



LeafGuard

Transforming Big Data with Data Science

Abstract

The LeafGuard project leverages Big Data processing and Data Science to develop an efficient and scalable application for detecting leaf diseases in crops. By utilizing machine learning models and image analysis, the system provides farmers with an accurate, cost-effective, and timely solution for identifying plant diseases, ultimately supporting sustainable agricultural practices.

Project Report

# Table of Contents

[Table of Contents 1](#_Toc177724592)

[**1** **Problem Statement:** 3](#_Toc177724593)

[**2** **Design Specifications** 3](#_Toc177724594)

[2.1 System Architecture 3](#_Toc177724595)

[2.1.1 Data Collection and Ingestion 3](#_Toc177724596)

[2.1.2 Data Pre-processing and Feature Extraction 3](#_Toc177724597)

[2.1.3 Data Management and Version Control 3](#_Toc177724598)

[2.1.4 Model Training and Evaluation 4](#_Toc177724599)

[2.1.5 Disease Prediction 4](#_Toc177724600)

[2.1.6 UserInterface(UI) and Reporting 4](#_Toc177724601)

[2.2 Technologies 5](#_Toc177724602)

[2.3 System Requirements 5](#_Toc177724603)

[2.3.1 Hardware Requirements 5](#_Toc177724604)

[2.3.2 Software Requirements 5](#_Toc177724605)

[2.4 Success Criteria 5](#_Toc177724606)

[**3** **User Flow Diagram** 6](#_Toc177724607)

[**4** **Test Data Used in the Project** 7](#_Toc177724608)

[4.1 Leaf Image Dataset 7](#_Toc177724609)

[4.1.1 Sample Datasets: 7](#_Toc177724610)

[4.1.2 Test Data Example: 8](#_Toc177724611)

[4.2 Disease Labels 8](#_Toc177724612)

[4.2.1 Sample Disease Labels: 8](#_Toc177724613)

[4.3 User Data (forTestingFunctionality) 11](#_Toc177724614)

[4.4 System Outputs 11](#_Toc177724615)

[4.4.1 Sample Outputs: 11](#_Toc177724616)

[4.5 Treatment Recommendations Data 12](#_Toc177724617)

[4.5.1 Sample Treatment Data: 12](#_Toc177724618)

[4.6 Feedback Data 13](#_Toc177724619)

[4.6.1 Sample Feedback Data: 13](#_Toc177724620)

[**5** **Test Data Used in the Project** 13](#_Toc177724621)

[5.1 Data Description: 13](#_Toc177724622)

[5.2 Sample Test Data Labels: 13](#_Toc177724623)

[5.3 Data Preprocessing: 14](#_Toc177724624)

[5.4 Evaluation Metrics: 14](#_Toc177724625)

[**6** **Project Installation Instructions** 14](#_Toc177724626)

[6.1 System Requirements: 14](#_Toc177724627)

[6.2 Installation 15](#_Toc177724628)

[**7** **Link of GitHub** 17](#_Toc177724629)

[**8** **Steps to Execute the Project** 17](#_Toc177724630)

[**9** **Blog Link** 18](#_Toc177724631)

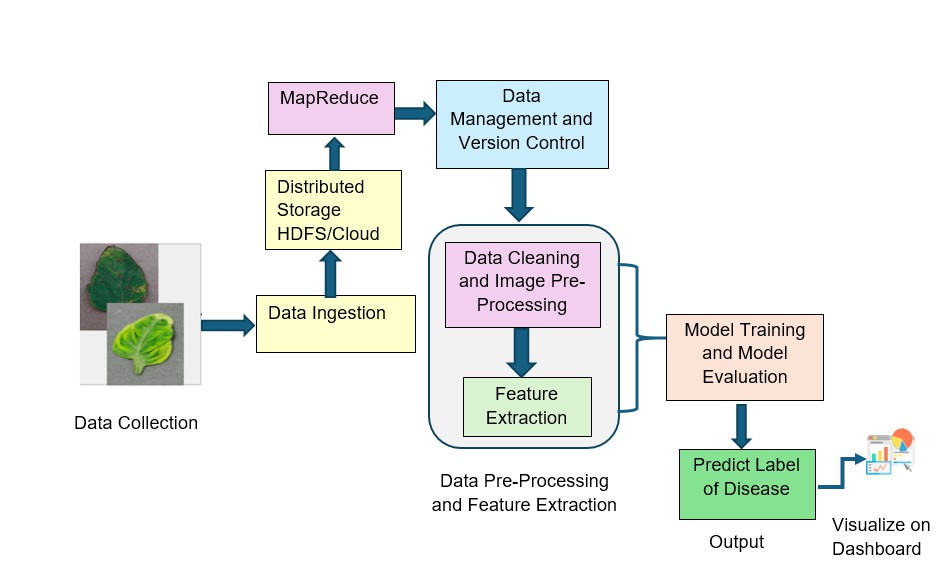
# **Problem Statement:**

The traditional method of detecting leaf diseases through manual inspection is time-consuming, labor-intensive, and inefficient, especially for large-scale farming operations. As global demand for agricultural productivity grows, driven by population increases and resource constraints, it becomes crucial to find more efficient methods of disease detection. Manual inspection is often prone to human error and requires significant labor, making it impractical for large farms where timely and accurate diagnosis is essential to prevent crop losses. With the rise of Data Science and Big Data processing, a promising solution emerges by utilizing data from various sources such as high-resolution images, weather patterns, and soil conditions. By analyzing these data streams, machine learning models can automate disease detection with greater accuracy and speed, helping to predict potential outbreaks and prevent widespread damage. This approach not only reduces the dependency on labor but also enhances resource optimization, improving agricultural yields while minimizing the use of pesticides and other inputs. Ultimately, leveraging these technologies can significantly reduce crop losses, increase productivity, and improve global food security, making agriculture more sustainable and efficient.

# **Design Specifications**

## System Architecture

LeafGuard employs a multi-stage architecture to achieve accurate disease detection:



### Data Collection and Ingestion

* Users upload high-resolution images of potentially diseased leaves.
* Alternatively, the system can integrate with external data sources (e.g., Kaggle) for training data.
* Images are stored in a distributed storage system like HDFS or cloud storage for scalability and accessibility.

### Data Pre-processing and Feature Extraction

* Images undergo pre-processing to eliminate noise, standardize formats, and enhance quality.
* Techniques like filtering, normalization, and resizing are employed for consistency.
* Feature extraction techniques like color analysis, texture analysis, and edge detection are used to identify key characteristics for disease classification.

### Data Management and Version Control

* Data management tools track and manage different data versions, ensuring reproducibility and consistency.
* Techniques like MapReduce are implemented for efficient processing and handling large datasets.

### Model Training and Evaluation

* **Pre-processing:** The pre-processed data is used to train machine learning models for disease classification. The data undergoes steps like resizing, normalization, and augmentation to enhance the training process.
* **Model Used:** The **ResNet-50** model (Residual Networks with 50 layers) is employed as the primary Convolutional Neural Network (CNN) for image classification. ResNet-50 is chosen for its ability to efficiently handle large-scale image recognition tasks while preventing the vanishing gradient problem through residual learning.
* **Alternative Models:** Along with ResNet-50, other machine learning models such as decision trees may be explored to handle contextual or tabular data related to plant health.
* **Cross-validation:** Cross-validation techniques, such as k-fold validation, are used during training to ensure model robustness and to prevent overfitting, allowing the model to generalize well to unseen data.
* **Model Evaluation:** After training, the model's performance is evaluated using standard metrics such as:
  + **Accuracy**
  + **Precision**
  + **Recall**
  + **F1 Score**

These metrics help determine the model's effectiveness in detecting and classifying leaf diseases. ResNet-50's architecture is particularly suited for extracting features from leaf images, leading to more accurate classification results.

### Disease Prediction

* New user-uploaded images are pre-processed and fed into the trained models.
* Models predict the presence and type of disease with corresponding confidence levels.

### UserInterface(UI) and Reporting

* A user-friendly interface allows farmers to upload images and access results.
* The UI displays the predicted disease type, confidence level, and recommended actions (e.g., treatment options).
* The system generates reports on disease occurrences, trends, and patterns over time.
* Visualizations like heatmaps and charts help users understand the spread and impact of diseases.

## Technologies

* **Data Storage:** HDFS, Apache HBase, or CSV
* **Backend:** Apache Spark or Apache Hive
* **Programming Language and IDE:** Python 3.11.4 or higher, Jupyter Notebook, Anaconda, or Google Colab
* **Libraries:** OpenCV, TensorFlow, scikit-learn, Pandas, NumPy, PyTorch, Matplotlib, and Seaborn
* **Visualization Tool:** Tableau Desktop

## System Requirements

### Hardware Requirements

* Processor: Intel Core i5/i7 or higher
* RAM: 8 GB or higher
* Display: Color SVGA
* Hard Disk Space: 500 GB
* Mouse and Keyboard

### Software Requirements

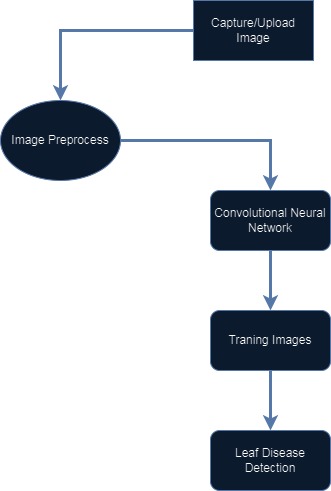
The system requires installation of the specified technologies mentioned above. Users should have access to a suitable environment for running the application.

## Success Criteria

The success of LeafGuard will be measured based on the following criteria:

* **Accuracy:** The application should achieve high accuracy in disease detection, exceeding a pre-defined threshold based on benchmark datasets.
* **Scalability:** The system should efficiently handle large volumes of image data and accommodate a growing user base.
* **Usability:** The user interface should be intuitive and user-friendly, catering to farmers with varying technical backgrounds.
* **Robustness:** The application should maintain consistent performance under diverse environmental conditions and with different plant species.
* **Reliability:** The system should consistently deliver accurate results and function reliably in various settings.

# **User Flow Diagram**



# **Test Data Used in the Project**

In the **LeafGuard** application, test data is essential for evaluating the accuracy of the machine learning model, functionality of the system, and ensuring that all components are working as expected. Below is a breakdown of the various types of test data that could be used in this project.

## Leaf Image Dataset

The core of the LeafGuard application is its ability to identify diseases from leaf images. To train and test the machine learning model, a large dataset of labeled leaf images is required. This dataset would typically include:

* **Healthy Leaves**: Images of leaves without any diseases.
* **Diseased Leaves**: Images showing symptoms of different diseases (e.g., blight, rust, mildew, bacterial infections).

### Sample Datasets:

* **PlantVillage Dataset**: A popular public dataset for plant disease classification. It includes thousands of labeled images of leaves from various plants with different diseases.
* **Custom Dataset**: Images collected specifically for the project from local plants or farms.

### Tomato Leaves Royalty-Free Images, Stock Photos & Pictures | ShutterstockTest Data Example:

* **Healthy Leaf (Tomato)**:
  + Label: Healthy
  + Image Dimensions: 256x256 pixels
  + Format: JPG/PNG
* **Diseased Leaf (Tomato, Bacterial Spot)**:
  + Label: Bacterial Spot
  + Image Dimensions: 256x256 pixels
  + Format: JPG/PNG

This data is split into **training** and **testing** sets to ensure that the model generalizes well to unseen images.

## Disease Labels

The images in the dataset need to be labeled with the corresponding disease or "healthy" status. For testing, the labels can be used to evaluate the accuracy of the model's predictions.

### Sample Disease Labels:

#### Tomato Leaf Diseases:

* + Bacterial Spot



* + Early Blight



* + Late Blight



* + Leaf Mold



* + Septoria Leaf Spot



* + Target Spot



* + Yellow Leaf Curl Virus



* + Healthy



Each image in the test set would have a corresponding label, and the system's prediction can be compared against these labels to compute accuracy, precision, recall, etc.

## User Data (forTestingFunctionality)

To test the complete functionality of the app, you would also need to test how user data is handled within the system, from registration to diagnosis.

|  |  |
| --- | --- |
| Field | Sample Value |
| Leaf Image | leaf\_bacterial\_spot.jpg |

## System Outputs

After processing the input images, the system will produce outputs such as disease reports and treatment suggestions. These outputs need to be compared against expected results to ensure the system is working correctly.

### Sample Outputs:

* **Input Image**: Leaf Image with Bacterial Spot



* **Expected Output**:
  + Disease Detected: Bacterial Spot
  + Confidence: 95%
  + Suggested Treatment: "Use copper-based fungicides and avoid overhead irrigation."
* **Input Image**: Healthy Leaf



* **Expected Output**:
  + Disease Detected: None
  + Confidence: 98%
  + Suggested Treatment: "No action needed. The leaf is healthy."

## **Treatment Recommendations Data**

The system needs to recommend treatments based on the detected disease. To test this feature, a set of treatments for each disease must be predefined.

### Sample Treatment Data:

|  |  |
| --- | --- |
| Disease | Treatment Suggestions |
| Bacterial Spot | Use copper-based fungicides, avoid overhead irrigation. |
| Late Blight | Use fungicides containing chlorothalonil, improve air circulation around plants. |
| Leaf Mold | Use sulfur or potassium bicarbonate-based fungicides, prune infected leaves. |

## **Feedback** Data

After a user has interacted with the app and received disease diagnosis and treatment suggestions, they may provide feedback on the recommendations. This feedback will help in testing how the app stores and processes user feedback.

### Sample Feedback Data:

|  |  |  |
| --- | --- | --- |
| User | Diagnosis | Feedback |
| testuser01 | Bacterial Spot | "The recommendation helped control the disease." |
| testuser02 | Healthy Leaf | "No issues found, everything was accurate." |

# **Test Data Used in the Project**

The test data in the LeafGuard project consists of images of various leaves infected with different diseases, along with healthy leaves, to evaluate the model’s ability to correctly identify and classify leaf diseases. The following are details about the test data:

## Data Description:

* **Dataset Format:** The test data is comprised of images in .jpg or .png format.
* **Number of Classes:** The dataset includes multiple classes of leaf conditions, including various types of diseases and healthy leaves.
* **Image Size:** Images are resized to a consistent size for the neural network, typically 224x224 pixels.
* **Number of Test Images:** The test data contains around 20% of the entire dataset, which has been split into training and test sets.

## Sample Test Data Labels:

* **Healthy Leaf**
* **Leaf Blight**
* **Leaf Curl**
* **Leaf Spot**
* **Powdery Mildew**
* **Rust**

## Data Preprocessing:

Before testing, the images are preprocessed:

* **Resizing:** All images are resized to a uniform shape (e.g., 224x224 pixels) to ensure compatibility with the Convolutional Neural Network (CNN).
* **Normalization:** Pixel values are normalized to a scale of [0, 1] by dividing by 255 to improve training and testing efficiency.
* **Data Augmentation (Optional):** Techniques like flipping, rotating, and zooming are applied during training, though the test data remains untouched to simulate real-world conditions.

## Evaluation Metrics:

The performance of the model on the test data is measured using the following metrics:

* **Accuracy**
* **Precision**
* **Recall**
* **F1 Score**

This test data is essential for evaluating the LeafGuard model’s generalization ability and performance on unseen data.

# **Project Installation Instructions**

## System **Requirements**:

* **Hardware**:
  + Intel Core i5/i7 Processor or higher
  + 8 GB RAM or higher
  + 500 GB Hard Disk space
  + SVGA color monitor
* **Software**:
  + OS: Windows/Linux/MacOS
  + Python 3.11.4 or higher
  + Anaconda 23.1.0 or higher (Optional)
  + Jupyter Notebook or Google Colab (for running Python notebooks)

## Installation

* **Install Jupyter Notebook:**

pip install opencv-python

You can install Jupyter using pip or conda. For pip:

pip install notebook

To start Jupyter Notebook:

jupyter notebook

* **Install TensorFlow:**

TensorFlow is required for building the Convolutional Neural Network (CNN). You can install it using pip:

pip install tensorflow

* **Install Pandas:**

Pandas is used for data manipulation and analysis:

pip install pandas

* **Install NumPy:**

NumPy is required for numerical operations and handling multi-dimensional arrays:

Install OpenCV (Optional, for Image Preprocessing):

pip install numpy

OpenCV is used for image processing tasks:

* **Install Matplotlib (Optional, for Visualization):**

Matplotlib is useful for visualizing data and results:

pip install matplotlib

* **Install Scikit-learn (Optional, for Preprocessing or Metrics):**

Scikit-learn may be used for data preprocessing or evaluating models:

pip install scikit-learn

* **Install Keras (Optional, if not included with TensorFlow):**

Keras is included within TensorFlow but can be installed separately if needed:

pip install keras

* **Install JupyterLab Extensions (Optional):**

If you're using JupyterLab and want additional functionality, install useful extensions:

pip install jupyterlab

Verify Installation: After installing these libraries, you can verify the installation by running the following in a Jupyter Notebook:

import tensorflow as tf

import pandas as pd

import numpy as np

import cv2

import matplotlib.pyplot as plt

from sklearn.model\_selection import train\_test\_splitbash

# **Link of GitHub**

# **Steps to Execute the Project**

**Step 1: Download the Project**

* Clone the LeafGuard project repository from GitHub:

git clone https://github.com/your-username/LeafGuard.git

* Navigate into the project directory:

cd LeafGuard

**Step 2: Open the Project in Jupyter Notebook or Google Colab**

* Open the LeafGuard.ipynb file in Jupyter Notebook or upload it to Google Colab.

**Step 3: Load Dataset**

* Upload the dataset to your notebook environment or use Google Drive to store it.
* Make sure to update the dataset paths in your code.

For uploading manually:

from google.colab import files

uploaded = files.upload()

For Google Drive:

from google.colab import drive

drive.mount('/content/drive')

**Step 4: Train the Model**

Run the cells in the notebook to train the model on the dataset using TensorFlow.

**Step 5: Evaluate the Model**

Evaluate the model's performance on the test dataset and generate accuracy metrics. You can also visualize training graphs using Matplotlib.

**Step 6: Use the Model for Prediction**

Run the final cells in the notebook to use the trained model for predicting new leaf disease samples.

**Step 7: Save and Export Results**

Save the trained model and export any results or visualizations as needed. You can save the model to your Google Drive or local machine.

Example:

model.save('/content/drive/MyDrive/leafguard\_model.h5')

By following these steps, you will have the LeafGuard project installed and running successfully.

# **Blog Link**