# \*\*Behavioral Cloning\*\*

## Writeup Template

\*\*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

```

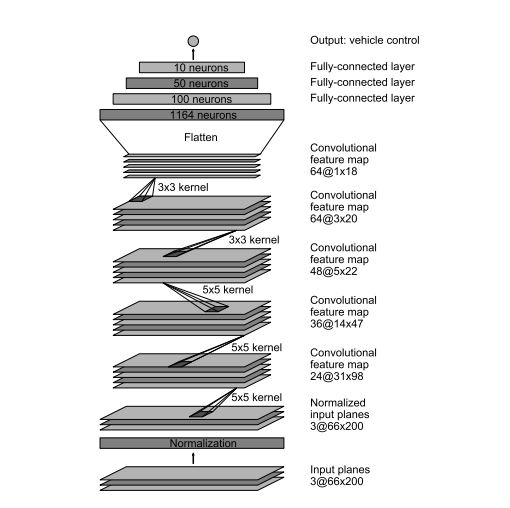
#### 3. Submission code is usable and readable

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

#### Overview

I decided to test the model provided by NVIDIA as suggested by Udacity. The model architecture is described by NVIDIA here. As an input this model takes in image of the shape (60,266,3) but our dashboard images/training images are of size (160,320,3). I decided to keep the architecture of the remaining model same but instead feed an image of different input shape .



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

My model consists of a convolution neural network with 2x2 filter sizes and depths between 24 and 64 (model.py lines 86-90).

##### Loading data

I am using OpenCV to load the images, by default the images are read by OpenCV in BGR format but we need to convert to RGB as in drive.py it is processed in RGB format.

Since we have a steering angle associated with three images we introduce a correction factor for left and right images since the steering angle is captured by the center angle.

I decided to introduce a correction factor of 0.2

For the left images I increase the steering angle by 0.2 and for the right images I decrease the steering angle by 0.2



##### Preprocessing

I decided to shuffle the images so that the order in which images comes doesn't matters to the CNN

Augmenting the data- i decided to flip the image horizontally and adjust steering angle accordingly, I used cv2 to flip the images.

In augmenting after flipping multiply the steering angle by a factor of -1 to get the steering angle for the flipped image.

So according to this approach we were able to generate 6 images corresponding to one entry in .csv file

##### Creation of the Training Set & Validation Set

I analyzed the Udacity Dataset and found out that it contains 9 laps of track 1 with recovery data. I was satisfied with the data and decided to move on.

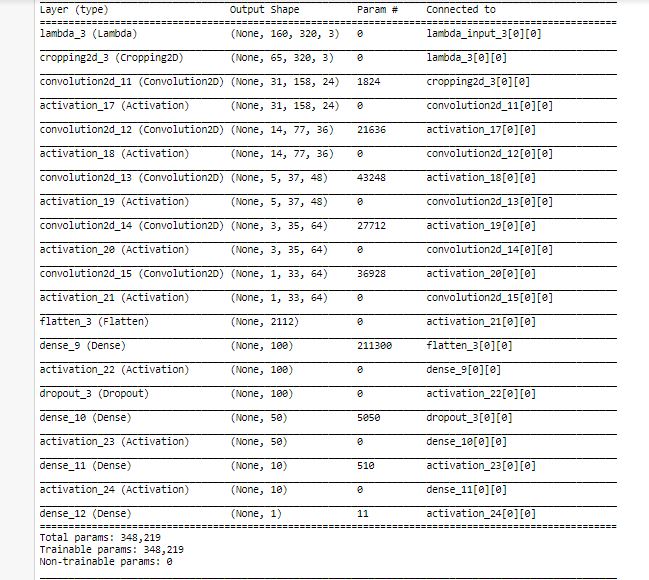
I decided to split the dataset into training and validation set using sklearn preprocessing library.

I decided to keep 15% of the data in Validation Set and remaining in Training Set

I am using generator to generate the data so as to avoid loading all the images in the memory and instead generate it at the run time in batches of 32. Even Augmented images are generated inside the generators.

##### Final Model Architecture

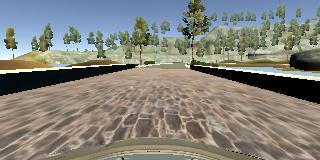
\* I made a little changes to the original NVIDIA architecture, my final architecture looks like in the image below



\* As it is clear from the model summary my first step is to apply normalization to the all the images.

\* As it is clear from the model summary my first step is to apply normalization to the all the images.

\* Second step is to crop the image 70 pixels from top and 25 pixels from bottom. The image was cropped from top because I did not wanted to distract the model with trees and sky and 25 pixels from the bottom so as to remove the dashboard that is coming in the images.



\* Next Step is to define the first convolutional layer with filter depth as 24 and filter size as (5,5) with (2,2) stride followed by ELU activation function

\* Moving on to the second convolutional layer with filter depth as 36 and filter size as (5,5) with (2,2) stride followed by ELU activation function

\* The third convolutional layer with filter depth as 48 and filter size as (5,5) with (2,2) stride followed by ELU activation function

\* Next we define two convolutional layer with filter depth as 64 and filter size as (3,3) and (1,1) stride followed by ELU activation funciton

\* Next step is to flatten the output from 2D to side by side

\* Here we apply first fully connected layer with 100 outputs

\* Here is the first time when we introduce Dropout with Dropout rate as 0.25 to combact overfitting

\* Next we introduce second fully connected layer with 50 outputs

\* Then comes a third connected layer with 10 outputs

\* And finally the layer with one output.

Here we require one output just because this is a regression problem and we need to predict the steering angle.

#### 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 (code line 14-15). 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 not tuned manually (model.py line 99).

No of epochs= 5

Optimizer Used- Adam

Learning Rate- Default 0.001

Validation Data split- 0.15

Generator batch size= 32

Correction factor- 0.2

Loss Function Used- MSE(Mean Squared Error as it is efficient for regression problem).

After a lot of testing on track 1 I was convinced that this is my final model.

#### 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. This was done for both tracks to help generalize the model.

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

### Model Architecture and Training Strategy

#### 1. Solution Design Approach

The model architecture I used was inspired from a similar network employed by NVIDIA team for steering control of an autonomous vehicle. I thought this model is appropriate because there are 5 convolutional layers and documented success of the network for steering control. In order to gauge how well the model was working, I split my image and steering angle data into a training and validation set in 80 to 20 ratio. The final step was to run the simulator to see how well the car was driving around the track. There were a few spots where the vehicle fell off the track on increasing the set speed in drive.py file. To improve the driving behavior in these cases, I augmented the training data with driving data from track 2.

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 85-95) consisted of a convolutional neural network with the following layers and layer sizes.

Layer Size

Input 65 x 320 x 3

Lambda (normalization) 65 x 320 x 3

Convolution with relu activation 5 x 5 x 24 with 2x2 filters

Convolution with relu activation 5 x 5 x 36 with 2x2 filters

Convolution with relu activation 5 x 5 x 48 with 2x2 filters

Convolution with relu activation 3 x 3 x 64

Convolution with relu activation 3 x 3 x 64

Flatten

Fully connected 100

Fully connected 50

Fully connected 10

Output 1

#### 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 navigate the turns instead of memorizing the track. These images show what a recovery looks like starting from right to center:







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 steering measurements thinking that this would help generalize the model. For example, here is a view from the first figure that has then been flipped:

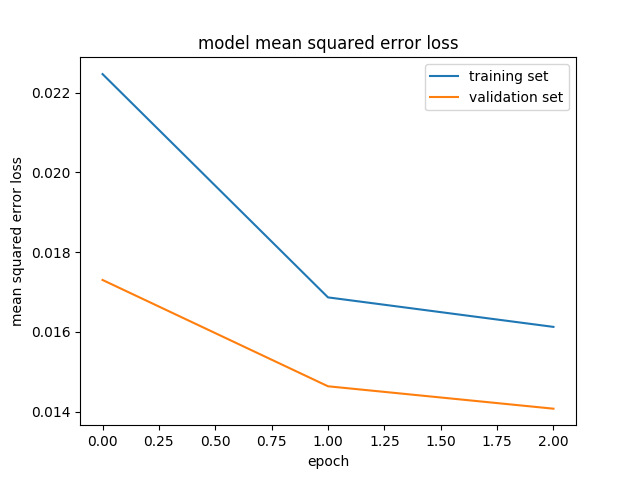
 

After the collection process, I had 47181 number of data points. I then preprocessed this data by cropping irrelevant data (sample below) from the top and bottom of the image. This led to a final image size of 65 x 320 x 3 which significantly reduced the computational requirements.

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 increasing loss after 3 epochs (see below). I used an adam optimizer so that manually training the learning rate wasn't necessary



Further improvements can be made to the simple PI controller used in drive.py to smoothen steering angles for the current model.