# Udacity Self-Driving Car Engineer Nanodegree - Behavioral Cloning Project.

## Introduction

The objective of this project is to teach computer to drive car on the basis of data collected in simulator provided by Udacity. Here we apply the concepts of Deep Learning and Convolutional Neural Networks to teach computer how to drive car autonomously.

We feed the data collected from Simulator to our model. This data is feed in the form of images captured by 3 dashboard cameras positioned center, left and right. The output data contains a file data.csv which has the mappings of center, left and right images and the corresponding steering angle, throttle, brake and speed.

Using Keras Deep learning framework we can create a model.h5 file which we can test later on simulator with the command "python drive.py model.h5". This drive.py connects our model to simulator. The challenge in this project is to collect sufficient training data so as to train the model to respond correctly in any type of situation.

## Files Submitted & Code Quality

### 1. Submission includes all required files and can be used to run the simulator in autonomous mode

My project includes following files:

- model.py containing the script to create and train the model
- drive.py for driving the car in autonomous mode
- model.h5 contains a trained convolution neural network
- writeup_report.pdf summarizing the process
- video.mp4, a video recording of vehicle driving autonomously around the track.

### 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**"

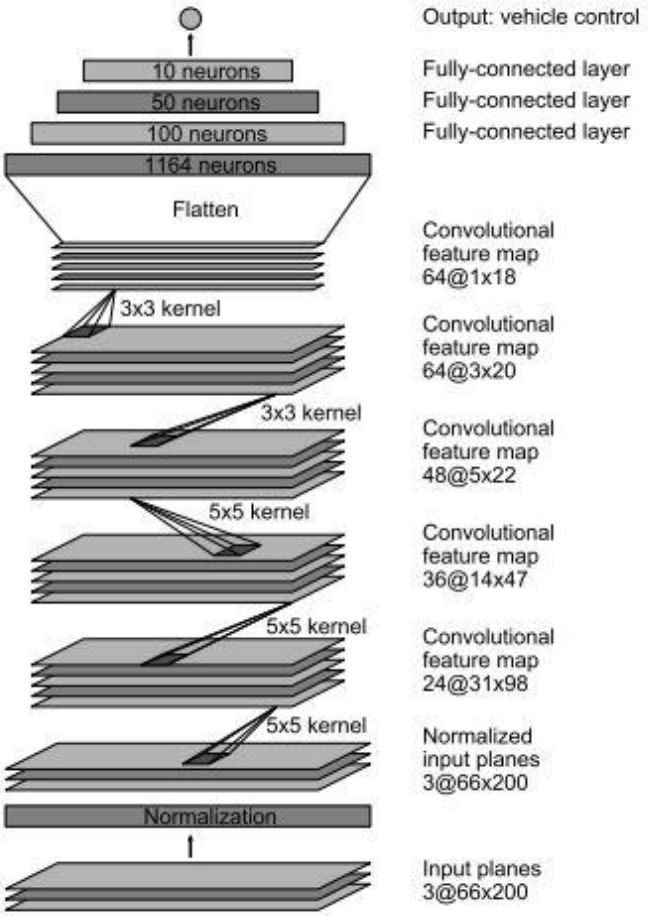
### 3. Submission code is usable and readable

The 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

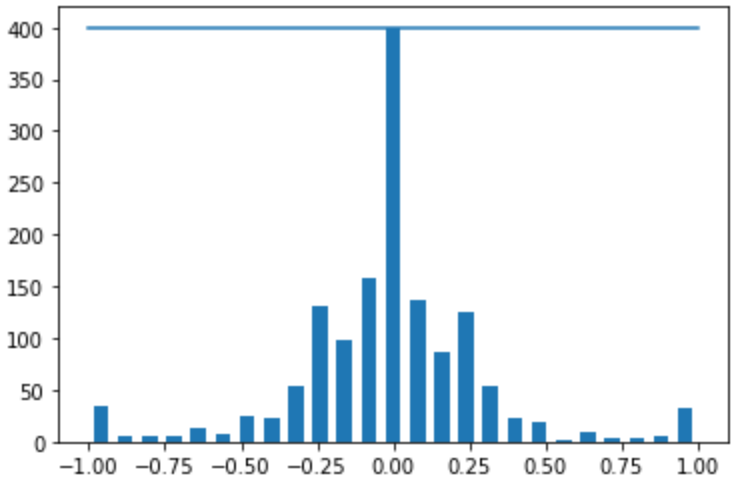
### Model Overview

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 which I will discuss later.




**Appropriate training data**

To ensure a balanced dataset and to gather more data thus helping the model to generalize, I used data from three laps in forward direction and another three laps in the reverse direction. Since this track when you first start has a left turn bias which will skew our data to one side. The collected data will be biased towards left turns. This creates a problem for the neural network as it could bias the model towards always predicting left turns which would have the car become biased towards driving left all the time crashing into the edges. But if we then drive in the opposite direction where we would now be taking mostly right turns the left and right steering angles in our data would become more balanced giving the model a new track to learn from helping the model generalize better




I am using panda liabrary to load the images

**Creation of the Training Set & Validation Set**

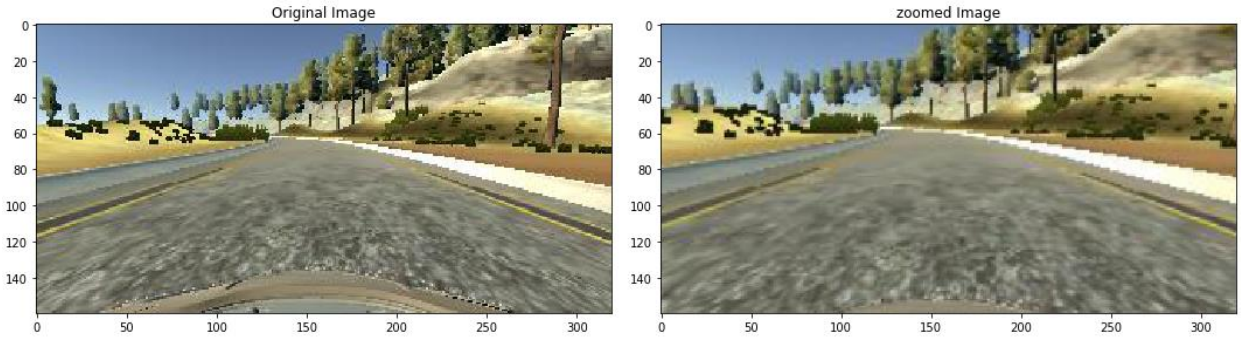
- I decided to split the dataset into training and validation set using sklearn preprocessing library.
- I decided to keep 20% of the data in Validation Set and remaining in Training Set
- Currently I am using ,training samples: 1170 and valid samples: 293
- I am using generator to generate the data so as to avoid loading all the images in the memory and instead generate it at the run time in batches of 100. Even Augmented images are generated inside the generators.

**Preprocessing**

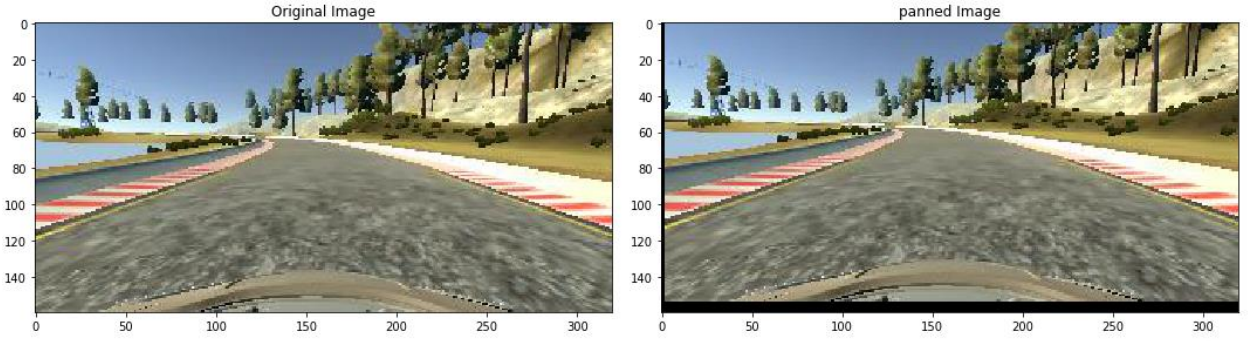
I decided to shuffle the images so that the order in which images comes doesn't matters to the CNN

Data augmentation is used to generalize the model and to reduce overfitting. This process consist of following steps

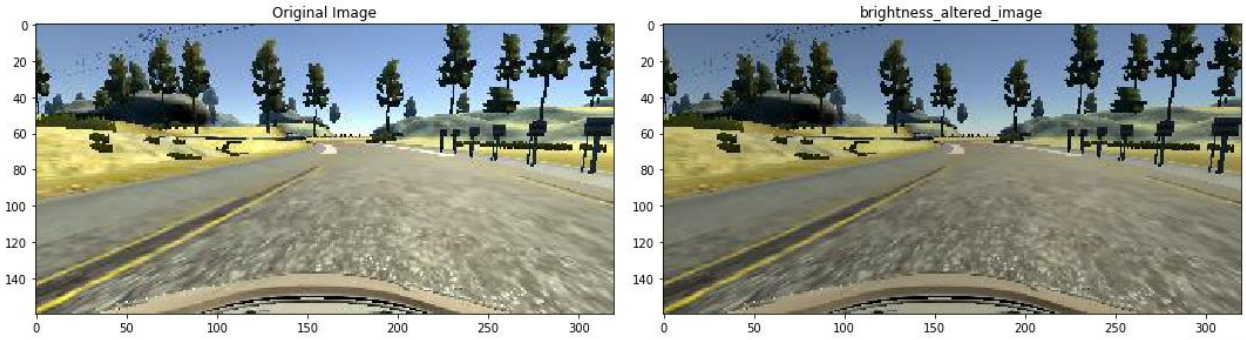
Zoom image:



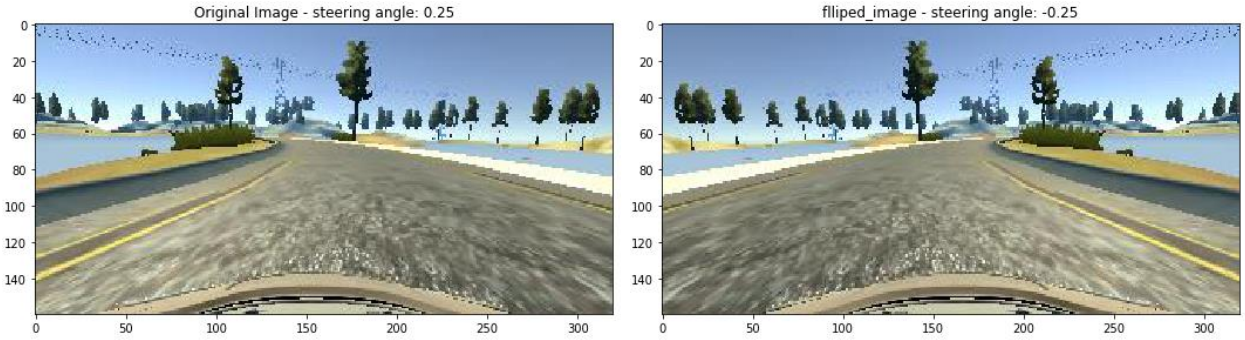

Pan image:




Brightness altered image




Flipped image




Final augmented image

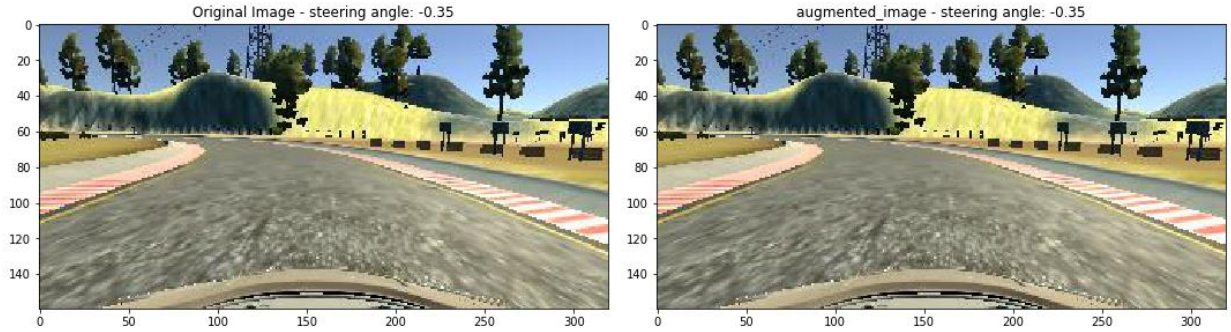
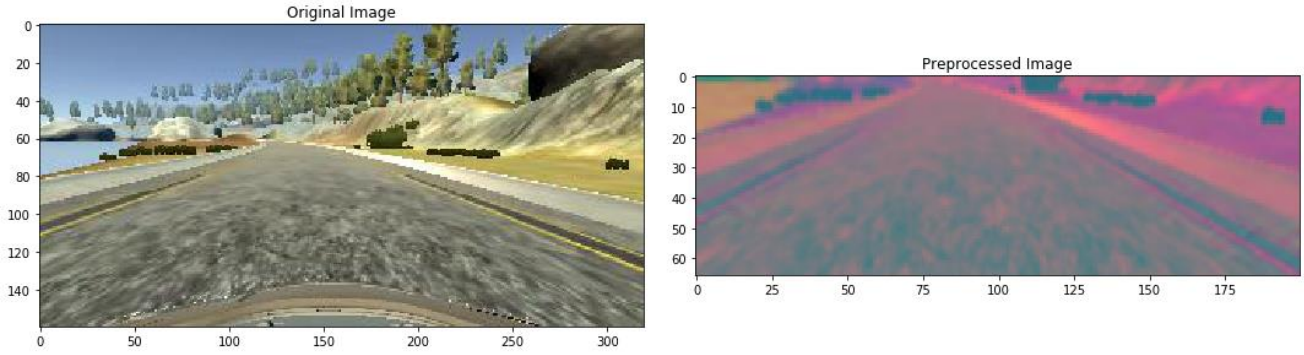



**Image preprocessing**

Steps involved are

- Cropping image as we are not interested in the scenery but only the road
- Changing the color space – This is relevant for pre-processing because the NVIDIA model architects themselves recommend that we use the YUV color space for a dataset as opposed to the default RGB format or even a greyscale image. For this reason we will also convert our color space to YUV
- Gaussian blur - This is applied using the CV2 gaussian blur function now the gaussian blur is useful to apply an image because it helps smoothing the image out and to reduce noise within the image.
- We are now going to resize our image and decrease its size. This will allow for faster computations as a smaller image is easier to work with. I used a CV2 command CV2 called resize which allows decreasing the size of our image to 200 by 66. Not only is this image smaller, it also matches the image size of the input images used by the nvidia model architecture.




## Final Model Architecture

Final model architecture contains following steps:

- First 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 RELU 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 RELU activation function
- The third convolutional layer with filter depth as 48 and filter size as (5,5) with (2,2) stride followed by RELU 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 RELU activation function
- Next step is to flatten the output from 2D to side by side
- Here we apply first fully connected layer with 100 outputs
- 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.

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_6 (Conv2D)            (None, 31, 98, 24)        1824
_________________________________________________________________
conv2d_7 (Conv2D)            (None, 14, 47, 36)        21636
_________________________________________________________________
conv2d_8 (Conv2D)            (None, 5, 22, 48)         43248
_________________________________________________________________
conv2d_9 (Conv2D)            (None, 3, 20, 64)         27712
_________________________________________________________________
conv2d_10 (Conv2D)           (None, 1, 18, 64)         36928
_________________________________________________________________
flatten_2 (Flatten)          (None, 1152)              0
_________________________________________________________________
dense_5 (Dense)              (None, 100)               115300
_________________________________________________________________
dense_6 (Dense)              (None, 50)                5050
_________________________________________________________________
dense_7 (Dense)              (None, 10)                510
_________________________________________________________________
dense_8 (Dense)              (None, 1)                 11
=================================================================
Total params: 252,219
Trainable params: 252,219
Non-trainable params: 0
_________________________________________________________________
None
```

**Attempts to reduce overfitting in the model**

I have used data augmentation so that the model generalizes on a track that it has not seen.

**Model parameter tuning**

- No of epochs= 10
- Optimizer Used- Adam
- Learning Rate- Default 0.001
- Validation Data split- 0.2
- Generator batch size= 100
- Loss Function Used- MSE (Mean Squared Error as it is efficient for regression problem).

**Output Video**

**https://www.youtube.com/watch?v=ceMT11-_Tio&feature=youtu.be**

**Important Sidenote**

- The data was cloned from github repository https://github.com/ChitraChaudhari/track, which is not included in submission.
- For data augmentation I am using imuaug liabrary, which might need to be installed