<u>Behavioural Cloning – Udacity Self-driving Car Simulator</u>

This document describes the process of **collecting image data** from Udacity Self-driving Car Simulator, **processing the images**, developing and **training a CNN model** that is capable of return an appropriate steering angle based on the given image, as well as **passing the model to the self-driving car simulator** to see how well the model clones our own driving behaviour.

First, to collect data of our own driving behaviour, we run the training mode in the simulator. This brings us to this page where we can start to record our own driving behaviour. Each frame of the drive will be captured and returned as the following output:

1. Front Left camera image
2. Front Centre camera image
3. Front Right camera image
4. Front Left camera image name
5. Front Centre camera image name
6. Front Right camera image name
7. Steering
8. Throttle
9. Reverse
10. Speed

Front Left camera image

Front Centre camera image

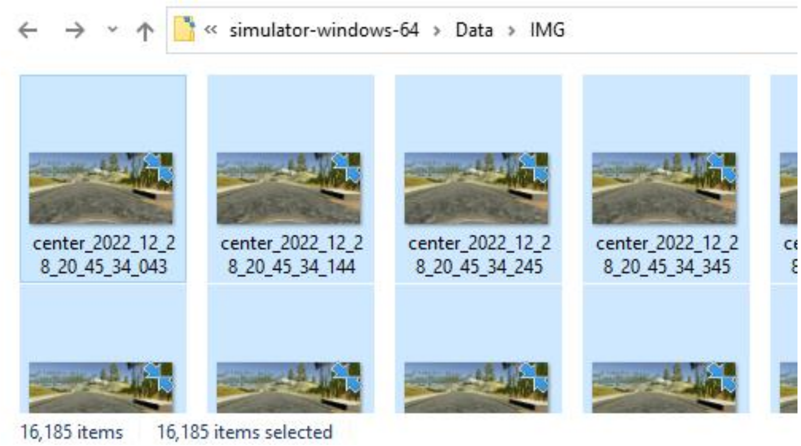
Front Right camera image

Driving log containing S/N 4 to 10

	A	B	C	D	E	F	G
1	C:\Users\benja\Desktop\data visualisatio	C:\Users\benja\Desktop\data visualisatio	C:\Users\benja\Desktop\data visualisatio	0	0	0	0.657895
2	C:\Users\benja\Desktop\data visualisatio	C:\Users\benja\Desktop\data visualisatio	C:\Users\benja\Desktop\data visualisatio	0	0	0	0.651296
3	C:\Users\benja\Desktop\data visualisatio	C:\Users\benja\Desktop\data visualisatio	C:\Users\benja\Desktop\data visualisatio	0	0	0	0.644763
4	C:\Users\benja\Desktop\data visualisatio	C:\Users\benja\Desktop\data visualisatio	C:\Users\benja\Desktop\data visualisatio	0	0	0	0.63701
5	C:\Users\benja\Desktop\data visualisatio	C:\Users\benja\Desktop\data visualisatio	C:\Users\benja\Desktop\data visualisatio	0	0	0	0.63062
6	C:\Users\benja\Desktop\data visualisatio	C:\Users\benja\Desktop\data visualisatio	C:\Users\benja\Desktop\data visualisatio	0	0	0	0.624294
7	C:\Users\benja\Desktop\data visualisatio	C:\Users\benja\Desktop\data visualisatio	C:\Users\benja\Desktop\data visualisatio	0	0	0	0.618032
8	C:\Users\benja\Desktop\data visualisatio	C:\Users\benja\Desktop\data visualisatio	C:\Users\benja\Desktop\data visualisatio	0	0	0	0.611833

As the track is oval in shape, I have to make sure I drive in both directions in order to balance out the number of left curves and right curves. To collect sufficient data, I drove about **3 laps in each direction**.

At the end of the recording, I collected **16,185 images**.



To upload this big set of data to train our model, we can make use of github and google colab. First, let's **upload the Data folder into github repository**. To do that, we run the following in command prompt.

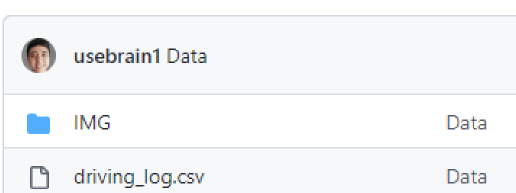
```
C:\Users\benja>cd Desktop\data visualisation\Udemy Course\Python Course\Machine Learning\Self-Driving Car Applied Deep Learning\simulator-windows-64\Data

C:\Users\benja\Desktop\data visualisation\Udemy Course\Python Course\Machine Learning\Self-Driving Car Applied Deep Learning\simulator-windows-64\Data>git init
Reinitialized existing Git repository in C:/Users/benja/Desktop/data visualisation/Udemy Course/Python Course/Machine Learning/Self-Driving Car Applied Deep Learning/simulator-windows-64/Data/.git/

C:\Users\benja\Desktop\data visualisation\Udemy Course\Python Course\Machine Learning\Self-Driving Car Applied Deep Learning\simulator-windows-64\Data>git add .
```

```
C:\Users\benja\Desktop\data visualisation\Udemy Course\Python Course\Machine Learning\Self-Driving Car Applied Deep Learning\simulator-windows-64\Data>git commit -m "Data"
```

```
C:\Users\benja\Desktop\data visualisation\Udemy Course\Python Course\Machine Learning\Self-Driving Car Applied Deep Learning\simulator-windows-64\Data>git remote add origin https://github.com/usebrain1/Track.git
```

```
C:\Users\benja\Desktop\data visualisation\Udemy Course\Python Course\Machine Learning\Self-Driving Car Applied Deep Learning\simulator-windows-64\Data>git push origin master
```

Now that the data has been uploaded to our github repository, we can **clone it** into google drive to be used in google colab.



```
!git clone https://github.com/usebrain1/Track
```

```
Cloning into 'Track'...
remote: Enumerating objects: 16189, done.
remote: Total 16189 (delta 0), reused 0 (delta 0), pack-reused 16189
Receiving objects: 100% (16189/16189), 204.72 MiB | 15.61 MiB/s, done.
Checking out files: 100% (16186/16186), done.
```

That concludes the collection and uploading of data. Now it is time to process the images and to develop and train our model. First, we need to import the following modules.

```python
import os
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
import keras
from keras.models import Sequential
from keras.optimizers import Adam
from keras.layers import Conv2D, MaxPooling2D, Dropout, Flatten, Dense
from sklearn.utils import shuffle
from sklearn.model_selection import train_test_split
from imgaug import augmenters as iaa
import cv2
import pandas as pd
import ntpath
import random
```

Next, we use Pandas to read the driving_log file and save the dataframe as "data".

```python
datadir = 'Track'
columns = ['center', 'left', 'right', 'steering', 'throttle', 'reverse', 'speed']
data = pd.read_csv(os.path.join(datadir, 'driving_log.csv'), names = columns)
pd.set_option('display.max_colwidth', -1)
data.head()
```

```
<ipython-input-5-d963de388c8a>:4: FutureWarning: Passing a negative integer is deprecated in version 1.0 and will not be supported in future version. Instead, use None to not limit the column width.
  pd.set_option('display.max_colwidth', -1)
```

	center	left	right	steering	throttle	reverse	speed
0	C:\Users\benja\Desktop\data visualisation\Udemy Course\Python Course\Machine Learning\Self-Driving Car Applied Deep Learning\simulator-windows-64\Data\IMG\center_2022_12_28_20_45_34_043.jpg	C:\Users\benja\Desktop\data visualisation\Udemy Course\Python Course\Machine Learning\Self-Driving Car Applied Deep Learning\simulator-windows-64\Data\IMG\left_2022_12_28_20_45_34_043.jpg	C:\Users\benja\Desktop\data visualisation\Udemy Course\Python Course\Machine Learning\Self-Driving Car Applied Deep Learning\simulator-windows-64\Data\IMG\right_2022_12_28_20_45_34_043.jpg	0.0	0.0	0	0.657895
1	C:\Users\benja\Desktop\data visualisation\Udemy Course\Python Course\Machine Learning\Self-Driving Car Applied Deep Learning\simulator-windows-64\Data\IMG\center_2022_12_28_20_45_34_144.jpg	C:\Users\benja\Desktop\data visualisation\Udemy Course\Python Course\Machine Learning\Self-Driving Car Applied Deep Learning\simulator-windows-64\Data\IMG\left_2022_12_28_20_45_34_144.jpg	C:\Users\benja\Desktop\data visualisation\Udemy Course\Python Course\Machine Learning\Self-Driving Car Applied Deep Learning\simulator-windows-64\Data\IMG\right_2022_12_28_20_45_34_144.jpg	0.0	0.0	0	0.651296
2	C:\Users\benja\Desktop\data visualisation\Udemy Course\Python Course\Machine Learning\Self-Driving Car Applied Deep Learning\simulator-windows-64\Data\IMG\center_2022_12_28_20_45_34_245.jpg	C:\Users\benja\Desktop\data visualisation\Udemy Course\Python Course\Machine Learning\Self-Driving Car Applied Deep Learning\simulator-windows-64\Data\IMG\left_2022_12_28_20_45_34_245.jpg	C:\Users\benja\Desktop\data visualisation\Udemy Course\Python Course\Machine Learning\Self-Driving Car Applied Deep Learning\simulator-windows-64\Data\IMG\right_2022_12_28_20_45_34_245.jpg	0.0	0.0	0	0.644763
3	C:\Users\benja\Desktop\data visualisation\Udemy Course\Python Course\Machine Learning\Self-Driving Car Applied Deep Learning\simulator-windows-64\Data\IMG\center_2022_12_28_20_45_34_345.jpg	C:\Users\benja\Desktop\data visualisation\Udemy Course\Python Course\Machine Learning\Self-Driving Car Applied Deep Learning\simulator-windows-64\Data\IMG\left_2022_12_28_20_45_34_345.jpg	C:\Users\benja\Desktop\data visualisation\Udemy Course\Python Course\Machine Learning\Self-Driving Car Applied Deep Learning\simulator-windows-64\Data\IMG\right_2022_12_28_20_45_34_345.jpg	0.0	0.0	0	0.637010
4	C:\Users\benja\Desktop\data visualisation\Udemy Course\Python Course\Machine Learning\Self-Driving Car Applied Deep Learning\simulator-windows-64\Data\IMG\center_2022_12_28_20_45_34_447.jpg	C:\Users\benja\Desktop\data visualisation\Udemy Course\Python Course\Machine Learning\Self-Driving Car Applied Deep Learning\simulator-windows-64\Data\IMG\left_2022_12_28_20_45_34_447.jpg	C:\Users\benja\Desktop\data visualisation\Udemy Course\Python Course\Machine Learning\Self-Driving Car Applied Deep Learning\simulator-windows-64\Data\IMG\right_2022_12_28_20_45_34_447.jpg	0.0	0.0	0	0.630620

We can make use of the ntpath.split() method to reduce the long image path name and return just the image name.

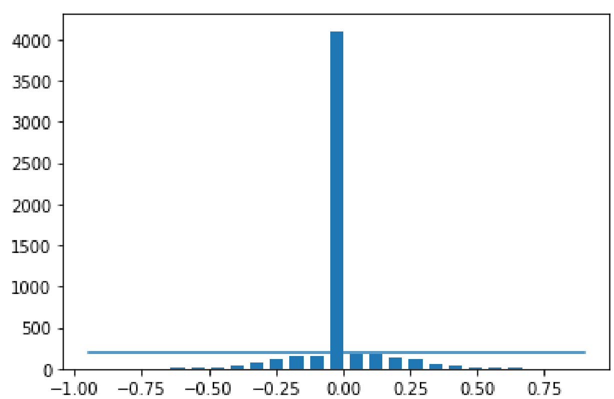
```
def path_leaf(path):
  head, tail = ntpath.split(path)
  return tail

data['center'] = data['center'].apply(path_leaf)
data['left'] = data['left'].apply(path_leaf)
data['right'] = data['right'].apply(path_leaf)

data.head()
```

	center	left	right	steering	throttle	reverse	speed
0	center_2022_12_28_20_45_34_043.jpg	left_2022_12_28_20_45_34_043.jpg	right_2022_12_28_20_45_34_043.jpg	0.0	0.0	0	0.657895
1	center_2022_12_28_20_45_34_144.jpg	left_2022_12_28_20_45_34_144.jpg	right_2022_12_28_20_45_34_144.jpg	0.0	0.0	0	0.651296
2	center_2022_12_28_20_45_34_245.jpg	left_2022_12_28_20_45_34_245.jpg	right_2022_12_28_20_45_34_245.jpg	0.0	0.0	0	0.644763
3	center_2022_12_28_20_45_34_345.jpg	left_2022_12_28_20_45_34_345.jpg	right_2022_12_28_20_45_34_345.jpg	0.0	0.0	0	0.637010
4	center_2022_12_28_20_45_34_447.jpg	left_2022_12_28_20_45_34_447.jpg	right_2022_12_28_20_45_34_447.jpg	0.0	0.0	0	0.630620

We have an **abnormally high number of images with steering angle of 0**, which is expected since most of the time we are driving straight.

```
num_bins = 25
samples_per_bin = 200
hist, bins = np.histogram(data['steering'], num_bins)
center = (bins[:-1]+ bins[1:]) * 0.5
plt.bar(center, hist, width=0.05)
plt.plot((np.min(data['steering']), np.max(data['steering'])), (samples_per_bin, samples_per_bin))
print('total data:', len(data))
```

```
total data: 5395
```



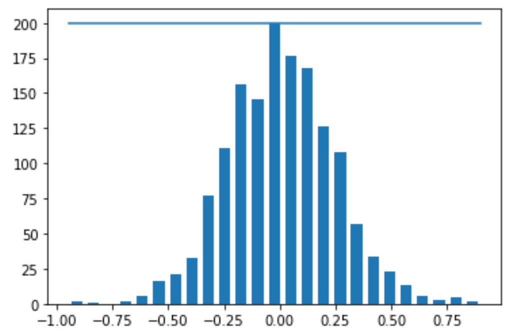
So as to ensure our model is not bias towards driving straight all the time, let's **cut down the number of images with 0 steering angle to just 200**.

```
#randomly remove some of the images with steering angle of 0 so that the distribution is not bias
remove_list = []
for j in range(num_bins):
  list_ = []
  for i in range(len(data['steering'])):
    if data['steering'][i] >= bins[j] and data['steering'][i] <= bins[j+1]:
      list_.append(i)
  list_ = shuffle(list_)
  list_ = list_[samples_per_bin:]
  remove_list.extend(list_)

print('removed:', len(remove_list))
data.drop(data.index[remove_list], inplace=True)
print('remaining:', len(data))
```

```
removed: 3902
remaining: 1493
```

```
hist, _ = np.histogram(data['steering'], (num_bins))
plt.bar(center, hist, width=0.05)
plt.plot((np.min(data['steering']), np.max(data['steering'])), (samples_per_bin, samples_per_bin))
```

```
[<matplotlib.lines.Line2D at 0x7fd9c0dbd700>]
```



Next, we define and run a function called load_img_steering() to return 2 lists of values. 1 list contains all the image path. Another list contains all the corresponding steering angle for each image.

```
print(data.iloc[1])
def load_img_steering(datadir, df):
  image_path = []
  steering = []
  for i in range(len(data)):
    indexed_data = data.iloc[i]
    center, left, right = indexed_data[0], indexed_data[1], indexed_data[2]
    image_path.append(os.path.join(datadir, center.strip()))
    steering.append(float(indexed_data[3]))
    # left image append
    image_path.append(os.path.join(datadir,left.strip()))
    steering.append(float(indexed_data[3])+0.15)
    # right image append
    image_path.append(os.path.join(datadir,right.strip()))
    steering.append(float(indexed_data[3])-0.15)
  image_paths = np.asarray(image_path)
  steerings = np.asarray(steering)
  return image_paths, steerings

image_paths, steerings = load_img_steering(datadir + '/IMG', data)
```

```
print(image_paths)
```

```
['Track/IMG/center_2022_12_28_20_45_35_358.jpg'
 'Track/IMG/left_2022_12_28_20_45_35_358.jpg'
 'Track/IMG/right_2022_12_28_20_45_35_358.jpg' ...
 'Track/IMG/center_2022_12_28_20_54_56_869.jpg'
 'Track/IMG/left_2022_12_28_20_54_56_869.jpg'
 'Track/IMG/right_2022_12_28_20_54_56_869.jpg']
```

```
print(steerings)
```

```
[-0.234633   -0.084633   -0.384633   ... 0.03580199  0.18580199
 -0.11419801]
```

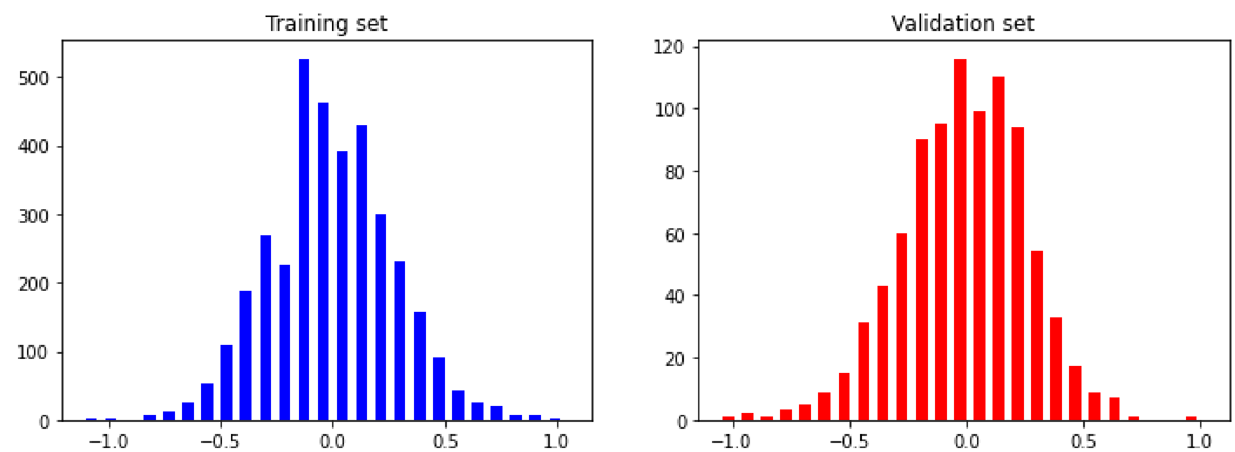
Now, we make use of **train_test_split()** to help us split the data into 80% training and 20% validation test sets.

```
X_train, X_valid, y_train, y_valid = train_test_split(image_paths, steerings, test_size=0.2, random_state=6)
print('Training Samples: {}\nValid Samples: {}'.format(len(X_train), len(X_valid)))
```

```
Training Samples: 3583
Valid Samples: 896
```

```
fig, axes = plt.subplots(1, 2, figsize=(12, 4))
axes[0].hist(y_train, bins=num_bins, width=0.05, color='blue')
axes[0].set_title('Training set')
axes[1].hist(y_valid, bins=num_bins, width=0.05, color='red')
axes[1].set_title('Validation set')
```

```
Text(0.5, 1.0, 'Validation set')
```



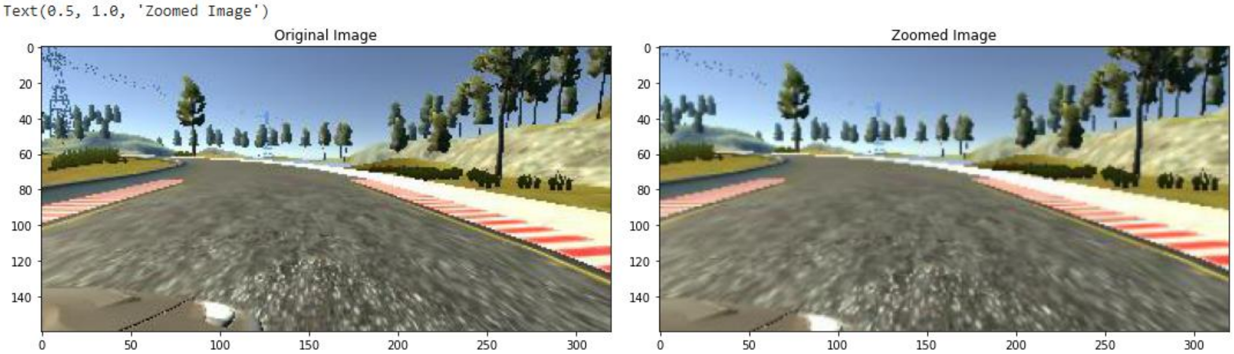
Good! Both datasets look rather similar in distribution.

To boost the variation of images we feed into our model, let's make use of the **augmenters module from imgaug library**.

Defining a function to zoom image by a random value of range 0% to 30%

```
def zoom(image):
  zoom = iaa.Affine(scale=(1, 1.3))                    #zoom in by a random value of range 0% to 30%
  image = zoom.augment_image(image)                    #apply zoom parameters to our image
  return image
image = image_paths[random.randint(0, 1000)]
original_image = mpimg.imread(image)
zoomed_image = zoom(original_image)

fig, axs = plt.subplots(1, 2, figsize=(15, 10))
fig.tight_layout()

axs[0].imshow(original_image)
axs[0].set_title('Original Image')

axs[1].imshow(zoomed_image)
axs[1].set_title('Zoomed Image')
```

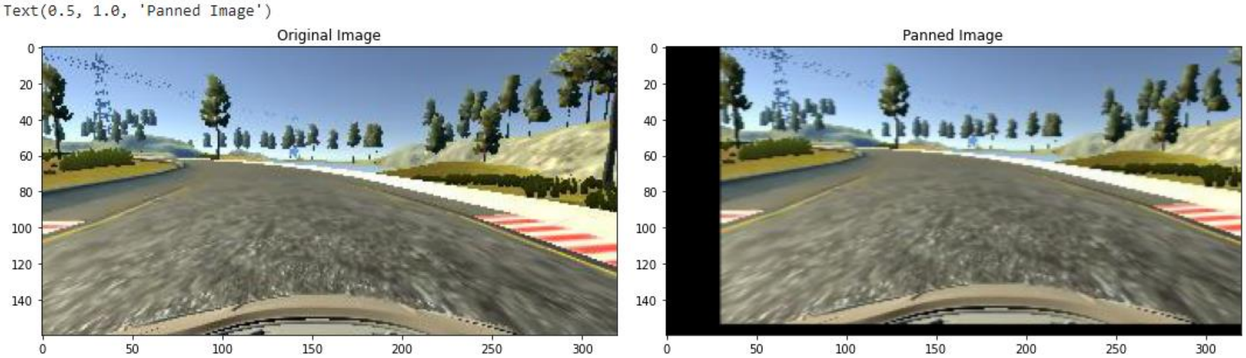
```
Text(0.5, 1.0, 'Zoomed Image')
```



Defining a function to translate image up to 10% in all 4 directions

```
def pan(image):
  pan = iaa.Affine(translate_percent= {"x" : (-0.1, 0.1), "y": (-0.1, 0.1)})     #translate up to 10% in all 4 directions
  image = pan.augment_image(image)                                                #apply translation parameters to our image
  return image
image = image_paths[random.randint(0, 1000)]
original_image = mpimg.imread(image)
panned_image = pan(original_image)

fig, axs = plt.subplots(1, 2, figsize=(15, 10))
fig.tight_layout()

axs[0].imshow(original_image)
axs[0].set_title('Original Image')

axs[1].imshow(panned_image)
axs[1].set_title('Panned Image')
```

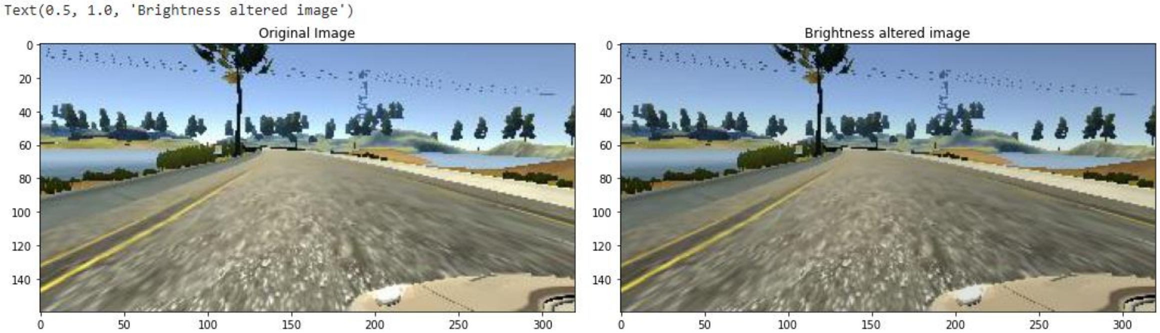
```
Text(0.5, 1.0, 'Panned Image')
```



Defining a function to multiply all pixel intensities by a range of 0.2 to 1.2 (0.2 means 80% decrease)

```
def img_random_brightness(image):
    brightness = iaa.Multiply((0.2, 1.2))              #multiply all pixel intensities by a range of 0.2 to 1.2 (0.2 means 80% decrease in brightness)
    image = brightness.augment_image(image)            #apply new brightness parameters to our image
    return image
image = image_paths[random.randint(0, 1000)]
original_image = mpimg.imread(image)
brightness_altered_image = img_random_brightness(original_image)

fig, axs = plt.subplots(1, 2, figsize=(15, 10))
fig.tight_layout()

axs[0].imshow(original_image)
axs[0].set_title('Original Image')

axs[1].imshow(brightness_altered_image)
axs[1].set_title('Brightness altered image')
```

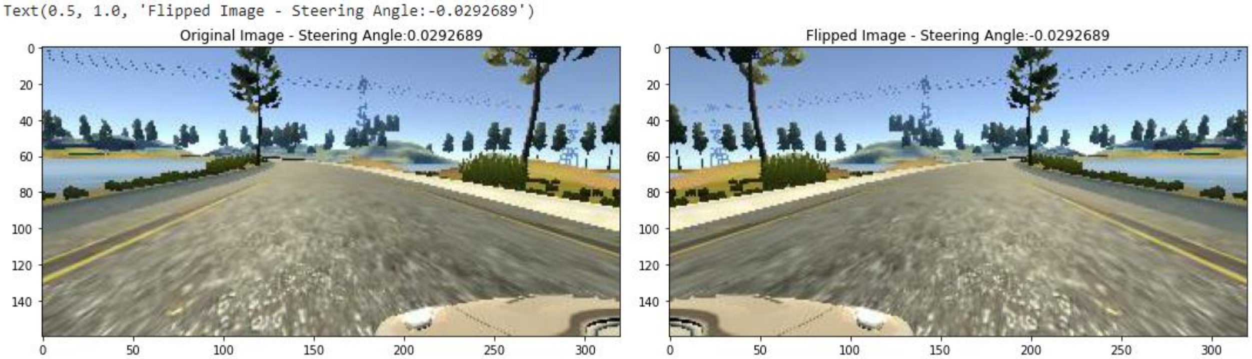
```
Text(0.5, 1.0, 'Brightness altered image')
```



Defining a function to flip images

```
def img_random_flip(image, steering_angle):
    image = cv2.flip(image,1)                  #flip the image to provide more and better distributed data in both directions
    steering_angle = -steering_angle           #need to flip the steering angle too
    return image, steering_angle

random_index = random.randint(0, 1000)
image = image_paths[random_index]
steering_angle = steerings[random_index]

original_image = mpimg.imread(image)
flipped_image, flipped_steering_angle = img_random_flip(original_image, steering_angle)

fig, axs = plt.subplots(1, 2, figsize=(15, 10))
fig.tight_layout()

axs[0].imshow(original_image)
axs[0].set_title('Original Image - ' + 'Steering Angle:' + str(steering_angle))

axs[1].imshow(flipped_image)
axs[1].set_title('Flipped Image - ' + 'Steering Angle:' + str(flipped_steering_angle))
```

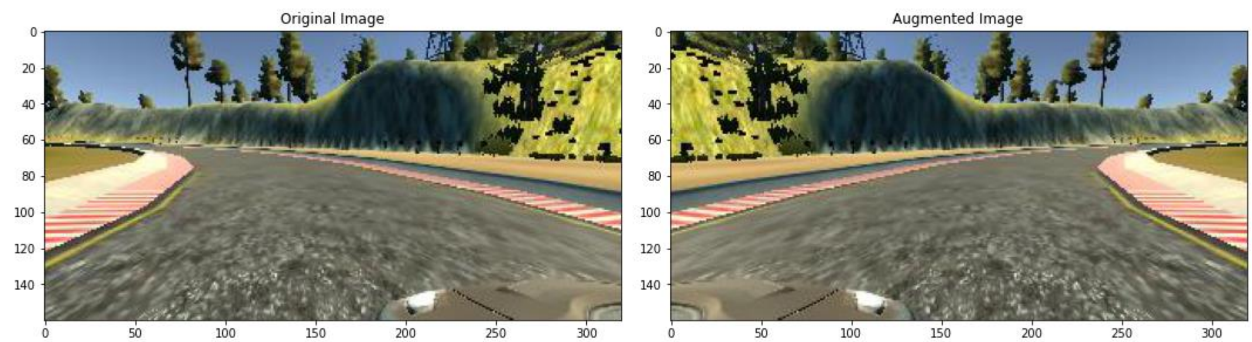
```
Text(0.5, 1.0, 'Flipped Image - Steering Angle:-0.0292689')
```

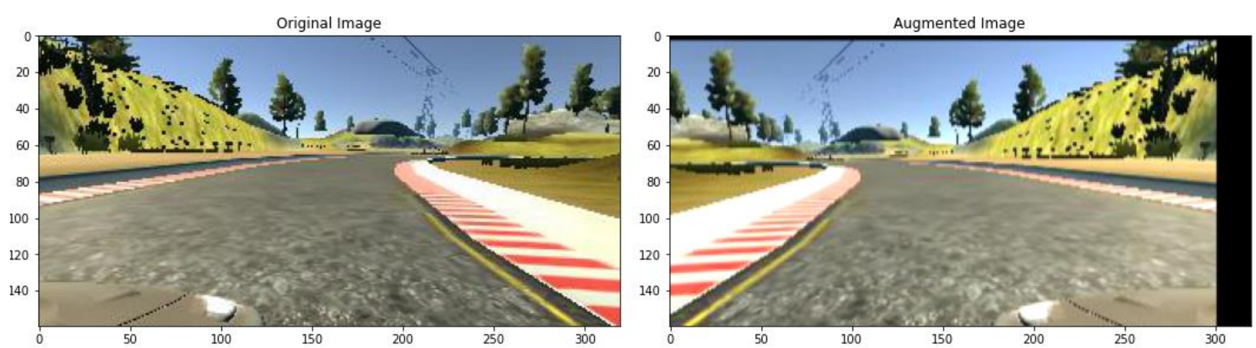


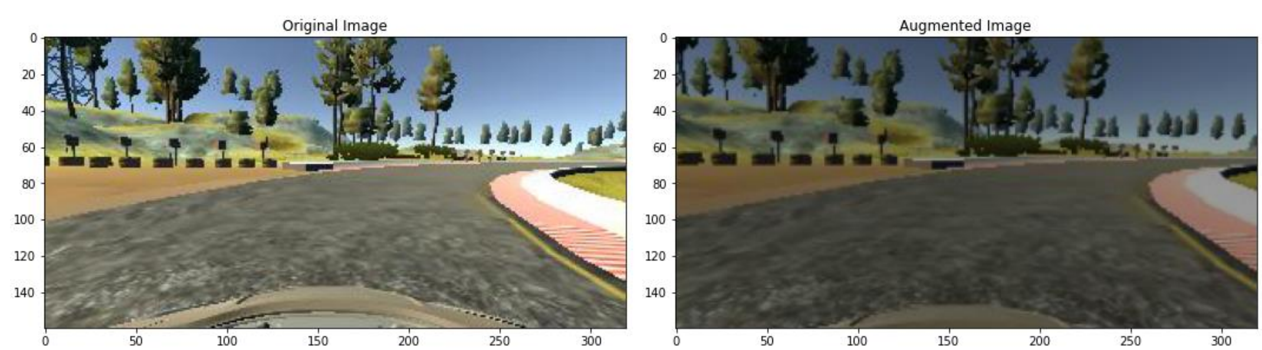
Now, we can **combine all 4 augment functions into 1 method called random_augment()**. What this method does is that for each image, there is a **50% probability** that it will apply each of the 4 functions to the image.

```
#function to randomly augment (i.e. zoom, translate, change intensity, flip) the image based on 50% probability
def random_augment(image, steering_angle):
    image = mpimg.imread(image)
    if np.random.rand() < 0.5:
      image = pan(image)
    if np.random.rand() < 0.5:
      image = zoom(image)
    if np.random.rand() < 0.5:
      image = img_random_brightness(image)
    if np.random.rand() < 0.5:
      image, steering_angle = img_random_flip(image, steering_angle)

    return image, steering_angle
```

Here are some examples of passing images through the random_augment() method.



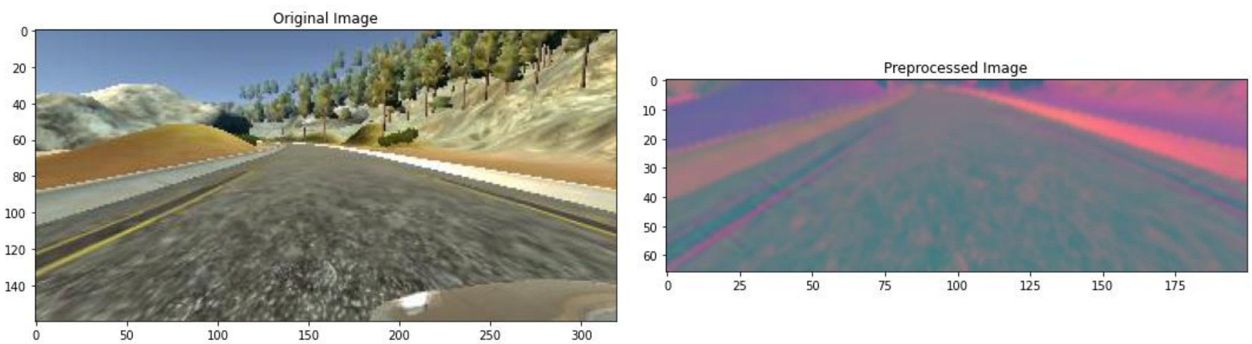




Before we start working on an image generator to create artificial images using the above random_augment() method, we need to develop a method to **preprocess images** so as to obtain a format that is suitable to be fed into our model. Refer to code comments below for details.

```
def img_preprocess(img):
    img = img[60:135,:,:]                          #remove unnecessary noise from image such as hood of car and mountain view
    img = cv2.cvtColor(img, cv2.COLOR_RGB2YUV)     #YUV format is recommended by Nvidia for use by their neural model
    img = cv2.GaussianBlur(img,  (3, 3), 0)        #for smoothing and removing noise
    img = cv2.resize(img, (200, 66))               #to match with Nvidia model
    img = img/255
    return img
```

This is how a processed image looks like.

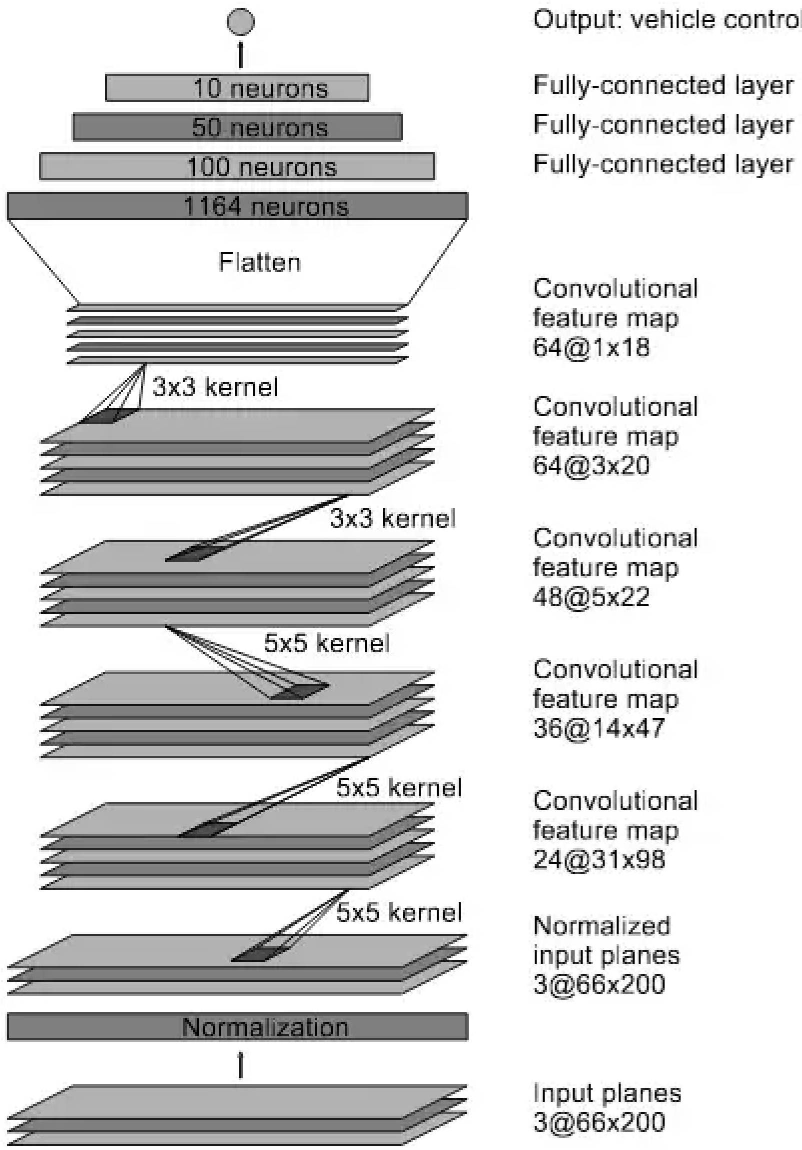


Next, we need to **create a batch generator** to generate artificial augmented images from our data. This helps to increase the variation and amount of our training data.

```
#batch generator allows the creation of augmented images when called instead of creating all at once and storing them (which takes up too much memory)

def batch_generator(image_paths, steering_ang, batch_size, is_training):
  while True:
    batch_img = []
    batch_steering = []

    for i in range(batch_size):
      random_index = random.randint(0, len(image_paths) - 1)
      if is_training:
        im, steering = random_augment(image_paths[random_index], steering_ang[random_index])
      else:
        im = mpimg.imread(image_paths[random_index])
        steering = steering_ang[random_index]

      im = img_preprocess(im)
      batch_img.append(im)
      batch_steering.append(steering)

    yield (np.asarray(batch_img), np.asarray(batch_steering))
```

With all these being done, it is finally time to develop our CNN model. For the model, we will **adopt the architecture used by Nvidia** for self-driving car behavioural cloning since it is widely known to return good results.

Nvidia CNN model



NVidia Convolutional Neural Network

Note that we are using **Mean Square Error (MSE)** for our loss function given that this is a **regression** task.

```
#Nvidia model

def nvidia_model():
  model = Sequential()
  model.add(Conv2D(24, kernel_size=(5,5), strides=(2,2), input_shape=(66,200,3),activation='elu'))
  model.add(Conv2D(36, kernel_size=(5,5), strides=(2,2), activation='elu'))
  model.add(Conv2D(48, kernel_size=(5,5), strides=(2,2), activation='elu'))
  model.add(Conv2D(64, kernel_size=(3,3), activation='elu'))
  model.add(Conv2D(64, kernel_size=(3,3), activation='elu'))
# model.add(Dropout(0.5))

  model.add(Flatten())

  model.add(Dense(100, activation = 'elu'))
# model.add(Dropout(0.5))

  model.add(Dense(50, activation = 'elu'))
# model.add(Dropout(0.5))

  model.add(Dense(10, activation = 'elu'))
# model.add(Dropout(0.5))

  model.add(Dense(1))

  optimizer = Adam(lr=1e-3)
  model.compile(loss='mse', optimizer=optimizer)
  return model
```

Next, we use the **fit_generator()** function to create artificial images in batches and fit the model.

```
history = model.fit_generator(batch_generator(X_train, y_train, 100, 1),  #batch size of 100
                              steps_per_epoch=300,                        #number of times you call batch generator to augment images
                              epochs=10,
                              validation_data=batch_generator(X_valid, y_valid, 100, 0),  #batch size of 100
                              validation_steps=200,
                              verbose=1,
                              shuffle = 1)
```

```
<ipython-input-32-0371458cbad2>:1: UserWarning: `Model.fit_generator` is deprecated and will be removed in a future version. Please u
  history = model.fit_generator(batch_generator(X_train, y_train, 100, 1),  #batch size of 100
Epoch 1/10
300/300 [==============================] - 127s 424ms/step - loss: 0.0587 - val_loss: 0.0376
Epoch 2/10
300/300 [==============================] - 127s 426ms/step - loss: 0.0471 - val_loss: 0.0292
Epoch 3/10
300/300 [==============================] - 122s 409ms/step - loss: 0.0402 - val_loss: 0.0271
Epoch 4/10
300/300 [==============================] - 125s 417ms/step - loss: 0.0355 - val_loss: 0.0228
Epoch 5/10
300/300 [==============================] - 123s 413ms/step - loss: 0.0352 - val_loss: 0.0219
Epoch 6/10
300/300 [==============================] - 125s 419ms/step - loss: 0.0330 - val_loss: 0.0253
Epoch 7/10
300/300 [==============================] - 122s 408ms/step - loss: 0.0308 - val_loss: 0.0224
Epoch 8/10
300/300 [==============================] - 123s 412ms/step - loss: 0.0299 - val_loss: 0.0223
Epoch 9/10
300/300 [==============================] - 125s 419ms/step - loss: 0.0296 - val_loss: 0.0218
Epoch 10/10
300/300 [==============================] - 123s 412ms/step - loss: 0.0284 - val_loss: 0.0214
```

Last step is to save the model as "**model.h5**" and **download** onto our computer.

```
model.save('model.h5')
```

```
from google.colab import files
files.download('model.h5')
```

With our model, we can now feed it through the Self-driving Car Simulator. To do that, I made use of this **drive.py script that I have downloaded from the web to connect our model and the simulator**. The script helps to automate the process of gathering frames of driving images, preprocess the data, run it through our model, and predict and return a steering angle to the simulator.

drive.py

```
import socketio
import eventlet
import numpy as np
from flask import Flask
from keras.models import load_model
import base64
from io import BytesIO
from PIL import Image
import cv2

sio = socketio.Server()

app = Flask(__name__) #'__main__'
speed_limit = 10
def img_preprocess(img):
    img = img[60:135,:,:]
    img = cv2.cvtColor(img, cv2.COLOR_RGB2YUV)
    img = cv2.GaussianBlur(img,  (3, 3), 0)
    img = cv2.resize(img, (200, 66))
    img = img/255
    return img

@sio.on('telemetry')
def telemetry(sid, data):
    speed = float(data['speed'])
    image = Image.open(BytesIO(base64.b64decode(data['image'])))
    image = np.asarray(image)
    image = img_preprocess(image)
    image = np.array([image])
    steering_angle = float(model.predict(image))
    throttle = 1.0 - speed/speed_limit
    print('{} {} {}'.format(steering_angle, throttle, speed))
    send_control(steering_angle, throttle)


@sio.on('connect')
def connect(sid, environ):
    print('Connected')
    send_control(0, 0)

def send_control(steering_angle, throttle):
    sio.emit('steer', data = {
        'steering_angle': steering_angle.__str__(),
        'throttle': throttle.__str__()
```

We are connected to the simulator! Let’s see how our self-driving car runs by itself! 😊

Select This

```
(myenviron) C:\Users\benja\Desktop\data visualisation\Udemy Course\Python Course\Machine Learning\Self-Driving Car Appli
ed Deep Learning\Behavioural Cloning>python drive.py
2022-12-30 15:37:38.727379: I tensorflow/core/platform/cpu_feature_guard.cc:193] This TensorFlow binary is optimized wit
h oneAPI Deep Neural Network Library (oneDNN) to use the following CPU instructions in performance-critical operations:
 AVX2
To enable them in other operations, rebuild TensorFlow with the appropriate compiler flags.
2022-12-30 15:37:38.730798: I tensorflow/core/common_runtime/process_util.cc:146] Creating new thread pool with default
inter op setting: 2. Tune using inter_op_parallelism_threads for best performance.
(21756) wsgi starting up on http://0.0.0.0:4567
(21756) accepted ('127.0.0.1', 54359)
Connected
1/1 [==============================] - 0s 291ms/step
-0.13627462089061737 0.781 2.19
1/1 [==============================] - 0s 24ms/step
-0.13627462089061737 0.7372 2.628
1/1 [==============================] - 0s 26ms/step
-0.14749176800251007 0.6934 3.066
1/1 [==============================] - 0s 23ms/step
-0.21763266623020172 0.8738900000000001 1.2611
1/1 [==============================] - 0s 25ms/step
-0.20208221673965454 0.9454 0.546
```

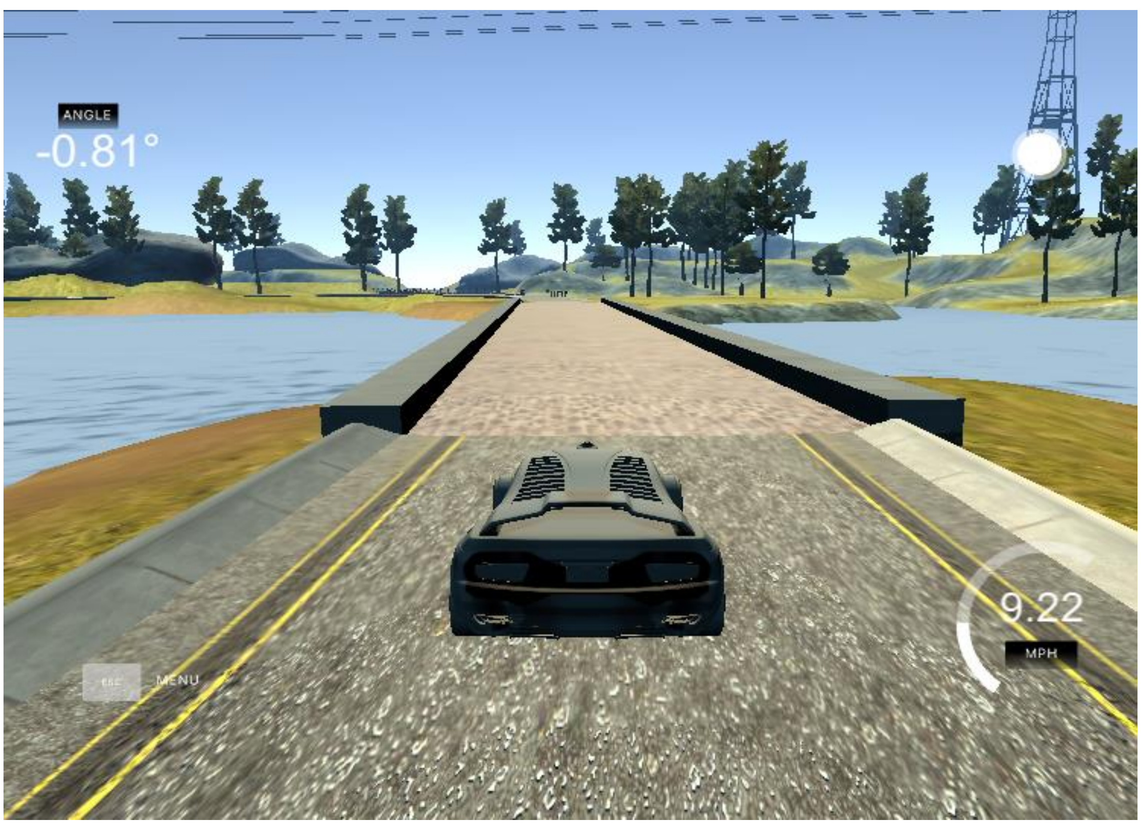

ANGLE
0.17°
ESC
MENU
9.22
MPH


ANGLE
-0.79°
ESC
MENU
9.22
MPH


ANGLE
-1.88°
ESC
MENU
9.23
MPH


ANGLE
-1.72°
ESC
MENU
9.22
MPH


ANGLE
-0.79°
ESC
MENU
9.22
MPH


ANGLE
-0.81°
ESC
MENU
9.22
MPH


ANGLE
0.24°
9.21
MPH
ESC
MENU


ANGLE
-1.60°
9.20
MPH
MENU


ANGLE
-4.39°
9.22
MPH


ANGLE
-7.11°
9.20
MPH


ANGLE
3.85°
ESC
MENU
9.21
MPH


ANGLE
2.60°
ESC
MENU
9.22
MPH


ANGLE
0.90°
ESC
MENU
9.22
MPH


ANGLE
-2.86°
ESC
MENU
9.21
MPH


ANGLE
-3.37°
9.22
MPH