#**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

###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:

1. Notebook that has the pipeline to –
    a. read images from the disk
    b. augment data
    c. preprocess them
    d. define model architecture (nvidia based)
    e. fit the model to training data and get training and validation accuracy
2. drive.py – driving the car in autonomous mode
   This has been modified with the preprocessing step which will be needed for every image that the camera loads while predicting
3. model-04.h5 containing a trained convolution neural network
4. A write up in the form of 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

####3. Submission code is usable and readable

The notebook 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

I used the nvidia model with a normalization at the top. The input image is of size 66x200 with 3 color channels

Layers description
Normalization Layer
Conv2D + Activation, Filter 24, size 5 x 5
Conv2D + Activation, Filter 36, size 5 x 5
Conv2D + Activation, Filter 48, size 5 x 5
Conv2D + Activation, Filter 64, size 5 x 5
Conv2D + Activation, Filter 64, size 5 x 5
Flatten Layer and Dense layer with output size of 100
Flatten Layer and Dense layer with output size of 50
Flatten Layer and Dense layer with output size of 1

For the activation we use ELU instead of RELU, because ELU which allows centering of values around zero by outputting a small negative activations for negative input and linear output for positive input. The data is normalized in the model using a Keras lambda layer.

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

Using both left and right camera images in addition to central images helps with generalization. Further by augmenting data with flipped images, and also adding more data with varying brightness, I make sure that we have enough data to avoid overfitting due to a smaller dataset.

####3. Model parameter tuning

The model used an adam optimizer, with a very small learning rate, to avoid any jumping around the minima.

####4. Appropriate training data

I used the training data provided in the repo. I started with a combination of center lane driving, recovering from the left and right sides of the road. I also augmented data with flipped versions of each of the center, left and right images, and finally added another version of the center image with its brightness changes.

###Model Architecture and Training Strategy

####1. Solution Design Approach

The overall strategy for deriving a model architecture was to start with Lenet and then slowing add data and try other established models. Lenet did quite well, but with a more powerful model like the one from nvidia would allow me to generalize better, hoping to run well on other tracks. Unfortunately, due to hardware and time constraints, I had to abandon some of the data augmentation techniques (which would blow up the amount of data and result in my computer to crash).

With just normalized and cropped images, the car started moving well on the track but soon crashed outside. Adding flipped images and then eventually adding images with brightness changes helped improve accuracy on the validation set, and drive well on the track.

####2. Final Model Architecture

I ended using nvidia architecture and to match its input, resize the images to 66x200.

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

I started building my own training set by driving around the track, but ended up using the training set provided, because my training dataset had too much noise, and would mean I would have to collect enough number of laps of data to do the training. This would generalize well, but would need much better hardware to train

I 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 accuracy went down and then started bumping up after the 5th epoch. I ended up recoding the video from the model output after 5th epoch, because the 6th and 7th video would have started overfitting

Video:

https://youtu.be/NxcBWMW4mHU