# Plant Disease Detection and Classification using Machine Learning and Deep Learning

A Mini Project report submitted in the partial fulfilment of the requirements for the award of the degree of

## Bachelor of Technology in Computer Science & Engineering (Artificial Intelligence and Machine Learning)


By

21071A66E5 Jayamangala Srujana

21071A66G0 Miyyapuram Srija

21071A66H4 Ravuri Joshini

22075A6616 Chitneni Harshitha


**Under the Guidance of**


Dr. N. Nirmala Jyothi
(Assistant Professor(AIML & IoT) )

CSE – AIML & IoT, VNR VJIET




**DEPARTMENT OF**
**CSE- (ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING & INTERNET OF THINGS)**


**August 2024**

VALLURUPALLI NAGESWARA RAO VIGNANA JYOTHI
INSTITUTE OF ENGINEERING AND TECHNOLOGY
An Autonomous, ISO 9001:2015 & QS I-Gauge Diamond Rated Institute, Accredited by NAAC with 'A++' Grade
NBA Accreditation for B.Tech. CE, EEE, ME, ECE, CSE, EIE, IT Programmes
Approved by AICTE, New Delhi, Affiliated to JNTUH, NIRF 135th Rank in Engineering Category
Recognized as "College with Potential for Excellence" by UGC
VignanaJyothi Nagar, Pragathi Nagar, Nizampet (S.O), Hyderabad – 500 090, TS, India. Telephone No: 040-2304 2758/59/60, Fax: 040-23042761

# CERTIFICATE

This is to certify that **Jayamangala Srujana (21071A66E5), Miyyapuram Srija (21071A66G0), Ravuri Joshini (21071A66H4), Chitneni harshitha (22075A6616)** have successfully completed their Mini project work at CSE-(AIML & IoT) Department of VNRVJIET, Hyderabad entitled **"Plant Disease Detection and Classification using Machine Learning and Deep Learning** in partial fulfilment of the requirements for the award of B. Tech degree during the academic year 2023-2024. This work is carried out under my supervision and has not been submitted to any other University/Institute for award of any degree/diploma.

**GUIDE**
Dr. N . Nirmala Jyothi
Assistant Professor (AIML & IoT)
Dept. of CSE (AIML & IoT)

**Head of the Department**
Dr. Sagar Yeruva
Associate Professor
Dept. of CSE (AIML & IoT)

# DECLARATION

This is to certify that our project titled "**Plant Disease Detection and Classification using Machine Learning and Deep Learning.**" submitted to Vallurupalli Nageswara Rao Vignana Jyothi Institute of Engineering and Technology in complete fulfilment of the requirement for the award of Bachelor of Technology in CSE- (Artificial Intelligence and Machine Learning)is a bonafide report to the work carried out by us under the guidance and supervision of Dr.Sagar yeruva, Associate Professor, Department of CSE-(AIML & IoT), Vallurupalli Nageswara Rao Vignana Jyothi Institute of Engineering and Technology. To the best of our knowledge, this has not been submitted in any form to another University/Institute for an award of any degree/diploma.


Jayamangala Srujana
21071A66E5
Dept. of CSE (AIML&IoT)

Miyyapuram Srija
21071A66G0
Dept. of CSE (AIML&IoT)

Ravuri Joshini
21071A66H4
Dept. of CSE (AIML&IoT))

Chitneni Harshitha
22075A6616
Dept. of CSE (AIML&IoT)

# ABSTRACT

## INDEX

# 1. INTRODUCTION

Agriculture plays a vital role in the economy of many countries, providing food, raw materials, and employment. However, the health of crops is often threatened by various plant diseases, which can lead to significant losses in yield and quality. Early and accurate detection of plant diseases is crucial for effective management and control, minimizing the impact on agricultural production.

In this project, titled "Plant Disease Detection Using Machine Learning and Deep Learning (Mango Leaf)," we focus on developing a system that can automatically detect diseases in mango leaves. Mango is a widely cultivated fruit crop in many tropical and subtropical regions, and it is susceptible to various diseases, such as anthracnose, powdery mildew, and bacterial blight. Traditional methods of disease detection often rely on manual inspection, which can be time-consuming, labor-intensive, and prone to human error.

To address these challenges, this project leverages the power of machine learning (ML) and deep learning (DL) techniques to develop a robust and efficient disease detection system. By analyzing images of mango leaves, the system can identify and classify different types of diseases, enabling timely intervention and treatment. The project involves the collection and preprocessing of leaf images, the design and training of ML/DL models, and the implementation of a real-time detection system using a Raspberry Pi device.

The proposed system not only aims to improve the accuracy and speed of disease detection but also to make it accessible and scalable for use by farmers and agricultural professionals. By automating the detection process, the system can contribute to sustainable agriculture and help protect one of the most important fruit crops in the world.

## 1.1 OBJECTIVE :

The purpose of this project is to develop a Virtual Mouse application that targets a few aspects of significant development. For starters, this project aims to eliminate the needs of having a physical mouse while able to interact with the computer system through webcam by using various image processing techniques. Other than that, this project aims to develop a Virtual Mouse application that can be operational on all kind of surfaces and environment.

## 1.2 PURPOSE:

The Virtual Mouse application is expected to replace the current methods of utilizing a physical computer mouse where the mouse inputs and positions are done manually. This application offers a more effortless way to interact with the computer system, where every task can be done by gestures. Furthermore, the Virtual Mouse application could assist the motor-impaired users where he/she could interact with the computer system by just showing the correct combination of colours to the webcam.

## 1.3 SCOPE:

- **Mobile Application**: In future this web application also able to use on Android devices, where touchscreen concept is replaced by hand gestures.
- **Smart Movement:** Due to the current recognition process are limited within 25cm radius, an adaptive zoom in/out functions are required to improve the covered distance, where it can automatically adjust the focus rate based on the distance between the users and the webcam.
- The system can be used to control robots and automation systems without the usage of devices

# 2. SRS DOCUMENT

A software requirements specification (**SRS**) is a document that describes what the software will do and how it will be expected to perform.

## 2.1 FUNCTIONAL REQUIREMENTS:

### *2.1.1Image Preprocessing and Enhancement:*

- The system should preprocess leaf images to enhance their quality and make them suitable for analysis. This includes tasks such as normalization, noise reduction, resizing, color space conversion and segmentation of the leaf from the rest of the image. The preprocessing pipeline should handle various image formats and resolutions, ensuring consistency and reliability in the input data for the deep learning model.

### 2.1.2 Disease Detection and Classification:

- The deep learning model should be capable of accurately detecting and classifying different corneal diseases, such as Anthracnose,Powdery Mildew and Bacterial Leaf Spot based on the input images.The system should provide a confidence score for each prediction, allowing user to assess the reliability of the detection.

### 2.1.3 Database Management:

- A database for storing mango leaf images, diseases they effect, and classification results. helps in improving detecting/identification accuracy over time.

### 2.1.4 Model Evaluation and Validation:

- Systematically evaluate ML and DL models to determine the optimal solution for disease detection.Ensure robust evaluation by partitioning data into training and validation sets.Assess the effectiveness of trained models using various metrics by calculating accuracy, precision, recall, F1-score, and generate confusion matrices.

## 2.2 NON-FUNCTIONAL REQUIREMENTS:

### *1 Performance*

- **Response Time**:The system should provide disease detection results within 2-5 seconds after an image is uploaded or captured. This includes time for image pre processing, feature extraction, and classification.

- **Throughput**:The system should be able to process at least 100 images per minute, especially in batch processing scenarios.
- **Scalability**:The system must scale to handle increased loads, such as a higher number of concurrent users or larger datasets, without significant degradation in performance.
- **Efficiency**:The system should be optimized to use minimal computational resources,especially when deployed on resource-constrained devices like Raspberry Pi.

### *Reliability*

- **Availability**:The system should have an uptime of at least 99.9%, ensuring it is available for use whenever needed.
- **Fault Tolerance**: The system should be resilient to hardware or software failures, automatically recovering from errors and maintaining functionality.
- **Accuracy**:The disease detection and classification models should achieve a minimum accuracy of 90%, with regular updates to maintain or improve this accuracy.
- **Consistency**: The system should provide consistent results for the same input images under similar conditions, ensuring that the detection process is repeatable and reliable.

### *Security*

- **Data Security**:User data, including images and personal information, must be encrypted both in transit and at rest.
- **Authentication and Authorization**: The system should implement user authentication and role-based access control to restrict access to sensitive data and functionalities.
- **Data Privacy**:The system should comply with relevant data protection regulations, such as GDPR, ensuring that user data is handled responsibly and only for the intended purposes.
- **Incident Response**: In case of a security breach, the system should have a documented incident response plan to mitigate the effects and notify affected users promptly.

### *Usability*

- **User Interface (UI) Design**: The UI should be intuitive, with clear instructions and easy navigation, enabling users with minimal technical knowledge to operate the system effectively.
- **Accessibility**: The system should be accessible to users with disabilities, adhering to standards such as the Web Content Accessibility Guidelines (WCAG).

- **User Documentation**:Comprehensive user guides and help resources should be available to assist users in understanding and using the system effectively.
- **Localization**: The system should support multiple languages, making it accessible to users from different regions.

## 2.3 HARDWARE REQUIREMENTS:

- Processor: Intel i5 or higher/GPU
- Laptop with webcam or an external webcam device for Desktop PC
- RAM: 8GB or higher
- Disk space: >=256 GB

## 2.4 SOFTWARE REQUIREMENTS:

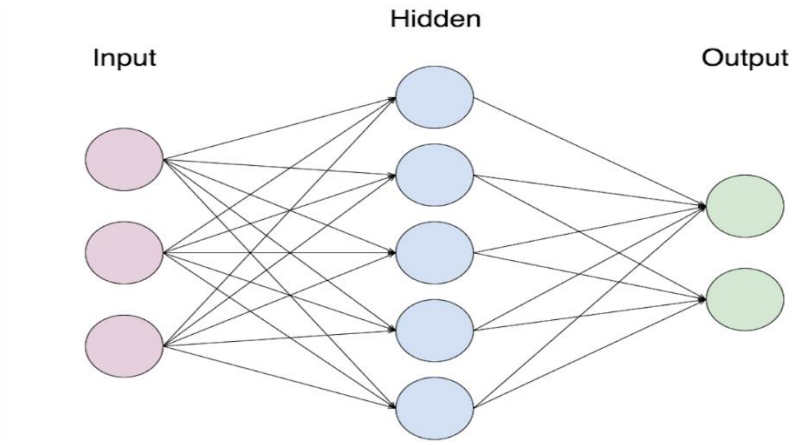
- Operating system: WINDOWS/LINUX(PREFERABLY)
- Programming languages: Python
- IDE: Visual studio, Anaconda

# 3. ANALYSIS

## 3.1 EXISTING SYSTEM:

In Existing system we only define the diseases of the diseased plant, but it may also contain medicinal properties .Optimizes model parameters using training dataset. Integrates trained models into software or hardware systems for automated disease detection. Compute texture features such as Gabor filters, Local Binary Patterns (LBP), or Gray Level Co-occurrence Matrix (GLCM) to capture patterns and textures in the plant images. Diseases often manifest as changes in texture, such as spots, lesions, or deformations. Analyze leaf venation patterns using techniques like fractal analysis or skeletonization to detect abnormalities caused by diseases affecting the vascular system of plants.

## 3.2 PROPOSED SYSTEM:




In this proposed project ,we are going to implement a real time model where we can identify the disease of a plant(especially in Mango Leaf).Explore the properties of plants known to possess disease-resistant qualities or compounds with potential therapeutic benefits. Engage with farmers, agricultural extension services, industry stakeholders, and healthcare professionals to understand their needs, challenges, and priorities related to plant disease management and medicinal plant utilization. Integration of advanced technologies with human expertise can lead to more effective and equitable plant disease detection and its medicinal properties.

Achieve high accuracy in disease classification and ensure the system's ability to differentiate between healthy and diseased plants. Create a user-friendly interface to facilitate easy input of images and provide clear disease detection results. Develop a robust and automated system for plant/crop disease detection using deep learning techniques.

In this system, we included that if a plant is diseased how it can be prevented Early detection and diagnosis of these diseases play a crucial role in implementing timely interventions and preventing extensive damage to the plant/crop.

# LITERATURE SURVEY

## RELATED WORK

We referenced seven distinct research papers encompassing disease detection, and early detection using Deep Learning Algorithms. These studies shed light on various methodologies, pros, and cons associated with their respective approaches.

| Title of the paper | Reference with DOI | Objective of the paper | Methodology | Published year |
|---|---|---|---|---|
| Medicinal Plant Parts Identification and Classification using Deep Learning Based on Multi Label Categories | Medicinal Plant Parts Identification and Classification using Deep Learning Based on Multi Label Categories \| Abebe Workneh - Academia.edu | The identification of the medicinal plant parts can be done using image processing and chemical ingredient extraction. But using its chemical ingredient consumes time and needs high expenditure for laboratory equipment. So, to reduce this problem image processing is the preferred approach. This approach can be handled with the help of deep learning. | Mobile Net with accuracy - 93% for training sets and 92% for testing sets. | 2021 |
| Mango Leaf Diseases Detection using Deep Learning | https://www.researchgate.net/profile/Sampada_Gulavani3/publication/362536693_Deep_Learning_for_Image_Based_Mango_Leaf_Disease_Detection/links/62ef2c5288b83e7320b5ac90/Deep-Learning-for-Image-Based-Mango-Leaf-Disease-Detection | By using technology one can detect diseases on a large scale. In the case of mango leaves; there are various types of diseases [4] present like powdery mildew, anthracnose, red rust etc. In the present work, a deep learning (DL) based model has been proposed for the classification of various mango leaf diseases (powdery mildew, anthracnose, red rust) at the initial stages. Accuracy, Recall, Precision and F-Score have been used to evaluate the model. | image classification, object detection,segmentation etc. The proposed CNN based model attains 90.36% of accuracy.. | 2022 |
| Image processing for smart farming: Detection of disease and fruit grading | https://sci-hub.se/10.1109/iciip.2013.6707647 | Due to the increasing demand industry, the need to effectively grow a plan yield is very important. In order to do so, monitor the plant during its growth period, as of harvest. In this paper image processing is monitor the diseases on fruits during far plantation | Back prop used for weight adjustment of training database. | 2013 |

| | | | | |
|---|---|---|---|---|
| | | to harvesting.Practical implementation has been done using MATLAB. | | |
| Comparison Of Machine Learning Algorithms For Detection Of Medicinal Plants | https://sci-hub.se/10.1109/iccmc48092.2020.iccmc-00010 | Various diseases that can be cured with these are also vast in number. So these plants have to be identified and preserved. If an automatic recognition system is available for feature extraction and classification of medicinal plants from the images, then it will be comfortable. | KNN and SVM are used for classification and a comparison is madeamong their performances. It was obtained as 93.23% accuracy for SVM. | 2020 |
| Classification of Watermelon Leaf Diseases Using Neural Network Analysis | https://sci-hub.se/10.1109/beiac.2013.6560170 | A few of infected leaf samples were collected and they were captured using a digital camera with specific calibration procedure under controlled environment. The classification on the watermelon's leaf diseases is based on color feature extraction from RGB color model where the RGB pixel color indices have been extracted from the identified Regions of Interest (ROI). | This work have shown that the type of leaf diseases achieved 75.9% of accuracy. | 2013 |
| Mango Leaf Diseases Identification Using Convolutional Neural Network | https://www.academia.edu/download/76332505/731.pdf | Convolutional Neural Network architectures such as VGG-16, VGG-19, Alexnet, a deep CNN architecture with three hidden layers has been proposed for this work. There is no precise rule in organizing the structure of the individual layer. | The testing accuracy resulted in 96.67%. | 2018 |
| BDMediLeaves: A leaf images dataset for Bangladeshi medicinal plants identification | https://www.sciencedirect.com/ | medicinal plant identification, ML, DL, and computer vision research, where it can serve as a benchmark for image recognition algorithms and classification models. Furthermore, this dataset holds relevance for public awareness and education initiatives, as well as for fields extending beyond traditional medicine, including agriculture, environmental monitoring, and remote sensing | architecture of pre-trained DenseNet201-FCNs and pre-trained InceptionResNetV2-FCNs, the pre-trained DenseNet201-FCNs, converges faster compared to the pre-trained InceptionResNetV2-FCNs architecture. | 2013 |

## SYSTEM STUDY

The system developed for mango leaf disease detection using machine learning and deep learning techniques is designed to efficiently identify and classify diseases based on images. The project involves several key components, starting with data collection, where images of both healthy and diseased mango leaves are gathered. These images undergo preprocessing steps such as resizing, noise reduction, and augmentation to enhance the quality and relevance of the features used for analysis. The core of the system is the model development phase, where machine learning and deep learning models are trained on the preprocessed images. These models are designed to accurately detect and classify various diseases by learning from the features extracted from the images. The system is capable of early disease detection, which is crucial for improving crop management and yield.

# 4. SOFTWARE DESCRIPTION

## 4.1 OpenCv:

OpenCV (Open-Source Computer Vision Library) is an open-source computer vision and machine learning software library. OpenCV was built to provide a common infrastructure for computer vision applications and to accelerate the use of machine perception in the commercial products. Being a BSD-licensed product, OpenCV makes it easy for businesses to utilize and modify the code.

The library has more than 2500 optimized algorithms, which includes a comprehensive set of both classic and state-of-the-art computer vision and machine learning algorithms. These algorithms can be used to detect and recognize faces, identify objects, classify human actions in videos, track camera movements, track moving objects, extract 3D models of objects, produce 3D point clouds from stereo cameras, stitch images together to produce a high resolution image of an entire scene, find similar images from an image database, remove red eyes from images taken using flash, follow eye movements, recognize scenery and establish markers to overlay it with augmented reality, etc. OpenCV has more than 47 thousand people of user community and estimated number of downloads exceeding 18 million. The library is used extensively in companies, research groups and by governmental bodies.

Along with well-established companies like Google, Yahoo, Microsoft, Intel, IBM, Sony, Honda, Toyota that employ the library, there are many start-ups such as Applied Minds, VideoSurf, and Zeitera, that make extensive use of OpenCV. OpenCV's deployed uses span the range from stitching street view images together, detecting intrusions in surveillance video in Israel, monitoring mine equipment in China, helping robots navigate and pick up objects at Willow Garage, detection of swimming pool drowning accidents in Europe, running interactive art in Spain and New York, checking runways for debris in Turkey, inspecting labels on products in factories around the world on to rapid face detection in Japan.

It has C++, Python, Java and MATLAB interfaces and supports Windows, Linux, Android and Mac OS. OpenCV leans mostly towards real-time vision applications and takes advantage of MMX and SSE instructions when available. A full-featured CUDA and OpenCL interfaces are being actively developed right now. There are over 500 algorithms and about 10 times as many functions that compose or support those algorithms. OpenCV is written natively in C++ and has a templated interface that works

## 4.2 Models

For detecting and classifying leaf diseases, several deep learning models can be used:

- **CNNs (Convolutional Neural Networks)**: Ideal for image feature extraction and classification.
- **VGG-16**: A deep CNN with 16 layers, effective for detailed image analysis.

- **ResNet50**: Uses residual connections to train deeper networks efficiently, preventing vanishing gradients.
- **Ensemble Learning**: Fine-tuning pre-trained models (e.g., ResNet, VGG) on plant leaf image datasets for improved accuracy.

These models can be combined or fine-tuned to enhance performance in corneal disease detection.

**Deep Learning Models**

**CNN**

Convolutional Neural Networks (CNNs) are ideal for detecting and classifying corneal diseases due to their ability to automatically extract spatial features from images. In implementation, start with data preprocessing, including resizing, normalization, and augmentation to improve model robustness. The CNN architecture typically consists of convolutional layers to extract features, followed by ReLU activation functions for non-linearity, and pooling layers to reduce spatial dimensions. The extracted features are then passed through fully connected layers for classification. CNNs are effective due to their ability to learn hierarchical features directly from the input images, making them well-suited for medical imaging tasks.

**RNN**

Recurrent Neural Networks (RNNs) are suitable for tasks involving sequential data, making them useful if your project involves analyzing sequences of corneal images over time. In implementation, RNNs process input sequences one step at a time, maintaining a hidden state that captures information about previous inputs. Start with data preprocessing, ensuring sequences are consistently formatted. The RNN architecture includes recurrent layers, which apply the same weights to each input in the sequence, allowing the model to learn temporal patterns. Outputs are typically passed through fully connected layers for final classification. RNNs excel at capturing temporal dependencies in data.

**VGG-16**

VGG16 is a deep convolutional neural network architecture renowned for its simplicity and effectiveness in image classification tasks.VGG16 consists of 16 weight layers, including 13 convolutional layers and 3 fully connected layers. The network employs 3x3 convolutional filters throughout, with max pooling layers to downsample feature maps. It is commonly used for transfer learning and feature extraction, thanks to its pretrained models on large datasets like ImageNet.

**ResNet50**

ResNet50 is a deep learning model part of the Residual Network (ResNet) family, designed to solve the problem of vanishing gradients in very deep networks. It consists of 50 layers, including convolutional layers, batch normalization, and shortcut connections (or "residual connections") that skip one or more layers, allowing the model to learn identity mappings. This enables training of deeper networks without degradation in performance.

**Performance Metrics**

Performance metrics provide a quantitative assessment of model effectiveness. Key metrics include accuracy, measuring overall correctness; precision, indicating true positive rate; recall, gauging the ability to identify positives; and F1 score, blending precision and recall.

- Accuracy: Measures the ratio of correctly predicted instances to the total instances.

  Formula: (TP + TN) / (TP + TN + FP + FN)

- Precision: Reflects the proportion of true positives among the predicted positive instances.

  Formula: TP / (TP + FP)

- Recall (Sensitivity or True Positive Rate): Represents the ratio of true positives among actual positive instances.

  Formula: TP / (TP + FN)

- F1 Score: Balances precision and recall, offering a harmonic mean of the two metrics. Formula: 2 * (Precision * Recall) / (Precision + Recall)

## 4.3 PYTHON:

Python is an interpreted, object-oriented, high-level programming language with dynamic semantics. Its high-level built in data structures, combined with dynamic typing and dynamic binding, make it very attractive for Rapid Application Development, as well as for use as a scripting or glue language to connect existing components together. Python's simple, easy to learn syntax emphasizes readability and therefore reduces the cost of program maintenance. Python supports modules and packages, which encourages program modularity and code reuse. The Python interpreter and the extensive standard library are available in source or binary form without charge for all major platforms, and can be freely distributed.

Often , programmers fall in love with Python because of the increased productivity it provides. Since there is no compilation step, the edit-test-debug cycle is incredibly fast. Debugging Python programs is easy: a bug or bad input will never cause a segmentation fault. Instead, when the interpreter discovers an error, it raises an exception..

# 5. PROJECT DESCRIPTION

## Methodology:

The methodology of project can be divided into following:

In our project on detecting and classifying leaf diseases, we adopt a comprehensive approach to ensure precise and reliable results. Our methodology encompasses thorough data preprocessing, careful data splitting, and rigorous hyperparameter tuning, followed by extensive model testing. This structured approach integrates advanced deep learning techniques to enhance accuracy and clinical relevance. By leveraging state-of-the-art models like CNNs, VGG-19, and ResNet-18, we aim to push the boundaries of machine learning in plant medical diagnostics. Our framework not only highlights the capabilities of deep learning but also advances the field of leaf disease diagnosis and treatment.

Data Preprocessing

Data preprocessing is crucial for the success of our plant disease detection project. This phase involves meticulous steps to ensure data quality and relevance. We perform thorough cleaning to remove noise and inconsistencies, normalize image data to standardize pixel values, and address any missing or incomplete data. By preparing our datasets with these rigorous techniques, we ensure that the deep learning models receive clean, consistent, and high-quality input, which is essential for accurate and reliable disease prediction and classification.

**Data Splitting**

For our leaf disease detection project, we partition the dataset using an 80:20 split, allocating 80% of the data for training and 20% for testing. This stratified approach ensures that both subsets accurately represent the various disease classes, minimizing bias and enhancing model performance.

Training Set: The larger portion of the data is used for training our models, allowing them to learn and identify patterns specific to leaf diseases.

Testing Set: The remaining data is reserved for testing. It serves as an independent evaluation tool to assess the model's ability to generalize and make accurate predictions on unseen data. This split is crucial for validating the model's performance and reliability in real-world scenarios.

**Algorithm selection**

In our project on detecting and classifying leaf diseases, choosing the right deep learning algorithms is crucial for accurate disease prediction. We select algorithms based on the specific characteristics of leaf images and the complexities involved in distinguishing between various leaf conditions. Our approach involves employing a range of deep learning models, including CNNs, VGG-16, ResNet50 and Ensemble Learning to compare their performance. This selection process is driven by the need to capture diverse image features and patterns, ensuring that our models are well-suited to the unique challenges of leaf disease classification and can deliver reliable, clinically relevant results.

**Plant Leaf Disease Prediction:**

Given the need to accurately identify and classify various leaf diseases, we employed several advanced deep learning models to address the unique challenges posed by leaf images.

**Hyper parameter Tuning**

Different algorithms demanded distinct parameters to optimize their predictive capabilities.

**GridSearchCV Approach**

To optimize model performance, we utilized the GridSearchCV technique to systematically explore hyperparameter combinations. This approach ensures thorough evaluation of various parameters to identify the most effective settings for each model.

**CNN Model**

For CNNs like VGG-16 and ResNet50, we fine-tuned parameters such as learning rates, batch sizes, and the number of epochs. GridSearchCV helped in determining the optimal configurations that enhance model accuracy and prevent overfitting.

**Model Testing**

**Hyperparameter-Informed Training**

In our leaf disease detection project, the model testing phase was significantly enhanced by using hyper parameters optimized through GridSearchCV. These finely tuned settings played a crucial role in training our deep learning models, such as CNNs like VGG-16 and ResNet50, improving their predictive accuracy.

**Optimal Configurations Empowerment**

By leveraging hyperparameters identified through GridSearchCV, we trained our models with configurations tailored to the specific needs of each architecture. This strategic approach allowed the models to learn from the data more effectively, enabling them to accurately differentiate between various corneal diseases.

**Accurate and Generalized Performance**

Utilizing GridSearchCV-derived hyperparameters ensured that our models not only excelled on the training dataset but also generalized well to unseen test data. This balance between training and generalization improved the models' adaptability and precision in real-world scenarios.

**Mitigation of Overfitting**

The incorporation of GridSearchCV-optimized hyperparameters was crucial in mitigating overfitting. By finding the right balance between model complexity and generalization, our models achieved better stability and performance, reducing the risk of overfitting to the training data.

**Elevated Prediction Metrics**

Model testing results demonstrated the effectiveness of GridSearchCV-informed hyperparameters, with notable improvements in accuracy, precision, and recall. These enhanced metrics highlight the models' capability to provide reliable and actionable predictions for leaf diseases.

**Pathway to Reliable Insight**

The combination of well-tuned hyperparameters and thorough model testing underscored the success of our approach. By ensuring optimal performance and minimizing errors, our methodology paves the way for dependable insights into corneal disease detection, potentially transforming healthcare practices.

# TESTING

## TEST RESULTS

The culmination of our Deep Learning model development lies in the rigorous testing and evaluation phase. This pivotal step aims to ascertain the predictive capabilities and performance of our models, further contributing to the refinement of disease prediction accuracy. Our evaluation metrics encompass a comprehensive suite of measurements, including:

**Accuracy**

Accuracy showcases the proportion of correctly predicted outcomes. A high accuracy rate signifies the models precision in differentiating disease classes and underscores their potential as diagnostic tools.

Formula: (TP + TN) / (TP + TN + FP + FN)

Accuracy of models for leaf Diseases:

| Model | Accuracy |
|---|---|
| VGG-16 | 81.95 |
| Resnet50 | 86.04 |
| Ensemble Learning (VGG16,Resnet50) | 98.39 |

# 6.SYSTEM DESIGN

## 6.1 INTRODUCTION TO UML:

Unified Modeling Language **(UML)** is a general-purpose modeling language. The main aim of UML is to define a standard way to **visualize** the way a system has been designed. It is quite similar to blueprints used in other fields of engineering. UML is not a programming language, it is rather a visual language. We use UML diagrams to portray the behaviour and structure of a system.

## 6.2 BUILDING BLOCKS OF THE UML:

The vocabulary of the UML encompasses three kinds of building blocks:

- Things
- Relationships
- Diagrams

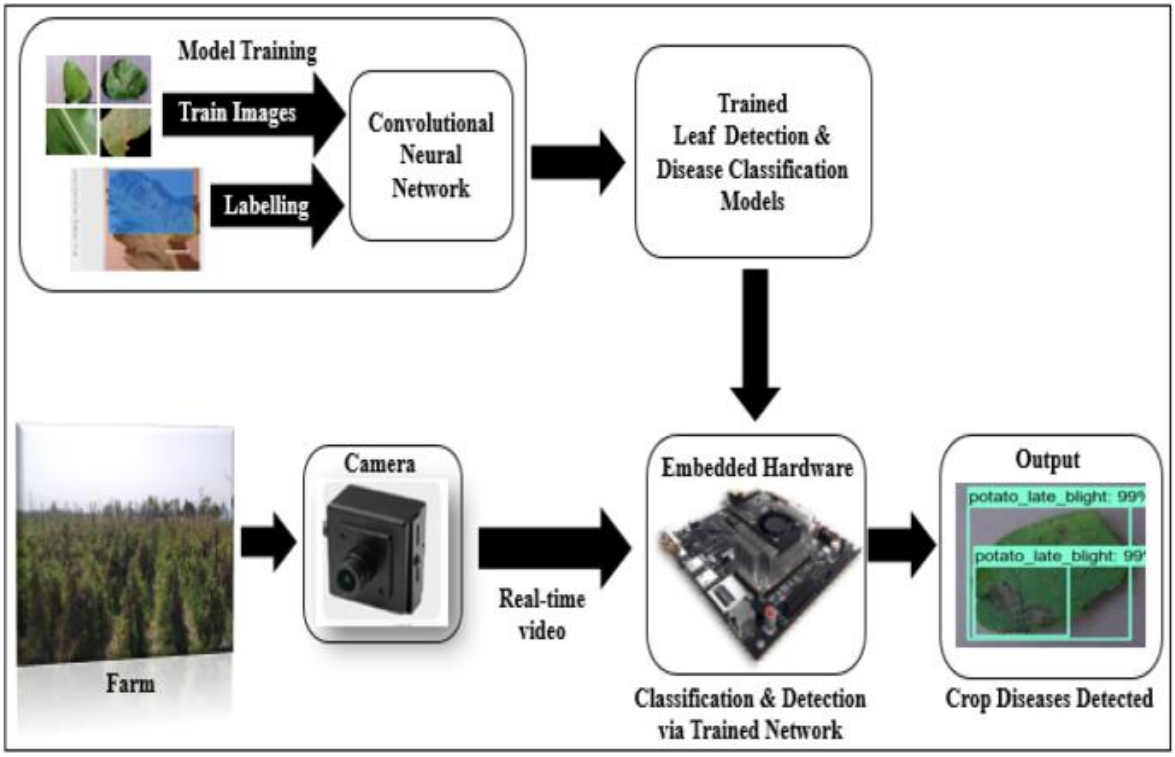
Things are the abstractions that are first-class citizens in a model; relationships tie these things together; diagrams group interesting collections of things.

**Things in the UML:**
There are four kinds of things in the UML:

Structural things
Behavioural things
Grouping things
Annotational things
These things are the basic object-oriented building blocks of the UML. You use them to write well-formed models.

## System Architecture:

System architecture refers to the conceptual model that defines the structure, behavior, and more views of a system. It encompasses the arrangement of components and their relationships, both internal and external, ensuring that the system meets the necessary requirements. A well-designed system architecture clearly outlines the high-level organization of the system, including key elements like hardware, software, data flow, communication protocols, and user interfaces. It acts as a blueprint, guiding developers and engineers in building and maintaining the system, ensuring scalability, performance, and security.




## 6.3 UML DIAGRAMS:

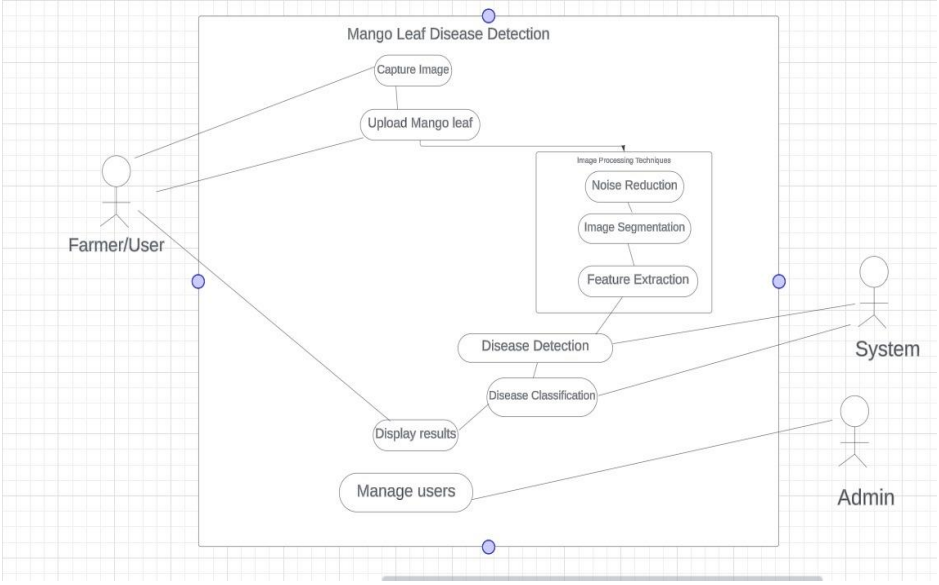
UML is a modern approach to modeling and documenting software. It is based on **diagrammatic representations** of software components. It is the final output and the diagram represents the system.

### UML includes the following

- Use case diagram
- Sequence diagram
- Flow diagram
- Activity diagram

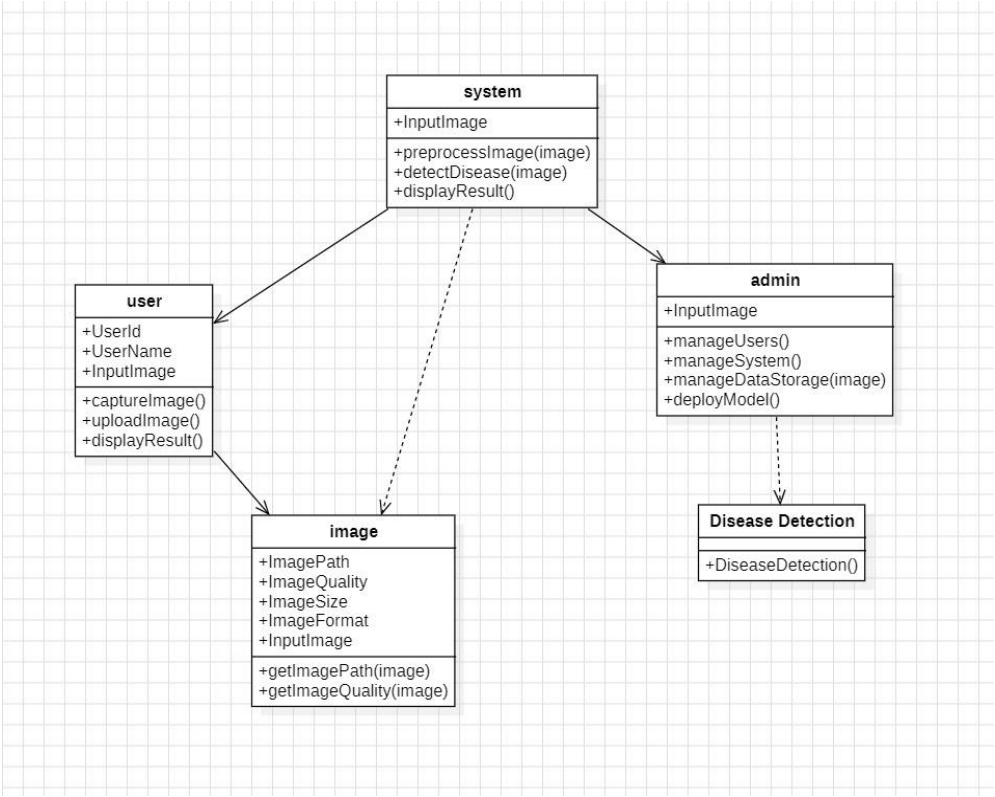
## USECASE DIAGRAM:

They are usually used to illustrate the various actions taken by the application. They also show the several users who can carry out these functions. Use-case diagrams fall under behaviour diagrams due to their emphasis on the tasks carried out and the users (actors) who carry out these tasks.

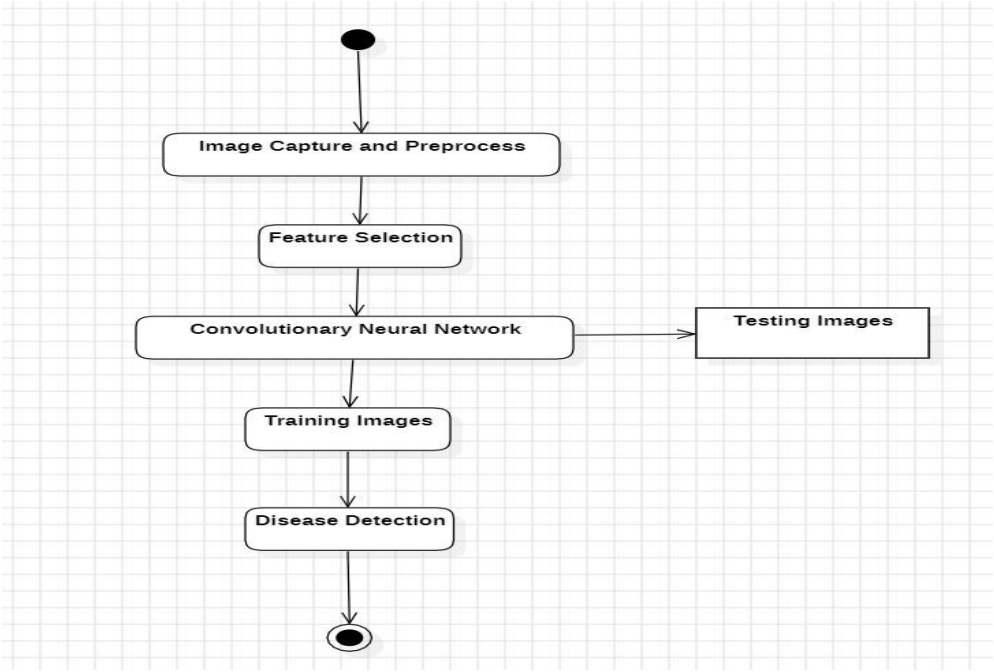



A **use case diagram** is a graphical depiction of a user's possible interactions with a system. A use case diagram shows various use cases and different types of users the system has and will often be accompanied by other types of diagrams as well. The use cases are represented by either circles or ellipses. The actors are often shown as stick figures.

## CLASS DIAGRAM:



## ACTIVITY DIAGRAM:




**Activity diagrams** are graphical representations of workflows of stepwise activities and actions with support for choice, iteration and concurrency. In the Unified Modeling Language, activity diagrams are intended to model both computational and organizational processes (i.e., workflows), as well as the data flows intersecting with the related activities. Although activity diagrams primarily show the overall flow of control, they can also include elements showing the flow of data between activities through one or more data stores.

# 7. DEVELOPMENT

## 7.1 OVERVIEW:

```
[ ] from google.colab import drive
    drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
!pip install gdown
```

```
Requirement already satisfied: gdown in /usr/local/lib/python3.10/dist-packages (5.1.0)
Requirement already satisfied: beautifulsoup4 in /usr/local/lib/python3.10/dist-packages (from gdown) (4.12.3)
Requirement already satisfied: filelock in /usr/local/lib/python3.10/dist-packages (from gdown) (3.15.4)
Requirement already satisfied: requests[socks] in /usr/local/lib/python3.10/dist-packages (from gdown) (2.32.3)
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-packages (from gdown) (4.66.5)
Requirement already satisfied: soupsieve>1.2 in /usr/local/lib/python3.10/dist-packages (from beautifulsoup4->gdown) (2.6)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests[socks]->gdown) (3.3.2)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests[socks]->gdown) (3.7)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests[socks]->gdown) (2.0.7)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests[socks]->gdown) (2024.7.4)
Requirement already satisfied: PySocks!=1.5.7,>=1.5.6 in /usr/local/lib/python3.10/dist-packages (from requests[socks]->gdown) (1.7.1)
```

```
[ ] import gdown

    # URL of the Google Drive file (adjusted for direct download)
    file_url = 'https://drive.google.com/file/d/1N07TnHVnUGGLPlbcYt55RjaHHv5stJ1l/view?usp=drive_link'

    # Path to save the downloaded file
    output = 'C:\Srujana_J\mango_data'

    # Download the file
    gdown.download(file_url, output, quiet=False)
```

```
import zipfile
import os

# Path to the downloaded ZIP file
zip_file_path = r'https://drive.google.com/file/d/1N07TnHVnUGGLPlbcYt55RjaHHv5stJ1l/view?usp=drive_link'

# Define the extraction directory
extract_to = r'https://drive.google.com/drive/folders/1oyT7cdDXiyByhzNurbyMBSRs7FLXfQvX?usp=drive_link'

# Extract the contents if it's a zip file
if zipfile.is_zipfile(zip_file_path):
    with zipfile.ZipFile(zip_file_path, 'r') as zip_ref:
        zip_ref.extractall(extract_to)
```

```
[ ] import os
    import time
    import shutil
    import pathlib
    import itertools
```

```
[ ] import cv2
    import numpy as np
    import pandas as pd
    import seaborn as sns
    sns.set_style('darkgrid')
    import matplotlib.pyplot as plt
    from sklearn.model_selection import train_test_split
    from sklearn.metrics import confusion_matrix, classification_report

    # import Deep learning Libraries
    import tensorflow as tf
    from tensorflow import keras
    from tensorflow.keras.models import Sequential
    from tensorflow.keras.optimizers import Adam, Adamax
    from tensorflow.keras.metrics import categorical_crossentropy
    from tensorflow.keras.preprocessing.image import ImageDataGenerator
    from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Activation, Dropout, BatchNormalization
```

```python
def load_and_resize_images_from_folder(folder, target_size=(224, 224)):
    images = []
    labels = []
    for class_label in os.listdir(folder):
        class_path = os.path.join(folder, class_label)
        if os.path.isdir(class_path):
            for filename in os.listdir(class_path):
                img_path = os.path.join(class_path, filename)
                img = cv2.imread(img_path)
                img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)  # Convert BGR to RGB
                img = cv2.resize(img, target_size)  # Resize the image
                images.append(img)
                labels.append(class_label)
    return np.array(images), np.array(labels)
data_path="/content/drive/MyDrive/MangoLeaf folder/MangoLeaf_Dataset"

x,y=load_and_resize_images_from_folder(data_path)
```

```python
[ ] x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.2,random_state=42)
    x_train,x_valid,y_train,y_valid=train_test_split(x_train,y_train,test_size=0.2,random_state=42)
```

```python
[ ] import random
    # displaying the train data
    random_indices = random.sample(range(len(x_train)), 15)


    plt.figure(figsize=(15, 8))
    for i, index in enumerate(random_indices, 1):
        plt.subplot(3,5, i)
        plt.imshow(x_train[index])
        plt.title(f'class: {y_train[index]}')
        plt.axis('off')

    plt.show()
```


```
from tensorflow.keras.applications import VGG16, ResNet50

# Load VGG16 model
vgg16_base = VGG16(include_top=False, weights="imagenet", input_shape=(224, 224, 3))
vgg16_base.trainable = False

# Load ResNet50 model
resnet50_base = ResNet50(include_top=False, weights="imagenet", input_shape=(224, 224, 3))
resnet50_base.trainable = False
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vgg16/vgg16_weights_tf_dim_ordering_tf_kernels_notop.h5
58889256/58889256 ━━━━━━━━━━━━━━━━━━━━ 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/resnet/resnet50_weights_tf_dim_ordering_tf_kernels_notop.h5
94765736/94765736 ━━━━━━━━━━━━━━━━━━━━ 1s 0us/step
```

```
[ ] from tensorflow.keras.layers import Input, Concatenate, GlobalAveragePooling2D
    from tensorflow.keras.models import Model
    from tensorflow.keras import regularizers

    # Define input layer
    input_layer = Input(shape=(224, 224, 3))

    # Extract features using VGG16
    vgg16_features = vgg16_base(input_layer)
    vgg16_features = GlobalAveragePooling2D()(vgg16_features)

    # Extract features using ResNet50
    resnet50_features = resnet50_base(input_layer)
    resnet50_features = GlobalAveragePooling2D()(resnet50_features)

    # Concatenate the features
    combined_features = Concatenate()([vgg16_features, resnet50_features])

    # Add a Dense layer with regularization
    output_layer = Dense(256, kernel_regularizer=regularizers.l2(0.016),
                         activity_regularizer=regularizers.l1(0.006),
                         bias_regularizer=regularizers.l1(0.006), activation='linear')(combined_features)
```

```
# Create the model
model = Model(inputs=input_layer, outputs=output_layer)
```

```
x_train_features = model.predict(x_train)
x_validation_features = model.predict(x_valid)
x_test_features = model.predict(x_test)

# Flatten the features
x_train_flatten = x_train_features.reshape(x_train_features.shape[0], -1)
x_validation_flatten = x_validation_features.reshape(x_validation_features.shape[0], -1)
x_test_flatten = x_test_features.reshape(x_test_features.shape[0], -1)
```

```
75/75 ━━━━━━━━━━━━━━━━━━━━ 1704s 23s/step
19/19 ━━━━━━━━━━━━━━━━━━━━ 422s 22s/step
24/24 ━━━━━━━━━━━━━━━━━━━━ 520s 22s/step
```

```
x_train.shape
```

```
(2386, 224, 224, 3)
```

```
from tensorflow.keras.utils import to_categorical
from sklearn.preprocessing import LabelEncoder

label_encoder = LabelEncoder()
y_train_numeric = label_encoder.fit_transform(y_train)
y_test_numeric = label_encoder.fit_transform(y_test)
y_valid_numeric = label_encoder.fit_transform(y_valid)
y_train_hot = to_categorical(y_train_numeric, 8)
y_test_hot=to_categorical(y_test_numeric, 8)
y_valid_hot=to_categorical(y_valid_numeric, 8)
```

```
x_train.shape
```

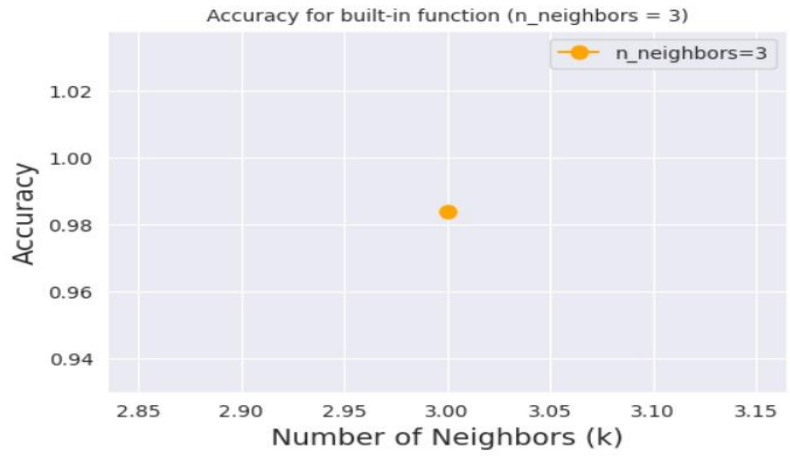
```
(2386, 224, 224, 3)
```

```
from tensorflow.keras.utils import to_categorical
from sklearn.preprocessing import LabelEncoder

label_encoder = LabelEncoder()
y_train_numeric = label_encoder.fit_transform(y_train)
y_test_numeric = label_encoder.fit_transform(y_test)
y_valid_numeric = label_encoder.fit_transform(y_valid)
y_train_hot = to_categorical(y_train_numeric, 8)
y_test_hot=to_categorical(y_test_numeric, 8)
y_valid_hot=to_categorical(y_valid_numeric, 8)
```

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.metrics import accuracy_score, multilabel_confusion_matrix


knn_classifier = KNeighborsClassifier(n_neighbors=3)

# Fitting
knn_classifier.fit(x_train_flatten, y_train_hot)

# predictions on test-set
y_pred = knn_classifier.predict(x_test_flatten)

# evaluate
accuracy = accuracy_score(y_test_hot, y_pred)
print(f'Accuracy: {accuracy}')
```

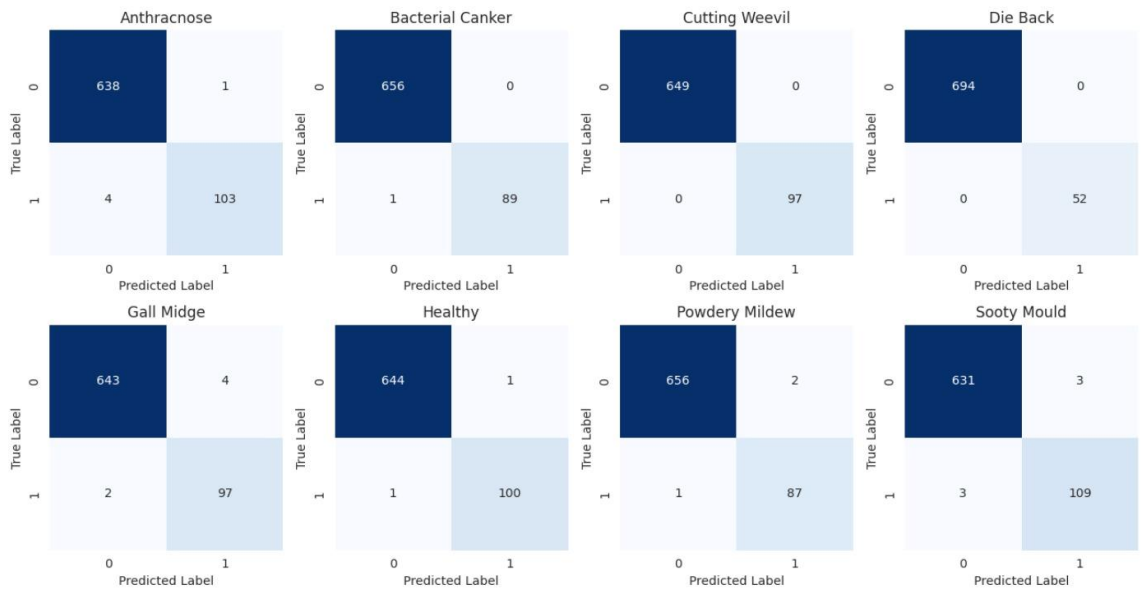
```
Accuracy: 0.9839142091152815
```

```
plt.figure(figsize=(6, 4))
plt.plot([3], [accuracy], marker='o', color='orange', label='n_neighbors=3', markersize=8)
plt.title('Accuracy for built-in function (n_neighbors = 3)', fontsize=10)
plt.xlabel('Number of Neighbors (k)', fontsize=14)
plt.ylabel('Accuracy', fontsize=14)
plt.grid(True)
plt.legend()
plt.show()

print("\n")
print(f'Accuracy for K=3 : {accuracy}')
```

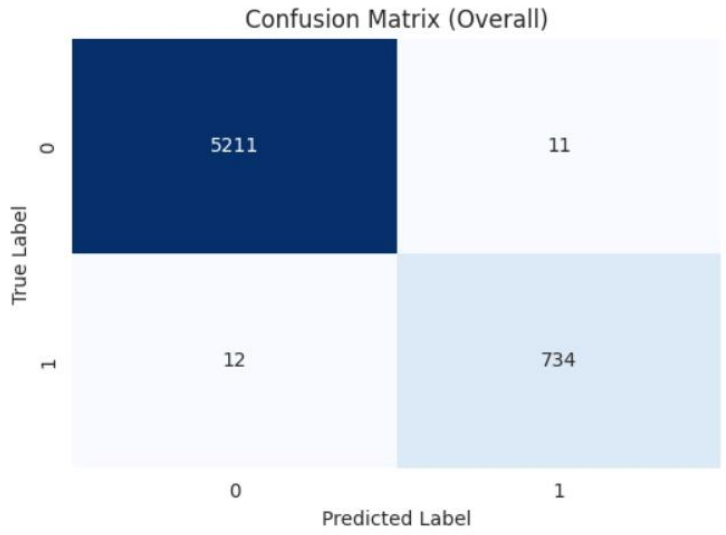



```
Accuracy for K=3 : 0.9839142091152815
```

```
class_names = ["Anthracnose", "Bacterial Canker", "Cutting Weevil", "Die Back",
               "Gall Midge", "Healthy", "Powdery Mildew", "Sooty Mould"]

conf_matrices = multilabel_confusion_matrix(y_test_hot, y_pred)

fig, axes = plt.subplots(nrows=2, ncols=4, figsize=(13, 7))

for i, (conf_matrix, ax) in enumerate(zip(conf_matrices, axes.flatten())):
    sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', cbar=False,
                xticklabels=['0', '1'], yticklabels=['0', '1'], ax=ax)
    ax.set_title(f'{class_names[i]}', fontsize=12)
    ax.set_xlabel('Predicted Label', fontsize=10)
    ax.set_ylabel('True Label', fontsize=10)

plt.tight_layout()
plt.show()
```




```python
from sklearn.metrics import multilabel_confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt

conf_matrix = multilabel_confusion_matrix(y_test_hot, y_pred)

# Plot the confusion matrix using seaborn
plt.figure(figsize=(6, 4))
sns.heatmap(conf_matrix.sum(axis=0), annot=True, fmt='d', cmap='Blues', cbar=False,
            xticklabels=['0', '1'], yticklabels=['0', '1'])
plt.title('Confusion Matrix (Overall)')
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.show()
```



```python
from sklearn.metrics import recall_score, precision_score, f1_score

recall = recall_score(y_test_hot, y_pred, average='macro')

precision = precision_score(y_test_hot, y_pred, average='macro')

f1 = f1_score(y_test_hot, y_pred, average='macro')

print("Recall:", recall)
print("Precision:", precision)
print("F1 score:", f1)
```

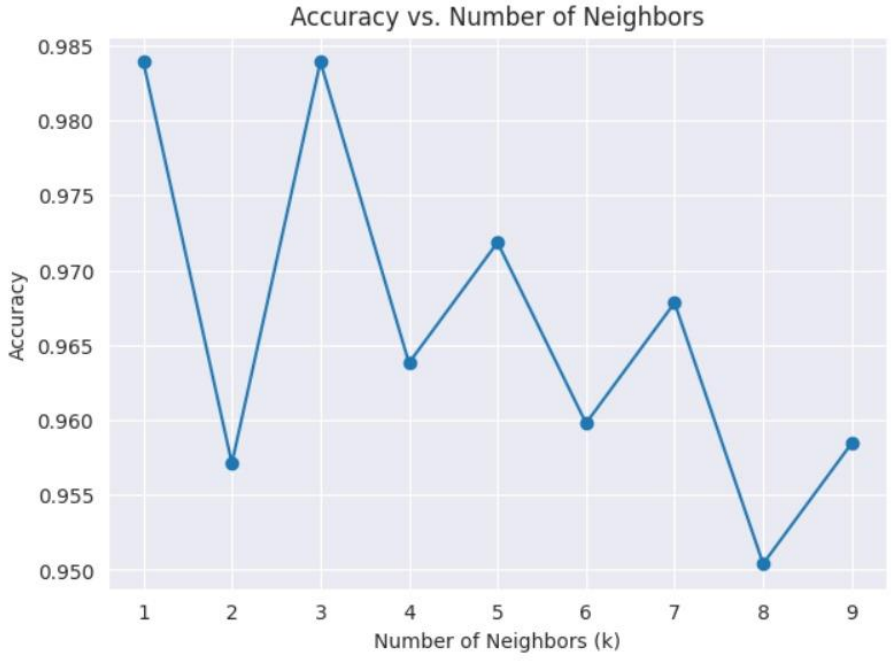
```
Recall: 0.9854066687960519
Precision: 0.9864527550614365
F1 score: 0.9858851085537097
```

```python
from sklearn.metrics import classification_report

class_names = ["Anthracnose", "Bacterial Canker", "Cutting Weevil", "Die Back",
               "Gall Midge", "Healthy", "Powdery Mildew", "Sooty Mould"]

print(classification_report(y_test_hot, y_pred, target_names=class_names))
```

```
                  precision    recall  f1-score   support

      Anthracnose       0.99      0.96      0.98       107
Bacterial Canker        1.00      0.99      0.99        90
  Cutting Weevil        1.00      1.00      1.00        97
       Die Back         1.00      1.00      1.00        52
     Gall Midge         0.96      0.98      0.97        99
        Healthy         0.99      0.99      0.99       101
  Powdery Mildew        0.98      0.99      0.98        88
     Sooty Mould        0.97      0.97      0.97       112

       micro avg        0.99      0.98      0.98       746
       macro avg        0.99      0.99      0.99       746
    weighted avg        0.99      0.98      0.98       746
     samples avg        0.98      0.98      0.98       746
```

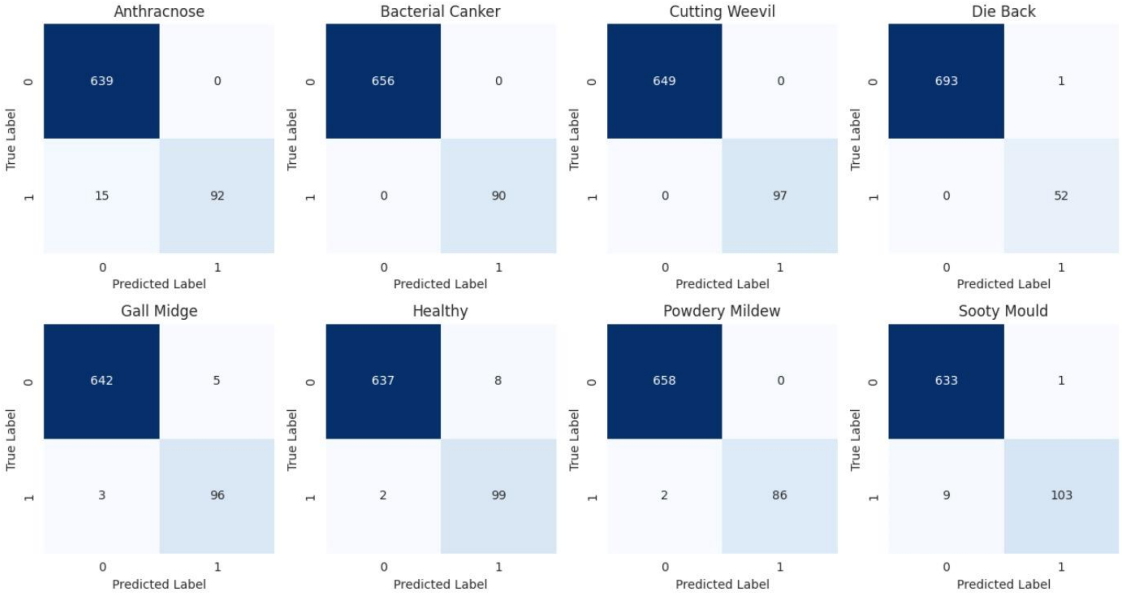
```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt
import numpy as np


neighbors_range = range(1, 10)
accuracies = []

for n_neighbors in neighbors_range:
    # fit
    knn_classifier = KNeighborsClassifier(n_neighbors=n_neighbors)
    knn_classifier.fit(x_train_flatten, y_train_hot)

    y_pred = knn_classifier.predict(x_test_flatten)

    # Evaluate
    accuracy = accuracy_score(y_test_hot, y_pred)
    accuracies.append(accuracy)
```

```
[ ] plt.figure(figsize=(7, 5))
    plt.plot(neighbors_range, accuracies, marker='o')
    plt.title('Accuracy vs. Number of Neighbors')
    plt.xlabel('Number of Neighbors (k)')
    plt.ylabel('Accuracy')
    plt.xticks(neighbors_range)
    plt.grid(True)
    plt.show()

    print("\n")
    print("Accuracies:", accuracies)
    print(f'Accuracy for K=10 : {accuracy}')
```

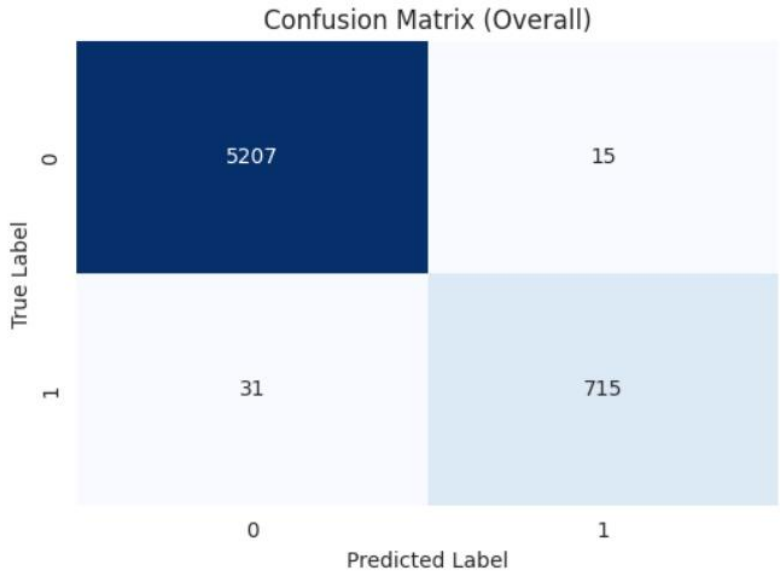



```
Accuracies: [0.9839142091152815, 0.9571045576407506, 0.9839142091152815, 0.9638069705093834,
Accuracy for K=10 : 0.9584450402144772
```

```
class_names = ["Anthracnose", "Bacterial Canker", "Cutting Weevil", "Die Back",
               "Gall Midge", "Healthy", "Powdery Mildew", "Sooty Mould"]

conf_matrices = multilabel_confusion_matrix(y_test_hot, y_pred)

fig, axes = plt.subplots(nrows=2, ncols=4, figsize=(13, 7))

for i, (conf_matrix, ax) in enumerate(zip(conf_matrices, axes.flatten())):
    sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', cbar=False,
                xticklabels=['0', '1'], yticklabels=['0', '1'], ax=ax)
    ax.set_title(f'{class_names[i]}', fontsize=12)
    ax.set_xlabel('Predicted Label', fontsize=10)
    ax.set_ylabel('True Label', fontsize=10)

plt.tight_layout()
plt.show()
```



```
conf_matrix = multilabel_confusion_matrix(y_test_hot, y_pred)

plt.figure(figsize=(6, 4))
sns.heatmap(conf_matrix.sum(axis=0), annot=True, fmt='d', cmap='Blues', cbar=False,
            xticklabels=['0', '1'], yticklabels=['0', '1'])
plt.title('Confusion Matrix (Overall)')
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.show()
```




```
from sklearn.metrics import recall_score, precision_score, f1_score

recall = recall_score(y_test_hot, y_pred, average='macro')

precision = precision_score(y_test_hot, y_pred, average='macro')

f1 = f1_score(y_test_hot, y_pred, average='macro')

print("Recall:", recall)
print("Precision:", precision)
print("F1 score:", f1)
```

```
Recall: 0.9633279572533355
Precision: 0.9809056731526347
F1 score: 0.9711539792259121
```

```
from sklearn.metrics import classification_report

class_names = ["Anthracnose", "Bacterial Canker", "Cutting Weevil", "Die Back",
               "Gall Midge", "Healthy", "Powdery Mildew", "Sooty Mould"]

print(classification_report(y_test_hot, y_pred, target_names=class_names))
```

```
                  precision    recall  f1-score   support

     Anthracnose       1.00      0.86      0.92       107
Bacterial Canker       1.00      1.00      1.00        90
  Cutting Weevil       1.00      1.00      1.00        97
        Die Back       0.98      1.00      0.99        52
     Gall Midge        0.95      0.97      0.96        99
         Healthy       0.93      0.98      0.95       101
  Powdery Mildew       1.00      0.98      0.99        88
     Sooty Mould       0.99      0.92      0.95       112

       micro avg       0.98      0.96      0.97       746
       macro avg       0.98      0.96      0.97       746
    weighted avg       0.98      0.96      0.97       746
     samples avg       0.96      0.96      0.96       746
```

```
from sklearn.ensemble import RandomForestClassifier

rf_model = RandomForestClassifier(n_estimators=100, random_state=42)
rf_model.fit(x_train_flatten, y_train_hot)

# Step 5: Make predictions on the test set and evaluate the performance
y_pred = rf_model.predict(x_test_flatten)
accuracy = accuracy_score(y_test_hot, y_pred)
print(f'Accuracy with selected features: {accuracy}')
```

```
Accuracy with selected features: 0.7962466487935657
```

```
# Extract features using the VGG16 model
new_image_features = vgg16_base.predict(new_image)

# Flatten the features into a 1D array
new_image_features_flat = new_image_features.flatten().reshape(1, -1)
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 751ms/step
```

```
import cv2
import numpy as np

# Load and preprocess the new image
new_image_path = '/content/sooth.jpg'
new_image = cv2.imread(new_image_path)

# Check if the image was loaded successfully
if new_image is None:
    print(f"Error: Could not load image from {new_image_path}. Please check the path and try again.")
else:
    # Preprocess the image (ensure it matches the preprocessing used during training)
    new_image = cv2.cvtColor(new_image, cv2.COLOR_BGR2RGB)
    new_image = cv2.resize(new_image, (224, 224))  # Assuming the model was trained on 224x224 images
    new_image = np.expand_dims(new_image, axis=0)  # Add batch dimension
```

```
    # Extract features using the pre-trained model
    new_image_features = model.predict(new_image)

    # Check the shape of the extracted features
    print(f"Extracted features shape: {new_image_features.shape}")

    # Ensure that the flattening matches the number of features used during training
    new_image_features_flatten = new_image_features.reshape(1, -1)
    print(f"Flattened features shape: {new_image_features_flatten.shape}")

    # Ensure the number of features matches what the KNN classifier expects
    if new_image_features_flatten.shape[1] != knn_classifier.n_features_in_:
        print(f"Error: Expected {knn_classifier.n_features_in_} features, but got {new_image_features_flatten.shape[1]} features.")
    else:
        # Predict the class using the KNN classifier
        prediction = knn_classifier.predict(new_image_features_flatten)
        predicted_class = label_encoder.inverse_transform([np.argmax(prediction)])
        print(f'Predicted class: {predicted_class[0]}')
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 925ms/step
Extracted features shape: (1, 256)
Flattened features shape: (1, 256)
```

```python
from PIL import Image
from transformers import pipeline
import matplotlib.pyplot as plt

# Initialize the image classification pipeline
pipe = pipeline("image-classification", model="DunnBC22/vit-base-patch16-224-in21k-Mango_leaf_Disease")

# Load the image
image_path = '/content/healthy.jpg'
image = Image.open(image_path)

# Use the pipeline to classify the image
results = pipe(image)

# Print only the class name with the highest accuracy
if results:
    # Find the result with the highest score
    top_result = max(results, key=lambda x: x['score'])
    predicted_class = top_result['label']
    score = top_result['score']

    # Display the image
    plt.imshow(image)
    plt.axis('off')  # Hide axes
    plt.title(f"Predicted class: {predicted_class} with score: {score:.4f}")
    plt.show()
else:
    print("No results returned.")
```

# 8. TEST CASES

## Test Cases:

| Test Case ID | Test Case Description | Test Steps | Preconditions | Test Data | Post Conditions |
|---|---|---|---|---|---|
| **1** | Predicting Anthrocnose | Pre-process the image and predict the Output. | Test data should meet model requirements | Images of leaf. | **Anthrocnose: No** |
| **2** | Predicting Healthy | Pre-process the image and predict the Output. | Test data should meet model requirements | Images of leaf. | **Healthy: No** |
| **3** | Predicting Powdery Mildew | Pre-process the image and predict the Output. | Test data should meet model requirements | Images of leaf. | **Powdery Mildew: Yes** |
| **4** | Predicting Die Back | Pre-process the image and predict the Output. | Test data should meet model requirements | Images of leaf. | **Die Back : No** |

**TEST RESULTS**

The culmination of our Deep Learning model development lies in the rigorous testing and evaluation phase. This pivotal step aims to ascertain the predictive capabilities and performance of our models, further contributing to the refinement of disease prediction accuracy. Our evaluation metrics encompass a comprehensive suite of measurements, including:

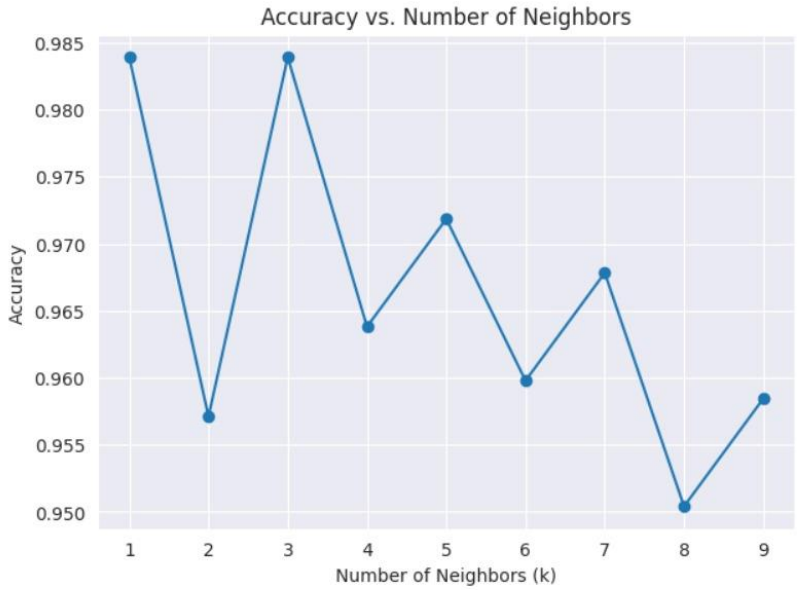
**Accuracy**

Accuracy showcases the proportion of correctly predicted outcomes. A high accuracy rate signifies the models precision in differentiating disease classes and underscores their potential as diagnostic tools.

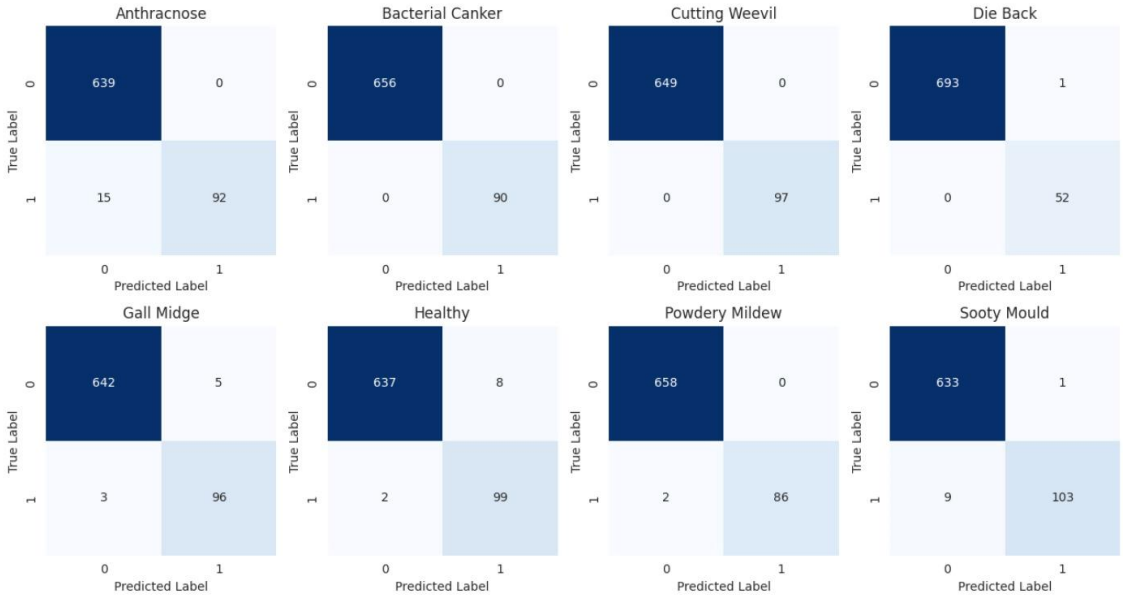
Formula: (TP + TN) / (TP + TN + FP + FN)

Accuracy of models for leaf Diseases:

| Model | Accuracy |
|---|---|
| VGG-16 | 81.95 |
| Resnet50 | 86.04 |
| Ensemble Learning (VGG16,Resnet50) | 98.39 |




```
Accuracies: [0.9839142091152815, 0.9571045576407506, 0.9839142091152815, 0.9638069705093834,
Accuracy for K=10 : 0.9584450402144772
```



# 9. CONCLUSION

The development of a plant disease detection system using machine learning and deep learning, specifically focused on mango leaves, represents a significant advancement in the field of agricultural technology. This system leverages modern image processing techniques, feature extraction methods, and sophisticated classification algorithms to accurately identify and classify diseases in mango leaves. By automating the disease detection process, the system offers farmers and agricultural professionals a powerful tool for early disease diagnosis, potentially leading to increased crop yields and reduced losses.

The use of deep learning models, particularly Convolutional Neural Networks (CNNs), allows for high accuracy in detecting subtle patterns and variations in leaf images that are indicative of specific diseases. The integration of a user-friendly interface ensures that the system is accessible to non-experts, enabling widespread adoption and practical application in real-world scenarios.

In conclusion, this project not only demonstrates the practical application of machine learning and deep learning in agriculture but also highlights the potential for these technologies to revolutionize plant disease management. The continued refinement and expansion of such systems could lead to more sustainable farming practices and contribute to global food security by protecting crops from disease outbreaks.

# 10 FUTURE SCOPE

The plant disease detection system using machine learning and deep learning for mango leaves holds significant potential for further development and expansion. Here are some key areas for future scope:

**Expansion to Other Crops**: The current system can be extended to detect diseases in a variety of other crops beyond mangoes. By training models on datasets from different plants, the system could become a versatile tool applicable across diverse agricultural sectors.

**Integration with IoT Devices**: Integrating the system with Internet of Things (IoT) devices, such as drones or field cameras, could enable real-time monitoring of large agricultural areas. This would allow for continuous surveillance and instant disease detection, leading to faster response times.

**Real-Time Disease Monitoring**: Developing the system into a real-time application that continuously monitors crops and automatically detects diseases can further enhance its utility. This could involve deploying the system on mobile platforms or edge devices like Raspberry Pi for on-site disease detection.

**Improved Model Accuracy and Generalization**: Ongoing research into more advanced deep learning architectures and techniques, such as transfer learning or ensemble methods, could improve the model's accuracy and its ability to generalize across different environmental conditions and disease variations.

**Automated Treatment Recommendations**: Future versions of the system could be enhanced to not only detect diseases but also to provide automated, context-specific treatment recommendations, integrating the system with agronomic databases and expert systems.

**Large-Scale Deployment and Cloud Integration** :Cloud-based platforms can be used to store vast amounts of image data and to deploy models at scale, making the system accessible to farmers globally. This could also facilitate collaborative learning where data from various regions contribute to improving the model.

**Multi model Data Integration**: Incorporating additional data types such as environmental data (e.g., temperature, humidity), soil data, or even genomic data could improve the model's predictive power and provide more comprehensive disease management strategies.

**Customization for Local Conditions**: The system can be customized to account for local agricultural practices, disease prevalence, and environmental conditions, making it more relevant and effective for specific regions or farming communities.

# 12 BIBLIOGRAPHY


- · **Chouhan, S. S., Kaul, A., Singh, U. P., & Jain, S. (2020).** "A data repository of leaf images: Practice towards plant conservation with plant pathology." *Data in Brief*, 25, 104414.
- **Sladojevic, S., Arsenovic, M., Anderla, A., Culibrk, D., & Stefanovic, D. (2016).** "Deep neural networks based recognition of plant diseases by leaf image classification."
- *Computational Intelligence and Neuroscience*, 2016, 1-11.
  · **Ferentinos, K. P. (2018).** "Deep learning models for plant disease detection and diagnosis."
- *Computers and Electronics in Agriculture*, 145, 311-318. **Dhaka, V. S., Meena, S. V., Rani, G., Sinwar, D., Dangayach, G. S., & Tanwar, S. (2021).** "A survey of deep learning techniques for analyzing plant diseases." *Computers and Electronics in Agriculture*, 173, 105383.
- **Mohanty, S. P., Hughes, D. P., & Salathé, M. (2016).** "Using deep learning for image-based plant disease detection." *Frontiers in Plant Science*, 7, 1419.

- **Brahimi, M., Boukhalfa, K., & Moussaoui, A. (2017).** "Deep learning for tomato diseases: Classification and symptoms visualization." *Applied Artificial Intelligence*, 31(4), 299-315.
- **Kamilaris, A., & Prenafeta-Boldú, F. X. (2018).** "Deep learning in agriculture: A survey." *Computers and Electronics in Agriculture*, 147, 70-90.
- **Ghosal, S., Blystone, D., Singh, A. K., Sarkar, S., Ganapathysubramanian, B., Singh, A., & Singh, S. (2018).** "An explainable deep machine vision framework for plant stress phenotyping." *Proceedings of the National Academy of Sciences*, 115(18), 4613-4618.
- **Kumar, S. V., & Mehta, P. (2021).** "Plant disease detection using machine learning: A review." *Materials Today: Proceedings*, 46, 5277-5281.
- **Picon, A., Alvarez-Gila, A., Seitz, M., Ortiz-Barredo, A., Echazarra, J., & Johannes, A. (2019).** "Deep convolutional neural networks for mobile capture device-based crop disease classification in the wild." *Computers and Electronics in Agriculture*, 161, 280-290.