Artificial Intelligence & Machine Learning

Project Documentation

1. Introduction

1.1 Project Title

Smart Sorting: Transfer Learning for Identifying Rotten Fruits and Vegetables

2. Project Overview

2.1 Purpose:

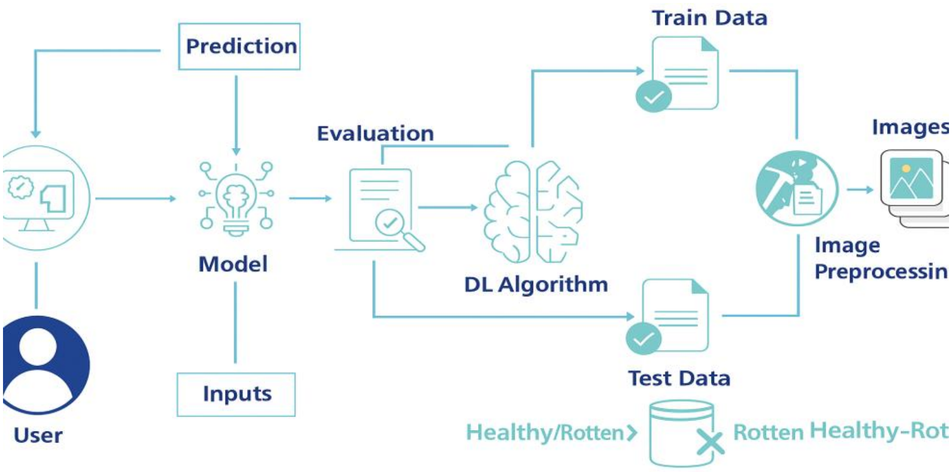
To develop an AI-powered web application that classifies uploaded images of fruits or vegetables into Healthy or Rotten categories, using Transfer Learning with VGG16, and provides:

1. A confidence score
2. A recommendation: *"Good to Eat"* or *"Don't Eat"*

2.2 Features:

1. Image upload with real-time classification
2. Real-time inference using VGG16 (Transfer Learning)
3. Confidence score display
4. "Good to Eat"/"Don't Eat" recommendation
5. Multi-class support (28 classes from real-world dataset)
6. Lightweight and optimized predictions
7. Feedback system for continuous learning
8. Error handling (invalid input, corrupt files, etc.)

3. Architecture



3.1 Backend (Flask)

- **Framework:** Flask (Python 3.9)
- **Model:** fruit_veg_disease_model.keras trained using VGG16 with Transfer Learning
- **Preprocessing:**
  - Image resizing, scaling, normalization
  - Data augmentation applied during training phase
- **Modules:**
  - **app.py:** Flask routes for prediction and feedback
  - **class_names.json:** JSON labels for 28 fruit/veg classes
  - **feedback_data:** Stores feedback in .json format

3.2 Frontend

- Built using HTML, Internal CSS & Jinja Templates
- **Pages:**
  - **index.html** – Upload interface
  - **result.html** – Prediction results with score and recommendation
  - **feedback.html** – Collects feedback

3.3 Model

- **Model:** VGG16 pretrained on ImageNet, fine-tuned with 28-class dataset
- **Libraries:**
  - Keras 2.10
  - TensorFlow 2.10
- **Optimizer:** Adam
- **Loss:** Categorical Crossentropy
- **Evaluation:**
  - Accuracy
  - F1 Score
- **Multiple versions:**
  - best_model.keras
  - final_model.keras
  - model_checkpoint.h5
  - model_checkpoint.keras
  - fruit_veg_disease_model.h5
  - healthy_vs_rotten_model.keras

3.4 Dataset

- **Source:** Kaggle Fruit and Vegetable Disease Dataset
- **Structure:**
  - **train:** Healthy & Rotten classes
  - **validation:** For evaluation
- **Preprocessing Techniques Used:**
  - Image resizing & normalization
  - Data augmentation
  - One-hot encoding for labels
  - Feature scaling

4. Setup Instructions

4.1 Prerequisites:

- Python 3.9+
- Flask 2.2.5
- TensorFlow 2.10.0
- Keras 2.10.0
- Other Python libraries (see requirements.txt)

4.2 Installation:

Step-by-step guide to clone, install dependencies, and set up the environment variables.

1. **Create a virtual environment:**
   1. conda create -n smart-sorting python=3.9
   2. conda activate smart-sorting
2. **Install dependencies:**
   1. pip install -r backend/requirements.txt
3. **Run the Flask server:**
   1. python backend/app.py

5. Folder Structure

- **Client:** Describe the structure of the **HTML, With Internal CSS** frontend.
- **Server:** Explain the organization of the **Flask** backend.

6. Running the Application

- **Start Flask server:** python backend/app.py

7. API Documentation

Route	Method	Description
/	GET	Load upload form
/predict	POST	Handle image upload & return prediction
/feedback	POST	Save user feedback to local JSON

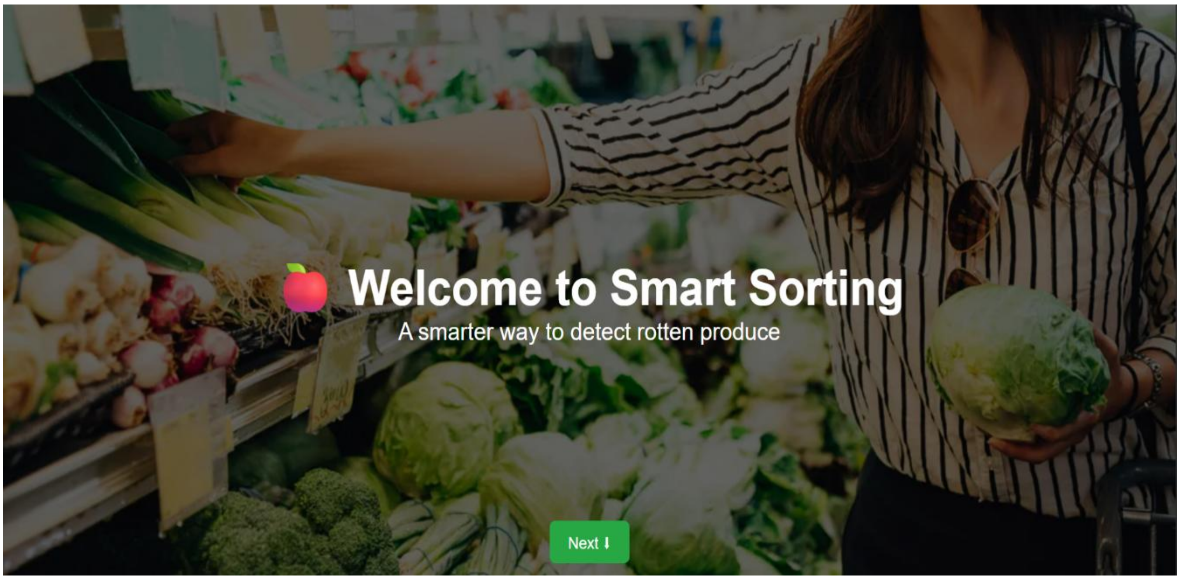
8. Authentication

- Not implemented.
- **Future Scope**: Added admin panel to review feedback and retrain models.

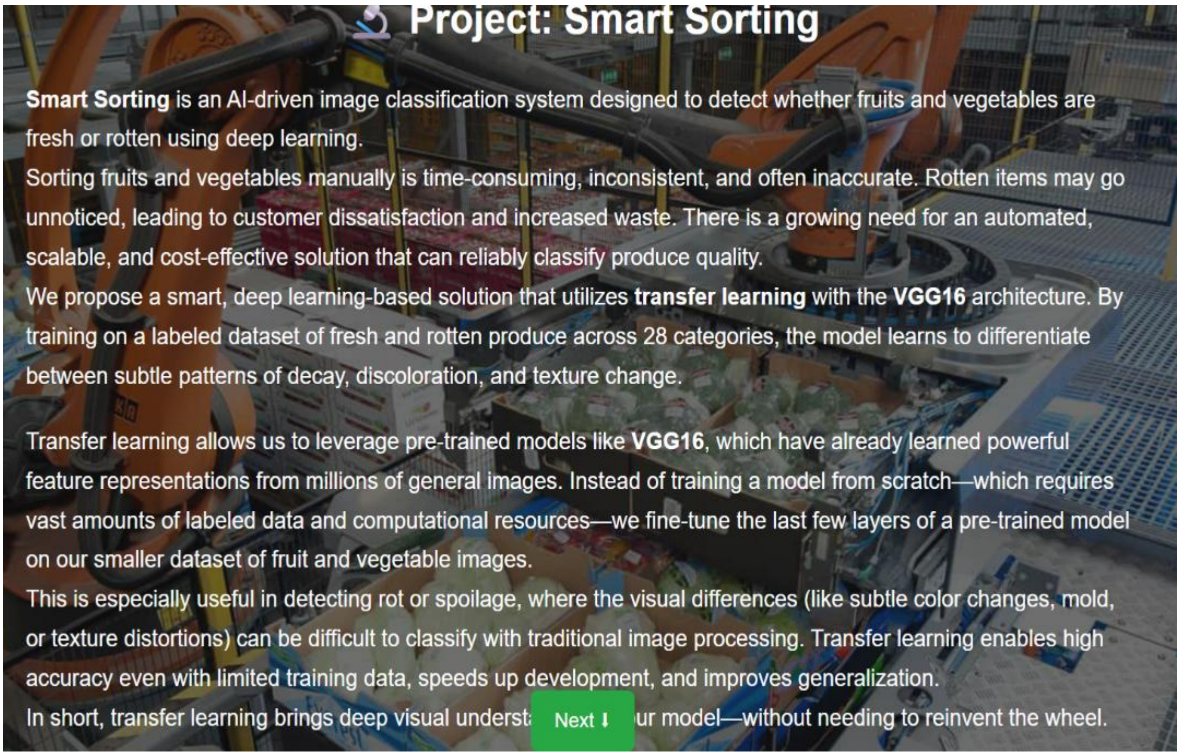
9. User Interface

9.1 By index.html code

- Starting page

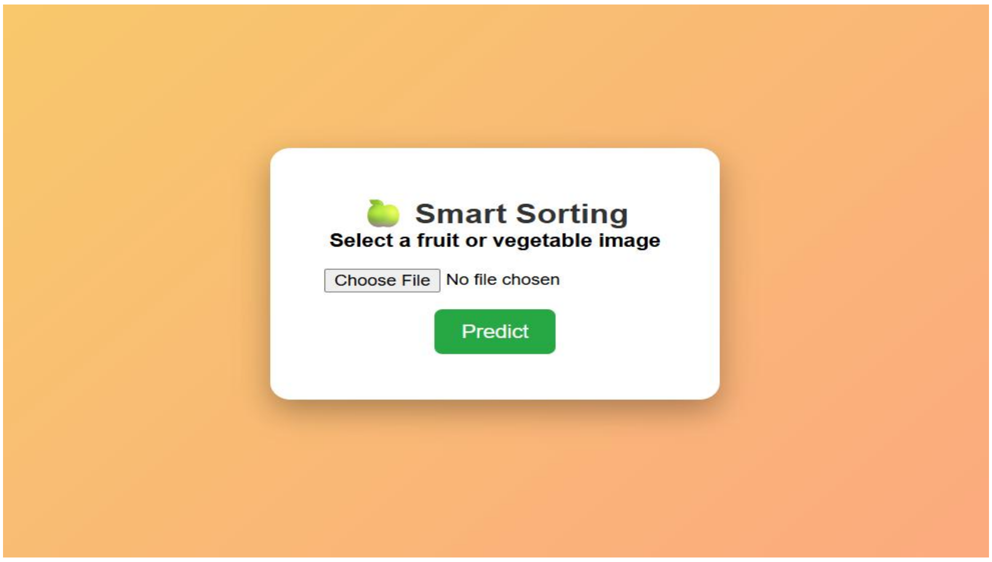


- Content Page

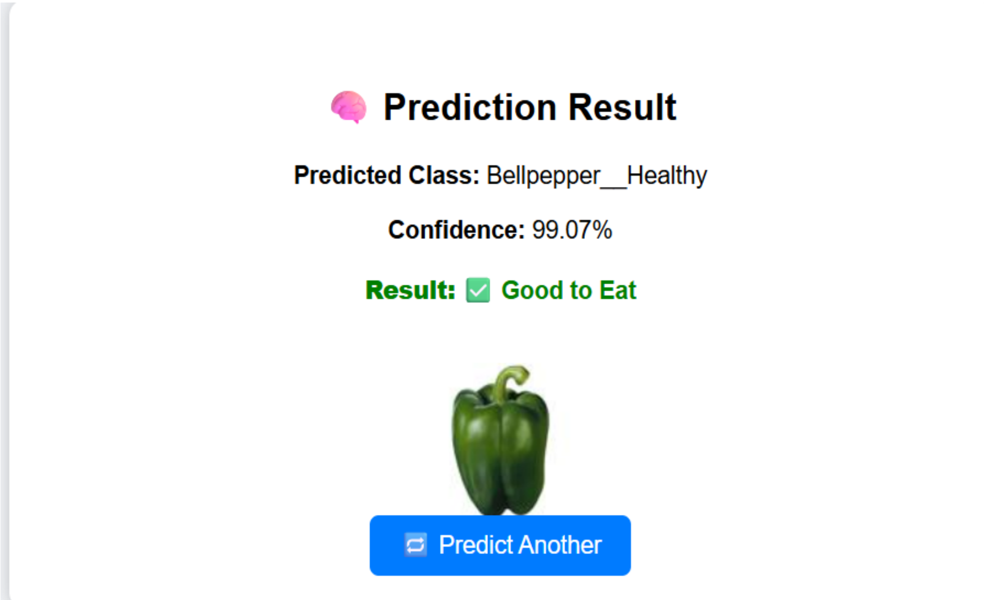


9.2 By result.html

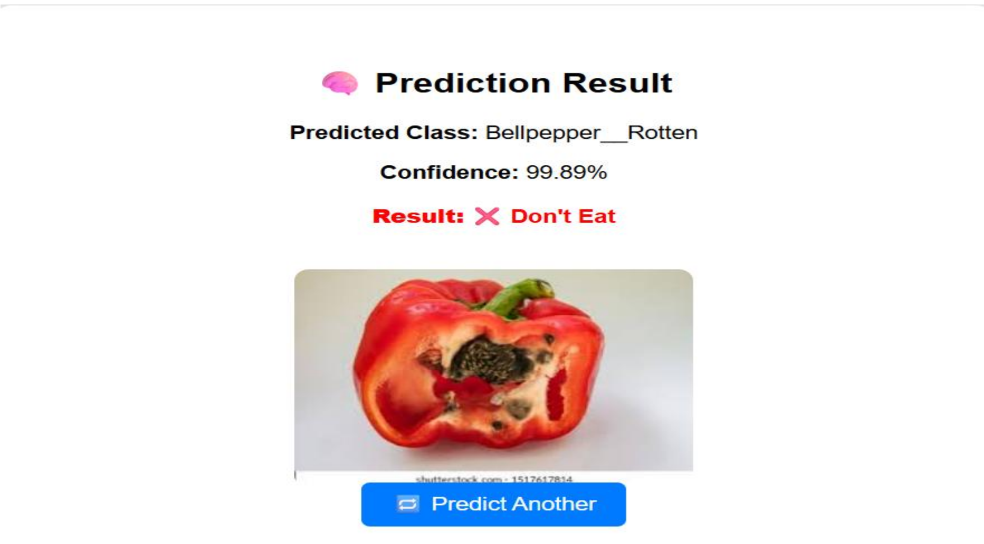
- Choose a file from the trained dataset to predict Rotten or Health



- Predicted Healthy – Good to Eat

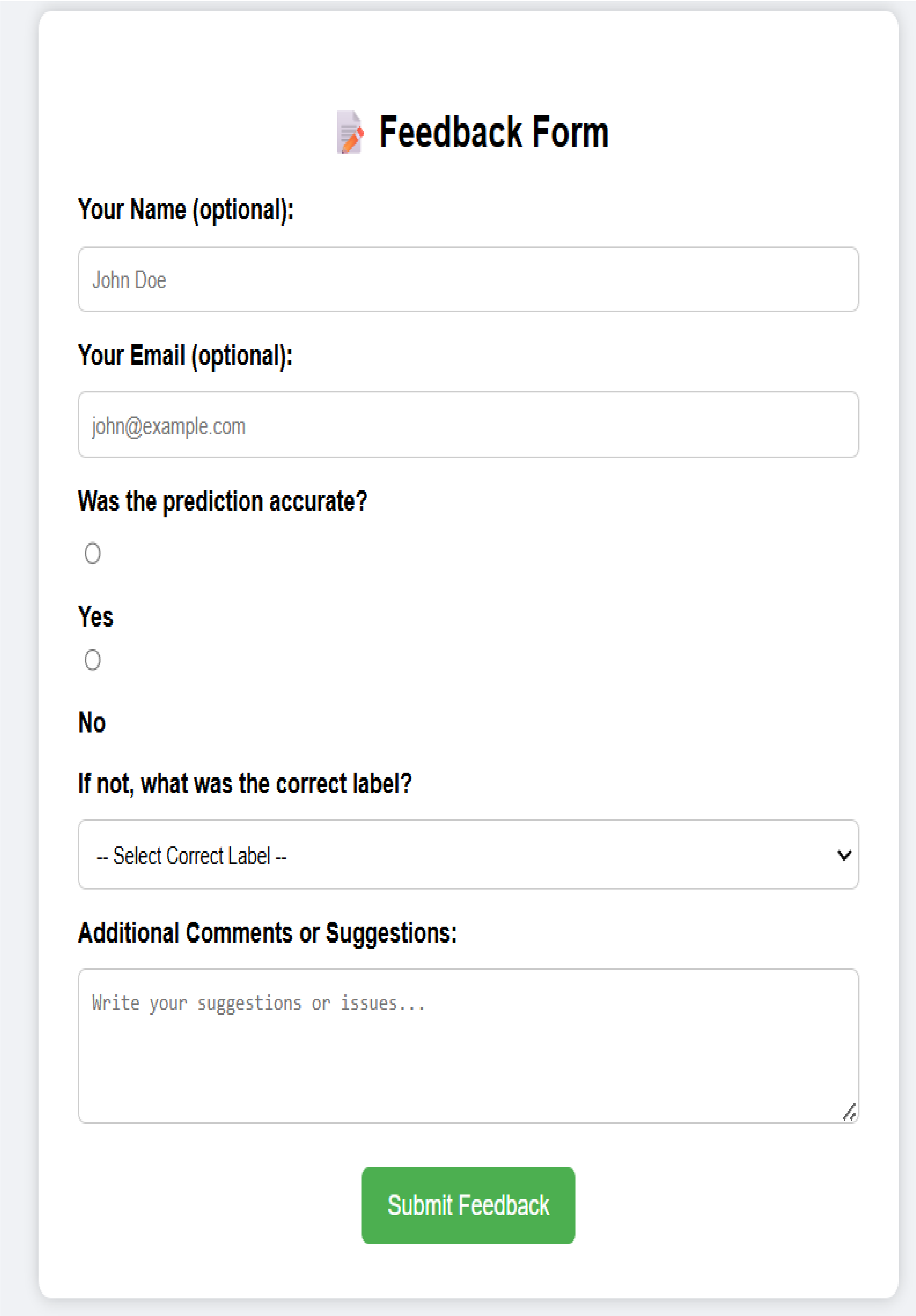


- Predicted Rotten – Don't Eat



9.3 By Feedback.html

- Feedback Form – Form Clients



10. Testing

10.1 Data Preparation & Sample Validation

- **Performed by**: *Mallela Suguna*
- **Responsibilities**:
    - Cleaned and structured dataset from Kaggle
    - Organized class-wise folders for healthy/rotten images
    - Manually checked 28-class image distribution for balance
- **Validation**:
    - Verified preprocessed images
    - Helped ensure training/validation split had proper class representation

10.2 Model Testing

- **Performed by**: *Durga Challa*
- **Responsibilities**:
    - Validated model performance on validation data
    - Measured accuracy, confidence score, and class prediction reliability
- **Results**:
    - **Training Accuracy**: 88.49%
    - **Validation Accuracy**: 88.26%
- **Tools & Libraries**:
    - TensorFlow 2.10,
    - Keras 2.10,
    - Scikit-learn
- **Techniques Used**:
    - Confusion matrix
    - Precision, Recall, F1 Score
    - Manual verification using test images

10.3 Flask & UI Integration Testing

- **Performed by**: *M M Bhavesh H R K*
- **Responsibilities**:
    - Integrated Flask with HTML templates
    - Ensured routing between /, /predict, and /feedback
    - Handled static assets (uploaded images, result images)
- **Tools Used**:
    - Flask debug server
    - Browser-based testing with real-time uploads

10.4 Web Application Testing & Feedback

- **Performed by**: *Poornima D C*
- **Responsibilities**:
    - Functional testing of the entire prediction and feedback flow
    - Validated UI pages and error handling
- **Test Cases**:
    - Upload valid/invalid image formats
    - Verify result and feedback routing
    - Test data storage in feedback_data/ (folder)

11. Demo

https://drive.google.com/file/d/1quqM_IQEYPyNyhenzheFxCVefvk2eQg1/view?usp=drive_link

12. Known Issues

- By split and copied the all the trained images and finally, I have fixed the issues.

```python
import os
import shutil
import random

# Paths
source_dir = r'C:\Users\JNTUK UCEK\OneDrive\Desktop\Smart Sorting\dataset\Fruit And Vegetable Diseases Dataset'
destination_dir = r'C:\Users\JNTUK UCEK\OneDrive\Desktop\Smart Sorting\Fruit and Vegetable Diseases Dataset'

train_dir = os.path.join(destination_dir, 'train')
val_dir = os.path.join(destination_dir, 'validation')

# Create train and validation directories
for base_dir in [train_dir, val_dir]:
    os.makedirs(base_dir, exist_ok=True)

# Split ratio
split_ratio = 0.8

# Loop through all class folders
for class_name in os.listdir(source_dir):
    class_path = os.path.join(source_dir, class_name)

    if os.path.isdir(class_path):
        images = [f for f in os.listdir(class_path) if f.lower().endswith(('.jpg', '.jpeg', '.png'))]
        random.shuffle(images)

        split_index = int(len(images) * split_ratio)
        train_images = images[:split_index]
        val_images = images[split_index:]

        # Create class subdirectories
        train_class_dir = os.path.join(train_dir, class_name)
        val_class_dir = os.path.join(val_dir, class_name)
        os.makedirs(train_class_dir, exist_ok=True)
        os.makedirs(val_class_dir, exist_ok=True)

        # Copy images to train folder
        for img in train_images:
            src = os.path.join(class_path, img)
            dst = os.path.join(train_class_dir, img)
            shutil.copy2(src, dst)

        # Copy images to validation folder
        for img in val_images:
            src = os.path.join(class_path, img)
            dst = os.path.join(val_class_dir, img)
            shutil.copy2(src, dst)

        print(f"✅ {class_name}: {len(train_images)} train, {len(val_images)} validation")

print("\n🎉 All classes split and copied successfully!")
```

```
✅ Apple__Healthy: 1950 train, 488 validation
✅ Apple__Rotten: 2340 train, 585 validation
✅ Banana__Healthy: 1599 train, 400 validation
✅ Banana__Rotten: 2237 train, 560 validation
✅ Bellpepper__Healthy: 488 train, 123 validation
✅ Bellpepper__Rotten: 472 train, 119 validation
✅ Carrot__Healthy: 495 train, 124 validation
✅ Carrot__Rotten: 463 train, 116 validation
✅ Cucumber__Healthy: 486 train, 122 validation
✅ Cucumber__Rotten: 474 train, 119 validation
✅ Grape__Healthy: 160 train, 40 validation
✅ Grape__Rotten: 160 train, 40 validation
✅ Guava__Healthy: 160 train, 40 validation
✅ Guava__Rotten: 160 train, 40 validation
✅ Jujube__Healthy: 160 train, 40 validation
✅ Jujube__Rotten: 160 train, 40 validation
✅ Mango__Healthy: 1450 train, 363 validation
✅ Mango__Rotten: 1797 train, 450 validation
✅ Orange__Healthy: 1660 train, 415 validation
✅ Orange__Rotten: 1748 train, 438 validation
✅ Pomegranate__Healthy: 160 train, 40 validation
✅ Pomegranate__Rotten: 160 train, 40 validation
✅ Potato__Healthy: 491 train, 123 validation
✅ Potato__Rotten: 467 train, 117 validation
✅ Strawberry__Healthy: 1282 train, 321 validation
✅ Strawberry__Rotten: 1276 train, 320 validation
✅ Tomato__Healthy: 483 train, 121 validation
✅ Tomato__Rotten: 476 train, 119 validation

🎉 All classes split and copied successfully!
```

- At last, all the trained dataset images are saved Succesfully

```python
# Step 14: Import required libraries
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout
from tensorflow.keras.preprocessing.image import ImageDataGenerator
import json

# Step 15: Set up paths
train_dir = r"C:\\Users\\JNTUK UCEK\\OneDrive\\Desktop\\Smart Sorting\\Fruit and Vegetable Diseases Dataset\\train"
val_dir = r"C:\\Users\\JNTUK UCEK\\OneDrive\\Desktop\\Smart Sorting\\Fruit and Vegetable Diseases Dataset\\validation"

# Step 16: Create data generators
train_datagen = ImageDataGenerator(rescale=1./255)
val_datagen = ImageDataGenerator(rescale=1./255)

train_generator = train_datagen.flow_from_directory(
    train_dir,
    target_size=(224, 224),
    batch_size=32,
    class_mode='categorical'
)

validation_generator = val_datagen.flow_from_directory(
    val_dir,
    target_size=(224, 224),
    batch_size=32,
    class_mode='categorical'
)

# Step 17: Build and compile the CNN model
model = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(224, 224, 3)),
    MaxPooling2D(2, 2),
    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D(2, 2),
    Conv2D(128, (3, 3), activation='relu'),
    MaxPooling2D(2, 2),
    Flatten(),
    Dense(128, activation='relu'),
    Dropout(0.5),
    Dense(train_generator.num_classes, activation='softmax')
])

model.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])

# Save class names
class_names = list(train_generator.class_indices.keys())
with open("class_names.json", "w") as f:
    json.dump(class_names, f)

# Train the model
history = model.fit(
    train_generator,
    epochs=10,
    validation_data=validation_generator
)

# Save the trained model
model.save("fruit_veg_disease_model.keras")
print("✅ Model trained and saved successfully!")
```

```
Found 23417 images belonging to 29 classes.
Found 5867 images belonging to 29 classes.
C:\ProgramData\anaconda3\Lib\site-packages\keras\src\layers\convolutional\base_conv.py:113: UserWarning: Do not pass an `input_shape`/`input_dim` ar
gument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
C:\ProgramData\anaconda3\Lib\site-packages\keras\src\trainers\data_adapters\py_dataset_adapter.py:121: UserWarning: Your `PyDataset` class should ca
ll `super().__init__(**kwargs)` in its constructor. `**kwargs` can include `workers`, `use_multiprocessing`, `max_queue_size`. Do not pass these arg
uments to `fit()`, as they will be ignored.
  self._warn_if_super_not_called()
Epoch 1/10
301/732 ━━━━━━━━━━━━━━━━━━━━ 9:32 1s/step - accuracy: 0.2145 - loss: 2.7656
C:\ProgramData\anaconda3\Lib\site-packages\PIL\Image.py:1056: UserWarning: Palette images with Transparency expressed in bytes should be converted t
o RGBA images
  warnings.warn(
732/732 ━━━━━━━━━━━━━━━━━━━━ 1148s 2s/step - accuracy: 0.3190 - loss: 2.3590 - val_accuracy: 0.6934 - val_loss: 1.0414
Epoch 2/10
732/732 ━━━━━━━━━━━━━━━━━━━━ 851s 1s/step - accuracy: 0.6298 - loss: 1.2333 - val_accuracy: 0.7825 - val_loss: 0.7461
Epoch 3/10
732/732 ━━━━━━━━━━━━━━━━━━━━ 830s 1s/step - accuracy: 0.7024 - loss: 0.9598 - val_accuracy: 0.8019 - val_loss: 0.6446
Epoch 4/10
732/732 ━━━━━━━━━━━━━━━━━━━━ 766s 1s/step - accuracy: 0.7562 - loss: 0.7607 - val_accuracy: 0.8120 - val_loss: 0.6214
Epoch 5/10
732/732 ━━━━━━━━━━━━━━━━━━━━ 731s 999ms/step - accuracy: 0.7931 - loss: 0.6454 - val_accuracy: 0.8478 - val_loss: 0.5084
Epoch 6/10
732/732 ━━━━━━━━━━━━━━━━━━━━ 730s 997ms/step - accuracy: 0.8185 - loss: 0.5370 - val_accuracy: 0.8483 - val_loss: 0.5076
Epoch 7/10
732/732 ━━━━━━━━━━━━━━━━━━━━ 730s 998ms/step - accuracy: 0.8449 - loss: 0.4634 - val_accuracy: 0.8596 - val_loss: 0.5148
Epoch 8/10
732/732 ━━━━━━━━━━━━━━━━━━━━ 723s 988ms/step - accuracy: 0.8626 - loss: 0.4120 - val_accuracy: 0.8587 - val_loss: 0.4980
Epoch 9/10
732/732 ━━━━━━━━━━━━━━━━━━━━ 793s 1s/step - accuracy: 0.8652 - loss: 0.3868 - val_accuracy: 0.8749 - val_loss: 0.4561
Epoch 10/10
732/732 ━━━━━━━━━━━━━━━━━━━━ 803s 1s/step - accuracy: 0.8849 - loss: 0.3402 - val_accuracy: 0.8826 - val_loss: 0.4252
✅ Model trained and saved successfully!
```

13. Future Enhancements

- Responsive frontend using React.js
- Admin dashboard to review feedback
- Feedback-based model retraining
- Real-time camera integration
- Model compression for mobile support