Final Thesis Report

Analysis of Training Time Optimization For Self-Driving Car Using Alternate Max Pooling Layers

Group Members

Kazi Ridwan Arefin

13101212

Md. Mashrukul Huque

13101232

Sheikh Sadaf Tasin

13321074

Ahmed Jawad Khan

14101258

Aareef Abrar

17301238

Supervisor:
Dr. Md. Ashraful Alam

Declaration

We hereby declare that this is an original report written by us with our own findings, and has not been published or presented in parts or as a whole for any other previous degree. Resources and materials by other researchers used as guidelines for our research are duly mentioned in reference citations.

Signature of the Authors:	Signature of the Supervisor:

_____
Kazi Ridwan Arefin

_____
Dr. Md. Ashraful Alam
Assistant Professor
Dept. of Computer Science and Engineering
BRAC University

_____
Md. Mashrukul Huque

_____
Sheikh Sadaf Tasin

_____
Ahmed Jawad Khan

_____
Aareef Abrar

ACKNOWLEDGEMENT

To start off, we would like to express our heartiest gratitude towards the Almighty Allah. Secondly, we would like to share our sincere gratitude to our advisor Dr. Md. Ashraful Alam for his constant motivation, support and immense knowledge to our research. It would not have been possible without his constant guidance in all phases of our project. Also, special thanks to Mr. Nazm Anwar, Ms. Tahseen Reza Anika, Ms. Afia Alam Toma and Ms Nuha Annoor Paboni for helping us with creating the architectural design of the project. We would also thank Mr.Khan Rafin Ahmed for motivating and helping us throughout the journey.

Finally, we would like to thank our gratefulness to parents, brothers and sisters for encouraging and supporting us from the very beginning.

Abstract

In the modern era, the vehicles are focused to be automated to give human driver relaxed driving. In the field of automobile various aspects have been considered which makes a vehicle automated. As Udacity, in 2016, with the Google Self-Driving Car founder Sebastian Thrun open sourced their self driving car simulation environment, a new door opened up in self driving vehicle research. In this paper we have focused on comparing between a popular neural network model introduced by NVIDIA and a model made with max pooling optimized neural network. Max pooling is a method of making Convolutional Neural Network simpler. We have also worked vigorously on creating a custom urban environment based on Dhaka, where we have run the car autonomously, gathered the training time data, and the instances it fails to drive along the trained roadway. The idea described in this paper is related to deep learning algorithm analysis and comparison, which is the core part of the self driving car. The analysis hence gives us a great realization of machine learning techniques and their effectiveness in practical situations. All being said, this paper, however approaches to solve one big problem, to find out practicality of a popular model compared to an unconventional model, put in a real scenario.

Keywords: Autonomous, Self Driving Car, NVIDIA Model, Max-pooling

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

Chapter 1

INTRODUCTION

Self driving technology has been coming of age in the automotive industry since last decade; a self driving vehicle is essentially a vehicle that can drive itself without any human intervention. Mankind have tried numerous strategies for its implementation throughout ages; form observing the history we can see that self driven cars have captivated the imagination of mankind since 1920s, and through failed attempts mankind have finally made driverless vehicle coming true through 80's.[1]

Through ages, self driving cars have tried to make use of concurrent technologies and an effective implementation of them. The first ever examples of successful self driving car, DARPA-funded Autonomous Land Vehicle (ALV), was based on an amazing technology of computer vision, which was new at the time. It could move about 2000 feet of complex terrain just based on it's vision sensors with magic of computer vision. Then in 1989, Carnegie Mellon University pioneered the use of neural networks to make self driving cars, which, still acts as the basis of driving control strategy of self driving cars.

Our thesis is centered around the fact of how powerful the modern field of neural network and machine learning is; and how it goes beyond just pattern recognition and actually mimics human behavior in real time with great accuracy.

1.1 Motivation

From the invention of the first pneumatic tire by Robert William Thomson to a full-fledged modern car, us, human beings have always thrived on looking for ways to make our day to day transportation easier, safer and more comfortable. Imagining a car without a driver was once completely unimaginable but with time, it is becoming a part of our reality. The safety and comfort of a passenger depends highly on the one who drives the vehicle. One of the

disadvantages we have compared to a robot (or an AI) is that, we, humans are susceptible to tiredness, fatigue and stress. According to a safety report of the National Highway Traffic Safety Administration of the United States (NHTSA) [2], there has been 56,000 reported cases of road crashes which happened due to the drowsiness and fatigue of the drivers in the United States of America. Another USA based research published by National Transportation Safety Board (NTSB) in 1999 showed that around 30 % fatal accidents were fatigue-related. [3]

Safety is without a doubt one of the most important issues. There are more problems that are caused by inefficient and careless drivers and one of them is the traffic jams. Gridlocks waste hours of our precious time and drain out the energy from our bodies, causing a huge loss of efficiency at our work. A research in 2013 showed that Bangladeshi economy had to face a loss of around USD 3868 million due to traffic congestion. [4]

We believe that it is possible to save precious lives and prevent huge economic failure by using a smart, reliable and efficient AI to drive the car. But there are a few obstacles on the path to achieve the ultimate smart autonomous vehicle. Implementing state-of-the-art methods of self-driving car in 3D models of Bangladeshi roads has its own set of challenges. A simulator usually provides a controlled environment but in the case of Bangladeshi roads, there are numerous challenges that occur. These challenges include, broken and uneven roads, ignorant pedestrians and drivers who do not abide by the traffic rules, unpredictable traffic flow, lack of road dividers, poor quality of road stripes and the list go on.

1.2 Objectives

We have created a self-driving car simulator based on sub-continental environment. This will perform as:

- First autonomous car simulator to run on actual, accurate 3D model of Bangladeshi roads
- Significant reduction in training time of an autonomous vehicle using max pooling layers.
- A stepping stone towards our goal of digitalization of roads and transport system

1.3 Report Overview

Chapter 1 contains the formal instructions for the report. This includes the introduction, motivation and objective of the proposed system. Chapter 2 contains the literature review; history of autonomous vehicles. In chapter 3 discussion is made on algorithm, system implementation. Chapter 4 contains experimental setup for the paper. Lastly, Chapter 6 contains the conclusion.

Chapter 2

LITERATURE REVIEW

2.1 History of Autonomous Vehicles

In the past, a self driving car was hard to imagine. Ernst Dickmann was a pioneer for the self driving car field in the 90's. He implemented a technique called Stop-and-go-behavior which is a basic maneuvering capability when driving behind a single vehicle. The first fully vision-based demonstrations of this capability were done in 1991 in the framework of an intermediate demo for the European project known as Prometheus which was held in Turin, Italy, on a separate test track. Performing this maneuver for Dickmann was a challenging task, the purpose was to roam around in a driverless vehicle safely in an urban environment with a multitude of different moving subjects (cars, vans, trucks, bicycles, humans, and animals of all kinds). He did not succeed yet. In 2005, the U.S.-Defense Advanced Research Project Agency (DARPA) initiated a challenge called "Urban Grand Challenge" that had set a price of $ 2 million for the autonomous vehicle capable of driving a 60-mile distance in an urban environment in the least amount of time (below 6 hours as additional constraint); this included reacting to other vehicles, also in stop-and-go traffic. The final test of this "Urban Grand Challenge" with a maximum allowed speed of 20 mph was planned for the 3rd of November 2007 in a mock-up town in the United States.[5]

Another pioneer Dean A. Pomerleau, who developed ALVINN (Autonomous Land Vehicle In a Neural Network) is mentionable because of his work with vehicle and Neural Network. With 3 layer back propagation designed for road following. Currently ALVINN takes images from a

camera and a laser range finder as input and produces as output the direction the vehicle should travel in order to follow the road.[6]

In March 2012, the state of Nevada in the United States amended the public roads laws and approved the test of autonomous vehicle. These regulations also stated that certain requirements must be fulfilled in order to test a particular company's vehicle. Many more countries are also joining in this cause and making the testing of self driven vehicle permissible. However, how safe is it? That is a question that remains. Several problems such as "co-operation of machine control to driver's operation", "safety on roads" and "reliability of automation technologies in self-driving cars", should be addressed. [7]

2.2 NVIDIA Model

Nvidia's motivation behind designing the algorithm was to create a system that does not require the complicated if and else statements that are required to be implemented for the recognition of terms such as the lane markings,guard rails, even other cars.

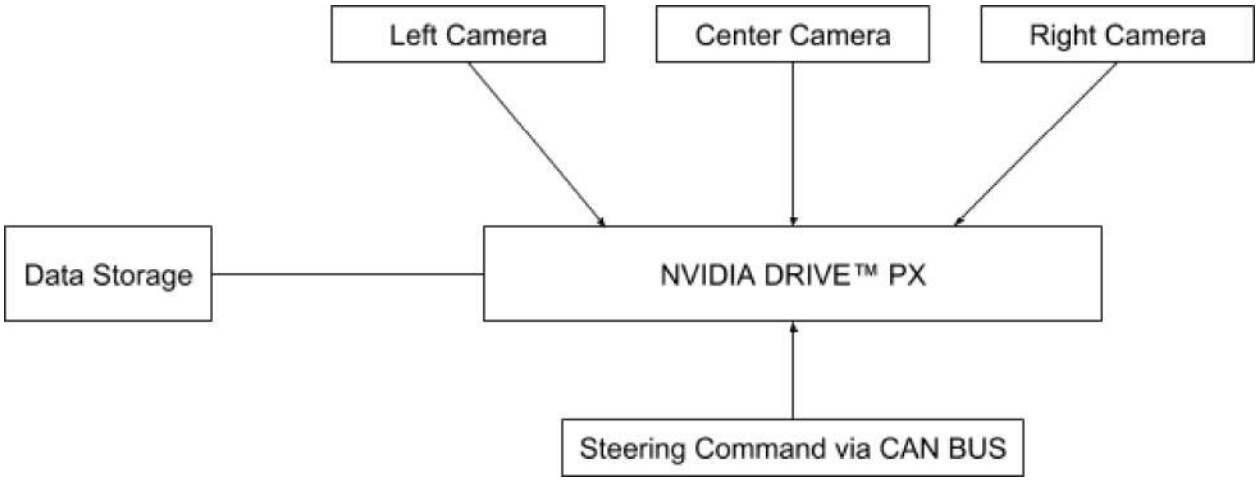

Figure 2.2.1: NVIDIA's Implementation Model

For the implementation of this algorithm, we need to start by first training the model with the data acquired by the three cameras mounted behind the windshield of the car that is designated to

collect the data for the process. Pictures that are captured from the cameras are combined with steering angles (collected through Controller Area Network(CAN)) of the car while a driver drives the vehicle. Steering command is represented by 1/r, where r is the radius in meters, and 1/r is used to prevent singularity when driving straight.

Obviously the training data on its own will not be enough for the implementation. For that we will have to teach the network to learn from its mistakes, and recover. Otherwise the car will drift off road.

Since 3D scanning is not being present,we will not be able to detect objects such as poles and trees, we are therefore assuming that all the points below the horizon are flat ground and the points above it are infinitely far away. Fortunately this does not cause a problem for the training of the machine. A block diagram of the training model is shown below. The CNN (Convolutional Neural Network) takes the pictures from the stored images and generates a proposed steering command. The actual command is compared to the generated command and the weights of the CNN are modified to bring the output closer to the desired output. This is done with the help of back propagation.

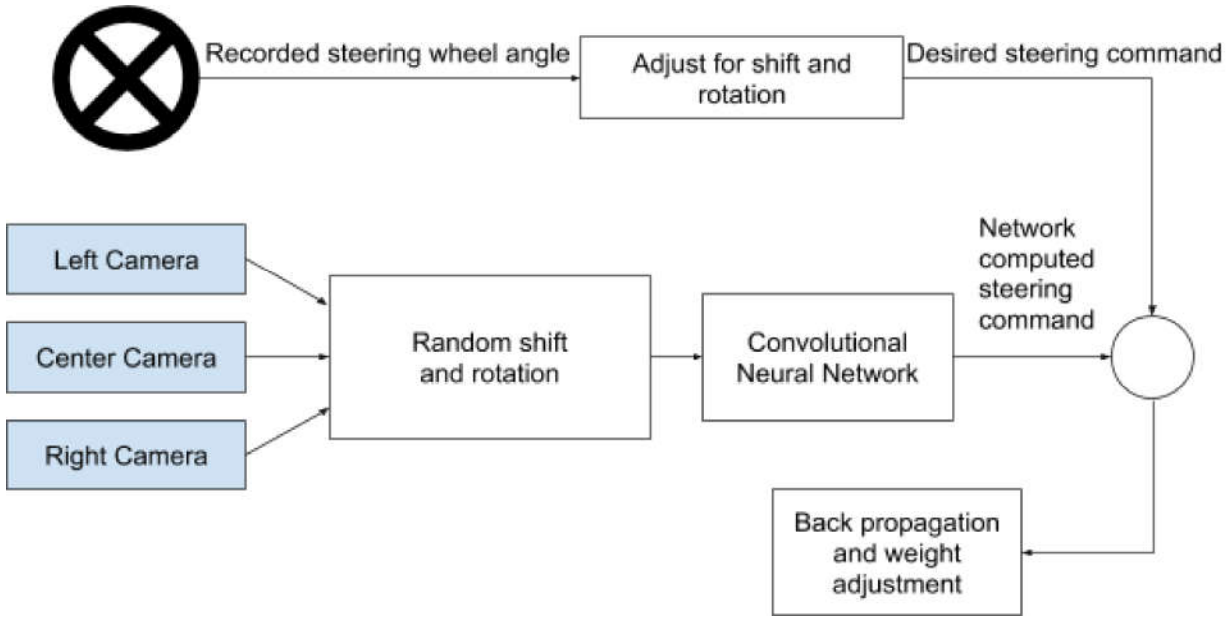

Figure 2.2.2: NVIDIA's Training Model

Once it is trained, the network can generate the angles required by the vehicle to travel through the roads with the help of a single center camera, as shown by the block diagram below. [8]

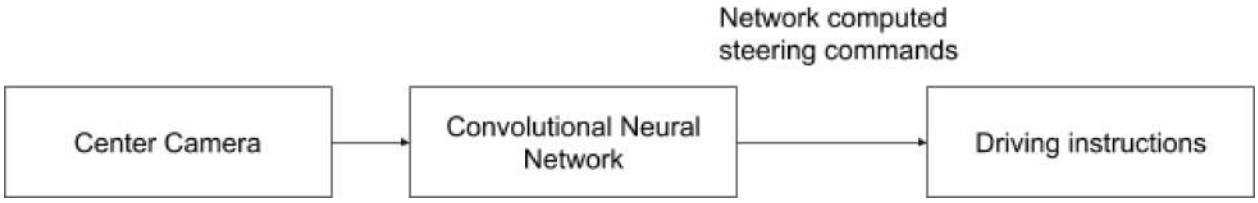

Figure 2.2.3: Decision making Process

2.3 Related Works

Autonomous cars are of state of the art technology for which in recent times it has become possible to implement through the vast expansion of Artificial Intelligence and Deep Learning. It understands the world by analyzing the context of a scene and focuses on the important objects than seeing the scene as a whole. With the development of computer programs and softwares that can read complex scene of objects to find meaningful information is a tremendous gift to the world. Many researchers have contributed to the field of autonomous vehicles and with the integration of deep learning it has become more real.

YOLO (You Only Look Once) is a real-time CNN methods that aims to detect objects from images and Road Lane Detector is used to detect road track from video's frames and to provide additional information that can be helpful for the decision-making process of the self-driving car. Shun-Feng Su and his team developed a simulator using YOLO as the object detector and polynomial regression as the road guidance in the real-world driving video simulations. The result shows a matching pair between those two methods for self-driving car environment and road lane guidance. [9]

In 2014, U. Lee, S. Yoon, H. Shim, P. Vasseur and C. Demonceaux introduced a local path planning algorithm for the self-driving car in a complex environment. The algorithm proposed by them consisted of three parts - the novel path representation, the collision detection and the path modification using a Voronoi Cell. Collision checking, modification of path and providing

continuous control input for steering wheel was made more convenient by the novel path representation than way point navigation. They applied the algorithm to the self-driving car, EureCar(KAIST) and validated its applicability and feasibility of real time use. [10]

In 2017, Zhilu Chen, Xinming Huang worked on self driving car which focused on lane keeping on self driving cars. Their paper presented an end-to-end learning approach to obtain the proper steering angle to maintain the car in the lane. The developed convolutional neural network (CNN) model takes in raw image frames as input and outputs the steering angles accordingly. The model is trained and evaluated using the comma.ai dataset, which contains the front view image frames and the steering angle data captured when driving on the road. [11]

Chapter 3

PROPOSED MODEL

3.1 Algorithm

The purpose of this thesis is to propose an enhanced algorithm for self driving car to clone behaviors of a good driver efficiently and accurately. The driving efficiency of the algorithm to drive the car will depend on the driver's skillfulness, but our main job is to clone the driver's behavior as accurately as possible.

The core approach that we are following is based on modern deep learning using multi-dimensional convolutional neural network. We will use a neural network architecture created by the NVIDIA Autonomous group as a base standard to compare our CNN model with, on several criterias such as accuracy of behavior cloning, error rate based, based on incremental amount of datasets from small to large.

Our convolutional neural network starts with (processed) images from the camera on a car, 3 sets of it, from front, left and right, then goes through our custom designed hidden layer, created for enhanced performance, and corresponds them to the output layer consisting of steering instructions.

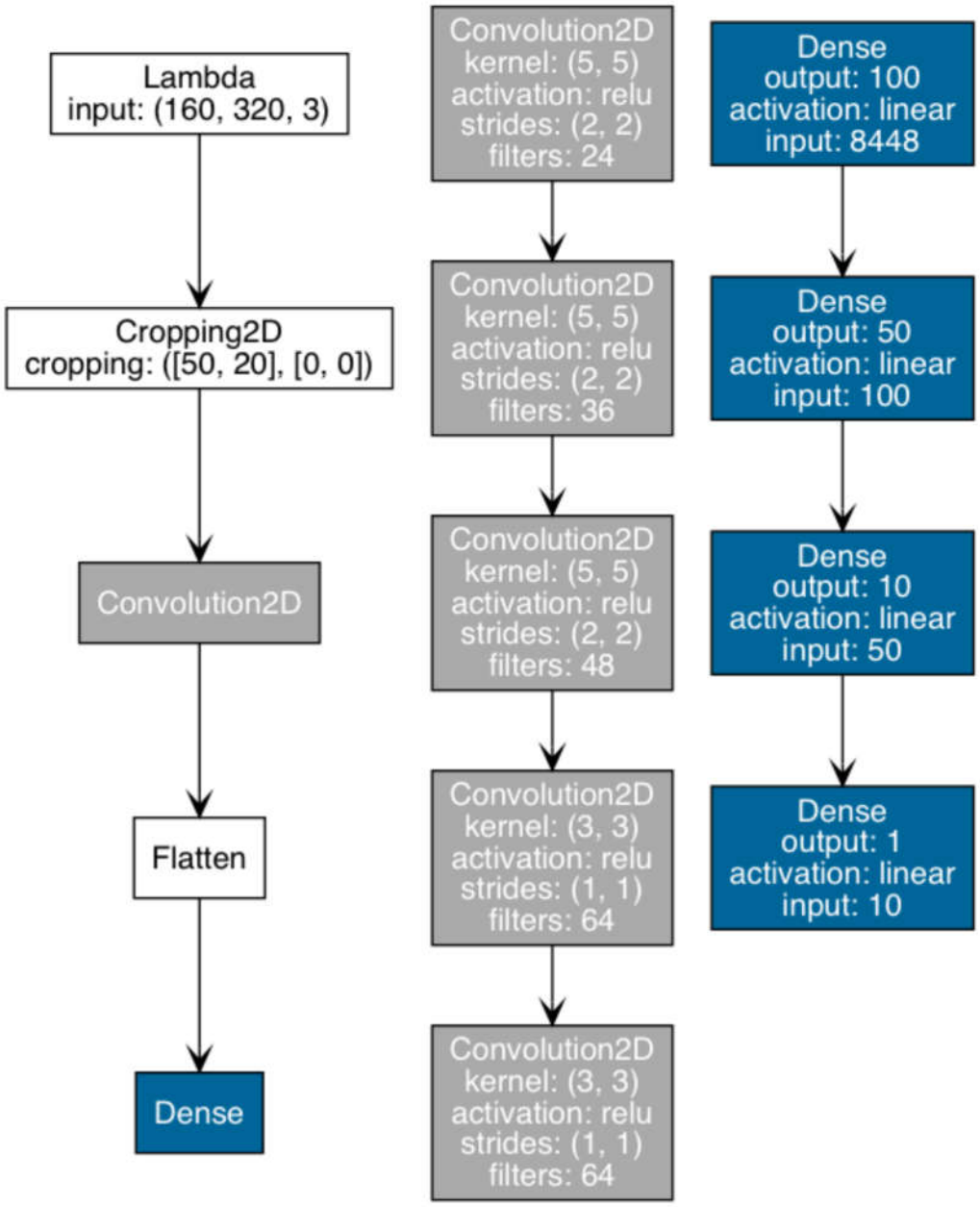

Figure 3.1.1: Proposed Model by the NVIDIA Group.

The core part of our algorithm lays in the hidden layers of the convolutional neural network that we designed. The dataset, from the input layer flows as below:

I. A two dimensional convolutional layer to extract features from the images

II. A layer for max pool output of the previous layer to pass through the high values

III. A two dimensional convolutional layer to make the features more evident using the weight value

IV. A layer for max pool output of the previous layer to pass through the high values again for

V. Then we perform densing consecutively for 120, 84 and 1 unit sequentially

A diagram of the optimized neural network is given below:

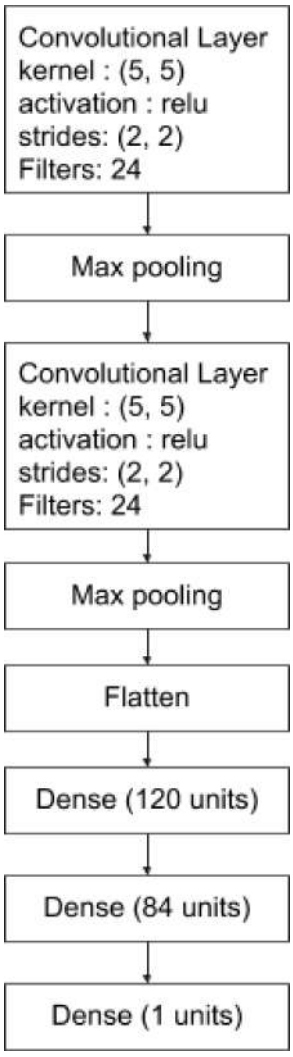

Figure 3.1.2: Optimized Neural Network

3.1.1 Max Pooling

Max pooling is a popular optimization technique in Convolutional Neural Network. It can be counted as *sample-based discretization process*. This process down-samples an input layer, that reduces its dimension and allow for assumptions to be made out of it. Although it makes the creation of the neural network model faster, but it is with the cost of accuracy.

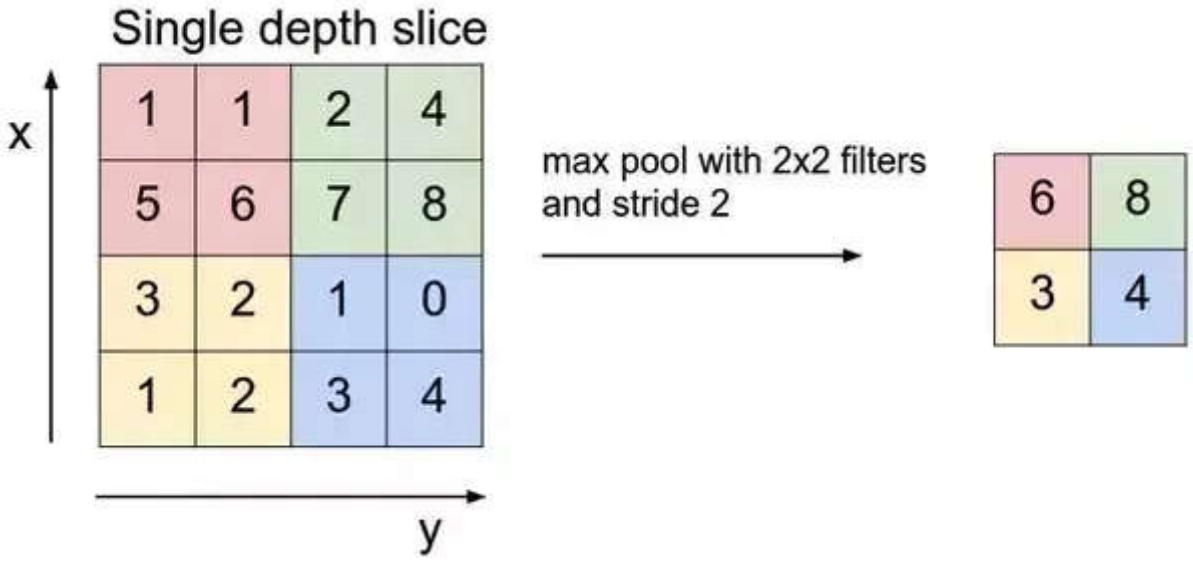

Figure 3.1.1.1: Demonstration of Max Pooling Process

3.2 System Implementation

Our system implementation includes two main parts:

I. Simulator

II. Training script

3.2.1 Simulation Environment

We have used Udacity's open source self driving car simulator to take advantage of a well built environment to test our algorithm . The simulator has 2 modes: Training mode and Autonomous

mode. The whole process consists of three steps, they are: data generation, training, and testing in autonomous mode.

3.2.1.1 Data Generation

Our data are generated from the simulator environment based on three virtual cameras held on center, left and right of the car. The steering angles and speed data are also recorded in the session which is then exported as a CSV file.

3.2.2 Training

The second step in the simulation process is training the environment using the generated data by our custom training script written in python, that implements our algorithm. The training phase outputs the generated model in a binary file.

3.2.3 Testing in Autonomous Mode

The last step of the simulation process is testing the car with the help of testing script, using our generated model.

3.2.4 Training Script

Our training script is written on python, and uses the generated data in the csv file, to export it as a Convolutional neural network model in form of model.h5 binary. The training script uses keras, a library written to build CNN models, to abstract away different layers of the model. The pseudo code for significant part of the training script is given below:

```
def build_model(args):
    model = Sequential()
    model.add(Convolution2D(6,5,5,activation='relu'))
    model.add(MaxPooling2D())
    model.add(Convolution2D(6,5,5,activation='relu'))
    model.add(MaxPooling2D())
    model.add(Flatten())
    model.add(Dense(120))
    model.add(Dense(84))
    model.add(Dense(1))
    model.summary()

    return model
```

Figure 3.2.2.1: Pseudocode of Training Script

There are two files the training script called model.py and testing script called drive.py. Therefore, after the data has been generated, our target is to build a model for the convolutional network that will read the generated data and will give the output steering command. Given below a list of items we will use for writing the training script along with their functions.

1. **Pandas**: This will be our data analysis toolkits.
2. **NumPy:** This will help us to do Matrix mathematics.
3. **Scikit-learn:** This will be used to split our training and testing data into sets.
4. **Keras:** This will be used as our machine learning model. It is going to be sequential as there will be linear stack of layers and then add them for gradient descent
5. **Model Checkpoints:** To save our model.
6. **Layer Types:**
7. **Helper Class Utils:** It's going to define our input shape and generate training images.
8. **Arc Parts:** Command line arguments.
9. **OS Modules:** For reading files.

The CSV file containing the driving data is loaded using Pandas into a single dataframe variable. Dataframe variables are easily parse able. Then we are going to take an input variable lets say x and store the three sets of images from the left, right and center. And another variable y to save the output data which is the steering command so we can concatenate that into one variable. Finally we have our input data and our output labels. Now the goal is to find the mapping between the two. his is a supervised learning problem and once we find the mapping between the two then given novel input data which is novel camera images of what a car sees. We can then output the predicted label which will be the steering command.

Now that we have our data we will split it into training and testing data. 80% of the data will be allocated to training and the rest 20% into testing. This can be done in the command line argument by multiplying the test size by 0.2. Finally we have our training data, our testing data and our validation data and we return it back. After that we are build our model.

Chapter 4

EXPERIMENTAL SETUP

This section describes the experimental setup of the components that are required to construct the simulator. Firstly, we describe the software components that are required to create the model for the simulator. Secondly, we explain the process of creating the 3D model of Hatirjheel and also the simulator with Unity Game Engine. Then later we discuss the training steps of the self driving car and how the car runs autonomously in the simulator.

4.1 Unity Game Engine

Unity is a cross-platform game engine developed by Unity Technologies in Denmark which is primarily used to develop both three-dimensional and two-dimensional video games and simulations for computers, consoles, and mobile devices. It has the following features

1. It is a multipurpose game engine that supports 2D and 3D graphics.
2. The users are able to make 2D and 3D video games (in our case, in the form of simulator) and it can work in any platform including iOS, Android, Tizen, Windows, Universal Windows Platform, Mac, Linux, WebGL, PlayStation 4, PlayStation Vita, Xbox One, Wii U, 3DS, Oculus Rift, Google Cardboard, Steam VR, PlayStation VR, Gear VR, Windows Mixed Reality, Daydream, Android TV, Samsung Smart TV, tvOS, Nintendo Switch, Fire OS, Facebook Gameroom, Apple ARKit, Google ARCore, and Vuforia.
3. Unity's editor has an easy drag and drop functionality when compared to other game engines like Unreal, and scripting using C# Language, BOO a programming language in Unity 5 and JavaScript got introduced in August 2017 with the release of Unity 2017.

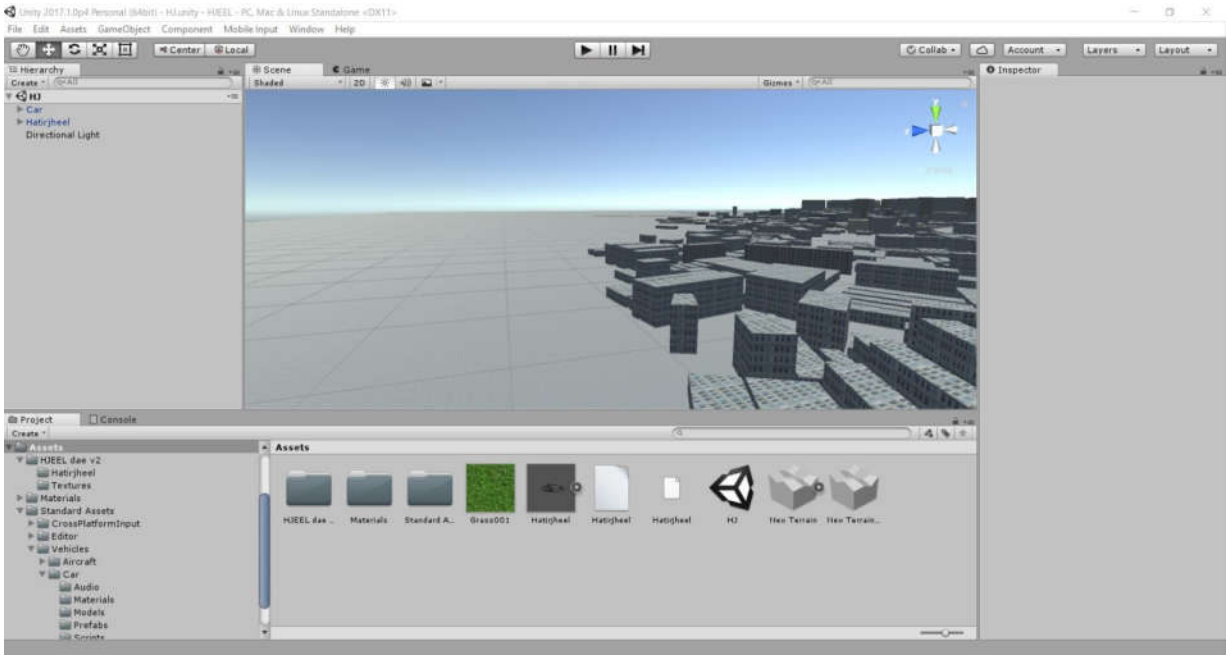

Figure 4.1: Unity 3D Game Engine

4.2 Sketchup

SketchUp which was formerly known as Google Sketchup, is a 3D modeling computer program for a wide range of drawing applications such as architectural, interior design, landscape architecture, civil and mechanical engineering, film and video game design. It is available as a freeware version, SketchUp Make, and a paid version with additional functionality, SketchUp Pro.

SketchUp is owned by Trimble Inc. a mapping, surveying and navigation equipment company. There is an online library of free model assemblies (e.g. windows, doors, automobiles), 3D Warehouse, to which users may contribute models. The program includes drawing layout functionality, allows surface rendering in variable "styles", supports third-party "plug-in" programs hosted on a site called Extension Warehouse to provide other capabilities (e.g. near photo-realistic rendering) and enables placement of its models within Google Earth.

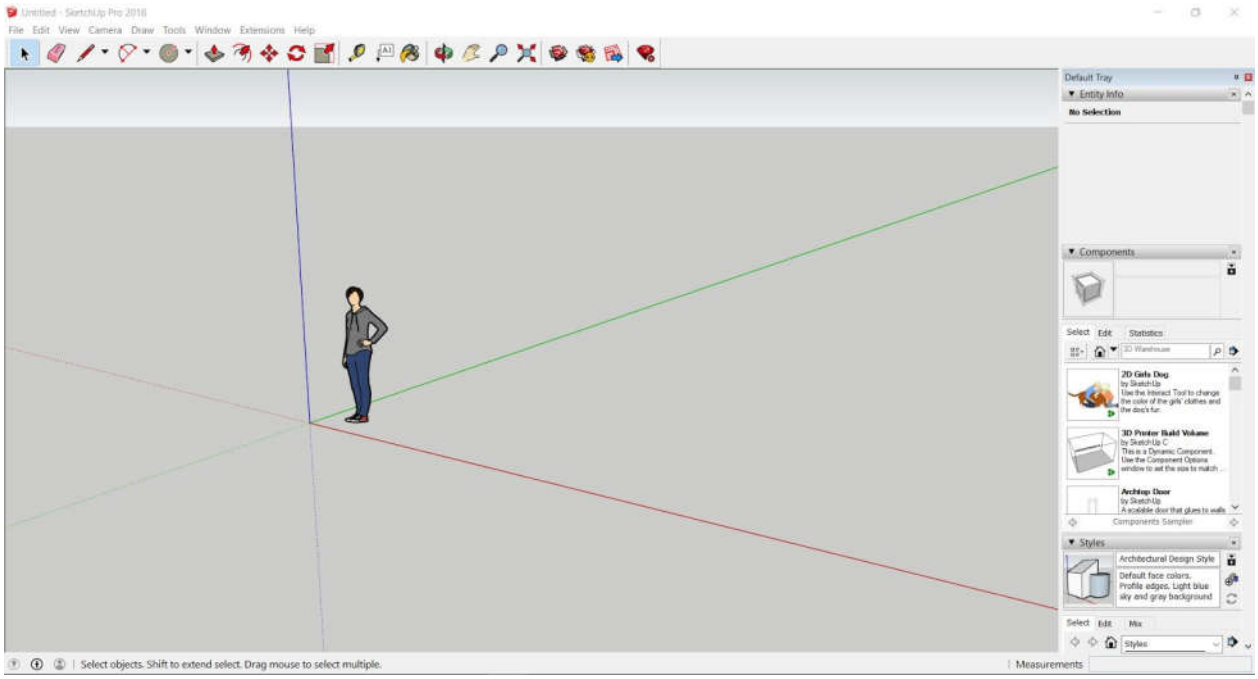

Figure 4.2: Sketchup Software

4.3 Making the 3D Model of Hatirjheel

The task was challenging not only to develop the algorithm but also to make the 3D model of a location that puts the car at ease during movement furthermore, the road needs to be wide enough. We decided to develop the 3D model of Hatirjheel as it fits to a suitable and urban environment for the car to be in free movement.

The model was developed with the help of Sketchup Software with reference to the design that is found on Google map. It is to be mentioned that the model does not replicate all architectural designs with the real life scenes that we see in Hatirjheel. We made effort to make the model as real as possible.

Figure 4.3.1: Hatirjheel seen on Google Map

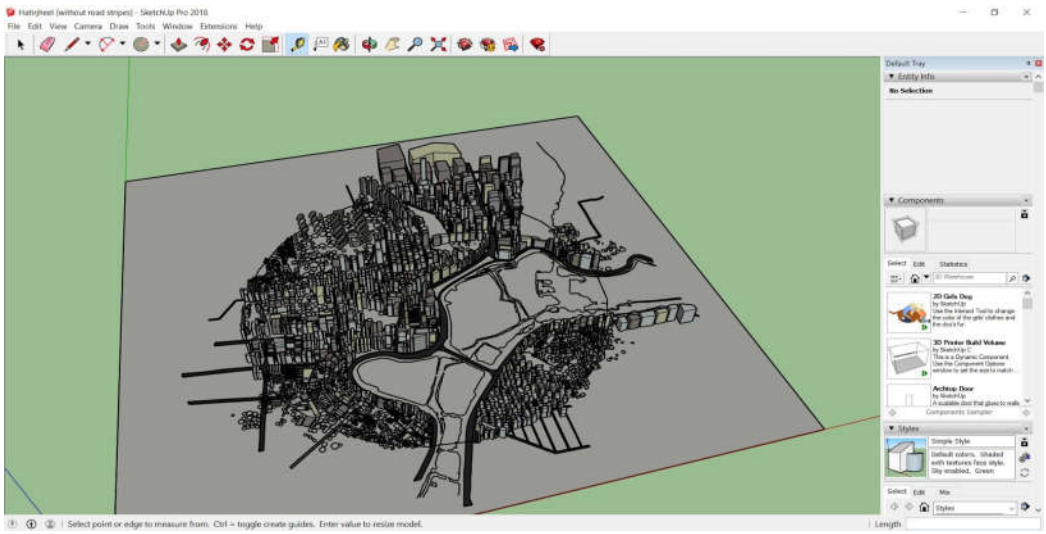

Figure 4.3.2: Hatirjheel model with Sketchup

We have covered the area ranging from Rampura, Gulshan and a part of Mohakhali. The model covers to be estimated 6.45 square kilometer. The road have an average thickness of about 23 meter. The model was exported to a 3d model format known as FBX (Filmbox) format in which we import it to the Unity Game Engine to complete the process of creating the simulator.

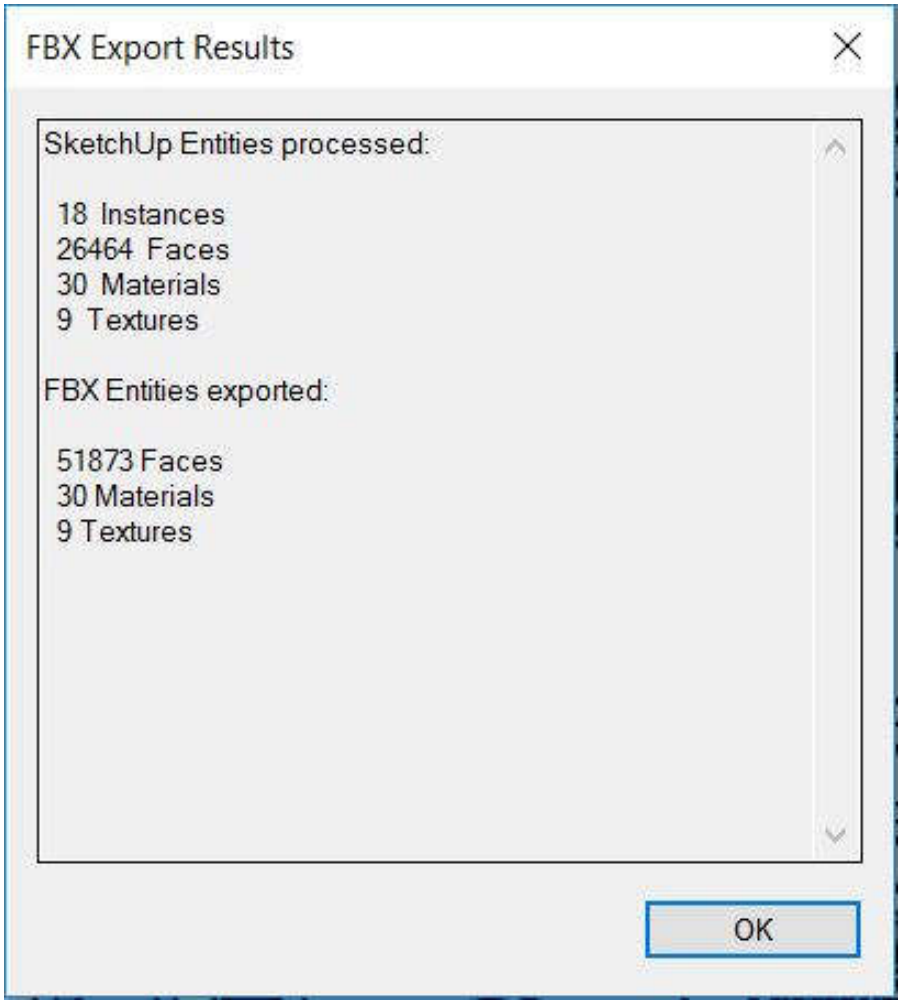

Figure 4.3.3: FBX Export Report of Hatirjheel 3D Model

4.4 Making the Simulator

Creating and designing the simulator is one of the vital part and has been given utmost importance. The simulator acts as an open world with which the car will react and response. Real world simulation is costly and the availability of the components are also scarce.

Udacity's open source self driving car simulator was built with Unity. It is to note that Unity Editor is a game development engine. The simulator comes with a number of pre-built scripts which allows us to simulate real world environmental features such as gravity, momentum and acceleration, river etc. These parameters also can be modified just by changing them in the code.

The built in simulator from Udacity comes with 2 modes namely Training Mode and Autonomous Mode. These two modes can be considered as two different simulators in which they differ by the internal features of the car.

It is to again notify that our working falls under supervised learning which means the system learns through datasets and we need to provide the datasets to the algorithm. In our project we provided sufficient amount of data that we applied to our algorithm so that the car could accomplish to drive by itself. And these datas are image datas from left camera, right camera and center camera, speed and steering angles of the car that are being recorded while driving in the simulator. These dataset we term it as training data and it is collected in the Training Mode feature of the simulator. We are not applying any algorithm yet, we are just collecting data.
The car that is featured in the Training Mode has three cameras attached to it.

1) Left Camera
2) Center Camera
3) Right Camera

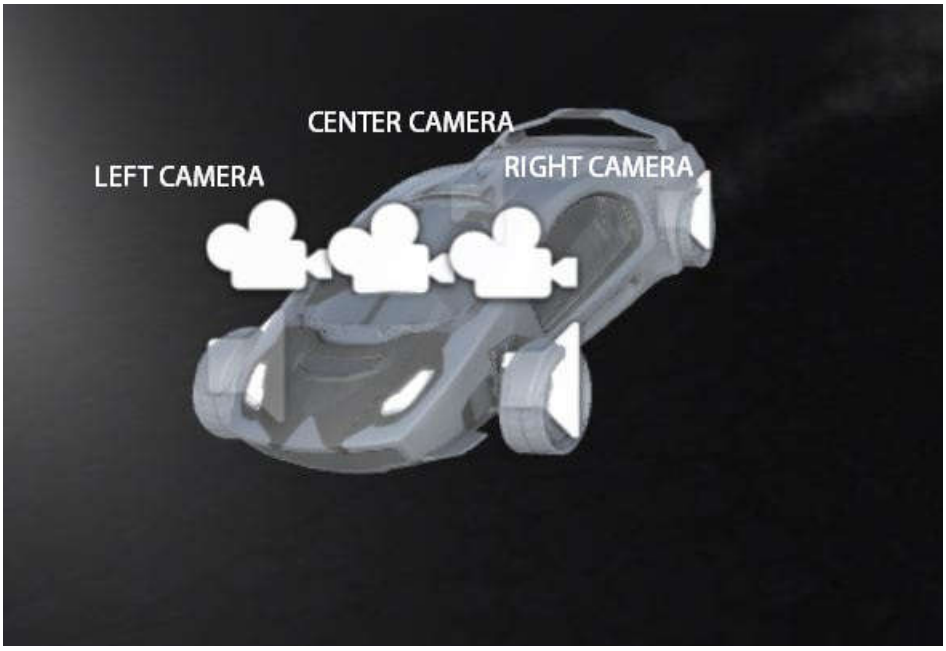

Figure 4.4.1: The Self Driving Car

In training mode, there is also a record feature when turned on captures frames from these three cameras of the car instantaneously. It also records the speed and steering angle at that instant. That means when the car is driven by a person in the simulator the image data, steering angles and speed are generated by the second. The image data is being saved to a folder and also a CSV file is generated that holds the record of speed and steering angles and also the location of the directory to where that image is saved.

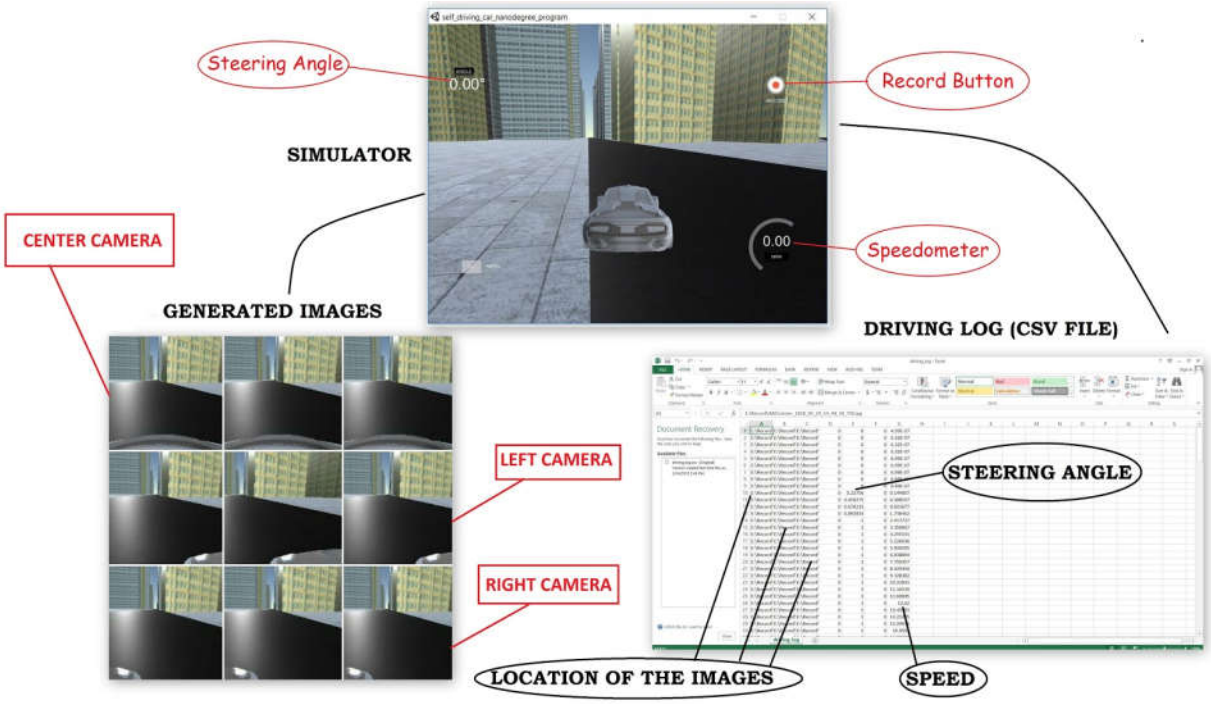

Figure 4.4.2: Functionality of Simulator in Training Mode

In autonomous mode, the car will not use the 3 cameras that is assembled to it. The training data that we collected from the simulator in training mode, we will apply our algorithm with these training data as input and after application of algorithm providing the resulting data as input to the car, it drives by itself. It means the car will use its previous experience when we were driving the car to collect training data as reference to take control of the car. This operation is done in the Autonomous Mode. We do not have the control to drive here. The car will drive by itself.

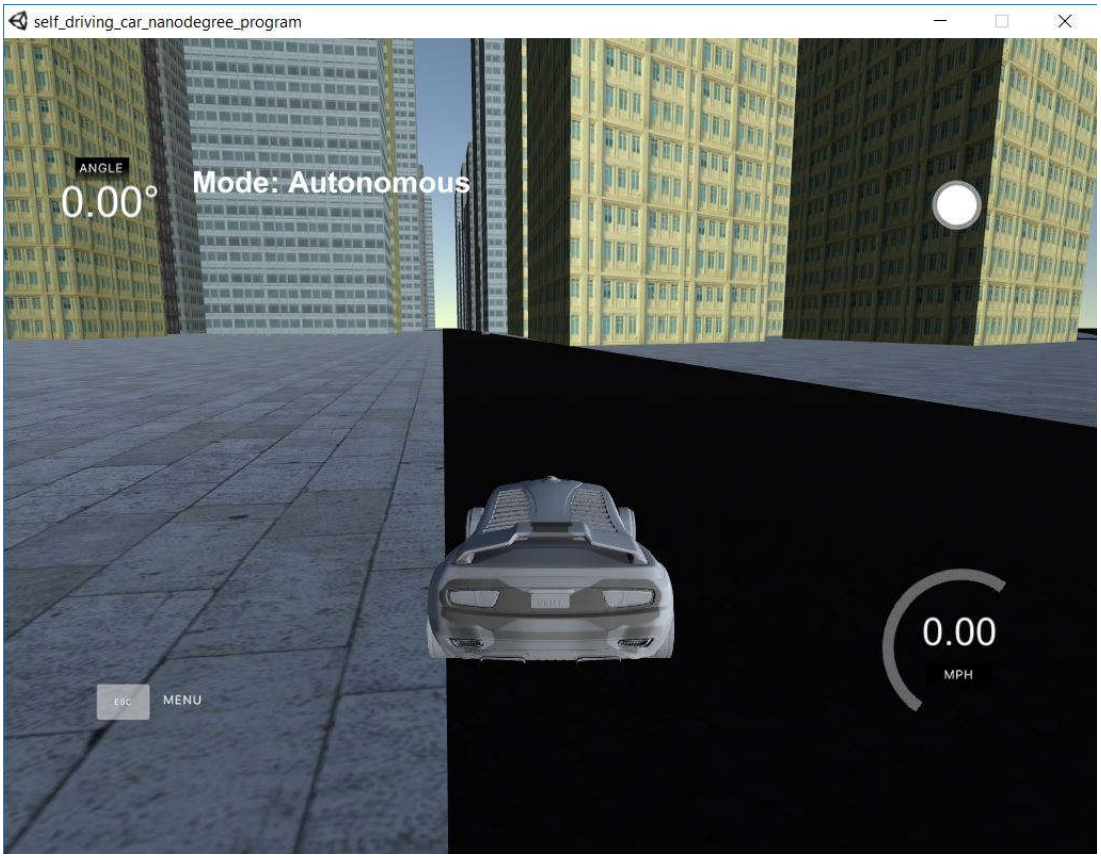

Figure 4.4.3: Autonomous Mode

One big advantage of the Udacity simulator is that it allows to create our own environment and we do this in Unity. The previously exported Hatirjheel fbx format 3D model from Sketchup is imported to the simulator and mounted on a terrain that we created. The imported Hatirjheel model was scaled to ten times to widen the roads so that we could add as many as car possible to fit in the road. We added textures to the model as in buildings, roads and also added river using the pre-built Unity Water Asset.

Figure 4.4.4: Hatirjheel Model in Unity Game Engine

4.5 Implementation of Traffic

For designing the traffic, we used a waypoint-based system which is quite simple to implement. To start off, we had to mark the entire map with blocks to define the route to be taken by the aforementioned traffic vehicles that we want to insert in our map. There is no limit here, which means, the greater the number of blocks, the greater the accuracy of the traffic vehicle hitting the block and activating the trigger. And we would want to activate the trigger since only on activation of the trigger the next waypoint is targeted on the system. This leads the vehicle(traffic) to travel the entire loop just by hitting and activating the triggers.

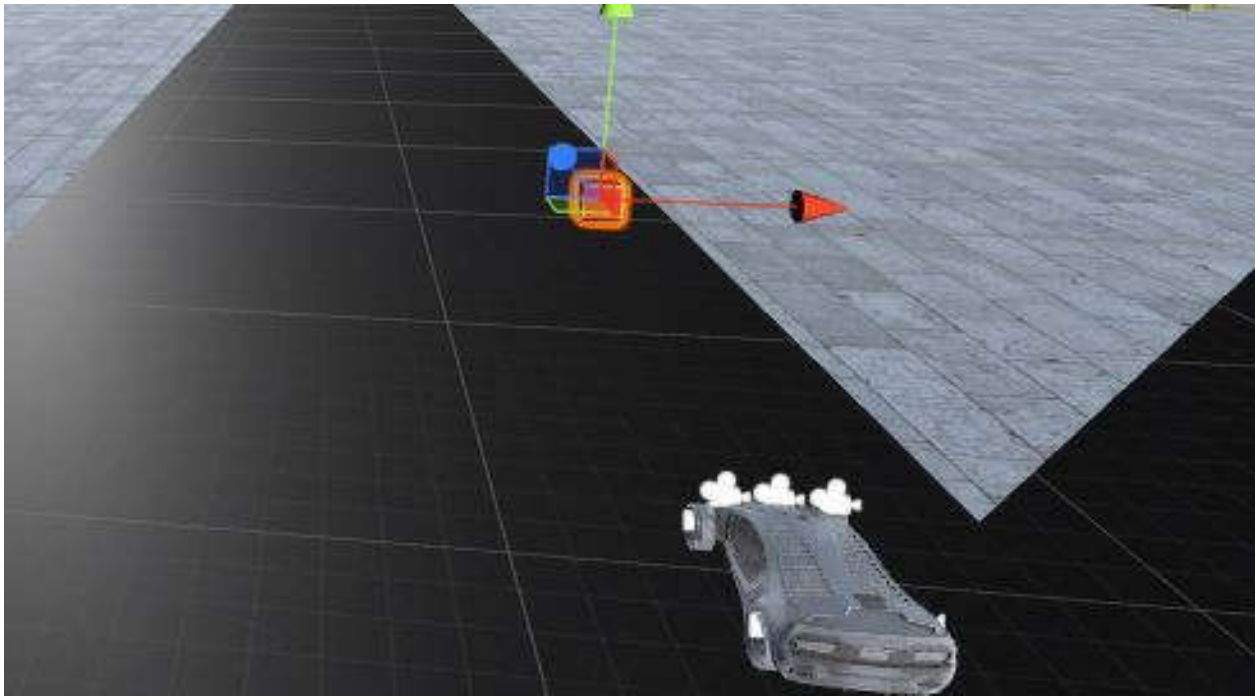

Figure 4.5.1: Creating the Waypoint Based System

In the figure above the vehicle is targeting the red cube, which is a waypoint in our case. As soon as it touches the cube, trigger that is set inside the cube is activated, which leads to the trigger setting automatically on to the next waypoint,shown in the figure below.

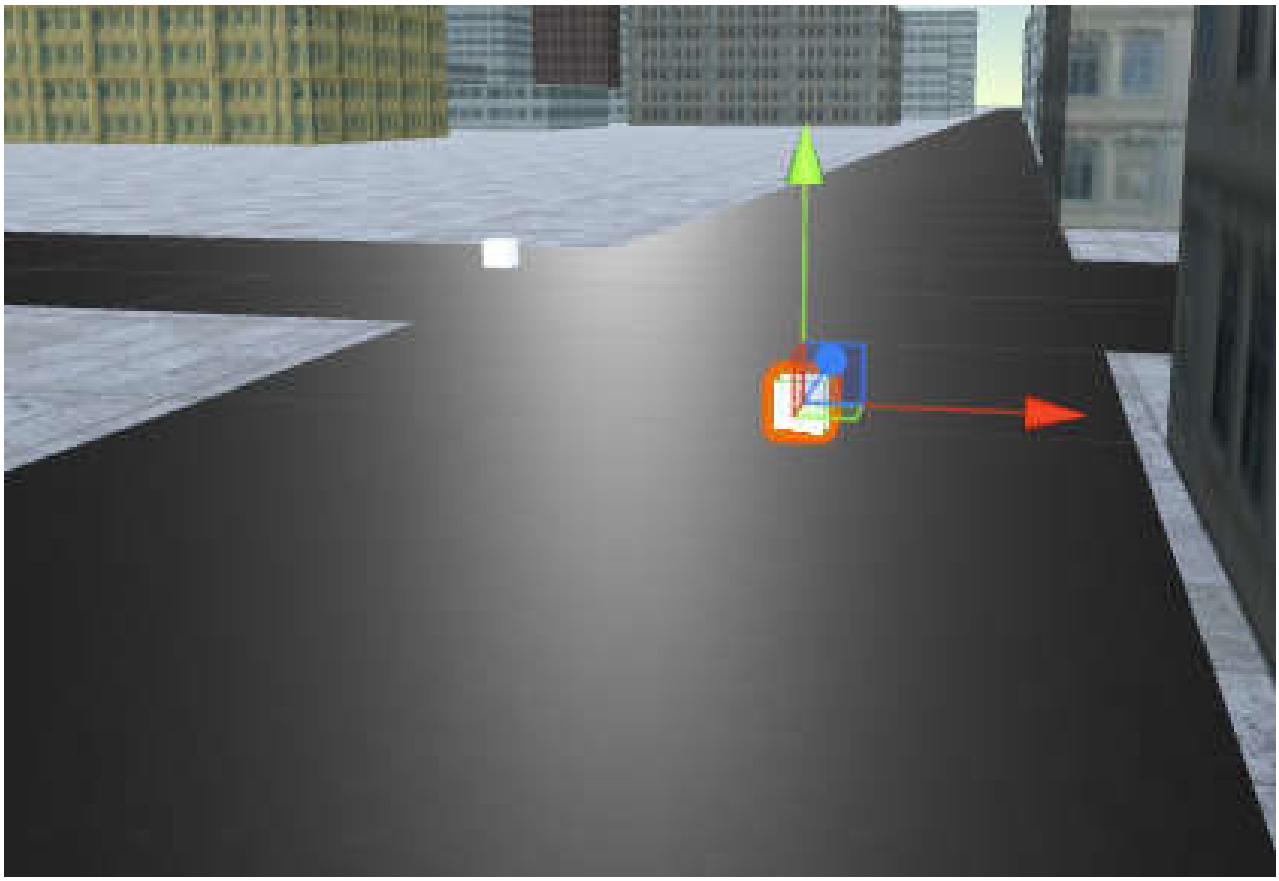

Figure 4.5.2: Waypoints

The white cube shown here is the next waypoint. The same process is repeated for the entire map for different traffic. But in the actual model, the blocks cannot be seen since even though we now understand how the traffic is moving throughout the city, we would not really want to see the actual blocks being present in the middle of the road. So for this the solution is quite simple thanks to Unity. We just had to uncheck “Mesh Renderer” named component, which leads to the blocks being present there, but we just won’t be able to see it when we are moving around in the roads of the city.

For the software aspect, the input r, for the steering wheel is taken as 1/r as it was found that the input taken like this helped the car to make smoother turns. Furthermore the autonomous angles in which the car moved independent of that the car moved independent of the geometry of the car.

The input data are the images from the three sets of cameras and the output label which would be the one binary value where all the steering angle data are extracted into. Basically, this means on what angle should the steering be rotated based on what we see through the cameras. All those inputs are listed on a CSV.

During the training process 72 hours of data generated by a human driver is produced, then it is possible to compare the error or loss between this and the autonomous mode. So the steering angle values and images from both the human driver generated data and autonomous data are taken. Both of those values are vectorized into one value. Then the difference of the vectorized values are found and that is the error existing between the two values. Then back propagation is used to update the weights based on the error value.

The images that are fed into the CNN from all the cameras as shown in the figure above. The CNN then computes a proposed steering command. The proposed command by the CNN is the compared to the desired command for that image. After comparing adjustments are made in order to minimize the difference between the CNN output and human generated desired output. And finally the adjustments are established through back propagations.

4.6 Training

During training mode, the simulator can be taken to be a server and the client is going to be the python program and the whole thing follows a server client architecture. This process can also be considered as a feedback loop. The client is piping in steering angles to the server and the server is piping back images from the car and steering angle so that it is properly trained. The steps in order to accomplish the training mode is explained below:

Step 1: Install all the dependencies

First of all we need to install all the dependencies. With the help of anaconda it can be easily done with just a line of code. From the environments.yml file all the 20 dependencies that are required, can be extracted. Then the conda environment is activated using another line of code. Basically, this means that the all the dependencies are stored in an anaconda container and the second line of code is used to activate it. After accomplishing this we can proceed to data generation.

Step 2: Generate data.

In order to generate data we drove our virtual vehicle on a custom created map of Hatirjheel in the Unity platform. When Unity is opened there will be two options training and autonomous. Initially, we would select training mode. By using w, a ,s and d keys of the keyboard as front, left, back and right control respectively we will drive the car. Thereby, we are able to generate the required human driving data from our virtual car by manually driving it Unity. The data is recorded by clicking the record button on the screen or by pressing R on the keyboard and then selecting a suitable location to save it. As we start recording the three cameras on the car keeps on capturing images frame by frame and also the four physics variables are also recorded. Ideally 5 laps should be completed. After we are finished with the 5 laps the recording should be terminated by pressing R again and when this is done the data is captured by replaying the whole thing and the input data is saved in a CSV file.

4.7 Testing

Measuring performance of the self driving car is done by letting the car undergo various tests using our driving scripts. The car will only be able to drive without any assistance if we provide enough data to it and we do it by training the car, meaning driving the car manually for a significantly long time as it picks up images from the cameras, and other components through sensors such as, picking up speed and steering angles. We selected a path of our choice closed around a loop as. The track we chose is approximately 1.1891 km in length, if we consider our average speed to be 45.0616 km/h. to be We drive the car around the track without any interruption for 1 to 24 laps, using both the CNN models. As it completes each lap we record the time. Since it is a supervised learning, it happens slower and it takes a great deal of time even for the car to learn to be on track. We also record instances where the car goes off track for each session.

4.7.1 Testing Procedure

For the testing procedure, we generate the data, and train the data with two Convolutional Neural Network models. The first one is based off of NVIDIA's Neural network model. Then, using our model that optimizes the training time. We started the training session and eventually generated the Convolutional Neural Network (CNN) model. After the training session was completed, we ran the car in autonomous mode for 24 laps that we did for training. Finally, we compared the results obtained from both the training and the autonomous session.

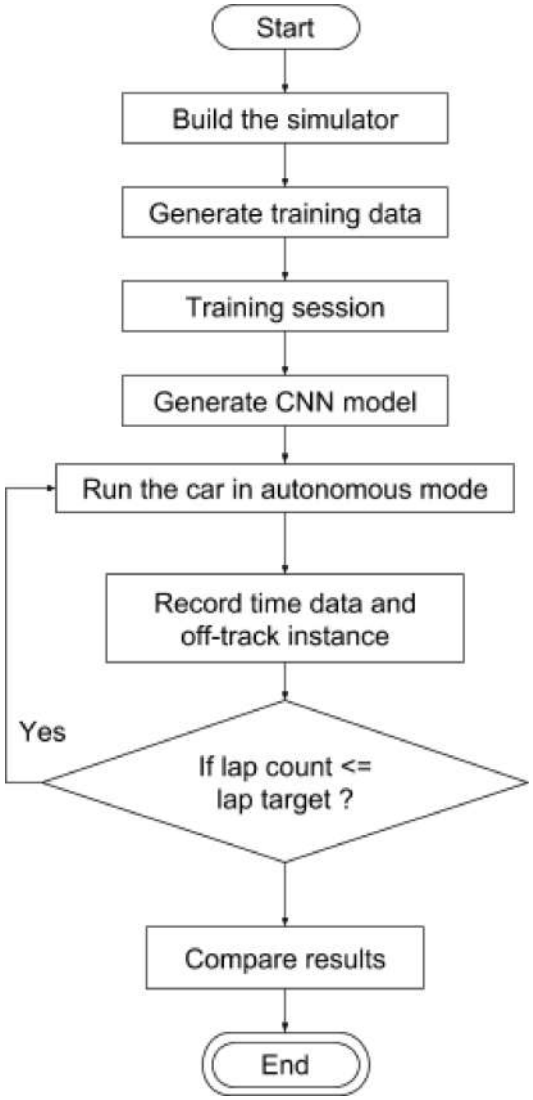

Figure 4.7.1: Training Process Flowchart

Chapter 5

EXPERIMENTAL RESULTS AND DISCUSSION

5.1 Training Time Comparison

Previously we tested our car by going through 24 laps and we gathered training datas.
Initially we generate data by driving the car upto 1 lap and we use our optimized algorithm in which we used 9 layers of convolutional layer to train our car and record the time that it takes to complete the training. We do the same with NVIDIA's model algorithm to train the same dataset of 1 lap and record the time that it completes training.

We repeat the same procedure again that we generate data by driving the car again from scratch upto 2 laps and use our optimized algorithm to train our car and recorded the time and also do the same with NVIDIA's model algorithm.
We keep repeating this procedure for 3 laps, 4 laps and continue upto 24 laps that means at 24th time or iteration we let our car to drive for 24 laps, we generate data and use this generated data to train with our optimized algorithm and NVIDIA's algorithm and record both of their duration respectively.

Comparison was done with respect to the duration of the training time that both of the algorithm takes to complete from 1 lap to 24 laps.

Lap count (units)	NVIDIA (hours)	Optimized (hours)	Training Time (NVIDIA) (hours: minutes)	Time(Optimized) (hours: minutes)	Difference in Training time (hours)
1	0.2333	0.1833	0:14	0:11	0.05
2	0.433	0.3833	0:26	0:23	0.0497
3	0.7666	0.5177	0:46	0:31	0.2489
4	1.0833	0.82	1:05	0:42	0.2633
5	1.266	0.8333	1:16	0:50	0.4327
6	1.5833	0.95	1:35	0:57	0.6333
7	1.7833	1.2	1:47	1:04	0.5833
8	2.05	1.366	2:03	1:16	0.684
9	2.25	1.5833	2:15	1:22	0.6667
10	2.566	1.8	2:34	1:33	0.766
11	2.85	1.95	2:51	1:45	0.9
12	3.2667	2.1167	3:16	2:07	1.15
13	3.45	2.266	3:27	2:18	1.184
14	3.7166	2.44	3:43	2:29	1.2766
15	4.0166	2.5833	4:01	2:39	1.4333
16	4.266	2.7	4:16	2:50	1.566
17	4.55	3	4:30	3:01	1.55
18	4.7667	3.3167	4:46	3:19	1.45
19	5.05	3.45	5:03	3:30	1.6
20	5.2833	3.7	5:17	3:43	1.5833
21	5.583	3.9333	5:35	3:55	1.6497
22	5.883	4.08333	5:53	4:06	1.79967
23	6.166	4.3333	6:10	4:19	1.8327
24	6.85	4.71	6:51	4:43	2.14

Table 5.1: Recorded Training Time for NVIDIA's Model and Optimized Model

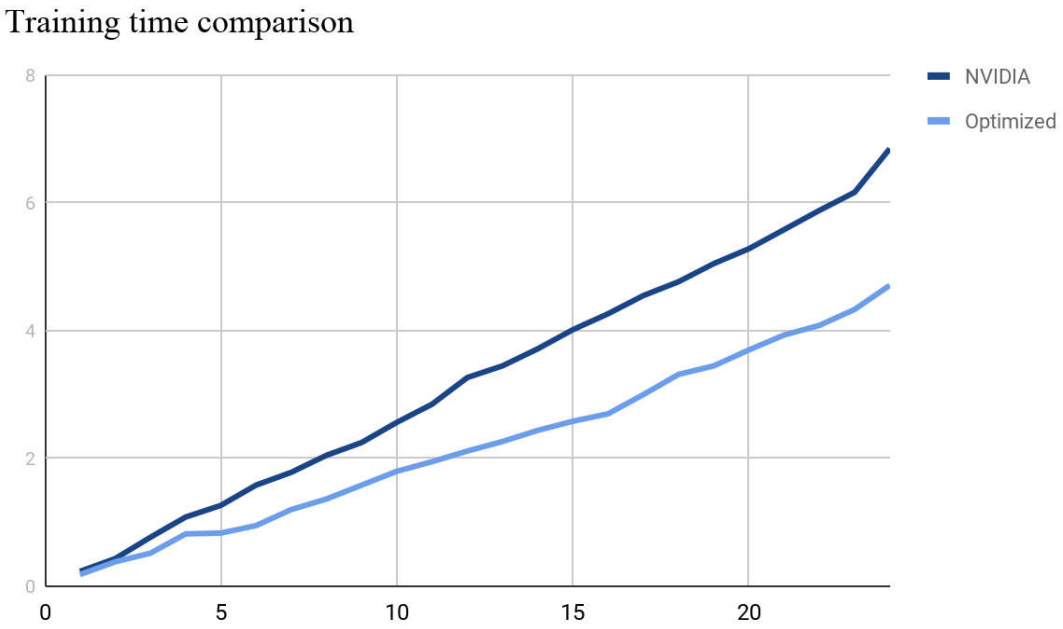

Figure 5.1.1: Training Time Comparison

In the graph the X axis represents the laps from 1 to 25 and the Y axis represents the duration in hour. The graph shows the duration of NVIDIA (represented by the navy blue graph) and our optimized algorithm (represented by the sky blue graph) that it takes to complete the training from lap 1 to 24. We observe that there is a deviation between the two lines and we observe that our optimized algorithm completes the training faster than NVIDIA's model algorithm. However, the performance will drastically fall as our model will perform badly with compared to NVIDIA's model. We are sacrificing performance for more faster training.

Following figure shows the difference between training duration of NVIDIA's algorithm and our optimized algorithm for each lap ranging from 1 to 24.

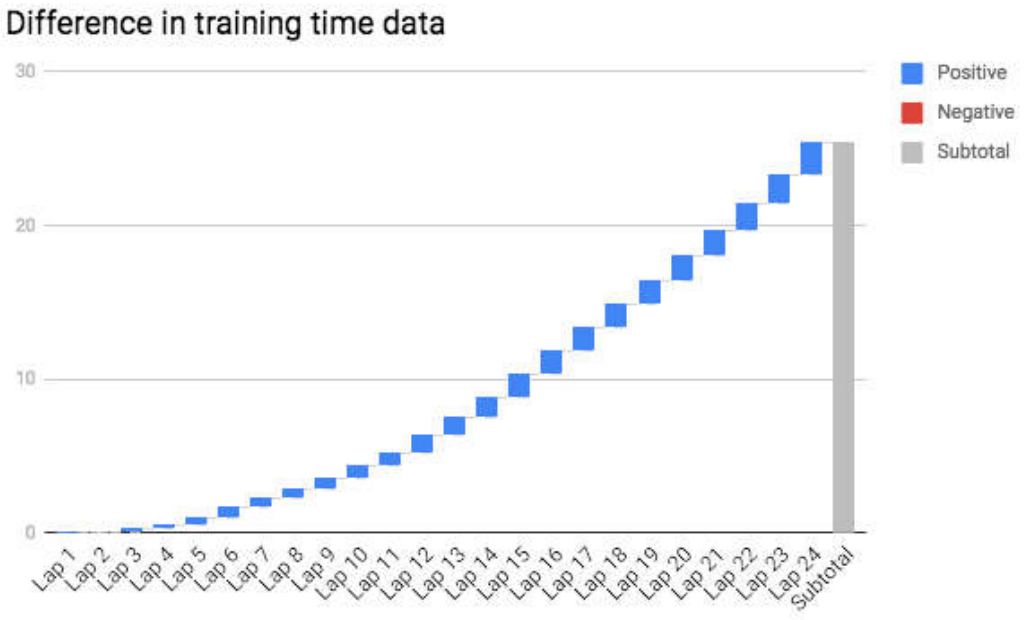

Figure 5.1.2: Difference in Training Time Data

We observe that in lap 1 the time difference seems to be negligible to an extent that the training may complete at the same time. However, as the lap increases to 3 and 4 a gap of time difference increases which approximates 24 minutes time difference. As we go beyond 20 laps the time difference increases to approximately about 90 minutes meaning that our algorithm will complete 90 minutes before the NVIDIA's model. In short, the difference between the time increases with the increase of the number of laps This shows that we are able to save a lot of time for the training process.

5.2 Instances of Going Off-track

As we mentioned earlier, we are sacrificing performance for faster training. The training does take a short time for our algorithm to complete but the performance of the car decreases by that we mean that when the car goes to autonomous mode the car has a high possibility of going off-track and it may not follow the lane. In comparison with NVIDIA's model whose training duration may be more but it has a short possibility of going off track than our algorithm. The graph below shows for 1 lap training in our optimized algorithm goes off the track 36% more than the NVIDIA model. Moving forward when the lap number increases to more than 10 laps the percentage varies 53-56% more to go off-track than the NVIDIA model. Last but not the least if the lap increases beyond 20 laps then the percentage reduces to about 48-50%. This shows that response of accuracy on the amount of dataset. Smaller dataset leads to high performance of the algorithm whereas larger datasets leads to a less performance. We show the performance of the two algorithms below on the graph below to illustrate the point.

Lap Count (units)	Session 1 Off Track Counts (units)	Session 2 Off Track Counts (units)	Session 3 Off Track Counts (units)	Average Off Track Counts (units)
1	12	11	9	10.66666667
2	12	12	11	11.66666667
3	10	11	12	11
4	11	9	8	9.333333333
5	9	7	12	9.333333333
6	10	9	7	8.666666667
7	7	7	8	7.333333333
8	6	5	5	5.333333333
9	6	5	3	4.666666667
10	5	6	5	5.333333333
11	4	2	3	3
12	2	3	4	3
13	2	1	4	2.333333333
14	4	3	1	2.666666667
15	0	1	2	1
16	2	3	3	2.666666667
17	3	1	0	1.333333333
18	2	0	1	1
19	1	1	1	1
20	1	2	1	1.333333333
21	0	1	0	0.3333333333
22	2	1	0	1
23	0	0	0	0
24	0	1	0	0.3333333333

Table 5.2.1: Recorded Average Off Track Instances for NVIDIA's Model

Lap Count (units)	Session 1 Off Track Counts (units)	Session 2 Off Track Counts (units)	Session 3 Off Track Counts (units)	Average Off Track Counts (units)
1	17	18	15	16.66666667
2	16	13	12	13.66666667
3	18	16	15	16.33333333
4	12	13	15	13.33333333
5	15	14	17	15.33333333
6	13	17	11	13.66666667
7	11	10	8	9.666666667
8	9	7	12	9.333333333
9	7	13	11	10.33333333
10	9	5	12	8.666666667
11	10	7	11	9.333333333
12	8	7	8	7.666666667
13	6	5	5	5.333333333
14	4	6	7	5.666666667
15	5	4	3	4
16	5	4	5	4.666666667
17	6	9	8	7.666666667
18	3	0	1	1.333333333
19	2	1	3	2
20	3	2	2	2.333333333
21	2	1	0	1
22	1	0	0	0.3333333333
23	1	1	3	1.666666667
24	2	1	0	1

Table 5.2.2: Recorded Average Off Track Instances for Optimized Model

Lap Count (units)	Average Off Track Counts (NVIDIA) (units)	Average Off Track Counts (Optimized) (units)	Difference in Average Off Track Counts (units)
1	10.66666667	16.66666667	6
2	11.66666667	13.66666667	2
3	11	16.33333333	5.333333333
4	9.333333333	13.33333333	4
5	9.333333333	15.33333333	6
6	8.666666667	13.66666667	5
7	7.333333333	9.666666667	2.333333333
8	5.333333333	9.333333333	4
9	4.666666667	10.33333333	5.666666667
10	5.333333333	8.666666667	3.333333333
11	3	9.333333333	6.333333333
12	3	7.666666667	4.666666667
13	2.333333333	5.333333333	3
14	2.666666667	5.666666667	3
15	1	4	3
16	2.666666667	4.666666667	2
17	1.333333333	7.666666667	6.333333333
18	1	1.333333333	0.3333333333
19	1	2	1
20	1.333333333	2.333333333	1
21	0.3333333333	1	0.6666666667
22	1	0.3333333333	-0.6666666667
23	0	1.666666667	1.666666667
24	0.3333333333	1	0.6666666667

Table 5.2.3: Recorded Average Off Track Instances for NVIDIA's Model and Optimized Model and Their Differences

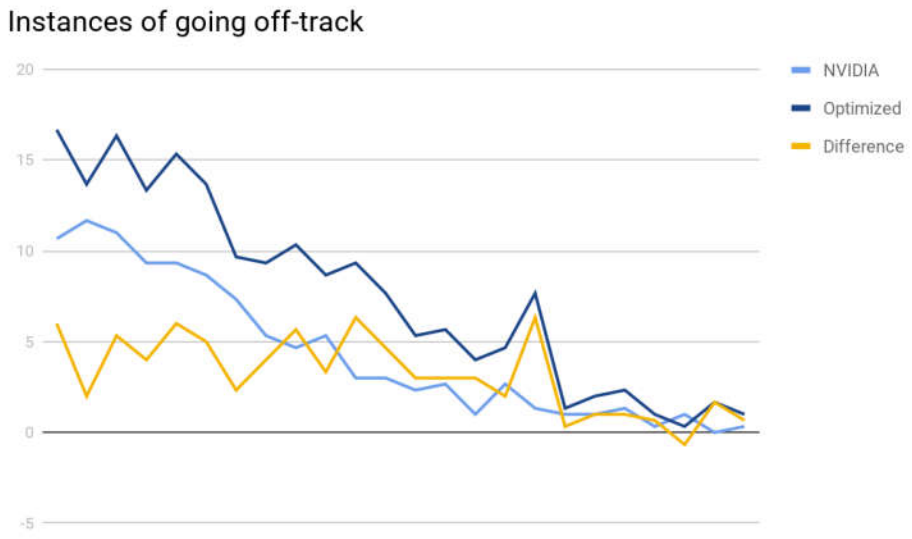

Figure 5.2.1: Instances of Going Off-track

5.3 Discussion of Results

Our extensive experimentation and result show two behaviors:

I. The optimized model reduces the training time for larger datasets significantly

II. Although it is less accurate for smaller dataset, the going-off-the-track instances are minimized in smaller datasets

Our observations also affirm that, by reducing the large number of convolutional layer, from 9 to two layers, and applying max pooling in alternate layers between them, we have reduced the processing time for training quite a bit. Our dataset shows that, from being 21.45% faster in small datasets, the efficiency of CNN based on altered max pooling layers increase to 31.24% when the dataset is larger.

This can be explained by:

I. The role of max pooling layer, which helps to reduce the feature map resolution.

II. The nature of the input images, which are high contrast street images, hence, max pooling does not lose much of the feature information from the dataset.

Chapter 6

CONCLUSION

6.1 Conclusion and Future Plan

Since, training is the most processor intensive task, our test was completely based on of comparing how different techniques of CNN layering affect training time. We also did our best to replicate a real urban environment, rather than a race track to emphasize on practicality of the analysis. The test data also shows, reducing convolutional layers, and sandwiching them between max pooling layers, may have less accuracy at first, but in the long run as we train more, it performs much faster and increases accuracy, with respect to conventional neural network used in NVIDIA's proposed model.

The potentiality for research and development of autonomous vehicles are unparalleled. Worldwide, throughout several countries a lot of researches have been conducted in this field. However, this is a relatively new topic in this part of the world, the sub-continental countries. Though, it is not an easy task, we plan to accomplish the feat of running a fully functioning autonomous vehicle on the streets of Dhaka in an organized environment like Hatirjheel, the location of our simulated model. We believe that, this will be a stepping stone towards our goal of digitalization of roads and transport system

References

[1] "*Phantom Auto will tour city*,"The Milwaukee Sentinel p. 1, Dec.8, 1926.

[2] W. W. Wierwille, L. A. Ellsworth, S. S. Wreggit, R. J. Fairbanks and C. L. Kirn, “Accident Scenarios” in *Research on vehicle-based driver status/performance monitoring; development, validation, and refinement of algorithms for detection of driver Drowsiness, Final Report,* United States, Joint Program Office for Intelligent Transportation Systems, 1994, ch. 1, pp. 1-23.
Available: https://rosap.ntl.bts.gov/view/dot/2578. [Accessed: 23- Mar- 2018].

[3] J. R. Aworemi, I. A. Abdul-Azeez, A. J. Oyedokun and J. O. Adewoye, "Efficacy of Drivers’ Fatigue on Road Accident in Selected Southwestern States of Nigeria", *International Business Research*, vol. 3, no. 3, pp. 225-232, July 2010
DOI: https://doi.org/10.5539/ibr.v3n3p225
http://www.ccsenet.org/journal/index.php/ibr/article/viewFile/4424/5109

[4] T. Khan and M. R. Islam, “Estimating Costs of Traffic Congestion in Dhaka City”, *International Journal of Engineering Science and Innovative Technology (IJESIT),* vol. 2, no. 3, pp. 281-289, May 2013
Available: http://www.ijesit.com/Volume%202/Issue%203/IJESIT201303_42.pdf

[5] E. Dickmanns, Dynamic vision for perception and control of motion. England: Springer London Ltd., 2010.pg.374

[6] D. Pomerleau: ALVINN: An Autonomous Land Vehicle in a Neural Network. In: Advances in Neural Information Processing Systems, Morgan Kaufmann, San Mateo, CA (1989).

[7] G. Obinata, T. Suzuki and T. Wada, "Tutorial I: Car robotics — Self-driving cars and human factor," *2012 Proceedings of SICE Annual Conference (SICE)*, Akita, Japan, 2012, pp. xiv-xiv.
Available:
http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6318316&isnumber=6318306

[8]Bojarski, M., Firner, B., Flepp, B., Jackel, L., Muller, U. and Zieba, K. (2018). End-to-End Deep Learning for Self-Driving Cars. [online] NVIDIA Developer Blog. Available at: https://devblogs.nvidia.com/deep-learning-self-driving-cars/

[9] B. Nugraha and S. Su, "Towards self-driving car using convolutional neural network and road lane detector - IEEE Conference Publication", Ieeexplore.ieee.org, 2018. [Online].
Available: http://ieeexplore.ieee.org/document/8253388/

[10] U. Lee, S. Yoon, H. Shim, P. Vasseur and C. Demonceaux, "Local path planning in a complex environment for self-driving car," *The 4th Annual IEEE International Conference on Cyber Technology in Automation, Control and Intelligent*, Hong Kong, 2014, pp. 445-450.
DOI: 10.1109/CYBER.2014.6917505
Available:
http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6917505&isnumber=6917419

[11] Z. Chen and X. Huang, "End-to-end learning for lane keeping of self-driving cars," *2017 IEEE Intelligent Vehicles Symposium (IV)*, Los Angeles, CA, 2017, pp. 1856-1860.
DOI: 10.1109/IVS.2017.7995975
Available:
http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=7995975&isnumber=7995688