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

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
- video.mp4 containing the video of track 1 in autonomous mode
- video_track2.mp4 containing the video of track 2 in autonomous mode

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 a convolution neural network with 5x5 and 3x3 filter sizes and depths between 24 and 64 (model.py lines 91-95)

The model includes RELU layers to introduce nonlinearity (code line 91-95), and the data is normalized in the model using a Keras lambda layer (code line 85).

2. Attempts to reduce overfitting in the model

The model was trained and validated on different data sets to ensure that the model was not overfitting (code line 26-68). 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 (model.py line 108).

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 and also the flipped data for center camera images.

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 use multiple convolutional layers to detect the road edges/lanes. Make the model deep enough so that it can learn how to drive by using correct steering angle.

My first step was to use a convolution neural network model similar to the LeNet having 2 convolutional layer and 3 fully connected layers. I thought this model might be appropriate because it has capabilities to detect the shapes and objects.

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 modified the model to add more convolutional layers and shuffled the data. Also normalized the data with Lambda layer.

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. To improve the driving behavior in these cases, I used data from right and left cameras with correction (+-0.1). Also used flipped center camera image data.

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

2. Final Model Architecture

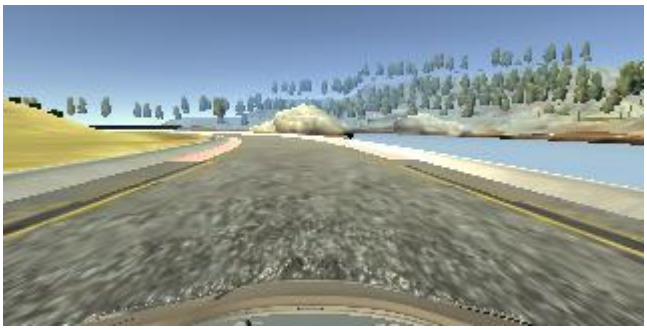
The final model architecture (model.py lines 91-105) consisted of a convolution neural network with the following layers and layer sizes:

- Convolutional layer with 24 depth and 5x5 filter
- Convolutional layer with 36 depth and 5x5 filter
- Convolutional layer with 48 depth and 5x5 filter
- Convolutional layer with 64 depth and 3x3 filter
- Convolutional layer with 64 depth and 3x3 filter
- Fully Connected layer with output 100
- Fully Connected layer with output 50
- Fully Connected layer with output 10
- Fully Connected layer with output 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:


I then recorded the vehicle driving in the reverse direction to cover more data and more examples of driving behavior for right turns. These images show 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.

After the collection process, I had 27680 number of data points. I then preprocessed this data by normalizing it using Lambda function. I also cropped the top 70 and bottom 20 pixels form each Image.

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 were 5. I used an adam

optimizer so that manually training the learning rate wasn't necessary. Below is the result of each Epoch’s training loss and validation loss.

```
Using TensorFlow backend.
Epoch 1/5
27680/27680 [==============================] - 601s - loss: 0.0122 - val_loss: 0.0109
Epoch 2/5
27680/27680 [==============================] - 598s - loss: 0.0101 - val_loss: 0.0110
Epoch 3/5
27680/27680 [==============================] - 609s - loss: 0.0094 - val_loss: 0.0107
Epoch 4/5
27680/27680 [==============================] - 618s - loss: 0.0091 - val_loss: 0.0125
Epoch 5/5
27680/27680 [==============================] - 628s - loss: 0.0089 - val_loss: 0.0125
```