**Behavioral Cloning**

**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

You can run Track 1 with a speed of 30 and Track 2 with a speed of 15.

**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**

My model consists of a convolution neural network that is largely based on the architecture of NVIDIA (https://devblogs.nvidia.com/deep-learning-self-driving-cars/) and has been extended by some dropouts between the fully connected layers.

From line 97 (function "build\_NVIDIA ()") the model is created.

**2. Attempts to reduce overfitting in the model**

The model contains dropout layers in order to reduce overfitting (model.py lines 108, 110, 112).

**3. Model parameter tuning**

The model used an adam optimizer, so the learning rate was not tuned manually (model.py line 115).

**4. Appropriate training data**

The training data consists of a separate set of training images of track 1 and track 2. All 3 images (left, center, middle) were used.

**Model Architecture and Training Strategy**

**1. Solution Design Approach**

The overall strategy for deriving a model architecture was to use an existing architecture and adapt it to my training data.

The first step was to use a convolution neural network similar to the NVIDIA. This is due to documented success a good starting position.

In order to gauge how well the model was working, I split my image and steering angle data into a training and validation set. I found that my first model had a low mean squared error on the training set but a high mean squared error on the validation set. This implied that the model was overfitting.

To combat the overfitting, I modified the model so that I add additional dropout layers.

The final step was to run the simulator to see how well the car was driving around track one. There were a few spots where the vehicle fell off the track. to improve the driving behavior in these cases, I record more training data for this spots.

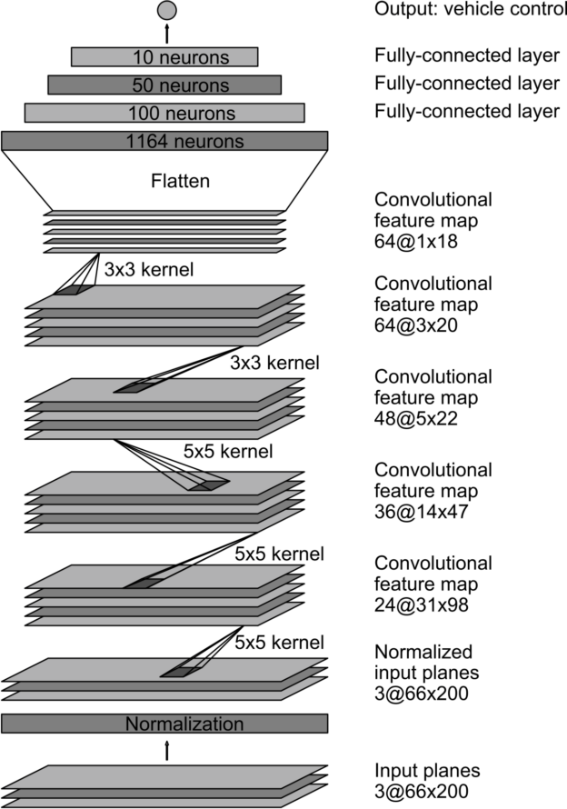
At the end of the process, the vehicle is able to drive autonomously around the track without leaving the road.

**2. Final Model Architecture**

The final model architecture (model.py lines 97-118) consisted of a convolution neural network with the following layers and layer sizes:

|  |  |
| --- | --- |
| Layer | Description |
| Input | 200x66x3 color Image |
| Convolution 5x5 | 2x2 stride, valid padding, RELU activation |
| Convolution 5x5 | 2x2 stride, valid padding, RELU activation |
| Convolution 5x5 | 2x2 stride, valid padding, RELU activation |
| Convolution 3x3 | 1x1 stride, valid padding, RELU activation |
| Convolution 3x3 | 1x1 stride, valid padding, RELU activation |
| Flatten | Output 1164 |
| Fully connected layer | Output 100, RELU activation |
| Dropout | Rate 0.2 |
| Fully connected layer | Output 50, RELU activation |
| Dropout | Rate 0.2 |
| Fully connected layer | Output 10, RELU activation |
| Dropout | Rate 0.2 |
| Output | Output 1 (Steering) |

Here is a visualization of the original NVIDIA architecture:



**3. Image Preprocessing**

Preprocessing the images takes 2 steps:  
- cropping (to remove unnecessary image data, e.g. Trees, the horizon, etc.)   
- resizing (to fit the 200x66 input size for NVIDIA architecture)

original Image

  
cropped image

  
resized image

**4. Creation of the Training Set & Training Process**

To capture good driving behavior, I first recorded two laps on each track using center lane driving. Here are example images of left, center and right lane driving (from Track 2):

I repeated this process on both tracks in order to get more data points.  
In addition to that, I also did the tracks counter-clockwise.

To modified the data, I flipped the images in 50% of the cases in which driving was not straight ahead.

I finally randomly shuffled the data set and put 20% of the data into a validation set (model.py line 116).

I used this training data for training the model. The validation set helped determine if the model was over or under fitting. The ideal number of epochs was 3 with a batch size of 128. I used an adam optimizer so that manually training the learning rate wasn't necessary. Here’s the plot of my „model mean squared error loss“.

