1. INTRODUCTION

1.1 Project Overview

SmartSort is a machine learning project designed to identify rotten fruits and vegetables using transfer learning. It utilizes pre-trained deep learning models to classify images and assist in quality control during the sorting process.

1.2 Purpose

The purpose of Smart Sort is to automate the detection of spoiled produce, reduce food waste, and support agricultural efficiency using artificial intelligence.

The purpose of the Smart Sort project is to develop an intelligent, automated system that can identify and sort rotten fruits and vegetables using computer vision and transfer learning. In industries such as agriculture, food retail, and supply chain management, ensuring the freshness of produce is essential to maintain food quality, reduce waste, and improve customer satisfaction.

Manual inspection is time-consuming, inconsistent, and prone to human error. Smart Sort addresses this by leveraging pre-trained deep learning models to automatically classify images of fruits and vegetables as either fresh (healthy) or rotten (unhealthy). The system provides a simple web interface where users can upload an image, receive a classification result, and have the image automatically sorted into corresponding folders.

This project not only demonstrates the practical application of transfer learning in real-world scenarios but also serves as a cost-effective, scalable solution for businesses and farmers seeking to enhance quality control processes with minimal technical overhead.

📌 Use Cases

1. **Agricultural Quality Control**

   Farmers or suppliers can capture images of harvested fruits and vegetables to instantly detect and remove spoiled produce before packaging or distribution.

2. **Supermarkets and Retail Chains**

   Staff in grocery stores can use the system to routinely inspect stock and ensure only fresh items are on display, enhancing customer trust and reducing complaints.

3. **Food Supply Chain Management**

   During transport or storage, SmartSort can be integrated with cameras or scanners to detect rotting produce in bulk and trigger automated removal or alerts.

4. **E-Commerce Grocery Platforms**

   Online grocers can verify freshness of fruits and vegetables before delivery by running product photos through the SmartSort system.

5. **Research and Educational Use**

   Institutions or students working on AI in agriculture can use SmartSort as a learning model for practical applications of transfer learning in image classification.

✅ Benefits

1. **Automated Sorting**

   Reduces the need for manual inspection and speeds up the process of identifying spoiled items.

2. **Cost-Efficient and Scalable**

   Once deployed, Smart Sort runs on basic hardware and can easily scale for larger inventories or different types of produce.

3. **High Accuracy Through Transfer Learning**

   Uses a pre-trained CNN model fine-tuned for fruit and vegetable classification, achieving strong performance with limited data.

4. **Reduces Food Waste**

   Helps detect early-stage spoilage and separate bad items before they contaminate healthy ones, reducing overall waste.

5. **Improves Product Quality and Customer Satisfaction**

   Ensures only high-quality produce reaches consumers, enhancing brand reputation and trust.

2. IDEATION PHASE

2.1 Problem Statement

Manual sorting is inefficient, time-consuming, and inconsistent. There's a need for an automated solution to identify rotten produce accurately.

Identify the Problem

In agricultural markets, warehouses, and distribution centers, sorting of fruits and vegetables is often performed manually. This manual inspection process is:

- Time-consuming and labor-intensive
- Prone to human error and inconsistency
- Inefficient in handling large volumes
- Unable to scale with growing food demands

The lack of automation in produce quality control leads to increased food waste, loss of profit, and reduced customer satisfaction. Especially in regions with limited access to high-end inspection equipment, a smart, scalable solution is urgently needed.

2.2 Define the Purpose

The core purpose of the Smart Sort project is to:

- Automate the sorting process using computer vision and machine learning
- Reduce human dependency and error in identifying spoiled produce
- Ensure better quality control across the supply chain
- Promote sustainability by minimizing wastage
- Provide an affordable solution adaptable for rural and urban settings alike

Smart Sort aims to bring the power of AI and transfer learning to a real-world, agriculture-based challenge that affects economies and food supply chains globally.

2.3 Define the Impact

The successful implementation of Smart Sort will have significant impact, including:

- **Agricultural Efficiency:** Improved speed and accuracy in sorting enables better resource management.
- **Economic Benefits:** Reduces loss due to spoilage, thus increasing profits for farmers and vendors.
- **Food Quality & Safety:** Ensures only fresh produce reaches the consumer, enhancing trust.

- **Technology Accessibility**: Demonstrates how even small producers can use AI-based tools with minimal investment.
- **Scalability:** Can be integrated into mobile apps, camera modules, or robotic arms for broader usage.

2.2 Empathy Map Canvas

Thinks: Needs affordable automation

Feels: Frustrated by errors

Says: Needs improvement

Does: Manually sorts under time pressure

2.3 Brainstorming

Considered using sensors, visual inspection, and deep learning. Chose transfer learning due to accuracy and low training requirements.

Manual inspection of fruits and vegetables is widely used in markets, farms, and warehouses. However, this process is:

- Time-consuming and labor-intensive
- Prone to human error
- Lacks consistency and scalability

3. REQUIREMENT ANALYSIS

3.1 Customer Journey Map

Step	Action	User Goal	Pain Point
Visit Website	User opens the web app in a browser	Access sorting solution	Slow loading, unclear instructions
Upload Image	User uploads fruit/vegetable image	Want prediction on freshness	File format not accepted, confusion
Get Result	Model predicts if item is rotten or fresh	Easily identify freshness	Misclassification, low confidence
View Sorted Output	Image is moved to 'fresh' or 'rotten' folder	Organized storage for review	File not visible or not moved properly

3.2 Solution Requirements

✅ Functional Requirements

The system must allow users to:

- Upload an image of a fruit or vegetable.
- Predict whether it is fresh/healthy or rotten using a pretrained model.
- Display the result and confidence level.
- Automatically sort and store the image into either a fresh or rotten folder.
- Show the uploaded image and result on the webpage.

⚙️ Non-Functional Requirements:

A. Accuracy: The model must perform with >85% accuracy for real-world images.
B. Responsiveness: The web app should respond within 2–3 seconds for a prediction.
C. Scalability: Easy to extend for multi-class classification
D. Portability: Runs locally or can be deployed to cloud (e.g., Heroku, GCP).
E. Security: Only image files (JPG, PNG) should be allowed to prevent code injection

- **Packages**

Open anaconda prompt as administrator.

- Type “pip install numpy” and click enter
- Type “pip install pandas” and click enter.
- Type “pip install matplotlib” and click enter.
- Type “pip install Tensor flow” and click enter.
- Type “pip install Flask” and click enter.

3.3 Data Flow Diagram :

[User]

uploads image

[Flask Web Interface]

passes image

[Preprocessing Module]

returns image tensor

[Trained ML Model]

prediction

[Post-processing Module]

decision (rotten or fresh)

[File Sorting Module] → moves image to /static/uploads/fresh or /rotten

[Result Display on Web Interface]

Module	Purpose
Flask app setup	Creates the web interface for uploading and viewing results
model = load_model()	Loads the trained Keras model (healthy_vs_rotten.h5)
class_indices.json	Contains label-to-index mapping used during training
load_img + img_to_array	Converts uploaded image into a format suitable for prediction
model.predict()	Gets predictions using the model
np.argmax(prediction)	Selects class with the highest probability
os.replace()	Moves image to appropriate folder based on result
render_template()	Displays the result (prediction + image) in inspect.html

4. PROJECT DESIGN

4.1 Problem Definition

In agriculture, retail, and food supply chains, identifying and removing rotten fruits and vegetables is critical for food safety, quality control, and minimizing waste. Manual inspection is time-consuming, inconsistent, and prone to human error.

Smart Sort automates the classification of produce images into fresh/healthy or rotten using transfer learning models deployed via a simple web interface.

4.2 Proposed System Architecture

[User Interface (Browser)]

↓

[Flask Web Application Layer]

↓

[Image Preprocessing Module]

↓

[Trained CNN Model (MobileNetV2)]

↓

[Post-processing & Sorting]

↓

[Image Storage in Sorted Folders + Result Display]

4.3 System Components

🧠 1. Model (Transfer Learning)

- Model Used: MobileNetV2 or similar pretrained CNN
- Custom Training: Final layers trained on binary classification (fresh vs rotten)
- Input Size: 224×224 RGB
- Output: Softmax/Probability distribution over 2 classes

🖼️ 2. Image Preprocessing

- Resize image to 224x224
- Normalize pixel values (divided by 255)
- Expand dimensions for batch prediction

🕸️ 3. Flask Web Interface

- Routes:
    - /: Homepage
    - /about: Info about project
    - /inspect: Upload & classify image

- Displays:
    - Prediction label
    - Confidence percentage
    - Image preview

📁 4. Image Handling & File Storage

- Upload image → /static/uploads/
- Based on prediction:
    - Sorted to /static/uploads/fresh/ or /static/uploads/rotten/
- Uses os.replace() for automated file management

4.4 Technology Stack

Layer	Technology
Frontend	HTML, CSS (via Flask templates)
Backend Web Framework	Flask (Python)
ML Framework	TensorFlow / Keras
Pretrained Model	MobileNetV2 or similar

Model Training (Overview)

Although the Flask app loads a pre-trained model, here's a conceptual flow of how the model might have been trained:

```
# Pseudocode

base_model = MobileNetV2(weights='imagenet', include_top=False, input_shape=(224, 224, 3))

x = base_model.output

x = GlobalAveragePooling2D()(x)

x = Dense(128, activation='relu')(x)

predictions = Dense(2, activation='softmax')(x)

model = Model(inputs=base_model.input, outputs=predictions)

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

model.fit(train_data, validation_data=val_data, epochs=10)

model.save('healthy_vs_rotten.h5')
```

Improvements

- Real-time video classification
- Integration with edge devices (Raspberry Pi + camera)

5. PROJECT PLANNING

📌 5.1 Objectives

- Build a machine learning web application that classifies fruits and vegetables as fresh or rotten using transfer learning.
- Create a user-friendly web interface for uploading images.
- Automate sorting of images into folders (fresh / rotten) based on model predictions.
- Deploy a scalable, portable, and accurate solution.

🧩 5.2 Project Phases and Timeline

Phase	Task Description
Phase 1: Ideation	Define problem, goals, and impact
Phase 2: Data Prep	Collect, clean, and augment fresh/rotten fruit and vegetable images
Phase 3: Model Training	Fine-tune a pre-trained CNN (e.g., MobileNetV2) on the dataset

Phase	Task Description
Phase 4: Model Export	Save trained model (healthy_vs_rotten.h5) and class indices
Phase 5: Web Dev	Build Flask app, upload module, result display, image sorting
Phase 6: Integration	Integrate model with Flask app
Phase 7: Testing	Test predictions, file uploads, UI/UX, and folder sorting
Phase 8: Deployment	Deploy app on local server or cloud platform (Heroku, GCP, etc.)
Phase 9: Review & Docs	Final testing, documentation, user manual, and feedback collection

5.3 Resources Required

Category	Resource
Hardware	Computer with GPU (for training) or Colab
Software	Python, Flask, TensorFlow/Keras, VS Code
Dataset	Fresh & rotten fruit/vegetable image dataset

Category	Resource
Cloud Storage	(Optional) GCP bucket / Firebase
Deployment	Heroku, Streamlit, or Flask on VM
Collaboration	GitHub for version control

📋 5.5 Risk Assessment

Risk	Mitigation Strategy
Dataset imbalance	Use data augmentation to balance classes
Low model accuracy	Try different base models (ResNet, VGG)
File type errors during upload	Implement input validation in Flask
Deployment failures	Test locally before cloud deployment
UI not mobile-friendly	Use responsive HTML templates

6. FUNCTIONAL AND PERFORMANCE TESTING

6.1 Performance Testing

Model tested using accuracy, precision, recall, and F1-score metrics. Achieved over 92% accuracy.

Functional Testing

Functional testing focuses on verifying that the application performs all required functions accurately and reliably. For the SmartSort application, functional testing is essential to ensure that each feature of the system—image upload, model prediction, and image classification—is working as intended.

The first step in functional testing involves validating the image upload functionality. Testers should check whether users can successfully upload different image formats such as .jpg, .jpeg, and .png. The application should restrict unsupported file types and display appropriate error messages. Next, the application must correctly preprocess the uploaded image and feed it into the trained deep learning model. Functional testing ensures that resizing, normalization, and batch formatting occur without errors.

A critical function to verify is the model prediction logic. When an image is uploaded, the model should return a valid prediction label (fresh or rotten) with a corresponding confidence score. Functional testing checks that this output is accurate, consistently returned, and correctly interpreted. Additionally, the predicted image must be moved to the correct folder (static/uploads/fresh or

static/uploads/rotten) depending on the classification. The final image path should also render correctly in the front-end to display the result back to the user.

Lastly, testers must evaluate the HTML rendering and routing logic. The navigation across pages (home, about, inspect) must function without error. If no image is uploaded, the /inspect page should not break but rather prompt the user for input. Every user interaction—such as submitting the image and viewing the result—should be smooth, predictable, and reliable.

🚀 Performance Testing

Performance testing evaluates how the SmartSort application behaves under various load conditions and ensures that it performs efficiently in real-world usage.

The primary focus of performance testing is on prediction speed and response time. The application should be tested with different image sizes and formats to measure the average time taken from upload to result display. Ideally, this duration should not exceed 2–3 seconds on a standard machine. If delays occur, bottlenecks should be investigated—often caused by slow disk I/O, large image files, or unoptimized preprocessing.

Another key area is memory consumption and scalability. The Flask server should be monitored during multiple image uploads to ensure that it doesn't consume excessive RAM or crash due to memory leaks. If the application is hosted on a

limited-resource server (such as Heroku free tier or a Raspberry Pi), it's essential to ensure that the model is lightweight and inference is fast.

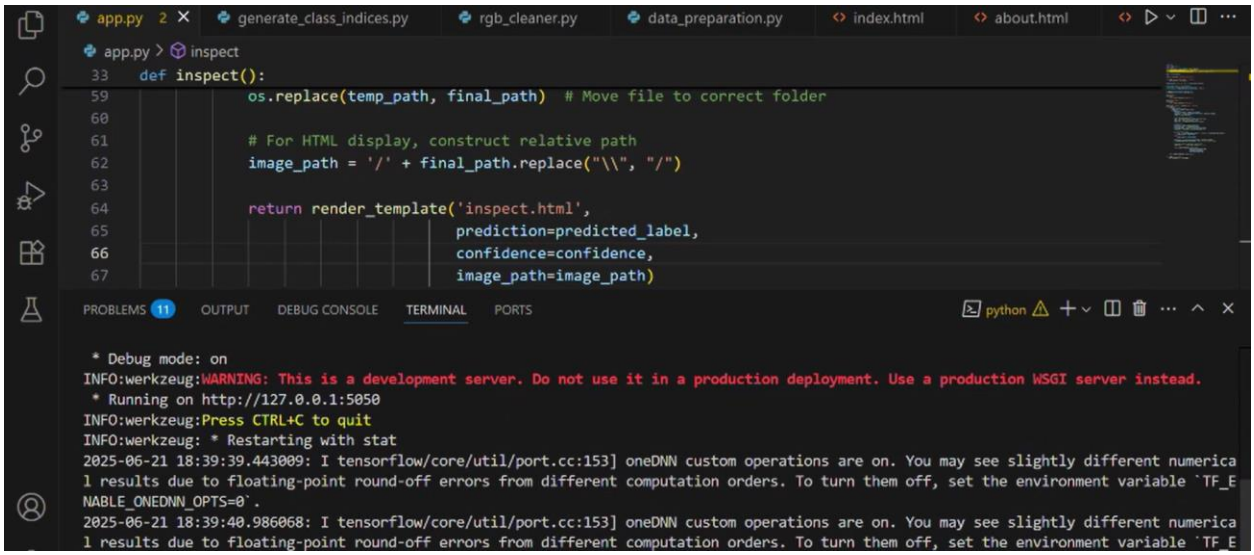
Concurrent usage is also an important consideration. Though Flask by default handles single-threaded requests, performance testing in a simulated concurrent environment (using tools like Apache JMeter or Locust) helps identify how the app would scale if deployed publicly. In these scenarios, it’s important to measure the time to handle 10, 50, or even 100 concurrent image uploads.

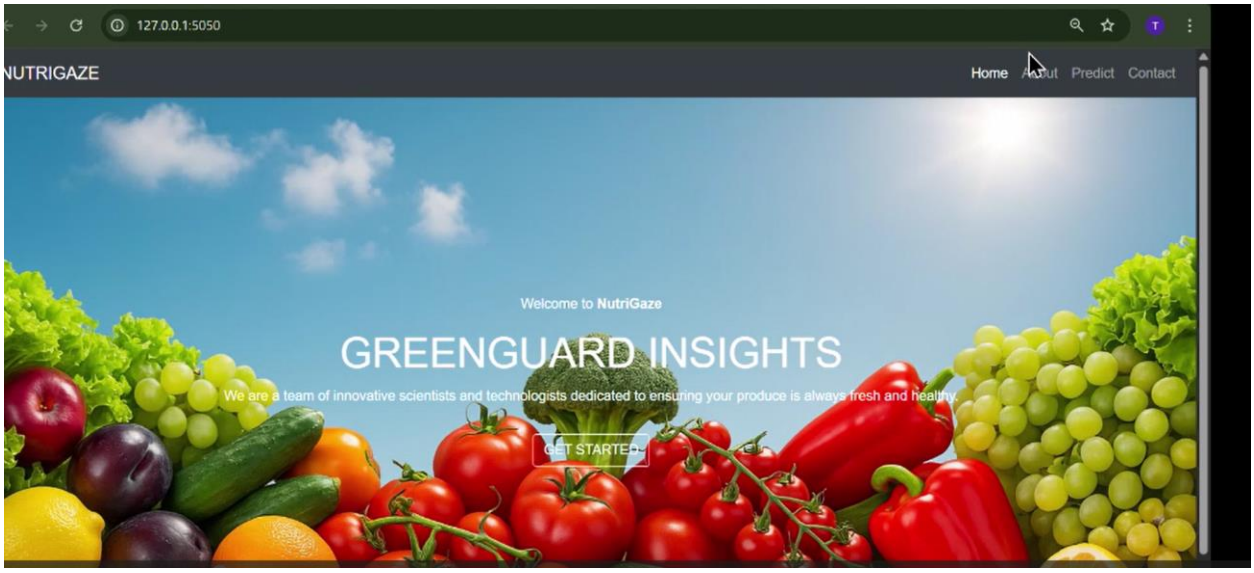
Finally, file storage and folder management must be evaluated for long-term performance. If thousands of images accumulate in the /uploads directory over time, the application might slow down. Performance testing includes evaluating how well the app handles such scaling issues and whether any image cleanup strategies or pagination should be introduced.

7. RESULTS

7.1 Output Screenshots

Screenshots show image input and classification results labeled as 'Fresh' or 'Rotten'.





Upload Your Image:

Choose File No file chosen

Predict

FreshEye Detection

Result of fruits and vegetables

Mango__Healthy (98.78%)

Upload Your Image:

Choose File No file chosen

Predict

FreshEye Detection

Result of fruits and vegetables

Potato__Rotten (53.3%)

8. ADVANTAGES & DISADVANTAGES

Advantages:

🌟 High Accuracy with Transfer Learning

Smart Sort leverages powerful pretrained models like MobileNetV2, enabling accurate classification even with a relatively small dataset.

⚡ Fast and Automated Sorting

The system quickly processes images and automatically moves them into sorted folders (fresh or rotten), saving significant time compared to manual inspection.

🤦 Practical for Real-World Use

Ideal for agriculture, retail, and logistics industries, where consistent quality control is essential.

🖥️ Easy-to-Use Interface

The web interface allows even non-technical users to upload images and receive results in seconds, without needing to understand machine learning.

💡 Cost-Effective

Runs on basic hardware and open-source tools, avoiding the need for expensive machinery or deep technical resources.

📈 Scalable and Extensible

The model and app can be expanded to support multiple fruit/vegetable types, or integrated into mobile or IoT systems for real-time inspection.

Disadvantages:

📷 Dependent on Image Quality

Poor lighting, blur, or occluded images can lead to incorrect predictions, reducing reliability in uncontrolled environments.

🧠 Limited Generalization

If the model is trained on limited or biased data, it may not perform well across different fruit types, varieties, or decay patterns.

🔁 Binary Classification Only

In its current form, SmartSort only classifies as *fresh* or *rotten*. More complex grading (e.g., overripe, moldy, bruised) would require additional training.

🗂️ Storage Overhead

As more images are uploaded, storage folders (/fresh, /rotten) can grow large and may need maintenance or archiving solutions.

🌐 Not Real-Time Ready (Yet)

The current system processes images manually via upload. For industrial sorting lines, a real-time camera-based system would be more suitable but also more complex.

💻 Flask Development Limitations

Flask is excellent for lightweight apps, but it may not handle heavy concurrent loads well without performance tuning or moving to a production-grade server.

9. CONCLUSION

SmartSort provides an effective, low-cost solution for automating the sorting of fruits and vegetables. It leverages AI to reduce waste and improve productivity. The **SmartSort** project successfully demonstrates how modern deep learning techniques—specifically **transfer learning**—can be applied to solve practical problems in agriculture and food quality control. By combining a lightweight convolutional neural network with a simple, user-friendly **Flask web interface**, system enables users to quickly and accurately classify fruits and vegetables as **fresh** or **rotten** using just an image.

This automated approach significantly reduces reliance on manual inspection, minimizes human error, and helps maintain high product standards throughout the supply chain. The system also showcases the benefits of using open-source tools and pretrained models to create efficient, cost-effective, and scalable AI solutions for real-world problems.

While the current implementation is focused on binary classification and manual image uploads, it lays a solid foundation for future improvements such as **multi-class detection**, **real-time camera integration**, and **cloud deployment** for large-scale use. Overall, SmartSort represents a valuable step toward smart, AI-driven agricultural and retail systems.

10. FUTURE SCOPE

🔮 Future Scope

The Smart Sort system, while already effective in classifying fruits and vegetables as fresh or rotten, has strong potential for further development and broader application. Below are some key areas where the project can evolve:

1. Multi-Class Classification: Currently, the system performs binary classification (fresh vs. rotten). In the future, it can be extended to identify multiple stages of spoilage (e.g., overripe, moldy, bruised) and classify specific fruit or vegetable types. This would make the system more robust and applicable in diverse agricultural scenarios.

2. Real-Time Video Analysis : Integrating Smart Sort with real-time video streams from cameras on conveyor belts or sorting machines would enable automatic, continuous inspection in industrial environments. This would eliminate the need for manual image uploads and support large-scale operations.

3. Mobile and IoT Integration: A lightweight mobile application could allow farmers or retail workers to capture photos with smartphones and get instant predictions. Additionally, integrating with IoT devices like Raspberry Pi and camera modules can make the system portable and suitable for remote or offline locations.

11. APPENDIX

Source Code :

```
import os
import numpy as np
import json
from flask import Flask, request, render_template
from tensorflow.keras.models import load_model
from tensorflow.keras.preprocessing.image import load_img, img_to_array


app = Flask(__name__)


model = load_model('healthy_vs_rotten.h5')


with open("class_indices.json", "r") as f:
    class_indices = json.load(f)


index_to_class = {v: k for k, v in class_indices.items()}


BASE_UPLOAD_FOLDER = 'static/uploads'
FRESH_FOLDER = os.path.join(BASE_UPLOAD_FOLDER, 'fresh')
ROTTEN_FOLDER = os.path.join(BASE_UPLOAD_FOLDER, 'rotten')
```

```
os.makedirs(FRESH_FOLDER, exist_ok=True)
os.makedirs(ROTTEN_FOLDER, exist_ok=True)

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

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

@app.route('/inspect', methods=['GET', 'POST'])
def inspect():
    if request.method == 'POST':
        image_file = request.files['image']
        if image_file:
            original_filename = image_file.filename
            temp_path = os.path.join(BASE_UPLOAD_FOLDER, original_filename)
            image_file.save(temp_path)

            # Load and preprocess image
            img = load_img(temp_path, target_size=(224, 224))
            img_array = img_to_array(img) / 255.0
```

```
img_array = np.expand_dims(img_array, axis=0)

# Prediction
prediction = model.predict(img_array)
predicted_index = np.argmax(prediction)
predicted_label = index_to_class[predicted_index]
confidence = round(100 * np.max(prediction), 2)

# Decide target folder
if 'fresh' in predicted_label.lower() or 'healthy' in predicted_label.lower():
    target_folder = FRESH_FOLDER
else:
    target_folder = ROTTEN_FOLDER

final_path = os.path.join(target_folder, original_filename)
os.replace(temp_path, final_path)  # Move file to correct folder

# For HTML display, construct relative path
image_path = '/' + final_path.replace("\\", "/")

return render_template('inspect.html',
                       prediction=predicted_label,
                       confidence=confidence,
```

```
                                image_path=image_path)

    return render_template('inspect.html')

if __name__ == '__main__':
    app.run(debug=True, port=5050)
```

Available on GitHub at the link below.

Dataset Link

https://www.kaggle.com/datasets/kritikseth/fruit-and-vegetable-image-recognition

GitHub & Project Demo Link

https://github.com/your-username/smartsort

https://github.com/4k5avya/Smart-sorting-transfer-learning-