Behavioral Cloning

Anthony Nixon – anixon604@gmail.com

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:
(*Please note: my cohort was assigned the project with drivetrain and model.json)

- •drivetrain.py containing the script to create and train the model

•drive.py for driving the car in autonomous mode

•model.json for containing the neural network

•model.h5 containing the trained weights for the 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.json
```

3. Submission code is usable and readable

The drivetrain.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 arcthiecture has been employed

My model is based on the NVIDIA architecture with some slight modifications. I have 5 convolutional layers with the following (depth, kernels):

(24, 4x4)*, (36, 4x4)*, (48, 4x4)*, (64, 2x2), (64, 2x2)

*first 3 layers also have a stride of (2x2)

These layers are followed by 4 flat layers of:

(100), (50), (10), (1)

Compared to the NVIDIA architecture, my kernels were changed from 5x5 → 4x4 and 3x3 → 2x2 for the first 3 convs and last 2 convs respectively. I also removed the initial normalization layer indicated in the NVIDIA paper. I did do a manual normalization from [0,1] on all inputs during preprocessing though. I then ADDED a single dropout layer with 0.3 probability after the first convolution.

All my layers use a RELU activation except for the final flat layer which I have used a TANH activation because it has an range [-1,1] which is conveniently matched to the normalized angle output range.

2. Attempts to reduce overfitting in the model

I experimented with dropout layers at various positions with my best strategy putting it after the first flat layer. Prior to placing the dropout, with my final parameters, I was getting a frozen response ~0.0008 steering angle repeating through all frames. I decided to address overfitting and after putting the dropout in, the car was responsive and smooth again.

I split the data into separate training and validation sets.

I used a generator function which randomized the data to be yielded as well as making random angle adjustments and random image flips to make sure the model was getting unique yet consistent input.

I also attached two CallBacks to the fit_generator function:
ModelCheckpoint – set to monitor val_loss and save the best model from all epoch
EarlyStopping – set to end training when val_loss no longer improves

The model was tested by running it on the simulator and confirming that it was staying on track and driving smoothly.

3. Model parameter tuning

The model used an adam optimizer, so the learning rate was adaptive but I initialized it to a smaller than default rate. Instead of 0.001, I used 0.0001.

I chose a batch size of 64 which was my starting point and generated good results and sufficient speed on the EC2 instance I was using.

I set number of Epochs to 50 as a limit. However, because I implemented EarlyStopping and ModelCheckpoint callbacks my actual number of Epochs is adaptive (usually completing 4-6 epochs on an average train).

I trained with samples_per_epoch at 50K and validated with nb_val_samples at 20% of samples_per_epoch.

4. Appropriate training data

Training data was a combination of one set of Udacity provided track1 data and three sets of generated data from manual track1 navigation.

Additional training data was also created by using the left and right camera viewpoints with an augmented angle towards center. This mimics a center camera with steering suited to the offset position.

The basic data described above was fed into a generator which created unique novel data on-the-fly by perturbing angles very slightly while maintaining the integrity of the steering position and by flipping images and negating their angle to create even more data.

I also normalized each image during generation to b = 1, a = 0. Using a manual X/255.

I also loaded more data 66% of Left/Right camera vs 33% of Center camera. I did this after realizing that the strong turns on the track were suffering. By feeding in more data containing centering correction I was able to improve immensly.

Model Architecture and Training Strategy

1. Solution Design Approach

The overall strategy I used was to start with something straight forward and get some small results then try to tune from there.

I went straight to the NVIDIA architecture first because after reading through the academic paper, I found it somewhat simple to understand. I also observed from slack messenger discussions that many people were having success with it.

I thought this would be appropriate because in their use case they were also going straight to actuation with their final layer. Additionally, the NVIDIA model was designed to handle real world inputs on real roads whereas I was dealing with lower resolution data on a closed track so I felt confident that it had enough depth and complexity to succeed in my case. Then, perhaps I could also have more flexibility with tuning.

After getting the basic model design implemented and trying just a few input images to make sure everything was working in terms of preprocessing and expected model outputs and driver function, I then focused on training data.

From researching I discovered a key component to having success on the track was to have a lot of data and enough variety of data to counter drifting off center or recovering from an unusual lane position. The reason this was difficult is because a regular driver data recording will have few instances of off center or heavy corrections. Some people created manual training data where they would swerve back to center from the extremes of the track. My technique was to utilize the left and right camera images with modified angles (+0.17 for left camera and -0.17 for right camera) to simulate a center camera offline which was driving back to center. I also fed in MORE of this offcenter data 66% vs 33% from center camera.

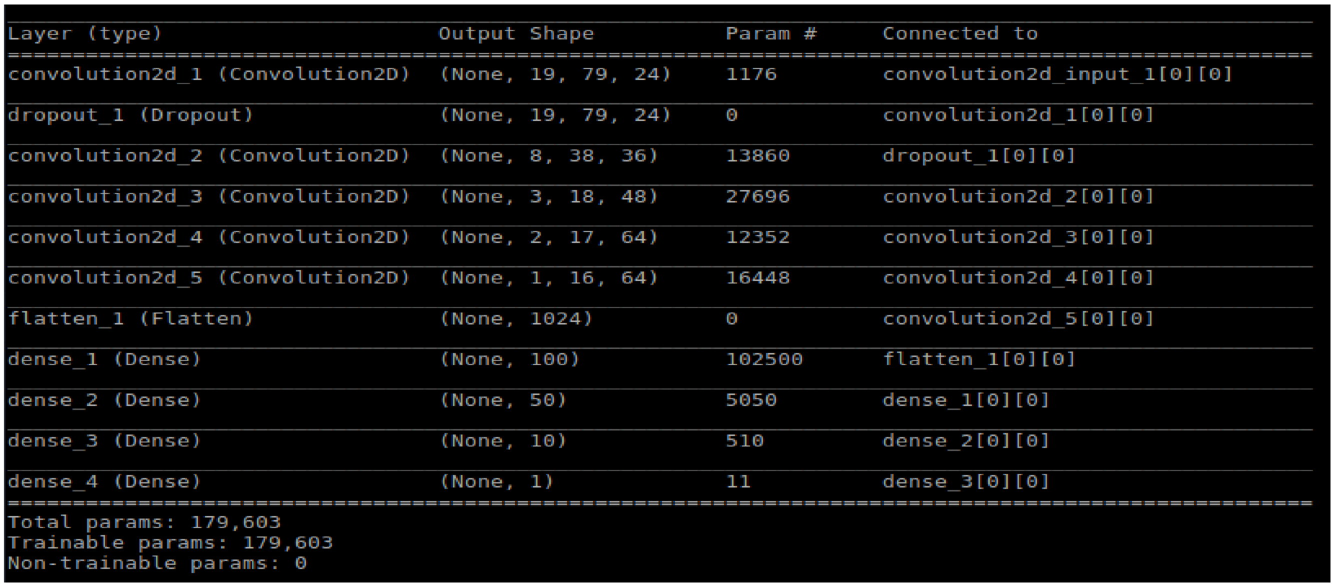
In order to check how well my model was working I had training and validation sets. My training set consisted of all three cameras and for the validation set I experimented with both, validation set with all three cameras, and validation set with only center camera images. I eventually settled on the latter as it gave slightly better results for me and with the rationale that the “real” testing and performance environment would only have center camera, and although the augmented angles of the other cameras simulated a center camera, it wouldn’t be quite as accurate. In all my testing the MSE of the training set was higher than the MSE on the validation set so I did not have overfitting issues.

Finally, I had the simulator drive autonomously around the track. I made sure that it did not veer off the track and tried to look for stability on straights as well. When the vehicle was able to complete 5+ rounds of the track without issue, on two separate test runs, I considered it a good model.

2. Final Model Architecture

The final model architecture consisted of a convolution neural network with the following layers and layer sizes ...

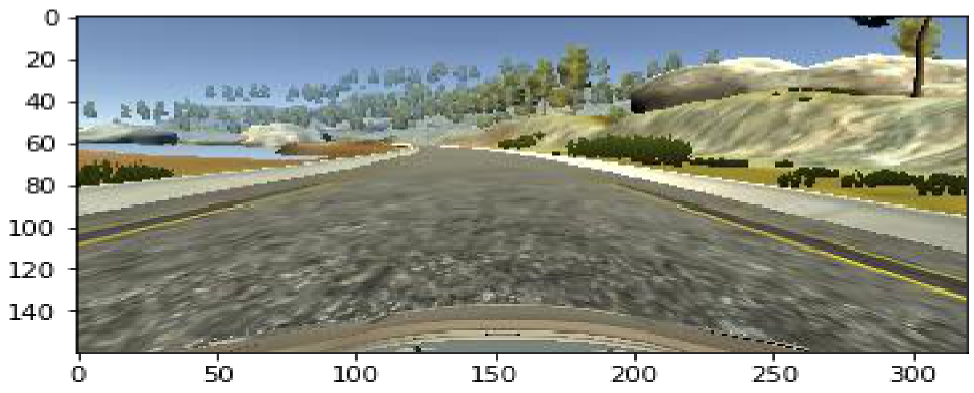
Here is a visualization of the architecture (note: visualizing the architecture is optional according to the project rubric)



```
____________________________________________________________________________________________________
Layer (type)                     Output Shape          Param #     Connected to
====================================================================================================
convolution2d_1 (Convolution2D)  (None, 19, 79, 24)    1176        convolution2d_input_1[0][0]
____________________________________________________________________________________________________
dropout_1 (Dropout)              (None, 19, 79, 24)    0           convolution2d_1[0][0]
____________________________________________________________________________________________________
convolution2d_2 (Convolution2D)  (None, 8, 38, 36)     13860       dropout_1[0][0]
____________________________________________________________________________________________________
convolution2d_3 (Convolution2D)  (None, 3, 18, 48)     27696       convolution2d_2[0][0]
____________________________________________________________________________________________________
convolution2d_4 (Convolution2D)  (None, 2, 17, 64)     12352       convolution2d_3[0][0]
____________________________________________________________________________________________________
convolution2d_5 (Convolution2D)  (None, 1, 16, 64)     16448       convolution2d_4[0][0]
____________________________________________________________________________________________________
flatten_1 (Flatten)              (None, 1024)          0           convolution2d_5[0][0]
____________________________________________________________________________________________________
dense_1 (Dense)                  (None, 100)           102500      flatten_1[0][0]
____________________________________________________________________________________________________
dense_2 (Dense)                  (None, 50)            5050        dense_1[0][0]
____________________________________________________________________________________________________
dense_3 (Dense)                  (None, 10)            510         dense_2[0][0]
____________________________________________________________________________________________________
dense_4 (Dense)                  (None, 1)             11          dense_3[0][0]
====================================================================================================
Total params: 179,603
Trainable params: 179,603
Non-trainable params: 0
```

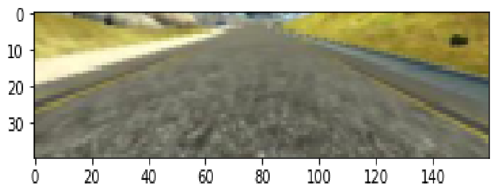
3. Creation of the Training Set & Training Process

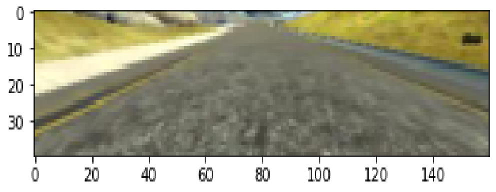
To capture good driving behavior, I made use of three extra data recordings in addition to the Udacity provided set. Here is an example image of center lane driving:

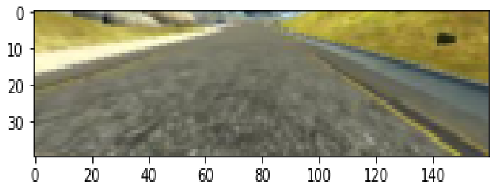


Every image was cropped with a slice of [55:135,:,:] and then resized to (40,160,3) (as shown below) this removed noise and background imagery like trees and clouds which did not have relevance to steering. It also made it slightly less load for training. I experimented with Greyscale but reverted back to RGB as it yielded stronger results on recognizing turns for me.

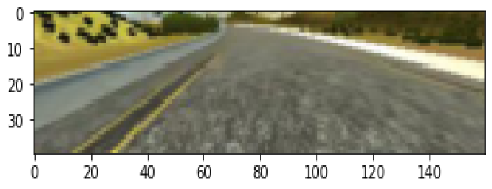
I then also made use of left side and right side cameras to simulate the car being further from center and modified the given angles to correct by +0.17 for left camera and -0.17 for right camera. The images L/R with angle correction can be interpreted as center camera views by the model. Below are cropped examples of center, left, and right camera views.

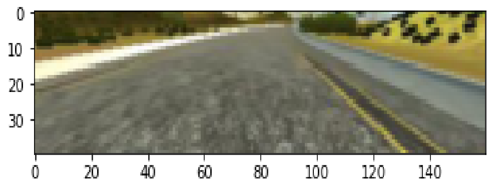






To augment the data sat, I gave each image a 50% chance of being angle perturbed (having angle adjusted minutely at a factor +/- 1.03) and 50% chance of being flipped with angle negated to create an opposing data image ... For example, here is an image that has been flipped:





In total I had 176700 (center + left + right camera) number of data points. I then applied a crop and X/255 normalization on each as preprocessing. Because I used a generator and image augmentation my accessible data is much higher though.

The generator randomized each selection for augmentation and feeding by taking a random image/angle pair from the full train data set. So I didn't have need to explicitly shuffle.

I used this training data for training the model. The validation set helped determine if the model was over or under fitting.