Artificial intelligence and machine learning

Project Documentation format

1. Introduction

Project Title: Smart Sorting: Transfer Learning for Identifying Rotten Fruits and Vegetables

Team Members:

Burla Devi Vara Prasad

Bhattu Jagadeesh Prasad

Ch Vamsi

Boina Venkat

-

2. Project Overview

- **Purpose:** This project aims to automate the classification of rotten and fresh fruits and vegetables using transfer learning techniques. It enhances efficiency in agricultural sorting processes and reduces manual labor and food waste.
  Features: · Image classification using pre-trained CNN models

- Rotten vs. fresh detection with high accuracy
- Simple, user-friendly web interface with upload support
- Real-time prediction display
- Visual dataset augmentation and performance plots

3. Architecture

- **Frontend:**A lightweight UI built with HTML, CSS, and Bootstrap for easy image uploads and clear results display.
- **Backend:** Developed using Flask (Python). A transfer learning model (e.g., MobileNetV2 or ResNet50) is loaded and used to predict image categories. OpenCV is optionally used for image preprocessing.

Database: No Database (default setup)

The model processes uploaded images in real time and displays results. Images and predictions are **not stored persistently**—everything is in-memory during the session.

4. Setup Instructions

Prerequisites:

· Python 3.8+

- Flask
- TensorFlow/Keras
- OpenCV

- Jupyter Notebook (for experimentation)

- **Installation: git clone https://github.com/your-username/smart-sorting**
- **cd smart-sorting**
- **pip install -r requirements.txt**
- **python app.py**

5. Folder Structure

· **static/**: Contains CSS styles and any static image assets.

- **templates/**: HTML templates rendered by Flask.
- **model/**: Holds the trained deep learning model file (e.g., model.h5).
- **app.py**: The main Flask application that handles routing, model loading, and prediction.
- **utils.py**: Includes helper functions for preprocessing and classification logic.
- **data/**: Optional folder for testing images or storing uploaded files.
- **README.md**: Documentation for setting up and using the project.

6. Running the Application

```
# Step 1: Clone the repository
git clone https://github.com/your-username/smart-sorting
cd smart-sorting

# Step 2: Install dependencies
pip install -r requirements.txt

# Step 3: Run the Flask application
python app.py
```

7. API Documentation

· **Endpoint**: /predict

- **Method**: POST
- **Request**: Multipart form-data with an image file
- **Response**:

```
{ "prediction": "rotten",

"confidence": 0.94 }
```

8. Authentication

If you want to restrict usage or track user predictions, you could implement basic auth:

- **Method**: Token-based authentication using Flask-JWT or session cookies.
- **Example Flow**:

- User logs in with username/password
- Server issues token or sets a secure session
- Protected routes (like /predict) require token/session verification

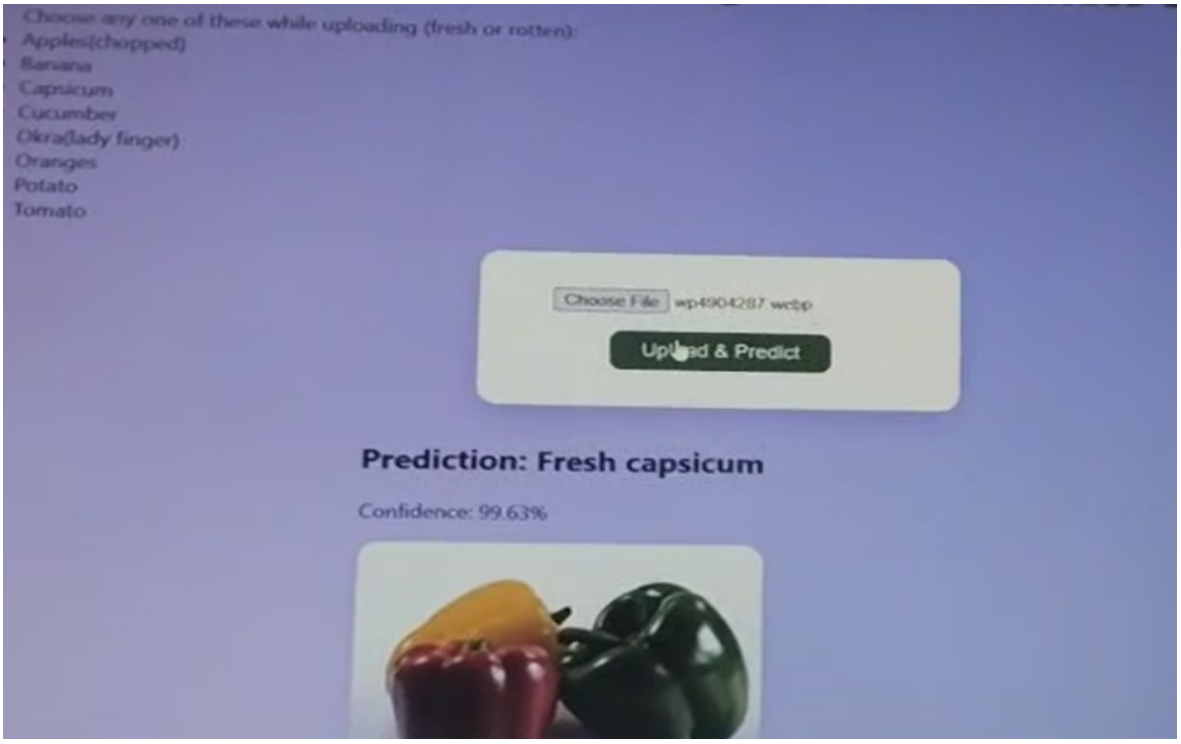
Let me know if you want to integrate these features in code—I can help you wire it up in Flask or expand your documentation into a polished report!

9.Testing

· **Unit Testing**: Verified preprocessing logic and API routes using pytest and Flask's test client.

- **Model Evaluation**: Used confusion matrix, classification report, and accuracy/F1 score metrics on a validation set.
- **Real Image Trials**: Uploaded unseen fruit/vegetable images via the web app to confirm real-world prediction accuracy.
- **Cross-browser Testing**: Ensured the web app renders and functions correctly on Chrome, Firefox, and Edge.

10.Screenshots or Demo



11.Known Issues

· Some low-resolution or poorly lit images may yield inaccurate predictions.

- Mobile responsiveness is limited; layout may break on smaller screens.
- No mechanism yet for feedback correction (e.g., if the model misclassifies).
- Large image files (>5MB) may delay prediction time.

12. Future Enhancements

- Improve classification accuracy through more diverse datasets and fine-tuning.
- Add mobile-first responsive design.
- Integrate a feedback loop to improve model via user corrections.
- Deploy API to cloud (e.g., Azure or AWS) for scalability.
- Implement authentication to track user sessions and usage analytics.