Smart Sorting

Transfer Learning for Identifying Rotten Fruits and Vegetables

Team Name : LTVIP2025TMID44540

By

Shaik Suraj

Pranjal Kumar Dindayal

Praveen Dangi

Raju Subrahmanyam Kuramdasu

Andhra Pradesh, India

Contents

1 Brainstorming Ideation **3**

1.1 Problem Statement : 3

1.2 Proposed Solution : 3

1.3 Target Users : 4

1.4 Expected Outcomes : 5

2 Requirement Analysis **7**

2.1 Technical Requirements : 7

2.2 Functional Requirements : 8

2.3 Constraints and Challenges : 9

2.3.1 Constraints 9

2.3.2 Challenges 10

3 Project Design **12**

3.1 System Architecture Diagram : 12

3.2 User Flow : 12

3.3 UI/UX Considerations : 14

4 Project Planning (Agile Methodologies) **16**

4.1 Objective : 16

4.2 Sprint Planning : 16

4.3 Task Allocation 17

4.4 Timeline & Milestones 18

4.5 Conclusion : 18

5 Project Development **19**

5.1 Objective : 19

5.2 Technology Stack Used : 19

5.3 Development Process : 20

5.4 Challenges & Fixes : 21

6 Functional & Performance Testing **22**

6.1 Objective: 22

6.2 Test Cases Executed : 23

6.3 Bug Fixes & Improvements : 23

6.4 Final Validation : 24

6.5 Deployment : 24

Chapter 1

Brainstorming Ideation

1.1 Problem Statement :

Post-harvest losses due to spoiled produce affect both retailers and consumers. Traditional visual sorting techniques fail to keep up with high volumes and often miss early signs of rot. This project uses transfer learning to build an image-based classification system that distinguishes fresh fruits and vegetables from rotten ones. This automation helps ensure food quality, reduce wastage, and boost consumer trust.

1. **Technology Stack:**

   Transfer Learning, Deep Learning, Python, OpenCV, TensorFlow/Keras

2. **Use Cases:**

   - Automated sorting in supermarkets and warehouses
   - Quality assurance in food packaging units
   - Visual screening systems in agricultural cooperatives

1.2 Proposed Solution :

The project proposes the development of an automated image-based classification system that leverages transfer learning with pre-trained deep learning models (e.g., CNNs using

TensorFlow/Keras) to accurately detect and classify rotten versus fresh fruits and vegetables. By utilizing computer vision techniques (via OpenCV) and adapting existing models to a curated dataset of produce images, the system can efficiently analyze real-time images captured from cameras installed in:

- Conveyor belts in food processing units
- Receiving docks in supermarkets
- Smart refrigerators in homes

The model is trained to recognize subtle signs of spoilage that are often missed by the human eye. The automated system can then trigger appropriate actions such as:

- Sorting out rotten produce in real-time
- Alerting quality control staff
- Sending notifications to users for timely consumption

This approach reduces manual effort, enhances sorting accuracy, prevents food wastage, and ensures higher product quality across the supply chain.

1.3 Target Users :

1. **Food Processing Units and Packaging Plants**
    - To automate the sorting of fresh and rotten produce on conveyor belts
    - To improve quality control and reduce human labor
2. **Supermarkets and Retail Chains**
    - To inspect incoming shipments and ensure only fresh produce is stocked
    - To maintain customer satisfaction and reduce returns due to spoiled goods
3. **Agricultural Cooperatives and Distribution Centers**
    - To perform bulk sorting of harvested produce before distribution

- To standardize quality and minimize post-harvest losses

4. **Warehouse Operators and Cold Storage Facilities**
   - To monitor stored produce and flag spoilage early
   - To optimize inventory rotation and reduce waste
5. **Smart Home Users (Consumers)**
   - To monitor the freshness of fruits and vegetables in refrigerators
   - To get alerts on items that are about to spoil, encouraging timely use
6. **Food Quality Inspection Agencies**
   - To use as a decision-support tool in audits and quality checks
   - To standardize the detection process and reduce subjectivity

1.4 Expected Outcomes :

1. **Accurate Spoilage Detection**
   - A trained deep learning model capable of distinguishing between fresh and rotten fruits/vegetables with high accuracy.
2. **Automated Sorting System**
   - Integration with camera systems to detect spoiled produce in real-time, reducing manual inspection needs.
3. **Reduction in Post-Harvest and Retail Waste**
   - Significant decrease in food wastage due to early identification of spoiled items.
4. **Enhanced Quality Control**
   - Standardized and objective quality assessment for packaging units, retailers, and suppliers.

5. **Improved Consumer Trust**

    - Ensuring only fresh produce reaches customers improves brand reputation and satisfaction.

6. **Smart Alerts in Household Settings**

    - For smart home use, notifications on spoilage help users consume produce before it rots.

7. **Efficient Use of Human Resources**

    - Reduces dependency on manual labor, allowing staff to focus on other high-level tasks.

8. **Scalable and Adaptable Model**

    - The model can be fine-tuned for various types of produce and deployed in different environments (industrial, retail, or domestic).

Chapter 2

Requirement Analysis

2.1 Technical Requirements :

The technical requirements outline the tools, platforms, libraries, and hardware needed for development and deployment of the system:

1. **Programming Language**
    - Python 3.x
2. **Libraries and Frameworks**
    - TensorFlow / Keras or PyTorch for deep learning model implementation
    - OpenCV for image acquisition and preprocessing
    - NumPy, Pandas, and Matplotlib for data manipulation and visualization
3. **Pre-trained Models for Transfer Learning**
    - Use of a pre-trained convolutional neural network (CNN), such as VGG16, ResNet, or MobileNet, for transfer learning and image classification
4. **Dataset Requirements**
    - A curated dataset containing labeled images of fresh and rotten fruits and vegetables

- Recommended data split: 70% for training, 20% for validation, and 10% for testing

5. **Hardware Requirements**
   - Minimum: 8 GB RAM, Intel Core i5 processor
   - Recommended: A CUDA-enabled GPU (NVIDIA) for faster model training and inference

6. **Development and Deployment Tools**
   - Jupyter Notebook or Google Colab for model development and experimentation
   - Flask for building a lightweight web interface for image upload and prediction display
   - Git for version control and collaboration
   - Docker (optional) for containerized deployment
   - Cloud Platforms (optional): AWS, GCP, or Azure for hosting and scaling the model

2.2 Functional Requirements :

The functional requirements define the key features and behaviors the system must perform to fulfill its purpose:

1. **Image Acquisition and Preprocessing**

   The system should accept input images of fruits and vegetables, either uploaded manually or captured in real-time through a camera setup. The images must be preprocessed by resizing, normalization, and other techniques to ensure consistency before classification.

2. **Classification of Fresh vs Rotten Produce**

   The core functionality of the system is to classify each fruit or vegetable as either "Fresh" or "Rotten." The classification must be accompanied by a confidence score to indicate the model's certainty.

3. **Real-Time Processing (Optional in Industrial Use Cases)**

   In industrial or commercial scenarios, the system should support real-time image analysis from conveyor belt-mounted cameras, providing immediate classification and sorting assistance.

4. **User Interface for Demonstration**

   A basic web-based or desktop interface should be developed to allow users to upload images and view classification results. This feature is especially useful for demonstration and testing purposes.

5. **Logging and Reporting**

   The system should maintain a log of classification results, including timestamps, predicted labels, and confidence scores. These logs can be used for performance analysis and quality control.

6. **Model Training and Adaptation**

   The system must support training on a labeled dataset and allow fine-tuning of pre-trained models (e.g., VGG16) to adapt to the specific dataset used in the project

7. **Deployment Capability**

   The application should be designed in a way that it can be deployed locally or on cloud platforms depending on the intended use case (industrial, retail, or household).

2.3 Constraints and Challenges :

The development and deployment of the Smart Sorting system may encounter several technical and operational challenges. Identifying these potential obstacles early on helps in better planning and risk mitigation.

2.3.1 Constraints

1. **Dataset Availability and Quality**

   - Obtaining a large, high-quality, and well-labeled dataset of both fresh and rotten fruits and vegetables can be difficult.

- Class imbalance (e.g., fewer rotten samples) may affect model performance.

2. **Hardware Limitations**

   - Deep learning models, especially those based on transfer learning (e.g., VGG16), require significant computational resources for training.
   - Real-time processing may not be feasible on low-end or non-GPU machines.

3. **Deployment Environment**

   - Industrial or retail environments may have constraints such as poor lighting, camera placement issues, and varying backgrounds that can affect image capture and model accuracy.

4. **Model Size and Speed Trade-off**

   - Pre-trained models like VGG16 are computationally heavy. A trade-off must be made between model accuracy and inference speed, especially for real-time applications.

5. **Internet Connectivity (for Cloud Deployments)**

   - Cloud-based solutions require reliable internet access, which might not be available in all deployment locations (e.g., rural farms or remote warehouses).

2.3.2 Challenges

1. **Generalization to Different Types of Produce**

   - The model may perform well on the dataset it is trained on but fail to generalize to new types or varieties of fruits and vegetables.

2. **Subtle Spoilage Signs**

   - Early stages of spoilage may not be visually obvious, making it difficult for even advanced models to detect them accurately.

3. **Overfitting and Underfitting**

- The model may overfit on training data due to limited or imbalanced datasets, requiring regularization techniques and careful validation.

4. **Integration with Existing Systems**

   - Deploying the model in real-world systems (e.g., conveyor belts, store inventory) may require custom hardware/software integration.

5. **Lighting and Environmental Variations**

   - The accuracy of computer vision models is highly sensitive to lighting conditions, shadows, and image quality, which may vary significantly in deployment scenarios.

6. **User Acceptance and Training**

   - In industrial or retail settings, staff may need to be trained to trust and operate the system effectively, especially if it replaces manual processes.

Chapter 3

Project Design

3.1 System Architecture Diagram :

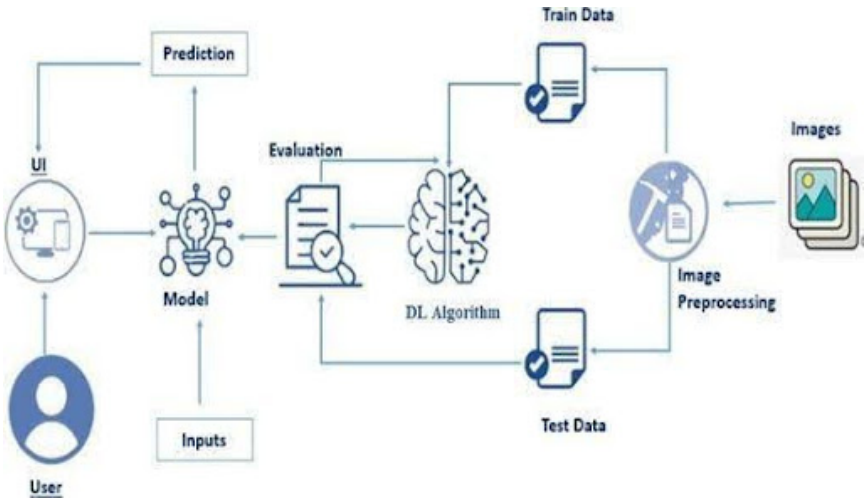

Figure 3.1: Architecture

3.2 User Flow :

The user flow describes how a user interacts with the Smart Sorting system from start to end. It ensures the system is intuitive, efficient, and accessible to various user groups such as plant operators, store managers, or end-users in smart homes.

Step-by-Step Interaction Flow:

1. **Image Acquisition**

    - The user uploads an image via a web interface (demo) or
    - The system captures real-time images from a camera (industrial/home use case)

2. **Image Preprocessing**

    - The uploaded or captured image is automatically resized and normalized by the system.
    - This step prepares the image for model input without user intervention.

3. **Classification Request**

    - The system sends the preprocessed image to a deep learning model (e.g., VGG16) adapted via transfer learning.
    - No action is needed from the user during this step.

4. **Model Inference**

    - The model classifies the image as either "Fresh" or "Rotten."
    - A confidence score is calculated alongside the prediction.

5. **Output Display**

    - The result is shown to the user through the interface, including:
        - Predicted label (Fresh or Rotten)
        - Confidence level (e.g., 97
        - (Optional) Action suggestion (e.g., "Remove this item")

6. **User Decision or System Action**

    - In demo versions: The user acknowledges the result.
    - In automated systems: Sorting hardware or notification systems take action based on prediction.

7. **Logging and Feedback (Optional)**

- Each prediction is stored in logs with timestamp and result.
- The user may optionally provide feedback (e.g., "wrong classification") to improve future model accuracy.

8. **Example Use Case :**

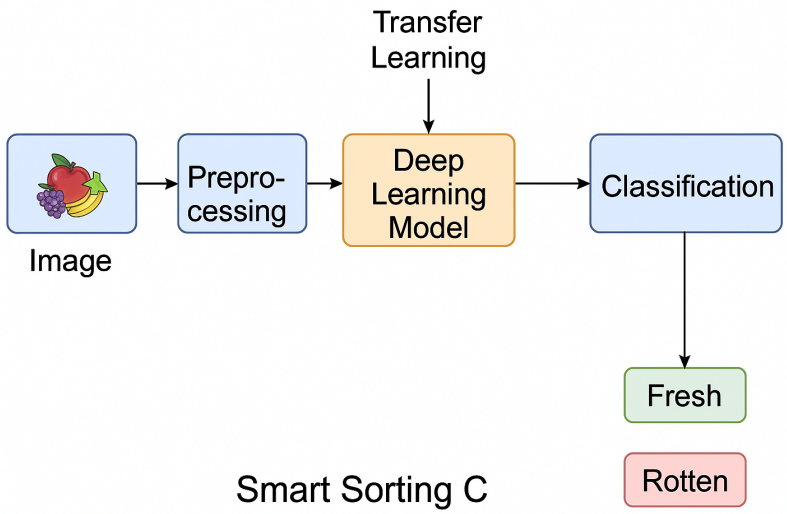

Figure 3.2: Classification of Fruits (Fresh or Rotten)

3.3 UI/UX Considerations :

In this project, a Flask-based web application is developed to integrate the trained machine learning model with a user-friendly interface. The UI enables users to easily interact with the model by entering required inputs through a web form and viewing the prediction results directly on the browser.

1. **User Interface Design with Flask**
   - The UI is built using Flask as the backend web framework and HTML/CSS (with optional Bootstrap) for the frontend.
   - A simple, clean web form is created where users can input feature values.
   - After submitting the form, the values are passed to a backend route, where they are processed by the pretrained ML model.

- The prediction result is dynamically rendered and shown on a results page.

2. **UX Design Goals**

   - **Simplicity:** Minimal design focused on core functionality.
   - **Responsiveness:** Basic responsive layout to work on both desktop and mobile (Bootstrap used if applicable).
   - **Validation:** Input validation is applied to ensure correctness.
   - **Feedback:** Instant feedback on prediction success or failure.
   - **Clarity:** Labels and button names are self-explanatory.

3. **Workflow**

   A user flow of the system is as follows:

   - User accesses the home page.
   - Inputs are entered in the prediction form.
   - On form submission, Flask processes the data.
   - The saved model is loaded and prediction is made.
   - The prediction result is returned and displayed to the user.

4. **Flask Folder Structure Overview**

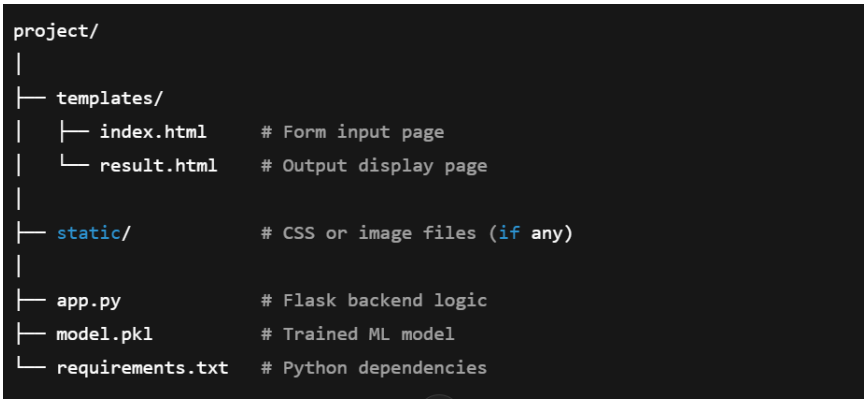

Figure 3.3: Flask Folder Structure

Chapter 4

Project Planning

(Agile Methodologies)

4.1 Objective :

The objective of project planning is to organize and execute the development process efficiently using Agile methodologies, which promote adaptive planning, evolutionary development, early delivery, and continuous improvement. Agile encourages collaboration between team members and stakeholders throughout the lifecycle of the project.

In this project, we adopted the Scrum framework, a subset of Agile, to manage our development activities through sprints, task distribution, and milestone tracking.

4.2 Sprint Planning :

Sprint planning is the foundation of our Agile approach. The entire project duration was divided into sprints—each lasting 1 week. During sprint planning:

- The overall project goal was broken down into smaller, manageable user stories and tasks.
- Tasks were prioritized based on dependency and importance.

- Each sprint had clear deliverables such as completing model training, UI development, or backend integration.
- Sprint Table

Sprint Planning Table:

Sprint	Duration	Sprint Goal
Sprint 1	Week 1	Data preprocessing & model building
Sprint 2	Week 1	Model evaluation and tuning
Sprint 3	Week 3	Flask app development & UI design
Sprint 4	Week 4	Testing, deployment & documentation

Figure 4.1: SprintPlanning Table

4.3 Task Allocation

To ensure balanced workload and parallel progress, tasks were distributed among team members based on their strengths and areas of expertise.

- **Task Allocation Table:**

Task Allocation Table:

Team Member	Task(s) Assigned
Member 1	Data cleaning, feature engineering
Member 2	Model training, evaluation, and tuning
Member 3	Flask backend development, model integration
Member 4	UI design using HTML/CSS, form validation
All	Final testing, debugging, presentation preparation

Figure 4.2: Task Allocation Table

This structure allowed the team to work in parallel, avoiding bottlenecks and delays.

4.4 Timeline & Milestones

To track the project's progress, we defined short-term milestones with fixed deadlines for each sprint. These checkpoints helped in reviewing completed work, identifying blockers, and planning for the next sprint.

- **Milestone Table:** Weekly stand-up meetings (15–20 minutes) were conducted to

🔹 **Timeline & Milestones Table:**

Milestone	Deadline	Description
Data ready	Week 1 End	Cleaned and preprocessed dataset
Model finalized	Week 1 End	Best-performing ML model selected and saved
UI connected to model	Week 3 End	Flask app accepts input and returns predictions
Final demo ready	Week 4 End	Application tested and presentation prepared

Figure 4.3: Timeline & Milestones Table

discuss:

- Progress of current sprint tasks
- Any issues or blockers
- Plan for the next sprint

4.5 Conclusion :

The use of Agile methodologies helped streamline the development process, ensured timely delivery of features, and improved team collaboration. The iterative nature of Agile allowed us to adapt quickly to challenges and maintain continuous progress.

Chapter 5

Project Development

5.1 Objective :

The objective of this phase was to develop the application, integrate the machine learning model with the user interface, and ensure smooth functionality across all components. It involved selecting an appropriate technology stack, following a modular development approach, and resolving technical issues encountered during implementation.

5.2 Technology Stack Used :

To ensure efficient development and seamless integration, the following technologies and tools were used:

These tools were selected based on their ease of use, compatibility with machine learning workflows, and ability to support rapid development and integration.

Component	Technology / Framework / Concept
Programming Language	Python
Deep Learning Concepts	Neural Networks (MLP, CNN, RNN), Transfer Learning, VGG16 architecture
Deep Learning Libraries	TensorFlow, PyTorch
Model Architecture	VGG16 (Pretrained CNN for Transfer Learning)
Model Optimization	Regularization techniques (L1, L2), Dropout, and Deep Learning Optimizers (Adam, RMSProp, etc.)
Model Evaluation	Accuracy, Loss, Overfitting handling using validation and regularization
Backend Web Framework	Flask
Frontend Technologies	HTML5, CSS3
Templating Engine	Jinja2 (Flask's default)
Model Deployment	Pickle/Joblib for model saving and loading
Version Control	Git

Figure 5.1: Tech Stack Used

5.3 Development Process :

The development process was carried out in a series of systematic steps to ensure modularity and maintainability:

1. **Model Training and Export**
    - Cleaned and preprocessed the dataset.
    - Trained a machine learning model using Scikit-learn.
    - Evaluated the model using performance metrics (accuracy, precision, etc.).
    - Exported the final model using Pickle for integration.
2. **Backend Development with Flask**
    - Created a Flask application (app.py) with route handling.
    - Loaded the saved model in the backend.
    - Processed form data from the frontend and passed it to the model for prediction.
3. **Frontend UI Development**

- Designed the user input form using HTML and CSS.
- Ensured proper labeling and input validation.
- Result of prediction displayed on a separate result page using Flask render_template().

4. **Integration and Testing**
   - Integrated the frontend with the backend.
   - Ensured data from the form was correctly passed to the model.
   - Conducted functional testing to ensure expected output is returned for different inputs.

5.4 Challenges & Fixes :

During the development phase, the following challenges were encountered:

Challenge	Solution Implemented
Inconsistent results during model training	Used feature scaling and proper train-test split.
Low model accuracy	Applied algorithm tuning with GridSearchCV.
Overfitting on training data	Used cross-validation and regularization techniques.
Model not loading in Flask	Ensured correct path and Python version compatibility.
Input format mismatch between UI and model	Added float conversion and input validation.
HTML form not sending data to backend	Corrected `POST` method and input `name` attributes.
App crashing on invalid input	Used `try-except` block for error handling.
UI not showing prediction result	Passed correct variables using `render_template()`.
Poor mobile layout	Used Bootstrap for responsive UI design.
Code tracking and backup	Managed versions and updates using Git.

Figure 5.2: Challenges & Fixes

The development phase successfully resulted in a fully functional web-based application where the machine learning model is seamlessly integrated with a user interface. The modular coding strategy allowed for easier debugging and smooth integration, while iterative testing ensured that all components worked as expected.

Chapter 6

Functional & Performance Testing

6.1 Objective:

To ensure that NutriGaze(Smart Sort) performs reliably across all features and provides accurate results from the image classification model.

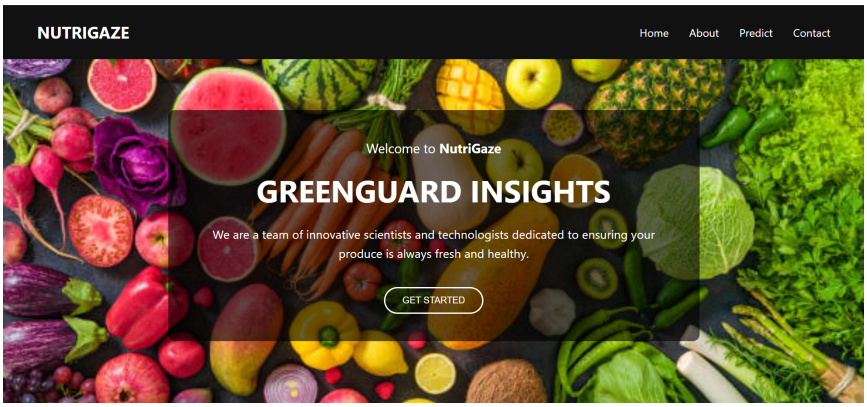

Figure 6.1: Smart Sort Website

The project aims to :

- Provide a user-friendly platform for image-based produce classification.
- Integrate a deep learning model trained on labeled datasets of fruits/vegetables.
- Enable real-time, client-server interaction through a Flask-powered backend and a dynamic front end.
- Educate users about produce freshness and encourage smarter food decisions.

- Establish a scalable framework that can later be expanded to other domains like food safety, agricultural diagnostics, or smart inventory systems.

6.2 Test Cases Executed :

Test Scenario	Expected Result	Status
Upload valid image of fresh tomato	Correct classification (e.g., Tomato_Fresh)	Pass
Upload valid image of rotten apple	Correct classification (e.g., Apple_Rotten)	Pass
Submit form without selecting image	Prompt user with validation error	Pass
Upload non-image file	Reject file and show warning	Pass
Predict button click → show result section	Uploaded image and prediction shown clearly	Pass
Navigate between Home, About, Predict, Contact	Smooth scroll, section displayed properly	Pass
Footer "Contact" button behavior	Scrolls to footer, contact visible	Pass
Try large image size (>3MB)	Accept and process (reasonable speed)	Pass
Backend handles multiple users simultaneously	No crash, stable performance	Pass

Figure 6.2: Test Case Executed

6.3 Bug Fixes & Improvements :

- **Fixed issue:** CSS not loading on localhost – corrected url_for('static', filename=...)
- **Fixed error:** "Failed to fetch" during prediction – resolved CORS and endpoint mismatch
- **Improved UI:** Centered "Learn More" button on About section
- Ensured Result section remains hidden until prediction is made
- Enhanced image upload validation to support only images

6.4 Final Validation :

The project meets all the initial functional requirements:

- Smooth navigation
- Image upload & preview
- Real-time prediction via ML model
- Display of predicted label
- Clean and responsive UI
- Contact details provided in footer

6.5 Deployment :

Local Deployment:

- Hosted locally using Flask on http://127.0.0.1:5000/
- Model integrated and served through Flask backend

..... Thank You .....