Smart Sorting: Transfer Learning for Identifying Rotten Fruits and Vegetables

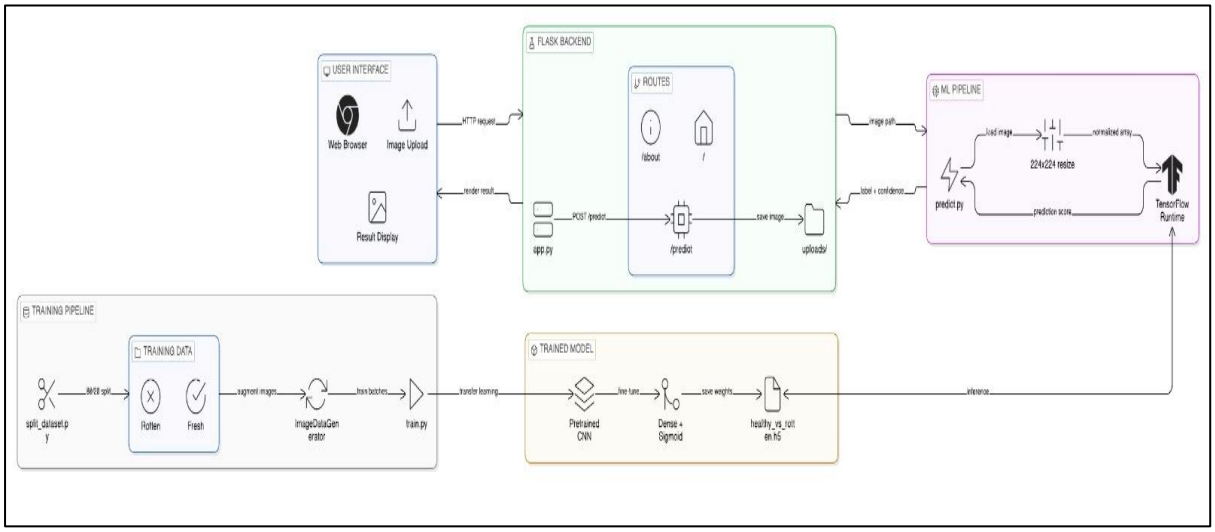
Project Description:

One of the major challenges affecting the agricultural supply chain and food industry is the identification of spoiled fruits and vegetables. A significant portion of food waste occurs due to improper quality inspection and delayed detection of rotten produce. As we know, maintaining food quality is extremely crucial for public health, economic stability, and sustainable resource management. There are various traditional techniques used for sorting and grading fruits and vegetables, but most of them rely on manual inspection, which is time-consuming, inconsistent, and prone to human error.

The detection of rotten fruits and vegetables is a difficult task when performed manually, especially in large-scale markets, warehouses, and supermarkets. Human inspection can be affected by fatigue, lighting conditions, and subjective judgment. By automating the identification process using intelligent systems, industries can reduce food waste, improve quality control, and minimize financial losses. This makes the study of automated fruit quality detection highly important in today's technology-driven environment. Machine Learning and Deep Learning techniques play a crucial role in solving such image-based classification problems effectively.

In this project, **Smart Sorting: Transfer Learning for Identifying Rotten Fruits and Vegetables**, we use deep learning-based image classification techniques to predict whether a fruit or vegetable is fresh or rotten. We implement transfer learning using pre-trained Convolutional Neural Network (CNN) architectures such as EfficientNet/ResNet to improve accuracy while reducing training time. The dataset is trained and tested using appropriate preprocessing and augmentation techniques. The best-performing model is saved in .h5 format and integrated into a Flask web application. Finally, the system is deployed with a user-friendly interface where users can upload an image and receive real-time predictions. This project demonstrates how AI-powered computer vision can be applied to build a smart, automated sorting system for real-world agricultural and retail applications.

Technical Architecture:



Pre requisites:

To complete this project, you must required following software's, concepts and packages

- **VS Code (Visual Studio Code):**
  - Install Visual Studio Code for writing and running the project.
  - Install Python extension in VS Code.
- **Python packages:**
  - Open VS Code Terminal as Administrator and install the following packages:
  - Type "pip install numpy" and click enter.
  - Type "pip install pandas" and click enter.
  - Type "pip install scikit-learn" and click enter.
  - Type "pip install matplotlib" and click enter.
  - Type "pip install scipy" and click enter.
  - Type "pip install tensorflow" and click enter.
  - Type "pip install seaborn" and click enter.
  - Type "pip install keras" and click enter.
  - Type "pip install opencv-python" and click enter.
  - Type "pip install Flask" and click enter.
  - Type "pip install pillow" and click enter.
  - Type "pip install h5py" and click enter.

Prior Knowledge:

You must have prior knowledge of following topics to complete this project.

ML Concepts

- Supervised Learning (Image Classification): https://www.javatpoint.com/supervised-machine-learning
- Convolutional Neural Network (CNN): https://www.javatpoint.com/convolutional-neural-network
- Transfer Learning: https://www.analyticsvidhya.com/blog/2020/04/what-is-transfer-learning-in-deep-learning/
- Image Classification: https://www.javatpoint.com/image-classification-in-machine-learning
- Data Augmentation: https://www.tensorflow.org/tutorials/images/data_augmentation
- Model Evaluation Metrics (Accuracy, Precision, Recall, F1-Score): https://www.analyticsvidhya.com/blog/2019/08/11-important-model-evaluation-error-metrics/
- TensorFlow and Keras Basics: https://www.tensorflow.org/tutorials
- **Flask Basics** : https://www.youtube.com/watch?v=lj4I_CvBnt0
- **Frontend Basics:** https://www.w3schools.com/

Project Objectives:

By the end of this project you will:

- Understand fundamental concepts of Deep Learning and Transfer Learning.

- Gain practical knowledge of image-based classification systems.
- Learn how to preprocess image datasets for training CNN models.
- Understand how pre-trained models improve performance and reduce training time.
- Build and integrate a deep learning model with a Flask web application.
- Develop a professional and user-friendly web interface for real-time prediction.

Project Flow:

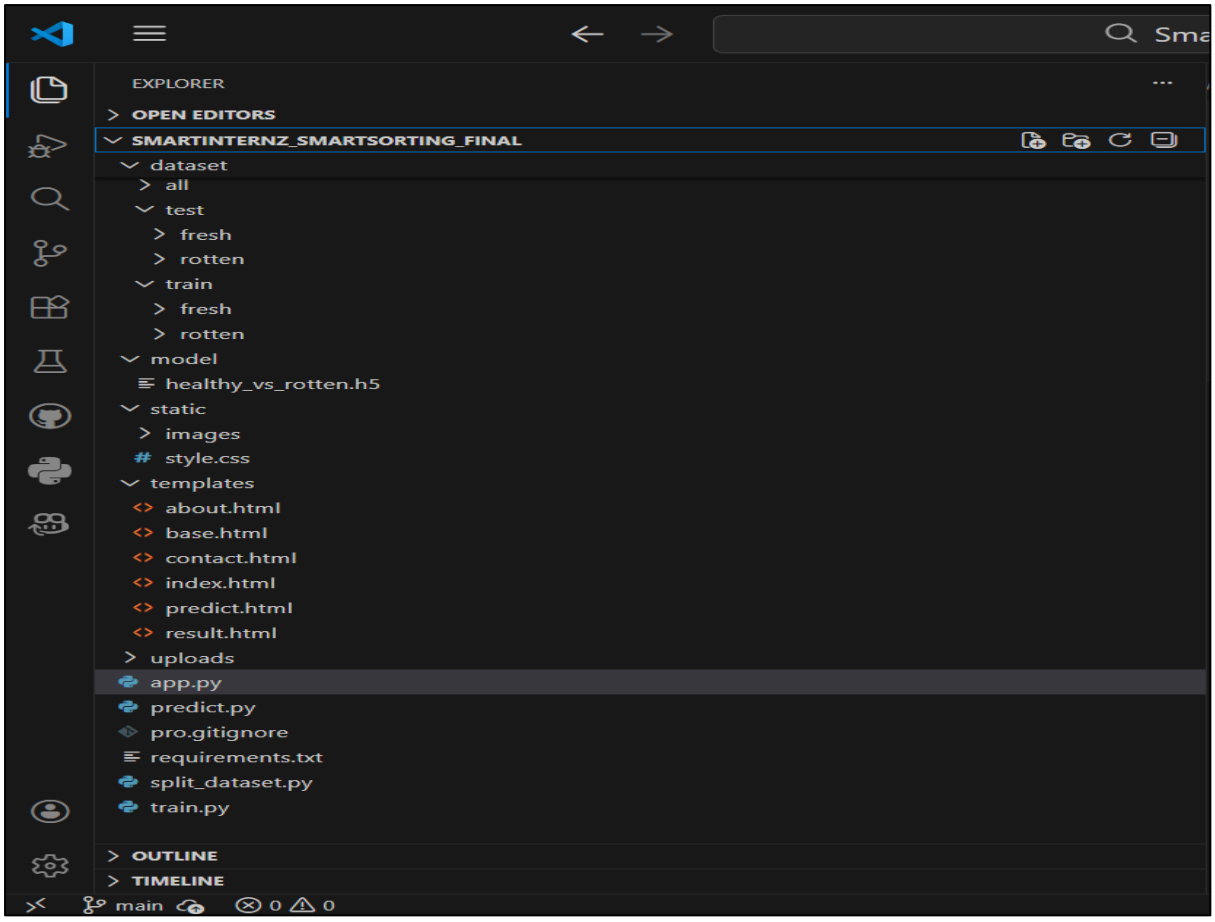
- User uploads an image of a fruit or vegetable through the web interface.
- The uploaded image is preprocessed (resized, normalized).
- The trained transfer learning model analyzes the image.
- The model predicts whether the fruit/vegetable is **Fresh or Rotten**.
- The prediction result along with confidence score is displayed on the UI.

To accomplish this, we have to complete all the activities listed below,

- Data collection
  - Collect dataset from public sources (Kaggle / open datasets)
  - Organize images into class folders (Fresh / Rotten for each fruit/vegetable)
- Visualizing and analyzing data
  - Checking number of images per class
  - Visual inspection of sample images
  - Class distribution analysis
  - Identifying imbalance in dataset
- Data pre-processing
  - Image resizing (e.g., 224x224)
  - Image normalization (scaling pixel values)
  - Data augmentation (rotation, flipping, zoom, brightness adjustment)
  - Splitting dataset into train, validation, and test sets
- Model building
  - Import deep learning libraries (TensorFlow / Keras)
  - Load pre-trained model (e.g., MobileNetV2 / EfficientNet)
  - Freeze base layers (feature extraction)
  - Add custom dense layers
  - Compile the model (optimizer, loss, metrics)
  - Train the model
  - Evaluate model performance (accuracy, confusion matrix)
  - Save the trained model in .h5 format
- Application Building
  - Create HTML files (Home, About, Predict, Contact pages)
  - Design professional UI using CSS
  - Write Flask backend code
  - Load trained model in Flask
  - Handle image upload and prediction
  - Display result with confidence score
  - Test application locally

Project Structure:

Create the Project folder which contains files as shown below



Flask Application

- app.py is the main Flask script that:
- Defines routes (/, /about, /predict, /contact)
- Handles image file uploads
- Calls the prediction function
- Renders HTML templates
- Displays prediction results with confidence score
- This file acts as the backbone of the web application.

Model Integration

- The Transfer Learning CNN model is trained using train.py.
- The trained model is saved as:
- model/healthy_vs_rotten.h5
- During inference:
- predict.py loads the .h5 model file.
- It defines a predict_image() function.
- This function is imported and used inside app.py to generate predictions.

Dataset Management

- The dataset/ folder stores all fruit and vegetable images.
- Images are categorized into **Fresh** and **Rotten** classes.
- split_dataset.py organizes the dataset into:
- Training set
- Testing set
- Structure example:

- dataset/
- ├── train/
- │ ├── fresh/
- │ └── rotten/
- └── test/
- ├── fresh/
- └── rotten/
- This structure is required for proper model training and evaluation.

User Interface

- All HTML pages are stored inside the templates/ folder.
- base.html is the common layout template shared by all pages.
- Other pages extend base.html for consistent design.
- The static/ folder contains:
- style.css → Professional dark-themed UI
- Animations and hover effects
- Responsive layout design
- Background images and icons
- This ensures a clean, modern, and interactive interface.

Runtime Uploads

- The `uploads/` folder temporarily stores images uploaded by users.
- Uploaded images are:
  - Used for prediction
  - Displayed on the Result page
- These files are **not part of the training dataset**.
- The folder can be cleared periodically to save storage.

Milestone 1: Data Collection

Machine Learning and Deep Learning models depend heavily on data. Data is the most crucial component that enables model training and accurate prediction. In image classification problems, having a well-structured and properly labeled dataset is essential for achieving high accuracy. This section allows you to download and organize the required dataset for training the Smart Sorting model.

Activity 1: Download the dataset

There are many popular open sources for collecting image datasets. Examples:

- kaggle.com
- UCI Repository
- Google Dataset Search

In this project, we use a **Fruits and Vegetables Fresh vs Rotten Image Dataset**. This dataset contains images categorized into:

- Fresh Fruits
- Rotten Fruits
- Fresh Vegetables

- Rotten Vegetables

The dataset is downloaded from Kaggle.

Please refer to the link given below to download the dataset:

Link:https://www.kaggle.com/datasets/sriramr/fruits-fresh-and-rotten-for-classification

Activity 2: Organize the Dataset

After downloading:

●Extract the dataset zip file.
● Create a folder named dataset inside your project directory.
● Organize the images into subfolders like:

```
dataset/
  ├── fresh/
  └── rotten/
```

Or (if **multiple classes are available):**

```
dataset/
  ├── train/
  │     ├── freshapples/
  │     ├── rottenapples/
  │     ├── freshbanana/
  │     └── rottenbanana/
  └── test/
        ├── freshapples/
        ├── rottenapples/
        ├── freshbanana/
        └── rottenbanana/
```

Proper folder structuring is necessary for training the CNN model using TensorFlow/Keras.

Milestone 2: Visualizing and analysing the data

After collecting and organizing the dataset, the next important step is to understand the data before training the model. In image classification problems, visualization helps us verify class distribution, detect imbalance, and understand image characteristics such as size, quality, and variations.

In this project, we are using a Fruits and Vegetables Fresh vs Rotten dataset downloaded from Kaggle.

Activity 1: Importing Required Libraries

To analyze and visualize the dataset, we import the necessary Python libraries:

- os – For directory handling

- matplotlib – For plotting images and graphs
- seaborn – For advanced visualization
- numpy – For numerical operations
- tensorflow.keras.preprocessing.image – For loading and preprocessing images

Example:

```
import os

import matplotlib.pyplot as plt

import seaborn as sns

import numpy as np

from tensorflow.keras.preprocessing import image
```

These libraries help in reading image data, plotting sample images, and analyzing class distribution.

Activity 2: Understanding Dataset Structure

The dataset is organized into folders like:

```
dataset/
├── train/
│   ├── fresh/
│   └── rotten/
└── test/
    ├── fresh/
    └── rotten/
```

Each subfolder represents a class label. TensorFlow automatically assigns labels based on folder names.

We first check:

- Total number of classes
- Number of images in each class
- Total dataset size

Example:

```
train_path = "dataset/train"
classes = os.listdir(train_path)
for cls in classes:
    print(cls, ":", len(os.listdir(os.path.join(train_path, cls))))
```

Observation:

- Helps identify if the dataset is balanced or imbalanced.
- Ensures that images are properly categorized into fresh and rotten.

Activity 3: Visualizing Sample Images

Before training, it is important to visually inspect sample images from each class.

```
plt.figure(figsize=(10,6))
for i, cls in enumerate(classes):
    img_path = os.path.join(train_path, cls, os.listdir(os.path.join(train_path, cls))[0])
    img = image.load_img(img_path)
    plt.subplot(1,2,i+1)
    plt.imshow(img)
    plt.title(cls)
    plt.axis("off")
    plt.show()
```

From Visualization We Can Observe:

- Color difference between fresh and rotten items
- Texture variations
- Background variations
- Image clarity

This helps us understand what visual features the CNN model will learn.

Activity 4: Class Distribution Analysis (Univariate Analysis)

Univariate analysis means analyzing a single feature — here, the class label (fresh or rotten).

We count the number of images in each category and plot a count graph.

```
import pandas as pd

data = {

   "fresh": len(os.listdir("dataset/train/fresh")),

   "rotten": len(os.listdir("dataset/train/rotten"))

}

sns.barplot(x=list(data.keys()), y=list(data.values()))

plt.title("Class Distribution")

plt.ylabel("Number of Images")

plt.show()
```

Inference:

- If both classes have nearly equal counts → dataset is balanced.
- If one class has significantly more images → dataset is imbalanced.
- Balanced datasets improve model accuracy and reduce bias.

Activity 5: Checking Image Dimensions

Different image sizes can affect training. We check whether images have uniform dimensions.

```
img = image.load_img(img_path)

print(img.size)
```

Observation:

- Most images may have different sizes.
- Therefore, resizing to 224×224 is necessary (as used in your model).

In this project, all images are resized to:(224, 224)

This matches the input size required for MobileNetV2 used in the Smart Sorting model.

Activity 6: Data Augmentation Visualization

Since real-world images vary in lighting and orientation, we use data augmentation during training.

Using ImageDataGenerator:

```
from tensorflow.keras.preprocessing.image import ImageDataGenerator
```

datagen = ImageDataGenerator(rotation_range=20,zoom_range=0.2,horizontal_flip=True)

We can visualize augmented images to ensure transformations are realistic.

Benefits:

- Prevents overfitting
- Improves model generalization
- Makes the model robust to real-world variations

Activity 7: Summary of Data Analysis

After completing visualization and analysis, we conclude:

1. Dataset contains two major classes:
    - Fresh
    - Rotten
2. Images vary in:
    - Background
    - Lighting
    - Texture
    - Shape
3. Image resizing to 224×224 is required.
4. Data augmentation improves performance and reduces overfitting.
5. Dataset is suitable for training a deep learning CNN model.

Milestone 3: Data Pre-processing

After analyzing the dataset in Milestone 2, the next important step is Data Pre-processing. Raw image data cannot be directly fed into a deep learning model. It must be cleaned, structured, resized, normalized, and prepared properly.

In this project, we use TensorFlow and Keras for preprocessing and training.

Why Data Pre-processing is Important?

Deep Learning models:

- Require uniform input size
- Perform better with normalized pixel values

- Need balanced and well-structured datasets
- Generalize better with augmented data

Without preprocessing, the model may:

- Overfit
- Perform poorly
- Learn irrelevant features

Activity 1: Dataset Organization

After downloading from Kaggle, the dataset is structured as:

dataset/

```
├── train/
│   ├── fresh/
│   └── rotten/
└── test/
    ├── fresh/
    └── rotten/
```

This folder structure is mandatory because flow_from_directory() automatically assigns labels based on folder names.

Activity 2: Splitting the Dataset (Train & Test)

If the dataset is not already split, we divide it into training and testing sets.

In the project, you created a script that:

- Reads all images from dataset/all
- Randomly shuffles images
- Splits them using 80:20 ratio
- Copies them into:
    - dataset/train
    - dataset/test

Example logic used:

split_ratio = 0.8

split_index = int(len(images) * split_ratio)

Why Splitting is Required?

- Training Set → Used to train the model
- Test Set → Used to evaluate model performance
- Prevents data leakage
- Ensures model generalization

Activity 3: Image Resizing

Images in the dataset may have different sizes.
But CNN models require fixed input dimensions.

In this project, images are resized to:

224 × 224 pixels

This size matches the requirement of MobileNetV2.

In the code:

IMG_SIZE = 224

target_size=(IMG_SIZE, IMG_SIZE)

Why 224×224?

- Pretrained ImageNet models expect 224×224 input
- Ensures compatibility with transfer learning
- Reduces computational cost

Activity 4: Pixel Value Normalization

Original pixel values range from:0 to 255

Deep learning models perform better when input values are scaled between:0 and 1

In the project, normalization is done using:rescale=1./255

And during prediction: img_array = image.img_to_array(img) / 255.0

Why Normalization?

- Speeds up convergence

- Prevents gradient explosion
- Improves numerical stability

Activity 5: Data Augmentation

Real-world images vary in:

- Lighting
- Orientation
- Zoom
- Position

To make the model robust, we apply augmentation using:

```
ImageDataGenerator(
   rescale=1./255,
   rotation_range=20,
   zoom_range=0.2,
   horizontal_flip=True
)
```

Transformations Used:

Technique	Purpose
Rotation	Handle different angles
Zoom	Detect close/far objects
Horizontal Flip	Improve generalization
Rescaling	Normalize pixel values

Why Augmentation?

- Prevents overfitting
- Increases dataset diversity
- Improves accuracy

Activity 6: Handling Class Labels

Since this is a binary classification problem:

- fresh → 0
- rotten → 1

In the code:

class_mode="binary"

This ensures:

- Output layer has 1 neuron
- Sigmoid activation is used
- Binary cross-entropy loss is applied

Activity 7: Creating Data Generators

Instead of loading all images into memory, we use generators:

train_gen = train_datagen.flow_from_directory(...)

val_gen = test_datagen.flow_from_directory(...)

Advantages of Generators:

- Memory efficient
- Automatically labels data
- Performs real-time augmentation
- Feeds data in batches

Batch size used in your project:

BATCH = 32

Activity 8: Preventing Overfitting

To avoid overfitting:

1. Data augmentation is used
2. Dropout layer added in model:Dropout(0.5)

This randomly drops 50% neurons during training to improve generalization.

Milestone 4: Model Building

After completing data collection, analysis, and preprocessing, the next step is Model Building. In this milestone, we design, train, evaluate, and save a Deep Learning model that can classify images as Fresh or Rotten.

In this project, we use Transfer Learning with MobileNetV2 implemented using TensorFlow and Keras.

Why Transfer Learning?

Training a CNN from scratch:

- Requires large dataset
- Takes long training time
- Needs high computational power

Instead, we use a pretrained model (trained on ImageNet dataset) and customize it for our Smart Sorting problem.

Advantages:

- Faster training
- Better accuracy
- Less data required
- Improved feature extraction

Activity 1: Import Required Libraries

import tensorflow as tf

from tensorflow.keras.applications import MobileNetV2

from tensorflow.keras.layers import Dense, Flatten, Dropout

from tensorflow.keras.models import Model

These libraries help in:

- Loading pretrained models
- Creating custom layers
- Building and compiling CNN

Activity 2: Load Pretrained Base Model

```
base_model = MobileNetV2(
    weights="imagenet",
```

```
    include_top=False,
    input_shape=(224,224,3)
)
```

Parameters Explanation:

Parameter	Purpose
weights="imagenet"	Uses pretrained ImageNet weights
include_top=False	Removes original classifier
input_shape=(224,224,3)	Input image size

Freezing Base Model

```
base_model.trainable = False
```

This prevents pretrained layers from updating during training.

Activity 3: Adding Custom Classification Layers

After loading the base model, we add our own classification layers:

```
x = Flatten()(base_model.output)
x = Dense(128, activation="relu")(x)
x = Dropout(0.5)(x)
output = Dense(1, activation="sigmoid")(x)
```

Layer Explanation:

1. Flatten Layer

Converts 2D feature maps into 1D vector.

2. Dense Layer (128 neurons, ReLU)

- Learns important features
- Adds non-linearity

3. Dropout (0.5)

- Randomly disables 50% neurons
- Prevents overfitting

4. Output Layer

- 1 neuron
- Sigmoid activation
- Used for binary classification

Activity 4: Creating Final Model

model = Model(inputs=base_model.input, outputs=output)

Now the complete CNN model is ready.

Activity 5: Compiling the Model

model.compile(optimizer="adam",loss="binary_crossentropy",metrics=["accuracy"])

Explanation:

Component	Purpose
Adam Optimizer	Efficient gradient descent
Binary Crossentropy	Used for 2-class problem
Accuracy	Evaluation metric

Activity 6: Training the Model

model.fit(train_gen,validation_data=val_gen,epochs=10)

Parameters:

Parameter	Description
train_gen	Training dataset
validation_data	Test dataset
epochs=10	Number of training cycles

During training:

- Model learns features from fresh and rotten images
- Accuracy and loss are monitored
- Validation performance is checked

Activity 7: Model Evaluation

After training, we evaluate performance:

- Training Accuracy
- Validation Accuracy
- Loss values

In this project:

- Achieved ~94% accuracy (as shown in about page)
- Good validation performance

If validation accuracy is close to training accuracy:

- Model is well generalized
- No overfitting

Activity 8: Saving the Model

After successful training:

model.save("model/healthy_vs_rotten.h5")

This saves:

- Model architecture
- Weights
- Optimizer state

The saved model is later loaded in the Flask app:

model = tf.keras.models.load_model("model/healthy_vs_rotten.h5")

Activity 9: Prediction Logic

In the predict.py:

```
pred = model.predict(img_array)[0][0]
if pred > 0.5:
    label = "Rotten"
else:
    label = "Fresh"
```

Why 0.5 Threshold?

Since sigmoid output ranges between 0 and 1:

- 0.5 → Rotten
- < 0.5 → Fresh

Confidence is calculated and displayed in UI.

Model Architecture Summary

Input (224×224×3 Image)
↓
MobileNetV2 (Feature Extraction)
↓
Flatten
↓
Dense (128, ReLU)
↓
Dropout (0.5)
↓
Dense (1, Sigmoid)
↓
Prediction (Fresh / Rotten)

Milestone 5: Application Building

After successfully training and saving the CNN model, the next step is to deploy it using a web application. In this milestone, we integrate the trained deep learning model with a web interface so users can upload fruit or vegetable images and receive predictions instantly.

In this project, the web application is built using **Flask** and the trained model is developed using **TensorFlow**.

Objective of Application Building

The goal of this phase is to:

- Create a user-friendly interface
- Allow users to upload an image
- Send the image to the trained model
- Display prediction results (Fresh / Rotten)
- Show confidence percentage
- Display uploaded image on result page

Application Architecture Overview

User (Browser)
↓
Flask Web Server
↓
Load Saved CNN Model (.h5 file)
↓
Process Uploaded Image

↓
Generate Prediction
↓
Display Result on Webpage

Activity 1: Building HTML Pages

All HTML files are stored inside the templates/ folder because Flask automatically looks for templates there.

You created the following pages:

- index.html → Home page
- predict.html → Image upload page
- result.html → Prediction output page
- about.html → Project details
- contact.html → Contact information

1. Home Page (index.html)

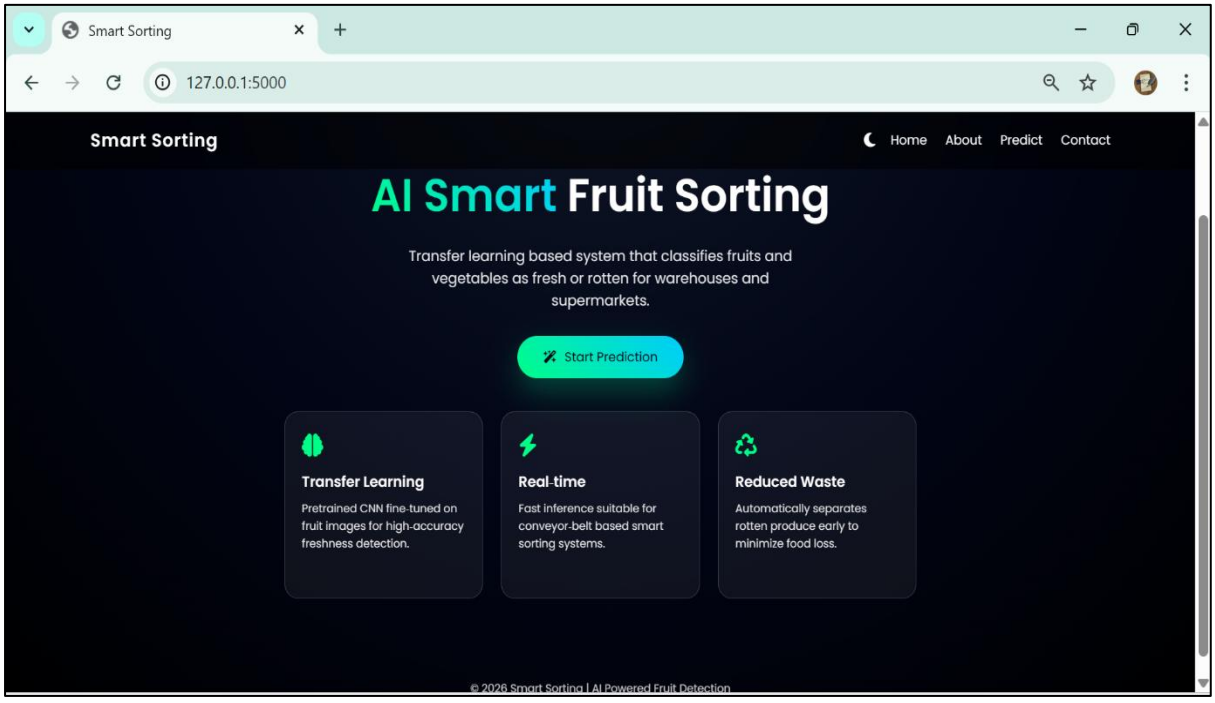
Purpose:

- Introduces the Smart Sorting project
- Contains navigation menu
- Redirects to prediction page

Route in Flask:

```
@app.route("/")
def home():
    return render_template("index.html")
```

When the user opens http://127.0.0.1:5000/, this page is displayed.



2. Predict Page (predict.html)

Purpose:

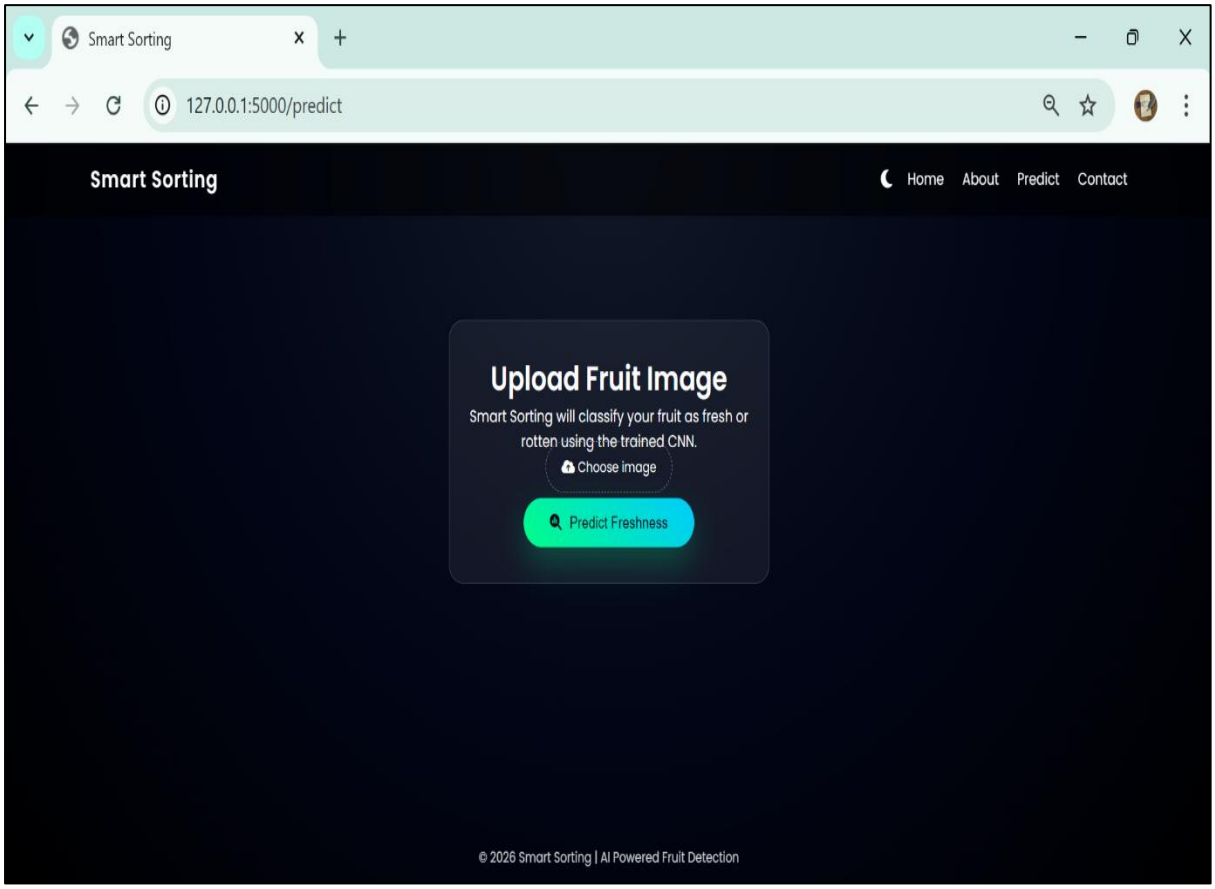
- Allows user to upload an image
- Uses HTML form with POST method
- Sends image file to Flask backend

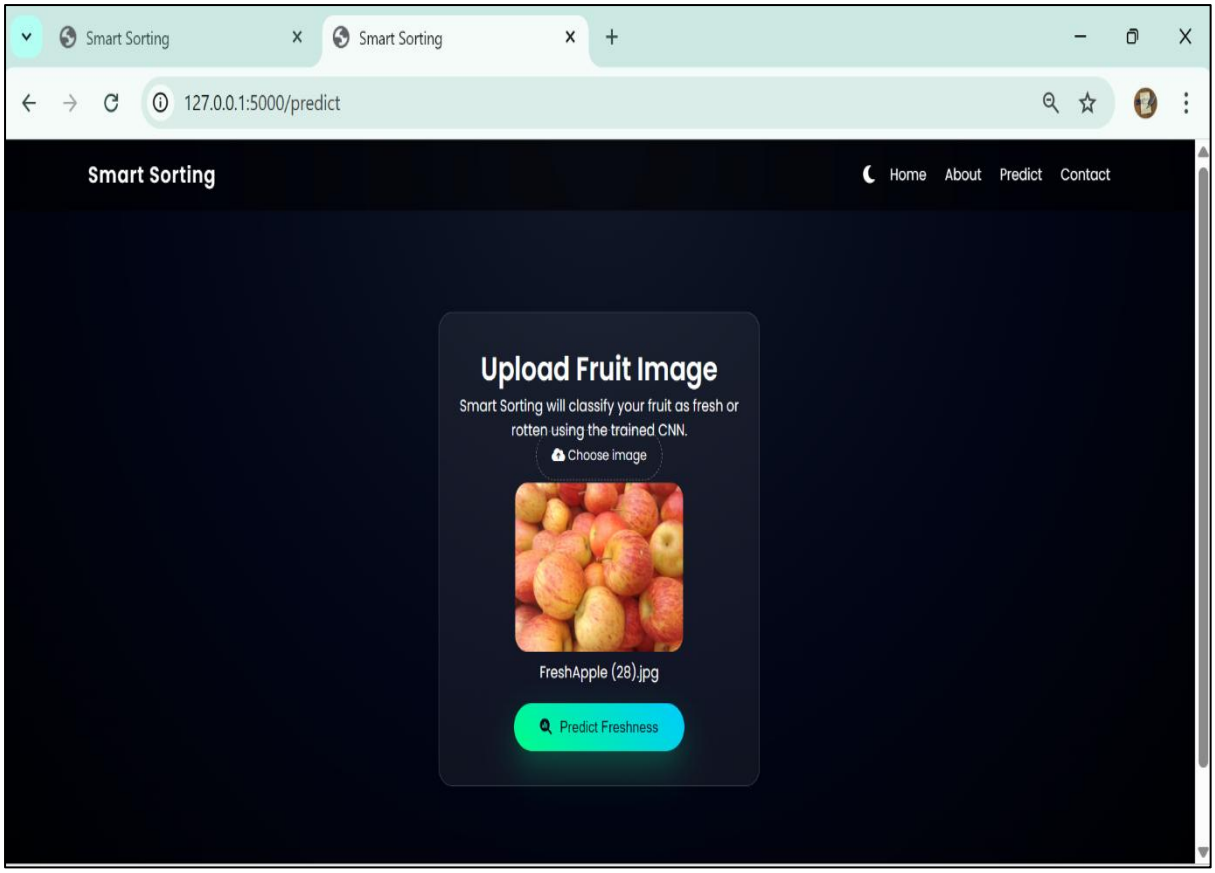
Important Form Elements:

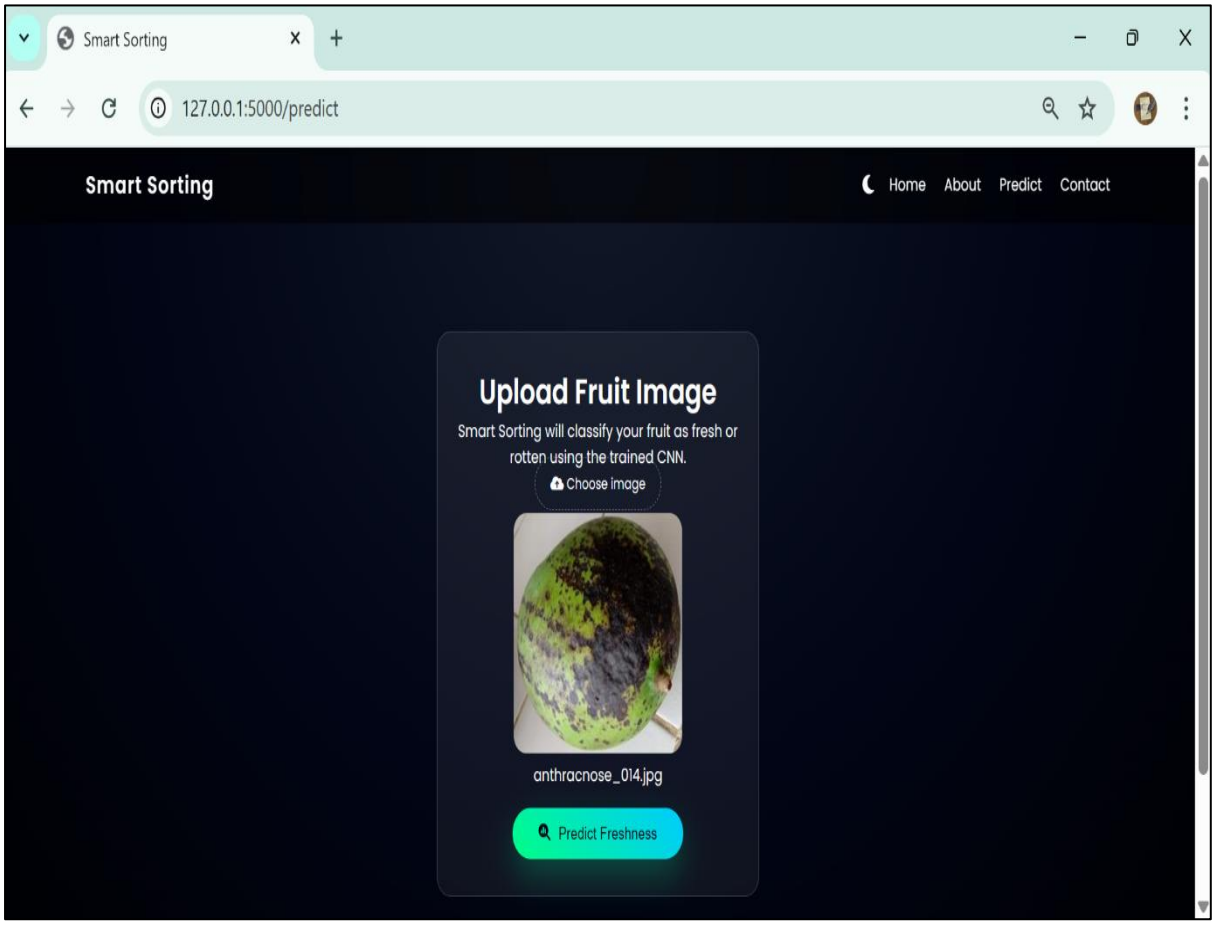
```
<form action="/predict" method="POST" enctype="multipart/form-data">
  <input type="file" name="file">
  <button type="submit">Predict</button>
</form>
```

Key Points:

- method="POST" → Sends data securely
- enctype="multipart/form-data" → Required for file upload
- name="file" → Used to retrieve image in Flask




Smart Sorting
Smart Sorting
127.0.0.1:5000/predict
Smart Sorting
Home
About
Predict
Contact
Upload Fruit Image
Smart Sorting will classify your fruit as fresh or rotten using the trained CNN.
Choose image
FreshApple (28).jpg
Predict Freshness


Smart Sorting
127.0.0.1:5000/predict
Smart Sorting
Home
About
Predict
Contact
Upload Fruit Image
Smart Sorting will classify your fruit as fresh or rotten using the trained CNN.
Choose image
anthracnose_014.jpg
Predict Freshness

3. Result Page (result.html)

Purpose:

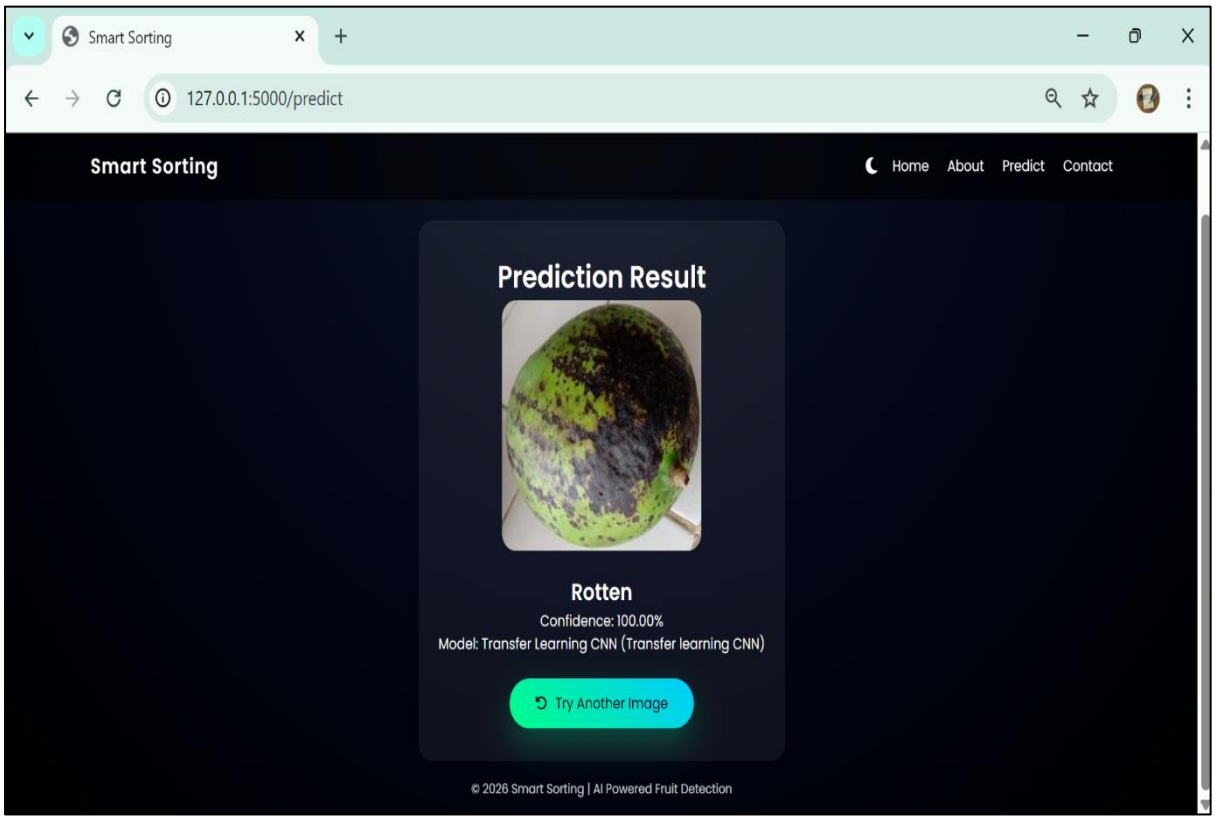
- Displays:
    - Uploaded image
    - Prediction label
    - Confidence score
    - Model name

Example variables passed:

```
return render_template(
    "result.html",
    prediction_label=label,
    confidence=f"{confidence:.2f}",
    model_name="Transfer Learning CNN",
    image_path=image_url,
)
```

On HTML page:

```
<img src="{{ image_path }}">
<h2>Prediction: {{ prediction_label }}</h2>
<p>Confidence: {{ confidence }}%</p>
```



4.About Page (about.html)

Purpose:

• Displays project information
• Shows model accuracy
• Displays number of classes
• Displays dataset size
• Describes model used (Transfer Learning CNN)

This page gives technical details about the Smart Sorting system.

Route in Flask:

```
@app.route("/about")
def about():
    accuracy = 94.6
    classes = 4
    dataset_size = 3200
    return
render_template("about.html",accuracy=accuracy,classes=classes,dataset_size=dataset_size,)
```

When the user clicks the **About** link in the navigation bar, this page is displayed.

Variables Passed to HTML:

• accuracy → Model accuracy percentage
• classes → Number of categories
• dataset_size → Total number of images

On HTML Page:

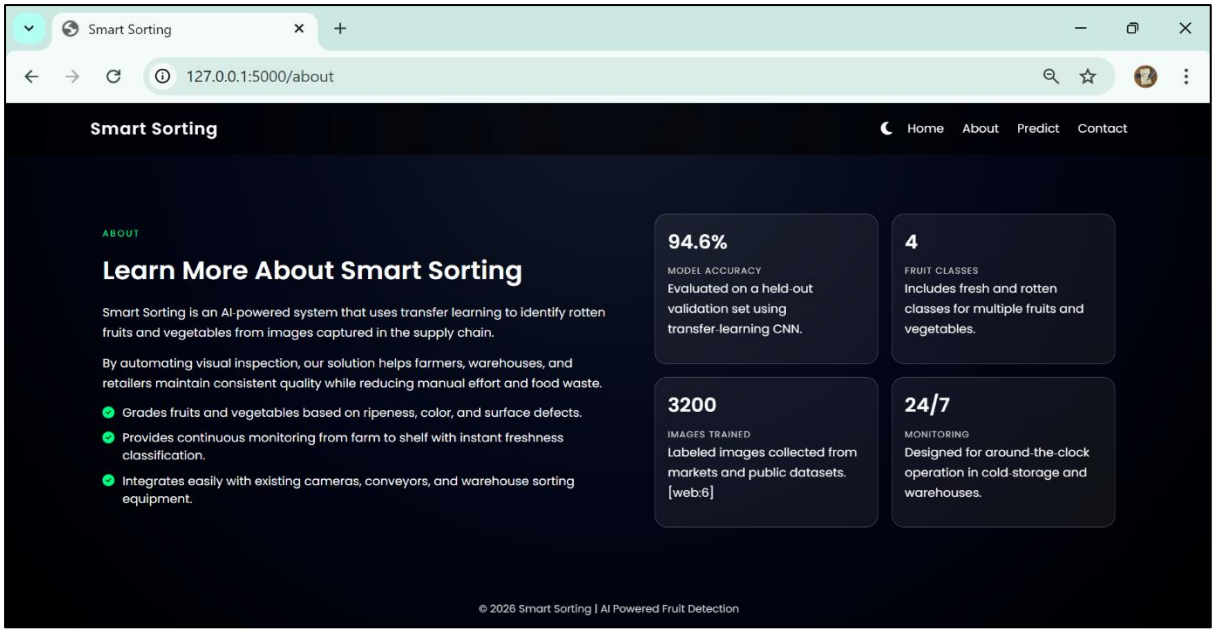
Example usage:

```
<h2>Model Accuracy: {{ accuracy }}%</h2>
<p>Total Classes: {{ classes }}</p>
<p>Dataset Size: {{ dataset_size }} images</p>
<p>Model Used: Transfer Learning CNN (MobileNetV2)</p>
```

What This Page Demonstrates:

• Model performance
• Dataset information
• Technical transparency
• Project credibility

This page is important for documentation because it shows the evaluation details of the trained CNN model.



5. Contact Page (contact.html)

Purpose:

• Displays developer details
• Shows contact information
• Provides communication details
• Makes the application look complete and professional

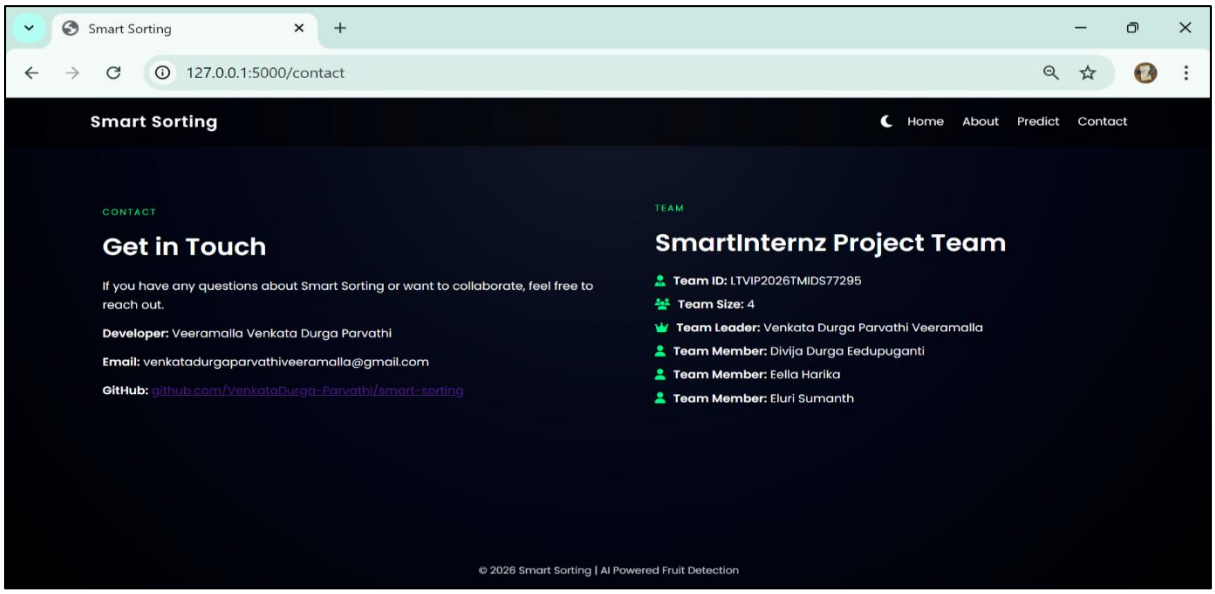
Route in Flask:

@app.route("/contact")

def contact():

    return render_template("contact.html")

When the user clicks **Contact** in the navigation menu, this page is displayed.



Activity 2: Building Server-Side Script (app.py)

Now we connect frontend with backend using Flask.

Step 1: Import Required Libraries

from flask import Flask, render_template, request, send_from_directory, url_for

import os

from predict import predict_image

Explanation:

Library	Purpose
Flask	Web framework
render_template	Load HTML files
request	Handle form data
send_from_directory	Serve uploaded images
url_for	Generate URLs
os	File handling

Step 2: Initialize Flask Application

app = Flask(__name__)

This creates a WSGI application instance.

Step 3: Configure Upload Folder

UPLOAD_FOLDER = os.path.join(app.root_path, "uploads")

app.config["UPLOAD_FOLDER"] = UPLOAD_FOLDER

os.makedirs(UPLOAD_FOLDER, exist_ok=True)

Purpose:

- Store uploaded images temporarily
- Ensure folder exists

Step 4: Prediction Route

@app.route("/predict", methods=["GET", "POST"])

def predict():

Handling POST Request

file = request.files.get("file")

If no file:

return render_template("predict.html", error="Please upload an image.")

Save Image

filepath = os.path.join(app.config["UPLOAD_FOLDER"], file.filename)

file.save(filepath)

Call Model Prediction

label, confidence = predict_image(filepath)

This calls your predict.py file where:

model = tf.keras.models.load_model("model/healthy_vs_rotten.h5")

The model processes the image and returns:

- Fresh or Rotten
- Confidence percentage

Step 5: Display Uploaded Image

To display the uploaded image:

image_url = url_for("uploaded_file", filename=file.filename)

Route to serve image:

@app.route("/uploads/<path:filename>")

def uploaded_file(filename):

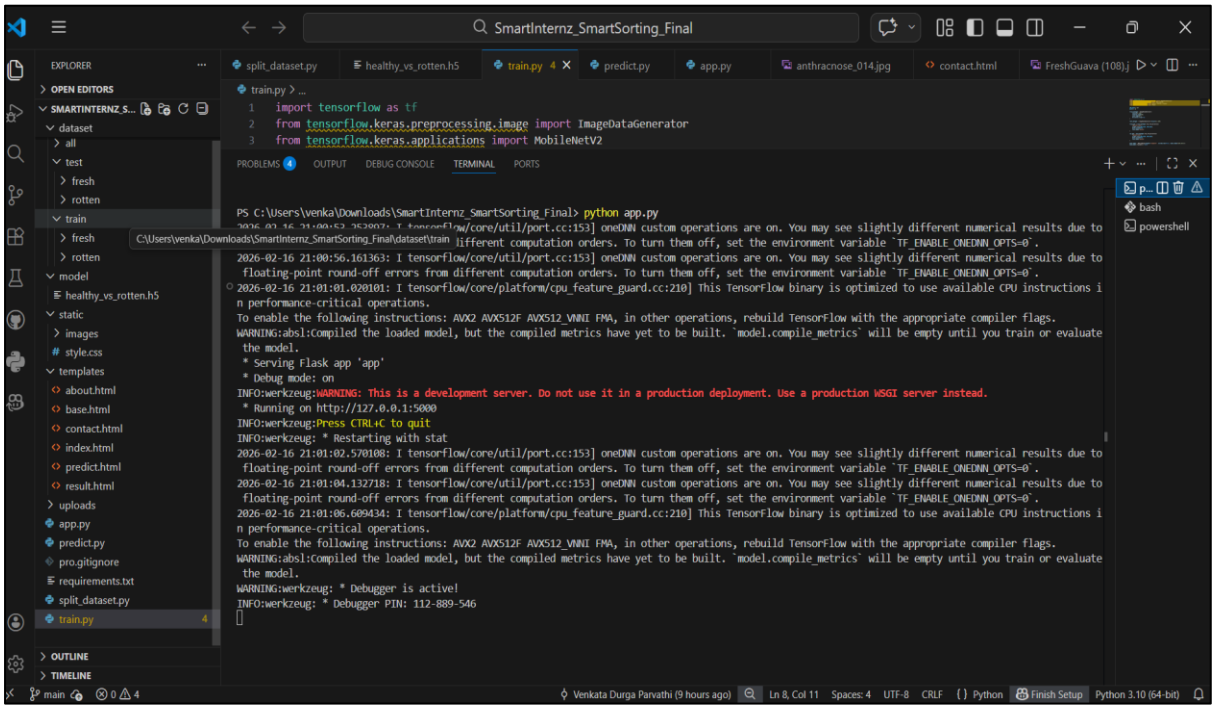
return send_from_directory(app.config["UPLOAD_FOLDER"], filename)

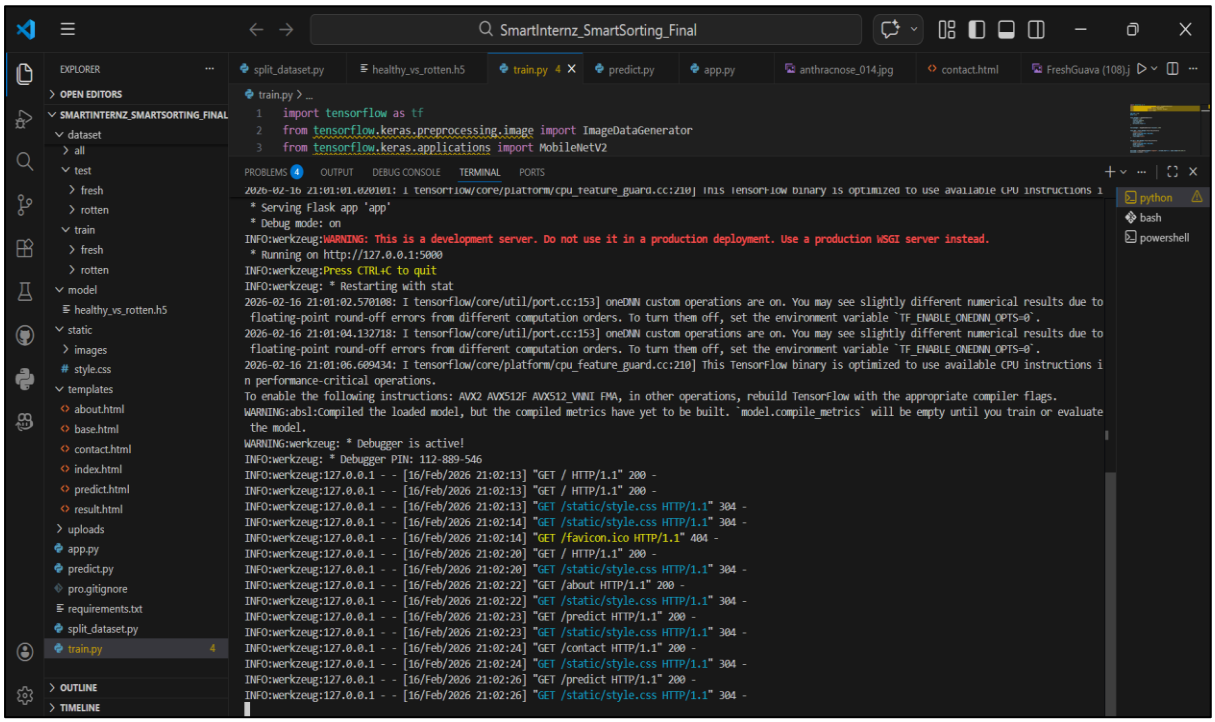
Activity 3: Running the Application

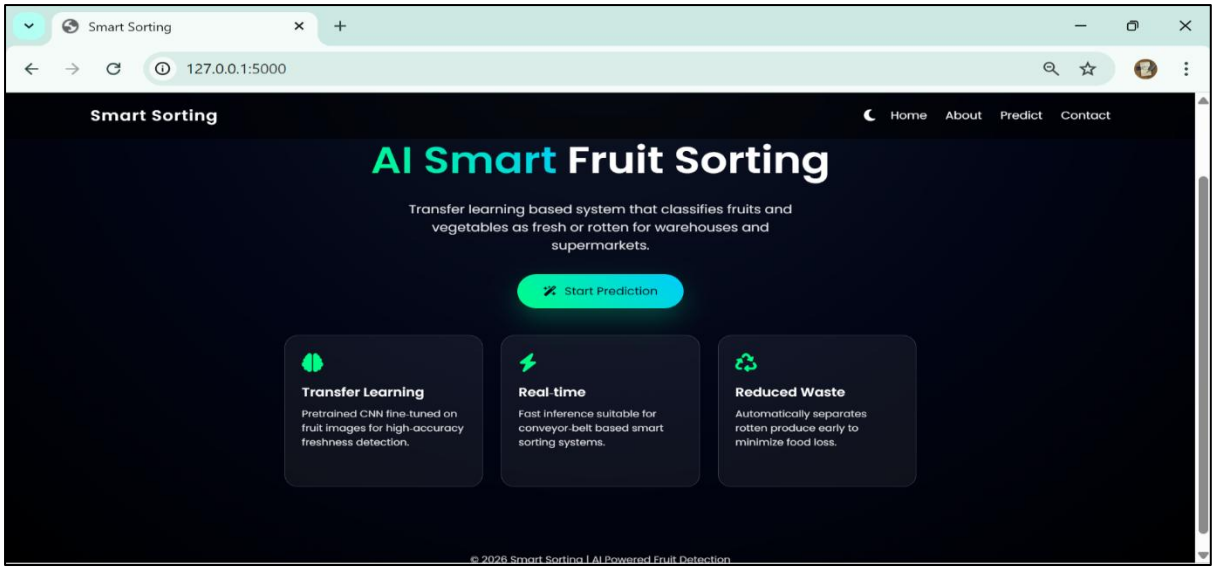
To run the project:

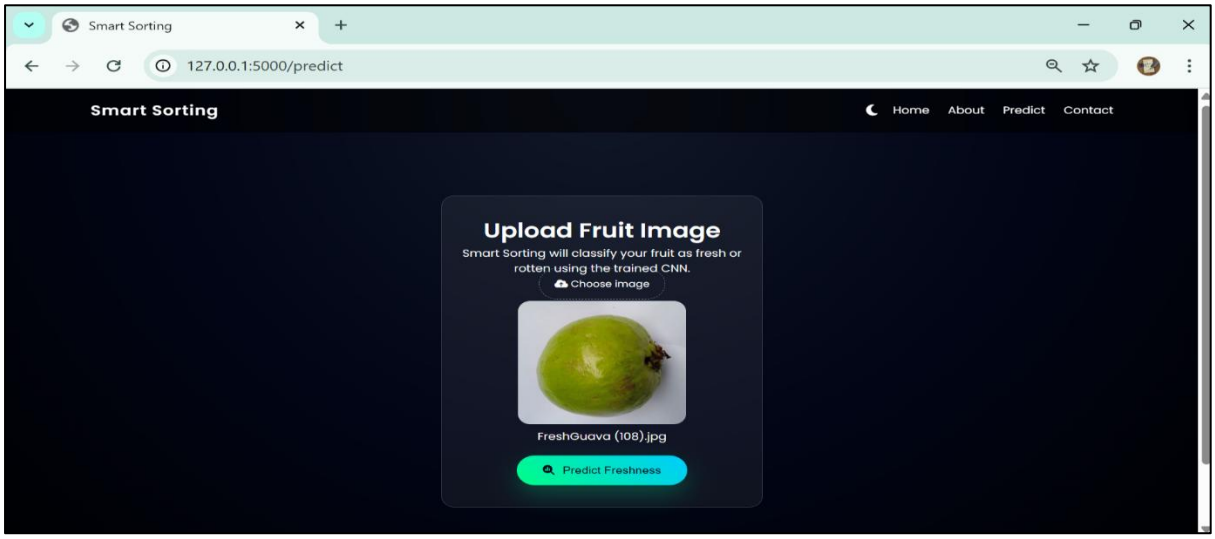
1. Open Command Prompt
2. Navigate to project directory
3. Run: python app.py

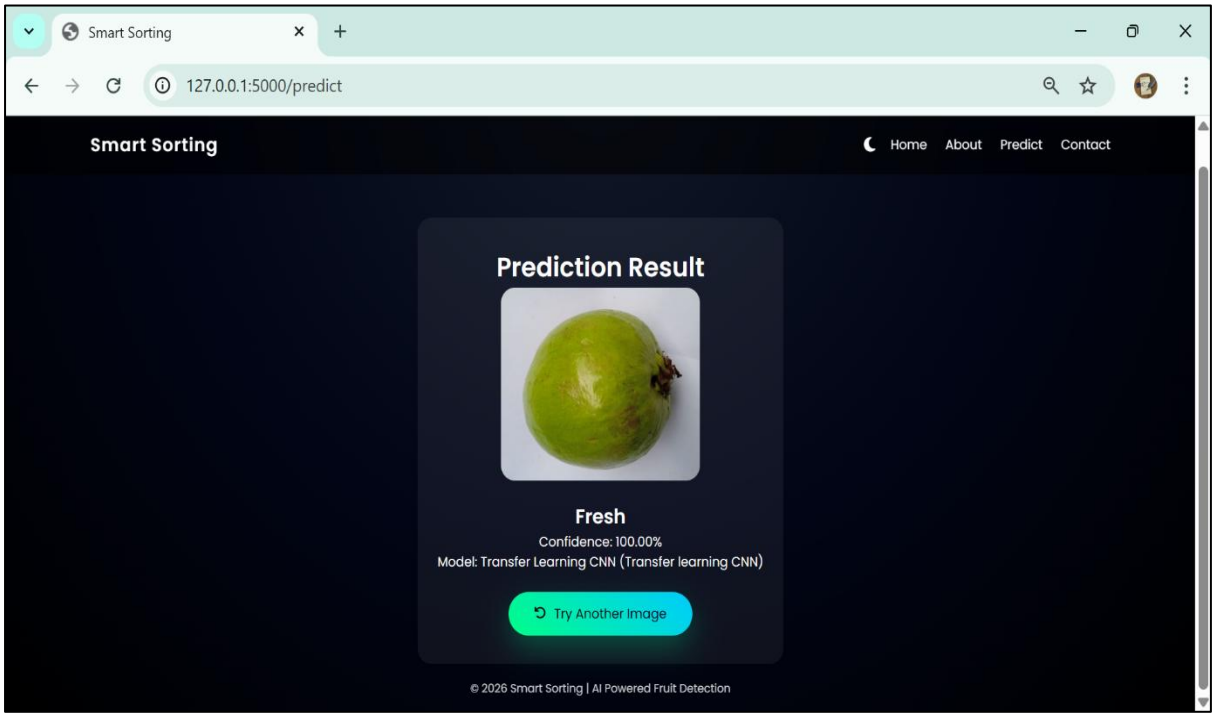
   Output:Running on http://127.0.0.1:5000/
4. Open browser
5. Go to: http://127.0.0.1:5000/
6. Upload image
7. Click Predict
8. View result


SmartInternz_SmartSorting_Final
EXPLORER
OPEN EDITORS
SMARTINTERNZ_S...
dataset
all
test
fresh
rotten
train
fresh
rotten
model
healthy_vs_rotten.h5
static
images
style.css
templates
about.html
base.html
contact.html
index.html
predict.html
result.html
uploads
app.py
predict.py
pro.gitignore
requirements.txt
split_dataset.py
train.py
OUTLINE
TIMELINE
split_dataset.py
healthy_vs_rotten.h5
train.py 4
predict.py
app.py
anthracnose_014.jpg
contact.html
FreshGuava (108).j
train.py > ...
1 import tensorflow as tf
2 from tensorflow.keras.preprocessing.image import ImageDataGenerator
3 from tensorflow.keras.applications import MobileNetV2
PROBLEMS 4
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
PS C:\Users\venka\Downloads\SmartInternz_SmartSorting_Final> python app.py
C:\Users\venka\Downloads\SmartInternz_SmartSorting_Final\dataset\train
different computation orders. To turn them off, set the environment variable `TF_ENABLE_ONEDNN_OPTS=0`.
2026-02-16 21:00:56.161363: I tensorflow/core/util/port.cc:153] oneDNN custom operations are on. You may see slightly different numerical results due to floating-point round-off errors from different computation orders. To turn them off, set the environment variable `TF_ENABLE_ONEDNN_OPTS=0`.
2026-02-16 21:01:01.020101: I tensorflow/core/platform/cpu_feature_guard.cc:210] This TensorFlow binary is optimized to use available CPU instructions in performance-critical operations.
To enable the following instructions: AVX2 AVX512F AVX512_VNNI FMA, in other operations, rebuild TensorFlow with the appropriate compiler flags.
WARNING:absl:Compiled the loaded model, but the compiled metrics have yet to be built. `model.compile_metrics` will be empty until you train or evaluate the model.
* Serving Flask app 'app'
* Debug mode: on
INFO:werkzeug: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
INFO:werkzeug:Press CTRL+C to quit
INFO:werkzeug: * Restarting with stat
2026-02-16 21:01:02.570108: I tensorflow/core/util/port.cc:153] oneDNN custom operations are on. You may see slightly different numerical results due to floating-point round-off errors from different computation orders. To turn them off, set the environment variable `TF_ENABLE_ONEDNN_OPTS=0`.
2026-02-16 21:01:04.132718: I tensorflow/core/util/port.cc:153] oneDNN custom operations are on. You may see slightly different numerical results due to floating-point round-off errors from different computation orders. To turn them off, set the environment variable `TF_ENABLE_ONEDNN_OPTS=0`.
2026-02-16 21:01:06.609434: I tensorflow/core/platform/cpu_feature_guard.cc:210] This TensorFlow binary is optimized to use available CPU instructions in performance-critical operations.
To enable the following instructions: AVX2 AVX512F AVX512_VNNI FMA, in other operations, rebuild TensorFlow with the appropriate compiler flags.
WARNING:absl:Compiled the loaded model, but the compiled metrics have yet to be built. `model.compile_metrics` will be empty until you train or evaluate the model.
WARNING:werkzeug: * Debugger is active!
INFO:werkzeug: * Debugger PIN: 112-889-546
p...
bash
powershell
main
Venkata Durga Parvathi (9 hours ago)
Ln 8, Col 11
Spaces: 4
UTF-8
CRLF
Python
Finish Setup
Python 3.10 (64-bit)


SmartInternz_SmartSorting_Final
EXPLORER
OPEN EDITORS
SMARTINTERNZ_SMARTSORTING_FINAL
dataset
all
test
fresh
rotten
train
fresh
rotten
model
healthy_vs_rotten.h5
static
images
style.css
templates
about.html
base.html
contact.html
index.html
predict.html
result.html
uploads
app.py
predict.py
pro.gitignore
requirements.txt
split_dataset.py
train.py
OUTLINE
TIMELINE
split_dataset.py
healthy_vs_rotten.h5
train.py 4
predict.py
app.py
anthracnose_014.jpg
contact.html
FreshGuava (108).j
train.py > ...
1 import tensorflow as tf
2 from tensorflow.keras.preprocessing.image import ImageDataGenerator
3 from tensorflow.keras.applications import MobileNetV2
PROBLEMS 4
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
* Serving Flask app 'app'
* Debug mode: on
INFO:werkzeug: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
INFO:werkzeug:Press CTRL+C to quit
INFO:werkzeug: * Restarting with stat
2026-02-16 21:01:02.570108: I tensorflow/core/util/port.cc:153] oneDNN custom operations are on. You may see slightly different numerical results due to floating-point round-off errors from different computation orders. To turn them off, set the environment variable `TF_ENABLE_ONEDNN_OPTS=0`.
2026-02-16 21:01:04.132718: I tensorflow/core/util/port.cc:153] oneDNN custom operations are on. You may see slightly different numerical results due to floating-point round-off errors from different computation orders. To turn them off, set the environment variable `TF_ENABLE_ONEDNN_OPTS=0`.
2026-02-16 21:01:06.609434: I tensorflow/core/platform/cpu_feature_guard.cc:210] This TensorFlow binary is optimized to use available CPU instructions in performance-critical operations.
To enable the following instructions: AVX2 AVX512F AVX512_VNNI FMA, in other operations, rebuild TensorFlow with the appropriate compiler flags.
WARNING:absl:Compiled the loaded model, but the compiled metrics have yet to be built. `model.compile_metrics` will be empty until you train or evaluate the model.
WARNING:werkzeug: * Debugger is active!
INFO:werkzeug: * Debugger PIN: 112-889-546
INFO:werkzeug:127.0.0.1 - - [16/Feb/2026 21:02:13] "GET / HTTP/1.1" 200 -
INFO:werkzeug:127.0.0.1 - - [16/Feb/2026 21:02:13] "GET / HTTP/1.1" 200 -
INFO:werkzeug:127.0.0.1 - - [16/Feb/2026 21:02:13] "GET /static/style.css HTTP/1.1" 304 -
INFO:werkzeug:127.0.0.1 - - [16/Feb/2026 21:02:14] "GET /static/style.css HTTP/1.1" 304 -
INFO:werkzeug:127.0.0.1 - - [16/Feb/2026 21:02:14] "GET /favicon.ico HTTP/1.1" 404 -
INFO:werkzeug:127.0.0.1 - - [16/Feb/2026 21:02:20] "GET / HTTP/1.1" 200 -
INFO:werkzeug:127.0.0.1 - - [16/Feb/2026 21:02:20] "GET /static/style.css HTTP/1.1" 304 -
INFO:werkzeug:127.0.0.1 - - [16/Feb/2026 21:02:22] "GET /about HTTP/1.1" 200 -
INFO:werkzeug:127.0.0.1 - - [16/Feb/2026 21:02:22] "GET /static/style.css HTTP/1.1" 304 -
INFO:werkzeug:127.0.0.1 - - [16/Feb/2026 21:02:23] "GET /predict HTTP/1.1" 200 -
INFO:werkzeug:127.0.0.1 - - [16/Feb/2026 21:02:23] "GET /static/style.css HTTP/1.1" 304 -
INFO:werkzeug:127.0.0.1 - - [16/Feb/2026 21:02:24] "GET /contact HTTP/1.1" 200 -
INFO:werkzeug:127.0.0.1 - - [16/Feb/2026 21:02:24] "GET /static/style.css HTTP/1.1" 304 -
INFO:werkzeug:127.0.0.1 - - [16/Feb/2026 21:02:26] "GET /predict HTTP/1.1" 200 -
INFO:werkzeug:127.0.0.1 - - [16/Feb/2026 21:02:26] "GET /static/style.css HTTP/1.1" 304 -
python
bash
powershell


Smart Sorting
127.0.0.1:5000
Smart Sorting
Home
About
Predict
Contact
AI Smart Fruit Sorting
Transfer learning based system that classifies fruits and vegetables as fresh or rotten for warehouses and supermarkets.
Start Prediction
Transfer Learning
Pretrained CNN fine-tuned on fruit images for high-accuracy freshness detection.
Real-time
Fast inference suitable for conveyor-belt based smart sorting systems.
Reduced Waste
Automatically separates rotten produce early to minimize food loss.
© 2026 Smart Sorting | AI Powered Fruit Detection


Smart Sorting
127.0.0.1:5000/predict
Smart Sorting
Home
About
Predict
Contact
Upload Fruit Image
Smart Sorting will classify your fruit as fresh or rotten using the trained CNN.
Choose Image
FreshGuava (108).jpg
Predict Freshness


Smart Sorting
127.0.0.1:5000/predict
Smart Sorting
Home
About
Predict
Contact
Prediction Result
Fresh
Confidence: 100.00%
Model: Transfer Learning CNN (Transfer learning CNN)
Try Another Image
© 2026 Smart Sorting | AI Powered Fruit Detection