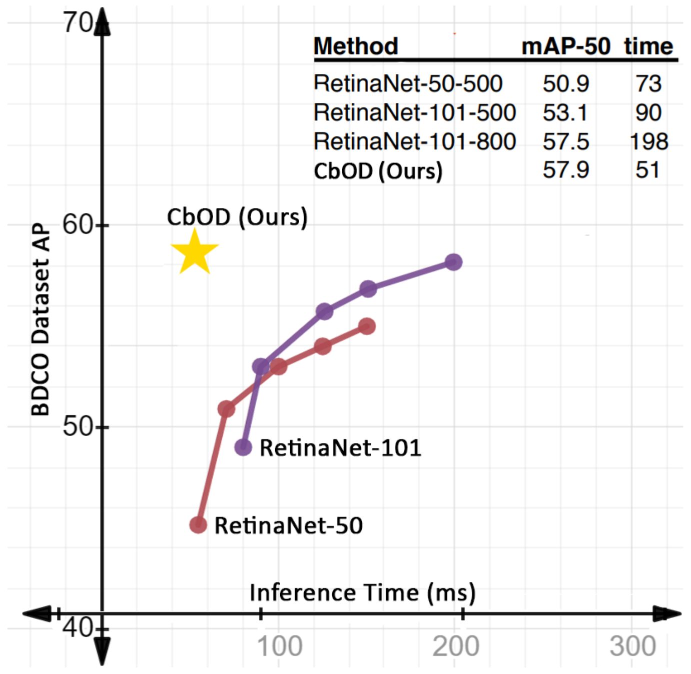


Figure 6.2: Intersection over Union.

We set the IOU at 0.5 so that it will reduce the chance of getting false positive and false negative predictions when we run our model. That is why we used the value of IOU at 0.5. Figure 6.3 shows that CbOD takes less time and provides better accuracy than other methods.



Method	mAP-50	time
RetinaNet-50-500	50.9	73
RetinaNet-101-500	53.1	90
RetinaNet-101-800	57.5	198
CbOD (Ours)	57.9	51

Figure 6.3: Inference time vs Average Precision graph.

6.1 Lane Detection(QLD)

In an environment like Bangladesh, it is very difficult to detect lanes as the lane lines are not clearly visible. In some cases there are no lane lines on the roads at all. To solve this difficult task we came up with our own lane detection model, Quick Lane Detection (QLD) especially for countries like Bangladesh. We have run two datasets in our model. Those are TUSimple and CULane datasets. The comparison between these two datasets are given in table 6.2 [46].

Dataset	#Frame	Train	Validation	Test	Resolution	#Lanes	#Scenarios	Environment
TUSimple	6408	3268	358	2782	1280X720	<=5	1	Highway
CULane	133235	88880	9675	34680	1640X590	<=4	9	Urban and Highway

Table 6.2: Comparison of datasets of lane detection.

Weather variation is one of the most challenging parts to deal with in terms of building an autonomous driving assistance system. We have tested these two datasets in different types of weather conditions. From table 6.3, we can analyze that CULane is more effective for Bangladeshi roads than TUSimple. This is why we used the CULane dataset in our lane detection model. The backbone of QLD is ResNet-18

Category	TUSimple	CULane
Normal	85.4	90.4
Crowded	61.7	70.2
Night	51.5	66.7
Curve	66.9	69.5
Total	63.6	72.3
Run Time	5.9	5.7

Table 6.3: Performance evaluation of lane detection datasets in different weather condition.

which performs better in all types of challenging weather conditions than all existing methods in terms of inference and accuracy. Though in some cases our method lacks accuracy in finding lanes, still it obtained the highest FPS and less runtime than all traditional models. Table 6.4 shows the comparison between ResNet-18 and other models in terms of different parameters[46].

Category	Res50-Seg[[31]]	SCNN [[28]]	FD-50[[40]]	Res34-SAD	SAD[[39]]	ResNet-18(Ours)
Normal	87.4	**90.6**	85.9	89.9	90.1	87.7
Crowded	64.1	64.1	**69.7**	63.6	68.5	66.0
Night	60.4	60.6	**66.1**	57.8	64.6	62.1
No Lane	38.1	38.1	**43.4**	40.6	42.2	41.6
Shadow	60.7	66.9	59.9	**67.7**	65.9	62.8
DazzleLight	54.1	58.5	57.0	59.9	**60.2**	58.4
Curve	59.8	64.4	65.2	**66.0**	65.7	57.9
Cross Road	2505	1990	7013	1960	1998	**1743**
Total	66.7	71.6	-	70.7	70.8	68.4
Run Time	-	133.5	-	50.5	13.4	3.1
FPS	-	7.5	-	19.8	74.6	322.5

Table 6.4: Comparison between different lane detection backbone.

6.2 Output

Bangladeshi Driving Assistant(BD-DA) is mainly based on cameras. Using the camera our model takes video frames as input to detect objects and lanes. It has four predefined zones. After performing all the calculations, depending on the zones our system provides driving suggestions.

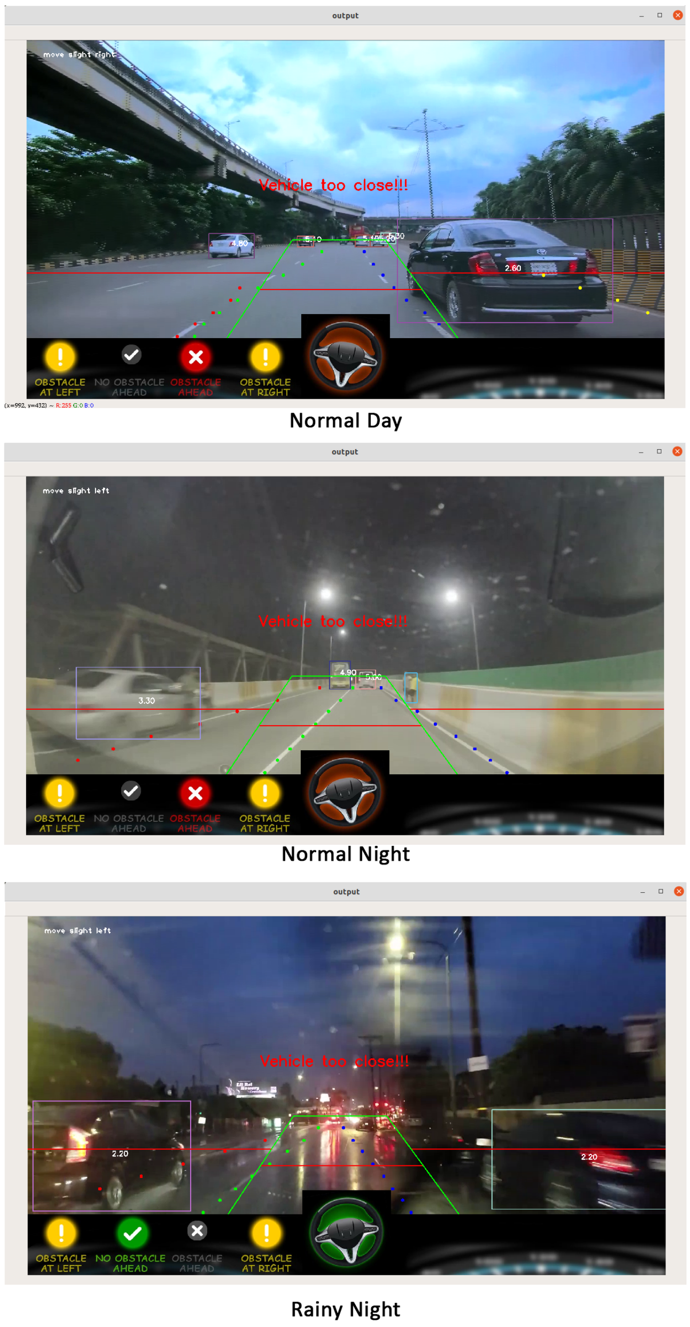


Figure 6.4: Output of the whole system.

It also can predict the approximate steering angle for staying in the lane. Figure 6.4 shows that our system is running all the operations smoothly in different weather conditions. Using CbOD our system also can identify the traffic signals. From that information it provides a warning when CbOD detects any traffic signal light. Figure 6.5 shows when the traffic light is detected in the right corner of the video frame our system is giving a warning in rainy weather.

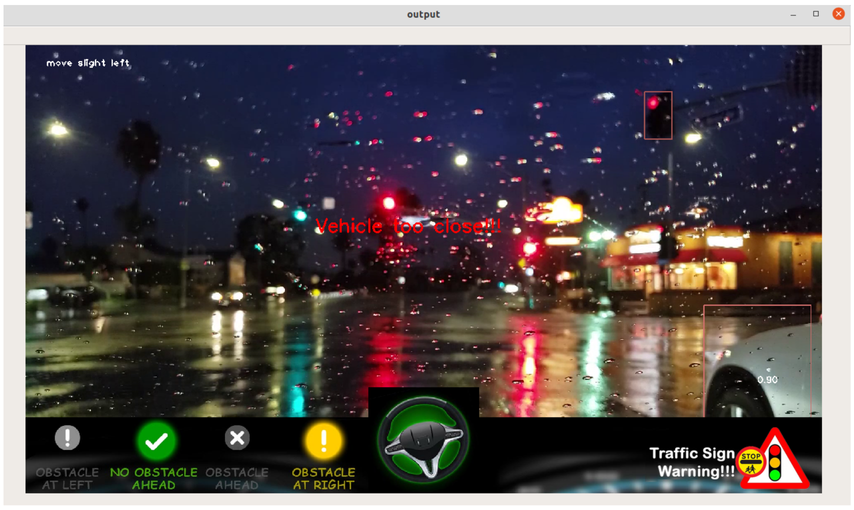


Figure 6.5: Traffic sign Warning.

Here, we are using one camera for the whole system. To increase the accuracy of the cascaded network we added two more cameras to collect data about the surroundings of the vehicle and surprisingly the accuracy rate increases and the system can make decisions more precisely. Figure 6.6 shows that by using extra two cameras our system can predict beforehand and can give decisions more accurately.

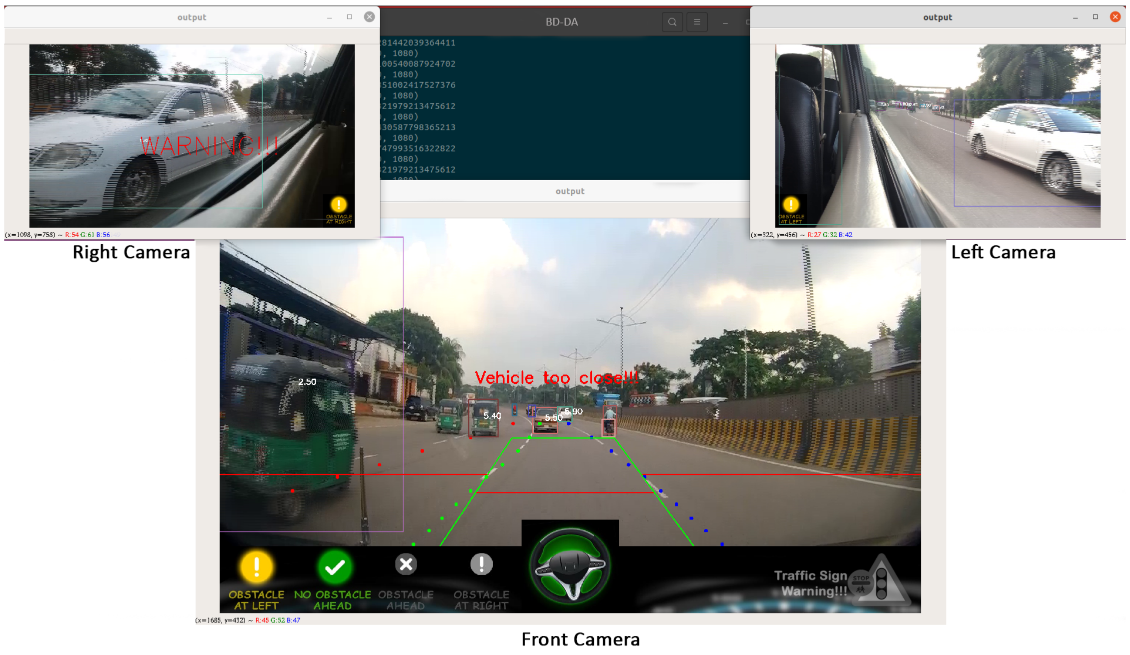

```
BD-DA
81442039364411
, 1080)
00540087924702
, 1080)
51002417527376
, 1080)
21979213475612
, 1080)
30587798365213
, 1080)
47993516322822
, 1080)
21979213475612
```

Figure 6.6: Output of the three different cameras.

Chapter 7

Conclusion and Future Work

Self-driving cars are not in people's dreams anymore. There are a lot of companies that are becoming more interested in investing in this platform by which this sector is expanding day by day. It is not easy for someone to accomplish a fully functional autonomous vehicle. Yet we will try to overcome all the barriers & research problems to get the maximum accuracy rate of the driverless autonomous car. The development, demand, advantages, costs, and travel implications of autonomous vehicles are all loaded with uncertainties. The numerous encounters with often-unpredictable people and cars make driving on public roadways very difficult. Before autonomous vehicles can function effectively in mixed urban traffic and various weather conditions of our country, significant development is required.

Our future goal is to implement our model on physical cars and bring change in the whole autonomous vehicle sector in the South-Asian continent. Moreover, researching thoroughly from other research sectors and gaining knowledge about small critical things and trying to improve the whole system by implementing more features. To improve our model, we tend to include a driver monitoring system by which we will be able to monitor the behaviour of the driver and show warnings accordingly. Moreover, by implementing lidar and radar, we will be able to detect lanes and objects with more efficiency. Autonomous vehicles can not reach their destination without the help of GPS. So installing GPS is a must. Our model will be more helpful with some features like emergency stop, maintaining speed, overtaking, turning and crossing on going vehicles etc which falls under the Urban analysis sector. We have listed our plans accordingly to accomplish these goals.

All the results that we have got after many days of collecting data and training them, we can say that our system indicates a system that is appropriate not only for an overcrowded place like Bangladesh but also for other countries with less environmental inconvenience.

Bibliography

[1] M. Minsky and S. Papert, “Perceptrons - an introduction to computational geometry,” 1969.

[2] P. Werbos, *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences*. Harvard University, 1975. [Online]. Available: https://books.google.ca/books?id=z81XmgEACAAJ.

[3] D. Rumelhart and J. L. McClelland, “Parallel distributed processing: Explorations in the microstructure of cognition, vol. 1: Foundations,” 1986.

[4] W. McCulloch and W. Pitts, “A logical calculus of the ideas immanent in nervous activity,” *Bulletin of Mathematical Biology*, vol. 52, pp. 99–115, 1990.

[5] J. Schmidhuber, “Learning complex, extended sequences using the principle of history compression,” *Neural Computation*, vol. 4, no. 2, pp. 234–242, 1992. DOI: 10.1162/neco.1992.4.2.234.

[6] I. Engel and N. Bershad, “A transient learning comparison of rosenblatt, backpropagation, and lms algorithms for a single-layer perceptron for system identification,” *IEEE Transactions on Signal Processing*, vol. 42, no. 5, pp. 1247–1251, 1994. DOI: 10.1109/78.295190.

[7] Y. LeCun, P. Haffner, L. Bottou, and Y. Bengio, “Object recognition with gradient-based learning,” in *Shape, Contour and Grouping in Computer Vision*. Berlin, Heidelberg: Springer Berlin Heidelberg, 1999, pp. 319–345.

[8] A. Girard, S. Spry, and J. Hedrick, “Intelligent cruise control applications: Real-time embedded hybrid control software,” *IEEE Robotics Automation Magazine*, vol. 12, no. 1, pp. 22–28, 2005. DOI: 10.1109/MRA.2005.1411415.

[9] K. Maniruzzaman and R. Mitra, “Road accidents in bangladesh,” *IATSS Research*, vol. 29, Dec. 2005. DOI: 10.1016/S0386-1112(14)60136-9.

[10] D. Steinkrau, P. Simard, and I. Buck, “Using gpus for machine learning algorithms,” *Eighth International Conference on Document Analysis and Recognition (ICDAR’05)*, 1115–1120 Vol. 2, 2005.

[11] M. Alam, S. M. Mahmud, and M. Hoque, “Road accident trends in bangladesh: A comprehensive study,” Dec. 2011.

[12] A. Shaout, D. Colella, and S. Awad, “Advanced driver assistance systems - past, present and future,” *2011 Seventh International Computer Engineering Conference (ICENCO’2011)*, pp. 72–82, 2011.

[13] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “Imagenet classification with deep convolutional neural networks,” in *Advances in Neural Information Processing Systems*, F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, Eds., vol. 25, Curran Associates, Inc., 2012. [Online]. Available: https://proceedings.neurips.cc/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf.

[14] D. Prasad, “Survey of the problem of object detection in real images,” *International Journal of Image Processing (IJIP)*, vol. 6, p. 441, Dec. 2012.

[15] 2014. [Online]. Available: https://news.jardinemotors.co.uk/lifestyle/the-history-of-car-technology.

[16] K. He, X. Zhang, S. Ren, and J. Sun, “Spatial pyramid pooling in deep convolutional networks for visual recognition,” *CoRR*, vol. abs/1406.4729, 2014. arXiv: 1406.4729. [Online]. Available: http://arxiv.org/abs/1406.4729.

[17] T.-Y. Lin, M. Maire, S. J. Belongie, L. D. Bourdev, R. B. Girshick, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, “Microsoft COCO: common objects in context,” *CoRR*, vol. abs/1405.0312, 2014. arXiv: 1405.0312. [Online]. Available: http://arxiv.org/abs/1405.0312.

[18] R. B. Girshick, “Fast R-CNN,” *CoRR*, vol. abs/1504.08083, 2015. arXiv: 1504.08083. [Online]. Available: http://arxiv.org/abs/1504.08083.

[19] D. Gonzalez Bautista, J. Pérez, V. Milanes, and F. Nashashibi, “A review of motion planning techniques for automated vehicles,” *IEEE Transactions on Intelligent Transportation Systems*, pp. 1–11, Nov. 2015. DOI: 10.1109/TITS.2015.2498841.

[20] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” *CoRR*, vol. abs/1512.03385, 2015. arXiv: 1512.03385. [Online]. Available: http://arxiv.org/abs/1512.03385.

[21] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. E. Reed, C.-Y. Fu, and A. C. Berg, “SSD: single shot multibox detector,” *CoRR*, vol. abs/1512.02325, 2015. arXiv: 1512.02325. [Online]. Available: http://arxiv.org/abs/1512.02325.

[22] J. Redmon, S. K. Divvala, R. B. Girshick, and A. Farhadi, “You only look once: Unified, real-time object detection,” *CoRR*, vol. abs/1506.02640, 2015. arXiv: 1506.02640. [Online]. Available: http://arxiv.org/abs/1506.02640.

[23] F. Furrer, M. Burri, M. Achtelik, and R. Siegwart, “Rotors – a modular gazebo mav simulator framework,” in. Jan. 2016, vol. 625, pp. 595–625, ISBN: 978-3-319-26054-9. DOI: 10.1007/978-3-319-26054-9_23.

[24] B. Paden, M. Cáp, S. Z. Yong, D. S. Yershov, and E. Frazzoli, “A survey of motion planning and control techniques for self-driving urban vehicles,” *CoRR*, vol. abs/1604.07446, 2016. arXiv: 1604.07446. [Online]. Available: http://arxiv.org/abs/1604.07446.

[25] J. Redmon and A. Farhadi, “YOLO9000: better, faster, stronger,” *CoRR*, vol. abs/1612.08242, 2016. arXiv: 1612.08242. [Online]. Available: http://arxiv.org/abs/1612.08242.

[26] E. Santana and G. Hotz, “Learning a driving simulator,” *CoRR*, vol. abs/1608.01230, 2016. arXiv: 1608.01230. [Online]. Available: http://arxiv.org/abs/1608.01230.

[27] A. Dosovitskiy, G. Ros, F. Codevilla, A. M. López, and V. Koltun, "CARLA: an open urban driving simulator," *CoRR*, vol. abs/1711.03938, 2017. arXiv: 1711.03938. [Online]. Available: http://arxiv.org/abs/1711.03938.

[28] X. Pan, J. Shi, P. Luo, X. Wang, and X. Tang, "Spatial as deep: Spatial CNN for traffic scene understanding," *CoRR*, vol. abs/1712.06080, 2017. arXiv: 1712.06080. [Online]. Available: http://arxiv.org/abs/1712.06080.

[29] S. D. Pendleton, H. Andersen, X. Du, X. Shen, M. Meghjani, Y. H. Eng, D. Rus, and M. H. Ang, "Perception, planning, control, and coordination for autonomous vehicles," *Machines*, vol. 5, no. 1, 2017, ISSN: 2075-1702. DOI: 10.3390/machines5010006. [Online]. Available: https://www.mdpi.com/2075-1702/5/1/6.

[30] 2018. [Online]. Available: https://software.intel.com/content/www/us/en/develop/download/developer-reference-for-intel-integrated-performance-primitives-intel-ipp-volume-2.html.

[31] L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille, "Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 40, no. 4, pp. 834–848, 2018. DOI: 10.1109/TPAMI.2017.2699184.

[32] J. Redmon and A. Farhadi, "Yolov3: An incremental improvement," *CoRR*, vol. abs/1804.02767, 2018. arXiv: 1804.02767. [Online]. Available: http://arxiv.org/abs/1804.02767.

[33] A. Sherstinsky, "Fundamentals of recurrent neural network (RNN) and long short-term memory (LSTM) network," *CoRR*, vol. abs/1808.03314, 2018. arXiv: 1808.03314. [Online]. Available: http://arxiv.org/abs/1808.03314.

[34] 2019. [Online]. Available: https://www.nhtsa.gov/technology-innovation/automated-vehicles-safety.

[35] M. Galvani, "History and future of driver assistance," *IEEE Instrumentation & Measurement Magazine*, vol. 22, pp. 11–16, 2019.

[36] M. Hannemose, J. Wilm, and J. Frisvad, "Superaccurate camera calibration via inverse rendering," Jun. 2019, p. 40. DOI: 10.1117/12.2531769.

[37] G. Heinzelman, "Autonomous vehicles, ethics of progress," Apr. 2019. DOI: 10.13140/RG.2.2.28046.31048.

[38] M. S. Hossen, "Analysis of road accidents in bangladesh," vol. 2, Mar. 2019. DOI: 10.28933/ajtl-2018-12-1608.

[39] Y. Hou, Z. Ma, C. Liu, and C. C. Loy, "Learning lightweight lane detection cnns by self attention distillation," *CoRR*, vol. abs/1908.00821, 2019. arXiv: 1908.00821. [Online]. Available: http://arxiv.org/abs/1908.00821.

[40] J. Philion, "Fastdraw: Addressing the long tail of lane detection by adapting a sequential prediction network," *CoRR*, vol. abs/1905.04354, 2019. arXiv: 1905.04354. [Online]. Available: http://arxiv.org/abs/1905.04354.

[41] P. Sun, H. Kretzschmar, X. Dotiwalla, A. Chouard, V. Patnaik, P. Tsui, J. Guo, Y. Zhou, Y. Chai, B. Caine, V. Vasudevan, W. Han, J. Ngiam, H. Zhao, A. Timofeev, S. Ettinger, M. Krivokon, A. Gao, A. Joshi, Y. Zhang, J. Shlens, Z. Chen, and D. Anguelov, "Scalability in perception for autonomous driving: Waymo open dataset," *CoRR*, vol. abs/1912.04838, 2019. arXiv: 1912.04838. [Online]. Available: http://arxiv.org/abs/1912.04838.

[42] R. J. Tan, *Breaking down mean average precision (map) - towards data science*, Mar. 2019. [Online]. Available: https://towardsdatascience.com/breaking-down-mean-average-precision-map-ae462f623a52#1a59.

[43] 2020. [Online]. Available: https://www.fatalerrors.org/a/detailed-explanation-of-yolov3-loss-function.html.

[44] A. Bochkovskiy, C.-Y. Wang, and H.-Y. M. Liao, "Yolov4: Optimal speed and accuracy of object detection," *CoRR*, vol. abs/2004.10934, 2020. arXiv: 2004.10934. [Online]. Available: https://arxiv.org/abs/2004.10934.

[45] F. Munir, S. Azam, and M. Jeon, "Ldnet: End-to-end lane detection approach usinga dynamic vision sensor," *CoRR*, vol. abs/2009.08020, 2020. arXiv: 2009.08020. [Online]. Available: https://arxiv.org/abs/2009.08020.

[46] Z. Qin, H. Wang, and X. Li, "Ultra fast structure-aware deep lane detection," *CoRR*, vol. abs/2004.11757, 2020. arXiv: 2004.11757. [Online]. Available: https://arxiv.org/abs/2004.11757.

[47] 2021. [Online]. Available: https://www.mathworks.com/help/simulink/simulation.html.

[48] X. Li, K.-Y. Lin, M. Meng, X. Li, L. Li, and Y. Hong, "Composition and application of current advanced driving assistance system: A review," *CoRR*, vol. abs/2105.12348, 2021. arXiv: 2105.12348. [Online]. Available: https://arxiv.org/abs/2105.12348.

[49] L. Liu, X. Chen, S. Zhu, and P. Tan, "Condlanenet: A top-to-down lane detection framework based on conditional convolution," *CoRR*, vol. abs/2105.05003, 2021. arXiv: 2105.05003. [Online]. Available: https://arxiv.org/abs/2105.05003.

[50] Z. Qu, H. Jin, Y. Zhou, Z. Yang, and W. Zhang, "Focus on local: Detecting lane marker from bottom up via key point," *CoRR*, vol. abs/2105.13680, 2021. arXiv: 2105.13680. [Online]. Available: https://arxiv.org/abs/2105.13680.

[51] [Online]. Available: https://www.aptiv.com/docs/default-source/white-papers/2021_aptiv_whitepaper_nextgenadasplatform.pdf.

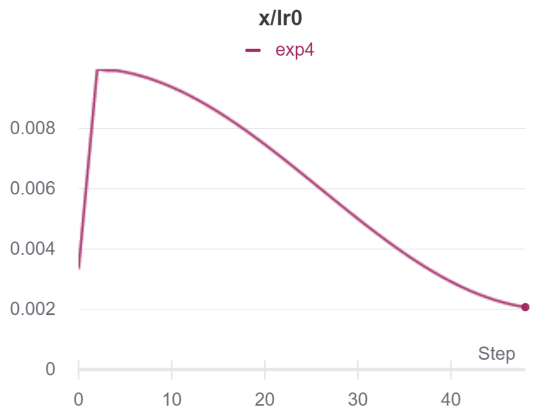
[52] I. Leneman, I. Verburg, and I. De, *PreScan, testing and developing active safety applications through simulation*. [Online]. Available: https://mediatum.ub.tum.de/doc/1145111/1145111.pdf.

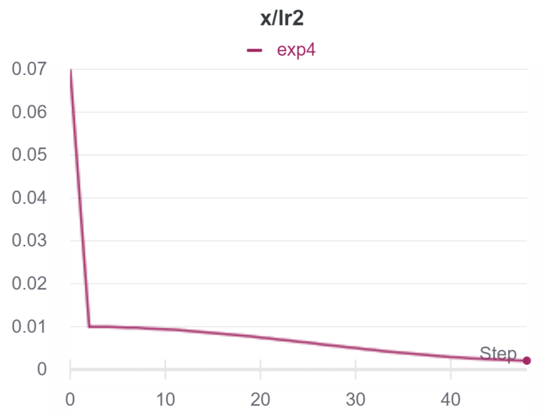
[53] shrekqie, *Detailed explanation of YOLOv3 loss function*. [Online]. Available: https://www.fatalerrors.org/a/detailed-explanation-of-yolov3-loss-function.html (visited on 09/01/2020).

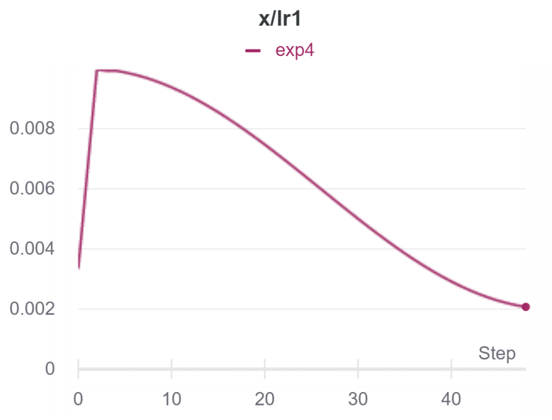
[54] *Weights and biases*. [Online]. Available: https://wandb.ai/site.

Appendix A

The learning rates during the training are shown in the following graphs. These graphs are produced from wandb during training.[54]

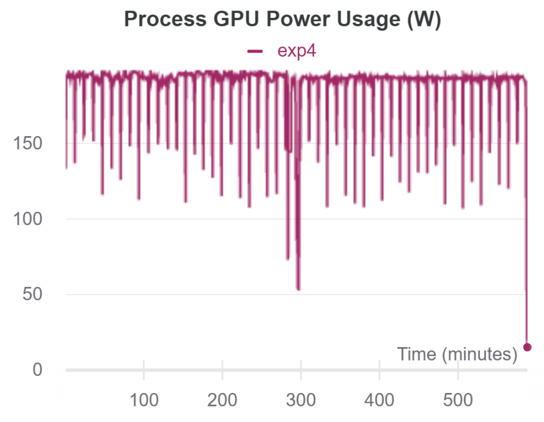


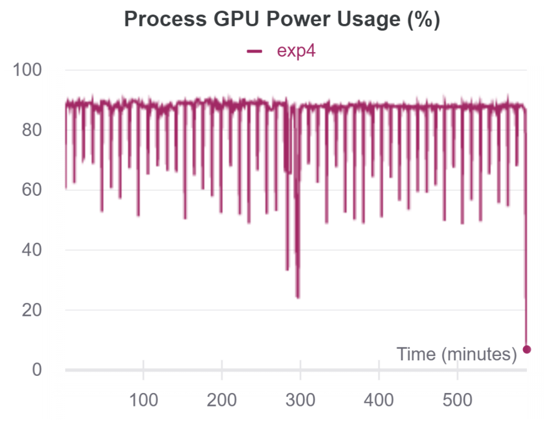


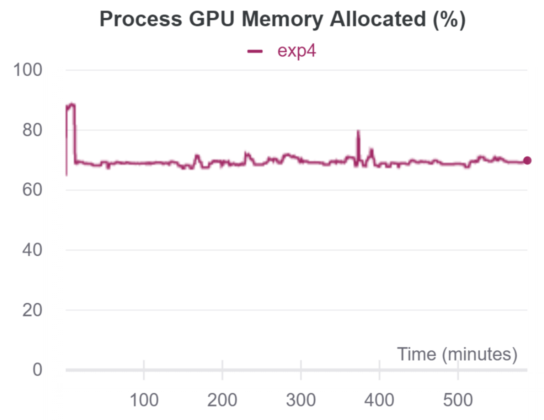


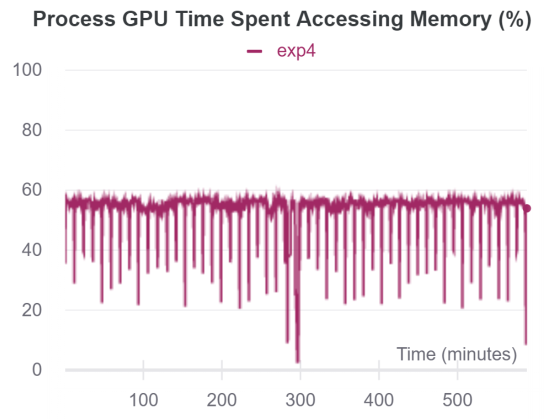
Appendix B

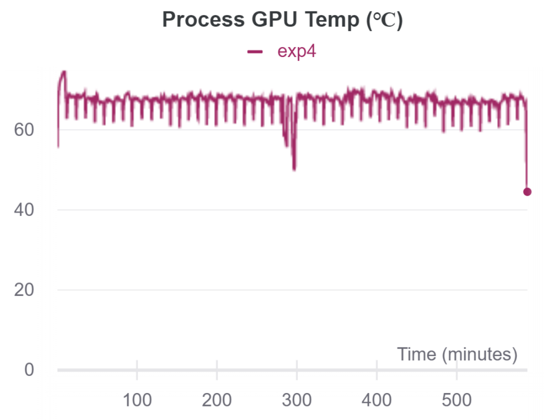
We have used AMD Ryzen 3700x, NVIDIA RTX 3070, 16gb ram for training our system. The following graphs show The GPU power usage, memory allocated percentage, temperature and utilization percentages. These graphs are generated from wandb.[54]

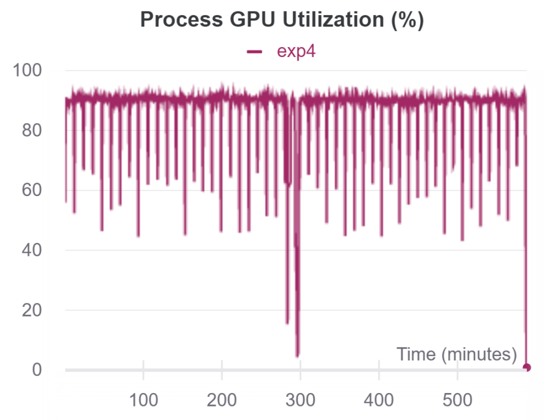

Process GPU Power Usage (W)
exp4
Time (minutes)


Process GPU Power Usage (%)
exp4
Time (minutes)


Process GPU Memory Allocated (%)
exp4
Time (minutes)


Process GPU Time Spent Accessing Memory (%)
exp4
Time (minutes)


Process GPU Temp (℃)
exp4
Time (minutes)


Process GPU Utilization (%)
exp4
Time (minutes)