Behavioral Cloning Project Writeup

Eddie Ferrufino

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

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

The model selected was the NVIDIA model extracted from the paper “End to End Learning for Self-Driving Cars”. I decided to go this route because after reading the paper it seemed like their use case was very similar to the Behavioral Cloning Project objective of training a neural network to drive based on images and steering angle data.

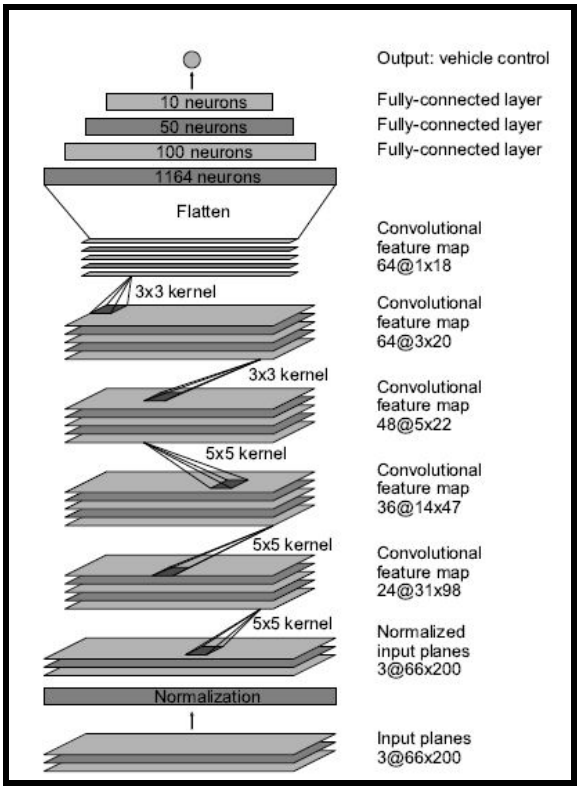

Image 1: NVIDIA Convolutional Neural Network Architecture implemented in model.py using Keras

Prior to being processed by the neural network the image is cropped to remove the hood and sky thus eliminating unimportant features which could impede learning since the color of the sky should not determine which direction to steer. After the image is cropped the image is normalized. All of this is done within Keras in order to leverage GPU processing. Implementing

these layers in Keras is also desirable since it makes the model instantly portable to drive.py since there is no need to reimplement cropping and normalization steps.

Image 2: Training image prior to cropping(320x160x3)

Image 3: Image after cropping(320x65x3)

```
model = Sequential()
model.add(Lambda(lambda x: (x / 255.0) - 0.5, input_shape=(160,320,3)))
model.add(Cropping2D(cropping=((70,25),(0,0))))
model.add(Convolution2D(24,5,5,subsample=(2,2), activation='relu'))
model.add(Convolution2D(36,5,5,subsample=(2,2), activation='relu'))
model.add(Convolution2D(48,5,5,subsample=(2,2), activation='relu'))
model.add(Convolution2D(64,3,3,activation='relu'))
model.add(Convolution2D(64,3,3,activation='relu'))
model.add(Flatten())
model.add(Dense(100))
model.add(Dense(50))
model.add(Dense(10))
model.add(Dense(1))
model.compile(optimizer='adam', loss='mse')
model.fit(X_train, y_train, validation_split=0.2, shuffle=True, nb_epoch=5)
model.save('model.h5')
```

Image 4: Keras implementation of the NVIDIA model described in Image 1

2. Attempts to reduce overfitting in the model

The generated data set is split into a training and validation data set. The training data set contains 80% generated data while the validation data set contains the remaining 20% of the

generated data set. Early stopping was utilized to prevent overfitting. The number of epochs selected to stop training the neural network was 5. I arrived at this number after performing empirical testing using the driving simulator. Training for more iterations had no impact or actually resulted in poor results when compared to a network that was trained for 5 epochs. The details of early stopping as a method for preventing overfitting are detailed in this paper Early Stopping - but when?.

3. Model parameter tuning

The model uses and adam optimizer, the learning rate was not tuned manually.

4. Appropriate training data

To train the model I generated training data with the provided driving simulator. It took me a couple of attempts to generate a dataset that produced a good result. Early datasets produced a model that pulled to the left, especially while driving over the bridge. After looking at the data set distribution I was able to determine that for a training data set with just one lap of center line driving more “left turn” data was present when compared to “right turn” training data. In order to offset this I added a lap driving in the reverse direction. This data also produced mixed results with the model barely making through the bridge and often careening of the road on turn 3. I had an idea that my model was not being shown enough variety of driving data to learn the task of driving in a general context.

To help the model learn to drive I thought if I showed it two different driving styles, it would have a more general understanding of what it means to drive. I recorded 4 laps total - 2 forward and 2 reverse. For each forward lap of driving I adopted two different styles of drivng the car, the first was normal center line driving where I tried as best I could to keep the car near the center of the track. The second style was a racing style of driving where I attempted to take turns more agressively, hit the corner apex’s, and use the entire track, not just the center. This final data set produced very good results. At a speed of 20mph the car is able to drive around the lap 5 times, possibly more if you have the patience to watch the simulator. Below is a link to a recording of my model driving the simulated car around the track.

https://www.youtube.com/watch?v=-VOEZxfdt8c

Model Architecture and Training Strategy
git
1. Solution Design Approach

My first approach for designing a solution to this project was to leverage the NVIDIA model mentioned in the paper that was previously mentioned. I did not deviate from this model since the researchers had demonstrated positive results in real world testing. For this project I

focused on generating a data set with diverse driving style(forward and reverse), I leveraged the left and right cameras to apply a steering correction factor incase the model was about to drive off road. If each camera counts as a “lap” then in total my generated data set effectively contains 12 laps of simulated data, with ⅔ of that data being “correction” data. The generated data set was split into 80% training data and 20% validation data. This allowed me to calculate model loss and gauge how well the model was learning. Loss did not prove to be the most definitive indicator of a well trained model. I found that empirical testing proved to be the most effective way to judge whether or not the model had learned the task of driving.

1. **Implemented NVIDIA model**

2. Driving the car around the simulator did not go very well to start.

3. With the standard data set: The car did not make it past turn one. Steering angle were wild and unpredictable.

4. With two laps of center driving(forward only): The car did not make it past turn one. Steering angle seemed almost random.

5. With two laps of center driving(forward and reverse): The car made it past turn one, would steer left at bridge and come to stop: Steering seemed ok for brief moments but random as well.

6. With two laps of center driving and two laps of “race car” driving(forward and reverse): The car made it past turn one, would steer left at bridge and come to stop: Steering seemed ok for brief moments but random as well.

7. **Added left and right images plus steering correction factor of 0.15**

8. With two laps of center driving and two laps of “race car” driving(forward and reverse): The car could drive several complete laps around the first track.

2. Final Model Architecture

The final model architecture consists the following layers.

Lambda layer(normalization): Output(320x160x3)
Crop Layer(320x65x3)
Convolution Layer(5x5): Output(158x31x24)
Convolution Layer(5x5): Output(77x14x36)
Convolution Layer(5x5): Output(37x5x48)
Convolution Layer(3x3): Output(35x3x64)
Convolution Layer(3x3): Output(33x1x64)

Flatten Layer: Output(2112x1)
Fully Connected Layer: Output(100x1)
Fully Connected Layer: Output(50x1)
Fully Connected Layer: Output(10x1)
Output: Predicted Steering Angle(1x1)

3. Creation of the Training Set & Training Process

The generated training data set was comprised of four total laps. The first two laps consisted of center line driving in both the forward and reverse direction. The 3rd and 4th laps consisted of “racing” style driving in both the forward and reverse direction. I decided to include “racing” style driving since this would show the neural network how it is acceptable to driving at the edges of the curves or straight part of the track while reinforcing the concept of staying within the bounds of the track. I hypothesize that adding data from different human drivers may provide an even richer source of data from which to learn from.

In addition to the four laps of driving, I then used the additional left and right images along with a generated steering angle to augment the center camera data set.

Left

Right

Center