1. Introduction

Project Title:

Rotten Fruits and Vegetables Detection System

Team Members:

- **Team Leader: Posina Sai Chakra Amrutha Varshini** – Responsible for overall project planning, coordination, documentation preparation, system integration, and supervision of model and web development.
- **Team Member: Pulugurtha Narasimha Rajesh** – Assisted in backend integration using Python and Flask, API handling, and system testing.
- **Team Member: Saideepak Rongali** – Contributed to frontend interface design using HTML, CSS, and JavaScript, and ensured smooth interaction between user interface and backend.
- **Team Member: Bhuvana Satya Samsani** – Worked on dataset preprocessing, CNN model training, evaluation, and performance optimization.

Food wastage is a major global issue, especially in agricultural markets and retail supply chains where fruits and vegetables are highly perishable. Manual inspection methods are time-consuming and prone to human error, often leading to incorrect classification of spoiled produce. This project introduces an automated detection system using Python and Deep Learning to classify fruits and vegetables as Fresh or Rotten based on image input.

The system is developed using a Convolutional Neural Network (CNN) model trained on labeled image datasets. A Flask-based web application is integrated with the trained model to allow users to upload images and receive instant predictions. This solution demonstrates the practical application of Machine Learning in agriculture and food quality control while reducing wastage and improving efficiency.

2. Project Overview

• Purpose:

The primary purpose of this project is to design and implement an intelligent image classification system capable of automatically detecting rotten fruits and vegetables using Deep Learning techniques. Food wastage is a significant issue in agricultural supply chains, retail markets, and storage facilities, primarily due to inefficient manual inspection processes. Human-based sorting methods are often inconsistent, time-consuming, and prone to fatigue-related errors. As a result, spoiled produce may either be mistakenly sold to consumers or fresh items may be unnecessarily discarded. This project aims to provide a technological solution to this problem by introducing an automated detection system that ensures more accurate and reliable classification.

The system is developed to reduce food wastage by enabling early identification of spoiled fruits and vegetables. By detecting external signs of decay through image analysis, the application assists vendors, warehouse managers, and distributors in maintaining better quality control. Faster and more consistent inspection processes directly contribute to improved supply chain efficiency, cost reduction, and enhanced consumer trust.

In addition to addressing a real-world problem, this project also serves as an academic implementation of Machine Learning and Deep Learning concepts. It demonstrates the practical application of dataset collection, image preprocessing, Convolutional Neural Network (CNN) model design, training, evaluation, and deployment using Flask. The project integrates theoretical knowledge with hands-on development experience, making it both socially relevant and technically significant.

• Features:

The Rotten Fruits and Vegetables Detection System includes several important features designed to ensure functionality, usability, and performance. One of the primary features of the application is the image upload functionality, which allows users to submit images of fruits or vegetables directly through a web-based interface. The interface is simple and user-friendly, ensuring that individuals with minimal technical knowledge can easily operate the system.

Once an image is uploaded, the backend performs a series of preprocessing operations before passing the image to the trained CNN model. These preprocessing steps include resizing the image to match the model's input dimensions, normalizing pixel values to improve model performance, and converting the image into an appropriate numerical format. Proper preprocessing ensures consistent and accurate predictions.

The trained Convolutional Neural Network model analyzes visual patterns such as color variations, texture differences, and surface irregularities to classify the image into two categories: Fresh or Rotten. The prediction result is displayed instantly on the user interface, providing quick and actionable feedback. The system is optimized to deliver fast response times, ensuring efficient real-time usage.

Another important feature of the system is its scalability. Although the current implementation focuses on binary classification (Fresh vs Rotten), the architecture supports future enhancements such as multi-class classification for different types of fruits and vegetables. Additionally, the system can be extended to include real-time camera-based detection, cloud deployment for large-scale usage, and integration with automated sorting systems.

Overall, the application combines accuracy, efficiency, user-friendliness, and scalability, making it a practical and extensible solution for automated food quality inspection.

3. Architecture

• Frontend:

The frontend of the Rotten Fruits and Vegetables Detection System is developed using HTML, CSS, and JavaScript. It provides a clean and simple interface that allows users to upload images for classification. The design focuses on usability and clarity so that even non-technical users can operate the system without difficulty.

The interface consists of:

- Image upload form
- Submit button
- Prediction result display section
- Basic navigation elements

The frontend communicates with the Flask backend through HTTP POST requests. When a user uploads an image, the file is sent to the backend server for processing. JavaScript performs basic client-side validation to ensure that only valid image files are uploaded.

The responsive design ensures that the application can be accessed smoothly on different screen sizes. The separation between HTML templates and static files (CSS/JS) improves maintainability and scalability.

• Backend:

The backend is implemented using **Python and Flask**, which acts as a lightweight web framework to manage routing and server-side operations.
The backend performs the following operations:

1. Receives the uploaded image from the frontend.
2. Validates the file type and saves it temporarily.
3. Loads the pre-trained CNN model (`trained_model.h5`).
4. Preprocesses the image (resizing to 64x64 pixels and normalizing pixel values).
5. Passes the processed image to the model for prediction.
6. Returns the predicted class (Fresh or Rotten) to the frontend.

The CNN model was built and trained using TensorFlow/Keras with the following architecture:

- Conv2D Layer (32 filters, 3x3 kernel, ReLU activation)
- MaxPooling2D Layer
- Flatten Layer
- Dense Layer (128 neurons, ReLU activation)
- Output Dense Layer (Softmax activation based on number of classes)

The model was trained using `ImageDataGenerator` with rescaling and validation split. After training, the model was saved in `.h5` format for reuse during deployment.
Flask handles:

- Routing
- Image preprocessing
- Model loading
- Prediction
- Error handling

• Database:

This project does not use a traditional database system. Since the primary focus is image classification, the system relies on:

- Dataset directories for training and validation
- Local storage of trained model (`trained_model.h5`)
- Label file (`labels.txt`) generated during training

The dataset is organized into separate class folders, and each folder represents one classification category. The labels are automatically extracted from folder names.
Although a database is not required in the current implementation, future versions can integrate a database to store:

- User prediction history
- Uploaded image logs
- Accuracy analytics
- Model performance metrics

4. Setup Instructions

• Prerequisites

To run the project locally, Python (3.x), Flask, TensorFlow or Keras, NumPy, and other required Python libraries must be installed. A code editor such as VS Code is recommended. The dataset should be properly organized into training and validation folders before model training.

• Installation:

- Download or clone the project files.
- Open the project folder in VS Code or any IDE.
- Install dependencies using**: pip install -r requirements.txt**
- Train the model using: **python projectprocess.py**
- This will:
  - Train the CNN model
  - Generate trained_model.h5
  - Create labels.txt
- Run the Flask application using: **python app.py**
- Open a browser and navigate to: **http://127.0.0.1:5000/**
- Upload an image to view prediction results.

5. Folder Structure

• Client:

The client side includes:

- HTML templates (for page layout)
- Static folder (CSS and JavaScript)
- Upload interface
- Result display section

Templates manage structure while CSS controls design and JavaScript handles client-side interactions.

• Server:

The server side handles all backend operations including image processing, model prediction, and response generation.

1. app.py (or run file)

This is the main Flask application file. It:

- Manages routing
- Handles HTTP requests
- Accepts image uploads
- Preprocesses images
- Loads the trained CNN model
- Returns prediction results

2. projectprocess.py

This script is used to:

- Preprocess the dataset
- Build and train the CNN model
- Save the trained model (trained_model.h5)
- Generate class labels (labels.txt)

3. trained_model.h5

This file stores the trained CNN model. It allows the system to perform predictions without retraining every time the application runs.

4. labels.txt

This file contains the class labels corresponding to the dataset categories. It ensures correct mapping between predicted index values and class names.

5. Dataset Directory

The dataset directory contains training and validation images organized into class-based subfolders.

The trained model is stored separately from application logic to allow easy updates or retraining without modifying the core application.

6. Running the Application

To run the application, ensure all dependencies are installed. Execute python app.py to start the Flask server. Once the server is running, open a web browser and navigate to the localhost address. Upload an image and view the prediction result displayed on the screen.

To run the application successfully, follow these steps:

1. Ensure all required dependencies (Flask, TensorFlow, NumPy, etc.) are installed.
2. Open the project directory in a terminal or IDE.
3. Start the Flask server by executing: python app.py
4. Once the server starts, open a web browser.
5. Navigate to: http://localhost:5000
6. Upload an image using the upload interface.
7. Click the prediction button to view the classification result.

The system will analyze the image and display whether the item is **Fresh** or **Rotten**.

7. API Documentation

The Flask backend exposes endpoints for image upload and prediction. The main endpoint accepts POST requests containing image files. The server processes the image and returns a JSON response indicating whether the item is Fresh or Rotten. Proper error responses are implemented for invalid inputs.

The backend exposes a RESTful API endpoint for image classification.

Endpoint

POST /predict

Request Format

- Method: POST
- Content Type: multipart/form-data
- Parameter: Image file

Processing Steps

1. Receive image from client
2. Validate file format
3. Resize image to 64x64 pixels
4. Normalize pixel values (scale to range 0–1)
5. Pass image to CNN model
6. Obtain predicted class

Error Handling

The system returns appropriate error responses for:

- Invalid file format
- Missing image file
- Corrupted image
- Internal server errors

This ensures robust and reliable API behavior.

8. Authentication

The current version of the project does not implement authentication because it is designed primarily as an academic demonstration of image classification using Convolutional Neural Networks.

However, authentication mechanisms can be integrated in future versions using:

- Flask-Login (session-based authentication)
- JWT (JSON Web Token) for token-based authentication
- Role-based access control for enterprise deployment

Adding authentication would help restrict unauthorized access and enable user-based prediction history tracking.

9. User Interface

The user interface is designed with simplicity and functionality in mind. It includes:

- Homepage with application title
- Image upload form
- Image preview display
- Prediction result section

The layout ensures:

- Clear navigation
- Minimal distractions
- Fast interaction

After uploading an image, users immediately receive a prediction result displayed in a clear and readable format. The interface is designed to be intuitive, requiring no prior technical knowledge.

10. Testing

Testing includes validating model accuracy using training and validation datasets. Metrics such as accuracy and loss are monitored. Functional testing ensures image upload and prediction features work correctly under different scenarios.

Testing is performed at two primary levels to ensure system reliability.

1. Model Testing

During training, the following metrics are monitored:

- Training Accuracy
- Validation Accuracy
- Training Loss
- Validation Loss

The validation dataset is used to evaluate model performance and prevent overfitting.

Confusion matrix and performance analysis can also be used for further evaluation.

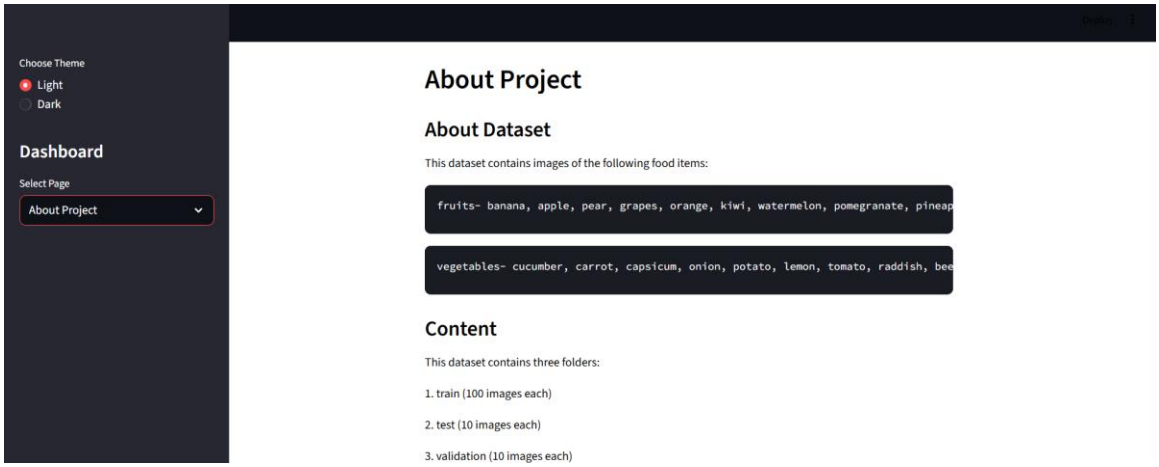
2. Functional Testing

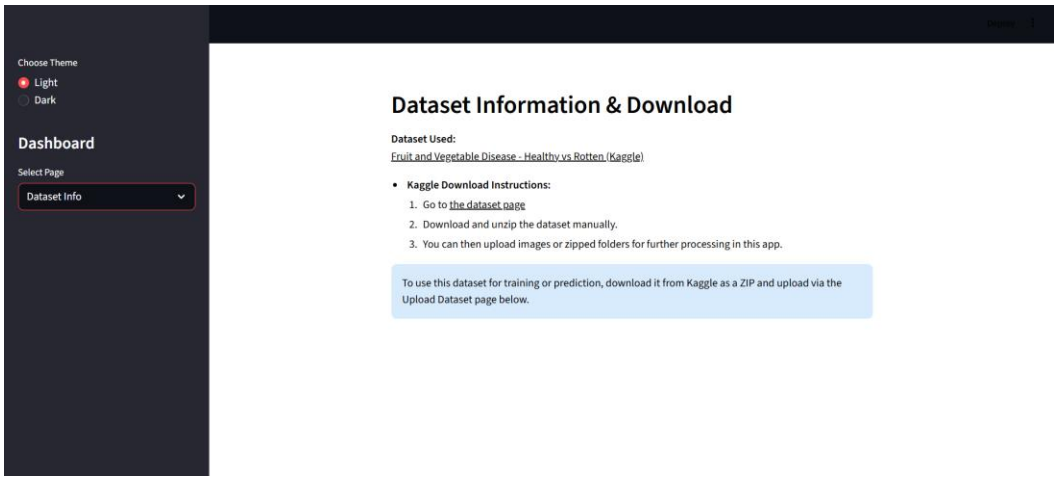
Functional testing ensures that:

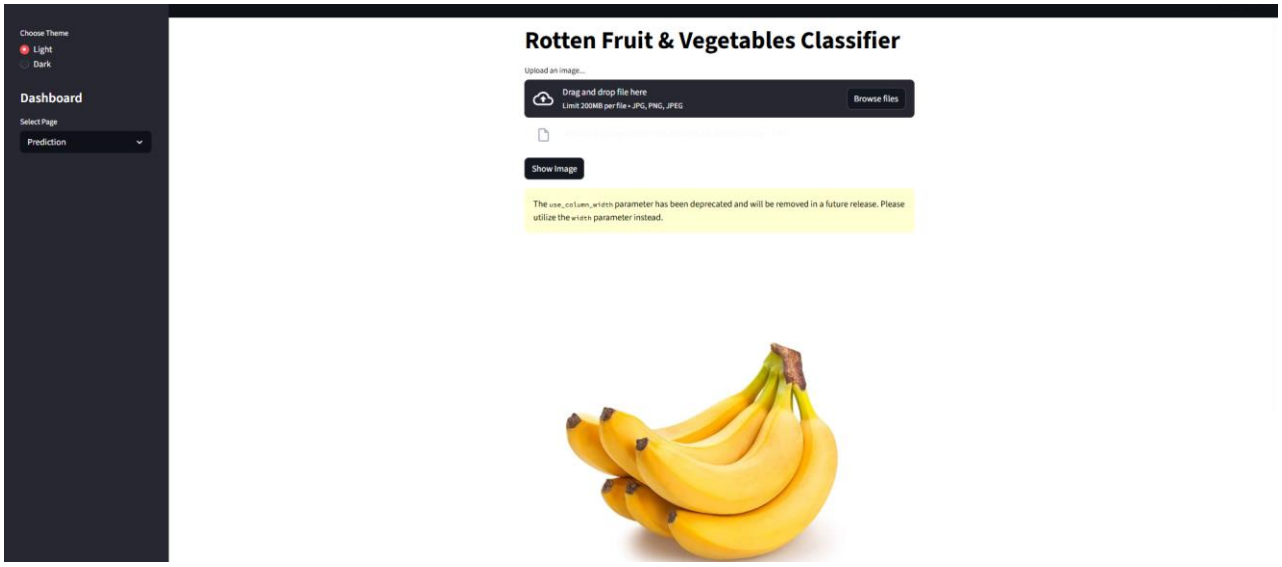
- Image upload works correctly
- Only valid file types are accepted
- The model prediction is returned correctly
- Error messages are displayed for invalid inputs
- Server runs without crashes

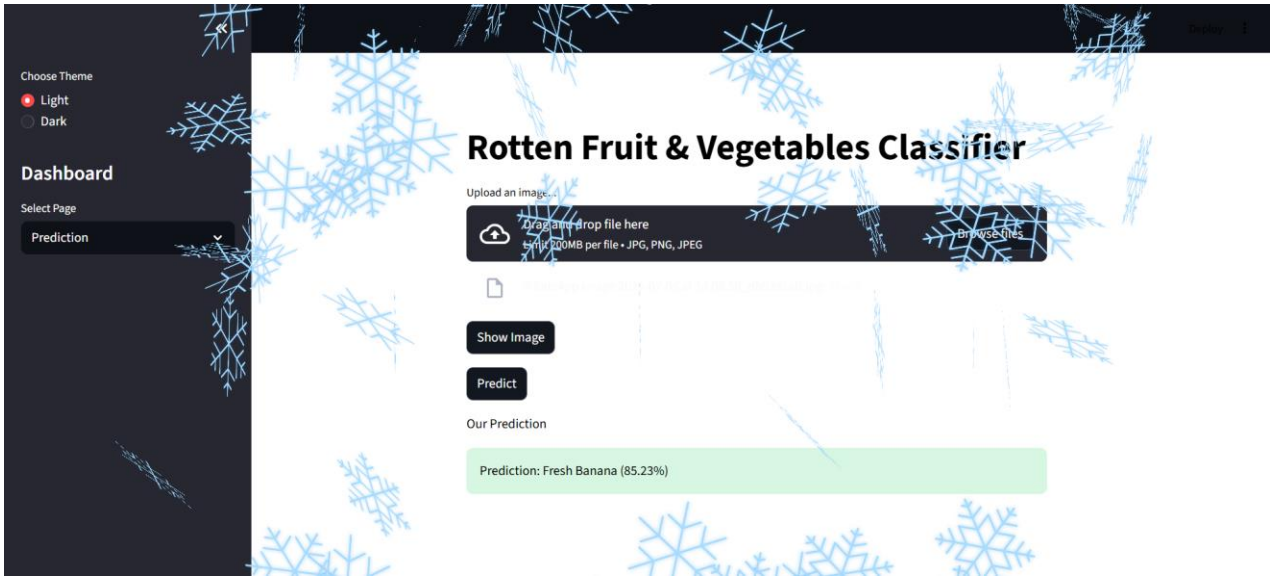
Both unit testing and manual testing methods are applied.

11. Screenshots or Demo






Choose Theme
Light
Dark
Dashboard
Select Page
Prediction
Rotten Fruit & Vegetables Classifier
Upload an image...
Drag and drop file here
Limit 200MB per file • JPG, PNG, JPEG
Browse files
Show Image
The use_column_width parameter has been deprecated and will be removed in a future release. Please utilize the width parameter instead.


Choose Theme
Light
Dark
Dashboard
Select Page
Prediction
Rotten Fruit & Vegetables Classifier
Upload an image...
Drag and drop file here
Limit 200MB per file • JPG, PNG, JPEG
Browse files
Show Image
Predict
Our Prediction
Prediction: Fresh Banana (85.23%)

12. Known Issues

Despite good performance, the system has some limitations:

1. The model detects only visible external spoilage.
2. Internal defects or hidden decay cannot be identified.
3. Image quality, lighting, and background noise may affect prediction accuracy.
4. Limited dataset size may reduce generalization ability.
5. Extreme image angles or partially visible objects may lead to incorrect classification.

These limitations highlight areas for improvement in future development.

13. Future Enhancements

The project can be enhanced further with advanced features such as:

1. Multi-class classification to identify specific fruit or vegetable types along with freshness status.
2. Real-time detection using webcam or mobile camera integration.
3. Cloud deployment using AWS, Azure, or Google Cloud for large-scale accessibility.
4. Mobile application integration for farmers and suppliers.
5. Expansion of dataset size for improved generalization.
6. Implementation of deeper CNN architectures (e.g., Transfer Learning using pre-trained models).
7. Integration of database for storing prediction history.
8. Integration with supply chain management systems for automated quality control.

These enhancements would transform the system from an academic prototype into a production-ready intelligent inspection solution.