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

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 NVIDIA architecture.

My model consists of a three convolution neural network with 5x5 filter sizes and depths between 24 and 48 , two convolution neural network with 3x3 filter size and depth 64 and four full connected layers with depths between 100 and 1(number of the unique classes)(*model.py* lines 85-103).

The model includes RELU layers to introduce nonlinearity (code lines 85-92), and the data is normalized in the model using a Keras lambda layer (code line 79).

2. Attempts to reduce overfitting in the model

The model contains dropout layers in order to reduce overfitting (*model.py* lines 90-95-102).

The model was trained and validated on different data sets to ensure that the model was not overfitting. 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 106).

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 and smooth turning over hard curves.

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 drive the car in the simulator autonomous mode

My first step was to use a convolution neural network model similar to the NVIDIA network I thought this model might be appropriate .

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.

To combat the overfitting, I modified the model so that i added three dropout layers with dropout = 50%.

The final step was to run the simulator to see how well the car was driving around track one.

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

2. Final Model Architecture

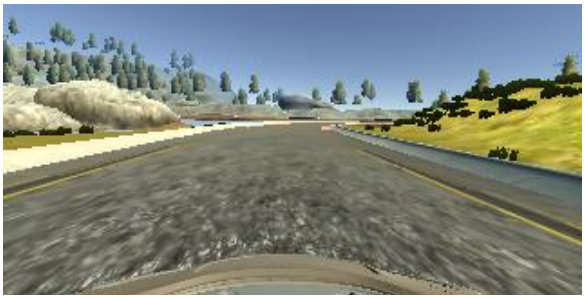
The final model architecture (*model.py* lines 76-107) consisted of a convolution neural network with the following layers.

Convolution layer with filter size 5x5, strides of 2x2 and depth of 24
Convolution layer with filter size 5x5, strides of 2x2 and depth of 36
Convolution layer with filter size 5x5, strides of 2x2 and depth of 48
Dropout 50%
Two Convolution layer with filter size 3x3 and depth of 64
Dropout 50%
Flatten layer
Full connected layer with depth of 100
Full connected layer with depth of 50
Full connected layer with depth of 10
Dropout 50%

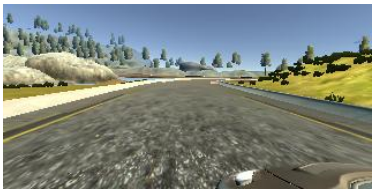
Final layer with depth of 1(number of the unique classes)
Loss function of mean square error
Adam optimizer with default settings

3. Creation of the Training Set & Training Process

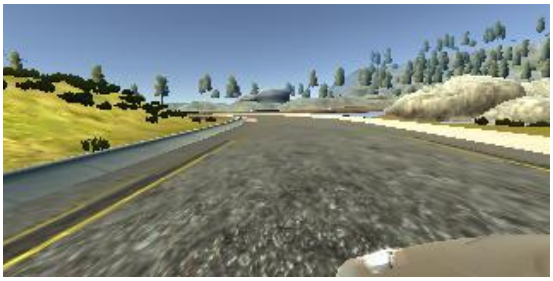
To capture good driving behavior, I first recorded two laps on track one using center lane driving. Here is an example image of center lane driving:



I then recorded the vehicle recovering from the left side and right sides of the road back to center so that the vehicle would learn to return back to the center of the road again examples of the images:



To augment the data sat, I also flipped images and angles thinking that this would increase the data to help the model learn more and the road always turn left so the model wont know what to do if turn right. For example, here is an image that has then been flipped:

After the collection process, I had (19320) number of data points. I then preprocessed this data by Lambda layer .
After i cropped the data images (sky, hills and car hood) so they don't distract the model using Cropping2D method on Keras.

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 model mean squared error loss .

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