**Python Code (Lane Detection) :**

import keras

from sklearn.model\_selection import train\_test\_split

import numpy as np

from keras.optimizers import Adam

from sklearn.utils import shuffle

import matplotlib.pyplot as plt

from keras.models import Sequential

import ntpath

import matplotlib.image as npimg

import pandas as pd

from keras.layers import Convolution2D, MaxPooling2D, Dropout, Flatten, Dense

import cv2

import os

import random

datadirect = 'Self-Driving-Car'

col = ['center1', 'left1', 'right1', 'steering1', 'throttle1', 'reverse1', 'speed1']

car\_info = pd.read\_csv(os.path.join(datadirect, 'file.csv'), names = col)

pd.set\_option('display.max\_colwidth', -1)

car\_info.head()

def leaf\_path(path):

head1, tail1 = ntpath.split(path)

return tail1

car\_info['center1'] = car\_info['center1'].apply(leaf\_path)

car\_info['left1'] = car\_info['left1'].apply(leaf\_path)

car\_info['right1'] = car\_info['right1'].apply(leaf\_path)

car\_info.head()

number\_of\_cbin = 25

cbin\_sample = 200

histo, cbin = np.histogram(car\_info['steering1'], number\_of\_cbin)

center1 = cbin[:-1] + cbin[1:] \* 0.5

plt.bar(center1, histo, width=0.05)

plt.plot((np.min(car\_info['steering1']), np.max(car\_info['steering1'])), (cbin\_sample,

cbin\_sample))

rem\_list = []

for j in range(number\_of\_cbin):

list1\_ = []

for i in range(len(car\_info['steering1'])):

angle\_steer = car\_info['steering1'][i]

if angle\_steer >= cbin[j] and angle\_steer <= cbin[j+1]:

list1\_.append(i)

list1\_ = shuffle(list1\_)

list1\_ = list1\_[cbin\_sample:]

rem\_list.extend(list1\_)

print('Total car\_info: {0}'.format(len(car\_info)))

car\_info.drop(car\_info.index[rem\_list], inplace=True)

histo, \_ = np.histogram(car\_info['steering1'], (number\_of\_cbin))

plt.bar(center1, histo, width=0.05)

plt.plot((np.min(car\_info['steering1']), np.max(car\_info['steering1'])), (cbin\_sample,

cbin\_sample))

print('Removed: {0}'.format(len(rem\_list)))

print('Remaining: {0}'.format(len(car\_info)))

def image\_steering1(datadirect, df):

path\_to\_image = []

steering1 = []

for i in range(len(car\_info)):

car\_info\_ind = car\_info.iloc[i]

center1, left1, right1 = car\_info\_ind[0], car\_info\_ind[1], car\_info\_ind[2]

path\_to\_image.append(os.path.join(datadirect, center1.strip()))

steering1.append(float(car\_info\_ind[3]))

paths\_to\_image = np.asarray(path\_to\_image)

steering1s = np.asarray(steering1)

return paths\_to\_image, steering1s

paths\_to\_image, steering1s = image\_steering1(datadirect + '/IMG', car\_info)

train\_data\_X, x\_val, train\_data\_Y, y\_val = train\_test\_split(paths\_to\_image, steering1s,

test\_size=0.2, random\_state=0)

print("Training Samples: {}\nValid Samples: {}".format(len(train\_data\_X), len(x\_val)))

fig, axes1 = plt.subplots(1, 2, figsize=(12, 4))

axes1[0].hist(train\_data\_Y, bins=number\_of\_cbin, width=0.05, color='green')

axes1[0].set\_title('Training set')

axes1[1].hist(y\_val, bins=number\_of\_cbin, width=0.05, color='yellow')

axes1[1].set\_title('Validation set')

def image\_preprocess(image):

image = npimg.imread(image)

image = image[60:135, :, :]

image = cv2.cvtColor(image, cv2.COLOR\_RGB2YUV)

image = cv2.GaussianBlur(image, (3, 3), 0)

image = cv2.resize(image, (100, 100))

image = image / 255

return image

images1 = paths\_to\_image[100]

origin\_images1 = npimg.imread(images1)

processed\_images1 = image\_preprocess(images1)

fig, axes1 = plt.subplots(1, 2, figsize=(15, 10))

fig.tight\_layout()

axes1[0].imshow(origin\_images1)

axes1[0].set\_title('Before Preprocess')

axes1[1].imshow(processed\_images1)

axes1[1].set\_title('After Preprocess')

train\_data\_X = np.array(list(map(image\_preprocess, train\_data\_X)))

x\_val = np.array(list(map(image\_preprocess, x\_val)))

plt.imshow(train\_data\_X[random.randint(0, len(train\_data\_X)-1)])

print(train\_data\_X.shape)

from keras.applications import ResNet50

resnet = ResNet50(weights='imagenet', include\_top=False, input\_shape=(100, 100, 3))

for layer in resnet.layers[:-4]:

layer.trainable = False

for layer in resnet.layers:

print(layer, layer.trainable)

def nvidia\_model():

mod\_classification\_lane = Sequential()

mod\_classification\_lane.add(resnet)

mod\_classification\_lane.add(Dropout(0.5))

mod\_classification\_lane.add(Flatten())

mod\_classification\_lane.add(Dense(100, activation='elu'))

mod\_classification\_lane.add(Dropout(0.5))

mod\_classification\_lane.add(Dense(50, activation='elu'))

mod\_classification\_lane.add(Dropout(0.5))

mod\_classification\_lane.add(Dense(10, activation='elu'))

mod\_classification\_lane.add(Dropout(0.5))

mod\_classification\_lane.add(Dense(1))

optimizer = Adam(lr=1e-3)

mod\_classification\_lane.compile(loss='mse', optimizer=optimizer, metrics=['accuracy'])

return mod\_classification\_lane

mod\_classification\_lane = nvidia\_model()

print(mod\_classification\_lane.summary())

history\_mod\_classification\_lane = mod\_classification\_lane.fit(train\_data\_X, train\_data\_Y,

epochs=25, validation\_data=(x\_val, y\_val), batch\_size=128, verbose=1, shuffle=1)

plt.plot(history\_mod\_classification\_lane.history['loss'])

plt.plot(history\_mod\_classification\_lane.history['val\_loss'])

plt.legend(['training', 'validation'])

plt.title('Loss')

plt.xlabel('Epoch')