Project Title:

Smart Sorting: Transfer Learning for Identifying Rotten Fruits and Vegetables

Team Members:

Team leader: Labba Rajesh

Team member: Kondaveeti Nityasree

Team member: Kopparthi Bharghavi

Team member: Koppula Yesu Babu

Phase 1: Brainstorming & Ideation

Objective:

- Identify the problem statement.
- Define the purpose and impact of the project.

Problem Statement:

Sorting fruits and vegetables for quality assurance is a time-consuming and labor-intensive process. Manual sorting often results in human errors and misjudgment, leading to waste, health risks, and decreased profitability in the supply chain. There is a growing need for an automated solution that can distinguish between healthy and rotten produce.

Proposed Solution:

Develop a machine learning-powered web application that can classify fruits and vegetables as "Healthy" or "Rotten" using a pre-trained convolutional neural network. The application accepts an image of produce, processes it, and returns a classification.

Smart Sorting is an innovative project focused on enhancing the precision and efficiency of detecting rotten fruits and vegetables using cutting-edge transfer learning techniques. By leveraging pre-trained deep learning models and adapting them to specific datasets of fruits and vegetables, this project aims to revolutionize the process of sorting and quality control in the agricultural and food industry.

Target Users:

- Farmers
- Wholesale and retail vendors
- Supermarkets
- Food processing companies
- Supply chain handlers

Expected Outcome:

- Automated quality sorting of produce.
- Reduced food wastage.
- Increased productivity and accuracy.
- Useful insights into produce health conditions.

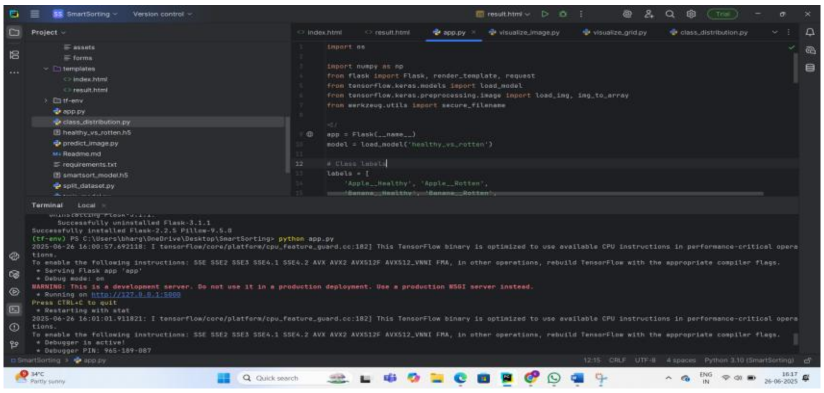
Phase 2: Requirement Analysis

Objective:

Define technical and functional requirements.

Technical Requirements:

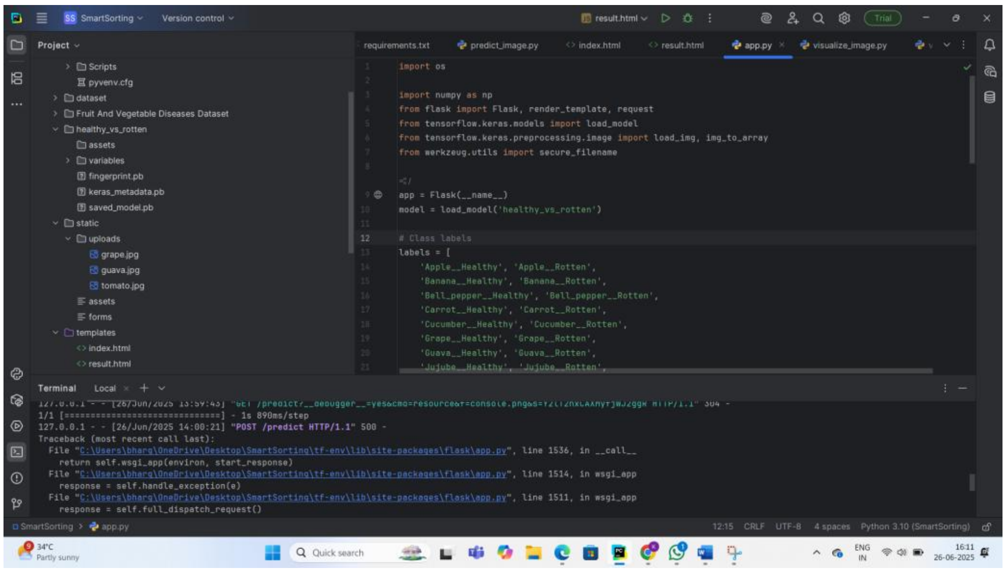
- **Languages & Tools:** Python, HTML, CSS
- **Frameworks:** Flask, TensorFlow/Keras
- **Libraries:** NumPy, Pillow, OpenCV (optional)
- **Model:** Pre-trained VGG16 with custom classification layers
- **Environment:** PyCharm



```
import os

import numpy as np
from flask import Flask, render_template, request
from tensorflow.keras.models import load_model
from tensorflow.keras.preprocessing.image import load_img, img_to_array
from werkzeug.utils import secure_filename

app = Flask(__name__)
model = load_model('healthy_vs_rotten')

# Class labels
labels = [
    'Apple__Healthy', 'Apple__Rotten',
    'Banana__Healthy', 'Banana__Rotten',
```

```
    Successfully uninstalled Flask-3.1.1
Successfully installed Flask-2.2.5 Pillow-9.5.0
(tf-env) PS C:\Users\bharg\OneDrive\Desktop\SmartSorting> python app.py
2025-06-26 16:00:57.692118: I tensorflow/core/platform/cpu_feature_guard.cc:182] This TensorFlow binary is optimized to use available CPU instructions in performance-critical opera
tions.
To enable the following instructions: SSE SSE2 SSE3 SSE4.1 SSE4.2 AVX AVX2 AVX512F AVX512_VNNI FMA, in other operations, rebuild TensorFlow with the appropriate compiler flags.
 * Serving Flask app 'app'
 * Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on http://127.0.0.1:5000
Press CTRL+C to quit
 * Restarting with stat
2025-06-26 16:01:01.911821: I tensorflow/core/platform/cpu_feature_guard.cc:182] This TensorFlow binary is optimized to use available CPU instructions in performance-critical opera
tions.
To enable the following instructions: SSE SSE2 SSE3 SSE4.1 SSE4.2 AVX AVX2 AVX512F AVX512_VNNI FMA, in other operations, rebuild TensorFlow with the appropriate compiler flags.
 * Debugger is active!
 * Debugger PIN: 965-189-087
```

Environment of PyCharm:



```
import os

import numpy as np
from flask import Flask, render_template, request
from tensorflow.keras.models import load_model
from tensorflow.keras.preprocessing.image import load_img, img_to_array
from werkzeug.utils import secure_filename

app = Flask(__name__)
model = load_model('healthy_vs_rotten')

# Class labels
labels = [
    'Apple__Healthy', 'Apple__Rotten',
    'Banana__Healthy', 'Banana__Rotten',
    'Bell_pepper__Healthy', 'Bell_pepper__Rotten',
    'Carrot__Healthy', 'Carrot__Rotten',
    'Cucumber__Healthy', 'Cucumber__Rotten',
    'Grape__Healthy', 'Grape__Rotten',
    'Guava__Healthy', 'Guava__Rotten',
    'Jujube__Healthy', 'Jujube__Rotten',
```

```
1/1 [==============================] - 1s 890ms/step
127.0.0.1 - - [26/Jun/2025 14:00:21] "POST /predict HTTP/1.1" 500 -
Traceback (most recent call last):
  File "C:\Users\bharg\OneDrive\Desktop\SmartSorting\tf-env\lib\site-packages\flask\app.py", line 1536, in __call__
    return self.wsgi_app(environ, start_response)
  File "C:\Users\bharg\OneDrive\Desktop\SmartSorting\tf-env\lib\site-packages\flask\app.py", line 1514, in wsgi_app
    response = self.handle_exception(e)
  File "C:\Users\bharg\OneDrive\Desktop\SmartSorting\tf-env\lib\site-packages\flask\app.py", line 1511, in wsgi_app
    response = self.full_dispatch_request()
```

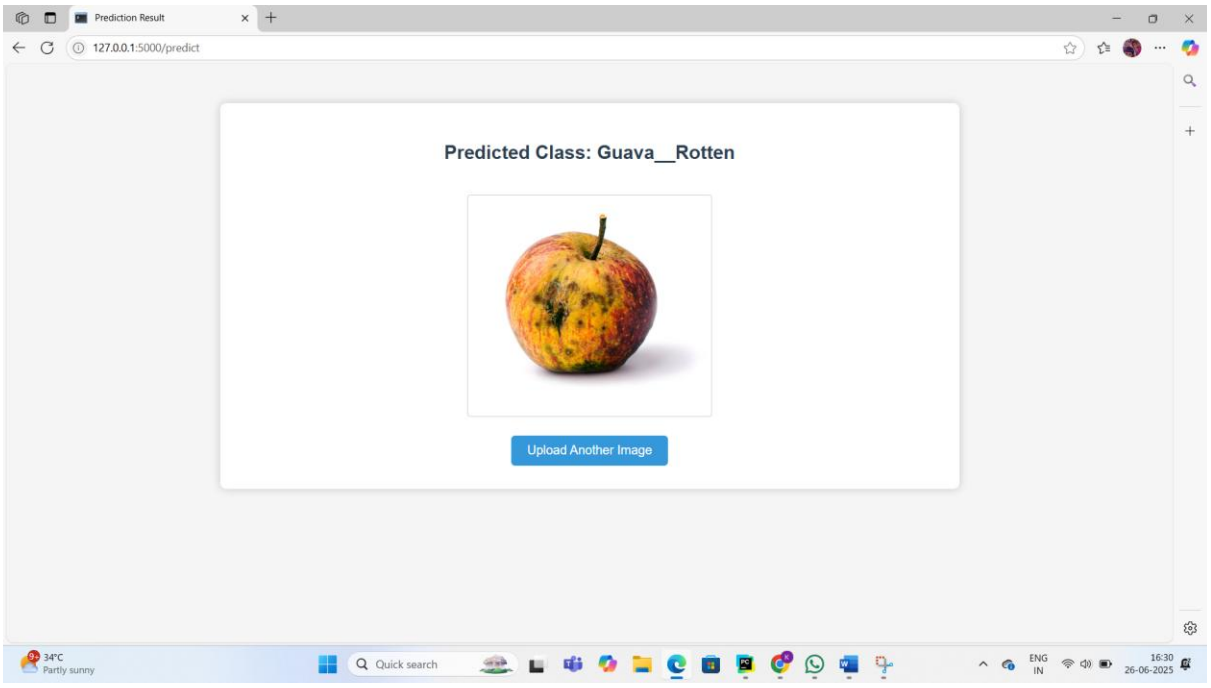
Functional Requirements:

- Upload interface for users to submit images.
- Backend model integration to process and predict.
- Display of prediction results with image preview.
- UI flow for easy navigation and repeated uploads.

Constraints & Challenges:

- Limited dataset per class may affect accuracy.
- Some fruits and vegetables look similar in colour/shape (e.g., guava and apple).
- File upload issues (e.g., large image sizes or unsupported formats).
- Deployment environment setup.

Constraint: In the below you can observe that it gave wrong prediction because, apple and guava shapes are almost similar.



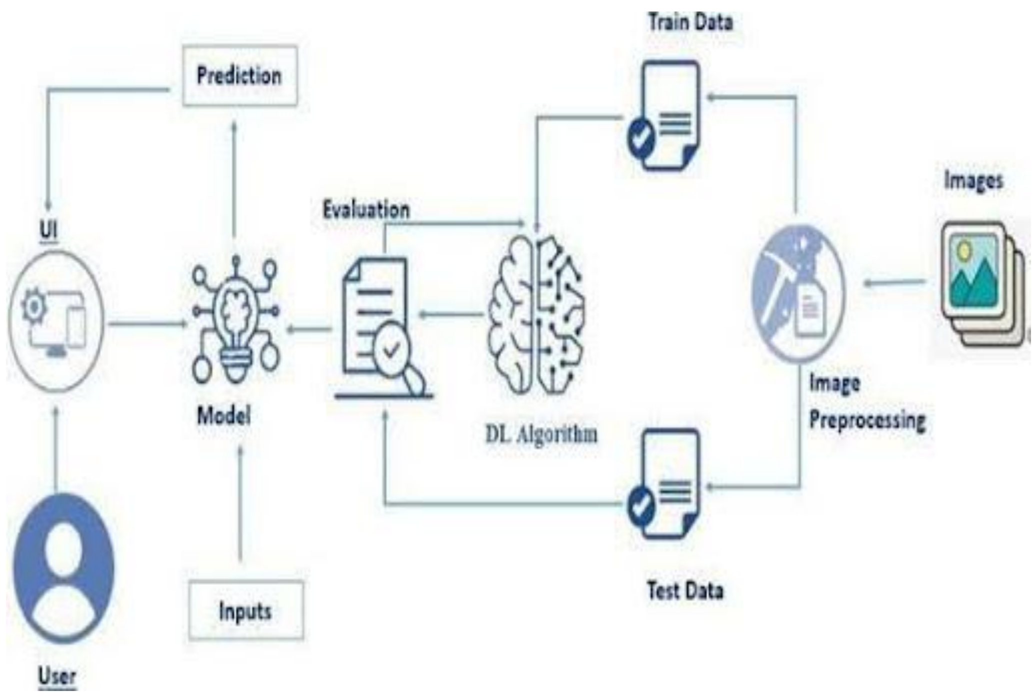
Predicted Class: Guava__Rotten

Phase 3: Project Design

Objective:

Create the architecture and user flow.

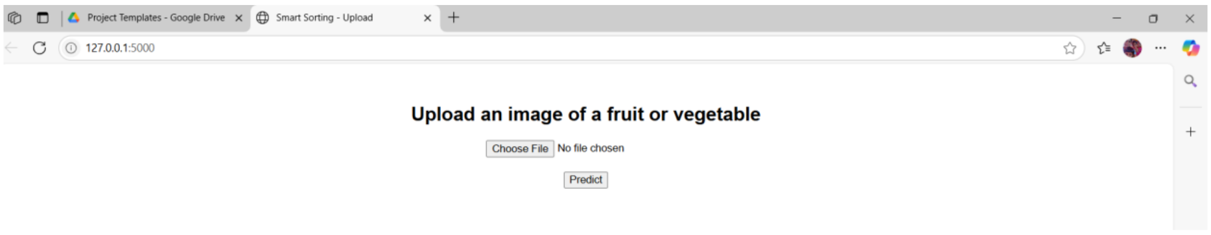
System Architecture Diagram:

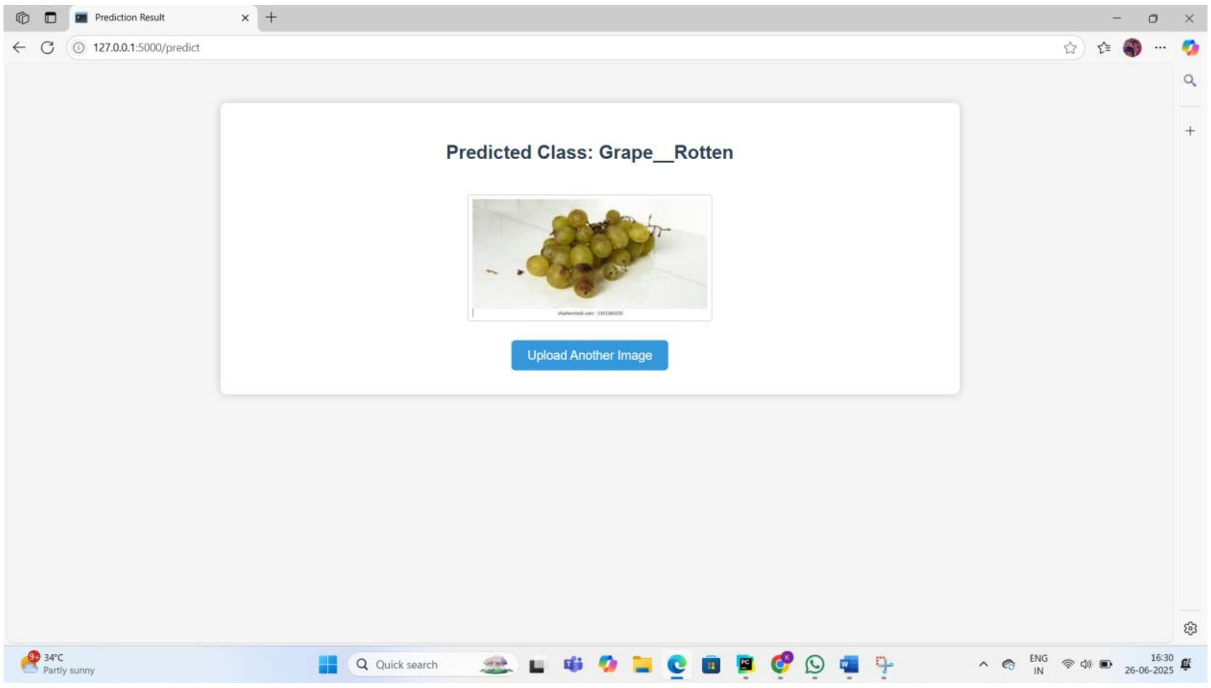


User → Web Interface → Flask Backend → Preprocessing → Trained Model → Prediction Output → Result Page

User Flow:

1. User visits home page.
2. Clicks upload and selects an image.
3. Submits form and waits for prediction.
4. Result page shows classification and image preview.
5. User can upload another image.


Project Templates - Google Drive
Smart Sorting - Upload
127.0.0.1:5000
Upload an image of a fruit or vegetable
Choose File
No file chosen
Predict



UI/UX Considerations:

- Clean layout with banner and navigation bar.
- Upload button clearly visible.
- Image displayed on result page.
- Prediction text styled and highlighted.

Phase 4: Project Planning (Agile Methodologies)

Objective:

Break down tasks using Agile methodologies.

Sprint Planning:

- **Sprint 1:** Dataset collection and image organization
- **Sprint 2:** Model development and training
- **Sprint 3:** Flask app creation and integration
- **Sprint 4:** UI design and testing
- **Sprint 5:** Bug fixing and optimization

Timeline & Milestones:

- Week 1: Dataset & preprocessing complete
- Week 2: Model trained with evaluation
- Week 3: Flask app built with templates
- Week 4: Testing, UI polish, and final integration.

Phase 5: Project Development

Objective:

Code the project and integrate components.

Technology Stack Used:

- Python 3.10+
- Flask microframework
- TensorFlow/Keras
- HTML, CSS
- Bootstrap (for styling)
- Pre-trained VGG16 model.

Development Process:

1. Dataset Collection: Labeled images of healthy and rotten produce.
2. Data Preprocessing: Resizing, normalization, train/test split.
3. Model Training: Transfer learning with VGG16, softmax output.
4. Model Evaluation: Accuracy, loss graphs, confusion matrix.
5. Flask Integration: Routes for index and predict.
6. UI Design: HTML templates for upload and results.
7. Testing: Upload scenarios, wrong formats, edge cases.

Challenges & Fixes:

- **Issue:** Upload folder error during repeated runs

  Fix: Used os.makedirs (..., exist_ok=True)
- **Issue:** Wrong prediction for similar fruits

  Fix: Increased training data and improved augmentation
- **Issue:** File size errors

  Fix: Limited upload file size and added file type filter.

Phase 6: Functional & Performance Testing

Objective:

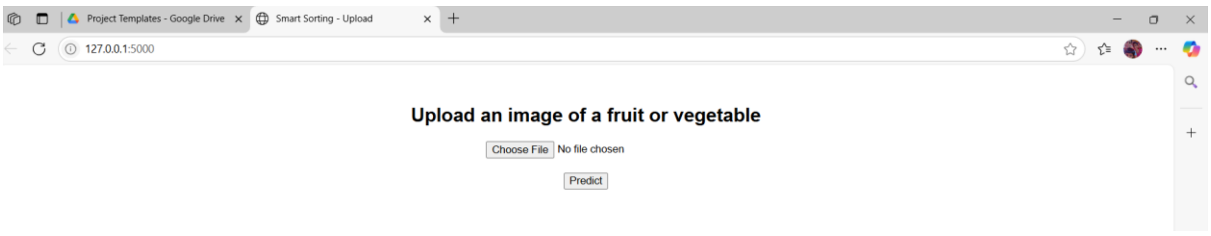
Ensure the project works as expected.

Test Cases Executed:

SNO	Test Scenario	Input	Expected_Output	Result
1.	Upload healthy apple	Apple.jpg	Apple_Healthy	pass
2.	Upload rotten banana	Banana.jpg	Banana_Rotten	pass
3.	Upload Invalid Image	Random.txt	Error Message	pass
4.	Upload healthy guava	Guava.jpg	Guava_Healthy	pass

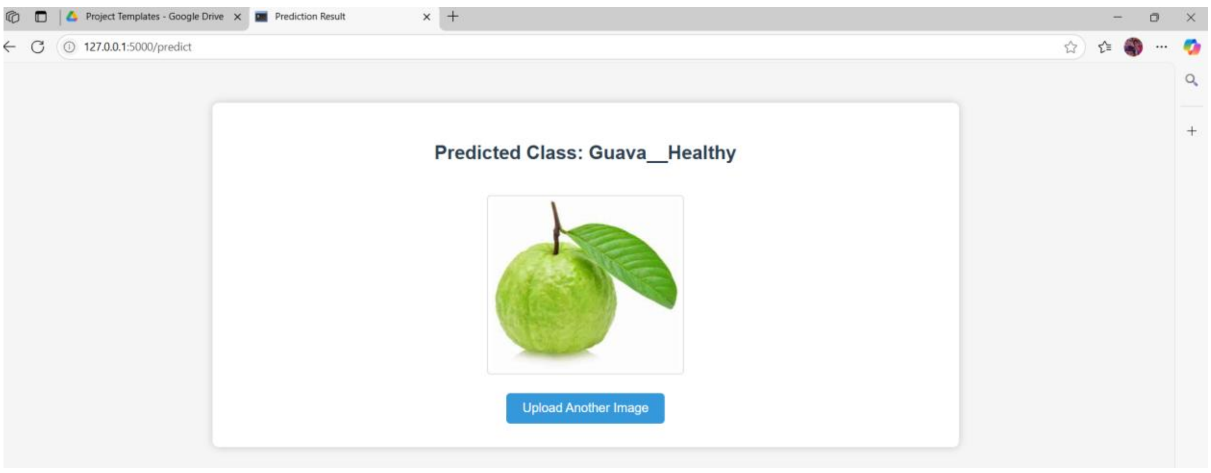
Results of app.py:

Image Upload Form:



Upload an image of a fruit or vegetable

Prediction Result Page:



Predicted Class: Guava__Healthy

Bug Fixes & Improvements:

- Improved model weights for better accuracy
- Cleaned up frontend with custom styling
- Added auto-clean script for uploaded images

Final Validation:

- Verified accuracy against unseen test set.
- Tested end-to-end flow.
- Verified UI responsiveness.

Deployment (if applicable):

- Hosted on local server.

  * Running on http://127.0.0.1:5000
- Future deployment on Render or Heroku.

Additional Sections

Dataset Overview:

- 28 classes: 14 fruits/vegetables × 2 (Healthy/Rotten)
- Balanced split for training/testing
- ~200-300 images per class

Utility Scripts:

- **split_dataset.py:** Divides data into train/test folders
- **visualize_image.py:** Shows single sample from each class
- **class_distribution.py:** Visualizes count of each class
- **visualize_grid.py:** Plots grid of random images

Future Enhancements:

- Expand to include more fruits and vegetables.
- Mobile app version.
- Real-time camera input for live sorting.
- Deploy on cloud with user authentication and tracking.

Project Structure:

```
SmartSorting/

│

├── app.py                    # Main Flask app

├── healthy_vs_rotten.h5        # Trained Keras model

├── requirements.txt           # Required Python packages

├── README.txt                  # Project documentation

│

├── static/                    # Static files

│   └── uploads/                # Uploaded images go here

│

├── templates/                  # HTML templates

│   ├── index.html              # Upload page

│   └── result.html             # Prediction result page

│

├── dataset/                    # (Optional) Dataset used for training

│   ├──Apple__Healthy/

│   ├──Apple__Rotten/

│   └── ... (other classes)

│

├── utils/                      # Utility scripts (optional)

│   ├── split_dataset.py        # Script to split data into train/test

│   ├── visualize_image.py       # Script to visualize single image

│   ├── visualize_grid.py        # Script to display grid of images

│   └── class_distribution.py
```