Codes to train VGG-16 model for detection of early leaf spot of groundnut

Step 1: Mounting Google Drive in Google Colab

from google.colab import drive

drive.mount('/content/gdrive')

Step 2: Importing required libraries

import os

import cv2

import numpy as np

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

import shutil

import matplotlib.pyplot as plt

%matplotlib inline

from tensorflow.keras.preprocessing.image import load\_img, img\_to\_array, array\_to\_img,

ImageDataGenerator

from keras.applications.vgg16 import VGG16

from keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout, InputLayer

from keras.models import Sequential

from keras.layers import BatchNormalization

from keras import optimizers

Step 3: Setting data path

main\_path = '/content/gdrive/MyDrive/Groundnut'

groundnut\_healthy = main\_path + '/Gnut healthy/'

groundnut\_leaf\_spot = main\_path + '/Gnut leaf spot/'

groundnut\_healthy\_data = [main\_path+'/Gnut healthy/'+f for f in os.listdir(groundnut\_healthy)]

groundnut\_leaf\_spot\_data = [main\_path+'/Gnut leaf spot/'+f for f in os.listdir(groundnut\_leaf\_spot)]

len(groundnut\_healthy\_data),len(groundnut\_leaf\_spot\_data)

Step 4: Print image

print(groundnut\_healthy\_data[0:5])

print(groundnut\_leaf\_spot\_data[0:5])

Step 5: Assign labels to the image dataset classes

images = np.array(groundnut\_healthy\_data + groundnut\_leaf\_spot\_data)

labels = np.array([0]\*len(groundnut\_healthy\_data)+[1]\*len(groundnut\_leaf\_spot\_data)).astype('float32')

Step 6: Dataset split into training, testing and validation dataset

validation\_ratio = 0.2

X\_train, X\_test, y\_train, y\_test = train\_test\_split(images,labels,test\_size=0.33)

X\_train, X\_validation, y\_train, y\_validation = train\_test\_split(X\_train,y\_train,test\_size=validation\_ratio)

X\_train.shape,X\_test.shape,y\_train.shape, y\_test.shape

len(X\_train),len(X\_test),len(y\_train),len(y\_test)

Step 7: Import modules/functions from TensorFlow.Keras library for data augmentation

from tensorflow.keras.preprocessing.image import load\_img, img\_to\_array, array\_to\_img, ImageDataGenerator

Step 8: Set image dimensions to resize images into uniform size

IMG\_WIDTH=224

IMG\_HEIGHT=224

IMG\_DIM = (IMG\_WIDTH, IMG\_HEIGHT)

Step 9: Image Pre-Processing

X\_train = np.array([img\_to\_array(load\_img(img, target\_size=IMG\_DIM)).astype('float32')/255 for img in X\_train])

X\_test = np.array([img\_to\_array(load\_img(img, target\_size=IMG\_DIM)).astype('float32')/255  for img in X\_test])

X\_validation = np.array([img\_to\_array(load\_img(img,target\_size=IMG\_DIM)).astype('float32')/255 for img in X\_validation])

Step 10: Preparing the data using ImageDataGenerator

dataGen = ImageDataGenerator(width\_shift\_range=0.2,

                             # 0.1 = 10%     IF MORE THAN 1 E.G 10 THEN IT REFFERS TO NO. OF  PIXELS EG 10 PIXELS

                             height\_shift\_range=0.6,

                             zoom\_range=0.3,  # 0.2 MEANS CAN GO FROM 0.8 TO 1.2

                             shear\_range=0.4,  # MAGNITUDE OF SHEAR ANGLE

                             rotation\_range=10)  # DEGREES

dataGen.fit(X\_train)

batches = dataGen.flow(X\_train, y\_train,batch\_size=20)  # REQUESTING DATA GENRATOR TO GENERATE IMAGES  BATCH SIZE = NO. OF IMAGES CREAED EACH TIME ITS CALLED

X\_batch, y\_batch = next(batches)

Step 11: Convert Numpy array into an image

array\_to\_img(X\_train[0])

print(X\_train[:10],y\_train[-10:])

Step 12: Modify VGG-16 model

from keras.models import Model

vgg = VGG16(include\_top=False, weights='imagenet', input\_shape = (IMG\_HEIGHT,IMG\_WIDTH,3))

output1 = vgg.layers[-1].output

output2 = Flatten()(output1)

vgg = Model(vgg.input, output2)

for layer in vgg.layers:

    layer.trainable = False

vgg.summary()

Step 13: Rebuild the model using ‘sequential’ API in keras

model = Sequential()

model.add(vgg)

input\_shape = (IMG\_HEIGHT,IMG\_WIDTH,3)

model.add(Flatten())

model.add(Dropout(0.3))

model.add(Dense(1, activation='sigmoid'))

Step 14: Update the model architecture

flatten\_layer = Flatten()

dense\_layer\_1 = Dense(500, activation='relu')

dense\_layer\_2 = Dense(500, activation='relu')

dropout\_layer = Dropout(0.2)

prediction\_layer = Dense(1, activation='sigmoid')

model = Sequential([

    vgg,

    flatten\_layer,

    dense\_layer\_1,

    dense\_layer\_2,

    dropout\_layer,

    prediction\_layer

])

Step 15: Compile the model

model.compile(loss='binary\_crossentropy', optimizer=optimizers.Adam(learning\_rate=0.001), metrics=['accuracy'])

model.summary()

Step 16: Model training

history = model.fit(dataGen.flow(X\_train, y\_train.reshape((-1,1))),

epochs=100,

validation\_data=(X\_validation,y\_validation),

shuffle=True,

verbose=1)

Step 17: Analyze the model performance

accuracy = history.history['accuracy']

val\_accuracy = history.history['val\_accuracy']

loss = history.history['loss']

val\_loss = history.history['val\_loss']

Step 18: Plot training and validation accuracy over epochs

epochs = range(len(accuracy))

plt.plot(epochs, accuracy, label='Training Accuracy')

plt.plot(epochs, val\_accuracy, 'b', label='Validation Accuracy')

plt.title('Training and Validation Accuracy')

plt.legend()

Step 19: Plot training and validation loss over epochs

epochs = range(len(loss))

plt.plot(epochs, loss, label='Training loss')

plt.plot(epochs, val\_loss, 'b', label='Validation loss')

plt.title('Training and Validation Loss')

plt.legend()

Step 20: Model accuracy on test set

score = model.evaluate(X\_test,y\_test)

print('Test Accuracy: {}'.format(score[1]))

Step 21: Make predictions on test set

predict = model.predict(X\_test)

predict

Step 22: Save and load model to Google drive using TensorFlow

import tensorflow as tf

#model.save('/content/gdrive/MyDrive/CNN/groundnutmodel.h5')

model = tf.keras.models.load\_model('/content/gdrive/MyDrive/CNN/groundnutmodel.h5')

Step 23: Create a new array ‘x’ based on the predicted values and print first 10 elements of ‘x’ array and ‘x\_test’ array

x = []

for i in predict:

if i<=0.5:

x.append(0)

else:

x.append(1)

x = np.array(x).astype('float32')

x[0:10]

X\_test[0:10]

Step 24: Import accuracy function from ‘sklearn.metrics’

from sklearn.metrics import accuracy\_score

Step 25: Calculate the accuracy score between the predicted labels (x) and the true labels

(y\_test) using the ‘accuracy\_score’ function

acc = accuracy\_score(x,y\_test)

print('Accuracy Score : ',acc\*100)

Step 26: Install Gradio library

!pip install gradio

import gradio as gr

import cv2

import numpy as np

Step 27: Make predictions on new input dataset

def groundnut\_disease(img):

img = np.asarray(img)

img = cv2.resize(img,(224,224))

img = img.reshape(1,224,224,3)

while True:

predict = model.predict(img)

if predict<=0.5:

return 'Healthy crop'

elif predict>=0.5:

return 'Leaf spot'

Step 28: Create Gradio interface for the developed model

#outputs = gr.output.Textbox()

app = gr.Interface(fn=groundnut\_disease,inputs="image",outputs="text",description='This is

Ground disease classification model')

Step 29: Launch the gradio interface and use the model for detection of early leaf spot of

groundnut

app.launch(debug=True)