Behavioral Cloning

Writeup Template

You can use this file as a template for your writeup if you want to submit it as a markdown file, but feel free to use some other method and submit a pdf if you prefer.

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](https://review.udacity.com/#!/rubrics/432/view) 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 model consists of 5 convolution neural network layers with 5x5 and 3X3 filter sizes and depths between 24,36,48 and 64 (model.py lines 65-72)

The model includes RELU layers to introduce nonlinearity, and the data is normalized in the model using a Keras lambda layer (code line 67).

2. Attempts to reduce overfitting in the model

The model contains dropout layers in order to reduce overfitting (model.py lines 77 and 79).

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 81).

4. Appropriate training data

Training data was chosen to keep the vehicle driving on the road. I used a combination of center lane driving, left images, right images and flipping all of them.

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 check from Udacity classroom lecture on "Even more powerful network" that is "NVIDIA architecture" and modifying on training data and including dropout layers.

My first step was to use a convolution neural network model similar to the NVIDIA model I thought this model might be appropriate because this has 5 convolutional layers and dense layers appropriate to train.

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 one dropout layer.

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 like after crossing the bridge, to improve the driving behavior in these cases, I added another dropout layer and flipped all the images to increase training data.

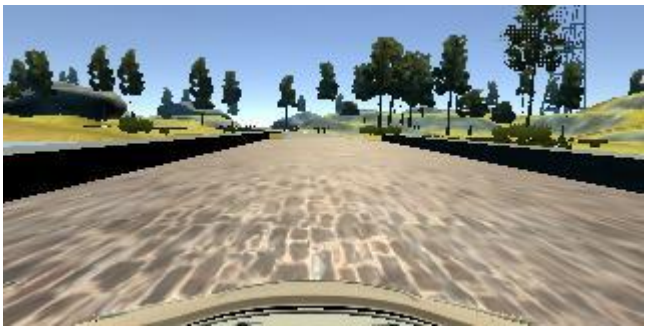
At the end of the process, the vehicle is able to drive autonomously around the track without leaving the road.

2. Final Model Architecture

The final model architecture (model.py lines 65-80) consisted of 5 convolution layers and layer depths 24, 48,64,64 and 3 Dense layers.

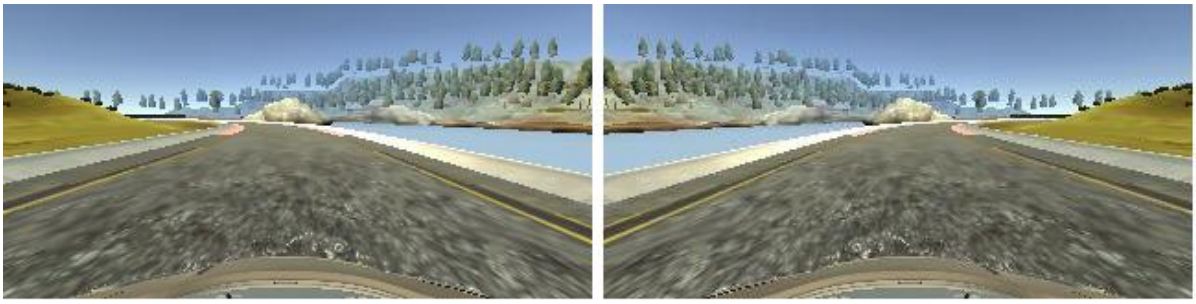
3. Creation of the Training Set & Training Process

To capture good driving behavior,I used Udacity's sample data. Here is an example image of center lane driving:



Here is an example image of left and right lane driving respectively:

To augment the data sat, I also flipped images that this would be helpful to train properly. For example, here is an image that has then been flipped:



After the collection process, I had 32144 number of data points. I then preprocessed this data by normalising using Lambda layers.

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 7 as evidenced by validation loss on every epochs, I used an adam optimizer so that manually training the learning rate wasn't necessary.