Behavioral Cloning Project

On my github P3 project repository I have placed the 5 required files, in addition to the brief readme file. **TL;DR—see the red bolded parts.**

Training Data

At first I decided to create the driving data files for each of the two tracks, and then discovered it was not so trivial (my gaming skills are limited). I **discovered the Udacity data and decided to use it**—it had about 8000 photos from each of 3 perspectives: center, left and right.

While I expected most of the driving samples to represent straight driving, with minimal steering input, the **skew to the data truly surprised me.** In the plot below, the frequency is plotted on a log-scale, just to show that there are a few samples in each bin. The steering data ranges from -1 to +1, which is mapped to a steering angle range of 25° left to 25° right.

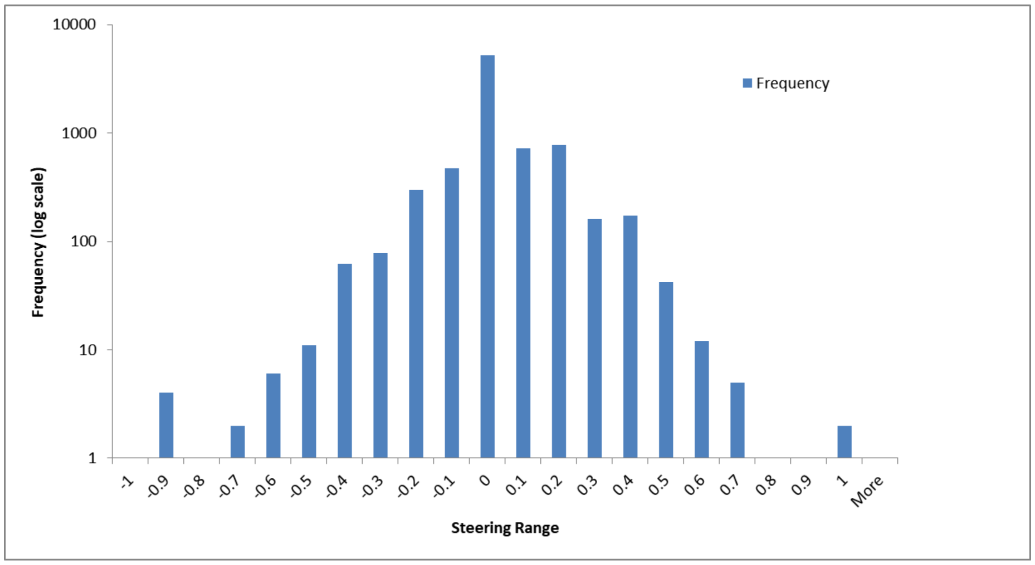

Figure 1. Of the 8036 samples in dataset, about 5200 samples had negligible steering angle.

In the data acquisition part of the model, I decided to randomly **omit 80% of the images with steering range under ±0.15,** leaving approximately 1200 images reflecting straight driving. Thus the data still has a slight bias towards straight driving, but not as excessive as before.

While I was experimenting with creating my own dataset, I drove the car on track 1 in the other direction to offset any handedness bias that would exist in any closed loop. I felt a more effective way of addressing this would be to **augment the data by L-R flipping and reversing (* -1) the steering angle** affiliated with that image. This resulted in a total available dataset size of about 6500 images, with 80% used for training and the remainder for validation .

Realizing that Keras is more efficient in cropping and normalizing data, I chose to **utilize the inherent parallelism in Keras' Cropping2D and Lambda** functions for cropping and normalizing the images. Another side benefit of this was that I did not need to modify *drive.py*.

With my very first network implementation (based on the single Flatten layer, and output layer covered in the lecture), my car went off course and in circles with steering locked at maximum value. On the other extreme, I considered using the Nvidia paper's model (multiple convolution layers & fully connected layers). I expected to use images from the left and right cameras as well, at which point I felt I would need to use generators to more efficiently use the memory on my system's GPU. As it turns out I got reasonably good results (eventually) with a far simpler model (than Nvidia's) and only using the center camera view for training. Thus I **ended up not using the generator functions.**

Model Architecture

After using the simple architecture covered in the lecture, as a proof of concept, I jumped to the other extreme of building a network architecture with 3 convolution layers and 3 fully connected layers. The result of the latter was that the car would generally drive at a slightly off-center, but constant, angle and quite quickly veer off the track. With the fully-connected layers, the parameters file: *model.h5* was 180MB!

At this point, after a lot of unsuccessful experimentation, I **decided to scale back my model** to just 1 convolution layer and 1 fully connected output layer. At this point I also decided to eliminate 80% of the data points with near-zero steering angles. This resulted in a mostly working autonomous driver that got off the course on Track1 at only 2 points (albeit at a slow speed of 9mph).

```
____________________________________________________________________________________________________
Layer (type)                     Output Shape          Param #     Connected to
====================================================================================================
cropping2d_1 (Cropping2D)        (None, 84, 220, 3)    0           cropping2d_input_1[0][0]
____________________________________________________________________________________________________
lambda_1 (Lambda)                (None, 84, 220, 3)    0           cropping2d_1[0][0]
____________________________________________________________________________________________________
convolution2d_1 (Convolution2D)  (None, 80, 216, 32)   2432        lambda_1[0][0]
____________________________________________________________________________________________________
maxpooling2d_1 (MaxPooling2D)    (None, 40, 108, 32)   0           convolution2d_1[0][0]
____________________________________________________________________________________________________
convolution2d_2 (Convolution2D)  (None, 36, 104, 64)   51264       maxpooling2d_1[0][0]
____________________________________________________________________________________________________
maxpooling2d_2 (MaxPooling2D)    (None, 18, 52, 64)    0           convolution2d_2[0][0]
____________________________________________________________________________________________________
convolution2d_3 (Convolution2D)  (None, 14, 48, 32)    51232       maxpooling2d_2[0][0]
____________________________________________________________________________________________________
maxpooling2d_3 (MaxPooling2D)    (None, 7, 24, 32)     0           convolution2d_3[0][0]
____________________________________________________________________________________________________
flatten_1 (Flatten)              (None, 5376)          0           maxpooling2d_3[0][0]
____________________________________________________________________________________________________
dense_1 (Dense)                  (None, 1)             5377        flatten_1[0][0]
====================================================================================================

Total params: 110,305
Trainable params: 110,305
Non-trainable params: 0
```

Figure 2. Summary of model architecture used in this project: **3 Convolution Layers + 1 FC Output layer.**

To eliminate the situations where the car got off track all the way to 30mph, I needed to add two more convolution layers. Training loss and validation losses converged very rapidly within 2-3 epochs, but I usually ran the models for 5 epochs.

As mentioned previously, cropping and normalization was done within Keras as it efficiently utilizes parallel computing operations. The incoming images of 160 x 320 had 46 pixels removed from the upper portion of the image, 30 from the bottom and 50 pixels from each side, as I estimated that the steering cues came from mostly the center portion of the image. Normalization was implemented with a Lambda function to center the pixel intensities around 0 and a scale of ±0.5 to help convergence.

Convolution layer #1 had a 5x5 filter, depth of 32 and ReLU activation, followed by MaxPooling to help reduce overfitting and help robustness.

This was followed by Convolution layer #2 also with a 5x5 filter, depth of 64 and ReLU activation; while the filtering further "squeezes" the data, important features could "grow" in the additional number of layers. This was also followed by MaxPooling.

The third and final Convolution layer had a 5x5 filter and depth of 32 plus ReLU activation, followed by its own MaxPooling layer. The output tensor was flattened to a linear array and then fed into the fully connected output layer with 1 output.

The resulting **parameter model file: 3convdataflip_final.h5 is 1.3MB in size**. I realized that adding fully connected layers drastically increases the size of these weights/ parameters, and at least for this project did not seem to contribute much. A smaller file size of weights/ parameters should also result in faster/ more efficient performance at the inference stage.

My penultimate model had an additional fully connected layer with 16 neurons prior to the output FC layer. This made the size of the model parameters jump to 12MB with no discernible performance gains.

I used **Keras' Adam optimizer** with its default learning rate of 0.001, and the loss function optimized was the mean square error. The **training data used was shuffled and split in an 80:20 ratio** for the training and validation sets. Training loss and validation loss converged rapidly to about 0.015 and 0.02, respectively, in 2-3 epochs.

Conclusion

This was a really good project, that I think helped me appreciate network training a little better. Keras makes it far too easy to slap on multiple layers very quickly, but a good network will not fix bad data. I realized again that really visualizing the training data is critical to a successful implementation.

Thanks for directing us to the Nvidia paper—though it made me wonder about the "value" or even need to explicitly do lane finding (I guess Project 4 will address that). However, for our situation a much simpler model works well. **I am working on getting my model to work on the entire stretch of Track 2, and will soon get an answer to more data? or bigger/better model, or both?**