Behavioral Cloning Project Overview

This project aims to learn from the human driving behavior using CNN, and drive the car in a simulator provided by Udacity. In the recording mode, the simulator records the training examples (images sampled from video at a frequency of 10 Hz), and the respective steering angle. The throttle and vehicle speed are not taken into consideration in this project. This training data is then fed to the network to build the final model.

Files Submitted

I have updated the following files:

- `model.py` containing the entire CNN architecture and processing pipeline
- `model.h5` the trained convolution neural network `data_pipe.py`, containing code for generators which stream the training images from disk and apply data augmentation
- `drive.py made minor modification to the code in order to change the colourscape of images to HSV`
- `Behavioral Cloning.pdf summarizing the project`
- `run1.mp4 the video recording of model running on Track1 in simulator`

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.h5
```

Implementation

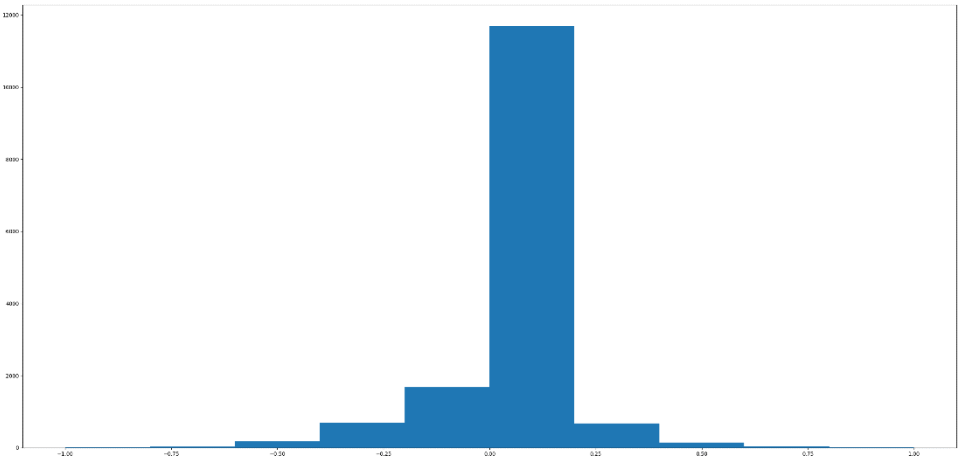
Training Data

A good driving behaviour data is essential for a good mode. I recorded the training data on Track 1. Some of the driving behaviours recorded are

- Two full laps on a nominal speed driving swiftly on the turns.
- One lap in the reverse direction.
- Some recovery driving samples also, where I took the car to one edge of the road, and steered it back to centre position.

Data Visualization

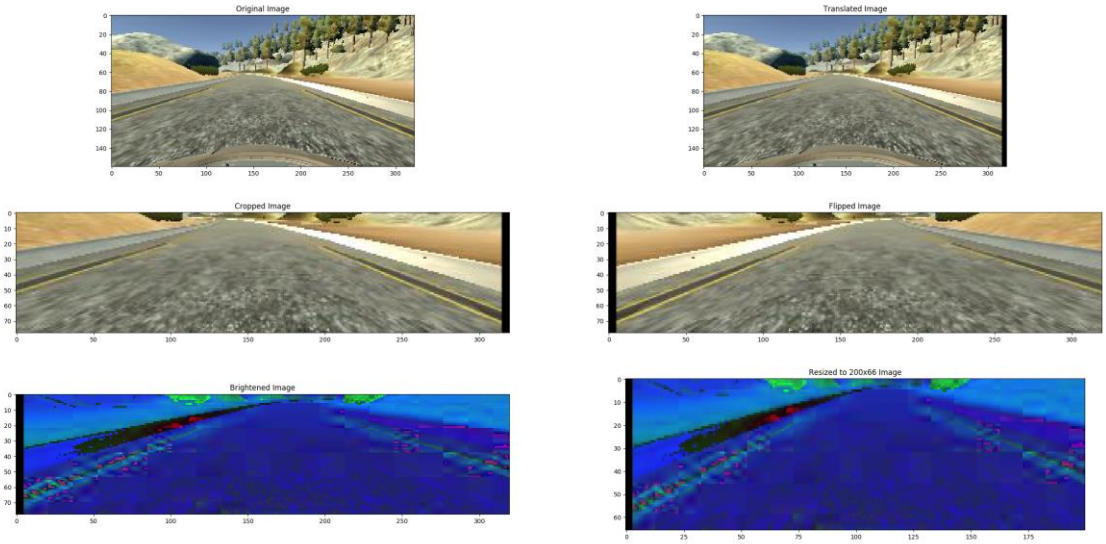
The training data is then plotted to see the spread of steering angles. There is a huge imbalance in the data and most of data is with steering angle of 0. Thereby some pre-processing techniques are applied to have a good data for neural network.



Data Pre-Processing

I have applied the following pre-processing techniques. For some of the techniques referred to existing blogs for help

1. Changed the brightness level of images randomly
2. Added a random translation to image and steering angles
3. Data is normalized with the use of lambda layers.
4. Images and angles are flipped
5. Correction factor of 0.25 is used for images coming from left and right cameras
6. Image is cropped to get the region of interest. Some percentage of image is cropped from top and bottom
7. The input image has some of the area on top and bottom which is not required for this model. Hence image is cropped by removing top ~35% and bottom ~10% area. And then the image is resized to be able to map to NVIDIA architecture.
8. All the images are fed to network in HSV colour space



Network Architecture

Training:

The convolutional neural network architecture is inspired by NVIDIA's End to End Learning for Self-Driving Cars paper. To add non-linearity to model ELU activations are used. And to deal with overfitting L2 regularization and dropouts are added to each convolutional and fully connected layer.

For training batch_size of 256 is selected. Since training data is large and limited memory, fit_generator API from Keras library is used. An Adam optimizer is used for optimization. This requires little or no tuning as the learning rate is adaptive. The model is trained for 10 epochs leading to training loss of 0.2040 and validation loss of 0.1904

Testing:

The trained model is tested on Track 1 and the recording of video is included. The performance is quite smooth with good results. I did not use training samples from Track 2, and testing on Track 2 did not show good results. There is more scope of improvement needed in the model to be able to generalize well for different types of tracks.

Conclusion and Further:

It was interesting to see the improvement in the final model by experimenting with different types of training data and CNN layers.
Initially it was challenging to get the vehicle to stay on the road and it would go out at times. However, after inserting in the recovery data, the performance had improved drastically.

1. Further improvement to the model can be made by making use of advanced image processing API's to augment the data (jitters, rotations,)
2. Since the performance was model was not good on Track 2, the model can further be generalized by making use of other techniques which help to reduce overfitting.
3. It will be interesting to learn later how other factors like speed and throttle are incorporated in a machine learning model.