Implementation Documentation

Crop Disease Detection using CNNs

Team Members:

- Ahmed Sameer Hashemi (66026)
- Ayaan Waheed (66344)

Presented to: Mr. Junaid Khan **Course:** Artificial Intelligence

---

1. Project Overview

This document outlines the technical implementation of the **Tomato Leaf Disease Detection System**. The software is designed to accept an image of a tomato leaf as input, process it through a Deep Learning model, and output the specific disease diagnosis along with a confidence score.

The system utilizes **Transfer Learning** based on the **EfficientNetB0** architecture to achieve high accuracy with a relatively small training dataset.

2. System Workflow

The overall working of the program follows a linear pipeline:

1. **Data Acquisition:** The system automatically downloads the "Tomato Leaf" dataset from Kaggle using the kagglehub library.
2. **Preprocessing:**
    - Images are resized to **224x224** pixels.
    - Pixel values are normalized using the specific preprocess_input function required by EfficientNet.
    - Data Augmentation (rotation, zoom, flips) is applied to training data to improve generalization.
3. **Model Inference:** The processed image is passed through the Convolutional Neural Network (CNN).

4. **Classification:** The final Dense layer (Softmax) calculates probabilities for all 10 classes (diseases).
5. **Output:** The class with the highest probability is displayed to the user.

3. Technical Stack & Libraries

The project is implemented in **Python** using the **Google Colab** environment. The following key libraries are used:

- **TensorFlow & Keras (2.x):** The core framework used for building, training, and saving the Deep Learning model.
- **Kagglehub:** Used for seamless, programmatic downloading of the dataset directly into the Colab environment.
- **NumPy:** Used for numerical array operations and image matrix manipulation.
- **Matplotlib & Seaborn:** Used for visualizing training performance (Accuracy/Loss graphs) and the Confusion Matrix.
- **Scikit-Learn:** Used to calculate evaluation metrics and generate the classification report.

4. Model Architecture Details

We utilized **EfficientNetB0** as the base model due to its balance of accuracy and computational efficiency.

- **Base Model:** EfficientNetB0 (Pre-trained on ImageNet).
- **Modifications:**
    - The top classification layer of EfficientNet was removed (include_top=False).
    - **Fine-Tuning:** The top 20 layers of the base model were unfrozen to allow the model to learn specific features of tomato leaves.
- **Custom Head:**
    - GlobalAveragePooling2D: Reduces spatial dimensions.
    - Dropout (0.5): Prevents overfitting by randomly deactivating neurons during training.
    - Dense (Output): 10 neurons with **Softmax** activation for multi-class classification.

5. Input and Output Formats

Input:

- **Format:** Digital Image (.jpg, .png, or .jpeg).
- **Content:** A close-up view of a tomato leaf (healthy or diseased).

Output:

- **Visual:** The uploaded image is displayed on the screen.
- **Text:** The predicted disease name (e.g., *Tomato_Bacterial_spot*).
- **Metric:** A confidence percentage (e.g., *98.5%*).

6. Execution Instructions

To run this project, follow these steps:

Step 1: Environment Setup

1. Open the provided .ipynb file in **Google Colab** (https://colab.research.google.com).
2. Go to **Runtime > Change runtime type** and select **T4 GPU**. This is critical for training speed.

Step 2: Dependencies & Data

1. Run the first cell to install kagglehub and import libraries.
2. Run the second cell. The system will automatically download the dataset. *Note: No manual API key upload is required with the updated code.*

Step 3: Training

1. Run the cells sequentially.
2. **Cell 6 (Training)** will take approximately 15–20 minutes to complete.
3. Once finished, the model saves automatically as best_tomato_model.keras.

Step 4: Testing & Prediction

1. Scroll to the final cell (Prediction).
2. Run the cell. A "Choose Files" button will appear.
3. Upload a clear image of a tomato leaf.
4. The system will process the image and print the result below the button.

7. Troubleshooting

- **Error: "Name 'model' is not defined":** This occurs if you run the prediction cell without running the training cell first. Ensure you run **Cell 5 (Build Model)** and **Cell 6 (Training)** before testing.
- **Low Accuracy:** Ensure preprocess_input is being used in the data generators. If validation accuracy is low, try increasing the number of epochs or the dropout rate.