Behavioral Cloning

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

####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 weights
* writeup_report.pdf summarizing the results
* video.mp4 a video recording the vehicle driving autonomously for one lap

####2. Submission includes functional code
Using the Udacity provided simulator and my drive.py file with some modification to make it to load weights to model instead of loading model because I could not save the model when I used Keras Lambda, the car can be driven autonomously around the track by executing

python drive.py model.h5

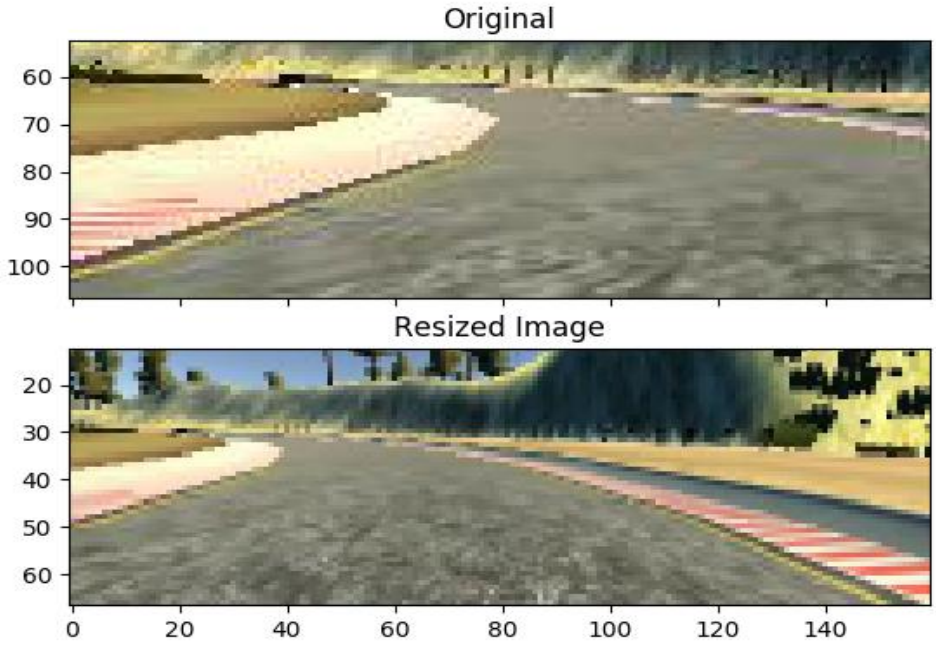
####3. Submission code is usable and readable

The model.py file contains the code for training and saving the convolution neural weights. The file shows the pipeline I used for training and validating the model, and it contains comments to explain how the code works.

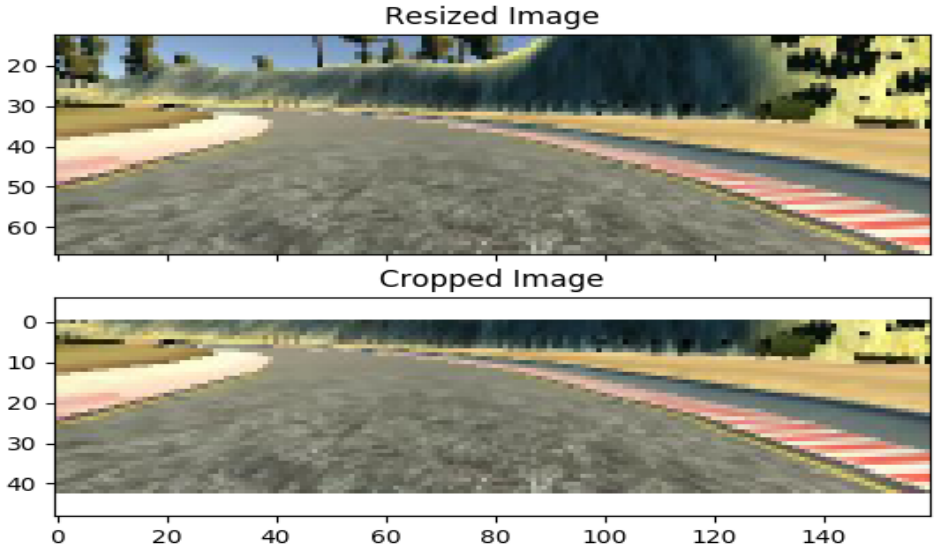
###Model Architecture and Training Strategy

####1. An appropriate model architecture has been employed

First I resized the input by half to increase the training process using Kera lambda and tensorflow resize_image function (code line 73). Here is the example



Then I cropped the image 25 pixels from top and 12 pixels from the bottom using Keras Cropping2D (code line 74).



Furthermore, I scale the data to (-1.0 and 1.0) using Keras Lambda (code line 75).

I used NVIDIA model which consists of a convolution neural network with 5x5 filter sizes and depths between 24 and 48, following by 3x3 filter sizes and depths 64. (model.py lines 18-24)

The model includes RELU layers to introduce nonlinearity (code line 78-90)

####2. Attempts to reduce overfitting in the model
The model contains dropout layers and BatchNormalization in order to reduce overfitting (model.py lines 79-80).

The model was trained and validated on different data sets to ensure that the model was not overfitting (model.py code line 19). The model was tested by running it through the simulator and ensuring that the vehicle could stay on the track.

####3. Model parameter tuning

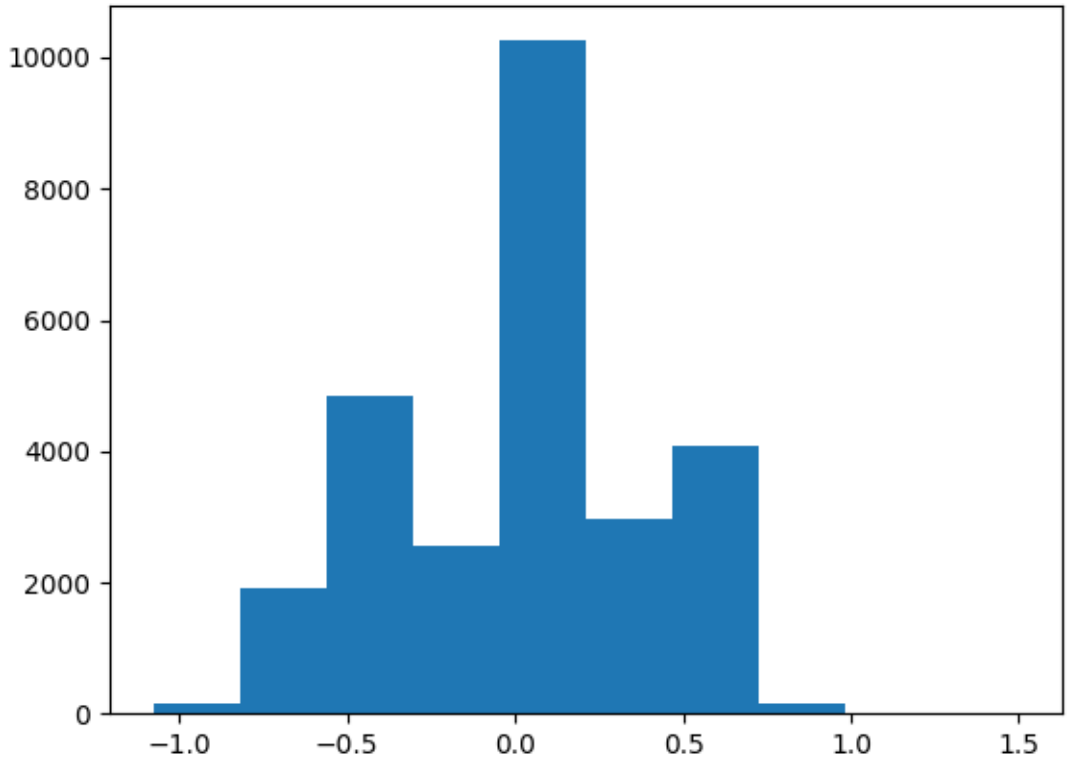
Kernel initializer for weight: truncated normal (model.py code line 77-84)
Optimizer: Adam (model.py code line 88)
Epoch: 150 (model.py code line 93) but Actual epoch should be 12 as shown below:

```
430s - loss: 0.3514 - val_loss: 0.1034
Epoch 2/150
560s - loss: 0.0830 - val_loss: 0.0819
Epoch 3/150
335s - loss: 0.0721 - val_loss: 0.0757
Epoch 4/150
206s - loss: 0.0669 - val_loss: 0.0695
Epoch 5/150
207s - loss: 0.0626 - val_loss: 0.0663
Epoch 6/150
208s - loss: 0.0583 - val_loss: 0.0625
Epoch 7/150
208s - loss: 0.0542 - val_loss: 0.0587
Epoch 8/150
208s - loss: 0.0500 - val_loss: 0.0523
Epoch 9/150
208s - loss: 0.0465 - val_loss: 0.0503
Epoch 10/150
207s - loss: 0.0435 - val_loss: 0.0424
Epoch 11/150
206s - loss: 0.0416 - val_loss: 0.0379
Epoch 12/150
204s - loss: 0.0396 - val_loss: 0.0355
Epoch 13/150
203s - loss: 0.0380 - val_loss: 0.0366
Model is saved
```

I used Keras early stopping callback to quit the training when validation loss is greater than the previous one.

####4. Appropriate training data

Training data was chosen to keep the vehicle driving on the road. I collected 3 laps of driving, one in the center of the road, two in the center but reverse, three recover from the side of the road. Here is the graph of steering angles:



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

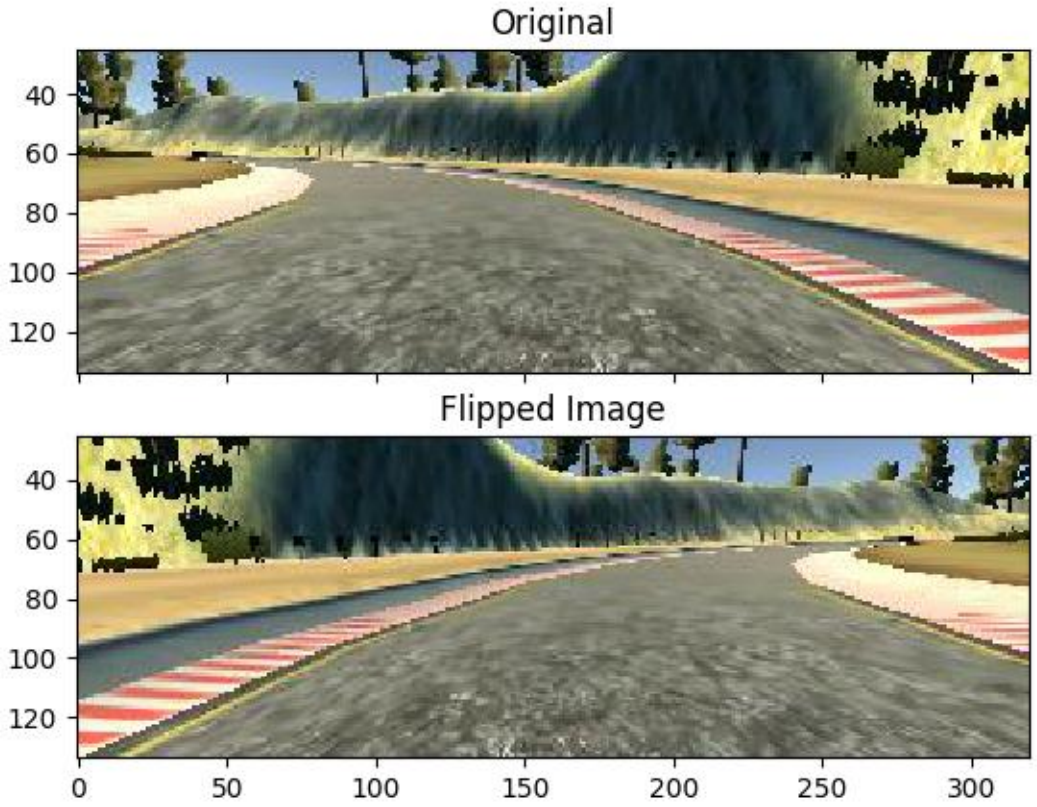
###Model Architecture and Training Strategy

####1. Solution Design Approach

My first step was to use a convolution neural network model similar to the NVIDIA architecture. I thought this model might be appropriate because it has been tested by NVIDIA and with the paper supporting the concept.

In order to gauge how well the model was working, I split my image and steering angle data into a training and validation set.

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 augmented the data by using multiple cameras, left and right with the correction angle of 0.5. (model.py code line 39-42). I also augmented the dataset by flipping the images using opencv (model.py code line 50).



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 78-85) consisted of a convolution neural network with the following layers and layer sizes:

layer 1: Conv2D: Filter size:5x5, Filter depth:24, strides:2x2 and activation: RELU
layer 2: Dropout
layer 3: Conv2D: Filter size:5x5, Filter depth:36, strides:1x1 and activation: RELU
layer 4: Dropout
layer 5: Conv2D: Filter size:5x5, Filter depth:48, strides:1x1 and activation: RELU
layer 6: Dropout
layer 7: Conv2D: Filter size:3x3, Filter depth:64, strides:1x1 and activation: RELU
layer 8: Dropout
layer 9: Conv2D: Filter size:3x3, Filter depth:64, strides:1x1 and activation: RELU
layer 10: Dropout
layer 11: Fully connected: input: 3x3x64, output:100
layer 12: Fully connected: input: 100, output:50
layer 13: Fully connected: input: 50, output:1