#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

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.pdf summarizing the results
- video.mp4 showing a recording of the vehicle driving autonomously around track 1.

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

My model consists of a convolution neural network with 5x5 and 3x3 filter sizes and depths between 24 and 64 (model.py lines 62-65)

The model includes RELU layers to introduce nonlinearity (code line 62-65), and the data is normalized in the model using a Keras lambda layer (code line 60).

The model finally includes 4 fully connected layers.

The validation loss appeared to be close to the training loss, indicating there is no overfitting. As such, no dropout layer or maxpooling layer were added.

The model used an adam optimizer, so the learning rate was not tuned manually (model.py line 72).

Training data

Training data was chosen to keep the vehicle driving on the road. I have tried multiple sets : one created by myself (center lane driving, recovery, driving smoothly around the curves), one downloaded from the forums (using a joystick instead of the keyboard to train the car) and finally the set provided by Udacity.

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

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 used in real life for the same purpose.

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 model had a low mean squared error on the training set and validation set. However, improving the model accuracy did not necessarily mean better performance in autonomous driving.

I tried different variations of the model such as adding more layers, trying different activation functions, adding regularization layers but these made very little difference in the accuracy of the model. Hence, I decided to stick with the origin model, similar to the nvidia one.

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, such as the bridge and the 2 final curves after the bridge. To improve the driving behaviour in these cases, I recorded additional data, focusing on the areas where the car was falling off the track.

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

Final Model Architecture

The final model architecture (model.py lines 59-70) consisted of a convolution neural network with the following layers and layer sizes :

Layer	Description
Input	160x320x3 image
Normalization	Lambda layer to normalize image
Cropping	Outputs 65x320x3
Convolution 5x5	2x2 stride, outputs 31x158x24
RELU	
Convolution 5x5	2x2 stride, outputs 14x77x36
RELU	
Convolution 3x3	2x2 stride, outputs 6x38x48
RELU	
Convolution 3x3	2x2 stride, outputs 2x18x64
RELU	
Flatten	Outputs 2304
Fully connected	Outputs 100
Fully connected	Outputs 50
Fully connected	Outputs 10
Fully connected	Outputs 1

Creation of the Training Set & Training Process

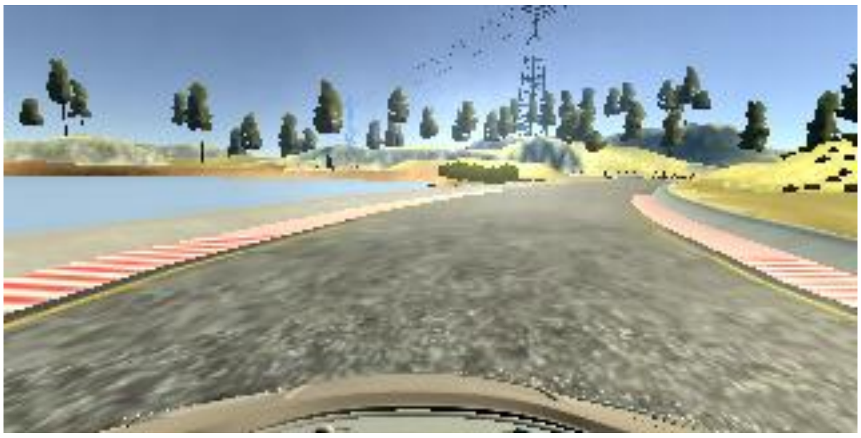
I recorded my own training data and compared the autonomous driving performance using it versus using the data set provided by Udacity.

With the one provided by Udacity, the car was falling off the track less frequently, so I decided to use it.

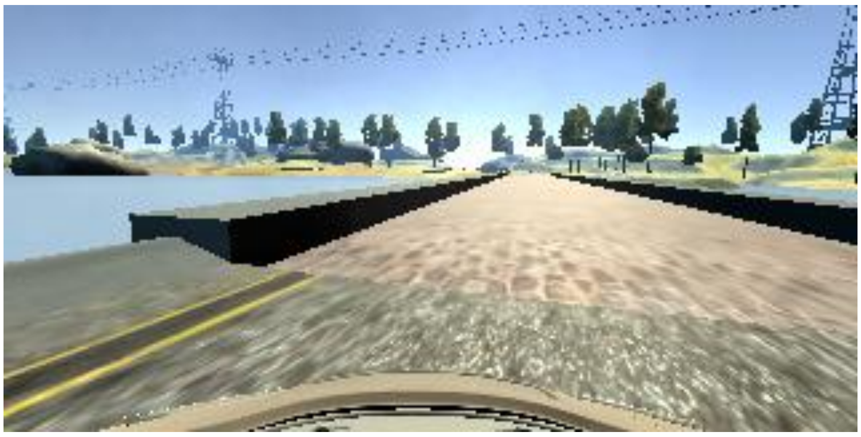
With this data set, the car was rolling on the ledges at the last curve of the track. To correct this, I captured good driving behaviour (centered) at this particular point of the track and appended it to the Udacity data set.

This produced a good result in a sense that the car was not rolling on the outer ledge anymore. It was still quite close to the ledge though. Hence I decided to append the same data once again to the CSV file in order to double its weight in the training. This step fixed the issue and the car was able to drive the curve in the center of the road.

Below is one of the images I added and shows good driving behaviour at the point where the car was having issues (last curve of the track) :



Another part of the track where the car was having issues is at the entrance of the bridge where the car was too close to the left border. The same method as described above was used to correct it.



To augment the data set, I also flipped images and angles in order to balance right and left turns. However, this did not make a difference in the quality of driving, so I removed the step. What made a difference is using the images from the side cameras (code lines 35-36) and associating to them the steering angle of the center image plus or minus a correction factor. Multiple trials showed a value of 0.1 for this factor produced the best results.

After the collection process, I had 9340 number of data points. The data augmentation consisting of using left and right cameras multiplied this number by 3. Form this total number of 28020 data points, I removed 40% of the data points with small steering angles (less than 0.8). This allowed to obtain a more stable driving. Eventually the number of points really used for training was around 15000.

I then preprocessed this data by changing the color space to YUV. The rest of the preprocessing was done in the keras model itself and consisted of cropping the image to get rid of the top part (tree, sky, mountain…) and bottom part (car hood) and normalizing it.

I finally randomly shuffled the data set and put 10% 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 3 as evidenced by the evolution of the training and validation loss. Additional epochs did not significantly improve the loss and even produced worse driving behaviour.

I used an adam optimizer so that manually training the learning rate wasn't necessary.

Below is the output of the training script, showing the training and validation losses for each epoch :

```
14958/14958 [============================] - 310s - loss: 0.0179 - val_loss: 0.0134
Epoch 2/3
14958/14958 [============================] - 302s - loss: 0.0153 - val_loss: 0.0121
Epoch 3/3
14958/14958 [============================] - 299s - loss: 0.0144 - val_loss: 0.0119
```