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 `sh 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 convolutional neural network model consists of three 5x5 filter sizes and two 3x3 filter sizes and depths between 24 and 64 (model.py lines 115-123).

The model includes RELU layers to introduce nonlinearity (code line 115,117,119,121,123), and the data is normalized in the model using a Keras lambda layer (code line 113).

2. Attempts to reduce overfitting in the model

The model contains dropout layers in order to reduce overfitting (model.py lines 116,118,120,122,126).

The model was trained and validated on different data sets to ensure that the model was not overfitting (code line 112-128). 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 (model.py line 131).

4. Appropriate training data

I used left,right,center camera image data, and the data augmentation as my training data.

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

Model Architecture and Training Strategy

1. Solution Design Approach

I'm going to test out an architecture published by the autonomous vehicle team at NVIDIA.This is the network they use for training a real car to drive autonomously.

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 but a high mean squared error on the validation set. This implied that the model was overfitting.

I found that the validation loss is increasing with every epoch.I'm going to try training it again for a fewer number of epochs.then I have added the dropout between the convolution layer to reduce the overfitting.

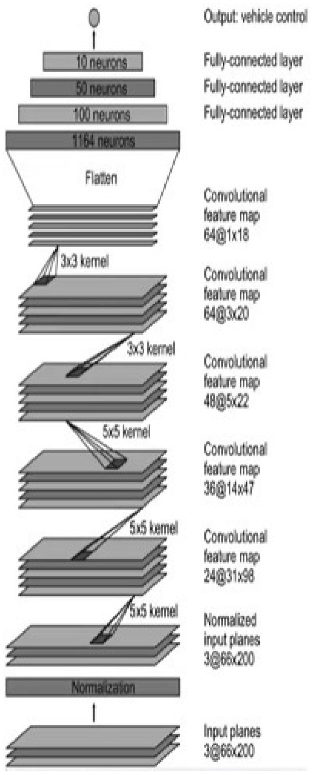
The final step was to run the simulator to see how well the car was driving around track one. There were a few spots where the vehicle fell off the track,to improve the driving behavior in these cases, I Continue in the simulator to continue on the track for 2 laps, plus 1 turn in the opposite direction.After three cameras and augmentation data, I get more data, and finally the training model gives a good result.

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

2. Final Model Architecture

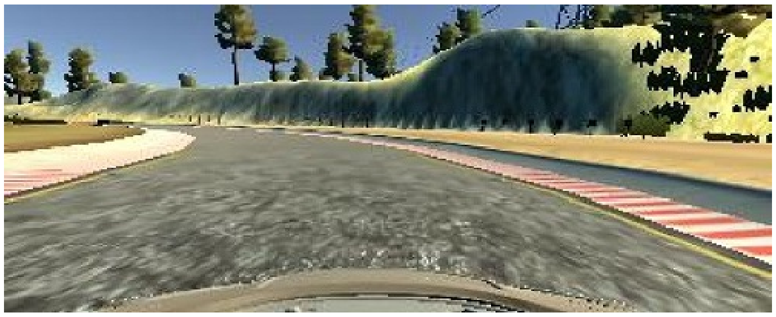
The final model architecture (model.py lines 112-128) consisted of a normalization layer and followed by five convolutional layers,followed by four fully connected layers.

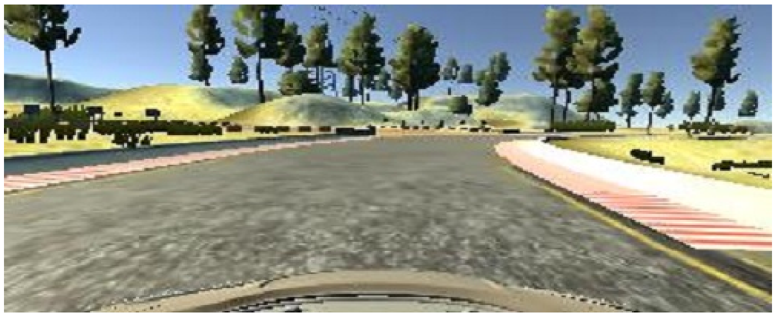
Here is a visualization of the architecture:



3. Creation of the Training Set & Training Process

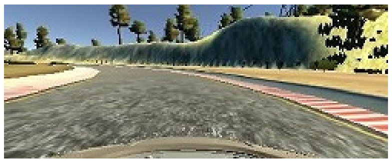
To capture good driving behavior, I first recorded two laps on track and drive around the track in the direction one using center lane driving. Here is an example image of center lane driving:

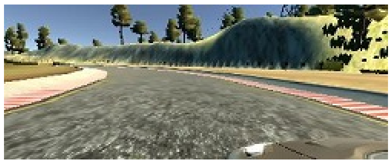


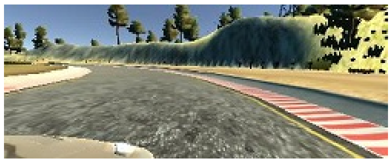


I then recorded the vehicle recovering from the left side and right sides of the road back to center so that the vehicle would learn to steer back to the center, if the vehicle starts drifting off to the side.

These images show what a recovery looks like starting from center, left, right :

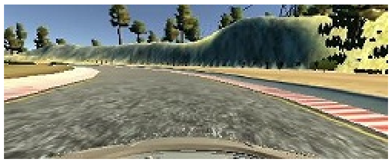


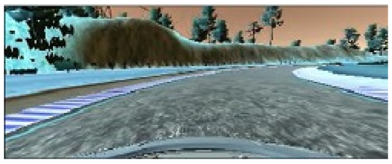




Then I repeated this process on track two in order to get more data points.

To augment the data sat, I also flipped images and angles thinking that this would Add more new data to fit the model better and solve the problem that the car seems to pull too hard to the left.(This actually makes sense,the training track is a loop, and the car drives counter-clockwise.so the most the time,the model learning to steer to the left.) For example, here is an image that has then been flipped:





After the collection process, I have three camera data and their total level reversal data. I then preprocessed this data by normalizing the data and mean centering the data.

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 too much epochs will overfitting I used an adam optimizer so that manually training the learning rate wasn't necessary.