**1.SAMPLE CODING**

**main.py**

import cv2

import numpy as np

import argparse, sys, os

from GUIdriver import \*

import pandas as pd

def endprogram():

print ("\nProgram terminated!")

sys.exit()

#Reading the image by parsing the argument

text = str(ImageFile)

print ("\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\nImage : " + ImageFile + "\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

img = cv2.imread(text)

img = cv2.resize(img ,((int)(img.shape[1]/5),(int)(img.shape[0]/5)))

original = img.copy()

neworiginal = img.copy()

cv2.imshow('original',img)

#Calculating number of pixels with shade of white(p) to check if exclusion of these pixels is required or not (if more than a fixed %) in order to differentiate the white background or white patches in image caused by flash, if present.

p = 0

for i in range(img.shape[0]):

for j in range(img.shape[1]):

B = img[i][j][0]

G = img[i][j][1]

R = img[i][j][2]

if (B > 110 and G > 110 and R > 110):

p += 1

#finding the % of pixels in shade of white

totalpixels = img.shape[0]\*img.shape[1]

per\_white = 100 \* p/totalpixels

'''

print 'percantage of white: ' + str(per\_white) + '\n'

print 'total: ' + str(totalpixels) + '\n'

print 'white: ' + str(p) + '\n'

'''

#excluding all the pixels with colour close to white if they are more than 10% in the image

if per\_white > 10:

img[i][j] = [200,200,200]

cv2.imshow('color change', img)

#Guassian blur

blur1 = cv2.GaussianBlur(img,(3,3),1)

#mean-shift algo

newimg = np.zeros((img.shape[0], img.shape[1],3),np.uint8)

criteria = (cv2.TERM\_CRITERIA\_EPS + cv2.TERM\_CRITERIA\_MAX\_ITER , 10 ,1.0)

img = cv2.pyrMeanShiftFiltering(blur1, 20, 30, newimg, 0, criteria)

cv2.imshow('means shift image',img)

#Guassian blur

blur = cv2.GaussianBlur(img,(11,11),1)

#Canny-edge detection

canny = cv2.Canny(blur, 160, 290)

canny = cv2.cvtColor(canny,cv2.COLOR\_GRAY2BGR)

#creating border around image to close any open curve cut by the image border

#bordered = cv2.copyMakeBorder(canny,10,10,10,10, cv2.BORDER\_CONSTANT, (255,255,255)) #function not working(not making white coloured border)

#bordered = cv2.rectangle(canny,(-2,-2),(275,183),(255,255,255),3)

#cv2.imshow('Canny on meanshift bordered image',bordered)

#contour to find leafs

bordered = cv2.cvtColor(canny,cv2.COLOR\_BGR2GRAY)

contours,hierarchy = cv2.findContours(bordered, cv2.RETR\_TREE, cv2.CHAIN\_APPROX\_NONE)

maxC = 0

for x in range(len(contours)): #if take max or one less than max then will not work in

if len(contours[x]) > maxC: # pictures with zoomed leaf images

maxC = len(contours[x])

maxid = x

perimeter = cv2.arcLength(contours[maxid],True)

#print perimeter

Tarea = cv2.contourArea(contours[maxid])

cv2.drawContours(neworiginal,contours[maxid],-1,(0,0,255))

cv2.imshow('Contour',neworiginal)

#cv2.imwrite('Contour complete leaf.jpg',neworiginal)

#Creating rectangular roi around contour

height, width, \_ = canny.shape

min\_x, min\_y = width, height

max\_x = max\_y = 0

frame = canny.copy()

# computes the bounding box for the contour, and draws it on the frame,

for contour, hier in zip(contours, hierarchy):

(x,y,w,h) = cv2.boundingRect(contours[maxid])

min\_x, max\_x = min(x, min\_x), max(x+w, max\_x)

min\_y, max\_y = min(y, min\_y), max(y+h, max\_y)

if w > 80 and h > 80:

#cv2.rectangle(frame, (x,y), (x+w,y+h), (255, 0, 0), 2) #we do not draw the rectangle as it interferes with contour later on

roi = img[y:y+h , x:x+w]

originalroi = original[y:y+h , x:x+w]

if (max\_x - min\_x > 0 and max\_y - min\_y > 0):

roi = img[min\_y:max\_y , min\_x:max\_x]

originalroi = original[min\_y:max\_y , min\_x:max\_x]

#cv2.rectangle(frame, (min\_x, min\_y), (max\_x, max\_y), (255, 0, 0), 2) #we do not draw the rectangle as it interferes with contour

cv2.imshow('ROI', frame)

cv2.imshow('rectangle ROI', roi)

img = roi

#Changing colour-space

#imghsv = cv2.cvtColor(img, cv2.COLOR\_BGR2HSV)

imghls = cv2.cvtColor(roi, cv2.COLOR\_BGR2HLS)

cv2.imshow('HLS', imghls)

imghls[np.where((imghls==[30,200,2]).all(axis=2))] = [0,200,0]

cv2.imshow('new HLS', imghls)

#Only hue channel

huehls = imghls[:,:,0]

cv2.imshow('img\_hue hls',huehls)

#ret, huehls = cv2.threshold(huehls,2,255,cv2.THRESH\_BINARY)

huehls[np.where(huehls==[0])] = [35]

cv2.imshow('img\_hue with my mask',huehls)

#Thresholding on hue image

ret, thresh = cv2.threshold(huehls,28,255,cv2.THRESH\_BINARY\_INV)

cv2.imshow('thresh', thresh)

#Masking thresholded image from original image

mask = cv2.bitwise\_and(originalroi,originalroi,mask = thresh)

cv2.imshow('masked out img',mask)

#Finding contours for all infected regions

contours,heirarchy = cv2.findContours(thresh, cv2.RETR\_TREE, cv2.CHAIN\_APPROX\_NONE)

Infarea = 0

for x in range(len(contours)):

cv2.drawContours(originalroi,contours[x],-1,(0,0,255))

cv2.imshow('Contour masked',originalroi)

#Calculating area of infected region

Infarea += cv2.contourArea(contours[x])

if Infarea > Tarea:

Tarea = img.shape[0]\*img.shape[1]

print ('\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n Perimeter: %.2f' %(perimeter)

+ '\n\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_')

print ('\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n Total area: %.2f' %(Tarea)

+ '\n\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_')

#Finding the percentage of infection in the leaf

print ('\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n Infected area: %.2f' %(Infarea)

+ '\n\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_')

try:

per = 100 \* Infarea/Tarea

except ZeroDivisionError:

per = 0

print ('\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n Percentage of infection region: %.2f' %(per)

+ '\n\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_')

print("\n\*To terminate press and hold (q)\*")

cv2.imshow('orig',original)

"""\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*update dataset\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"""

#Updating a dataset file to maintain log of the leaf images identified.

print("\nDo you want to run the classifier(Y/N):")

n = cv2.waitKey(0) & 0xFF

if n == ord('q' or 'Q'):

endprogram()

#import csv file library

import csv

directory = 'datasetlog'

filename = directory+'/Datasetunlabelledlog.csv'

imgid = "/".join(text.split('/')[-2:])

while True:

if n == ord('y'or'Y'):

fieldnames = ['fold num', 'imgid', 'feature1', 'feature2', 'feature3']

print ('Appending to ' + str(filename)+ '...')

try:

log = pd.read\_csv(filename)

logfn = int(log.tail(1)['fold num'])

foldnum = (logfn+1)%10

L = [str(foldnum), imgid, str(Tarea), str(Infarea), str(perimeter)]

my\_df = pd.DataFrame([L])

my\_df.to\_csv(filename, mode='a', index=False, header=False)

print ('\nFile ' + str(filename)+ ' updated!' )

except IOError:

if directory not in os.listdir():

os.system('mkdir ' + directory)

foldnum = 0

L = [str(foldnum), imgid, str(Tarea), str(Infarea), str(perimeter)]

my\_df = pd.DataFrame([fieldnames, L])

my\_df.to\_csv(filename, index=False, header=False)

print ('\nFile ' + str(filename)+ ' updated!' )

finally:

import classifier

endprogram()

elif n == ord('n' or 'N') :

print ('File not updated! \nSuccessfully terminated!')

break

else:

print ('invalid input!')

break

**leaf.py**

#Import necessary libraries

from flask import Flask, render\_template, request

import numpy as np

import os

from tensorflow.keras.preprocessing.image import load\_img

from tensorflow.keras.preprocessing.image import img\_to\_array

from tensorflow.keras.models import load\_model

filepath = r'D:\plant leaf\SOURCE CODE\SOURCE CODE\Source code/model.h5'

model = load\_model(filepath)

print(model)

print("Model Loaded Successfully")

def pred\_tomato\_dieas(tomato\_plant):

test\_image = load\_img(tomato\_plant, target\_size = (128, 128)) # load image

print("@@ Got Image for prediction")

test\_image = img\_to\_array(test\_image)/255 # convert image to np array and normalize

test\_image = np.expand\_dims(test\_image, axis = 0) # change dimention 3D to 4D

result = model.predict(test\_image) # predict diseased palnt or not

print('@@ Raw result = ', result)

pred = np.argmax(result, axis=1)

print(pred)

if pred==0:

return "Tomato - Bacteria Spot Disease", 'Tomato-Bacteria Spot.html'

elif pred==1:

return "Tomato - Early Blight Disease", 'Tomato-Early\_Blight.html'

elif pred==2:

return "Tomato - Healthy and Fresh", 'Tomato-Healthy.html'

elif pred==3:

return "Tomato - Late Blight Disease", 'Tomato - Late\_blight.html'

elif pred==4:

return "Tomato - Leaf Mold Disease", 'Tomato - Leaf\_Mold.html'

elif pred==5:

return "Tomato - Septoria Leaf Spot Disease", 'Tomato - Septoria\_leaf\_spot.html'

elif pred==6:

return "Tomato - Target Spot Disease", 'Tomato - Target\_Spot.html'

elif pred==7:

return "Tomato - Tomoato Yellow Leaf Curl Virus Disease", 'Tomato - Tomato\_Yellow\_Leaf\_Curl\_Virus.html'

elif pred==8:

return "Tomato - Tomato Mosaic Virus Disease", 'Tomato - Tomato\_mosaic\_virus.html'

elif pred==9:

return "Tomato - Two Spotted Spider Mite Disease", 'Tomato - Two-spotted\_spider\_mite.html'

# Create flask instance

app = Flask(\_\_name\_\_)

# render index.html page

@app.route("/", methods=['GET', 'POST'])

def home():

return render\_template('index.html')

# get input image from client then predict class and render respective .html page for solution

@app.route("/predict", methods = ['GET','POST'])

def predict():

if request.method == 'POST':

file = request.files['image'] # fet input

filename = file.filename

print("@@ Input posted = ", filename)

file\_path = os.path.join(r'D:\plant leaf\SOURCE CODE\SOURCE CODE\Source code\static/upload', filename)

file.save(file\_path)

print("@@ Predicting class......")

pred, output\_page = pred\_tomato\_dieas(tomato\_plant=file\_path)

return render\_template(output\_page, pred\_output = pred, user\_image = file\_path)

# For local system & cloud

if \_\_name\_\_ == "\_\_main\_\_":

app.debug = True

app.run(threaded=False,port=8000)