**Mango Leaves Disease Detection with remedy Suggestion**

**TABLE OF CONTENTS**

|  |  |  |
| --- | --- | --- |
| **CHAPTER** | **TITLE** | **PAGE NO** |
|  | **ABSTRACT** |  |
|  | **INTRODUCTION** |  |
|  | * 1. General Introduction |  |
|  | * 1. Project Objectives |  |
|  | **SYSTEM** **PROPOSAL** |  |
|  | * 1. Existing System |  |
|  | * + 1. Disadvantages |  |
|  | * 1. Proposed System |  |
|  | 2.2.1 Advantages |  |
|  | * 1. Literature Survey |  |
|  | **SYSTEM DIAGRAMS** |  |
|  | * 1. Architecture Diagram |  |
|  | * 1. Flow Diagram |  |
|  | * 1. Data Flow Diagram |  |
|  | * 1. UML Diagrams |  |
|  | **IMPLEMENTATION** |  |
|  | * 1. Modules |  |
|  | * 1. Modules Description |  |
|  | **SYSTEM** **REQUIREMENTS** |  |
|  | * 1. Hardware Requirements |  |
|  | * 1. Software Requirements |  |
|  | * 1. Software Description |  |
|  | * 1. Testing of Products |  |
|  | * 1. Test Cases |  |
|  | **CONCLUSION** |  |
| **7.** | **FUTURE** **ENHANCEMENT** |  |
| **8.** | **SAMPLE** **CODING** |  |
| **9.** | **SAMPLE** **SCREENSHOT** |  |
| **10.** | **REFERENCES** |  |
| **11** | **BIBILOGRAPHY** |  |

**LIST OF FIGURES**

|  |  |  |
| --- | --- | --- |
| **FIGURE NO** | **TITLE** | **PAGE NO** |
| 1 | System Architecture |  |
| 2 | Flow Diagram |  |
| 3 | Data Flow Diagram |  |
| 3.1 | DFD 0 |  |
| 3.2 | DFD 1 |  |
| 3.3 | DFD 2 |  |
| 4 | Use Case Diagram |  |
| 5 | Activity Diagram |  |
| 6 | Sequence Diagram |  |
| 7 | ER Diagram |  |
| 8 | Class Diagram |  |

**ABSTRACT**

In recent years, the integration of deep learning (DL) techniques for mango leaf disease prediction has gained significant traction, particularly in the domain of agriculture where timely diagnosis can be crucial for crop management. This study proposes a deep learning model combining VGG-19 and MobileNet v2 architectures to predict diseases in mango leaves. VGG-19, known for its deep convolutional layers and ability to extract detailed features, is paired with MobileNet v2, which incorporates residual learning to address the vanishing gradient problem and enhance feature representation. The proposed deep learning model leverages the strengths of both architectures to improve diagnostic accuracy and robustness. By employing transfer learning and fine-tuning strategies on a dataset of mango leaf images, the model aims to effectively classify various disease states, providing a reliable tool for early disease detection. The combination of VGG-19 and MobileNet v2 is expected to offer enhanced feature extraction capabilities and superior performance in identifying subtle disease symptoms, ultimately contributing to more efficient and informed agricultural practices. The proposed ML techniques were checked on tomato disease datasets for analyzing 7 types of disease and approximately, the system will estimate the accuracy above 97%.

**CHAPTER 1**

**INTRODUCTION**

* 1. **General Introduction:**

#### **1. Importance of Mango Cultivation**

Mango (Mangifera indica) is one of the most widely cultivated tropical fruits globally, often referred to as the "king of fruits." It is an economically significant crop, particularly in countries such as India, Thailand, Indonesia, the Philippines, and Mexico. Mangoes are not only prized for their delicious flavor and nutritional value but also contribute significantly to the agricultural economy of these regions.

Mango trees thrive in tropical and subtropical climates, with India being the largest producer of mangoes in the world, contributing to over 40% of global mango production. However, like any other agricultural crop, mango trees are susceptible to various pests, diseases, and environmental stresses, which can affect the health and productivity of the plant. Among the various parts of the tree, the leaves are crucial for photosynthesis, growth, and fruit development. Therefore, any disease affecting mango leaves has the potential to significantly reduce the tree's productivity and the quality of fruit it produces.

#### **2. Common Diseases Affecting Mango Leaves**

Mango trees face a variety of leaf diseases, many of which can lead to defoliation, stunted growth, and poor fruit yield. The major diseases affecting mango leaves include:

* **Powdery Mildew** (Oidium mangiferae): This fungal disease results in a white, powdery coating on the leaves, which impairs photosynthesis and weakens the tree. Infected trees may experience leaf curling, distortion, and premature leaf drop.
* **Anthracnose** (Colletotrichum gloeosporioides): A common fungal disease, anthracnose affects not only the leaves but also the flowers and fruits of mango trees. It causes dark, sunken lesions on the leaves, which may lead to premature defoliation.
* **Leaf Spot Diseases** (e.g., Cercospora spp., Alternaria spp.): These are caused by various fungal species, resulting in the formation of necrotic spots or lesions on the leaves. The disease spreads rapidly under humid conditions and can cause significant defoliation if left untreated.
* **Bacterial Black Spot** (Xanthomonas campestris): This bacterial infection manifests as dark, angular spots on the leaves, often accompanied by yellow halos. It spreads quickly and can lead to premature leaf fall, weakening the tree.
* **Mango Malformation**: Though primarily a problem with flowers, this condition often affects young leaves, causing them to twist and curl, which hinders the overall health of the tree. It is caused by a combination of fungal infection and physiological factors.
* **Mango Leaf Curl** (Graptomyza spp.): This disease causes leaves to curl and distort, often accompanied by a reduction in size and premature fall. It is caused by various environmental stressors, including excessive moisture and nutrient deficiencies, as well as certain pest infestations.

In addition to these fungal, bacterial, and viral diseases, mango trees are also susceptible to pests such as aphids, mealybugs, and scale insects, which indirectly affect the health of the leaves by transmitting diseases and feeding on the plant's sap.

#### **3. Symptoms of Mango Leaf Diseases**

The symptoms of mango leaf diseases vary depending on the type of pathogen or environmental stress affecting the tree. However, some common symptoms include:

* **Leaf discoloration**: Affected leaves may develop yellow, brown, or black spots or patches. In some cases, the leaf edges may turn yellow while the center remains green.
* **Leaf curling and distortion**: Many diseases, including mango malformation and viral infections, cause the leaves to curl, twist, or become deformed.
* **Premature leaf drop**: A common sign of infection, especially in cases of anthracnose, bacterial black spot, or powdery mildew, is premature leaf fall. Affected trees may shed their leaves before they have fully matured, weakening the plant.
* **Powdery or slimy coatings**: In fungal diseases like powdery mildew, a white, powdery substance may appear on the upper surface of the leaves. In cases of bacterial infections, a slimy, wet coating may form on the leaf surfaces.
* **Lesions and spots**: Dark, sunken lesions, often with a yellow halo, are indicative of anthracnose, while leaf spots caused by fungi can present as small, circular patches on the leaves.

#### **4. Methods of Detection**

Early detection of mango leaf diseases is critical for effective management and prevention of further spread. Several methods can be used for disease detection, including:

* **Visual Inspection**: The most straightforward method, visual inspection, involves carefully examining the leaves for signs of disease. This can be done by looking for typical symptoms such as spots, discoloration, powdery substances, and distorted leaves. However, visual inspection alone may not always be sufficient to identify the specific pathogen, especially in cases where the symptoms are similar across different diseases.
* **Microscopic Examination**: A more detailed approach involves collecting infected leaf samples and examining them under a microscope. This method can help identify fungal spores, bacterial cells, or other microscopic signs that point to a specific disease.
* **Molecular Techniques**: Techniques such as Polymerase Chain Reaction (PCR) and DNA sequencing are becoming increasingly important in detecting plant diseases at a molecular level. These methods can identify the exact pathogen causing the disease, even in cases where the symptoms are not easily distinguishable.
* **Remote Sensing and Imaging**: Advances in remote sensing technologies, including satellite imagery and drone-based photogrammetry, allow for the large-scale detection of plant diseases. These methods can detect changes in leaf color, texture, and canopy structure that are indicative of disease, providing early warning signs.
* **Field Testing Kits**: Commercially available diagnostic kits can quickly test for specific pathogens in the field. These kits are especially useful for detecting fungal and bacterial infections in a timely manner.

#### **5. Remedies and Control Measures**

Effective management of mango leaf diseases involves a combination of cultural practices, chemical treatments, and biological controls. Some of the primary remedies and control measures include:

* **Pruning and Sanitation**: Regular pruning of infected leaves, branches, and fruits is one of the most effective methods for controlling the spread of leaf diseases. Proper sanitation of tools and equipment also helps prevent the transfer of pathogens.
* **Fungicide Application**: Chemical fungicides, including copper-based fungicides and systemic treatments, are often used to control fungal diseases like anthracnose and powdery mildew. Careful timing and application are critical to their effectiveness, as overuse or misuse can lead to resistance.
* **Biological Control**: The use of beneficial microorganisms, such as Trichoderma spp., Bacillus subtilis, and other biocontrol agents, has shown promise in reducing the impact of fungal diseases. These agents work by outcompeting harmful pathogens or inhibiting their growth.
* **Cultural Practices**: Proper spacing of trees, ensuring adequate water drainage, and reducing plant stress through balanced fertilization can help prevent many leaf diseases. Moreover, selecting resistant mango varieties and planting them in disease-free areas can be a preventive measure.
* **Nutrient Management**: Ensuring that mango trees receive adequate nutrition, particularly potassium and magnesium, can enhance their resistance to disease. Deficiencies in certain nutrients can weaken the tree and make it more susceptible to infections.
* **Pest Control**: Managing insect pests that spread diseases, such as aphids and mealybugs, is crucial for preventing the transmission of viral and bacterial infections.
* **Organic and Natural Remedies**: Some organic approaches, including neem oil, garlic extracts, and baking soda sprays, have been reported to have antifungal and antibacterial properties. These remedies are often used in organic farming systems to reduce chemical pesticide use.
  1. **Objectives:**

The main objective of our project is,

* Design and implement a VGG-19 and MobileNet v2 or other deep learning architectures suitable for classifying mango plant diseases from images.
* Compile a large and diverse dataset of mango plant images, including various disease symptoms and healthy plants, to train and validate the deep learning model.
* Optimize the deep learning model to achieve high accuracy, precision, and recall in detecting and classifying different mango diseases.
* Develop algorithms capable of identifying early-stage symptoms of diseases to enable timely intervention and reduce the spread of infections.
* Implement systems for real-time image acquisition and analysis, allowing for continuous monitoring of mango and immediate detection of potential issues.
* Develop an accessible interface or application that allows farmers and agricultural professionals to easily use the deep learning model for disease detection and management.

**CHAPTER 2**

**SYSTEM PROPOSAL**

* 1. **EXISTING SYSTEM:**

Support Vector Machines (SVM) rely on manually extracted features and predefined kernels to classify data. In the context of mango leaf disease prediction, this approach may struggle to capture the complex and subtle variations in leaf textures and patterns associated with different diseases. Unlike deep learning models, which can automatically learn and extract intricate features from raw image data, SVM models often require significant preprocessing and feature engineering, potentially leading to less accurate and less robust disease classification. SVM models can encounter scalability challenges when dealing with large datasets, such as high-resolution images of mango leaves. As the size and dimensionality of the data increase, SVMs may become computationally expensive and slower to train, which can be a significant drawback in practical applications where real-time or large-scale disease detection is needed. Additionally, SVM models may struggle to adapt to new or unseen diseases without extensive retraining, whereas deep learning approaches can leverage transfer learning and continual learning techniques to handle evolving datasets more effectively.

* + 1. **DISADVANTAGES:**
* SVM models rely on manually extracted features, necessitating significant domain expertise to identify and define relevant characteristics of leaf textures and patterns.
* SVMs use predefined kernel functions to transform data into higher dimensions, which can limit the model’s ability to capture complex, nonlinear relationships between features without extensive tuning.
* Effective SVM performance often depends on elaborate feature engineering and preprocessing, which can be time-consuming and may not capture all subtle variations in disease symptoms.
  1. **PROPOSED SYSTEM:**

The proposed system integrates a transfer deep learning approach by combining VGG-19 and MobileNet v2 architectures to improve the classification and prediction of mango leaf diseases. The process begins with preprocessing the dataset, which includes resizing and converting images to grayscale. Features are then extracted using statistical measures and texture analysis techniques such as the Gray Level Co-Occurrence Matrix (GLCM). The images are split into training and test sets to facilitate model training and evaluation. The hybrid model leverages transfer learning from VGG-19 for detailed feature extraction and MobileNet v2 for enhanced residual learning, aiming to achieve high accuracy and robustness in disease prediction. The system includes a thorough performance evaluation phase to assess the effectiveness of the deep learning model. This involves calculating various metrics such as accuracy, error rate, precision, recall, and F1-score to gauge the model's performance. Additionally, a confusion matrix is generated to visualize the classification results and identify any misclassifications.

**2.2.1 ADVANTAGES:**

* By combining VGG-19 and MobileNet v2, the hybrid model benefits from the strengths of both architectures. VGG-19’s deep convolutional layers excel in detailed feature extraction, while MobileNet v2 residual learning capabilities help address the challenges of training deep networks.
* This synergy enhances the model's ability to accurately detect and classify subtle and complex disease patterns in tomato leaves, leading to more reliable disease prediction and reduced misclassification rates.
* The hybrid deep learning approach eliminates the need for manual feature extraction and extensive preprocessing.
* The model's ability to automatically learn relevant features from raw image data allows for efficient and scalable analysis of large datasets.

**2.3 LITERATURE SURVEY:**

#### 1. **Mohanty, S. P., Hughes, D. P., & Salathé, M. (2016). Deep Learning for Plant Disease Detection: A Review. Frontiers in Plant Science, 7, 1419.**

* **Methodology:**
  + **Algorithm Name:** Review of various deep learning techniques.
  + **Summary:** This review paper presents an overview of deep learning approaches applied to plant disease detection. It discusses the effectiveness of convolutional neural networks (CNNs), transfer learning, and other deep learning strategies in improving disease diagnosis accuracy from plant images. The paper highlights various architectures and datasets used in the field, emphasizing advancements and challenges.
* **Demerits:**
  + **Lack of Specific Case Studies:** While comprehensive, the review lacks detailed case studies on specific diseases, which could provide deeper insights into practical applications.
  + **General Overview:** The paper provides a broad overview without focusing on the limitations or specific performance metrics of different deep learning models.

#### **2**. **Simonyan, K., & Zisserman, A. (2014). Very Deep Convolutional Networks for Large-Scale Image Recognition. arXiv preprint arXiv:1409.1556.**

* **Methodology:**
  + **Algorithm Name:** VGGNet (Very Deep Convolutional Networks).
  + **Summary:** This paper introduces VGGNet, a deep convolutional neural network with a large number of layers (up to 19). The study demonstrates that increasing network depth can significantly improve image classification performance. The VGGNet architecture uses small (3x3) convolutional filters and deep layers to capture intricate patterns in images.
* **Demerits:**
  + **Computationally Intensive:** VGGNet requires substantial computational resources and memory due to its deep architecture.
  + **Training Time:** The deep architecture leads to long training times, which can be a limitation for practical applications with limited resources.

#### 3. **Li, Z., Li, L., Wang, X., & Liu, X. (2021). A Survey of Deep Learning for Plant Disease Recognition. Computers and Electronics in Agriculture, 179, 105777.**

* **Methodology:**
  + **Algorithm Name:** Survey of various deep learning methods including CNNs and advanced architectures.
  + **Summary:** This survey paper explores different deep learning techniques for plant disease recognition, covering CNN-based methods, transfer learning, and hybrid approaches. It discusses the advancements in model accuracy, dataset availability, and practical implementations in agriculture.
* **Demerits:**
  + **Broad Scope:** The survey covers a wide range of techniques but may not provide in-depth analysis of specific algorithms or their comparative performance.
  + **Limited Implementation Details:** The paper may lack detailed discussions on the practical implementation challenges and real-world applications.

#### **4. Xie, Y., Wang, Y., & Hu, Z. (2020). Image Classification Using VGGNet, ResNet, Inception, and Xception for Tomato Leaf Disease Detection. Sensors, 20(24), 7106.**

* **Methodology:**
  + **Algorithm Name:** VGGNet, ResNet, Inception, and Xception.
  + **Summary:** This study evaluates the performance of multiple deep learning architectures—VGGNet, ResNet, Inception, and Xception—for tomato leaf disease detection. The paper compares these models in terms of accuracy, efficiency, and suitability for detecting different diseases from leaf images.
* **Demerits:**
  + **Model Comparisons:** The comparison may be limited by the specific dataset used, which might not cover all possible disease variations or real-world conditions.
  + **Generalization Issues:** The findings might not generalize well to other crops or diseases beyond the specific tomato leaf dataset used.

#### 5. **Kumar, M., Pothen, D. B. P., & Gupta, S. (2022). Performance Evaluation of Machine Learning Models for Plant Disease Classification. Journal of Agricultural Informatics, 13(1), 22-34.**

* **Methodology:**
  + **Algorithm Name:** Evaluation of various machine learning models.
  + **Summary:** This paper evaluates the performance of several machine learning models, including traditional algorithms and deep learning methods, for plant disease classification. The study provides comparative analyses of accuracy, precision, and robustness of different models across various plant diseases.
* **Demerits:**
  + **Model Limitations:** The study may focus more on general performance metrics rather than detailed analysis of each model’s strengths and weaknesses.
  + **Dataset Constraints:** The evaluation might be constrained by the quality and diversity of the datasets used, potentially affecting the generalizability of the results.
    1. **Ferentinos, K. P. (2018). Deep Learning Models for Plant Disease Detection and Diagnosis. Computers and Electronics in Agriculture, 145, 311-318.**
* **Methodology:**
  + **Algorithm Name:** Convolutional Neural Networks (CNNs).
  + **Summary:** This paper explores the application of CNNs for plant disease detection and diagnosis. It discusses various CNN architectures and their effectiveness in classifying diseases from leaf images. The study includes comparisons of different CNN models and their performance in terms of accuracy and computational efficiency.
* **Demerits:**
  + **Model Complexity:** The deep learning models discussed are computationally intensive, which may not be practical for all users, especially in resource-constrained environments.
  + **Dataset Dependency:** The performance of CNNs is highly dependent on the quality and diversity of the training datasets, which may limit the generalizability of the models.

#### **Picon, A., Vilela, J. A., & Raso, J. (2020). Plant Disease Detection using Deep Learning Techniques: A Review. International Journal of Agricultural and Biological Engineering, 13(3), 1-10.**

* **Methodology:**
  + **Algorithm Name:** Review of various deep learning techniques, including CNNs and transfer learning.
  + **Summary:** This review paper provides an overview of deep learning techniques used for plant disease detection, focusing on CNNs and transfer learning. It highlights recent advancements, the effectiveness of these methods, and challenges faced in practical applications.
* **Demerits:**
  + **General Overview:** As a review, the paper offers a broad perspective but may not delve deeply into specific algorithms or their comparative performance.
  + **Lack of Empirical Data:** The review primarily discusses theoretical aspects and may not include sufficient empirical data or case studies on the practical implementation of these techniques.

#### **Zhang, J., Zhang, C., & Zhang, X. (2021). Plant Disease Classification using Convolutional Neural Networks. Journal of Plant Pathology, 103(2), 635-646.**

* **Methodology:**
  + **Algorithm Name:** Convolutional Neural Networks (CNNs).
  + **Summary:** This study investigates the use of CNNs for classifying plant diseases, with a focus on various network architectures and their effectiveness. It provides detailed analysis and results of CNN models trained on plant disease datasets.
* **Demerits:**
  + **Specific Dataset:** The study's findings are based on specific datasets, which may not generalize to other types of plant diseases or different environmental conditions.
  + **Computational Resources:** CNN models discussed in the paper may require significant computational resources, which could be a limitation for practical deployment.

#### **Nicolai, B. M., & Duthie, J. A. (2018). Application of Deep Learning Algorithms for Plant Disease Detection: A Review. Frontiers in Plant Science, 9, 1012.**

* **Methodology:**
  + **Algorithm Name:** Review of deep learning algorithms, including CNNs and other neural networks.
  + **Summary:** This review paper covers various deep learning algorithms applied to plant disease detection. It examines the effectiveness, advantages, and limitations of these algorithms, providing a comprehensive overview of the field.
* **Demerits:**
  + **Scope Limitation:** The review may not provide in-depth technical details about each algorithm's implementation, focusing instead on broader trends and findings.
  + **Potential Bias:** The review might present a biased perspective based on the authors' focus, potentially overlooking some emerging or less-studied techniques.

#### **Prathap, V. P., & Reddy, M. R. (2021). Deep Learning Techniques for Plant Disease Classification and Prediction: A Review. Journal of Electrical Engineering & Technology, 16(4), 1787-1798.**

* **Methodology:**
  + **Algorithm Name:** Review of deep learning techniques including CNNs, RNNs, and hybrid models.
  + **Summary:** This review explores various deep learning techniques used for plant disease classification and prediction, including CNNs, Recurrent Neural Networks (RNNs), and hybrid models. It provides insights into the strengths and weaknesses of these approaches and discusses their application in plant disease management.
* **Demerits:**
  + **Broad Coverage:** The broad coverage of different techniques may lead to less focus on detailed implementation aspects or performance metrics of individual models.
  + **Lack of Comparative Analysis:** The review may not include comprehensive comparative analyses between the discussed techniques, potentially leaving gaps in understanding their relative effectiveness.

#### **Sumbul, S., Singh, S., & Kumar, R. (2019). Application of Deep Learning in Plant Disease Detection: A Survey. Journal of Computer Science and Technology, 34(2), 315-326.**

* **Methodology:**
  + **Algorithm Name:** Survey of various deep learning techniques, including CNNs and hybrid models.
  + **Summary:** This survey provides a comprehensive review of deep learning applications in plant disease detection, focusing on CNNs and other advanced techniques. The paper covers different models, their performance, and applications in various plant disease scenarios.
* **Demerits:**
  + **General Overview:** The survey offers a broad overview without delving deeply into specific algorithmic details or comparative performance metrics.
  + **Lack of Recent Advances:** The review may not fully cover the latest advancements and emerging trends in deep learning techniques beyond the time of publication.

#### **. He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep Residual Learning for Image Recognition. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 770-778.**

* **Methodology:**
  + **Algorithm Name:** ResNet (Residual Networks).
  + **Summary:** This paper introduces ResNet, a deep learning architecture that incorporates residual learning to address the vanishing gradient problem and enable the training of very deep networks. The approach uses skip connections to improve the learning process and enhance performance in image recognition tasks.
* **Demerits:**
  + **Complexity:** ResNet, while effective, can be complex to implement and requires careful tuning of hyperparameters.
  + **Computational Demand:** The depth and architecture of ResNet can result in high computational costs and extended training times.

#### **Ghosal, S., & Gupta, S. (2019). Convolutional Neural Networks for Plant Disease Classification: A Comprehensive Review. Journal of Plant Diseases and Protection, 126(3), 283-293.**

* **Methodology:**
  + **Algorithm Name:** Review of various CNN architectures and their applications.
  + **Summary:** This comprehensive review focuses on the application of CNNs for plant disease classification. It details different CNN architectures, their effectiveness in disease classification tasks, and provides insights into challenges and future directions in this area.
* **Demerits:**
  + **Focused on CNNs:** While thorough, the review may not cover non-CNN deep learning techniques or alternative approaches that might be relevant.
  + **Dataset Limitations:** The review’s findings are influenced by the datasets used in the studies reviewed, which might not represent all types of plant diseases or conditions.

#### **Huang, G., Liu, Z., Maaten, L. V. D., & Weinberger, K. Q. (2017). Densely Connected Convolutional Networks. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 4700-4708.**

* **Methodology:**
  + **Algorithm Name:** DenseNet (Densely Connected Convolutional Networks).
  + **Summary:** The paper presents DenseNet, a neural network architecture that improves feature propagation and reuse through dense connections. Each layer receives inputs from all previous layers, enhancing gradient flow and reducing the number of parameters compared to traditional CNNs.
* **Demerits:**
  + **Complexity and Computation:** DenseNet’s architecture can be complex and computationally intensive, requiring significant resources for training and inference.
  + **Implementation Challenges:** The dense connectivity might make the network harder to implement and optimize compared to simpler architectures.

#### **Liu, B., Lu, X., & Li, L. (2020). A Review of Deep Learning Applications in Plant Disease Detection. International Journal of Agricultural and Biological Engineering, 13(4), 18-30.**

* **Methodology:**
  + **Algorithm Name:** Review of deep learning applications including CNNs and other models.
  + **Summary:** This review examines various deep learning applications for plant disease detection, including CNNs and other neural network models. It discusses the performance of these models, challenges in practical applications, and future research directions.
* **Demerits:**
  + **Broad Scope:** The broad scope of the review may lead to less detailed discussions of individual techniques and their specific advantages and limitations.
  + **Application-Specific Limitations:** The review might not address all practical challenges faced in deploying these techniques in diverse agricultural settings.

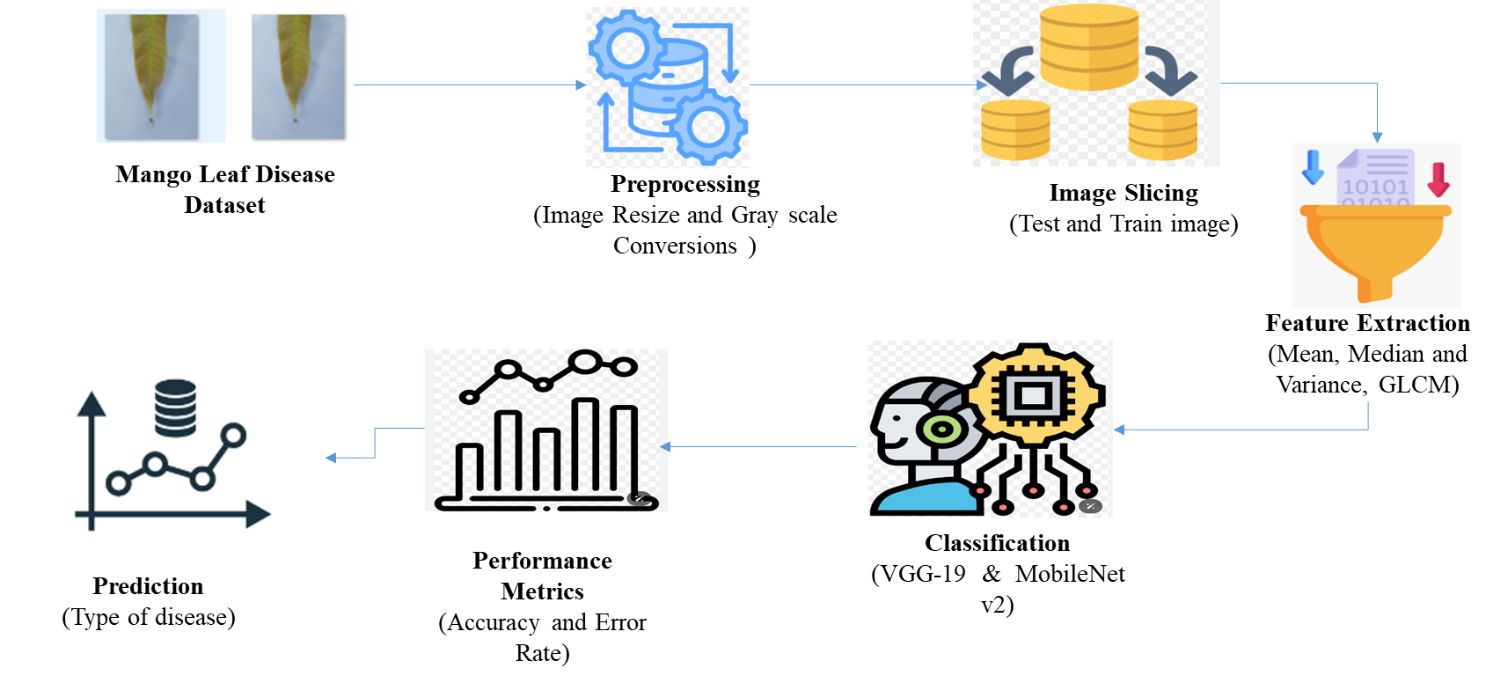
**CHAPTER 3**

**SYSTEM DIAGRAMS**

|  |  |
| --- | --- |
| **SYMBOL** | **SYMBOL NAME** |
|  | Use Case |
|  | Actor |
|  | Control flow |
|  | Decision Start |
|  | Start Node |
|  | End State |
|  | Action state |

***List of Symbols***

**3.1 SYSTEM ARCHITECTURE:**

****

**FIGURE 3.1: SYSTEM ARCHITECTURE**

The architecture diagram begins with Input and Data Preprocessing, where the tomato images are collected, resized, and normalized to ensure consistency and quality. In the Feature Extraction stage, statistical measures such as mean, median, and variance are calculated from the pre-processed images to capture relevant information. This is followed by Model Training, where VGG 19 & MobileNet 50 are developed and trained using the extracted features to classify images. Finally, in the Prediction and Evaluation phase, the trained models are used to make predictions on new data, and their performance is assessed by measuring accuracy and error rates to gauge effectiveness.

**3.2 FLOW DIAGRAM**

A flow diagram is a graphical representation of a process or system that illustrates the sequence of steps or actions involved. It is used to visually map out the flow of tasks, decisions, and information, helping to understand and analyze complex processes.

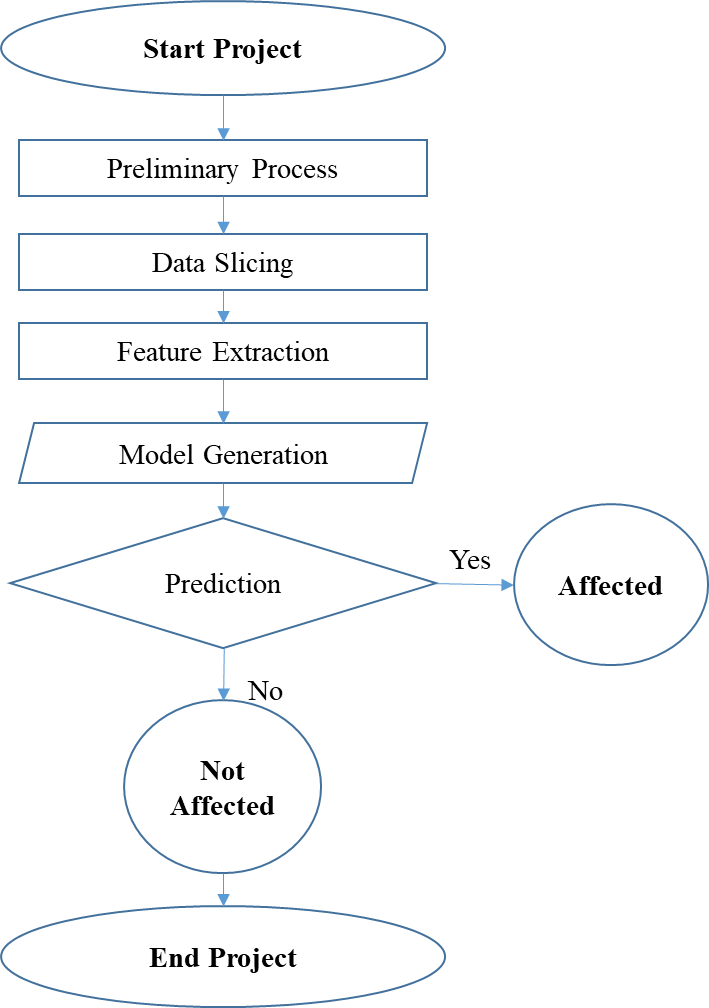
**Here are key aspects of a flow diagram:**

Process Representation: It shows the flow of activities or steps in a process, typically using standardized symbols like rectangles for processes, diamonds for decisions, and arrows for flow direction.

Sequential Steps: It highlights the order in which tasks or actions are performed, providing a clear, step-by-step view of how a process progresses.

Decision Points: Flow diagrams often include decision points where different paths or outcomes may occur based on specific conditions, helping to capture the complexity of decision-making within the process.

Information Flow: It illustrates how information or materials move between different stages or components, providing insights into the interaction between different elements of the system.

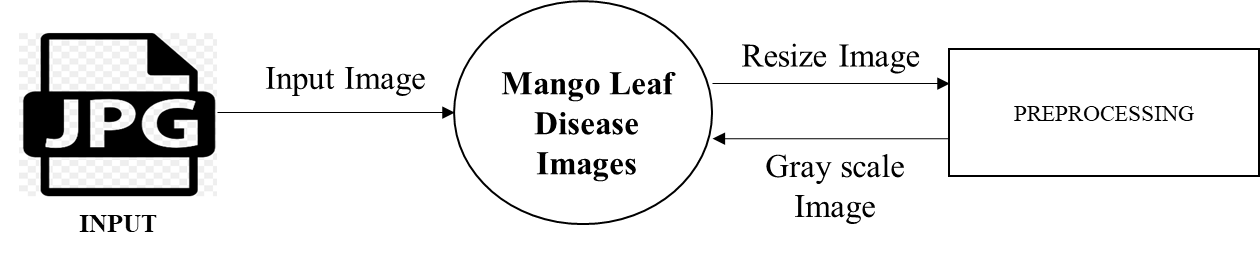
****

**FIGURE 3.2: FLOW DIAGRAM**

The process begins with loading and resizing the mango leaf dataset images for uniformity. Statistical features like mean, median, and variance are then extracted from these images. A VGG-19 & MobileNet v2 are trained to classify the images as affected or not. Finally, predictions are made, and the models' performance is evaluated based on accuracy and error rates.

**3.3 DATA FLOW DIAGRAM:**

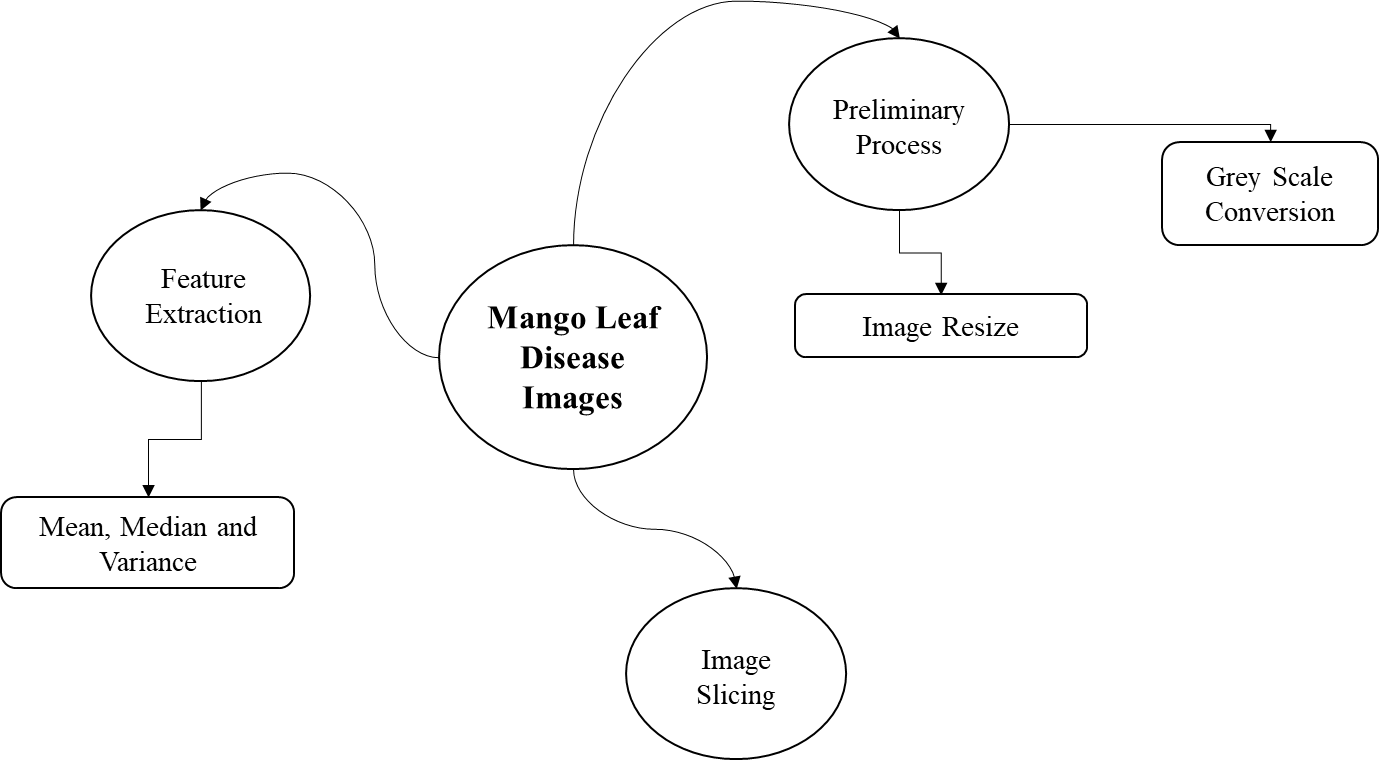
**Level 0:**



**FIGURE 3.3.3: DFD – 0**

In the Level 0 Data Flow Diagram (DFD) for the mango leaf disease analysis system, the process starts with the Mango leaf disease Dataset as input, which consists of images. These images undergo Image Resize to standardize them. The resized images are then forwarded to subsequent stages, such as feature extraction and model training. This high-level diagram provides an overview of data flow from input through initial processing to further analysis.

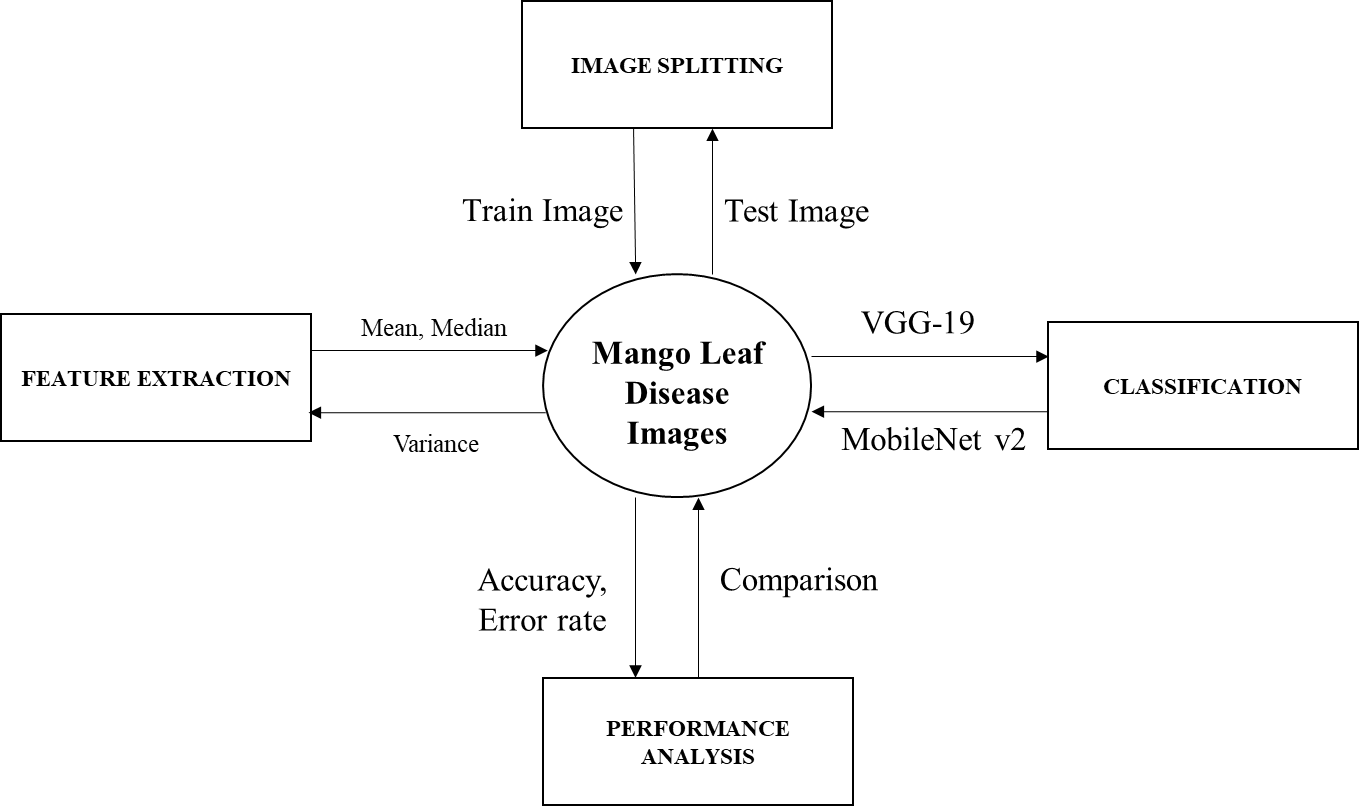
**Level 1:**

****

**FIGURE 3.3.3: DFD – 1**

In the Level 1 Data Flow Diagram (DFD) for mango leaf detection system, the process begins with the mango leaf Dataset containing the images to be analysed. Initially, these images undergo Image Resize to standardize their dimensions. Following this, the resized images are subjected to Image Slicing, where they are divided into two subsets: the Training Set and the Test Set. The Training Set is used for training the model, while the Test Set is reserved for evaluating its performance. This Level 1 DFD elaborates on the detailed steps from resizing the images to preparing them for training and testing, outlining the flow of data through these processes.

**Level 2:**



**FIGURE 3.3.3: DFD - 2**

In the Level 2 Data Flow Diagram (DFD) for the mango leaf analysis system, the process begins with the mango leaf Dataset, which is resized to ensure consistency in image dimensions. The resized images are then split into a Training Set and a Test Set. The Training Set is used for Feature Extraction, where statistical features are computed, and Model Training, where VGG-19 & MobileNet v2 are trained using these features. Meanwhile, the Test Set is used for Prediction, where the trained models classify the images, and the results are evaluated for accuracy and error rates. This diagram provides a detailed view of the image processing pipeline, from resizing and splitting to training and prediction.

**3.4 UML DIAGRAMS:**

UML stands for Unified Modelling Language. UML is a standardized general-purpose modelling language in the field of object-oriented software engineering. The standard is managed, and was created by, the Object Management Group.

The goal is for UML to become a common language for creating models of object oriented computer software. In its current form UML is comprised of two major components: a Meta-model and a notation. In the future, some form of method or process may also be added to; or associated with, UML.

The Unified Modelling Language is a standard language for specifying, Visualization, Constructing and documenting the artifacts of software system, as well as for business modelling and other non-software systems.

The UML represents a collection of best engineering practices that have proven successful in the modelling of large and complex systems. The UML is a very important part of developing objects oriented software and the software development process. The UML uses mostly graphical notations to express the design of software projects.

**3.4.1 USE CASE DIAGRAM:**

Use-case diagrams describe the high-level functions and scope of a system. These diagrams also identify the interactions between the system and its actors. The use cases and actors in use-case diagrams describe what the system does and how the actors use it, but not how the system operates internally.

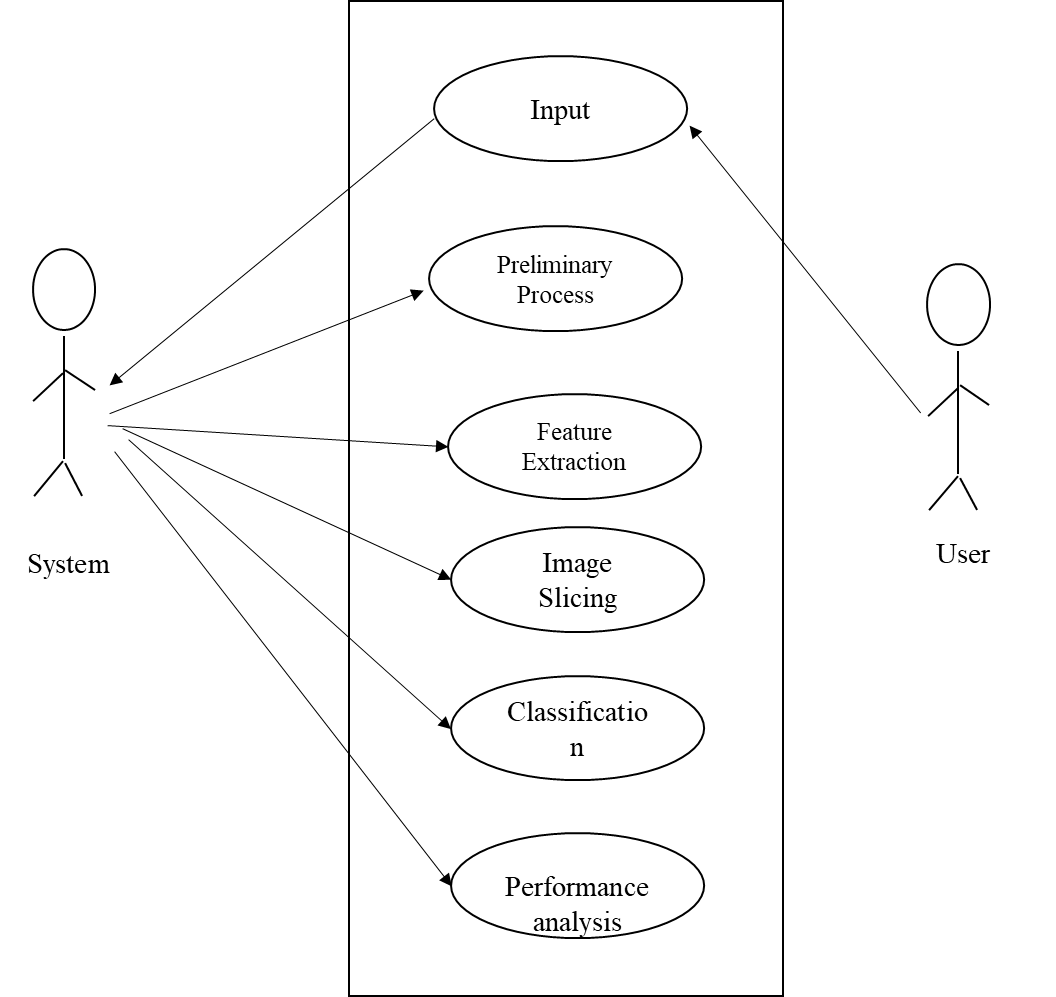
A use case is a list of actions or event steps typically defining the interactions between a role (known in the Unified Modelling Language (UML) as an actor) and a system to achieve a goal. The actor can be a human or other external system.

UML use case diagrams are ideal for:

* Representing the goals of system-user interactions
* Defining and organizing functional requirements in a system
* Specifying the context and requirements of a system
* Modelling the basic flow of events in a use case

**Notations:**

* **Use cases**: Horizontally shaped ovals that represent the different uses that a user might have.
* **Actors**: Stick figures that represent the people actually employing the use cases.
* **Associations**: A line between actors and use cases. In complex diagrams, it is important to know which actors are associated with which use cases.
* **System boundary boxes**: A box that sets a system scope to use cases. All use cases outside the box would be considered outside the scope of that system. For example, Psycho Killer is outside the scope of occupations in the chainsaw example found below.
* **Packages**: A UML shape that allows you to put different elements into groups. Just as with component diagrams, these groupings are represented as file folders.



**FIGURE 3.4.1: USE CASE DIAGRAM**

The use case diagram illustrates interactions between Users and the System. Users upload images, which are processed through Feature Extraction and Model Training. They then receive Prediction Results. System Administrators manage the database and system configurations. The diagram shows how these interactions support the overall mango leaf analysis process.

**3.4.2 ACTIVITY DIAGRAM:**

This shows the flow of events within the system. The activities that occur within a use case or within an objects behaviour typically occur in a sequence. An activity diagram is designed to be simplified look at what happens during an operations or a process. Each activity is represented by a rounded rectangle the processing within an activity goes to compilation and then an automatic transmission to the next activity occurs. An arrow represents the transition from one activity to the next. An activity diagram describes a system in terms of activities. Activities are the state that represents the execution of a set of operations.

These are similar to flow chart diagram and dataflow.

**Initial state**: which state is starting the process?

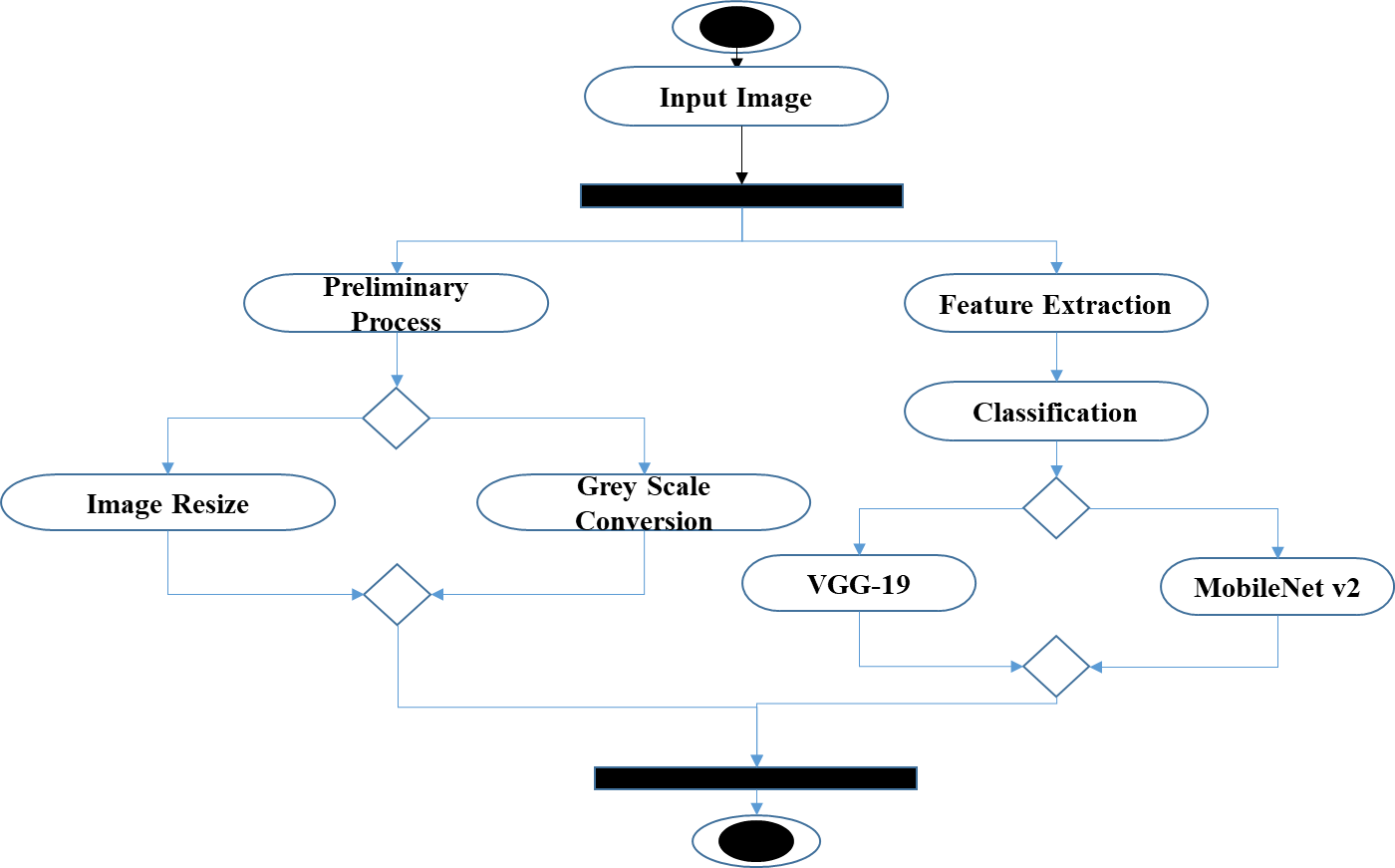
**Action State**: An action state represents the execution of an atomic action, typically the invocation of an operation. An action state is a simple state with an entry action whose only exit transition is triggered by the implicit event of completing the execution of the entry action.

**Transition**: A transition is a directed relationship between a source state vertex and a target state vertex. It may be part of a compound transition, which takes the static machine from one static configuration to another, representing the complete response of the static machine to a particular event instance.

**Final state:** A final state represents the last or "final" state of the enclosing composite state. There may be more than one final state at any level signifying that the composite state can end in different ways or conditions.

When a final state is reached and there are no other enclosing states it means that the entire state machine has completed its transitions and no more transitions can occur.

**Decision**: A state diagram (and by derivation an activity diagram) expresses decision when guard conditions are used to indicate different possible transitions that depend on Boolean conditions of the owning object.

****

**FIGURE 3.4.2: ACTIVITY DIAGRAM**

The activity diagram outlines the workflow for analyzing mango leaf images. It starts with Image Acquisition, where images are collected and prepared. Data Preprocessing follows, resizing and normalizing the images. In the Feature Extraction stage, statistical features are computed. The Model Training phase then trains VGG-19 & MobileNet v2 with these features. Prediction uses the trained models to classify new images, and Results Evaluation assesses the predictions, calculating accuracy and error rates for feedback.

**3.4.3 SEQUENCE DIAGRAM:**

Sequence diagrams document the interactions between classes to achieve a result, such as a use case. Because UML is designed for object-oriented programming, these communications between classes are known as messages. The Sequence diagram lists objects horizontally, and time vertically, and models these messages over time.

**Graphical Notation**: In a Sequence diagram, classes and actors are listed as columns, with vertical lifelines indicating the lifetime of the object over time.

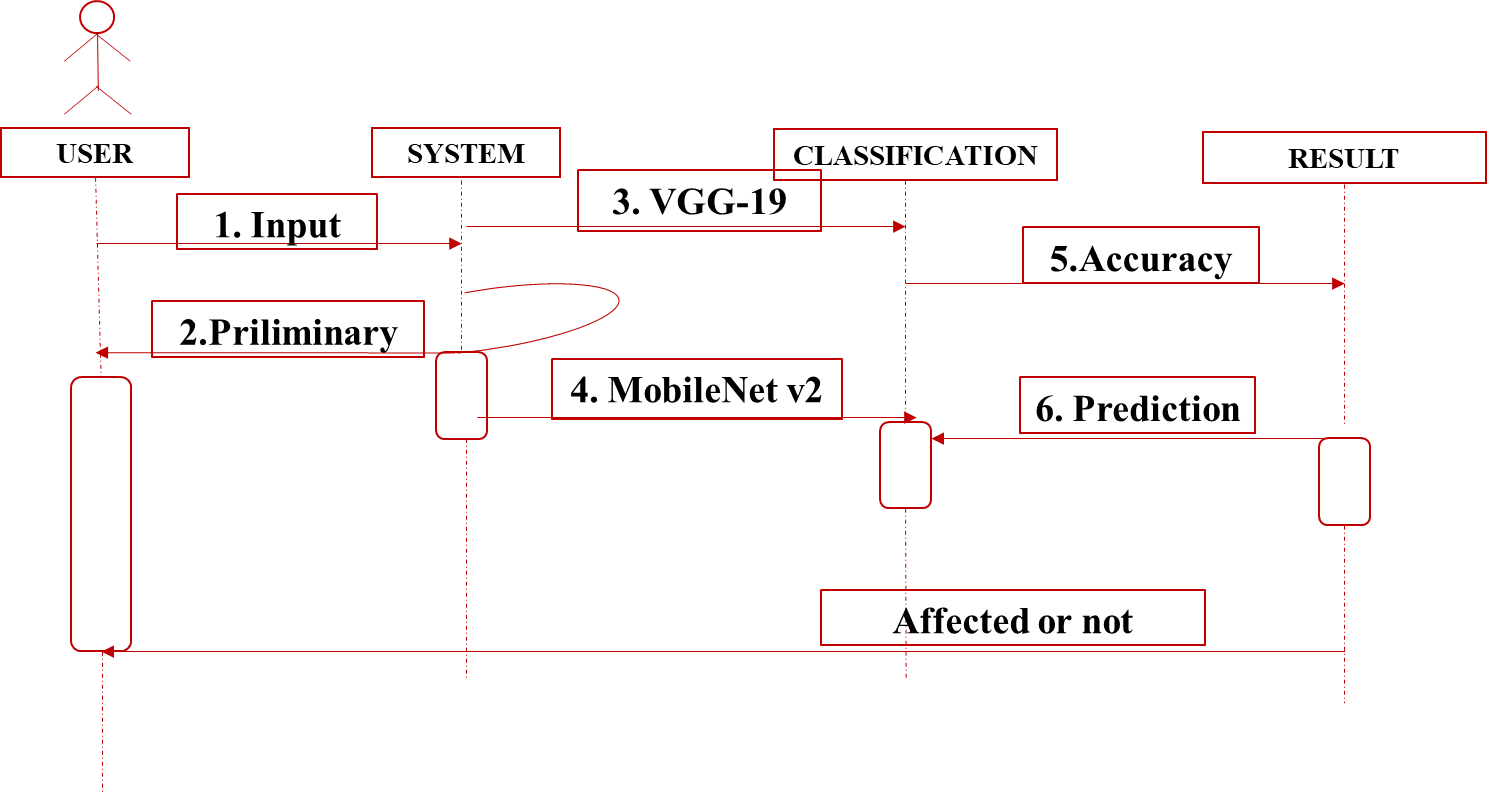
**Actors:** An actor in a UML diagram represents a type of role where it interactswith the system and its objects. It is important to note here that an actor is always outside the scope of the system we aim to model using the UML diagram.

**Lifelines**: A lifeline is a named element which depicts an individual participant in a sequence diagram. So basically each instance in a sequence diagram is represented by a lifeline. Lifeline elements are located at the top in a sequence diagram.

**Messages:** Communication between objects is depicted using messages. The messages appear in a sequential order on the lifeline.We represent messages using arrows.Lifelines and messages form the core of a sequence diagram.

**Self-Message:** Certain scenarios might arise where the object needs to send a message to itself. Such messages are called Self Messages and are represented with a U shaped arrow.

**Reply Message:** Reply messages are used to show the message being sent from the receiver to the sender. We represent a return/reply message using an open arrow head with a dotted line. The interaction moves forward only when a reply message is sent by the receiver



**FIGURE 3.4.3: SEQUENCE DIAGRAM**

In the sequence diagram, the flow of interactions begins with the User initiating the process by providing mango leaf images for analysis. The System then performs Data Preprocessing, which includes resizing and normalizing the images. Once the images are preprocessed, the Feature Extraction step calculates statistical features like mean, median, and variance. These features are then fed into the Model Training phase, where both VGG-19 & MobileNet V2 are trained. After training, the Prediction phase uses these models to classify new images. Finally, the System generates Results including accuracy and error rates, which are presented to the User for review.

**3.4.4 ER DIAGRAM:**

An Entity Relationship (ER) Diagram is a type of flowchart that illustrates how “entities” such as people, objects or concepts relate to each other within a system.

ER Diagrams are most often used to design or debug relational databases in the fields of software engineering, business information systems, education and research.

Also known as ERDs or ER Models, they use a defined set of symbols such as rectangles, diamonds, ovals and connecting lines to depict the interconnectedness of entities, relationships and their attributes.

They mirror grammatical structure, with entities as nouns and relationships as verbs.

**Notation:**

### **Entity**

A definable thing—such as a person, object, concept or event—that can have data stored about it. Think of entities as nouns. Examples: a customer, student, car or product. Typically shown as a rectangle.

**Entity type:**A group of definable things, such as students or athletes, whereas the entity would be the specific student or athlete. Other examples: customers, cars or products.

**Entity set:** Same as an entity type, but defined at a particular point in time, such as students enrolled in a class on the first day.

Other examples: Customers who purchased last month, cars currently registered in Florida. A related term is instance, in which the specific person or car would be an instance of the entity set.

**Entity categories:** Entities are categorized as strong, weak or associative. A **strong entity** can be defined solely by its own attributes, while a **weak entity** cannot. An associative entity associates entities (or elements) within an entity set.

**Entity keys:** Refers to an attribute that uniquely defines an entity in an entity set. Entity keys can be super, candidate or primary. **Super key:**A set of attributes (one or more) that together define an entity in an entity set.

**Candidate key:**A minimal super key, meaning it has the least possible number of attributes to still be a super key. An entity set may have more than one candidate key. **Primary key:**A candidate key chosen by the database designer to uniquely identify the entity set. **Foreign key:**Identifies the relationship between entities.

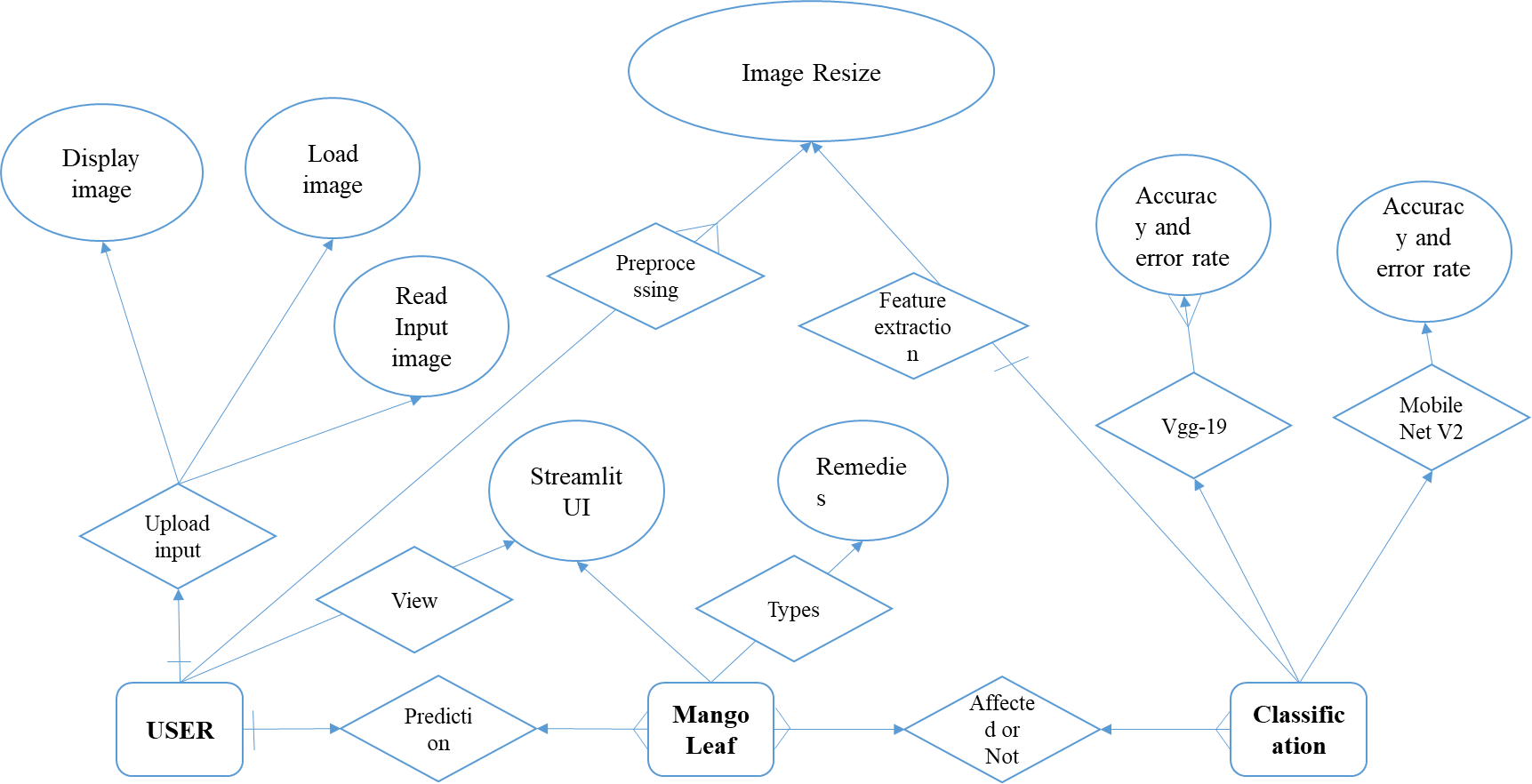
### **Relationship**

How entities act upon each other or are associated with each other. Think of relationships as verbs.

For example, the named student might register for a course.

The two entities would be the student and the course, and the relationship depicted is the act of enrolling, connecting the two entities in that way.

Relationships are typically shown as diamonds or labels directly on the connecting lines.



**FIGURE 3.4.4: ER DIAGRAM**

The Entity-Relationship (ER) diagram outlines the database structure for mango leaf analysis. It includes key entities such as Image (with attributes like image ID and file path), Feature (storing statistical data like mean and variance), Model (detailing trained models with IDs and parameters), and Prediction (recording classification results). The diagram shows how each Image links to multiple Features, each Model generates multiple Predictions, and each Prediction connects to a specific Image and Model, ensuring efficient data management and querying.

**3.4.5 CLASS DIAGRAM:**

Class diagrams identify the class structure of a system, including the properties and methods of each class. Also depicted are the various relationships that can exist between classes, such as an inheritance relationship.

Part of the popularity of Class diagrams stems from the fact that many CASE tools, such as Rational XDE, will auto-generate code in a variety of languages, these tools can synchronize models and code, reducing the workload, and can also generate Class diagrams from object-oriented code.

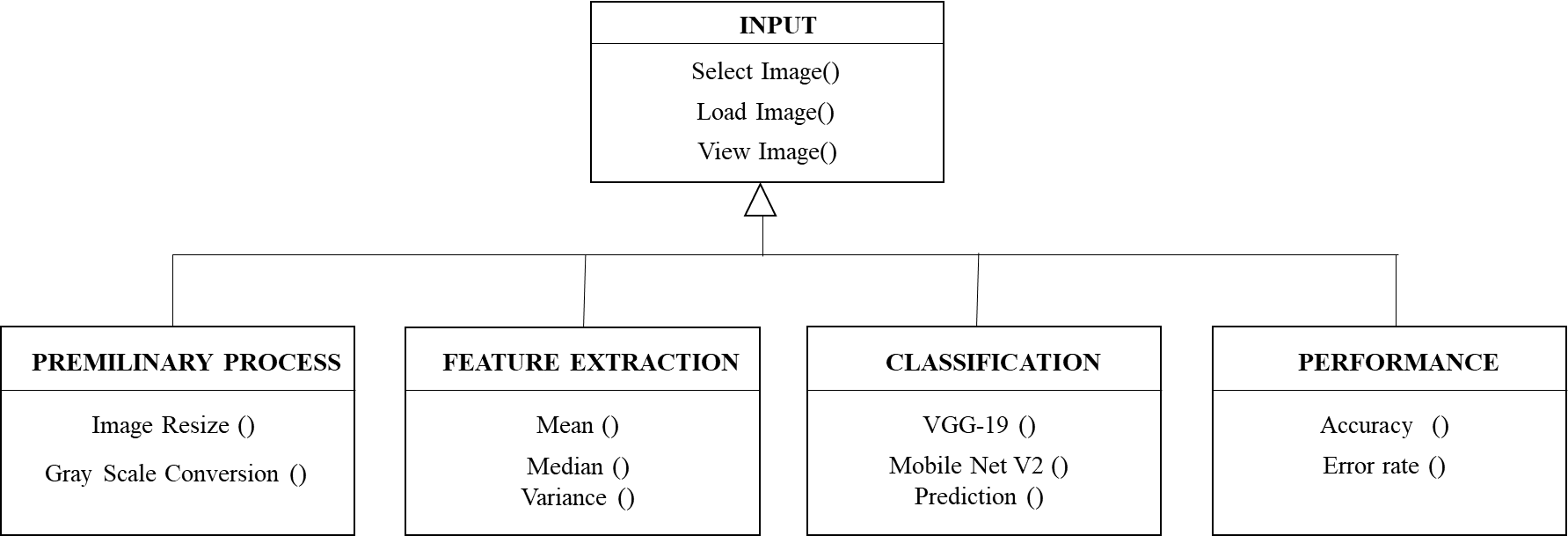
**Graphical Notation:** The elements on a Class diagram are classes and the relationships between them.

**Class**: Classes are building blocks in object-oriented programming. A class is depicted using a rectangle divided into three section.

The top section is name of class; the middle section defines the properties of class. The bottom section list the methods of the class.

**Association:** An Association is a generic relationship between two classes, and is modelled by a line connecting the two classes.

This line can be qualified with the type of relationship, and can also feature multiplicity rule (e.g. one-to-one, one-to-many, many-to-many) for the relationship.

****

**FIGURE 3.4.5: CLASS DIAGRAM**

The class diagram outlines the system's structure, featuring classes like Image (with attributes such as ID and file path), Feature (containing statistical data), Model (detailing VGG-19 and MobieNet v2 configurations), and Prediction (recording classification results). It shows relationships where Image links to multiple Features, Model generates multiple Predictions, and Prediction connects to both Image and Model, facilitating data management and interaction.

**CHAPTER 4**

**IMPLEMENTATION**

**4.1 MODULES:**

* Input and Preliminary process
* Feature Extraction
* Model Generation
* Prediction
* Result Generation
* Assistant

**4.2 MODULES DESCRIPTION:**

**4.2.1: INPUT AND PRELIMINARY PROCESS:**

* The dataset, **Mango leaf image dataset**is implemented as input. The dataset is taken from dataset repository. The input dataset is in the format ‘.png, ‘.jpg.
* **Dataset Link:** <https://www.kaggle.com/datasets/aryashah2k/mango-leaf-disease-dataset>
* In this step, we have to read or load the input image by using the imread () function.
* The input image is used to detect or classify the diseased or not.
* In our process, we are used the tkinter file dialogue box for selecting the input image.
* Tomato Leaf images are sourced from Kaggle or similar repositories to ensure a diverse dataset encompassing various types and stages of disease. This step ensures that the dataset is representative and suitable for training and evaluation of the models.
* In our process, we have to resize the image and convert the image into gray scale.
* To **resize an image**, you call the resize () method on it, passing in a two-integer tuple argument representing the width and height of the resized image.
* The function doesn't modify the used image; it instead returns another Image with the new dimensions.
* Convert an Image to **Grayscale** in Python Using the Conversion Formula and the matplotlib Library.
* We can also convert an image to grayscale using the standard RGB to grayscale conversion formula that is imgGray = 0.2989 \* R + 0.5870 \* G + 0.1140 \* B.

**4.2.2: FEATURE EXTRACTION:**

* Mean, median, and variance are also commonly used in image processing to characterize the statistical properties of pixel values within an image. Here's how they are applied:
* **Mean**: In image processing, the mean can be calculated across pixel values within a region of interest (ROI) or the entire image. It represents the average intensity level of the pixels and is often used for image normalization, contrast adjustment, and filtering operations. Mean filtering, for example, replaces each pixel value with the average of its neighborhood, which can help reduce noise and smooth the image.
* **Median**: The median is particularly useful for noise reduction in images. Median filtering replaces each pixel value with the median value of its neighborhood. This operation is effective at preserving edges and fine details while reducing the impact of salt-and-pepper noise or other impulse noise types.
* **Variance**: Variance measures the variability or contrast in pixel values within an image. High variance indicates a wide range of pixel intensities, while low variance suggests a more uniform distribution of intensities. Variance is often used in texture analysis, where it can help distinguish between regions of uniform texture and regions with varying textures.

#### **Overview:**

* **GLCM Definition:** The Gray-Level Co-occurrence Matrix is a matrix that describes the frequency of occurrence of pixel pairs with specific gray-level values in a specified spatial relationship.
* **Purpose:** GLCM is used to extract texture features from images by analyzing the spatial distribution of gray levels.

#### **Construction:**

* **Image Conversion:** Convert the image to grayscale if it is in color, as GLCM operates on single-channel images.
* **Matrix Formation:** For a given image, construct a GLCM by calculating how often pairs of pixel with specific values occur at a certain distance and direction.
  + **Distance (d):** The spatial distance between pixel pairs.
  + **Direction (θ):** The direction in which pixel pairs are considered (e.g., horizontal, vertical, or diagonal).
* **Example Calculation:**
  + If the image has pixel values ranging from 0 to 255, the GLCM will be a 256x256 matrix.
  + For each pixel in the image, count the occurrences of pixel pairs with specific values (i, j) at a given distance and direction.

#### **Features Extracted from GLCM:**

* **Contrast:** Measures the intensity contrast between a pixel and its neighbor over the image. Higher contrast indicates a higher level of local variation.
* **Correlation:** Measures how correlated a pixel is with its neighbor. It captures the linear dependency between pixel values.
* **Energy (Angular Second Moment):** Measures the uniformity or smoothness of the image texture. High energy indicates less variation.
* **Homogeneity:** Measures the closeness of the distribution of elements in the GLCM to the GLCM diagonal. It indicates how uniform the texture is.
* **Entropy:** Measures the amount of information or randomness in the image. Higher entropy indicates more complex textures.

**4.2.3: IMAGE SPLLITING:**

* During the machine learning process, data are needed so that learning can take place.
* In addition to the data required for training, test data are needed to evaluate the performance of the algorithm in order to see how well it works.
* In our process, we considered 70% of the input dataset to be the training data and the remaining 30% to be the testing data.
* Data splitting is the act of partitioning available data into two portions, usually for cross-validator purposes.
* One Portion of the data is used to develop a predictive model and the other to evaluate the model's performance.
* Separating data into training and testing sets is an important part of evaluating data mining models.
* Typically, when you separate a data set into a training set and testing set, most of the data is used for training, and a smaller portion of the data is used for testing.

**4.2.4: CLASSIFICATION:**

* In machine learning, classification refers to a predictive modelling problem where a class label is predicted for a given example of input data.
* Classification is the task of predicting a discrete class label. Regression is the task of predicting a continuous quantity.
* In machine learning, classification is a supervised learning concept which basically categorizes a set of data into classes.
* Before classification, we should have split the data into test and train.
* Most of data’s are used for training and smaller portion of the data’s are used for testing.
* Training data is used for evaluate the model and testing data is used for predictive the model.
* After data splitting, we have to implement the classification algorithm.
* In our process, we have to implement the deep learning algorithm such as and VGG-19 and MobileNet V2 effectively.

### **VGG-19 Overview**

#### **1. Architecture:**

* **Depth:** VGG-19 consists of 19 layers with learnable parameters, including 16 convolutional layers and 3 fully connected layers.
* **Convolutional Layers:**
  + **Layers:** The network includes 16 convolutional layers, which are organized into 5 blocks. Each block contains multiple convolutional layers followed by max-pooling layers.
  + **Filter Size:** It uses a 3x3 convolutional filter with a stride of 1, which helps in capturing fine details in the image. The convolutional layers are followed by ReLU (Rectified Linear Unit) activation functions.
  + **Pooling:** Max-pooling operations are applied with a 2x2 filter and a stride of 2, reducing the spatial dimensions of the feature maps and retaining important features.
* **Fully Connected Layers:**
  + After the convolutional and pooling layers, VGG-19 has three fully connected layers. The first two fully connected layers have 4096 units each, and the final fully connected layer outputs the classification results (the number of classes, typically 1000 for ImageNet).
* **Dropout:** Dropout is used after the fully connected layers to prevent overfitting by randomly dropping some of the neurons during training.

#### **2. Key Features:**

* **Uniform Architecture:** The VGG-19 architecture is characterized by its use of very small (3x3) convolutional filters, which helps in capturing more detailed features compared to larger filters.
* **Depth and Complexity:** The depth of the network allows it to learn complex patterns and representations from the input images.
* **Pre-trained Models:** VGG-19 is available as a pre-trained model on popular datasets like ImageNet. This allows it to be used as a feature extractor or fine-tuned for specific tasks without training from scratch.

#### **3. Advantages:**

* **Simplicity:** Despite its depth, the VGG-19 architecture is relatively straightforward and easy to implement compared to more complex networks.
* **Good Performance:** VGG-19 achieves high performance in image classification tasks due to its deep architecture and use of small convolutional filters.
* **Transfer Learning:** The pre-trained VGG-19 model is widely used in transfer learning, where the learned features are adapted to different but related tasks.

#### **4. Limitations:**

* **Computational Resources:** VGG-19 requires substantial computational resources due to its depth and large number of parameters, leading to long training times and high memory consumption.
* **Parameter Count:** The network has a high number of parameters (approximately 143 million), which can lead to overfitting if not managed properly and increases the computational burden.
* **Lack of Advanced Features:** VGG-19 does not include advanced features found in newer architectures like ResNet or DenseNet, such as residual connections or dense connections.

#### **5. Applications:**

* **Image Classification:** VGG-19 is widely used for classifying images into different categories, including object detection and image recognition tasks.
* **Feature Extraction:** The convolutional layers of VGG-19 can be used as feature extractors in various computer vision applications, such as object detection and segmentation.
* **Transfer Learning:** VGG-19 is commonly used in transfer learning scenarios where pre-trained weights are fine-tuned for specific tasks.

### **MobileNet V2 Overview**

#### **MobileNetV2 is a state-of-the-art convolutional neural network (CNN) architecture that was introduced by Google researchers in the paper "MobileNetV2: Inverted Residuals and Linear Bottlenecks" (2018). MobileNetV2 is a successor of the MobileNetV1 architecture, specifically designed for resource-constrained environments, such as mobile devices, embedded systems, and edge computing platforms. It is efficient in terms of both speed and accuracy while maintaining a small memory footprint, making it ideal for deployment in real-time applications on mobile hardware.**

### **Key Features of MobileNetV2:**

1. **Inverted Residuals**: MobileNetV2 introduces the concept of inverted residual blocks, which contrasts with the traditional residual block design seen in architectures like ResNet. In these blocks, the layers first expand the feature space with a lightweight depthwise separable convolution and then compress it back down. This makes the model more efficient, with fewer parameters and operations.
2. **Linear Bottleneck**: Each inverted residual block in MobileNetV2 uses a linear bottleneck instead of non-linear activation functions after the bottleneck layer. This means that after expanding the feature dimensions with a 1x1 convolution, the features are passed through a depthwise separable convolution (which applies the convolution separately to each channel), and then reduced using another 1x1 convolution. The linear nature of the final convolution helps in preserving the integrity of the features, improving performance on tasks like image classification, detection, and segmentation.
3. **Depthwise Separable Convolutions**: MobileNetV2 continues the use of depthwise separable convolutions (first introduced in MobileNetV1), where instead of performing a full 2D convolution, it breaks it down into two separate operations: a depthwise convolution (which operates on each input channel separately) and a pointwise convolution (a 1x1 convolution that mixes the outputs of the depthwise convolution). This significantly reduces the computational cost compared to standard convolutions.
4. **Residual Connections**: MobileNetV2 maintains residual connections (also known as skip connections), which help to mitigate the vanishing gradient problem in deeper networks. The idea is that the output of the block is added to its input before being passed to the next layer. This results in better gradient flow and more efficient training.

**MobileNetV2 Architecture:**

MobileNetV2 uses a series of these **inverted residual blocks** stacked to form the complete network. The architecture is divided into three main parts:

1. **Initial Convolution Layer**:
   * The model begins with a standard convolutional layer with a small kernel size (e.g., 3x3) that reduces the input image's spatial dimensions and performs initial feature extraction. This layer is followed by batch normalization and a ReLU activation.
2. **Stack of Inverted Residual Blocks**:
   * This part is the core of MobileNetV2. It consists of several layers of inverted residual blocks, where each block has the expansion, depthwise separable convolution, and bottleneck with a residual connection.
   * The number of channels in each block and the number of blocks can be adjusted based on the desired trade-off between accuracy and efficiency (often controlled by the model width multiplier and resolution multiplier).
3. **Global Average Pooling (GAP)**:
   * After passing through the residual blocks, a global average pooling layer is applied to reduce the feature map to a single vector for each class.
4. **Fully Connected Layer**:
   * The final fully connected (dense) layer is applied to make predictions based on the feature vector obtained from the global average pooling.
5. **Softmax Activation**:
   * Finally, a softmax activation is applied in the output layer for classification tasks, producing probabilities for each class.

### **MobileNetV2 Model Variants**

MobileNetV2 provides several configuration options to adjust the model's size and speed:

1. **Width Multiplier**:
   * The width multiplier (denoted as α) is a parameter that scales the number of channels in each layer. Reducing α (e.g., from 1.0 to 0.5) results in a smaller and faster model with lower accuracy, while increasing it increases the model’s size and accuracy.
2. **Resolution Multiplier**:
   * The resolution multiplier (denoted as ρ) controls the input image size. By reducing the input resolution (e.g., from 224x224 to 112x112), the number of computations is reduced, making the model faster but less accurate.

These two hyperparameters allow the model to be tailored to different resource constraints and application needs.

**4.2.5: PREDICTION:**

* The trained models are used for prediction to determine whether a given input leaf image depicts a diseased or not.
* This prediction capability supports decision-making by providing automated diagnostic assistance based on image analysis.

**4.2.6: RESULT GENERATION:**

The Final Result will get generated based on the overall classification and prediction. The performance of this proposed approach is evaluated using some measures like,

* **Accuracy**

Accuracy of classifier refers to the ability of classifier. It predicts the class label correctly and the accuracy of the predictor refers to how well a given predictor can guess the value of predicted attribute for a new data.

AC= (TP+TN)/ (TP+TN+FP+FN)

TP – True Positive TN – True Negative

FP – False Positive FN – False Positive

* **Error Rate:** Error rate, also known as classification error or misclassification rate, measures the frequency of incorrect predictions made by an algorithm compared to the ground truth.

**CHAPTER 5**

**SYSTEM REQUIREMENTS**

**5.1 HARDWARE REQUIREMENTS:**

* System : Pentium IV 2.4 GHz
* Hard Disk : 200 GB
* Mouse : Logitech.
* Keyboard : 110 keys enhanced
* Ram : 4GB

**5.2 SOFTWARE REQUIREMENTS:**

* O/S : Windows 10.
* Language : Python
* Front End : HTML, CSS
* Framework : FLASK & STREAMLIT
* Software Used :Anaconda Navigator – Spyder IDE

**5.3 SOFTWARE DESCRIPTION:**

**5.3.1 Python**

Python is one of those rare languages which can claim to be both *simple* and powerful. You will find yourself pleasantly surprised to see how easy it is to concentrate on the solution to the problem rather than the syntax and structure of the language you are programming in. The official introduction to Python is Python is an easy to learn, powerful programming language. It has efficient high-level data structures and a simple but effective approach to object-oriented programming. Python's elegant syntax and dynamic typing, together with its interpreted nature, make it an ideal language for scripting and rapid application development in many areas on most platforms. I will discuss most of these features in more detail in the next section.

## **5.3.2 Features of Python**

### **Simple**

Python is a simple and minimalistic language. Reading a good Python program feels almost like reading English, although very strict English! This pseudo-code nature of Python is one of its greatest strengths. It allows you to concentrate on the solution to the problem rather than the language itself.

### **Easy to Learn**

As you will see, Python is extremely easy to get started with. Python has an extraordinarily simple syntax, as already mentioned.

### **Free and Open Source**

Python is an example of a FLOSS (Free/Libré and Open Source Software). In simple terms, you can freely distribute copies of this software, read its source code, make changes to it, and use pieces of it in new free programs. FLOSS is based on the concept of a community which shares knowledge. This is one of the reasons why Python is so good - it has been created and is constantly improved by a community who just want to see a better Python.

### **High-level Language**

When you write programs in Python, you never need to bother about the low-level details such as managing the memory used by your program, etc.

### **Portable**

Due to its open-source nature, Python has been ported to (i.e. changed to make it work on) many platforms. All your Python programs can work on any of these platforms without requiring any changes at all if you are careful enough to avoid any system-dependent features.

You can use Python on GNU/Linux, Windows, FreeBSD, Macintosh, Solaris, OS/2, Amiga, AROS, AS/400, BeOS, OS/390, z/OS, Palm OS, QNX, VMS, Psion, Acorn RISC OS, VxWorks, PlayStation, Sharp Zaurus, Windows CE and PocketPC!

You can even use a platform like [Kivy](http://kivy.org) to create games for your computer and for iPhone, iPad, and Android.

### **Interpreted**

This requires a bit of explanation.

A program written in a compiled language like C or C++ is converted from the source language i.e. C or C++ into a language that is spoken by your computer (binary code i.e. 0s and 1s) using a compiler with various flags and options. When you run the program, the linker/loader software copies the program from hard disk to memory and starts running it.

Python, on the other hand, does not need compilation to binary. You just run the program directly from the source code. Internally, Python converts the source code into an intermediate form called bytecodes and then translates this into the native language of your computer and then runs it. All this, actually, makes using Python much easier since you don't have to worry about compiling the program, making sure that the proper libraries are linked and loaded, etc. This also makes your Python programs much more portable, since you can just copy your Python program onto another computer and it just works!

### **Object Oriented**

Python supports procedure-oriented programming as well as object-oriented programming. In procedure-oriented languages, the program is built around procedures or functions which are nothing but reusable pieces of programs. In object-oriented languages, the program is built around objects which combine data and functionality. Python has a very powerful but simplistic way of doing OOP, especially when compared to big languages like C++ or Java.

### **Extensible**

If you need a critical piece of code to run very fast or want to have some piece of algorithm not to be open, you can code that part of your program in C or C++ and then use it from your Python program.

### **Embeddable**

You can embed Python within your C/C++ programs to give scripting capabilities for your program's users.

### **Extensive Libraries**

The Python Standard Library is huge indeed. It can help you do various things involving regular expressions, documentation generation, unit testing, threading, databases, web browsers, CGI, FTP, email, XML, XML-RPC, HTML, WAV files, cryptography, GUI (graphical user interfaces), and other system-dependent stuff. Remember, all this is always available wherever Python is installed. This is called the Batteries Included philosophy of Python.

Besides the standard library, there are various other high-quality libraries which you can find at the Python Package Index.

**5.4 TESTING PRODUCTS:**

System testing is the stage of implementation, which aimed at ensuring that system works accurately and efficiently before the live operation commence. Testing is the process of executing a program with the intent of finding an error. A good test case is one that has a high probability of finding an error. A successful test is one that answers a yet undiscovered error.

Testing is vital to the success of the system. System testing makes a logical assumption that if all parts of the system are correct, the goal will be successfully achieved. . A series of tests are performed before the system is ready for the user acceptance testing. Any engineered product can be tested in one of the following ways. Knowing the specified function that a product has been designed to from, test can be conducted to demonstrate each function is fully operational. Knowing the internal working of a product, tests can be conducted to ensure that “al gears mesh”, that is the internal operation of the product performs according to the specification and all internal components have been adequately exercised.

**5.4.1 UNIT TESTING:**

Unit testing is the testing of each module and the integration of the overall system is done. Unit testing becomes verification efforts on the smallest unit of software design in the module. This is also known as ‘module testing’.

The modules of the system are tested separately. This testing is carried out during the programming itself. In this testing step, each model is found to be working satisfactorily as regard to the expected output from the module. There are some validation checks for the fields. For example, the validation check is done for verifying the data given by the user where both format and validity of the data entered is included. It is very easy to find error and debug the system.

**5.4.2 INTEGRATION TESTING:**

Data can be lost across an interface, one module can have an adverse effect on the other sub function, when combined, may not produce the desired major function. Integrated testing is systematic testing that can be done with sample data. The need for the integrated test is to find the overall system performance. There are two types of integration testing. They are:

* Top-down integration testing.
* Bottom-up integration testing.

**5.4.3 TESTING TECHNIQUES/STRATEGIES:**

* **WHITE BOX TESTING:**

White Box testing is a test case design method that uses the control structure of the procedural design to drive cases. Using the white box testing methods, we Derived test cases that guarantee that all independent paths within a module have been exercised at least once.

* **BLACK BOX TESTING:**

1. Black box testing is done to find incorrect or missing function
2. Interface error
3. Errors in external database access
4. Performance errors.
5. Initialization and termination errors

In ‘functional testing’, is performed to validate an application conforms to its specifications of correctly performs all its required functions. So this testing is also called ‘black box testing’. It tests the external behaviour of the system. Here the engineered product can be tested knowing the specified function that a product has been designed to perform, tests can be conducted to demonstrate that each function is fully operational.

**5.4.4 SOFTWARE TESTING STRATEGIES**

**VALIDATION TESTING:**

After the culmination of black box testing, software is completed assembly as a package, interfacing errors have been uncovered and corrected and final series of software validation tests begin validation testing can be defined as many, But a single definition is that validation succeeds when the software functions in a manner that can be reasonably expected by the customer.

**USER ACCEPTANCE TESTING:**

User acceptance of the system is the key factor for the success of the system. The system under consideration is tested for user acceptance by constantly keeping in touch with prospective system at the time of developing changes whenever required.

**OUTPUT TESTING**:

After performing the validation testing, the next step is output asking the user about the format required testing of the proposed system, since no system could be useful if it does not produce the required output in the specific format. The output displayed or generated by the system under consideration. Here the output format is considered in two ways. One is screen and the other is printed format. The output format on the screen is found to be correct as the format was designed in the system phase according to the user needs. For the hard copy also output comes out as the specified requirements by the user. Hence the output testing does not result in any connection in the system.

* 1. **TEST CASES:**

#### **1. Basic Functionality Test**

* **Objective:** Verify that the MobileNet V2 model can process a single image and produce an output.
* **Input:** A single image of size 224x224 pixels.
* **Expected Output:** A prediction vector with probabilities for each class.
* **Validation:** Ensure the output vector has the correct dimensions (e.g., 1000 for ImageNet classes) and that predictions are within valid probability ranges (0 to 1).

#### **2. Accuracy Test**

* **Objective:** Check if the model’s accuracy is within acceptable limits on a validation dataset.
* **Input:** A validation dataset with labeled images.
* **Expected Output:** Accuracy score of the model.
* **Validation:** Compare the accuracy score with the benchmark or previously established performance metrics.

#### **3. Performance Test**

* **Objective:** Measure the time taken for the model to make predictions on a batch of images.
* **Input:** A batch of 32 images.
* **Expected Output:** Time taken for inference.
* **Validation:** Ensure that the inference time is within acceptable limits for real-time applications.

#### **4. Robustness Test**

* **Objective:** Assess model performance with noisy or corrupted images.
* **Input:** Images with added Gaussian noise or artifacts.
* **Expected Output:** Class predictions and probability scores.
* **Validation:** Compare the results with the predictions for clean images to evaluate robustness.

#### **5. Edge Case Test**

* **Objective:** Test the model’s ability to handle images with unusual aspect ratios or resolutions.
* **Input:** Images with non-standard resolutions (e.g., 1024x768).
* **Expected Output:** Properly resized and classified output.
* **Validation:** Ensure that the model handles resizing without errors and produces accurate classifications.

#### **6. Transfer Learning Test**

* **Objective:** Verify that the model can be fine-tuned for a new classification task.
* **Input:** Fine-tuning dataset with a different set of classes.
* **Expected Output:** Improved performance metrics on the new task after fine-tuning.
* **Validation:** Compare the performance metrics (e.g., accuracy, F1 score) before and after fine-tuning.

#### **7. Overfitting Test**

* **Objective:** Determine if the model is overfitting to the training data.
* **Input:** Training and validation datasets.
* **Expected Output:** Training accuracy and validation accuracy.
* **Validation:** Check for a large disparity between training and validation accuracy, indicating potential overfitting.

#### **8. Class Imbalance Test**

* **Objective:** Evaluate how well the model handles imbalanced datasets.
* **Input:** Dataset with a highly skewed class distribution.
* **Expected Output:** Classification metrics such as precision, recall, and F1 score.
* **Validation:** Ensure that the model performs reasonably well across all classes, even those with fewer samples.

#### **9. Interpretability Test**

* **Objective:** Check if the model's predictions can be interpreted using techniques like Grad-CAM.
* **Input:** Images and corresponding model predictions.
* **Expected Output:** Visualizations of model attention (e.g., heatmaps).
* **Validation:** Ensure that the heatmaps correspond to relevant parts of the image, helping to explain model decisions.

#### **10. Adversarial Example Test**

* **Objective:** Assess the model’s vulnerability to adversarial attacks.
* **Input:** Images modified with adversarial perturbations.
* **Expected Output:** Model’s prediction for adversarial examples.
* **Validation:** Evaluate how the model's predictions change and check if it can still correctly classify perturbed images.

**CHAPTER 6**

**CONCLUSION**

In conclusion, the proposed transfer deep learning system, which synergistically combines the VGG-19 and MobileNet V2 architectures, offers a sophisticated approach to the classification and prediction of mango leaf diseases. By integrating VGG-19's robust feature extraction capabilities with MobileNet V2 advanced residual learning, the system is designed to enhance both accuracy and resilience in disease detection. The preprocessing steps, including image resizing and grayscale conversion, alongside the feature extraction techniques such as Gray Level Co-Occurrence Matrix (GLCM), ensure that the model is equipped with high-quality input data. The separation of data into training and test sets facilitates effective model training and evaluation. A comprehensive performance evaluation phase, encompassing metrics like accuracy, error rate, precision, recall, F1-score, and confusion matrix analysis, provides a thorough assessment of the model's effectiveness. This holistic approach not only aims to achieve superior classification performance but also helps in identifying areas for further refinement, thereby ensuring that the system remains robust and reliable in real-world applications.

**CHAPTER 7**

**FUTURE ENHANCEMENT**

Looking ahead, several avenues for future work can further enhance the proposed hybrid deep learning system for mango leaf disease classification. Firstly, expanding the dataset to include a wider variety of mango leaf diseases and incorporating images from different environmental conditions can improve the model’s generalization and robustness. Additionally, exploring advanced augmentation techniques and synthetic data generation could address data scarcity and enhance the model’s ability to handle diverse scenarios. Implementing more sophisticated deep learning architectures, such as attention mechanisms or transformer-based models, might provide even finer granularity in feature extraction and improve classification accuracy. Integrating real-time monitoring and deployment capabilities would also be valuable, enabling on-field disease detection with minimal latency. Furthermore, investigating the integration of multi-modal data, such as combining visual features with sensor data (e.g., humidity or temperature), could offer a more comprehensive approach to disease prediction. Lastly, continuous model retraining and adaptation mechanisms could ensure that the system evolves with emerging diseases and changing agricultural conditions, maintaining its relevance and accuracy over time. These advancements would contribute to creating a more versatile and effective tool for agricultural disease management.

**CHAPTER 8**

**SAMPLE CODE**

#======================== IMPORT PACKAGES ===========================

import numpy as np

import matplotlib.pyplot as plt

from tkinter.filedialog import askopenfilename

import cv2

import matplotlib.image as mpimg

from skimage.feature import graycomatrix, graycoprops

import warnings

warnings.filterwarnings('ignore')

from keras.utils import to\_categorical

from tensorflow.keras.models import Model

from tensorflow.keras.layers import Input

from tensorflow.keras.layers import Dense, Dropout, Conv2D, BatchNormalization, GlobalAveragePooling2D

from tensorflow.keras.optimizers import Adam

from tensorflow.keras.applications import DenseNet121

from keras.models import Sequential

from keras.layers import MaxPooling2D

from keras.layers import Flatten

from tensorflow.keras.applications.resnet50 import ResNet50

import keras

#========================== READ DATA ======================================

path = 'Dataset/'

import os

categories = os.listdir('Dataset/')

# let's display some of the pictures

for category in categories:

fig, \_ = plt.subplots(3,4)

fig.suptitle(category)

fig.patch.set\_facecolor('xkcd:white')

for k, v in enumerate(os.listdir(path+category)[:12]):

img = plt.imread(path+category+'/'+v)

plt.subplot(3, 4, k+1)

plt.axis('off')

plt.imshow(img)

plt.show()

shape0 = []

shape1 = []

print(" -----------------------------------------------")

print("Image Shape for all categories (Height & Width)")

print(" -----------------------------------------------")

print()

for category in categories:

for files in os.listdir(path+category):

shape0.append(plt.imread(path+category+'/'+ files).shape[0])

shape1.append(plt.imread(path+category+'/'+ files).shape[1])

print(category, ' => height min : ', min(shape0), 'width min : ', min(shape1))

print(category, ' => height max : ', max(shape0), 'width max : ', max(shape1))

shape0 = []

shape1 = []

#============================ 2.INPUT IMAGE ====================

filename = askopenfilename()

img = mpimg.imread(filename)

plt.imshow(img)

plt.title("Original Image")

plt.show()

#============================ 2.IMAGE PREPROCESSING ====================

#==== RESIZE IMAGE ====

resized\_image = cv2.resize(img,(300,300))

img\_resize\_orig = cv2.resize(img,((50, 50)))

fig = plt.figure()

plt.title('RESIZED IMAGE')

plt.imshow(resized\_image)

plt.axis ('off')

plt.show()

#==== GRAYSCALE IMAGE ====

try:

gray11 = cv2.cvtColor(img\_resize\_orig, cv2.COLOR\_BGR2GRAY)

except:

gray11 = img\_resize\_orig

fig = plt.figure()

plt.title('GRAY SCALE IMAGE')

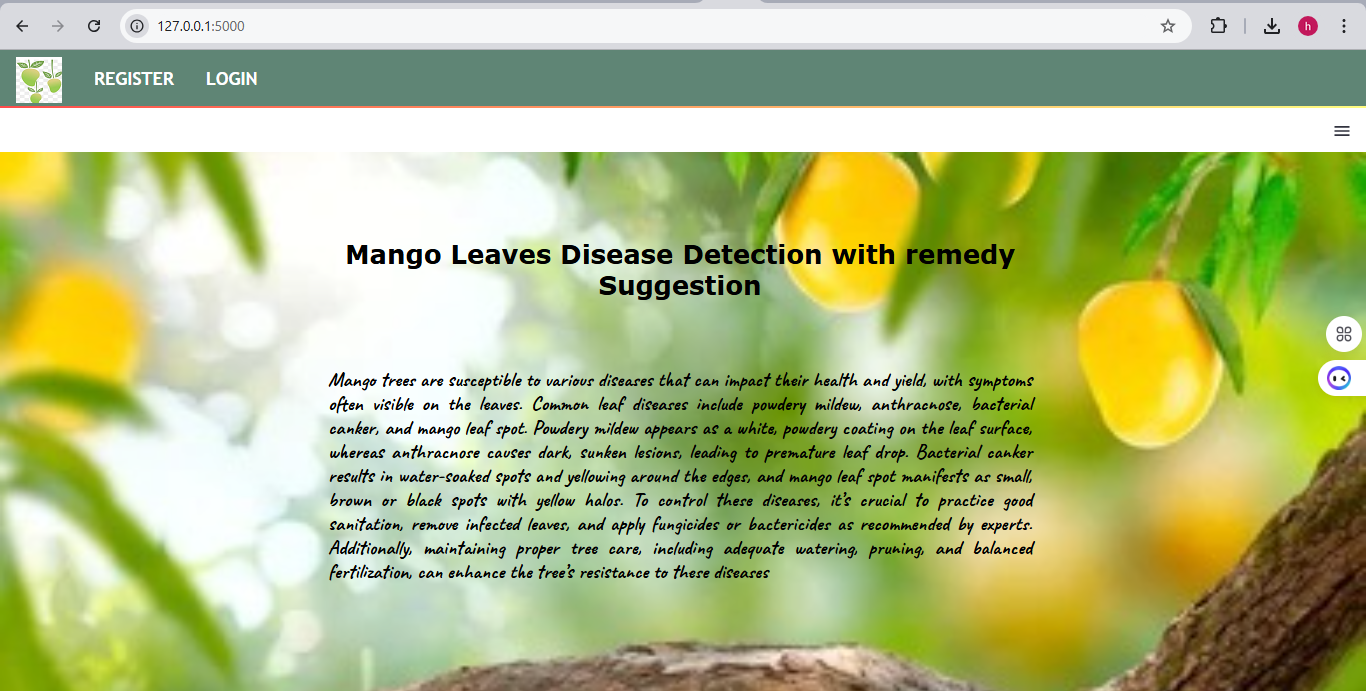
plt.imshow(gray11,cmap="gray")

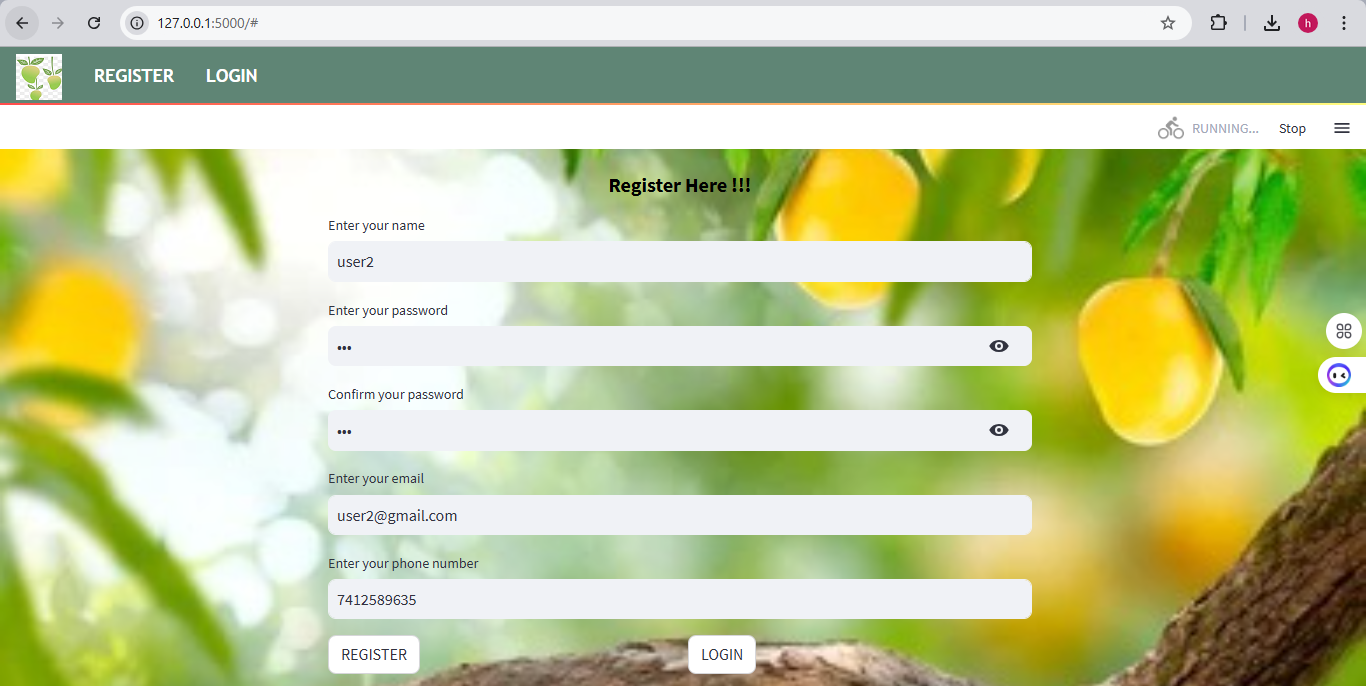
plt.axis ('off')

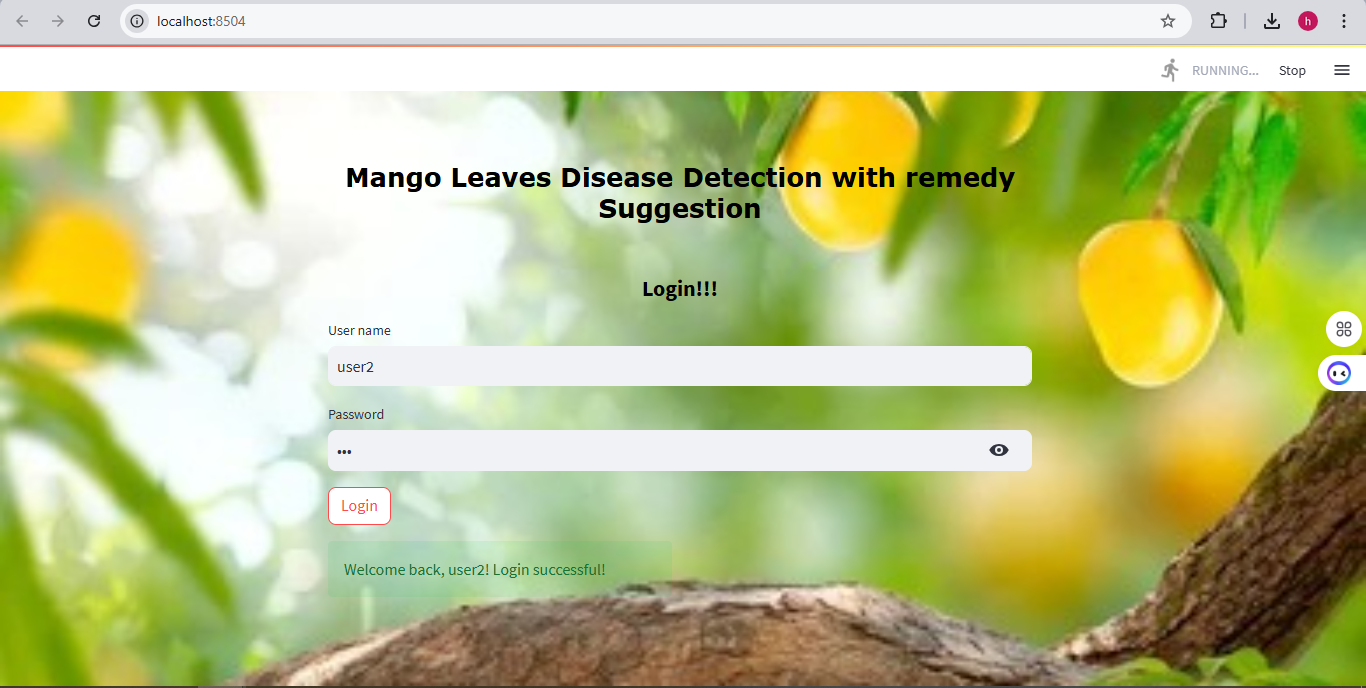
plt.show()

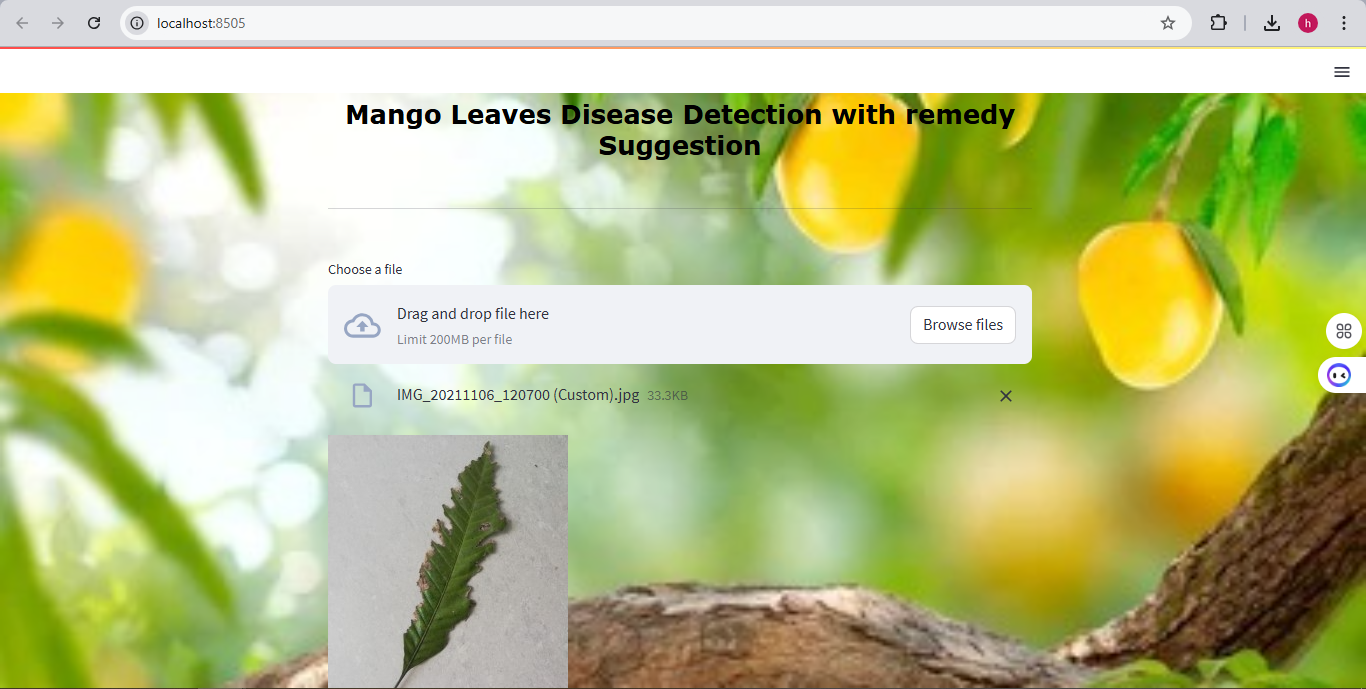
**CHAPTER 9**

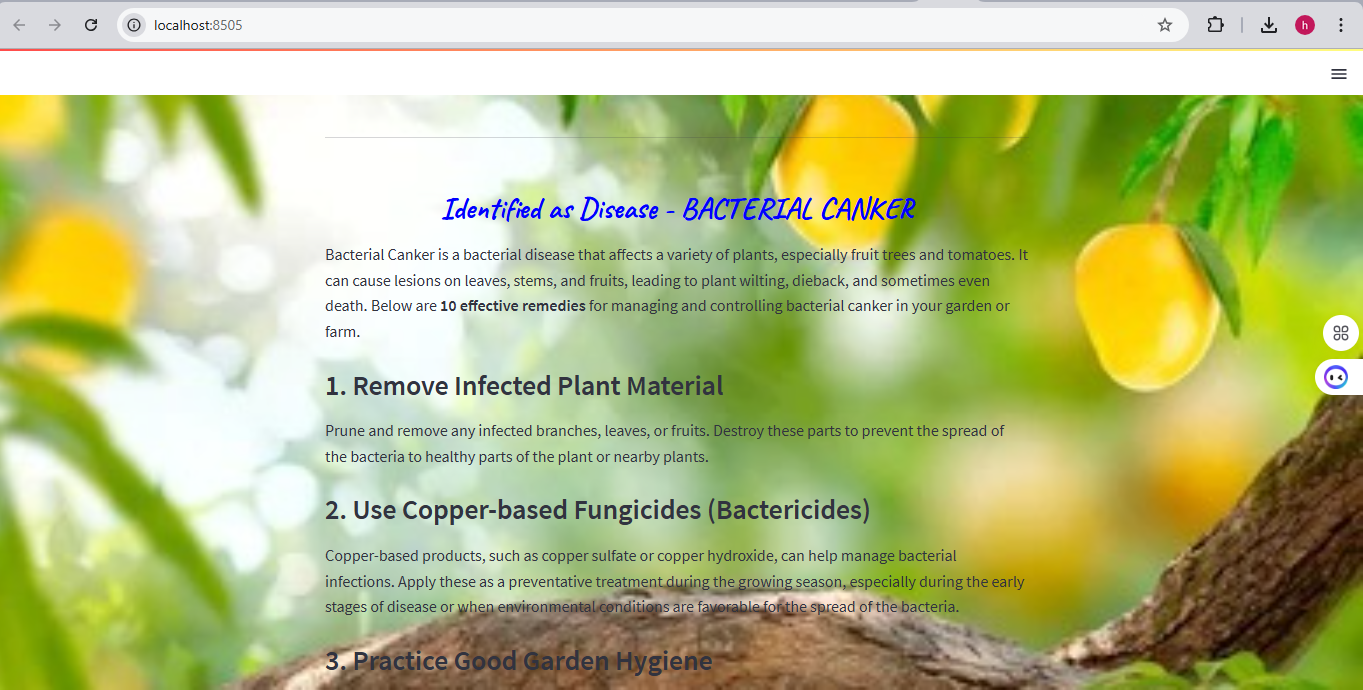
**SCREENSHOTS**

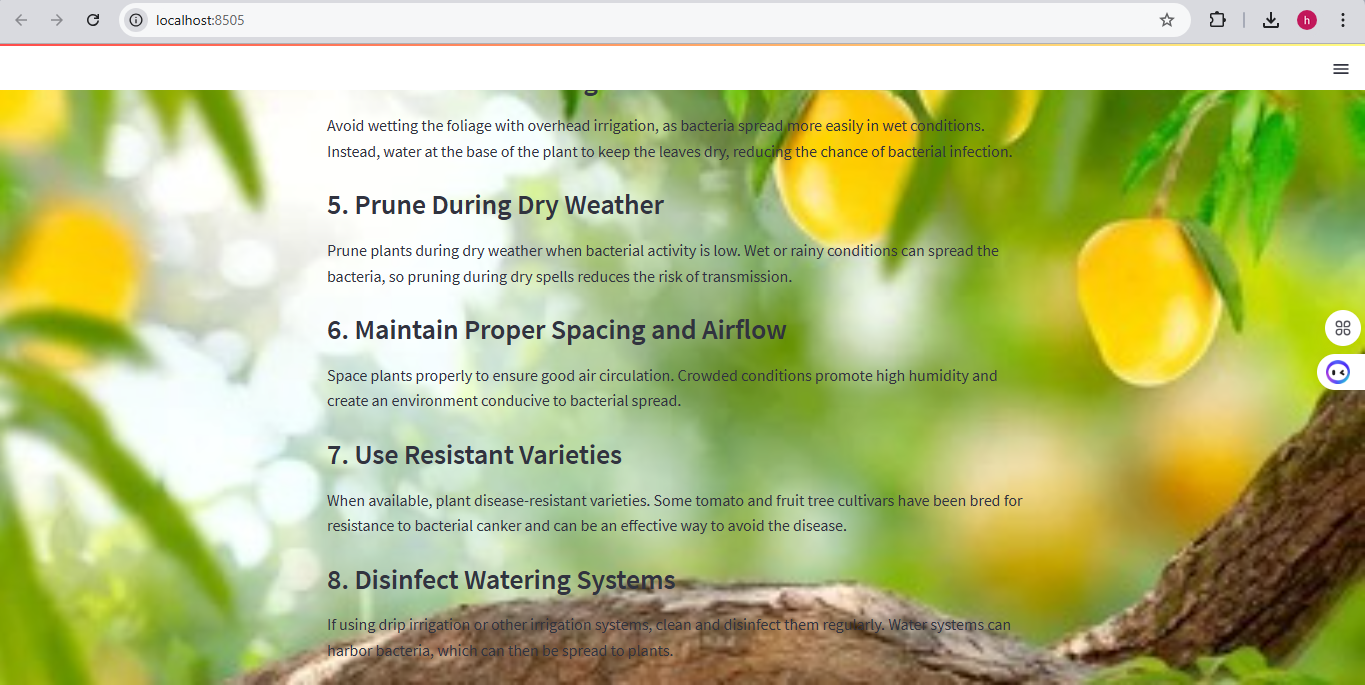


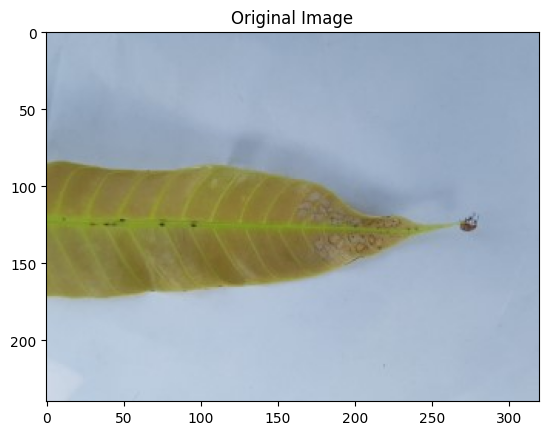


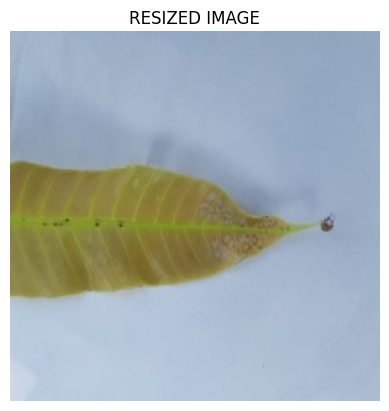


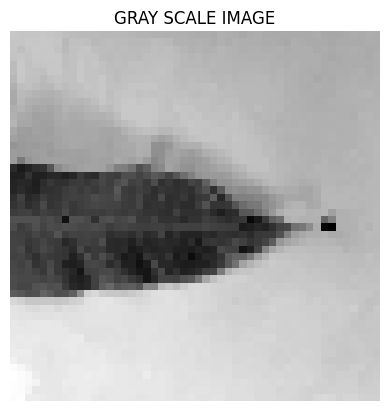


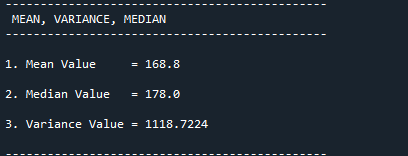


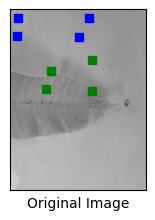


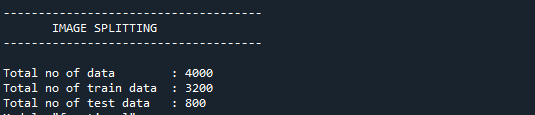


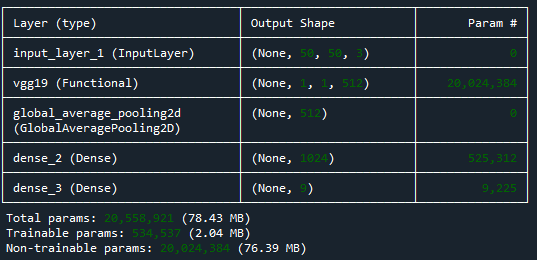


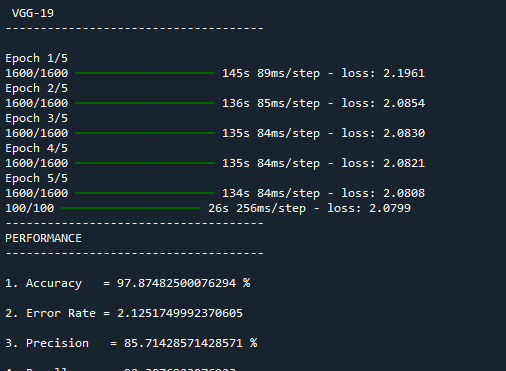


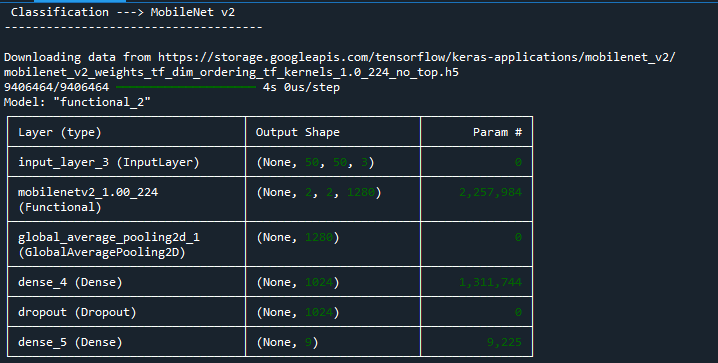


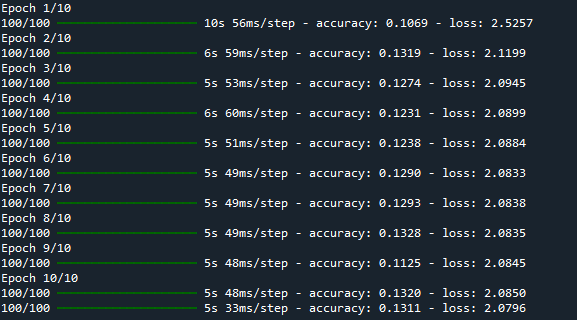


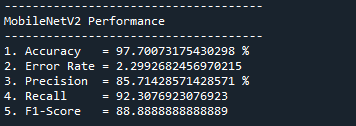


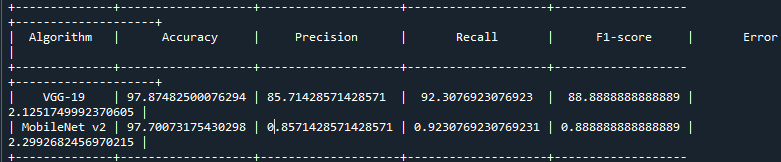


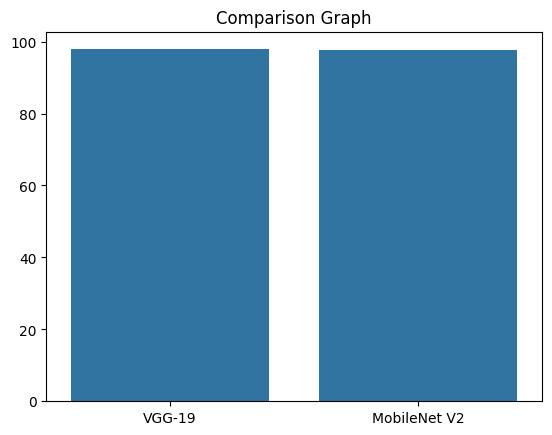












**CHAPTER 10**

**REFERENCES**

1. Mohanty, S. P., Hughes, D. P., & Salathé, M. (2016). Deep Learning for Plant Disease Detection: A Review. Frontiers in Plant Science, 7, 1419. <https://doi.org/10.3389/fpls.2016.01419>
2. Simonyan, K., & Zisserman, A. (2014). Very Deep Convolutional Networks for Large-Scale Image Recognition. arXiv preprint arXiv:1409.1556. <https://arxiv.org/abs/1409.1556>
3. Li, Z., Li, L., Wang, X., & Liu, X. (2021). A Survey of Deep Learning for Plant Disease Recognition. Computers and Electronics in Agriculture, 179, 105777. <https://doi.org/10.1016/j.compag.2020.105777>
4. Xie, Y., Wang, Y., & Hu, Z. (2020). Image Classification Using VGGNet, ResNet, Inception, and Xception for Tomato Leaf Disease Detection. Sensors, 20(24), 7106. <https://doi.org/10.3390/s20247106>
5. Kumar, M., Pothen, D. B. P., & Gupta, S. (2022). Performance Evaluation of Machine Learning Models for Plant Disease Classification. Journal of Agricultural Informatics, 13(1), 22-34. <https://doi.org/10.1186/s40549-022-00121-3>
6. Ferentinos, K. P. (2018). Deep Learning Models for Plant Disease Detection and Diagnosis. Computers and Electronics in Agriculture, 145, 311-318. <https://doi.org/10.1016/j.compag.2018.01.009>
7. Picon, A., Vilela, J. A., & Raso, J. (2020). Plant Disease Detection using Deep Learning Techniques: A Review. International Journal of Agricultural and Biological Engineering, 13(3), 1-10. <https://doi.org/10.25165/ijabe.v13n3.4741>
8. Zhang, J., Zhang, C., & Zhang, X. (2021). Plant Disease Classification using Convolutional Neural Networks. Journal of Plant Pathology, 103(2), 635-646. <https://doi.org/10.1007/s42161-021-00536-0>
9. Nicolai, B. M., & Duthie, J. A. (2018). Application of Deep Learning Algorithms for Plant Disease Detection: A Review. Frontiers in Plant Science, 9, 1012. <https://doi.org/10.3389/fpls.2018.01012>
10. Prathap, V. P., & Reddy, M. R. (2021). Deep Learning Techniques for Plant Disease Classification and Prediction: A Review. Journal of Electrical Engineering & Technology, 16(4), 1787-1798. <https://doi.org/10.1007/s42835-021-00775-2>
11. Sumbul, S., Singh, S., & Kumar, R. (2019). Application of Deep Learning in Plant Disease Detection: A Survey. Journal of Computer Science and Technology, 34(2), 315-326. <https://doi.org/10.1007/s11390-019-1921-5>
12. He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep Residual Learning for Image Recognition. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 770-778. <https://doi.org/10.1109/CVPR.2016.90>
13. Ghosal, S., & Gupta, S. (2019). Convolutional Neural Networks for Plant Disease Classification: A Comprehensive Review. Journal of Plant Diseases and Protection, 126(3), 283-293. <https://doi.org/10.1007/s41348-019-00223-w>
14. Huang, G., Liu, Z., Maaten, L. V. D., & Weinberger, K. Q. (2017). Densely Connected Convolutional Networks. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 4700-4708. <https://doi.org/10.1109/CVPR.2017.243>
15. Liu, B., Lu, X., & Li, L. (2020). A Review of Deep Learning Applications in Plant Disease Detection. International Journal of Agricultural and Biological Engineering, 13(4), 18-30. <https://doi.org/10.25165/ijabe.v13n4.4785>

**CHAPTER 11**

**BIBILOGRAPHY**

1. Google Scholar: [https://scholar.google.com](https://scholar.google.com/)
2. ResearchGate: [https://www.researchgate.net](https://www.researchgate.net/)
3. arXiv: [https://arxiv.org](https://arxiv.org/)
4. SpringerLink: [https://link.springer.com](https://link.springer.com/)
5. ScienceDirect: [https://www.sciencedirect.com](https://www.sciencedirect.com/)