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* file 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 a video recording of autonomous driving around the track for few laps

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
```

The results were very positive, with reliable driving for multiple laps as shown in video.

3. Submission code is usable and readable

The model.py contains the code for training and saving the convolution neural network. The file shows the NVidea pipeline I used for training and validating the model, with some 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 5 Convolutional layers followed by 3 Fully Connected layers.

- Preprocess: the data is normalized in the model using a Keras lambda layer, followed by cropping to remove unneeded image sections.

- The Convo layers use 5x5 and 3x3 filter sizes, with depths 24, 36, 48, 64 and 64. Convo layers include RELU to introduce nonlinearity.
- Then there are 3 Fully Connected layers, with gradual reduction to 100, 50, 10, followed by a linear activation output layer (instead of RELU) for detecting steering angle.

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

The model used lots of training data to reduce overfitting.

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, combined with use of all 3 camera angles for training.

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 NVidea's. I thought this model might be appropriate because it worked for many others and was simple enough to implement.

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, and also low mean squared error on the validation set. This implied that the model was not overfitting.

The final step was to run the simulator to see how well the car was driving around track one. Initially, there were a few spots where the vehicle steered off the track, without corrective steering. So recreated more data driving zigzag along road ends, to generate data to steer back to central lane. Such driving was repeated for multiple laps to get a good mean steering angle.

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 consisted of a convolution neural network with the following layers and layer sizes

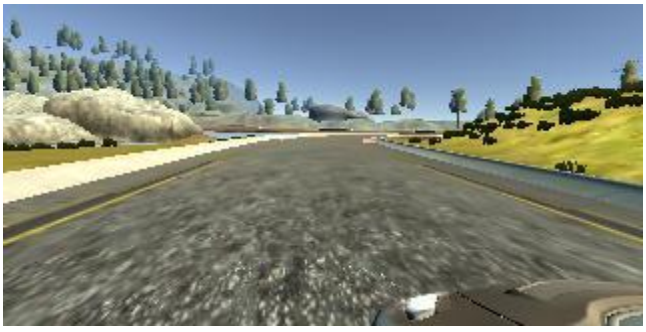
- Preprocess: the data is normalized in the model using a Keras lambda layer, followed by cropping to remove unneeded image sections.
- The Convo layers use 5x5 and 3x3 filter sizes, with depths 24, 36, 48, 64 and 64. Convo layers include RELU to introduce nonlinearity.
- Then there are 3 Fully Connected layers, with gradual reduction to 100, 50, 10, followed by a linear activation output layer (instead of RELU) for detecting steering angle.

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

- To capture good driving behavior, and lots of data, started with provided input data. Then using 10Hz simulator on a windows PC with only CPU with keyboard input,
- I recorded few laps on track one.
- Then redid more driving to generate more data with smooth driving in both directions, using 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 steer back.
- To augment the data sat, and create more data in steering back from sides, took data from all 3 camera angles. The images from left and right camera angles were applied steering angle correction. This helped both increase number of test cases, and teaches cars that are off center to steer back to the middle.
- While calling the model using Keras, randomly shuffled the data set with 20% allotted to 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 4 as evidenced by reducing loss. I used an adam optimizer so that manually training the learning rate wasn't necessary.

  Sample Images:

  Left image



  Center image

Right Image

####4. Implementation Summary

- Managed to get results for first track with default 10hz simulator and keyboard input available on std Windows PC and CPU, with no smoothing and filter algorithms.
- Just used adam optimizer with default learning rate for 4 EPOCHs and no successive refinement.
- The key for generating enough data, covering non center of lane driving, so model learns how to steer back.
- Image argumentation using 3 cameras with .3 steering angle adjustments, but not flipping image.
- Used AWS GPU for training. Did not resize input image, retaining original size. When started getting Memory error loading images into memory on AWS, instead of using Python generators, culled some of my training data :)

And i have a model which can make multiple laps, reliably.