**Behavioral Cloning**

**Files Submitted**1. My project includes the following files:

* *model.py* – generates and saves the convolution neural network used to model the driver behavior
* *model\_retrain.py* – loads a previously saved model and trains it on new data
* *drive.py* – connects to the driving simulator and uses the specified model to autonomously control the simulated vehicle
* *video.py* – generates a video from the output frames from the simulator
* *model\_trained.h5* – the trained model
* *output.mp4* – the video of the simulator in autonomous mode, being controlled by the model

2. The Udacity simulator can be run using my mode with the following command:

python drive.py model\_trained.h5

3. Running *model.py* will attempt to train the model using a convnet on data generated using the Udacity simulator in training mode. The model is afterwards saved locally as *model.h5*. This module shows the pipeline used to pull and process the training data, and also shows the structure of the convnet used.

Running *model\_retrain.py* will attempt to load a previously saved model and train it on a new data set. This module is used to incrementally train the model in case more training data is needed to further train a somewhat successful model.

### **Model Architecture**

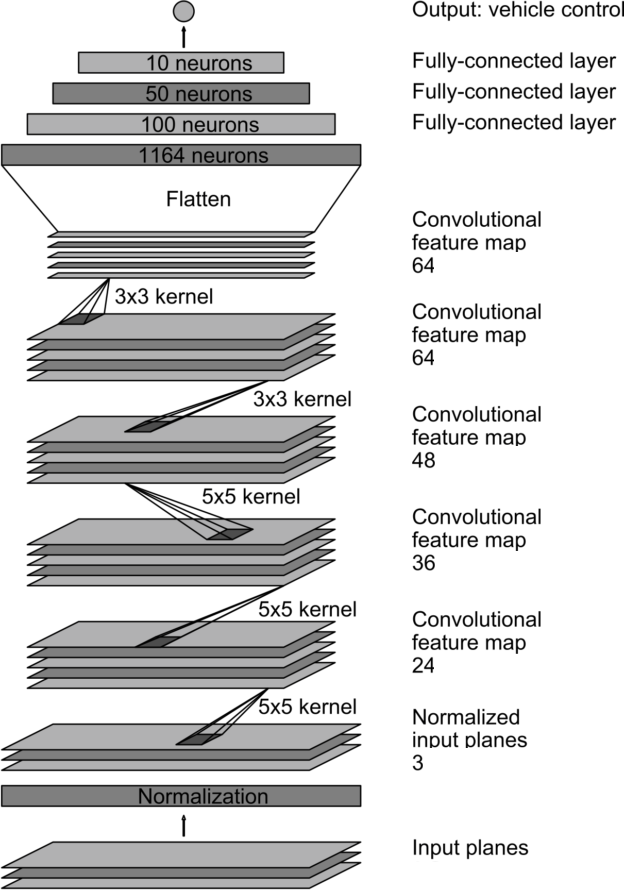
**1. Pre-Processing**

The only pre-processing done is normalization on line 60 of *model.py*, which normalizes the pixel color values to a range of [-0.5, 0.5].

**2. Model Architecture**

The model I used is a convolutional neural network based on nVidia’s end-to-end deep learning CNN for autonomous driving, found here <https://devblogs.nvidia.com/parallelforall/deep-learning-self-driving-cars/>

A visual model is provided by nVidia:



In this application, the three input planes are the RGB layers of the vehicle cameras, and the output is the steering angle. The model is implemented between lines 59 and 95 of *model.py*. This model features 5 convolution layers as seen above and 4 full-connected layers with ReLU activations between each layer to introduce nonlinearity.

**3. Model Parameters**

The ‘out-of-the-box’ Adam optimizer from Keras was used to train the network, so the learning rate was left at the default 0.0001. Otherwise, no other parameters were changed.

**4. Reducing Overfitting**

Separate data sets were used for training and validation to identify when the model was overfitting based on the training and validation performance. The split is carried out on line 17 of *model.py* and line 58 of *model\_retrain.py*.

### **Training Strategy & Results**

**1. Approach**

I decided to use the Udacity simulator to generate my own training data so that I could get a better sense of how the model behaves based on the kind of data I generated. My approach would be to train the model on my simulated data, then test out the model on the autonomous mode in the Udacity simulator. For the driving behavior, I planned to stay in the center of the lane as best as possible and to make my steering inputs as accurate and as smooth as possible.



*Vehicle located in the center of the lane*

I also planned to drive the course in the reverse direction to eliminate any left turn / right turn bias. Based on the performance, I would decide what to do next. In general, I always stuck for 5 epochs when training the model.

**2. Initial Results**

After training the model using ~1500-2000 data points, the results were abysmal. The vehicle would veer off the road almost immediately after starting the sim. I realized I probably needed more data points to train the model. I generated more training data (~8000-9000 points) and trained the model again, and this time the behavior was better but still not satisfactory. The vehicle would start moving along the road and slightly follow the initial bend of the first turn, but as the turn radius decreased, the vehicle would slowly veer off road.

**3. Improving the Model with More Data**

Since the vehicle was having trouble successfully completing a turn, I thought I again needed more data. To generate more points, I decided to use the images from the left and right cameras and the adjusted steering angles for these images, thereby tripling my data. Since I was dealing with a large amount of data now, I started to think of ways to cut down processing time. In the end, these efforts wasted much more time than it saved, as explained below.

**4. Handling More Data**

With so much training data (close to 30,000 points), I decided that maybe I should try to compress each image by resizing it to half its original size. I tried to integrate the resizing into the model using a Lambda layer, but this was *not* helpful. In order to save and reload the image properly, I had to define a standalone function to be used in the Lambda layer that explicitly imported tensorflow from the keras backend. This significantly increased the training time by 10-15 times the original training time for reasons that I’m still not sure of.

Because I tripled the amount of data points and added some more processing at the same time (calculating the adjusted steering angles for the left/right camera images), I figured they were causing the slow performance and I didn’t realize the resize layer was the real culprit. The model architecture was still usable with the Lambda resize layer, but training took several hours (if it didn’t shut down due to OOM issues). Once I discovered the reason for the slowdown, I removed it and decided to use the full-sized image.

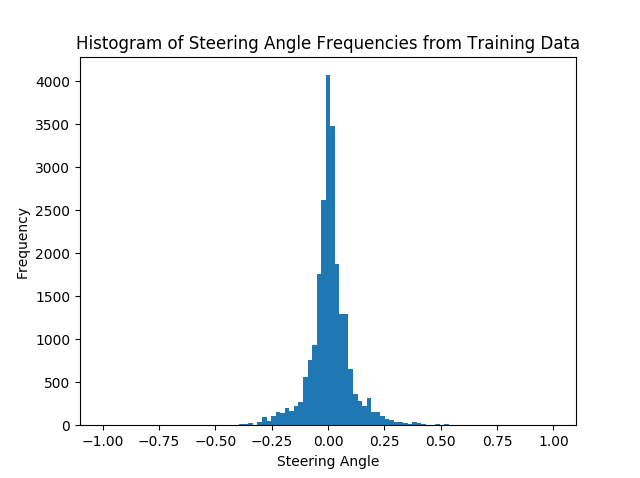
**5. Final Data Collection**

To move towards a solution faster, I then decided to discard the left/right camera images as it would remove a parameter (the steering adjustment value) and instead generate more data by flipping each image and accompanying it with the negative of the steering angle. After training on ~21,000 images, the results were improved – the model was able to take the vehicle completely around the first gentle bend and across the bridge. However, as soon as it got to a tight turn (after the bridge, as pictured below), it failed as the vehicle went off-road.



*Problematic turn ahead*

Based on this performance, I noticed the model was able to handle large-radius turns well but didn’t know how to handle tight turns. I realized that since the majority of the train is either very gentle turns or straight runs, so much of the data is for steering angles close to 0.0 – the histogram below illustrates this clearly:



I decided I would also include a small amount of examples of ‘recovery’ in my original data set (~1500 example images), where the vehicle starts close to a lane divider and gently moves back towards the center of the lane. I trained a model on the entire data set. An example of a gentle recover is illustrated below:

*Vehicle starts towards left divider ---> Vehicle steers slightly to right ---> Vehicle eventually recovers*

The performance was better and the vehicle was getting closer to making that turn, but was still going off road because it wasn’t turning sharply enough.

To teach the model to be more ‘comfortable’ about taking sharp turns, I generated a new set of data that included both aggressive recovery examples and also examples of exaggerated, slightly over-shot turns with eventual recovery. I figured a slightly over-shot turn with eventual recovery would more forcefully influence the model to take sharper turns. An example of an aggressive recovery is illustrated below:

*Vehicle starts towards left divider ---> Vehicle steers sharply to right ---> Vehicle quickly recovers*

To make sure these ~5000 data points were more influential, I took the existing model, already trained on the entire set, and trained it on these points with a larger learning rate of 0.001 as opposed to the 0.0001 used for the entire set. This is accomplished in the *model­\_retrain*.*py* module, which loads in a previously saved model on line 43 and trains it on new data on line 67.

**6. Final Results**

After training the model on both the original data set and the new one that included aggressive recoveries and exaggerated turns, the model was finally able to take the vehicle around track 1 successfully. See the video *output.mp4* for the results. It can be seen in the video that the vehicle stays towards the center of the road for the entire duration and does not touch the ledges on either side.