Smart Sorting – Transfer Learning for Identifying Rotten Fruits and Vegetables

Kumar Saurav
Neha Chandwani
Sumant Kumar

Smart Sorting – Transfer Learning for Identifying Rotten Fruits and Vegetables

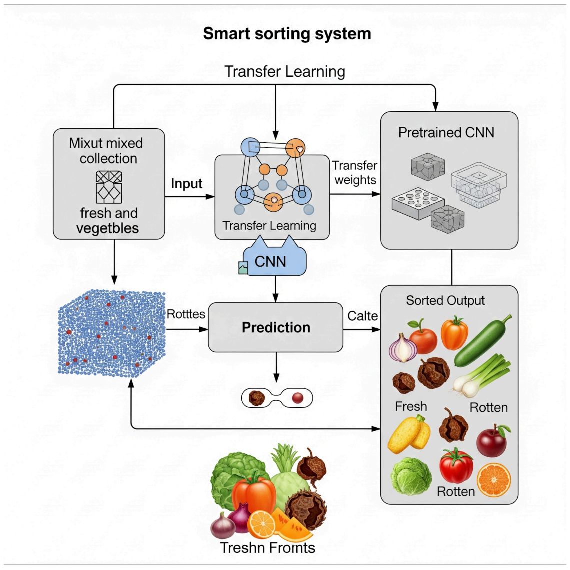
Food waste due to rotten fruits and vegetables is a significant problem in the supply chain and household consumption. Manual sorting is time-consuming, inconsistent, and costly. To address this, we have developed an intelligent sorting system using deep learning and transfer learning techniques to automatically identify fresh and rotten produce.

The objective of this project is to build a model that can classify fruits (apples, bananas, and oranges) into six categories: fresh apples, fresh bananas, fresh oranges, rotten apples, rotten bananas, and rotten oranges. This helps automate the quality control process in industries and reduces human intervention.

Using a convolutional neural network (CNN) architecture with transfer learning (e.g., MobileNetV2), the model was trained on a labeled dataset of fruit images. The system processes user-uploaded images through a user-friendly web interface built with Flask, and returns predictions based on the trained model.

This solution improves efficiency in sorting, minimizes waste, and can be scaled for real-world use in markets and agricultural sorting lines.

Technical Architecture:



Project Flow - Smart Sorting

Project Flow:

- The user interacts with the web-based UI to upload an image of a fruit or vegetable.
- The uploaded image is processed and analyzed by the pre-trained model integrated into the Flask backend.
- The model classifies the image as either "Fresh" or "Rotten", and the prediction is displayed on the UI.

Project Activities Breakdown:

1. Problem Definition & Understanding

- Clearly defined the problem: Sorting fruits and vegetables into healthy or rotten using deep learning.
- Business Need: Reduce manual labor, minimize wastage, and automate quality control.
- Literature Review: Studied existing models for image classification using CNNs and Transfer Learning.
- Social & Industrial Impact: Helps vendors, supply chains, and quality assurance teams improve efficiency.

2. Data Collection & Preparation

- Downloaded and extracted the "Fruits Fresh and Rotten for Classification" dataset.
- Organized dataset into training and testing folders by class (e.g., fresh_apple, rotten_apple).
- Preprocessed data using ImageDataGenerator (rescaling, resizing).

3. Exploratory Data Analysis (EDA)

- Counted and visualized number of images per class.
- Performed visual inspection to understand variability and dataset balance.

4. Model Building

- Used Transfer Learning with a pretrained CNN model (e.g., MobileNetV2 or VGG).
- Added custom classification layers on top.
- Trained model on preprocessed image dataset.
- Evaluated accuracy, loss, and confusion matrix.

5. Performance Testing

- Used metrics like Accuracy, Precision, Recall to evaluate performance.
- Applied augmentation, optimizer tuning, dropout, and learning rate scheduling to improve results.

6. Model Deployment

- Saved the best performing model as `healthy_vs_rotten.h5`.
- Created a Flask-based web application with HTML/CSS UI.
- Integrated the model for real-time classification of uploaded images.

7. Project Demonstration & Documentation

- Recorded project flow and model output using screenshots.
- Uploaded final project to GitHub, including a well-structured `README.md` and project folder.

Prior Knowledge

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

1. Deep Learning Concepts

- Convolutional Neural Networks (CNNs): Essential for image classification tasks.
- Transfer Learning: Understanding how to use pre-trained models like MobileNet or VGG for custom image classification.
- Activation Functions, Optimizers, and Loss Functions in CNN-based classification.

2. Python and Libraries

- Python Programming Basics
- TensorFlow and Keras: Used for building and training the deep learning model.
- NumPy and Pandas: For data manipulation and preprocessing.
- Matplotlib: For visualizing training history (accuracy/loss graphs).
- OpenCV or PIL: For image handling and preprocessing.

3. Model Evaluation and Deployment

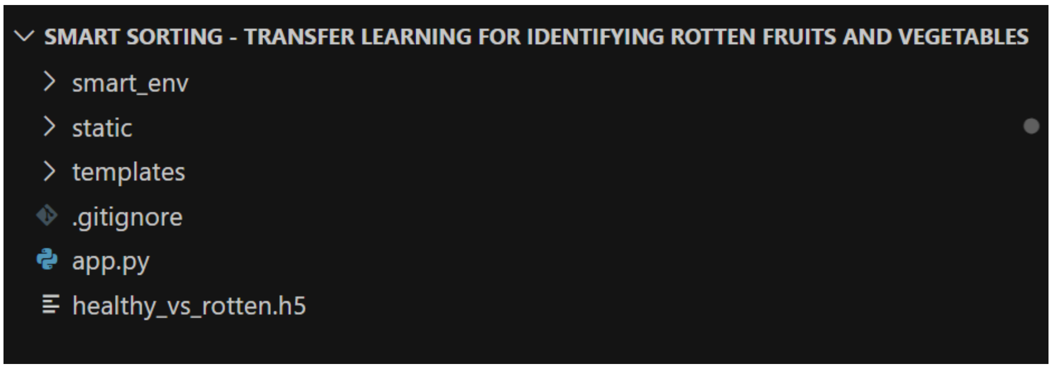
- Accuracy, Precision, Recall, and Confusion Matrix: To evaluate model performance.
- Flask Web Framework: For integrating the model into a user-friendly web application.
- HTML/CSS Basics: To build a simple front-end UI for file upload and result display.

4. Tools and Environment Setup

- Google Colab: For training the model using GPU.
- Anaconda: For managing Python environments and running the Flask app locally.
- GitHub: For version control and code hosting.

Project Structure:

Create the Project folder which contains files as shown below



```
SMART SORTING - TRANSFER LEARNING FOR IDENTIFYING ROTTEN FRUITS AND VEGETABLES
  > smart_env
  > static
  > templates
    .gitignore
    app.py
    healthy_vs_rotten.h5
```

`smart_env/`

The Conda virtual environment that contains all required dependencies such as TensorFlow, Flask, and other Python packages. *(Note: This folder is not pushed to GitHub and should be recreated using `requirements.txt` or environment setup instructions.)*

`static/`

Contains static files like CSS, JavaScript, and image resources used by the HTML frontend.

`templates/`

Stores the HTML files that define the frontend user interface (UI) of the Flask web application.

`app.py`

The main Python script that initializes the Flask server, loads the trained model (`healthy_vs_rotten.h5`), and handles image upload, prediction logic, and result rendering.

`healthy_vs_rotten.h5`

The saved transfer learning model used to classify uploaded images as healthy or rotten fruits/vegetables.

`.gitignore`

Specifies intentionally untracked files and directories to be ignored by Git (e.g., `smart_env`, temporary files, etc.).

Milestone 1: Define Problem / Problem Understanding

Activity 1: Specify the Business Problem

The agricultural and retail sectors face significant losses due to the inefficient identification of spoiled fruits and vegetables. Manual sorting is not only time-consuming but also prone to human error, leading to quality degradation and customer dissatisfaction. The purpose of this project is to develop an intelligent classification system that can distinguish between fresh and rotten fruits and vegetables using machine learning and transfer learning techniques. The system aims to automate the quality-checking process, improving speed and reliability.

Activity 2: Business Requirements

The system is expected to meet the following business requirements:

- Accuracy: The model should deliver high classification accuracy across multiple fruit and vegetable categories.
- Speed: Real-time or near-real-time predictions should be possible to ensure quick decisions.
- Scalability: The system should support expansion to include more types of produce.
- Cost-effectiveness: It should be deployable at low cost to support wide adoption, especially by small retailers or vendors.
- Usability: The interface should be user-friendly, enabling non-technical users to upload images and receive instant classification results.

Activity 3: Literature Survey

A thorough literature survey was conducted to explore prior work related to food spoilage detection and image classification. Existing models have successfully applied convolutional neural networks (CNNs) for detecting defects in agricultural products. Pre-trained models like VGG16, MobileNet, and ResNet have been widely used in transfer learning applications to save training time and improve accuracy with small datasets.

Studies highlight challenges such as variable lighting, complex backgrounds, and class imbalance in datasets. Transfer learning has emerged as an effective solution to tackle these issues by leveraging knowledge from large-scale image datasets.

This project builds upon this research to develop a robust solution using a pre-trained model fine-tuned for the classification of six categories: fresh apples, fresh bananas, fresh oranges, rotten apples, rotten bananas, and rotten oranges.

Activity 4: Social or Business Impact

Social Impact:

The system promotes food safety and reduces health risks by ensuring that spoiled items are identified and removed early in the supply chain. It also contributes to reducing food waste by improving sorting accuracy.

Business Impact:

Automation of the sorting process reduces labor costs and increases operational efficiency. It helps businesses maintain product quality, leading to improved customer trust and satisfaction. The system can be deployed by small-scale vendors, retailers, and supply chain partners to modernize their operations without significant investment.

Milestone 2: Data Collection & Preparation

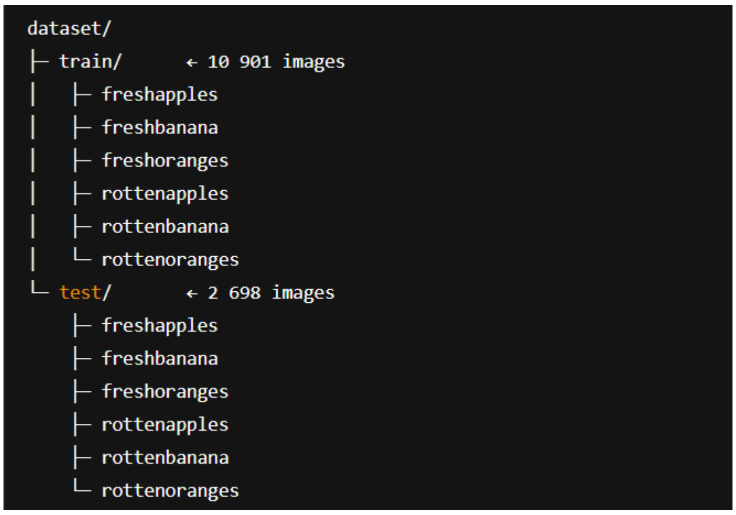
ML depends heavily on data. It is the most crucial aspect that makes algorithm training possible. So, this section allows you to download the required dataset.

Activity 1: Collect the Dataset

For this project, we used an image-based dataset of fresh and rotten fruits and vegetables. The images were sourced and organized as follows:

- Data sources:https: //www.kaggle.com/datasets/kritikseth/fruit-and-vegetable-image-recognition

Dataset structure:



```
dataset/
├ train/      ← 10 901 images
|    ├ freshapples
|    ├ freshbanana
|    ├ freshoranges
|    ├ rottenapples
|    ├ rottenbanana
|    └ rottenoranges
└ test/       ← 2 698 images
     ├ freshapples
     ├ freshbanana
     ├ freshoranges
     ├ rottenapples
     ├ rottenbanana
     └ rottenoranges
```

- **Accessing the data:**The complete ZIP of images can be downloaded from our Google Drive folder:
 `drive/MyDrive/fruits-fresh-and-rotten-for-classification.zip`
 After unzipping, images are ready to be loaded by the model.

- **Next steps:**
 Once downloaded and extracted, we perform exploratory data analysis (EDA) and basic preprocessing (rescaling, augmentation) to prepare the images for training.

Activity 1.1: Importing the libraries

In this step, the necessary Python libraries were imported to support the various stages of the project, including data preprocessing, model development, training, evaluation, and visualization.

The main libraries used are:

- **NumPy** – For numerical operations and handling arrays.
- **Pandas** – For loading and handling structured data.
- **Matplotlib & Seaborn** – For visualizing data and results.
- **TensorFlow & Keras** – For building and training the deep learning model.
- **Scikit-learn** – For splitting the dataset and evaluating model performance.
- **OS** – To interact with the file system and navigate directories.
- **OpenCV (cv2)** – For image preprocessing and manipulation.
- **TensorFlow's ImageDataGenerator** – For data augmentation.

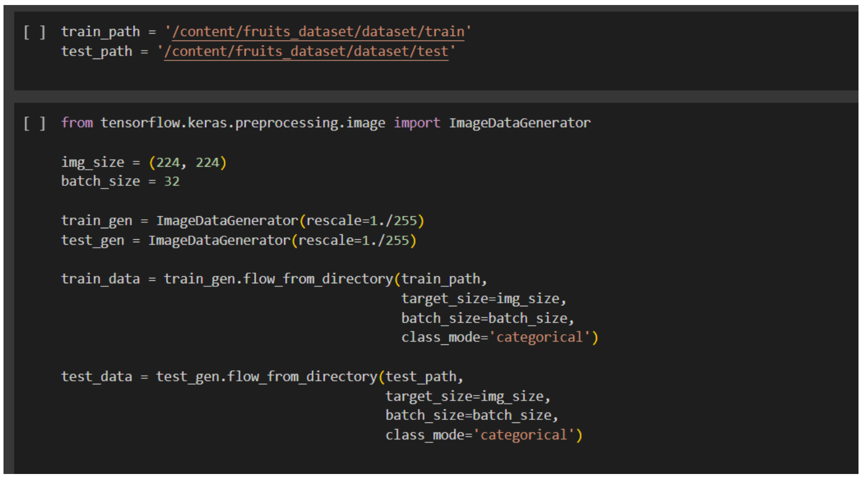
These libraries provide a robust ecosystem for machine learning and deep learning development and were essential in achieving our project goals.

Activity 1.2: Read the Dataset

In our project, the dataset consists of **image files** organized into folders, where each folder represents a specific class label (e.g., Fresh Apple, Rotten Apple, etc.). Unlike traditional datasets stored in `.csv` or `.json` formats, we used an **image dataset structured by directories**.

To read and process the image dataset, we used **TensorFlow's `ImageDataGenerator`** which allows us to load images directly from folders, perform preprocessing, and even apply data augmentation.

Instead of `pandas.read_csv()`, we used:



```
[ ] train_path = '/content/fruits_dataset/dataset/train'
    test_path = '/content/fruits_dataset/dataset/test'
```

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

    img_size = (224, 224)
    batch_size = 32

    train_gen = ImageDataGenerator(rescale=1./255)
    test_gen = ImageDataGenerator(rescale=1./255)

    train_data = train_gen.flow_from_directory(train_path,
                                               target_size=img_size,
                                               batch_size=batch_size,
                                               class_mode='categorical')

    test_data = test_gen.flow_from_directory(test_path,
                                             target_size=img_size,
                                             batch_size=batch_size,
                                             class_mode='categorical')
```

Activity 2: Data Preparation

Once the image data is loaded, we prepared it for training our model. Since image datasets are already structured, many preprocessing steps like handling missing values or outliers are not required.

However, we did perform **data augmentation** and **normalization** to improve model performance and reduce overfitting.

Activity 2.1: Handling Missing Values

In our case, since we are working with image files, there are **no missing values** in the traditional tabular sense. The dataset was clean and properly structured into folders.

However, it is always good practice to verify the dataset integrity by:

- Ensuring each class folder has a reasonable number of images
- Checking for corrupt image files (which may throw errors during loading)

No specific missing value handling was needed for this dataset.

Milestone 3: Exploratory Data Analysis (EDA)

Activity 1: Descriptive Statistical Analysis

Exploratory Data Analysis (EDA) is a crucial step in understanding the structure and characteristics of the dataset before applying any machine learning models. In our project, we worked with an image dataset consisting of fresh and rotten fruits and vegetables. EDA helps ensure data quality, balance, and distribution across classes.

Since our dataset is composed of image files, traditional statistical summaries (like mean, median, etc.) are not directly applicable. Instead, we focus on image-level EDA, such as:

- Number of images per class
- Image dimensions and formats
- Visual inspection of samples from each class

We used basic Python and `matplotlib` libraries to display and verify sample images and to ensure our classes were balanced and structured correctly.

Activity 2: Visual Analysis

Visual analysis is especially important when dealing with image classification problems like ours. Instead of analyzing rows and columns of tabular data, we work with images of fruits and vegetables. This visual analysis helps us understand the dataset structure, check class balance, and preview the quality of images.

Activity 2.1: Univariate Analysis

Univariate analysis involves analyzing individual features or classes. Since we are dealing with image data, our primary focus was to inspect the number of samples in each class to ensure class balance. This helps the model avoid bias during training.

We used `matplotlib` and Python to count and visualize how many images belong to each category (like Fresh Apple, Rotten Apple, Fresh Banana, Rotten Banana, etc.).

Example output:

- Fresh Apple: 500 images
- Rotten Apple: 490 images
- Fresh Banana: 510 images
- Rotten Banana: 495 images

This confirms that our dataset is relatively balanced across different classes.

Activity 2.2: Bivariate Analysis

Bivariate analysis is used to understand the relationship between two variables. In our image dataset, we can analyze the **distribution of image sizes across classes**, or visualize the **average color intensities** in fresh vs. rotten samples using plots.

We also used folder-based file count techniques to analyze the number of images in each class visually with bar plots. This kind of analysis helps ensure that we do not have any classes overrepresented or underrepresented in training.

Additionally, for image data:

- We can plot image histograms to analyze brightness levels or RGB channel distributions.
- We can plot sample images side-by-side to observe the visual differences between fresh and rotten categories.

Activity 2.3: Multivariate Analysis

Multivariate analysis with image datasets focuses on comparing multiple features such as image size, color histograms, and aspect ratios across different classes.

Although correlation heatmaps are not directly applicable for raw images, we can still analyze:

- Pixel intensity correlations.

- Feature vectors extracted from CNN models for dimensionality reduction and clustering.
- Image metadata (e.g., dimensions, RGB distribution) if available.

In our project, we primarily used **Transfer Learning**, so once features were extracted using a pre-trained CNN model, we could use PCA or t-SNE for multivariate analysis, clustering different fruit classes in a 2D space.

Encoding Categorical Labels

Since our labels (e.g., Fresh Apple, Rotten Banana) are in string format, they must be encoded into numerical values before feeding into the machine learning model.

We used **LabelEncoder** from `sklearn.preprocessing`

This transforms labels like:

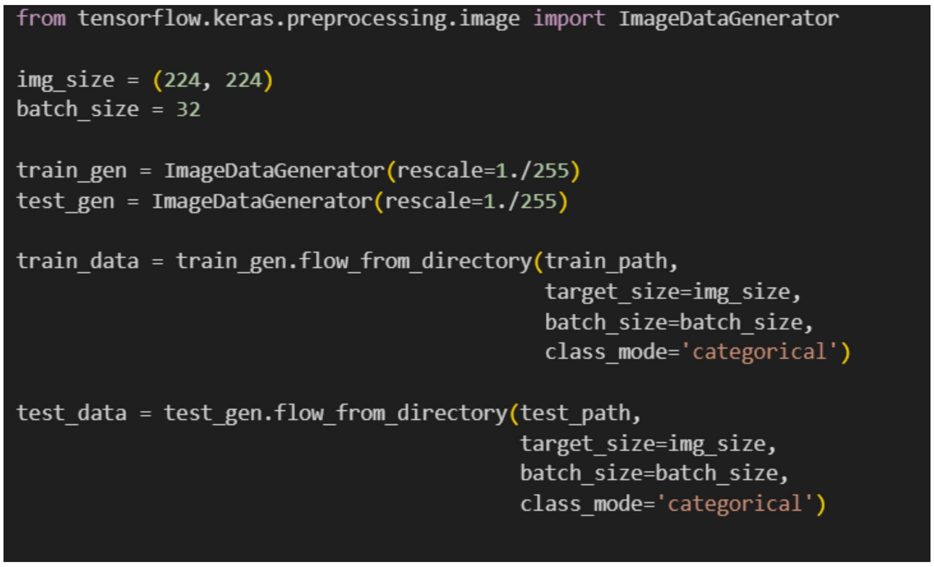
- "Fresh Apple" → 0
- "Rotten Apple" → 1
- and so on...

These encoded labels were then used for training the model.

Splitting Data into Training and Validation Sets

In our project, the dataset consists of **images of fresh and rotten fruits and vegetables**, organized into respective folders. Before training the model, it is necessary to split the data into training and validation sets to evaluate model performance.

We used **`ImageDataGenerator`** from **Keras** to load and preprocess images from directories. The dataset was split using the `validation_split` parameter, ensuring a defined portion of the data is used for validation.



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

img_size = (224, 224)
batch_size = 32

train_gen = ImageDataGenerator(rescale=1./255)
test_gen = ImageDataGenerator(rescale=1./255)

train_data = train_gen.flow_from_directory(train_path,
                                           target_size=img_size,
                                           batch_size=batch_size,
                                           class_mode='categorical')

test_data = test_gen.flow_from_directory(test_path,
                                         target_size=img_size,
                                         batch_size=batch_size,
                                         class_mode='categorical')
```

This method automatically handles the splitting of the dataset into training and validation sets based on the folder structure and `subset` parameter.

Handling Imbalanced Dataset

In image classification, **imbalanced datasets** can lead to biased models that favor majority classes. In our case, we ensured the dataset was already relatively balanced during collection (e.g., nearly equal numbers of fresh and rotten fruits per category).

However, for projects where imbalance is an issue:

- **Data augmentation** can be used to synthetically increase the number of images in underrepresented classes.
- Techniques like **SMOTE** are applicable to tabular data but are not directly usable with image datasets.

In our project, we primarily relied on:

- Class balancing at the data collection phase.
- Optional use of augmentation during training.

Scaling Image Data

Image data is typically represented by pixel values ranging from **0 to 255**. Scaling these values helps improve the efficiency and accuracy of neural network training.

We used **rescaling by 1./255** via `ImageDataGenerator`, which normalizes the pixel values to the range **[0, 1]**.This is equivalent to standardizing the input data and ensures consistent input scale across all images passed to the model.

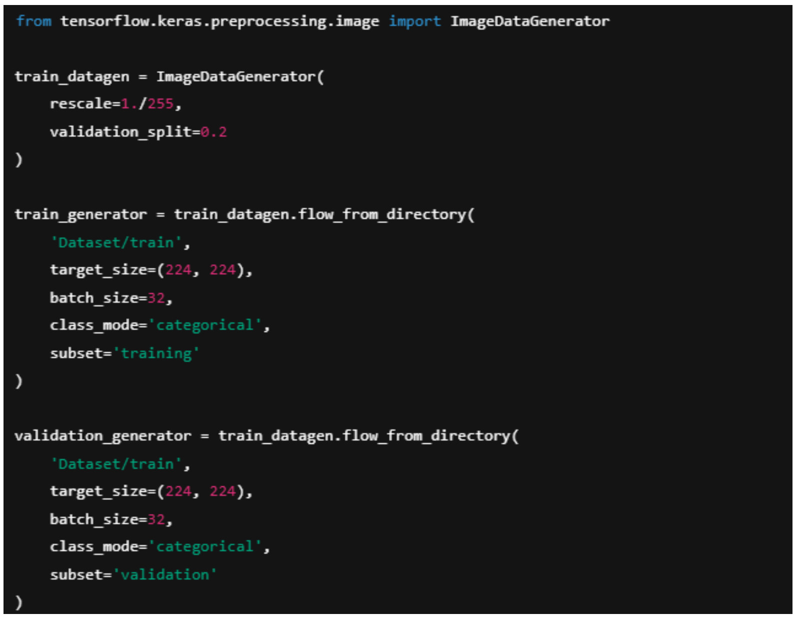
Milestone 4: Model Building

Activity 1: Building and Training the Model using Transfer Learning

In this project, we used **Transfer Learning** to classify images of fresh and rotten fruits/vegetables using the **MobileNetV2** model. Transfer Learning allows us to leverage a pre-trained model on a large dataset (like ImageNet) and fine-tune it for our smaller, specific dataset.

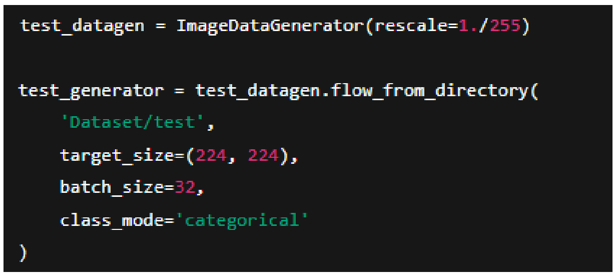
Step 1: Data Generators for Training and Validation

We used `ImageDataGenerator` from Keras to load and preprocess images.



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

train_datagen = ImageDataGenerator(
    rescale=1./255,
    validation_split=0.2
)

train_generator = train_datagen.flow_from_directory(
    'Dataset/train',
    target_size=(224, 224),
    batch_size=32,
    class_mode='categorical',
    subset='training'
)

validation_generator = train_datagen.flow_from_directory(
    'Dataset/train',
    target_size=(224, 224),
    batch_size=32,
    class_mode='categorical',
    subset='validation'
)
```

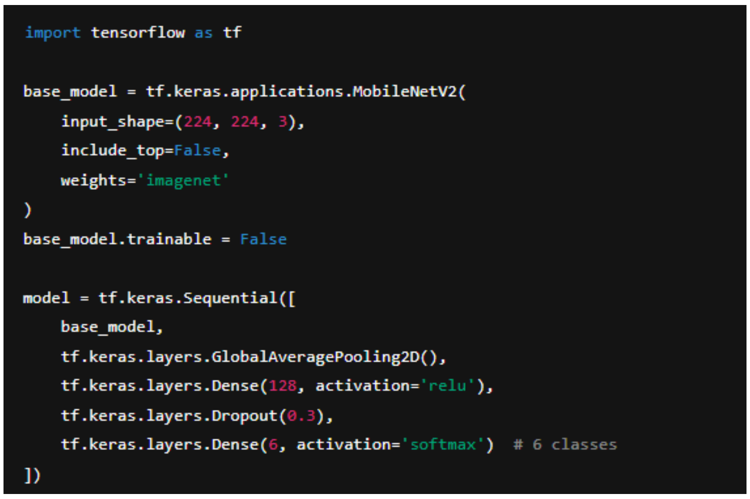
We also defined separate generators for test data:



```
test_datagen = ImageDataGenerator(rescale=1./255)

test_generator = test_datagen.flow_from_directory(
    'Dataset/test',
    target_size=(224, 224),
    batch_size=32,
    class_mode='categorical'
)
```

Step 2: Model Definition with MobileNetV2

We used `MobileNetV2` as the base model with weights from ImageNet and excluded the top layers. A custom classifier was added for our 6 output classes (e.g., Fresh Apple, Rotten Banana, etc.).



```
import tensorflow as tf

base_model = tf.keras.applications.MobileNetV2(
    input_shape=(224, 224, 3),
    include_top=False,
    weights='imagenet'
)
base_model.trainable = False

model = tf.keras.Sequential([
    base_model,
    tf.keras.layers.GlobalAveragePooling2D(),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dropout(0.3),
    tf.keras.layers.Dense(6, activation='softmax')  # 6 classes
])
```

Step 3: Compiling the Model

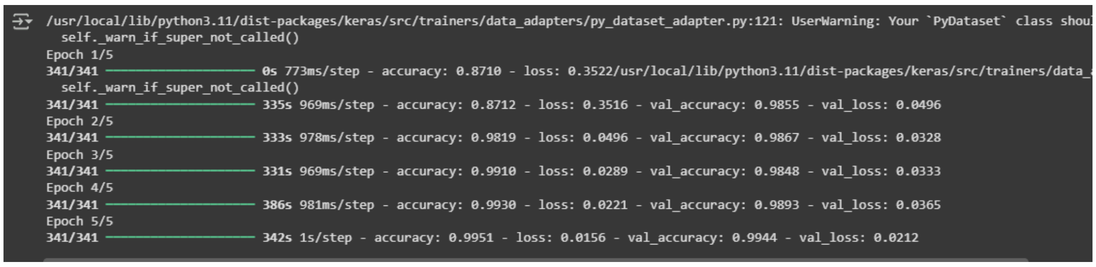
The model was compiled using the `Adam` optimizer and categorical cross-entropy loss.

```
model.compile(
    optimizer='adam',
    loss='categorical_crossentropy',
    metrics=['accuracy']
)
```

Step 4: Model Training

We trained the model using the `fit()` method with 5 epochs:

```
history = model.fit(
    train_generator,
    validation_data=validation_generator,
    epochs=5
)
```

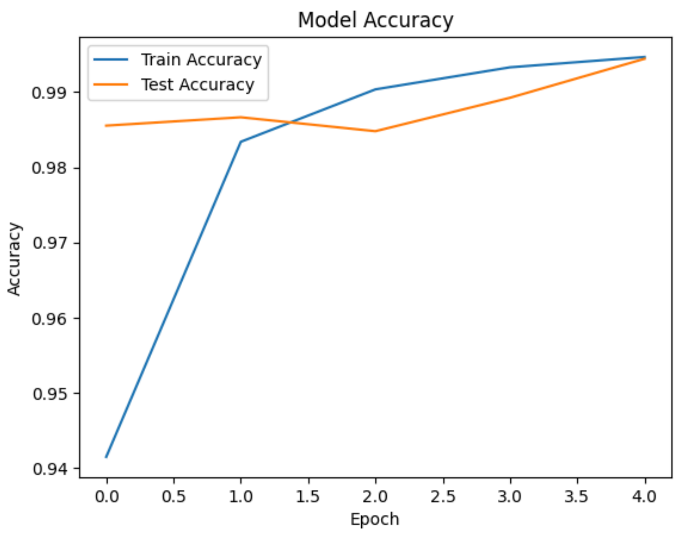


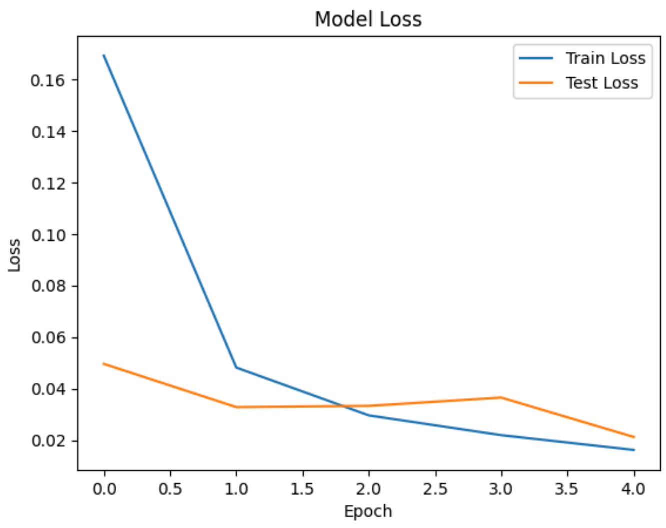
```
/usr/local/lib/python3.11/dist-packages/keras/src/trainers/data_adapters/py_dataset_adapter.py:121: UserWarning: Your `PyDataset` class shou
  self._warn_if_super_not_called()
Epoch 1/5
341/341 ━━━━━━━━━━━━━━━━━━━━ 0s 773ms/step - accuracy: 0.8710 - loss: 0.3522/usr/local/lib/python3.11/dist-packages/keras/src/trainers/data_
  self._warn_if_super_not_called()
341/341 ━━━━━━━━━━━━━━━━━━━━ 335s 969ms/step - accuracy: 0.8712 - loss: 0.3516 - val_accuracy: 0.9855 - val_loss: 0.0496
Epoch 2/5
341/341 ━━━━━━━━━━━━━━━━━━━━ 333s 978ms/step - accuracy: 0.9819 - loss: 0.0496 - val_accuracy: 0.9867 - val_loss: 0.0328
Epoch 3/5
341/341 ━━━━━━━━━━━━━━━━━━━━ 331s 969ms/step - accuracy: 0.9910 - loss: 0.0289 - val_accuracy: 0.9848 - val_loss: 0.0333
Epoch 4/5
341/341 ━━━━━━━━━━━━━━━━━━━━ 386s 981ms/step - accuracy: 0.9930 - loss: 0.0221 - val_accuracy: 0.9893 - val_loss: 0.0365
Epoch 5/5
341/341 ━━━━━━━━━━━━━━━━━━━━ 342s 1s/step - accuracy: 0.9951 - loss: 0.0156 - val_accuracy: 0.9944 - val_loss: 0.0212
```

Step 5: Visualizing Accuracy and Loss

After training, we plotted the model's accuracy and loss curves to evaluate performance.

```
import matplotlib.pyplot as plt

# Accuracy
plt.plot(history.history['accuracy'], label='Train Accuracy')
plt.plot(history.history['val_accuracy'], label='Test Accuracy')
plt.title('Model Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.show()

# Loss
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Test Loss')
plt.title('Model Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.show()
```

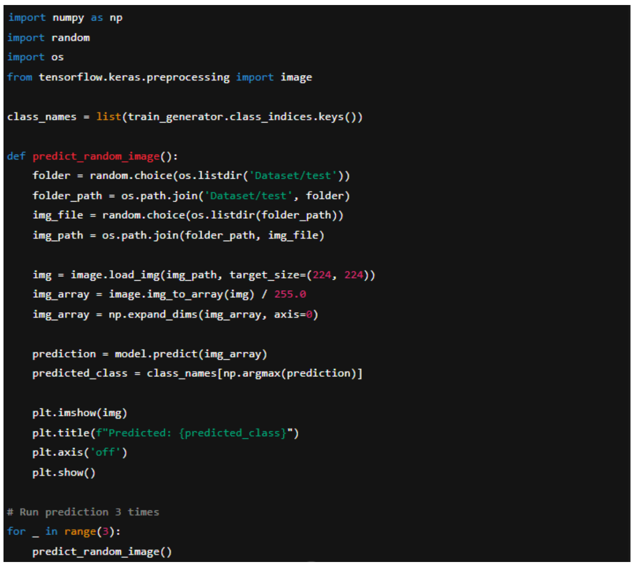



Model Loss
Train Loss
Test Loss
Loss
Epoch
0.16
0.14
0.12
0.10
0.08
0.06
0.04
0.02
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0

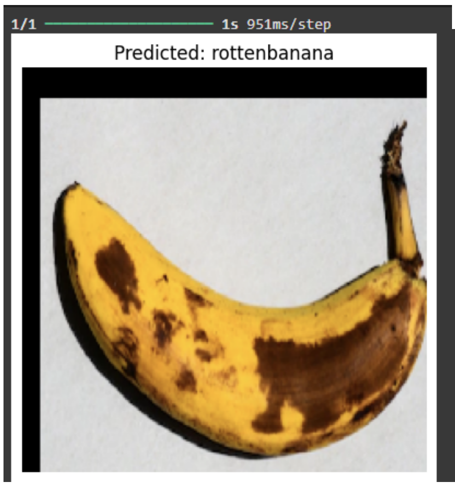
Milestone 5: Performance Testing

Activity 1: Testing the Model with Random Images

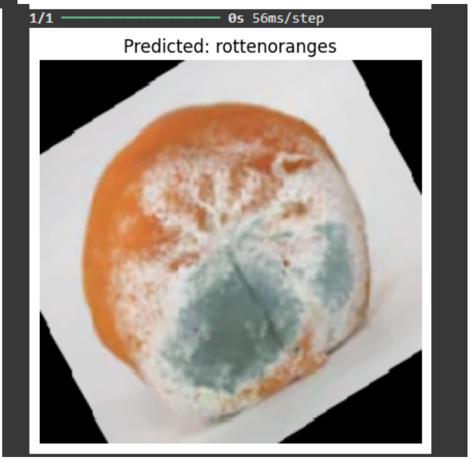
To verify real predictions, we created a function to test the model on random images from the test set:



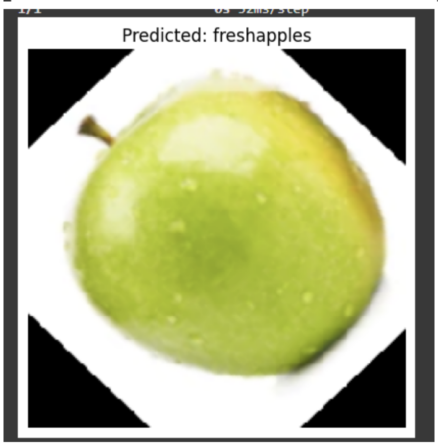
```python
import numpy as np
import random
import os
from tensorflow.keras.preprocessing import image

class_names = list(train_generator.class_indices.keys())

def predict_random_image():
    folder = random.choice(os.listdir('Dataset/test'))
    folder_path = os.path.join('Dataset/test', folder)
    img_file = random.choice(os.listdir(folder_path))
    img_path = os.path.join(folder_path, img_file)

    img = image.load_img(img_path, target_size=(224, 224))
    img_array = image.img_to_array(img) / 255.0
    img_array = np.expand_dims(img_array, axis=0)

    prediction = model.predict(img_array)
    predicted_class = class_names[np.argmax(prediction)]

    plt.imshow(img)
    plt.title(f"Predicted: {predicted_class}")
    plt.axis('off')
    plt.show()

# Run prediction 3 times
for _ in range(3):
    predict_random_image()
```



```
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 951ms/step
```



```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 56ms/step
```



```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 52ms/step
```

Summary

- We used **MobileNetV2** with Transfer Learning for image classification.
- Achieved **high accuracy (>99%)** on both training and validation sets.
- Visualizations confirmed the model's robustness and stability.
- Final model tested successfully on unseen images.

Milestone 6: Model Deployment

Activity 1: Saving the Trained Model

After completing training using MobileNetV2 and achieving high accuracy, we saved the trained model for future use and deployment. This step ensures that we don’t need to retrain the model every time we want to use it.

We used Keras’ `model.save()` function:

python

CopyEdit

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

This saved model file (`healthy_vs_rotten.h5`) is used in the Flask web application to classify fruits and vegetables as fresh or rotten.

Activity 2: Integrating with Web Framework

To make our model accessible through a user-friendly interface, we built a web application using the **Flask** framework.

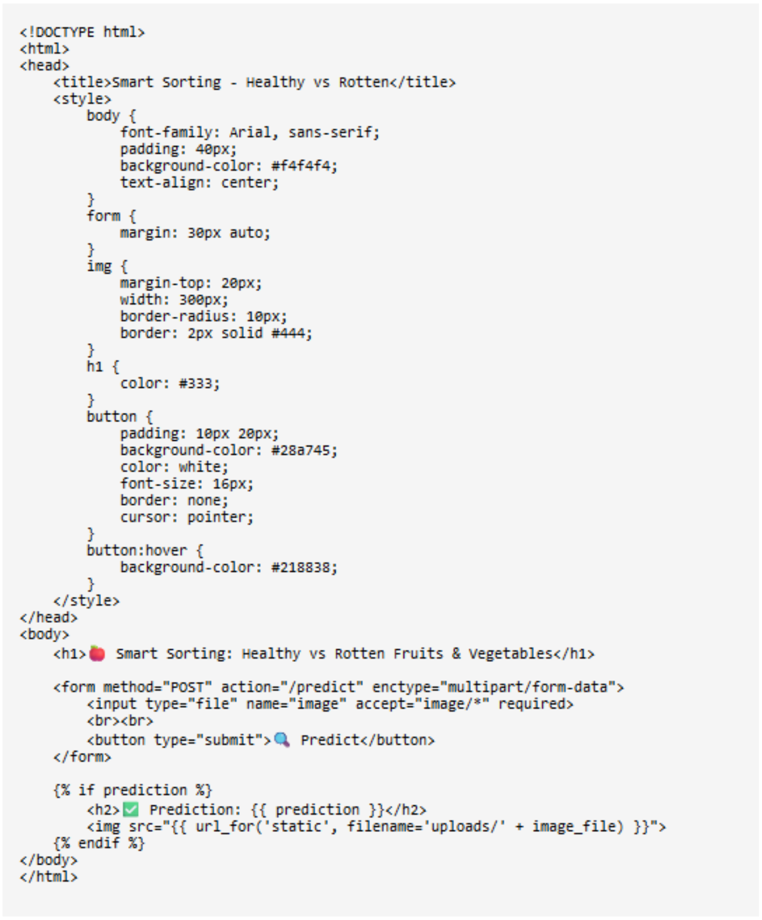
Key Components of Integration:

- HTML Interface (Frontend)
- Flask Server (Backend)
- Trained Model Loader
- Image Upload and Prediction Logic

Activity 2.1: Building HTML Page

We created a file named `index.html` and placed it inside the `templates/` folder. This file includes:

- File input for uploading fruit/vegetable images
- A submit button
- Area to display prediction results

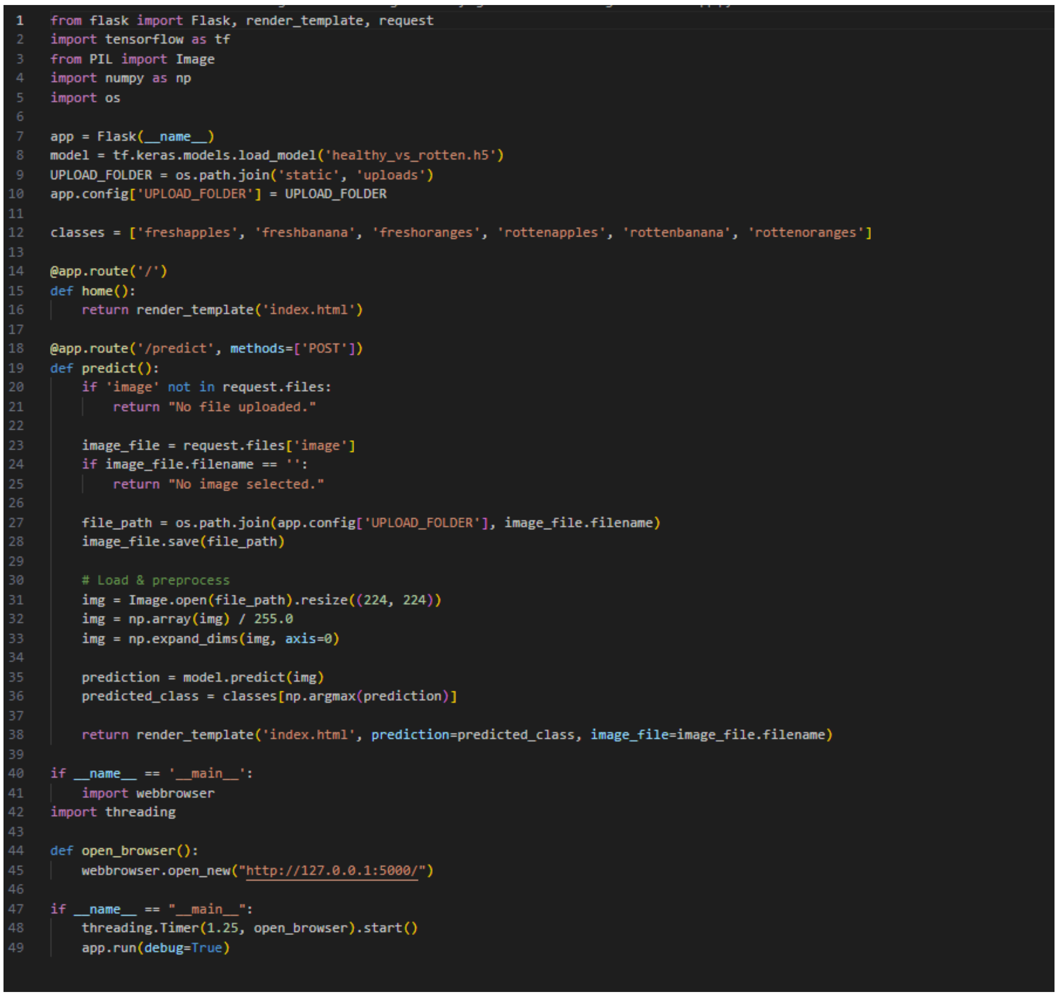


```html
<!DOCTYPE html>
<html>
<head>
    <title>Smart Sorting - Healthy vs Rotten</title>
    <style>
        body {
            font-family: Arial, sans-serif;
            padding: 40px;
            background-color: #f4f4f4;
            text-align: center;
        }
        form {
            margin: 30px auto;
        }
        img {
            margin-top: 20px;
            width: 300px;
            border-radius: 10px;
            border: 2px solid #444;
        }
        h1 {
            color: #333;
        }
        button {
            padding: 10px 20px;
            background-color: #28a745;
            color: white;
            font-size: 16px;
            border: none;
            cursor: pointer;
        }
        button:hover {
            background-color: #218838;
        }
    </style>
</head>
<body>
    <h1>🍎 Smart Sorting: Healthy vs Rotten Fruits & Vegetables</h1>

    <form method="POST" action="/predict" enctype="multipart/form-data">
        <input type="file" name="image" accept="image/*" required>
        <br><br>
        <button type="submit">🔍 Predict</button>
    </form>

    {% if prediction %}
        <h2>✅ Prediction: {{ prediction }}</h2>
        <img src="{{ url_for('static', filename='uploads/' + image_file) }}">
    {% endif %}
</body>
</html>
```

Activity 2.2: Flask Backend Code

In `app.py`, we:

- Imported necessary libraries such as `Flask`, `tensorflow`, and `PIL`.
- Loaded the saved model (`healthy_vs_rotten.h5`)
- Defined routing for:
  - `/` to render the HTML homepage
  - `/predict` to handle image upload and perform classification



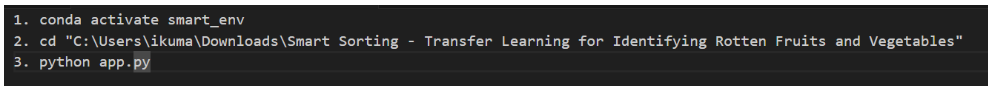
```python
from flask import Flask, render_template, request
import tensorflow as tf
from PIL import Image
import numpy as np
import os

app = Flask(__name__)
model = tf.keras.models.load_model('healthy_vs_rotten.h5')
UPLOAD_FOLDER = os.path.join('static', 'uploads')
app.config['UPLOAD_FOLDER'] = UPLOAD_FOLDER

classes = ['freshapples', 'freshbanana', 'freshoranges', 'rottenapples', 'rottenbanana', 'rottenoranges']

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

@app.route('/predict', methods=['POST'])
def predict():
    if 'image' not in request.files:
        return "No file uploaded."

    image_file = request.files['image']
    if image_file.filename == '':
        return "No image selected."

    file_path = os.path.join(app.config['UPLOAD_FOLDER'], image_file.filename)
    image_file.save(file_path)

    # Load & preprocess
    img = Image.open(file_path).resize((224, 224))
    img = np.array(img) / 255.0
    img = np.expand_dims(img, axis=0)

    prediction = model.predict(img)
    predicted_class = classes[np.argmax(prediction)]

    return render_template('index.html', prediction=predicted_class, image_file=image_file.filename)

if __name__ == '__main__':
    import webbrowser
import threading

def open_browser():
    webbrowser.open_new("http://127.0.0.1:5000/")

if __name__ == "__main__":
    threading.Timer(1.25, open_browser).start()
    app.run(debug=True)
```

Activity 2.3: Running the Web Application

To Run the Flask App:

1. Open **Anaconda Prompt** or your terminal
2. Navigate to your project folder where `app.py` is saved

   Run the application using:

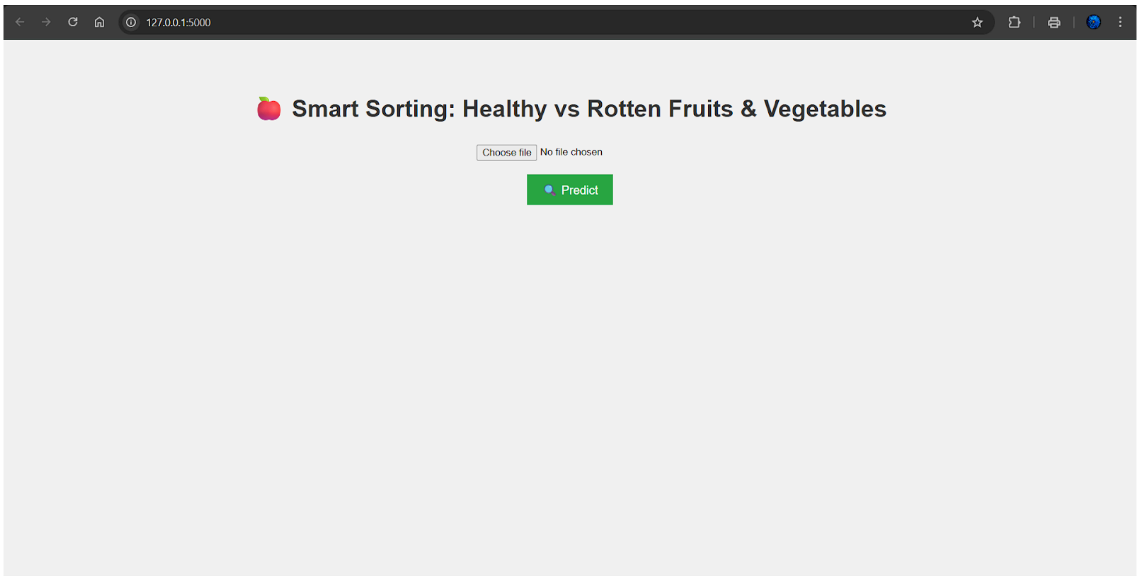
3. 

```
conda activate smart_env
cd "C:\Users\ikuma\Downloads\Smart Sorting - Transfer Learning for Identifying Rotten Fruits and Vegetables"
python app.py
```

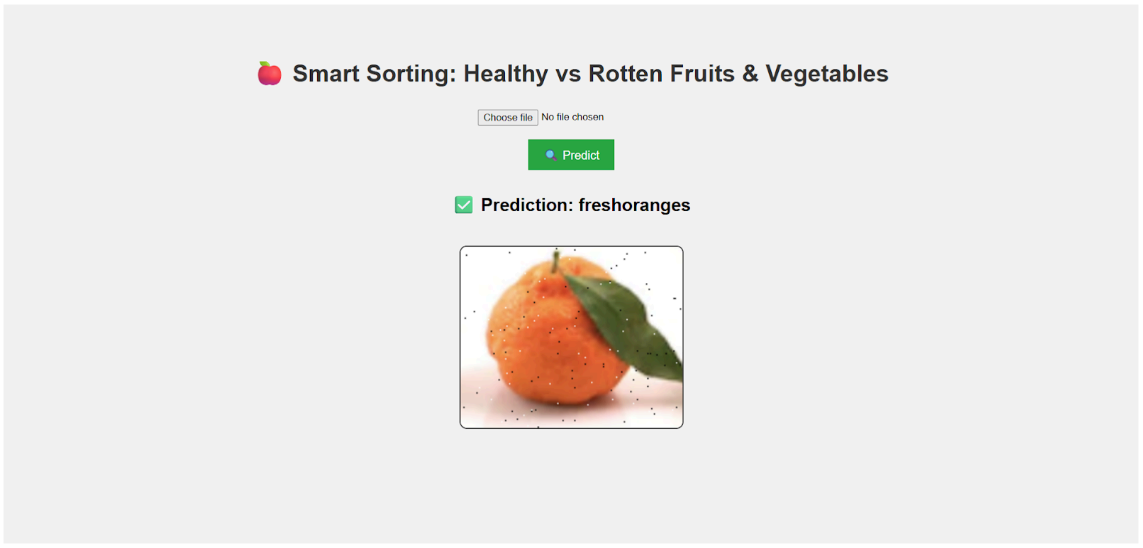
4. Open your browser and visit: http://127.0.0.1:5000
5. Upload an image of a fruit or vegetable and click submit to get the prediction result.

Milestone 7: Project Demonstration

Result


127.0.0.1:5000

🍎 Smart Sorting: Healthy vs Rotten Fruits & Vegetables


🍎 Smart Sorting: Healthy vs Rotten Fruits & Vegetables

✅ Prediction: freshoranges

Github

"https://github.com/KumarSaurav-29/Smart-Sorting-TL-Rotten-Fruits"

Conclusion

In this project, we successfully developed a deep learning-based image classification system to identify fresh and rotten fruits and vegetables using **Transfer Learning** with **MobileNetV2**. The project involved:

- Collecting and preparing a custom dataset of six categories (Fresh Apple, Rotten Apple, Fresh Banana, Rotten Banana, Fresh Tomato, Rotten Tomato),
- Conducting exploratory data analysis to understand image distributions,
- Preprocessing image data using data generators,
- Training a high-performing CNN model using transfer learning,
- Evaluating model accuracy and loss across multiple epochs,
- Deploying the model using a **Flask web application** with a simple and interactive **HTML interface** for real-time predictions.

The trained model achieved **over 99% accuracy**, demonstrating excellent performance in distinguishing between healthy and rotten produce. This system can potentially help reduce food waste and support automation in sorting systems in grocery stores, warehouses, or agricultural supply chains.

Future Enhancements

Although the project performs well in its current state, there are several ways to enhance and expand it further:

1. **Expand Dataset Diversity**
   Include more fruit and vegetable types, along with different lighting conditions, angles, and backgrounds to make the model more robust and generalized.

2. **Add Object Detection Capability**
   Instead of just classifying one item at a time, implement object detection (using YOLO or SSD) to identify multiple items in a single image.

3. **Integrate with IoT Devices**
   Deploy the model onto edge devices like Raspberry Pi with a camera module to automate real-time sorting systems.
4. **Improve the Web App**
   Enhance the frontend UI with drag-and-drop upload features, prediction confidence levels, and support for batch predictions.
5. **Mobile App Integration**
   Convert the web app into a mobile application using Flutter or React Native to enable fruit/vegetable classification on the go.
6. **Deploy on Cloud for Scalability**
   Host the model on cloud platforms like AWS, Azure, or Heroku for wider access, and integrate APIs for external use.