ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING

Project Documentation format

1.Introduction:

Project Title:

Smart Sorting: Identifying Rotten Fruits and Vegetables Using Transfer Learning

Team Members:

Name	Role
Lakshmi Narayana	Full Stack & ML Dev
Kushulatha	UI/UX Designer
Devi Vara Prasad	Dev
Heshwanth	Dev

2.Project Overview

Purpose:

The primary goal of this project is to build a smart AI-powered system capable of detecting and classifying the freshness of fruits and vegetables through image recognition. Leveraging advanced machine learning techniques—particularly transfer learning with pre-trained models like MobileNetV2—this system identifies whether a fruit or vegetable is fresh or rotten based on user-uploaded images.

This technology has practical implications in agriculture, retail, and supply chain sectors, where timely identification of spoiled produce can significantly reduce waste, enhance food safety, and improve operational efficiency. By providing real-time feedback with high accuracy, this solution supports farmers, vendors, and consumers in making informed decisions, ultimately contributing to sustainable food distribution and quality assurance.

Key Features:

- Image Upload for Prediction:
  Users can easily upload images of fruits or vegetables through a simple web interface for instant analysis.
- Real-Time Classification:
  The system provides immediate classification (Fresh or Rotten) along with a confidence score, enhancing decision-making at the point of inspection.
- Transfer Learning with MobileNetV2:
  A lightweight, high-performance deep learning model (MobileNetV2) is fine-tuned using a curated dataset to improve accuracy and reduce training time.
- User-Friendly Web Interface:
  The front end, built using basic HTML and CSS, is minimalistic yet effective, ensuring a seamless user experience for uploading images and viewing results.
- Flask-Based REST API:
  The backend API is built using Flask, allowing for smooth communication between the user interface and the AI model for prediction tasks.
- Cloud Deployment on Render:
  The entire application is deployed on Render, making it accessible from anywhere with an internet connection and demonstrating real-world usability.

3.Architecture

Frontend Architecture:

The frontend is designed to be simple, responsive, and user-friendly using standard web technologies:

Technologies Used:

- **HTML5** – Structure of the web page.
- **CSS3** – Styling and responsive layout.

Functional Flow:

1. **User Interface (UI):**
    - A single-page HTML layout provides an upload form where users can select fruit/vegetable images from their local device.
2. **Image Upload:**
    - The uploaded image is submitted via an HTML <form> using the POST method to the Flask backend (/predict endpoint).
3. **Prediction Display:**
    - Once the response is received from the backend, the page dynamically displays:
        - The predicted class (Fresh or Rotten).
        - The model's confidence score.
        - A preview of the uploaded image.
4. **Error Handling (Basic):**

- Alerts or messages are shown in case of invalid input (e.g., no image uploaded or unsupported file type).

Backend Architecture Outline:

The backend is built using **Flask**, a lightweight Python web framework, and integrates a trained **TensorFlow model** for inference.

Technologies Used:

- **Flask** – For handling web requests and serving predictions.
- **TensorFlow / Keras** – For model loading and inference.
- **Pillow (PIL)** – For image preprocessing.
- **NumPy** – For array manipulation during preprocessing.

Component-wise Outline:

1. **Model Loading:**
   - The trained .h5 model (healthy_vs_rotten.h5) is loaded once at server startup using TensorFlow's load_model().
2. **API Endpoints:**
   - / (GET): Serves the HTML frontend.
   - /predict (POST): Accepts an uploaded image file, processes it, performs prediction, and returns a JSON response.
3. **Image Preprocessing:**
   - Uploaded images are resized, normalized, and reshaped to match the input format expected by MobileNetV2.

4. **Prediction Logic:**
    - The processed image is passed to the model.
    - The prediction score is interpreted and mapped to class labels: “Fresh” or “Rotten”.
5. **Response Format:**
    - JSON object returned with:
        - prediction: Class label
        - confidence: Prediction probability
6. **File Handling:**
    - Temporary in-memory image storage (no persistent uploads).
    - Optional cleanup after prediction to save memory.
7. **Security & Performance:**
    - Basic file validation (image types).
    - Can be enhanced with throttling, file size checks, and API key validation.

Database:

- No persistent database used.
- In-memory operations process the uploaded images.
- Optional logs of predictions can be stored in flat files or a NoSQL store if extended.

4.Setup Instructions

Prerequisites

Before setting up and running the Smart Sorting project, ensure you have the following tools and libraries installed on your development machine:

- Python 3.x
  Python is the primary programming language used for building the backend application and integrating the machine learning model. Any recent version of Python 3 (e.g., 3.7, 3.8, or newer) is supported.
- pip
  pip is the Python package installer. It allows you to easily install all required dependencies listed in the requirements.txt file.
- Flask
  Flask is a lightweight Python web framework used to create the REST API endpoints and serve the web application. It handles routing, request processing, and response generation.
- TensorFlow
  TensorFlow is an open-source machine learning framework used for loading and running the trained deep learning model (MobileNetV2) to classify the images.
- NumPy
  NumPy is a fundamental Python library for scientific computing. It is used for manipulating image data arrays during preprocessing before passing them to the model.

- Pillow
  Pillow is a popular Python imaging library. It provides tools to open, process, and convert uploaded images into the appropriate format and size required by the model.

Installation:

Clone the Repository:

```
git clone [your-repo-url]

cd [project-directory]
```

Create a Virtual Environment

```
python -m venv venv

On Windows: venv\Scripts\activate
```

Install Dependencies:

```
pip install -r requirements.txt
```

Set Environment Variables):

```
set FLASK_APP=app.py

set FLASK_ENV=development
```

5.Folder Structure

```
smart-sorting/
|
├── app.py                  # Flask backend with REST API routes
├── healthy_vs_rotten.h5    # Trained TensorFlow model
├── requirements.txt        # Python dependencies
├── templates/
│   └── index.html          # Frontend HTML page
├── static/
│   └── styles.css         # CSS styling
└── uploads/               # Temporary storage for uploaded images
```

Client (Frontend):

- **templates/index.html**
  This file contains the core HTML markup for the web interface. It provides:
    - A clean, user-friendly page layout where users can select and upload images of fruits or vegetables.
    - An upload form (<form>) configured to send the selected file to the backend API endpoint (/predict) via a POST request.
    - Dynamic placeholders or sections to display:
        - The uploaded image preview.
        - The predicted classification (Fresh or Rotten).
        - The model's confidence score.
- **static/styles.css**
  This stylesheet is used to customize the appearance and layout of the web interface, ensuring a professional and responsive design. It includes:
    - Styling for the upload form and buttons.
    - Rules for image preview sizing.

    - Fonts, colors, and spacing to improve readability and visual appeal.
    - Basic responsiveness to make the page usable on different devices (laptops, tablets, smartphones).

Server (Backend):

- **app.py**
  This Python script is the main entry point of the backend application. It is responsible for:
    - Initializing the Flask server and defining the routes (/ for the main page and /predict for handling predictions).
    - Loading the pre-trained MobileNetV2 model at startup.
    - Handling incoming POST requests containing image files.
    - Performing image preprocessing (resizing, normalization, and reshaping) to prepare the image data for prediction.
    - Running inference using TensorFlow and interpreting the output probabilities.
    - Returning a JSON response containing the predicted class and confidence score to the frontend.
- **healthy_vs_rotten.h5**
  This is the pre-trained TensorFlow model file containing the learned weights and architecture of the MobileNetV2 model fine-tuned for classifying fresh versus rotten produce. It serves as the core intelligence behind the prediction system.
    - It is loaded once when the Flask server starts to optimize performance.

  - During prediction, it receives processed image data and outputs the prediction probabilities.
  - This file is essential for the system to make accurate classifications.

6.Running the Application

Starting the Frontend and Backend Servers Locally

To run the Smart Sorting application on your local development machine, follow these steps:

Backend Server (Flask API)

The backend is built with Flask, which serves the REST API and handles image classification.

Steps:

1. Navigate to your project directory:

```
cd smart-sorting
```

2. Activate your virtual environment:

```
venv\Scripts\activate
```

2. Install all dependencies:

```
pip install -r requirements.txt
```

3. Set environment variables (development mode):

```
set FLASK_APP=app.py
set FLASK_ENV=development
```

4. Start the Flask server:

```
flask run
```

By default, this will start the backend at:

http://127.0.0.1:5000

Functionality:

- Serves the main HTML page (index.html) for uploading images.
- Accepts image uploads via the /predict endpoint.
- Loads the MobileNetV2 model (healthy_vs_rotten.h5) at startup.
- Returns JSON predictions to the frontend.

Frontend (HTML/CSS Static Files)

This project uses static HTML and CSS, served directly by Flask.
No separate frontend server is required.
When you start Flask (flask run), it automatically makes the web interface available at:

http://127.0.0.1:5000

If you prefer to test the HTML separately (without Flask), you can open templates/index.html directly in your browser, but image uploads will not work unless the Flask backend is running.

Typical Workflow

Here’s what you do to get everything running:

1. Open your terminal or command prompt.
2. Start the Flask server as shown above.
3. In your browser, go to:

http://127.0.0.1:5000

4. Upload an image to test classification.
5. View the prediction result displayed on the page.

Example Directory Commands

Below is an example of a complete command sequence for clarity:

```
cd smart-sorting
python -m venv venv
 venv\Scripts\activate
pip install -r requirements.txt
export FLASK_APP=app.py
export FLASK_ENV=development
flask run
```

Troubleshooting:

- If you get a ModuleNotFoundError, double-check that all required packages are installed.
- Make sure your healthy_vs_rotten.h5 model file is in the correct location.
- Confirm no other processes are using port 5000.

Summary:

- Backend: Flask server (flask run)

- Frontend: Static HTML/CSS served by Flask (no npm needed)

7.API Documentation

This project exposes a simple REST API with two endpoints for interacting with the model and serving the user interface.

Endpoints Overview

Endpoint	Method	Description
/	GET	Render the main HTML upload page
/predict	POST	Process an uploaded image and return a freshness prediction

---

GET /

- Description:
  Serves the web interface (index.html) where users can upload an image of a fruit or vegetable for classification.
- Request Parameters:
  None.
- Response:
  Returns an HTML page containing:
  - An image upload form.
  - Instructions for using the system.
  - A placeholder area for displaying results.

POST /predict

- Description:
  Accepts an image file uploaded by the user, processes it through the trained model, and returns a JSON prediction.
- Request Headers:

Content-Type: multipart/form-data

- Request Parameters:

Parameter	Type	Description
file	File	Image file to be classified (JPEG, PNG)

http://127.0.0.1:5000/predict

- Response Format:
  JSON object containing:
  - prediction: String label ("Fresh" or "Rotten")
  - confidence: Float value between 0 and 1 representing prediction certainty.
- Response Example:

```
{
  "prediction": "Rotten",
  "confidence": 0.89
}
```

Error Responses

If an error occurs during the request (e.g., no file provided, unsupported format), the API will return a JSON object with an error message:

Response Example:

```
{
  "error": "No file part in the request"
}
```

or

```
{
  "error": "Invalid file type"
}
```

Additional Notes

- Accepted Image Types:
  .jpg, .jpeg, .png
- Maximum File Size:
  Typically limited by Flask configuration (can be set in app.py).
- Inference Time:
  Predictions are returned in real-time, usually within 1–2 seconds.
- Security:
  Currently, no authentication is implemented—any client can submit a request. In production, you should add API keys or token-based authentication.

8.Authentication

Current State

- The current version of this application is designed as a public demo, and no authentication mechanism is implemented.
- All API endpoints are open-access, meaning:
  - Any client or user with the URL can submit requests.
  - No credentials, API keys, or tokens are required.
- This setup is suitable for:
  - Educational purposes.
  - Testing and demonstrations.
  - Local development environments.

Potential Enhancements

To improve security and control access in a real-world scenario, you can implement one or more of the following authentication and authorization strategies:

---

1. API Key Validation

- Description:
  - Each client must include a unique API key in the request header or query parameters.
  - The server validates this key against a list of authorized keys.
- How it works:

    - When a request is received, the backend checks the provided key.
    - If the key is missing or invalid, the request is rejected with a 401 Unauthorized response.
- Benefits:
    - Simple to implement.
    - Suitable for controlling usage among a limited set of trusted clients.

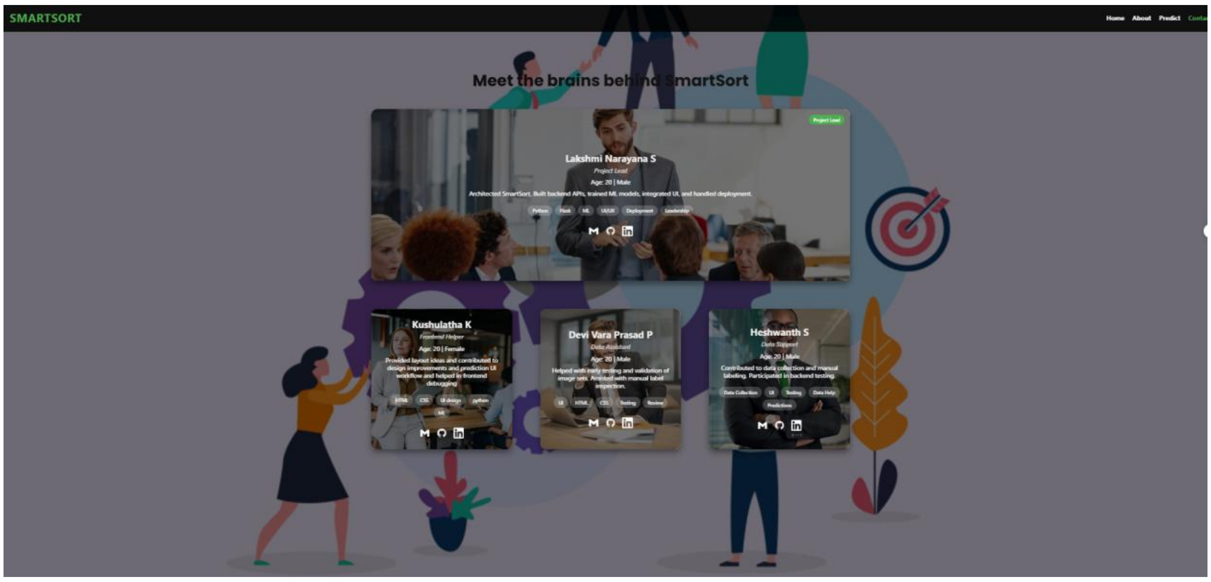
---

2. JWT (JSON Web Token) Authentication

- Description:
    - Users or applications authenticate by logging in (or registering), receiving a signed token.
    - The token must be included in each request's Authorization header.
- How it works:
    - The server verifies the token's signature and validity.
    - Tokens can include claims (e.g., user roles, expiry timestamps).
- Benefits:
    - Scalable for multiple users.
    - Supports fine-grained permissions (e.g., read-only, admin).

9.User interface

Applies a gradient color effect to the section heading “Meet the brains behind SmartSort”.



Meet the brains behind SmartSort

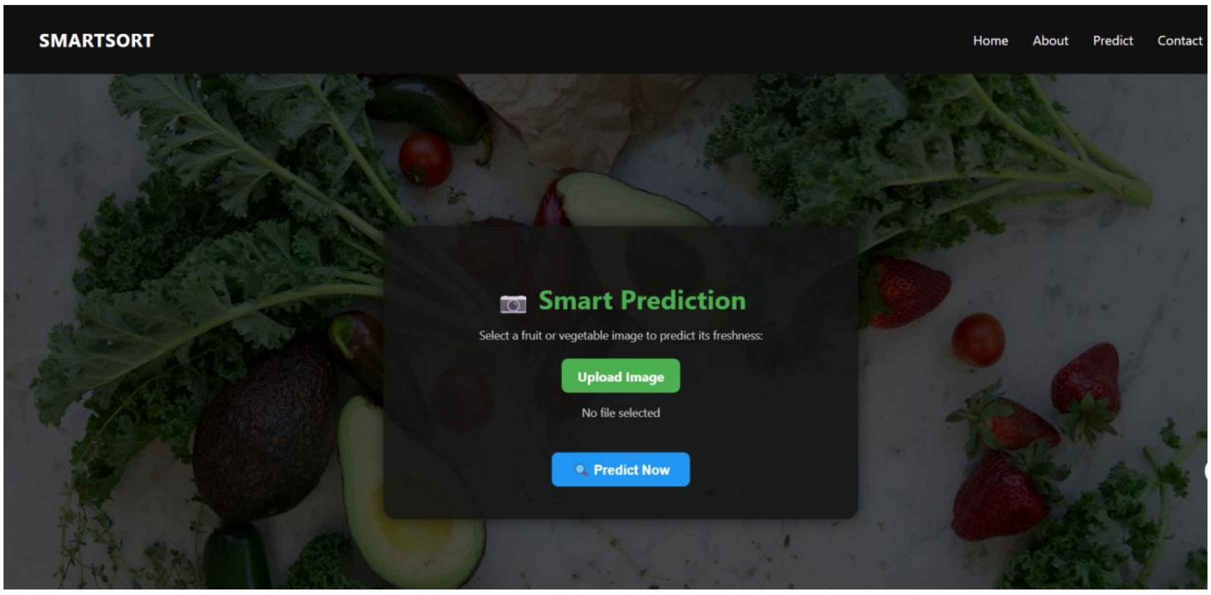
Lakshmi Narayana S
Project Lead
Age: 20 | Male
Architected SmartSort. Built backend APIs, trained ML models, integrated UI, and handled deployment.

Kushulatha K
Frontend Helper
Age: 20 | Female
Provided layout ideas and contributed to design improvements and prediction UI workflow and helped in frontend debugging

Devi Vara Prasad P
Data Assistant
Age: 20 | Male
Helped with early testing and validation of image sets. Assisted with manual label inspection.

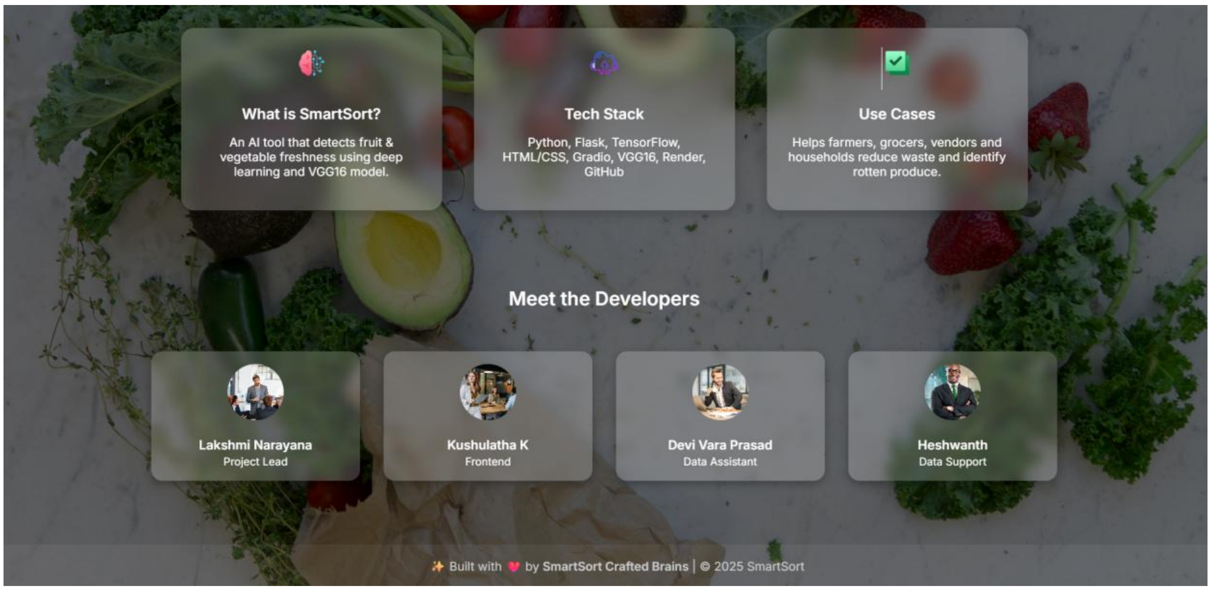
Heshwanth S
Data Support
Age: 20 | Male
Contributed to data collection and manual labeling. Participated in backend testing.

Adds **smooth scroll-based animations** as each element comes into view



Select a fruit or vegetable image to predict its freshness:

Team Grid with Responsive Cards



What is SmartSort?
An AI tool that detects fruit & vegetable freshness using deep learning and VGG16 model.

Tech Stack
Python, Flask, TensorFlow, HTML/CSS, Gradio, VGG16, Render, GitHub

Use Cases
Helps farmers, grocers, vendors and households reduce waste and identify rotten produce.

Meet the Developers

Lakshmi Narayana
Project Lead

Kushulatha K
Frontend

Devi Vara Prasad
Data Assistant

Heshwanth
Data Support

Built with by SmartSort Crafted Brains | © 2025 SmartSort

10.Testing

Strategy:

- Manual testing using sample images.
- Model validation accuracy evaluated during training.
- Integration testing with Flask's test client.

Tools:

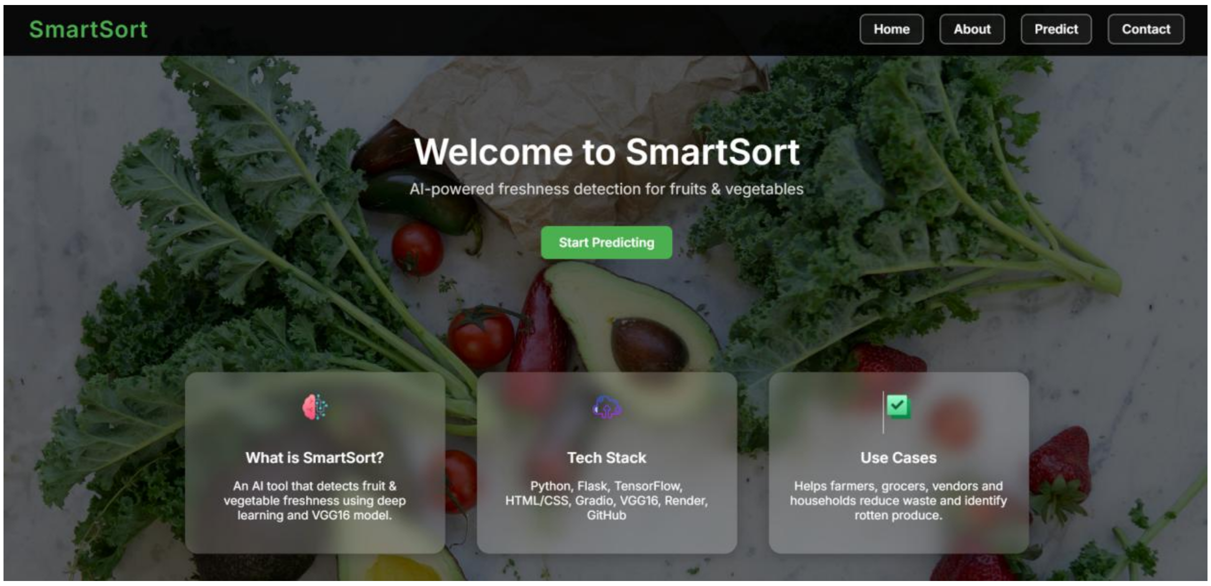
- pytest (optional)
- Flask's built-in testing utilities.

11. Screenshots or Demo

- **Live Demo:**
  Demo URL on Render *(smartsort-backend.onrender.com/)*

- **Screenshots:**

  Home page:



Welcome to SmartSort

AI-powered freshness detection for fruits & vegetables

What is SmartSort?

An AI tool that detects fruit & vegetable freshness using deep learning and VGG16 model.

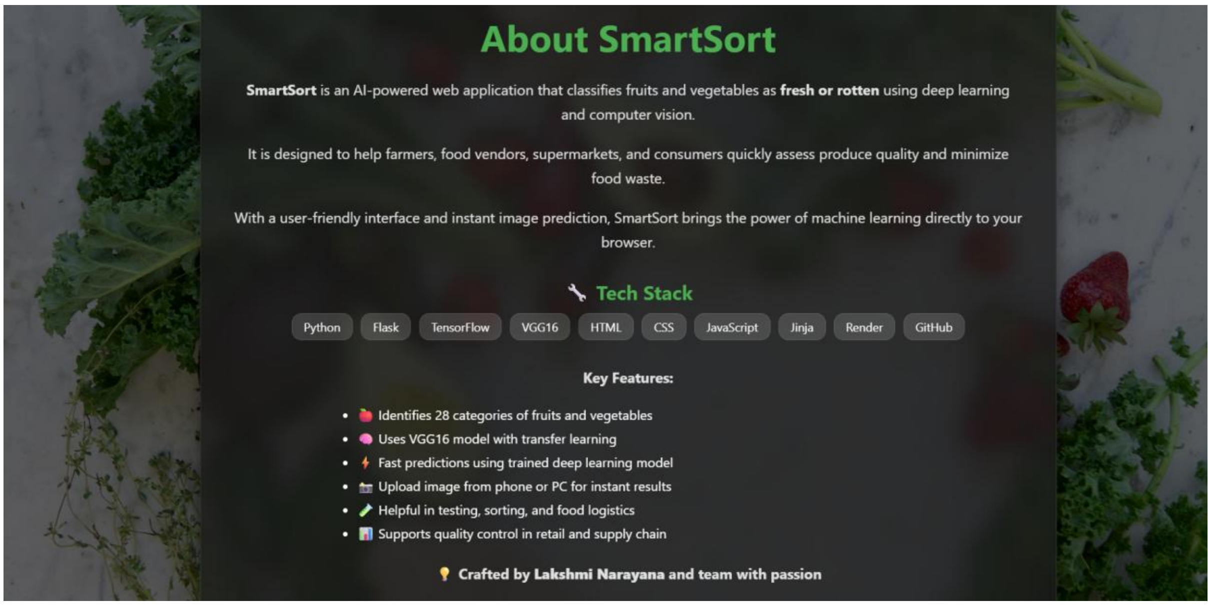
Tech Stack

Python, Flask, TensorFlow, HTML/CSS, Gradio, VGG16, Render, GitHub

Use Cases

Helps farmers, grocers, vendors and households reduce waste and identify rotten produce.

About page:



About SmartSort

SmartSort is an AI-powered web application that classifies fruits and vegetables as **fresh or rotten** using deep learning and computer vision.

It is designed to help farmers, food vendors, supermarkets, and consumers quickly assess produce quality and minimize food waste.

With a user-friendly interface and instant image prediction, SmartSort brings the power of machine learning directly to your browser.

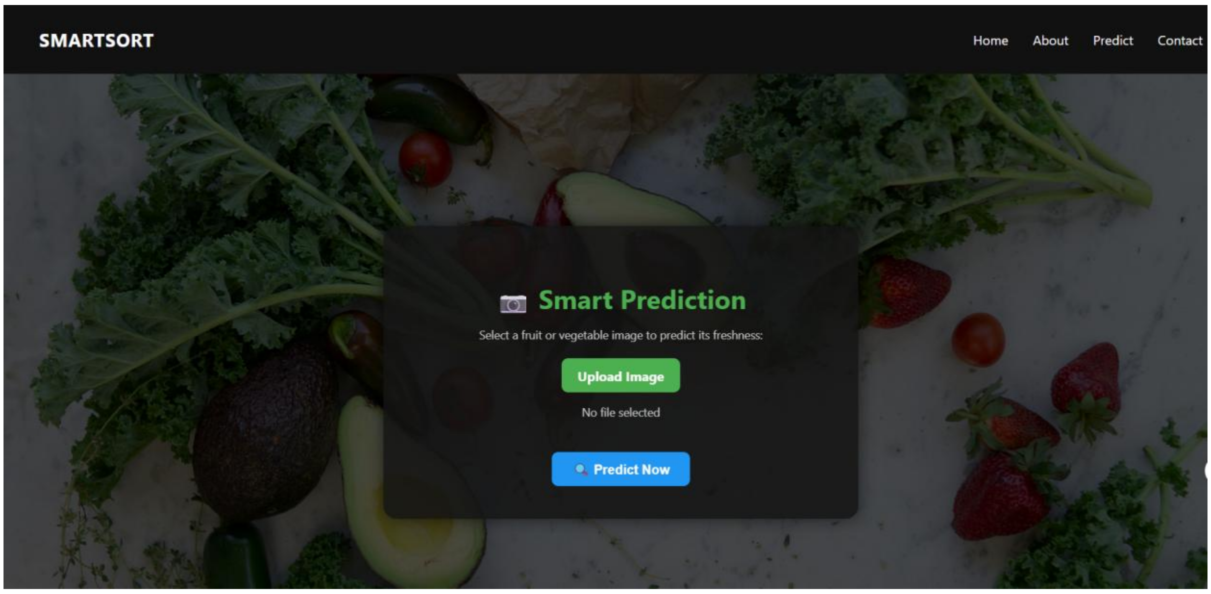
Tech Stack

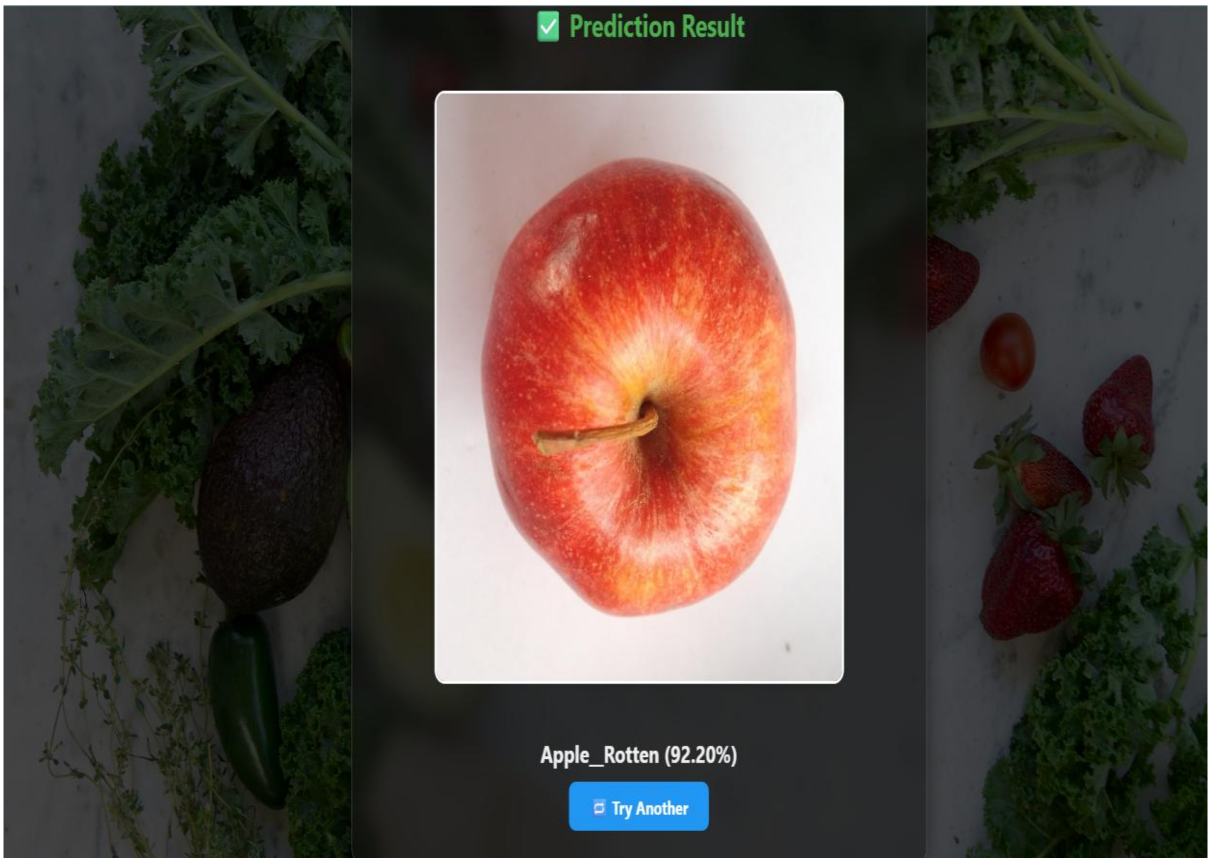
Key Features:

- Identifies 28 categories of fruits and vegetables
- Uses VGG16 model with transfer learning
- Fast predictions using trained deep learning model
- Upload image from phone or PC for instant results
- Helpful in testing, sorting, and food logistics
- Supports quality control in retail and supply chain

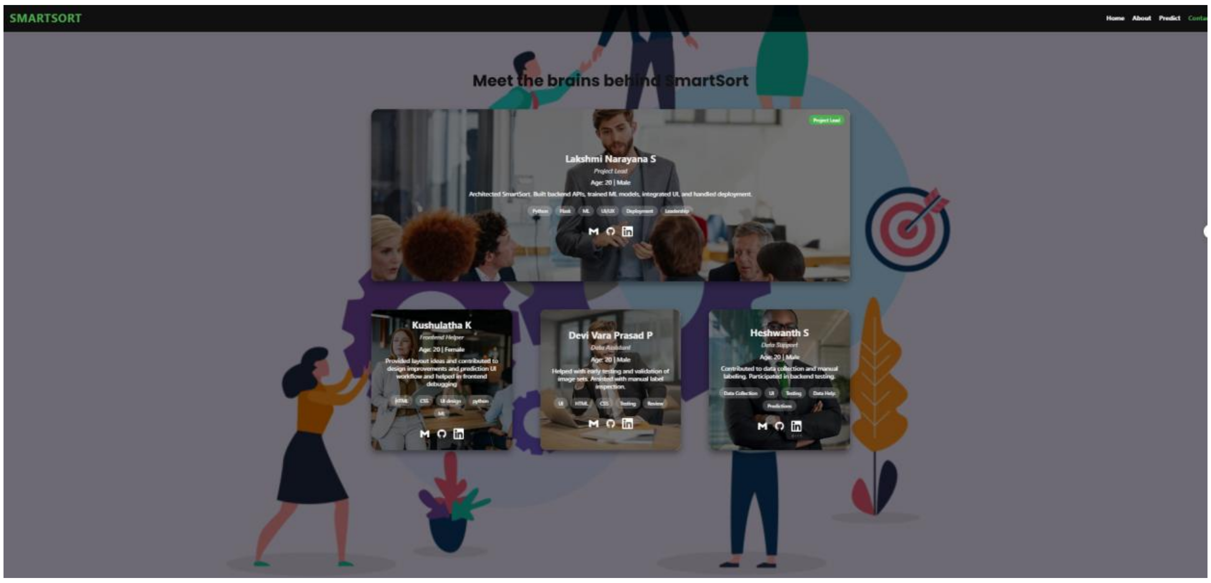
Crafted by Lakshmi Narayana and team with passion

Predict page:


SMARTSORT
Home
About
Predict
Contact
Smart Prediction
Select a fruit or vegetable image to predict its freshness:
Upload Image
No file selected
Predict Now


Prediction Result
Apple_Rotten (92.20%)
Try Another

Contact page:



Meet the brains behind SmartSort

Lakshmi Narayana S
Project Lead
Age: 20 | Male
Architected SmartSort. Built backend APIs, trained ML models, integrated UI, and handled deployment.

Kushulatha K
Frontend Helper
Age: 20 | Female
Provided layout ideas and contributed to design improvements and prediction UI workflow and helped in frontend debugging

Devi Vara Prasad P
Data Assistant
Age: 20 | Male
Helped with early testing and validation of image sets. Assisted with manual label inspection.

Heshwanth S
Data Support
Age: 20 | Male
Contributed to data collection and manual labeling. Participated in backend testing.

12. Known Issues

- Model performance may degrade on low-resolution or poor-light images.
- No rate limiting—server could be overloaded with large requests.
- No persistent storage for logs or uploaded images.

---

13. Future Enhancements

- Add user authentication and dashboards.
- Store prediction logs in a database.

- Support multiple categories beyond fresh/rotten.
- Improve model accuracy with a larger dataset.
- Implement API rate limiting and error handling.
- Containerize with Docker for easier deployment.