Simulation of Self Driving Car using CNN

Geetanjali Bhosale[1], Bhumika Bhagia[1], Akanksha Hatle[1] and Dr. Mrudul Dixit[2]
[1]*Student*
[2] *Assistant Professor*
Department of Electronics and Telecommunication,
MKSSS's Cummins College of Engineering for Women, Pune, India.

***Abstract*:** *A self-driving car is a step towards making our transportation intelligent. Such a car senses its environment and makes decisions independently without any human intervention. Simulation is the first step in evaluating any idea before it can be implemented in the real world. This paper deals with simulating an autonomous car in the Udacity Simulator. A model has been trained using CNN algorithm which enables the car to steer autonomously along the track. A regression problem has been solved in this paper where the CNN model is fed with a labeled dataset which consists of images and corresponding steering angle values from the simulator. The trained model is thus able to predict steering angle values from the new images which the cameras on the car capture while driving in autonomous mode.*

Keywords: **Autonomous car, CNN, Regression, Simulation, Udacity Simulator, Flask and SocketIO.**

1. Introduction

A self-driving car is capable of performing all the driving tasks autonomously by sensing its environment. It is able to do so with no human interaction or attention at any time during the drive. The benefits of these cars are endless. Availability of autonomous cars can improve mobility of people, especially senior citizens and people with disabilities who are often dependent on others to move around. They are a great solution for an intelligent transport system which will be instrumental in the development of smart cities. They have a high potential to reduce congestion on road and inturn reduce the CO_2 emissions worldwide. A lot of parking lots will be free for other uses like community centers , parks etc.[1].

Making transportation automated can help reduce crashes. It is observed that over 90 percent of road crashes occur due to human error. Automated vehicles can reduce the risk of accidents due to dangerous driver behavior like speeding, drunken driving and distraction.

These cars rely on sensors like video cameras to perceive the road and detect other vehicles, traffic signs, pedestrians etc. on the road. Sophisticated algorithms like machine

learning algorithms process all this sensory input and help the car take decisions and navigate on the road independently.[1]

Deep learning algorithms like convolutional neural networks (CNN) is used by these systems to recognize different parts of the road and in turn help make appropriate decisions. CNN can easily model spatial information like images and are excellent in extracting features from them. Each layer in CNN can capture a different pattern. For example, initial layers can capture edges of the road while deep layers are capable of capturing more complex features like trees, vehicles etc.[2]

Our goal is to simulate a self-driving car which is successfully able to traverse the track provided by the Udacity simulator without any human intervention. The simulation provides fast and less expensive insights about the model before one can go for the physical prototyping stage. This accelerates the time to market and reduces cost.

For this project the car is first driven manually in the training mode of the simulator using arrow keys. The data is collected in csv format containing path to images and the corresponding steering angles. This information is preprocessed and fed to the model. The model is trained in such a way that it is able to predict steering angles when it sees new data. Finally the model is tested out in the autonomous mode of the simulator.

2. System Overview

Figure 1 shows the steps involved in training a model capable of autonomously driving a car in the simulator.

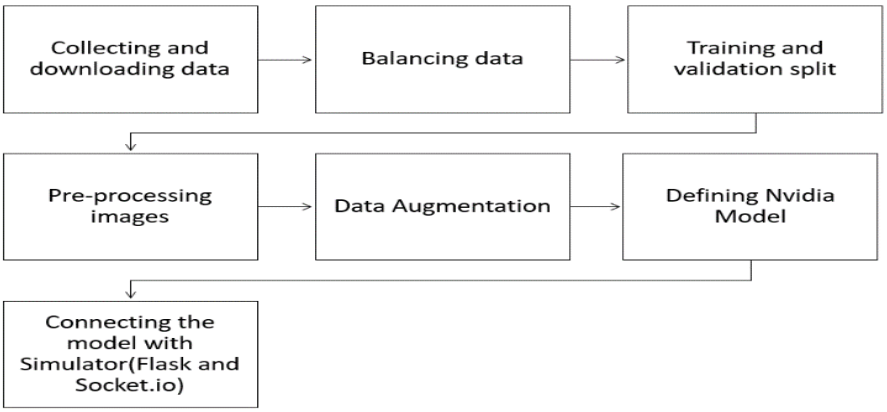

Figure 1. Steps Involved In Training A Model For A Self Driving Car

2.1. Collection and downloading data:

The data has been collected in the form of images, by running the car on Udacity Simulator.

Figure 2 shows the images captured by the simulator

A CSV file is generated by the simulator which contains information like image paths, steering angles and throttle values that is required for training the model.

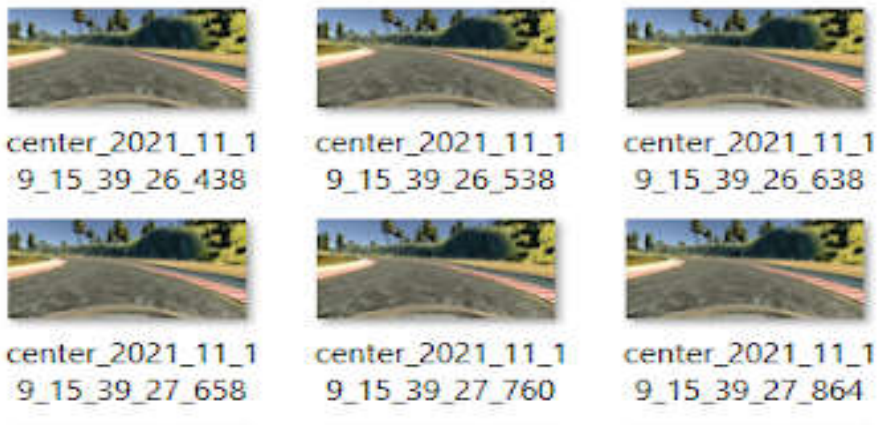

Figure 2. Images Captured By The Simulator

2.2. Balancing Data:

Data collected contains more zero angles as most of the time the car was driven in the center of the track. If the model is trained based on this data, this could bias it towards driving in the center of the road. Such a model will be befuddled when it encounters sharp turns.

To overcome this, all samples after a certain threshold are rejected. This creates a data set having a comparable amount of left, right and center steering angle values, ensuring that the model is not heavily skewed towards predicting the center angle. The data will be skewed towards the center even after balancing but as our initial goal is to drive the car in the center of the road, it is justified.

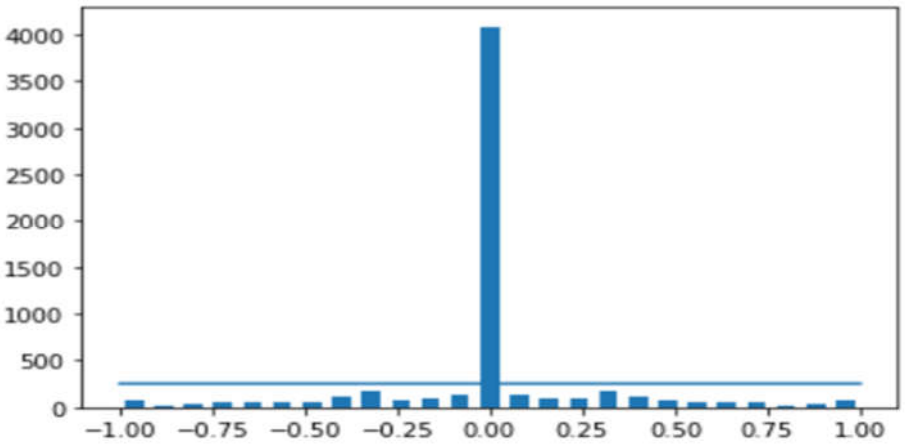

Figure 3. Unbalanced Steering angle values

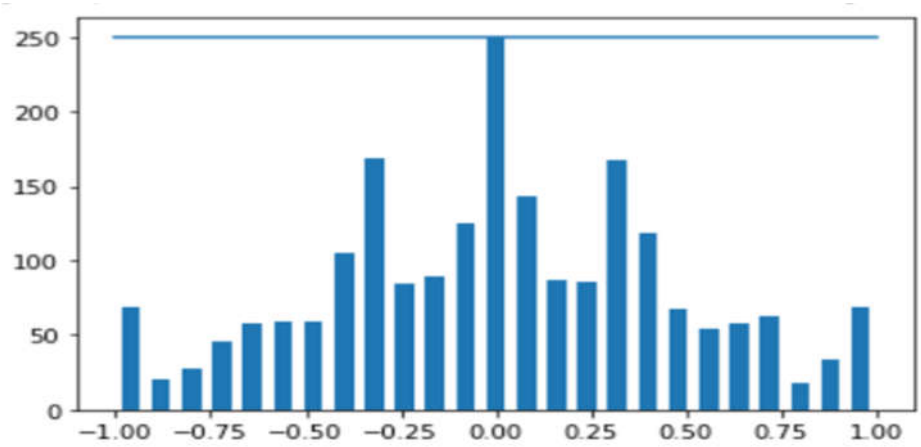

Figure 4. Balanced Steering angle values

Figure 3 and Figure 4 represent that by selecting an appropriate threshold the data of steering angles becomes balanced.

2.3. Preprocessing of Images

To ameliorate the performance of the model and to get optimized results preprocessing of images is performed. This step helps clean the data, and gets rid of unwanted features. Different preprocessing techniques applied on the data collected in the simulator are:

1. Crop image: The image contains some features(trees, mountains, hood of the car) which are not relevant for the car to determine the steering angles, so it is better to omit these. This can be done by using the technique of numpy array slicing. Image is represented as a 3-dimensional array where [height, width, channel].Here only the height dimension has been sliced.
2. Modifying Color Space: RGB to YUV Conversion: In YUV, Y denotes brightness of image and UV denotes the chrominance which means adding color to the image, this color space usually used for color image processing.
3. Gaussian Blur: This technique helps to remove noise from the image, and hence smoothens it.
4. Image resizing: By reducing the size of the image the computation complexity is reduced.
5. Normalization: This technique speeds up the learning process. In this, the pixel values are converted to a range [0,1]

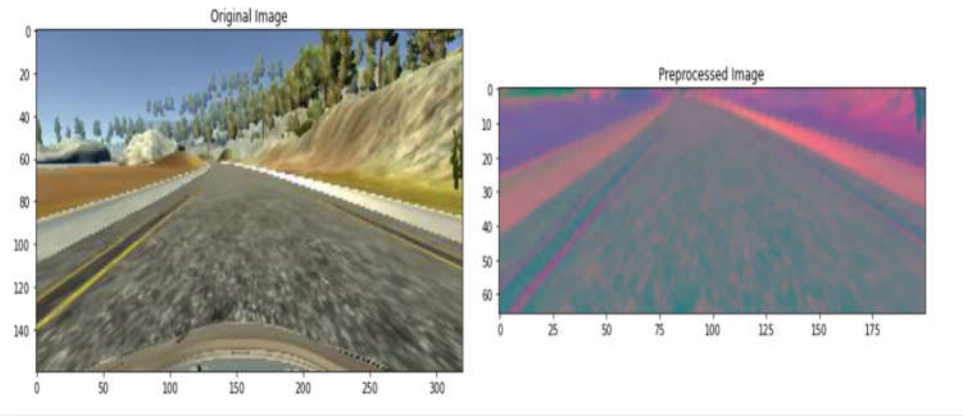

Figure 5. Preprocessing Of Images

Figure 5 is the result of applying all the preprocessing techniques to the image.The first image is the original image and the second one is the preprocessed image

2.4. Augmentation of Dataset

In order to include variety in the dataset various augmentation techniques are applied to the data captured. These applied transformations to the current set of data help the model train efficiently as it has more data to work with. Two categories of augmentation:

1. Spatial Augmentation: Different techniques of spatial augmentation:

- Zoom: This technique is applied in order to get a potentially closer look at some of the features of the image. Affine function from Imgaug library is used to perform this task. Affine function mainly deals with transformations that preserve straight lines and planes, zooming of an image fits in this category.

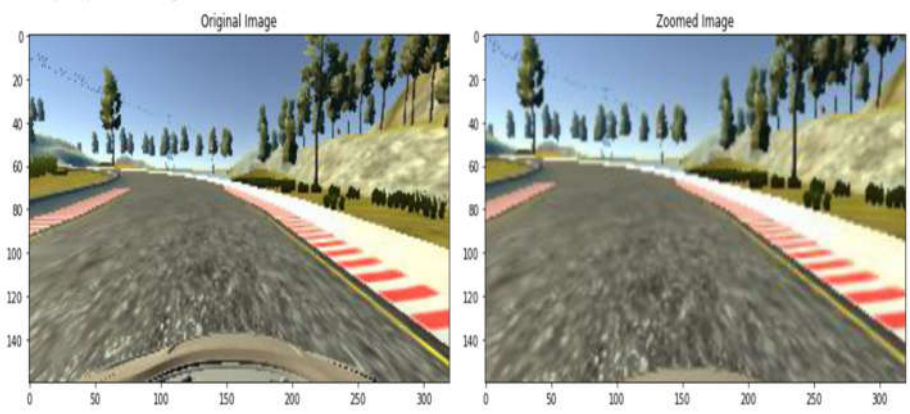

Figure 6. Zooming Augmentation Technique

Figure 6 shows the zoomed image which is obtained by applying zoom augmentation technique to the original image.

- Panning: In this technique horizontal and vertical translations are applied to images. Affine function mainly deals with transformations that preserve straight lines and planes, panning of an image fits in this category.

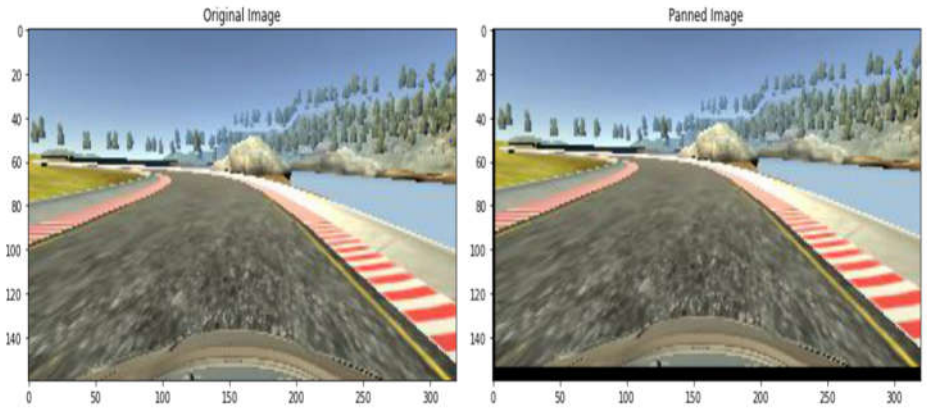

Figure 7. Panning Augmentation Technique

Figure 7 shows the panned image which is obtained by applying panning augmentation technique to original image

2. Pixel Augmentation: Altering brightness of image: Images of various intensities are generated. Multiply function is used to perform this task, thus any pixel intensity multiplied by a value less than 1 will become dark and equal to or more than one will become bright.

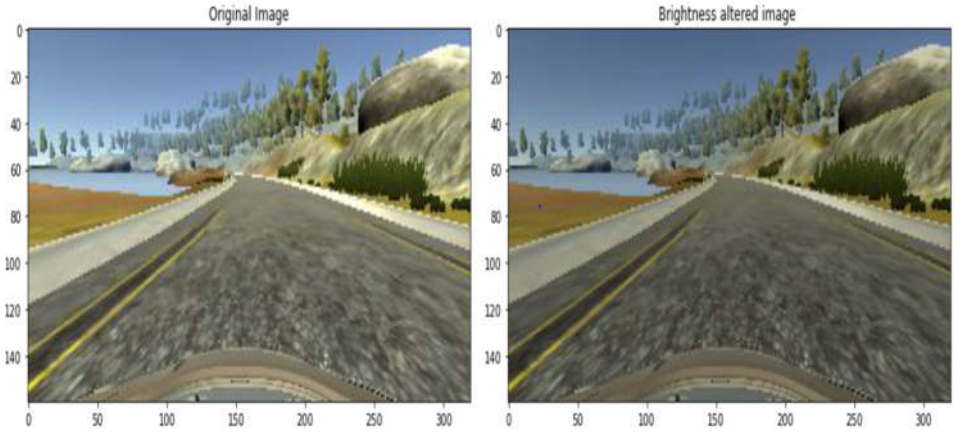

Figure 8. Pixel Augmentation

Figure 8 shows the image with altered brightness which is obtained by applying an altering brightness augmentation technique to the original image.

3. Flipping Augmentation Technique: The data has been already balanced well for both right and left steering angles in the data balancing step but to provide additional balancing to the dataset flipping augmentation technique. Flip function from CV2 library is used to perform this task. The parameters for flip function are flip(img,0/1/-1) where 0 indicates vertical flip,1 indicates horizontal flip and -1 indicates vertical and horizontal flip.

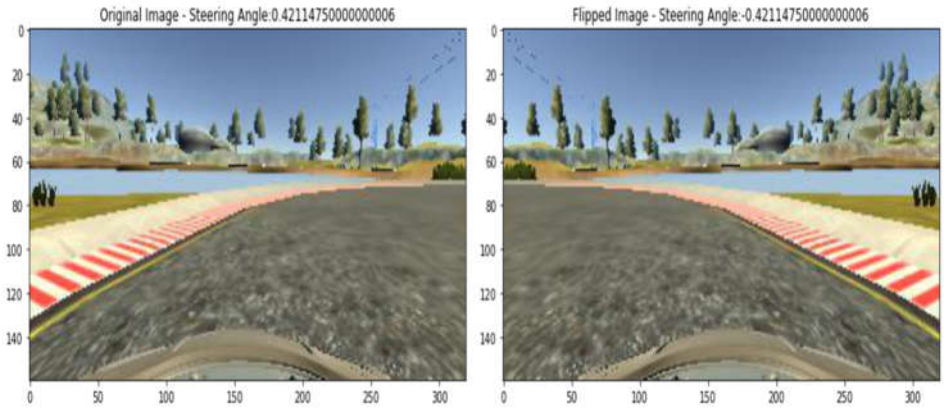

Figure 9. Flipping Augmentation Technique

Figure 9 shows the flipped image which is obtained by applying flipping augmentation technique to the original image

2.5. Defining and Training the CNN Model

This is a Regression problem where the model is given the labeled data with images and corresponding steering angles as input and it has to predict steering angle values as output for new unlabeled data. The network is trained to minimize loss between actual and predicted output. A

CNN network is designed having the following architecture:

- The input passes through 5 convolutional layers which perform the task of feature extraction.
- The number of filters are increased in succeeding layers to capture complex features .
- Next comes the flatten layer that converts the data into a 1D array for inputting it to the next layer.
- 3 fully connected layers come after flatten layer and final one is the output layer.

Figure 10 shows the pictorial representation of the designed CNN model.
The CNN model is developed with help of keras library in python. From keras sequential model is used which arranges the keras layers in a consecutive manner so that the information flows from one layer to another in that order until the data finally reaches the output layer.

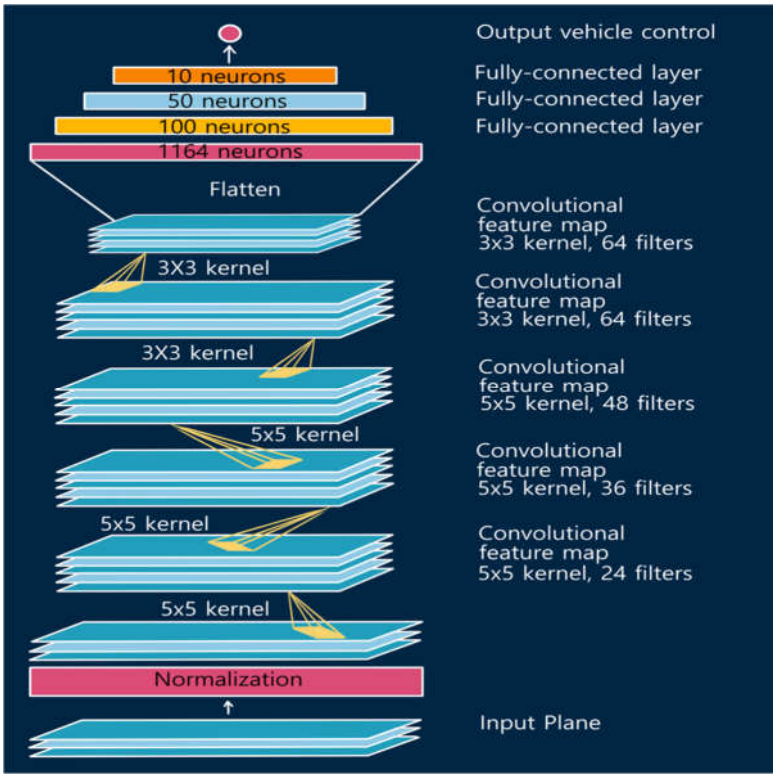

Figure 10. CNN Model Architecture

2.5.1. Activation Function: It is used to determine the output of a neuron .

For Hidden layers: Typically, a differentiable and a non linear activation function is used. This enables the model to comprehend functions with high complexity.

$$\square\square\square(\square) = \{\square \quad \square\square\ \square > 0 \ , \ \square(\square^{\square} - 1) \quad \square\square\ \square < 0\}$$

(1)

α is a positive constant

Exponential linear unit (Elu) function is used.
It is more or less immune to the vanishing gradient problem and also produces accurate results. It is immune to the dead relu problem.

For output layer: Linear activation function f(x)=x is used. The output layer directly outputs the prediction.

2.5.2. Optimizer and loss function:
Adam optimizer from keras is used. It implements an optimal gradient descent algorithm to correct the weights of the network and in turn helps the model train effectively by minimizing the loss. For optimization error should be computed.

The loss function used is Mean squared error loss (MSE).It is calculated by averaging the squared difference between the actual and the prediction made by the model.

2.5.3. Batch generator and fit_generator:

The amount of information obtained from the simulator was not sufficient to train the model .This was the reason to go for augmentation techniques. Augmenting the entire dataset at once and feeding it to the model will lead to huge memory requirements and will in turn hamper the performance.

Defining a batch generator solves this problem. The batch generator is like an iterator and takes as input a batch size. The generator randomly selects the training images and applies random augmentation techniques to them and thus generates new images according to the batch size. The fit_generator function accepts the batch of data generated, performs backpropagation and corrects the weights in our models. The process is repeated till desired performance is achieved.

2.6. Connecting model to Simulator

The trained model is used to drive the car autonomously in the simulator. To connect this model to the simulator Flask and SocketIO is required.

Flask is useful for creating servers in a quick and easy manner. It is a python microframework whose aim is to have a simple and extensible core [5].

SocketIO gives Flask applications access to low latency bidirectional communication between the client and the server.

It is a way to have sockets open between a client and a server to send messages in real time.[6] It is not the typical request - response model of the web. Here there is an open connection that always exists between the client and the server which keeps listening for new events and updates.

Figure 11 shows the persistent two way communication link established between the client and the server enabling them to communicate in real time.

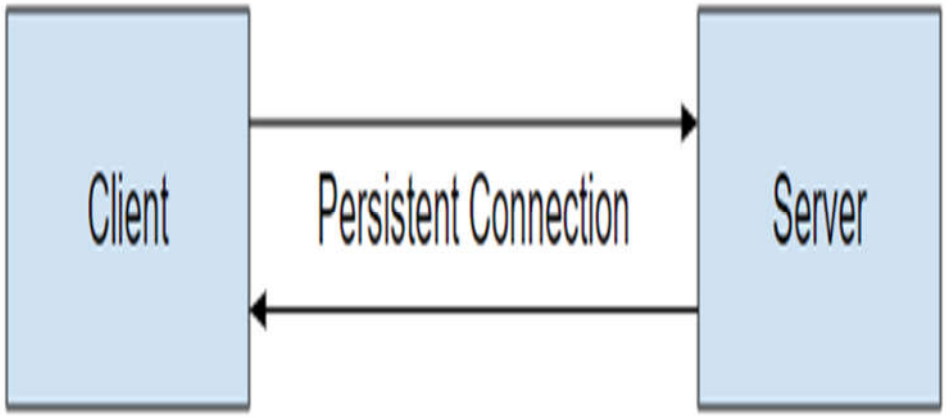

Figure 11. Bidirectional client - server communication

A python Anaconda environment is created and all the necessary packages like tensorflow, keras, flask, socketIO, pillow and eventlet are installed. The socket server is linked with the flask python web application. Eventlet library is used to open up a listening socket and send the request to the web application.

The data received from the simulator contains the current captured image along with other telemetry data such as throttle value, speed and steering angle value. The image is preprocessed before it is given to the model. The model takes this preprocessed image and predicts the corresponding steering angle value.

The current speed and a predetermined speed limit is used to estimate the throttle value so that the speed can be controlled. The new steering angle and throttle values are sent to the simulator and used to drive the car. Again the simulator sends back the current image and the process continues. In this way the car is able to drive autonomously.

The block diagram in figure 12 shows how the car is able to drive autonomously in the simulator.

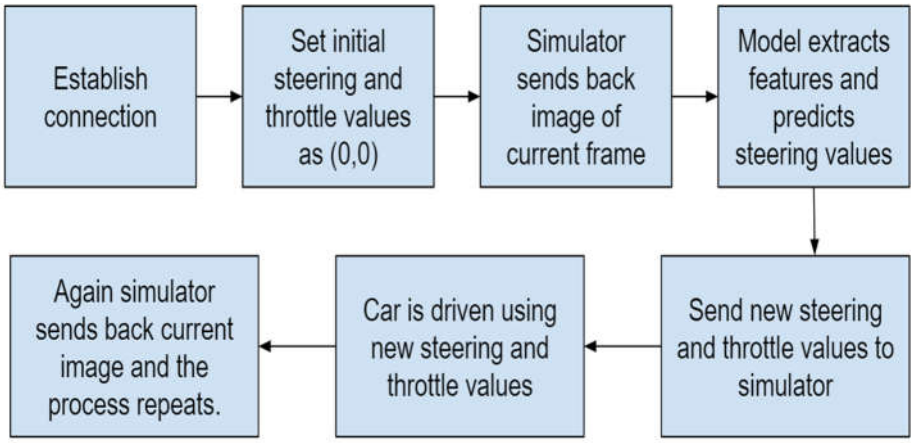

Figure 12. Connecting model to simulator

The steering angle, throttle value and speed of the car is regulated so that the car is able to make gradual and sharp turns and maintain its speed below a certain speed limit.

3. Results

To evaluate the performance of the model, the training loss and validation loss vs number of epochs is plotted.

To get a good result, the loss value should decrease significantly with each epoch and shouldn't be a constant value. From the graph it can be observed that this is the case for our model.

To avoid overfitting, there should not be a large gap between the training and validation loss plots.

Figure 13 shows the graph of training and validation loss. It can be seen that the gap reduces with each epoch and is minimal at the last one.

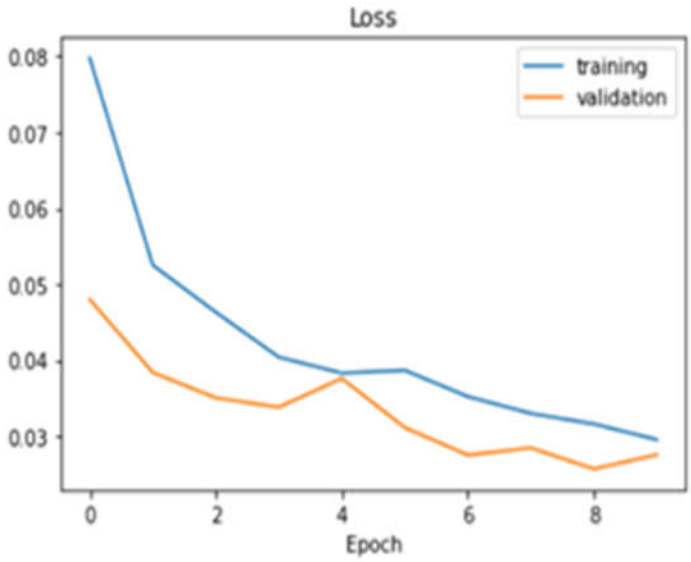

Figure 13. Plot of training loss and validation loss

The car driving autonomously can be seen in figure 14. The car is able to steer autonomously mostly in the center of the road based on the trained CNN model. It shows good performance on both left and right turns. The model is able to adapt to a completely new track. It is also able to drive the car successfully on gradual slopes and sharp turns.

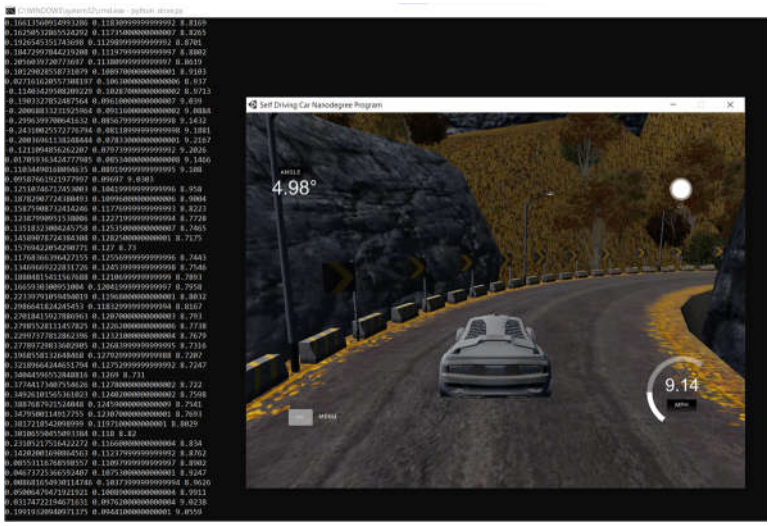

Figure 14. Testing the model in autonomous mode

4. Conclusion

A vehicle that is qualified to sense its surroundings and navigate cautiously through it without human intervention has been implemented and successfully driven in the simulator. The development of Self-driving cars is at an early stage, so along with many benefits there are some limitations of having these cars on the road which cannot be ignored. Security is a major concern as the software which helps these cars drive autonomously on the road can be hacked. Such a security breach can cause complete mayhem. Unemployment is another concerning factor as the road transport system

provides employment to many people. Weather related problems such as heavy rain can cause the sensors to malfunction and then it is really difficult for the car to read traffic signs. Considering all these factors, self-driving cars are still not a reality and a lot of research is going on to actually overcome these factors and get these cars on the road.

5. References

[1] "What Is An Autonomous Car? – How Self-Driving Cars Work|Synopsys"Https://Www.Synopsys.Com/Automotive/What-Is-Autonomous-Car.Html

[2] Nilesh Barla "Self-Driving Cars With Convolutional Neural Networks (cnn)" neptune.ai https://neptune.ai/blog/self-driving-cars-with-convolutional-neural-networks-cnn (August 25th, 2021)

[3] Davide Del Testa, Mariusz Bojarski (2016) 'end-to-end deep learning for self-driving cars'

[4] "Flask documentation" flask.palletsprojects https://flask.palletsprojects.com/en/2.1.x/

[5] "What Socket.IO is" socket.io https://socket.io/docs/v4/#what-socketio-is

[6] Javier Del Egido Sierra, Luis Miguel Bergasa Pascua (2019) "Self-Driving A Car In Simulation Through A Cnn",Ieee Transactions On Intelligent Vehicles, vol. 4, no. 2

[7] Minh-Thien Duong, Truong-Dong Do And My-Ha Le (2018) ‘ Navigating Self-Driving Vehicles Using Convolutional Neural Network’ 4th International Conference On Green Technology And Sustainable Development (Gtsd)

[8] Mohamed A. A. Babiker, Mohamed A. O. Elawad,Azza H. M. Ahmed (2019) ‘convolutional Neural Network For A Self-Driving Car In A Virtual Environment’ International Conference On Computer, Control, Electrical, And Electronics Engineering (Iccceee19)

[9] Shahzeb Ali (2021),"Self-Driving Cars: Automation Testing Using Udacity Simulation",International Research Journal Of Engineering And Technology (Irjet), Apr 2021,E-Issn: 2395-0056,vol.:08,issue: 04

[10] Yue Kang , Hang Yin , And Christian Berger (2019) ‘test Your Self-Driving Algorithm: An Overview Of Publicly Availabe