

# Smart Sorting: Transfer Learning for Identifying Rotten Fruits and Vegetables

## Project Documentation format

### 1. Introduction

- **Project Title:** [Smart Sorting: Transfer Learning for Identifying Rotten Fruits and Vegetables]
- **Team Members:**
  1. YALAKANTI NARASIMHA [23HM5A0525]
  2. SHAIK MASOOD [23HM5A0518]
  3. SHAIK ZEBA [22HM1A05C4]
  4. URUKUNDAPAGARI RAMUDU [22HM1A05D7]

### 2. Project Overview

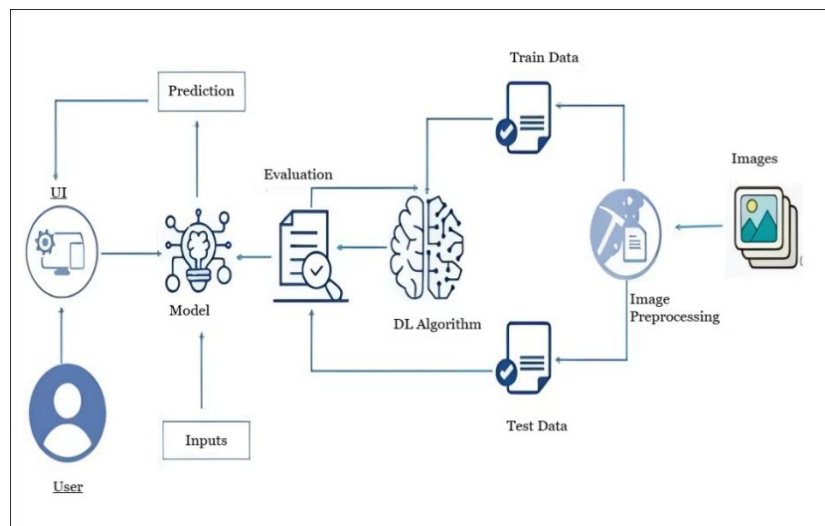
- **Purpose:**

The primary purpose of the Smart Sorting project is to automate the identification and sorting of rotten fruits and vegetables using transfer learning-based deep learning models, aiming to improve efficiency, accuracy, and scalability in food quality control systems. Demonstrate the power of AI and machine learning in solving agricultural problems. Use transfer learning to reduce the need for large datasets and training time, the system enables accurate, fast, and consistent detection of spoilage, ensuring only high-quality produce reaches retailers and consumers. Ultimately, the project promotes food safety, reduces wastage, enhances operational efficiency, and builds consumer trust in the food supply chain.
- **Goals:**
  - Reduce the reliance on manual inspection.
  - Enable real-time sorting of fruits and vegetables in industrial or retail settings.
  - To minimize human error and subjectivity in traditional manual sorting processes.
  - To reduce post-harvest losses by enabling early detection of spoilage.
  - Early detection of spoilage helps remove rotten items quickly, preserving surrounding produce.
  - To utilize deep learning (transfer learning with VGG16) for accurate, real-time classification.
  - To develop an automated image-based system for classifying fruits and vegetables as fresh or rotten.
  - To ensure consistent food quality, enhancing consumer safety and satisfaction.
  - To build a scalable and reliable solution that can be adapted to large-scale food supply chains.
  - To contribute towards reducing food wastage and promoting sustainable consumption.

- **Features:**

- **Transfer Learning with VGG16:** Utilizes a pre-trained VGG16 model, fine-tuned for high accuracy in detecting subtle signs of spoilage.
- **User-Friendly Web Interface:** Simple web application where users can upload images and receive immediate classification results.
- **Real-Time Prediction:** Fast and efficient classification suitable for real-time use in supermarkets, warehouses, and food processing plants.
- **Confidence Score Display:** Provides a confidence percentage with each prediction to indicate model certainty.
- **Supports Multiple Classes:** Capable of detecting freshness or spoilage across various fruits and vegetables, not limited to a single type.
- **Visual Result Presentation:** Displays the uploaded image along with the classification result for easy verification.
- **Extensibility:** Designed to allow easy addition of more fruit or vegetable classes in the future.
- **Prevention of Food Waste:** Helps in early detection of spoilage, reducing wastage and improving food quality control.
- **Scalable Dataset Handling:** Can work with large datasets, making it adaptable for industrial use.

### 3. Architecture



### 4. Setup Instructions

- **Prerequisites:**

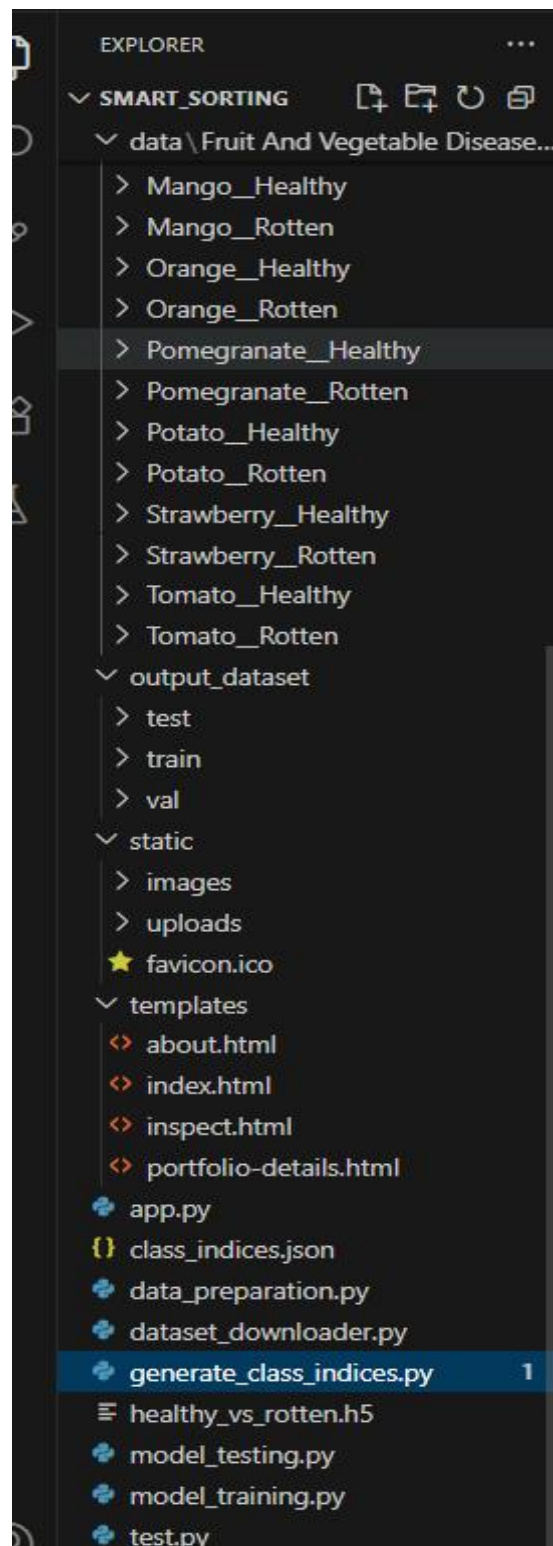
To complete this project, you must require the following software and packages.

- Software Requirements:
  - Visual Studio Code (VS Code) or any Python-supported IDE
  - Python 3.10 for better suitable to all packages
- Python packages:
  - Open VS code terminal prompt etc.,

- Type “pip install numpy” and click enter.
  - Type “pip install pandas” and click enter.
  - Type “pip install scikit-learn” and click enter.
  - Type “pip install matplotlib” and click enter.
  - Type “pip install scipy” and click enter.
  - Type “pip install seaborn” and click enter.
  - Type “pip install tensorflow” and click enter.
  - Type “pip install Flask” and click enter.
- **Installation:**
    1. Create a Virtual Environment (Optional but Recommended):  
`python -m venv .venv`  
`source .venv/Scripts/activate` For Windows  
`source .venv/bin/activate` For Mac/Linux
    2. Install Required Packages:  
Ensure you run:  
`pip install numpy`  
`pip install pandas`  
`pip install scikit-learn`  
`pip install matplotlib`  
`pip install scipy`  
`pip install seaborn`  
`pip install tensorflow`  
`pip install Flask`  
`pip install Pillow`
    3. Download Dataset:  
Place the dataset inside the `data` folder or run your `dataset\_downloader.py` to fetch from Kaggle.
    4. Prepare the Dataset:  
`python data_preparation.py`
    5. Clean the Dataset (Optional for Transparency Issues):  
`python rgb_cleaner.py`
    6. Train the Model:  
`python model_training.py`
    7. Test the Model (Optional):  
`python model_testing.py`
    8. Run the Flask Web Application:  
`python app.py`
    9. Access the Application:  
Open your browser and visit: [\[http://127.0.0.1:5050\]](http://127.0.0.1:5050)

**NOTE:** After successful setup, you can upload fruit or vegetable images through the web interface to detect freshness or spoilage.

## 5. Folder Structure



## 6. Running the Application

### A. Backend (Flask Application):

The Flask backend serves both the API and the frontend templates (HTML pages).

Commands to run the backend:

Open the terminal (preferably inside your project directory).

Activate the virtual environment if created:

```
source .venv/Scripts/activate # For Windows
```

```
source .venv/bin/activate # For Mac/Linux
```

Run the Flask application:

```
python app.py
```

The server will start at:

```
http://127.0.0.1:5050
```

### B. Frontend:

The project uses Flask's Jinja templates (index.html, about.html, inspect.html) for the frontend. No separate command is needed for the frontend. Once the Flask server starts, the frontend will be accessible through your web browser at:

<http://127.0.0.1:5050>

The following endpoints are exposed by the Flask backend for the NutriGaze application:

## 7. API Documentation

### a) Home Page

- **URL:** /
- **Method:** GET
- **Description:** Displays the landing page of the application.
- **Response:** Returns the index.html template.

### b) About Page

- **URL:** /about
- **Method:** GET
- **Description:** Displays information about the project.
- **Response:** Returns the about.html template.

### c) Inspect Page (Upload & Predict UI)

- **URL:** /inspect
- **Method:** GET
- **Description:** Displays the image upload form for prediction.
- **Response:** Returns the inspect.html template.

### d) Image Prediction API

- **URL:** /predict
- **Method:** POST
- **Description:** Accepts an image file, performs classification using the trained model, and returns the prediction.

#### Request Parameters:

Name	Type	Description
image	File	The image file to be uploaded (JPG, PNG, etc.).

#### Example Request using HTML Form:

```
<form method="POST" action="/predict" enctype="multipart/form-data">
  <input type="file" name="image" required>
  <button type="submit">Predict</button>
</form>
```

#### Example Response (Rendered on Inspect Page):

Upon successful prediction, the following details are displayed:

- Uploaded image preview
- Predicted class label (e.g., "Apple\_\_Healthy")
- Confidence score (percentage)

#### Example Backend Response (if it were JSON API):

(Note: Your current implementation renders a template, but if converted to pure JSON API, it would look like this)

```
{ "predicted_label": "Banana__Rotten",
  "confidence": 97.35,
  "image_path": "static/uploads/banana.jpg"
}
```

## 8. Authentication

### Current Status:

The current version of the NutriGaze project does not implement authentication or authorization mechanisms. The application is designed as a publicly accessible image classification tool intended for demonstration purposes, allowing any user to:

- ✓ Access the website
- ✓ Upload images for freshness classification
- ✓ View results without requiring login or registration

### Future Scope for Authentication (Optional Enhancements):

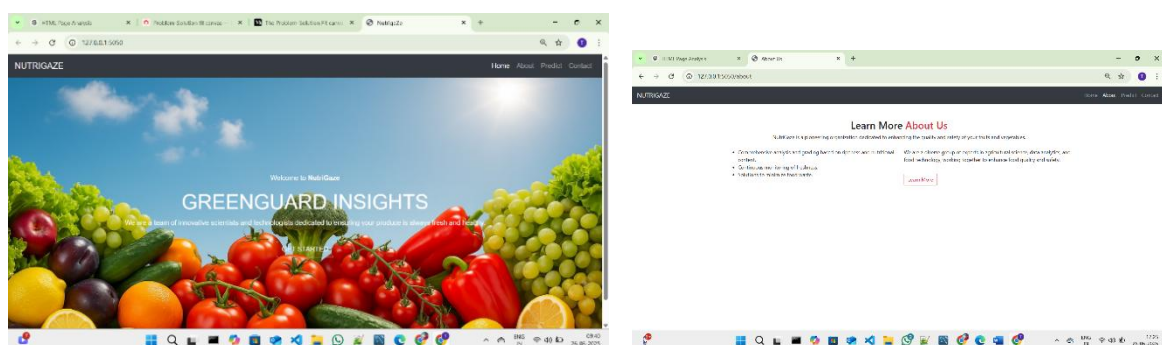
For improved security and controlled access in future versions, the following authentication and authorization methods can be implemented:

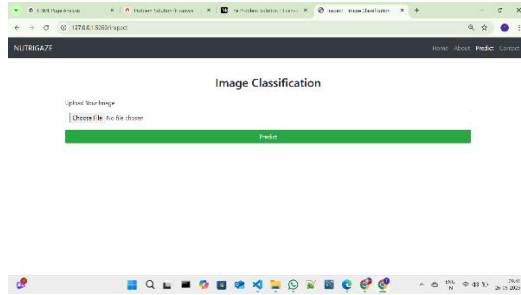
Method	Description
<b>Session-Based Authentication</b>	User credentials are verified on login, and a secure session is maintained using cookies.
<b>Token-Based Authentication (JWT)</b>	Users receive a JSON Web Token (JWT) upon successful login, which must be provided with each request to access protected resources.
<b>Role-Based Access Control (RBAC)</b>	Different user roles (e.g., Admin, Food Plant Head, Supermarket Manager) can be defined to restrict or grant access to specific features.

### Recommended Future Features:

- ✓ Admin Login: Only authorized personnel can retrain or upload new datasets.
- ✓ User Dashboard: Registered users can track prediction history.
- ✓ API Access Tokens: Protect REST APIs for mobile or external application integration.

## 9. User Interface





## 10. Testing

### Testing Strategy

To ensure the robustness and accuracy of the NutriGaze fruit and vegetable freshness classification system, a combination of manual and automated testing techniques were used:

- **Model Evaluation Testing:**
  - Evaluated the trained deep learning model using separate validation and test datasets.
  - Performance metrics such as accuracy, confusion matrix, and classification report were generated.
- **External Image Testing:**
  - Manually tested the model by uploading real-world images not included in the training dataset.
  - Observed model predictions to assess generalization ability.
- **Web Application Functional Testing:**
  - Tested all web pages (Home, About, Predict) for correct navigation and content display.
  - Verified image upload functionality through the web interface.

### Tools Used

Tool/Library	Purpose
TensorFlow/Keras	Model training, evaluation, and predictions.
scikit-learn	Generated confusion matrix and classification reports.
Matplotlib/Seaborn	Visualization of confusion matrix and performance metrics.
Flask (Debug Mode)	Web application testing and debugging.
Manual Testing	Verified functionality of file uploads and predictions through browser.

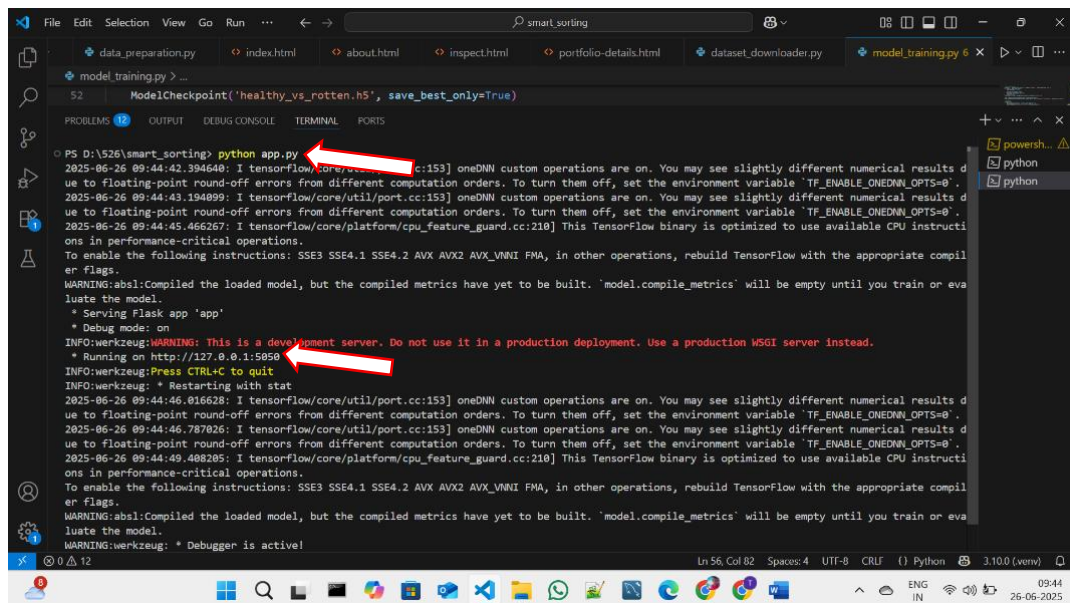


## 11. Screenshots or Demo

### Screenshots

The complete execution of the Smart Sorting application is shown in the images step by step as shown below.

**Step 1:** Run the app.py code and you will get a link in terminal as <https://127.0.0.1:5050> to access web page and to do the other process.



```
File Edit Selection View Go Run ... smart_sorting
data_preparation.py index.html about.html inspect.html portfolio-details.html dataset_downloader.py model_training.py x
model_training.py > ...
52 ModelCheckpoint('healthy_vs_rotten.h5', save_best_only=True)
PROBLEMS 12 OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS D:\526\smart_sorting> python app.py
2025-06-26 09:44:42.394648: I tensorflow/core/util/port.cc:153] oneDNN custom operations are on. You may see slightly different numerical results due to floating-point round-off errors from different computation orders. To turn them off, set the environment variable 'TF_ENABLE_ONEDNN_OPTS=0'.
2025-06-26 09:44:43.194899: I tensorflow/core/util/port.cc:153] oneDNN custom operations are on. You may see slightly different numerical results due to floating-point round-off errors from different computation orders. To turn them off, set the environment variable 'TF_ENABLE_ONEDNN_OPTS=0'.
2025-06-26 09:44:45.466267: I tensorflow/core/platform/cpu_feature_guard.cc:210] This TensorFlow binary is optimized to use available CPU instructions in performance-critical operations.
To enable the following instructions: SSE3 SSE4.1 SSE4.2 AVX AVX2 AVX_VNNI FMA, in other operations, rebuild TensorFlow with the appropriate compiler flags.
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.
* Serving Flask app 'app'
* Debug mode: on
INFO:werkzeug:WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
* Running on http://127.0.0.1:5050
INFO:werkzeug:Press CTRL+C to quit
INFO:werkzeug: * Restarting with stat
2025-06-26 09:44:46.016628: I tensorflow/core/util/port.cc:153] oneDNN custom operations are on. You may see slightly different numerical results due to floating-point round-off errors from different computation orders. To turn them off, set the environment variable 'TF_ENABLE_ONEDNN_OPTS=0'.
2025-06-26 09:44:46.787026: I tensorflow/core/util/port.cc:153] oneDNN custom operations are on. You may see slightly different numerical results due to floating-point round-off errors from different computation orders. To turn them off, set the environment variable 'TF_ENABLE_ONEDNN_OPTS=0'.
2025-06-26 09:44:49.488205: I tensorflow/core/platform/cpu_feature_guard.cc:210] This TensorFlow binary is optimized to use available CPU instructions in performance-critical operations.
To enable the following instructions: SSE3 SSE4.1 SSE4.2 AVX AVX2 AVX_VNNI FMA, in other operations, rebuild TensorFlow with the appropriate compiler flags.
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:werkzeug: * Debugger is active!
```

Fig 7.1.1: Code running in Terminal

**Step 2:** Click on that link a web page of Nutrigaze will be open in the web browser.

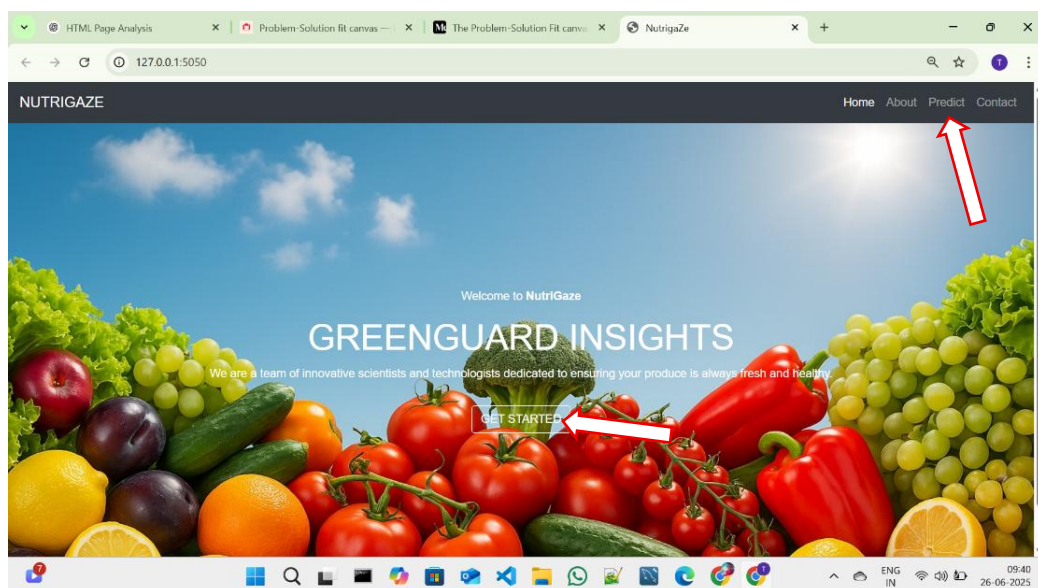
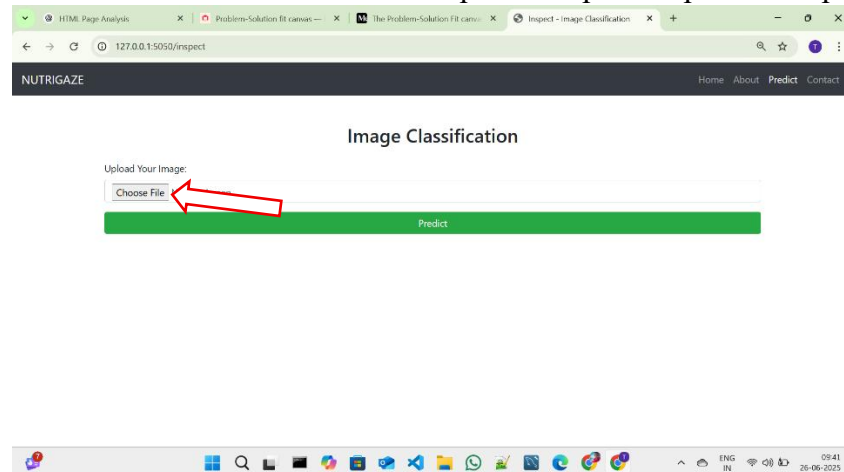


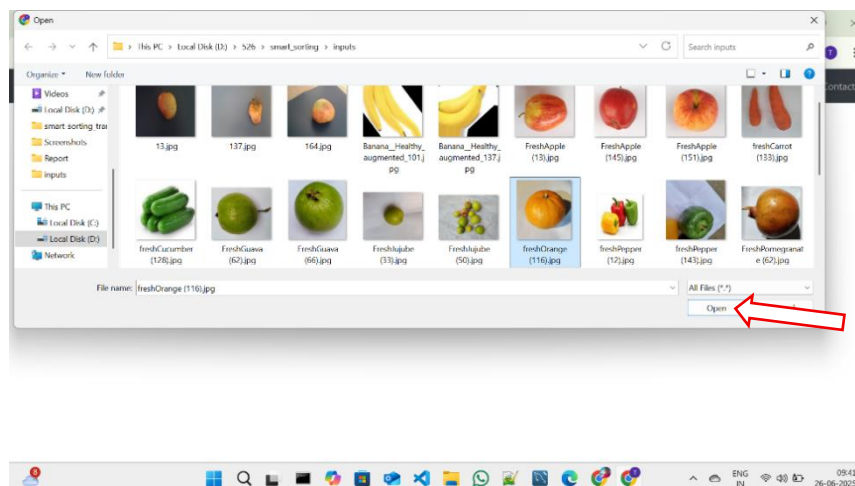
Fig 7.1.2: NUTRIGAZE Home Page

**Step 3:** Click on GET STARTED or PREDICT option to open the prediction page.



**Fig 7.1.3: Prediction page in NUTRIGAZE**

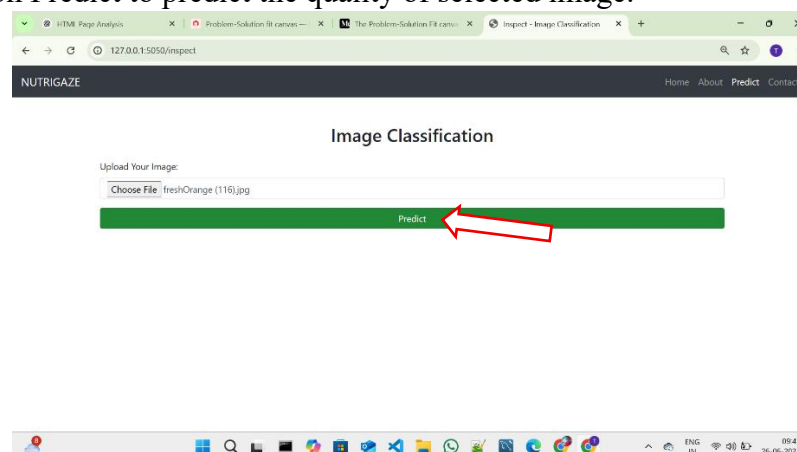
**Step 4:** Click on choose file option to choose the images that need to predict.



**Fig 7.1.4: Window to choose image for prediction**

Select any image for prediction and click on Open.

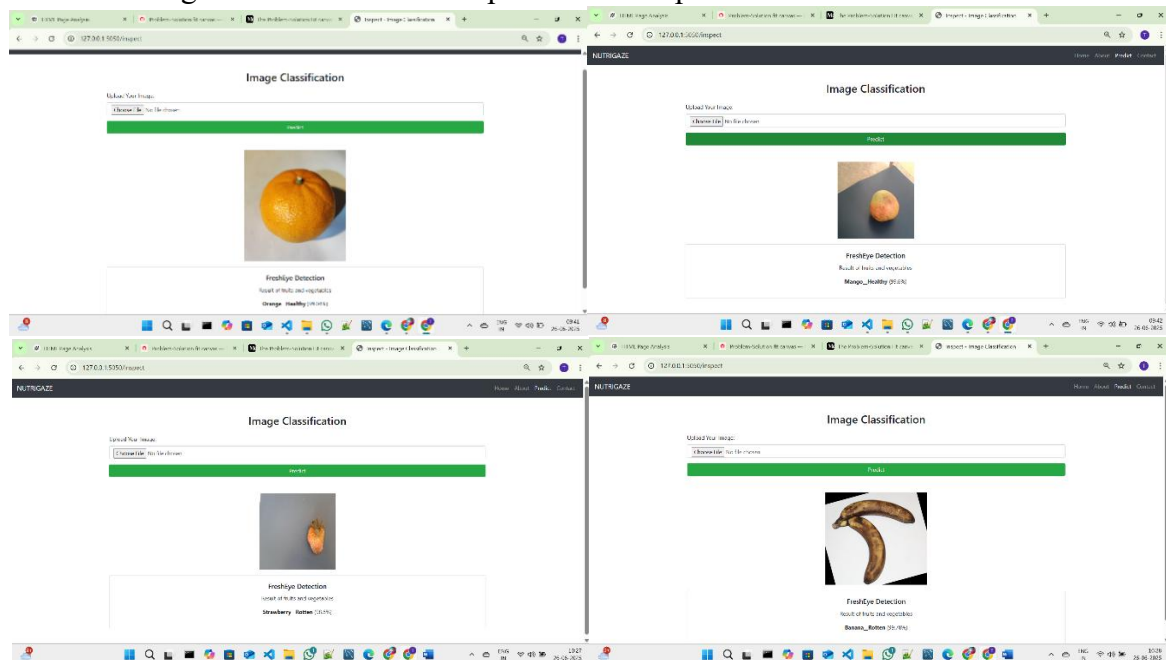
**Step 5:** Click on Predict to predict the quality of selected image.



**Fig 7.1.5: Image selected in prediction page**

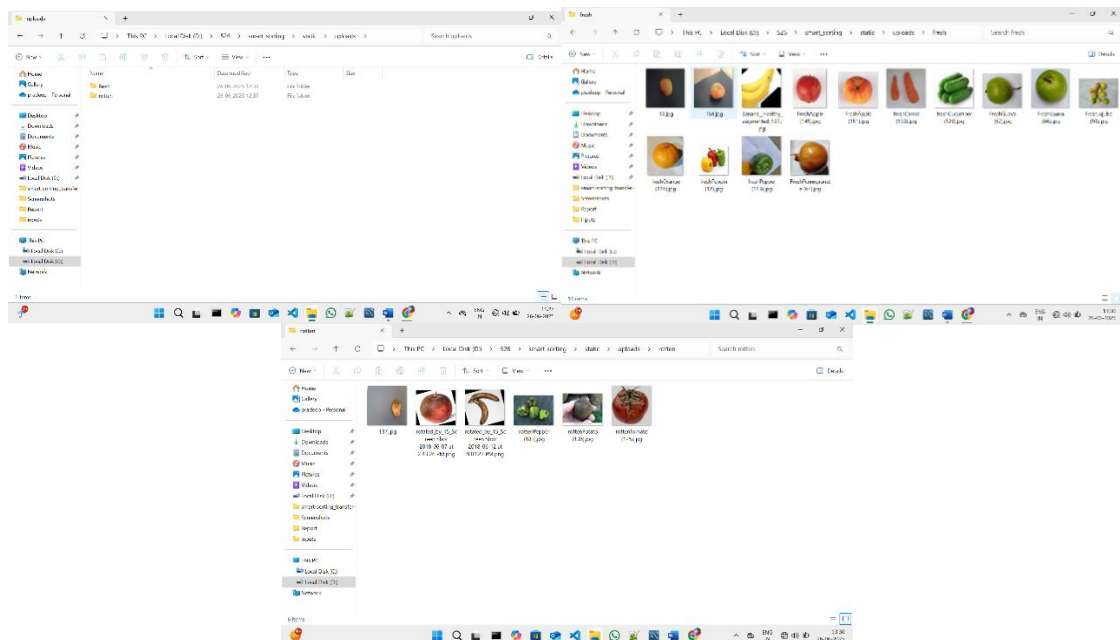
**Step 6:** After clicked on the predict button the model predicts the image quality and displays the quality of image.

The below images are the some of samples tested for prediction.



**Fig 7.1.6: Prediction output for the several inputs with accuracy**

After the Images Predicted the images will be stored in the uploads folder as fresh and rotten as shown in the figure given below.



**Fig 7.1.7: Folder Structure to store predicted images**

**Project Demo Link:**

<https://drive.google.com/file/d/15Er4XMIVQakaXRwXQOqdOFN96thbvckt/view?usp=sharing>

## 12. Known Issues

### Functional Issues

- **Confidence Variability:**  
Predictions on external, real-world images sometimes produce low confidence scores even when the classification is correct.
- The model may need retraining over time with new data.
- Running deep learning models on edge devices (like smartphones or Raspberry Pi) may face hardware limitations.
- **Model Misclassification:**  
The model occasionally misclassifies images that have poor lighting, background clutter, or images taken from unusual angles.

### File Handling Issues

- **Duplicate File Names:**  
If a user uploads a file with the same name as an existing file in the static/uploads/fresh/ or static/uploads/rotten/ directories, the existing file will be overwritten.

### UI/UX Issues

- **Image Display Delay:**  
In some cases, large image files may take a few extra seconds to display after prediction due to processing and browser loading time.
- **No Image Preview Before Submission:**  
Users cannot preview the image before submitting it for prediction.

### Security Issues

- **No File Type Validation:**  
The current system does not strictly validate uploaded file types, which may allow unsupported or malicious files to be uploaded.

## 13. Future Enhancements

### Model Improvements

- **Integration with IoT and Smart Cameras:**  
Embed the model into **edge devices**, smart cameras, or IoT systems for **24/7 real-time monitoring** in warehouses, cold storage, or supermarkets.
- **Multi-Class Classification:**  
Extend the model to classify not just fresh or rotten but also detect specific defects, diseases, or categorize different types of fruits and vegetables.

## Web Application Features

- **Image Preview Before Submission:**  
Add functionality to allow users to preview uploaded images before submitting them for prediction.
- **Drag and Drop Uploads:**  
Improve user experience by supporting drag-and-drop image uploads.
- **Batch Prediction:**  
Enable users to upload multiple images at once and receive predictions for all of them.

## File Management

- **Automatic File Renaming:**  
Implement a system to avoid overwriting existing files by appending timestamps or unique IDs to uploaded file names.
- **Image History & Gallery:**  
Provide users with access to previously uploaded images and their prediction results.

## Security Enhancements

- **Strict File Validation:**  
Ensure that only valid image files (e.g., .jpg, .png) are accepted to prevent potential misuse.
- **Authentication & User Management:**  
Implement user login and role-based access control to secure the application for different types of users (e.g., Admin, General User).

## Reporting & Analytics

- **Downloadable Reports:**  
Generate downloadable reports (e.g., PDF or CSV) summarizing predictions made during a session.
- **Usage Analytics:**  
Track application usage statistics, such as number of images analyzed, classification distribution, etc.

## Deployment & Scalability

- **Cloud Deployment:**  
Deploy the application to a cloud platform (e.g., AWS, Heroku) for public access.
- **Mobile-Friendly Design:**  
Optimize the web interface for mobile devices to improve accessibility.

## Mobile Application Development

- Develop a lightweight **mobile app** to allow farmers, vendors, and consumers to scan produce using their **smartphone cameras**.