M. Tech. dissertation on

Autonomous driving simulation, based on Udacity self-driving car simulator

IS SUBMITTED IN PARTIAL FULFILLMENT OF THE MASTER OF TECHNOLOGY IN EMBEDDED SYSTEMS

Submitted By

WAFA ABDULLA V.T
2019PEB5458

Under joint supervision of

Internal Supervisor
Rakesh Bairathi
Associate Professor,
E.C.E Dept. MNIT, Jaipur

External Supervisor
Anurag Modi
Software Developer Engineer
Xilinx, Hyderabad

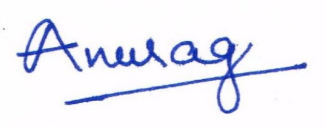
Department of Electronics and Communication Engineering Malaviya National Institute of Technology, Jaipur, India June, 2021

CERTIFICATE

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING
MALAVIYA NATIONAL INSTITUTE OF TECHNOLOGY JAIPUR – 302017,
RAJASTHAN, INDIA

This is to certify that this Dissertation report entitled "**Autonomous driving simulation, based on Udacity self-driving car simulator**" by **Wafa Abdulla V.T** (2019PEB5458), is the work completed under joint supervision and guidance, hence approved for submission in partial fulfilment for the award of the degree of **Master Of Technology** in **Embedded Systems** to the Department of Electronics and Communication Engineering, Malaviya National Institute of Technology, Jaipur in the academic session 2020- 2021 for full time post-graduation program of 2019-2021. The contents of this dissertation work, in full or in parts, have not been submitted to any other Institute or University for the award of any degree or diploma.

Date:



(Rakesh Bairathi)
Associate Professor,
ECE Dept. MNIT Jaipur

(Anurag Modi)
Software Developer Engineer
Xilinx, Hyderabad

DECLARATION

I, hereby declare that the work which is being presented in this project entitled " **Autonomous driving simulation, based on Udacity self-driving car simulator** " in partial fulfilment of the degree of Master of Technology in Embedded Systems is an authentic record of my own work carried out under the joint supervision and guidance of Rakesh Bairathi, Associate Prof., Dept. Of ECE, in Department of Electronics and Communication Engineering, Malaviya National Institute of Technology, Jaipur MNIT Jaipur & Anurag Modi, Software developer Engineer, Xilinx Hyderabad. I am fully responsible for the matter embodied in this project in case of any discrepancy found in the project and the project has not been submitted for the award of any other degree. I also confirm that I have consulted the published work of others, the source is clearly attributed, and I have acknowledged all main sources of help.

Wafa Abdulla V.T
2019PEB5458

ACKNOWLEDGMENT

I open this opportunity to express my deep sense of gratitude and respect towards my Supervisor (Guide), **Rakesh Bairathi**, Associate Professor, Department of Electronics and Communications Engineering, Malviya National Institute of Technology, Jaipur. I am very much indebted to him for the generosity, expertise and guidance, I have received from him while working on this project and throughout my studies. Without his support and timely guidance, the completion of my project would have seemed a far-fetched dream. In this respect, I find myself lucky to have him as my Project Guide. He has guided me not only with the subject matter, but also taught me the proper style and techniques of working. I would also like to thank Anurag Modi, Software Developer Engineer at Xilinx Hyderabad, for his cooperation and help rendered in numerous ways for the successful completion of this work. I take this opportunity to express my regards and obligation to my husband and other family members whose support and encouragement, I can never forget in my life. Lastly, I am thankful to All those who have supported me directly or indirectly during the dissertation work.

Wafa Abdulla V.T
2019PEB5458

ABSTARCT

Autonomous driving simulation, based on Udacity self-driving car simulator

Autonomous driving is a golden area of research and study since the end of 20th century, since it is offering multiple useful things, like increase in safety, reduction in traffic congestion and saves time. Autonomous car is a good revolution in the automotive industry. In this thesis explains the process of Autonomous driving on Udacity's self-driving car simulator. In this work two models are trained and simulated. Model1 is my self learning outcome of reference paper given in 22, which is modified NVIDIA model and model 2 is actual NVIDIA model which is referenced form document given in the reference no 34 are used for training to predict steering angle of car based on input pre-processed images which is obtained using manual movement of car in a selected track in Udacity's self-driving car simulator.

The trained models (model1(modified NVIDIA model) and model2 Actual NVIDIA model[34]) can predict steering angle for an autonomous driving car with input as autonomous driving car centre cameras pre-processed image. The trained model/client is communicated with Udacity simulator/server via socket io module. After successful connection establishment the models correctly predicts steering angle and autonomous car moves according to decisions for each instant captured by autonomous car camera.

Here Model1(Modified Nvidia model) is an outcome of my activities including self learning with the help of various sources for the implementation of my base paper[22] concept.

By performance point of view the model1(modified NVIDIA model) took less training time around 45 min and good behaviour in autonomous movement compared to model2(Actual NVIDIA model[34])which took around 1hr to 1hr+15 minutes training time and little bit abrupt behaviour while autonomous model. But in terms of mean square error of (Steering angle predicted by model and input steering angle to model), model2 performs better than model1 by a slight difference.

Keywords: - Neural network, Udacity, Simulator, Autonomous, NVIDIA.

Contents

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

ML - Machine learning

CARLA - Car Learning to Act

rFpro - A simulation environment for the automotive and motorsport industries

ADAS - Advanced driver-assistance systems

SDV - Static Driver Verifier

LIDAR - Light Detection and Ranging

OPAL-RT – Real time simulation platform

TSRD – Traffic sign recognition Database

GTSRB - German Traffic Sign Recognition Benchmark

CSV - comma-separated values

GPU - Graphics processing unit

CPU - Central processing unit

MSE – Mean square error

ADAM - active design and analysis modelling

HDF - Hierarchical Data **Format**

PID – Proportional integral derivative

IO – Input output

WSGI - Web Server Gateway Interface

VGGNET - Visual Geometry Group at University of Oxford

CNN -Convolutional neural network

CHAPTER 1

INTRODUCTION

1.1 Motivation

An autonomous car is a vehicle capable of capturing its surroundings and operate without human involvement. A human passenger is not required to take actions of the vehicle at any time, even a human passenger not required to be present in the vehicle at all. An autonomous car can go anywhere as a traditional car goes and do everything that an experienced human driver does. Here is an autonomous vehicle trained and tested using open-source simulator (Udacity's self-driving car simulator). It is applicable for further processing (e.g.: testing of autonomous vehicle actions towards traffic signs/obstacles etc). The intension of a Self-driving car project is to develop a good autonomous driver. The car should be capable to drive itself, to navigate itself without falling on the track.

Autonomous driving is a golden area of research and study since the end of 20th century, since it is offering multiple useful things, like increase in safety, reduction in traffic congestion and saves time. Autonomous car is a good revolution in the automotive industry.

Nowadays most of around 70percent of road accidents can be related to human miss behaviour. Safety will be the first measure to consider while developing a human friendly autonomous car. Any autonomous car drives automatically without human support or intervention. It can be referred as a smart car also.

Any system study or analysis simulation will be the first initial methodology to implement. Designers or developers should ensure to consider a scenario matching to real time behaviour so as to innovate a human friendly system.

Artificial intelligence and machine learning are the true methodologies to be considered for a model to train to behave good in autonomous mode. Machine learning is a kind of artificial intelligence which results machines the ability to understand without programming. Machine learning has various possibilities like learn or understand from data and predicts on new data. For this thesis our main intension is to simulate the autonomous driving system according to modes trained, which is going to predict steering angle for autonomous car. Unity is providing game engines similar to video 3d games, can be utilized for this.

1.2 Problem definition:

Udacity created self driving cars open source simulator to imitate a real-time scenario. The strategy is to follow the driving behaviour of a human on the simulator platform with the help of a models trained by deep neural networks using pytorch framework.

The simulator involves 2 tracks and 2 modes, named as, training mode and autonomous mode. The dataset is generated in training mode of simulator using keyboard control keys. This dataset is the model training input data. This is followed by training the model, testing the trained model on the track, to verify how the deep learning model performs after being trained by the user data. There are 2 different neural network models has been considered for training, model1 is neural network layer reduced version of NVIDIA mode/modified NVIDIA model and model2 is actual NVIDIA neural network model. Train both the models separately and analyse mean square error loss of steering angle (error among model predicted steering angle and input image steering angle), Deploying the trained model locally and simulate autonomous car movement with both models individually.

Nowadays most of, around 70 percent of road accidents can be related to human misbehaviour. Safety is the first measure to consider while developing a human friendly autonomous car. Any autonomous car drives automatically without human support or intervention. It can be referred as a smart car also.

1.3 Literature survey

1.3.1 The simulators of autonomous vehicles

CARLA [12] is developed to support training, and validation of autonomous driving models. In extra to open-source code , CARLA [12] provides different tracks (urban layouts, vehicles, buildings) it can be used for research and study purpose. The simulation platform supports flexible to environmental conditions or whether conditions, full control of all static and dynamic elements, and much more.

rFpro [23] gives driving simulation software, ADAS and Vehicle Dynamics development, testing and validation. rFpro [23]simulator can be used to train, test and validate Supervised Learning systems for **ADAS and Autonomous applications** [23]. rFpro's weather and physically modelled atmosphere, outputs real-time reflections.

Wipro's [24] introduced driverless car simulator (also known as SDV in a box or self-driving car in a box) is a simulator applicable to test and validate the navigation algorithms of autonomous vehicles. It acts as a simulation platform for the cars before being pulled out onto the roads. The simulator is capable of taking the vehicle into its limits of operations by navigating its through challenging real-life scenarios.

OPAL-RT's [25] systems overcome difficulties which automotive industry faces when testing autonomous vehicle by putting physical testbeds onto simulation environments. OPAL-RT systems are flexible enough to integrate each new technology as it is introduced to the vehicle, from data fusion and deep learning to new sensors such as LIDAR.

Udacity self-driving simulator [22]: Udacity self driving car simulator's idea behind to teach students on training different models on different tracks with innovative scenarios. Udacity self driving car simulator designed using Unity engines platform. This simulator will capture images from 3 cameras left, centre, and right cameras to autonomous car. Neural networks are

the main technology going to train the model which will support autonomous driving. The models will predict steering angle for autonomous movement. Udacity provides a simulator which has two modes of operation names training and autonomous ,where in training mode it will collect data and store in some directory to consider the data later time during training. We can train the model using any active libraries which support inbuilt neural network libraries and train it. After train it can be possible to test the trained model accuracy using autonomous mode of simulator.

Using deep neural networks 2 different phases are involved. Training and interreference faces.[34].During training time network weights are incrementally updated with its backpropagation errors which it gets from the example training data's. Once the model training completed goes to inference section, there our trained model capable to analyse new data inputs. Training phase requires high computational performance compared to testing phase.

Using Udacity self driving car simulator it is capable to edit and stretch the simulator tracks via its backend development codes to create suitable tracks matching to our scenario[21].Which is going to perform as similar to the same process how Udacity simulator works for other models. That is data collection via training mode and ,further train the model separately , and interface trained model with simulator in autonomous mode. I have kept the paper basis as a supporting document for my future works.

1.3.2 Basics of AI, Machine learning, deep learning and Neural networks.

Artificial intelligence (AI): is a technology in which human brain invention in to machines. How humans or animals take decisions or actions to situations by similar fashion machine will be capable to do. For human being neural system performs a vital role on control actions or decision making, similarly for artificial intelligence neural networks are the basic and fundamental base to be designed for proper decision making.

Machine learning (ML) :is a research area that can be improved by learning more and more data. It is variant of Artificial intelligence. There are various machine learning algorithms present. These algorithms develop or build a model based on input data or using data which we are passing for training. which can make decisions or predict after model training, without an explicit programs written in any language. Its usages are open to wide variety of applications in classification, autonomous car etc, only thing is proper model to be developed which will support our system after training.[29].

Deep learning: Deep learning is a part of machine learning where will implement mainly neural networks. Different types of neural networks available. Out of which our scope of interest for this thesis is convolutional neural networks. Neural networks layer to layer feature transition happens in using convolution technique.[30]

Neural networks:

Features:Features are the fundamental unit of Convolutional neural networks. It highlights the character of an image. Like edges, corners, borders etc. It gives a good understanding on analysing image similarities.

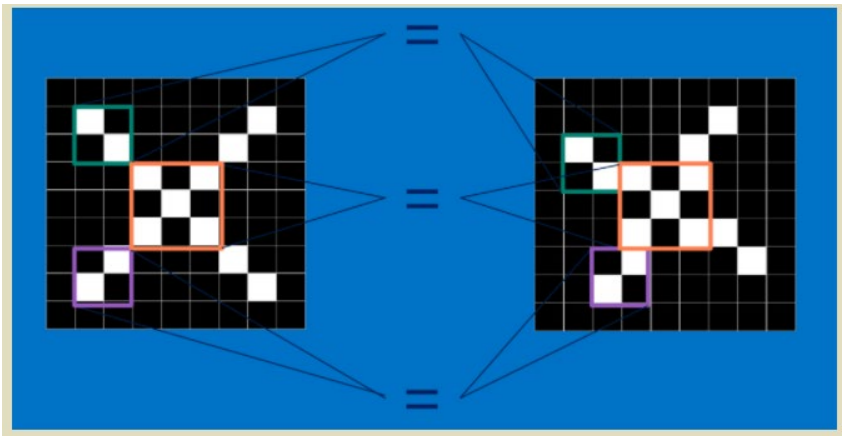


Figure 1. 1 CNN features [5]

Each feature depicts a mini image with a two dimensional set of values. If 2 images feature matches implies similarities among both of the images.

Convolution:

With a new image convolutional neural networks performs convolution operation with the kernal we defined by placing each and every position on the input image to depict where exactly the feature exists. If required there will be zero padding also included.

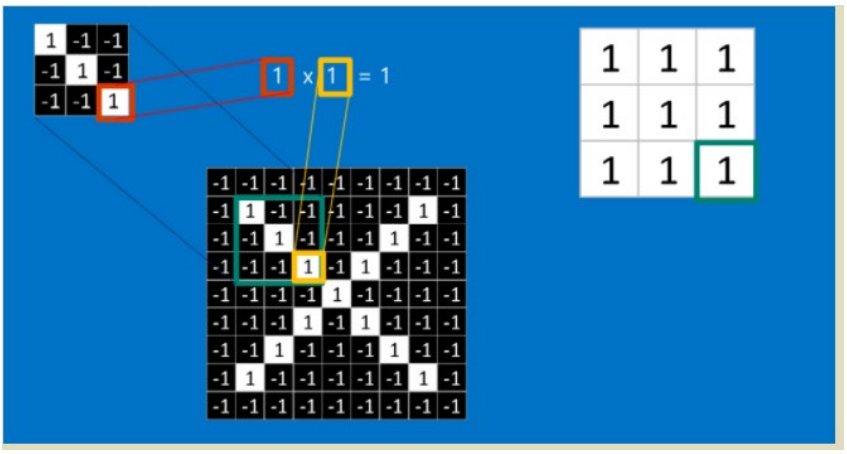


Figure 1. 2 Convolution [5]

Pooling:

Next important utility in convolutional neural networks is pooling. It a kind of minimizing the input image dimensions by keeping the important features or information. Window method is used to achieve pooing, by passing a 2 or 3 pixel window ,will take highest value in the window for to create output matrix. Here highest value will be best information gathering feature.

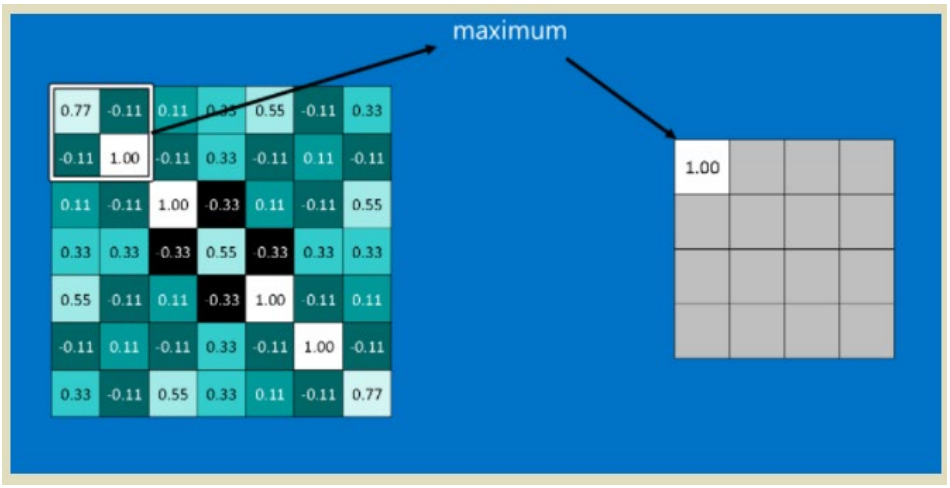


Figure 1. 3 Pooling [5]

Relu (Rectifier linear network)

Relu is Rectified Linear Unit, Its simply putting in the incoming matrix by zero for all -ve values.The advantage of Relu unitis to keep CNN mathematically good by learned values from getting stuck near 0 .

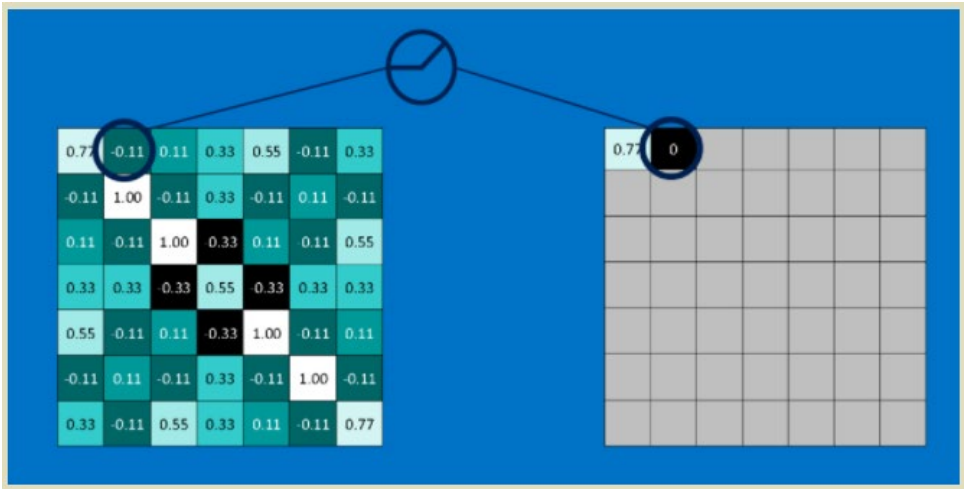


Figure 1. 4 Rectifier linear network [5]

Fully connected layers:

Fully connected layers take the high-level filtered images which is going to translate in to class label votes. And it will be primary building block of neural networks, Instead of treating input images as a two-dimensional array, fully connected layers will be treating as a single list array and all values considered identically. Some of the values will be best for describing that particular image is of our targeted category or not, those values or features which imitates our image quality will get high votes among others.

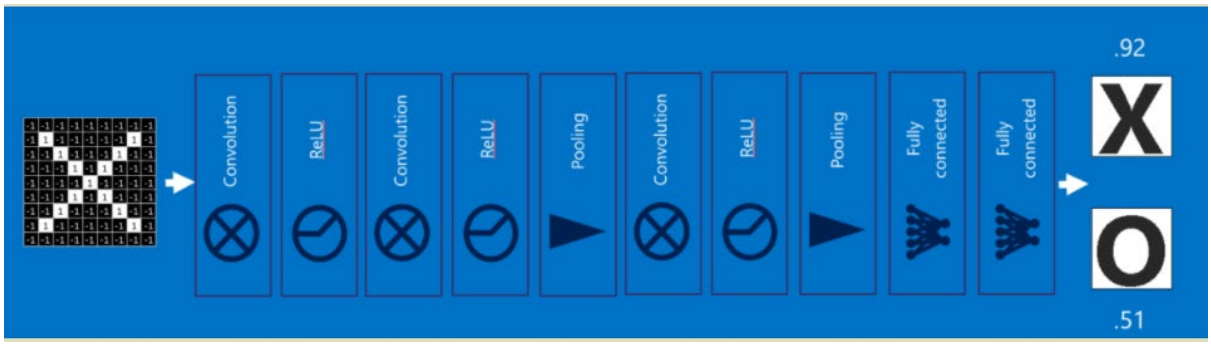


Figure 1. 5 Overall neural network model [5]

1.4 Outline of thesis

This thesis presents the simulation of an autonomous driving vehicle using Udacity self-driving car open-source simulator, and road traffic sign recognition and classification using germen dataset as an enhancement study. This thesis trained a neural network model and establishing connection with server (self-driving car simulator) to achieve autonomous driving.

In this section we have a glance at different chapters included. Starting with

chapter 2 Describes the workspace setup required before start working on thesis, introduces the software used and usage.

chapter 3 introduces the Udacity self-driving car simulator's features and data collection for to train the neural network model.

Chapter 4 explains input data pre-processing procedures used before training neural network model for a smooth training process. Which includes cropping the image as to focus the image in to road track only, flipping the image if its more right aligned or left aligned to make sure always car to be located at the centre of the track.

Chapter 5 includes 2 neural networks models used for training [Model1 (Layer reduced version of NVIDIA model) and model2 (Actual NVIDIA model)] its training process on google colab platform and mean square error obtained for each epochs.

Chapter 6 Consist of deploying the trained models(both model1 and model2 separately) locally and interfacing /establish server client connection for models testing, using server engine IO module.

Chapter 7 Describes simulation and autonomous driving testing using both trained models(model1 and model2), Establishing server (self-driving car simulator) and client (trained neural network model) connection and execution results.

CHAPTER 2

SETUP WORKSPACE

2.1 Udacity self driving car simulator

This Udacity self driving car simulator was created by **Udacity**[26], to learn students how to train or how to train neural network models and how to navigate on different road tracks using machine learning or deep learning.

Download the Udacity self-driving car simulator from Udacity website (term1 simulator) beta version. Extract it. term1 is the basic simulator created for study purpose. New release simulators also available in Udacity website, for advanced study purposes.

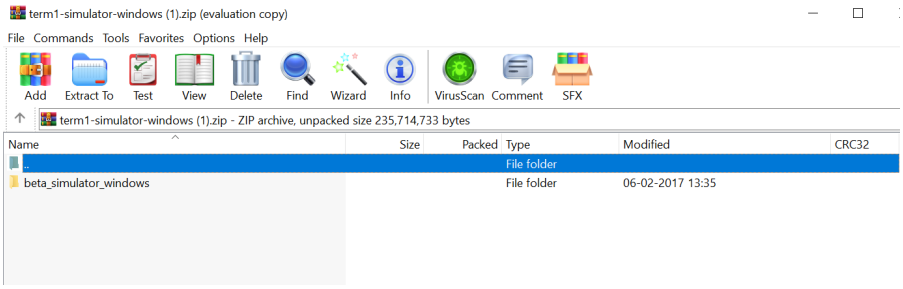


Figure 2. 1 Downloaded term1 Udacity self-driving car simulator application

Inside the simulator folder, there is one data folder and executable .exe file.

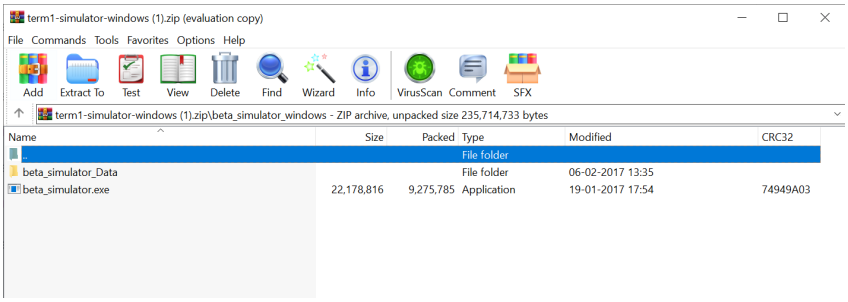


Figure 2. 2 Extracted term1 Udacity self-driving car simulator data

Run that .exe file it opens a window which has Udacity self-driving car simulators screen resolution data's as shown in fig 2.3

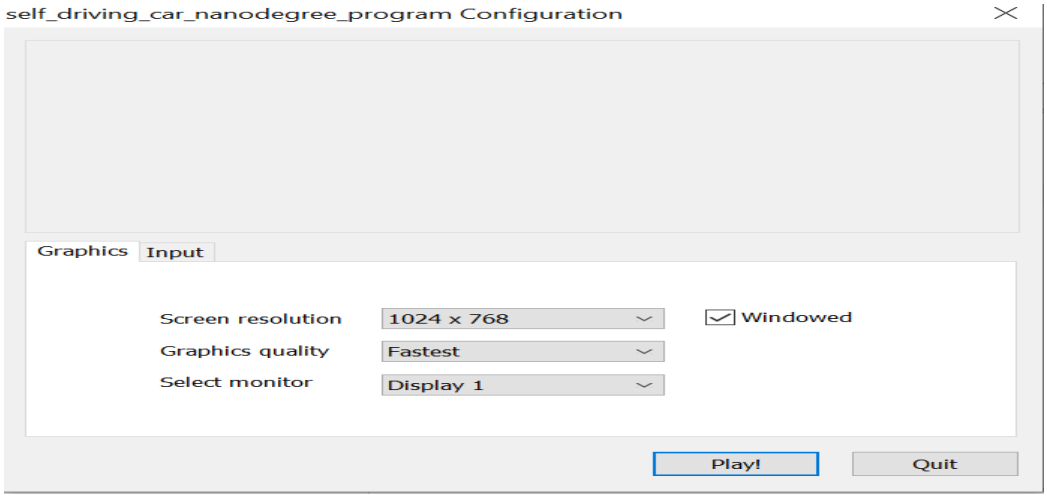


Figure 2. 3 Screen resolution window

The data folder shown in fig 2.4 inside simulator app has backend resources and later logs file gets saved once we start working on it.

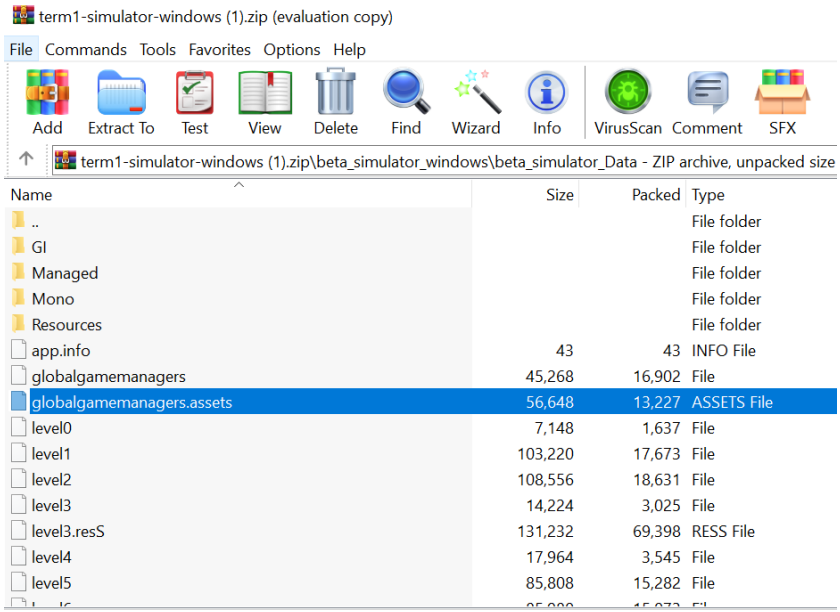


Figure 2. 4 Udacity simulator data folder

Choose required screen resolution and click on play to go to main window. It opens a window similar to fig 2.5.

Figure 2. 5 Udacity simulators main window.

Simulator setup done.

2.2 Training environment:

For training neural network models(here both model1 and model2) best platform is google colab. Colab notebooks execute code on Google's cloud servers, meaning you can leverage the power of Google hardware, including GPUs and TPUs.

Google colab disk information

```
[1]  !df -h

     Filesystem      Size  Used Avail Use% Mounted on
     overlay         108G   39G   70G  36% /
     tmpfs            64M     0   64M   0% /dev
     tmpfs           6.4G     0  6.4G   0% /sys/fs/cgroup
     shm             5.9G     0  5.9G   0% /dev/shm
     tmpfs           6.4G   28K  6.4G   1% /var/colab
     /dev/sda1        76G   41G   36G  54% /etc/hosts
     tmpfs           6.4G     0  6.4G   0% /proc/acpi
     tmpfs           6.4G     0  6.4G   0% /proc/scsi
     tmpfs           6.4G     0  6.4G   0% /sys/firmware
```

Figure 2. 6 Google colab disk information

41GB of SD card partition space available. Which can be used for training.

see CPU specifications execute **!cat /proc/cpuinfo**

For memory execute command - **!cat /proc/meminfo**

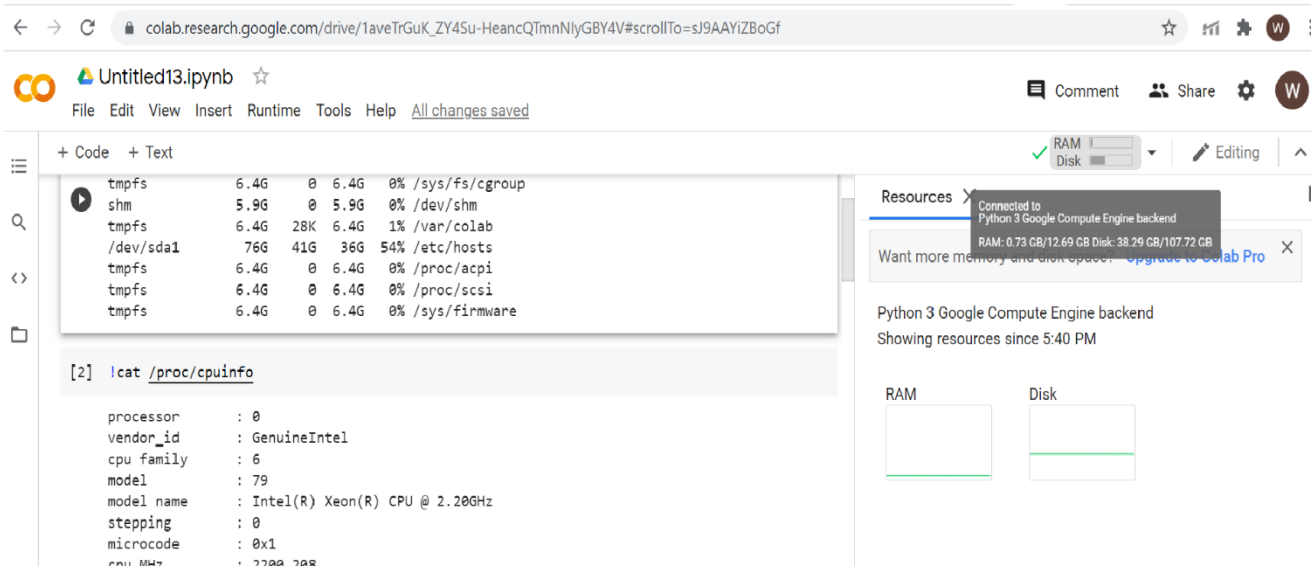


Figure 2. 7 Google colab notebook structure

Google colab provides 12.69 GB RAM, 107.72 GB disk space as shown in fig 2.7.

Colab has most of the libraries preinstalled, we can directly import it and use for testing.

If any utilities are not imported then it's possible to install through pip command.

Pip installs {utility}. Click on code button to add sections, and click on the play icon for execution. It is possible to upload the notebook in to GitHub, download the saved notebook to local machine, or even save data to drive for future use.

2.3 Server (Udacity simulator) to client (Trained model) interfacing workspace:

Anaconda Individual Edition is the world's most popular **Python** distribution platform.

Install anaconda from anaconda website. Used anaconda version 4.9.2 and python version 3.7.6.

(base) C:\Users\Wafa>conda --v
conda 4.9.2
(base) C:\Users\Wafa>python -V
Python 3.7.6
Jupiter notebook, spider and every dependency gets installed along with anaconda installation. We can enable required utilities during installation. After that also it is possible to install though conda command or pip command.

After anaconda installation there is one prompt for user inputs as shown in figure 2.8. We can change directory to our project directory and execute commands over there.

Figure 2. 8 Anaconda prompt

Check, list of utilities available in conda by using **conda list** command.

And we required one python idle which can be easy to use for code correction / project corrections. Spyder/Jupiter notebook is fine, I have used **atom** application for editing and manipulating the python code.

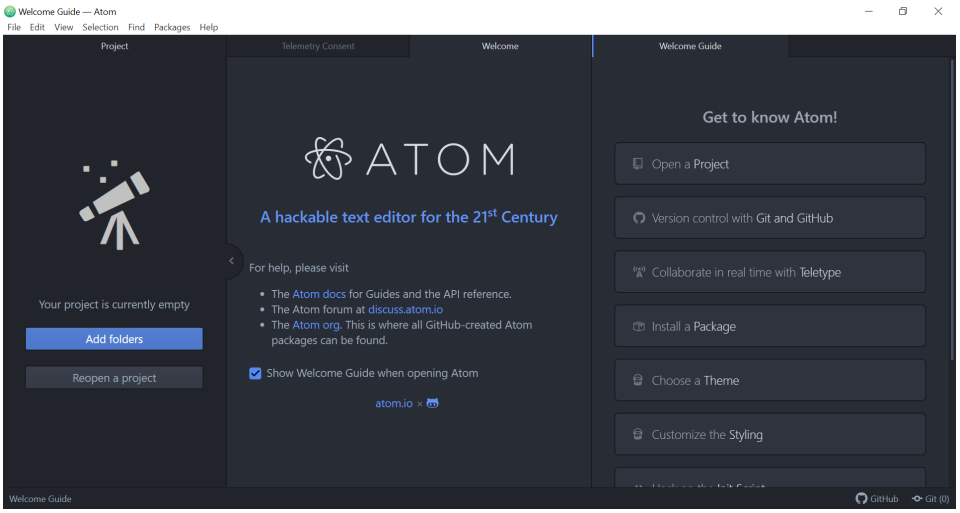


Figure 2. 9 atom application

Atom is a open-source and free source code editor for Linux, Mac OS and Microsoft Windows with support for plug-ins written in JavaScript, and embedded Git Control, developed by GitHub. **Atom** is a desktop **application** built using web technologies. Where project structures can be visible in left panel and right side editor window is visible.

CHAPTER 3

DATA COLLECTION FOR TRAINING.

Udacity self-driving car simulator is an open source which can be downloaded for to train cars using deep learning. A self-driving car simulator created with Unity.

Unity is game engine developed by Unity Technologies, first announced and released in June 2005 at Apple Inc.'s Several major versions of Unity have been released since its launch. The latest stable version, 2021.1.0, was released on 23 March 2021;

Figure 3. 1 Unity Engine

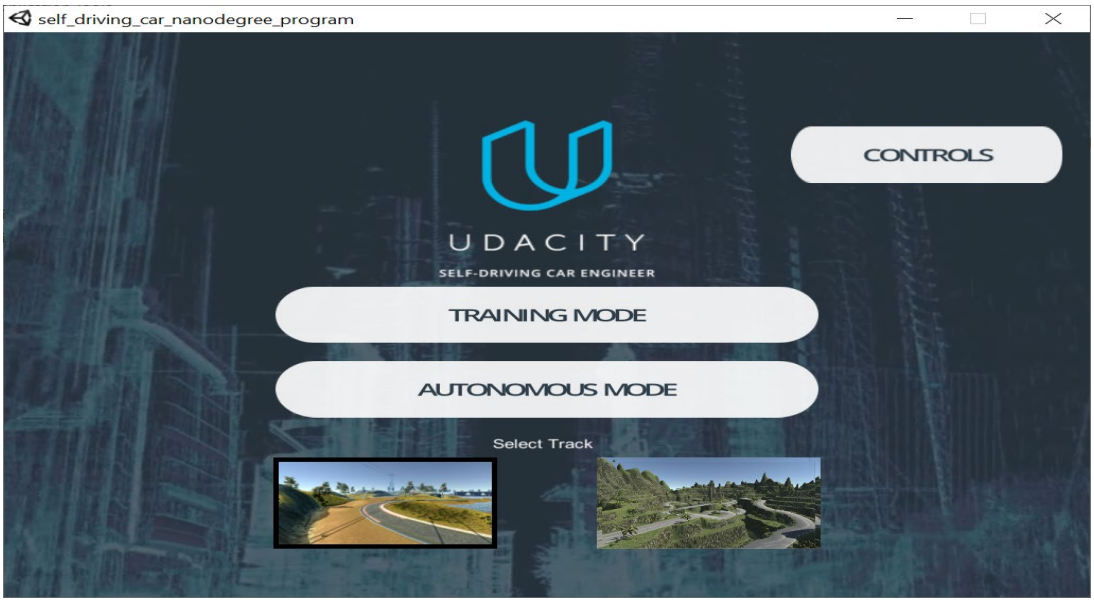


Figure 3. 2 Udacity self-driving car simulator

Using our keyboard to drive the car, we were able to direct the simulated vehicle to turn left, right, move straight accordingly. The main attraction of this simulator is that it can be used for training as well as testing the model. It has two modes of operation: (I) Training mode, and (ii) Autonomous mode as shown in Fig. 3.2

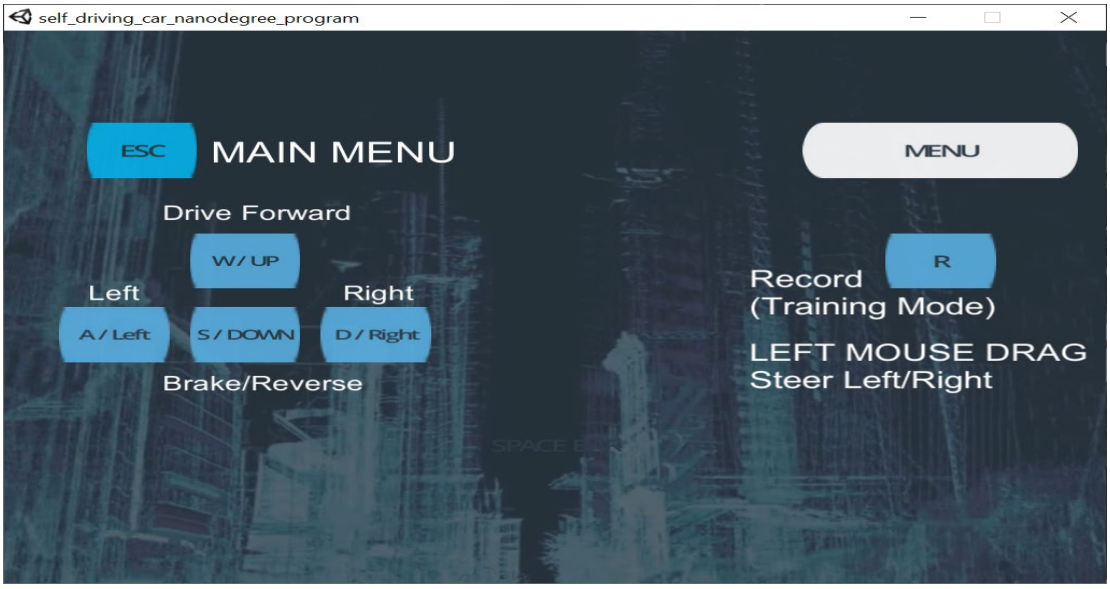


Figure 3. 3 Udacity simulator's key board control keys

The training mode is used to collect the training data and the autonomous mode is used to simulate the trained model or simply for to test model. There are two kinds of road tracks available in the simulator - lake track and the jungle track. The lake track is smaller and easy to handle the car when compared with the jungle track as shown in Fig. 3.4 and Fig. 3.5. The simulator records data when the car is moving around the track, using left and right key borad keys to control the steering angles of car and up ,down key board key arrows to control speed.

Figure 3. 4 lake track

Figure 3. 5 jungle track

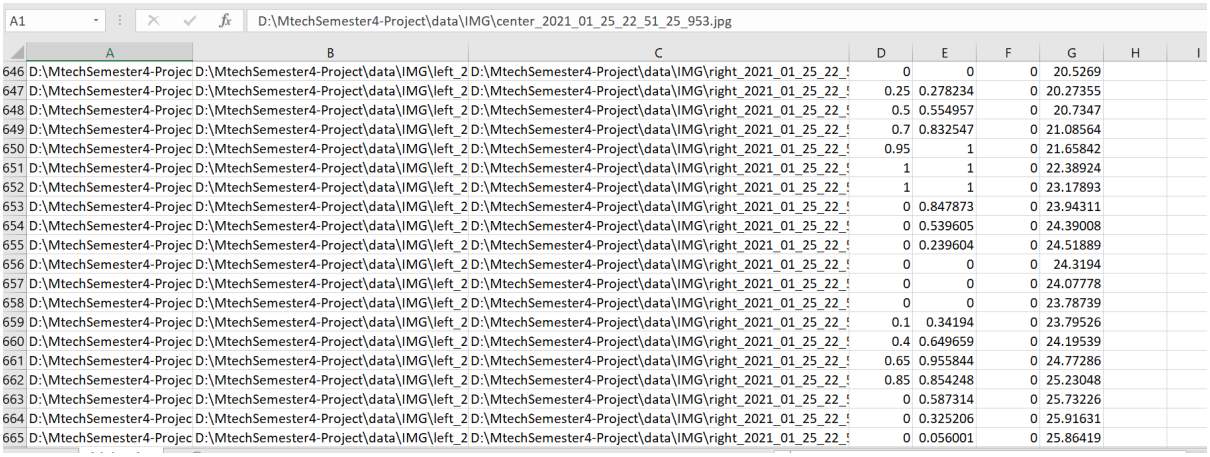
The simulator generates a folder named data which contains recorded images and one CSV file. The image folder contains three images for every snap captured by the left, centre and right camera and every row in the CSV file contains four metrics - steering angle, speed, throttle and brake, captured frame. Fig. 3.6, Fig. 3.7. and Fig. 3.8 shows the left, centre and right image, for single instance image.

Figure 3. 6 Left Image

Figure 3. 7 Centre Image

Figure 3. 8 Right image

After data collection there are total **5,505** images including right, left and centre camera images has saved in data folder. It took around 30 min for training data collection.



A1 D:\MtechSemester4-Project\data\IMG\center_2021_01_25_22_51_25_953.jpg

	A	B	C	D	E	F	G	H	I
646	D:\MtechSemester4-Projec	D:\MtechSemester4-Project\data\IMG\left_2	D:\MtechSemester4-Project\data\IMG\right_2021_01_25_22_	0	0	0	20.5269		
647	D:\MtechSemester4-Projec	D:\MtechSemester4-Project\data\IMG\left_2	D:\MtechSemester4-Project\data\IMG\right_2021_01_25_22_	0.25	0.278234	0	20.27355		
648	D:\MtechSemester4-Projec	D:\MtechSemester4-Project\data\IMG\left_2	D:\MtechSemester4-Project\data\IMG\right_2021_01_25_22_	0.5	0.554957	0	20.7347		
649	D:\MtechSemester4-Projec	D:\MtechSemester4-Project\data\IMG\left_2	D:\MtechSemester4-Project\data\IMG\right_2021_01_25_22_	0.7	0.832547	0	21.08564		
650	D:\MtechSemester4-Projec	D:\MtechSemester4-Project\data\IMG\left_2	D:\MtechSemester4-Project\data\IMG\right_2021_01_25_22_	0.95	1	0	21.65842		
651	D:\MtechSemester4-Projec	D:\MtechSemester4-Project\data\IMG\left_2	D:\MtechSemester4-Project\data\IMG\right_2021_01_25_22_	1	1	0	22.38924		
652	D:\MtechSemester4-Projec	D:\MtechSemester4-Project\data\IMG\left_2	D:\MtechSemester4-Project\data\IMG\right_2021_01_25_22_	1	1	0	23.17893		
653	D:\MtechSemester4-Projec	D:\MtechSemester4-Project\data\IMG\left_2	D:\MtechSemester4-Project\data\IMG\right_2021_01_25_22_	0	0.847873	0	23.94311		
654	D:\MtechSemester4-Projec	D:\MtechSemester4-Project\data\IMG\left_2	D:\MtechSemester4-Project\data\IMG\right_2021_01_25_22_	0	0.539605	0	24.39008		
655	D:\MtechSemester4-Projec	D:\MtechSemester4-Project\data\IMG\left_2	D:\MtechSemester4-Project\data\IMG\right_2021_01_25_22_	0	0.239604	0	24.51889		
656	D:\MtechSemester4-Projec	D:\MtechSemester4-Project\data\IMG\left_2	D:\MtechSemester4-Project\data\IMG\right_2021_01_25_22_	0	0	0	24.3194		
657	D:\MtechSemester4-Projec	D:\MtechSemester4-Project\data\IMG\left_2	D:\MtechSemester4-Project\data\IMG\right_2021_01_25_22_	0	0	0	24.07778		
658	D:\MtechSemester4-Projec	D:\MtechSemester4-Project\data\IMG\left_2	D:\MtechSemester4-Project\data\IMG\right_2021_01_25_22_	0	0	0	23.78739		
659	D:\MtechSemester4-Projec	D:\MtechSemester4-Project\data\IMG\left_2	D:\MtechSemester4-Project\data\IMG\right_2021_01_25_22_	0.1	0.34194	0	23.79526		
660	D:\MtechSemester4-Projec	D:\MtechSemester4-Project\data\IMG\left_2	D:\MtechSemester4-Project\data\IMG\right_2021_01_25_22_	0.4	0.649659	0	24.19539		
661	D:\MtechSemester4-Projec	D:\MtechSemester4-Project\data\IMG\left_2	D:\MtechSemester4-Project\data\IMG\right_2021_01_25_22_	0.65	0.955844	0	24.77286		
662	D:\MtechSemester4-Projec	D:\MtechSemester4-Project\data\IMG\left_2	D:\MtechSemester4-Project\data\IMG\right_2021_01_25_22_	0.85	0.854248	0	25.23048		
663	D:\MtechSemester4-Projec	D:\MtechSemester4-Project\data\IMG\left_2	D:\MtechSemester4-Project\data\IMG\right_2021_01_25_22_	0	0.587314	0	25.73226		
664	D:\MtechSemester4-Projec	D:\MtechSemester4-Project\data\IMG\left_2	D:\MtechSemester4-Project\data\IMG\right_2021_01_25_22_	0	0.325206	0	25.91631		
665	D:\MtechSemester4-Projec	D:\MtechSemester4-Project\data\IMG\left_2	D:\MtechSemester4-Project\data\IMG\right_2021_01_25_22_	0	0.056001	0	25.86419		

Figure 3. 9 drivng_log.csv file generated by simulator's training mode

Where the CSV file generated by Udacity simulator consists of following data.

- Column 1, 2, 3: data images collected during training mode, as centre, right and left respectively
- Column 4: steering angle value, 0 indicates straight motion, negative value is left turn and positive value is right turn.
- Column 5: value of acceleration or throttle at each particular instance.
- Column 6: value of brakes or deceleration at each particular instance
- Column 7: speed of the car in during data collection

CHAPTER 4

DATA PRE-PROCESSING

The topmost problem was generalizing the behaviour of the car on lake track which we are using for this thesis, In a real-life situation, we can never train a self-driving car model for every track, as the data size to be trained is too huge. Also, it is not possible to collect data's for all different tracks and with different lighting conditions. Thus, there is a need to come up with an idea of simplifying the behaviour of autonomous car on various tracks/roads. This problem is solved up to limit by means of image pre-processing and augmentation techniques, which is discussed in the following section:

- Split the samples in to 2 sets , training and validation (80 percent to training and 20 percent of total data in to validation)
- Training process is computationally large and time-consuming even for GPUs. Since the dataset to be considered for training is huge. The trick is to parallelize this process by taking data in batchesused 32 batches , augmenting them and sending to the model to train. In Pytorch, used the Dataset class and the Data loader function to demonstrate the same.
- After choosing final set of datas , the data is augmented or added by cropping and rotations to teach the network on how to overcome from a bad position or alignment. During augmenting, randomly choose right, left or centre images, randomly flip the images left/right according to Y axis and adjust the steering angle. The steering angle is adjusted by +0.4 for the left image and -0.4 for the right image.

```
import cv2
import torch.utils.data as data
import numpy as np
from google.colab.patches import cv2_imshow

def augment(imgName, angle):
  name = 'data/IMG/' + imgName.split('/')[-1]
  current_image = cv2.imread(name)
  cv2_imshow(current_image)
  print('\n')
  current_image1 = current_image[65:-25, :, :]
  cv2_imshow(current_image1)
  if np.random.rand() < 0.5:
    current_image1 = cv2.flip(current_image1, 1)
    angle = angle * -1.0
  return current_image1, angle
```

Figure 4. 1 Image pre-processing functionality view [14].

Here function named augment shown in fig 4.1 has inputs as image name and steering angle.

That is passed from driving_log.csv file generated by simulators training mode during data collection.

First read the image, crop the image using image name[x1:y1:::] function. Given the values appropriately.[33]

The cropped image is then passed to an if loop if the np.random.rand() [20] returns floating value is less than 0.5. If the image is in the if loop, image is flipped with respect to Y axis and its corresponding steering agle multiplied by -1.

Figure 4. 2 Image before crop

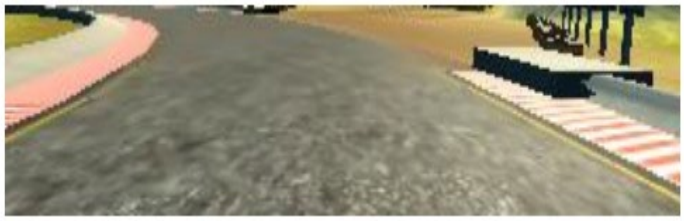


Figure 4. 3 Image after crop

As a part of pre-processing left images steering angle get added to 0.4, right images steering angle gets subtracted by 0.4 with steering angle received at training to maintain the car centre located.

```
: class Dataset(data.Dataset):

    def __init__(self, samples, transform=None):

        self.samples = samples
        self.transform = transform

    def __getitem__(self, index):

        batch_samples = self.samples[index]

        steering_angle = float(batch_samples[3])

        center_img, steering_angle_center = augment(batch_samples[0], steering_angle)
        left_img, steering_angle_left = augment(batch_samples[1], steering_angle + 0.4)
        right_img, steering_angle_right = augment(batch_samples[2], steering_angle - 0.4)

        center_img = self.transform(center_img)
        left_img = self.transform(left_img)
        right_img = self.transform(right_img)

        return (center_img, steering_angle_center), (left_img, steering_angle_left), (right_img, steering_angle_right)

    def __len__(self):
        return len(self.samples)
```

Figure 4. 4 Torch framework's Dataset class usage for data augmentation

In figure 4.4 Pytorch frameworks Dataloader class is defined. Passed index wised values from driving log.csv file generated by Udacity self-driving car simulators training mode.

While class invoke, it saugment the data accordingly. In the same function it is shown centre camera image, steering angle passed to augment function as same as value in driving_log.csv file generated.

From left and right images shifted the angles by +0.4, -0.4 with actual steering angle received at driving_log.csv file generated during data collection.

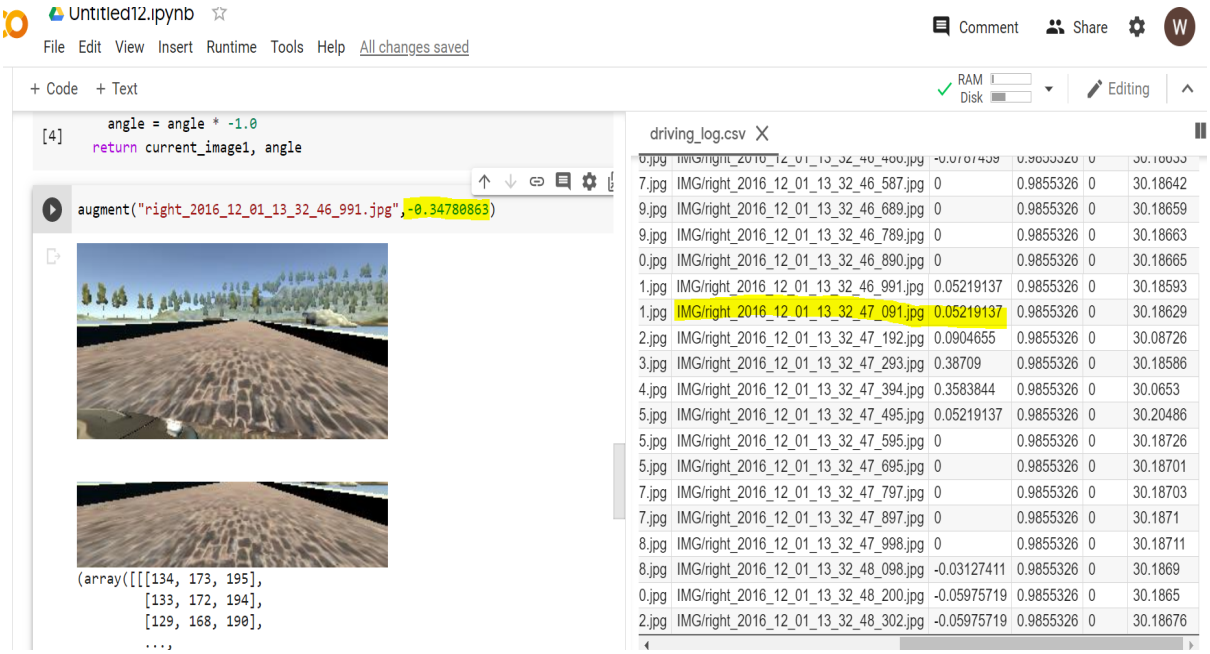


Figure 4. 5 Right camera image after pre-processing (crop + angle shifted)

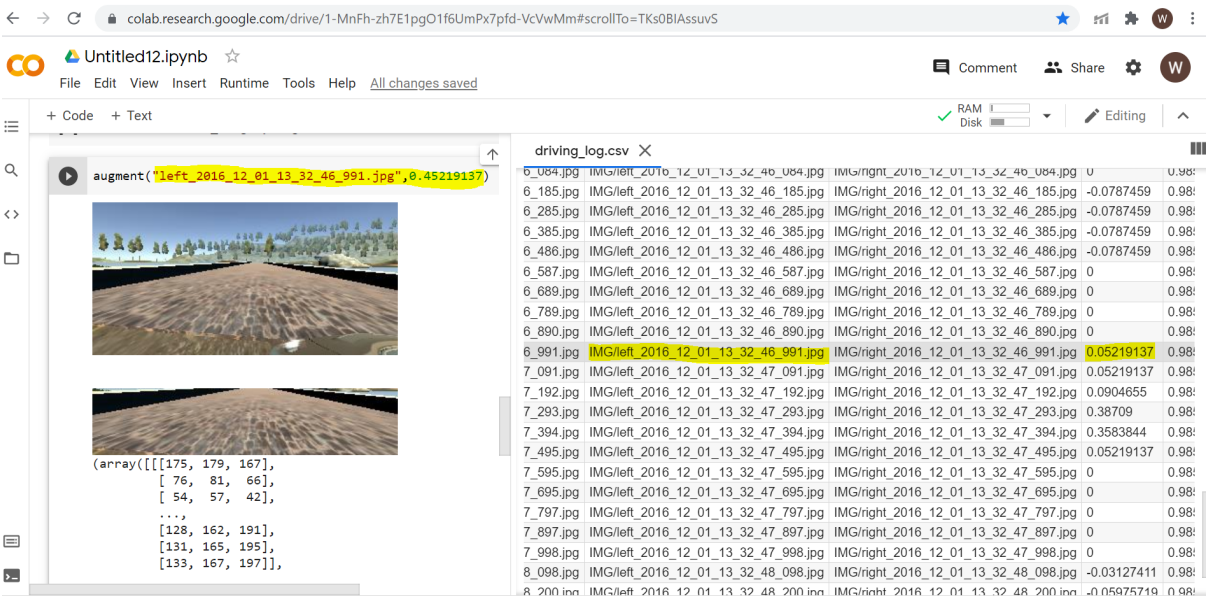


Figure 4. 6 Left camera image after pre-processing (crop + angle shifted)

In fig 4.5 there the first image is the image which we received at data collection. Driving log.csv file is opened at the right corner of the same image. Th steering angle was 0.05219137

During augment function we augments data such that right images shifted angle by -0.4 that is (0.5219137-0.4 = -0.347880863) is passed to augment function. The resultant image obtained is shown in the same image second one.

Similarly for left camera image added +0.4 with the steering angle obtained at driving_log.csv file, and called augment function the resultant is shown in fig 4.6.

CHAPTER 5

CNN MODELS AND TRAINING

In this chapter 5, 2 neural network models considered for training the input data. Model1(Modified NVIDIA network) is neural network layer reduced version of Actual NVIDIA model with(9 convolutional layers) as shown in fig 5.3 and model2 is Actual Nvidia neural network model as shown in figure 5.5.

During training mode Self driving car collects data with left, centre, right aligned cameras.

The simulator gives camera images, speed, throttle, steering angle values for each instant and recorded in a driving_log.csv file .The images from each camera is stored in a folder where user chooses as data collection directory. During pre-processing stage steering angle of the images are adjusted to keep the car movement steady and located to the middle of the track.

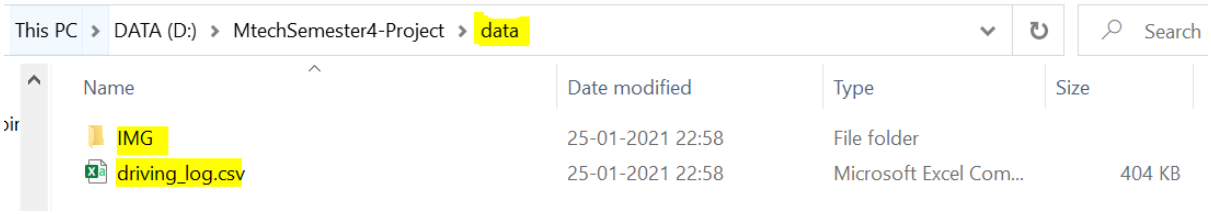


Figure 5. 1 Training image collection data structure

Once the data/images pre-processed for training the model, the images are passed to model1 (Modified NVIDIA model)and model2(Actual NVIDIA model[34] to train. **The training process is same ,only the CNN model used for training (either model1 or model2) in training system as shown in figure 5.2 will vary.**

5.1 Training system

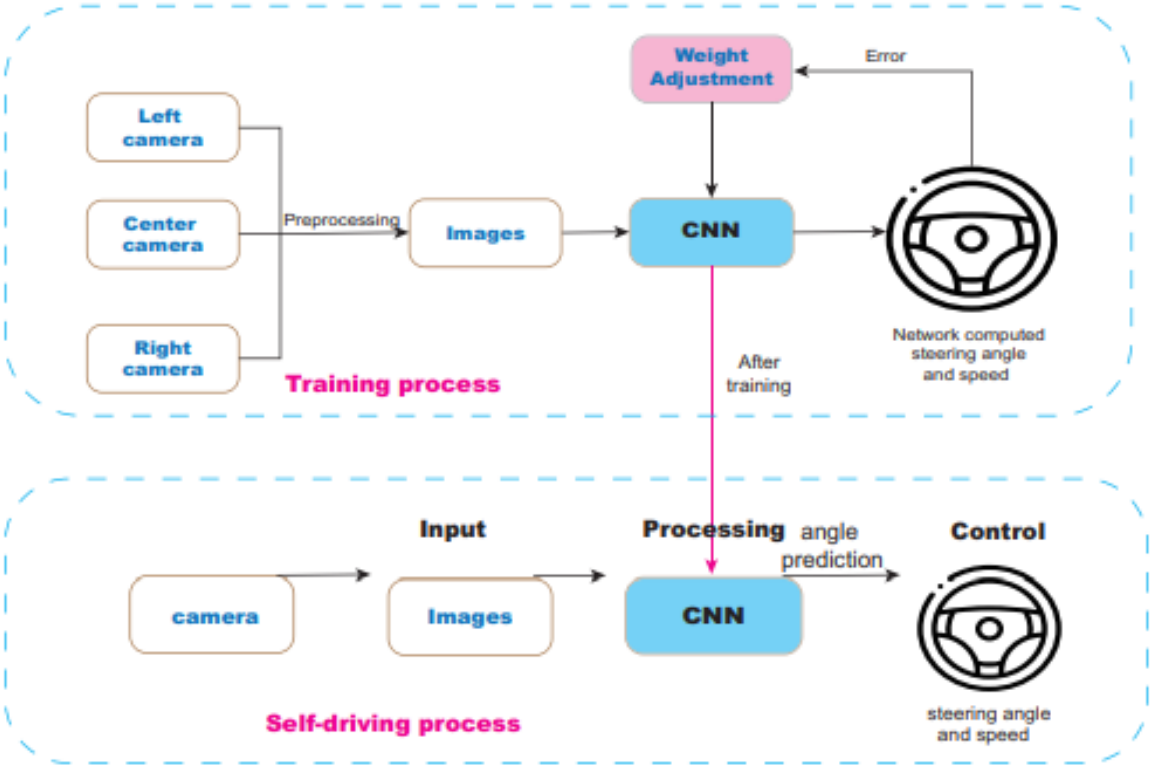


Figure 5. 2 Block diagram of training system [22]

During training the images collected by left,centre, and right cameras are recorded and stored in directoy,a driving_log.csv file which containes images names with steering angle ,throttle/accelaration,break and speed values of each instant. The data processed as described in chapter 4 is passed to the two models- model1(Modified NVIDIA model) and model2(Actual NVIDIA model[34]) for training as shown in fig 5.2 training process.

Come to self drivinf process of figure 5.2, the car is in Udacity self driving car simulators autonomous mode. The images capture by centre camera of the car is pre-processed and passed as an input to trained model. The trained model predicts the steering angle for that corresponding image.In autonomous mode driving the server is our Udacity simulator and cliend is our trained model.Once after successful server to client connection established the image send from server reaches to client ,predict the angle and pass to server back.This is how autonomous driving achieved.

Total 22 epoches are used while training.In each epoch calculated steering angle mean square error loss between model predicted steering angle and pre-processed steering angle of each image data.

5.2 Model 1:Modified NVIDIA model

Here Model1(Modified Nvidia model) is an outcome of my activities including self learning with the help of various sources for the implementation of my base paper[22] concept.

Model summary of first trained model is given in fig 5.3, total 9 neural network layers used.In fig 5.3 output shape(tensor output shape-pytorch framework uses tensors similar to numpy arrays for storing shapes/dimension values) of each layer also given in the same figure.

Neural Network layers were organized in sequential and various combinations of Convolution layers ,Max Pooling, Flatten, Dropout, Dense and so on are used in architectures. The best performing ones are shown in detail. The **feature** is the output of one filter(kernel used in convolution) applied to the previous sequential layer. A given kernel moves throughout the entire previous layer, moved one pixel at a time. Each position results in an activation of the **neuron** and the output are collected in the **feature map.**

Maximum pooling, is a **pooling** operation that calculates the **maximum**, or highest value in each window of each feature map. The pooling operation results feature maps that highlight the most present feature in the image by passing the window kernel(a 2x2 or 3x3 matrix) .

Fully Connected layers in neural networks are **layers** where all the inputs from one **layer** are **connected** to every activation unit of the next **layer**. In almost all neural networks models, the last few **layers** are **fully connected layers** which compiles the data extracted by previous **layers** to form the final output (here model predicted steering angle).

```
torch.Size([2, 48, 4, 19])
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1         [-1, 24, 34, 159]             672
               ELU-2         [-1, 24, 34, 159]               0
            Conv2d-3          [-1, 48, 16, 79]          10,416
         MaxPool2d-4           [-1, 48, 4, 19]               0
           Dropout-5           [-1, 48, 4, 19]               0
            Linear-6                 [-1, 50]         182,450
               ELU-7                 [-1, 50]               0
            Linear-8                 [-1, 10]             510
            Linear-9                  [-1, 1]              11
================================================================
Total params: 194,059
Trainable params: 194,059
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.26
Forward/backward pass size (MB): 2.50
Params size (MB): 0.74
Estimated Total Size (MB): 3.50
----------------------------------------------------------------
```

Figure 5. 3 Model summary of model1(Modified Nvidia model)

Model 1's corresponding torch framework implementation is shown in fig 5.4

```
class NetworkLight(nn.Module):

    def __init__(self):
        super(NetworkLight, self).__init__()
        self.conv_layers = nn.Sequential(
            nn.Conv2d(3, 24, 3, stride=2),
            nn.ELU(),
            nn.Conv2d(24, 48, 3, stride=2),
            nn.MaxPool2d(4, stride=4),
            nn.Dropout(p=0.25)
        )
        self.linear_layers = nn.Sequential(
            nn.Linear(in_features=48*4*19, out_features=50),
            nn.ELU(),
            nn.Linear(in_features=50, out_features=10),
            nn.Linear(in_features=10, out_features=1)
        )


    def forward(self, input):
        input = input.view(input.size(0), 3, 70, 320)
        output = self.conv_layers(input)
#        print(output.shape)
        output = output.view(output.size(0), -1)
        output = self.linear_layers(output)
        return output
```

Figure 5. 4 Torch framework implementation of model 1 defined in fig 5.3[35]

In Table 5.1 training loss and validation loss of all 22 epochs data trained with model1 is given.There were 4 co-works defined and they worked parallelly at a time. After complete training the optimizer took average of all errors (Error between pre processed steering angle and model predicted steering angles mean square error) among all coworkers and set the average as meansqaure error as the estimated error of model1.

Co-working means data splitted and works parallaly at a time.

For 22 epochs data training around 45 minutes of cpu time elapsed for model1.

5.3 Model 2:Actual NVIDIA model[34]

Model summary of second trained model is given in fig 5.6, total 16 neural network layers used.In fig 5.6 output shape(tensor output shape-pytorch framework uses tensors similar to numpy arrays for storing shapes/dimension values) of each layer also given in the same figure.

Used convolutions with strides in the first three convolutional layers with a 2×2 stride and a 5×5 kernel and a convolution without strides with a 3×3 kernel size in the last two convolutional layers.We can use pytorch neural network libraries for the implementation of the model. **nn**.**Sequential** is a pytorch neural network Module which incorporates other layers, and adds those in sequentail to produce its output. Every Linear layers calculates output from input using a linear function, and holds internal Tensors with its weights. Sequential libraries primary intension is to work one layer next to other ,complete working of layer1 to last layer sequentially.

Dropout is a regularisation technique that can be added for a model to avoid overfitting.

Exponential Linear Unit or ELU is a function covers cost to zero faster and results good and accurate outcome. Different to other activation functions, ELU has an extra alpha constant which should be positive number, By default its value is

1. torch.nn.ELU (*alpha=1.0*, *inplace=False*)—Pytorch funtion

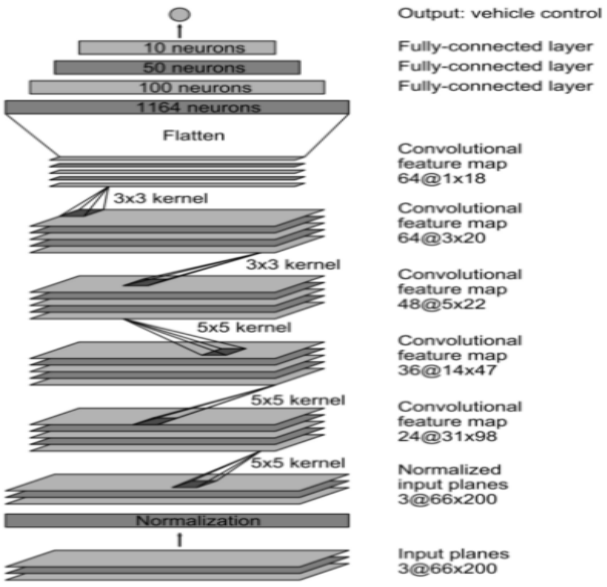


Figure 5. 5 model2 Actual NVIDIA model[34]

```
torch.Size([2, 64, 2, 33])
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1         [-1, 24, 33, 158]           1,824
               ELU-2         [-1, 24, 33, 158]               0
            Conv2d-3          [-1, 36, 15, 77]          21,636
               ELU-4          [-1, 36, 15, 77]               0
            Conv2d-5           [-1, 48, 6, 37]          43,248
               ELU-6           [-1, 48, 6, 37]               0
            Conv2d-7           [-1, 64, 4, 35]          27,712
               ELU-8           [-1, 64, 4, 35]               0
            Conv2d-9           [-1, 64, 2, 33]          36,928
           Dropout-10           [-1, 64, 2, 33]               0
           Linear-11                 [-1, 100]         422,500
              ELU-12                 [-1, 100]               0
           Linear-13                  [-1, 50]           5,050
              ELU-14                  [-1, 50]               0
           Linear-15                  [-1, 10]             510
           Linear-16                   [-1, 1]              11
================================================================
Total params: 559,419
Trainable params: 559,419
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.26
Forward/backward pass size (MB): 2.91
Params size (MB): 2.13
Estimated Total Size (MB): 5.30
----------------------------------------------------------------
```

Figure 5. 6 shows model summary of model2 (Actual NVIDIA model)

github.com/WafaAbdulla-tech/Autonomous-Vehicle-Design/blob/master/model_test.ipynb

In [ ]:

```
import torch.nn as nn
import torch.nn.init as init
import torch.optim as optim
import torch.nn.functional as F

class NetworkDense(nn.Module):

    def __init__(self):
        super(NetworkDense, self).__init__()
        self.conv_layers = nn.Sequential(
            nn.Conv2d(3, 24, 5, stride=2),
            nn.ELU(),
            nn.Conv2d(24, 36, 5, stride=2),
            nn.ELU(),
            nn.Conv2d(36, 48, 5, stride=2),
            nn.ELU(),
            nn.Conv2d(48, 64, 3),
            nn.ELU(),
            nn.Conv2d(64, 64, 3),
            nn.Dropout(0.25)
        )
        self.linear_layers = nn.Sequential(
            nn.Linear(in_features=64 * 2 * 33, out_features=100),
            nn.ELU(),
            nn.Linear(in_features=100, out_features=50),
            nn.ELU(),
            nn.Linear(in_features=50, out_features=10),
            nn.Linear(in_features=10, out_features=1)
        )

    def forward(self, input):
        input = input.view(input.size(0), 3, 70, 320)
        output = self.conv_layers(input)
        print(output.shape)
        output = output.view(output.size(0), -1)
        output = self.linear_layers(output)
        return output
```

Figure 5. 7 pytorch implementation of model2 (Actual NVIDIA model) [35]

Table 5.1 shown below contains values of means square error of model1 and model2 for all 22 epochs. Finally each model overwrites all parameter with last epoch values which is trained. Error shows slight difference in the value of steering angle predicted compared to input image steering angle for training and validation data set. This mean square error effected the autonomous car movement slightly in terms of steering angle decisions. Simple abrupt movement produced during simulation.

For model 1 The Mean square error for each(Total 22) epoch is shown in Table 5.1, it was around 0.04 average value.

For model 2 The Mean square error for each(Total 22) epoch is shown in Table 5.1, it was around 0.025 average value.

Mean square error means squared difference between CNN(neural network predicted) steering angle and input steering angle(The steering angle of input image after data pre-processing process)

Model1's training time via google colab is less around 45 minutes but mean square error is little bit high compared to actual NVIDIA model(model2).Model2 is NVIDIA neural network model with around 16 convolutional layers as shown in figure 5.6. It took around 1-1.5 hr training time via google colab, but received lesser mean square error compared to model1.

Epoch no	Training loss - Model1	validation loss - model1	Training loss - model2	Validation loss - model2
1	0.074	0.0193	0.081	0.185
2	0.059	0.162	0.063	0.178
3	0.058	0.2	0.049	0.151
4	0.053	0.16	0.051	0.175
5	0.043	0.121	0.049	0.142
6	0.05	0.123	0.049	0.149
7	0.05	0.142	0.041	0.168
8	0.044	0.105	0.044	0.144
9	0.046	0.196	0.046	0.108
10	0.045	0.136	0.039	0.138
11	0.046	0.122	0.042	0.155
12	0.041	0.162	0.032	0.125
13	0.043	0.137	0.039	0.131
14	0.046	0.117	0.038	0.094
15	0.041	0.151	0.029	0.098
16	0.044	0.134	0.034	0.104
17	0.043	0.139	0.031	0.088
18	0.042	0.148	0.023	0.075
19	0.041	0.143	0.017	0.067
20	0.043	0.105	0.017	0.068
21	0.043	0.11	0.022	0.079

Table 5. 1 shows training and validation errors of model1 and model2[35]

5.4 Torch framework libraries:

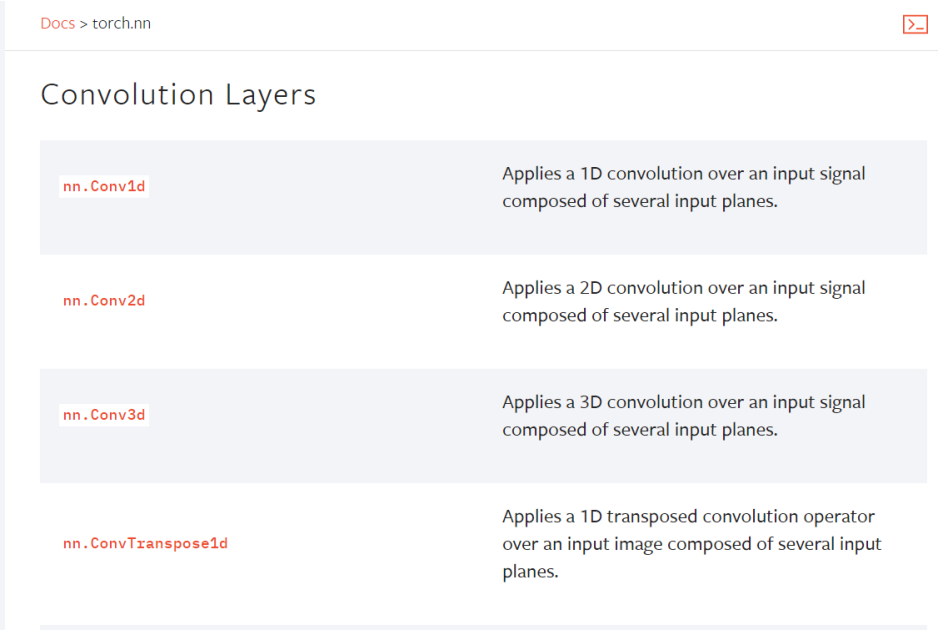


Figure 5. 8 torch neural network libraries [36]

Fig 5.8 shows various possible pytorch neural network libraries that cen be used for model implementation using python language.

Mean Square Error (MSE) is one of the most commonly used regression **loss** function. **MSE** is the sum of **squared** distances between our target value and predicted values.

For our case predicted value is output steering angle and target value is input pre-processed steering angle.

Adam optimizer used as Gradient descent optimization algorithms.[34]

Adaptive Moment Estimation (Adam) is a method that computes adaptive learning rates for each parameterin each convolutional layers. In addition to storing an exponentially decaying average of past squared gradients, Adam also keeps an exponentially decaying average of past gradients.

Parameter	Value
Batch_size	32
Epoch	22
Laerning rate	0.0001

Table 5. 2 Adam optimizer parameter

```
optimizer = optim.Adam(model.parameters(), lr=0.0001)
```

Figure 5. 9 Adam optimizer function

The defined 2 models used a single learning rate of 0.0001 for all layers.

All optimizers implement a `step()` method, that updates the parameters. It can be used in the fgiven way.

Optimizer.step()

This is a simplified version supported by most optimizers. The function can be called once the gradients are computed using e.g., `backward()`.

When we are calling. step function it updates the parameters in the optimizer.

Below figure 5.12 shows the pytorch way to store a trained model(example shown is for model2) in to an .h5 file for further usage.

```
In [28]: state = {
             'model': model.module if device == 'cuda' else model,
             }

         torch.save(state, 'model_Testmodel1_NVIDIA.h5')
```

Figure 5. 10 Save model to .h5 file

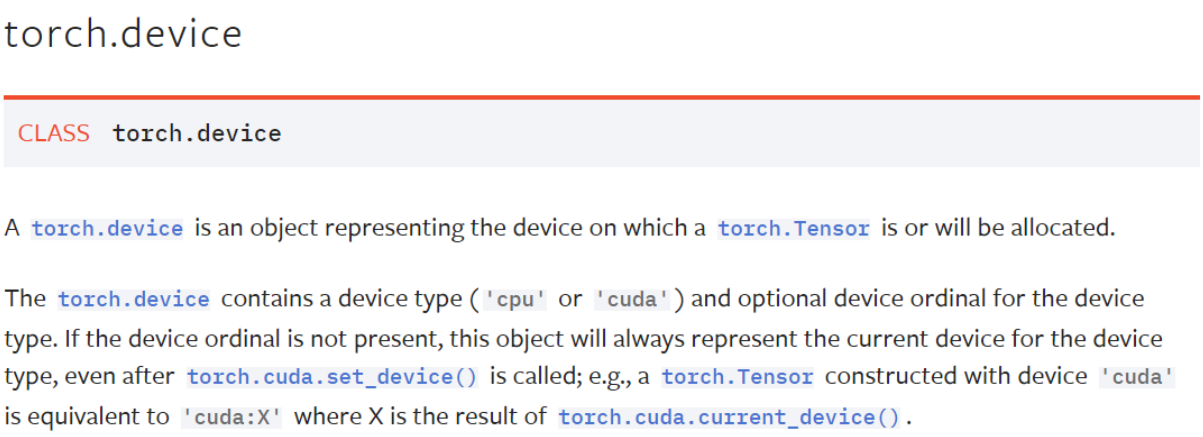


Figure 5. 11 torch device[38]

Finally save the trained models in to .h5 files for both model1 and model2 and downloaded to local (if it is trained via google Colab).

Finally save the trained model weights in to .h5 file. An H5 file is a data file format saved in the Hierarchical Data Format (HDF). It contains multidimensional arrays of data which we are saving.

For model1 trained model saved in to **model1_Modified-Nvidia.h5** file and for model2 trained model saved in to an h5 file named **model2_Actual-Nvidia.h5**. Both h5 files are downloaded in to local machine/workspace for simulation testing.

CHAPTER 6

TRAINED MODEL'S LOCAL DEPLOYMENT

Neural network models are trained and saved in to respective .h5 files as per chapter 5. Downloaded the trained model .h5 files(for mode1 - **model1_Modified-Nvidia.h5 and for** model2 **- model2_Actual-Nvidia.h5**) which is generated after successful training, in to local machine.

Virtual environment:

Python virtual environments main intension is to create an isolated environment for Python based projects. This means that each project can have its own dependencies in their environment.

Steps to create virtual environment:

- conda create -name venv
- activate venv
- install all dependencies inside that
- list of dependencies includes
- (venv) (base) C:\Users\Wafa\Sem4 Project\DeepLearningForSelfDrivingCars-master>pip list
  - Package Version
  - ---------- ---------
  - absl-py 0.12.0
  - astunparse 1.6.3
  - bidict 0.21.2
  - cachetools 4.2.2
  - certifi 2020.12.5
  - chardet 4.0.0
  - click 7.1.2
  - dnspython 1.16.0
  - eventlet 0.30.2
  - Flask 1.1.2
  - flatbuffers 1.12
  - gast 0.3.3
  - google-auth 1.30.0
  - google-auth-oauthlib 0.4.4
  - google-pasta 0.2.0
  - greenlet 1.0.0
  - grpcio 1.32.0
  - h5py 2.10.0
  - idna 2.10
  - importlib-metadata 4.0.1
  - itsdangerous 1.1.0

- Jinja2 2.11.3
- Keras 2.4.3
- Keras-Preprocessing 1.1.2
- Markdown 3.3.4
- MarkupSafe 1.1.1
- numpy 1.19.5
- oauthlib 3.1.0
- opt-einsum 3.3.0
- Pillow 8.2.0
- pip 21.1.1
- protobuf 3.15.8
- pyasn1 0.4.8
- pyasn1-modules 0.2.8
- python-engineio 3.13.2
- python-socketio 4.6.1
- PyYAML 5.4.1
- requests 2.25.1
- requests-oauthlib 1.3.0
- rsa 4.7.2
- scipy 1.6.3
- setuptools 41.2.0
- six 1.15.0
- tensorboard 2.5.0
- tensorboard-data-server 0.6.0
- tensorboard-plugin-wit 1.8.0
- tensorflow 2.4.1
- tensorflow-estimator 2.4.0
- termcolor 1.1.0
- torch 1.8.1+cpu
- torchaudio 0.8.1
- torchvision 0.9.1+cpu
- typing-extensions 3.7.4.3
- urllib3 1.26.4
- websocket-client 0.58.0
- Werkzeug 1.0.1
- wheel 0.36.2
- wrapt 1.12.1
- zipp 3.4.1

6.1 Autonomous driver:

The image which we are collecting from the centre camera of autonomous car pre-processed (cropped to focus only on road track), and sent to the trained model as input. The trained model we are sending as one of the arguments in the python driving command.

Setting autonomous car speed as constant a value of 9.

```
@sio.on('telemetry')
def telemetry(sid, data):
    if data:
        # The current steering angle of the car
        steering_angle = float(data["steering_angle"])
        # The current throttle of the car
        throttle = float(data["throttle"])
        # The current speed of the car
        speed = float(data["speed"])
        # The current image from the center camera of the car
        imgString = data["image"]
        image = Image.open(BytesIO(base64.b64decode(imgString)))
        image_array = np.asarray(image)
        image = image_array[65:-25, :, :]
        image = transformations(image)
        image = torch.Tensor(image)
        print(image.shape)
        image = image.view(1, 3, 70, 320)
        image = Variable(image)
        print(image.shape)
        steering_angle = model(image).view(-1).data.numpy()[0]
        throttle = controller.update(float(speed))
        print(steering_angle, throttle)
        send_control(steering_angle, throttle)
```

Figure 6. 1 The driving function of autonomous car.

In the second part of fig 5.2, the training system where image obtained with autonomous car's centre camera is pre-processed and sending to trained model for to predict steering angle.

The predicted steering angle and throttle are sending as input to send_control function.

6.2 Creating a client instance (Trained model)

The Socket.IO protocol is event based. When a server wants to initiate communication with a client it *ejects* an event. Every events are associated with a name, and a list of inputs or arguments to be passed during communication. The client stores event handler functions with the **socketio.Client.event()** or **socketio.Client.on()** .

```
@sio.event
def message(data):
    print('I received a message!')

@sio.on('my message')
def on_message(data):
    print('I received a message!')
```

Figure 6. 2 Socket io synchronous mode [1]

In the first example the event name is visible in the definition or name of the handler function. By the same time event name and event handler function name can be different also.

```
86  @sio.on('connect')
87  def connect(sid, environ):
88      print("connect ", sid)
89      send_control(0, 0)
90
```

Figure 6. 3 Initiate/connect a server client communication

The connect() method depicts the connection with server

Once after successful connection, server assigns the client a unique session identifier or sid.

We can print the sid/session id and verify the session created, as below.

```
print ('my sid is', sio.sid)
```

For the first time connection establishment setting steering angle and throttle as 0.

And another thing for functional usage defined an argument parser function which defines what is my python execution function looks like.

```
if __name__ == '__main__':
    parser = argparse.ArgumentParser(description='Remote Driving')
    parser.add_argument(
        'model',
        type=str,
        help='Path to model h5 file. Model should be on the same path.'
    )
    parser.add_argument(
        'image_folder',
        type=str,
        nargs='?',
        default='',
        help='Path to image folder. This is where the images from the run will be saved.'
    )
    args = parser.parse_args()

```

Figure 6. 4 Argument parser function

The argument parser function defines parser arguments which can be added in python command for server client communication

Example usage: For model1 :**python drive.py model1_Modified-Nvidia.h5 datastore**

For model2: **python drive.py model2_Actual-Nvidia.h5 datastore**

Here drive.py is the main file which contains autonomous driving SIO establishment, argument parser functions etc. Second attribute is trained model file (for model1 **model1_Modified-Nvidia.h5** and for model2 - **model2_Actual-Nvidia.h5)** file which is our trained model, should be on the same directory. And third parameter is optional, we can define a folder where output images/images during autonomous driving gets stored.

6.3 Creating a server instance (Udacity self driving car simulator)

A Socket.IO server is an instance of class **socketio.Server**. This instance can be transformed into a standard WSGI application by wrapping it with the **socketio.WSGIApp** class:

WSGI: WSGI is the Web Server Gateway Interface. Is an methodology that explaines how a web server communicates with web applications, and how web applications can be chained together to process a request.

A **middleware** function is a function that gets executed for every incoming connection.

```
    # wrap Flask application with engineio's middleware
    app = socketio.Middleware(sio, app)
    print(app)
```

Figure 6. 5 server client connection establishment

Here sio is the client instance, app is the server instance. For each client instance there is different session id, that is going to pass to middle ware function.

The WSGI utility provides an easy way to start an event-based WSGI server.

To launch a wsgi server, simply create a socket and call `eventlet.wsgi.server()` with it:

```
from eventlet import wsgi
import eventlet

def hello_world(env, start_response):
    start_response('200 OK', [('Content-Type', 'text/plain')])
    return ['Hello, World!\r\n']

wsgi.server(eventlet.listen(('', 8090)), hello_world)
```

Figure 6. 6 eventlet wsgi server example usage [15]

There 4567 is the port number of Udacity self-driving car simulator.

```
# deploy as an eventlet WSGI server
eventlet.wsgi.server(eventlet.listen(('', 4567)), app)
```

Figure 6. 7 eventlet wsgi server actual usage for our testcase

Server(Udacity simulator) to client interaction (Trained model)

Socket.IO is a protocol that connects real-time two directional event-based communication between clients (for our case the trained model) and a server(Udacity self driving car simulator).

Version compatibility The Socket.IO protocol has been through a number of revisions or releases, and some of these introduced backward incompatible changes, which implies that the client and the server must use compatible versions for establish connection between them. For the Python client and server, the easiest way to make sure compatibility is to use the same version of this package for the client and the server as shown in the below table.The version compatibility chart given below shows versions of this package to versions of the JavaScript reference implementation and the versions of the Socket.IO and Engine.IO protocols.

JavaScript Socket.IO version	Socket.IO protocol revision	Engine.IO protocol revision	python-socketio version	python-engineio version
0.9.x	1, 2	1, 2	Not supported	Not supported
1.x and 2.x	3, 4	3	4.x	3.x
3.x and 4.x	5	4	5.x	4.x

Table 6. 1 Client and server compatibility check[37]

The table shown in 6.1 specifies the versions of socketio and engineio compatibilities which is the version to be matched in a specified environment ,then only server to client interaction establish.

I have used below version for python enginio and python socketio

python-engineio 3.13.2

python-socketio 4.6.1

CHAPTER 7

SIMULATION AND RESULTS

Change directory to your project directory through anaconda prompt, and activate virtual environment which we created for our project.

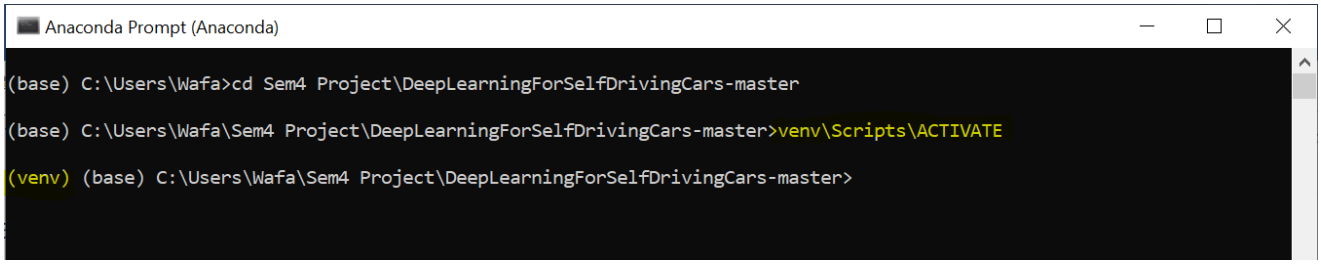


Figure 7. 1 Activate virtual environment

Open Udacity self-driving car beta version application in **autonomous mode**. Steering angle and speed of the car are set to zero.

Figure 7. 2 Udacity self-driving car simulator in autonomous mode

7.1 model1 simulation(Modified Nvidia model)

Execute the command **python drive.py model1_Modified-Nvidia.h5** on anaconda prompt.

Where **model1_Modified-Nvidia.h5** is the trained model saved file for model1. Given model summary print option in drive file so as to verify the model used.

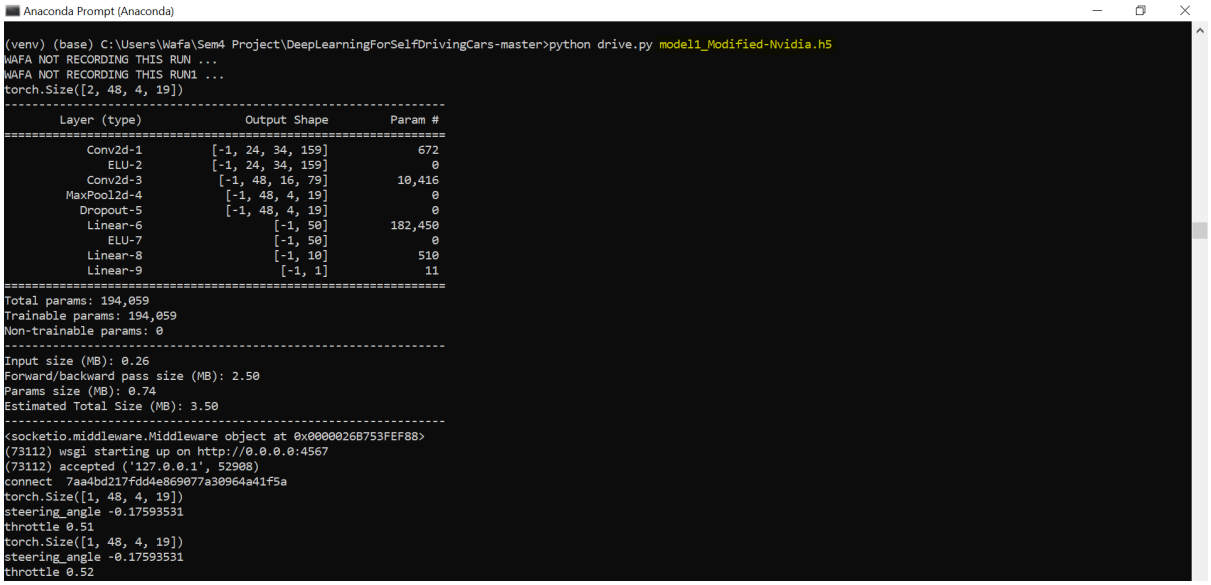


Figure 7. 3 Server to client connection established with model1

From figure 7.3 it is visible as:

Server instance created is:

socketio.middleware.Middleware object at 0x0000026B753FEF88

Client instance created is: **connect 7aa4bd217fdd4e869077a30964a41f5a**

wsgi starting up on http://0.0.0.0:4567

where 4567 is our simulator port number.

Figure 7. 4 autonomous driving steering angle, throttle, and image shape for each instance with model1

Figure 7. 5 Glimpse of autonomous mode driving capture image for model1

7.2 Model2 simulation(Actual Nvidia model)

Execute the command **python drive.py model2_Actual-Nvidia.h5** on anaconda prompt.

Where **model2_Actual-Nvidia.h5** is the trained model saved file for model2. Given model summary print option in drive file so as to verify the model used

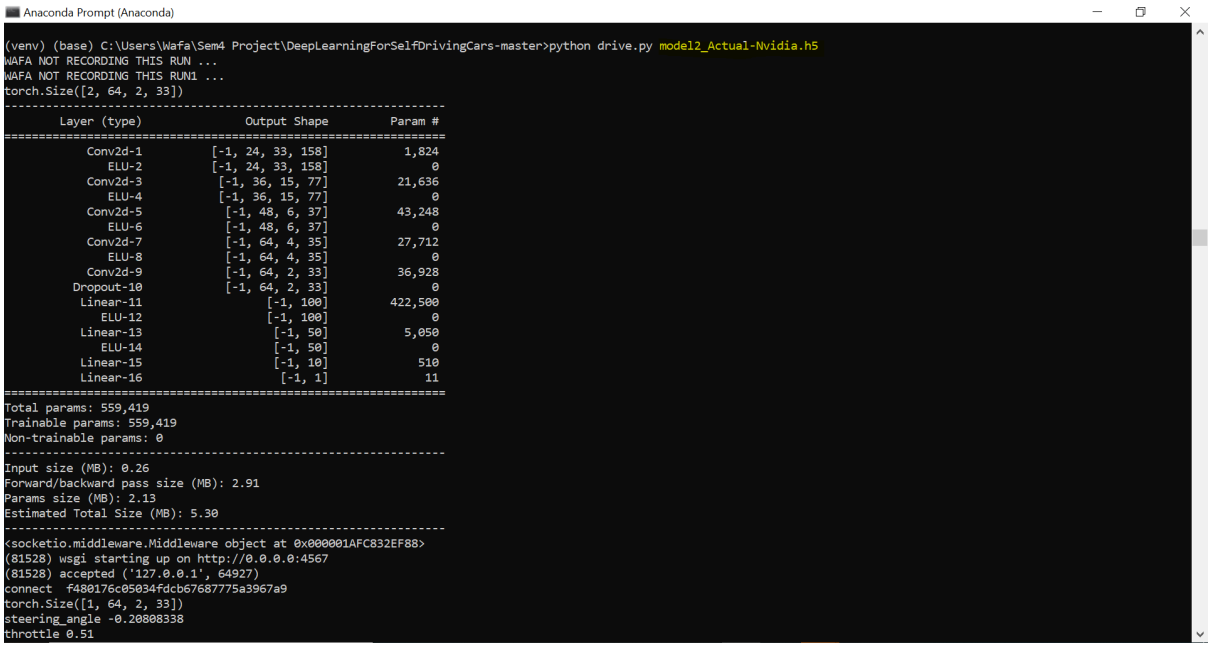


Figure 7. 6 Server to client connection established with model2

From figure 7.6 it is visible as :

Server instance created is :Simulator instance

socketio.middleware.Middleware object at 0x000001AFC832EF88

Client instance created is:Model2 instance

connect f480176c05034fdcb67687775a3967a9

wsgi starting up on http://0.0.0.0:4567

where 4567 is our simulator port number.

```
torch.Size([1, 64, 2, 33])
steering_angle -0.22420329
throttle 0.51
torch.Size([1, 64, 2, 33])
steering_angle -0.22420329
throttle 0.52
torch.Size([1, 64, 2, 33])
steering_angle -0.20585275
throttle 0.53
torch.Size([1, 64, 2, 33])
steering_angle -0.20461737
throttle 0.5163768
torch.Size([1, 64, 2, 33])
steering_angle -0.15606125
throttle 0.461062
torch.Size([1, 64, 2, 33])
steering_angle -0.15606125
throttle 0.4693272
torch.Size([1, 64, 2, 33])
steering_angle -0.17834772
throttle 0.416882
torch.Size([1, 64, 2, 33])
steering_angle -0.1852308
```

Figure 7. 7 autonomous driving steering angle, throttle, and image shape for each instance with model2

Figure 7. 8 Glimpse of autonomous mode driving capture image for model2

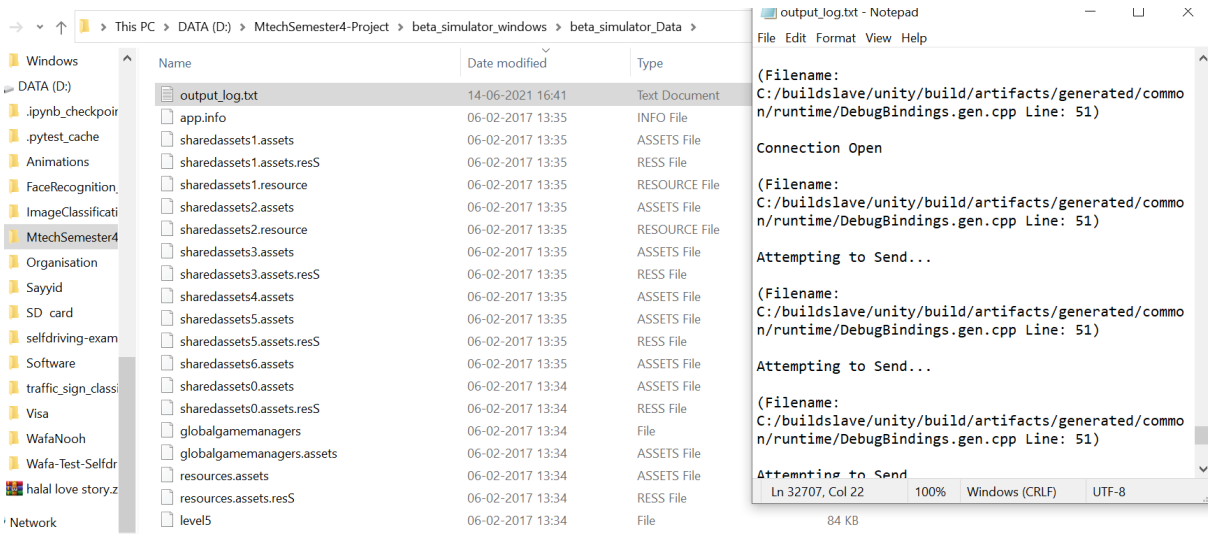


Figure 7. 9 Server side log.txt during execution

In this chapter included simulation results of 2 models separately. And figure 7.9 shows the server side logs view when successful connection established between Udacity simulator and model.

model 1	model2	Steering angle differnece between 2 nearby instance with model1	Steering angle differnece between 2 nearby instance with model2
steering_angle -0.17517355	steering_angle -0.20468177		
steering_angle -0.17517355	steering_angle -0.20468177	0	0
steering_angle -0.17523444	steering_angle -0.20476434	0.00006089	0.00008257
steering_angle -0.17517355	steering_angle -0.22420329	-0.00006089	0.01943895
steering_angle -0.15767907	steering_angle -0.16479248	-0.01749448	-0.05941081
steering_angle -0.15767907	steering_angle -0.16479248	0	0
steering_angle -0.23634721	steering_angle -0.26937395	0.07866814	0.10458147
steering_angle -0.20692784	steering_angle -0.27068183	-0.02941937	0.00130788
steering_angle -0.25332102	steering_angle -0.27068183	0.04639318	0
steering_angle -0.25332102	steering_angle -0.22769688	0	-0.04298495
steering_angle -0.30807513	steering_angle -0.24041672	0.05475411	0.01271984
steering_angle -0.25166547	steering_angle -0.24041673	-0.05640966	0.00000001
steering_angle -0.1377066	steering_angle -0.24041674	-0.11395887	0.00000001

Table 7. 1 Steering angle differences of model1 and model2 for few instances captured in autonomous mode.

In table 7.1 shows steering angle for few instances obtained in autonomous mode with the same track by both models(model1 – Modified NVIDIA model and model2 – Actual NVIDIA model) and the steering angle difference between nearby instances are given.

A smooth driver is capable to manage very low steering angle variations. While considering model1's(Modified NVIDIA model) steering angle difference between near by instances there is a small ad smooth gradual variation observed as per table 7.1 So model1 offers slightly smooth movement in autonomous mode compare to model2. Minute variations only ,but it is more recognisable during autonomous car running on track.

CHAPTER 8

CONCLUSION AND FUTURE WORK

CONCLUSION

An autonomous driving simulation using Udacity's self-driving car simulator is shown in this thesis. The images captured by left, right and centre cameras and steering angle during training mode of simulator is pre-processed and send to model for training. There are 2 models considered for training and simulation process. Trained models are saved and used for autonomous driving simulation. During autonomous driving the images collected by centre camera of the car pre-processed and send to model as input. Model predicts the steering angle that passed to the car in the autonomous mode after successful server client connection establishment. Speed set as constant throughout autonomous journey to monitor the movement properly. From the 2 model's(model1 – modified NVIDIA model and model2 – Actual NVIDIA model) the first model was comparatively good than NVIDIA model, since car was always focussing centre. Convolutional neural network model training time was less for model1 compared to model2.

FUTURE WORK

Traffic sign recognition and classification on Udacity self driving car track ,and implementation of the same with necessary control actions a future scope of this thesis.

Using the trained road traffic sign model, it is possible for to check and make decisions on Udacity simulator track during autonomous driving. It can be achieved by editing / creating a track with relevant road traffic signs ,deploy the trained model in to local, and test the model with simulators track images while running.

Since autonomous driving is a current trending thing, many other future studies can be emulated with same simulator for study purpose.

BIBLIOGRAPHY

[1] "The Socket.IO Server — python-socketio documentation," Readthedocs.io. [Online]. Available: https://python-socketio.readthedocs.io/en/latest/server.html.

[2] J. Kitson and J. Kitson, "Installing Tensorflow with CUDA, cuDNN and GPU support on Windows 10," Towards Data Science, 03-Apr-2019. [Online]. Available: https://towardsdatascience.com/installing-tensorflow-with-cuda-cudnn-and-gpu-support-on-windows-10-60693e46e781.

[3] "Self-Driving with Valohai," Valohai.com. [Online]. Available: https://valohai.com/blog/self-driving-with-valohai/.

[4] "tf.concat," Tensorflow.org. [Online]. Available: https://www.tensorflow.org/api_docs/python/tf/concat.

[5] "No title," E2eml.school. [Online]. Available: https://e2eml.school/how_convolutional_neural_networks_work.html.

[6] soumya, "Lane line detection," Kaggle.com, 02-Sep-2019. [Online]. Available: https://www.kaggle.com/soumya044/lane-line-detection.

[7] "Before you continue to YouTube," Youtube.com. [Online]. Available: https://www.youtube.com/watch?v=nHk53YnrE5k.

[8] "Save and load models," Tensorflow.org. [Online]. Available: https://www.tensorflow.org/tutorials/keras/save_and_load.

[9] J. Kocić, N. Jovičić, and V. Drndarević, "An end-to-end deep neural network for autonomous driving designed for embedded automotive platforms," Sensors (Basel), vol. 19, no. 9, p. 2064, 2019.

[10] "API Reference — python-socketio documentation," Readthedocs.io. [Online]. Available: https://python-socketio.readthedocs.io/en/latest/api.html.

[11] "tf.compat.v1.train.Saver," Tensorflow.org. [Online]. Available: https://www.tensorflow.org/api_docs/python/tf/compat/v1/train/Saver.

[12] CARLA Team, "CARLA," Carla.org. [Online]. Available: https://carla.org/.

[13] R. Khandelwal, "Convolutional neural network: Feature map and filter visualization," Towards Data Science, 18-May-2020. [Online]. Available: https://towardsdatascience.com/convolutional-neural-network-feature-map-and-filter-visualization-f75012a5a49c.

[14] "Python OpenCV – cv2.flip() method - GeeksforGeeks," Geeksforgeeks.org, 23-Jun-2020. [Online]. Available: https://www.geeksforgeeks.org/python-opencv-cv2-flip-method/.

[15] "wsgi – WSGI server — Eventlet 0.31.0 documentation," Eventlet.net. [Online]. Available: https://eventlet.net/doc/modules/wsgi.html.

[16] "Downloads/DFGTSD - visual cognitive systems laboratory," Vicos.si. [Online]. Available: https://www.vicos.si/Downloads/DFGTSD.

[17] "German traffic sign benchmarks," Rub.de. [Online]. Available: https://benchmark.ini.rub.de/?section=gtsrb&subsection=dataset.

[18] "Sklearn.Model_selection.Train_test_split — scikit-learn 0.24.2 documentation," Scikit-learn.org. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html.

[19] "Python PIL," Geeksforgeeks.org, 29-Jul-2019. [Online]. Available: https://www.geeksforgeeks.org/python-pil-imagedraw-draw-rectangle/.

[20] "numpy.random.rand() in Python - GeeksforGeeks," Geeksforgeeks.org, 24-Aug-2017. [Online]. Available: https://www.geeksforgeeks.org/numpy-random-rand-python/.

[21] L. H. Meftah and R. Braham, “A virtual simulation environment using deep learning for autonomous vehicles obstacle avoidance,” in 2020 IEEE International Conference on Intelligence and Security Informatics (ISI), 2020, pp. 1–7.
[22] J. Zhang, H. Huang, and Y. Zhang, “A convolutional neural network method for self-driving cars,” in 2020 Australian and New Zealand Control Conference (ANZCC), 2020, pp. 184–187.
[23] “Driving Simulation for autonomous driving, ADAS, vehicle dynamics and motorsport,” Rfpro.com. [Online]. Available: https://www.rfpro.com/.
[24] “Self-Driving Car Simulator,” Wipro.com. [Online]. Available: https://www.wipro.com/engineeringNXT/self-driving-vehicle-in-a-box-a-global-scale-simulator-for-autonomous-vehicles/.
[25] “Real-Time simulation │ Real-Time Solutions │ OPAL-RT,” Opal-rt.com, 02-Mar-2016. [Online]. Available: https://www.opal-rt.com/.
[26] “Traffic sign recogntion database,” Ia.ac.cn. [Online]. Available: http://www.nlpr.ia.ac.cn/pal/trafficdata/recognition.html.
[27] Wikipedia contributors, “Artificial intelligence,” Wikipedia, The Free Encyclopedia, 30-May-2021. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Artificial_intelligence&oldid=1025930579.
[28] Wikipedia contributors, “Machine learning,” Wikipedia, The Free Encyclopedia, 18-Jun-2021. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Machine_learning&oldid=1029169603.
[29] Wikipedia contributors, “Deep learning,” Wikipedia, The Free Encyclopedia, 16-Jun-2021. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Deep_learning&oldid=1028832472.
[30] Unity Technologies, “Unity - Unity,” Unity.com. [Online]. Available: https://unity.com/.
[31] “numpy.random.rand() in Python - GeeksforGeeks,” Geeksforgeeks.org, 24-Aug-2017. [Online]. Available: https://www.geeksforgeeks.org/numpy-random-rand-python/.
[32] “NumPy quickstart — NumPy v1.20 Manual,” Numpy.org. [Online]. Available: https://numpy.org/doc/stable/user/quickstart.html.
[33] “torch.optim — PyTorch 1.9.0 documentation,” Pytorch.org. [Online]. Available: https://pytorch.org/docs/stable/optim.html.
[34] M. G. Bechtel, E. Mcellhiney, M. Kim, and H. Yun, “DeepPicar: A low-cost deep neural network-based autonomous car,” in *2018 IEEE 24th International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, 2018.
[35] Autonomous-Vehicle-Design. . https://github.com/WafaAbdulla-tech/Autonomous-Vehicle-Design
[36] “torch.nn — PyTorch 1.9.0 documentation,” *Pytorch.org*. [Online]. Available: https://pytorch.org/docs/stable/nn.html.
[37] “Getting Started — python-socketio documentation,” *Readthedocs.io*. [Online]. Available: https://python-socketio.readthedocs.io/en/latest/intro.html.
[38] “Tensor Attributes — PyTorch 1.9.0 documentation,” *Pytorch.org*. [Online]. Available: https://pytorch.org/docs/stable/tensor_attributes.html.

Thesis Report

ORIGINALITY REPORT

16%	11%	5%	11%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	Submitted to SASTRA University Student Paper	1%
2	python-socketio.readthedocs.io Internet Source	1%
3	www.wipro.com Internet Source	1%
4	Submitted to Florida College Student Paper	1%
5	Submitted to City University Student Paper	1%
6	pypi.org Internet Source	1%
7	pytorch.org Internet Source	1%
8	iq.opengenus.org Internet Source	1%
9	towardsdatascience.com Internet Source	<1%

10	wikimili.com Internet Source	<1 %
11	Submitted to tsi Student Paper	<1 %
12	arxiv.org Internet Source	<1 %
13	www.rfpro.com Internet Source	<1 %
14	Submitted to The Robert Gordon University Student Paper	<1 %
15	mafiadoc.com Internet Source	<1 %
16	Submitted to National University of Ireland, Maynooth Student Paper	<1 %
17	doras.dcu.ie Internet Source	<1 %
18	machinelearningmastery.com Internet Source	<1 %
19	Submitted to Queen Mary and Westfield College Student Paper	<1 %
20	Submitted to The Scientific & Technological Research Council of Turkey (TUBITAK) Student Paper	<1 %

21	en.wikipedia.org Internet Source	<1 %
22	Submitted to Sogang University Student Paper	<1 %
23	fileinfo.com Internet Source	<1 %
24	www.cours-gratuit.com Internet Source	<1 %
25	Sangita Lade, Parth Shrivastav, Saurabh Waghmare, Sudarshan Hon, Sushil Waghmode, Shubham Teli. "Simulation of Self Driving Car Using Deep Learning", 2021 International Conference on Emerging Smart Computing and Informatics (ESCI), 2021 Publication	<1 %
26	backend.orbit.dtu.dk Internet Source	<1 %
27	codelabs.developers.google.com Internet Source	<1 %
28	realpython.com Internet Source	<1 %
29	ar.player.fm Internet Source	<1 %
30	Submitted to Ain Shams University Student Paper	<1 %

31	I Sonata, Y Heryadi, L Lukas, A Wibowo. "Autonomous car using CNN deep learning algorithm", Journal of Physics: Conference Series, 2021 Publication	<1 %
32	en.unionpedia.org Internet Source	<1 %
33	Submitted to Shri Guru Gobind Singhji Institute of Engineering and Technology Student Paper	<1 %
34	medium.com Internet Source	<1 %
35	"Machine Learning and Intelligent Communications", Springer Science and Business Media LLC, 2018 Publication	<1 %
36	Submitted to Kaunas University of Technology Student Paper	<1 %
37	Submitted to Birla Institute of Technology and Science Pilani Student Paper	<1 %
38	Submitted to Georgia Institute of Technology Main Campus Student Paper	<1 %
39	researcharchive.lincoln.ac.nz Internet Source	<1 %

40	yunusmuhammad007.medium.com Internet Source	<1 %
41	"Intelligent Computing, Information and Control Systems", Springer Science and Business Media LLC, 2020 Publication	<1 %
42	Submitted to Benedictine University Student Paper	<1 %
43	lib.buet.ac.bd:8080 Internet Source	<1 %
44	Submitted to California State University, Sacramento Student Paper	<1 %
45	Submitted to Rajasthan Technical University, Kota Student Paper	<1 %
46	Huihui Wu, Deyun Lv, Tengxiang Cui, Gang Hou, Masahiko Watanabe, Weiqiang Kong. "SDLV: Verification of Steering Angle Safety for Self-Driving Cars", Formal Aspects of Computing, 2021 Publication	<1 %
47	Submitted to University of Southern California Student Paper	<1 %
48	Wen-Yen Lin, Wang-Hsin Hsu, Yi-Yuan Chiang. "A Combination of Feedback Control and	<1 %

	Vision-Based Deep Learning Mechanism for Guiding Self-Driving Cars", 2018 IEEE International Conference on Artificial Intelligence and Virtual Reality (AIVR), 2018 Publication	
49	bmcbioinformatics.biomedcentral.com Internet Source	<1 %
50	campar.in.tum.de Internet Source	<1 %
51	documents.mx Internet Source	<1 %
52	www.da6nci.com Internet Source	<1 %

Exclude quotes	Off	Exclude matches	Off
Exclude bibliography	On		