FINAL CODE

SOURCE CODE

```
import tensorflow as tf
import time
import numpy as np
import os

start = time.time()
#try:
# Total iterations
final_iter = 1000

# Assign the batch value
batch_size = 20

# 20% of the data will automatically be used for validation
validation_size = 0.2
img_size = 128
num_channels = 3
train_path = r'data\Train'

# Prepare input data
if not os.path.exists(train_path):
print("No such directory")
raise Exception
classes = os.listdir(train_path)
num_classes = len(classes)

# We shall load all the training and validation images and labels into memory
using openCV and use that during training
data = dataset.read_train_sets(train_path, img_size, classes,
validation_size=validation_size)

# Display the stats
print("Complete reading input data. Will Now print a snippet of it")
print("Number of files in Training-set:\t\t{}".format(len(data.train.labels)))
print("Number of files in Validation-set:\t{}".format(len(data.valid.labels)))
session = tf.compat.v1.Session()
x = tf.compat.v1.placeholder(tf.float32, shape=[None, img_size, img_size,
num_channels], name='x')

## labels
y_true = tf.compat.v1.placeholder(tf.float32, shape=[None, num_classes],
name='y_true')
y_true_cls = tf.argmax(y_true, dimension=1)

##Network graph params
filter_size_conv1 = 3
num_filters_conv1 = 32

filter_size_conv2 = 3
```

```
num_filters_conv2 = 32

filter_size_conv3 = 3
num_filters_conv3 = 64

fc_layer_size = 128


def create_weights(shape):
return tf.Variable(tf.random.truncated_normal(shape, stddev=0.05))


def create_biases(size):
return tf.Variable(tf.constant(0.05, shape=[size]))



def make_generator_model(input,
                               num_input_channels,
                               conv_filter_size,
                               num_filters):
## We shall define the weights that will be trained using create_weights function.
weights = create_weights(shape=[conv_filter_size, conv_filter_size,
num_input_channels, num_filters])
## We create biases using the create_biases function. These are also trained.
biases = create_biases(num_filters)

## Creating the convolutional layer
layer = tf.nn.conv2d(input=input,
filter=weights,
strides=[1, 1, 1, 1],
padding='SAME')

    layer += biases

## We shall be using max-pooling.
layer = tf.nn.max_pool(value=layer,
ksize=[1, 2, 2, 1],
strides=[1, 2, 2, 1],
padding='SAME')
## Output of pooling is fed to Relu which is the activation function for us.
layer = tf.nn.relu(layer)

return layer


# Function to create a Flatten Layer
def create_flatten_layer(layer):
# We know that the shape of the layer will be [batch_size img_size img_size
num_channels]
    # But let's get it from the previous layer.
layer_shape = layer.get_shape()

## Number of features will be img_height * img_width* num_channels. But we shall
calculate it in place of hard-coding it.
num_features = layer_shape[1:4].num_elements()

## Now, we Flatten the layer so we shall have to reshape to num_features
layer = tf.reshape(layer, [-1, num_features])
```

```
return layer


# Function to create a Fully - Connected Layer
def create_fc_layer(input,
                    num_inputs,
                    num_outputs,
                    use_relu=True):
# Let's define trainable weights and biases.
weights = create_weights(shape=[num_inputs, num_outputs])
    biases = create_biases(num_outputs)

# Fully connected layer takes input x and produces wx+b.Since, these are matrices,
we use matmul function in Tensorflow
layer = tf.matmul(input, weights) + biases
if use_relu:
        layer = tf.nn.relu(layer)

return layer


# Create all the layers
layer_conv1 = make_generator_model(input=x,
num_input_channels=num_channels,
conv_filter_size=filter_size_conv1,
num_filters=num_filters_conv1)
layer_conv2 = make_generator_model(input=layer_conv1,
num_input_channels=num_filters_conv1,
conv_filter_size=filter_size_conv2,
num_filters=num_filters_conv2)

layer_conv3 = make_generator_model(input=layer_conv2,
num_input_channels=num_filters_conv2,
conv_filter_size=filter_size_conv3,
num_filters=num_filters_conv3)

layer_flat = create_flatten_layer(layer_conv3)

layer_fc1 = create_fc_layer(input=layer_flat,
num_inputs=layer_flat.get_shape()[1:4].num_elements(),
num_outputs=fc_layer_size,
use_relu=True)

layer_fc2 = create_fc_layer(input=layer_fc1,
num_inputs=fc_layer_size,
num_outputs=num_classes,
use_relu=False)

y_pred = tf.nn.softmax(layer_fc2, name='y_pred')

y_pred_cls = tf.argmax(y_pred, dimension=1)
session.run(tf.compat.v1.global_variables_initializer())
cross_entropy = tf.nn.softmax_cross_entropy_with_logits_v2(logits=layer_fc2,
labels=y_true)
cost = tf.reduce_mean(cross_entropy)
optimizer = tf.compat.v1.train.AdamOptimizer(learning_rate=1e-4).minimize(cost)
correct_prediction = tf.equal(y_pred_cls, y_true_cls)
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
```

```
session.run(tf.compat.v1.global_variables_initializer())


# Display all stats for every epoch
def show_progress(epoch, feed_dict_train, feed_dict_validate, val_loss, 
total_epochs):
    acc = session.run(accuracy, feed_dict=feed_dict_train)
    val_acc = session.run(accuracy, feed_dict=feed_dict_validate)
    msg = "Training Epoch {0}/{4} --- Training Accuracy: {1:>6.1%}, Validation 
Accuracy: {2:>6.1%},  Validation Loss: {3:.3f}"
print(msg.format(epoch + 1, acc, val_acc, val_loss, total_epochs))


total_iterations = 0

saver = tf.compat.v1.train.Saver()

print("")


# Training Function
def train(num_iteration):
global total_iterations

for i in range(total_iterations,
                   total_iterations + num_iteration):

        x_batch, y_true_batch, _, cls_batch = data.train.next_batch(batch_size)
        x_valid_batch, y_valid_batch, _, valid_cls_batch = 
data.valid.next_batch(batch_size)

        feed_dict_tr = {x: x_batch,
                        y_true: y_true_batch}
        feed_dict_val = {x: x_valid_batch,
                         y_true: y_valid_batch}

        session.run(optimizer, feed_dict=feed_dict_tr)

if i % int(data.train.num_examples / batch_size) == 0:
            val_loss = session.run(cost, feed_dict=feed_dict_val)
            epoch = int(i / int(data.train.num_examples / batch_size))
# print(data.train.num_examples)
            # print(batch_size)
            # print(int(data.train.num_examples/batch_size))
            # print(i)

total_epochs = int(num_iteration / int(data.train.num_examples / batch_size)) + 1
show_progress(epoch, feed_dict_tr, feed_dict_val, val_loss, total_epochs)
            saver.save(session, 'trained_model')

    total_iterations += num_iteration


train(num_iteration=final_iter)


#except Exception as e:
    #print("Exception:",e)
```

```
# Calculate execution time
end = time.time()
dur = end-start
print("")
if dur<60:
print("Execution Time:",dur,"seconds")
elif dur>60 and dur<3600:
    dur=dur/60
print("Execution Time:",dur,"minutes")
else:
    dur=dur/(60*60)
print("Execution Time:",dur,"hours")
from flask import Flask, render_template, flash, request, session,send_file
from flask import render_template, redirect, url_for, request
import warnings
import datetime
import cv2
import tensorflow as tf
import numpy as np

from tkinter import *
import os


app = Flask(__name__)
app.config['DEBUG']
app.config['SECRET_KEY'] = '7d441f27d441f27567d441f2b6176a'

@app.route("/")
def homepage():

return render_template('index.html')





@app.route("/Test")
def Test():
return render_template('Test.html')





@app.route("/train", methods=['GET', 'POST'])
def train():
if request.method == 'POST':
import model as model

return render_template('Tranning.html')






@app.route("/testimage", methods=['GET', 'POST'])
def testimage():
if request.method == 'POST':
        file = request.files['fileupload']
        file.save('data/alien_test/Test.jpg')


img = cv2.imread('data/alien_test/Test.jpg')



        train_path = r'data\train'
if not os.path.exists(train_path):
print("No such directory")
raise Exception
# Path of testing images
dir_path = r'data\alien_test'
if not os.path.exists(dir_path):
print("No such directory")
raise Exception

# Walk though all testing images one by one
for root, dirs, files in os.walk(dir_path):
for name in files:

print("")
                image_path = name
                filename = dir_path + '\\' + image_path
print(filename)
                image_size = 128
num_channels = 3
images = []

if os.path.exists(filename):

# Reading the image using OpenCV
image1 = cv2.imread(filename)

                    import_file_path = filename

                    image = cv2.imread(import_file_path)
                    fnm = os.path.basename(import_file_path)
                    filename = 'Test.jpg'
cv2.imwrite(filename, image)
# print("After saving image:")

print("\n*********************\nImage : " + fnm + "\n*********************")
                    img = cv2.imread(import_file_path)
if img is None:
print('no data')

                    img1 = cv2.imread(import_file_path)
print(img.shape)
                    img = cv2.resize(img, ((int)(img.shape[1] / 5),
(int)(img.shape[0] / 5)))
                    original = img.copy()
                    neworiginal = img.copy()
                    cv2.imshow('original', img1)
                    gray = cv2.cvtColor(img1, cv2.COLOR_BGR2GRAY)
                    cv2.imshow('Original image', img1)
                    orimage = 'static/Out/Test.jpg'
cv2.imwrite(orimage, img1)


                    cv2.imshow('Gray image', gray)

                    gry = 'static/Out/gry.jpg'

cv2.imwrite(gry, gray)


                    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

totalpixels = img.shape[0] * img.shape[1]
                    per_white = 100 * p / totalpixels
if per_white >10:
                        img[i][j] = [500, 300, 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)

                    noise = 'static/Out/noise.jpg'

cv2.imwrite(noise, img)


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

                    blur = cv2.GaussianBlur(img, (11, 11), 1)
# Canny-edge detection
canny = cv2.Canny(blur, 160, 290)
                    canny = cv2.cvtColor(canny, cv2.COLOR_GRAY2BGR)
# 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 len(contours[x]) > maxC:
                            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)

                    roi12 = 'static/Out/roi.jpg'

cv2.imwrite(roi12, 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)

# Resizing the image to our desired size and preprocessing will be done exactly as
done during training
```

```
image = cv2.resize(image1, (image_size, image_size), 0, 0, cv2.INTER_LINEAR)
                    images.append(image)
                    images = np.array(images, dtype=np.uint8)
                    images = images.astype('float32')
                    images = np.multiply(images, 1.0 / 255.0)

# The input to the network is of shape [None image_size image_size num_channels].
Hence we reshape.
x_batch = images.reshape(1, image_size, image_size, num_channels)

# Let us restore the saved model
sess = tf.compat.v1.Session()
# Step-1: Recreate the network graph. At this step only graph is created.
saver = tf.compat.v1.train.import_meta_graph('models/trained_model.meta')
# Step-2: Now let's load the weights saved using the restore method.
saver.restore(sess, tf.train.latest_checkpoint('./models/'))

# Accessing the default graph which we have restored
graph = tf.compat.v1.get_default_graph()

# Now, let's get hold of the op that we can be processed to get the output.
                    # In the original network y_pred is the tensor that is the
prediction of the network
y_pred = graph.get_tensor_by_name("y_pred:0")

## Let's feed the images to the input placeholders
x = graph.get_tensor_by_name("x:0")
                    y_true = graph.get_tensor_by_name("y_true:0")
                    y_test_images = np.zeros((1, len(os.listdir(train_path))))

# Creating the feed_dict that is required to be fed to calculate y_pred
feed_dict_testing = {x: x_batch, y_true: y_test_images}
                    result = sess.run(y_pred, feed_dict=feed_dict_testing)
# Result is of this format [[probabiliy_of_classA probability_of_classB ....]]
print(result)

# Convert np.array to list
a = result[0].tolist()
                    r = 0

# Finding the maximum of all outputs
max1 = max(a)
                    index1 = a.index(max1)
                    predicted_class = None

# Walk through directory to find the label of the predicted output
count = 0
for root, dirs, files in os.walk(train_path):
for name in dirs:
if count == index1:
                                predicted_class = name
                            count += 1

# If the maximum confidence output is largest of all by a big margin then
                    # print the class or else print a warning
for i in a:
if i != max1:
if max1 - i < i:
                                r = 1
```

```
out = ''

pre = ""
if r == 0:
print(predicted_class)

if (predicted_class == "Black spot"):
                            out = predicted_class

                            pre = 'Griffin  Fertilizer  reducing the fungus'

elif (predicted_class == "canker"):
                            out = predicted_class
                            pre = 'sprayed with Bordeaux mixture 1.0 per cent.'


elif (predicted_class == "greening"):
                            out =  predicted_class
                            pre =  'Mn-Zn-Fe-B micronutrient fertilizer'

elif (predicted_class == "healthy"):
                            out =  predicted_class
# messagebox.showinfo("Uses", '')
elif (predicted_class == "Melanose"):
                            out = predicted_class
                            pre =  'strobilurin fungicide'




else:

                        out = 'Could not classify with definite confidence'


else:
print("File does not exist")






        org = 'static/Out/Test.jpg'
gry ='static/Out/gry.jpg'
noise = 'static/Out/noise.jpg'
roi12 = 'static/Out/roi.jpg'




return
render_template('Test.html',result=out,org=org,gry=gry,inv=noise,noi=roi12,fer=pre
)
```

```
def sendmsg(targetno,message):
import requests

requests.post("http://smsserver9.creativepoint.in/api.php?username=fantasy&passwor
d=596692&to=" + targetno + "&from=FSSMSS&message=Dear user  your msg is " +
message + " Sent By FSMSG
FSSMSS&PEID=1501563800000030506&templateid=1507162882948811640")
```

```
if __name__ == '__main__':
    app.run(debug=True, use_reloader=True)
import cv2
import os
import glob
from sklearn.utils import shuffle
import numpy as np


def load_train(train_path, image_size, classes):
    images = []
    labels = []
    img_names = []
    cls = []

print('Going to read training images')
for fields in classes:
        index = classes.index(fields)
print('Now going to read {} files (Index: {})'.format(fields, index))
        path = os.path.join(train_path, fields, '*g')
        files = glob.glob(path)
for fl in files:
            image = cv2.imread(fl)
            image = cv2.resize(image, (image_size, image_size),0,0,
cv2.INTER_LINEAR)
            image = image.astype(np.float32)
            image = np.multiply(image, 1.0 / 255.0)
            images.append(image)
            label = np.zeros(len(classes))
            label[index] = 1.0
labels.append(label)
            flbase = os.path.basename(fl)
            img_names.append(flbase)
            cls.append(fields)
    images = np.array(images)
    labels = np.array(labels)
    img_names = np.array(img_names)
    cls = np.array(cls)

return images, labels, img_names, cls
```

```
class DataSet(object):

def __init__(self, images, labels, img_names, cls):
self._num_examples = images.shape[0]

self._images = images
self._labels = labels
self._img_names = img_names
self._cls = cls
self._epochs_done = 0
self._index_in_epoch = 0

@property
def images(self):
return self._images

@property
def labels(self):
return self._labels

@property
def img_names(self):
return self._img_names

@property
def cls(self):
return self._cls

@property
def num_examples(self):
return self._num_examples

@property
def epochs_done(self):
return self._epochs_done

def next_batch(self, batch_size):
"""Return the next `batch_size` examples from this data set."""
start = self._index_in_epoch
self._index_in_epoch += batch_size

if self._index_in_epoch >self._num_examples:
# After each epoch we update this
self._epochs_done += 1
start = 0
self._index_in_epoch = batch_size
assert batch_size <= self._num_examples
    end = self._index_in_epoch

return self._images[start:end], self._labels[start:end],
self._img_names[start:end], self._cls[start:end]


def read_train_sets(train_path, image_size, classes, validation_size):
class DataSets(object):
pass
data_sets = DataSets()
```

```
  images, labels, img_names, cls = load_train(train_path, image_size, classes)
  images, labels, img_names, cls = shuffle(images, labels, img_names, cls)

if isinstance(validation_size, float):
    validation_size = int(validation_size * images.shape[0])

  validation_images = images[:validation_size]
  validation_labels = labels[:validation_size]
  validation_img_names = img_names[:validation_size]
  validation_cls = cls[:validation_size]

  train_images = images[validation_size:]
  train_labels = labels[validation_size:]
  train_img_names = img_names[validation_size:]
  train_cls = cls[validation_size:]

  data_sets.train = DataSet(train_images, train_labels, train_img_names,
train_cls)
  data_sets.valid = DataSet(validation_images, validation_labels,
validation_img_names, validation_cls)

return data_sets
import tensorflow as tf
import numpy as np

from tkinter import *
import os
from tkinter import filedialog
import cv2
import time
from matplotlib import pyplot as plt
from tkinter import messagebox





def endprogram():
print ("\nProgram terminated!")
   sys.exit()









def training():

import Training as tr
```

```
def imgtraining():
    import_file_path = filedialog.askopenfilename()

    image = cv2.imread(import_file_path)
    filename = 'Test.jpg'
cv2.imwrite(filename, image)
print("After saving image:")

    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

    cv2.imshow('Original image', image)
    cv2.imshow('Gray image', gray)
# import_file_path = filedialog.askopenfilename()
print(import_file_path)
    fnm = os.path.basename(import_file_path)
print(os.path.basename(import_file_path))

from PIL import Image, ImageOps

    im = Image.open(import_file_path)
    im_invert = ImageOps.invert(im)
    im_invert.save('lena_invert.jpg', quality=95)
    im = Image.open(import_file_path).convert('RGB')
    im_invert = ImageOps.invert(im)
    im_invert.save('tt.png')
    image2 = cv2.imread('tt.png')
    cv2.imshow("Invert", image2)

""""----------------------------------------------"""

img = image

gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    cv2.imshow('Original image', img)
#cv2.imshow('Gray image', gray)
dst = cv2.fastNlMeansDenoisingColored(img, None, 10, 10, 7, 21)
    cv2.imshow("Nosie Removal", dst)

gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)



print("\n********************\nImage : " + fnm + "\n********************")
    img = cv2.imread(import_file_path)
if img is None:
print('no data')

    img1 = cv2.imread(import_file_path)
print(img.shape)
    img = cv2.resize(img, ((int)(img.shape[1] / 5), (int)(img.shape[0] / 5)))
original = img.copy()
neworiginal = img.copy()
    cv2.imshow('original', img1)
    gray = cv2.cvtColor(img1, cv2.COLOR_BGR2GRAY)

    cv2.imshow('Original image', img1)
# cv2.imshow('Gray image', gray)
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

totalpixels = img.shape[0] * img.shape[1]
    per_white = 100 * p / totalpixels
if per_white >10:
        img[i][j] = [500, 300, 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)
    cv2.imshow('Noise Remove', blur)
    corners = cv2.goodFeaturesToTrack(gray, 27, 0.01, 10)
    corners = np.int0(corners)

# we iterate through each corner,
    # making a circle at each point that we think is a corner.
for i in corners:
        x, y = i.ravel()
        cv2.circle(image, (x, y), 3, 255, -1)

    plt.imshow(image), plt.show()




def testing():
global testing_screen
    testing_screen = Toplevel(main_screen)
    testing_screen.title("Testing")
# login_screen.geometry("400x300")
testing_screen.geometry("600x450+650+150")
    testing_screen.minsize(120, 1)
    testing_screen.maxsize(1604, 881)
    testing_screen.resizable(1, 1)
# login_screen.title("New Toplevel")

Label(testing_screen, text='''Upload Image''', background="#d9d9d9",
disabledforeground="#a3a3a3",
foreground="#000000", bg="turquoise", width="300", height="2", font=("Calibri",
16)).pack()
    Label(testing_screen, text="").pack()
    Label(testing_screen, text="").pack()
    Label(testing_screen, text="").pack()
    Button(testing_screen, text='''Upload Image''', font=(
'Verdana', 15), height="2", width="30", command=imgtest).pack()
```

```
def imgtest():
    import_file_path = filedialog.askopenfilename()

    image = cv2.imread(import_file_path)
print(import_file_path)
    filename = 'data/alien_test/Test.jpg'
cv2.imwrite(filename, image)
print("After saving image:")
```

```
def main_account_screen():
from PIL import Image, ImageTk
global main_screen
    main_screen = Tk()
    width = 600
height = 600
screen_width = main_screen.winfo_screenwidth()
    screen_height = main_screen.winfo_screenheight()
    x = (screen_width / 2) - (width / 2)
    y = (screen_height / 2) - (height / 2)
    main_screen.geometry("%dx%d+%d+%d" % (width, height, x, y))
    main_screen.resizable(0, 0)
# main_screen.geometry("300x250")
main_screen.title("Leaf Disease classification")

    Label(text="Leaf Disease classification", bg="turquoise", width="300",
height="5", font=("Calibri", 16)).pack()
    Label(text="").pack()
    Label(text="").pack()

    image = ImageTk.PhotoImage(Image.open('gui/12344.jpg'))

    Label(main_screen, text='Hello', image=image, compound='left', height="100",
width="200",).pack()

    Button(text="Training", font=(
'Verdana', 15), height="2", width="30", command=training,
highlightcolor="black").pack(side=TOP)
    Label(text="").pack()
    Button(text="Testing", font=(
'Verdana', 15), height="2", width="30", command=testing).pack(side=TOP)

    Label(text="").pack()

    main_screen.mainloop()


main_account_screen()
```