

NAME-ANIRBAN DEY

PROJECT TITLE-SELF-DRIVING CAR

SELF-DRIVING CAR USING CAR SIMULATOR ENVIRONMENT

Table of contents

Abstract

Introduction

- 1.problem faced
- 2.Technology used
- 3.CNN

Devices

Training process/Methodology

Image pre-processing

Network architecture

Conclusion

References

ABSTRACT

In this project it is about a simple self-driving car, how it will work and implemented it will be discussed. The car is used to detect the road signals, taking the right and left turn accordingly using the steering angle. To determine the whole system, we need to analyse it with different process. The whole system capable of taking right decision with excellent accuracy.

INTRODUCTION

A self-driving car is a vehicle that capable of sensing its environment and navigating without human input. Self-driving cars can detect surroundings, using a variety of techniques such as radar, GPS, and computer vision. An autonomous or self-driving car that capable of sensing its environment and navigating without human inputs. Fully autonomous vehicles will be available on the market by the end of the decade – but, for now, barriers to widespread adoption remain. The potential for connected vehicles during the interim, however, is an exciting, multi-faceted and high-growth area in the IoT's development – enabling enhanced road safety, smart traffic management, advanced navigation assistance, passenger entertainment and much more.

Problem statement

In this project we are trying that how a self-driving car works using different technologies. The car will depict at what steering angle the car will turn left, right or straight in a road path and give correct decision to the car.

Technology used

Technologies that are used in the implementation of this project are as follows

TensorFlow: This an open-source library for data flow programming. It is used for machine learning applications. It is used for functions in math library and large computation techniques. For this project, high level API uses TensorFlow as the backend is used. KERAS used in building the machine learning or AI models as it is more user friendly.

Different libraries in machine learning in python help in machine learning. Many of those libraries improved the performance of the projects. Few of them are as follows first "NUMPY" which is used to provides high level math function for multidimensional metrices or arrays. This is used for faster communications in neural network. Second, is "scikit-learn" is a machine learning library for python features different algorithms and machine learning python packages. Another one is "OpenCV" which is used for image processing and other techniques.

Convolutional Neural Network

CNN is a network architecture which is used for deep learning which learns directly from data. CNNs are used for finding patterns in images to recognize objects, faces. They are also helpful for identifying different data such as audio, time series and signal data. Object recognition and computer vision-such as self-driving vehicles and faces recognition applications-rely heavily in CNNs.

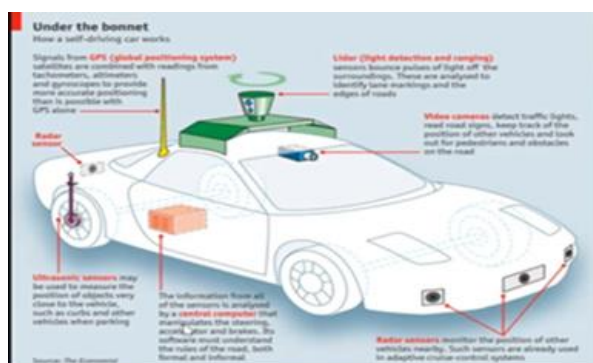
Time distributed layers

Another type of layers sometimes used in deep learning networks is a Time-distributed layer. Time-Distributed layers are provided in KERAS as wrapper layers. Every temporal slice of an input is applied with this wrapper layer. The requirement for input is that to be at least three-dimensional, first index can be considered as temporal dimension. These Time-Distributed can be applied to a dense layer to each of the timesteps, independently or even used with Convolutional Layers.

DEVICES

INPUTS

- 1.cameras(front ,rear ,side)
- 2.LIDAR
- 3.RADAR
- 4.Ultrasonic sensors
- 5.GPS



Video camera-It is used to detect traffic light, read road lights, keep track on the position of other vehicles and look out for pedestrian and obstacles on the road

Lidar (light detection and ranging)-it is a sensor bounce pounces of light of the surroundings. These are used to analyse the line markings and the edges of road.

GPS(Global positioning system)-This satellites are combined with readings from tachometers, altimeter and gyroscopes to provide more accurate positioning than is possible with GPS alone.

Radar sensors-it is used to monitor the position of other vehicles nearby. Such sensors are used in adaptive cruise control systems.

Ultrasonic sensors-Used to measure the position of the objects which is very close to the vehicles such as curbs and other vehicles when parking.

Download the dataset:

I have downloaded the dataset using Kaggle. Here are some images and the screenshot of the csv file.

Some images from the dataset:



Right



Center



Left

A sample of my csv file is shown

Column A, B, C: provide the paths of images in middle, right and left.

Column D: contains the steering angle value as 0 means straight, positive value is right turn and negative value is left turn

Column E: provides the acceleration at that instance.

Column F: contains the brakes at that instance

Column G: contains the speed of the vehicle

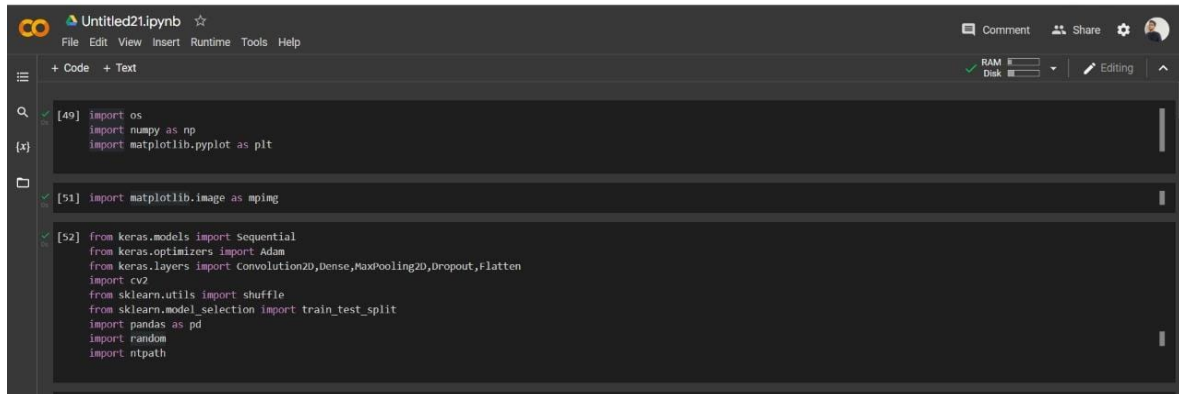
	A	B	C	D	E	F	G
6	/home/aman-py/Desktop/Self-Driving-Car/data/IMG/left_2018_10_17_16_36_31_484.j	/home/aman-py/Desktop/Self-Driving-Car/data/IMG/right_2018_10_17_16_36_31_484.j	/home/aman-py/Desktop/Self-Driving-Car/data/IMG/center_2018_10_17_16_36_31_484.j	0	0	0	7.83E-05
7	/home/aman-py/Desktop/Self-Driving-Car/data/IMG/left_2018_10_17_16_36_31_619.j	/home/aman-py/Desktop/Self-Driving-Car/data/IMG/right_2018_10_17_16_36_31_619.j	/home/aman-py/Desktop/Self-Driving-Car/data/IMG/center_2018_10_17_16_36_31_619.j	0	0	0	7.80E-05
8	/home/aman-py/Desktop/Self-Driving-Car/data/IMG/left_2018_10_17_16_36_31_755.j	/home/aman-py/Desktop/Self-Driving-Car/data/IMG/right_2018_10_17_16_36_31_755.j	/home/aman-py/Desktop/Self-Driving-Car/data/IMG/center_2018_10_17_16_36_31_755.j	0	0	0	7.81E-05
9	/home/aman-py/Desktop/Self-Driving-Car/data/IMG/left_2018_10_17_16_36_31_862.j	/home/aman-py/Desktop/Self-Driving-Car/data/IMG/right_2018_10_17_16_36_31_862.j	/home/aman-py/Desktop/Self-Driving-Car/data/IMG/center_2018_10_17_16_36_31_862.j	0	0	0	7.82E-05
10	/home/aman-py/Desktop/Self-Driving-Car/data/IMG/left_2018_10_17_16_36_31_964.j	/home/aman-py/Desktop/Self-Driving-Car/data/IMG/right_2018_10_17_16_36_31_964.j	/home/aman-py/Desktop/Self-Driving-Car/data/IMG/center_2018_10_17_16_36_31_964.j	0	0	0	7.94E-05
11	/home/aman-py/Desktop/Self-Driving-Car/data/IMG/left_2018_10_17_16_36_32_081.j	/home/aman-py/Desktop/Self-Driving-Car/data/IMG/right_2018_10_17_16_36_32_081.j	/home/aman-py/Desktop/Self-Driving-Car/data/IMG/center_2018_10_17_16_36_32_081.j	0	0	0	7.90E-05
12	/home/aman-py/Desktop/Self-Driving-Car/data/IMG/left_2018_10_17_16_36_32_196.j	/home/aman-py/Desktop/Self-Driving-Car/data/IMG/right_2018_10_17_16_36_32_196.j	/home/aman-py/Desktop/Self-Driving-Car/data/IMG/center_2018_10_17_16_36_32_196.j	0	0	0	7.85E-05
13	/home/aman-py/Desktop/Self-Driving-Car/data/IMG/left_2018_10_17_16_36_32_336.j	/home/aman-py/Desktop/Self-Driving-Car/data/IMG/right_2018_10_17_16_36_32_336.j	/home/aman-py/Desktop/Self-Driving-Car/data/IMG/center_2018_10_17_16_36_32_336.j	0	0	0	7.85E-05
14	/home/aman-py/Desktop/Self-Driving-Car/data/IMG/left_2018_10_17_16_36_32_461.j	/home/aman-py/Desktop/Self-Driving-Car/data/IMG/right_2018_10_17_16_36_32_461.j	/home/aman-py/Desktop/Self-Driving-Car/data/IMG/center_2018_10_17_16_36_32_461.j	0	0	0	7.79E-05
15	/home/aman-py/Desktop/Self-Driving-Car/data/IMG/left_2018_10_17_16_36_32_563.j	/home/aman-py/Desktop/Self-Driving-Car/data/IMG/right_2018_10_17_16_36_32_563.j	/home/aman-py/Desktop/Self-Driving-Car/data/IMG/center_2018_10_17_16_36_32_563.j	0	0	0	7.79E-05
16	/home/aman-py/Desktop/Self-Driving-Car/data/IMG/left_2018_10_17_16_36_32_654.j	/home/aman-py/Desktop/Self-Driving-Car/data/IMG/right_2018_10_17_16_36_32_654.j	/home/aman-py/Desktop/Self-Driving-Car/data/IMG/center_2018_10_17_16_36_32_654.j	0	0	0	7.80E-05
17	/home/aman-py/Desktop/Self-Driving-Car/data/IMG/left_2018_10_17_16_36_32_765.j	/home/aman-py/Desktop/Self-Driving-Car/data/IMG/right_2018_10_17_16_36_32_765.j	/home/aman-py/Desktop/Self-Driving-Car/data/IMG/center_2018_10_17_16_36_32_765.j	0	0	0	7.96E-05
18	/home/aman-py/Desktop/Self-Driving-Car/data/IMG/left_2018_10_17_16_36_32_899.j	/home/aman-py/Desktop/Self-Driving-Car/data/IMG/right_2018_10_17_16_36_32_899.j	/home/aman-py/Desktop/Self-Driving-Car/data/IMG/center_2018_10_17_16_36_32_899.j	0	0	0	7.89E-05
19	/home/aman-py/Desktop/Self-Driving-Car/data/IMG/left_2018_10_17_16_36_33_015.j	/home/aman-py/Desktop/Self-Driving-Car/data/IMG/right_2018_10_17_16_36_33_015.j	/home/aman-py/Desktop/Self-Driving-Car/data/IMG/center_2018_10_17_16_36_33_015.j	0	0	0	7.77E-05
20	/home/aman-py/Desktop/Self-Driving-Car/data/IMG/left_2018_10_17_16_36_33_149.j	/home/aman-py/Desktop/Self-Driving-Car/data/IMG/right_2018_10_17_16_36_33_149.j	/home/aman-py/Desktop/Self-Driving-Car/data/IMG/center_2018_10_17_16_36_33_149.j	0	0.2235782	0	0.2044514

The Process/Methodology

For the process of working of the self-driving car, we have to upload the images that we are recorded.

We will be using Google Collab for doing the training process.

We will now import all the libraries needed for training, we will use KERAS at the front end and TensorFlow at the backend

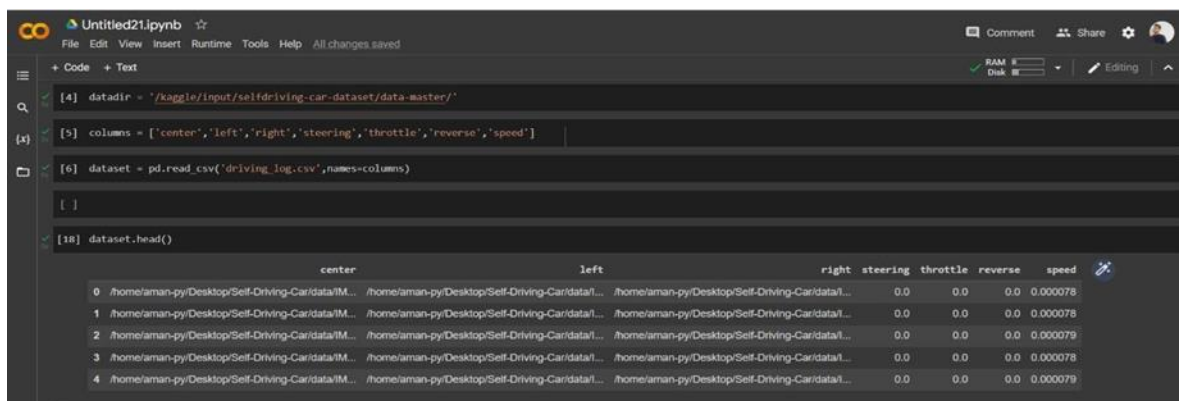


```
[49] import os
import numpy as np
import matplotlib.pyplot as plt

[51] import matplotlib.image as mpimg

[52] from keras.models import Sequential
from keras.optimizers import Adam
from keras.layers import Convolution2D,Dense,MaxPooling2D,Dropout,Flatten
import cv2
from sklearn.utils import shuffle
from sklearn.model_selection import train_test_split
import pandas as pd
import random
import ntpath
```

We will show the first five values for the csv on the desired format. We will use a filename given to the folder itself and then take the parameters.



```
[4] datadir = '/kaggle/input/selfdriving-car-dataset/data-master/'

[5] columns = ['center','left','right','steering','throttle','reverse','speed']

[6] dataset = pd.read_csv('driving_log.csv',names=columns)

[ ]

[18] dataset.head()
```

	center	left	right	steering	throttle	reverse	speed
0	/home/aman-py/Desktop/Self-Driving-Car/data/IM...	/home/aman-py/Desktop/Self-Driving-Car/data/IM...	/home/aman-py/Desktop/Self-Driving-Car/data/IM...	0.0	0.0	0.0	0.000078
1	/home/aman-py/Desktop/Self-Driving-Car/data/IM...	/home/aman-py/Desktop/Self-Driving-Car/data/IM...	/home/aman-py/Desktop/Self-Driving-Car/data/IM...	0.0	0.0	0.0	0.000078
2	/home/aman-py/Desktop/Self-Driving-Car/data/IM...	/home/aman-py/Desktop/Self-Driving-Car/data/IM...	/home/aman-py/Desktop/Self-Driving-Car/data/IM...	0.0	0.0	0.0	0.000079
3	/home/aman-py/Desktop/Self-Driving-Car/data/IM...	/home/aman-py/Desktop/Self-Driving-Car/data/IM...	/home/aman-py/Desktop/Self-Driving-Car/data/IM...	0.0	0.0	0.0	0.000078
4	/home/aman-py/Desktop/Self-Driving-Car/data/IM...	/home/aman-py/Desktop/Self-Driving-Car/data/IM...	/home/aman-py/Desktop/Self-Driving-Car/data/IM...	0.0	0.0	0.0	0.000079

```
[x] def removePath(path):
    base,tail = ntpath.split(path)
    return tail

[20] dataset['center']=dataset['center'].apply(removePath)

[21] dataset['left'] = dataset['left'].apply(removePath)

[22] dataset['right'] = dataset['right'].apply(removePath)

[23] dataset.head()
```

	center	left	right	steering	throttle	reverse	speed
0	center_2018_10_17_16_36_30_865.jpg	left_2018_10_17_16_36_30_865.jpg	right_2018_10_17_16_36_30_865.jpg	0.0	0.0	0.0	0.000078
1	center_2018_10_17_16_36_30_966.jpg	left_2018_10_17_16_36_30_966.jpg	right_2018_10_17_16_36_30_966.jpg	0.0	0.0	0.0	0.000078
2	center_2018_10_17_16_36_31_098.jpg	left_2018_10_17_16_36_31_098.jpg	right_2018_10_17_16_36_31_098.jpg	0.0	0.0	0.0	0.000079
3	center_2018_10_17_16_36_31_253.jpg	left_2018_10_17_16_36_31_253.jpg	right_2018_10_17_16_36_31_253.jpg	0.0	0.0	0.0	0.000078
4	center_2018_10_17_16_36_31_366.jpg	left_2018_10_17_16_36_31_366.jpg	right_2018_10_17_16_36_31_366.jpg	0.0	0.0	0.0	0.000079

To get the center distribution we need to take the no of values as odd number. We will execute it through a histogram having data as “steering”.

```
[22] num_bins = 25

[23] hist,bins = np.histogram(dataset['steering'],num_bins)

[24] print(hist)
print(bins)
```

```
[ 48 16 13 15 21 36 48 69 161 174 337 239 5454 56
 160 82 97 99 37 25 32 15 12 5 8]
[-1. -0.92 -0.84 -0.76 -0.68 -0.6 -0.52 -0.44 -0.36 -0.28 -0.2 -0.12
 -0.04 0.04 0.12 0.2 0.28 0.36 0.44 0.52 0.6 0.68 0.76 0.84]
```

0s completed at 12:39 PM

```
[25] center = (bins[:-1]+bins[1:])*0.5

[26] center

array([-0.96, -0.88, -0.8 , -0.72, -0.64, -0.56, -0.48, -0.4 , -0.32,
       -0.24, -0.16, -0.08, 0. , 0.08, 0.16, 0.24, 0.32, 0.4 ,
        0.48, 0.56, 0.64, 0.72, 0.8 , 0.88, 0.96])

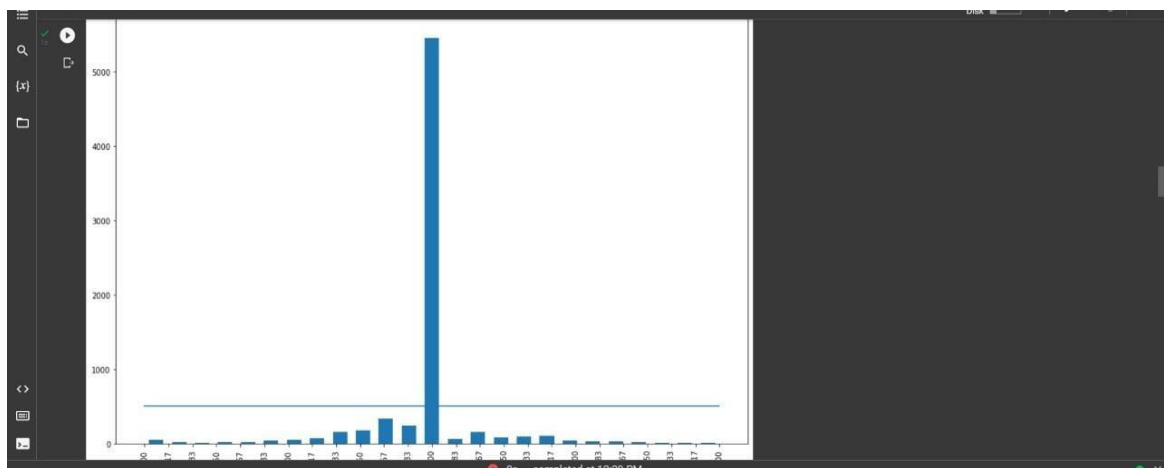
[27] center1 = []
for i in range(0,len(bins)-1):
    x = (bins[i] + bins[i+1]) * 0.5
    center1.append(x)

[28] center1

[-0.96,
 -0.88,
 -0.8,
 -0.72,
 -0.6399999999999999,
 -0.56,
```


Display the histogram

```
0.32000000000000006,  
0.4,  
0.48,  
0.56,  
0.64,  
0.72,  
0.8,  
0.88,  
0.96]  
  
[29] threshold = 500  
plt.figure(figsize=(15,10))  
plt.bar(center,hist,width=0.05)  
plt.xticks(np.linspace(-1,1,25),rotation=90)  
(x1,x2) = (np.min(dataset['steering']),np.max(dataset['steering']))  
(y1,y2) = (threshold,threshold)  
plt.title('Steering Angles')  
plt.plot((x1,x2),(y1,y2))
```



Now, we will provide a variable. We will identify the samples which we want to remove through a looping statement in which every bin will iterate the steering data as because it is unstructured. It will now randomly shuffle the data to make it uniformly structured.

We are doing this to provide the output of the distribution of steering angle to be uniform.

```

[30] remove_list = []
for i in range(num_bins):
    list = []
    for j in range(len(dataset['steering'])):
        if dataset['steering'][j] >= bins[i] and dataset['steering'][j] <= bins[i+1]:
            list.append(j)
    list = shuffle(list)
    list = list[threshold:]
    remove_list.extend(list)

len(dataset['steering'])
7259

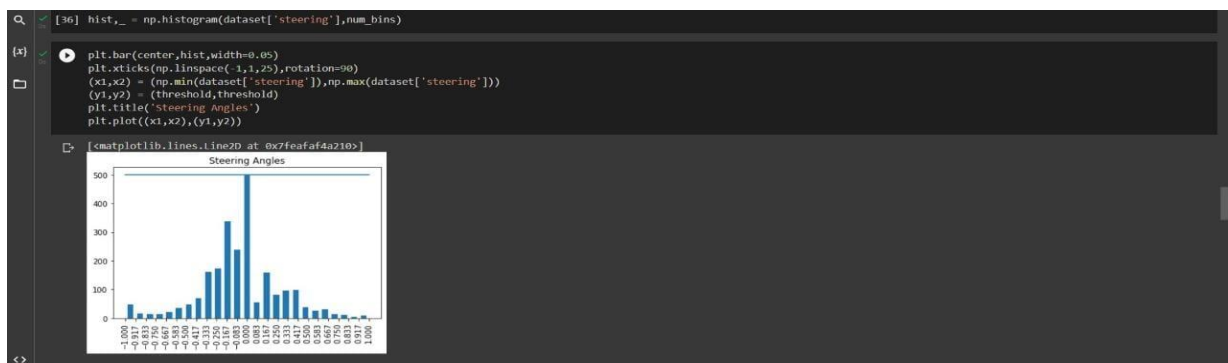
[32] len(remove_list)
4954

[33] dataset.drop(dataset.index[remove_list],inplace=True)

[34] hist,_ = np.histogram(dataset['steering'],num_bins)

```

Plot on it



```

dataset.iloc[1]

```

center	center_2018_10_17_16_36_33_479.jpg
left	left_2018_10_17_16_36_33_479.jpg
right	right_2018_10_17_16_36_33_479.jpg
steering	0.0
throttle	1.0
reverse	0.0
speed	3.178461
Name: 22, dtype: object	

We will now load the image to operate them accordingly by using function. There will be a image path as an empty list and steering as a empty list.

```
[38] def loadImageSteering(dataset):
    imagePath = []
    steeringPath = []
    for i in range(len(dataset)):
        center = dataset.iloc[i][0]
        steering = float(dataset.iloc[i][3])
        imagePath.append(os.path.join(datadir, center))
        steeringPath.append(steering)
    imagePath = np.asarray(imagePath)
    steeringPath = np.asarray(steeringPath)
    return imagePath, steeringPath
```

```
[39] dataset.iloc[0][0]
'center_2018_10_17_16_36_32_196.jpg'

[40] imagePath, steeringPath = loadImageSteering(dataset)

imagePath[0]
'/kaggle/input/selfdriving-car-dataset/data-master/center_2018_10_17_16_36_32_196.jpg'

[42] len(steeringPath)
2305
```

We will be dividing the image path using train test split and store the arrays accordingly.

```
from sklearn.model_selection import train_test_split

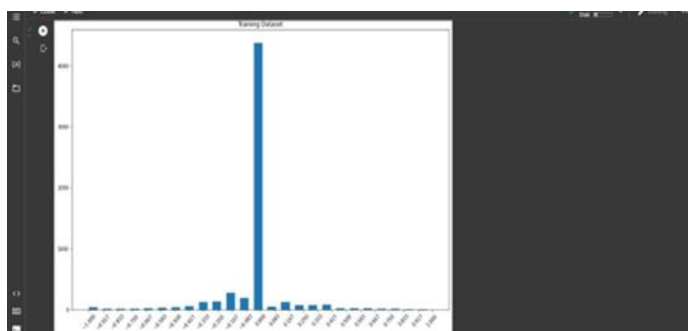
[15] from sklearn.model_selection import train_test_split
     x_train, x_test, y_train, y_test = train_test_split(imagePath, steeringPath, random_state=0, test_size=0.2)

[41] len(x_train)
5807
```

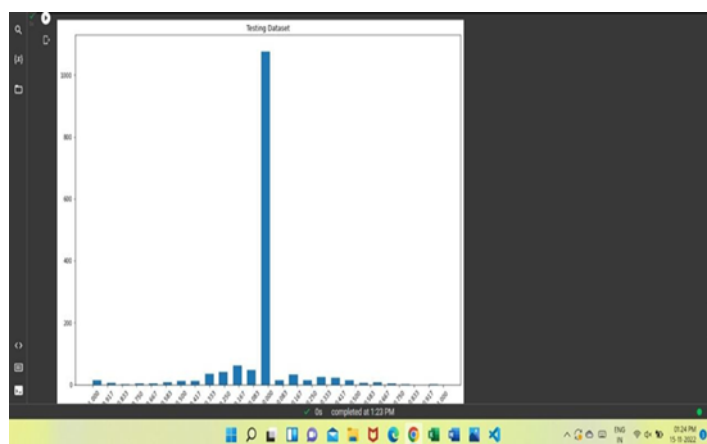
We will now display the histogram of training and testing data set

```
plt.figure(figsize=(15,10))
plt.hist(y_train,bins=num_bins,width=0.05)
plt.xticks(np.linspace(-1,1,25),rotation=45)
plt.title("Training Dataset")
plt.show()
```

```
plt.figure(figsize=(15,10))
plt.hist(y_test,bins=num_bins,width=0.05)
plt.xticks(np.linspace(-1,1,25),rotation=45)
plt.title("Testing Dataset")
plt.show()
```



Training dataset



Testing dataset

Image pre-processing

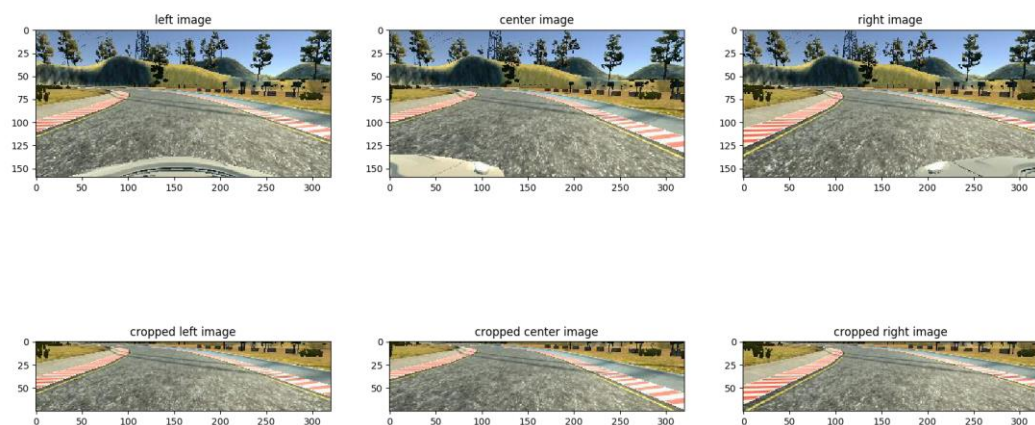
This is the process through which we can evaluate to process in different ways. In real life it is difficult to train a self-driving car model as the data is very huge, thus we need to generalize the conduct on different tasks. The problem is solved using image pre-processing.

```
[43] def imagePreprocessing(img):  
    img = mpimg.imread(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
```

Zoom

The images in the dataset have applicable features in the end part where the road is visible. The external environment above a certain image will not be used to depict the output and hence it is needed to crop.

Cropped images can be looked like this:



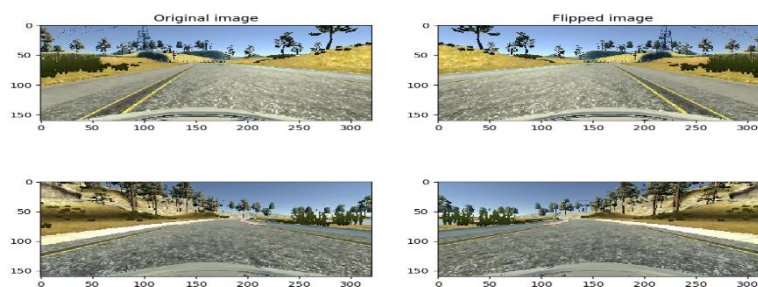
The code are as follows:

```
def zoom(image):
    zoom = iaa.Affine(scale=(1, 1.3))
    image = zoom.augment_image(image)
    return image
image = image_paths[random.randint(0, 1000)]
original_image = mpimg.imread(image)
zoomed_image = zoom(original_image)
fig, axes = plt.subplots(1, 2, figsize=(15, 10))
fig.tight_layout()
axes[0].imshow(original_image)
axes[0].set_title("Original Image")
axes[1].imshow(zoomed_image)
axes[1].set_title("Zoomed Image")
```

Flip/horizontal

The image can be flipped horizontally (i.e. the mirror image of the original image). The aim is that to be the model get trained for similar kind of turns on the opposite sides.

The images after flipping can be shown like this:



The code are as follows

```

def random_flip(image, steering_angle):
    image = cv2.flip(image, 1)
    steering_angle = -steering_angle
    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 = random_flip(original_image, steering_angle)
fig, axes = plt.subplots(1, 2, figsize=(15, 10))
fig.tight_layout()
axes[0].imshow(original_image)
axes[0].set_title("Original Image - " + "Steering Angle:" + str(steering_angle))
axes[1].imshow(flipped_image)
axes[1].set_title("Flipped Image - " + "Steering Angle:" + str(flipped_steering_angle))

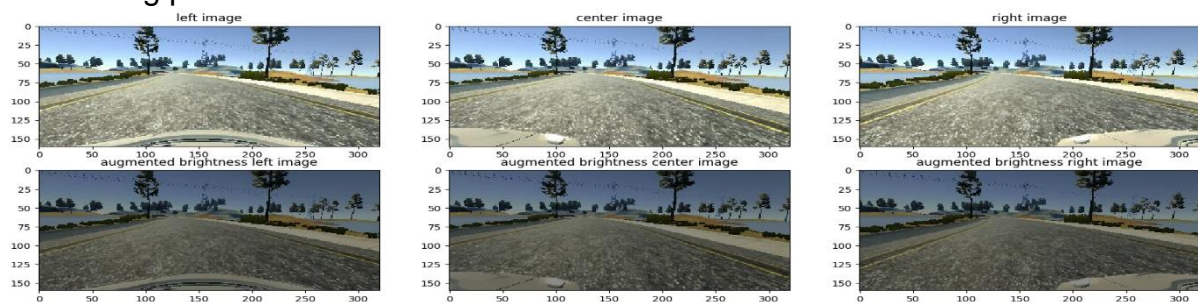
```

BRIGHTNESS

To summarize the weather condition with a sunny day or a cloudy day, or in lowlight condition, the brightness segment is very usual in this case.

The images can be seen like this:

The coding part are as follows:



```

def random_brightness(image):
    brightness = iaa.Multiply((0.2, 1.2))
    image = brightness.augment_image(image)
    return image

image = image_paths[random.randint(0, 1000)]
original_image = mpimg.imread(image)
brightness_altered_image = random_brightness(original_image)
fig, axes = plt.subplots(1, 2, figsize=(15, 10))
fig.tight_layout()
axes[0].imshow(original_image)
axes[0].set_title("Original Image")
axes[1].imshow(brightness_altered_image)
axes[1].set_title("Brightness altered image ")

```

NETWORK -ARCHITECTURE

The design of the network is based on the NVIDIA model, which has been used by NVIDIA for the end-to-end self-driving test. As such, it is well suited for the project. It is a deep convolution network which works well with supervised image classification / regression problems. As the NVIDIA model is well documented, I was able to focus how to adjust the training images to produce the best result with some adjustments to the model to avoid overfitting and adding non-linearity to improve the prediction. I've added the following adjustments to the model.

- I used Lambda layer to normalized input images to avoid saturation and make gradients work better.
- I've added an additional dropout layer to avoid overfitting after the convolution layers.
- I've also included ELU for activation function for every layer except for the output layer to introduce non-linearity. In the end model looks like
- Image normalization
- Convolution: 5x5, filter: 24, strides: 2x2, activation: ELU
- Convolution: 5x5, filter: 36, strides: 2x2, activation: ELU
- Convolution: 5x5, filter: 48, strides: 2x2, activation: ELU
- Convolution: 3x3, filter: 64, strides: 1x1, activation: ELU
- Convolution: 3x3, filter: 64, strides: 1x1, activation: ELU
- Drop out (0.5)
- Fully connected: neurons: 100, activation: ELU

Fully connected: neurons: 50, activation: ELU

- Fully connected: neurons: 10, activation: ELU
- Fully connected: neurons: 1 (output)

As per the NVIDIA model, the convolution layers are meant to handle feature engineering and the fully connected layer for predicting the steering angle. However, as stated in the NVIDIA document, it is not clear where to draw such a clear distinction. Overall, the model is very functional to clone the given steering angle. The below is a model structure output from the Keras which gives more details on the shapes and the number of parameters. We will design our model architecture. We have to classify the traffic signs too that's why we need to shift from Lenet5 model to NVIDIA model. With behavioural cloning, our dataset is much more complex than any dataset we have used. We are dealing with images that have (200,66)

dimensions. Our current dataset has 5386 images to train but MNSIT has around 60,000 images to train with. Our behavioural cloning code has simply had to return appropriate steering angle which is a regression type example. For these things, we need a more advanced model which is provided by NVIDIA and known as NVIDIA model.

For defining the model architecture, we need to define the model object. Normalization state can be skipped as we have already normalized it. We will add the convolution layer. As compared to the model, we will organize accordingly. The NVIDIA model uses 24 filters in the layer along with a kernel of size 5,5. We will introduce sub sampling. The function reflects to stride length of the kernel as it processes through an image, we have large images. Horizontal movement with 2 pixels at a time, similarly vertical movement to 2 pixels at a time. As this is the first layer, we have to define input shape of the model too i.e., (66,200,3) and the last function is an activation function that is "Elu"

Revising the model, we see that our second layer has 36 filters with kernel size (5,5) same subsampling option with stride length of (2,2) and conclude this layer with activation 'Elu'.

According to NVIDIA model, it shows we have 3 more layers in the convolutional neural network. With 48 filters, with 64 filters (3,3) kernel 64 filters (3,3) kernel Dimensions have been reduced significantly so for that we will remove subsampling from 4th and 5th layer.

Next, we add a flatten layer. We will take the output array from previous convolution neural network to convert it into a one-dimensional array so that it can be fed to fully connected layer to follow.

We end the architecture of NVIDIA model with a dense layer containing a single output node which will output the predicted steering angle for our self-driving car. Now we will use function `model.compile()` to compile our architecture as this is a regression type example the metrics that we will be using will be mean squared error and optimize as Adam. We will be using relatively a low learning rate that it can help on accuracy. We will use dropout layer to avoid overfitting the data. Dropout Layer sets the input of random fraction of nodes to "0" during each update. During this, we will generate the training data as it is forced to use a variety of combination of nodes to learn from the same data. We will have to separate the convolution layer from fully connected layer with a factor of 0.5 is added so it converts 50 percent of the input to 0. We Will define the model by calling the NVIDIA model itself. Now we will have the model training process.

As we follow with the code:

ADVANTAGES OF SELF-DRIVING CARS

- 1.Reducing human error
- 2.More independence for the elder persons or disabilities
- 3.Fewer accidents or deaths
- 4.Oppurtunities for more efficient driving
- 5.Traffic efficiency

DISADVANTAGES OF SELF-DRIVING CARS

- 1.Initial cost
- 2.Security issues
- 3.Danger of hacking
- 4.Accidents due to failed software updates and technology glitches

CONCLUSION

In this project a simple demonstration of self-driving is implemented. This project is used to conclude that at how much accuracy the training and testing dataset can be determine using the model and take decisions accordingly, it is also determining the angle of the steering .This we have done using different machine learning techniques with the help of different libraries such as OpenCV, TensorFlow, etc .As our technology is improving we can built several self-driving cars so that our work can be made more faster and easier to overcome the time management. As technology expands throughout the world, self-driving cars will become the future mode of transportation universally.

OVERVIEW

In this project, we use deep neural networks and convolutional neural networks to clone driving behavior. The model is trained, validated and tested using KERAS. The model outputs a steering angle to an autonomous vehicle. The autonomous vehicle is provided as a simulator. Image data and steering angles are used to train a neural network and drive the simulation car autonomously around the track.

This project started with training the models and tweaking parameters to get the best performance on the track.

The use of CNN for getting the spatial features and RNN for the temporal features in the image dataset makes this combination a great fit for building fast and lesser computation required neural networks. Substituting recurrent layers for pooling layers might reduce the loss of information and would be worth exploring in the future projects.

It is interesting to find the use of combinations of real-world dataset and simulator data to train these models. Then I can get the true nature of how a model can be trained in the simulator and generalized to the real world or vice versa. There are many experimental implementations carried out in the field of self-driving cars and this project contributes towards a significant part of it.

REFERENCES

GITHUB-[UDACITY-self-driving-car/term1_project3_behavioral_cloning_at_master · woges/UDACITY-self-driving-car · GitHub](#)

DEEP LEARNING FOR SELF_DRIVING CAR-[Deep Learning for Self-Driving Cars | by Manajit Pal | Towards Data Science](#)

GITHUB-[GitHub - llsourcell/How_to_simulate_a_self_driving_car: This is the code for "How to Simulate a Self-Driving Car" by Siraj Raval on Youtube](#)

PDF-[fx719m76s \(calstate.edu\)](#)