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

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
- data_process.py for generation of processed training and validation data.
- writeup_report.pdf summarizing the results
- video.mp4 capture the video of two laps of autonomous driving.

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
```

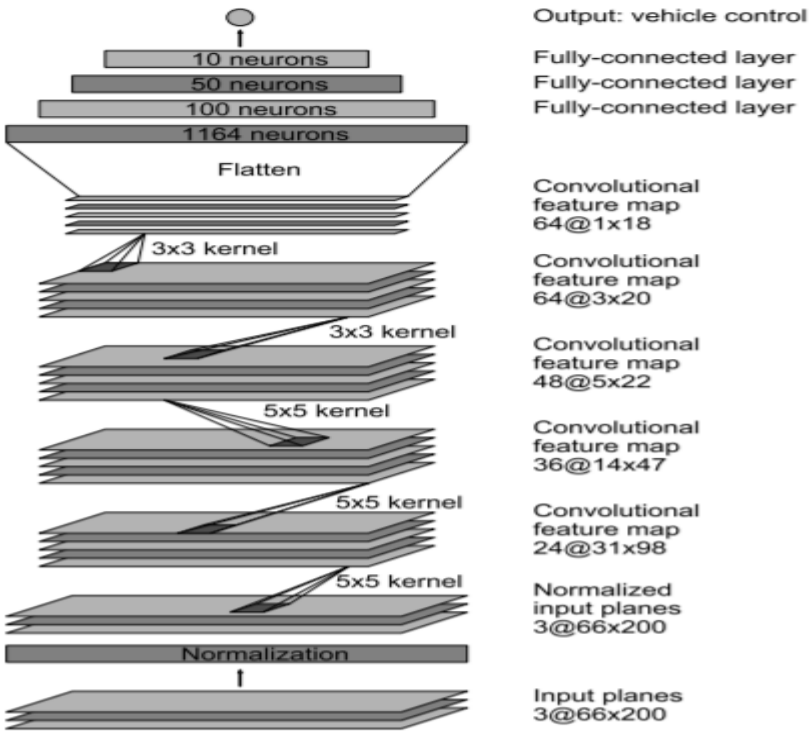
3. Submission code is usable and readable

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

The model is based on NVIDIA's "End to End Learning for Self-Driving Cars" paper. The Network structure is as follows.



It consists of a convolution neural networks with 5x5 and 3x3 filter sizes and depths between 24 and 64.The model includes RELU layers to introduce nonlinearity, and the data is normalized in the model using a Keras lambda layer. Intermittent dropout layers are also used in between the convolution layers and all the fully connected layers.

2. Attempts to reduce over-fitting in the model

The model was trained and validated on different data sets to ensure that the model was not over-fitting (code line 70-76). The model was tested by running it through the simulator and ensuring that the vehicle could stay on the track.

The model contains dropout layers in order to reduce over-fitting (model.py lines 38 56 61 and 66).

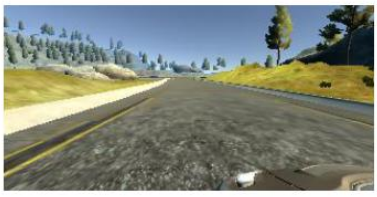
3. Model parameter tuning

The model used an adam optimizer, so the learning rate was not tuned manually (model.py line 67).

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.

At a given time step three images taken from left, center, and right cameras.

Left	Center	Right
		

All these images are used for training by applying steering coefficient to the left and right camera images.

Various data augmentation techniques have been applied to induce the desired behavior. For details about how I created the training data, see the next section.

Model Architecture and Training Strategy

1. Solution Design Approach

The Model used convolution neural network model similar to the NVIDIA's End to End Learning for Self-Driving Cars paper which has been proven to work for this problem.

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

To combat the over fitting, the model was modified by adding dropout layers. Max-Pooling layers are used which reduces the training time.

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 max-pooling and drop-out layers.

Layer #	Description	Location
1.	Normalization layer	model.py lines 20
2.	5x5 Convolution layer with RELU activation	model.py lines 23-24
3.	Max Pooling Layer	model.py lines 26
4.	5x5 Convolution layer with RELU activation	model.py lines 28-29
5.	Max Pooling Layer	model.py lines 31
6.	5x5 Convolution layer with RELU activation	model.py lines 33-34
7.	Max Pooling Layer	model.py lines 36
8.	Dropout Layer with probability : 0.5	model.py lines 38
8.	3x3 Convolution layer with RELU activation	model.py lines 40-41

9.	Max Pooling Layer	model.py lines 43
10.	3x3 Convolution layer with RELU activation	model.py lines 45-46
11.	Max Pooling Layer	model.py lines 48
12.	Flattened Layer	model.py lines 50
13.	Fully Connected Layer (1164) with RELU activation	model.py lines 53-54
14.	Dropout Layer with probability : 0.5	model.py lines 56
15.	Fully Connected Layer (100) with RELU activation	model.py lines 58-59
16.	Dropout Layer with probability : 0.5	model.py lines 61
17.	Fully Connected Layer (50) with RELU activation	model.py lines 63-64
18.	Dropout Layer with probability : 0.5	model.py lines 66
19.	Fully Connected Layer (10) with RELU activation	model.py lines 68-69
20.	Fully Connected Layer (1) with RELU activation	model.py lines 71

3. Creation of the Training Set & Training Process

Taking into consideration the images from all the three cameras, the dataset consists of 24108 images (8036 images per camera angle). Most of the recorded steering angles are zeroes because the training track contains a lot of shallow turns and straight road segments. Therefore, preprocessing images and respective steering angles are necessary in order to generalize the training model for unseen tracks such as our validation track.

Data Augmentation

a) Random Flipped

To augment the data set, I also flipped images and angles thinking that this would reduce the left/right skew due to driving the car around the track in one direction only and can be eliminated by flipping each recorded image and its

corresponding steering angle. For example, here is an image that has then been flipped:

b) Random Shear

Image is sheared horizontally and the steering angle was changed proportionally to the shearing angle. This had the effect of making curvy road pieces appear just as often in the training set as straight parts.

c) Resize

The images are resized to 64x64 in order to reduce training time. A sample resized image is shown in the following figure. Resized images are fed into the neural network.

d) Cropping

The images which are captured during training contains a lot of data which does not add any information to the learning. E.g. the sky, trees and the scenery and thus are cropped from the image. The following figure shows the result of cropping operation applied to an image.

Thus image distortions, random brightness corrections and crops, together generate a practically infinite number of training images from the little training data that was gathered.

Training

Training dataset was very large and it could not fit into the main memory. Hence, we used `fit_generator` API of the Keras library for training our model.
We created two generators namely:

- `train_gen = data_process.generate_next_batch()`
- `valid_gen = data_process.generate_next_batch()`

Batch size of both `train_gen` and `valida_gen` was 64. I used 20000 images per training epoch. It is to be noted that these images are generated on the fly also 6400 images were used for Validation which were also generated using the generator.

Result and Conclusion

After various Iterations of training and testing for various speeds, the final results which met all the points of the rubric was achieved and demonstrated by the video captured and submitted with the project.

The current model works for a lower speed range of 10-15 mph. For more speed the model requires more training data and tuning of the hyper-parameters.