# Project : Behavioral Cloning

28.12.2018

Submitted By:

Lalu Prasad Lenka

## Overview

The project aimed at building a CNN based deep neural network that will be able to clone the driving behaviour of human driver, thus being able to drive autonomously on a simulator provided by Udacity.

## Goals

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 without leaving the road.
5. Summarize the results with a written report.

## Code Overview:

The code is structured as follows:

- config.py: project configuration and hyperparameters
- model.py: model definition and training
- drive.py: Customized the given drive.py a bit to preprocess the input images.
- load_data.py: definition of data generator & handling data augmentation
- visualize_data.py: exploratory visualization of the dataset, use in this writeup.
- Root folder contains pretrained model architecture and weights

## Model Architecture and Training Strategy

### 1. Choosing the Training data

Data for this task can be gathered with the Udacity simulator. When the   simulator is set to training mode, the car is controlled by the human though the keyboard, and frames and steering directions are stored to disk.

I downloaded the simulator locally and collected data by using it. But uploading data to Udacity workspace was very slow. So I decided to train using Udacity's own data.

Udacity training set is constituted by 8036 samples. For each sample, two main information are provided:

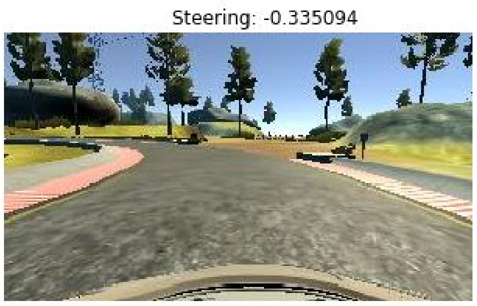
i) Three frames from the frontal, left and right camera respectively

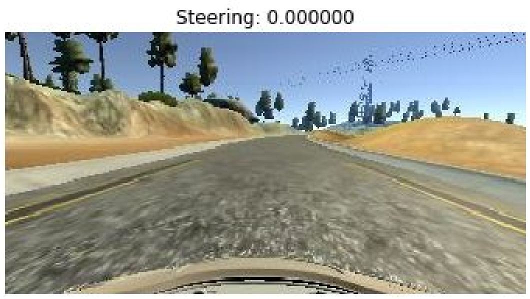
ii) The corresponding steering direction

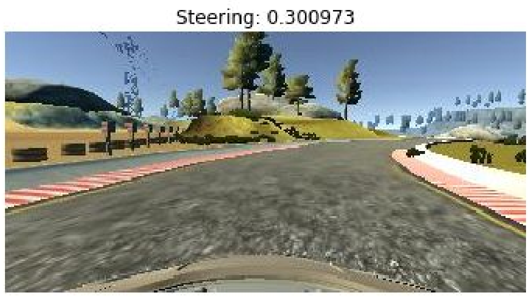
## 2. Visualization of Training data

Here's how a typical sample looks like. We have three frames from frontal camera as well as the associated steering direction.

All my visualization are done using ***visualize_data.py*** file.

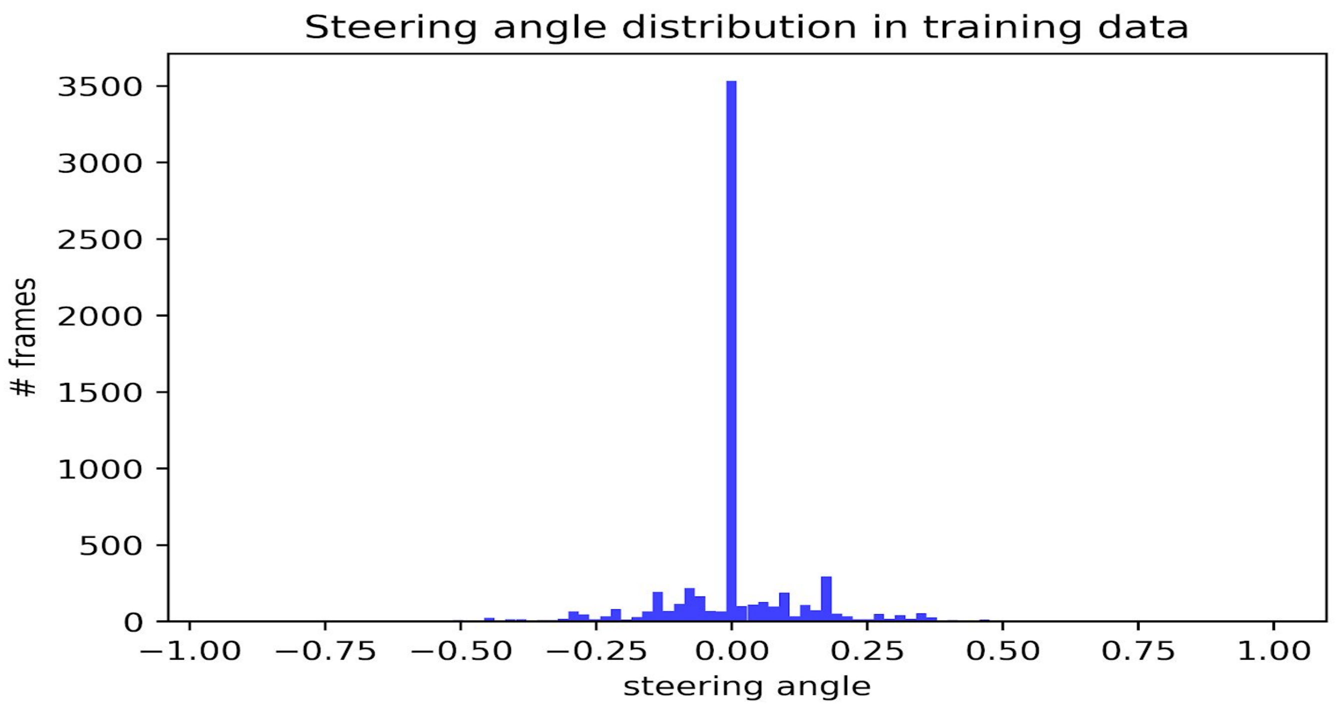






As we see, each frame is associated to a certain steering angle. Unfortunately, there's a huge skew in the ground truth data distribution: as we can see the steering angle distribution is strongly biased towards the zero.

I have visualised the training data distribution using a histogram plot.



## 3. Preprocessing of Training data & Data Augmentation

All the preprocessing done in preprocess() , *line 29* in ***load_data.py***
generate_data_batch() in *line 139* implements the generator to pre-process that on real-time.

Every frame is preprocessed by cropping the upper and lower part of the frame: in this way we discard information that is probably useless for the task of predicting the steering direction. Now our input frames look like these:

Cropping also helps speed up the training as the image size is reduced.

Due to the aforementioned data imbalance, it's easy to imagine that every learning algorithm we would train on the raw data would just learn to predict the steering value 0. Furthermore, we can see that the "test" track is completely different from the "training" one form a visually point of view. Thus, a network naively trained on the first track would *hardly* generalize to the second one. Various forms of data augmentation can help us to deal with these problems.

### Exploiting left and right cameras

For each steering angle we have available three frames, captured from the frontal, left and right camera respectively. Frames from the side cameras can thus be employed to augment the training set, by appropriately correcting the ground truth steering angle. This way of augmenting the data is also reported in the NVIDIA paper.

### Brightness changes

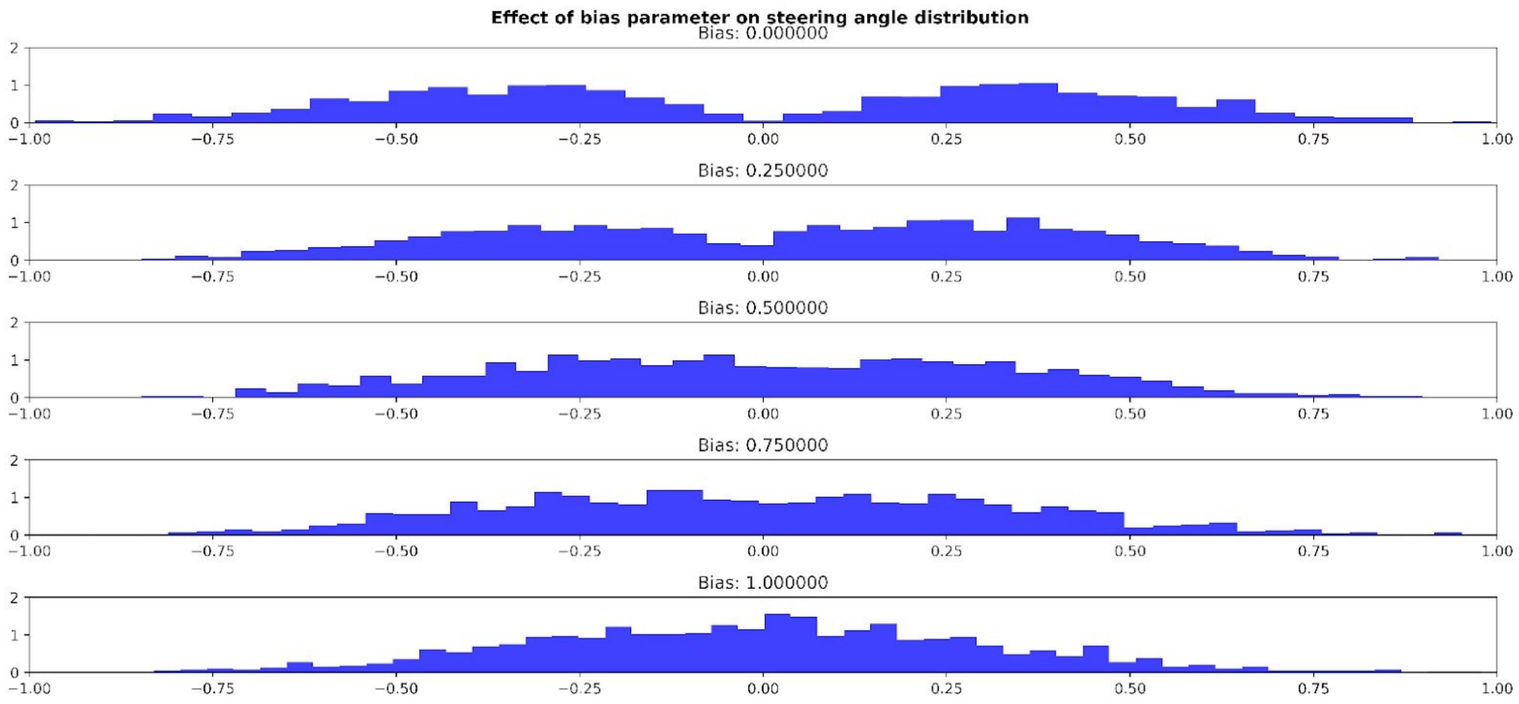
Before being fed to the network, each frame is converted to HSV and the Value channel is multiplied element-wise by a random value in a certain range. The wider the range, the more different will be on average the augmented frames from the original ones.

## Normal noise on the steering value

Given a certain frame, we have its associated steering direction. However, being steering direction a continuous value, we could argue that the one in the ground truth is not *necessarily* the only working steering direction. Given the same input frame, a slightly different steering value would probably work anyway. For this reason, during the training a light normal distributed noise is added to the ground truth value. In this way we create a little more variety in the data without completely twisting the original value.

## Shifting the bias

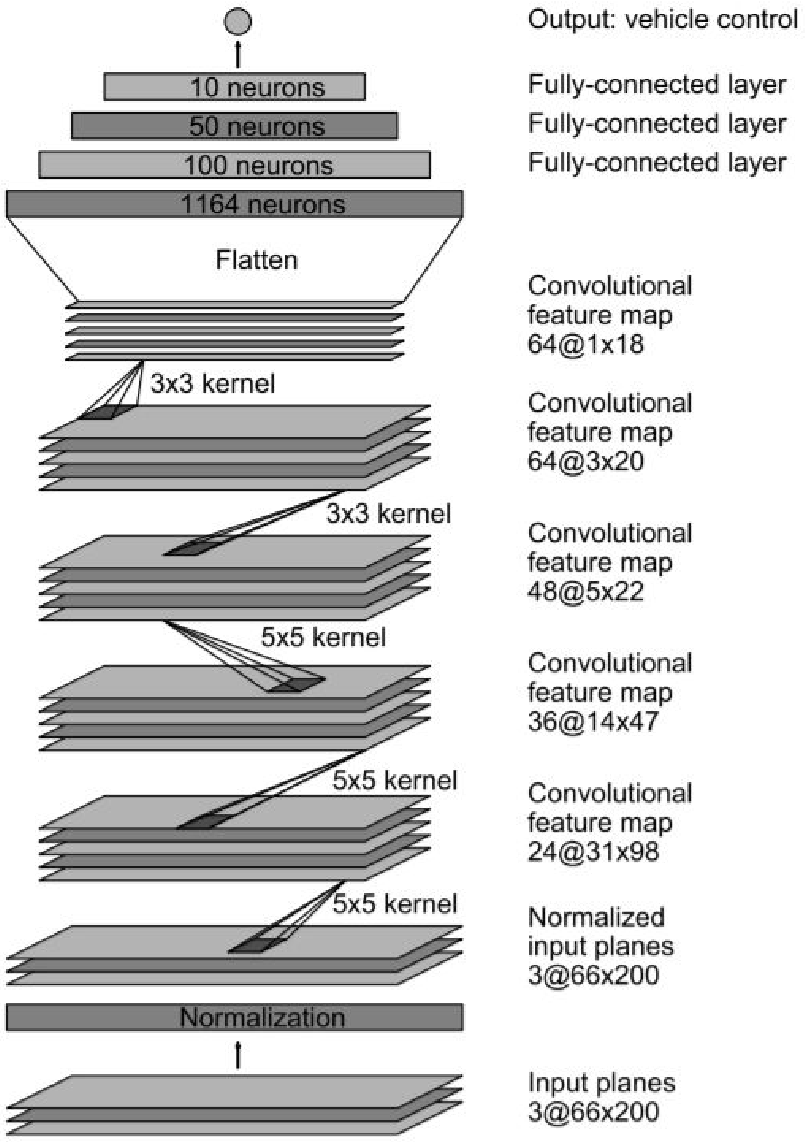
Finally, we introduced a parameter called bias belonging to range [0, 1] in the data generator to be able to mitigate the bias towards zero of the ground truth. Every time an example is loaded from the training set, a *soft* random threshold `r = np.random.rand()` is computed. Then the example i is discarded from the batch if `steering(i) + bias < r`. In this way, we can tweak the ground truth distribution of the data batches loaded. The effect of the bias parameter on the distribution of the ground truth in a batch of 1024 frames is shown below:



Final bias considered is **0.8** defined in ***config.py***, It helps to get almost uniform distribution of steering angles.

## 4. Network Architecture

Network architecture is borrowed from the aforementioned NVIDIA paper in which they tackle the same problem of steering angle prediction, just in a slightly more unconstrained environment.



The architecture is *relatively shallow* and is shown above.

```
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         (None, 66, 200, 3)        0

lambda_1 (Lambda)            (None, 66, 200, 3)        0

conv2d_1 (Conv2D)            (None, 31, 98, 24)        1824

elu_1 (ELU)                  (None, 31, 98, 24)        0

dropout_1 (Dropout)          (None, 31, 98, 24)        0

conv2d_2 (Conv2D)            (None, 14, 47, 36)        21636

elu_2 (ELU)                  (None, 14, 47, 36)        0

dropout_2 (Dropout)          (None, 14, 47, 36)        0

conv2d_3 (Conv2D)            (None, 5, 22, 48)         43248

elu_3 (ELU)                  (None, 5, 22, 48)         0

dropout_3 (Dropout)          (None, 5, 22, 48)         0

conv2d_4 (Conv2D)            (None, 3, 20, 64)         27712

elu_4 (ELU)                  (None, 3, 20, 64)         0

dropout_4 (Dropout)          (None, 3, 20, 64)         0

conv2d_5 (Conv2D)            (None, 1, 18, 64)         36928

elu_5 (ELU)                  (None, 1, 18, 64)         0

dropout_5 (Dropout)          (None, 1, 18, 64)         0

flatten_1 (Flatten)          (None, 1152)              0

dense_1 (Dense)              (None, 100)               115300

elu_6 (ELU)                  (None, 100)               0

dropout_6 (Dropout)          (None, 100)               0
```

```
dense_2 (Dense)              (None, 50)            5050
_____________________________________________________
elu_7 (ELU)                  (None, 50)            0
_____________________________________________________
dropout_7 (Dropout)           (None, 50)            0
_____________________________________________________
dense_3 (Dense)              (None, 10)            510
_____________________________________________________
elu_8 (ELU)                  (None, 10)            0
_____________________________________________________
dense_4 (Dense)              (None, 1)            11
=====================================================
Total params: 252,219
Trainable params: 252,219
Non-trainable params: 0
```

Input normalization is implemented through a Lambda layer, which constitutes the first layer of the model. In this way input is standardized such that lie in the range [-1, 1]: of course this works as long as the frame fed to the network is in range [0, 255].

The choice of ELU activation function (instead of more traditional ReLU) come from this model of CommaAI, which is born for the same task of steering regression. On the contrary, the NVIDIA paper does not explicitly state which activation function they use.

Convolutional layers are followed by 3 fully-connected layers: finally, a last single neuron tries to regress the correct steering value from the features it receives from the previous layers. The model is defined in ***model.py***.

## 5. Preventing Overfitting

Despite the strong data augmentation mentioned above, there's still room for the major nightmare of the data scientist, a.k.a. overfitting. In order to prevent the network from falling in love with the training track, dropout layers are aggressively added after each convolutional layer (*drop prob=0.2*) and after each fully-connected layer (*drop prob=0.5*) but the last one.

## 6. Training Details

Model was compiled using ***Adam optimizer with default parameters*** (like learning_rate, beta etc.) and mean squared error loss w.r.t. the ground truth steering angle. Training took a couple of minutes in  Udacity's workspace. During the training bias parameter was set to 0.8, frame brightness (V channel) was augmented in the range [0.2, 1.5] with respect to the original one. Normal distributed noise added to the steering angle had parameters *mean=0*, *std=0.2*. Frame flipping was random with probability *0.5*.

## 7. Testing The Model

The model was tested in track 1. The model performed decent in this track.
 I have uploaded the video "**video.mp4**" created from center camera images In autonomous mode. I have updated the file ***drive.py***  to preprocess the input images and feed them to the network.

I have also uploaded a  "**screen_Recorder_track_1.mp4**" which is recorded through A screen recorder and the network driving in simulator along with output **(steering_angel, throttle)**  in another window side-by-side.

## Discussion

In my opinion, these were the two main challenges in this project:

1. Skew distribution of training data (strong bias towards 0)
2. Relatively few training data, in one track only (risk of overfitting)

Both these challenges has been solved, or at least mitigated, using aggressive data augmentation and dropout. The main drawback I notice is that now the network has some difficulties in going just straight: it tends to steer a little too much even when no steering at all is needed. Beside this aspect, the network is able to safely drive on both tracks, never leaving the drivable portion of the track surface.

There's still a lot of room for future improvements. Among these, would be interesting to predict the car *throttle* along with the steering angle. In this way the car would be able to maintain its speed constant even on the second track which is plenty of hills. Furthermore, at the current state enhancing the graphic quality of the simulator leads to worse results, as the network is not able to handle graphic details such as *e.g.* shadows. Collecting data from the simulator set with better graphic quality along with the appropriate data augmentation would likely mitigate this problem.