Project 3 - Behavioral Cloning

The goals / steps of this project are the following:

- •Use the simulator to collect data of good driving behavior or Use data provided by UDACITY
- •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

Files Submitted

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
- •model.json :- containing saved model in json format
- •writeup_report :- summarizing the results

2. Submssion includes functional code

Using the Udacity provided simulator and my drive.py file, the car can be driven autonomously around the track by executing

```
python drive.py model.json
```

3. Submssion code is usable and readable

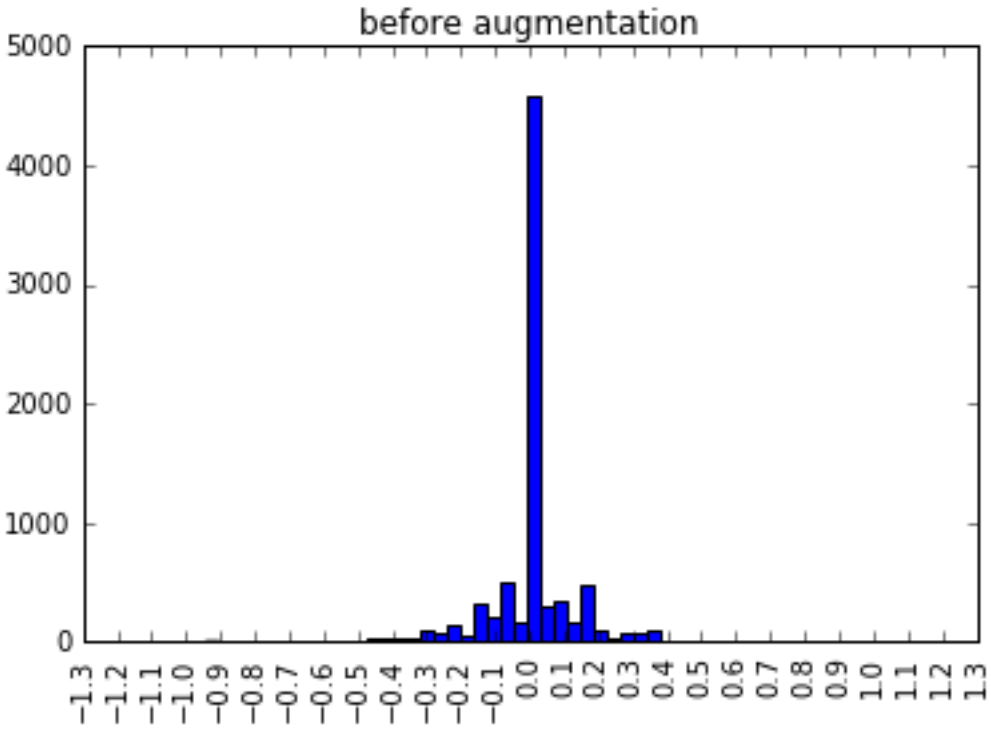
The model.py file contains the code for training and saving the convolution neural network. The file shows the pipeline I used for training and validating the model, and it contains comments to explain how the code works.

Model.py ---

Only udacity dataset of images are used and augmentation is done on images to drive the car round the both track 1 and track 2. Simulator was run on fastest mode as recommended by UDACITY.

In first few lines, libraries are imported which will be needed for this project including numpy,csv,cv2. This is followed by defining the functions which will be used for reducing & cropping & resizing & rgb2yuv conversion & flipping the images. Same functions are used in drive.py as well.

Paths of Images are read from the **driving_log.csv** file provided for project 3. Corresponding images are read from the **data folder** and histogram is plotted to see the variation of no.of images with respect to steering angles.



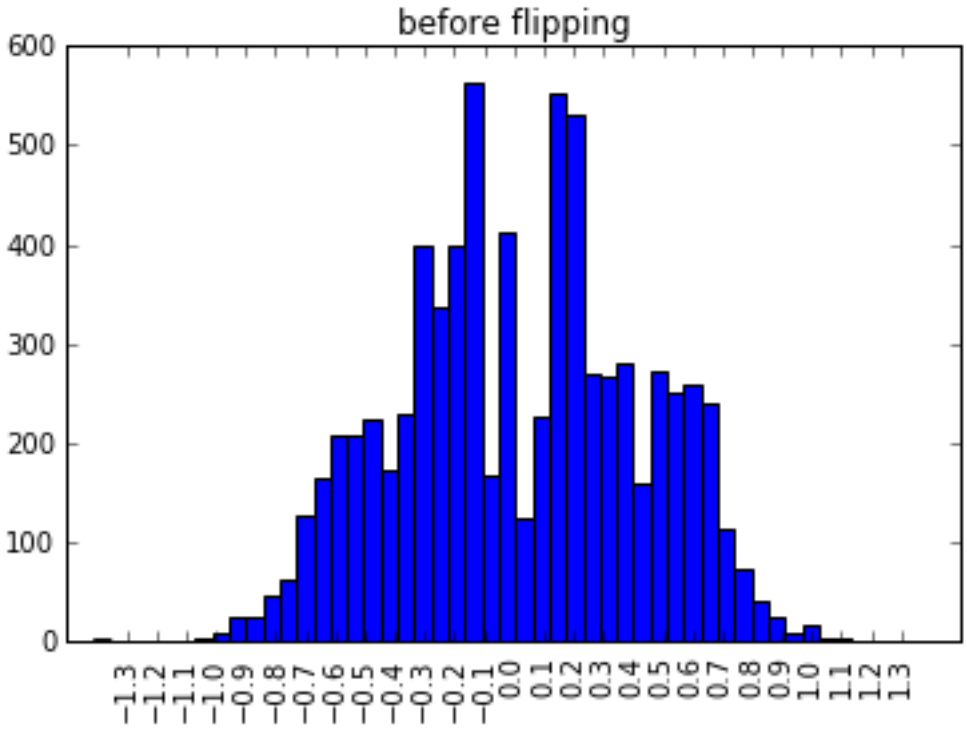
Images with low steering angles are seperated with rest of images. It has been found that there are lot of images with low steering angles. The no.of images with low steering angles

are reduced in quantity using **random function** so that model will not be inclined to low steering angles. Out of approximately 8000 images which were provided in udacity data, **only around 3000 images are kept including low and not low steering angles.** Then these images (which are the images from center camera of car) are combined with similar no. of left camera and right camera images,which makes the total of around 7500 images.

no. of steeringAngle before flipping: 7521

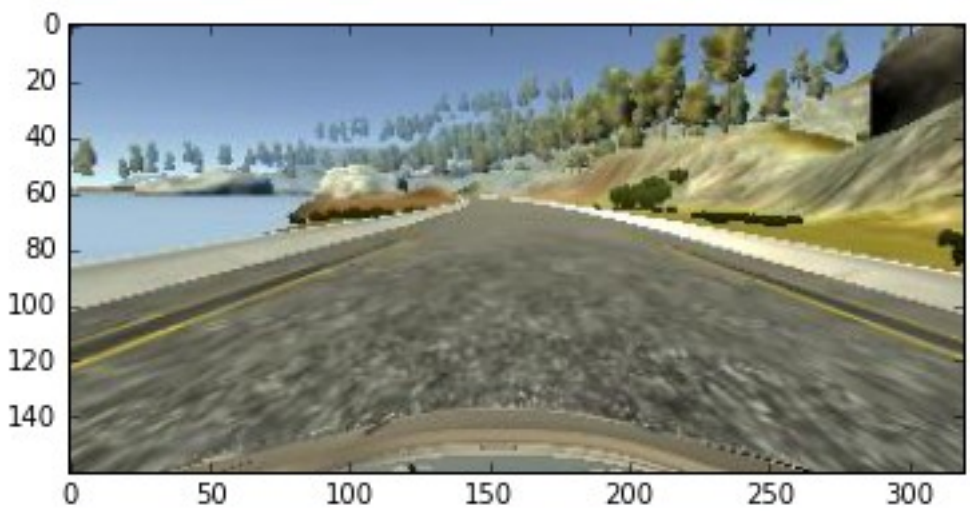
no. of imagesTogether before flipping: 7521

Histogram before flipping images--

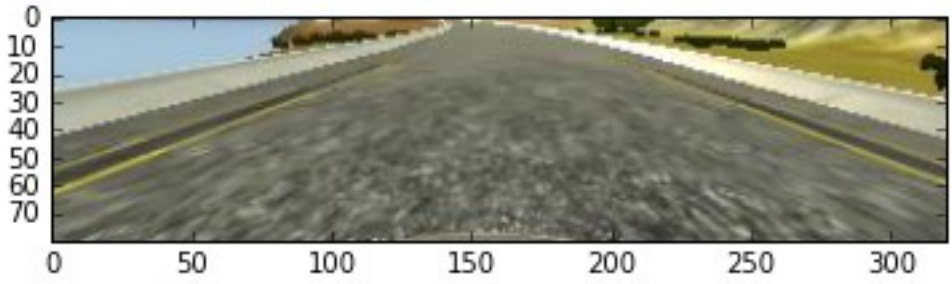


Cropping of images are done to remove unwanted pixel values from top and bottom of image. **Resizing and RGB to YUV conversion of images** are done to match the specifications of images mentioned inm **nvidia model (**explained later on**).**

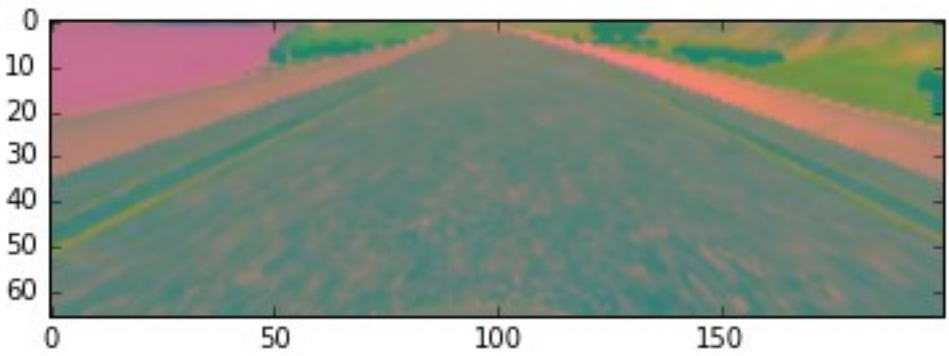
Original Image-



Cropped Image-



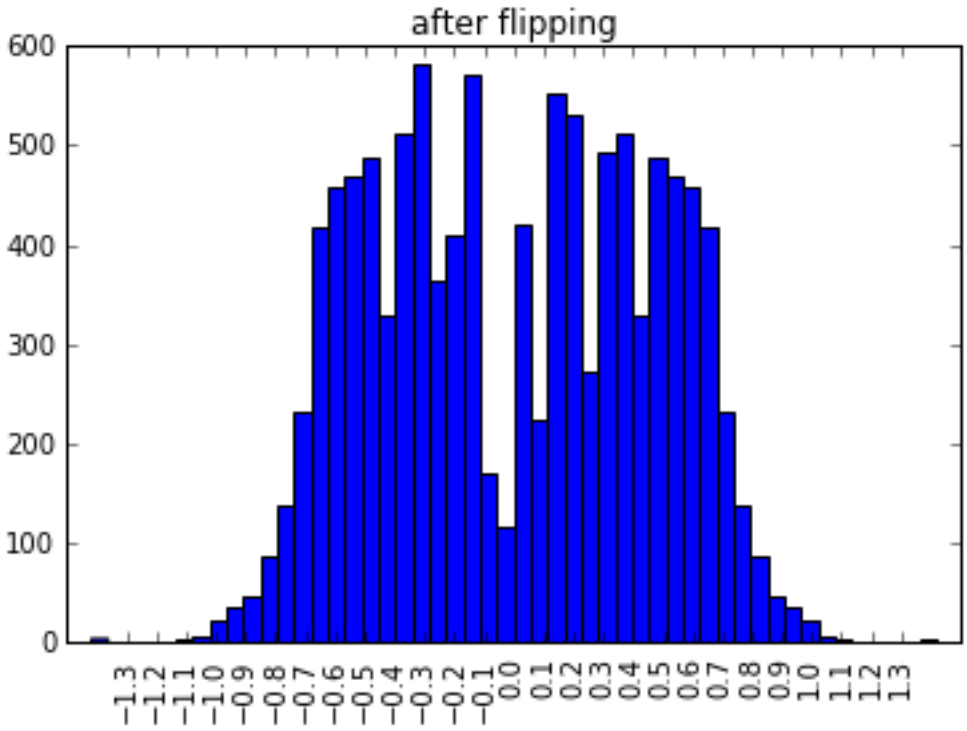
RGB to YUV converted image-



Then no.of images are further augmented using **flipping function for the sharp curves of the road.** After flipping, no. on images become near to 11000. Histogram is shown below.

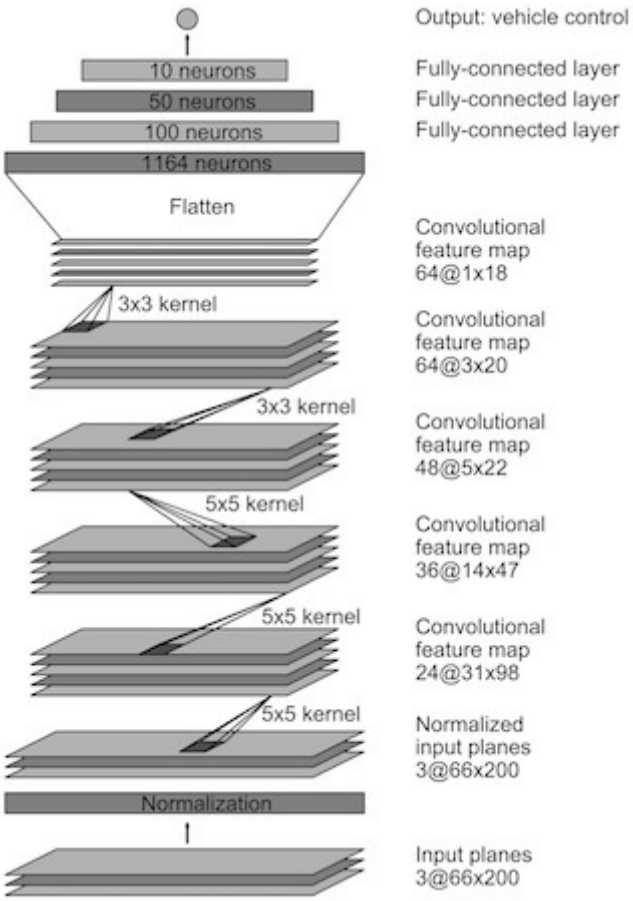
no. of steeringAngle after flipping: 11219

no. of imagesTogether after flipping: 11219



Convnet Architecture

The architecture used in this project is similar to **nvidia model.**



The convnet contains the following nine layers:

- Image preprocessing layer using Lambda layer: making image preprocessing part of the network
- Convolutional layer 1: 24 5x5 kernels, stride 2, activation=relu, dropout=0.2

•Convolutional layer 2: 36 5x5 kernels, stride 2, activation=relu, dropout=0.2

•Convolutional layer 3: 48 5x5 kernels, stride 2, activation=relu, dropout=0.2

•Convolutional layer 4: 64 5x5 kernels, stride 1, activation=relu, dropout=0.2

•Convolutional layer 5: 64 5x5 kernels, stride 1, activation=relu, dropout=0.2

•Flatten layer

•Fully connected layer 1: 100 neurons,, activation=relu, dropout=0.5

•Fully connected layer 2: 50 neurons, activation=relu

•Fully connected layer 3: 10 neurons, activation=relu, dropout=0.5

The last fully connected layer is connected to a single neuron that contains the predicted steering angle.

Loss and Optimizer-

As this project is a regression proble, hence **mean square error(mse)** is udes as loss function and **Adam** optimizer is used with **learning rate=0.0001.**

EPOCHS-

After a lot of traial and run, no. of EPOCHS are taken to be 8 and loss was not decreasing even if no. of EPOCHS are increased.

Python Generator-

Python generator is used to train the model in batches of 32.

Saving the model-

Finally the model is saved in **json and h5 format.**

Youtube link of final running video on track 1 and track 2 --

https://www.youtube.com/watch?v=wrPC7OJWoWg

Simulation Overview shown below---

```
_____
Layer (type)                 Output Shape              Param #     Connected to
=================================================================
=================================================================
======
```

Normalization (Lambda)	(None, 66, 200, 3)	0	lambda_input_1[0][0]
Conv1 (Convolution2D)	(None, 31, 98, 24)	1824	Normalization[0][0]
dropout_1 (Dropout)	(None, 31, 98, 24)	0	Conv1[0][0]
Conv2 (Convolution2D)	(None, 14, 47, 36)	21636	dropout_1[0][0]
dropout_2 (Dropout)	(None, 14, 47, 36)	0	Conv2[0][0]
Conv3 (Convolution2D)	(None, 5, 22, 48)	43248	dropout_2[0][0]
dropout_3 (Dropout)	(None, 5, 22, 48)	0	Conv3[0][0]
Conv4 (Convolution2D)	(None, 3, 20, 64)	27712	dropout_3[0][0]
dropout_4 (Dropout)	(None, 3, 20, 64)	0	Conv4[0][0]
Conv5 (Convolution2D)	(None, 1, 18, 64)	36928	dropout_4[0][0]
dropout_5 (Dropout)	(None, 1, 18, 64)	0	Conv5[0][0]
flatten_1 (Flatten)	(None, 1152)	0	dropout_5[0][0]
FC2 (Dense)	(None, 100)	115300	flatten_1[0][0]
dropout_6 (Dropout)	(None, 100)	0	FC2[0][0]
FC3 (Dense)	(None, 50)	5050	dropout_6[0][0]
FC4 (Dense)	(None, 10)	510	FC3[0][0]
dropout_7 (Dropout)	(None, 10)	0	FC4[0][0]

```
_________________
final_output (Dense)         (None, 1)         11        dropout_7[0][0]
==================================================================
==================================================================
=======
```

Total params: 252,219
Trainable params: 252,219
Non-trainable params: 0

```
_________________
here
Epoch 1/8
11106/11106 [==============================] - 96s - loss:
0.1711 - acc: 0.0267 - val_loss: 0.0907 - val_acc: 0.0088
Epoch 2/8
11106/11106 [==============================] - 94s - loss:
0.1293 - acc: 0.0258 - val_loss: 0.0705 - val_acc: 0.0088
Epoch 3/8
11106/11106 [==============================] - 94s - loss:
0.1237 - acc: 0.0264 - val_loss: 0.0682 - val_acc: 0.0088
Epoch 4/8
11106/11106 [==============================] - 95s - loss:
0.1197 - acc: 0.0257 - val_loss: 0.0662 - val_acc: 0.0088
Epoch 5/8
11106/11106 [==============================] - 94s - loss:
0.1176 - acc: 0.0258 - val_loss: 0.0672 - val_acc: 0.0088
Epoch 6/8
11106/11106 [==============================] - 94s - loss:
0.1168 - acc: 0.0255 - val_loss: 0.0648 - val_acc: 0.0088
Epoch 7/8
11106/11106 [==============================] - 95s - loss:
0.1180 - acc: 0.0260 - val_loss: 0.0636 - val_acc: 0.0088
Epoch 8/8
11106/11106 [==============================] - 94s - loss: 0.1140 - acc: 0.0260 -
val_loss: 0.0632 - val_acc: 0.0088
```