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

###Here I will consider the rubric points 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 consists of a convolution neural network with 3x3 and 5x5 filter sizes and depths between 24 and 64 (model.py lines 79-87)

The model includes ELU layers to introduce nonlinearity (the benefits of ELU's over ReLU's have been published), and the data is normalized in the model using a Keras lambda layer (code line 18).

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

The model is mostly inspired from the Nvidia one and, as a matter of fact, doesn't contains dropout layers in order to reduce overfitting (model.py lines 52-59).

The model was trained and validated on different data sets 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.

To combat the overfitting, I also reduced the number of epochs when it was necessary.

####3. Model parameter tuning

The model used an adam optimizer, so the learning rate was not tuned manually (model.py line 90).

####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 by adding 0.1rad to the left steering angle and subtracting 0.1rad to right one.

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

###Model Architecture and Training Strategy

####1. Solution Design Approach

My first step was to use a convolution neural network model similar to the Nvidia convent architecture, I thought this model might be appropriate because it achieve a low validation loss with a low overfitting. In addition, it is pretty fast to compute due to its few parameters in comparison with other Convnets.

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 so that the number of features be higher. I mainly achieved it by driving the car counter wise on the track.

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 modified the model so that the number of features be higher by taking the right and left camera features and associate it to a modified steering angle (+/-0.1 rad)

At the end of the process, the vehicle was 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

Layer (type)	Output Shape	Param #	Connected to
lambda_1 (Lambda)	(None, 160, 320, 3)	0	lambda_input_1[0][0]
cropping2d_1 (Cropping2D)	(None, 65, 320, 3)	0	lambda_1[0][0]
convolution2d_1 (Convolution2D)	(None, 31, 158, 24)	1824	cropping2d_1[0][0]
convolution2d_2 (Convolution2D)	(None, 14, 77, 36)	21636	convolution2d_1[0][0]
convolution2d_3 (Convolution2D)	(None, 5, 37, 48)	43248	convolution2d_2[0][0]
convolution2d_4 (Convolution2D)	(None, 3, 35, 64)	27712	convolution2d_3[0][0]
convolution2d_5 (Convolution2D)	(None, 1, 33, 64)	36928	convolution2d_4[0][0]
flatten_1 (Flatten)	(None, 2112)	0	convolution2d_5[0][0]
dense_1 (Dense)	(None, 100)	211300	flatten_1[0][0]
dense_2 (Dense)	(None, 50)	5050	dense_1[0][0]
dense_3 (Dense)	(None, 1)	51	dense_2[0][0]

Total params: 347,749

Here is a visualization of the training and validation loss over the epochs

```
Epoch 1/7
19284/19284 [==============================] - 218s - loss: 0.0127 - val_loss: 0.0103
Epoch 2/7
19284/19284 [==============================] - 219s - loss: 0.0111 - val_loss: 0.0098
Epoch 3/7
19284/19284 [==============================] - 218s - loss: 0.0108 - val_loss: 0.0097
Epoch 4/7
19284/19284 [==============================] - 214s - loss: 0.0105 - val_loss: 0.0095
Epoch 5/7
19284/19284 [==============================] - 211s - loss: 0.0103 - val_loss: 0.0096
Epoch 6/7
19284/19284 [==============================] - 216s - loss: 0.0100 - val_loss: 0.0093
Epoch 7/7
19284/19284 [==============================] - 216s - loss: 0.0098 - val_loss: 0.0097
```

####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:

Note that I hadn't had to record the vehicle recovering from the left side and right sides of the road as it was driving well without it.

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

I finally haven't repeated this process on track as it as causing underfitting.

With my model I hadn't to flip anything though I must admit the model, even if it drove perfectly on track one, drove terribly on track two.

After the collection process, I had X number of data points. I then preprocessed this data by ...

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 the ratio between the training and validation loss which was very low on epoch 7. I used an adam optimizer so that manually training the learning rate wasn't necessary.