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
- writeup_report.md or writeup_report.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 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

My model consists of a convolution neural network with 5x5 filter sizes and 6 and 16 depths. (model.py lines 106 and 111)

The model includes RELU layers to introduce nonlinearity (model.py lines 106 and 111), and the data is normalized in the model using a Keras lambda layer (model.py lines 101).

2. Attempts to reduce overfitting in the model

The model was trained and validated on different data sets to ensure that the model was not overfitting (model.py lines 36). The model was tested by running it through the simulator and ensuring that the vehicle could stay on the track. (It's NOT yet trace the track all the way.)

3. Model parameter tuning

The model used an adam optimizer, so the learning rate was not tuned manually (model.py lines 126).

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.

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

Model Architecture and Training Strategy

1. Solution Design Approach

The overall strategy for deriving a model architecture was to collect enough and appropriate driving data and have the model learned by their data.

My first step was to use a convolution neural network model similar to the LeNet architecture I used at the project 2. I thought this model might be appropriate because it can learn right driving line to remember the road character and the appropriate steering angle.

The next 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 made the mirror images of my driving data and inserted them into my data set.

At the end of the process, the vehicle is able to drive autonomously around the track without leaving the road. (Sorry, I lie to you. It's NOT yet trace the track all the way.)

2. Final Model Architecture

The final model architecture (model.py lines between 99 and 121) consisted of a convolution neural network with the following layers and layer sizes .

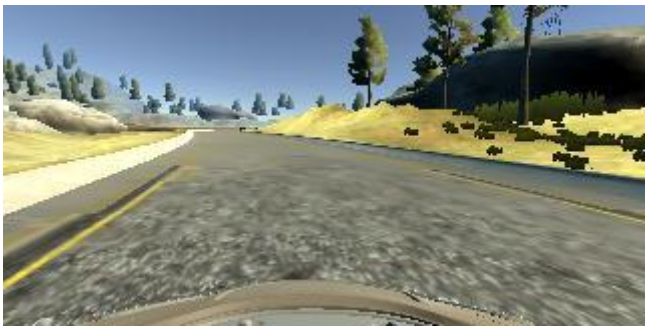
Layer	Description
Input	65x320x3 image
Convolution 5x5	1x1 stride, valid padding, outputs 61x316x6
RELU	
Max pooling	2x2 stride, outputs 31x80x6
Convolution 5x5	1x1 stride, valid padding, outputs 25x76x16

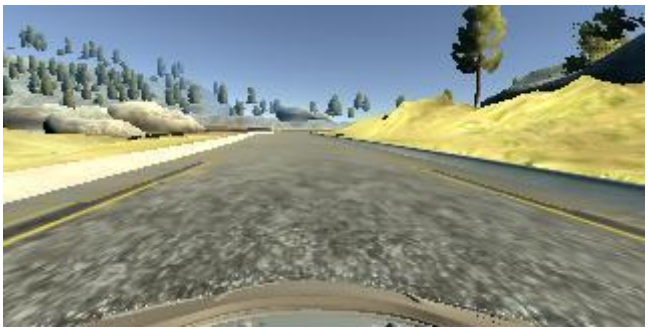
Layer	Description
RELU	
Max pooling	2x2stride, outputs 13x38x16
Flatten	Output 7940
Fully connected	Output 120
Fully connected	Output 84
Fully connected	Output 1

3. Creation of the Training Set & Training Process

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

I then recorded the vehicle recovering from the left side and right side of the road back to center so that the vehicle would learn to go back from the both sides to center. These images show what a recovery looks like starting from the left side of the road. :





To augment the data set, I also flipped images and angles thinking that this would make model more robust for the reverse move of the recorded data. For example, here is an image that has then been flipped:

After the collection process, I had 11586 number of data points. (They include the flipped images.) I then preprocessed this data by normalization and cropping.

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. The ideal number of epochs was 3 as evidenced by the low and stable loss value of the epoch 2 and 3. (the values

are between 0.03 and 0.04. ; model.ipynb output cell 12 line 12 and 14) I used an adam optimizer so that manually training the learning rate wasn't necessary.