Behavioral Cloning Project

Required Files

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.pdf summarizing the results

Quality of Code

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.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

4. An appropriate model architecture has been employed
The model is based on “End to End Learning for Self-Driving Car”

My model consists of a convolution neural network with 3x3 and 5x5 filter sizes and depths between 24 and 64.The activation in every layer was relu. Then four normal networks were added.

The data is normalized using lambda.
model.add(Lambda(lambda x: x/255.0 - 0.5, input_shape=(160,320,3)))

5. Attempts to reduce overfitting in the model

First i think to use dropout layers to avoid overfitting. But i prefer to start with the advice of the lesson "running your network" and use more data, so i started to increase the amount of data using batches of images from the simulation until the behavior was the desire. The model was tested by running it through the simulator and ensuring that the vehicle could stay on the track.

6. Model parameter tuning

The model used an Adam optimizer, so the learning rate was not tuned manually (

7. Appropriate training data

Training data was chosen to keep the vehicle driving on the road. I used a center lane driving images from four different data sets to training the model. The images

was taken while the car was going in the center of the road, when the car was going to the left side and then recovery to the center and when the car was going to the right side and then recovery to the center.

Model Architecture and Training Strategy

1. Solution Design Approach

I started following the examples in the videos of the class until the "Even more powerful network". When i obtained a model that allowed the car to navigate a section of the track i started to add more data until the car was able to cross the bridge. In this section i tried to different versions of data:

- The first was using data from the center, left and right image.
- The second used data only from the center image.

But this two version had troubles crossing the curve next to the bridge. This is when i added the new model based in the article from envidia. This didn't work either. Then I decided to add more data to the model, considering the specific sections when the car had bad behavior.

This allowed me to avoid overfitting and to successfully use the model to navigate the car inside the road. 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 model is based on “End to End Learning for Self-Driving Car”

My model consists of a convolution neural network with 5x5 filter sizes and depth of 24, a convolution neural network with 5x5 filter sizes and depth of 36 and a convolution neural network with 5x5 filter sizes and depth of 48. A subsample of (2,2) was used. Then the model has two convolution neural network with 3x3 filter sizes and depth of 64. The activation in every layer was relu.

```
model.add(Convolution2D(24,5,5, subsample =  (2,2), activation = "relu"))
model.add(Convolution2D(36,5,5, subsample =  (2,2), activation = "relu"))
model.add(Convolution2D(48,5,5, subsample =  (2,2), activation = "relu"))
model.add(Convolution2D(64,3,3, activation = "relu"))
model.add(Convolution2D(64,3,3, activation = "relu"))
```

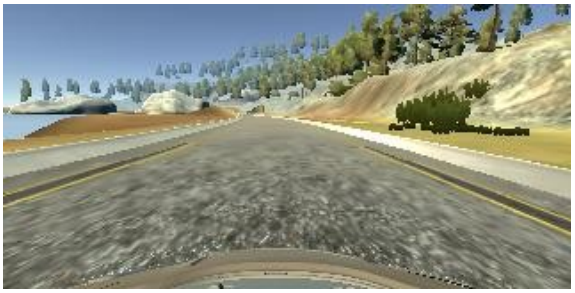
Finally the model has four normal layers.

```
model.add(Dense(100))
model.add(Dense(50))
model.add(Dense(10))
model.add(Dense(1))
```

3. Creation of the Training Set & Training Process

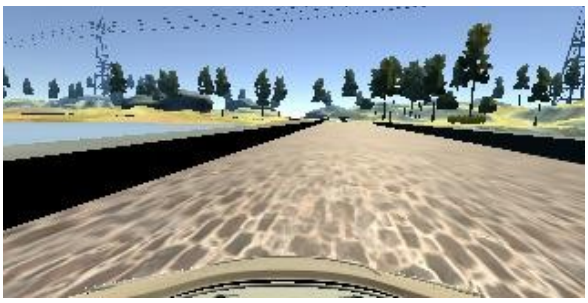
First i used a small amount of data from the simulator. Then i added a full cycle of the road, next i added a specific section of the road when the car was having troubles. Finally i added more data from sections when the car was off the road.

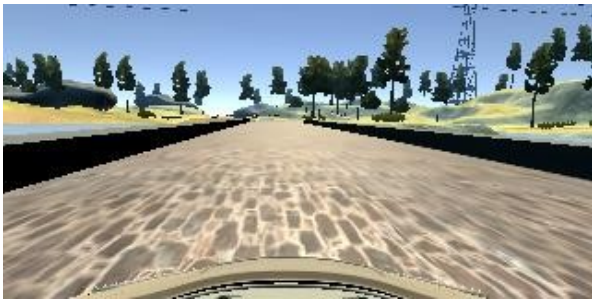
The first data set contains almost 2 laps from the road. Here are a couple of images



Second i recorded a new lap of the road. In this part the travel was made (by mistake) near the left lane line

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 how to do it. This was made mainly in the bridge.





Then i collected two sets more of data from specific parts of the road where the car was having trouble.

I finally randomly shuffled the data set and put 20% of the data into a 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 5 as evidenced by having near zero value of loss for the validation set. I used an Adam optimizer so that manually training the learning rate wasn't necessary.