# Behavioral Cloning

**Behavioral Cloning Project**
The goals / steps of this project are the following:

1. Use the simulator to collect data of good driving behavior
2. Build a convolution neural network in Keras that predicts steering angles from images
3. Train and validate the model with a training and validation set
4. Test that the model successfully drives around track one and track two without leaving the road
5. Summarize the results with a written report

1 . My project includes the following files:

| File | Description |
|---|---|
| clone.py | containing the script to create and train the model |
| helper.py | Containing helper functions to read lines from CSV file, read images, flip images and generate training data |
| helper-test.py | Test the functionality of helper.py |
| linear.py | Linear regression model |
| lenet.py | LeNet convolutional neural network |
| nvidia.py | Nvidia convolutional neural network |
| nvidia-model-track-2-plus-track1.h5 | Trained Nvidia convolutional model with training data of track 2 (2 laps) and track 1 (1 lap) |
| drive.py | Driving car in autonomous mode |
| video.py | Filming autonomous driving car |

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
```

3. Submission code is usable and readable

The linear.py, lenet.py and nvidia.py file contains the code for building the convolution neural network.

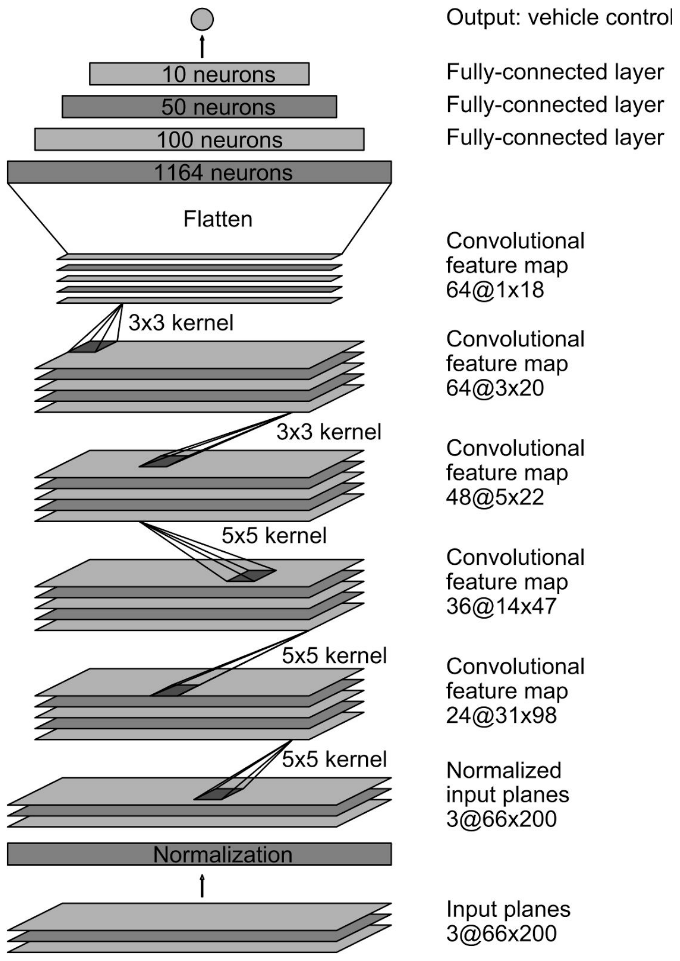
The helper.py contains functions that help to read lines from CSV file, parse the line and generate training data so that they can be fit in main memory.

The clone.py contains the pipeline I used for training and validating the model.

## Model Architecture and Training Strategy

### 1. An appropriate model architecture has been employed

I have built 3 models: linear regression model, lenet model and the model that's developed by Nvidia. I trained and tested all 3 models and found the Nvidia model best. Here's how the model looks like:



[1] credit: Nvidia model

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

When training the LeNet neural network, dropout layer can be added to avoid overfitting. However, according to Nvidia model, there's no dropout layer. I mainly restricted the epoch number to avoid overfitting.

### 3. Model parameter tuning

The model used an adam optimizer, so the learning rate was not tuned manually. However, I do print out the training loss and validation loss each epoch and select the favorable epoch. In the case of Nvidia model, I choose an epoch of 8.

4. Appropriate training data

Training data was chosen to keep the vehicle driving on the road. For track 1, I basically keep the car in the middle of the road while for track 2, I tried very hard to keep the car in the right lane.

For details about how I created the training data, see the next section.

## Model Architecture and Training Strategy

1. Solution Design Approach

A good self driving car's brain = model + training data.

In order to set up the development environment, I adopt the simplest model at first -- the linear regression model, and I use the sample training data provided by the course. In order to train the network, I deployed the model to AWS GPU instance and the training speed is very fast!

After the model is trained, I used 'scp' command to copy the model from my AWS GPU instance to the local machine and run the simulator. Looks okay but can't complete every the most basic track.

Then I decided to gather more training data. I trained the car for 1 lap of the basic track and 2 laps of the advanced track. Then I generated 6 times more training data by using center, left and right camera views and also flipping the images.

I also upgraded the brain from linear to LeNet and finally the Nvidia neural network! With the tuning of epoch steps, the car could drove a very long distance even in track 2! But the problem is that in track 1, the car tended to drive very close to the border since I only used the training data from track 2.

I helped the car to improve the driving skills by combining training data from track 1 and track 2. After that, the car can drive at the center of road in track 1 and at one lane in track 2! The vehicle is able to drive autonomously around the track without leaving the road!

2. Final Model Architecture

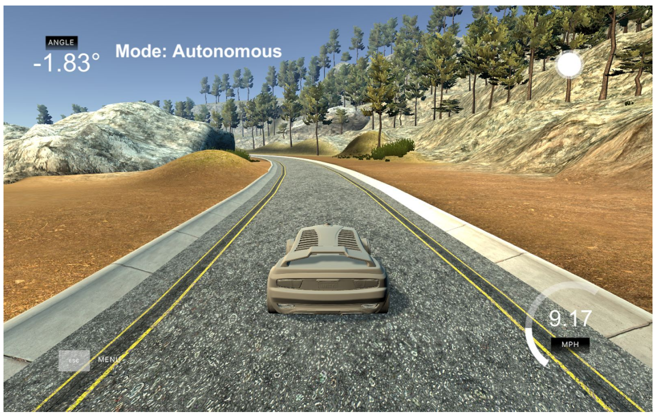
As I mentioned before, the final model architecture is the Nvidia model. Here's a table of model and training data that shows the performance:

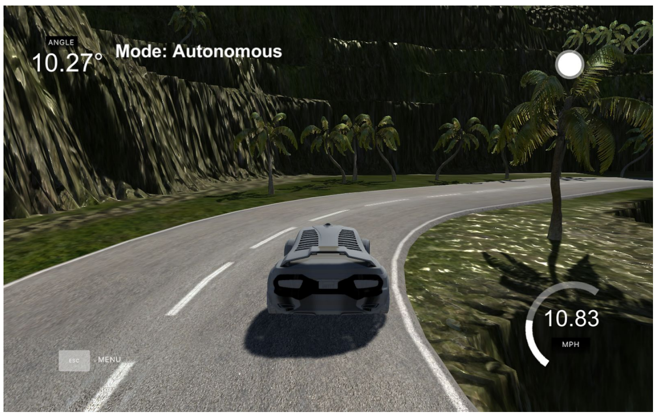
| Training data | LeNet | | Nvidia | |
|---|---|---|---|---|
| | Train loss | Validation loss | Train loss | Validation loss |
| Track 1 | 0.0722 | 0.0714 | 0.0462 | 0.0417 |
| Track 2 - 1 lap | 0.1154 | 0.1284 | 0.0695 | 0.1093 |

|  |  |  |  |  |
|---|---|---|---|---|
| Track 2 - 2 laps | N/A | N/A | 0.0750 | 0.1121 |
| Track 2 + Track 1 | N/A | N/A | 0.0729 | 0.0986 |

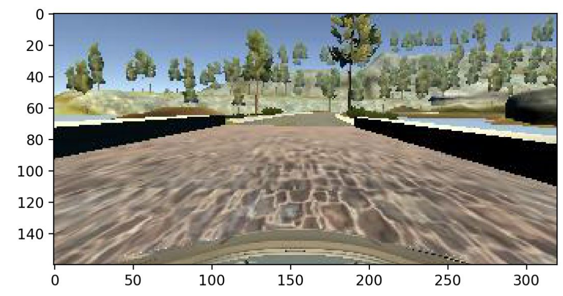
## 3. Creation of the Training Set & Training Process

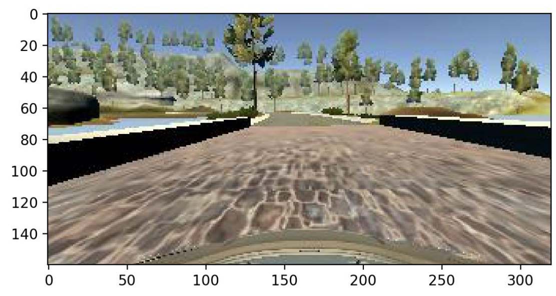
To capture good driving behavior, I first recorded 1 lap on track one using center lane driving. Then I recorded 2 laps on track two and made the car stay in one lane. Here is an example image of center lane driving (even though the images are gained from autonomous mode, this was how I trained the car):





To augment the data sat, I also flipped images and angles thinking that this would ... For example, here is an image that has then been flipped:





After the collection process, I had **102966** number of data points. I then pre-processed this data by adding a ***normalization*** layer and a ***cropping*** layer.

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 8 after since I found the validation loss slightly increases after that. I used an adam optimizer so that manually training the learning rate wasn't necessary.