ABSTRACT

The project titled "Plant Health Check Assistance" is an innovative framework that integrates advanced technologies such as OpenCV, TensorFlow, Keras, and various supporting libraries to provide an efficient solution for plant leaf disease detection and classification. The project aims to assist researchers, agricultural practitioners, and technology enthusiasts by offering a versatile tool for analyzing leaf images and accurately diagnosing plant health.

The core of the framework utilizes OpenCV for image pre-processing and manipulation tasks, preparing leaf images for subsequent analysis. Advanced deep learning models, trained on extensive datasets of plant leaves exhibiting various diseases, are employed to recognize patterns associated with different diseases and accurately diagnose the health status of the plant.

A unique feature of this project is the inclusion of a chatbot designed to assist users by addressing their plant health-related queries, providing tips, and offering solutions for plant health problems. This interactive tool enhances the user experience, making the system more accessible and user-friendly.

By accurately identifying plant diseases at an early stage, the project facilitates timely intervention and treatment, thereby potentially improving crop yields in agriculture and enhancing the vibrancy of gardens. This could have far-reaching implications, contributing to food security, environmental sustainability, and economic growth. The project serves as a testament to the role of technological innovation in promoting sustainable and resilient farming practices, demonstrating its usefulness to society.

LIST OF FIGURES

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

1.1 Introduction

Plant diseases pose a significant threat to global food security and agricultural productivity. Early and accurate detection of plant diseases can lead to effective interventions, minimizing the damage and ensuring healthy crop yield. This is where the application of Deep Learning in plant leaf disease detection comes into play. Deep Learning, a subset of Machine Learning, uses neural networks with many layers (hence the ‘deep’ in Deep Learning) to analyze various factors and parameters. In the context of plant leaf disease detection, these factors could be color, texture, shape, and spots on the leaves. The process begins with the collection of plant leaf images. These images form the dataset that the deep learning model will be trained on. The dataset includes images of healthy leaves and leaves affected by various diseases. Each image in the dataset is labelled with its corresponding class, i.e., the type of disease present or healthy. The next step is reprocessing. The images are resized to a uniform size, and their pixel intensities are normalized. This standardization is crucial for the model to process the images effectively.

The pre-processed images are then fed into a Convolutional Neural Network (CNN), a class of deep neural networks highly effective in analysing visual data. The CNN consists of multiple layers, including convolutional layers, pooling layers, and fully connected layers. Each layer of the network extracts features from the images, starting from low-level features like edges and textures to high-level features like shapes and patterns specific to different diseases.

The CNN, through a process of forward propagation, backpropagation, and optimization algorithms like Stochastic Gradient Descent (SGD), learns the optimal weights and biases. The goal is to minimize the difference between the predicted and actual labels of the images, i.e., the loss.

Once trained, the model can take in a new leaf image and predict the presence and type of disease with high accuracy. This prediction can then be used to advise appropriate treatment for the plant, thus saving it from further damage. Deep Learning provides a powerful tool for plant leaf disease detection. It automates the process, making it faster and more accurate than manual detection. This technology holds great promise for the future of sustainable agriculture, contributing to food security and the well-being of our planet.

This is a brief introduction to plant leaf disease detection using deep learning. The actual implementation involves more detailed steps and fine-tuning to achieve optimal results. However, the core idea remains the same: using the power of deep learning to protect our plants and ensure a healthy, abundant harvest.

1.2 Objective of the Project

Automate Disease Detection: To develop a system that can automatically detect and classify plant diseases based on leaf images. This reduces the need for manual inspection and provides a faster and more efficient method of disease detection.

Improve Accuracy: To improve the accuracy of plant disease diagnosis. Deep learning models, especially Convolutional Neural Networks (CNNs), have shown high accuracy in image classification tasks.

Early Detection: To enable early detection of diseases so that appropriate measures can be taken to prevent further spread. Early detection can minimize crop loss and increase agricultural productivity.

Aid Non-Experts: To assist farmers, gardeners, and non-experts in identifying plant diseases without the need for expert knowledge or consultation.

Smart Assistance: The smart assistance chatbot helps to educate users and assist them to their problems and query's.

Contribute to Sustainable Agriculture: By aiding in disease detection and prevention, the project contributes to sustainable agriculture practices, which is crucial for food security and environmental preservation.

Expandable Knowledge Base: The model can be trained on more types of diseases as more data becomes available, making the system more robust and comprehensive over time.

1.3 Motivation of the Project

Agricultural Development: There is a recognition that while many sectors have seen significant advancements, the area of plant leaf disease detection has not seen comparable progress. This project aims to address this gap.

Food Security: Plant diseases pose a threat to food security globally. By improving the accuracy and speed of disease detection, the project seeks to mitigate this risk.

Technological Advancement: The project leverages recent breakthroughs in deep learning, particularly convolutional neural networks (CNNs), to improve the accuracy of plant disease identification over traditional methods.

Global Impact: With the widespread availability of smartphones and advances in computer vision, the project has the potential for global reach, assisting even smallholder farmers in remote areas to diagnose crop diseases.

Overall, the project is driven by the need to enhance agricultural productivity through technology, ensuring better disease management and contributing to the stability of food supplies worldwide.

1.4 Literature Survey

Reference	Year	Methodology	Dataset	Performance	Key Contributions
[1] Mohanty et al. (2016)	2016	CNN-based approach with transfer learning	Plant Village	Accuracy: 99.35%	Transfer learning from ImageNet pre-trained models. Utilization of multiple CNN architectures for feature extraction.
[2] Sladojevic et al. (2016)	2016	Texture analysis combined with CNN	Swedish leaf dataset	Accuracy: 95.7%	Combined texture analysis with CNNs for improved disease recognition. Dataset augmentation techniques employed to increase dataset diversity.
[3] Ferentinos (2018)	2018	Ensemble of CNNs and transfer learning	Plant Village	Accuracy: 97.89%	Utilization of ensemble learning to improve classification accuracy. Transfer learning to adapt pre-trained models to the plant disease detection task.

[4] Brahimi et al. (2017)	2017	Transfer learning with AlexNet and SVM	Plant Village	Accuracy: 98.60%	Utilized AlexNet pre-trained model with SVM classifier. Data augmentation improved performance.
[5] Li et al. (2020)	2020	Hybrid deep learning model with data augmentation	Plant Village	Accuracy: 98.79%	Hybrid model combining CNN and traditional image processing. Extensive data augmentation applied.
[6] Too et al. (2019)	2019	Custom CNN with data augmentation and transfer learning	Plant Village	Accuracy: 99.70%	Developed custom CNN model. Enhanced performance with data augmentation and transfer learning.

CHAPTER 2

USER APPLICATION ENVIRONMENT

2.1 INTRODUCTION

User application environment means in which environment we are going to build our system and also which are the technologies we are going to use.

In our system we are using **Python, Jupyter, OpenCV, TensorFlow and Keras, SciPy, Matplotlib, Pillow, Seaborn.**

2.2 Python

Python is an interpreted, high-level, general-purpose programming language. It was created by Guido van Rossum and first launched in 1991. Python's design philosophy emphasizes code readability, and it notably uses significant whitespace. Its constructs and object-oriented approach aim to help programmers write clear, logical code for both small and large-scale projects.

Python is dynamically typed and supports multiple programming paradigms, including procedural, object-oriented, and functional programming. Python is often described as a "batteries included" language due to its comprehensive standard library. Python is a multi-paradigm programming language. It fully supports object-oriented programming and structured programming, and many of its features support functional programming and aspect-oriented programming (including by metaprogramming and metaobjects (magic methods)). Many other paradigms are supported via extensions, including design by contract and logic programming.

2.3 Jupyter

Jupyter is an open-source web app that allows creation and sharing of documents with live code, equations, visuals, and text. It supports multiple languages including Python, Julia, R, and Scala. Jupyter notebooks, its core feature, combine code and markdown cells for data exploration, analysis, visualization, and research sharing. It integrates with libraries like NumPy, pandas, Matplotlib, and TensorFlow, making it a popular choice for interactive computing in data science, machine learning, research, education, and journalism.

2.4 OpenCV

OpenCV, or Open-Source Computer Vision Library, is a comprehensive open-source software library designed for computer vision and machine learning tasks. It provides a vast collection of algorithms and functions that enable developers and researchers to perform various image and video processing tasks. Originally developed by Intel, OpenCV has evolved into a community-driven project supported by contributors worldwide.

One of the key features of OpenCV is its versatility and flexibility. It supports multiple programming languages, including C++, Python, Java, and MATLAB, making it accessible to a wide range of developers. This cross-language compatibility allows users to leverage OpenCV's capabilities in their preferred programming environment.

OpenCV offers a rich set of functionalities, including image and video I/O, image processing, feature detection, object detection and tracking, machine learning algorithms, and more. These features enable applications such as facial recognition, object tracking, augmented reality, robotics, medical image analysis, and autonomous vehicles.

OpenCV is known for its high performance and efficiency, with optimizations tailored for different hardware architectures. It is widely used in both academia and industry, powering applications across various domains, from research laboratories to commercial products.

Overall, OpenCV is a versatile, powerful, and widely adopted library that continues to drive innovation in the fields of computer vision and machine learning. Its open-source nature encourages collaboration and community-driven development, ensuring its continued relevance and evolution in the years to come.

2.5 TensorFlow and Keras

TensorFlow is an open-source machine learning framework developed by Google Brain for building and training deep learning models. It provides a comprehensive ecosystem of tools and libraries for developing and deploying machine learning applications across various domains. At its core, TensorFlow represents computations as computational graphs, where nodes represent mathematical operations and edges represent the flow of data between operations. This flexible and scalable architecture enables efficient execution of complex machine learning models on a wide range of hardware platforms, from CPUs and GPUs to specialized accelerators like TPUs (Tensor Processing Units).

Keras, on the other hand, is a high-level neural networks API written in Python that serves as an interface for building and training deep learning models. Initially developed as an independent project, Keras was integrated into TensorFlow as its official high-level API, making it the preferred interface for TensorFlow users. Keras provides a user-friendly and intuitive interface for constructing neural networks, abstracting away the complexities of low-level TensorFlow operations.

It allows developers to quickly prototype and experiment with different model architectures, loss functions, and optimizers, while maintaining compatibility with TensorFlow's powerful backend for efficient training and deployment. Together, TensorFlow and Keras empower researchers, developers, and data scientists to leverage the power of deep learning to solve a wide range of real-world problems, from image recognition and natural language processing to reinforcement learning and generative modelling.

2.6 SciPy

SciPy is an open-source scientific computing library for Python that builds upon the capabilities of NumPy, another popular library for numerical computing. It provides a vast array of functions and tools for solving mathematical, scientific, and engineering problems efficiently. SciPy covers a wide range of domains, including optimization, interpolation, integration, linear algebra, signal processing, and statistics. One of the key features of SciPy is its extensive collection of optimized numerical routines, implemented in efficient C and Fortran code. These routines allow users to perform complex mathematical computations with high performance and accuracy, making SciPy well-suited for both research and production environments.

SciPy's integration with NumPy enables seamless interoperability with other Python libraries, such as Matplotlib for data visualization and Pandas for data manipulation. This integration fosters a cohesive ecosystem of tools for scientific computing and data analysis in Python. SciPy also provides specialized modules for specific scientific disciplines, such as SciPy. Spatial for spatial algorithms, SciPy. Signal for signal processing, SciPy. Stats for statistical functions, and SciPy. Optimize for optimization algorithms. These modules offer advanced functionality and capabilities tailored to the needs of researchers and practitioners in various fields.

Overall, SciPy serves as a powerful and versatile toolkit for scientific computing in Python, empowering users to tackle complex problems across a wide range of domains. Its rich collection of functions, ease of use, and interoperability with other libraries make it an essential tool for scientists, engineers, and data analysts alike.

2.7 Matplotlib

Matplotlib is a comprehensive data visualization library for Python widely used in scientific computing and data analysis. It provides a flexible and user-friendly interface for creating high-quality plots, charts, and graphs to visualize data and communicate insights effectively. Matplotlib is highly customizable, allowing users to control every aspect of their visualizations, including colours, labels, fonts, and styles.

At its core, Matplotlib adopts a hierarchical structure where figures contain one or more axes objects, each representing an individual plot or subplot. Users can create a wide variety of plot types, including line plots, scatter plots, bar plots, histograms, pie charts, and more, using simple function calls. Matplotlib also supports advanced features such as 3D plotting, animation, and interactive plotting through integration with other libraries like mpl3d and Plotly. Matplotlib's versatility extends beyond basic plotting functionalities, offering support for complex layouts, annotations, legends, and axes customization. It seamlessly integrates with other Python libraries, such as NumPy for numerical computing and Pandas for data manipulation, making it a powerful tool for exploratory data analysis and presentation.

Matplotlib's popularity and extensive documentation have led to a vibrant community of users and contributors who continuously develop new features, share resources, and provide support through forums and online resources. Whether for scientific research, data visualization, or educational purposes, Matplotlib remains a cornerstone of the Python ecosystem for creating visually compelling and informative plots.

2.8 Pillow

Pillow, formerly known as Python Imaging Library (PIL), is a powerful and user-friendly library for image processing and manipulation in Python. It provides a wide range of functionalities for opening, manipulating, and saving images in various formats, including JPEG, PNG, TIFF, and BMP.

Pillow is designed to be easy to use, with a simple and intuitive interface that makes it accessible to both beginners and experienced developers. One of the key features of Pillow is its extensive support for basic image processing operations, such as resizing, cropping, rotating, and filtering. These operations can be performed with just a few lines of code, making Pillow ideal for tasks like batch processing, image enhancement, and data pre-processing in machine learning applications.

Pillow also offers advanced image manipulation capabilities, including support for transparency, alpha compositing, color manipulation, and pixel-level access. This enables users to create sophisticated visual effects, composite images, and perform advanced image analysis tasks.

Additionally, Pillow is actively maintained and regularly updated with new features and bug fixes, ensuring compatibility with the latest Python versions and operating systems. Its open-source nature encourages community contributions and extensions, further expanding its capabilities and versatility.

Overall, Pillow serves as a valuable tool for a wide range of applications, including web development, scientific research, digital art, computer vision, and multimedia processing. Its simplicity, versatility, and comprehensive documentation make it a popular choice among Python developers for image processing tasks.

2.9 Seaborn

Seaborn is a powerful Python data visualization library based on Matplotlib that specializes in creating attractive and informative statistical graphics. It provides a high-level interface for producing complex visualizations with minimal code, making it particularly useful for exploratory data analysis and presentation.

One of Seaborn's key features is its ability to create sophisticated statistical plots with ease. It offers a wide range of plot types, including scatter plots, line plots, bar plots, histograms, box plots, violin plots, and heatmaps, among others. These plots are designed to reveal insights into the underlying data distribution, relationships, and patterns, facilitating data-driven decision-making and interpretation.

Seaborn also provides seamless integration with Pandas, a popular data manipulation library in Python, allowing users to directly plot data stored in Pandas Data Frames. This tight integration streamlines the visualization workflow and enables rapid iteration and exploration of data. Another notable feature of Seaborn is its support for styling and theming, which allows users to customize the appearance of their plots easily. Seaborn comes with several built-in themes and color palettes, as well as tools for fine-tuning plot aesthetics, such as font sizes, grid styles, and plot sizes.

Overall, Seaborn is a versatile and user-friendly library that empowers Python developers and data scientists to create professional-quality statistical visualizations efficiently. Its rich functionality, ease of use, and aesthetic appeal make it a valuable tool for data visualization tasks in various domains, including academic research, business analytics, and data journalism.

CHAPTER 3

SYSTEM ANALYSIS

3.1 Introduction

Systems analysis is "the process of studying a procedure in order to identify its goals and purposes and create system and procedures that will achieve them in an efficient way". Another view sees system analysis as a problem-solving technique that breaks down a system into its component pieces for the purpose of the studying how well those component parts work and interact to accomplish their purpose.

3.2 Scope of the project

The “Plant Health Check Assistance” The scope of the “Plant Leaf Disease Detection using Deep Learning” project encompasses several key areas:

Deep Learning Model Development: The project involves the creation and training of deep learning models to accurately identify and classify various plant diseases based on leaf images. This includes data collection, pre-processing, model selection, training, validation, and testing.

Image Processing: The project utilizes OpenCV for image pre-processing and manipulation tasks, preparing leaf images for subsequent analysis.

This includes image resizing, color space conversion, noise reduction, edge detection, and contour extraction.

Web User Interface: A user-friendly web interface is developed to allow users to easily interact with the system. Users can upload leaf images, view the analysis results, and receive recommendations for disease treatment.

- **Chatbot Integration:** The project includes a chatbot that assists users by answering their plant health-related queries, providing tips, and offering solutions for plant health problems. This interactive tool enhances the user experience, making the system more accessible and user-friendly.

- **Real-time Assistance:** The system is designed to provide real-time assistance to farmers, gardeners, and researchers, helping them maintain the health and productivity of their plants.

- **Societal Impact:** By accurately identifying plant diseases at an early stage, the project has the potential to significantly improve crop yields in agriculture and enhance the vibrancy of gardens. This could contribute to food security, environmental sustainability, and economic growth.

- **Future Enhancements:** The project's scope also includes potential future enhancements such as expanding the range of detectable diseases, improving the accuracy of the deep learning models, integrating with other agricultural management systems, and developing a mobile application for easier access.

In summary, the project aims to develop an intelligent, user-friendly system capable of detecting and identifying plant diseases at an early stage, thereby facilitating timely intervention and treatment. It serves as a testament to the role of technological innovation in promoting sustainable and resilient farming practices.

3.3 Requirement Specification

The presentation of the Software Requirements Specification (SRS) gives a review of the whole SRS with reason, scope, definitions, abbreviations, contractions, references and diagram of the SRS. The point of this report is to assemble, dissect, and give a top to bottom knowledge of the total "Plant Health Check Assistance" by the difficult articulation in detail.

3.3.1 Purpose

The Purpose of the Software Requirements Specification is to give the specialized, Functional and non-useful highlights, needed to build up a console application. The whole application intended to give client adaptability to finding the briefest as well as efficient way. To put it plainly, the motivation behind this SRS record is to give an itemized outline of our product item, its boundaries and objectives. This archive depicts the task's intended interest group and its UI, equipment and programming prerequisites. It characterizes how our customer, group and crowd see the item and its usefulness.

3.4 Hardware/Software Requirements

All computer software needs certain hardware components or other software resources to be present on a computer. The most common set of requirements defined by any operating system or software application is the physical computer resources, also known as Hardware. Software requirements deal with defining software resource requirements and prerequisites that need to be installed on a computer to provide optimal functioning of an application.

3.4.1 Hardware Requirements

- RAM: MINIMUM OF 8 GB RECOMMENDED
- STORAGE: 100 GB
- CPU: A modern multi-core processor (Intel i5, i7, or equivalent AMD processors)
- GPU: A dedicated Graphics Processing Unit (GPU) is highly recommended for deep learning tasks.

3.4.2 Software Requirements

- FRAMEWORK: FLASK, BOTPRESS CHATBOT API
- LANGUAGE: PYTHON
- SOFTWARE: JUPYTER NOTEBOOK/ VS CODE
- FRONT END: HTML5, CSS3, JAVSCRIPT
- LIBRARIES: TENSERFLOW, KERAS, NUMPY, PANDAS, SEABORN, OPEN CV, SCIPY, PILLOW, MATPLOTLIB

3.5 User Requirements

User requirements are functions or features that must be included in the system to satisfy the desired needs and be acceptable by the user. Based on this, the user requirements for this project are as follows:

- Users should be able to upload images of plant leaves to the web interface.
- Users should be able to view the analysis results, including the type of disease (if any) detected in the uploaded leaf image.
- Users should be able to interact with the chatbot, ask questions, and receive responses related to plant health and disease treatment.

3.6 product Functions

The “Plant Leaf Disease Detection using Deep Learning” project has several key functions:

Image Upload: Users can upload images of plant leaves to the web interface. The system accepts these images as input for analysis.

Image Processing: The system uses OpenCV to pre-process and manipulate the uploaded images, preparing them for disease detection. This includes tasks such as resizing, color space conversion, noise reduction, and extraction of the region of interest.

Disease Detection: The system employs deep learning models to analyze the processed images and detect any signs of disease. These models have been trained on extensive datasets of plant leaves with various diseases, enabling them to recognize patterns associated with different diseases.

Disease Classification: In addition to detecting the presence of disease, the system also classifies the type of disease present. This is done by comparing the patterns in the image with those learned by the deep learning models during training.

Results Presentation: The system presents the results of the analysis to the user, indicating whether a disease was detected and, if so, what type of disease it is.

Chatbot Assistance: The system includes a chatbot that can interact with users, answer their questions about plant health, and provide tips and solutions for treating plant diseases.

Real-time Assistance: The system is designed to provide real-time assistance, helping users maintain the health and productivity of their plants by providing immediate analysis and recommendations.

These functions work together to provide a comprehensive solution for detecting and classifying plant diseases, making the system a valuable tool for farmers, gardeners, researchers, and anyone else interested in plant health.

3.6.1 Functional Requirements

1. Image Upload: Users should be able to upload images of plant leaves to the system for disease detection.

2. Disease Detection: The system must accurately detect and classify diseases in plant leaves based on the uploaded images.

3. User Interface: The system should have a user-friendly interface that allows users to easily upload images and view disease detection results.

4. Feedback Mechanism: Users should have the ability to provide feedback on the accuracy of disease detection results, which can be used to improve the system.

5. Scalability: The system should be able to handle a large volume of image uploads and process them efficiently.

3.6.2 Non-Functional Requirements

1. Accuracy: The system must achieve a high level of accuracy in disease detection to provide reliable results to users.

2. Performance: The system should be able to process images and provide disease detection results within a reasonable timeframe to ensure user satisfaction.

3. Reliability: The system should be reliable and available for use whenever users need to upload images for disease detection.

4. Security: The system must ensure the confidentiality and integrity of user data, including uploaded images and disease detection results.

5. Usability: The user interface should be intuitive and easy to navigate, even for users with limited technical knowledge.

6. Compatibility: The system should be compatible with a variety of devices and browsers to ensure accessibility for all users.

7. Maintainability: The system should be easy to maintain and update, allowing for improvements and bug fixes to be implemented efficiently.

3.7 Feasibility Study

The feasibility study helps to find solutions to the problems of the project. The solution is given how looks like a new system look like.

There are four types of feasibility study, they are

- Technical Feasibility
- Economic Feasibility
- Operational Feasibility
- Behaviour Feasibility

1. Technical Feasibility

The technical feasibility assesses whether the project can be implemented with current technology and resources. For this project:

Deep Learning Models: The use of TensorFlow and Keras is technically feasible as they are well-established frameworks for building deep learning models.

Image Processing: OpenCV provides robust tools for image processing, which are crucial for the project and are widely supported.

Web UI and Chatbot: Modern web technologies like HTML, CSS, and JavaScript, along with backend frameworks, can support the development of the web UI and chatbot.

Hardware: The availability of high-performance computing resources, such as GPUs for model training and inference, is feasible but may require investment.

2. Economic Feasibility

This aspect evaluates the cost-effectiveness of the project:

Development Costs: Initial costs include software development, purchasing of hardware, and acquisition of datasets.

Operational Costs: Recurring costs involve hosting the web application, maintaining the system, and updating the models.

Return on Investment: By improving crop yields and reducing losses due to diseases, the project has the potential for a significant positive economic impact.

3. Operational Feasibility

This area examines the practicality of the project's operation within the current operational capability:

User Adoption: The system's ease of use and real-time analysis can lead to high adoption rates among target users.

Maintenance: Ongoing maintenance is feasible with a dedicated team for system updates and customer support.

Integration: The system can be integrated with existing agricultural management tools, enhancing its operational value.

4. Behavioural Feasibility

Behavioural feasibility looks at how well the project aligns with human behaviour and social context:

User Engagement: The interactive chatbot and user-friendly web interface are designed to engage users effectively.

Training Needs: Users will require minimal training, as the system is designed to be intuitive.

Social Impact: The project aligns with societal goals of sustainable farming practices and food security, which can drive positive behaviour towards technology adoption.

In conclusion, the "Plant Leaf Disease Detection using Deep Learning" project appears to be feasible in all examined areas. It leverages current technology, offers economic benefits, is operationally practical, and aligns well with user behaviours and social goals.

CHAPTER 4

SYSTEM DESIGN

4.1 Introduction to Design document

The Software Design Document for our Plant Leaf Health check assistance project serves as the blueprint for the development and implementation of our innovative solution. This document outlines the architecture, components, and functionalities of our software system, aimed at accurately identifying and diagnosing diseases in plant leaves through advanced deep learning techniques. By meticulously detailing the system's design principles, data flow, algorithms, and user interfaces, this document ensures clarity, coherence, and efficiency throughout the development lifecycle, empowering our team to deliver a robust and effective solution for agricultural disease management.

4.1.1 Scope

The software design scope for the “Plant Health Check Assistance” project is a comprehensive system designed to detect and manage plant diseases at an early stage. The project’s scope includes the development of a system architecture that integrates deep learning models, image pre-processing techniques, and data storage solutions. This ensures the system’s scalability and flexibility, allowing for future enhancements and adaptations.

The project leverages advanced deep learning methodologies for disease detection and classification. This involves implementing algorithms for image analysis, feature extraction, and disease classification. The system is capable of fine-tuning pre-trained models, designing custom neural network architectures, and optimizing computational resources for efficient processing of large-scale image datasets.

In addition to the technical aspects, the project also focuses on user interaction. It includes the development of user-friendly interfaces for seamless interaction with the system. Users can upload images, view diagnosis results, and access supplementary information on detected diseases through these interfaces.

The project also considers system integration with existing agricultural management platforms and compatibility with diverse hardware environments. This enhances its operational value and broadens its applicability.

The project aims to provide a robust, efficient, and effective solution for early detection and management of plant leaf diseases. By doing so, it contributes to improved crop health and food security, demonstrating its usefulness to society.

4.2 Existing System

The current system for plant identification heavily relies on manual methods, where human observation and expertise play a central role. However, this approach is marred by several limitations. Firstly, it is time consuming and labour-intensive, hindering prompt decision-making in agricultural practices and research.

Human errors are inherent in the process, leading to potential misidentifications and overlooking early signs of diseases. Scalability is a challenge, making it impractical for large-scale plant studies or agricultural operations.

Moreover, the effectiveness of identification is highly dependent on the expertise of individuals, posing difficulties for non-experts. The lack of early disease detection further impacts crop health and yield. Additionally, the system is hindered by seasonal variations, resource-intensive training requirements, limited accessibility in remote areas, and cumbersome data handling.

The Automated Plant Leaf Detection System aims to address these limitations by providing a technologically advanced, efficient, and user-friendly alternative to manual plant identification, thereby revolutionizing current practices.

4.2.1 Disadvantages of Existing System

- Limited Early Disease Detection
- Data Management
- Time-Consuming Processes
- Scalability Challenges
- Expertise Dependency
- Resource Intensiveness
- Limited Adaptability

4.3 Proposed System

The proposed system for the plant leaf disease detection project aims to leverage advanced deep learning techniques to overcome the limitations of existing methods. It involves the development of a comprehensive software solution consisting of several key components. Firstly, the system will feature a robust image pre-processing module to enhance the quality and consistency of input data, ensuring optimal performance of the subsequent analysis.

The core of the system will be built upon state-of-the-art deep learning models, specifically convolutional neural networks (CNNs), trained on extensive datasets of labelled plant leaf images. These models will be fine-tuned and optimized to accurately detect and classify various diseases across a wide range of plant species.

Additionally, the proposed system will incorporate adaptive algorithms for real-time disease monitoring and early detection, enabling timely intervention and mitigation strategies to be deployed.

A smart assistance in the form of a chatbot will be provided that would guide the user to use the application and not only that it will provide assistance in solving their queries and needs of information.

Furthermore, to enhance accessibility and usability, the system will include intuitive user interfaces accessible through web-based platforms. This interface will allow users, including farmers and agricultural professionals, to easily upload images, receive diagnosis results, and access relevant information and recommendations for disease management.

Overall, the proposed system aims to provide a reliable, scalable, and user-friendly solution for plant leaf disease detection, contributing to improved crop health, yield, and agricultural sustainability.

4.3.1 Advantages of Proposed System

- **Efficiency Gains:** The proposed system will significantly enhance efficiency by automating the plant identification process, enabling rapid and accurate species classification compared to time-consuming manual methods.

- **Early Disease Detection:** Incorporating advanced algorithms, the system will offer early detection of diseases or abnormalities in plants, providing timely information for proactive disease management and safeguarding crop health.

- **Technological Precision:** Leveraging computer vision and machine learning ensures a higher level of precision in plant identification, allowing the system to adapt and learn from diverse datasets for improved accuracy.

- **User-Friendly Interface:** A user-friendly interface will be designed for accessibility, making the system easily adoptable by individuals with varying levels of technical expertise, promoting inclusivity and widespread use.

- **Scalability**: The proposed system is designed to scale effectively, accommodating large-scale plant studies and agricultural operations, providing a practical solution for varied applications.

- **Adaptability:** The system's adaptability ensures it can handle diverse plant species and environmental conditions, offering a versatile tool for researchers, farmers, and botanists.

- **Reduced Environmental Impact:** By eliminating the need for physical sample collection, the system minimizes the environmental impact associated with traditional plant identification methods, contributing to ecological sustainability.

- **Cost-Effectiveness:** While initial implementation may involve costs, the long-term benefits of increased efficiency and reduced errors contribute to overall cost effectiveness in agricultural and research operations.

4.4 Modules

4.4.1 Collection datasets:

• We are going to collect datasets for the prediction from the kaggle.com

• The data sets consist of 3 Classes

4.4.2 Data Pre-Processing:

• In data pre-processing we are going to perform some image pre-processing techniques on the selected data

• Image Resize.

• And Splitting data into train and test.

4.4.3 Feature Extraction and Data Modelling:

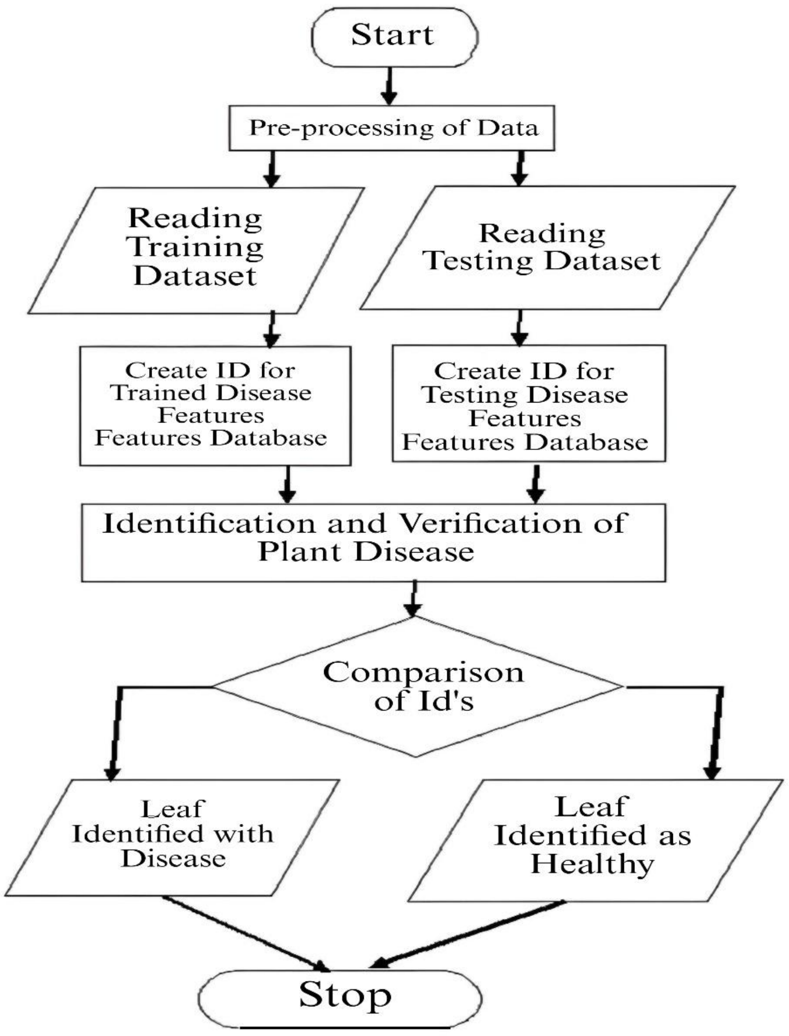
- The spitted train data are passed as input to the CNN algorithm, which helps in feature extraction.
- The feature extracted model from images are stored as model file,

4.4.4 Classification:

- The feature is split into train and test using build model.
- The feature is passed as input to the machine learning classification algorithms like Naïve bayes, SVM and random forest tree.
- Check for the accuracy result.

4.5 System Architecture Diagram

In below figure this is architecture of our project. User can capture and upload the image that want to predicate. The train module which consists of different plant leaf disease that already train and store in database. If new image inserted that match with dataset and show appropriate result to the user.


Start
Pre-processing of Data
Reading Training Dataset
Reading Testing Dataset
Create ID for Trained Disease Features Features Database
Create ID for Testing Disease Features Features Database
Identification and Verification of Plant Disease
Comparison of Id's
Leaf Identified with Disease
Leaf Identified as Healthy
Stop

System Architecture Diagram

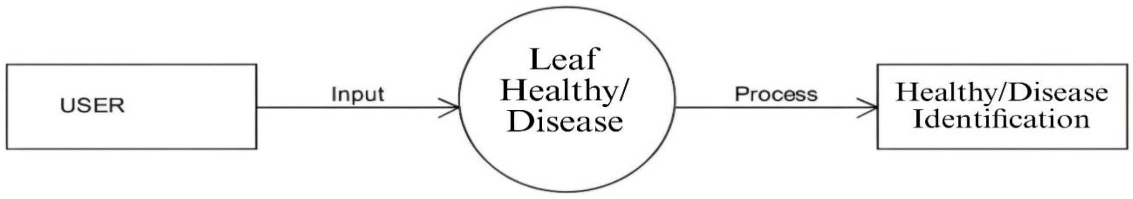
4.6 Data Flow Diagram

4.6.1 DFD 1:

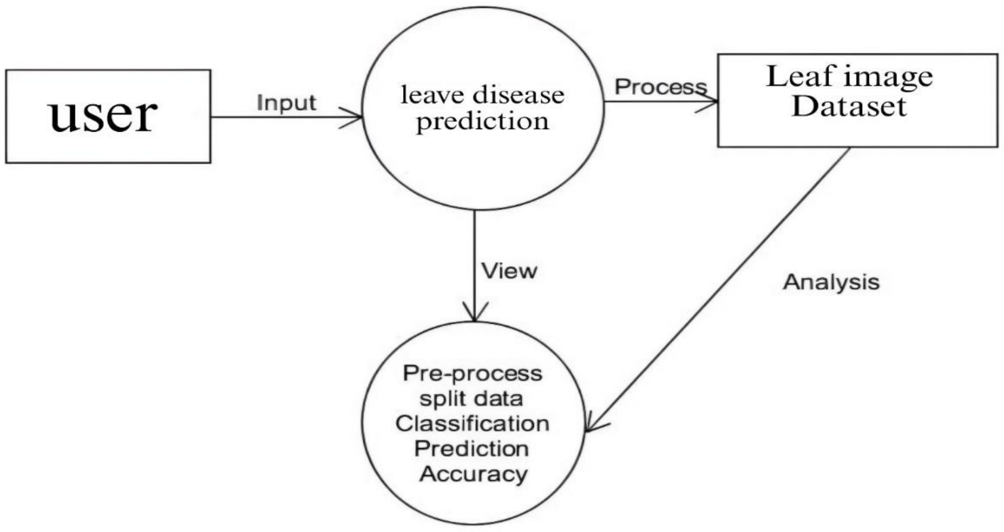
A data flow diagram (DFD) maps out the flow of information for any process or system. It uses defined symbols like rectangles, circles and arrows, plus short text labels, to show data inputs, outputs, storage points and the routes between each destination.

Data flowcharts can range from simple, even hand-drawn process overviews, to in-depth, multi-level DFDs that dig progressively deeper into how the data is handled. They can be used to analyze an existing system or model a new one. Like all the best diagrams and charts, a DFD can often visually “say” things that would be hard to explain in words, and they work for both technical and nontechnical audiences, from developer to CEO.

DFD 0

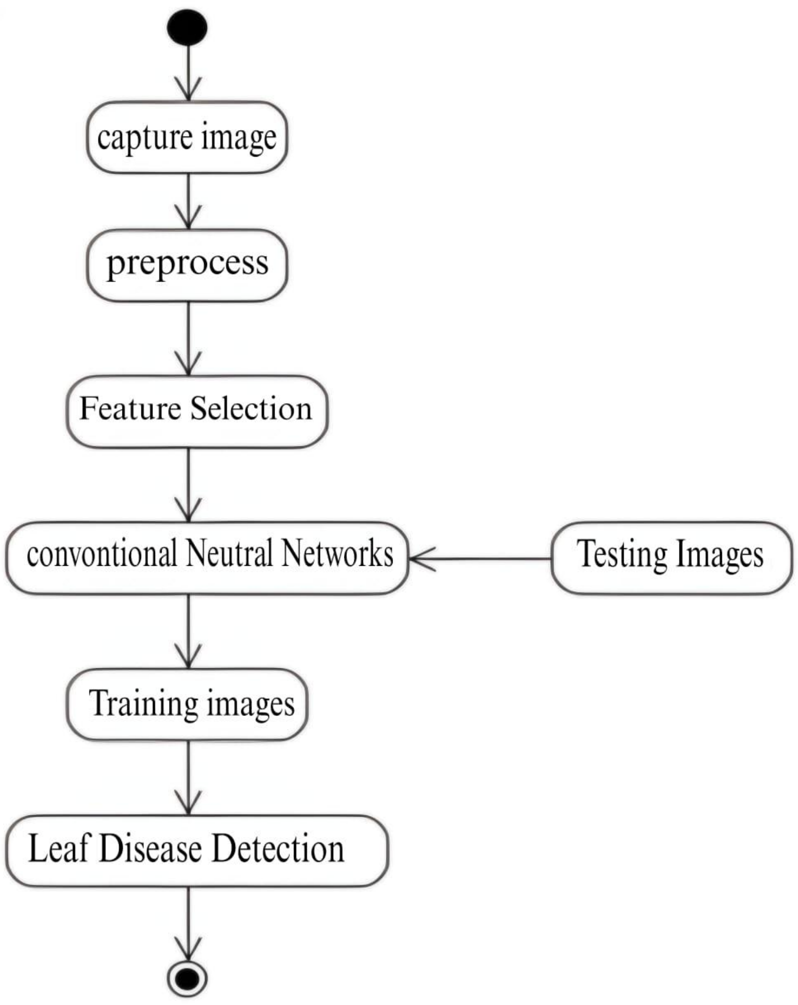


DFD 1



4.7 Activity diagram

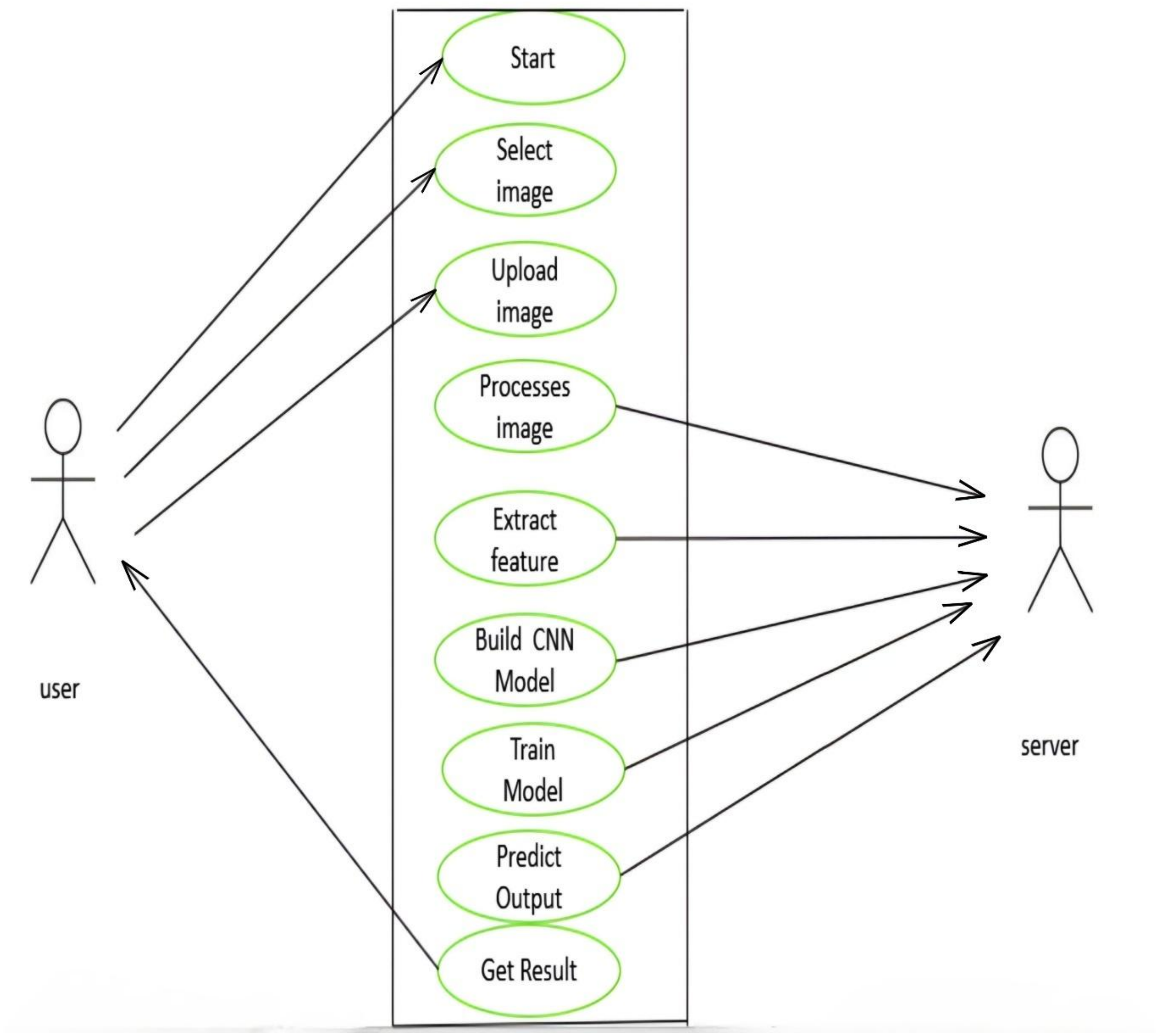
Activity Diagrams describe how activities are coordinated to provide a service which can be at different levels of abstraction. Typically, an event needs to be achieved by some operations, particularly where the operation is intended to achieve a number of different things that require coordination, or how the events in a single use case relate to one another, in particular, use cases where activities may overlap and require coordination. It is also suitable for modelling how a collection of use cases coordinates to represent business workflows.



Activity diagram

4.8 Use Case Diagram

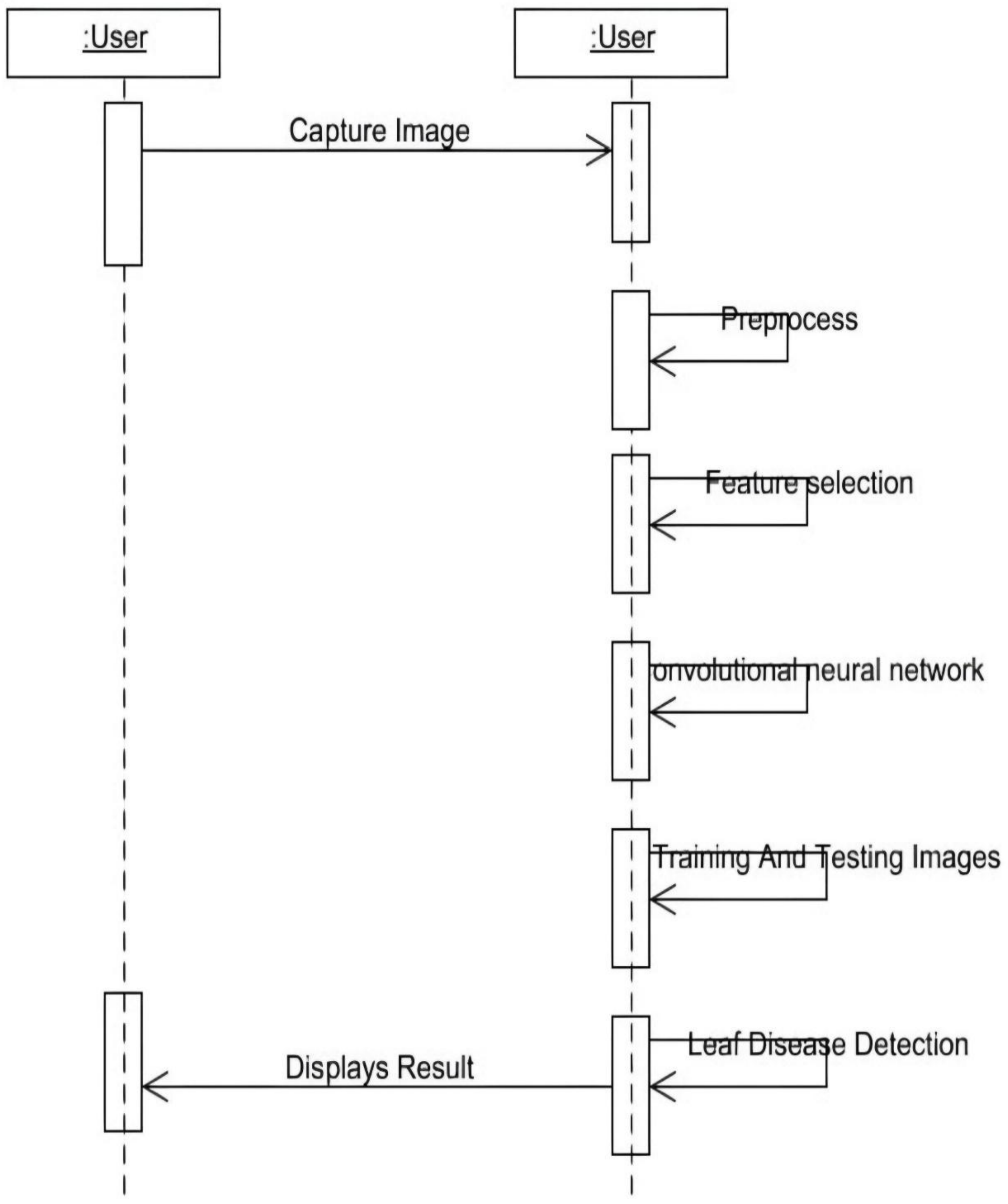
A UML use case diagram is the primary form of system/software requirements for a new software program underdeveloped. Use cases specify the expected behaviour (what), and not the exact method of making it happen (how). Use cases once specified can be denoted both textual and visual representation (i.e., use case diagram). A key concept of use case modelling is that it helps us design a system from the end user's perspective. It is an effective technique for communicating system behaviour in the user's terms by specifying all externally visible system behaviour.



Use Case Diagram

4.9 Sequence Diagram

The below is a sequence diagram is an interaction diagram that demonstrates the order in which processes collaborate with one another. It is made up of message sequence diagrams, also known as event diagrams, sequence diagrams, and message flow diagrams.



Sequence Diagram

CHAPTER 5

SYSTEM IMPLEMENTATION

5.1 Description

The plant Health check assistance project is developed using Python, harnessing both object-oriented and procedural programming methodologies. Python's dynamic typing and automatic garbage collection optimize development efficiency, while its support for multiple programming paradigms, including procedural, object-oriented, and functional programming, provides adaptability. Deep learning techniques like CNN (Convolutional neural networks) are employed to analyze leaf images for disease detection. To foster user engagement and confidence, users are educated about the system's benefits, offered guidance, and informed of the necessity for the server program to be operational for viewing results. Implementation involves deploying the software in its real-world environment, ensuring it satisfies user requirements and operates seamlessly. Continuous user feedback guides ongoing improvements, aiming to enhance the system's usability and efficacy.

By adhering to these principles, the plant leaf disease detection system aims to provide a user-friendly, efficient, and reliable solution for identifying and managing plant diseases using advanced deep learning techniques within the Python programming language ecosystem.

5.2 Machine Learning vs Deep Learning

In the realm of plant Health check assistance using deep learning, distinctions between machine learning (ML) and deep learning (DL) exist, though experts continue to debate and explore these concepts. ML predates DL and can be considered a broader concept encompassing various techniques, including DL. Here are the key differences:

1) Data Requirements: DL algorithms typically require large volumes of data to effectively learn and extract meaningful patterns from the input. In contrast, ML models may achieve satisfactory performance with smaller datasets by leveraging expert-defined features.

2) Feature Extraction: DL algorithms autonomously learn to extract relevant features directly from the input data, whereas in ML, features are often handcrafted by domain experts.

3) Hardware Requirements: DL models often demand high-performance computing resources, such as GPUs, due to their complex architecture and computational intensity. ML algorithms, on the other hand, can operate efficiently on standard CPUs.

4) Problem Solving Approach: ML approaches often involve breaking down the problem into manageable components, solving them individually, and combining the results. DL, however, tends to tackle problems end-to-end, learning hierarchical representations directly from raw data.

5) Training Time: DL algorithms typically require longer training times compared to traditional ML algorithms due to their deeper architectures and the vast amount of data they process.

In the context of plant leaf disease detection, leveraging DL offers the potential for more accurate and robust models by automatically learning relevant features directly from leaf images. However, it comes with the trade-off of increased computational resources and training time compared to traditional ML approaches.

5.3 Deep Learning Overview

The term Deep Learning or Deep Neural Network refers to Artificial Neural Networks (ANN) with multi layers. Over the last few decades, it has been considered to be one of the most powerful tools, and has become very popular in the literature as it is able to handle a huge amount of data. The interest in having deeper hidden layers has recently begun to surpass classical methods performance in different fields; especially in pattern recognition.

One of the most popular deep neural networks are the Convolutional Neural Network (CNN).

It takes this name from mathematical linear operation between matrixes called convolution. CNN have multiple layers; including convolutional layer, non-linearity layer, pooling layer and fully connected layer.

The convolutional and fully- connected layers have parameters but pooling and non-linearity layers don't have parameters. The CNN has an excellent performance in machine learning problems. Specially the applications that deal with image data, such as largest image classification data set (Image Net), computer vision, and in natural language

processing (NLP) and the results achieved were very amazing. In this paper we will explain and define all the elements and important issues related to CNN, and how these elements work.

In addition, we will also state the parameters that effect CNN efficiency.

This paper assumes that the readers have adequate knowledge about both machine learning and artificial neural network.

Convolutional Neural Network has had ground breaking results over the past decade in a variety of fields related to pattern recognition; from image processing to voice recognition. The most beneficial aspect of CNNs is reducing the number of parameters in ANN. This achievement has prompted both researchers and developers to approach larger models in order to solve complex tasks, which was not possible with classic ANNs.

The most important assumption about problems that are solved by CNN should not have features which are spatially dependent. In other words, for example, in a chest X ray detection application, we do not need to pay attention to where the chest X rays are located in the images. The only concern is to detect them regardless of their position in the given images. Another important aspect of CNN, is to obtain abstract features when input propagates toward the deeper layers.

5.4 CNN

Convolutional Neural Networks have a different architecture than regular Neural Networks. Regular Neural Networks transform an input by putting it through a series of hidden layers. Every layer is made up of a set of neurons, where each layer is fully connected to all neurons in the layer before. Finally, there is a last fully-connected layer — the output layer — that represent the predictions.

Convolutional Neural Networks are a bit different. First of all, the layers are organised in 3 dimensions: width, height and depth. Further, the neurons in one layer do not connect to all the neurons in the next layer but only to a small region of it. Lastly, the final output will be reduced to a single vector of probability scores, organized along the depth dimension.

- Module 1: Region Proposal. Generate and extract category independent region proposals, e.g., candidate bounding boxes.

- Module 2: Feature Extractor. Extract feature from each candidate region, e.g., using a deep convolutional neural network.

- Module 3: Classifier. Classify features as one of the known classes, e.g., CNN classifier model.

5.4.1 Convolutional Neural Networks have the following layers

- Convolutional
- ReLU Layer
- Pooling
- Fully Connected Layer

Step1: Convolution Layer

Convolutional neural networks apply a filter to an input to create a feature map that summarizes the presence of detected features in the input.

Step2: ReLU Layer

In this layer, we remove every negative value from the filtered images and replaces them with zeros. It is happening to avoid the values from adding up to zero. Rectified Linear unit (ReLU) transform functions only activates a node if the input is above a certain quantity. While the data is below zero, the output is zero, but when the information rises above a threshold. It has a linear relationship with the dependent variable.

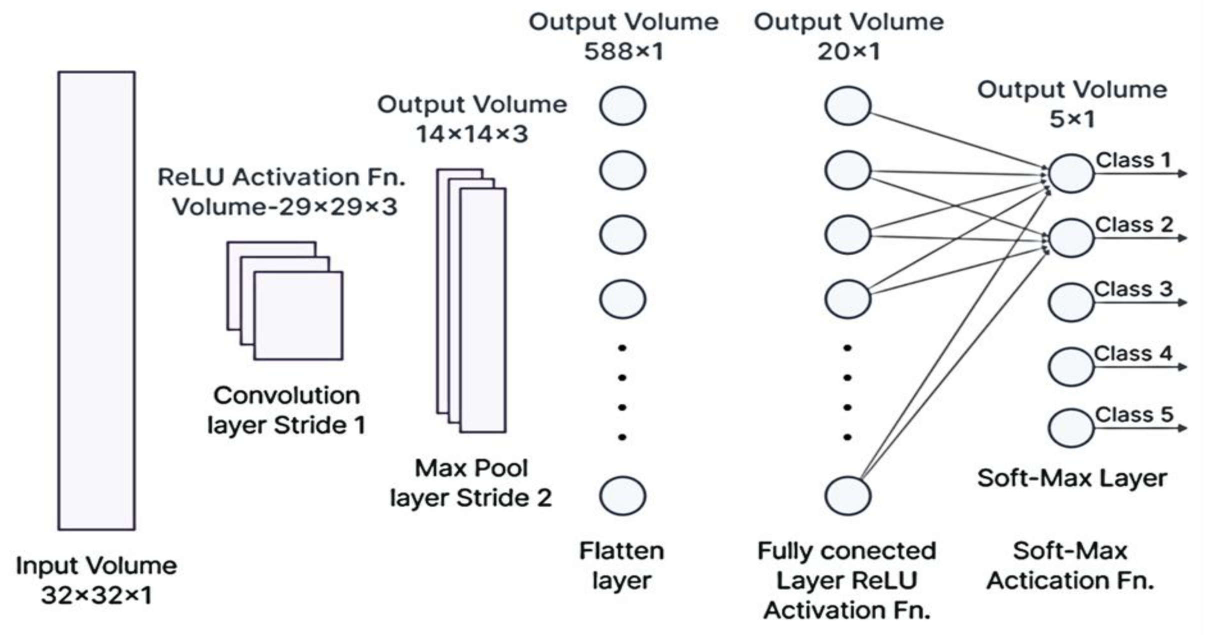
Step3: Pooling Layer

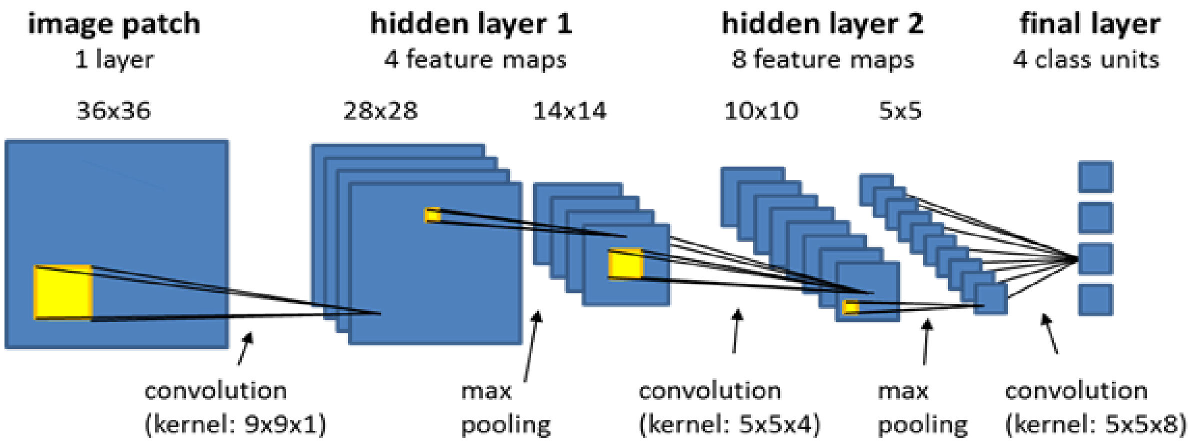
In the layer, we shrink the image stack into a smaller size. Pooling is done after passing by the activation layer. We do by implementing the following 4 steps:

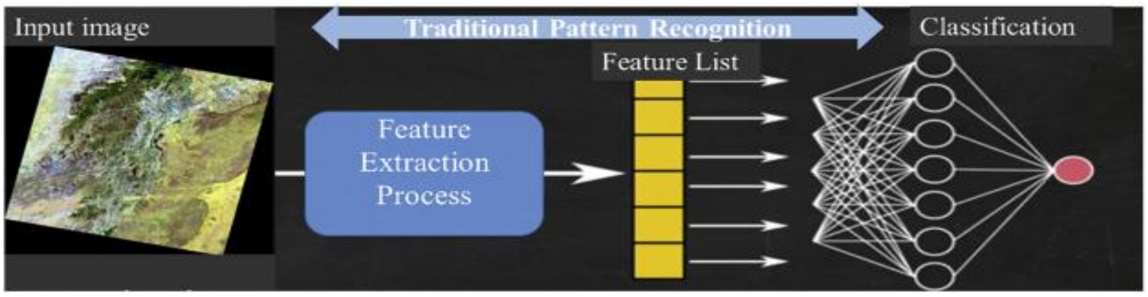
- Pick a window size (often 2 or 3)
- Pick a stride (usually 2)
- Walk your Window across your filtered images
- From each Window, take the maximum value

Step4: Fully Connected Layer

The last layer in the network is fully connected, meaning that neurons of preceding layers are connected to every neuron in subsequent layers. This mimics high-level reasoning where all possible pathways from the input to output are considered. Then, take the shrunk image and put into the single list, so we have got after passing through two layers of convolution relu and pooling and then converting it into a single file or a vector.







CNN Layer Working Diagram

5.5 Naïve Bayes

• Derivation:

• D: Set of tuples

• Each Tuple is an 'n' dimensional attribute vector

• X: (x1, x2, x3, …. xn)

• Let there be 'm' Classes: C1, C2, C3…Cm

• Naïve Bayes classifier predicts X belongs to Class Ci iff

• P (Ci/X) > P(Cj/X) for 1<= j <= m, j <> i Maximum Posteriori Hypothesis

• P(Ci/X) = P(X/Ci) P(Ci) / P(X)

• Maximize P(X/Ci) P(Ci) as P(X) is constant with many attributes, it is computationally expensive to evaluate P(X/Ci). Naïve Assumption of "class conditional independence"

• ∏= = n k P X Ci P xk Ci 1 (xk./ Ci)

• P(X/Ci) = P(x1/Ci) * P(x2/Ci) *…* P(xn/ Ci)

P(A|B) = Fraction of worlds in which B is true that also have A true

P(A ^ B) P(A|B) =------------------- P(B)

Corollary:

P(A ^ B) = P(A|B) P(B) P(A|B)+P(¬A|B) = 1

5.6 Information Gain

```
infoGain(examples, attribute, entropyOfSet)
gain = entropyOfSet
for value in attributeValues(examples, attribute):
sub = subset(examples, attribute, value)
gain -= (number in sub)/(total number of examples) * entropy(sub)
return gain
```

Entropy

```
entropy(examples)
```

CHAPTER 6

TESTING

6.1 Introduction

Testing is the way toward running a framework with the expectation of discovering blunders. Testing upgrades the uprightness of the framework by distinguishing the deviations in plans and blunders in the framework.

Testing targets distinguishing blunders – prom zones. This aides in the avoidance of mistakes in the framework. Testing additionally adds esteems to the item by affirming the client's necessity.

The primary intention is to distinguish blunders and mistake get-prom zones in a framework. Testing must be intensive and all around arranged. A somewhat tried framework is as terrible as an untested framework. Furthermore, the cost of an untested and under-tried framework is high.

The execution is the last and significant stage. It includes client preparation, framework testing so as to guarantee the effective running of the proposed framework. The client tests the framework and changes are made by their requirements. The testing includes the testing of the created framework utilizing different sorts of information. While testing, blunders are noted and rightness is the mode.

6.2 Objectives of Testing

• Testing in a cycle of executing a program with the expectation of discovering mistakes.

• An effective experiment is one that reveals an up 'til now unfamiliar blunder.

Framework testing is a phase of usage, which is pointed toward guaranteeing that the framework works accurately and productively according to the client's need before the live activity initiates. As expressed previously, testing is indispensable to the achievement of a framework. Framework testing makes the coherent presumption that if all the framework is right, the objective will be effectively accomplished.

A progression of tests is performed before the framework is prepared for the client acknowledgment test.

6.3 Testing Methods

System testing is a stage of implementation. This helps the weather system works accurately and efficiently before live operation commences. Testing is vital to the success of the system. The candidate system is subject to a variety of tests: online response, volume, stress, recovery, security, and usability tests series of tests are performed for the proposed system are ready for user acceptance testing.

6.3.1 Unit Testing

Unit testing chiefly centres around the littlest unit of programming plan. This is known as module testing. The modules are tried independently. The test is done during the programming stage itself. In this progression, every module is discovered to be working acceptably as respects the normal yield from the module.

6.3.2 Integration Testing

Mix testing is an efficient methodology for developing the program structure, while simultaneously leading tests to reveal blunders related with the interface. The goal is to take unit tried modules and manufacture a program structure. All the modules are joined and tried in general.

6.3.3 Performance Testing

Evaluate the performance of the system in terms of speed, accuracy, and resource utilization. Measure factors such as inference time, memory usage, and processing efficiency to ensure the system meets performance requirements.

6.3.4 Regression Testing

Repeatedly test the system after making changes or updates to ensure that new modifications do not introduce regressions or unexpected behaviour.

6.3.5 Output Testing

After validation testing, the next step is output testing of the proposed system, as it’s crucial for the system to produce the required output in a specific format. The on-screen output format was found to be correct, designed during system configuration according to user needs. The hard copy output also met the user’s predefined requirements. Therefore, no revisions were needed after output testing.

6.3.6 Chatbot Testing

Testing whether the chatbot is functional and can respond to natural languages and typography of humans and provide the right answer to the user queries.

6.3.7 User Acceptance Testing

Client acknowledgment of a framework is the vital factor for the achievement of any framework. The framework viable is tried for client acknowledgment by continually staying in contact with the imminent framework clients at the hour of creating and making changes at whatever point required.

6.4 Validation

Toward the consummation of the reconciliation testing, the product is totally amassed as bundle interfacing blunders have been revealed and adjusted and a last arrangement of programming tests starts in approval testing. Approval testing can be characterized from multiple points of view; however, a straightforward definition is that the approval succeeds when the product work in a way that is normal by the client. After approval test has been directed as follows:

• The capacity or execution qualities adjust to detail and are acknowledged.

• A deviation from the particular is revealed and a lack list is made.

• Proposed framework viable has been tried by utilizing an approval test and discovered to be working acceptably.

6.5 Test Reports

The users test the developed system when changes are made according to the needs. The testing phase involves the testing of the developed system using various kinds of data. An elaborate testing of data is prepared and system is tested using the test data. Test cases are used to check for outputs with different set of inputs.

6.6 Test Cases

1. **Test Case: Image Upload**

   Input: User selects an image of a plant leaf.
   Action: User uploads the image to the system.
   Expected Outcome: The system successfully receives the image and prepares it for analysis.

2. **Test Case: Image Processing**

   Input: Uploaded image of a plant leaf.
   Action: The system processes the image using OpenCV functions.
   Expected Outcome: The system successfully processes the image and extracts relevant features for disease detection.

3. **Test Case: Disease Detection**

   Input: Processed image of a plant leaf.
   Action: The system analyzes the image using the deep learning model.
   Expected Outcome: The system successfully identifies whether the leaf has a disease and, if so, classifies the type of disease.

4. **Test Case: Results Presentation**

   Input: Analysis results from the deep learning model.
   Action: The system presents the results to the user.
   Expected Outcome: The user successfully receives the results, including the disease status and type (if any).

5. **Test Case: Chatbot Interaction**

   Input: User types a question or command into the chatbot interface.
   Action: The chatbot processes the user's input and generates a response.
   Expected Outcome: The user receives a relevant and helpful response from the chatbot.

CHAPTER 7

CODES

7.1 backend code

```
import os
import tensorflow as tf
import numpy as np
from tensorflow.keras.preprocessing import image
from PIL import Image
import cv2
from keras.models import load_model
from flask import Flask, request, render_template
from werkzeug.utils import secure_filename
from tensorflow.keras.preprocessing.image import load_img, img_to_array
app = Flask(__name__)

model =load_model('model.keras')
print('Model loaded. Check http://127.0.0.1:5000/')

labels = {0: 'Dead', 1: 'Healthy', 2: 'NonLeaf', 3: 'Powdery', 4: 'Rust'}

def getResult(image_path):
    img = load_img(image_path, target_size=(225,225))
    x = img_to_array(img)
    x = x.astype('float32') / 255.
    x = np.expand_dims(x, axis=0)
    predictions = model.predict(x)[0]
    return predictions
@app.route('/', methods=['GET'])
def index():
    return render_template('index.html')


@app.route('/predict', methods=['GET', 'POST'])
def upload():
    if request.method == 'POST':
        f = request.files['file']


        basepath = os.path.dirname(__file__)
        file_path = os.path.join(
            basepath, 'uploads', secure_filename(f.filename))
        f.save(file_path)
        predictions=getResult(file_path)
        predicted_label = labels[np.argmax(predictions)]
        return str(predicted_label)
    return None

if __name__ == '__main__':
    app.run(debug=True)
```

7.2 Frontend code [main page]

```
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Plant Health Check Assistance</title>
  <link rel="preconnect" href="https://fonts.googleapis.com">
  <link rel="preconnect" href="https://fonts.gstatic.com" crossorigin>
  <link
href="https://fonts.googleapis.com/css2?family=Nunito:ital,wght@0,200..1000;1,200..10
00&display=swap"
    rel="stylesheet">
  <style>
    body {
      font-family: "Nunito", sans-serif;
      background-image: url(./2024-04-04_03-54-45_6532.png);
      background-color: #1e1f1ceb;
      background-repeat: no-repeat;
      background-attachment: fixed;
      background-size: 100%;
      background-position: 70%;
    }

    .hero {
      padding: 20px;
    }
```

```
h2 {
    color: #ffffff;
    font-size: 60px;
    text-align: center;
}

p {
    color: rgb(226, 233, 237);
    font-weight: 400;
    font-size: 30px;
}

.btn {

    justify-content: center;
    align-items: center;
    border-radius: 20px;
    font-size: 20px;
    margin: 10px;
    padding: 10px;
    background-color: #687D7A;
    border: 1px solid azure;

}

#plink {
    justify-content: center;
```

```
            align-items: center;

            text-decoration: none;

            color: #18181B

        }


        .bg {

            margin: 15px;

            padding: 8px;

            background-color: #18181b55;

            border-radius: 20px;

        }


        #quote {

            color: rgb(226, 233, 237);

            font-weight: 400;

            font-size: 25px;

        }

    </style>

</head>


<body>

    <div class="hero">

        <div class="bg">

            <h2>Welcom to plant Health Check Assistance</h2>

            <p>“Cultivating Growth, Harvesting Wellness: Unearth the Power of Plants with
Us!”</p>

                <div id="quote"></div>

            <button class="btn"><a id="plink" href="http://127.0.0.1:5000/"
target="_blank">Health Check</a></button>
```

```
        </div>
    </div>
    <script>
        var quotes = [
          "The glory of gardening: hands in the dirt, head in the sun, heart with nature.",
          "To nurture a garden is to feed not just the body, but the soul.",
          "Plant and your spouse plants with you; weed and you weed alone.",
          "The love of gardening is a seed once sown that never dies."
        ];

        function displayQuote() {
          var index = Math.floor(Math.random() * quotes.length);
          document.getElementById("quote").innerHTML = quotes[index];
        }

        // Display a quote every 10 seconds
        setInterval(displayQuote, 10000);
        </script>
        <script src="https://cdn.botpress.cloud/webchat/v1/inject.js"></script>
        <script src="https://mediafiles.botpress.cloud/ /webchat/config.js" defer></script>
</body>
</html>
```

7.3 MODEL TRAINING CODE

```
import os
def total_files(folder_path):
    num_files = len([f for f in os.listdir(folder_path) if
os.path.isfile(os.path.join(folder_path, f))])
    return num_files

train_files_healthy = "Dataset/Train/Healthy"
train_files_powdery = "Dataset/Train/Powdery"
train_files_rust = "Dataset/Train/Rust"
train_files_nonleaf = "Dataset/Train/NonLeaf"
train_files_dead = "Dataset/Train/Dead"

test_files_healthy = "Dataset/Test/Healthy"
test_files_powdery = "Dataset/Test/Powdery"
test_files_rust = "Dataset/Test/Rust"
test_files_nonleaf = "Dataset/Test/NonLeaf"
test_files_dead = "Dataset/Test/Dead"

valid_files_healthy = "Dataset/Validation/Healthy"
valid_files_powdery = "Dataset/Validation/Powdery"
valid_files_rust = "Dataset/Validation/Rust"
valid_files_nonleaf = "Dataset/Validation/NonLeaf"
valid_files_dead = "Dataset/Validation/Dead"

print("Number of healthy leaf images in training set", total_files(train_files_healthy))
print("Number of powder leaf images in training set", total_files(train_files_powdery))
print("Number of rusty leaf images in training set", total_files(train_files_rust))
print("Number of non-leaf images in training set", total_files(train_files_nonleaf))
print("Number of dead images in training set", total_files(train_files_dead))
print("=========================================================")
print("Number of healthy leaf images in test set", total_files(test_files_healthy))
print("Number of powder leaf images in test set", total_files(test_files_powdery))
print("Number of rusty leaf images in test set", total_files(test_files_rust))
print("Number of non-leaf images in test set", total_files(test_files_nonleaf))
print("Number of dead images in test set", total_files(test_files_dead))
print("=========================================================")
print("Number of healthy leaf images in validation set", total_files(valid_files_healthy))
print("Number of powder leaf images in validation set", total_files(valid_files_powdery))
print("Number of rusty leaf images in validation set", total_files(valid_files_rust))
print("Number of non-leaf images in validation set", total_files(valid_files_nonleaf))
print("Number of dead images in validation set", total_files(valid_files_dead))

from PIL import Image
import IPython.display as display
image_path = 'Dataset/Train/Healthy/8ce77048e12f3dd4.jpg'

with open (image_path, 'rb') as f:
    display.display(display.Image(data=f.read(), width=500))
image_path = 'Dataset/Train/Rust/80f09587dfc7988e.jpg'

with open(image_path, 'rb') as f:
    display.display(display.Image(data=f.read(), width=500))
image_path = 'Dataset/Train/Powdery/802f7439ec1ef0cd.jpg'

with open(image_path, 'rb') as f:
    display.display(display.Image(data=f.read(), width=500))
image_path = 'Dataset/Train/Dead/download (2).jpeg'

with open(image_path, 'rb') as f:
    display.display(display.Image(data=f.read(), width=500))
image_path = 'Dataset/Train/NonLeaf/download (1).jpeg'

with open(image_path, 'rb') as f:
    display.display(display.Image(data=f.read(), width=500))
from tensorflow.keras.preprocessing.image import ImageDataGenerator

train_datagen = ImageDataGenerator(
    rescale=1./255,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True)

test_datagen = ImageDataGenerator(rescale=1./255)
train_generator = train_datagen.flow_from_directory('Dataset/Train',
                                                    target_size=(225, 225),
                                                    batch_size=32,
                                                    class_mode='categorical')

validation_generator = test_datagen.flow_from_directory('Dataset/Validation',
                                                        target_size=(225, 225),
                                                        batch_size=32,
                                                        class_mode='categorical')
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D, Flatten, Dense

model = Sequential()
model.add(Conv2D(32, (3, 3), input_shape=(224, 224, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Flatten())
model.add(Dense(64, activation='relu'))
model.add(Dense(5, activation='softmax'))
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
history = model.fit(train_generator,
            batch_size=16,
            epochs=8,
            validation_data=validation_generator,
            validation_batch_size=16
            )

from matplotlib import pyplot as plt
from matplotlib.pyplot import figure

import seaborn as sns
sns.set_theme()
sns.set_context("poster")

figure(figsize=(25, 25), dpi=100)

plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'val'], loc='upper left')
plt.show()
model.save("model.keras")
from tensorflow.keras.preprocessing.image import load_img, img_to_array
import numpy as np

def preprocess_image(image_path, target_size=(225, 225)):
    img = load_img(image_path, target_size=target_size)
    x = img_to_array(img)
    x = x.astype('float32') / 255.
    x = np.expand_dims(x, axis=0)
    return x

x = preprocess_image('Dataset/Test/Rust/82f49a4a7b9585f1.jpg')
predictions = model.predict(x)
predicted_index = np.argmax(predictions)
labels = train_generator.class_indices
labels = {v: k for k, v in labels.items()}
labels
predicted_label = labels[predicted_index]
print(predicted_label)
```

CHAPTER 8

CONCLUSION AND FUTURE ENHANCEMENT

8.1 Conclusion

The “Plant Health Check Assistance” project is a sophisticated web-based application that employs Convolutional Neural Networks (CNN) for the prediction of plant diseases with a high accuracy of 96%. The system is architected to analyze leaf images from a diverse dataset and utilize historical data for pattern recognition in plant diseases.

A significant feature of this system is its extensive coverage of various plant leaves, providing farmers with a comprehensive database that can aid in crop selection. This is further enhanced by the integration of historical data production, offering insights into market trends and demand for specific plants.

In addition to these functionalities, the system incorporates an interactive chatbot. This chatbot serves as a digital assistant, capable of addressing user queries related to plant health, providing tips, and offering solutions for plant health issues. The integration of this chatbot enhances the user experience, making the system more accessible and user-friendly.

From a technical perspective, the project demonstrates the effective application of deep learning methodologies, image processing techniques, and natural language processing in a real-world context. It showcases the potential of integrating various technologies to create a comprehensive solution that addresses a critical issue - plant health.

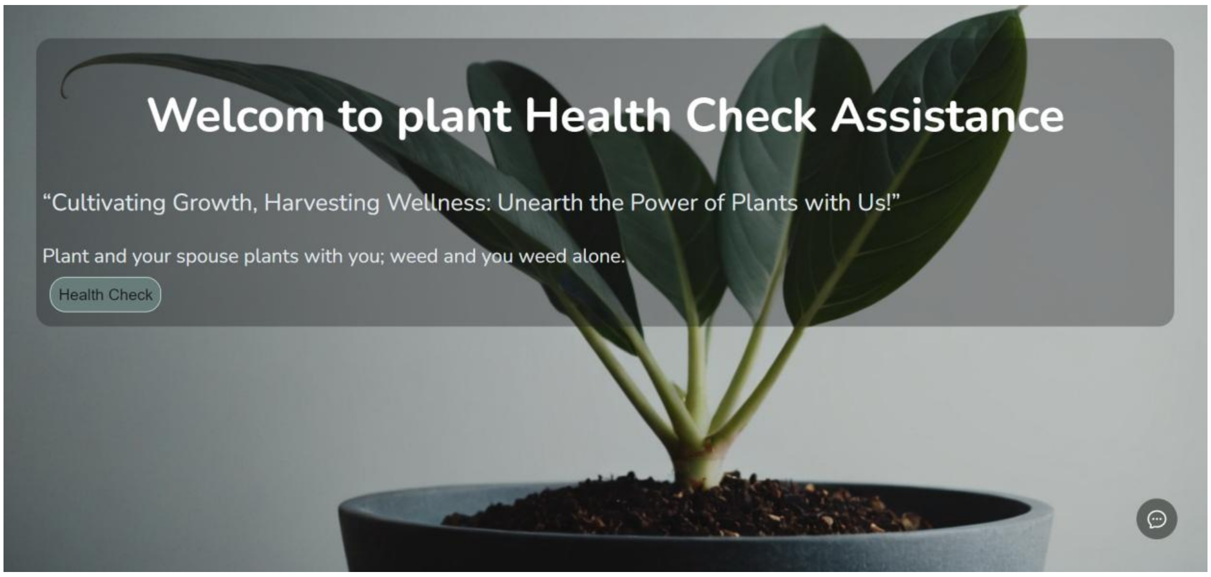
In conclusion, the “Plant Health Check Assistance” project is a testament to the potential of technological innovation in promoting sustainable and resilient farming practices. By accurately predicting plant diseases, it contributes to improved crop health and yield, thereby playing a crucial role in ensuring food security. The project, with its deep learning models, image processing techniques, and chatbot integration, stands as a significant contribution to the field of agricultural technology.

8.2 Future Enhancement

To improve recognition rate of final classification process hybrid algorithms like Artificial Neural Network, Bayes classifier, Fuzzy Logic can also be used.

- Mobile application can be developed which is handy and easy to use.
- An extension of this work will focus on automatically estimating the severity of the detected disease.
- As future enhancement of the project chat bot to develop the open multimedia (Audio/Video) about the diseases and their solution automatically once the disease is detected.
- Improvement in to provide audio guidance and video remedies and Tips.

SCREENSHOTS

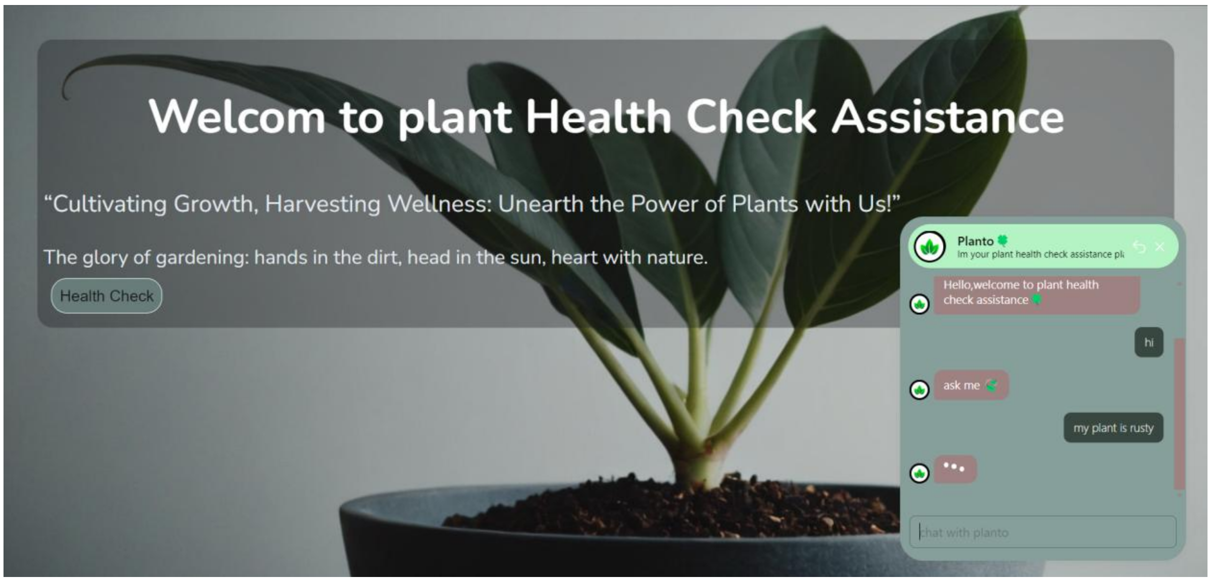


Welcom to plant Health Check Assistance

"Cultivating Growth, Harvesting Wellness: Unearth the Power of Plants with Us!"

Plant and your spouse plants with you; weed and you weed alone.

Image of Home Page



Welcom to plant Health Check Assistance

"Cultivating Growth, Harvesting Wellness: Unearth the Power of Plants with Us!"

The glory of gardening: hands in the dirt, head in the sun, heart with nature.

Im your plant health check assistance pl

Hello,welcome to plant health check assistance

hi

ask me

my plant is rusty

Image of accessing Chatbot in Home Page

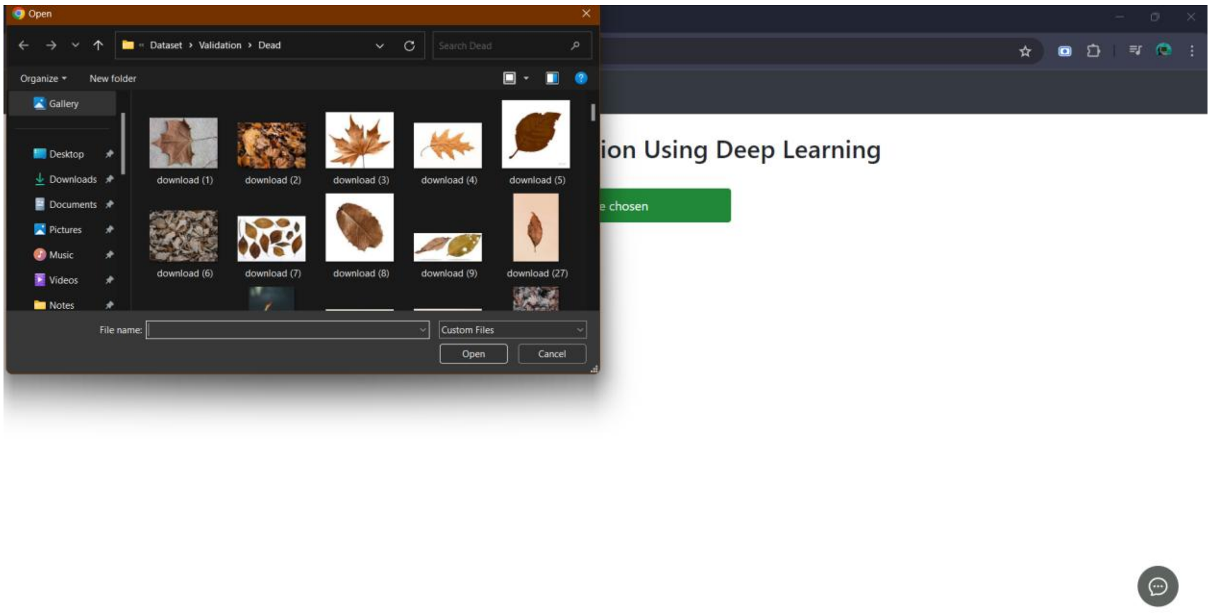
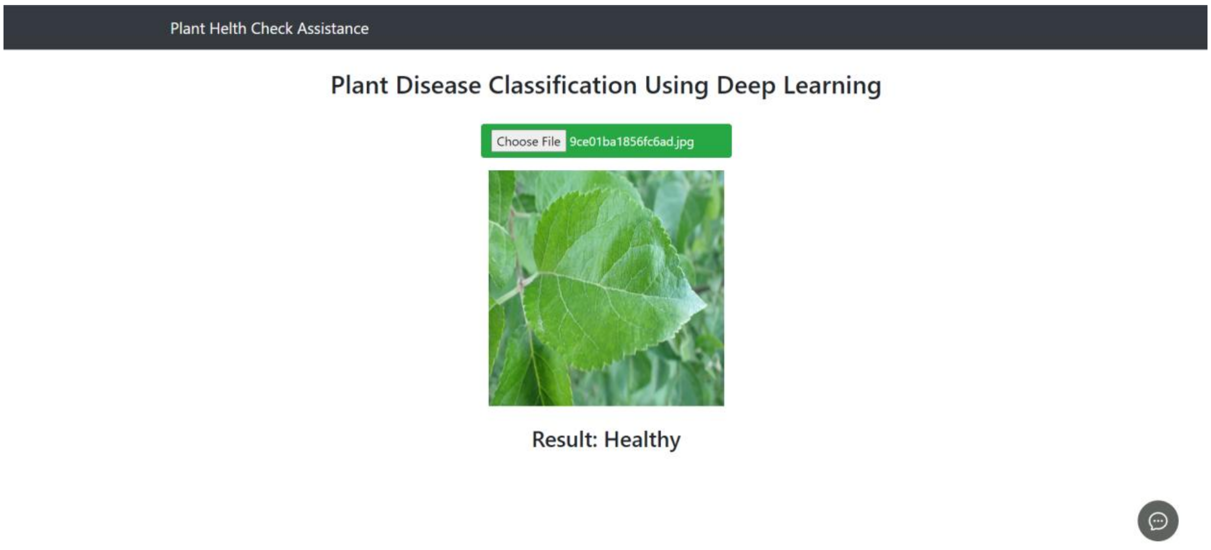
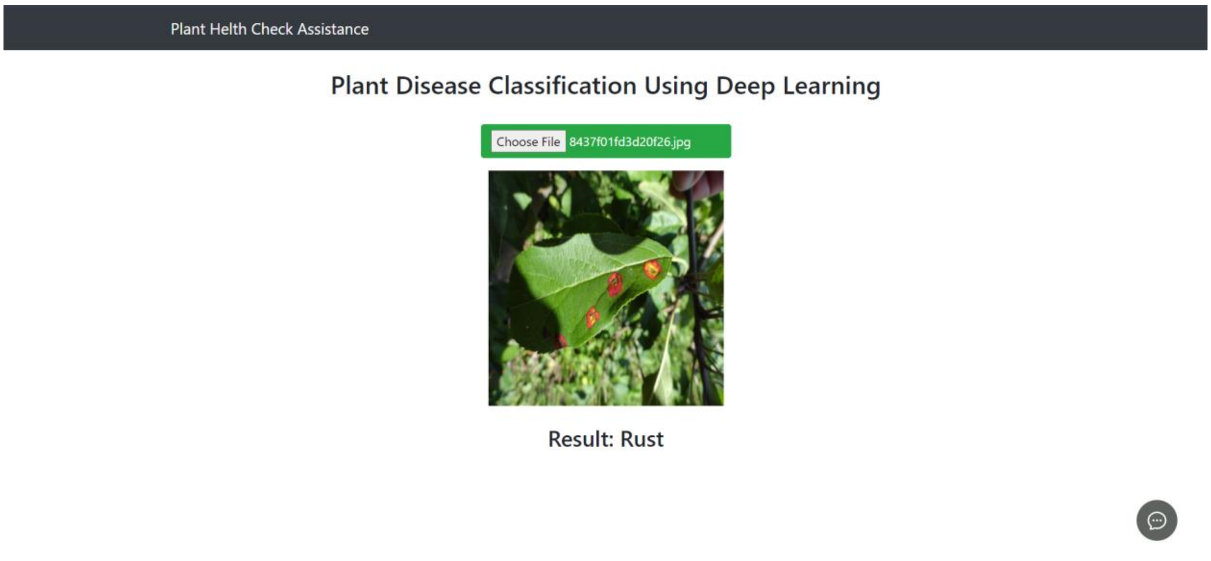


Image of choosing media for getting an output


Plant Helth Check Assistance
Plant Disease Classification Using Deep Learning
Choose File
9ce01ba1856fc6ad.jpg

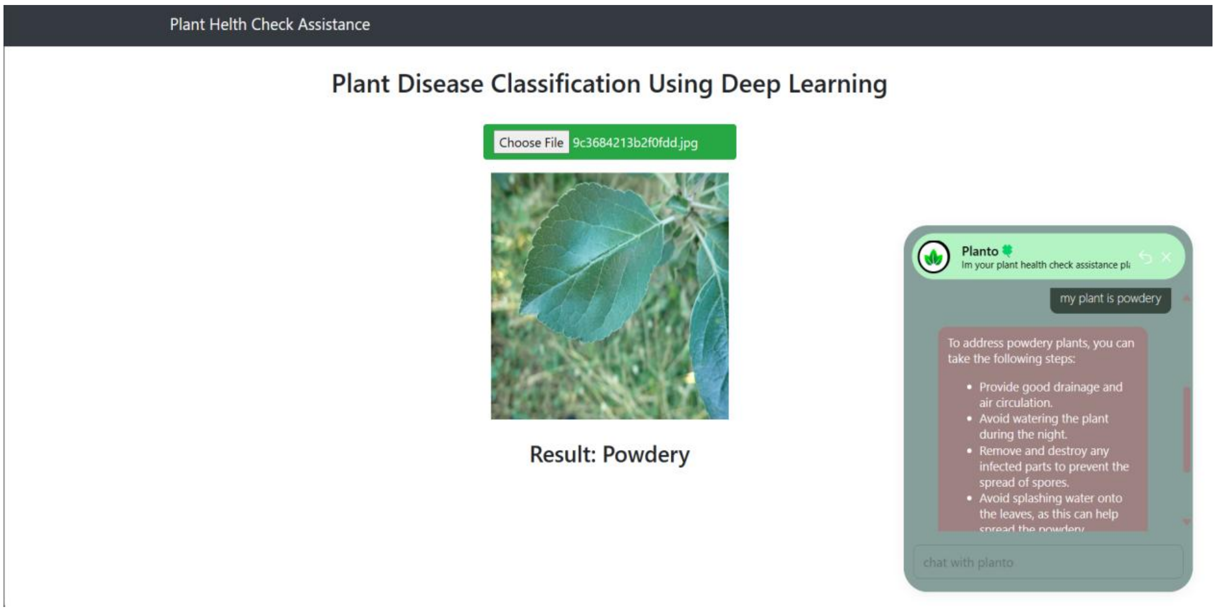
Result: Healthy

Image of an output classifying leave is Healthy


Plant Helth Check Assistance
Plant Disease Classification Using Deep Learning
Choose File 8437f01fd3d20f26.jpg

Result: Rust

Image of an output classifying leave is Rusty


Plant Helth Check Assistance
Plant Disease Classification Using Deep Learning
Choose File 9c3684213b2f0fdd.jpg
Planto
Im your plant health check assistance pl
chat with planto

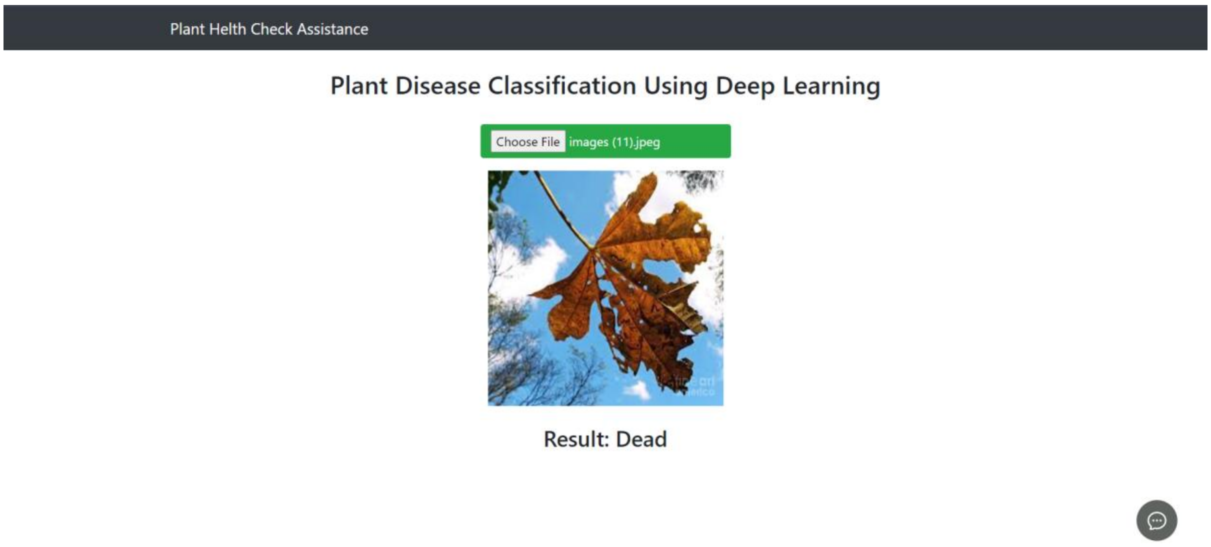
Result: Powdery

my plant is powdery

To address powdery plants, you can take the following steps:

- Provide good drainage and air circulation.
- Avoid watering the plant during the night.
- Remove and destroy any infected parts to prevent the spread of spores.
- Avoid splashing water onto the leaves, as this can help spread the powdery

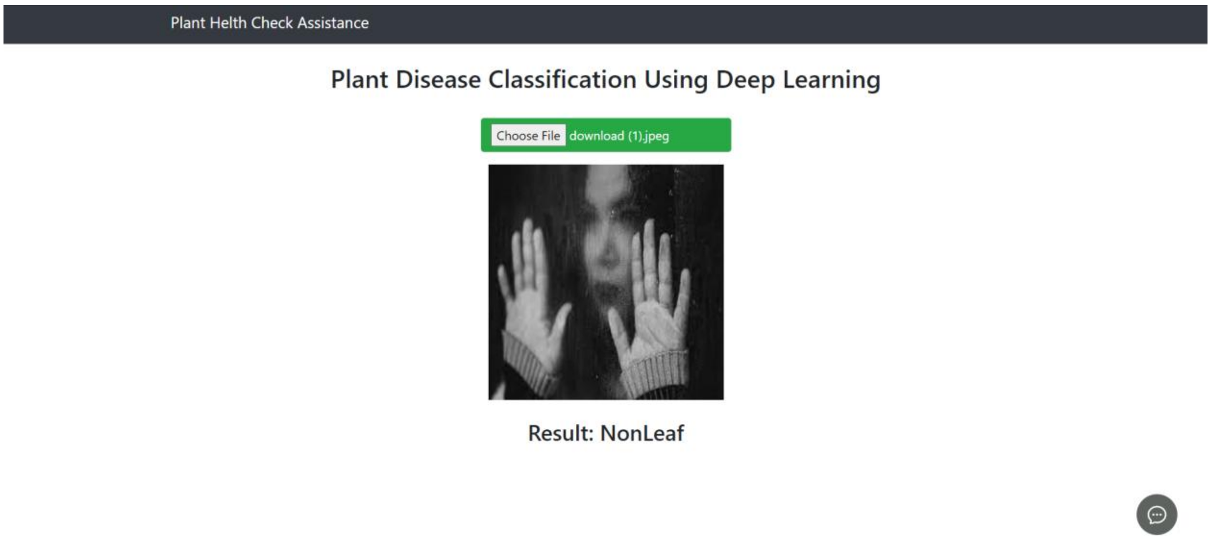
Image of an output classifying leave is Powdery


Plant Helth Check Assistance

Plant Disease Classification Using Deep Learning

Result: Dead

Image of an output classifying leave is Dead


Plant Helth Check Assistance

Plant Disease Classification Using Deep Learning

Result: NonLeaf

Image of an output classifying input image is Not a Leafe

BIBLIOGRAPHY

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. doi:10.3389/fpls.2016.01419

2. 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, 3289801. doi:10.1155/2016/3289801

3. Ferentinos, K. P. (2018). Deep learning models for plant disease detection and diagnosis. Computers and Electronics in Agriculture, 145, 311-318. doi:10.1016/j.compag.2018.01.009

4. Brahimi, M., Boukhalfa, K., & Moussaoui, A. (2017). Deep learning for tomato diseases: Classification and symptoms visualization. Applied Artificial Intelligence, 31(4), 299-315. doi:10.1080/08839514.2017.1315516

5. Li, Y., Zhang, J., & Yang, L. (2020). Plant disease detection and classification by deep learning—A review. IEEE Access, 8, 40926-40935. doi:10.1109/ACCESS.2020.2971044

6. Too, E. C., Yujian, L., Njuki, S., & Yingchun, L. (2019). A comparative study of fine-tuning deep learning models for plant disease identification. Computers and Electronics in Agriculture, 161, 272-279. doi:10.1016/j.compag.2018.03.032

7. Botpress | the Generative AI platform for ChatGPT Chatbots

8. Kaggle Dataset- https://www.kaggle.com/datasets/rashikrahmanpritom/plant-disease recognition-dataset

9. GitHub reference- https://github.com/Chando0185/plant_disease_detection