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

Here I will consider the [rubric points](https://review.udacity.com/#!/rubrics/432/view) individually and describe how I addressed each point in my implementation.

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:
* 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.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.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 was inspired by Nvidia’s CNN for Self-driving car (described in https://images.nvidia.com/content/tegra/automotive/images/2016/solutions/pdf/end-to-end-dl-using-px.pdf)

The model consists of 5 convolution layers, followed by a dropout layer and 4 dense fully connected layers.

In the convolution network, the first three layers use a 2x2 stride and a 5x5 kernel size while the next two layers use a non-strided convolution with a 3×3 kernel size.

The model also performs normalization in the second layer after performing cropping in the first.

The model is defined in model.py (164 - 178).

2. Attempts to reduce overfitting in the model

The model contains dropout layers in order to reduce overfitting (model.py lines 173).

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 and the default learning rate worked well for my case (model.py line 178)

4. Appropriate training data

Training data was chosen to keep the vehicle driving on the road. I used a combination of center lane driving, recovering from the left and right sides of the road.

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

Model Architecture and Training Strategy

1. Solution Design Approach

As a first base model I chose the established Nvidia self-driving CNN model.

For my initial data collection strategy I used the guidelines in the Udacity self driving course.
I collected the following data:

a. Two laps of center lane driving
b. One lap of counter clockwise centre lane driving

c. one lap of recovery driving from the sides
d. one lap focusing on driving smoothly around curves
e. One lap going around curves smoothly in counter clockwise direction.

With this I noticed that the model had trouble navigating the dirt boundary, a patch of road with tree shadow on it and the left turn after the dirt boundary.

I decided to provide more data for these scenarios by doing the following for each:

a. Drove clockwise on the same patch starting at laterally different start points.
b. Drove counter clockwise on the same patch starting at laterally different start points.
c. Recorded a few recovery maneuvers.

I retrained the model with this data with bad results. The car started to go outside the road boundaries in more situation. So I decided to use this newly collected data only for retraining and with much fewer epochs.

2. Final Model Architecture

I did not modify the architecture of my base model.

3. Creation of the Training Set & Training Process

Apart from the data collection techniques described above I also chose to augment my data by flipping the images and the sign of the corresponding angle also using the camera images from the left and the right camera and adding 0.2 and subtracting 0.2 from the corresponding angles. These were arbitrarily chosen numbers that gave good results. (model.py line 140 - 142).

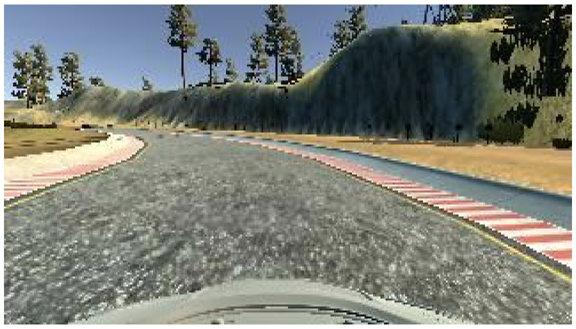
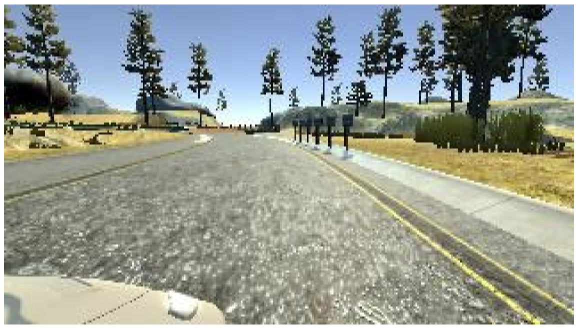
This improved the model performance by making the car more steady and helped in recovery from the sides.

I also performed normalization of the data by scaling the pixels between -1.0 to 1.0. (model.py line 166)

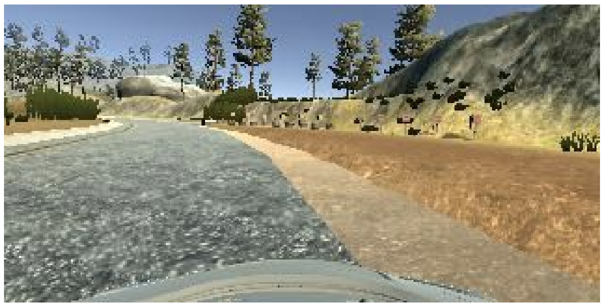
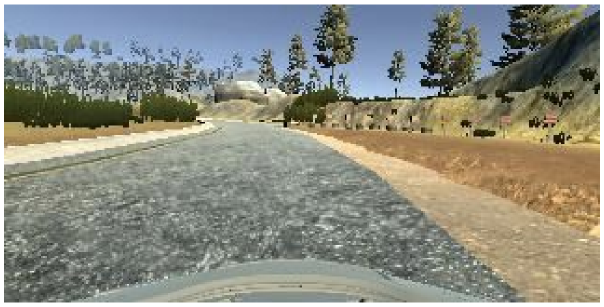
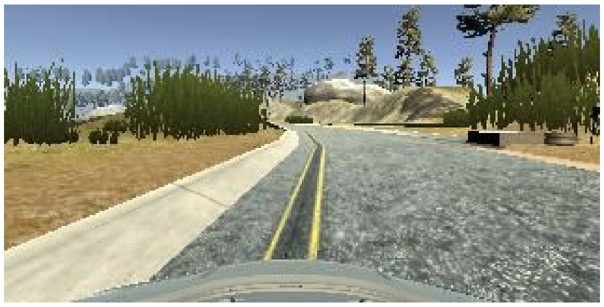
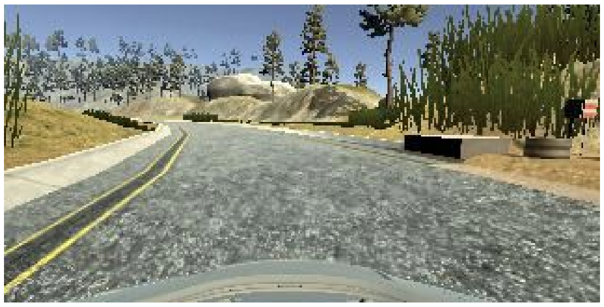
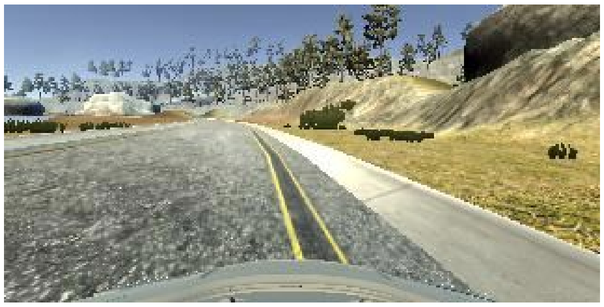
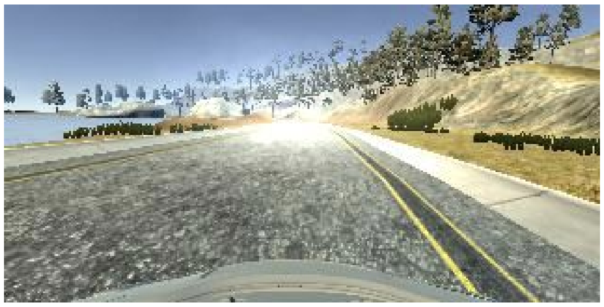
I also performed cropping of the image to remove background noise. (model.py line 165)

I created the validation set by assigning 20 percent of the data set as the validation set. (model.py line 118).

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:

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 .... These images show what a recovery looks like starting from ... :

Driving to the side	**Recovery In progress**
	
	
	

To augment the data sat, I also flipped images and angles thinking that this would.
For example, here is an image that has then been flipped:

Image	Flipped Image