#**Behavioral Cloning**

##Writeup Template

###You can use this file as a template for your writeup if you want to submit it as a markdown file, but feel free to use some other method and submit a pdf if you prefer.

---

**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](https://review.udacity.com/#!/rubrics/432/view) 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
* 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
```sh
python drive.py model.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.

###Model Architecture and Training Strategy

####1. An appropriate model architecture has been employed
In model at first, images are cropped to take out correct information and not irrelevant information. (model.py line 141)

My model consists of a 5 convolution layers with 3 of the layers 5x5 filter sizes and other 2 convolution layers with 3x3 filter sizes and depths 24, 36, 48, 64 and 64 for layers from first convolution to 5th convolutional layer. After each of the 1st 3 convolutional layers pooling/subsampling is done with sample size 2x2. (model.py lines 142-146)

The model includes RELU layers to introduce nonlinearity (model.py lines 142-146), and the data is normalized in the model using a Keras lambda layer (model.py lines 140).

| | KIND OF LAYER | LAYER DETAILS |
|---|---|---|
| 1 | Convolutional Layer | Depth-24, 5x5 filter, Activation-relu, subsample- 2x2 |
| 2 | Convolutional Layer | Depth-36, 5x5 filter, Activation-relu, subsample- 2x2 |

| | | |
|---|---|---|
| 3 | Convolutional Layer | Depth-48, 5x5 filter, Activation-relu, subsample- 2x2 |
| 4 | Convolutional Layer | Depth-64, 3x3 filter, Activation-relu |
| 5 | Convolutional Layer | Depth-64, 3x3 filter, Activation-relu |
| 6 | Flatten layer | |
| 7 | Fully Connected layer | Number of neurons - 100 |
| 8 | Fully Connected layer | Number of neurons - 60 |
| | Fully | Number of neurons - 40 |

| | | |
|---|---|---|
| 9 | Connected layer | |
| 10 | Fully Connected layer | Number of neurons - 10 |

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

The model was trained and validated on different data sets to ensure that the model was not overfitting (model.py lines 10-110). The model was tested by running it through the simulator and ensuring that the vehicle could stay on the track.
Two Dropout layers had been added in the network to reduce overfitting. (model.py lines 146,149)

####3. Model parameter tuning

The model used an adam optimizer, so the learning rate was not tuned manually (model.py lines 159).

#####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 from the left and right sides of the road.

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 use CNN along with full layer networks. The data that the model should be trained on should contain all possible to situations car may face in reality/test data.

My first step was to use a convolution neural network model similar to the Lenet. I then used some more complicated model provided by NVIDIA self-driving engineers. Then to this model i added some more fully-connected layers.

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 number epochs are reduced from 5 to 3.

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 may not be upto the mark and to improve the driving behavior in these cases, I thought more such kind of data should be recored in training.

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 140-157) consisted of a convolution neural network with the following layers and layer sizes:
5 Convolutional Layers - with depths 24, 36, 48, 64 and 64 respectively.
One Flattening layer
4 Fully-Dense Layers

####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 the left side and right sides of the road back to center so that the vehicle would learn to recover if in case. These images show what a recovery looks like:

To augment the data sat, I also flipped images. As lap driving mostly is counter clockwise, model was biased towards left and so data is augmented.

After the collection process, I had (16483-Train,4121-Validation) number of data points. I then preprocessed this data by normalisation.

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 validation error. I used an adam optimizer so that manually training the learning rate wasn't necessary.