#**Behavioral Cloning**

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:

- model.py containing the script to create and train the model titled: (gen_bc_aws.py)
- drive.py for driving the car in autonomous mode this file has not been modified from the original from udacity (drive_udacity.py)
- model.h5 containing a trained convolution neural network this file is titled (model3.h5)
- writeup_report.pdf summarizing the results (this file)

####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_udacity.py model3.h5
```

####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. This code was based on the lessons and the Q & A walkthrough. Although I did also add some extras into the code by using the Keras cheat sheet to stop the epochs when it was trained enough.

###Model Architecture and Training Strategy

####1. An appropriate model architecture has been employed by using the NVIDIA End to End driving model.

My model consists of a convolution neural network:

That had Convolution layers of 24, 36, 48, with 5x5 and strides of (2,2).

This model also had convolution layers of 64 twice with 3x3 and stride of (1,1)

Then the model was Flattened.

Then the dense layers were used of 1164, 100, 50, 10, and finally 1.

The model includes RELU layers to introduce nonlinearity (code line 90-109 as needed), and the data is normalized in the model using a Keras lambda layer (code line 87).

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

The model contains dropout layers in order to reduce overfitting after each convolutional layer (gen_bc_aws.py lines 92-104 as needed).

The model was trained and validated on different data sets to ensure that the model was not overfitting (code line 16-17). The model was tested by running it through the simulator and ensuring that the vehicle could stay on the track.

####3. Model parameter tuning

The model used an adam optimizer, so the learning rate was tuned manually to 1.0e-4 as this was considered optimal for this type of model according to the Keras cheat sheet. (gen_bc_aws.py line 116).

####4. Appropriate training data

Training data was chosen to keep the vehicle driving on the road. I used a combination of center lane driving, recovering to account for recovering from the edge of the road I used the other camera images from the left and right side. For details about how I created the training data, see the next section.

###Model Architecture and Training Strategy

####1. Solution Design Approach

The overall strategy for deriving a model architecture was to ...start by using the examples in class.

My first step was to use a convolution neural network model similar to the LeNet architecture as this model might be appropriate because it showed some promise in driving the car.

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 included dropout.

Then I decided to change the model to a stronger model the NVIDIA self driving car model.

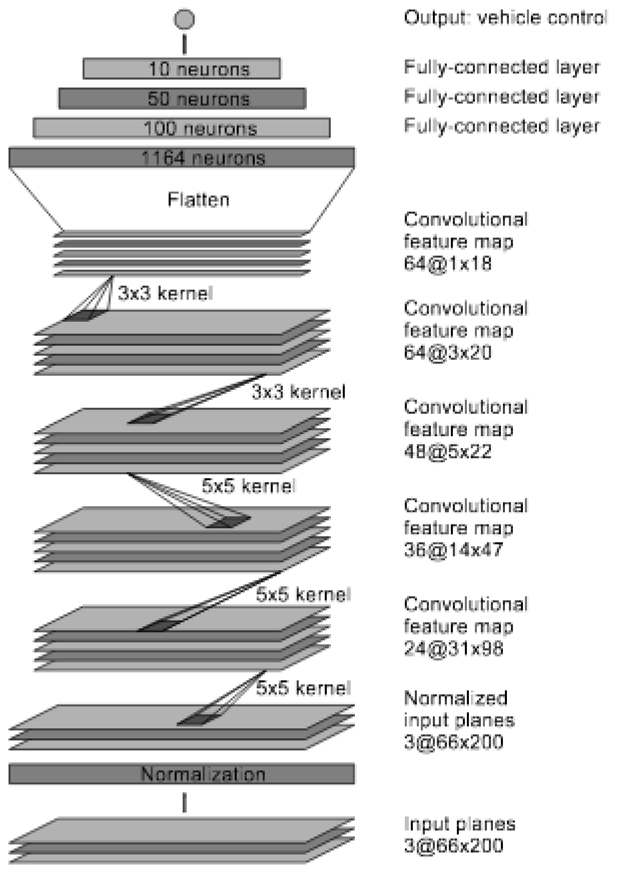
The final step was to run the simulator to see how well the car was driving around track one. There were a few spots where the vehicle fell off the track... to improve the driving behavior in these cases, I added more data on those parts of the road, example bridge and stipes.

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

####2. Final Model Architecture

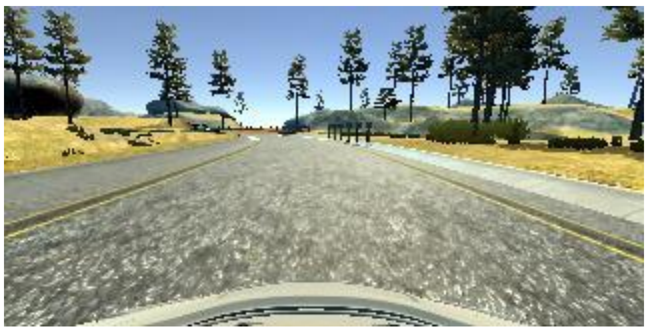
The final model architecture (gen_bc_aws.py lines 87-110) consisted of a convolution neural network with the following layers and layer sizes ...

Here is a visualization of the architecture (note: visualizing the architecture is optional according to the project rubric) From:
https://images.nvidia.com/content/tegra/automotive/images/2016/solutions/pdf/end-to-end-dl-using-px.pdf

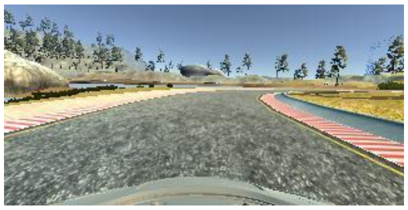
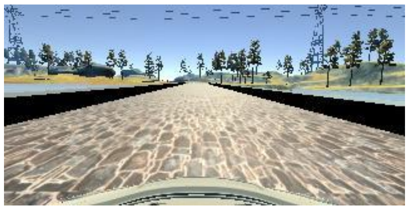


####3. Creation of the Training Set & Training Process

To capture good driving behavior, I first recorded two laps on track one using center lane driving. Here is an example image of center lane driving:



I then recorded the vehicle recovering from stripes and bridge parts of the roads These images show what a recovery looks like starting from my driving data:

Then I repeated this process on track two in order to get more data points.

To augment the data sat, I also flipped images and angles thinking this would give me more data to work with and would be like driving the track backwards.

I also used the Left and Right camera images and flipped those as well to give myself more data to work with.

After the collection process, I had 12,299 center image points with a total number of data points of 36,897 I then preprocessed this data by adding more data by flipping the images so I had a total data images of 73,794

I finally randomly shuffled the data set and put 20% of the data into a validation set.

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 as evidenced by process stopping after the 3 epochs. I used an adam optimizer and I manually changed the learning rate to 1e-4.