Jeffrey Wolberg

**Project Writeup**

Output files:

Model.py file and drive.py file can be found in the working directory, **run4\_my\_model\_26.h5** can be found in the folder trained\_models, and **run4**.**mp4** can be found in the directory output\_videos.

Model Architecture:

The model first takes the (160,320,3) image and crops 50 pixels from the top and 20 from the bottom to produce (90, 320, 3)

I then use a lambda layer to normalize the model.

I then use the nvidia architecture (which contains 5 convolutional layers, each one followed by relu. The number of output features for each layer are 24,36,48,64,64, and I use either 3x3 or 5x5 kernels. I use maxpooling after each layer to shrink the input space), with a slight modification. Instead of using stride, I use a maxPool2D layer with a pool size of (2,2) to lower the size of the feature map by a factor of 4. Pooling, as opposed to striding, helps you glean the most out of a feature map because we first collect all the convolutions and then keep the strongest one, instead of skipping over certain convolutions with a stride>1, which can cause you to lose valuable parts of the feature map (the downsize is that this takes more time). A convolutional neural network is also ideal for this type of task because it is necessary to extract spatially changing features from the image (although the features in this task are usually found in the same region of the image), which is what CNNs are great at. I used 5 convolutional layers because each layer progressively determines more complex features, from edges and shapes in the beginning layers, to more intricate features later on.

After that, I use a dropout layer to prevent overfitting by ensuring that the model is not too reliant on any given feature.

Then, I flatten the feature map, and pass it through Dense fully connected layers, which produces outputs layers with sizes 100, 50, 10, and finally, 1. The one output is the steering angle for the input frame.

Training Strategy:

I used a learning rate of .0005 (which is half the default .001), and I used the Adam optimizer. I also used only 2 epochs. I did this because I had quite a lot of training data (I collected my own as well as used Udacity’s). I also used a batch size of 32, which updates the weights after accumulating the loss for only 32 images. I had len(data)/32 updates to my weights per epoch. I used 80% of my data for training, 20% for validation.

Training Data Documentation:

I used udacity’s data in addition to collecting my own. I drove counterclockwise once in order to add diversity to the model. I also used the left and right camera images, and added/subtracted a correction to the steering angle in order to simulate more data that was taken from the center camera (the network is being trained to work off of center images only). I also flipped the center image as well. Therefore, for every center image, there were 3 more images, which boosted the amount of data that I had. I also included data of the vehicle turning back from the side of the lane into the center, in order to expose the network to cases where the vehicle is driving close to the side of the road. Please check out **sample\_training\_data/IMG\_train11/** for examples of this.

Output files:

**run4\_my\_model\_26.h5** can be found in the folder trained\_models, and **run4**.**mp4** can be found in the directory output\_videos.