DATE	20-02-2026
TEAM ID	LTVIP2026TMIDS90622
PROJECT NAME	SMART SORTING : TRANSFER LEARNING FOR IDENTIFYING ROTTEN FRUITS AND VEGETABLES
MAXIMUM MARKS	4 MARKS

6.5 APPLICATION BUILDING

In this section, we will be building a web application that is integrated into the model we built. A UI is provided for the uses where he has to enter the values for predictions. The enter values are given to the saved model and prediction is showcased on the UI.

This section has the following tasks

- ✓ Building HTML Pages
- ✓ Building server-side script

Building HTML Pages:

For this project create two HTML files namely

- ✧ index.html
- ✧ Portfolio-details.html

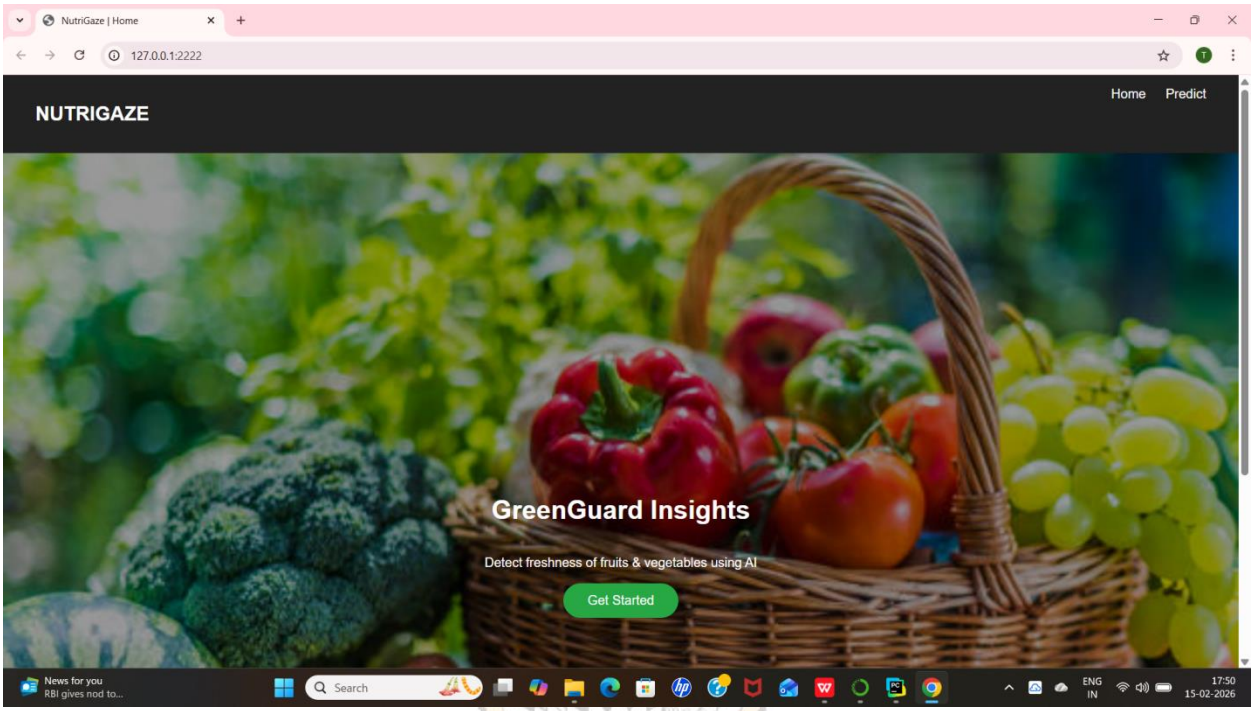
And save them in the templates folder.

- ✧ Style.css(in static =>css => style.css)
- ✧ Image(in static => image=>img.jpg)

UI Image preview:

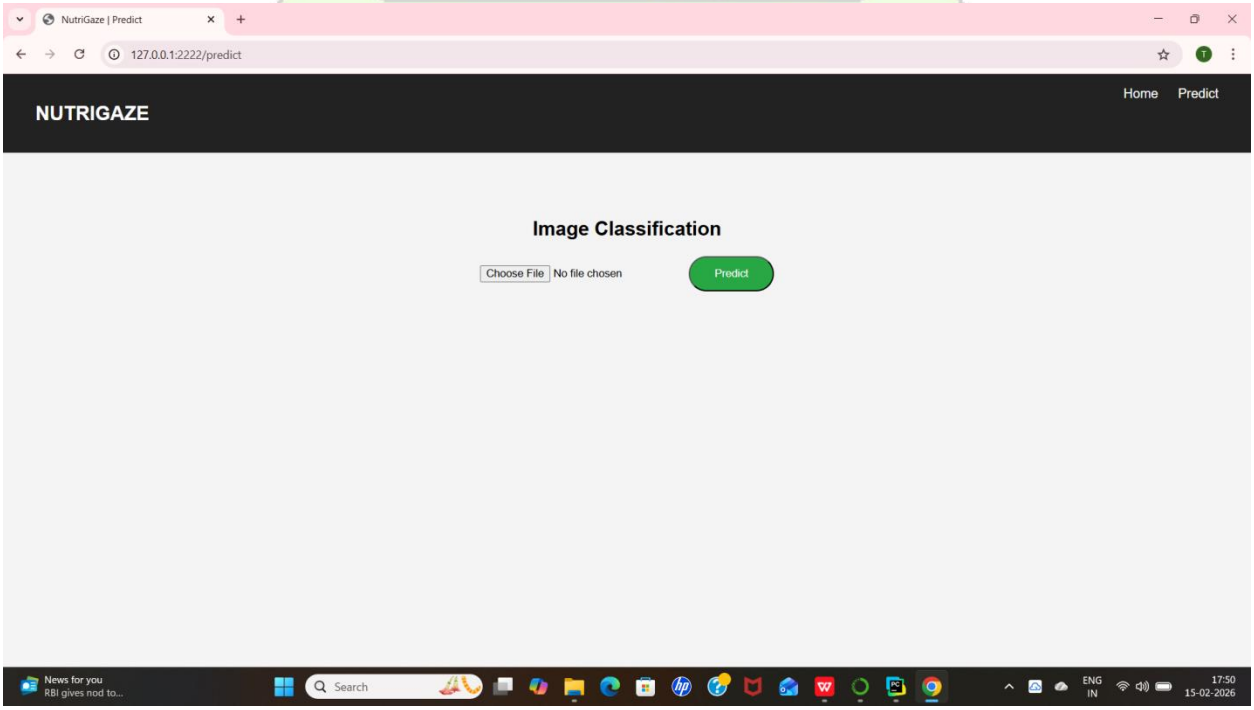
This html pages and css spreadsheet will help to enhance the beauty of our web page “Neutrigaze”.

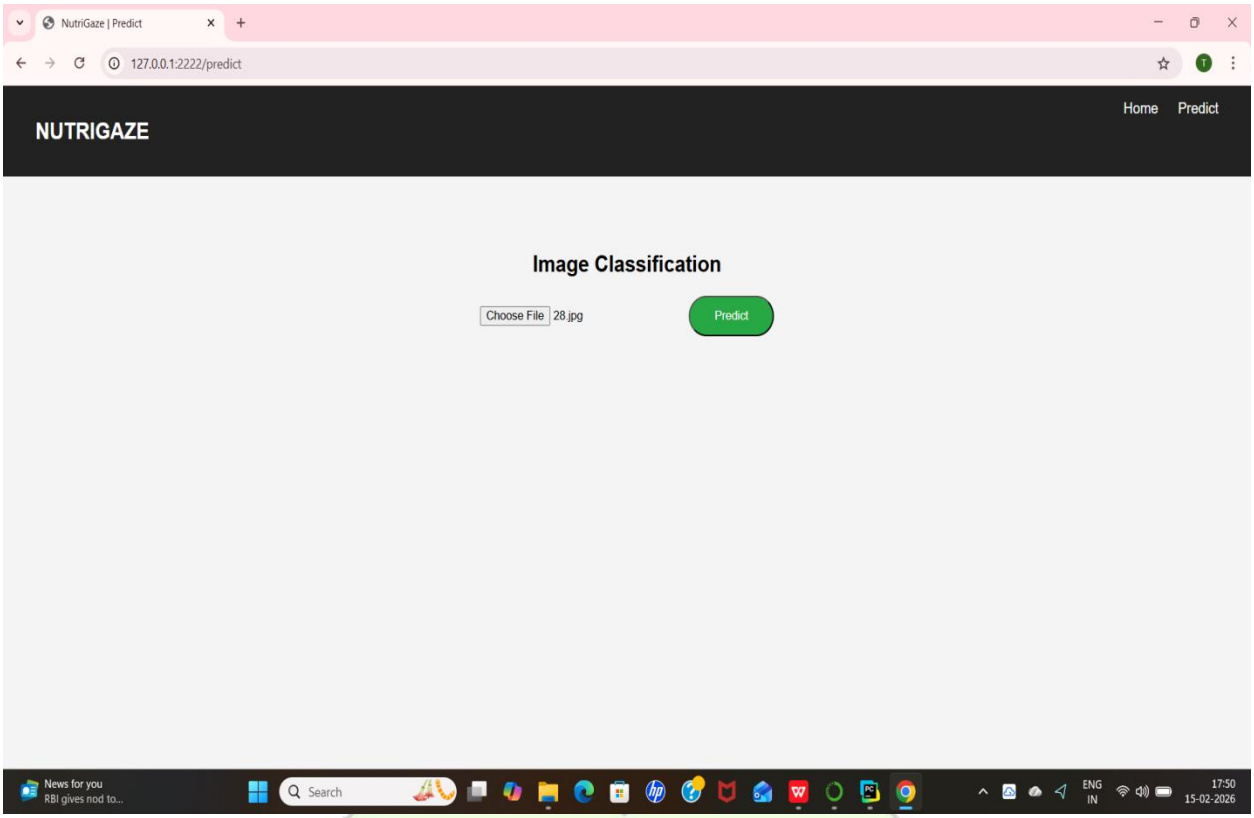
It provide home page and predict page in the preview as shown in the image given below .

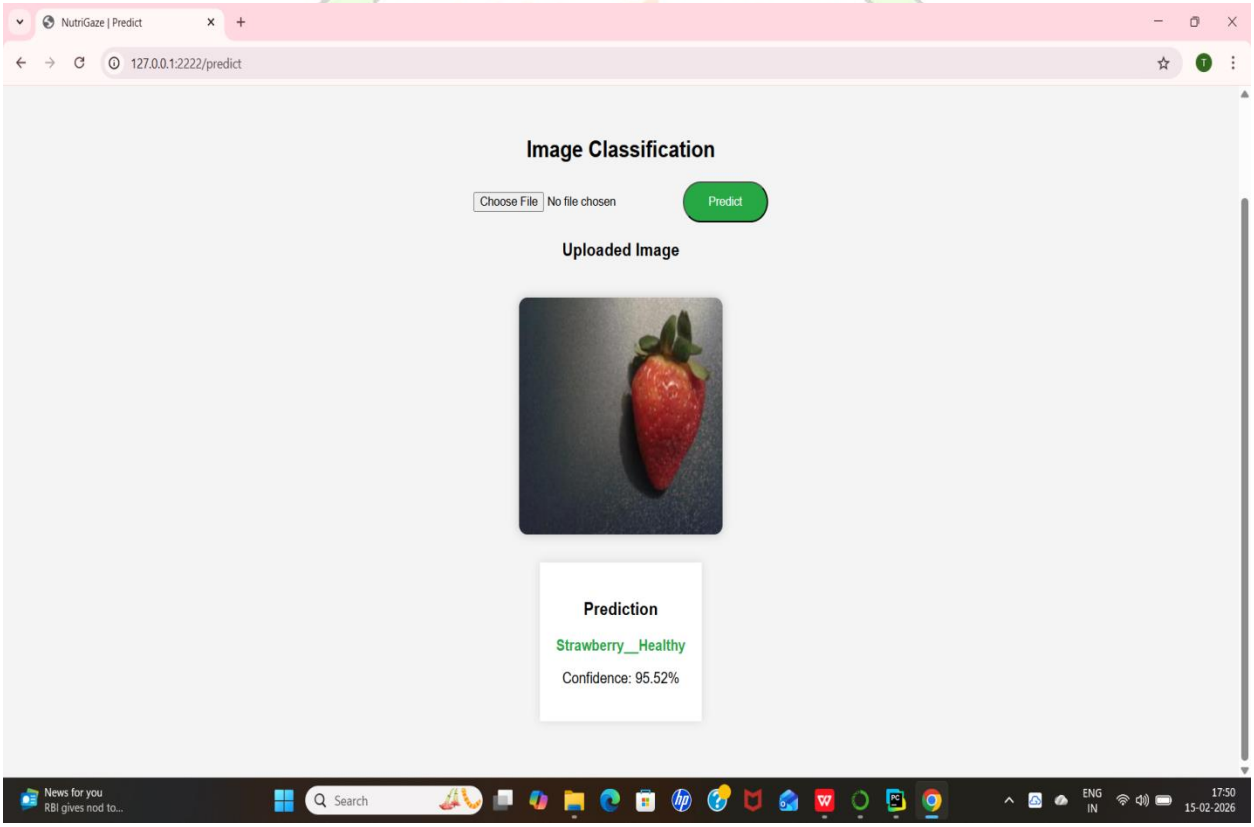


Now when you click on the predict button further in the top right corner you will get redirected to portfolio-details.html

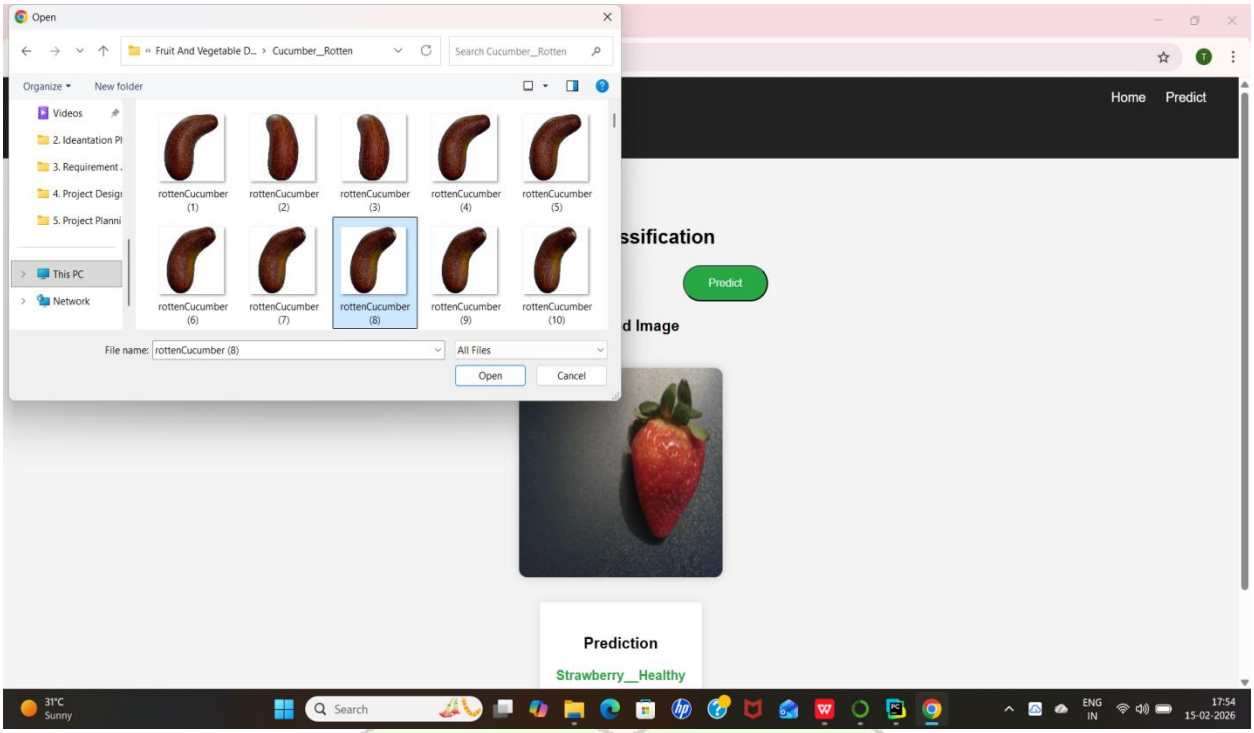
Let's look at what our inner.html file looks like and test the model:



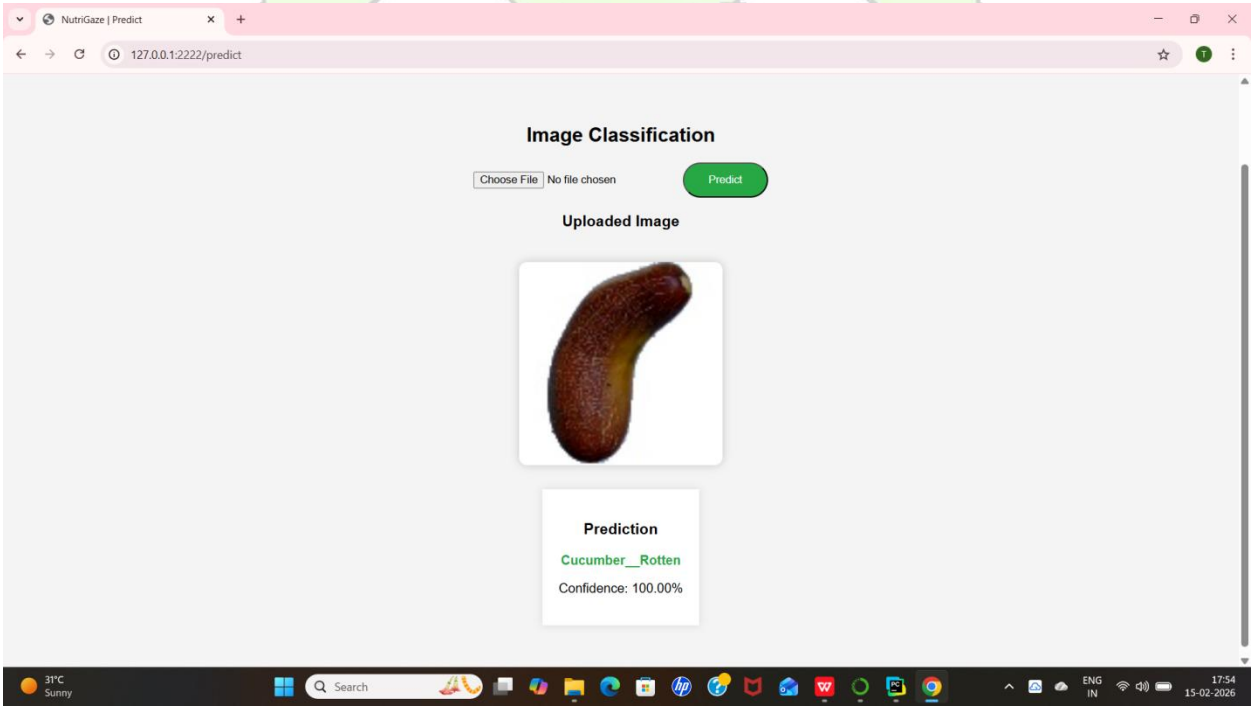

NutriGaze | Predict
127.0.0.1:2222/predict
NUTRIGAZE
Home
Predict
Image Classification
Choose File
28.jpg
Predict


NutriGaze | Predict
127.0.0.1:2222/predict
Image Classification
Choose File
No file chosen
Predict
Uploaded Image
Prediction
Strawberry__Healthy
Confidence: 95.52%

Test Class rotten cucumber (8):



Now when you click on predict button you will get output down to the image itself.Let's look how our output looks like:



Build Python code:

Import the libraries

```
from flask import Flask, render_template,request,jsonify,url_for,redirect
from tensorflow.keras.preprocessing.image import load_img,img_to_array
from PIL import Image
import numpy as np
import os
import tensorflow as tf
```

Load the saved model. Importing the Flask module in the project is mandatory. An object of the Flask class is our WSGI application. The Flask constructor takes the name of the current module (__name__) as argument.

```
app=Flask(__name__)
model = tf.keras.models.load_model('healthy_vs_rotten.h5')

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

Here we will be using the declared constructor to route to the HTML page which we have created earlier.

In the above example, the '/' URL is bound with the index.html function. Hence, when the index page of the web server is opened in the browser, the html page will be rendered. Whenever you enter the values from the html page the values can be retrieved using POST Method.

Retrieves the value from UI:

```
@app.route('/predict', methods=['GET','POST'])
def output():
    if request.method =='POST':
        f=request.files['pc_image']
        img_path = "static/uploads/" + f.filename
        f.save(img_path)
        img=load_img(img_path,target_size=(224,224))
        # Resize the image to the required size
        # Convert the image to an array and normalize it
        image_array = np.array(img)
        # Add a batch dimension
        image_array = np.expand_dims(image_array, axis=0)
        # Use the pre-trained model to make a prediction
        pred=np.argmax(model.predict(image_array),axis=1)
        index=['Apple__Healthy (0)','Apple__Rotten (1)','Banana__Healthy (2)',
               'Banana__Rotten (3)','Bellpepper__Healthy (4)','Bellpepper__Rotten (5)',
               'Carrot__Healthy (6)','Carrot__Rotten (7)', 'Cucumber__Healthy (8)','Cucumber__Rotten (9)','Grape__Healthy (10)',
               'Grape__Rotten (11)','Guava__Healthy (12)','Guava__Rotten (13)',
               'Jujube__Healthy (14)',
               'Jujube__Rotten (15)', 'Mango__Healthy (16)', 'Mango__Rotten (17)', 'Orange__Healthy (18)',
               'Orange__Rotten (19)', 'Pomegranate__Healthy (20)', 'Pomegranate__Rotten (21)', 'Potato__Healthy (22)',
               'Potato__Rotten (23)', 'Strawberry__Healthy (24)', 'Strawberry__Rotten (25)', 'Tomato__Healthy (26)', 'Tomato__Rotten (27)'
              ]
        prediction = index[int(pred)]
        print("prediction")
        #predict = prediction
        return render_template("portfolio-details.html", predict = prediction)
```

Here we are routing our app to the output() function. This function retrieves all the values from the HTML page using a Post request. That is stored in an array. This array is passed to the model. Predict () function. This function returns the prediction. This prediction value will rendered to the text that we have mentioned in the output.html page earlier.

Main Function:

```
if __name__=='__main__':
    app.run(debug = True,port = 2222)
```

Run the web application

- ✓ Open Anaconda prompt from the start menu
- ✓ Navigate to the folder where your Python script is.
- ✓ Now type the “app.py” command
- ✓ Navigate to the local host where you can view your web page.
- ✓ Click on the inspect button from the top right corner, enter the inputs, click on the predict button, and see the result/prediction on the web.

```
In [1]: runfile('C:/Users/santu/Downloads/flask2/app.py', wdir='C:/
Users/santu/Downloads/flask2')
WARNING:absl:Compiled the loaded model, but the compiled metrics have
yet to be built. `model.compile_metrics` will be empty until you train
or evaluate the model.
WARNING:absl:Error in loading the saved optimizer state. As a result,
your model is starting with a freshly initialized optimizer.
 * Serving Flask app 'app'
 * Debug mode: on
INFO:werkzeug:WARNING: This is a development server. Do not use it in
production deployment. Use a production WSGI server instead.
 * Running on http://127.0.0.1:2222
INFO:werkzeug:Press CTRL+C to quit
INFO:werkzeug: * Restarting with watchdog (windowsapi)
```

TEST.PY CODE:

```
from flask import Flask, render_template, request
from tensorflow.keras.preprocessing.image import load_img,
img_to_array
import numpy as np
import tensorflow as tf
import os
app = Flask(__name__)
# Load trained model
model = tf.keras.models.load_model("healthy_vs_rotten.h5")
# Threshold: If confidence is below 80%, we say "Out of Dataset"
# Adjust this (0.0 to 1.0) based on your testing!
```

```
CONFIDENCE_THRESHOLD = 0.92  # Requires 92% certainty
classes = [
    'Apple__Healthy',       'Apple__Rotten',       'Banana__Healthy',
'Banana__Rotten',
    'Bellpepper__Healthy',  'Bellpepper__Rotten',  'Carrot__Healthy',
'Carrot__Rotten',
    'Cucumber__Healthy',    'Cucumber__Rotten',    'Grape__Healthy',
'Grape__Rotten',
    'Guava__Healthy',       'Guava__Rotten',       'Jujube__Healthy',
'Jujube__Rotten',
    'Mango__Healthy',       'Mango__Rotten',       'Orange__Healthy',
'Orange__Rotten',
    'Pomegranate__Healthy', 'Pomegranate__Rotten', 'Potato__Healthy',
'Potato__Rotten',
    'Strawberry__Healthy',  'Strawberry__Rotten',  'Tomato__Healthy',
'Tomato__Rotten'
]
@app.route("/")
def home():
    return render_template("index.html")
from tensorflow.keras.applications.vgg16 import preprocess_input # Add
this import
@app.route("/predict", methods=["GET", "POST"])
def predict():
    prediction = None
    confidence = None
    image_path = None
    filename = None
    if request.method == "POST":
        file = request.files.get("pc_image")
        if file:
            filename = file.filename
            image_path = os.path.join("static/uploads", filename)
            os.makedirs("static/uploads", exist_ok=True)
            file.save(image_path)

            # ✅ 1. Standard VGG16 Preprocessing
            img = load_img(image_path, target_size=(224, 224))
```

```
            img_array = np.array(img)
            # If you used rescale=1/255 in training, keep the next
line.
            #    If    not,    comment    it    out    and    use
preprocess_input(img_array)
            img_array = img_array / 255.0
            img_array = np.expand_dims(img_array, axis=0)
            # ✅ 2. Get All Probabilities
            probs = model.predict(img_array)[0]
            class_index = np.argmax(probs)
            max_prob = probs[class_index]
            # ✅ 3. Calculate "Confusion" (Gap between top 2 guesses)
            sorted_probs = np.sort(probs)
            top_choice = sorted_probs[-1]
            second_choice = sorted_probs[-2]
            gap = top_choice - second_choice
            # ✅ 4. STRICT FILTERING
            # A tiger shouldn't have a high gap, and its confidence
shouldn't be massive.
            # We require at least 85% confidence AND a 40% gap over
the next best guess.
            if top_choice < 0.85 or gap < 0.40:
                prediction = "Out of Dataset"
                confidence = top_choice * 100
            else:
                prediction = classes[class_index]
                confidence = top_choice * 100

    return                render_template("portfolio-details.html",
predict=prediction,
                          confidence=confidence,
image_path=filename)
if __name__ == "__main__":
    app.run(debug=True, port=2222)
```