#Behavioral Cloning****

##Writeup Template

Rubric Points

###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

* **"BehavioralCloning.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

####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

Similar with NVIDA model, the model consists of a convolution neural network with **three times 5x5 filter and 2 times 3x3 filter** (model.py **lines 299-333**) **.**

The model includes **RELU** layers to introduce nonlinearity , and the data is normalized in the model using a Keras lambda layer (code line **from 316 to 320**).

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

The model contains dropout layers in order to reduce **overfitting (model.py lines between 337 and 341). Both L2 regulation and dropout are good approaches to reduce overfitting.**

The model was trained and validated on **different data sets** to ensure that the model was not overfitting (code line **between 31 and 54**).

####3. Model parameter tuning

The model used an adam optimizer, so **the learning rate was manually tuned to 0.0001.** (**model.py line 344**).

####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.

0.20 is added to the steering angle for the images from left camera.

0.20 is reduced to the steering angle for the images from right camera. See model.py line 87 to 91.

Recovering data was constructed (line 119 to 137).

Besides, **three different data sets are used for training including udacity data, normal driving data and data with opposite driving direction**.

###Model Architecture and Training Strategy

####1. Solution Design Approach

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

My first step was to **testing the simple neural from udacity and NVIDA models. I implemented them. However, RGB color was not converted from BGR in "drive.py". The parameter tuning was not correctly feedbacked due to this color conversion issue. A big lesson learned.**

To combat with overfitting, I used dropout and L2 regularization.

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 the lake in the last corner**. to improve the driving behavior in these cases, **I had to increase the steering angle correction from 0.1 to 0.2 for images from left and right cameras. See line 96 and 99**

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

####2. Final Model Architecture

I tried basic neural network from Udacity and NVIDA model architecture. These two approaches were not successfully due to color conversion issues in drive.py. I also tried comma.ai approach. However, I did not have time to improve it.

Due to issue of local carnd-term1 environment setup, I was not able to use padding feature until 20 hours ago. Finally, I had to borrow An Nguyen's model structure (NIVDA model architecture) for using his padding feature.

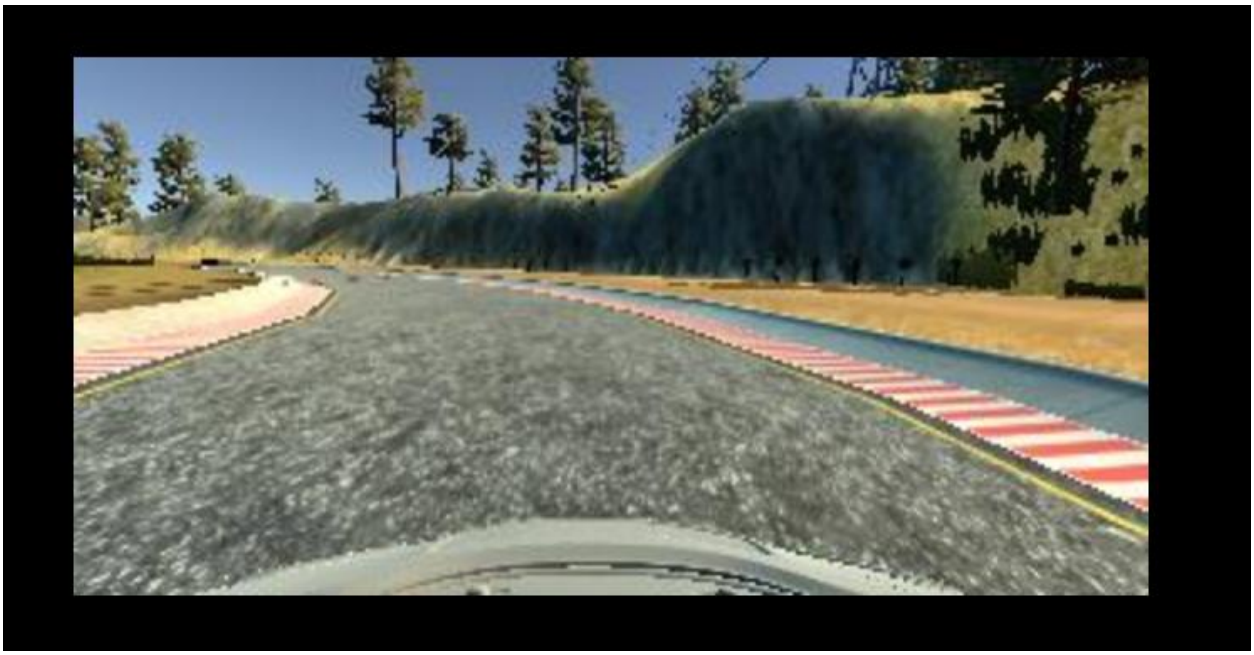
The final model architecture consisted of **three times 5x5 filter and 2 times 3x3 filter (model.py lines 299-333).**

Here is a visualization of the architecture (note: visualizing the architecture is optional according to the project rubric)

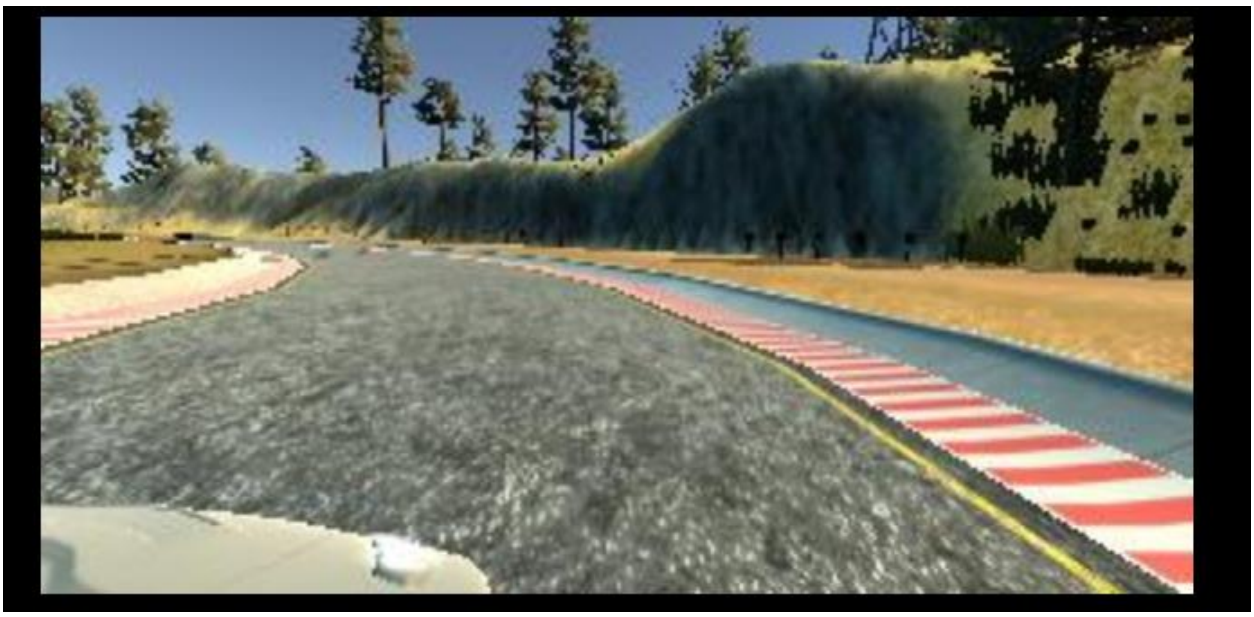
Layer (type)	Output Shape	Param #
lambda_2 (Lambda)	(None, 64, 64, 3)	0
conv2d_1 (Conv2D)	(None, 30, 30, 24)	1824
activation_1 (Activation)	(None, 30, 30, 24)	0
conv2d_2 (Conv2D)	(None, 13, 13, 36)	21636
activation_2 (Activation)	(None, 13, 13, 36)	0
conv2d_3 (Conv2D)	(None, 5, 5, 48)	43248
activation_3 (Activation)	(None, 5, 5, 48)	0
conv2d_4 (Conv2D)	(None, 3, 3, 64)	27712
activation_4 (Activation)	(None, 3, 3, 64)	0
conv2d_5 (Conv2D)	(None, 1, 1, 64)	36928
activation_5 (Activation)	(None, 1, 1, 64)	0
flatten_1 (Flatten)	(None, 64)	0
dense_1 (Dense)	(None, 80)	5200
dropout_1 (Dropout)	(None, 80)	0
dense_2 (Dense)	(None, 40)	3240
dropout_2 (Dropout)	(None, 40)	0
dense_3 (Dense)	(None, 16)	656
dropout_3 (Dropout)	(None, 16)	0
dense_4 (Dense)	(None, 10)	170
dense_5 (Dense)	(None, 1)	11

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

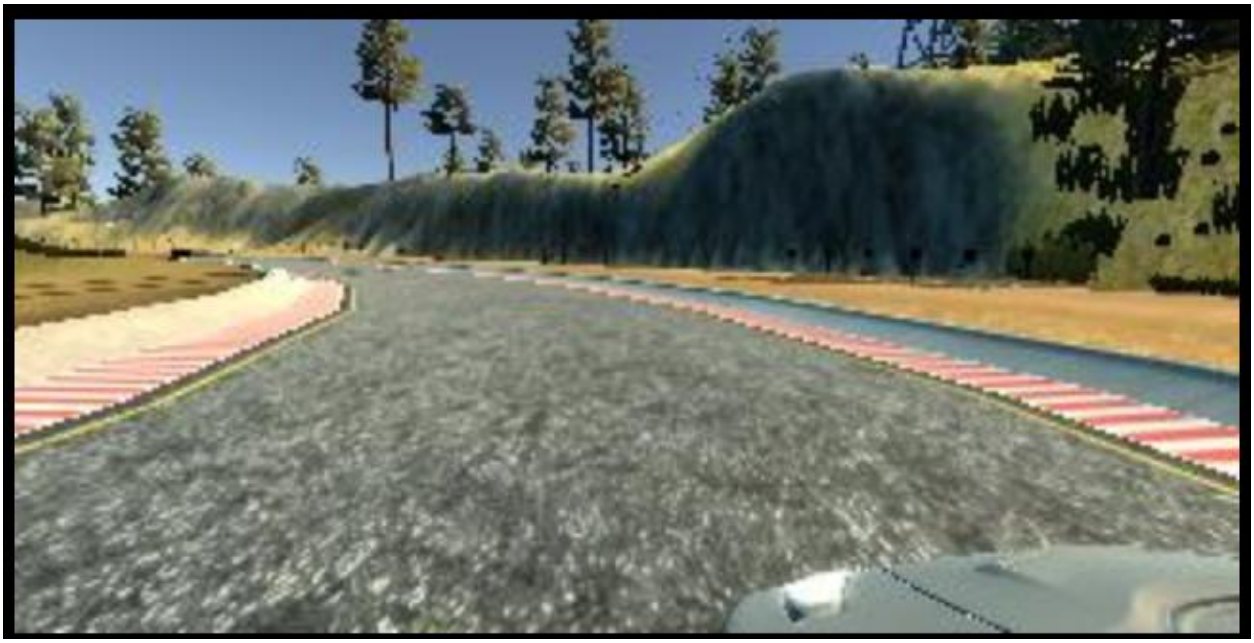
To capture good driving behavior, I first recorded **three** laps on track one using center lane driving. Here is an example image of center lane driving:



I also collected the images from left and right cameras, see image from right camera in below:



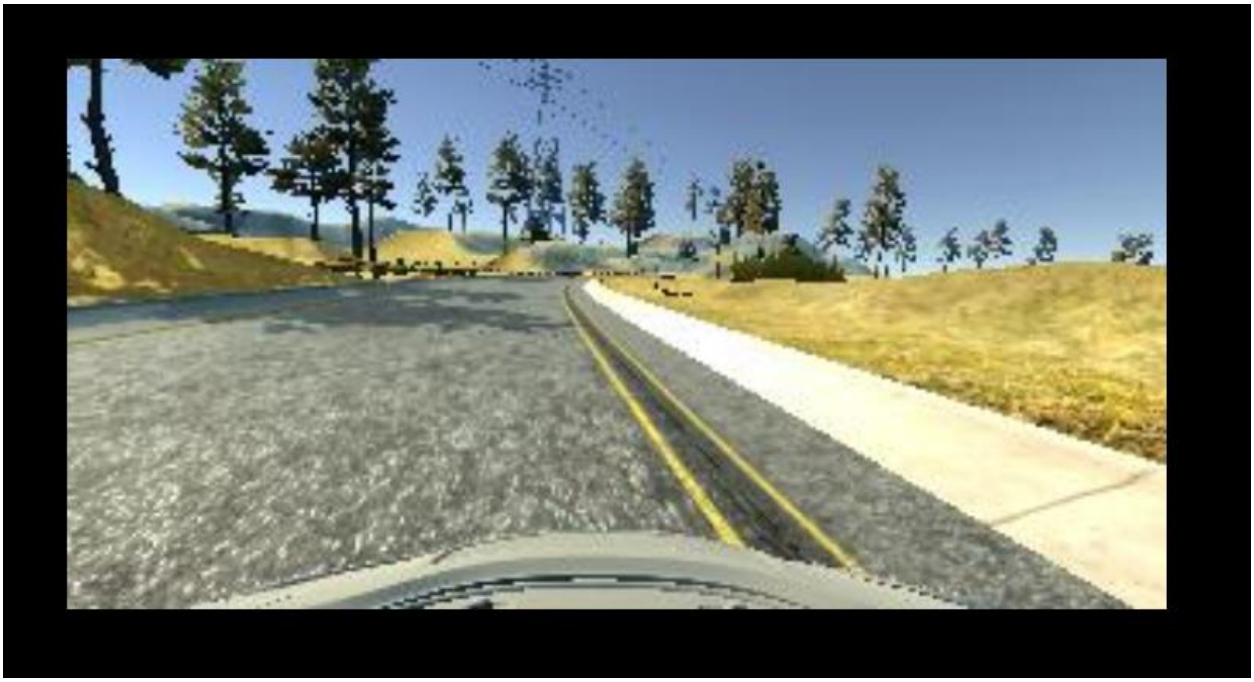
An example image from left camera in below:



Then, I reduced steering angle 0.1 for the image from right camera and add steering angle 0.1 for the left camera. Then I repeated this process in order to get more data points.

To augment the data sat, I also **randomly flipped images (code line from 248 to line 251) during training process.**

I also drive 3 rounds in opposite direction to generalize the training data, see an example image in below:



Finally, I have 7945 images per epoch for training.

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.

Initially I set epoch number as 20. However, I found the driving performance is better with lower epoch number, even if the training loss and validation loss is still high. Finally, 4 was chosen as the epoch number.

For Adam optimizer, I manually set it to 0.0001.