Behavioral Cloning Project

- Subhendu Mishra (subhendu.mishra20@gmail.com)

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

###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.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 consists of 10 layers. First is the cropped image layer where the image cropped to include the area consisting primarily of the road and road signs. 2nd layer is the normalization layer using keras lambda with output shape of (64,64,3). 3rd layer is a convolutional layer with (5x5) kernel and output of (30,30,24). 4th layer is a convolutional layer with (5x5) kernel and output size of (13,13,36). 5th layer is again a convolutional layer with (5x5) kernel and output size of (5,5,48). Here I tried to implement Dropout, but the car is deviating in the simulation, so I chose to omit it.

Next 6th layer is a convolutional layer with (3x3) kernel and output layer of size (3,3,64). 7th layer is also a convolutional layer with kernel size (3x3) and output of (1,1,64). All the convolutional layers from 3 – 7 include RELU layers to introduce nonlinearity. Layers 8 – 10 are fully connected layers giving more flexibility to the neural network.

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

Dropout was added to reduce overfitting, but the vehicle went off course each time dropout layer was used. So, finally the Dropout layer was removed.

The model was trained and validated on different data sets to ensure that the model was not overfitting. 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

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

The overall strategy for deriving a model architecture was to ...

My first step was to use a convolution neural network model similar to the Nvidia architecture. I thought this model might be appropriate because the model worked well with as image classifier for traffic signs.

In order to gauge how well the model was working, I split my image and steering angle data into a training and validation set. I found that my first model had a low mean squared error on the training set but a high mean squared error on the validation set. This implied that the model was overfitting.

To combat the overfitting, I added more data for the model to process. I used images from all the three cameras mounted on the vehicle. I also augmented the data by flipping each image, thus training the model 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 especially around the bridge. To improve the driving behavior in these cases, I change the dataset deliberately driving along the edges of the bridge to train the model better.

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 (model.py) consisted of a convolution neural network with the following layers and layer sizes:

Layer 1 has the cropped image removing the part which does not include the road

Layer 2 adds the normalized and resized plane 3@64x64

Layer 3 Convolutional Feature Map input = (64,64,3) output = (30,30,24)

Layer 4 Convolutional Feature Map input = (30,30,24) output = (13,13,36)

Layer 5 Convolutional Feature Map input = (13,13,36) output = (5,5,48)

Layer 6 Convolutional Feature Map input = (5,5,36) output = (3,3,64)

Layer 7 Convolutional Feature Map input = (3,3,36) output = (1,1,64)

Layer 8 Fully connected layer

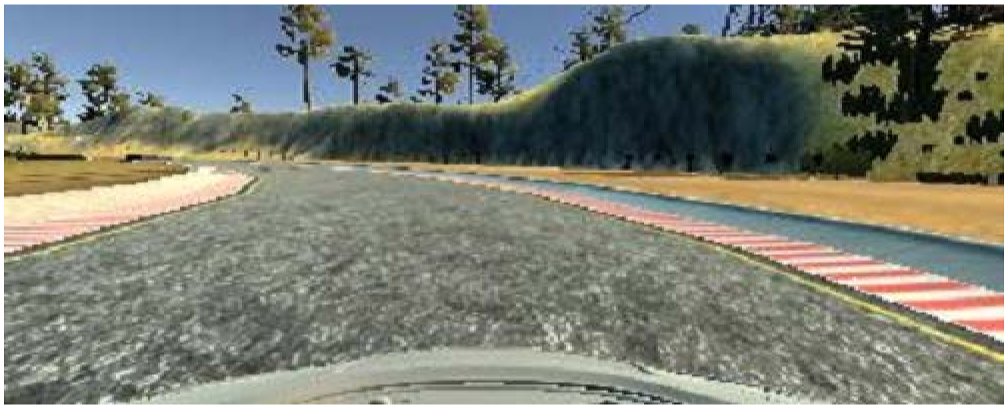
Layer 9 Fully connected layer

Layer 10 Fully connected layer

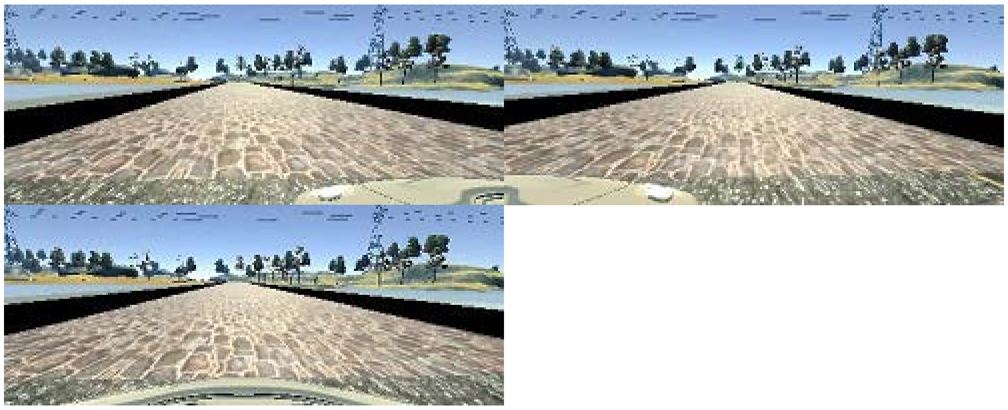
Output: Vehicle Control

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

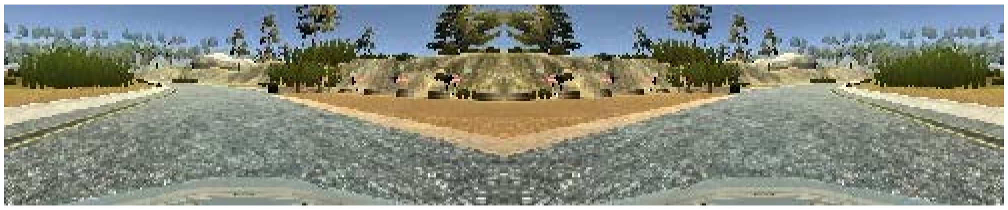
To capture good driving behavior, I first recorded two laps on track one using center lane driving. Here is an example image of center lane driving:



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 stay in middle of the road. These images show what a recovery looks like starting from being too near the left and far away from the right to being in the center of the road.



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:



After the collection process, I had 40,488 number of data points. I then preprocessed this data by normalizing the images and resizing each image to (64x64x3) 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 7 as evidenced by constant value of validation loss. I used an adam optimizer so that manually training the learning rate wasn't necessary.