

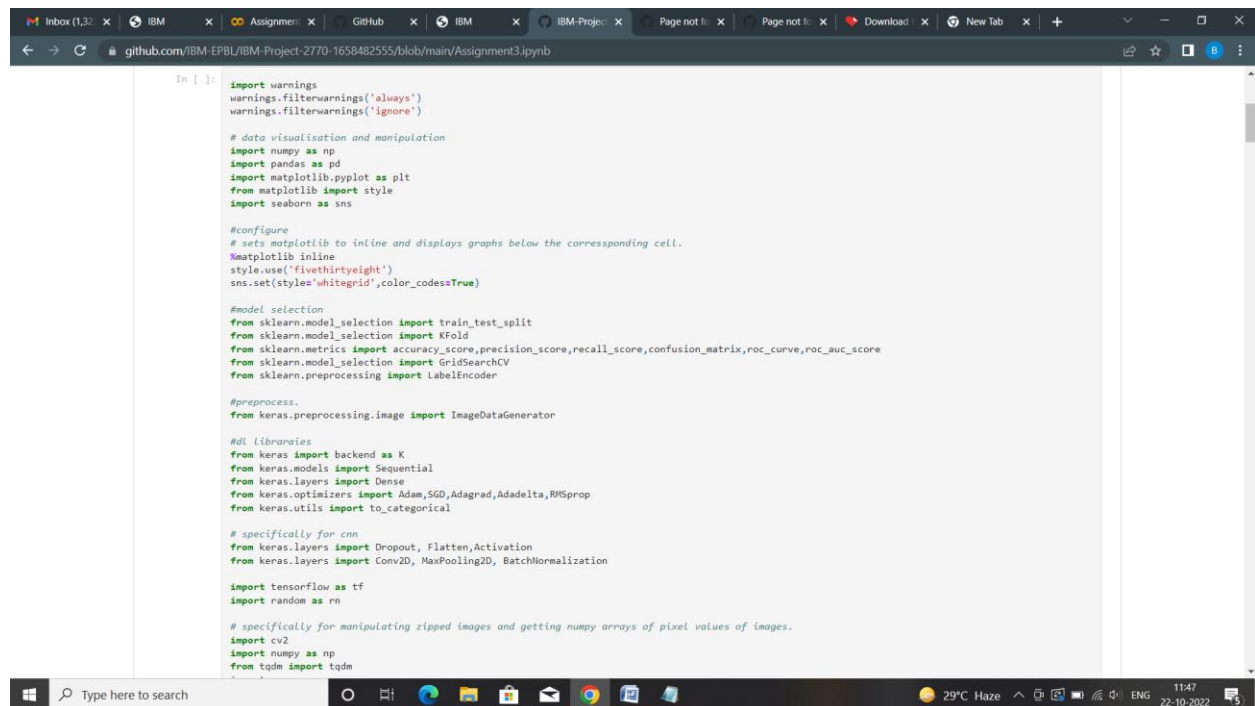
Team No:-PNT2022TMID08666

Assignment 3

Student Name	A.B.Brindha
Student Roll Number	19BCS051
Maximum Marks	2 Marks

Problem Statement :- Build CNN Model for Classification Of Flowers

Screenshots:-



```
In [ ]: import warnings
warnings.filterwarnings('always')
warnings.filterwarnings('ignore')

# data visualisation and manipulation
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from matplotlib import style
import seaborn as sns

#configure
# sets matplotlib to inline and displays graphs below the corresponding cell.
%matplotlib inline
style.use('fivethirtyeight')
sns.set(styles='whitegrid',color_codes=True)

#model selection
from sklearn.model_selection import train_test_split
from sklearn.model_selection import KFold
from sklearn.metrics import accuracy_score,precision_score,recall_score,confusion_matrix,roc_curve,roc_auc_score
from sklearn.model_selection import GridSearchCV
from sklearn.preprocessing import LabelEncoder

#preprocess.
from keras.preprocessing.image import ImageDataGenerator

#DL Libraries
from keras import backend as K
from keras.models import Sequential
from keras.layers import Dense
from keras.optimizers import Adam,SGD,Adagrad,Adadelta,RMSprop
from keras.utils import to_categorical

# specifically for cnn
from keras.layers import Dropout, Flatten,Activation
from keras.layers import Conv2D, MaxPooling2D, BatchNormalization

import tensorflow as tf
import random as rn

# specifically for manipulating zipped images and getting numpy arrays of pixel values of images.
import cv2
import numpy as np
from tqdm import tqdm
```

```
from PIL import Image

In [ ]: import os
print(os.listdir('/content/drive/MyDrive/flowers'))

['daisy', 'dandelion', 'rose', 'sunflower', 'tulip']

In [ ]: from google.colab import drive
drive.mount('/content/drive')

Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).

In [ ]: X=[]
Z=[]
IMG_SIZE=150
FLOWER_DAISY_DIR="/content/drive/MyDrive/flowers/daisy"
FLOWER_SUNFLOWER_DIR="/content/drive/MyDrive/flowers/sunflower"
FLOWER_TULIP_DIR="/content/drive/MyDrive/flowers/tulip"
FLOWER_DANDI_DIR="/content/drive/MyDrive/flowers/dandelion"
FLOWER_ROSE_DIR="/content/drive/MyDrive/flowers/rose"

In [ ]: def assign_label(img, flower_type):
return flower_type

In [ ]: def make_train_data(flower_type, DIR):
for img in tqdm(os.listdir(DIR)):
label=assign_label(img, flower_type)
path = os.path.join(DIR, img)
img = cv2.imread(path, cv2.IMREAD_COLOR)
img = cv2.resize(img, (IMG_SIZE, IMG_SIZE))

X.append(np.array(img))
Z.append(str(label))

In [ ]: pip install tqdm

Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Requirement already satisfied: tqdm in /usr/local/lib/python3.7/dist-packages (4.64.1)
```

```
In [ ]: import tqdm

In [ ]: make_train_data('Daisy', FLOWER_DAISY_DIR)
print(len(X))

0it [00:00, ?it/s]
0

In [ ]: make_train_data('Sunflower', FLOWER_SUNFLOWER_DIR)
print(len(X))

100% [██████████] 66/66 [00:16<00:00, 4.06it/s]
66

In [ ]: make_train_data('Tulip', FLOWER_TULIP_DIR)
print(len(X))

100% [██████████] 984/984 [00:17<00:00, 57.00it/s]
1050

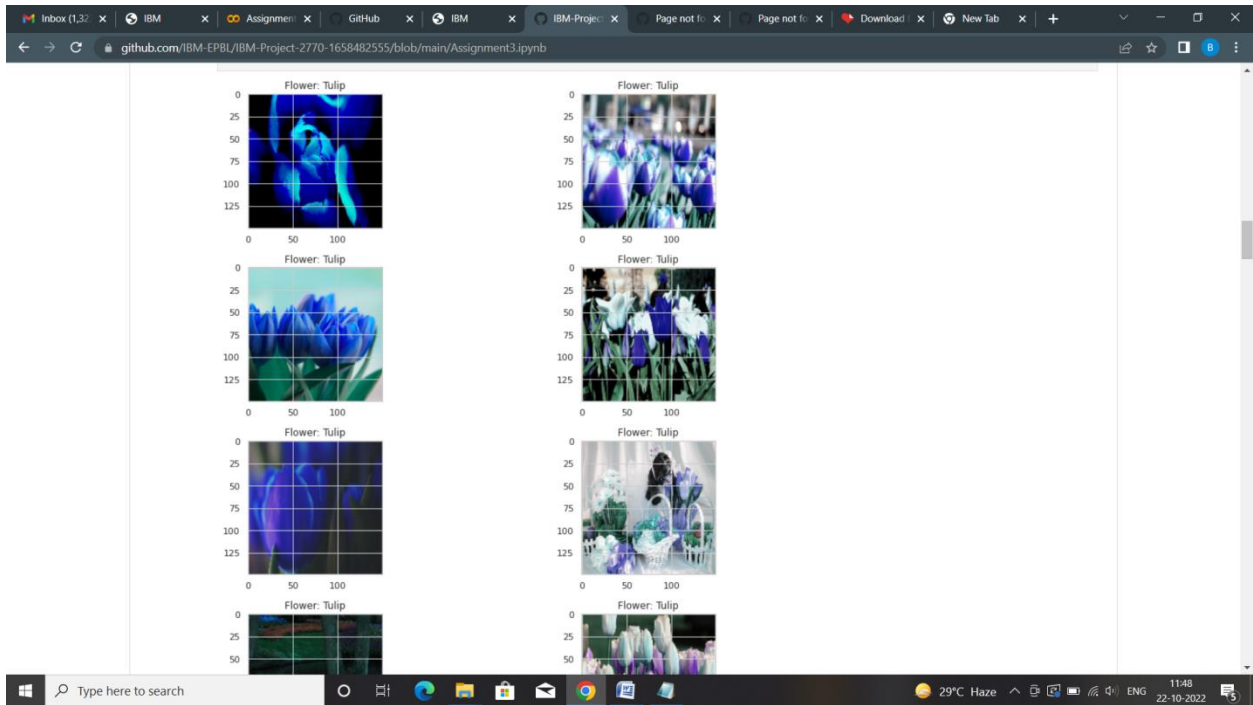
In [ ]: make_train_data('Dandelion', FLOWER_DANDI_DIR)
print(len(X))

0it [00:00, ?it/s]
1050

In [ ]: make_train_data('Rose', FLOWER_ROSE_DIR)
print(len(X))

0it [00:00, ?it/s]
1050

In [ ]: fig, ax=plt.subplots(5,2)
fig.set_size_inches(15,15)
for i in range(5):
```



```

In [ ]:
le=LabelEncoder()
Y=le.fit_transform(Z)
Y=to_categorical(Y,5)
X=np.array(X)
X=X/255

In [ ]:
x_train,x_test,y_train,y_test=train_test_split(X,Y,test_size=0.25,random_state=42)

In [ ]:
import tensorflow

In [ ]:
np.random.seed(42)
rn.seed(42)
tf.random.set_seed(42)

In [ ]:
model = Sequential()
model.add(Conv2D(filters = 32, kernel_size = (5,5),padding = 'Same',activation = 'relu', input_shape = (150,150,3)))
model.add(MaxPooling2D(pool_size=(2,2)))

model.add(Conv2D(filters = 64, kernel_size = (3,3),padding = 'Same',activation = 'relu'))
model.add(MaxPooling2D(pool_size=(2,2), strides=(2,2)))

model.add(Conv2D(filters = 96, kernel_size = (3,3),padding = 'Same',activation = 'relu'))
model.add(MaxPooling2D(pool_size=(2,2), strides=(2,2)))

model.add(Conv2D(filters = 96, kernel_size = (3,3),padding = 'Same',activation = 'relu'))
model.add(MaxPooling2D(pool_size=(2,2), strides=(2,2)))

model.add(Flatten())
model.add(Dense(512))
model.add(Activation('relu'))
model.add(Dense(5, activation = "softmax"))

In [ ]:
batch_size=128
epochs=50

from keras.callbacks import ReduceLRonPlateau

```

```
model.summary()

Model: "sequential_2"
-----
Layer (type)                 Output Shape              Param #
-----
conv2d_8 (Conv2D)            (None, 150, 150, 32)      2432
max_pooling2d_8 (MaxPooling2D) (None, 75, 75, 32)        0
conv2d_9 (Conv2D)            (None, 75, 75, 64)        18496
max_pooling2d_9 (MaxPooling2D) (None, 37, 37, 64)        0
conv2d_10 (Conv2D)           (None, 37, 37, 96)        55392
max_pooling2d_10 (MaxPooling2D) (None, 18, 18, 96)        0
conv2d_11 (Conv2D)           (None, 18, 18, 96)        83040
max_pooling2d_11 (MaxPooling2D) (None, 9, 9, 96)          0
flatten_2 (Flatten)          (None, 7776)              0
dense_4 (Dense)              (None, 512)               3981824
activation_2 (Activation)     (None, 512)               0
dense_5 (Dense)              (None, 5)                 2565
-----
Total params: 4,143,749
Trainable params: 4,143,749
Non-trainable params: 0

In [ ]: History = model.fit_generator(datagen.flow(x_train,y_train, batch_size=batch_size),
epochs = epochs, validation_data = (x_test,y_test),
verbose = 1, steps_per_epoch=x_train.shape[0] // batch_size)

Epoch 1/50
```

```
Epoch 12/50
6/6 [=====] - 47s 8s/step - loss: 0.1775 - accuracy: 0.9408 - val_loss: 0.1784 - val_accuracy: 0.9316
Epoch 13/50
6/6 [=====] - 49s 8s/step - loss: 0.1779 - accuracy: 0.9378 - val_loss: 0.1735 - val_accuracy: 0.9316
Epoch 14/50
6/6 [=====] - 47s 9s/step - loss: 0.1621 - accuracy: 0.9378 - val_loss: 0.1545 - val_accuracy: 0.9316
Epoch 15/50
6/6 [=====] - 46s 9s/step - loss: 0.1675 - accuracy: 0.9469 - val_loss: 0.1890 - val_accuracy: 0.9392
Epoch 16/50
6/6 [=====] - 48s 8s/step - loss: 0.1829 - accuracy: 0.9347 - val_loss: 0.1490 - val_accuracy: 0.9316
Epoch 17/50
6/6 [=====] - 46s 8s/step - loss: 0.1610 - accuracy: 0.9347 - val_loss: 0.1456 - val_accuracy: 0.9430
Epoch 18/50
6/6 [=====] - 46s 7s/step - loss: 0.1472 - accuracy: 0.9469 - val_loss: 0.1417 - val_accuracy: 0.9430
Epoch 19/50
6/6 [=====] - 46s 7s/step - loss: 0.1527 - accuracy: 0.9393 - val_loss: 0.1300 - val_accuracy: 0.9544
Epoch 20/50
6/6 [=====] - 49s 10s/step - loss: 0.1564 - accuracy: 0.9454 - val_loss: 0.1153 - val_accuracy: 0.9620
Epoch 21/50
6/6 [=====] - 50s 8s/step - loss: 0.1380 - accuracy: 0.9499 - val_loss: 0.1192 - val_accuracy: 0.9582
Epoch 22/50
6/6 [=====] - 48s 8s/step - loss: 0.1147 - accuracy: 0.9530 - val_loss: 0.1045 - val_accuracy: 0.9582
Epoch 23/50
6/6 [=====] - 54s 10s/step - loss: 0.1589 - accuracy: 0.9408 - val_loss: 0.1502 - val_accuracy: 0.9468
Epoch 24/50
6/6 [=====] - 46s 8s/step - loss: 0.1478 - accuracy: 0.9484 - val_loss: 0.1226 - val_accuracy: 0.9696
Epoch 25/50
6/6 [=====] - 46s 7s/step - loss: 0.1250 - accuracy: 0.9499 - val_loss: 0.1906 - val_accuracy: 0.9354
Epoch 26/50
6/6 [=====] - 53s 9s/step - loss: 0.1437 - accuracy: 0.9453 - val_loss: 0.1359 - val_accuracy: 0.9620
Epoch 27/50
6/6 [=====] - 54s 9s/step - loss: 0.1358 - accuracy: 0.9518 - val_loss: 0.1229 - val_accuracy: 0.9468
Epoch 28/50
6/6 [=====] - 46s 9s/step - loss: 0.1220 - accuracy: 0.9530 - val_loss: 0.1085 - val_accuracy: 0.9620
Epoch 29/50
6/6 [=====] - 48s 8s/step - loss: 0.1402 - accuracy: 0.9439 - val_loss: 0.0874 - val_accuracy: 0.9658
Epoch 30/50
6/6 [=====] - 53s 9s/step - loss: 0.1189 - accuracy: 0.9440 - val_loss: 0.0897 - val_accuracy: 0.9582
Epoch 31/50
6/6 [=====] - 47s 8s/step - loss: 0.1192 - accuracy: 0.9499 - val_loss: 0.1170 - val_accuracy: 0.9696
Epoch 32/50
6/6 [=====] - 46s 7s/step - loss: 0.1149 - accuracy: 0.9560 - val_loss: 0.1140 - val_accuracy: 0.9506
Epoch 33/50
6/6 [=====] - 50s 8s/step - loss: 0.1229 - accuracy: 0.9560 - val_loss: 0.1043 - val_accuracy: 0.9696
Epoch 34/50
6/6 [=====] - 46s 8s/step - loss: 0.1212 - accuracy: 0.9575 - val_loss: 0.0964 - val_accuracy: 0.9696
```

