Self Driving Car in a Constrained Environment

B. Harish and Durairaj Thenmozhi(✉)

SSN College of Engineering, Chennai, India
harish17049@cse.ssn.edu.in, theni_d@ssn.edu.in

Abstract. The purpose of the research is to build a machine learning model which can drive a car on the tracks of Udacity's Car simulator without any human intervention. This is achieved by mimicking the human driving behaviour in the training mode on a track. A dataset is generated by the simulator based on the human driving behaviour in the training mode and a deep learning model is built using this dataset which is then used to drive the car autonomously on any unseen track. Initially the model performed well only on the already seen track and failed to perform well on new unseen tracks. The simulator track in which the car was trained with didn't consist of any sharp turns or elevations or any other road barriers, but the real world tracks do contain them, so in order to overcome this problem image processing techniques like zooming, changing brightness, flipping images, panning were used and in order to avoid over-fitting problem more dataset was generated using data augmentation techniques. Finally a model was built which was able to generalise the tracks and drive the car autonomously on the unseen track of the simulator.

Keywords: Self-driving car · Deep learning · Automation · Machine learning · Udacity · Unity · Track · Roads · ELU activation function · Constrained environment

1 Introduction

This section describes the need of autonomous cars and the objectives of our research work.

1.1 Motivation

Today, driving has become a stress full job, people going on long trips get tired because of driving and lose their focus many times due to which accidents happen, this can be solved through model-based and learning-based [3] approaches in order to achieve full vehicle autonomy without or with minimum human intervention. Perceiving the scenes, controlling the vehicles and choosing a best and

Published by Springer Nature Switzerland AG 2022
E. J. Neuhold et al. (Eds.): ICCCSP 2022, IFIP AICT 651, pp. 325–339, 2022.
https://doi.org/10.1007/978-3-031-11633-9_23

safe path to drive remains a challenge. A machine learning model is needed which can aid the car to drive autonomously in the real world tracks which has lot of constraints like pot holes, elevations, speed breakers etc. With advancement in technologies and many things getting automated it's time to automate the driving task by building a model which can handle a lot of edge cases and the error produced by it should be extremely small.

1.2 Objectives

The governing objectives of the study are:

- To create a driving dataset using Udacity Simulator which can provide high definition pictures and video for the development of a deep learning model, similar to the famous NVIDIA model [1].
- To identify the right tools and techniques which can help in providing a safe vehicle autonomy.
- The car must be capable of driving within the lanes of the track.

2 Related Work

Muddassir Ahmed Khan [2] has undertaken this research work 4 years ago. For data generation, images were picked from the metadata provided driving_log.csv file and passed through the augmentor to get the training data. The top and bottom 25 pixels of the images are ignored, in order to get rid of the front of the car from the images, he has used various augmentation techniques like image flipping, brightness variation, adding shadows to the images. When he trained the network using such data alone the network went off the track and wasn't able to drive properly. He then included the left and right images with various offsets such as 0.3, 0.2, but these values led to large shifts in the steering and the car would wear off the track. 0.3 worked fine when training the network for few epochs but when the number of epochs was increased the vehicles started to move in a zig-zag fashion across the road with more and more hard turns. At the last moment he changed the offset to 0.09 and started getting better results.

In 2017, Albin Falk and David Granqvist from Chalmers University of Technology, University of Gothenburg [8], Sweden proposed that by integrating deep learning and conventional computer vision based techniques, redundancy can be introduced and it can minimize the unsafe behaviour in autonomous vehicles. A control algorithm was constructed to combine the advantages of a lane detection algorithm with a deep neural network which ensures the vehicle stays within the appropriate lane. The proposition makes it evident that the system shows better performance when a combination of the two technologies were run in a simulator than using them independently.

In 2018, Aditya Babhulkar from California State University, Sacramento has undertaken this research work [9]. The dataset was collected by driving the car on Track_1 of the simulator in the training mode, and the simulator provided

the dataset in a csv file which consisted of steering angle and the path to then directory in which the images of the track were stored. During the autonomous mode the simulator acts as a server and the images captured from the single center camera of the car are sent to the client python program which uses the built neural network model and send back a steering angle to the car to drive autonomously on the Track_2 of the simulator in autonomous mode. The model was built using sequential models provided by Keras.

3 Proposed Methodology

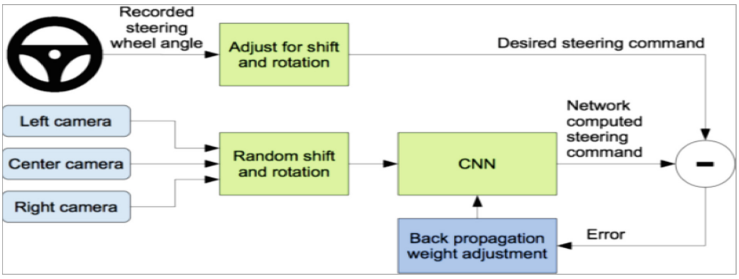

Fig. 1. Bird's eye view of the system's working

Figure 1 shows an overview of our system. The images from the three cameras attached to the car in the training mode of the simulator, are fed to the CNN model which predicts a steering angle for that part of the track, this predicted steering angle is compared to the desired steering angle proposed by human for that part of the track. And the weights of the CNN are adjusted to bring the CNN output closer to the desired output. Weight adjustment is accomplished using back propagation. Once trained, the network is able to generate steering commands for driving left, straight or right from the video images of a single center camera provided by the simulator in autonomous mode. Figure 2 shows this configuration.

4 Algorithm

- Open Unity Self Driving car simulator powered by Udacity.
- Select Track-1, the primitive track and choose training mode.
- Turn on recording mode and start collecting the data - track images and steering angle.
- Perform Data Manipulation i.e. changing or altering the collected data.
- Perform Data Augmentation.
- Pre-process the data.

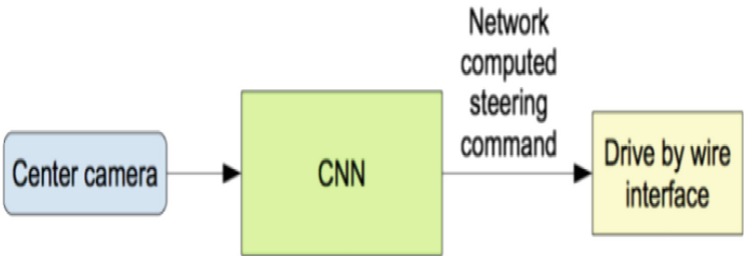

Fig. 2. The CNN predicts a steering angle for the images from the single center camera of the car in the autonomous mode of the simulator.

- Train the data using CNN and build the ML model.
- In the simulator, select Track-2 the unseen track and choose autonomous mode.
- In this mode simulator acts as a server and a python code built using Flask framework contains the built ML model acts as a client.
- The track images are sent to the python script containing the ML model.
- ML model sends the predicted steering angle for that point of the track back to the simulator.
- Test the performance of the model in the simulator.

4.1 Udacity Simulator

Unity simulator by Udacity [7] is used for training the self-driving car. The simulator provides two types of tracks, Track-1 is a circular track with a few sharp turns and no elevations, this is the track which was used for data collection purposes. Track-2 is a very complex track with a lot of sharp turns, ups and downs through out the end of the track, this is the track where we tested the car using the built machine learning model. The simulator has two modes - Training mode, where an user can move the car using his keyboard and Autonomous mode where the car is moved by instructions given by ML model. The car in the training mode has 3 cameras attached to it and in autonomous mode has a single camera attached to it.

4.2 Assumptions and Limitations

- No other cars are present along the track.
- No speed breakers are present.
- No human interventions along the track.
- Performance is affected by hardware specification, performance will be good only if run on machines with atleast 8 GB of Ram and quad cores.
- No split lanes in the track. Assuming just a one way track.
- Udacity has released a newer version where we can train the car to handle the lane changes, it will be considered in the future works.

- Road traffic with few vehicles, hurdles and risks will be considered in the future works.
- A speed limit variable is set to control the speed of the car while driving autonomously. It's value can range from 5 MPH to 30 MPH (max speed of the car).

4.3 Data Collection

Steps involved in the data collection are

- Training mode is turned on for Track-1 and data collection process is started.
- Various images of the track are captured.
- The simulator also records the Steering angle and speed of the car.
- The simulator provides the recorded data in CSV format.
- Size of the collected dataset = 25641 images of Track-1
- Amount of data used for Training = 15384
- Amount of data used for Testing on Track-2 = 10257

4.4 Data Manipulation

The car provided by the simulator in the training mode contains 3 cameras attached in it's front part. The cameras are located on the left, center and right side in the front part of the car. The idea is to make the images captured by the left and the right cameras as though they were captured by the center camera, this is done by adding a bias to their steering angles, this is necessary because in the autonomous mode the simulator provides only a single center camera to the car. Also by utilizing all the 3 cameras we get a bigger dataset of the track images to train our model, which helped in improving the accuracy of our machine learning model. The distance-arc length between center and left camera (or right camera) is called as bias After trial and error we came up with 0.09 bias, which gave good predictions when trained with our machine learning model. For the images captured by the left camera, 0.09 bias was added to it's steering angle. For the images captured by the right camera, -0.09 bias was added to it's steering angle. These images from left and right cameras after adding the bias were appended to the images captured by the center camera.

In Fig. 3, x-axis shows the range of steering angle and y-axis shows the count of images for that particular steering angle.

Since amount of images available for 0.0, 0.09 and -0.09 steering angle is more, our model will always tend to predict these steering angles making our model predict poorly, so, we deliberately delete some images containing these steering angles from various parts of the track, Fig. 4 depicts the new distribution of steering angle after that deletion.

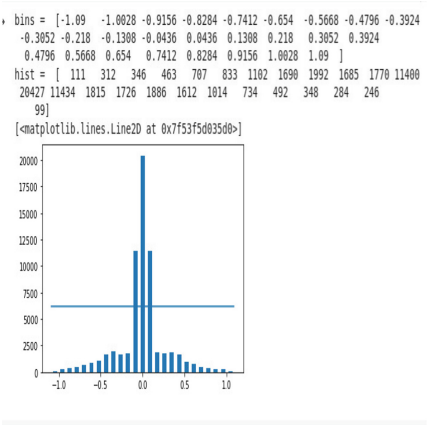

Fig. 3. Histogram of steering angle-1

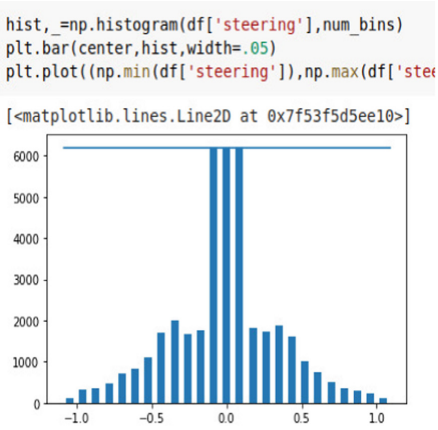

Fig. 4. Histogram of steering angle-2

4.5 Data Augmentation

The biggest challenge was to generalize the car behaviour on Track_2 (unseen track) of the simulator. In real world, we can never train a machine learning model for every available track as it will require a humongous amount of data to train and process it. Real world contains a variety of weather conditions, and it is impossible to build a dataset for all such weather conditions and roads, however by using data augmentation techniques we can create a dataset closer to the real world.

Zooming. Images were zoomed, to get a better view of the track. OpenCV techniques were used to achieve this. Figure 5 shows the original and zoomed track.

Changing Brightness. By changing the brightness of the track images we can generalize the weather conditions. By increasing brightness we can get a track which looks as if it was captured during a sunny day and by decreasing the brightness we can make the track image look like as though it was captured during night time or a cloudy day or a shadow of a building/tree falls on the track. The change in brightness for various images of the track can be seen in the Fig. 6.

Panning/Translation. The image is moved along the x-axis or y-axis, which make the neural network to look everywhere in the image to capture the vital data. The original and translated images are shown in Fig. 7.

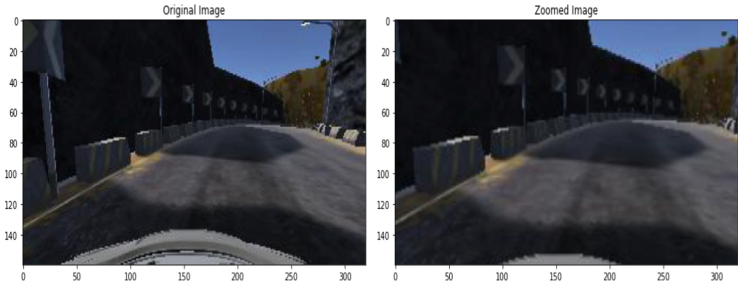

Fig. 5. Zoomed image

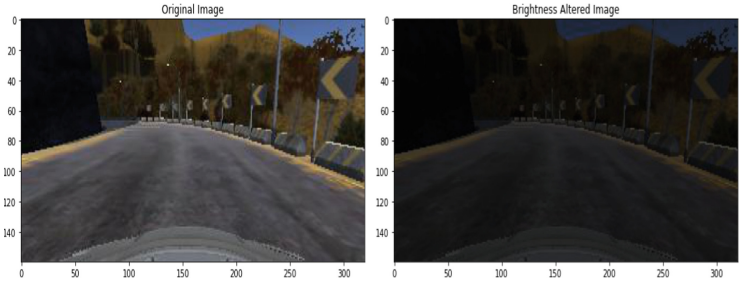

Fig. 6. Brightness altered image

Horizontal Flipping. Sometimes when we collect data, the data may be skewed to left side (or right side), i.e. if the track that we trained contained more no. of left turns compared to the no. of right turns, our model will always predict a left steering angle. Where as in real life tracks have an equal no. of left and right turns. To solve this problem, we randomly select some tracks and flip it horizontally and negating it's corresponding steering angle. This helped in making the dataset to contain nearly equal counts for left and right steering angles. Figure 8 shows the flipped image.

4.6 Pre-processing

The height of the image is reduced to bring focus on the track thereby ignoring the landscapes surrounding the track. The color images(RGB) are converted to YUV(Y-Brightness and UV - color) encoding pattern. Gaussian Blur of size 3 by 3 was applied to the image. Images were resized to the shape 200 by 66. Normalization of the images are done (Fig. 9).

4.7 Training

The recorded data is splitted into x_train and y_train. Figure 12 shows the splitted data, x-axis represents the steering angle, y-axis represents the count of

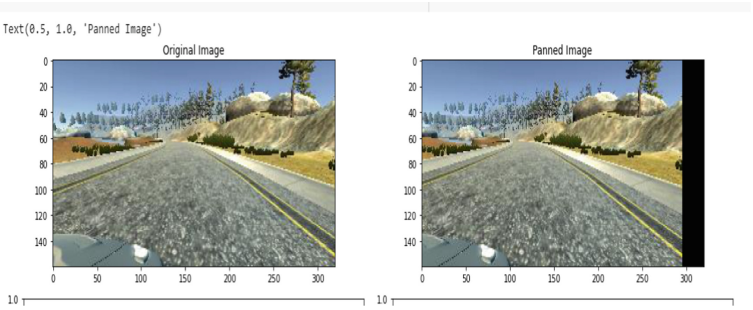

Fig. 7. Panned image

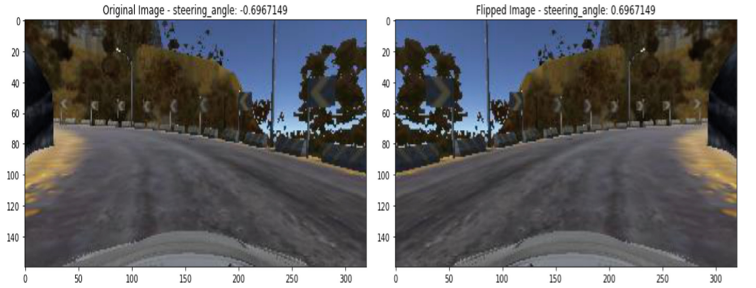

Fig. 8. Flipped image

images for each steering angle. Blue and red plot shows the histogram distribution for training and testing data respectively. x_train contains the images of the track and y_train contains the steering angle of the car.

A convolutional neural network model, is built and 80% of the collected dataset is used for training remaining 20% is used for testing. The weights of the network are adjusted by back propagation to minimize the mean-squared error between the predicted steering angle and the desired steering command of the human driver. Figure 13 shows the network architecture, The network architecture contains nine layers of which there are five convolution layers, three fully connected layers and a normalization layer. The image from the simulator is pre-processed and is converted into YUV planes and is passed to the network.

First layer of the network is normalization layer which performs normalization of the input image. Feature extraction is done by the convolution layers. 2×2 stride convolution and 5×5 kernel are used in the first three convolution layers and the last two convolution layers consist of a non-strided convolution and a 3×3 kernel. The convolution layers are designed to perform feature extraction, and are chosen through a series of experiments that vary layer configurations. We then use strided convolutions in the first three convolution layers with a 2×2 stride and a 5×5 kernel, and a non-strided convolution with a 3×3 kernel size in the last two convolution layers.

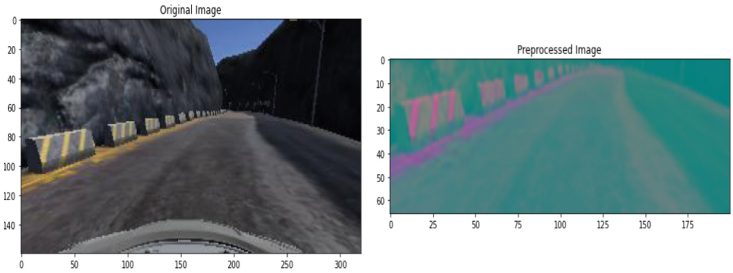

Fig. 9. Pre-processed image

The five convolution layers are shown in Fig. 13. The three fully connected layers which are designed to act as controller for steering, outputs the steering angle. To avoid over-fitting problem, we need a large amount of data to train on, so using data augmentation technique the batch generator creates large amount of data on the fly i.e. only during training after this the training begins. The Rectified Linear Unit-ReLU activation function-R(z) = max(0, z) was used during the initial stages of building and testing the model, however ReLU was not a good choice for this problem because ReLU function returns zero for values less than zero but our steering angles consist of both positive and negative values. So if a neuron gets an input of negative number it will return a value of zero, so the gradient at this point is zero and the weight of this neuron will never be changed because the back propogation uses the gradient value to change the weight values of the neurons. This phenomenon is called as dead ReLU. Due to this the neuron will always receive a value of zero and will always feed forward the value of zero and there is no learning. Sigmoid function can't be used as it would create vanishing gradient problem as our network is complex. So we chose Exponential Linear Unit-ELU activation function which is similar to ReLU in positive region but in negative region it returns the negative number. So unlike ReLU, ELU has a non-zero gradient value in the negative region, which means that it can always recover and fix it's weight parameters to decrease it's error i.e. it is always capable of learning and contributing to the model, unlike ReLU which can essentially die. Figures 10 and 11 shows the ELU, ReLU and Sigmoid functions. Based on the accuracy, the CNN model is modified by changing the no. of neurons, by adding dropout layers, changing the learning rate, data augmentation etc. to improve the accuracy.

Using PYTHON and Flask the created CNN model is given to the simulator and the car is driven in autonomous mode in the selected track. A speed limit is set to the car, the speed of the car increases till that limit and begins to gradually slow down after that.

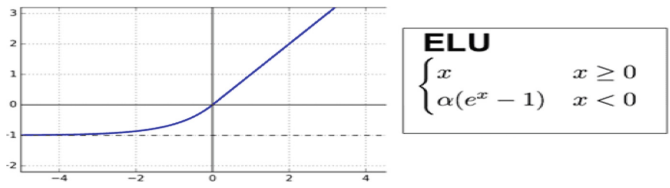

Fig. 10. ELU function

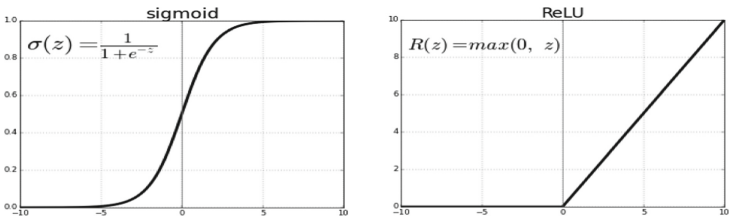

Fig. 11. ReLU and sigmoid functions

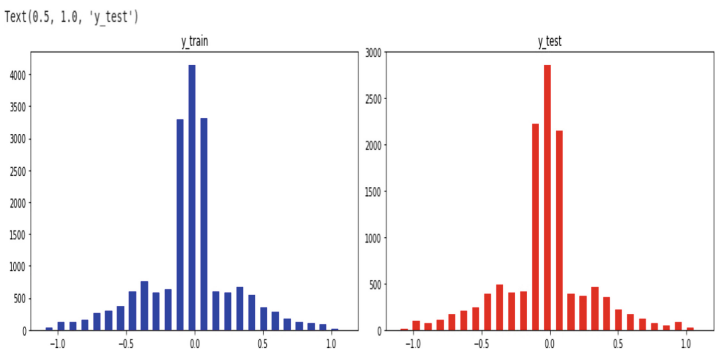

Fig. 12. Histogram distribution of steering angles after train-test split (Color figure online)

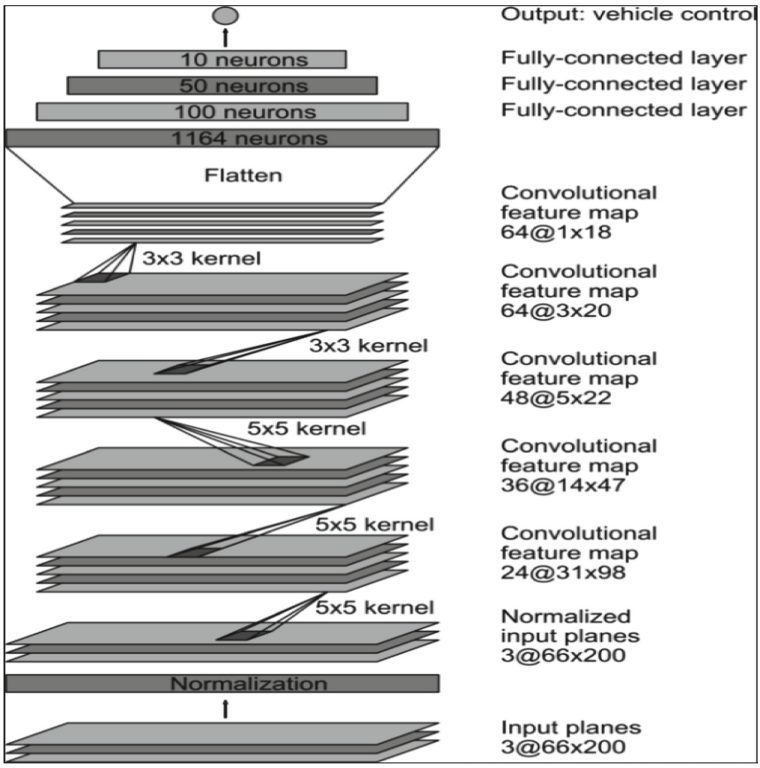

Fig. 13. Architecture

5 Testing

The built model is used in a python application which uses socketio to connect to the udacity simulator (server) by acting as a client. The Udacity simulator in autonomous mode acts as a server. In terminal the python driver file is run after loading the model. Inside the simulator, autonomous mode is selected and Track-2 - the unseen track is chosen. Unseen track is the one which the car has never seen before while training. The simulator sends the track images to the client (python application), which uses the built model to predict the steering angle for the car at that point of the track. The client code returns the steering angle to the simulator. To avoid the car going very fast, a speed_limit variable is used to control the throttle of the car. Response time is the sum of, time to send the track image to ML model, time for ML model to predict the steering angle for that point of track, time to send back the steering angle to the simulator. Here sending back steering angle takes very less time compared to other two factors. It takes a lot of time to send an image from simulator to model, but however since the model is available in local machine we can decrease this time by using it on a machine with decent hardware specs like atleast 8 GB ram

and quad cores. And the ML model predicting the steering angle also depends on the hardware specs, so by using the above hardware specs we can altogether reduce the latency i.e. the response time. Figures 14 and 15 shows the car driving without human intervention in an unseen track i.e. in Track-2 of the simulator.

Fig. 14. Car driving in autonomous mode

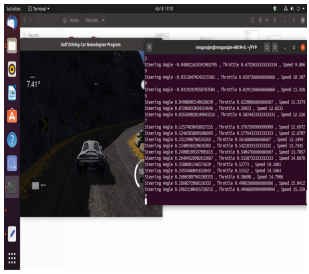

Fig. 15. Car driving in autonomous mode

6 Results

This section shows how our approach is different from the existing approaches and how well the built model has predicted in various parts of the unseen track (Track-2).

Our approach though it uses the same augmentation techniques as with the existing approaches mentioned in the literature survey, how we have used it matters. The existing approaches use all the augmentation techniques at once during data augmentation, whereas in our approach we apply the 4 augmentation techniques (zooming, panning, flipping images and changing brightness) randomly, we select an image at random from the training dataset and then we generate a random number between 0 and 1, if the value is greater than 0.5 we apply zooming technique to it, or else we don't apply zooming to that image, and again we repeat the same for other augmentation techniques. So a particular image chosen for augmentation may be applied with all the 4 augmentation techniques or 3 or 2 or 1 or even none. This approach produces a dataset with increased randomness in the images compared to the existing approaches. Also, we have used ELU activation function which drastically improved the performance in the unseen track i.e. Track-2.

Images below on the left column represent the true value of steering angle (TSA) for that point in the track.

Images on the right column represent the predicted value of steering angle (PSA) for that point in the track.

6.1 Left Curved Roads

See Figs. 16, 17, 18 and 19.

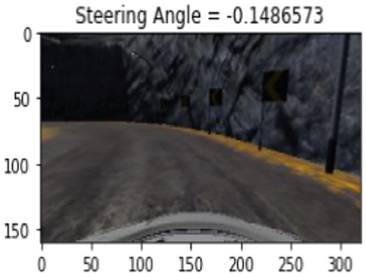

Fig. 16. True SA = −0.148

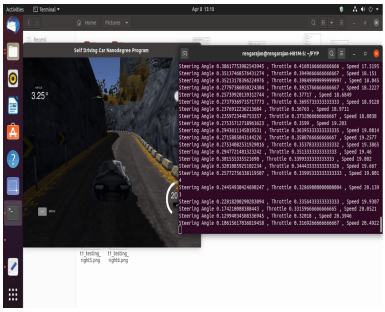

Fig. 17. Predicted SA = −0.106

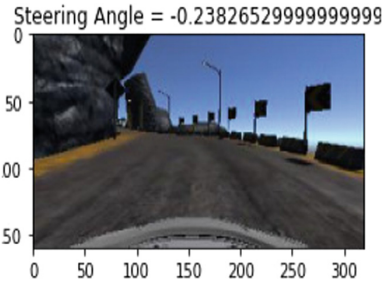

Fig. 18. True SA = −0.238

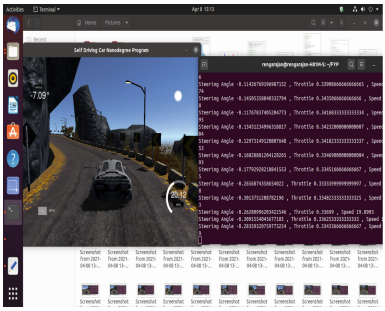

Fig. 19. Predicted SA = −0.283

6.2 Right Curved Roads

See Figs. 20, 21, 22 and 23.

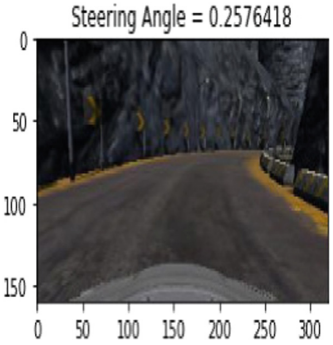

Fig. 20. True SA = 0.257

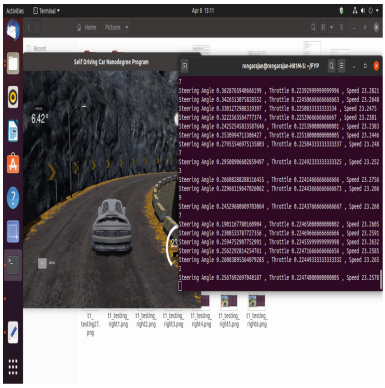

Fig. 21. Predicted SA = 0.256

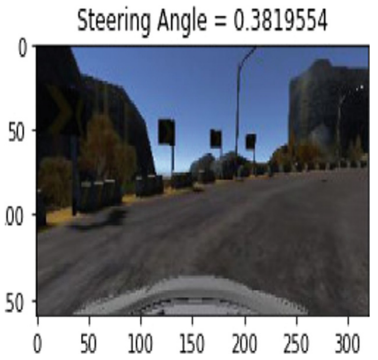

Fig. 22. True SA = 0.381

Fig. 23. Predicted SA = 0.430

S. No	Figure No	TSA	PSA	$\|PSA - TSA\|$	$\|PSA - TSA\|^2$
1	16,17	−0.148	−0.106	0.042	0.00176
2	18,19	−0.238	−0.283	−0.045	0.00202
3	20,21	0.257	0.256	−0.001	1.0000e−06
4	22,23	0.381	0.430	0.049	0.002401

$$\text{MSE(Mean Squared Error)} = \sum_{i=1}^{4} \|PSA - TSA\|^2/4 = 0.00154$$
$$\text{RMSE(Root Mean Squared Error)} = \sqrt{\sum_{i=1}^{4} \|PSA - TSA\|^2/4} = 0.03934$$

The table above shows the difference between True Steering Angle (TSA) and Predicted Steering Angle (PSA) i.e. $\|PSA - TSA\|$, $\|PSA - TSA\|^2$, for the various road tracks.

7 Scope of Deployment in Real World

Since this research focused on working in a constrained environment, in the real world it could be used in places like industries where unmanned cars can be used for transporting goods. The minimum hardware requirement is a camera to capture the track images, servo/induction motors to drive the car based on predicted steering angle and a computer to run the model and give instructions to the car. We can use a Programmable Logic Controller (PLC) as a computer which would cost a minimum of Rs. 20000. The cost of a good quality single camera compatible with PLC would be around Rs. 5000. The cost of servo motors for real cars would cost around Rs. 20000. The cost of building the entire setup would be around Rs. 20000. So the total cost to deploy this work with minimum hardware configurations in a constrained environment in the real world would be around Rs. 65000.

8 Conclusions and Future Work

This research started with capturing images of the tracks, using computer vision [4,6] OpenCV techniques to add shadows to tracks, zooming of tracks, etc. Initially the model performed well only on the already seen track and failed to perform well on new unseen tracks. So, many models were built by changing the parameters like the number of neurons or the activation function so that the model was able to generalize road conditions and achieve the similar performance on different tracks. The spatial features were obtained by using CNN [5]. In this research we used only the data obtained from the simulator, future works can be carried out by combining simulator data with the real world data and already many experimental configurations are being carried out in the field of autonomous cars. The models used in this research were built sequentially using Keras, future works can be done by trying parallel network layers for learning the specific track behaviour which can lead to an increase in performance. In this research the simulated environment didn't consist of traffic environment, multiple lanes, obstacles or other cars along the track, when these are placed on the tracks they would make it more closer to a real world environment and a challenge for the self-driving cars.

References

1. Bojarski, M., et al.: End to end learning for self-driving cars. arXiv preprint arXiv:1604.07316 (2016)
2. Khan, M.A.: Making A Virtual Self-Driving Car, Published on 1 April 2017. https://muddassirahmed.medium.com/making-a-virtual-self-driving-car-2d81f3f539e8
3. Fridman, L., et al.: MIT autonomous vehicle technology study: large-scale deep learning based analysis of driver behavior and interaction with automation. arXiv preprint arXiv:1711.06976 (2017)
4. Dai, J., et al.: Deformable convolutional networks. In: Proceedings of the IEEE International Conference on Computer Vision, pp. 764–773 (2017)
5. Jmour, N., Zayen, S., Abdelkrim, A.: Convolutional neural networks for image classification. In: 2018 International Conference on Advanced Systems and Electric Technologies (ICASET), pp. 397–402. IEEE (2018)
6. Badrinarayanan, V., Galasso, F., Cipolla, R.: Label propagation in video sequences. In: 2010 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 3265–3272. IEEE (2010)
7. Naoki, Introduction to Udacity Self-Driving Car Simulator, Published on 25 February 2017. https://naokishibuya.medium.com/introduction-to-udacity-self-driving-car-simulator-4d78198d301d
8. Falk, A., Granqvist, D.: Combining deep learning with traditional algorithms in autonomous cars (2017)
9. Babhulkar, A.: Self-driving car using udacity's car simulator environment and trained by deep neural networks (2019)