Behavioral Cloning

The aim of this project is to design and train a neural network to drive a car autonomously along a given track in the simulator provided by Udacity. The goals / steps of this project are the following:

- Use the simulator to collect data of good driving behavior
- Build, a convolution neural network in Keras that predicts steering angles from images
- Train and validate the model with a training and validation set
- Test that the model successfully drives around track one without leaving the road
- Summarize the results with a written report

Rubric Points

Here the rubric points are considered individually and described how each point is addressed in the implementation.

Files Submitted & Code Quality

1. Submission includes all required files and can be used to run the simulator in autonomous mode

My project includes the following files:

- Model.py containing the script to create and train the model
- drive.py for driving the car in autonomous mode
- model.h5 containing a trained convolution neural network
- writeup_report summarizing the results

2. Submission includes functional code

Using the Udacity provided simulator and the drive.py file. The car can be driven autonomously around the track by executing the following command. The neural network controls only the lateral motion of the vehicle by predicting the steering angle. The longitudinal motion is controlled by a PI controller in the drive.py program with a pre defined speed set point of 9 mph. The model is able to drive the vehicle in both the track 1 and track 2 in the simulator. The videos of the test runs are also store in .mp4 format.

```
python drive.py model.h5
```

3. Submission code is usable and readable

The Model.py file contains the code for training and saving the convolution neural network. The file shows the pipeline I used for training and validating the model, and it contains comments to explain how the code works.

The code contains mainly three sections, the first section of the code reads log file from the simulator and stores the file names of the images and the steering angle data in a pandas dataframe. The second section of the code contains the generator function, which reads the images and the steering values and feeds the network with input values and the reference values. The third section of the code contains the neural network implemented in keras which trained using the data from the generator and finally the network is stored as model.h5 file.

Model Architecture and Training Strategy

Preprocessing

The images captured by the simulator are given to the network. The input images are coloured images of size 160 x 320 pixel size. The image is pre-processed in three steps before being feed to the network. The image captured from the simulator is a three channel colour image of format BGR. But the drive.py which is used to run the simulator in autonomous mode feeds the network with image of RGB format. So as a first step the image is converted from BGR to RGB format. Then the input images are 8 bit images with values of pixels between 0 and 255. So the image is normalized to have zero mean and equal variance in the horizontal and vertical direction. The normalization is done using the following formula.

$$X^{'} = a + \frac{(X - X_{min})(b - a)}{X_{max} - X_{min}}$$

$X_{min} = 0,$

$X_{max} = 255,$

$a = -0.5$

$b = 0.5$

The input image not only contains the picture of the road but also the road environment such as trees, mountains, parts of car etc. But for calculating the steering angle does not need this information so these unwanted sections of the images are cropped before it is feed to the neural network.

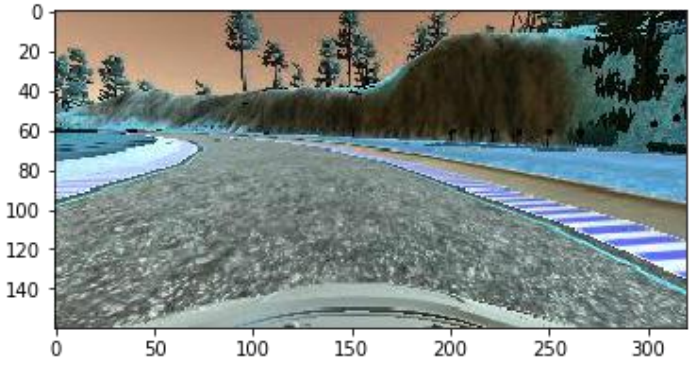


Fig 1: Image before pre-processing

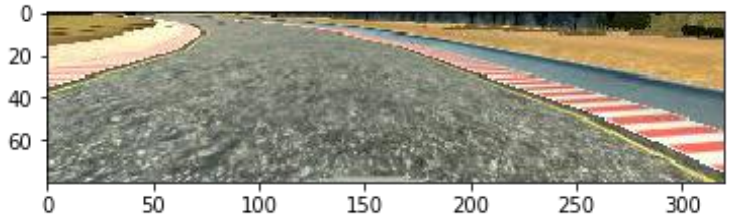


Fig 2: Image after pre-processing

Data Augmentation and Multiple Cameras

In order to train the network with more data, augmentation is used. Images from multiple cameras are used where the cameras are placed in the center, left side and right side of the car. The image from side cameras make the car appear in the side of the road even if the image was taken when the car is in the center. The following images (Fig 3 and Fig 4) show the images from the left and right cameras when the center camera has taken the image shown in the Fig 1.

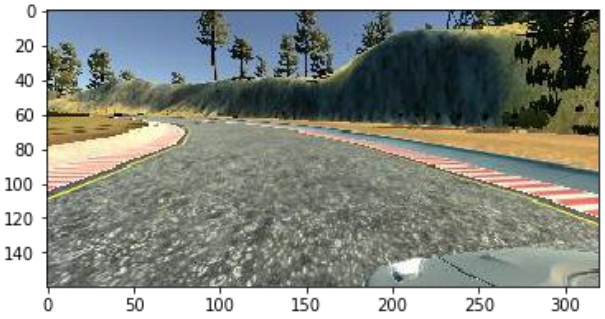


Fig 3: Image from the left camera

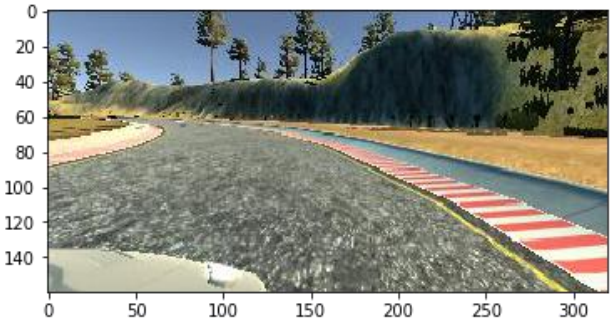
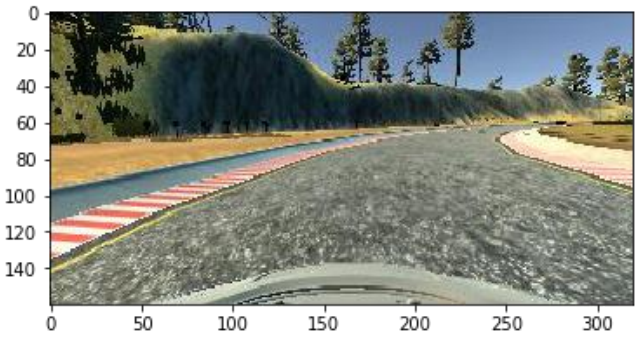


Fig 4: Image from the right camera

In order to correct the offset by the side cameras a correction factor is used to compensate the steering angle. The correction factor is added to the steering angle for the left camera and subtracted from the steering angle for the right camera. This is because the negative values of steering angle corresponds to turning the car left and positive values corresponds to turning the car right. The correction factor of the

steering angle for makes the car move towards the center. The left and right images along with the correction factor provide some recovery data for the network.

In addition to the multiple cameras the image from the center camera is flipped to augment the data further. For the flipped image the sign of steering angle is also changed. The track one in the simulator contains mostly left turns, the flipped images helps the training data more generic. The following image shows the flipped image of the image shown in the Fig 1.



Model Architecture

The architecture of the neural network is based on Lenet architecture used for classification of handwritten digits. Since the behavior cloning problem is a regression problem the Lenet network is modified. The neural network for the behavior cloning has two convolution layers followed by max pooling layers and three fully connected layers. The activation function Rectified Linear Unit (RELU) is used for the activation of the convolution layers and the activation function TanH are used for the fully connected layers. The main difference between the Lenet architecture and the behavior cloning neural network are the activation functions and the input image size. The activation function RELU and TanH provide non-linearity to the network. Since the steering angle value ranges from -1 to 1 the TanH function is a natural choice for activation. The network architecture used for this project is shown in the table below.

Layer	**Description**
Input layer	Image size 160 x 320 x 3
Normalization and Mean centering	Converts the data from range 0 to 255 to -0.5 to 0.5
Cropping	The unwanted region in the image is cropped
Convolution layer	Filters = 6, kernel size = (5,5), stride 1 step
Activation layer	Rectified Linear Unit (RELU)
Maximum Pooling	Kernel size = 2,2 and stride 1 step
Convolution layer	Filters = 16, kernel size = (5,5), stride 1 step
Activation layer	Rectified Linear Unit (RELU)
Maximum Pooling	Kernel size = 2,2 and stride 1 step
Flatten	Flattens the 2D array image to 1D array
Fully connected layer	No. of layers = 128
Activation Function	Tanh

Fully connected layer	No. of layers = 84
Activation Function	Tanh
Dropout Layer	Drop probability = 0.4
Fully connected layer	No. of layers = 1
Activation Function – Output Layer	Tanh

Table 1: Network Architecture

Prevention of Overfitting

Between the output layer and the last hidden layer, a dropout layer is used during the training process. The dropout layer randomly deletes few neurons in the network and the network relies on the remaining nodes. Thus the network becomes more robust and independent of weights of single neurons. This prevents the network from overfitting to the training data. The dropout function in keras uses drop probability instead of keep probability in Tensor Flow which means that the probability value given in the keras dropout function represents the probability that a neuron is dropped from the network. A moderate value of 0.4 is used in this network which is large enough to prevent overfitting and also not so large to prevent the network from learning the training data.

Another method used to prevent overfitting was to stop the training process as soon possible when the mean square loss reached the minimum. The parameters of the network such as batch size, learning rate, and batch size were tuned in such a way that both the losses during training and the validation reduce monotonically. If the mean square loss of validation increases or oscillates during the end of training process it suggests that the network is over-fitted to the training data. In such case, the network is trained again with less number of epochs.

Model Parameter Tuning

The neural network model is trained based on the mean square loss between the steering angles values in the training data and the angles predicted by the network. Mean square loss is used instead of the cross-entropy loss because this is a regression network. For the optimizer Adam optimizer is used, which is an extension of the stochastic gradient descent algorithm. The Adam algorithm is computationally more efficient than the stochastic gradient descent algorithm. The Adam optimizer does not keep the learning rate constant instead it computes the adaptive learning rates of each of the parameters based on the first and second order moments of the gradients. This makes the optimizer more efficient. As Adam optimizer is used the tunable parameters in the network are the batch size, epochs and the dropout rate. As mentioned in the previous section the network parameters are tuned such a way that the mean square loss for both training and validation datasets reduces monotonically. The following parameters are the final parameters of the network.

Parameter	**Value**
Batch Size	32
Epochs	3
Dropout rate	0.4

Table 2: Final Model parameters

The following figure shows the mean square loss in each epoch for the training and validation dataset.

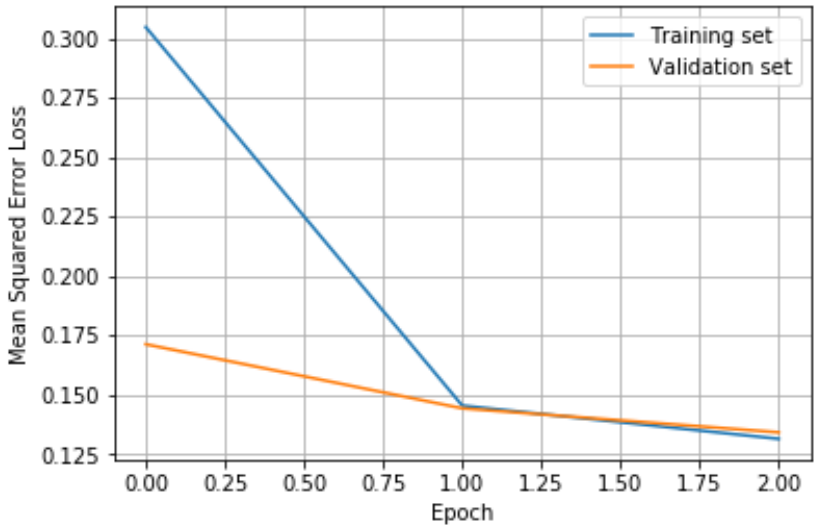


Fig 5: Mean Square Loss of the Neural Network

Solution Approach and Training Data

Solution Approach

The neural network was developed initially by using a simple neural network with only one fully connected layer. Then features were implemented to the network such as data augmentation, using multiple cameras and normalization of image. Then the network was tested in the simulator repeatedly and based on the performance in the simulator the network was improved by collecting more data, changing the network architecture and tuning of model parameters.

In the Initial stages of development it was noticed in the training data that the steering angle recorded contains high frequency oscillation and the network could not predict correct steering angles for sharp turns. There was an idea to use filtered steering angle values during the training of the network to improve its performance. A simple running average filter (filter size 5) was used to filter the steering angle values. This improved the performance of the network which was visible in the mean square loss and also in the simulator. But the performance improvement was only visible when the training data was small. When the training data was large in the later stage of development the improvement was not visible in the simulator. So filtering of the steering angle was not used in the final model. The picture below shows the mean square loss of the neural network with filtered steering angle with the same training data and parameters. It can be clearly seen the mean square loss of the network at the end of training process is lower than the network without the filter for steering angle. As mentioned before the performance of the network in simulator did not improve when it was trained with filtered steering angle. When the filter order was increased the network could not drive safely through the track.

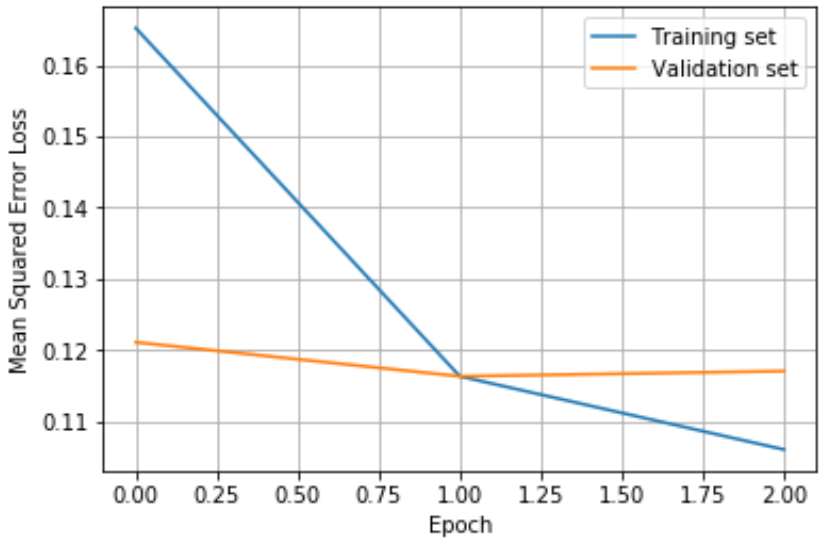


Fig 6: Mean Square Loss of the Neural Network with Filtered Steering Angle

After the implementation of preprocessing and the data augmentation steps, the Lenet based neural network explained in the previous section was applied. Once this architecture was implemented there was a big improvement in the performance of the network but the network could not completely drive along the track 1 in the simulator as the training data was small. There was a thought to use more complex network architecture, which was published by Nvidia, which contains five convolution layer and four fully connected layers. But this was not done because the Nvidia network architecture consumes large memory and it could not be run on the machine. So the Lenet based network architecture is used, which is complex enough to drive the vehicle along both the track 1 and track 2 in the simulator with large training data.

Data Collection Strategy

The training data are collected from the simulator by running it in the training mode. The training data contains the following:

- Driving in the center of the road
    - 2 laps in Track 1 in forward direction
    - 1 lap in the opposite direction in Track 1
    - 2 laps in the Track 2 in forward direction
    - 1 lap in the opposite direction in Track 2
- Recovery driving
    - 1 lap in Track 1
    - 1 lap in Track 2
    - More data in problematic locations in both track 1 and track2
        - Turns in track 1 where there are no shoulder markings
        - Double hairpin bend in track 2

- Uphill zig-zag turn

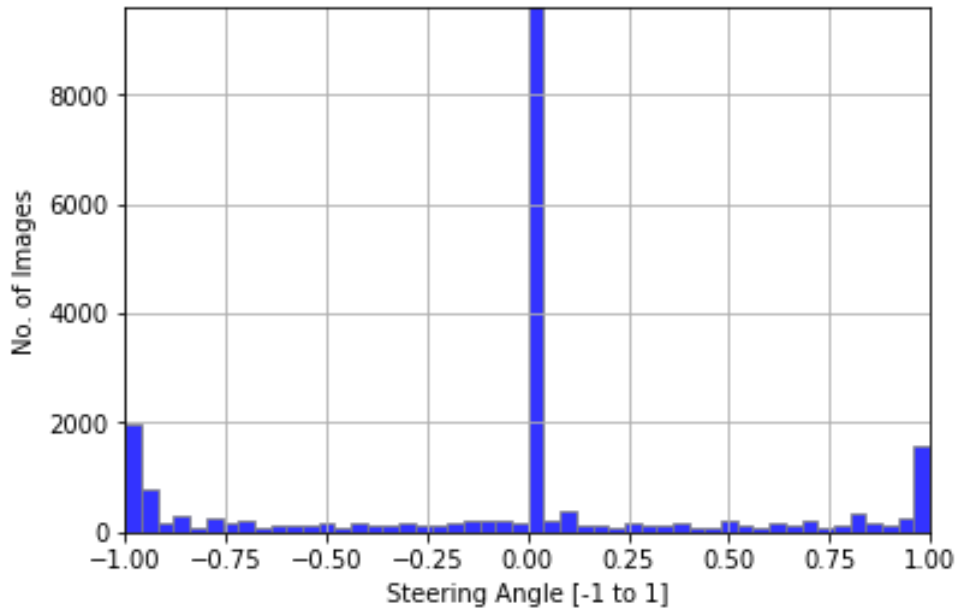


Fig 7: Steering Angle Histogram of the collected data

Initially the network was trained only with the data from driving in the center of the road in both tracks 1 and 2. With this data the network was able drive in track 1 along corners with large radius but the network could not make sharp turns. And it could not bring the car to the center of the road if it is in the side of the road. To train the network to bring the car to the center of the road recovery data were collected. Recovery data was collected by driving the car to the left of the road and to the right of the road alternatively like a sinusoidal curve. But the data was recorded only when the car was driven towards the center and it was not recorded when car was driven towards the sides. With this data the network could bring the car to the center of the road if it was in the sides and also it was able to drive the sharp corners. Even with recovery data the network was not able to drive completely along the tracks 1 and 2 there were some problem in some sections for example in track 1 there are some turns in the road where the shoulder marking are missing. The network could not drive safely along these sections. After collecting more recovery data in these special sections the network was able to drive safely along both the tracks in the simulator.