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 convolutional 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.json
```

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

Network Architecture

For this project, I was able to create a neural network that can steer a car with constant throttle in a simulated environment. I implemented the Nvidia Paper on End-to-end learning for Self-Driving Cars for my neural architecture which has demonstrated the ability to steer a car in the real world. The model takes in as an input 66x200X3 images. The model consists of 24 filters followed by 36 filters, 48 filters and two 64 filter segments. Each segment has 2x2 max pooling, a dropout layer, and a relu activation function. The final segment flattens the input data then consists of fully several fully connected layers (dense layer of 1164 neurons, a dropout layer, dense layer of 100, dense layer of 50, dense layer of 10, and finally a dense layer of 1 to generate the output). It is important to note that it is not possible to distinguish which parts of the network do feature extraction and which parts of the network providing the steering prediction. (model.py 146-177)

2. Attempts to reduce overfitting in the model

The model contains ‘relu’ dropout layers in order to reduce overfitting (model.py lines 150, 155, 160, 164, 168, 173).

The model was trained and validated on different data sets to ensure that the model was not overfitting (code line 121-127). 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 184).

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

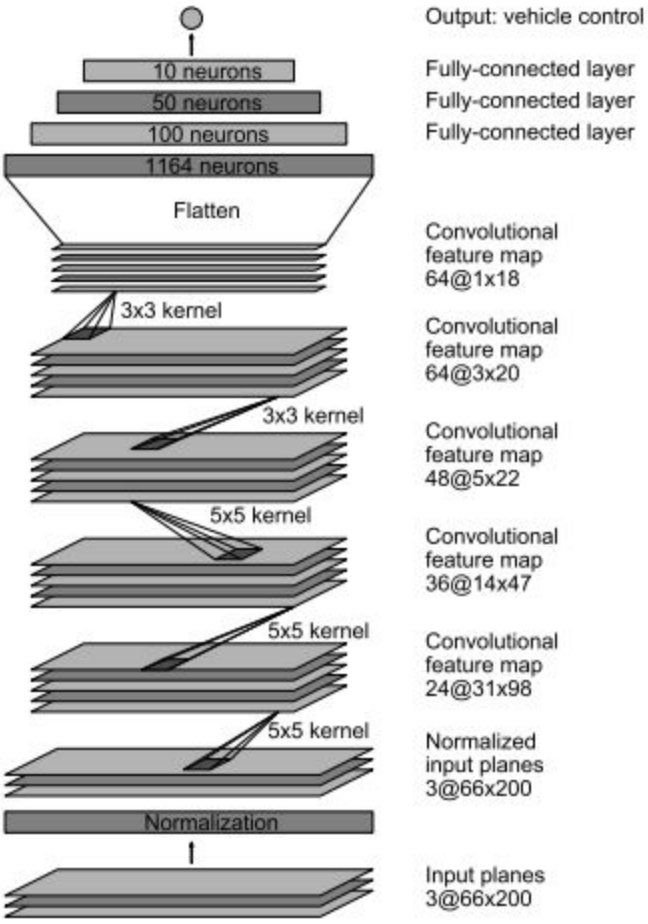
The overall strategy for deriving a model architecture was to pick at random a left, right or center image for each training batch. In addition, each image was flipped at random. These procedures helped provide training examples that could generalize the model.

My first step was to use a convolutional neural network model similar to the NVIDIA model used in the end-to-end learning for self driving cars paper. I thought this model might be appropriate because it was proven effective at predicting steering angles in real world environments.

In order to gauge how well the model was working, I split my image and steering angle data into a training and validation set. Only the center images were chosen for the validation set and no image flipping occurred. It is interesting to note that the validation error was slightly less than the the training error. This could be because of the lack of data augmentation in the validation set as well as the dropout layers used to reduce overfitting in the network architecture.

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

2. Final Model Architecture



Source: NVIDIA paper

3. Creation of the Training Set & Training Process

I used the Udacity training data provided for the project. I trained with 16 epochs in total. After every 2 epochs, I changed the number of training examples with large steering angles incrementally. This was to ensure that the large steering angles were not diluted and well represented in training since the majority of training examples had steering angles of 0.