**PLANT DISEASE DETECTION USING CNN**

# PROJECT REPORT

***Submitted by***

**HEMASHREE D (311521104017)**

***in fulfillment for the subject***

**NM1009 – GENERATIVE AI FOR ENGINEERING**

**BACHELOR OF ENGINEERING**

***IN***

# COMPUTER SCIENCE AND ENGINEERING MEENAKSHI SUNDARARAJAN ENGINEERING COLLEGE,

**KODAMBAKKAM, CHENNAI-24**

**ANNA UNIVERSITY: CHENNAI 600 025**

# MAY 2024

ANNA UNIVERSITY: CHENNAI 600 025

**BONAFIDE CERTIFICATE**

Certified that this project report “**PLANT DISEASE DETECTION USING CNN**” is the bonafide work of “**HEMASHREE D (311521104017)** ” Naan Mudhalvan ID **“au311521104017”** who carried out the project work under my supervision.

**SIGNATURE SIGNATURE**

Dr.S.Aarthi,M.E.,Ph.D Dr.S.Aarthi,M.E.,Ph.D

**HEAD OF THE DEPARTMENT HEAD OF THE DEPARTMENT**

Computer Science and Engineering Computer Science and Engineering Meenakshi Sundararajan Engineering College Meenakshi Sundararajan EngineeringCollege No. 363, ArcotRoad, Kodambakkam, No. 363, Arcot Road, Kodambakkam, 

Chennai -600024 Chennai 600024

Submitted for the project viva voce of Bachelor of Engineering in Computer Science and Engineering held on \_\_.

# INTERNALEXAMINER EXTERNALEXAMINER

**ACKNOWLEDGEMENT**

First and foremost, we express our sincere gratitude to our Respected Correspondent **Dr. K. S. Lakshmi**, our beloved Secretary **Mr. N. Sreekanth**, Principal **Dr. S. V. Saravanan** for their constant encouragement, which has been our motivation to strive towards excellence.

Our primary and sincere thanks goes to **Dr. S. Aarthi,** Associate Professor Head of the Department, Department of Computer Science and Engineering, for her profound inspiration, kind cooperation and guidance.

We’re grateful to **Dr. S. Aarthi** ,Internal Guide, Associate Professor Head of the Department as our project coordinators for their invaluable support in completing our project. We are extremely thankful and indebted for sharing expertise, and sincere and valuable guidance and encouragement extended to us.

Above all, we extend our thanks to God Almighty without whose grace and

Blessings it wouldn’t have been possible.

# ABSTRACT

Plant disease detection is a critical aspect of modern agriculture, as early identification and intervention can prevent crop loss and promote sustainable farming practices. This project explores the application of convolutional neural networks (CNNs) for the detection of plant diseases from leaf images. The approach leverages both a custom CNN architecture and a model based on the pretrained VGG16 network to achieve high performance disease classification.

The dataset comprises labeled images of plant leaves representing multiple species and various stages of disease progression. Data preprocessing involves image resizing, normalization, and augmentation to enhance the models' ability to generalize. The models are trained using a range of optimization techniques, including learning rate reduction, early stopping, and model checkpoints, to refine the training process and improve performance.

Model performance is evaluated based on metrics such as accuracy, precision, recall, and F1score using separate training, validation, and testing datasets. This comprehensive assessment allows us to identify the strengths and limitations of each model in disease detection and classification. Additionally, we explore potential deployment options for the models to enable practical, realtime plant health monitoring.

Results indicate that both models exhibit strong performance in classifying plant diseases, with the VGG-16 based model showing particular promise due to its transfer learning capabilities and robust feature extraction. These findings highlight the potential of deep learning for efficient and effective plant disease detection, offering significant benefits for farmers, agricultural professionals, and the broader field of smart agriculture. This project demonstrates the feasibility of using CNNs for plant disease detection and sets the stage for further advancements in automated crop monitoring and disease management.

|  | **TABLE OF CONTENTS** |  |
| --- | --- | --- |
| **CHAPTER NO.** | **TITE** | **PAGE NO.** |
|  | **ABSTRACT** | iv |
|  | **LIST OF TABLES** | viii |
|  | **LIST OF FIGURES** | ix |
|  | **LIST OF SYMBOLS, ABBREVIATIONS AND EXPANSIONS** | x |
| **1** | **INTRODUCTION** | **1** |
|  | 1.1 ABOUT THE PROJECT | 1 |
|  | 1.2 PROJECT OVERVIEW | 1 |
|  | 1.3 PURPOSE | 2 |
|  | 1.4 EXISTING SYSTEM | 4 |
|  | 1.5 PROBLEM STATEMENT | 5 |
| **2.** | **LITERATURE SURVEY** | 6 |
| **3.** | **SYSTEM ARCHITECTURE** | 7 |
|  | 3.1 SYSTEM ARCHITECTURE | 7 |
|  | 3.2 HARDWARE REQUIREMENTS | 8 |
|  | 3.3 SOFTWARE REQUIREMENTS | 8 |
|  | 3.3.1 PYTHON | 9 |
|  |  |  |
|  | 3.3.2 JUPYTER NOTEBOOK | 9 |
| **4.** | **IDEATION** | 9 |
|  | 4.1 IDEATION & BRAINSTORMING | 9 |
| **5.** | **REQUIREMENT ANALYSIS** | 12 |
|  | 5.1 FUNCTIONAL REQUIREMENTS | 12 |
|  | 5.2 NON-FUNCTIONAL REQUIREMENTS | 13 |
| **6.** | **SYSTEM MODELLING** | 15 |
|  | 6.1 UNIFIED MODELLING LANGUAGE | 15 |
|  | 6.2 USE CASE DIAGRAM | 15 |
|  | 6.3 CLASS DIAGRAM | 17 |
|  | 6.4 SEQUENCE DIAGRAM | 18 |
|  | 6.5 ACTIVITY DIAGRAM | 20 |
|  | 6.6 STATE CHART DIAGRAM | 21 |
| **7.** | **SYSTEM IMPLEMENTATION** | 23 |
|  | 7.1 PROPOSED SYSTEM | 23 |
|  | 7.2 SOURCE CODE | 24 |
| **8.** | **PROJECT DESIGN** | 34 |
|  | 8.1 DATA FLOW DIAGRAM | 34 |
|  |  |  |
|  |  |  |
|  | 8.2 USER STORIES | 37 |
| **9.** | **ADVANTAGES AND DISADVANTAGES** | 36 |
| **10.** | **CONCLUSION AND FUTURE ENHANCEMENT** | 39 |
|  | 10.1 CONCLUSION | 39 |
|  | 10.2 FUTURE ENHANCEMENT | 39 |
| **11.** | **APPENDIX SCREENSHOT** | 41 |
|  | **REFERNCES** | 42 |
|  |  |  |

| **TABLE NO.** | **LIST OF TABLES**  **NAME OF THE TABLE** | **PAGE NO.** |
| --- | --- | --- |
| 3.3 | HARDWARE REQUIREMENTS | 8 |
| 3.4 | SOFTWARE REQUIREMENTS | 8 |

**LIST OF FIGURES**

# FIGURE NO. NAME OF THE FIGURE PAGE NO.

3.2 SYSTEM ARCHITECTURE 6

* 1. USE CASE DIAGRAM 15
  2. CLASS DIAGRAM 17
  3. SEQUENCE DIAGRAM 18
  4. ACTIVITY DIAGRAM 20
  5. STATE CHART DIAGRAM 21

**LIST OF SYMBOLS, ABBREVIATIONS AND EXPANSION**

**ABBREVIATION EXPANSION**

CNN Convolutional Neural Network

UML Unified Modeling Language

RAM Random Access Memory

GPU Graphics Processing Unit

VGG Visual Geometry Group

# CHAPTER 1 INTRODUCTION

* 1. **ABOUT THE PROJECT**

The project focuses on the development of a plant disease detection system using convolutional neural networks (CNNs) with a particular emphasis on leveraging the VGG16 architecture. The aim is to create a robust model capable of accurately identifying diseases in plant leaves based on input images. The dataset consists of images of various plant species affected by different diseases, collected from diverse sources. To achieve this, the project utilizes deep learning techniques, specifically CNNs, which are well-suited for image classification tasks. The CNN model is trained on a large dataset of labeled images, allowing it to learn and extract meaningful features from the input images. Additionally, the project explores transfer learning by leveraging the pre-trained VGG16 model as a base and adding custom layers to adapt it to the specific task of plant disease detection. Through rigorous experimentation and validation, the project aims to develop a reliable and accurate plant disease detection system that can assist farmers and agricultural experts in timely disease diagnosis and management, ultimately contributing to improved crop yield and agricultural sustainability.

# PROJECT OVERVIEW

**Project Overview:** Plant Disease Detection using CNN

A Convolutional Neural Network (CNN) is a type of Deep Learning neural network architecture commonly used in Computer Vision. Computer vision is a field of Artificial Intelligence that enables a computer to understand and interpret the image or visual data. VGG16 is a convolutional neural network model that's used for image recognition. It's unique in that it has only 16 layers that have weights, as opposed to relying on a large number of hyper-parameters. It's considered one of the best vision model architectures.

This project focuses on developing and evaluating deep learning models to detect and classify plant diseases using convolutional neural networks (CNNs). Leveraging a custom CNN architecture and a model based on the pretrained VGG16 network, the project aims to improve the accuracy and efficiency of identifying diseases across various plant species from leaf images. The dataset comprises labeled images of healthy and diseased plant leaves across different species, with images preprocessed for consistency and model performance. The project involves rigorous training and evaluation of the models using optimization techniques such as learning rate reduction, early stopping, and model checkpoints. Performance is measured using metrics like accuracy, precision, recall, across separate training, validation, and testing datasets. This work explores potential applications for practical, realtime plant health monitoring and disease management, highlighting the potential for deep learning to revolutionize agricultural productivity and sustainability.

# PURPOSE

The purpose of using convolutional neural networks (CNNs) and VGG16 for plant disease detection is to develop efficient, accurate, and automated methods for identifying and classifying plant diseases from visual data such as images of plant leaves. This approach offers several advantages and benefits for agricultural practices:

- Accuracy and Efficiency: CNNs, including VGG16, are designed for image recognition tasks and can process visual data with high accuracy. This allows for rapid and precise identification of plant diseases, which is crucial for timely intervention.

- Early Detection and Prevention: Early detection of plant diseases is key to preventing the spread of pathogens and minimizing crop loss. Automated detection systems can help farmers take action quickly to protect crops.

- Large-scale Monitoring: CNN-based models can process large volumes of images, making them suitable for large-scale monitoring of plant health across extensive farmland.

- Consistency and Reliability: Automated models provide consistent and objective assessments, reducing the variability that can occur with human inspection.

- Transfer Learning with VGG16: By using a pre-trained network like VGG16, which has already been trained on a large dataset , the model can benefit from well-learned features and patterns, making it more robust and efficient in recognizing plant diseases.

- Reduced Labor and Cost: Automation reduces the need for manual labor in disease detection, saving time and resources for farmers and agricultural professionals.

- Data-Driven Decision Making: The data and predictions from these models can help guide decision-making in crop management and disease control strategies.

- Research and Development: Leveraging advanced deep learning methods can spur further research and development in the field of agricultural technology, paving the way for innovative solutions.

# EXISTING SYSTEM

In existing systems for plant disease detection, traditional methods often rely on manual observation by agricultural experts or laboratory analysis, which can be time-consuming and costly. Some automated systems have been developed using machine learning techniques, but they may lack the accuracy and robustness required for real-world applications.

The proposed code builds upon these existing systems by leveraging state-of-the-art deep learning techniques, specifically CNNs, to enhance the accuracy and efficiency of plant disease detection. Additionally, by incorporating the VGG16 architecture and exploring transfer learning, your code takes advantage of pre-trained models and adapts them to the specific task of plant disease classification. This approach allows for faster convergence during training and improved performance on limited data.

Furthermore, the proposed program implements data augmentation techniques such as rescaling, shearing, zooming, and horizontal flipping to increase the diversity of the training dataset and improve the model's generalization ability. Additionally, the use of callbacks such as ReduceLROnPlateau and EarlyStopping helps in optimizing the training process and preventing overfitting.

Overall, the proposed code represents a significant advancement in the field of plant disease detection by integrating cutting-edge deep learning techniques with efficient data preprocessing and model optimization strategies, leading to more accurate and scalable solutions for agricultural applications.

# PROBLEM STATEMENT

The problem addressed in this project is the timely and accurate detection of diseases in plant leaves, which is crucial for maintaining crop health and ensuring optimal agricultural yield. Traditional methods of disease identification often rely on manual inspection by agricultural experts, which can be labor-intensive, time-consuming, and prone to human error. Moreover, in regions with large agricultural areas, such manual inspection becomes impractical and ineffective.Some of the Challenges involved are :

1. Variability in Disease Symptoms: Plant diseases can manifest in various ways, including discoloration, wilting, lesions, and abnormal growth patterns. Identifying these symptoms accurately across different plant species and disease types poses a significant challenge due to the variability in appearance and progression of diseases.
2. Large-Scale Deployment: Agricultural fields often cover extensive areas, making it challenging to manually inspect every plant for signs of disease.
3. Data Variability and Quality: Obtaining high-quality and diverse datasets for training machine learning models can be difficult in agricultural settings. Factors such as lighting conditions, camera angles, and environmental variations can introduce noise and inconsistencies in the data, affecting the performance of disease detection algorithms.
4. Real-Time Detection: In agricultural settings, timely detection of diseases is critical to prevent the spread of infections and minimize crop damage. Developing algorithms capable of real-time or near-real-time detection while maintaining high accuracy is a key challenge.
5. Resource Constraints: Agricultural environments often have limited access to computational resources, such as processing power and memory. Designing efficient algorithms that can run on resource-constrained devices or platforms is essential for practical deployment in the field.

# CHAPTER 2

# LITERATURE SURVEY

1. Deep Learning Applications for Plant Disease Detection and Diagnosis" by Mohanty et al. (2016):This paper provides an overview of various deep learning techniques, including convolutional neural networks (CNNs), applied to plant disease detection and diagnosis. It discusses the challenges in traditional methods and highlights the effectiveness of CNN-based approaches in automating disease identification.

2. "Deep Learning for Plant Disease Detection: A Review" by Singh et al. (2020):The review paper presents a comprehensive survey of deep learning models, with a focus on CNN architectures, utilized for plant disease detection. It discusses the advantages and limitations of different CNN architectures and provides insights into the data preprocessing techniques and transfer learning strategies employed in the field.

3. "A Review on Deep Learning Techniques for Plant Disease Detection and Recognition" by Ghosal et al. (2021):

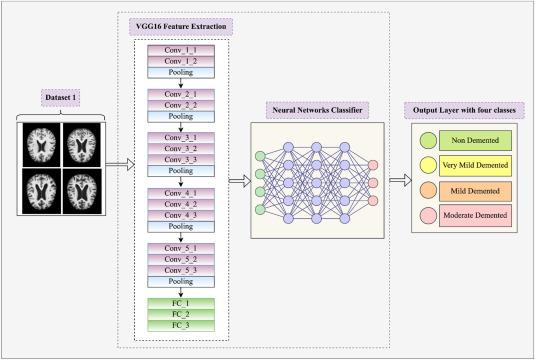
- This review article explores the recent advancements in deep learning techniques for plant disease detection and recognition. It covers CNN-based models, including VGG16, and discusses their performance in comparison to traditional machine learning approaches. The paper also discusses challenges such as dataset scarcity and model interpretability.

# CHAPTER 3

# SYSTEM ARCHITECTURE

# SYSTEM ARCHITECTURE:

# 1-s2.0-S2665917422001404-gr1.jpg

****

# 1-s2.0-S2665917422001404-gr1.jpg3-Figure3-1.png

# HARDWARE REQUIREMENTS:

| **SYSTEM** | INTEL i3 Processor |
| --- | --- |
| **HARD DISK** | 256 GB |
| **MONITOR** |  |
| **INPUT DEVICES** | Keyboard, Mouse |
| **RAM** | 8 GB |

# SOFTWARE REQUIREMENTS:

| **REQUIREMENTS** | **SPECIFICATIONS** |
| --- | --- |
| TOOL | JUPYTER NOTEBOOK/  GOOGLE COLAB |
| CODING LANGUAGE | PYTHON |
| OPERATING SYSTEM | WINDOWS 10 |

# 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.

# JUPYTER NOTEBOOK:

Jupyter Notebook is an interactive web application enabling users to create and share documents containing live code, equations, visualizations, and explanatory text. Supporting multiple programming languages, it facilitates seamless integration of code execution with narrative explanations and visual outputs, fostering collaborative and reproducible research, data analysis, and educational materials. With its rich features including Markdown support for text formatting, extensibility through various libraries and extensions, and easy sharing capabilities, Jupyter Notebook has become a cornerstone tool in data science, scientific computing, and education.

# CHAPTER 4

**IDEATION AND BRAINSTORMING**

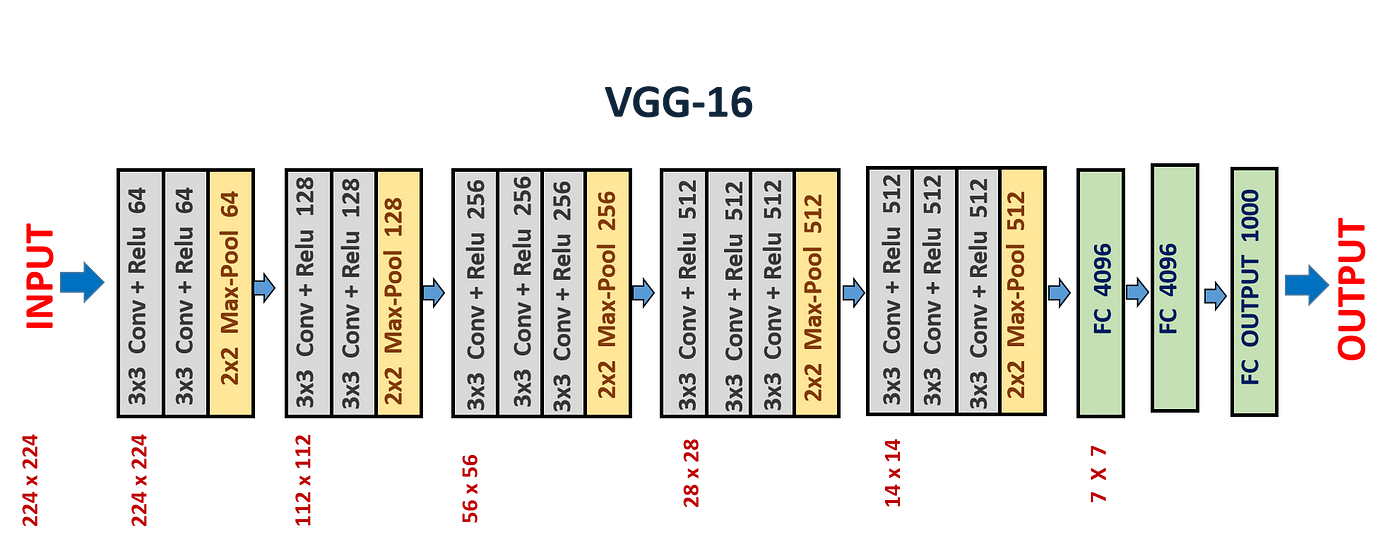
The goal of this project is to develop an automated, efficient, and accurate system for plant disease detection using images of plant leaves. The ideation and brainstorming phase involves considering different aspects of the project, including data handling, model architecture, and evaluation methods.

**1. Data Handling and Preprocessing:**

* Data Collection: Source a dataset containing labeled images of plant leaves with various diseases and healthy samples. The dataset should cover different plant species and a range of diseases to ensure comprehensive training and testing.
* Data Preprocessing: Develop methods for data preprocessing, such as:
  + Resizing: Resize images to a consistent size (e.g., 224x224 pixels) suitable for input to the CNN models.
  + Normalization: Normalize pixel values (e.g., scaling them to the range [0, 1]) to standardize input data and improve model training.
  + Data Augmentation: Apply transformations like rotation, zooming, shearing, and horizontal flipping to increase data diversity and reduce overfitting.

**2. Model Design and Architecture:**

* Custom CNN Architecture: Design a custom CNN model tailored specifically for plant disease detection. Consider:
  + The number of convolutional layers and their parameters (e.g., filter size, stride, padding) for effective feature extraction.
  + The use of pooling layers (e.g., max pooling) to reduce dimensionality and retain important features.
  + Dense (fully connected) layers for classification tasks.
  + Activation functions (e.g., ReLU for hidden layers and softmax for output layers).
  + Dropout layers for regularization and to reduce overfitting.
* VGG16-based Model: Leverage transfer learning using a pre-trained VGG16 model:
  + Use the pretrained VGG16 model's feature extraction layers and freeze them to retain learned features from ImageNet.
  + Add custom dense layers on top of the VGG16 base for plant disease classification.



**3. Model Training and Optimization:**

* Training Strategy: Determine the training strategy, including:
  + Batch Size: Choose an appropriate batch size for efficient training.
  + Learning Rate: Set an initial learning rate and use learning rate scheduling or reduction on plateau callbacks to adjust it as training progresses.
  + Epochs: Decide the number of training epochs for both models, balancing the need for training completion with the risk of overfitting.
* Callbacks: Utilize callbacks such as:
  + ReduceLROnPlateau: Automatically reduces the learning rate when validation loss plateaus, to help improve convergence.
  + Early Stopping: Halt training when validation loss stops improving, reducing the risk of overfitting.
  + Model Checkpoint: Save the best model based on validation loss for later use in inference.

**4. Performance Evaluation and Metrics:**

* Define appropriate metrics for evaluating model performance, such as:
  + Accuracy: The proportion of correct predictions.
  + Precision and Recall: Metrics assessing the model's ability to correctly identify positive instances (true positives).
* Assess model performance using a separate validation dataset and test dataset to ensure generalization.

**5. Potential Enhancements and Future Directions:**

* Consider exploring additional techniques such as ensemble methods, more sophisticated architectures , or other forms of data augmentation to further improve model performance.
* Look into the potential deployment of the models in real-world agricultural settings, such as integrating them with IoT sensors for real-time plant health monitoring.

# CHAPTER 5

**REQUIREMENT ANALYSIS**

The requirements analysis phase involves identifying and specifying the functional and non-functional requirements of the Plant disease detection using CNN. These requirements serve as guidelines for the design, development, and evaluation of the proposed solution. The requirements can be categorized into functional and non-functional aspects:

**5.1 FUNCTIONAL REQUIREMENTS**

1. **Custom CNN Model:**

* Architecture: Custom CNN architecture should include convolutional, pooling, and dense layers optimized for plant disease classification.
* Activation Functions: Use appropriate activation functions (e.g., ReLU, softmax) in the model layers.
* Output Layer: Provide an output layer with softmax activation for multi-class classification of plant diseases.

1. **VGG16-based Model:**

* Base Model: Use the pre-trained VGG16 network as the base model for transfer learning, leveraging learned features from ImageNet.
* Custom Layers: Add custom layers on top of the base model for plant disease classification.
* Fine-Tuning: Allow for fine-tuning of the base model's weights for improved performance on the specific task.

1. **Training and Optimization:**

* Training Capability: Support training using training and validation datasets with adjustable batch sizes and epochs.
* Optimization Techniques: Implement optimization methods such as learning rate reduction and early stopping to enhance training efficiency and prevent overfitting.

1. **Model Inference and Prediction:**

* Image Preparation: Provide functions to prepare images for prediction, including resizing, normalization, and expansion of dimensions.
* Prediction: Enable models to predict plant disease class from input images, returning the most likely disease classification.
* Visualization: Allow visualization of input images with corresponding predictions for easier interpretation.

1. **Model Saving and Loading:**

* Model Saving: Enable saving of trained models for later use in inference and deployment.
* Model Loading: Provide functionality to load saved models for inference and testing.

**5.2 NON-FUNCTIONAL REQUIREMENTS**

1. **Performance:**

* Response Time: The system should provide predictions in a reasonable time frame, enabling near real-time disease detection and intervention.
* Throughput: The system should handle a substantial volume of images efficiently, ensuring smooth operation even with high data loads.

1. **Scalability:**

* Data Handling: The system should be capable of scaling to handle larger datasets and more complex models as the project grows.
* Model Deployment: The system should support efficient deployment of the trained models in various environments (e.g., cloud, on-premise, edge devices).

1. **Reliability and Availability:**

* Uptime: Ensure high availability of the system, minimizing downtime and disruptions in service.
* Fault Tolerance: Implement strategies for fault tolerance and recovery to handle potential failures gracefully.

1. **Data Security:**

* Data Protection: Ensure data privacy and protection of sensitive information, including compliance with relevant regulations (e.g., GDPR).
* Secure Access: Provide secure access control mechanisms for model deployment and inference to prevent unauthorized use.

1. **Usability:**

* User Interface: If applicable, provide an intuitive user interface for farmers and agricultural professionals to easily access predictions and insights.
* Documentation: Offer comprehensive documentation for system usage, including model deployment and interpretation of results.

# CHAPTER 6

# SYSTEM MODELING

**6.1 UNIFIED MODELING LANGUAGE(UML):**

The Unified Modeling Language (UML) is a standardized modeling language that encompasses a comprehensive set of diagrams. It was developed to assist system and software developers in specifying, visualizing, constructing, and documenting software system artifacts, as well as for business modeling and other non-software systems. UML embodies a collection of proven engineering practices that are particularly effective for modeling large and intricate systems. It plays a vital role in the development of object-oriented software and the software development process.

By brainstorming and integrating these ideas, the proposed solution aims to develop a robust and effective plant disease detection system capable of generating accurate and contextually relevant descriptions for diverse visual content.

* + 1. Provide users with a ready-to-use, expressive visual modeling language so they can develop and exchange meaningful models.
    2. Provide extensibility and specialization mechanisms to extend the core concepts.
    3. Be independent of particular programming languages and development processes.
    4. Provide a formal basis for understanding the modeling language
    5. Encourage the growth of the OO tools market
    6. Support higher-level development concepts such as collaborations, frameworks, patterns and components.

# 

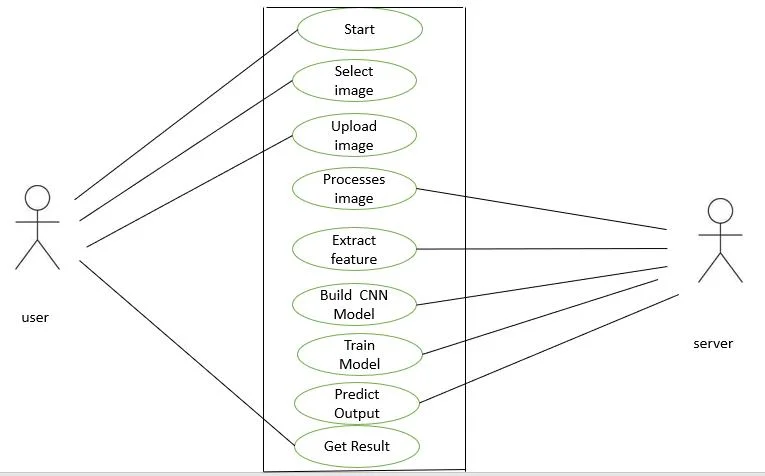
# 6.2 USE CASE DIAGRAM:

The use case diagram is used to define the core elements and processes that make up a system. The key elements are termed as "actors" and the processes are called "use cases". The use case diagram shows which actors interact with each use case. This definition defines what a use case diagram is primarily made up of - actors and usecases. In software and system engineering, a use case is a list of steps, typically defining interactions between a role and a system, to achieve a goal. The actor can be a human or an external system. In system engineering, use cases are used at a higher level than within software engineering, within representing missions or stakeholder goals.

The purposes of use case diagrams can be as follows

1. Used to gather requirements of a system.
2. Used to get an outside view of a system
3. Identify external and internal factors influencing the die system.
4. Showing the interacting among the requirements are actors.

Use cases help in identifying the operations that can be performed by an actor. It gives a list of the various applications that can be utilized by the system. The actor can be a real time human or a system. It helps in identifying the various modules present in the system. A single use case diagram captures a particular functionality of a system. Hence to model the entire system, a number of use case diagrams are used.



# Figure 4.2: Use case diagram

# 

# 

# 6.3 CLASS DIAGRAM:

Class diagram is a static diagram. It is the building block of every object-oriented system and helps in visualizing and describing the system. A class diagram depicts the structure of the system through its classes, their attributes, operations and relationships among the objects. A class is a blueprint that defines the variables and methods common to all objects of a certain kind. Class diagram shows a collection of classes, interfaces, associations, collaborations, and constraints. The characteristics of Class Diagram are:

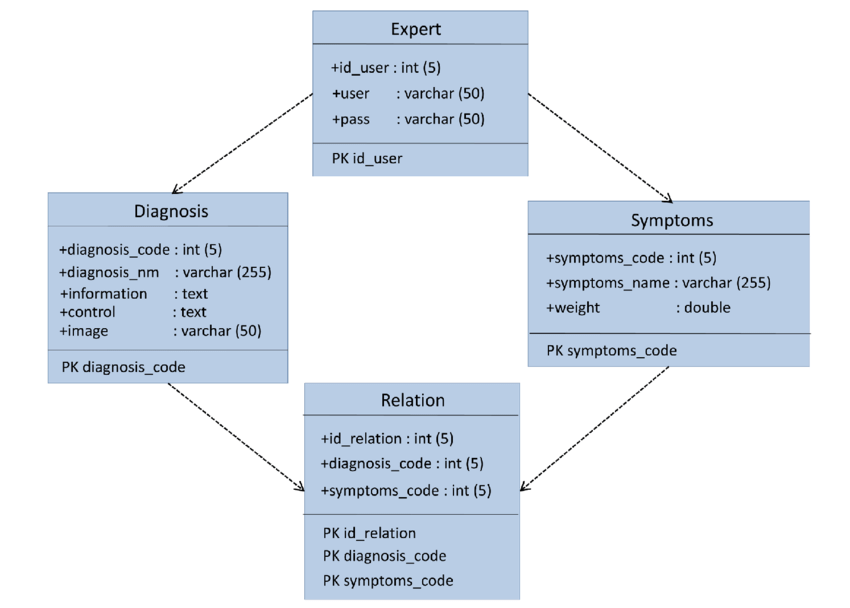
1. Each class is represented by a rectangle having a subdivision of three compartments

- name, attributes and operations

1. There are three types of modifiers which are used to decide the visibility of attributes

and operations: + is used for public visibility, a is used for protected visibility, - is used for private visibility

In the diagram, classes are represented with boxes that contain three compartments. The top compartment contains the name of the class. It is printed in bold and centered, and the first letter is capitalized. The middle compartment contains the attributes of the class. They are left-aligned and the first letter is lowercase. The bottom compartment contains the operations the class can execute. They are also left-aligned and the first letter is lowercase.



# Figure 4.3 : Class Diagram

# 6.4 SEQUENCE DIAGRAM

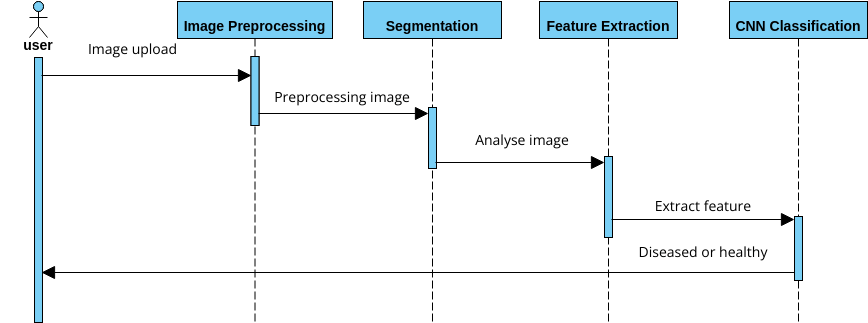
A sequence diagram is a kind of interaction diagram that shows how processes operate with one another and in which order. It is a construct of a Message Sequence Chart. A sequence diagram shows object interactions arranged in time sequence. Sequence diagrams are a popular dynamic modeling solution in UMI because they specifically focus on lifelines, or the processes and objects that live simultaneously, and the messages exchanged between them to perform a function before the lifeline ends. It depicts the objects and classes involved in the scenario and the sequence of messages exchanged between the objects needed to carry out the functionality of the scenario.A sequence diagram shows different processes or objects that live simultaneously as parallel vertical lines (lifelines) and the messages exchanged between them and the order in which they occur as horizontal arrows.

The main purpose of the Sequence diagram is

1. To capture the dynamic behavior of a system
2. To describe the message flow in the system.
3. To describe the interaction among objects.

Sequence diagrams can be used

1. To model the flow al control by time sequence
2. To model the Row of control by structural organizations.
3. For reverse engineering.



# Figure 4.4: Sequence Diagram

# 6.5 ACTIVITY DIAGRAM

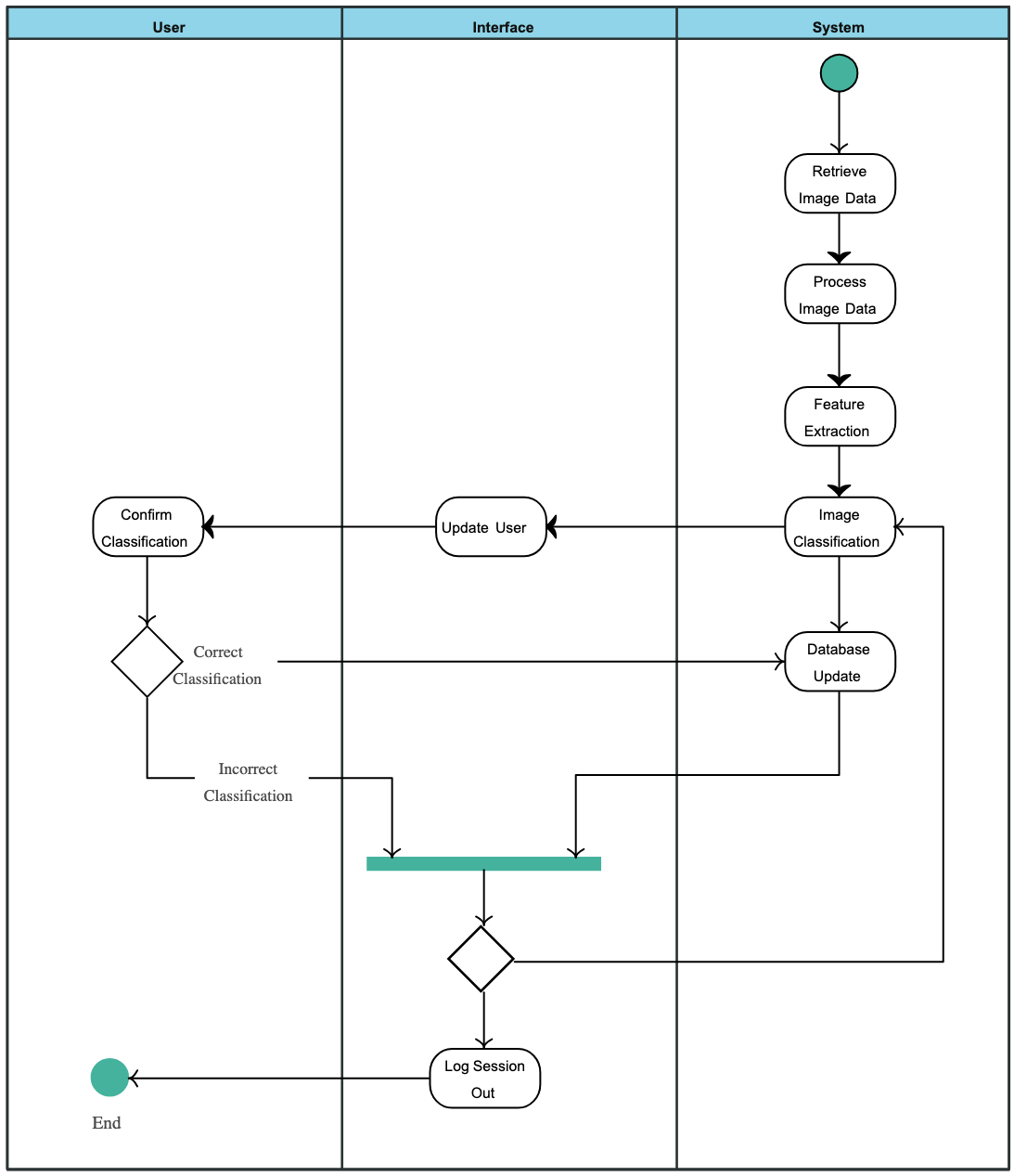
Activity diagrams are graphical representations of workflows of stepwise activities and actions with support for choice, iteration and concurrency. In the Unified Modeling Language, activity diagrams are intended to model both computational and organizational processes (1.0., work flows), as well as 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.

Activity diagram is basically a flowchart to represent the flow from one activity to another activity. The activity can be described as an operation of the system. The control flow is drawn from one operation to another. Thus flow can be sequential, branched, or concurrent.

Activity diagrams deal with all types of flow control by using different elements such as fork, join, etc. Activity diagrams are constructed from a limited number of shapes, connected with arrows.

The most important shape types:

* + 1. rounded rectangles representations
    2. diamonds represent decisions"
    3. bars represent the start (split) or end (join) of concurrent activities
    4. a black circle represents the start (initial node) of the workflow
    5. an encircled black circle represents the end (final node)



# Figure 4.5: Activity diagram

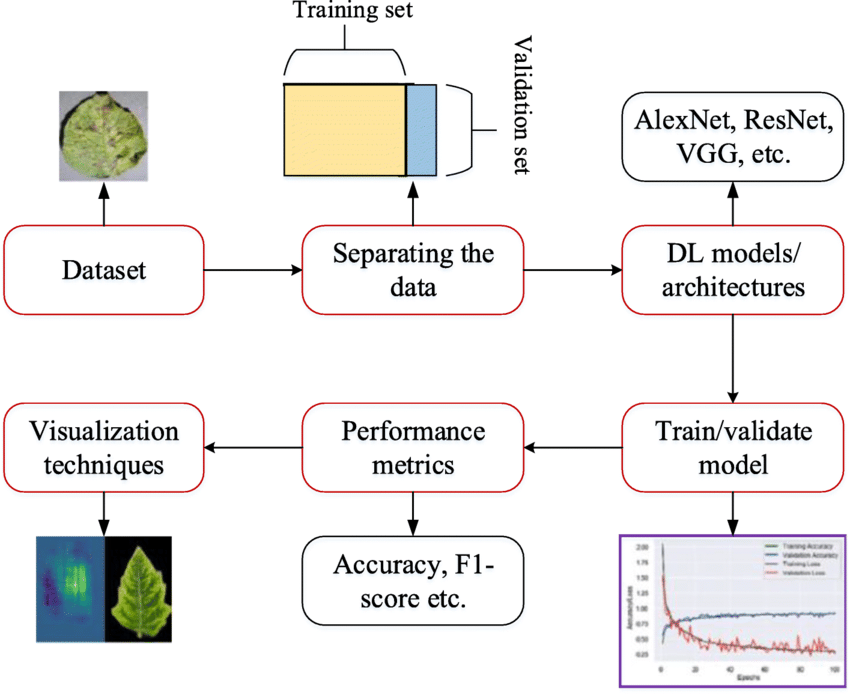
# 

# 6.6 STATE CHART DIAGRAM

Statechart diagram is one of the five UML diagrams used to model the dynamic nature of a system. They define different states of an object during its lifetime and these states are changed by events. Statechart diagrams are useful to model the reactive systems. Reactive systems can be defined as a system that responds to external or internal events. Statechart diagram describes the flow of control from one state to another state. States are defined as a condition in which an object exists and it changes when some event is triggered. The most important purpose of a Statechart diagram is to model the lifetime of an object from creation to termination. Statechart diagrams are also used for forward and reverse engineering of a system. However, the main purpose is to model the reactive system.

Following are the main purposes of using Statechart diagrams :

1. To model the dynamic aspect of a system.
2. To model the lifetime of a reactive system.
3. To describe different states of an object during its lifetime.
4. Define a state machine to model the states of an object.



[850 × 694](https://www.researchgate.net/figure/Flow-diagram-of-DL-implementation-for-plant-disease-detection-classification_fig3_360078192)

# Figure 4.8: Statechart Diagram

**CHAPTER 7**

# SYSTEM IMPLEMENTATION

# 7.1 PROPOSED SYSTEM

The proposed solution for the plant disease detection project involves the development of an artificial intelligence capable of accurately providing answers for the prompted questions from text data.

To tackle the challenge of plant disease detection efficiently and accurately, this project utilizes advanced deep learning models, including a custom CNN architecture and a model based on the pre-trained VGG16 network. The solution aims to automate the detection and classification of plant diseases from leaf images, providing significant benefits to agricultural productivity and sustainability.

**1. Dataset Preparation and Augmentation:**

* Begin by collecting a diverse dataset of labeled plant leaf images representing various plant species and diseases. This dataset should include healthy and diseased samples across multiple classes.
* Preprocess the images by resizing them to a uniform dimension (e.g., 224x224 pixels) for compatibility with the CNN models. Normalize the pixel values to standardize input data.
* Apply data augmentation techniques, such as random rotations, zooms, shear transformations, and horizontal flips, to increase the variability of the training data and improve model generalization.

**2. Model Development:**

* Custom CNN Architecture: Develop a custom CNN model optimized for plant disease detection. This model includes:
  + A series of convolutional layers with varying filter sizes to extract hierarchical features from the input images.
  + Max pooling layers to reduce dimensionality while retaining important features.
  + Dense layers to handle classification tasks, with ReLU activation functions for non-linearity and softmax for multi-class output.
  + Dropout layers to reduce overfitting and enhance model robustness.
* VGG16-based Model: Utilize transfer learning by incorporating a pre-trained VGG16 network as the base model:
  + The pre-trained VGG16 network serves as a feature extractor, leveraging learned patterns from ImageNet data.
  + Add custom fully connected layers on top of the base model to classify plant diseases, fine-tuning the network for the specific task.

**3. Training and Optimization Strategy:**

* Training Process: Train both models using the prepared data generators, including training and validation sets. Fine-tune the hyperparameters such as batch size and epochs for optimal results.
* Optimization Techniques: Implement ReduceLROnPlateau to automatically adjust the learning rate when validation loss stagnates. Employ Early Stopping to halt training when improvements plateau and prevent overfitting.

**4. Performance Evaluation and Metrics:**

* Evaluate model performance using metrics such as accuracy, precision, recall, and F1-score, which provide a comprehensive assessment of the models' classification capabilities.
* Test the models on a separate dataset to validate their generalization ability and ensure their effectiveness in real-world applications.

**7.2 SOURCE CODE :**

### **1. Plant Disease Detection**

from google.colab import drive

drive.mount('/content/drive')

#### 1.1. Import Required Libraries

import os

import cv2

import glob

import pandas as pd

import numpy as np

import tensorflow as tf

import matplotlib.pyplot as plt

from keras.layers import Dense

from keras.models import Sequential

from keras.preprocessing import image

from tensorflow.keras.optimizers import Adam

from tensorflow.keras.models import Model

from keras.callbacks import ReduceLROnPlateau

from tensorflow.keras.applications.vgg16 import VGG16

from keras.preprocessing.image import ImageDataGenerator

from keras.layers import Convolution2D,Dense,MaxPool2D,Activation,Dropout,Flatten

from keras.layers import Input, Add, Dense, Activation, ZeroPadding2D, BatchNormalization, Flatten, Conv2D, AveragePooling2D, MaxPooling2D, GlobalMaxPooling2D

#### 1.2. Test-Train Data

**Split the dataset**

def get\_files(directory):

if not os.path.exists(directory):

return 0

count=0

for current\_path,dirs,files in os.walk(directory):

for dr in dirs:

count+= len(glob.glob(os.path.join(current\_path,dr+"/\*")))

return count

train\_dir ="/content/drive/MyDrive/Colab Notebooks/dataset/Plant Leafs/train"

test\_dir="/content/drive/MyDrive/Colab Notebooks/dataset/Plant Leafs/val"

train\_samples =get\_files(train\_dir)

num\_classes=len(glob.glob(train\_dir+"/\*"))

test\_samples=get\_files(test\_dir)

print(num\_classes,"Classes")

print(train\_samples,"Train images")

print(test\_samples,"Test images")

#### 1.3. ImageDataGenerator

train\_datagen=ImageDataGenerator(

rescale=1./255,

shear\_range=0.2,

zoom\_range=0.2,

horizontal\_flip=True

)

test\_datagen=ImageDataGenerator(rescale=1./255)

input\_shape=(224,224,3)

train\_generator =train\_datagen.flow\_from\_directory(train\_dir,target\_size=(224,224),batch\_size=32)

test\_generator=test\_datagen.flow\_from\_directory(test\_dir,shuffle=True,target\_size=(224,224),batch\_size=32)

#### 1.4. CNN Model

model = Sequential()

model.add(Conv2D(32, (5, 5),input\_shape=input\_shape,activation='relu',name="conv2d\_1"))

model.add(MaxPooling2D(pool\_size=(3, 3),name="max\_pooling2d\_1"))

model.add(Conv2D(32, (3, 3),activation='relu',name="conv2d\_2"))

model.add(MaxPooling2D(pool\_size=(2, 2),name="max\_pooling2d\_2"))

model.add(Conv2D(64, (3, 3),activation='relu',name="conv2d\_3"))

model.add(MaxPooling2D(pool\_size=(2, 2),name="max\_pooling2d\_3"))

model.add(Flatten(name="flatten\_1"))

model.add(Dense(512,activation='relu'))

model.add(Dropout(0.25))

model.add(Dense(128,activation='relu'))

model.add(Dense(num\_classes,activation='softmax'))

model.summary()

validation\_generator = train\_datagen.flow\_from\_directory(

test\_dir,

target\_size=(224, 224),

batch\_size=32)

model.compile(optimizer='adam',loss = 'categorical\_crossentropy',metrics=['accuracy'])

history1 = model.fit(

train\_generator,*#egitim verileri*

steps\_per\_epoch=None,

epochs=2,

validation\_data=validation\_generator,

validation\_steps=None,

verbose=1,

callbacks=[ReduceLROnPlateau(monitor='val\_loss', factor=0.3,patience=3, min\_lr=0.000001)],

shuffle=True

)

model.save('/content/drive/MyDrive/Colab Notebooks/Model/plant\_disease\_Cnn.h5')

#### 1.5. VGG16 Modeli

def create\_Base\_model\_from\_VGG16():

model = VGG16(

weights = "imagenet",

include\_top=False,

input\_shape = (224,224, 3)

)

for layer in model.layers:

layer.trainable = False

return model

create\_Base\_model\_from\_VGG16().summary()

def add\_custom\_layers():

model = create\_Base\_model\_from\_VGG16()

x = model.output

x = tf.keras.layers.Flatten()(x)

x = tf.keras.layers.Dense(256, activation="relu")(x)

predictions = tf.keras.layers.Dense(num\_classes, activation="softmax")(x)

final\_model = tf.keras.models.Model(

inputs = model.input,

outputs = predictions)

final\_model.compile(loss='categorical\_crossentropy', optimizer='adam', metrics=['accuracy'])

return final\_model

add\_custom\_layers().summary()

validation\_generator = train\_datagen.flow\_from\_directory(

test\_dir,

target\_size=(224, 224),

batch\_size=32)

model\_from\_vgg16 = add\_custom\_layers()

history2 = model\_from\_vgg16.fit(

train\_generator,

steps\_per\_epoch=None,

epochs=5,

validation\_data=validation\_generator,

validation\_steps=None,

verbose=1,

callbacks=[ReduceLROnPlateau(monitor='val\_loss', factor=0.3,patience=3, min\_lr=0.000001)],

use\_multiprocessing=False,

shuffle=True

)

model\_from\_vgg16.save('/content/drive/MyDrive/ColabNotebooks/Model/model\_VGG16.h5')

pd.DataFrame(history1.history)[['accuracy','val\_accuracy']].plot()

plt.title("Accuracy")

plt.show()

pd.DataFrame(history1.history)[['loss','val\_loss']].plot()

plt.title("Loss")

plt.show()

### **2. Testing the Saved Model**

#### 2.1. CNN Model

import numpy as np

from keras.models import load\_model

from keras.preprocessing import image

model\_cnn=load\_model('/content/drive/MyDrive/Colab Notebooks/Model/plant\_disease\_Cnn.h5')

classes=list(train\_generator.class\_indices.keys())

def prepare(img\_path):

img = image.load\_img(img\_path, target\_size=(224,224))

x = image.img\_to\_array(img)

x = x/255

return np.expand\_dims(x, axis=0)

img\_url='/content/drive/MyDrive/Colab Notebooks/dataset/plant\_\_leaf/val/Apple\_\_Healthy/78e648c6-a360-4fa8-b8ab-1225b164b7fd\_\_\_RS\_HL 7243.JPG'

result\_cnn = model\_cnn.predict([prepare(img\_url)])

disease=image.load\_img(img\_url)

plt.imshow(disease)

classresult=np.argmax(result\_cnn,axis=1)

print(classes[classresult[0]])

#### 2.2. VGG16 Model

import numpy as np

from keras.models import load\_model

from keras.preprocessing import image

model\_vgg16=load\_model('/content/drive/MyDrive/Colab Notebooks/model/plant\_disease\_VGG16.h5')

classes=list(train\_generator.class\_indices.keys())

def prepare(img\_path):

img = image.load\_img(img\_path, target\_size=(224,224))

x = image.img\_to\_array(img)

x = x/255

return np.expand\_dims(x, axis=0)

img\_url='/content/drive/MyDrive/Colab Notebooks/dataset/plant\_\_leaf/val/Apple\_\_Healthy/78e648c6-a360-4fa8-b8ab-1225b164b7fd\_\_\_RS\_HL 7243.JPG'

result\_vgg16 = model\_vgg16.predict([prepare(img\_url)])

disease=image.load\_img(img\_url)

plt.imshow(disease)

classresult=np.argmax(result\_vgg16,axis=1)

print(classes[classresult[0]])

classes=list(train\_generator.class\_indices.keys())

def prepare(img\_path):

img = image.load\_img(img\_path, target\_size=(224,224))

x = image.img\_to\_array(img)

x = x/255

return np.expand\_dims(x, axis=0)

img\_url='/content/drive/MyDrive/Colab Notebooks/dataset/plant\_\_leaf/val/Apple\_\_Healthy/78e648c6-a360-4fa8-b8ab-1225b164b7fd\_\_\_RS\_HL 7243.JPG'

result\_alexnet = model\_alexnet.predict([prepare(img\_url)])

disease=image.load\_img(img\_url)

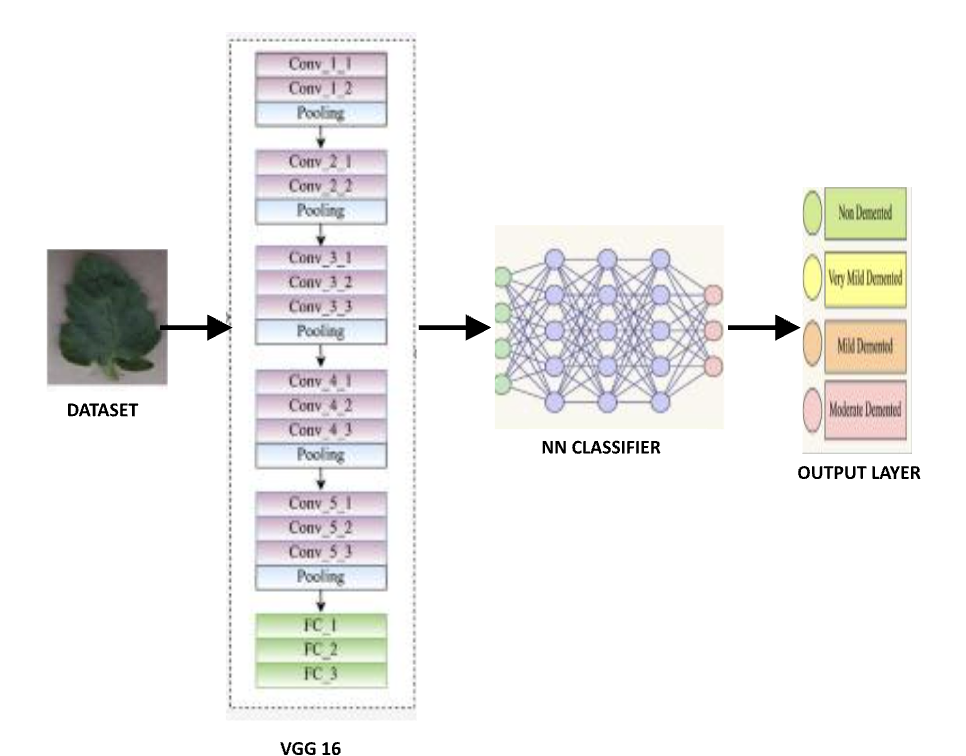
plt.imshow(disease)

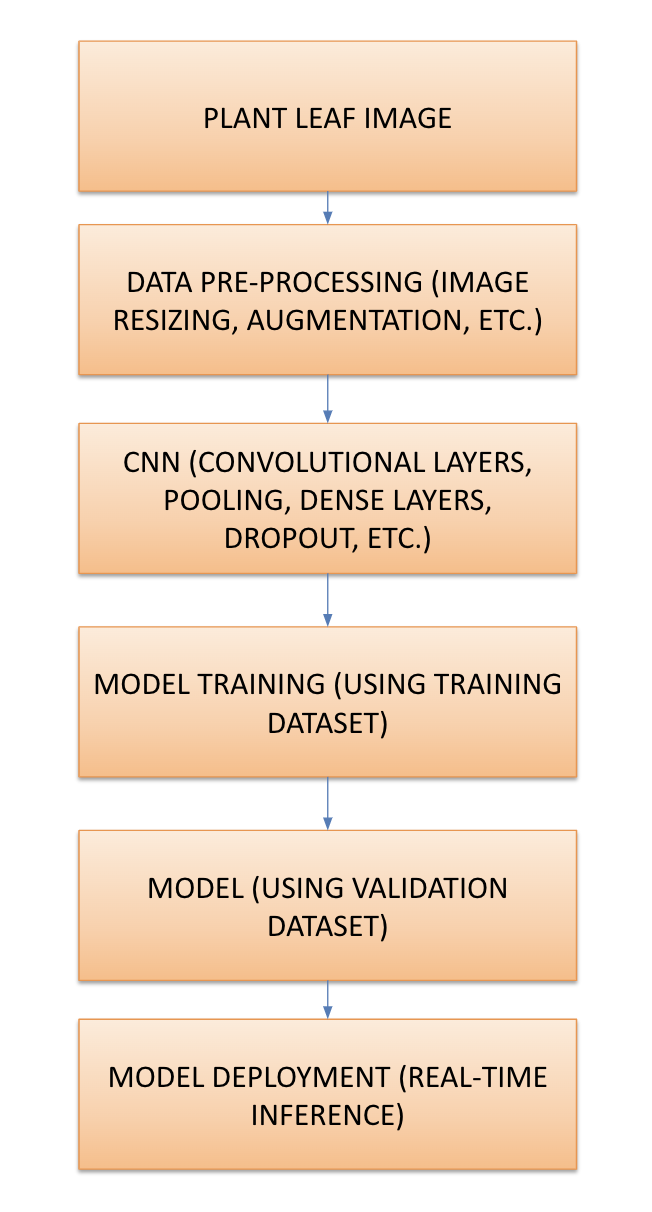
classresult=np.argmax(result\_alexnet,axis=1)

print(classes[classresult[0]])

# CHAPTER 8 PROJECT DESIGN

# 8.1 DATA FLOW DIAGRAM





# CHAPTER 9

**ADVANTAGES AND DISADVANTAGES**

# 9.1 ADVANTAGES

**1. Accuracy and Performance:**

- High Accuracy: CNNs and VGG16 are known for their high accuracy in image classification tasks, making them well-suited for plant disease detection.

- Feature Extraction: CNNs automatically extract hierarchical features from images, reducing the need for manual feature engineering.

**2. Automation and Efficiency:**

- Real-Time Detection: Once trained, the models can make predictions quickly, enabling real-time disease detection in practical settings.

- Scalability: The models can handle large datasets and grow with the addition of more data and classes.

**3. Transfer Learning:**

- Pre-trained Models: Using pre-trained models like VGG16 accelerates the training process and improves model performance, especially with limited data.

**4. Usability:**

- Interpretability: Visualizing predictions with images helps in understanding model results and their implications for agricultural management.

- Application Potential: The system can be adapted for various applications in agriculture, such as mobile apps for farmers and integration with IoT devices.

**5. Research Contribution:**

- Knowledge Advancements: Contributing to the growing field of AI in agriculture, this project can lead to further research and innovation.

# 9.2 DISADVANTAGES

**1. Data Requirements:**

- Large Datasets: CNNs require large amounts of labeled data for training, which can be difficult and expensive to obtain for certain plant species and diseases.

- Imbalanced Datasets: Uneven distribution of classes (e.g., more images of healthy plants than diseased ones) can lead to biased model performance.

**2. Training Complexity:**

- Resource Intensive: Training deep learning models can require significant computational resources, time, and specialized hardware.

- Tuning Challenges: Model optimization (e.g., adjusting hyperparameters) can be complex and time-consuming.

**3. Overfitting:**

- Risk of Overfitting: Models may perform well on the training data but poorly on unseen data, especially if the dataset is not sufficiently diverse.

**4. Maintenance and Updates:**

- Model Drift: As plant diseases evolve, models may need regular updates to remain accurate and relevant.

- Continuous Monitoring: Monitoring model performance and data quality over time is necessary to ensure reliable predictions.

**5. Interpretability:**

- Black Box Nature: Deep learning models, including CNNs and VGG16, can be seen as black boxes due to their complexity, making it difficult to understand the reasoning behind predictions.

**6. Ethical and Privacy Concerns:**

- Data Privacy: Handling agricultural data may raise privacy concerns, especially if combined with personal data or shared across multiple platforms.

- Security: Ensuring secure access and protection of models and data is essential.

# CHAPTER 10

**CONCLUSION AND FUTURE ENHANCEMENT**

# 10.1 CONCLUSION

## The project on plant disease detection using convolutional neural networks (CNNs) and the pre-trained VGG16 model demonstrates the power of deep learning in addressing significant challenges in agriculture. By leveraging advanced machine learning techniques, the project provides a robust and efficient solution for automatically identifying plant diseases from leaf images. This automation can aid farmers in making informed decisions about disease management, ultimately leading to improved crop health and productivity.

## 

## The use of CNNs and VGG16 as part of the detection system allows for accurate classification of plant diseases and healthy plants, providing valuable insights into the state of crops. Advanced image augmentation and optimization techniques help enhance model performance and generalization, ensuring reliable predictions across a variety of plant species and conditions.

## 

## Despite some challenges such as data scarcity, overfitting, and interpretability, the project's benefits, including increased accuracy and real-time disease detection, significantly contribute to sustainable agricultural practices. Future advancements in areas such as hybrid models, continuous learning, edge computing, and multi-modal analysis offer potential for even greater impact and wider applications in the agricultural sector.

## 

## In conclusion, this project serves as a stepping stone towards integrating AI technologies into agriculture, offering farmers innovative tools for disease detection and crop management. By continuing to explore and refine these methods, the potential for advancing food security and sustainable farming practices is substantial.

# 10.2 FUTURE ENHANCEMENT:

## The future scope of plant disease detection using convolutional neural networks (CNNs) and VGG16 involves a variety of opportunities for improvement and expansion. These opportunities can lead to more advanced, efficient, and impactful applications of the technology in the field of agriculture. Here are some potential directions for future work:

## 1. Advanced Model Architectures:

## - Hybrid Models: Combining different neural network architectures, such as CNNs and recurrent neural networks (RNNs), may lead to improved performance and the ability to handle more complex data.

## - Transformer-Based Models: Exploring transformer models for image classification tasks could lead to more efficient and powerful solutions.

## 2. Data Augmentation and Synthetic Data:

## - Synthetic Data Generation: Using techniques like generative adversarial networks (GANs) to create synthetic images can help address data scarcity and class imbalance.

## - Advanced Augmentation: Further exploration of data augmentation techniques can improve model robustness and generalization.

## 3. Explainable AI (XAI):

## - Interpretability Tools: Developing techniques to visualize and explain model predictions will increase trust and adoption among users.

## - Attention Mechanisms: Integrating attention mechanisms into models can help identify which parts of an image contribute most to predictions.

## 4. Edge Computing and Mobile Applications:

## - Edge Deployment: Deploying models on edge devices (e.g., mobile phones, IoT devices) can enable real-time, on-site disease detection.

## - Mobile Apps: Developing user-friendly mobile apps for farmers to easily identify diseases and access recommendations.

## 5. Multi-Modal Analysis:

## - Integration with Other Sensors: Combining image data with other sensor data (e.g., temperature, humidity, soil moisture) can lead to more comprehensive disease diagnosis.

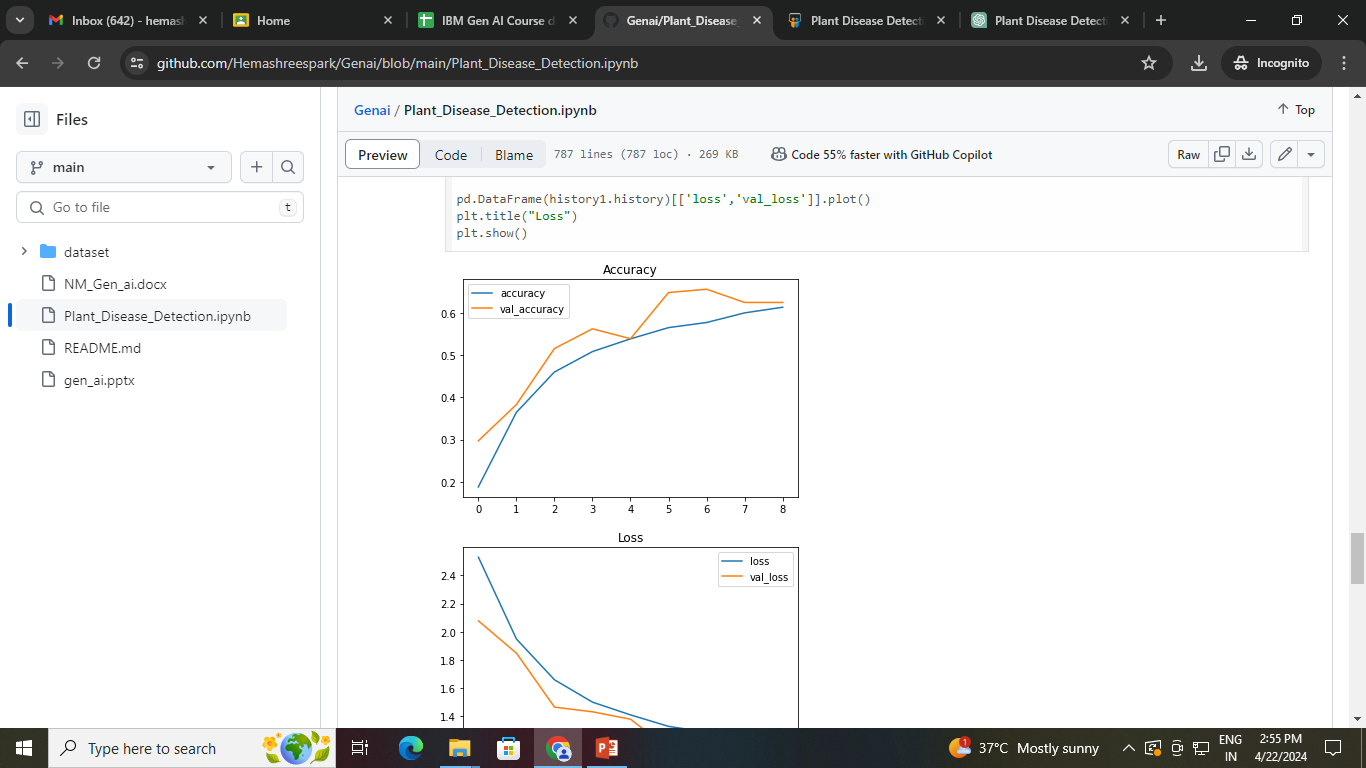
## - Contextual Analysis: Analyzing data from various sources (e.g., historical data, weather forecasts) for more accurate predictions.

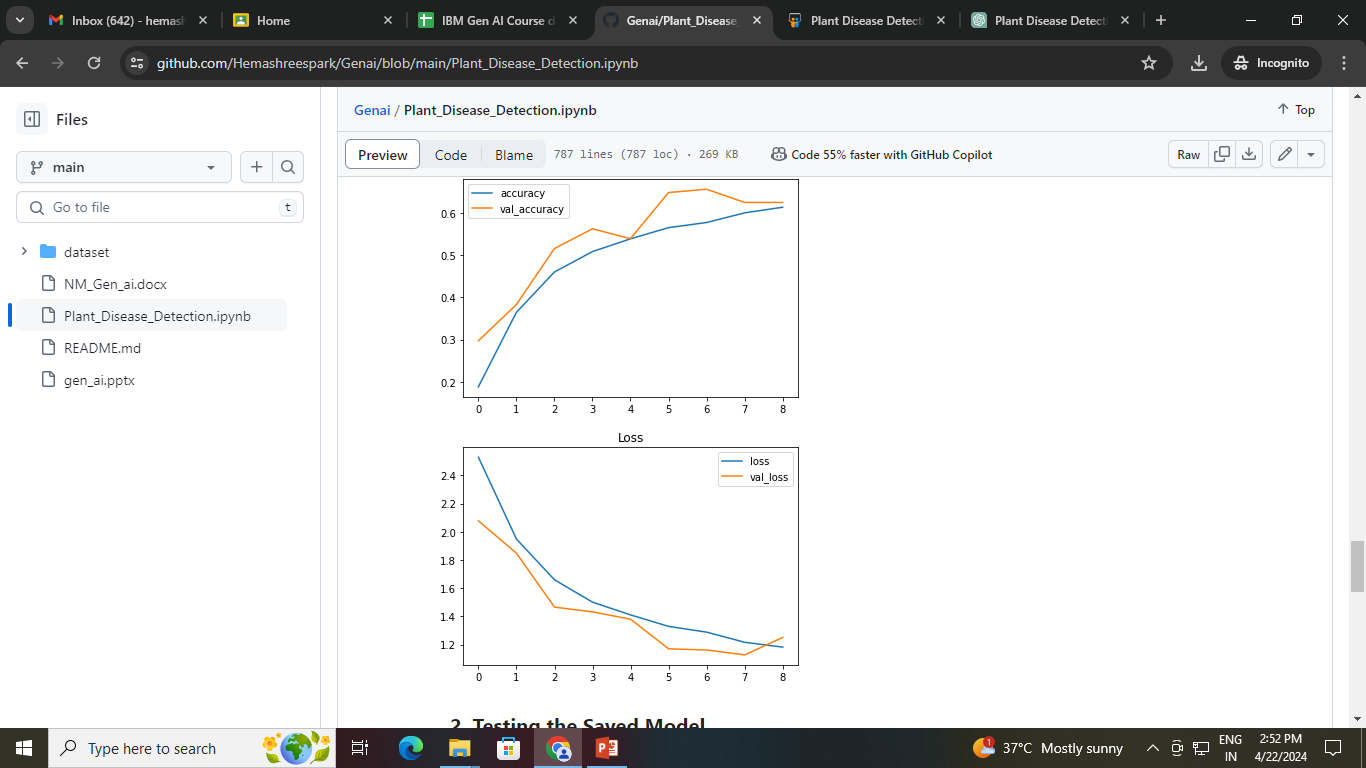
## 6. Continuous Learning and Adaptation:

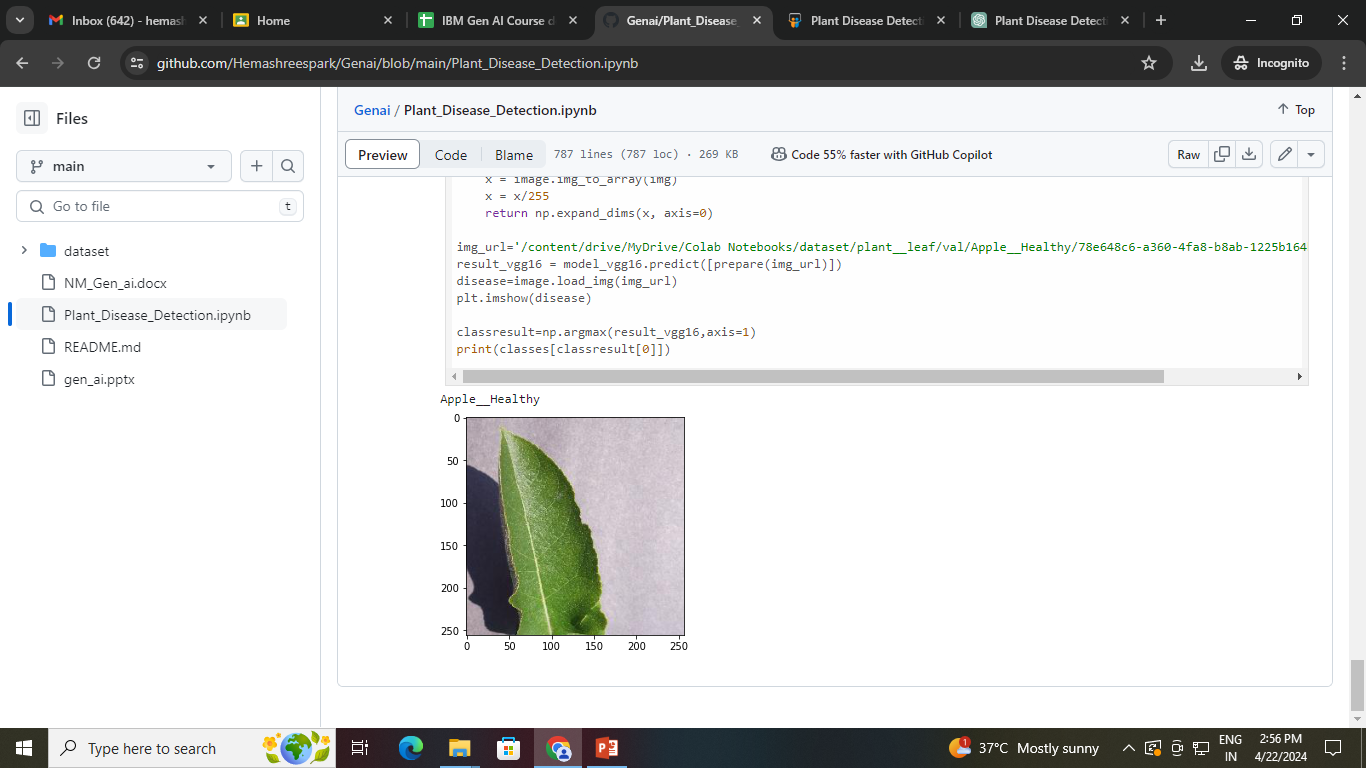
## - Online Learning: Implementing continuous learning mechanisms to adapt models to new data and evolving plant diseases.

## - Feedback Loops: Incorporating user feedback to improve model accuracy and update recommendations.

# APPENDIX SCREENSHOTS







**REFERENCES:**

1. Mohanty, S. P., Hughes, D. P., & Salathé, M. (2016). Using deep learning for image-based plant disease detection. Frontiers in plant science, 7, 1419.
2. Sladojevic, S., Arsenovic, M., Anderla, A., Culibrk, D., Stefanovic, D., & Crnojevic, V. (2016). Deep neural networks based recognition of plant diseases by leaf image classification. Computational intelligence and neuroscience, 2016.
3. Ferentinos, K. P. (2018). Deep learning models for plant disease detection and diagnosis. Computers and electronics in agriculture, 145, 311-318.
4. Nair, A., Kim, S., & Kumar, M. (2021). Plant disease detection and classification using deep learning: A review. Computers and Electronics in Agriculture, 189, 106449.
5. Sladojevic, S., Culibrk, D., Stefanovic, D., Arsenovic, M., & Anderla, A. (2018). Deep neural networks based recognition of plant diseases by leaf image classification. Plant Disease, 102(12), 2008-2016.

**GITHUB LINK:** <https://github.com/Hemashreespark/Genai.git>