**READ ME FILE**

**Implementation of Autonomous Vehicles using Deep Learning**

**1.Dataset Generation:**

This section consists of training a Deep-Learning model using the dataset gathered by our experiment as human subject, whose behaviour is to be cloned. This dataset was recorded using the Udacity Self-Driving Car Simulator [5]. The simulator consists of two modes:

1. Training Mode - It is used for recording the dataset

2. Autonomous Mode - It is used for testing the model.

The simulator is based on Unity Engine. The car has three cameras attached to it: one in the front, and two at each side of the car. Figures 3.4, 3.5 show the training track and testing track in Udacity simulator. Path provided in the simulator during the training phase

involves a one-way track, which had multiple curvatures, obstacles, bridges, and rivers. Testing track is an unknown mountain area, which has up-down scenarios and non-uniform

curvatures. The first mode, Training Mode, is used to gather the dataset. Therefore, they used this mode to gather the dataset. They drove multiple rounds each in forward as well as in backward direction on the first track.

Initially the dataset contained around 4053 frames along with the respective labels. Since, the dataset was highly skewed towards 0 degree, they under sampled it to get a more balanced dataset. They selected a threshold value of 350 i.e. corresponding to each angle we can have a maximum of 350 frames. This threshold was selected keeping in mind the data-imbalance condition as well as to maintain a decent count of the frames available for training our model. The dataset was then split into Training Set and Validation set in the ratio of 4:1.

Udacity has built a simulator for self-driving cars and made it open source for the enthusiasts, so they can work on something close to a real-time environment. It is built on Unity, the video game development platform. The simulator consists of a configurable resolution and controls setting and is very user friendly.

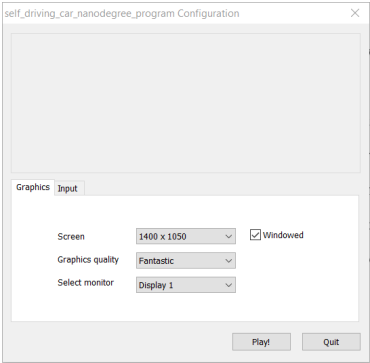


Figure 1.1: Configuration screen

The graphics and input configurations can be changed according to user preference and machine configuration as shown in Figure 3.1 . The user pushes the “Play!” button to enter the simulator user interface. We can enter the Controls tab to explore the keyboard controls, quite similar to a racing game which can be seen in Figure 3.2.

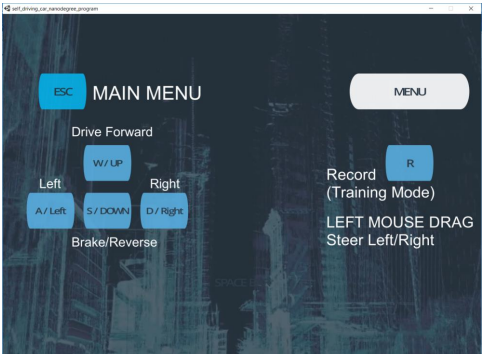


Figure 1.2: Controls Configuration

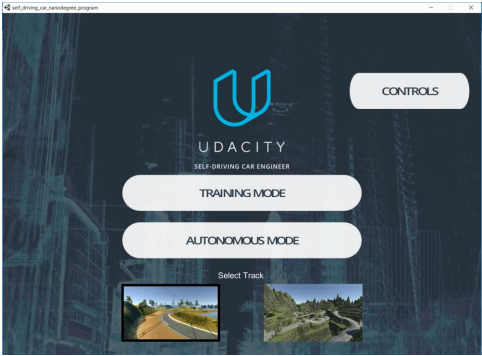


Figure 1.3: First Screen

The first actual screen of the simulator can be seen in Figure 3.3 and its components are discussed below. The simulator involves two tracks. One of them can be considered as simple and another one as complex that can be evident in the screenshots attached in Figure 3.4 and Figure 3.5. The word “simple” here just means that it has fewer curvy tracks and is easier to drive on, refer Figure.3.5.



Figure 1.4: Track\_1

The “complex” track has steep elevations, sharp turns, shadowed environment, and is tough to drive on, even by a user doing it manually. Please refer Figure.3.6



Figure 1.5: Track\_2

There are two modes for driving the car in the simulator:

(1) Training mode and

(2) Autonomous mode.

The training mode gives you the option of recording your run and capturing the training dataset. The small red sign at the top right of the screen in the Figure 3.4 depicts the car is being driven in training mode. The autonomous mode can be used to test the models to see if it can drive on the track without human intervention. Also, if you try to press the controls to get the car back on track, it will immediately notify you that it shifted to manual controls. The mode screenshot can be as seen in Figure 3.6.



Figure 1.6: Autonomous mode

The simulator’s feature to create your own dataset of images makes it easy to work on the problem. Some reasons why this feature is useful are as follows:

• The simulator has built the driving features in such a way that it simulates that there are three cameras on the car. The three cameras are in the center, right and left on the front of the car, which captures continuously when we record in the training mode.

• The stream of images is captured, and we can set the location on the disk for saving the data after pushing the record button. The image set are labeled in a sophisticated manner with a prefix of center, left, or right indicating from which camera the image has been captured.

• Along with the image dataset, it also generates a datalog.csv file. This file contains the image paths with corresponding steering angle, throttle, brakes, and speed of the car at that instance. A few images from the dataset are shown in Figures 3.7, 3.8, 3.9, 3.10, 3.11, 3.12.

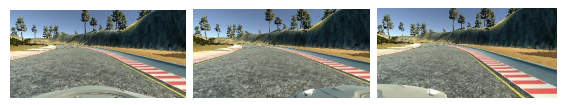


Figure1.9:Left0001

Figure1.8:Right0001

Figure 1.7:Center0001



Figure1.11:Right0099

Figure1.10:Center0099

Figure1.12:Left0099

A sample of datalog.csv file is shown in Figure 3.13.

Column 1, 2, 3: contains paths to the dataset images of center, right and left respectively

Column 4: contains the steering angle Column value as 0 depicts straight, positive value is right turn and negative value is left turn.

Column 5: contains the throttle or acceleration at that instance

Column 6: contains the brakes or deceleration at that instance

Column 7: contains the speed of the vehicle

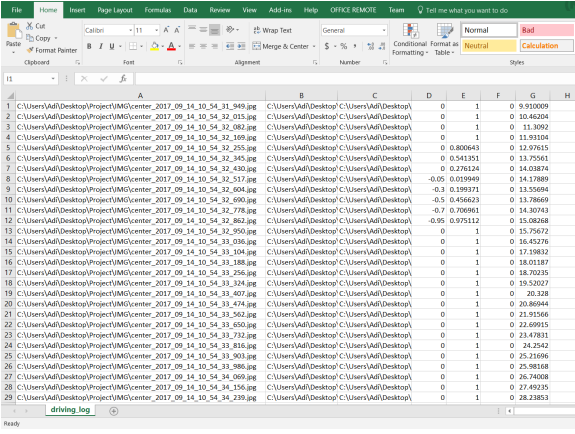


Figure1.13:driving\_log.csv

**2.Preprocessing and Image Augmentation:**

After having our dataset in place, the next step was to preprocess the data before providing it to CNN [4] based model. To preprocess the data we harnessed the power of

OpenCV 2 [6] library. The preprocessing applied to the input data were cropping the

image, converting the Channel from RGB to YUV, applying Gaussian Blur, resizing the image and normalization. Therefore, each frame, that is passed to the model is first preprocessed as explained in the pseudo-code below:

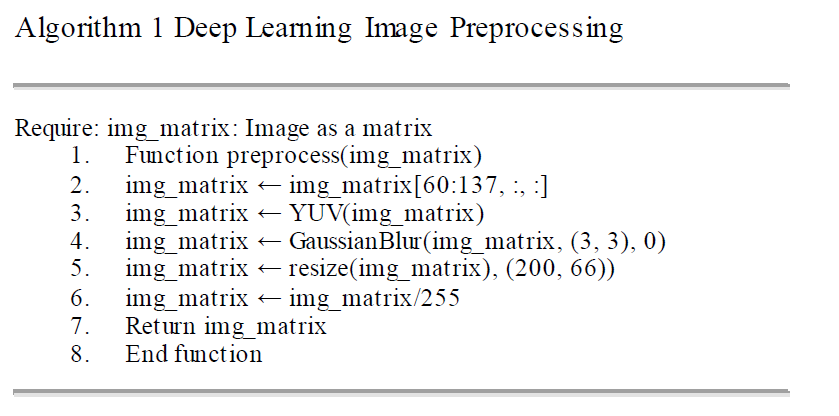
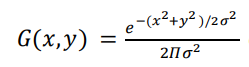


Figure1.14: Image preprocessing algorithm

Conversion of the channel is performed to get the pixels in YUV format. This is performed because YUV channel is more efficient and reduces the bandwidth more than what RGB can

capture [28]. In Gaussian Blur, the image is convolved using a Gaussian filter to minimise the noise in the image [19]. Gaussian filter basically uses Gaussian function, which is applied to each pixel in the image. The Gaussian function for two-dimensions is shown in below equation :



Further, normalization was performed to reduce each pixel to a similar data distribution which allows faster convergence of the model. The formula for Normalization is illustrated in equation shown below:



Our next step was to leverage the power of a concept known as Image Augmentation, which in the past has proved to be highly beneficial for any CNN [4] based model. Zooming() function is used for producing a new image from the given image by either zooming in or zooming out. Panning() function is used for producing translation in the image. brightness () function is used to multiply each intensity value of pixels in Image to brighten or darken the image. flipping ()

function flips the image Horizontally as well as the corresponding Steering Angle. In our model we used ELU activation function [21]. Adam optimizer [22] is used with default parameters along with Mean Square Error as the regression loss function. Batch size of 32 was selected. The pseudo-code for the process of Image Augmentation is described in algorithm 2:

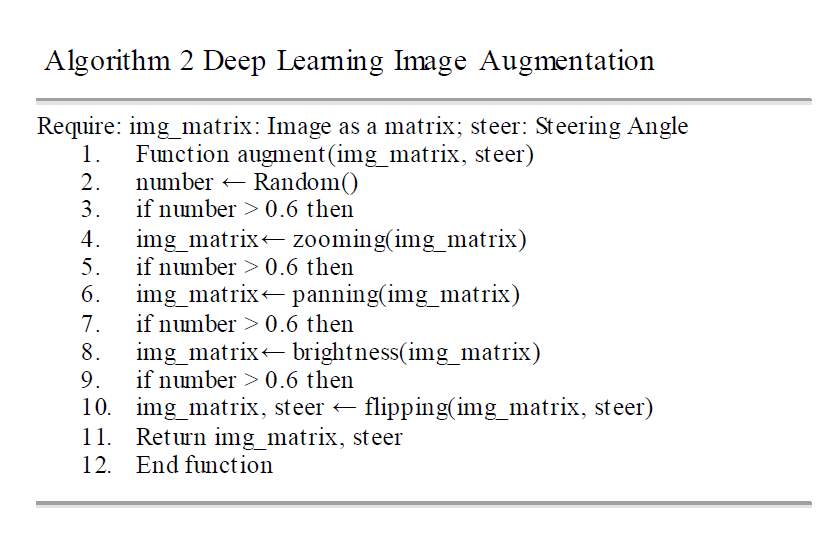


Figure1.15: Deep Learning Image Augmentation Algorithm

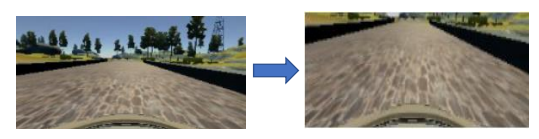


Figure 1.16: Crop Image



Figure 1.17: Flip Image

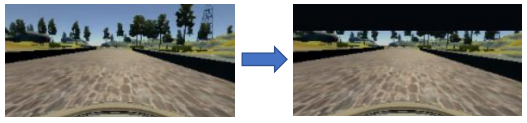


Figure 1.18: Shift Image vertical

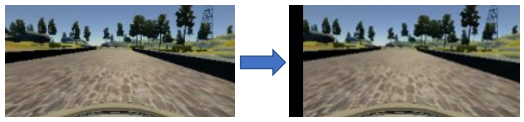


Figure 1.19: Shift Image horizontal

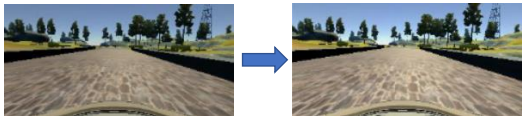


Figure 1.20: Brightness increase

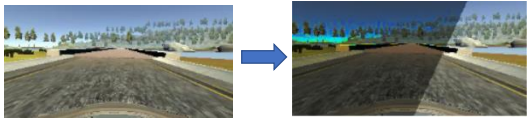


Figure 1.21: Random Shadows



Figure 1.22: Random blur

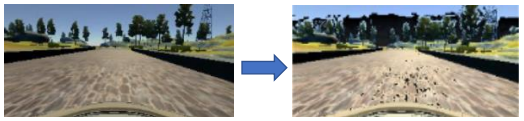


Figure 1.23: Random Noise

**3.Training and Validation:**

This section consists of the configurations used to set up the models for training the Python Client to provide the Neural Network outputs that drive the car on the simulator. The tweaking of parameters and rigorous experiments were tried to reach the best combination. Though each of the models had their unique behaviors and differed in their performance with each tweak, the following combination of configuration can be considered as the optimal:

• The sequential models built on Keras with deep neural network layers are used to train the data.

• Models are only trained using the dataset from Track\_1.

• 80% of the dataset is used for training, 20% is used for testing.

• Epochs = 50, i.e. number of iterations or passes through the complete dataset. Experimented with larger number of epochs also, but the model tried to “overfit”. In other words, the model learns the details in the training data too well, while impacting the performance on new dataset.

• Batch-size = 40, i.e. number of image samples propagated through the network, like a subset of data as complete dataset is too big to be passed all at once.

• Learning rate = 0.0001, i.e. how the coefficients of the weights or gradients change in the network

• ModelCheckpoint() is the function provided in Keras to save checkpoints and to save the best epoch according to the validation loss. There are different combinations of Convolution layer, Time-Distributed layer, MaxPooling layer, Flatten, Dropout, dense and so on, that can be used to implement the Neural Network models. Out of around ten different architectures I tried, three of the best ones are discussed in the chapter on network architectures.

**4.Testing:**

After training the model with the proposed CNN architecture with the preprocessed image, the model is connected to the simulator to predict the steering angles in real-time. The frames were pre-processed, and the model passed back the Steering Angles and Throttle values back to the simulator in real-time. We trained our model using the frames obtained from the first track. For testing the model, we allowed the model to drive the car in a track it had never seen before, i.e different track .Throttle value is calculated using the equation. The entire process can be summarized using the workflow given in Figure 3.24 .

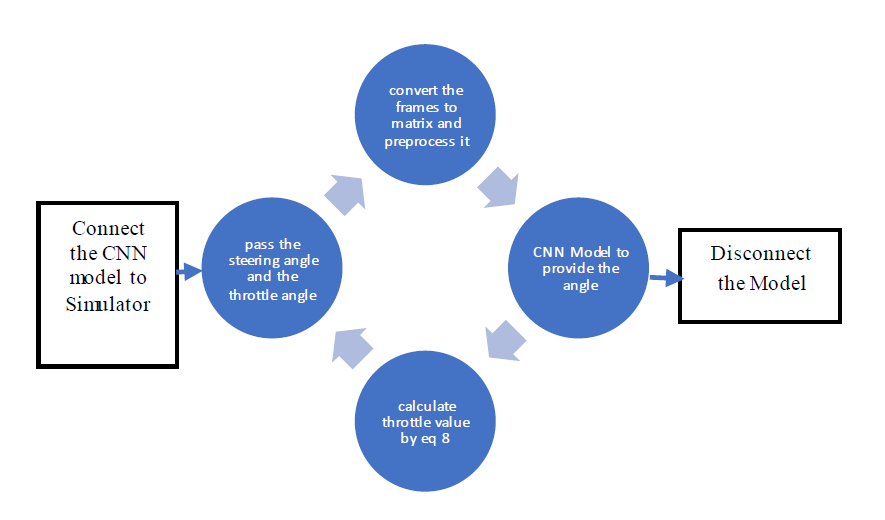


Figure 1.24: Workflow diagram of the model with simulator