#**Behavioral Cloning**

##Writeup Template

Behavioral Cloning Project

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

###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:

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

####2. Submission 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_add.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.

###Model Architecture and Training Strategy

####1. An appropriate model architecture has been employed

My model is based primarily on NVIDIA network and modified to make it even more accurate and robust. It consists of a 1by1 conv layer followed by 6 conv layers with initial filter size of 5by5 followed by 3by3.

Model includes ELU layers to introduce non linearity. I tried RELU as well. Does not have much of a performance improvement.

####2. Attempts to reduce overfitting in the model

The model contains several dropout layers in order to reduce overfitting.

The model was trained and validated on udacity's training data set as well as two of the recorded datasets to ensure that the model was not overfitting (code line 10-16). The model was tested by running it through the simulator and ensuring that the vehicle could stay on the track.

####3. Model parameter tuning

The model used an adam optimizer, so the learning rate was not tuned manually.

####4. Appropriate training data

Initially I trained the model on Udacity's dataset which led to some success. Then I gathered additional data to improve the model. This included driving

1) A lap around the track
2) Driving in the opposite direction
3) Recovering from the side lines

For details about how I created the training data, see the next section.

###Model Architecture and Training Strategy

####1. Solution Design Approach

The overall strategy was to first test the NVIDIA architecture on Udacitys training data which was fruitful once I resolved Image colorscale issues as well as scaling and cropping images in the model as well as drive.py in parallel.

Next, the goal was to train the model on additional training set recorded to improve the performance.

I created a generator to sample data from the set for training as well as validation. I used the generator images to gauge how my model was going to see them. Using the fit.generator command, I created a training set to train the model as well as validate validation set simultaneously.

To combat the overfitting, I modified the model so that there were several dropout layers which helped the model to generalize better

The final step was to run the simulator to see how well the car was driving around track one. There were a few spots where the vehicle fell off the track. With the help from fellow Udacity members, I cut the training set where the steering angles were close to zero. I also added left and right camera images with a compensation to the measurement such that if the car sees the left or right image as the center image, it steers hard to get on track. I also used YUV colorscale to better fit the model as suggested in the forums. The biggest improvement I noticed was by cutting the dataset and viewing a smoother Gaussian distribution in the measurements.

At the end of the process, the vehicle is able to drive autonomously around the track without leaving the road.

####2. Final Model Architecture

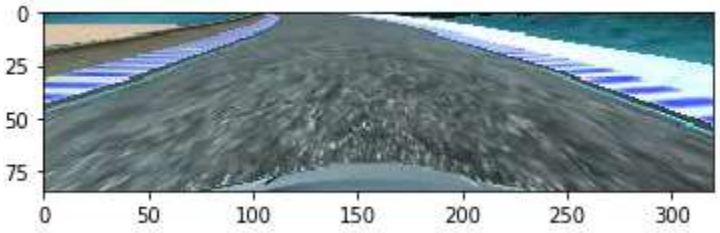
The final model architecture consisted of a convolution neural network with the following layers and layer sizes

1) Convolution 2d – depth-3 , filter (1,1), Activation - ELU
2) Convolution 2d – depth-24 , filter (5,5), Activation - ELU
3) Convolution 2d – depth-36 , filter (5,5), Activation - ELU
4) Dropout (40%)
5) Convolution 2d – depth-48 , filter (5,5), Activation - ELU
6) Convolution 2d – depth-64 , filter (3,3), Activation - ELU
7) Convolution 2d – depth-64 , filter (3,3), Activation - ELU

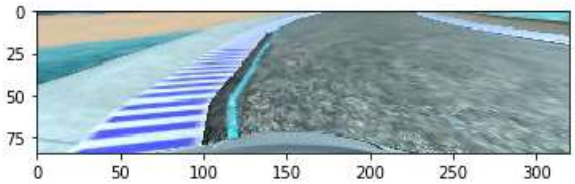
8) Dropout (30%)
9) Dense size- 1164
10) Dense size- 100
11) Dense size- 50
12) Dense size- 10
13) Dense size- 1

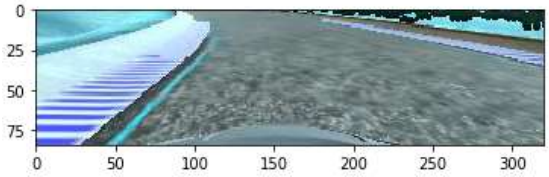
####3. Creation of the Training Set & Training Process

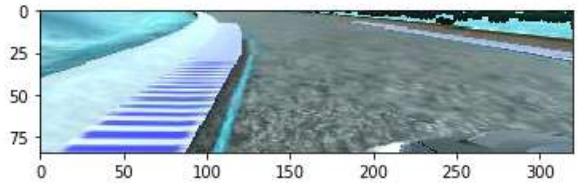
To capture good driving behavior, I first recorded two laps on track one using center lane driving. Here is an example image of center lane driving after cropping the image:



I then recorded the vehicle recovering from the left side and right sides of the road back to center so that the vehicle would learn to recover when it went off the lane. Along with this I attached the left and right camera images to the dataset with a hard steer compensation. The images of that are as below:



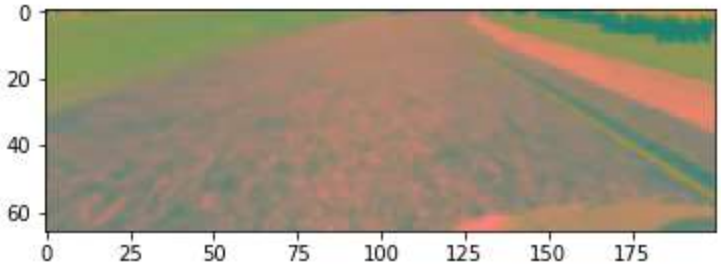


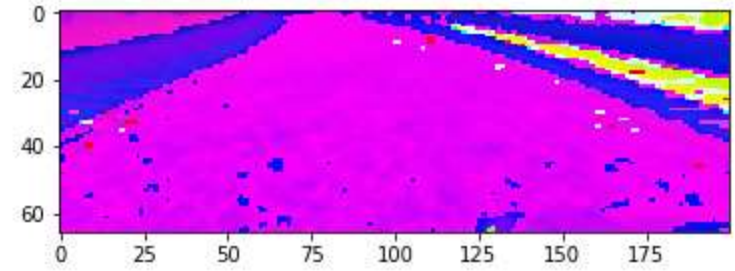


Then I repeated this process on track two in order to get more data points.

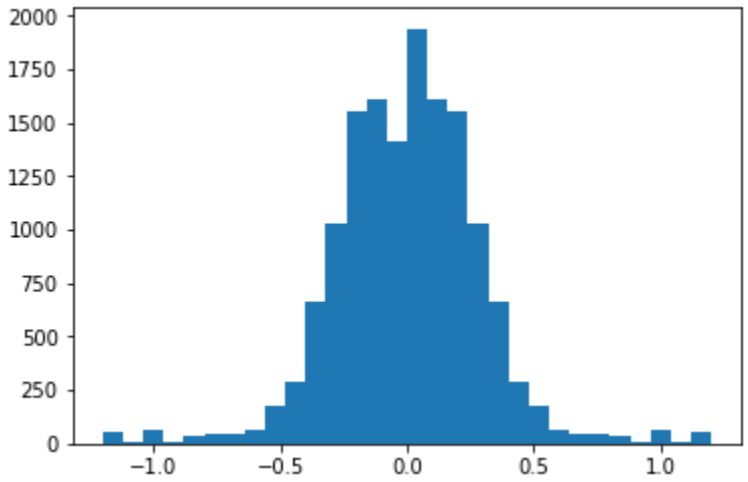
To augment the data sat, I also flipped images and angles that would cover a lot more possibilities and help the model to train.

I then preprocessed this data by adding a Gaussian blur as well as changing the colorscale to YUV along with Normalizing the image. The image below is using YUV scale followed by an image which has been normalized.





I deleted about 85 to 90 percent of the data where the angles were less than 0.03 so that the model does not have a lot of straight steering to train on. I used a histogram to view the distribution and tried to keep it looking like a Gaussian distribution with a large variance.



I finally randomly shuffled the data set and used 30 images per generator execution for validation set.

I used this training data for training the model. The validation set helped determine if the model was over or under fitting. The ideal number of epochs was 6. I used an adam optimizer so that manually training the learning rate wasn't necessary.

Improvements to be made:

Collect more data and train on second circuit

Add augmentation like shadows for which functions are already in place.

Remove the braking issue where the car brakes every second or two.

Increase the speed of the car and have a smooth ride.