# Running the code instructions:

## 1) Place folder of your images in the same directory and call it “images”. Then run the following data preparation code (codes from this step and step 2 can be joined together in one “.py” file and run in a command line terminal as (python file\_name.py or python3 file\_name.py). Already joined for you in preperation\_and\_training.py

## from tensorflow.keras.preprocessing.image import ImageDataGenerator

## # Define your data augmentation

## data\_augmentation = ImageDataGenerator(

## rescale=1./255,

## rotation\_range=40,

## width\_shift\_range=0.2,

## height\_shift\_range=0.2,

## shear\_range=0.2,

## zoom\_range=0.2,

## horizontal\_flip=True,

## fill\_mode='nearest',

## validation\_split=0.2 # Using this for validation split as cross-validation is more complex and not directly supported

## )

## # Training data generator

## train\_generator = data\_augmentation.flow\_from\_directory(

## 'images/', # Assuming this is your dataset directory with subfolders for each class

## target\_size=(224, 224), # Adjust based on your model input size

## batch\_size=32,

## class\_mode='categorical', # Assuming a multi-class classification problem

## subset='training' # Use this for splitting dataset

## )

## # Validation data generator

## validation\_generator = data\_augmentation.flow\_from\_directory(

## 'images/',

## target\_size=(224, 224),

## batch\_size=32,

## class\_mode='categorical',

## subset='validation' # Use this for splitting dataset

## )

## 2) Run training and testing code:

## from tensorflow.keras.applications import MobileNetV2

## from tensorflow.keras.layers import Dense, GlobalAveragePooling2D

## from tensorflow.keras.models import Model

## # Load MobileNetV2 as the base model

## base\_model = MobileNetV2(weights='imagenet', include\_top=False, input\_shape=(224, 224, 3))

## # Freeze the base model

## base\_model.trainable = False

## # Add custom layers on top for the classification task

## x = base\_model.output

## x = GlobalAveragePooling2D()(x)

## x = Dense(1024, activation='relu')(x)

## predictions = Dense(train\_generator.num\_classes, activation='softmax')(x)

## # Final model

## model = Model(inputs=base\_model.input, outputs=predictions)

## # Compile the model

## model.compile(optimizer='adam', loss='categorical\_crossentropy', metrics=['accuracy'])

## # Train the model

## model.fit(

## train\_generator,

## epochs=100, # Adjust based on requirements

## validation\_data=validation\_generator

## )

## 3) Saving the model:

## model.save('my\_model.h5')

4) Running the flask website:

Place this code in a file and call it app.py:

from tensorflow.keras.preprocessing.image import ImageDataGenerator

import tensorflow as tf

from flask import Flask, request, render\_template

from tensorflow.keras.preprocessing import image

import numpy as np

import os

from io import BytesIO

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

# Load the trained model (ensure the model is saved after training and path is correct)

model = tf.keras.models.load\_model('your\_model.h5')

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

def index():

return render\_template('index.html')

class\_labels = {0: 'afan', 1: 'alshees', 2: 'depelodia', 3: 'fyozariomi', 4: 'khayas', 5: 'lafha', 6: 'leaf', 7: 'mayalan', 8: 'tabaqqu3', 9: 'takashor', 10: 'tashteeb', 11: 'thobool'}

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

def predict():

img\_file = request.files['image']

# Convert the FileStorage to a BytesIO object

img\_bytes = BytesIO(img\_file.read())

# Load the image directly from BytesIO

img = image.load\_img(img\_bytes, target\_size=(224, 224))

img\_array = image.img\_to\_array(img) / 255.0

img\_array = np.expand\_dims(img\_array, axis=0) # Add batch dimension

# Perform prediction

predictions = model.predict(img\_array)

predicted\_class\_index = np.argmax(predictions, axis=1)[0]

predicted\_label = class\_labels.get(predicted\_class\_index, 'Unknown') # Handle case where index is not in class\_labels

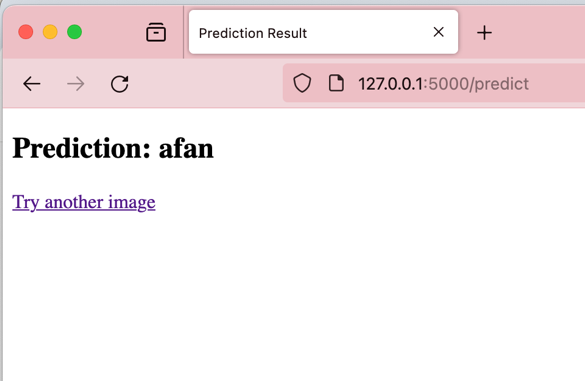
return render\_template('result.html', prediction=predicted\_label)

if \_\_name\_\_ == '\_\_main\_\_':

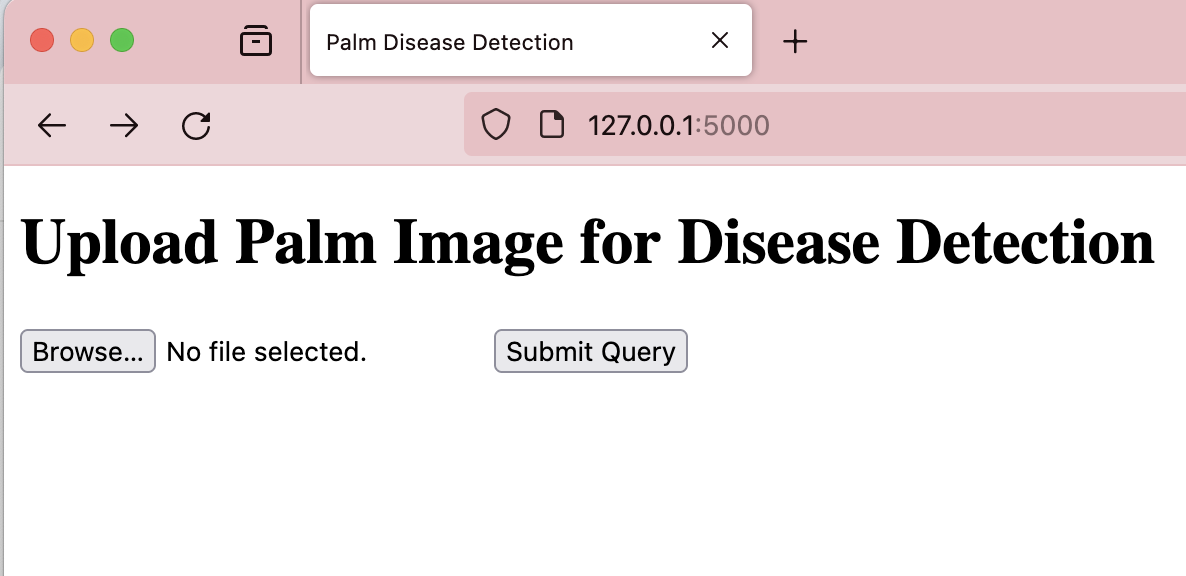
app.run(debug=True)

5) At the same directory as app.py, add a folder called templates. Templates folder will include the two html files I attached.

- result.html (the page that presents the prediction results)



- index.html (the page that where you upload the image and submit)



6) From the terminal at the same directory of the app.py, run the following command depending on if you have python 2 or python 3: python app.py or python3 app.py

You will see a localhost link in the output, put it on the browser:

