ADA PROJECT FINAL REPORT

SELF-DRIVING CAR USING UDACITY'S CAR SIMULATOR ENVIRONMENT AND TRAINED BY CONVOLUTIONAL NEURAL NETWORKS

Deep Learning for Behavioral Cloning.

Submitted by
Jai Chaudhry (2K18/SE/069)
Ishaan Jain (2K18/SE/067)

Under the supervision of

DR. RAHUL GUPTA
Professor
Department of Computer Science and Engineering
Delhi Technological University

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

DELHI TECHNOLOGICAL UNIVERSITY

(FORMERLY DELHI COLLEGE OF ENGINEERING)
SHAHABAD DAULATPUR, BAWANA ROAD,
DELHI – 110042

November 20, 2020

Abstract

Self-driving cars have become a trending domainbecause of significant enhancement in processing power of computers and performance and training of Machine and Deep Learning Algorithms. The project purpose is to train a convolutional neural network to drive acar agent on the track of Udacity's Simulator environment. Udacity had released the simulator about around 2 years ago as an open source software for learning purpose. Driving a car in an autonomous manner requires learning to control steering angle, throttle and brakes. Behavioral cloning technique is used to mimic human driving behavior using the data collected in the training mode of the simulator. After training on this data, the Deep Learning model then drives the car in autonomous mode.

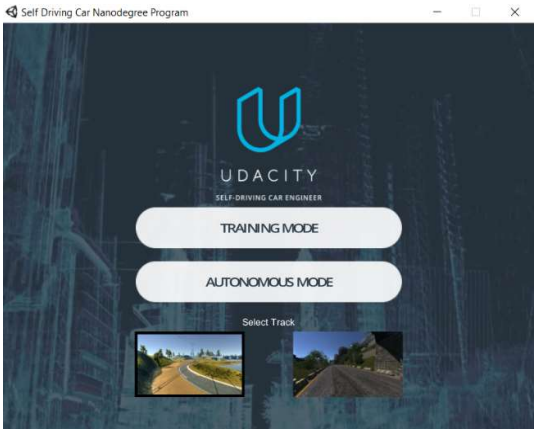
Table of Contents

1. Introduction

1.1 Problem Definition

The challenge is to mimic the driving behavior of a human on the simulator with the help of a model trained by deep neural networks. The concept is called Behavioral Cloning, to mimic how humans the drive car on the simulator. The simulator contains two tracks and two modes, namely, training mode and autonomous mode. The dataset is generated from the simulator by the user, driving the car in training mode. This is followed by testing on the track, seeing how the deep learning model performs after being trained by that user data.

Simulator Download link: https://github.com/udacity/self-driving-car-sim



About Dataset:

In training mode, a dataset in produced in a certain specified folder which contains images of the front view of track as seen by car and a csv file driving_log.csv as shown in below image which contains the image paths along with the value of steering angle, brake, throttle and speed of car in simulator at that moment.

	center	left	right	steering	throttle	break	speed
0	Desktop\track1data\IMG\center_2019_04_02_19_25...	Desktop\track1data\IMG\left_2019_04_02_19_25_3...	Desktop\track1data\IMG\right_2019_04_02_19_25_...	0.0	0.000000	0	0.000007
1	Desktop\track1data\IMG\center_2019_04_02_19_25...	Desktop\track1data\IMG\left_2019_04_02_19_25_3...	Desktop\track1data\IMG\right_2019_04_02_19_25_...	0.0	0.000000	0	0.000003
2	Desktop\track1data\IMG\center_2019_04_02_19_25...	Desktop\track1data\IMG\left_2019_04_02_19_25_3...	Desktop\track1data\IMG\right_2019_04_02_19_25_...	0.0	0.048016	0	0.002267
3	Desktop\track1data\IMG\center_2019_04_02_19_25...	Desktop\track1data\IMG\left_2019_04_02_19_25_3...	Desktop\track1data\IMG\right_2019_04_02_19_25_...	0.0	0.281203	0	0.175589

1.2 Solution Approach

The high-level architecture of the implementation can be seen in Figure (…)

The problem is solved in the following steps:

• The simulator is used to collect data by driving the car in the training model keyboard, providing the so called “good-driving” behaviourinput data in form of a driving_log (.csv file) and a set of images. The simulator acts as a server and pipes these images and data log to the Python client.

• The client (Python program) is the machine learning model built using Deep NeuralNetworks. These models are developed on Keras (a high-level API over TensorFlow). Keras provides sequential models to build a linear stack of network layers. Such models are used in the project to train over the datasets as the second step. Detailed description of CNN models can be seen in further sections of this report.

• Once the model is trained, it outputs steering angle when an image is given as input and this angle is sent to Udacity simulator server.

• These modules, or inputs, are piped back to the server and are used to drive the car autonomously in the simulator and keep it from falling off the track.

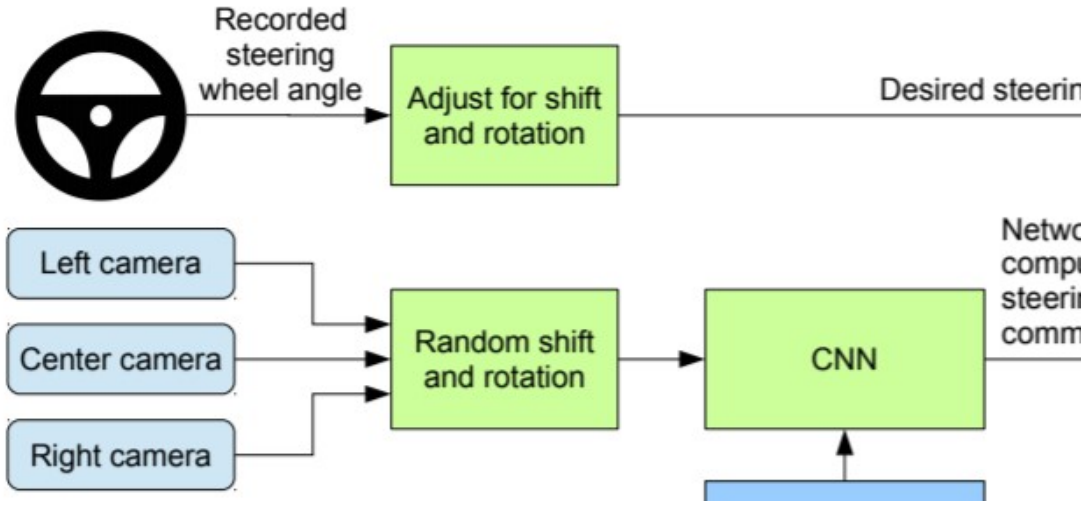


Image Source: https://arxiv.org/pdf/1604.07316v1.pdf

2. Tools and Technology used:

The various technologies that are used in the implementation of this project are described in this section.

TensorFlow: This is an open-source library for implementation of deep neural networks. It is widely used for machine learning applications. It is also used as both a math library and for large computations. For this project, Keras, which is a high-level API on TensorFlow was used. Keras facilitates in building the models easily as it more user friendly with respect to code.
Different libraries are available in Python that help in machine learning projects.

Several of those libraries have improved the performance of this project. Few of them are mentionedbelow: -

1) **Numpy:** provides with high-level math function collection to support multi-dimensional metrices and arrays. This is used for faster computations over the weights (gradients) in neural networks.
2) **Matplotlib:**which is a machine learning library for Python which features different plotting and visualization techniques.
3) **OpenCV:** (Open Source Computer Vision Library) which is designed for image preprocessing and augmentation techniques.
4) **Pandas:** used to read csv file.

The machine on which this project was built, is a personal computer with following configuration:

• Processor: Intel(R) Core i5-8300H @ 2.3GHz

• RAM: 8GB

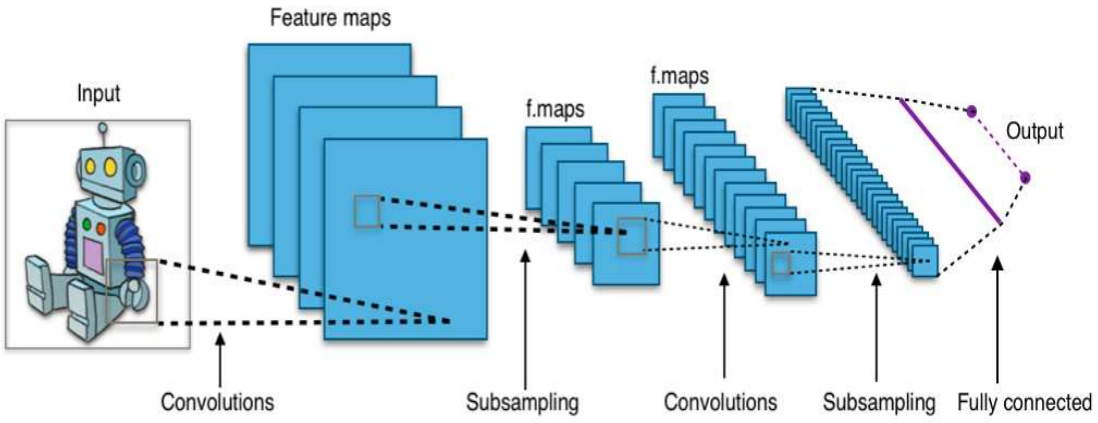
• System: 64bit OS, x64 processor

3. Explanation of different modules with snapshots

3.1 About the chosen Neural Network CNN

CNN is a type of feed-forward neural network computing system that can be used to learn from input data. Learning is accomplished by determining a set of weights or filter values that allow the network to model the behavior according to the training data. The desired output and the output generated by CNN initialized with random weights will be different. This difference (generated error) is backpropagated through the layers of CNN to adjust the weights of the neurons, which in turn reduces the error and allows us to produce output closer to the desired one.

CNN is good at capturing hierarchical and spatial data from images. It utilizes filters that look at regions of an input image with a defined window size and map it to some output. It then slides the window by some defined stride to other regions, covering the whole image. Each convolution filter layer thus captures the properties of this input image hierarchically in a series of subsequent layers, capturing the details like lines in image, then shapes, then whole objects in later layers. CNN can be a good fit to feed the images of a dataset and classify them into their respective classes.



3.2 EXPERIMENTAL CONFIGURATIONS

This section consists of the configurations used to set up the model for training the Python Client to provide the Neural Network outputs that drive the car on the simulator.

The tweaking of parameters and rigorous experiments were tried to reach the best combination. Though each of the models had their unique behaviors and differed in their performance with each tweak, the following combination of configuration can be considered as the optimal:

- The sequential models built on Keras with deep neural network layers are used to train the data.
- Models are only trained using the dataset from Track_1.
- 90% of the dataset is used for training, 10% is used for validation.
- Epochs = 60, i.e. number of iterations or passes through the complete dataset. Experimented with larger number of epochs also, but the model tried to "overfit". In other words, the model learns the details in the training data too well, thus impacting the performance during testing.
- Batch-size = 32, i.e. number of image samples sent to the model during training, like a subset of data as complete dataset is too big to be passed all at once.
- Learning rate's = 0.0001, 0.00005, 0.000007, i.e. how the coefficients of the weights or gradients change in the network.
- ModelCheckpoint() is the function provided in Keras to save checkpoints and to save the best epoch according to the validation loss. There are different combinations of Convolution layer, Flatten, Dropout, Dense and so on, that can be used to implement the Neural Network models. Out of the various different architectures that were tried, the best one is discussed in further sections of this report.

3.3 NETWORK ARCHITECTURE

Numerous combinations of architectures tried for predicting the steering angle, that is input for the car to drive in autonomous mode. Neural Network layers were organized in series and various combinations of Time-Distributed Convolution layers, Flatten, Dropout, Dense and so on are used in architectures. But the final architecture doesn't include any Dropout Layer.

```
model.add(Conv2D(48, kernel_size=(4,4), activation='relu',st
model.add(Conv2D(64, kernel_size=(3,3), activation='relu'))
model.add(Conv2D(64, kernel_size=(3,3), activation='relu'))
model.add( Flatten() )
model.add( Dense(100, activation = 'relu') )
model.add( Dense(50, activation = 'relu') )
model.add( Dense(10,activation='relu') )
```

The model takes in the image (after pre-processing), with input shape as (66,200,3).

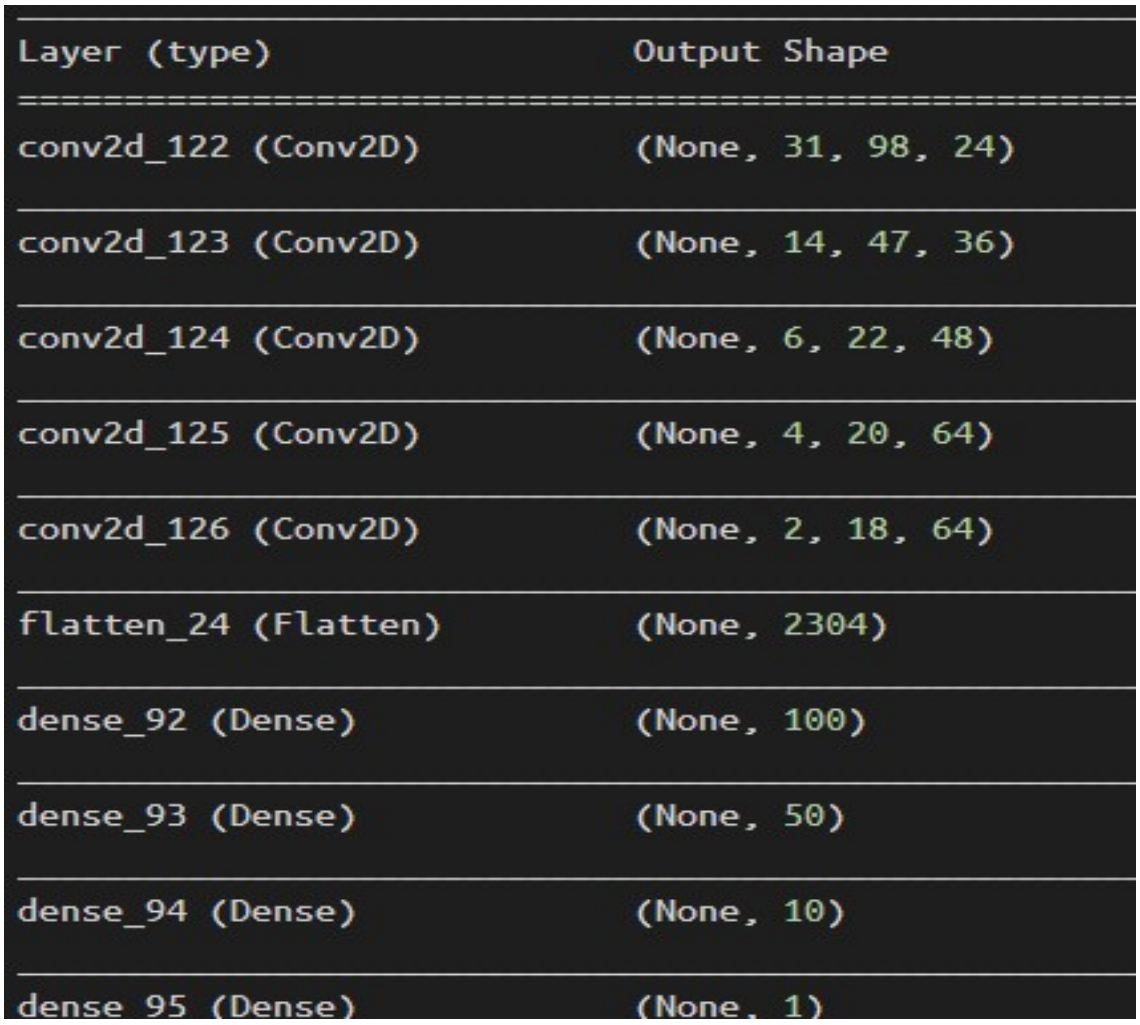
Then convolutions are applied on the image using kernels with dimensions 4 x 4. There are 44 such kernels in the first layer and this leads to the creation of 44 filters. Also, the stride is kept as (2,2) instead of the default (1,1) to ensure faster reduction in dimensions of the final feature layer before Flattening to reduce the total number of parameters.

Then again in second and third layer kernel convolution is done but this time their shape is (3x3) and the number of filters chosen is 64. (since an even power of 2 generally results in good feature learning in Deep Learning Area).

Then the whole matrix produced after CNN layers is Flattened. This matrix contains the feature values.

Then Dense Layers are added with reduction in the size in power of 2 starting from

100 → 50 → 10 → 1.



Layer (type)	Output Shape
conv2d_122 (Conv2D)	(None, 31, 98, 24)
conv2d_123 (Conv2D)	(None, 14, 47, 36)
conv2d_124 (Conv2D)	(None, 6, 22, 48)
conv2d_125 (Conv2D)	(None, 4, 20, 64)
conv2d_126 (Conv2D)	(None, 2, 18, 64)
flatten_24 (Flatten)	(None, 2304)
dense_92 (Dense)	(None, 100)
dense_93 (Dense)	(None, 50)
dense_94 (Dense)	(None, 10)
dense_95 (Dense)	(None, 1)

Network Architecture

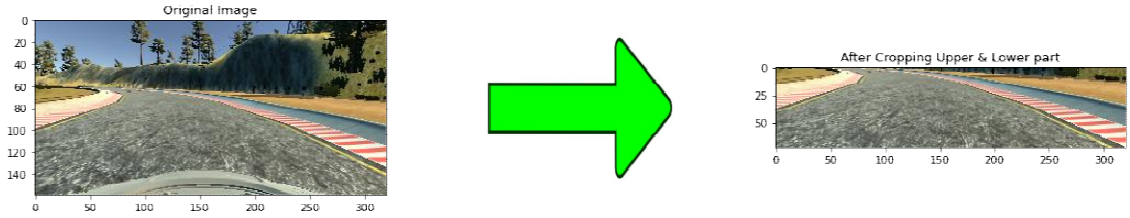
4. Implementation code:

4.1 Data Augmentation using Image Processing:

- Crop

The images in the dataset have relevant features in the lower part where the road is visible. The external environment above a certain image portion will never be used to determine the output and thus can be cropped. Approximately, 40% of the top portion of the image is cut and passed in the training set. The snippet of code and transformation of an image after cropping and resizing it to original image can be seen inbelow images.

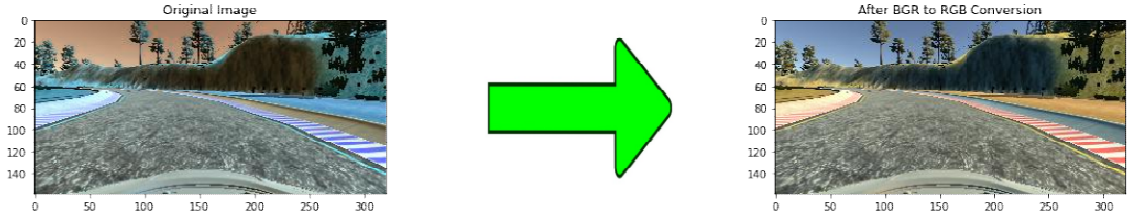
```
def crop_image(img):
    return img[62:-25,:]
```



- Read image (BGR to RGB)

To load the images from disk into a variable and change the channel configuration from BGR (default of OpenCV) to RGB.

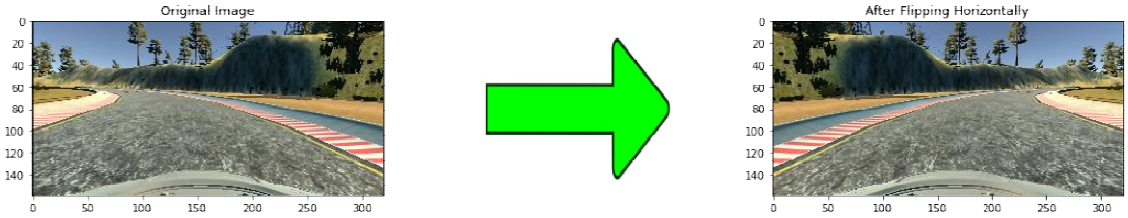
```
def read_image(path):
    img = cv.imread(path)
    img = cv.cvtColor(img,cv.COLOR_BGR2RGB)
    return np.array(img)
```



- Flip (horizontal)

The image is flipped horizontally (i.e. a mirror image of the original image is passed to the dataset). The motive behind this is that the model gets trained for similar kinds of turns on opposite sides too. This is important because Track_1 includes mostly left turns. The snippet of code and transformation of an image after flipping it can be seen in below images.

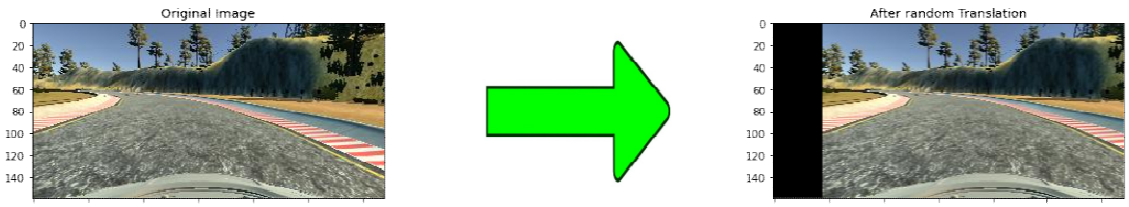
```
def flip_image(img,angle):
    flipp = np.random.choice(2)
    if flipp==1:
        img = cv.flip(img,1)
        angle = -angle

    return img,angle
```



- Translation (horizontal and vertical)

The image is shifted by a small amount both horizontally and vertically.
The snippet of code and transformation of an image after translation can be seen inbelow images.

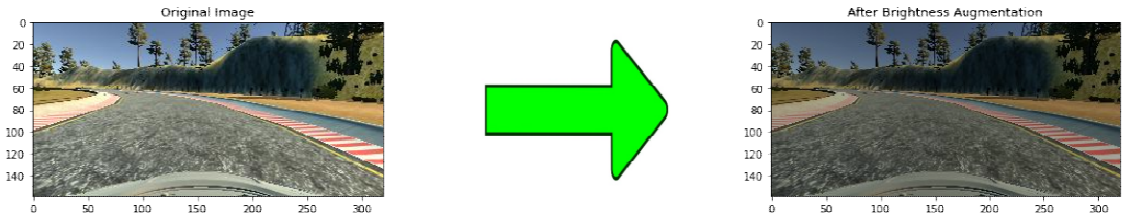
```
def trans_image(image, steer, trans_range=100):
    # Translation
    tr_x = trans_range * np.random.uniform() - trans_range / 2
    steer_ang = steer + tr_x / trans_range * 2 * .2
    tr_y = 0
    Trans_M = np.float32([[1, 0, tr_x], [0, 1, tr_y]])
    col, row = image.shape[:2]
    image_tr = cv.warpAffine(image, Trans_M, (row, col))

    return image_tr, steer_ang
```



- Brightness

To generalize the lowlight conditions, the brightness augmentation proved to be very useful. The snippet of code and transformation of an image after Brightness change can be seen in below images.

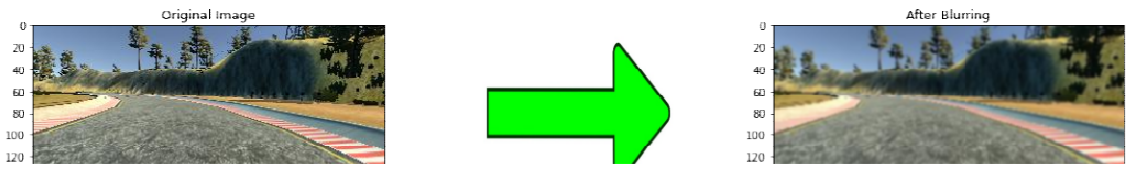
```
def change_brightness(image):
    image1 = cv.cvtColor(image, cv.COLOR_RGB2HSV)
    image1 = np.array(image1, dtype=np.float64)
    random_bright = .5 + np.random.uniform()
    image1[:, :, 2] = image1[:, :, 2] * random_bright
    image1[:, :, 2][image1[:, :, 2] > 255] = 255
    image1 = np.array(image1, dtype=np.uint8)
    image1 = cv.cvtColor(image1, cv.COLOR_HSV2RGB)
    return image1
```



- Random blur

To take care of the distortion effect in the camera while capturing the images, this augmentation is used as an image captured is not clear every time. Sometimes, thecamera goes out of focus, but the car still needs to fit that condition and keep the car steady. This random blur augmentation can take such scenarios into consideration. The code snippet and the transformation can be seen in below images.

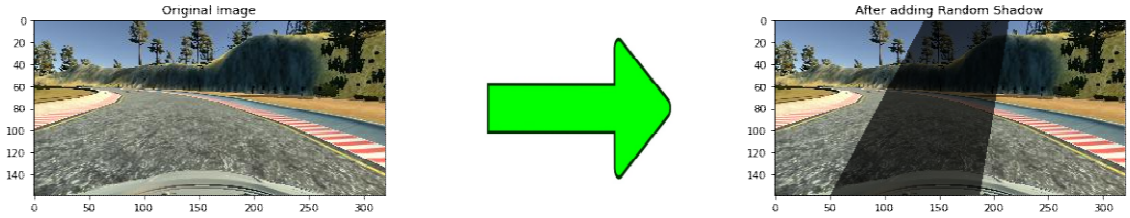
```
def blur_image(img, f_size=5):
    """
    Applies Gaussir Blur to smoothen the image.
    This in effect performs anti-aliasing on the provided image
    """
    img = cv.GaussianBlur(img,(f_size, f_size),0)
    img = np.clip(img, 0, 255)
    return img.astype(np.uint8)
```



- Random Shadows

Even after taking into considerations the light conditions, there are still chances that there are shadows on the road. This will give an instance of half lit and half lowlight scenes in the image. To cast random shadows and solve this shadow fitting problem, this augmentation is applied on the dataset. The snippet of code and transformation of an image after adding random shadow can be seen in below images.

```
def add_random_shadow(img, w_low=0.6, w_high=0.85):
    """
    Overlays supplied image with a random shadow polygon
    The weight range (i.e. darkness) of the shadow can be configured via the interval [w_low, w_high)
    """
    cols, rows = (img.shape[0], img.shape[1])

    top_y = np.random.random_sample() * rows
    bottom_y = np.random.random_sample() * rows
    bottom_y_right = bottom_y + np.random.random_sample() * (rows - bottom_y)
    top_y_right = top_y + np.random.random_sample() * (rows - top_y)
    if np.random.random_sample() <= 0.5:
        bottom_y_right = bottom_y - np.random.random_sample() * (bottom_y)
        top_y_right = top_y - np.random.random_sample() * (top_y)

    poly = np.asarray([[ [top_y,0], [bottom_y, cols], [bottom_y_right, cols], [top_y_right,0]]], dtype=np.int32)

    mask_weight = np.random.uniform(w_low, w_high)
    origin_weight = 1 - mask_weight

    mask = np.copy(img).astype(np.int32)
    cv.fillPoly(mask, poly, (0, 0, 0))

    return cv.addWeighted(img.astype(np.int32), origin_weight, mask, mask_weight, 0).astype(np.uint8)
```



4.2 Generate Dataset function :

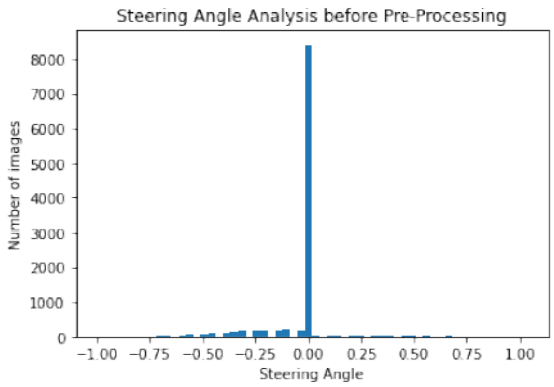
Loads image data according to the specified parameter size.

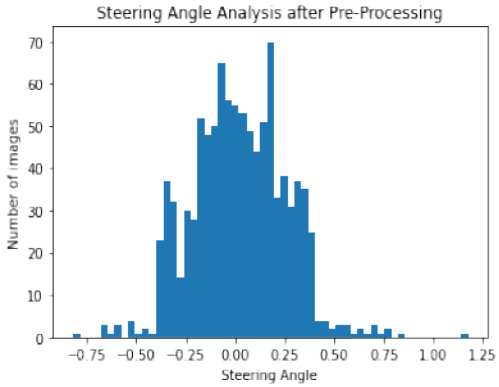
Uses random generator since we have 3 images for each time frame of the simulator car when it was running in training mode.

Then data augmentation is done first by applying basic processing then advanced processing.

```
def generate_dataset(size):

    # read csv file column values
    df = pd.read_csv(base_path + "driving_log.csv")
    df.columns = header

    # GET THE PATHS TO IMAGES IN OUR DIRECTORY
    img_paths = list(df[['left','right','center']].values)
    for i in range(0,len(img_paths)):
        for j in range(0,3):
            img_paths[i][j] = img_paths[i][j].split("IMG")


    # GET THE OUTPUT STEERING ANGLES VALUES
    output = np.array(df['steering'].values)


    data_X = np.empty((size,66,200,3))
    data_Y = np.empty((size))

    for i in range(0,size):
        angle = output[i]
        random_index = np.random.choice(3)
        img = read_image(base_path+"IMG/"+img_paths[i][ran

        if random_index == 0:
            angle += 0.2
        elif random_index == 1:
            angle -= 0.2

        img = crop_image(img)
        img = resize_image(img)
        img , angle = flip_image(img,angle)

        img,angle = trans_image(img,angle)
```

Because of the numerous processing functions the initial steering angles get transformed to a widespread area thus allowing the training of a generalized model.





4.3 Checkpoint function:

Made to store the best trained model while training with numerous epochs. Best models are stored on the basis of their loss and validation loss during training.

```
val_checkpoint = ModelCheckpoint('./models/model_Val_Loss-{val_loss:.4f}.h5',
                                 monitor = 'val_loss',
                                 save_best_only = True,
                                 mode = 'auto' )

loss_checkpoint = ModelCheckpoint('./models/model_Loss-{loss:.4f}.h5',
                                 monitor = 'loss',
                                 save_best_only = True,
                                 mode = 'auto'
                                 )
```

4.4 Model Building Function:

This was made to speed up the process of manual tuning of model with various tryouts of different-different processing and augmentation techniques.

```
def get_model():
    model = Sequential()

# ------------------------------------------------------------------------------
    model.add(Conv2D(24, kernel_size=(5,5), activation='relu',input_shape = IMG_SHA
    model.add(Conv2D(36, kernel_size=(5,5), activation='relu', strides=(2, 2)))
    model.add(Conv2D(48, kernel_size=(4,4), activation='relu',strides=(2, 2)))
    model.add(Conv2D(64, kernel_size=(3,3), activation='relu'))
    model.add(Conv2D(64, kernel_size=(3,3), activation='relu'))
# ------------------------------------------------------------------------------

    model.add( Flatten() )
    model.add( Dense(100, activation = 'relu') )
    model.add( Dense(50, activation = 'relu') )
    model.add( Dense(10,activation='relu') )
    model.add( Dense(1) )
    model.summary()
```

4.5 Model Training Code:

The model is trained using this code snippet. X_train is the input images, y_train is the output steering angles corresponding to the images in X_train.

Validation is performed on X_test, and y_test data which is 10% of the total data taken into consideration. Validation is done to ensure the model trains in a right way and to catch any error in training or preprocessing before testing it on simulator.

Then there are callbacks which are functions that are called upon finish of one epoch to save the best model.

Shuffle = True ensures that the data that is taken from X_train and y_train for model training is randomly chosen.

```
from keras import backend as K

K.set_value(model.optimizer.learning_rate, 0.000
model.fit( X_train, y_train,
           batch_size=32, epochs=20,
           validation_data=(X_test, y_test),
           callbacks=[val_checkpoint,loss_checkp
```

5. Results:

Full track autonomous movement of the car in the simulator was successfully accomplished after a lot of hit and trials with manual tuning of hyper-parameters of the model and the pre-processing functions along with changing the size of the dataset taken for training.

Data Augmentation proved to be extremely helpful because most of the training data was centered around 0 value with respect to steering angle.

Simulator Download link: https://github.com/udacity/self-driving-car-sim

Video link: https://drive.google.com/file/d/1rGX95tF-3ixq8wO0nBBdcbLhtv2EGs98/view?usp=sharing

GitHub Link : https://github.com/jaichaudhry323/Udacity_Self_Driving_Simulation_Project

6. Conclusions:

Data Pre-processing is an essential step in in Deep Learning.

Data Augmentation is necessary when the dataset is too biased.

A low loss or validation loss doesn't mean that the model is correct. It can be verified only by practically employing it to the task for which it was built.

Too much processing is harmful as it tends to distort the input data and hence the model output on the actual data is no longer accurate.

Even a small mistake like forgetting BGR to RGB conversion can lead to zero performance of the model on the simulator and hence everything should be checked thoroughly.

CNN performs well when at least 3-4 layers of Time Distributed layers are employed along with Dense Layers for feature mapping to output.

Deep Learning Models don't learn if the number of parameters is way too large as compared to the approximate number of features.

A CNN with big architecture and small number of parameters is the best choice as it trains faster and better than others with higher parameters or smaller architecture.

7. Bibliography

1. https://arxiv.org/pdf/1604.07316v1.pdf

2. Du, S., Guo, H., & Simpson, A. (2017). Self-driving car steering angle prediction based on image recognition.
3. Department of Computer Science, Stanford University, Tech. Rep. CS231–626.

4. https://github.com/udacity/CarND-Behavioral-Cloning-P3/blob/master/drive.py

5. https://docs.opencv.org/master/d9/df8/tutorial_root.html

6. http://images.nvidia.com/content/tegra/automotive/images/2016/solutions/pdf/end-to-end-dl-using-px.pdf