Chapter-12

12.1 SOURCE CODE

APP.PY CODE :

```
from sklearn.model_selection import train_test_split  # Fixed import
statement
import os
import shutil
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from IPython.display import Image, display
import random
from sklearn.model_selection import train_test_split
from tensorflow.keras.applications.vgg16 import VGG16
from tensorflow.keras.layers import Dense, Flatten
from tensorflow.keras.models import Model
from keras.optimizers import Adam
from tensorflow.keras.applications.resnet50 import ResNet50
from tensorflow.keras.layers import Dense, Flatten
from tensorflow.keras.models import Model
from keras.preprocessing import image
from keras.applications.vgg16 import preprocess_input
from    tensorflow.keras.preprocessing.image    import    load_img,
img_to_array
import numpy as np
from sklearn.model_selection import train_test_split
from keras.callbacks import EarlyStopping
from keras.optimizers import Adam
# Set the path to the dataset - Update this path to the correct location
of your dataset
dataset_dir                                                          =
'C:/Users/tejas/PycharmProjects/PythonProject/content/Fruit       And
Vegetable Diseases Dataset'  # CHANGE THIS to your actual dataset path
with full absolute path
# Check if the directory exists before proceeding
if not os.path.exists(dataset_dir):
    raise  FileNotFoundError(f"The  dataset  directory  '{dataset_dir}'
does not exist. Please check the path.")
```

```
# Create directories for train, val, and test sets
output_dir = 'output_dataset'
os.makedirs(output_dir, exist_ok=True)
os.makedirs(os.path.join(output_dir, 'train'), exist_ok=True)
os.makedirs(os.path.join(output_dir, 'val'), exist_ok=True)
os.makedirs(os.path.join(output_dir, 'test'), exist_ok=True)
classes = os.listdir(dataset_dir)
for cls in classes:
    os.makedirs(os.path.join(output_dir, 'train', cls), exist_ok=True)
    os.makedirs(os.path.join(output_dir, 'val', cls), exist_ok=True)
    os.makedirs(os.path.join(output_dir, 'test', cls), exist_ok=True)
    class_dir = os.path.join(dataset_dir, cls)
    images = os.listdir(class_dir)[:200]
   print(cls, len(images))
    train_and_val_images, test_images = train_test_split(
        images, test_size=0.2, random_state=42
    )
    train_images, val_images = train_test_split(
        train_and_val_images, test_size=0.25, random_state=42
    )  # 0.25 x 0.8 = 0.2
    # Copy images to respective directories
    for img in train_images:
        shutil.copy(
            os.path.join(class_dir, img),
            os.path.join(output_dir, 'train', cls, img)
        )
    for img in val_images:
        shutil.copy(
            os.path.join(class_dir, img),
            os.path.join(output_dir, 'val', cls, img)
        )
    for img in test_images:
        shutil.copy(
            os.path.join(class_dir, img),
            os.path.join(output_dir, 'test', cls, img)
        )
print("Dataset split into training, validation, and test sets.")
# Define directories
```

```
dataset_dir = 'output_dataset'
train_dir = os.path.join(dataset_dir, 'train')
val_dir = os.path.join(dataset_dir, 'val')
test_dir = os.path.join(dataset_dir, 'test')
# Define image size expected by the pre-trained model
IMG_SIZE = (224, 224)  # Common size for many models like ResNet, VGG,
MobileNet
# Create ImageDataGenerators for resizing and augmenting the images
train_datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=20,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest'
)
val_test_datagen = ImageDataGenerator(rescale=1./255)
# Load and resize the images from directories
train_generator = train_datagen.flow_from_directory(
    train_dir,
    target_size=IMG_SIZE,
    batch_size=32,
    class_mode='binary'  # Assuming binary classification for healthy
vs rotten
)
val_generator = val_test_datagen.flow_from_directory(
    val_dir,
    target_size=IMG_SIZE,
    batch_size=32,
    class_mode='binary'
)
test_generator = val_test_datagen.flow_from_directory(
    test_dir,
    target_size=IMG_SIZE,
    batch_size=32,
    class_mode='binary',
    shuffle=False  # Do not shuffle test data
```

```
)
# Print class indices for reference
print(train_generator.class_indices)
print(val_generator.class_indices)
print(test_generator.class_indices)
# Specify the path to your image folder
folder_path = 'output_dataset/train/Apple__Healthy'  # Replace with
the actual path to your image folder
# List all files in the folder
image_files  =  [f  for  f  in  os.listdir(folder_path)  if
f.endswith(('.jpg', '.png', '.jpeg'))]
# Select a random image from the list
selected_image = random.choice(image_files)
# Display the randomly selected image
image_path = os.path.join(folder_path, selected_image)
display(Image(filename=image_path))
# Specify the path to your image folder
folder_path = 'output_dataset/test/Strawberry__Healthy'  # Replace
with the actual path to your image folder
# List all files in the folder
image_files  =  [f  for  f  in  os.listdir(folder_path)  if
f.endswith(('.jpg', '.png', '.jpeg'))]
# Select a random image from the list
selected_image = random.choice(image_files)
# Display the randomly selected image
image_path = os.path.join(folder_path, selected_image)
display(Image(filename=image_path))
# Specify the path to your image folder
folder_path = 'output_dataset/test/Cucumber__Rotten'  # Replace with
the actual path to your image folder
# List all files in the folder
image_files  =  [f  for  f  in  os.listdir(folder_path)  if
f.endswith(('.jpg', '.png', '.jpeg'))]
# Select a random image from the list
selected_image = random.choice(image_files)
# Display the randomly selected image
image_path = os.path.join(folder_path, selected_image)
display(Image(filename=image_path))
# Specify the path to your image folder
```

```
folder_path = 'output_dataset/test/Strawberry__Rotten'  # Replace with
the actual path to your image folder
# List all files in the folder
image_files    =    [f    for    f    in    os.listdir(folder_path)    if
f.endswith(('.jpg', '.png', '.jpeg'))]
# Select a random image from the list
selected_image = random.choice(image_files)
# Display the randomly selected image
image_path = os.path.join(folder_path, selected_image)
display(Image(filename=image_path))
trainpath = "output_dataset/train"
testpath = "output_dataset/test"
train_datagen = ImageDataGenerator(
    rescale=1./255,
    zoom_range=0.2,
    shear_range=0.2
)
test_datagen = ImageDataGenerator(rescale=1./255)
train = train_datagen.flow_from_directory(
    trainpath,
    target_size=(224, 224),
    batch_size=20
)
test = test_datagen.flow_from_directory(
    testpath,
    target_size=(224, 224),
    batch_size=20
)  # 5, 15, 32, 50
vgg = VGG16(include_top=False, input_shape=(224, 224, 3))
for layer in vgg.layers:
    print(layer)
len(vgg.layers)
for layer in vgg.layers:
    layer.trainable = False
x = Flatten()(vgg.output)
output = Dense(30 , activation='softmax')(x)
vgg16 = Model(vgg.input, output)
vgg16.summary()
opt = Adam(learning_rate=0.0001)
```

```
# Assuming you have defined your VGG16 model as vgg16
# Define Early Stopping callback
early_stopping = EarlyStopping(
    monitor='val_accuracy',
    patience=3,
    restore_best_weights=True
)
# Compile the model (you may have already done this)
vgg16.compile(
    optimizer="Adam",
    loss='categorical_crossentropy',
    metrics=['accuracy']
)
# Train the model with early stopping callback
history = vgg16.fit(
    train,
    validation_data=test,
    epochs=15,
    steps_per_epoch=20,
    callbacks=[early_stopping]
)
vgg16.save('healthy_vs_rotten.h5')
```

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)
```

INDEX.HTML CODE :

```
<!DOCTYPE html>
<html>
<head>
    <title>NutriGaze | Home</title>
    <link      rel="stylesheet"      href="{{      url_for('static',
filename='css/style.css') }}">
</head>
<body>
<header class="navbar">
    <h2>NUTRIGAZE</h2>
    <nav>
        <a href="{{ url_for('home') }}">Home</a>
        <a href="{{ url_for('predict') }}">Predict</a>
    </nav>
</header>
<section class="hero">
    <h1>GreenGuard Insights</h1>
    <p>Detect freshness of fruits & vegetables using AI</p>
    <a href="{{ url_for('predict') }}" class="btn">Get Started</a>
</section>
</body>
</html>
```

PORTFOLIO-DETAILS.HTML CODE:

```
<!DOCTYPE html>
<html>
<head>
    <title>NutriGaze | Predict</title>
    <link      rel="stylesheet"      href="{{      url_for('static',
filename='css/style.css') }}">
</head>
<body>
<header class="navbar">
    <h2>NUTRIGAZE</h2>
```

```
    <nav>
        <a href="{{ url_for('home') }}">Home</a>
        <a class="active" href="{{ url_for('predict') }}">Predict</a>
    </nav>
</header>
<section class="predict-section">
    <h2>Image Classification</h2>
    <form method="POST" enctype="multipart/form-data">
        <input type="file" name="pc_image" required>
        <button class="btn green" type="submit">Predict</button>
    </form>
  {% if image_path %}
<div class="preview">
    <h3>Uploaded Image</h3>
    <img src="{{ url_for('static', filename='uploads/' + image_path)
}}" alt="Preview">
</div>
{% endif %}
{% if predict %}
<div class="result-box">
    <h3>Prediction</h3>
    <p><strong style="color: {{ 'red' if predict == 'Out of Dataset'
else '#28a745' }}">
        {{ predict }}
    </strong></p>
    <p>Confidence: {{ "%.2f"|format(confidence) }}%</p>
</div>
{% endif %}</section>
</body></html>
```

STYLE.CSS CODE:

```
body {
    font-family: Arial, sans-serif;
    background: #f4f4f4;
    margin: 0;
}
.navbar {
```

```
    background: #222;
    padding: 15px 40px;
    color: white;
    display: flex;
    justify-content: space-between;
}
.navbar a {
    color: white;
    margin-left: 20px;
    text-decoration: none;
}
.hero {
    padding: 120px;
    text-align: center;
}
.btn {
    background: #28a745;
    color: white;
    padding: 12px 30px;
    border-radius: 25px;
    text-decoration: none;
}
.predict-section {
    padding: 60px;
    text-align: center;
}
.preview img {
    width: 250px;
    margin-top: 20px;
    border-radius: 10px;
    box-shadow: 0 0 10px rgba(0,0,0,.2);
}
.result-box {
    margin-top: 25px;
    background: white;
    padding: 20px;
    display: inline-block;
    box-shadow: 0 0 10px rgba(0,0,0,.1);
}
```

```
.hero {
    padding: 120px;
    text-align: center;
    color: white;              /* ensures text is readable */
    background: url('../images/bg-home.jpg') no-repeat center center;
    background-size: cover;     /* makes it fill the section */
    position: relative;
}
/* Optional: add overlay for better contrast */
.hero::before {
    content: "";
    position: absolute;
    top: 0; left: 0; right: 0; bottom: 0;
    background: rgba(0,0,0,0.4);  /* dark overlay */
    z-index: 0;
}
.hero h1,
.hero p,
.hero .btn {
    position: relative;  /* keep text above overlay */
    z-index: 1;
}
.hero {
    height: 100vh;   /* full viewport height */
    display: flex;
    flex-direction: column;
    justify-content: center;
    align-items: center;}
```

BG-HOME.JPG IMAGE :