ADITYA COLLEGE OF ENGINEERING AND TECHNOLOGY

Department of Artificial Intelligence and Machine Learning

Surampalem

INTERNSHIP REPORT

ON

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

Submitted in partial fulfillment of the requirements of the

Virtual Internship Program

Organized by

SMART BRIDGE

Submitted by

Chinni Rakesh (23P31A42F6)

Dasari Sirisha (24A95A0105)

Koppoju Sasi Kala (23A91A0499)

Korada Leela Ram Prasad (23A91A04F9)

TEAM ID:

LTVIP2025TMID34503

Under the Mentorship of

Mr. Anji Babu

Smart Bridge

June 2025

Index

BRAINSTORMING &IDEATION

Objective:

Problem Context

In the agriculture and food industry, identifying rotten fruits and vegetables remains a manual, error-prone, and time-consuming task. Whether in large food processing units, supermarkets, or homes, the absence of automated sorting leads to inefficiencies, food waste, and reduced quality assurance. Traditional methods lack precision and scalability, making it hard to maintain consistency in quality checks.

Purpose of the Project

Smart Sorting was developed to address these issues by implementing a deep learning-powered visual inspection system capable of:
• Detecting rotten fruits and vegetables using transfer learning on real-time images.
• Classifying produce based on freshness for better decision-making.
• Integrating with conveyor belts, supermarket docks, or smart fridges.
• Improving food quality, reducing waste, and increasing operational efficiency using AI.

Key Points:

- **Problem Statement**

  • Food processing, retail, and household setups rely heavily on manual sorting of fruits and vegetables.
  • Rotten produce often goes undetected, leading to food waste and health risks.
  • There is a lack of automated systems that use AI to ensure consistent, real-time quality checks.

- **Proposed Solution**

- **Food Processing Plants:** To automate sorting lines using AI-powered visual inspection.
- **Supermarkets:** To scan incoming shipments and flag spoiled produce instantly.
- **Smart Homes:** To monitor fridge contents and alert users about decaying items.
- **Developers/Admins:** To manage the model performance, updates, and integration with hardware.

➢ Target Users

- Quality Control Teams – For ensuring high accuracy in sorting fresh vs. rotten produce.
- Warehouse Managers – To reduce waste and streamline shipment inspections.
- Homeowners – To reduce food spoilage and save costs through fridge alerts.
- Retail Chains – To maintain consistent quality across multiple locations.
- Tech Developers – For integrating AI with IoT and automation systems.

➢ Expected Outcomes

- A real-time fruit and vegetable sorting system powered by transfer learning.
- Early detection of spoilage to reduce food waste and improve health outcomes.
- AI-driven dashboards with freshness status, trends, and wastage insights.
- Increased efficiency and trust in automated quality control systems.

REQUIREMENT ANALYSIS

Objective:

Functional Requirements:

The Smart Sorting system is designed to enhance the automation and accuracy of identifying rotten fruits and vegetables across various environments. At the core is a deep learning-based image classification model, fine-tuned using transfer learning techniques on a dataset of fresh and rotten produce. The system processes real-time images captured through cameras and classifies each item based on freshness. Depending on the environment—food processing plant, supermarket, or smart home—the system can trigger appropriate actions such as sorting, flagging, or alerting. A user-friendly web interface allows stakeholders to upload images, view predictions, and analyze performance metrics. Additionally, a visual dashboard provides insights on spoilage rates, freshness trends, and detection history. The model continuously improves through incremental learning based on newly labeled data, ensuring adaptability and accuracy across different settings. All image inputs, classification results, and timestamped logs are persistently stored in a backend database to support traceability and analytics.

Technical Requirements:

The Smart Sorting solution is implemented using a modern and modular AI stack. The backend uses Python with the Flask framework, enabling clean routing for prediction, logging, and dashboard APIs. Image classification is powered by pre-trained deep learning models like VGG16 or EfficientNet, adapted using TensorFlow or Keras libraries. The frontend is built using HTML, CSS, and JavaScript to provide an interactive interface for users to upload images, view classification results, and monitor freshness trends. Image data, model outputs, and historical logs are stored in a structured database such as SQLite or PostgreSQL. Configuration parameters, including model paths and environment variables, are managed securely through .env files and a centralized config.py. The system is designed for performance, targeting prediction response times under 3 seconds and supporting multiple concurrent requests. Security measures such as input validation and safe file handling are enforced to protect system integrity. Unit and integration tests are included to ensure system reliability and maintainability.

Key points:

➢ **Technical Requirements**:

• Backend developed using Flask (Python) with modular routing for predictions and logging.
• Frontend created using HTML, CSS, and JavaScript for interactive UI and image upload.
• AI integration handled via pre-trained models (e.g., VGG16, EfficientNet) using TensorFlow/Keras.
• Database setup using SQLite or PostgreSQL for storing predictions and image metadata.
• Sensitive data and configuration managed through .env files and config.py module.
• Prediction latency kept under 3 seconds for real-time sorting experience.
• Scalable and modular design to support different environments (plants, stores, homes).
• Input validation and safe file handling for data security and robustness.
• Unit and integration tests included to ensure quality and consistent model behavior.

➢ **Functional Requirements:**

• Enable real-time classification of fruits and vegetables as fresh or rotten.
• Accept images via upload or camera and return freshness prediction.
• Display classification results in an intuitive and user-friendly web interface.
• Log prediction history with timestamps and metadata for future analysis.
• Visualize spoilage trends and freshness stats via a dynamic dashboard.
• Allow system retraining with new labeled data for improved accuracy.

• Store all inputs, model outputs, and logs securely in a backend database.

Constraints & Challenges:

- **Model Size and Resource Limitations:**
  - o Large pre-trained models may require GPU or optimization for local deployment.
- **Real-Time Inference Speed:**
  - o Image preprocessing and model prediction must remain low-latency for usability.
- **Classification Accuracy:**
  - o Visual similarity of some fresh/rotten cases may lead to false positives/negatives.
- **Data Privacy & Storage:**
  - o Uploaded images and prediction logs must be handled securely.
- **UI Simplicity:**
  - o Interfaces must be intuitive for non-technical users in industries and homes.
- **Deployment & Infrastructure:**
  - o Hosting model files, APIs, and dashboards may require scalable cloud.

PROJECT DESIGN

Objective:

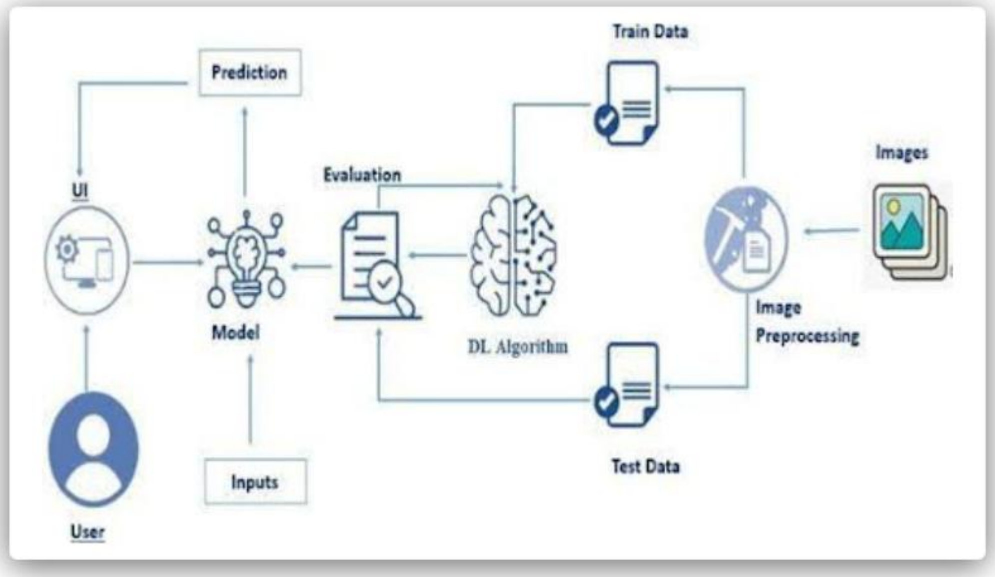
The objective of the Project Design phase is to define the overall structure, flow, and technical organization of the Smart Sorting system. It includes the layout of core components, interaction between modules, and the architecture that enables real-time image classification, freshness prediction, data logging, and dashboard visualization. This design ensures that the system remains scalable, modular, and maintainable, while delivering an efficient and accurate sorting experience for both industrial users and consumers.

Key points:

The Smart Sorting system follows a modular Flask architecture with organized components for routes, model logic, templates, and static files, ensuring clarity and ease of maintenance. Image uploads are handled via upload_routes.py, processed by model_inference.py using a pre-trained deep learning model, and the freshness result is displayed in real-time through predict.js. Uploaded images and classification outputs are stored securely in the database. Admins and users access performance trends through dashboard_routes.py, which retrieves and visualizes spoilage rates and prediction history. Image preprocessing and session logging ensure accuracy and traceability. Configuration settings such as model paths and database URIs are managed via .env files and config.py, supporting a smooth and scalable AI-based sorting workflow.

- **Architecture Diagram:**

Architecture:



➢ User flow:

- A user accesses the Smart Sorting platform and uploads an image of a fruit or vegetable through the web interface.
- The image is sent to the backend, where the deep learning model processes it and returns a freshness classification.
- The user receives a real-time result indicating whether the item is fresh or rotten, along with confidence level.
- An admin accesses the dashboard to review prediction logs, monitor spoilage trends, and track model accuracy.
- The system stores image history and classification data, allowing for future reference and trend-based analysis.

➢ UI/UX Consideration:

The Smart Sorting platform is designed with a clean, intuitive, and responsive interface to support users in industrial, retail, and home environments. The image upload section provides real-time classification feedback using smooth transitions and toast notifications for user actions. The layout leverages Bootstrap for responsive design, ensuring seamless usability across desktops, tablets, and mobile devices.

High-contrast visuals and accessible font sizes are implemented for better readability and accessibility. Upload buttons, result displays, and confidence indicators are clearly structured to guide users effortlessly. The dashboard uses interactive charts and simplified metrics to help administrators monitor spoilage trends and prediction performance at a glance. The overall design focuses on clarity, responsiveness, and ease of interaction to build user confidence and streamline quality control tasks.

PROJECT PLANNING

Objective:

The objective of the Project Planning phase is to structure the development of the Smart Sorting system into manageable tasks and timelines using an Agile methodology. By organizing work into focused sprints and assigning specific responsibilities to each team member, the project ensures consistent progress, timely iteration, and successful delivery of each component. This structured plan includes sprint objectives, task assignments, and milestone tracking.

Key Points:

➢ **Sprint Planning:**

- **Sprint 1: Project Setup**

  • Initialize Flask project with app.py, config.py, .env, and install necessary dependencies.

- **Sprint 2: Model Integration**

  • Load and test the pre-trained deep learning model using model_inference.py and prepare routing in predict_routes.py.

- **Sprint 3: Image Upload & Prediction**

  • Create image upload UI, connect it with backend prediction logic, and display results in predict.html.

- **Sprint 4: Dashboard Development**

  • Build an admin dashboard using dashboard_routes.py to visualize spoilage rates, history, and performance metrics.

- **Sprint 5: Image Preprocessing**

  • Implement image validation and preprocessing steps using helpers.py to ensure model compatibility and accuracy.

- **Sprint 6: Database Setup**

  • Define schema in models.py, manage records with init_db.py, and log predictions with timestamps.

- **Sprint 7: Testing & Optimization**

  • Conduct unit/integration testing, validate model predictions, and reduce latency for real-time use.

- **Sprint 8: Deployment**

  • Set up deployment config, prepare demo data, and finalize frontend UI and project documentation.

Task Allocation:

Members	Tasks
Chinni Rakesh	• Backend architecture (Flask) • Model integration and inference logic • API routing and response handling • Deployment configuration and setup • Database schema and storage design • Final system integration and optimization
Dasari Sirisha	• Frontend UI design • User image upload interface • Result display components • Styling and responsiveness (HTML/CSS/JS)
Koppoju Sasi Kala	• Dashboard analytics • Spoilage trend visualization • Admin panel layout and data presentation • Integration with prediction logs
Koppoju Sasi Kala	• Testing & validation • Unit and integration tests • Code review support • Documentation and presentation support

TimeLine and Milestones:

Date	Milestone
Week-1	• Flask project setup • Pre-trained model integration • Image upload & prediction module
Week-2	• Image preprocessing logic • Database design and connectivity • Prediction logging and result display
Week 3	• Dashboard implementation • Spoilage trend visualization • Admin panel integration
Week 4	• Final testing and bug fixes • UI polish and optimization • Documentation and cloud deployment

PROJECT DEVELOPMENT

Objective:

The objective of the Project Development phase is to implement, integrate, and test all modules of the Smart Sorting system, transforming the design into a fully functional application. This includes building the image classification engine, preprocessing pipeline, result display interface, admin dashboard, and backend database integration, ensuring all components work seamlessly together.

Key Points:

➢ **Technology Stack Used:**

• Backend Framework: Python with Flask (for modular API routing and image classification logic)
• Frontend Technologies: HTML, CSS, JavaScript (with Bootstrap for responsive design)
• AI Integration: Pre-trained CNN models (e.g., VGG16 or EfficientNet via TensorFlow/Keras) for freshness classification
• Image Preprocessing: Custom Python module using OpenCV and NumPy for real-time image handling
• Database: SQLite (development) / PostgreSQL (production-ready) for storing predictions and image metadata
• Configuration & Secrets: .env file and config.py for secure environment-based setup
• Version Control: Git and GitHub for source code tracking and collaboration
• Deployment (Optional): Docker for containerization and cloud hosting (e.g., Render, Heroku, AWS) for scalable deployment

➢ **Development Process:**

The development of Smart Sorting followed a structured and modular approach. The project was divided into independent yet connected components, ensuring flexibility, scalability, and easier debugging. Each major feature was implemented and tested step by step, starting from the backend setup to model integration and user interface development. Below are the key stages of the development process**.**

Flask Project Initialization:

• Created the basic Flask structure with app.py, config.py, and .env for secure and modular project setup.
• Installed necessary dependencies including Flask, TensorFlow/Keras for model integration, and SQLAlchemy for database interaction.

Prediction Module Implementation:

• Developed predict_routes.py to handle incoming image uploads and trigger model predictions.
• Integrated the deep learning model via model_inference.py to classify fruit and vegetable freshness.
• Designed predict.html and predict.js for real-time user interaction and result display on the frontend.

Prediction Logging & Data Handling:

• Created a form/interface to collect uploaded images and linked it to predict_routes.py for backend handling.
• Processed each image through model_inference.py to classify it as fresh or rotten using the trained model.
• Stored image metadata, prediction results, timestamps, and confidence scores into the database using SQLAlchemy models.

Dashboard Development:

• Built dashboard_routes.py to serve freshness prediction logs, spoilage trends, and classification statistics.
• Designed dashboard.html with embedded charts (using Chart.js) to visualize real-time analytics and historical data.
• Displayed freshness counts, spoilage rates, and model performance metrics to provide actionable insights for admin users.

Prediction Context Enhancement:

• Implemented session-based tracking to log user activity and associate predictions with specific sessions.
• Used helpers.py to manage preprocessing history and metadata, enabling result consistency and potential batch insights.
• Laid the foundation for adaptive learning, where future versions could tailor predictions based on usage context or prior results.

Database Integration:

• Defined all database schemas (e.g., ImageRecord, PredictionLog, UserSession) in models.py to store image metadata and classification results.
• Created init_db.py to initialize and reset tables during development and testing phases.
• Connected the application to SQLite for development and PostgreSQL for production-ready persistent storage of predictions, timestamps, and session data.

Testing and Final Integration:

• Performed module-wise testing of prediction logic, image upload, database logging, and dashboard rendering.
• Resolved synchronization issues between frontend components (predict.html, dashboard.html) and backend Flask routes.
• Ensured smooth end-to-end functionality from image upload and classification to real-time visualization on the admin dashboard.

➢ **Challenges & Fixes:**

• **Model Integration Issues:**
Initial model loading failed due to mismatched input dimensions. Fixed by standardizing image preprocessing in model_inference.py and resizing all inputs to the required dimensions.

• **Slow Prediction Response:**
Initial predictions exceeded expected latency due to unoptimized model execution. Resolved by caching the model and minimizing unnecessary processing in the Flask route.

• **Frontend and Backend Sync:**
Prediction results were not displaying correctly. Fixed

by ensuring consistent JSON response structures and proper async handling in predict.js.

- **Image Format Compatibility:**

Certain uploaded images caused runtime errors. Addressed by validating file types and applying fallback conversions using PIL before model inference.

- **UI Freezing & Reloads:**

Image upload triggered full-page reloads. Resolved by implementing asynchronous JavaScript with real-time result updates via AJAX.

- **Database Connection Errors:**

Faced issues with saving prediction logs due to uninitialized tables. Fixed by refining table schema in models.py and using init_db.py to reset the structure during testing.

FUNCTIONAL & PERFOMANCE TESTING

Objective:

The primary objective of the Functional and Performance Testing phase is to ensure that all features of the Smart Sorting system operate correctly and that the application performs reliably under real-time usage conditions. This phase involved testing each module individually and then as a complete system, verifying accuracy, response speed, and overall stability. The goal was to identify and fix bugs, inconsistencies, and performance issues before deployment.

Key Points:

➢ Test Cases Executed:

• **Image Upload & Prediction:**
Verified real-time image uploads and accurate freshness classification from the deep learning model.

• **Prediction Logging:**
Tested logging of image metadata, prediction results, and timestamps into the database.

• **Dashboard Visualization:**
Validated the accuracy of spoilage trends, freshness counts, and chart rendering in the admin panel.

• **Image Preprocessing:**
Ensured images of various formats and sizes were correctly resized and normalized for inference.

• **UI/UX Interaction:**
Checked responsiveness of upload forms, result displays, and seamless transitions without reloads.

• **Database Integrity:**
Confirmed proper schema implementation, data consistency, and reliable storage using SQLite/PostgreSQL.

• **Performance Testing:**
Measured classification latency (<3s), stress-tested multi-user prediction requests, and validated smooth UI rendering.

➢ Bug Fixes & Improvements:

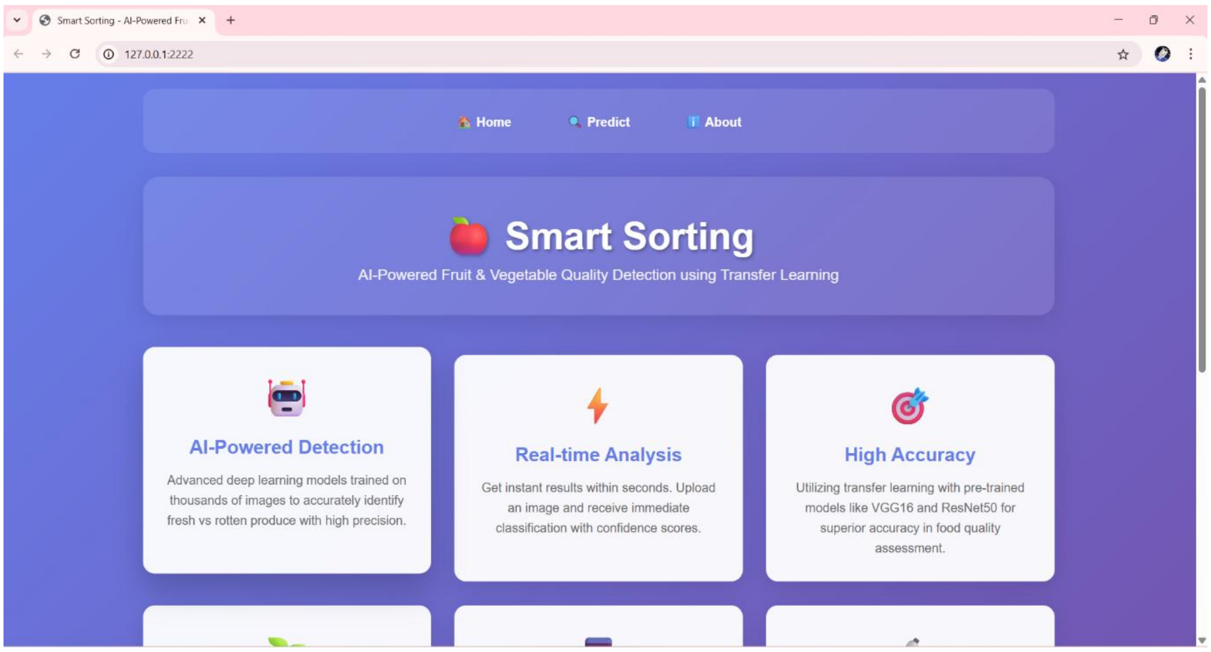
Several issues were identified and fixed during testing:

• **Model Loading Delay:**
Reduced model load time by initializing once and reusing the loaded instance across sessions.
• **Incorrect Predictions:**
Refined image preprocessing pipeline and ensured consistent input dimensions to improve classification accuracy.
• **Frontend-Backend Mismatch:**
Standardized JSON response formats to ensure smooth communication between prediction routes and the UI.
• **Dashboard Chart Errors:**
Fixed data parsing issues for timestamps and freshness labels to ensure accurate chart rendering.
• **UI Freezing During Prediction:**
Integrated async handling in JavaScript and added loading indicators to enhance user experience.

➢ Final Validation:

After thorough testing, the Smart Sorting system was confirmed to:
• Handle real-time image uploads and classify freshness accurately.
• Process, store, and log prediction results reliably in the database.
• Display spoilage trends and analytics correctly on the dashboard.
• Respond quickly (within 3 seconds) and support concurrent users.
• Meet all functional and performance requirements defined in the planning phase.

➢ Deployment:



Smart Sorting

AI-Powered Fruit & Vegetable Quality Detection using Transfer Learning

AI-Powered Detection

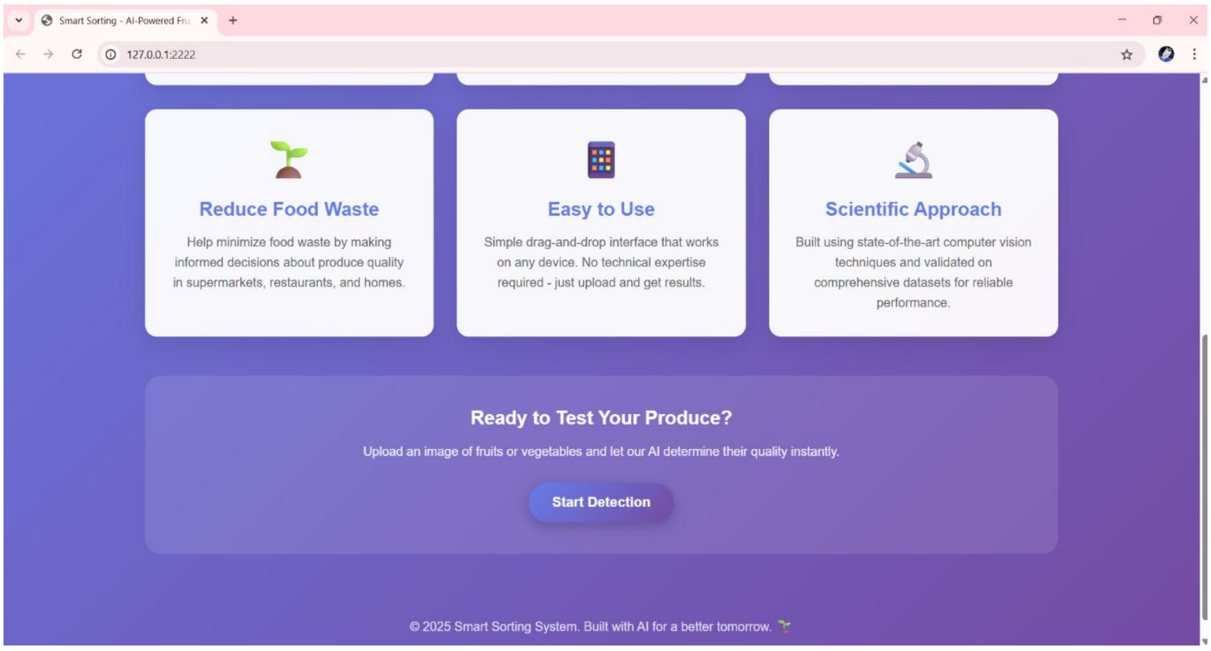
Advanced deep learning models trained on thousands of images to accurately identify fresh vs rotten produce with high precision.

Real-time Analysis

Get instant results within seconds. Upload an image and receive immediate classification with confidence scores.

High Accuracy

Utilizing transfer learning with pre-trained models like VGG16 and ResNet50 for superior accuracy in food quality assessment.



Reduce Food Waste

Help minimize food waste by making informed decisions about produce quality in supermarkets, restaurants, and homes.

Easy to Use

Simple drag-and-drop interface that works on any device. No technical expertise required - just upload and get results.

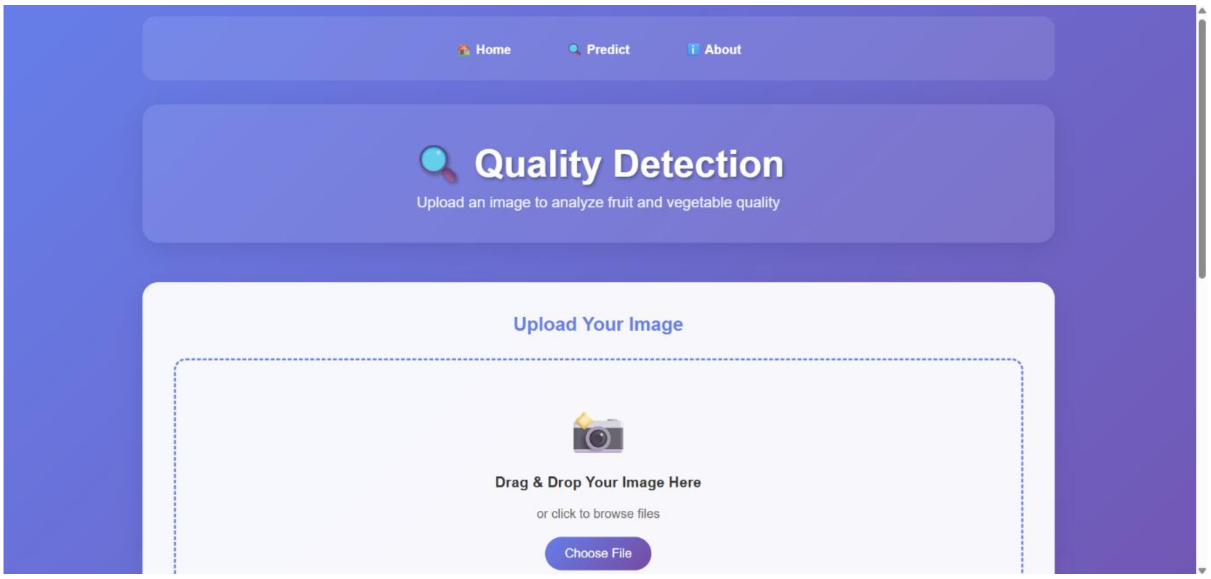
Scientific Approach

Built using state-of-the-art computer vision techniques and validated on comprehensive datasets for reliable performance.

Ready to Test Your Produce?

Upload an image of fruits or vegetables and let our AI determine their quality instantly.

© 2025 Smart Sorting System. Built with AI for a better tomorrow.



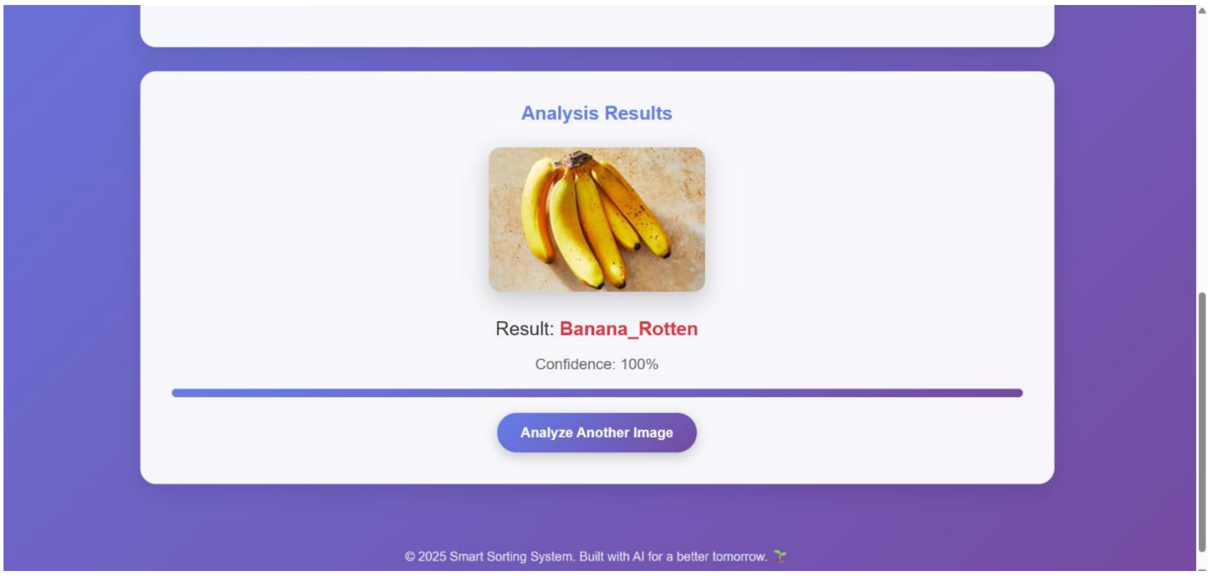
🔍 Quality Detection

Upload an image to analyze fruit and vegetable quality

Upload Your Image

Drag & Drop Your Image Here

or click to browse files


Analysis Results

Result: Banana_Rotten

Confidence: 100%

© 2025 Smart Sorting System. Built with AI for a better tomorrow. 🌱