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
- common.py for camera and image pre-processing for feeding into the model
- edge.py for all the functions used for finding edges and surfaces
- model.h5 containing a trained convolution neural network
- 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.

The common.py file contains the code for image/camera pre-processing that returns the featured image that contains both raw camera data and engineered features to aid in keeping the vehicle on the track.

The edge.py file contains the code functions for the pre-processing of the data. The preprocessed data finds edges and luminance surfaces to aid in the training of the network.

Model Architecture and Training Strategy

1. An appropriate model architecture has been employed

My model consists of the model architecture, as outlined in the paper "End to End Learning for Self-Driving Cars" by Nvidia, Publication "arXiv:1604.07316" with an added convolutional layer at the end.

The model consist of three convolution neural network with 5x5 filter sizes and depths of 24, 36 and 48, followed by three convolution neural network with 3x3 filter sizes and all the same depths of 64. This then followed by two final connect layers with 100 and 50 nodes respectively, down to outputting a single a steer angle (model.py lines 129-218)

The model uses ELU activation to introduce nonlinearity, and the data is normalized between each layer.

The Model has been implemented in Keras, using TensorFlow as the backend.

2. Attempts to reduce overfitting in the model

The model contains dropout layers in order to reduce overfitting between all layers, excluding the last fully connected layer.

The model was trained and validated on two different track data sets to help ensure that the model was not overfitting on the one track. Data was obtain by driving in both directions of the track, as well as flipping image data and negating steer angles to create inverse data to train on (model.py line 10-48).

The model was tested by running it through the simulator and ensuring that the vehicle could stay on the track. A video of the run on one of the tracks is supplied as "run3.mp4"

3. Model parameter tuning

The model used an adam optimizer, so the learning rate was not tuned manually (model.py line 213).

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 as well as obtaining data from multiple tracks and flipping images to double the data. Multiple runs both forward and backwards on the two supplyed tracks were obtained to increase the data for training.

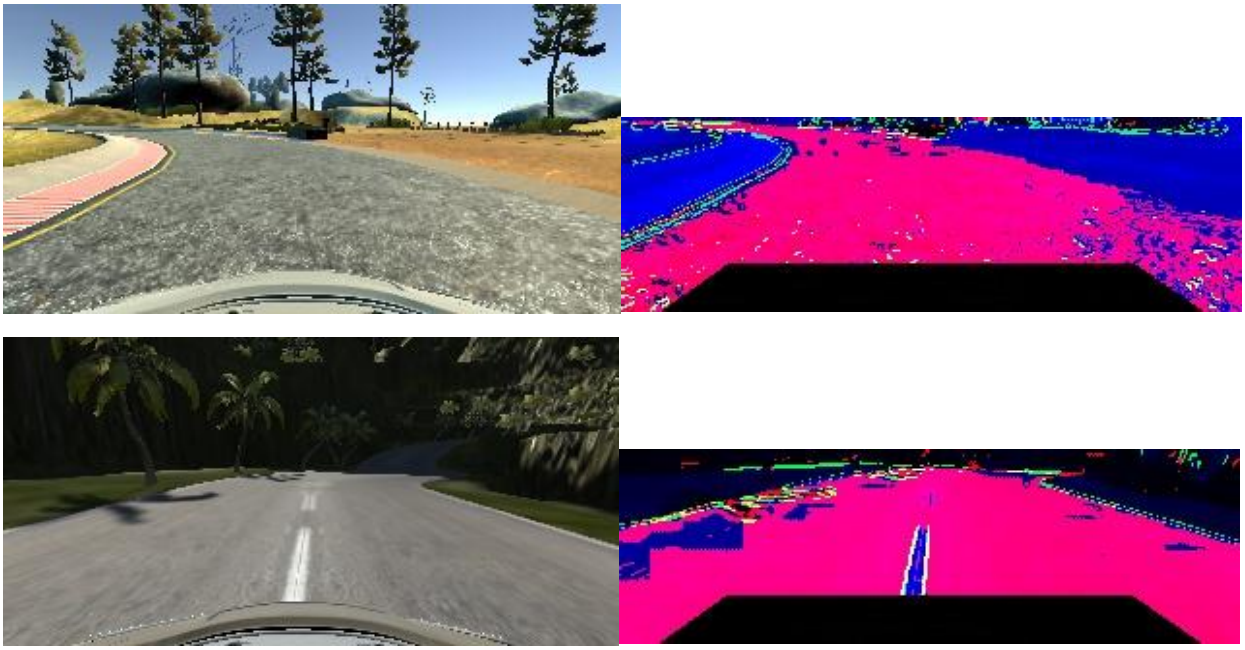
For details about how I created the training data, see the next section.

Model Architecture and Training Strategy

1. Solution Design Approach

The overall strategy for deriving a model architecture was to pre-process each image too both remove unwanted data and add edge and surface detection features for aiding the model in learning the track.

As outline in the common.py file. Sobel edge detection was used on the L channel of the HSL image data as well as a thresholded binary map of the S channel for distinguishing the dark road and other surfaces. Examples of the non and pre-process images are below:



My first step was to use a convolution neural network model similar to the method outlined in the "End to End Learning for Self-Driving Cars" by Nvidia, Publication "arXiv:1604.07316" paper. I thought this model might be appropriate because of the results outlined in the paper and was a good starting point for testing.

In order to gauge how well the model was working, I split my image and steering angle data into a training and validation set. The original model, as outlined in the paper was found to not keep the vehicle accurately driving on the track.

To combat the inaccuracy, I modified the model both by adding an extra convolutional layer to minimize the features size discrepancy as well as performed feature engineering on the data to help the model learn the track. More data was also collected at this point as well.

Four EPOCHs were chosen for training the model as the training low mean squared error began to no longer converge on the training set as well as producing a similar mean squared error on the validation set. This implied that the model was fitting the data with very little overfitting.

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 129-218) as outlined above in section "1. An appropriate model architecture has been employed"

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:

Then I repeated this process on track two in order to get more data points and to create a more versatile model for use on other tracks. Here is an example image of center lane driving from track two:

To augment the data set, I also flipped images and angles to both double the data and provide an opposite steering angle. Here is an example image that has then been flipped:

After the collection process, I had 48036 number of data images.

I finally randomly shuffled the data set and put 15% 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 four. This was chosen as there was no more significant decrease in the training loss and also produced a similar validation loss. I used an adam optimizer so that manually training the learning rate wasn't necessary.