Smart Sorting: Transfer Learning for Identifying Rotten Fruits and Vegetables

Rayankula Lakshmi Chaithanya
Seshadri Rao Gudlavalleru Engineering College

June 28, 2025

Abstract

This document provides a comprehensive overview of a deep learning project focused on classifying fruits and vegetables as healthy or rotten. The project leverages transfer learning with pre-trained Convolutional Neural Networks (CNNs) and integrates the trained model into a user-friendly web application using Flask. The documentation covers dataset preparation, model training, and the functionality of the web interface.

1 Introduction

Project Title: Smart Sorting: Transfer Learning for Identifying Rotten Fruits and Vegetables
Team Members: Rayankula Lakshmi Chaithanya (Developer), Seshadri Rao Gudlavalleru Engineering College (Affiliation)

2 Project Overview

2.1 Purpose

This project aims to develop an automated system for identifying whether a fruit or vegetable is healthy or rotten using image recognition. The primary purpose is to contribute to reducing food waste by enabling early and accurate detection of spoilage at various stages of the supply chain or in household settings. By providing a quick and easy way to assess produce quality through visual inspection, this system can assist individuals and businesses in making informed decisions about food consumption, inventory management, and waste reduction. The core of the system is a deep learning model, specifically a Convolutional Neural Network (CNN), trained effectively using the technique of transfer learning. This model is seamlessly integrated into a user-friendly web application developed with Flask, making advanced Artificial Intelligence accessible and practical for everyday needs, contributing to more sustainable food practices.

2.2 Features

- **Image Classification:** The fundamental feature of this system is its ability to accurately classify uploaded images of various fruits and vegetables. Each image is categorized into one of two states: 'Healthy' or 'Rotten'. This classification is performed by a trained deep learning model that has learned to identify subtle visual cues, textures, and color changes indicative of spoilage or freshness.

- **Transfer Learning Implementation:** At the heart of the classification mechanism is the principle of transfer learning. This powerful technique utilizes pre-trained Convolutional Neural Networks (CNNs), such as VGG16 or ResNet50, which have already learned highly effective feature representations from vast general image datasets (like ImageNet). By using these models as a starting point and fine-tuning them on our specific fruit and vegetable dataset, the project achieves efficient and high-performance model training. This approach significantly reduces the computational resources and the extensive amount of domain-specific labeled data typically required for training a deep learning model from scratch.

- **Data Augmentation for Robustness:** To enhance the model's robustness and effectively prevent overfitting, a comprehensive suite of data augmentation techniques is extensively employed during the training phase. These techniques create synthetic, yet realistic, variations of the existing images by applying transformations such as random rotations (up to 20 degrees), width and height shifts (up to 20

- **Intuitive Web Interface (Flask-based):** A simple, intuitive, and user-friendly web application, built using the Flask micro-framework in Python, serves as the primary interface for users. This web interface provides a straightforward mechanism for users to easily upload images of their fruits or vegetables. Upon submission, the application processes the image and instantly displays the classification predictions, making the powerful deep learning model accessible and actionable for non-technical users.

- **Comprehensive Classification Categories:** The trained model is designed to support a wide and comprehensive range of produce types, offering classification across 28 distinct classes. This extensive coverage includes 14 different types of common fruits and vegetables, with each type having a specific 'Healthy' state and a corresponding 'Rotten' state (e.g., Apple_Healthy, Apple_Rotten; Banana_Healthy, Banana_Rotten; Bellpepper_Healthy, Bellpepper_Rotten; and so forth, covering Carrots, Cucumbers, Grapes, Guavas, Jujubes, Mangoes, Oranges, Pomegranates, Potatoes, Strawberries, and Tomatoes). This granular classification allows for precise identification of the produce's condition.

3 Architecture

The project employs a client-server architecture, where a single Python Flask application handles both the presentation layer (frontend HTML serving) and the core business logic (backend operations, including machine learning model inference and image processing).

3.1 Frontend Implementation

The frontend layer is engineered for simplicity and ease of interaction, relying on foundational web technologies.

- **HTML Templates (Jinja2):** The presentation logic is managed by Flask's integration with the Jinja2 templating engine. Specifically, the `portfolio-details.html` file serves as the primary user interface. This HTML template is dynamically rendered by the Flask backend, allowing it to inject server-side data, such as prediction results ({{ `predict` }}) and paths to uploaded images ({{ `image_path` }}), directly into the web page before it's sent to the user's browser. This provides a flexible way to present dynamic content.

- **User Interaction via Forms:** The user interaction primarily revolves around an HTML `<form>` element. Users select an image file using an `<input type="file" name="pc_image">` field. When the form is submitted via a POST request to the `/predict` endpoint, the Flask backend processes the uploaded file. While the current implementation relies heavily on backend processing, the `portfolio-details.html` could be extended with basic client-side JavaScript to provide features like immediate image preview (displaying the selected image before actual upload) or client-side form validation to enhance the user experience by providing instant feedback without a round trip to the server.

- **Styling with Standard CSS:** The visual aesthetics and responsiveness of the web interface are achieved through standard Cascading Style Sheets (CSS). These styles are typically linked within the HTML templates

```
<link rel="stylesheet" href="{{ url_for('static', filename='
    css/style.css') }}">
```

  . The CSS ensures a clean, intuitive, and responsive design that adapts well to various screen sizes (e.g., desktops, tablets, mobile phones), focusing on clarity and ease of navigation for efficient image uploads and result viewing.

3.2 Backend Implementation

The backend is a robust and efficient Flask application, written entirely in Python, serving as the central processing unit for all web requests and machine learning tasks.

- **Flask Framework Core (app.py):** The application is initialized using `app = Flask(__name__)`. Flask handles the fundamental web server operations, including defining URL routes using decorators like `@app.route('/')` for the home page and `@app.route('/predict', methods=['POST'])` for handling image submissions. It is responsible for parsing incoming HTTP requests (both GET and POST), extracting data (including file uploads via `request.files`), and orchestrating the rendering and sending of HTML responses back to the client.

- **Image Processing Pipeline (TensorFlow/Keras, NumPy, Pillow):** Upon receiving an uploaded image, the backend initiates a meticulous image processing pipeline crucial for preparing the image for the deep learning model. It leverages:

    1. `load_img(img_path, target_size=(224, 224))`:
        - Source: `tensorflow.keras.preprocessing.image`
        - Function: Loads image from path and resizes to 224×224 pixels
        - Purpose: Standard input size for VGG16/ResNet models
    2. `img_to_array(img)`:
        - Converts Pillow image → NumPy array
        - Output shape: (224, 224, 3) for RGB images
    3. `preprocess_input(x)`:
        - Source: `tensorflow.keras.applications.vgg16`
        - Operations:
            * Pixel scaling (-1 to 1 range)
            * ImageNet mean subtraction
        - Ensures compatibility with pretrained weights
    4. `np.expand_dims(x, axis=0)`:
        - Input shape: (224, 224, 3)
        - Output shape: (1, 224, 224, 3)
        - Purpose: Adds batch dimension for model input

- **Machine Learning Model Inference (healthy_vs_rotten.h5):** The core of the prediction happens when the `healthy_vs_rotten.h5` Keras model is loaded into memory via `tf.keras.models.load_model()`. This model represents the fine-tuned deep learning classifier. Once loaded, the preprocessed image data is passed to it for inference using `model.predict(x)`. The output of this prediction is typically a probability distribution over the classes. `np.argmax(pred, axis=1)` is then used to extract the index of the class with the highest predicted probability, which corresponds to the model's most confident classification.

- **Temporary File Handling (static/uploads):** The backend manages temporary storage of uploaded image files. When a file is received via `request.files`, it is saved to a designated `static/uploads` directory using `file.save(img_path)`. The `os.makedirs(upload_dir, exist_ok=True)` command ensures that this directory exists before any files are written to it, preventing common `FileNotFoundError` issues. The path to this temporarily saved image is then passed to the HTML template to allow the browser to display the uploaded image alongside the prediction. Error handling with `try-except` blocks is implemented to gracefully manage issues during image loading, processing, or prediction.

3.3 Database Implementation

This project's design prioritizes demonstrating the machine learning classification functionality, and as such, it **does not integrate a traditional persistent database system** (such as MongoDB, SQL databases like PostgreSQL/MySQL, or cloud-based solutions like Firestore). The data management is handled through simpler, in-memory or file-system-based mechanisms.

- **File System Storage for Ephemeral Data:** Uploaded images are stored temporarily on the local file system within the `static/uploads` directory. This is managed by standard Python file operations (`file.save()`) and `os` module functions (`os.makedirs()`, `os.path.join()`). This storage is inherently ephemeral for the purpose of immediate processing and displaying the results. These files are not indexed or managed by a database and are not designed for long-term archival, querying historical predictions, or complex data relationships across application restarts. They serve as transient storage for the web server's static assets.

- **In-Memory Class Labels (CLASS_INDEX):** The mapping of numerical prediction indices (output by the machine learning model) to their corresponding human-readable class labels (e.g., 'Apple_Healthy', 'Banana_Rotten', 'Pomegranate_Healthy') is maintained as a static Python list named `CLASS_INDEX`. This list is hardcoded directly within the `app.py` script. It is loaded into the application's memory when the Flask server starts. This approach is suitable for a fixed set of classes and removes the overhead and complexity of managing a separate database for label lookup. If the number of classes were dynamic or managed externally, a database would be necessary.

The absence of a persistent database simplifies the project setup and deployment, aligning with its core objective of showcasing ML inference. For applications requiring user management, historical data logging, or complex data relationships, a dedicated database would be a necessary addition.

4 Setup Instructions

To set up and run this project locally, enabling both the deep learning model and the Flask web interface, follow these detailed instructions. This guide covers all necessary software installations and project-specific configurations.

4.1 Prerequisites

Before proceeding with the project installation, ensure your system meets the following software dependencies. These are fundamental for running Python applications and managing their respective libraries.

- **Python 3.x:** The entire project, encompassing the Flask web application, the machine learning model training script (`Tranfer_Learning_Fruits_Vegs.ipynb`), and the inference logic (`app.py`), is developed and runs exclusively on Python 3.x. It is strongly recommended to use Python 3.8 or a newer version to ensure compatibility with the latest features and versions of libraries like TensorFlow. You can download Python from its official website: `https://www.python.org/downloads/`.
- **pip:** This is the standard package-management system used to install and manage software packages written in Python. pip is typically included by default with most modern Python installations. You can verify its installation by running `pip --version` in your terminal.
- **Git:** A widely used distributed version control system. Git is essential for cloning the project repository from a remote source (such as GitHub, GitLab, or Bitbucket) to your local machine. If you do not have Git installed, you can download it from `https://git-scm.com/downloads`.
- **Required Python Libraries:** These specific Python libraries are fundamental for the project's operation and must be installed within your Python environment. They will be installed collectively using pip.
  - `Flask`: The lightweight Python micro-web framework used to build the backend server and handle web requests.
  - `tensorflow`: The comprehensive open-source machine learning platform developed by Google. This library is crucial for loading, running, and potentially training the deep learning model. For systems without a compatible GPU, `tensorflow-cpu` can be installed instead to avoid GPU-specific dependencies and potential installation complexities.
  - `numpy`: A foundational package for scientific computing in Python. It provides powerful array objects and tools for working with numerical data, essential for processing image pixel data as arrays.
  - `Pillow` (`PIL`): A friendly fork of the original Python Imaging Library. It is used by Keras's image preprocessing utilities (`load_img`,

`img_to_array`) to handle various image file formats and perform basic image manipulations.

  - `scikit-learn`: While the core model training uses TensorFlow, scikit-learn is specifically required for the `train_test_split` utility. This function is used within the `Tranfer_Learning_Fruits_Vegs.ipynb` notebook to divide the dataset into training, validation, and test subsets.

4.2 Installation

Follow these step-by-step instructions to get the project installed and ready for execution on your local development machine.

1. **Clone the repository:** Begin by obtaining all the project files. Open your terminal or command prompt and use the `git clone` command, replacing `<repository_url>` with the actual URL of your project's Git repository. After cloning, navigate into the project's root directory.

```
git clone <repository_url>
cd <repository_name>

```

2. **Create a virtual environment (recommended):** It is highly recommended to set up a Python virtual environment for this project. A virtual environment creates an isolated space for your project's Python dependencies, preventing conflicts with other Python projects or your system's global Python packages.

```
python -m venv venv

# On Windows, activate with:
.\venv\Scripts\activate

# On macOS/Linux, activate with:
source venv/bin/activate

```

   Once activated, your terminal prompt will typically show `(venv)` indicating you are in the virtual environment.

3. **Install dependencies:** With your virtual environment activated, use pip to install all the necessary Python libraries listed in the Prerequisites section. This command will fetch and install all required packages into your isolated virtual environment.

```
pip install Flask tensorflow numpy Pillow scikit-learn

```

4. **Obtain the trained model (`healthy_vs_rotten.h5`):** The Flask application relies on a pre-trained Keras model file to perform classifications.

Ensure that the `healthy_vs_rotten.h5` model file is physically present and accessible directly in the root directory of your project (i.e., in the same directory where `app.py` is located). This `.h5` file is the output generated after successfully executing the model training process detailed in the `Tranfer_Learning_Fruits_Vegs.ipynb` Jupyter notebook. If you haven't trained the model yourself, you will need to acquire this file.

5. **Dataset Setup (for re-training/data exploration):** This step is only necessary if you intend to re-run the `Tranfer_Learning_Fruits_Vegs.ipynb` Jupyter notebook to train the model from scratch or to explore the dataset's structure. In this case, ensure your raw image dataset is organized in a specific folder structure: `data/Fruit and Vegetable Diseases Dataset/<Class Name>/<Image Files>`. If your dataset adheres to a different organizational scheme, you will need to modify the data loading paths and configurations within the Jupyter notebook accordingly to match your setup.

5 Folder Structure

The project adheres to a clear, organized, and modular folder structure. This layout is typical for Flask applications that integrate machine learning components, facilitating logical separation of concerns, ease of development, maintenance, and potential deployment.

- `/`: This represents the project's root directory, which is the top-level folder of your cloned repository.
  - `app.py`: This is the central Python script that defines and launches the Flask web application. It encapsulates the core logic for routing web requests, loading the machine learning model, processing user-uploaded images, and rendering dynamic HTML responses to the client.
  - `healthy_vs_rotten.h5`: This file serves as the container for the trained Keras deep learning model. It holds the model's architecture, weights, and optimizer state, making it the fundamental component responsible for performing fruit and vegetable classification.
  - `Tranfer_Learning_Fruits_Vegs.ipynb`: This Jupyter notebook contains the detailed Python code for the entire machine learning pipeline. This includes steps such as preparing the dataset (splitting into training, validation, and test sets), implementing data augmentation techniques, constructing and compiling the deep learning model (e.g., using VGG16 or ResNet50 as a base), training the model, and initial evaluation of its performance.
  - `static/`: This directory is a standard Flask convention for serving static files directly to the web browser. Files placed here are not processed by Python logic before being sent.

    * `uploads/`: A crucial sub-directory within `static/`. This is the designated location where all image files uploaded by users through the web interface are temporarily saved on the server's file system for subsequent processing by the Flask backend and the machine learning model.
    * `css/`: Contains all Cascading Style Sheets (CSS) files. These files define the visual presentation, layout, and styling rules for the web pages, ensuring a consistent look and feel.
    * `img/`: Stores any static image assets that are part of the web interface design itself. This might include logos, icons, background images, or any other images that are not dynamically uploaded by users but are part of the application's fixed visual elements.
    * `js/`: Contains JavaScript files. These scripts handle client-side interactivity, enhancing the user experience directly within the web browser without requiring constant server communication.
  - `templates/`: This directory is another standard Flask convention, holding all HTML template files. Flask uses these templates (powered by Jinja2) to render dynamic web pages by inserting data from the server.
    * `portfolio-details.html`: This is the primary HTML template that defines the structure and content of the user interface for image uploading. It also serves as the page where the classification results are displayed dynamically after a prediction is made.
    * `error.html`: A simple HTML template designed to display user-friendly error messages. This page is rendered when an unexpected issue or an invalid request occurs on the backend, providing feedback to the user.
  - `data/`: (Optional) This directory typically houses the raw, unprocessed image dataset used for training the machine learning model. If you intend to run the training notebook (`Tranfer_Learning_Fruits_Vegs.ipynb`), your dataset should ideally be organized here, following a structure like `data/Fruit and Vegetable Diseases Dataset/<Class Name>/<Image Files>`.
  - `output_dataset/`: (Optional) This directory is automatically created and populated when the data splitting and preprocessing steps within the `Tranfer_Learning_Fruits_Vegs.ipynb` notebook are executed. It contains the organized subsets of the dataset, specifically the training, validation, and test sets, ready for model training and evaluation.

6 Running the Application

To launch and interact with the Flask web application on your local development machine, enabling both the user interface and the backend machine learning inference, follow these simple, command-line based instructions:

1. **Activate your virtual environment:** This is a critical first step. If you followed the recommended setup instructions, you would have created a Python virtual environment. Activating it ensures that the application runs with its specific, isolated set of dependencies, preventing potential conflicts with other Python projects or your system's global Python installation.

   ```
   # On Windows, execute:
   .\venv\Scripts\activate

   # On macOS/Linux, execute:
   source venv/bin/activate

   ```

   Upon successful activation, your terminal prompt will typically change (e.g., prepend `(venv)`) to indicate that you are operating within the virtual environment.

2. **Navigate to the project root directory:** Using your terminal or command prompt, change your current working directory to the project's root folder. This is the directory where the `app.py` file is located.

3. **Run the Flask application:** Execute the `app.py` script directly using the Python interpreter. This command will initiate the Flask development server.

   ```
   python app.py

   ```

   Once executed, the terminal will display messages indicating that the Flask server is starting up. It will typically show the address where the application is hosted.

4. **Access the application in your web browser:** After the server has successfully started, you will see output similar to `* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)` in your terminal. Copy this URL (commonly `http://127.0.0.1:5000`, which refers to your local machine on port 5000) and paste it into the address bar of your preferred web browser. Press Enter, and the web application's user interface will load, ready for interaction.

7 API Documentation

The Flask backend for the "Smart Sorting" project exposes a single, yet crucial, API endpoint dedicated to the image classification service. This section provides detailed documentation for this endpoint, including its functionality, expected input parameters, and the structure of its typical responses.

7.1 /predict

This endpoint serves as the interface for users to submit images for classification. It orchestrates the entire backend process from image reception to prediction output.

- **Method:** `POST`
- **Description:** This endpoint is specifically designed to accept an image file uploaded as part of a multi-part form data request. Upon receiving the image, the backend performs a series of critical operations. These include:

    1. **Image Preprocessing:** The raw uploaded image is meticulously prepared for the deep learning model. This involves resizing the image to the model's required input dimensions (224x224 pixels), converting its pixel data into a numerical NumPy array, and applying specific normalization or scaling (`preprocess_input`) that aligns with how the pre-trained CNN was originally trained. A batch dimension is also added to the image array.
    2. **Model Inference:** The preprocessed image is then fed as input to the loaded TensorFlow/Keras deep learning model (`healthy_vs_rotten.h5`). The model processes the image and generates a prediction, typically in the form of probability scores for each possible class.
    3. **Class Label Extraction:** From the model's raw output, the index corresponding to the highest probability score is identified using `np.argmax`. This numerical index is then mapped to its corresponding human-readable class label (e.g., "Apple_Healthy", "Banana_Rotten") using the predefined `CLASS_INDEX` list.

    Finally, instead of returning raw JSON, the backend renders an HTML page (`portfolio-details.html`) that dynamically displays the prediction result along with the uploaded image, providing a direct visual feedback loop to the user.

- **Parameters (Form Data):**
    - **Parameter Name:** `pc_image`
    - **Type:** File
    - **Description:** This parameter represents the image file that the user wishes to classify. It must be sent as part of a `multipart/form-data` request. The Flask backend accesses this file through `request.files['pc_image']`. Accepted image formats typically include JPEG, PNG, etc., as supported by the Pillow library.
- **Example Request (Conceptual - via HTML form):** Users typically interact with this endpoint through a standard HTML form. The form's `action` attribute is set to `/predict`, its `method` is `POST`, and crucially, its `enctype` is set to `multipart/form-data` to correctly handle file uploads.

```html
<form action="/predict" method="post" enctype="multipart/form-
    data">
  <p>Select image to upload:</p>
  <input type="file" name="pc_image" id="pc_image" accept="
    image/*">
  <input type="submit" value="Upload Image" name="submit">
</form>

```

Listing 1: Example HTML Form Structure (Conceptual from portfolio-details.html)

8 Authentication

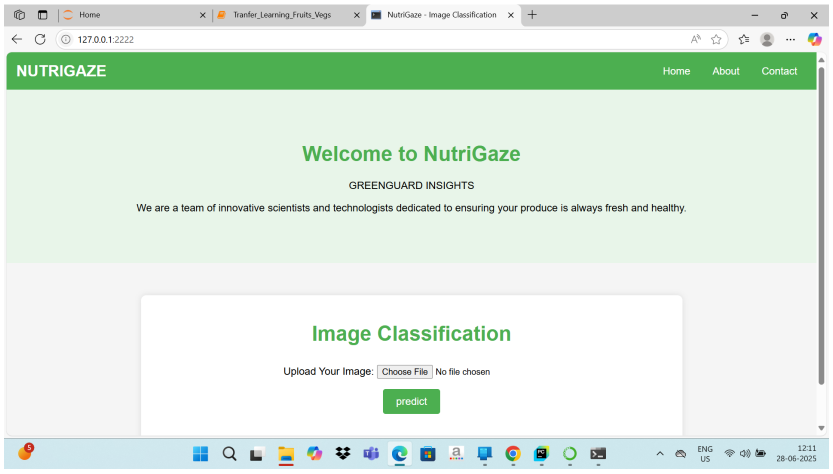
This project is primarily conceptualized and developed as a demonstration of a machine learning model's capability in image classification. Consequently, it is designed with simplicity in mind and **does not incorporate any user authentication or authorization mechanisms**.

- **No User Accounts or Profiles:** The application does not include any functionality for user registration, user login, or the creation and management of user profiles. There are no provisions to store or retrieve user-specific data.
- **Public Accessibility:** All functionalities, specifically the image upload and classification prediction, are publicly accessible. Any user can interact with the system without needing to provide any form of credentials, log in, or register. This design choice significantly simplifies the setup, deployment, and immediate usability of the application for demonstration and rapid prototyping purposes, allowing focus on the core ML task.
- **Absence of Tokens and Sessions:** Given the lack of user accounts, the project does not implement or manage any form of user sessions, authentication tokens (such as JSON Web Tokens - JWTs), or other access control methods tied to individual user identities. The application operates in a stateless manner regarding user identity.

In summary, this straightforward approach is adopted to keep the focus squarely on the deep learning classification task without adding the complexity of a secure user management system. For any future expansion into production environments or applications that require user-specific data security, personalization, or restricted access, a robust authentication and authorization system would be an indispensable enhancement.

9 User Interface

The user interface (UI) of the "Smart Sorting" web application is meticulously designed for clarity, simplicity, and ease of use. It allows users to effortlessly upload an image of a fruit or vegetable and immediately view the classification result. The UI is built using standard HTML and is styled with CSS to provide a clean and intuitive experience, with all pages served directly by the Flask backend application.


NUTRIGAZE

Home About Contact

Welcome to NutriGaze

GREENGUARD INSIGHTS

We are a team of innovative scientists and technologists dedicated to ensuring your produce is always fresh and healthy.

Image Classification

Upload Your Image: Choose File No file chosen

predict

Figure 1: The main interface of the NutriGaze web application. This screen presents the primary image classification upload section. Users interact with a "Choose File" button to select an image from their local device and then initiate the prediction process by clicking the 'predict' button. The interface features a prominent title "NutriGaze" and a welcoming introductory text, providing a clear call to action.

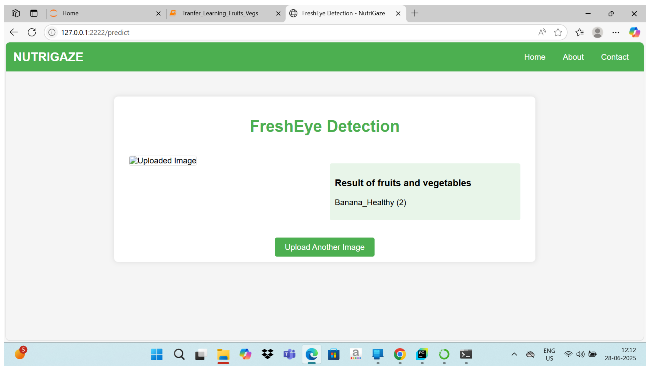


Figure 2: Prediction display after image upload. Following a successful image upload and backend classification, this screen is displayed. It prominently shows the uploaded image on the left. On the right, the classification result is clearly indicated (e.g., "Banana_Healthy (2)"). A user-friendly "Upload Another Image" button is provided to facilitate continuous interaction and further classifications.