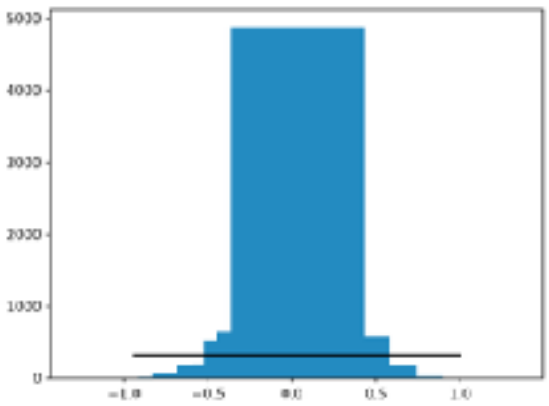
End to End Drive Car using Behavioral Cloning

This project mainly uses behavioral cloning and deep neural networks to drive the car in a simulated environment. It is a very exciting project because what ever we learned so far can be applied to drive the car. Driving the car in a simulator is not same as real car but it will capture the main idea of using behavioral cloning using CNN.

Training Data Analysis

The goal is to collect data while driving in a simulator and train a model to mimic that behavior. Trained the model with Udactiy provided data set instead of my own because it didn't produce good results. I realized while driving a car in a simulator require more practice otherwise collected data will be skewed towards bad dataset, which internally not produced good behavior while driving in autonomous mode.

Dataset has center, left and right images of the road and steering, throttle and speed. Also data is biased towards center lane images. That means most of the time car travelled in center of the lane while capturing the data due to that steering angle for these images are zero. In order to obtain training data, shuffle and split the data into training and validation set as 90/10 split using ***train_test_split*** from ***sklearn***. The below bar graph represents the histogram of steering angle with bin size 25



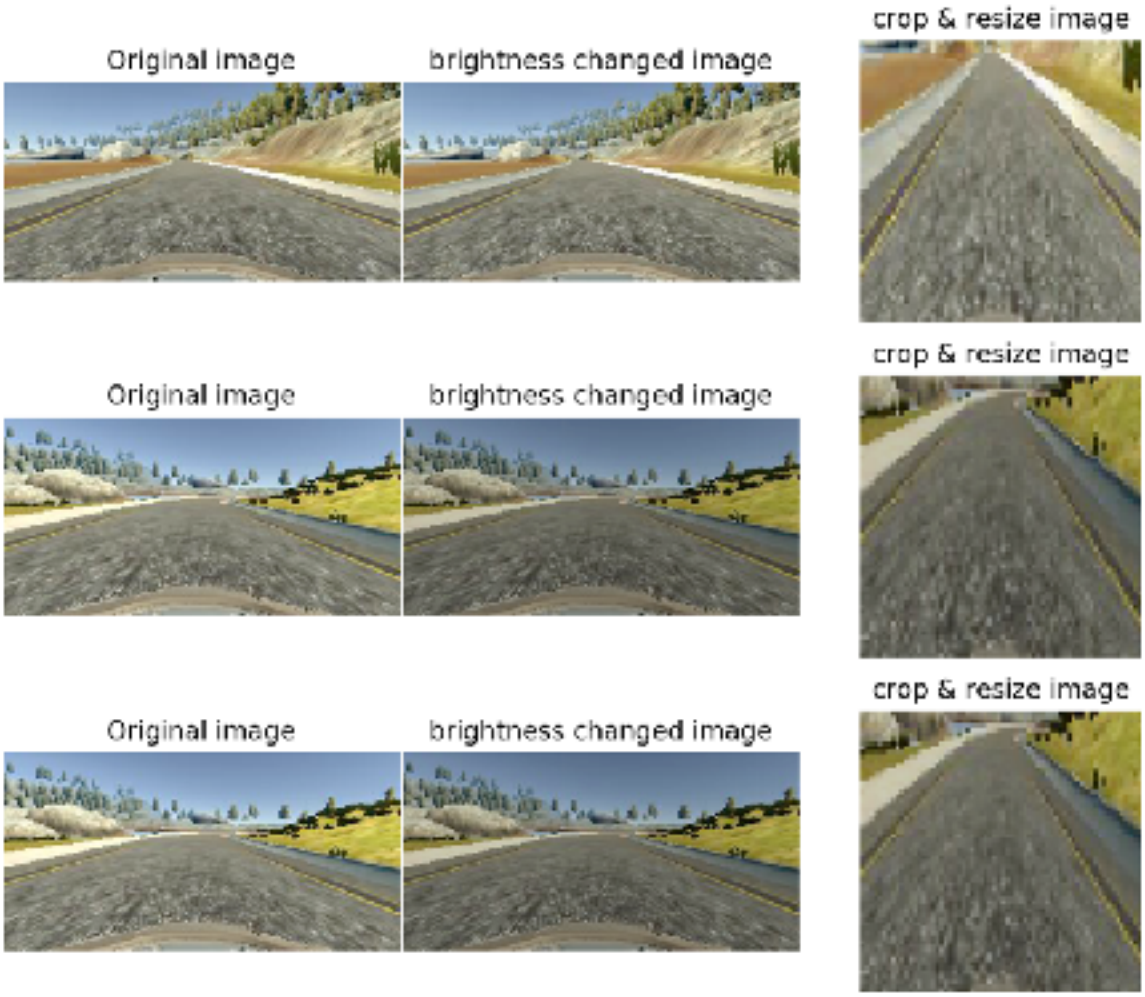
Augmentation

After splitting training data from Udacity data, further applied following rules to augment and filter data

- Separated driving straight, left and right data by applying steering angle, >0.15 as right and <-0.15 as left
- Applied further adjustment angle to driving left and driving right steering angles to recover, if the car deviated from center
- Merged driving straight, left and right as single dataset
- Applied random brightness filter by adjusting V channel
- Randomly flipped some of the images vertically to introduce opposite traffic
- Cropped the image and resize to 64x64. This is mainly to improve the speed of model training

Preprocessing

In order to avoid learning other part of the image cropped the image sky and car deck out of the images. Which reduced the original image from 160x320 to 78x320. To help further, train the model faster, further scaled down to 64x64. Final images looks like



Training the model

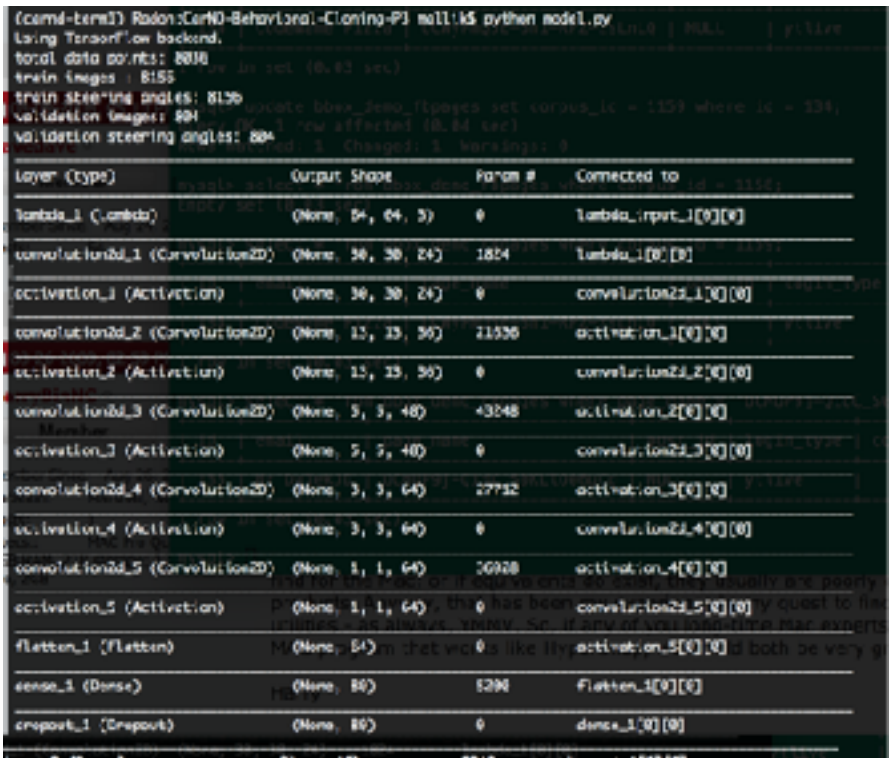
This part took me quite a while for tuning the parameters by trail and error. I picked Nvidia model and modified it. Modified following things to work the model for this dataset

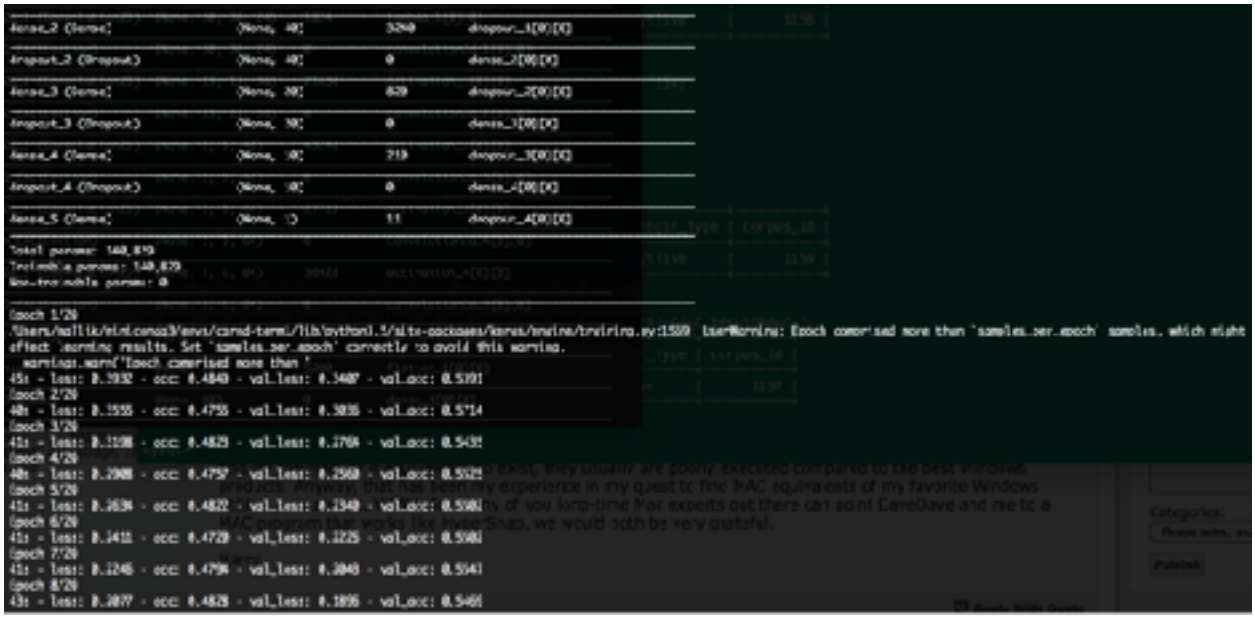
- Original Nvidia model uses 66x200 images size but reduced image size to 64x64 to train faster
- Used Keras fit_generator() with 128 batch size. First started with 64 and settled down with 128
- Applied RELU activation
- Applied 50% dropout and L2 regularized to avoid overfit the data
- Used Adam optimizer with MSE and rate as 0.0001

Model contains following layers

- 1st layer normalized the data
- Three convolution layers with 5x5 filter, 2x2 strides and 24, 36 and 64 depth
- Two convolution layers with 3x3 filter, 1x1 stride and 64 depth
- Flatten out and applied dense layer with 80, 40, 20, 10 and 1. Each layer applied 50% dropout and L2 regularizer

Model output





```
Epoch 9/20
43s - loss: 0.1961 - acc: 0.4807 - val_loss: 0.1786 - val_acc: 0.5480
Epoch 10/20
43s - loss: 0.1832 - acc: 0.4844 - val_loss: 0.1681 - val_acc: 0.5357
Epoch 11/20
43s - loss: 0.1724 - acc: 0.4813 - val_loss: 0.1626 - val_acc: 0.5525
Epoch 12/20
43s - loss: 0.1637 - acc: 0.4830 - val_loss: 0.1505 - val_acc: 0.5268
Epoch 13/20
42s - loss: 0.1549 - acc: 0.4751 - val_loss: 0.1414 - val_acc: 0.5401
Epoch 14/20
43s - loss: 0.1462 - acc: 0.4768 - val_loss: 0.1352 - val_acc: 0.5312
Epoch 15/20
43s - loss: 0.1380 - acc: 0.4806 - val_loss: 0.1306 - val_acc: 0.5268
Epoch 16/20
42s - loss: 0.1301 - acc: 0.4974 - val_loss: 0.1233 - val_acc: 0.5502
Epoch 17/20
43s - loss: 0.1257 - acc: 0.4833 - val_loss: 0.1141 - val_acc: 0.5558
Epoch 18/20
42s - loss: 0.1194 - acc: 0.4862 - val_loss: 0.1107 - val_acc: 0.5458
Epoch 19/20
42s - loss: 0.1123 - acc: 0.4741 - val_loss: 0.1058 - val_acc: 0.5379
Epoch 20/20
41s - loss: 0.1061 - acc: 0.4885 - val_loss: 0.1018 - val_acc: 0.5725
Training finished
Model generation successful
```

Testing

Saved the trained model as a *model.json* and weights in *model.h5*. Changed *drive.py* to scale the image to 64x64, speed and read the trained model from json file.

It worked fine in Track1 but didn't perform well in track 2. Captured video from simulator while driving in autonomous driving mode.

Further improvements

I can spend more time on improving the model to run in track2. Here are the few things didn't consider in current implementation

- Need to apply shadow effects on images to train the model to accommodate night and shadow scenarios
- Tuning model parameters taking lot of time with trail and error. Need more time to tune it
- Collect own training data while driving in simulator and train the model. Some how I didn't get good results with my data. I drove very bad in the simulator, require more time to practice it and collect good data
- Experiment with other models like RNN and comma.ai