RV College of Engineering®
Mysore Road, RV Vidyaniketan Post,
Bengaluru - 560059, Karnataka, India

Go, change the world®

DEPARTMENT OF ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING

Project Report

On

Plant Disease Detection: A Vision Based Solution

Submitted in partial fulfilment of the requirements for the V Semester
ARTIFICIAL NEURAL NETWORK AND DEEP LEARNING
AI253IA

By

1RV22AI005	**Akshita Chavan**
1RV22AI008	**Ankush Arunkumar**
1RV22AI038	**Pavithra C**

Department of Artificial Intelligence and Machine Learning
RV College of Engineering®
Bengaluru – 560059

Academic year
2024-25

RV COLLEGE OF ENGINEERING®

(Autonomous Institution Affiliated to Visvesvaraya Technological University, Belagavi)

DEPARTMENT OF ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING

Bengaluru– 560059

CERTIFICATE

This is to certify that the project entitled "**Plant Disease Detection: A Vision Based Solution"** submitted in partial fulfillment of Artificial Neural Networks and Deep Learning (21AI63) of V Semester BE is a result of the bonafide work carried out by Akshita Chavan (1RV22AI005), Ankush Arunkumar (1RV22AI008) and Pavithra C (1RV22AI038) during the Academic year 2024-25

Faculty In charge

Date :

Head of the Department

Date :

RV COLLEGE OF ENGINEERING®

(Autonomous Institution Affiliated to Visvesvaraya Technological University, Belagavi)

DEPARTMENT OF ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING

Bengaluru– 560059

DECLARATION

We, Akshita Chavan (1RV22AI005), Ankush Arunkumar (1RV22AI008) and Pavithra C (1RV22AI038), students of Fifth Semester BE hereby declare that the Project titled **"Plant Disease Detection: A Vision Based Solution**" has been carried out and completed successfully by us and is our original work.

Date of Submission: **Signature of the Student**

ACKNOWLEDGEMENT

We are profoundly grateful to our guide, **Dr. Somesh Nandi,** Assistant Professor, RV College of Engineering, for his wholehearted support, valuable suggestions, and invaluable advice throughout the duration of our project. His guidance and encouragement were instrumental not only in the successful completion of the project but also in the preparation of this report.We also extend our special thanks to **Dr. Anupama Kumar** for her invaluable insights, support, and constructive feedback, which significantly contributed to the improvement of our work.

We would like to express our sincere thanks to our Head of the Department, **Dr. Satish Babu,** for his constant encouragement and for fostering an environment of innovation and learning that greatly aided our progress.

We extend our heartfelt gratitude to our beloved Principal, **Dr. K. N. Subramanya,** for his unwavering appreciation and support for this Experiential Learning Project, which motivated us to give our best.

Lastly, we take this opportunity to thank our family members and friends for their unconditional support and encouragement throughout the project. Their backup and motivation were crucial in helping us overcome challenges and successfully complete our work.

ABSTRACT

The agricultural sector plays a critical role in the global economy, yet farmers face persistent challenges in managing crop health, particularly with the spread of plant diseases. These diseases, if undetected, can lead to substantial losses, affecting food production and economic stability. Current methods of plant disease diagnosis often rely on manual inspection, which can be time-consuming and error-prone. To address these issues, this project presents an innovative solution by leveraging Convolutional Neural Networks (CNNs) to automate the detection of diseases such as Apple Scab, Potato Early Blight, and Corn Northern Leaf Blight in their respective crops. By analyzing leaf images, the system is able to classify plant diseases and provide actionable insights, ultimately helping farmers make informed decisions about crop management.

For this project, a pre-trained ResNet50 model is fine-tuned to recognize various plant diseases in crops such as apples,potatoes ,corn and tomatoes . The dataset, sourced from Mendeley Data, contains labeled images of healthy and diseased plant leaves and includes diseases such as Apple Scab, Potato Early Blight, and Corn Northern Leaf Blight. To improve model performance and generalize better to unseen data, image augmentation techniques such as rotation, flipping, and scaling are applied during training. The model is built using TensorFlow and Keras, and MLFlow is integrated into the workflow to track experiments and monitor the model's performance across different stages of training. The system outputs predictions of disease types, enabling farmers to quickly identify issues with their crops. An interactive user interface is built using streamlite for better visualization.

The results of this project demonstrate the effectiveness of the CNN model in accurately classifying plant leaf diseases, establishing it as a valuable tool for early detection and timely intervention. The project is successfully implemented with an accuracy of 97.5% for the given data set. With real-time predictions, farmers can take immediate action to manage and mitigate the effects of plant diseases, potentially saving crops and reducing the need for excessive pesticide use. The project not only demonstrates the power of deep learning in solving real-world agricultural problems but also provides a platform for future advancements, such as expanding the disease database, enhancing the model's accuracy, and incorporating remedy suggestions for each identified disease. Ultimately, this system has the potential to improve agricultural productivity and sustainability, supporting farmers in maintaining healthy crops while minimizing resource wastage.

Table of Contents

List of Figures

Chapter 1: Introduction

This chapter gives the description of the project Plant DIsease Detection .It also includes theory and concepts used followed by report organization.

1.1 Project Description

Agriculture is the backbone of the global economy, employing nearly 40% of the workforce and contributing over $10 trillion annually [7]. However, plant diseases continue to threaten food security, causing up to 40% of crop losses globally and resulting in annual economic damage of more than $220 billion [3]. These challenges are particularly severe in developing countries, where smallholder farmers, who produce 80% of the world's food, struggle with limited access to disease management resources [6].

Timely detection and management of plant diseases can reduce yield losses by as much as 30%, yet traditional methods are slow, costly, and reliant on experts [10]. To overcome this, our project employs state-of-the-art computer vision techniques, specifically the ResNet-50 architecture, known for its high accuracy in image classification tasks. Trained on the diverse PlantVillage dataset, containing over 50,000 annotated leaf images, this system ensures precise identification of diseases like early blight, rust, and bacterial spots [4].

By enabling farmers to upload leaf images and receive real-time diagnoses along with actionable remedies, this approach not only reduces reliance on pesticides by up to 20% but also promotes sustainable practices [9]. Studies predict that integrating AI-driven tools in agriculture could boost global food production by 70% by 2050, meeting the needs of an expanding population [2]. This project exemplifies the transformative potential of AI in agriculture, inspiring innovative solutions to enhance food security and sustainable farming [8]

Theory and concept

1. Computer Vision and Image Recognition

Computer vision is a field of artificial intelligence (AI) that enables computers to interpret and understand visual information from the world, such as images or videos. In the context of plant disease detection, computer vision techniques are employed to analyze images of plant leaves to detect signs of diseases. These techniques involve extracting meaningful features from images, such as texture, color,

shape, and edges, which can help distinguish between healthy and diseased plants. Common image processing methods used include edge detection, segmentation, and feature extraction, which aid in detecting anomalies caused by diseases.

2. Deep Learning and Convolutional Neural Networks (CNNs)

Deep learning, a subset of machine learning, involves the use of neural networks with many layers (hence "deep") to model complex patterns in large datasets. Convolutional Neural Networks (CNNs) are a class of deep learning algorithms particularly well-suited for image analysis. CNNs work by applying convolutional layers that automatically detect relevant features in images, such as textures and shapes, which are crucial for understanding visual patterns. CNNs have proven to be highly effective in image recognition tasks, including identifying diseases in plants. In this project, the CNN architecture, specifically ResNet50 (a deep CNN pre-trained on ImageNet), will be employed to detect plant diseases based on input leaf images.

3. Image Preprocessing and Augmentation

Before feeding images into a model, it is essential to preprocess them to standardize their size and normalize their pixel values. In this project, images are resized to a consistent dimension (224x224 pixels) to match the input size of the CNN model (ResNet50). Image augmentation, on the other hand, is a technique used to artificially expand the training dataset by applying random transformations to the images, such as rotation, flipping, and scaling. This helps improve the robustness of the model by exposing it to a variety of image variations, thus preventing overfitting and enhancing the model's generalization capability.

4. Transfer Learning

Transfer learning is a technique in machine learning where a model developed for one task is reused for another related task. In this project, a pre-trained ResNet50 model (trained on ImageNet) will be fine-tuned for the task of plant disease detection. By using a pre-trained model, the project leverages the feature extraction capabilities learned from a massive dataset like ImageNet, which can be adapted to the plant disease dataset with relatively less data and computational cost. Fine-tuning the model involves adjusting the last few layers of the network to adapt it to the specific task of classifying plant diseases, while keeping the earlier layers intact for efficient feature extraction.

5. Categorical Cross-Entropy Loss Function

Categorical cross-entropy is used as the loss function in classification tasks, like plant disease detection,where the goal is to assign images to multiple categories. It calculates the difference between

predicted and actual labels, guiding the model to minimize this discrepancy for improved classification accuracy.

6. Model Evaluation Metrics

Evaluation metrics, such as accuracy,validation_accuracy, loss and validation _loss , are used to assess model performance. Accuracy measures the proportion of correct classifications, while precision, recall, and F1-score are essential when dealing with imbalanced classes. The formulas are given below

$$1.1)\ Accuracy\ =\ \frac{Number\ of\ Correct\ Predictions}{Total\ Number\ of\ Predictions}\times 100$$

$$1.2)\ Validation\ Accuracy\ =\ \frac{Number\ of\ Correct\ Predictions\ on\ validation\ data}{Total\ Validation\ Data\ Samples}\times 100$$

$$1.3)\ Loss\ =\ \frac{1}{N}\sum_{i=1}^{N} Loss\ per\ sample$$

7. MLFlow and Experiment Tracking

MLFlow is used to track and manage the machine learning lifecycle, logging experiments, parameters, and model metrics. This allows for better organization, version control, and easy comparison of different model versions, ensuring reproducibility and efficient model management.

8. Model Deployment and Integration

After training, the model is deployed via an interactive web or mobile application, allowing farmers to upload images of their plants for disease prediction. The application provides an easy-to-use interface for real-time predictions, integrating the model into a practical tool for farmers to access disease detection services.

9. Real-Time Prediction and Farmer Empowerment

The system enables real-time plant disease detection, reducing the reliance on expert knowledge and manual inspection. By providing accurate diagnoses and treatment suggestions, the system empowers farmers to take quick, informed actions, helping improve crop health, reduce pesticide use, and increase yield.

1.2 Report Organization

The report is structured to provide a comprehensive understanding of the plant disease detection project. It begins with the **Introduction**, offering an overview of the project's background, significance, and objectives in tackling the challenges of plant disease detection using advanced technologies like computer vision and machine learning. The **Project Description** elaborates on the scope, employed methodologies, and anticipated outcomes, setting the stage for the rest of the report.

The **Report Organization** section guides readers through the report's layout, ensuring clarity and logical progression. Following this, the **Literature Review** delves into previous research, current systems, and the proposed innovations, detailing the tools and technologies used, along with hardware and software requirements. The **Software Requirement Specifications** section describes the software's functional and non-functional requirements, external interfaces, and design constraints, offering a clear understanding of the system's capabilities and limitations.

Next, the **System Design** segment includes the architectural design, data flow diagrams, and a detailed description of the algorithms and neural networks employed, particularly focusing on the deep learning architecture used. The **Implementation** section showcases key code snippets and discusses the results with supporting screenshots, highlighting the system's effectiveness and accuracy.

The report concludes with a **Conclusion** summarizing the project's achievements and its impact on agriculture. The **Future Enhancements** section suggests potential improvements and future directions, followed by the **References** listing all cited sources, ensuring the report's comprehensiveness and scholarly rigor.

This chapter provides an overview of the plant disease detection project, highlighting its importance in agriculture and the use of advanced technologies like computer vision and deep learning for accurate disease diagnosis. It also outlines the theory and concepts that underpin the project, setting the stage for detailed discussions in later chapters.

Chapter 2: Literature Survey

This chapter presents a literature survey on plant disease detection, summarizing various machine learning, deep learning, and computer vision techniques used across multiple studies to enhance early disease detection, classification accuracy, and real-time application in agriculture.

2.1 Literature Survey

The authors M. Kumar and I. Sethi of paper [1] introduces a hybrid model combining Convolutional Neural Networks (CNN) and Support Vector Machines (SVM) to detect rice spot disease and assess its severity. The model, trained on an extensive rice spot dataset, achieved an impressive accuracy rate of 95.59%. This system offers a practical solution for farmers to improve crop yield by enabling early disease detection and precise disease management.In [2] R. Sathya and et al utilizes supervised machine learning to recognize diseases in brinjal leaves, such as Cercospora solani and Tobacco Mosaic Virus. Using feature extraction methods like LIV, PCA, and GLCM, the model achieved an accuracy of 98.48% with SVM RBF. This high-performing system provides an effective solution for detecting and diagnosing brinjal leaf diseases with high precision.Meanwhile in [3] Rutuja Rajendra Patil and et al proposed the integration of IoT-based systems equipped with environmental sensors to track parameters like humidity, temperature, and soil moisture for early disease detection. Combining IoT with AI enhances the precision of disease forecasting, improving yield prediction and reducing pesticide overuse in agriculture.In [4] Emmanuel Moupojou and et al describes many deep learning models trained on lab datasets, like PlantVillage, fail under field conditions due to complex backgrounds.

The FieldPlant dataset addresses this gap by using annotated field images, leading to improved disease classification under real-world scenarios. Models like MobileNetV2 and YOLO have shown high accuracy in plant disease detection, making them effective solutions for real-time applications.In [5] Anupam Bonkra and et al explores collaborative ML/DL approaches, including CNNs and hybrid techniques, for classifying plant diseases like scab, rust, and black rot in apple leaves. Key pre-processing steps, such as segmentation and feature extraction, are crucial for improving the model's accuracy.

In [6] Hassan Mustafa and et al emphasizes the importance of early plant disease detection for improving agricultural productivity. It proposes a five-layer CNN model trained on 20,000 augmented images to automatically identify diseases in plant leaves. The optimized model achieves 99.99%

accuracy in distinguishing healthy from bacterial-infected pepper bell leaves, showcasing its potential as a real-time disease detection tool in agriculture.The authors in [7] Deepkiran Munjal and et al reviews the use of ML, DL, and computer vision techniques for early plant disease detection, outlining their advantages over traditional methods. It discusses state-of-the-art approaches, highlighting their strengths, limitations, and their potential to revolutionize disease management in agriculture.The authors R. K. Lakshmi and N. Savarimuthu in paper [8], the research presents a transfer learning-based, optimized EfficientDet deep learning framework for plant disease detection.

The model, evaluated using mean average precision (mAP), achieved an mAP of 74.10%. This system offers a practical solution for automated disease detection, delivering quick results with fewer computational resources compared to other models, making it ideal for deployment on handheld devices.In [9] J. Madake and et al focuses on detecting wilted plants using image processing and feature extraction techniques, such as ORB and SIFT. Various classifiers like Random Forest, AdaBoost, Decision Tree, and Logistic Regression were used, with the Random Forest classifier yielding an accuracy of 85.41%. The system provides insights into plant health, helping in the detection of indoor plant watering needs.Meanwhile in [10],authors B.Nageswara Rao Naik adn et al focus on detecting and classifying five different chilli leaf diseases using image-based classification. The study compares the performance of machine learning and deep learning models, testing twelve pretrained networks. DarkNet53 achieved the highest accuracy of 99.12% with augmentation, while a novel squeeze-and-excitation CNN (SECNN) model further enhanced the accuracy to 99.28% across 43 plant leaf classes, including chilli and other datasets.

This highlights the potential of SECNN for plant disease detection.Here’s the content description for the papers you mentioned with the respective numbers.In [11] J. Arun Pandian and et al introduces a 14-layer deep convolutional neural network (DCNN) that achieves 99.97% accuracy for detecting plant diseases across 58 plant leaf classes. Data augmentation techniques like BIM, GANs, and NST are used to enhance the diversity of the dataset, improving the model’s robustness.

High-performance GPUs and TensorFlow were utilized for efficient training and testing, ensuring the scalability of the system for large-scale applications.In paper [12] ,A. A. Alatawi and et al leverages the VGG-16 model to classify plant diseases in a dataset of 15,915 plant leaf images (both healthy and diseased) from the PlantVillage dataset, achieving 95.2% accuracy. The model uses CNN for efficient disease classification across 19 plant disease classes, enabling timely interventions for disease management. With a testing loss of 0.4418, the study demonstrates the model’s scalability for agricultural disease management applications.The authors K. L. R and N. Savarimuthu of paper [13] investigates the use of computer vision-based object detection methods, such as YOLOv4, EfficientDet,

and Scaled-YOLOv4, for early plant disease detection. The study utilizes the PlantVillage dataset and highlights the effectiveness of Scaled-YOLOv4 in detecting small infected areas in real-time.

This method offers a quick and efficient solution for early diagnosis, which is crucial for reducing crop losses and ensuring better disease management.In [14] Prakhar Bansal and et al proposes a model for classifying diseases in apple leaves using an ensemble of pre-trained deep learning models. The proposed model outperforms previous models, achieving an accuracy of 96.25%. Deep learning techniques, particularly convolutional neural networks (CNNs), are found to be particularly effective in image classification .

Sankaranarayanan Nalini of paper [15] includes preprocessing images using k-means clustering for disease region segmentation, and extracting color, shape, and texture features. A novel deep learning approach with optimized weights and biases is employed, significantly outperforming traditional support vector machine (SVM) algorithms. The model achieves an impressive 96.96% accuracy, providing a robust solution for disease classification in plant leaves.In [16] Saumya Yadav and et al develops a convolutional neural network (CNN) model for detecting bacteriosis in peach crops using leaf images. Trained on augmented datasets, the model achieved 98.75% accuracy, processing images in just 0.185 seconds per image. The model performs well on both field and laboratory images, offering potential as an early warning tool for real-world farming conditions. It also holds promise for integration with unmanned aerial vehicles (UAVs) for practical field applications.In [17] R. Biswas and et al introduces a mobile or web application powered by deep learning and computer vision for the automated identification of plant diseases.

The system targets five plant diseases, offering users insights on treatment and prevention, improving accessibility to timely agricultural interventions.In [18] Usama Mokhtar and et al focuses on detecting tomato leaf health using image processing techniques with GLCM (Gray Level Co-occurrence Matrix) for feature extraction and SVM for classification. The model achieved an accuracy of 99.83% using 800 images and N-fold cross-validation with a linear kernel. This highly accurate system offers a reliable tool for early detection of tomato leaf diseases, which is crucial for improving crop yield and health.Meanwhile, in [19] A. S. M. Farhan Al Haque and et al focus on developing an automated system using CNN to detect diseases in guava crops, specifically anthracnose, fruit rot, and fruit canker. Using images of diseased and healthy guava, the model achieved a high accuracy of 95.61%.

This system serves as an early detection tool that helps mitigate economic losses in guava production by providing actionable insights into potential disease management and curative actions for farmers.In [20] Muhammad Hammad Saleem and et al discusses how advancements in deep learning (DL) have

revolutionized plant disease detection and classification, surpassing traditional machine learning techniques. It explores the performance of DL models, visualization techniques, and metrics, while identifying gaps that need to be addressed for enhancing early, symptom-free disease detection in plants.

Therefore the reviewed studies demonstrate significant progress in leveraging ML, DL, and IoT technologies for efficient plant disease detection and management. These advancements hold great potential for improving crop yield, reducing economic losses, and fostering sustainable farming practices.

2.2 Summary of the literature survey:

The following are the observations from the literature survey:

- **Deep Learning (DL) Dominance:** Techniques like CNNs, VGG-16, and transfer learning models (EfficientDet, YOLO, etc.) are widely used for plant disease detection due to high accuracy and scalability (Summaries 1, 13, 14, 17).[1,13,14,17]
- **Machine Learning (ML) & Hybrid Approaches:** Some papers combine traditional ML with DL for improved performance, emphasizing feature extraction and SVM-based classification (Summaries 2, 6, 9).
- **Image Processing Techniques:** Preprocessing, feature extraction (GLCM, ORB, SIFT), and augmentation play crucial roles in boosting model performance (Summaries 7, 18).
- **High Accuracy Models:** Many studies report models achieving 95–99% accuracy, demonstrating potential for real-world deployment .
- **Dataset Importance:** Augmentation and diverse datasets (e.g., PlantVillage, FieldPlant) enhance model robustness for real-world conditions .

Identified Gaps:

- **Data Challenges:** Models often underperform in certain scenarios due to complex backgrounds and environmental noise.
- **Early Detection Focus:** Develop models capable of detecting diseases before visible symptoms appear.
- **Remedy Suggestion:** Provide remedies for the identified diseases along with the prediction of the disease.
- **Need for Comparative Analysis of Deep Learning Architectures:**Existing studies lack comparative analyses of deep learning architectures for plant disease detection.

Objectives

1. Implement a system that automatically identifies plant diseases from images without manual intervention.
2.Enable early identification of diseases to minimize crop damage and improve yields.

3. Provide actionable treatment suggestions tailored to the detected disease.
4. Ensure the model delivers reliable and precise predictions for effective decision-making.

2.3 Existing and Proposed system

Existing System:

1. **Problem Statement:**

   Traditional methods of plant disease detection are slow, prone to human error, and often ineffective in real-world field conditions due to complex backgrounds. Many deep learning models, such as those trained on lab datasets like PlantVillage, struggle to generalize to field scenarios, limiting their practical applicability. Early detection of plant diseases, which is crucial for reducing crop losses and economic impacts, remains inadequately addressed by existing systems. Furthermore, the lack of tools combining real-time detection, scalability, and accessibility for farmers hinders effective disease management. To address these challenges, the proposed system aims to develop a robust, scalable, and efficient automated solution for early detection and classification of plant diseases across diverse crops and environments. By leveraging datasets like PlantVillage and FieldPlant, augmented using techniques such as GANs, NST, BIM, and k-means clustering, the system enhances data diversity and addresses imbalances. Advanced feature extraction methods, including GLCM, ORB, and SIFT, are utilized alongside segmentation techniques like k-means clustering to isolate disease regions effectively. The proposed system adopts state-of-the-art models like EfficientDet, Scaled-YOLOv4, and ensemble CNNs, optimized through hyperparameter tuning and transfer learning, to achieve high accuracy and efficiency. Evaluation metrics such as accuracy, testing loss, and mean average precision (mAP) ensure model performance, with real-time detection capabilities and processing times as low as 0.185 seconds per image. The system is designed to be scalable and adaptable to field conditions, addressing challenges like complex backgrounds and small infection areas while supporting diverse plant types and diseases, including rice spot, guava anthracnose, and tomato leaf health. Accessibility is enhanced through mobile and web applications, providing farmers with real-time diagnostics, accuracy scores, and bounding boxes

for detected diseases. Compatibility with handheld devices and UAVs ensures practical field use. Additionally, the system integrates innovative data augmentation techniques, advanced CNN architectures, and novel feature extraction methods to optimize performance. Overall, the proposed solution offers a comprehensive framework for accurate, real-time, and accessible plant disease detection and management, effectively addressing the needs of farmers and stakeholders.

Proposed System:

Problem Statement and Scope of the Project

The goal of this project is to develop a robust plant disease detection system capable of identifying diseases in major crops like apple, corn, potato, and tomato based on leaf images. By leveraging deep learning and computer vision, this system aims to assist farmers and agricultural professionals in early disease diagnosis, thereby preventing crop loss and ensuring better yields. The system focuses on providing accurate, accessible, and scalable disease detection solutions while addressing challenges like diverse environmental conditions and dataset limitations.

Methodology Adopted in the Proposed System

The project employs a structured methodology comprising three key modules:

1. **Data Collection and Preprocessing**: A curated dataset of plant leaf images was used, ensuring diversity in disease types and environmental conditions. Preprocessing steps like resizing, normalization, and augmentation were applied to enhance the dataset's robustness and improve the model's performance.

2. **Implementation of Deep Learning Algorithm**: A ResNet-50-based architecture was fine-tuned for multi-class classification. Transfer learning was utilized to leverage pretrained weights, while additional dense layers were added for domain-specific training.
3. **Testing and Validation**: The trained model was evaluated on an unseen test dataset to assess its generalization ability. Performance metrics such as accuracy, precision, recall, and F1-score were calculated to validate the system's reliability.

Technical Features of the Proposed System

- **Deep Learning Integration**: Utilizes ResNet-50, a state-of-the-art convolutional neural network, for feature extraction and classification.

- **Data Augmentation**: Enhances model robustness by introducing variations in the training data, such as rotations, flips, and shifts.
- **Transfer Learning**: Reduces computational overhead and improves accuracy by fine-tuning a pretrained model.
- **MLFlow Integration**: Facilitates experiment tracking, model versioning, and artifact logging, ensuring a streamlined development process.
- **Scalable Deployment**: Designed for adaptability, with potential for deployment on edge devices, mobile platforms, or cloud services

2.4 Tools and Technologies used

1. Deep Learning Framework:

- TensorFlow: For implementing and fine-tuning the ResNet-50 architecture.
- Keras: To build and train the model with a user-friendly API.

2. Data Processing and Augmentation:

- OpenCV: For image preprocessing tasks like resizing and normalization.
- Albumentations: For advanced data augmentation techniques, including rotations, flips, and brightness adjustments.

3. Experiment Tracking and Management:

- MLFlow: To track experiments, log metrics, manage model versions, and streamline the workflow.

4. Development Environment:

- Google Colab: For GPU-accelerated model training and testing.
- VS Code: For code development and debugging.

5. Visualization Tools:

- ML FLow : For plotting accuracy, loss curves.
- Streamlit: For building an interactive frontend to showcase predictions and insights.

6. Hardware Acceleration:

- NVIDIA GPUs: To accelerate deep learning model training and inference.

2.5 Hardware and Software Requirements

Hardware Requirements:

1. **Processor:**
    - **Minimum:** Intel i5 or equivalent processor.
    - **Recommended:** Intel i7 or higher for faster and more efficient processing, especially when dealing with complex models.
2. **CPU/GPU:**
    - **Minimum:** NVIDIA GTX 1050 Ti for effective model training and inference.
    - **Recommended:** GPUs with higher processing power such as NVIDIA RTX series are ideal for faster deep learning model training.
3. **RAM:**
    - **Minimum:** 8 GB of RAM.
    - **Recommended:** 16 GB for better handling of large datasets and to ensure the smooth operation of machine learning tasks.
    -
4. **Camera (for real-world testing):**
    - A camera capable of capturing images with a minimum resolution of 224x224 pixels, suitable for field use to take pictures of plant leaves for disease detection.

Software Requirements:

1. **Programming Language:**
    - **Python** (version 3.8 or higher), which supports machine learning libraries and frameworks.
2. **Libraries & Frameworks:**
    - **TensorFlow (Version 2.x) or PyTorch (Version 1.10 or above):** Popular deep learning frameworks used for model development.
    - **OpenCV (Version 4.x):** Used for image preprocessing tasks such as resizing, filtering, and data augmentation.
    - **Scikit-learn (Version 1.x):** Provides tools for data analysis and evaluation metrics like precision, recall, and F1 score.
3. **Integrated Development Environment (IDE):**

- **Jupyter Notebook**, **PyCharm**, or **VS Code** are recommended for writing and executing code in an efficient manner, with Jupyter Notebook being preferred for easy experimentation and visualization.

4. **Operating System:**
    - **Windows 10/11**, **Linux (Ubuntu 20.04 or above)**, or **macOS**: All of these platforms support the necessary software tools and frameworks for the project.

The Literature survey concludes with the Hardware and software requirements for the proposed system which is CNN ResNet -50 .This section describes the scope of improvement of the existing systems by achieving the objectives.

Chapter 3: Software Requirement Specifications

This chapter introduces to definitions, acronyms and abbreviations used in the report , additionally it gives the general description of the product . It also describes the functional ,non functional requirements and external interface requirements.

3.1 Introduction

Definitions:

- **CNN (Convolutional Neural Network)**: A Convolutional Neural Network (CNN), also known as ConvNet, is a specialized type of deep learning algorithm mainly designed for tasks that necessitate object recognition, including image classification, detection, and segmentation.
- **ML (Machine Learning)**: A subset of artificial intelligence where models learn from data and make predictions without explicit programming.
- **MLFlow**: An open-source platform used to manage the machine learning lifecycle, including tracking experiments, packaging code, and deploying models.
- **OpenCV**: Open Source Computer Vision Library, widely used for image processing tasks in computer vision.
- **TensorFlow**: An open-source machine learning framework developed by Google, primarily used for deep learning and neural network-based tasks.
- **PyTorch**: An open-source deep learning framework developed by Facebook, popular for its flexibility and efficiency in training deep learning models.

Acronyms:

- **CNN**: Convolutional Neural Network
- **ML**: Machine Learning
- **MLFlow**: Machine Learning Flow
- **OpenCV**: Open Source Computer Vision
- **ResNet** : Residual Neural Network

Overview

This project focuses on the development of a computer vision-based system for detecting plant diseases, specifically designed to be accessible and beneficial to farmers. It combines advanced machine learning and image processing techniques to identify diseases from plant images. To ensure

smooth and efficient execution, certain hardware and software requirements are necessary for both training and real-world application.

3.2 General Description

Product Perspective

The plant disease detection system is designed to address a critical challenge faced by farmers: the timely identification of plant diseases. This system uses computer vision and machine learning techniques to detect diseases in crops like tomatoes, potatoes, and corn, and provides actionable insights for farmers. By using images of plant leaves, the system identifies potential diseases, thus helping in early intervention and reducing crop yield losses. The system is intended to be scalable, low-cost, and easy to use for farmers, especially those in rural and underdeveloped regions.

The product will be available as a mobile or web application, integrated with machine learning models that provide predictions based on real-time input from farmers. The primary stakeholders include farmers, agricultural technology companies, and NGOs. The system will be available for use in the field, enabling farmers to capture leaf images with their smartphones or cameras and receive disease predictions immediately.

Product Functions

1. **Disease Detection:**
   The system will identify diseases in plant leaves using deep learning models, providing a disease classification from the available classes (e.g., Tomato Early blight, Potato Healthy).

2. **Treatment Recommendations:**
   Upon detecting a disease, the system will provide treatment recommendations, including eco-friendly alternatives, to mitigate further spread and damage.

3. **Graphical User Interface:**
   The application will feature an intuitive, easy-to-navigate interface, making it accessible for farmers with minimal technical experience.

User Characteristics

Primary Users (Farmers):

- **Skill Level:** Primarily non-technical users, likely to have little experience with technology or machine learning.
- **Technical Needs:** Simple and intuitive interfaces for easy interaction with the system.
- **Goal:** To use the app for timely disease identification and to implement appropriate treatments.

Secondary Users (Agricultural Technicians, Researchers, NGOs):

- **Skill Level:** Technically proficient, with an understanding of agriculture and plant diseases.
- **Technical Needs:** Access to advanced features for analyzing system performance, updating models, and providing large-scale solutions.
- **Goal:** To monitor disease trends, update the system, and deploy it to larger groups of farmers in rural and underdeveloped regions.

End Users (Agri-tech Companies, Government Agencies):

- **Skill Level:** Highly technical, involved in large-scale deployment and optimization.
- **Technical Needs:** High scalability, integration with existing agricultural tools, and the ability to optimize for specific crops and regions.
- **Goal:** To deploy the system on a larger scale for commercial and developmental purposes, ensuring that it reaches farmers in various regions.

General Constraints

1. **Accuracy of Disease Detection:**
   The system's performance is dependent on the quality of the dataset. The accuracy of predictions may vary depending on environmental conditions, lighting, and image quality.

2. **Hardware Limitations:**
   The mobile devices or cameras used by farmers need to meet certain hardware specifications to capture clear and usable images for disease detection.

3. **Internet Connectivity:**
   Although the system can work offline for capturing images, the disease detection and

model updates might require internet connectivity for uploading and processing data, especially in rural areas with poor connectivity.

4. **Model Performance and Scalability:**
   The deep learning models may not perform equally well across all environments. They may need to be retrained or adjusted periodically to improve accuracy and adaptability in real-world field conditions.

Assumptions and Dependencies

1. **Dataset Availability and Quality:**
   The system assumes the availability of a diverse, up-to-date dataset covering various plant diseases and target crops.
2. **Farmer Access to Smartphones or Cameras:**
   It assumes farmers have access to smartphones or cameras capable of capturing high-resolution plant images. If not, alternative detection methods would be required.
3. **Agri-Tech Infrastructure:**
   The system depends on the presence of agri-tech infrastructure and support from NGOs and governments for large-scale deployment in remote areas.
4. **Model Update and Training:**
   The system will be periodically updated with new data, with models retrained to enhance accuracy based on the latest research and user feedback.
5. **User Support and Training:**
   The system assumes that basic training and support will be provided to farmers to ensure effective use of the app, especially for those with limited tech experience.

3.3 Functional Requirement

1. Introduction

The Plant Disease Detection System leverages deep learning techniques to automatically detect diseases in plant leaves based on images. This system uses a pre-trained ResNet-50 model, fine-tuned on a custom dataset of plant leaves. The system is designed to assist farmers, agricultural researchers, and horticulturists in quickly identifying plant diseases and taking timely action to prevent the spread of

these diseases. The system provides an easy-to-use interface where users can upload images of plant leaves, and receive predictions about the disease detected.

2. Input

The system requires the following inputs:

- **Leaf Images**: High-quality images of plant leaves, captured using a smartphone or camera. These images should ideally be clear, focused, and properly cropped to show the leaf in detail.
- **File Format**: The images can be uploaded in common image formats like JPEG, PNG, or BMP.
- **User Inputs (optional)**: A dropdown or selection box to choose the type of plant (if the dataset involves multiple plants) or additional metadata about the image (such as geographical location or plant variety) to improve accuracy.

3. Processing

The system performs the following key processes:

- **Image Preprocessing**:
    - **Resizing**: The uploaded image is resized to a standard resolution to match the model's input requirements.
    - **Normalization**: Pixel values are normalized to a scale between 0 and 1, to help with model convergence.
    - **Data Augmentation (during training)**: Techniques such as rotation, flipping, and color adjustments are applied to the training dataset to improve model robustness.
- **Disease Detection**:
    - The preprocessed image is passed through the fine-tuned ResNet-50 model, which classifies the image into one of the disease categories based on its features.
    - The model processes the image through multiple layers to extract hierarchical features and make a prediction about the disease.
- **Post-Processing**:
    - The results are interpreted and presented in a human-readable format.
    - Additional information about the disease (such as symptoms, treatment recommendations, and prevention methods) is fetched from a predefined database and shown to the user.

4. Output

The system provides the following outputs:

- **Predicted Disease**: It displays the predicted disease,
- **Disease Information**: Detailed information about the detected disease is provided, including:
    - Symptoms of the disease.
    - Recommended treatments and remedies.
    - Prevention methods and precautions.

.By processing the input images it provides the class label ,i.e the disease classification , the system helps to accurately identify and take timely actions to manage plant diseases.

3.4 Non-Functional Requirements

1. Performance

The system should process and return prediction results within 2-3 seconds for efficient user experience, especially for agricultural settings where quick feedback is crucial.

2. Reliability

The system should be fault-tolerant and capable of recovering from errors, ensuring uninterrupted service and minimizing downtime.

3. Usability

The system should have an intuitive and easy-to-use interface for all users, with clear instructions and minimal user effort to navigate through the features.

4. Security

Data security should be prioritized, with encryption of user-uploaded images and secure transmission (e.g., HTTPS), ensuring user privacy and protection.

5. Maintainability

The system should have modular architecture, well-structured code, and comprehensive documentation for easy updates and long-term maintenance.

3.5 External Interfaces Requirements

1. Hardware Interface

The system requires the following hardware components for operation:

1. **Smartphone/Camera**: A high-quality camera or smartphone is necessary for capturing clear images of plant leaves for disease detection.
2. **Server**: A server or high-performance workstation for processing the images and running the machine learning model. This can either be a physical machine or a cloud-based solution (e.g., AWS, Google Cloud, or Microsoft Azure).
3. **Storage Device**: Storage for saving uploaded images, model artifacts (trained models), and results. This can be local storage or cloud storage depending on deployment.

3.6 Design Constraints

1. Standard Compliance

a. The system should comply with industry standards for machine learning and AI, ensuring it follows best practices for data preprocessing, model training, and evaluation.
b. The image processing and model deployment should adhere to relevant standards, such as GDPR for user data privacy and accessibility standards for the user interface.
c. The model architecture (e.g., ResNet-50) should be compatible with standard deep learning frameworks like TensorFlow or PyTorch, following established norms for model structure and evaluation.

2. Hardware Limitations

a. The system should be optimized to run on standard agricultural field devices, which may have limited computational resources such as low-power CPUs and GPUs.
b. The image processing and model inference should be light enough to function efficiently on low-end devices without significantly affecting performance.
c. The system should also be able to work on devices with limited memory (e.g., 4GB or less) by ensuring that the model size and memory usage are minimized while maintaining accuracy.
d. Offline functionality may need to be supported for areas with limited internet access, requiring lightweight, offline versions of the model to be deployed on local hardware.

This chapter outlines the software requirements for a plant disease detection system using machine learning and computer vision, detailing the functional, non-functional, and external interface requirements. It describes the system's design, key features, user characteristics, and necessary hardware and software components for effective disease identification and treatment recommendations.

Chapter 4: System Design

The System Design of our Project gives an overview of the workflow architecture ,data flow diagrams of level 0 and 1 .Additionally it describes the architecture of the CNN ResNet-50 Model .

4.1 Architectural Design of the Project

The architectural design of the plant disease detection system is divided into three primary modules: **Data Collection and Preprocessing**, **Implementation of ANN/DL Algorithm**, and **Testing and Validation**. Each module plays a crucial role in ensuring the overall effectiveness and efficiency of the system. Below is a detailed explanation of the architectural flow for each module.

Block Diagram

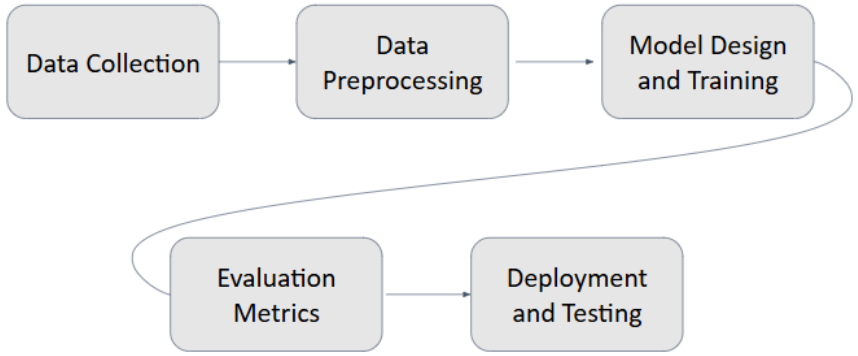

Figure 4.1 Block diagram

1. **Data Collection**

The data for the project is gathered from Kaggle called as Plant Village dataset [22]. Images of plant leaves are labeled with corresponding disease types or marked as healthy, ensuring diversity in the dataset by including variations in plant types, disease stages, and environmental conditions.

2. **Data Preprocessing**

Images are resized to a consistent resolution (e.g., 224x224 pixels) and normalized to a range of 0 to 1 for faster model convergence. Data augmentation techniques like rotation and flipping are applied to increase the model's robustness. The dataset is split into training, validation, and test sets to ensure the model generalizes well.

3. **Model Testing and Training**

A pre-trained model like ResNet-50 is fine-tuned using the training data to learn to classify plant diseases. During training, hyperparameters like learning rate and batch size are adjusted, and the model is trained for several epochs until convergence. After training, the model is tested on unseen data to evaluate its generalization performance.

4. **Evaluation Metrics**

Model performance is assessed using metrics like accuracy, precision, recall, F1-score, and confusion matrix. These metrics provide insights into the model's ability to correctly classify diseases and its effectiveness in handling imbalances between different classes.

5. **Testing and Validation**

Cross-validation is performed to ensure that the model's performance is consistent and not dependent on a specific data split. Hyperparameter tuning is done to optimize the model's performance, and overfitting and underfitting are monitored to ensure that the model generalizes well to new data.

Data definition

The success of any plant disease detection system depends on the quality, diversity, and relevance of the dataset used for training and evaluation. For this project, we utilized the widely recognized **PlantVillage Dataset**, a publicly available dataset hosted on Kaggle. This dataset is renowned for its extensive collection of labeled plant leaf images covering a broad range of crops and diseases.

Figure 4.2 Apple Scab

Figure 4.3 Corn Healthy

Figure 4.4 Potato Early Blight

Figure 4.5 Tomato Leaf Mold

Dataset Description

Image Resolution	**256x256**
Apple Scab	**630**
Apple Black Rot	**621**
Cedar Apple Rust	**275**
Healthy Apple	**1,645**
Corn Grey Leaf Spot	**513**
Corn Common Rust	**1,192**
Corn Healthy	**1,162**
Corn Northern Leaf Blight	**985**
Potato Early Blight	**1,000**
Potato Late Blight	**1,000**
Potato Healthy	**151**
Tomato Bacterial Spot	**2,127**
Tomato Early Blight	**1,000**
Tomato Late Blight	**1,909**
Tomato Leaf Mold	**952**
Tomato Healthy	**1,591**
Total Images	**16,753**

Dataset Overview

The PlantVillage dataset comprises thousands of high-resolution images of plant leaves, categorized by crop type and disease condition. It includes healthy and diseased leaf samples for a variety of crops, such as tomato, potato, apple, and corn, among others. These images are meticulously labeled and span multiple disease categories, including bacterial, fungal, and viral infections, making the dataset an ideal choice for building a robust and accurate model.

Data Augmentation

Given the vastness of the PlantVillage dataset, we carefully reduced its scope to focus on four major crops: **Apple, Corn, Potato, and Tomato**. This decision was based on several key factors:

1. **Focus on Economically Significant Crops**
    - The selected crops play a critical role in global food security and agricultural output. For instance, tomatoes and potatoes are widely cultivated staples, while apples and corn are essential for both direct consumption and industrial uses.
    - Diseases affecting these crops have a substantial economic impact, with yield losses potentially resulting in significant financial strain for farmers.
2. **Practical Relevance**
    - By targeting commonly grown crops, the system aligns with the practical needs of farmers. These crops are more likely to require disease detection systems due to their prevalence and susceptibility to a variety of diseases.
    - Crops with limited geographical or economic relevance were excluded to ensure the system's real-world utility for the majority of users.
3. **Dataset Balance and Representation**
    - The original dataset contained some crops and diseases with insufficient data for training. To ensure a balanced and well-represented dataset, we excluded classes with limited samples. This helped prevent model bias and enhanced the generalization capabilities of the trained system.
4. **Computational Efficiency**
    - Training deep learning models on large-scale datasets can be resource-intensive. Reducing the dataset size to focus on specific crops ensured that training could be performed efficiently without compromising on accuracy or robustness.

Dataset Composition

After this refinement, the dataset included the following crop-disease combinations:

- **Apple**: Apple Scab, Black Rot, Cedar Apple Rust, and Healthy.
- **Corn**: Cercospora Leaf Spot (Gray Leaf Spot), Common Rust, Northern Leaf Blight, and Healthy.
- **Potato**: Early Blight, Late Blight, and Healthy.
- **Tomato**: Early Blight, Late Blight, Leaf Mold, Bacterial Spot, and Healthy.

This selection captures the most common and economically impactful diseases for each crop, ensuring that the system is applicable in diverse agricultural scenarios.

Image Preprocessing

To prepare the dataset for model training, several preprocessing steps were applied:

1. **Resizing**: All images were resized to a resolution of **224x224 pixels**, which matches the input size required by the ResNet50 model used in this project.
2. **Normalization**: Pixel values were normalized to ensure consistency and compatibility with the deep learning framework.
3. **Augmentation**: Data augmentation techniques, such as rotation, flipping, zooming, and shifting, were employed to artificially expand the dataset and improve the model's ability to generalize to unseen data.

The reduction of the dataset to focus on Apple, Corn, Potato, and Tomato was driven by practical, agricultural, and technical considerations. This approach ensures that the project addresses crops with the greatest economic and nutritional significance while maintaining dataset balance and computational efficiency. By concentrating on these high-priority crops, this system offers a targeted and impactful solution for plant disease detection in real-world agricultural practices.

Module specification

Providing insights into three primary modules implemented in the project: **Data Collection and Preprocessing**, **Implementation of Artificial Neural Network/Deep Learning Algorithm**, and

Testing and Validation. Each module plays a critical role in building an efficient and accurate plant disease detection system.

Module 1: Data Preprocessing

Input:

The inputs for this module consist of images of plant leaves, both healthy and diseased. These images are sourced primarily from the publicly available **PlantVillage Dataset**, supplemented by field-collected images or IoT devices like cameras deployed in agricultural settings. Additionally, metadata such as plant type, disease category, and environmental conditions is associated with the images.

Process:

1. **Resizing**: Images are resized to a standardized dimension of **224x224 pixels** to ensure compatibility with the ResNet-50 model.
2. **Normalization**: Pixel values are scaled to a range of **0 to 1**, enhancing consistency across the dataset and optimizing performance during training.
3. **Data Augmentation**: Techniques such as flipping, rotation, and zooming are applied to artificially expand the dataset. This process increases diversity, reduces overfitting, and improves the model's ability to generalize across varying conditions.

Output:

The output of this module is a **preprocessed dataset** that is clean, normalized, and ready to be used for training and testing. The dataset includes augmented variations of the original images to enhance robustness and reliability.

Module 2:Plant Disease detection

Input:

The inputs for this module include:

- The **preprocessed dataset**, divided into training and validation subsets.
- The network configuration, which in this case is the **ResNet-50 architecture**, pre-trained on the ImageNet dataset.
- Hyperparameters, such as learning rate, batch size, and the number of epochs, to guide the training process.

Process:

1. **Model Construction**: The ResNet-50 architecture is adapted for multi-class classification of plant diseases. Residual blocks, a key feature of ResNet, are leveraged to minimize vanishing gradient issues and enhance training efficiency.
2. **Training**: The model is trained using the preprocessed dataset. A **categorical cross-entropy loss function** is employed to measure prediction errors, while the **Adam optimizer** adjusts model weights for improved accuracy.
3. **Validation**: After each training epoch, the model's performance is evaluated on the validation subset to monitor progress and detect potential overfitting.
4. **Model Saving**: The trained model is saved in a format such as **.h5**, enabling further testing, evaluation, and eventual deployment.

Output:

The output of this module is a **trained ResNet model file** containing the learned weights and architecture. Additionally, training logs, including loss and accuracy curves, are generated to track the performance across epochs.

Module 3: Testing and Validation

Input:

The inputs for this module include:

- The **trained ResNet model** from Module 2.
- A **test dataset**, containing images not seen during training or validation.
- Metrics for evaluation, such as accuracy, precision, recall, F1 score, and a confusion matrix.

Process:

1. **Testing**: The trained model is tested on unseen images from the test dataset. Predictions generated by the model are compared with the actual ground truth labels.
2. **Validation**: The model's generalizability is evaluated by analyzing performance on diverse test samples. Misclassifications are closely examined to identify potential weaknesses.
3. **Visualization**: A **confusion matrix** is generated to illustrate the classification performance. Additional visual outputs, such as ROC curves and samples of misclassified images, are used to interpret results effectively.

Output:

The final outputs of this module include:

- **Performance Metrics**: A detailed report containing accuracy, precision, recall, F1 score, and insights into model strengths and limitations.
- **Refined Model**: A thoroughly evaluated model ready for deployment, ensuring high accuracy and robustness in real-world scenarios.

The modular approach described above provides a clear, organized, and effective process for building a reliable plant disease detection system. Each module works together, playing a crucial role in creating a practical solution that helps farmers quickly and accurately identify plant diseases. This system is designed to be both user-friendly and impactful in addressing real-world agricultural challenges.

Module 4: Deploy in MLFlow

Input

The deployment module requires the following inputs:

- The trained ResNet model from the previous module.
- MLflow Tracking Server URL for experiment logging and artifact storage.
- Metadata such as model parameters, training metrics, and validation results.

Process

1. **MLflow Integration**: The model is integrated with MLflow, a platform for tracking machine learning experiments. This enables the logging of metrics, parameters, and artifacts during the deployment process.
2. **Model Artifact Logging**: The trained ResNet model is saved and logged as an artifact in MLflow. This ensures that the model is accessible for future use, evaluation, or further fine-tuning.
3. **Experiment Tracking**: MLflow tracks the entire deployment pipeline, including the experiment ID, run details, and associated metrics such as accuracy, precision, and loss.
4. **Version Control**: By leveraging MLflow's versioning capabilities, different iterations of the model can be stored and compared to ensure optimal performance.
5. **Serving and Access**: The deployed model is prepared for serving via MLflow's built-in model serving feature, making it accessible through APIs for real-time predictions or batch processing.

Output

- A fully deployed and versioned model stored in MLflow.
- Logged experiment details, including metrics, parameters, and artifacts, accessible through the MLflow Tracking UI.
- A model endpoint ready for integration into applications or APIs for real-time or batch inference.

This deployment ensures the system is production-ready, enabling scalability, easy tracking, and continuous improvements.

4.2 Data Flow Diagram

Level 0

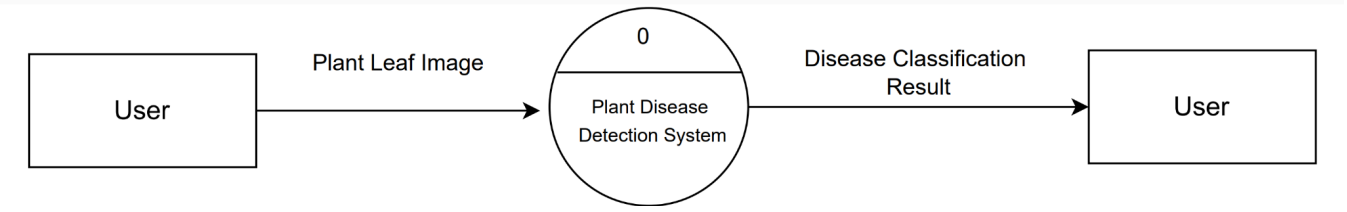

Figure 4.6 Data Flow Diagram Level 0

1. At this level, the system is represented as a single process, "Plant Disease Detection System."
2. **External entities**:
    - User/Farmer: Provides the input (plant leaf images).
    - System Output: Sends back the detection results (healthy or diseased with remedy suggestions).
3. **Data flows**:
    - The user uploads an image of a plant leaf.
    - The system processes the image and returns the result to the user.

Level 1

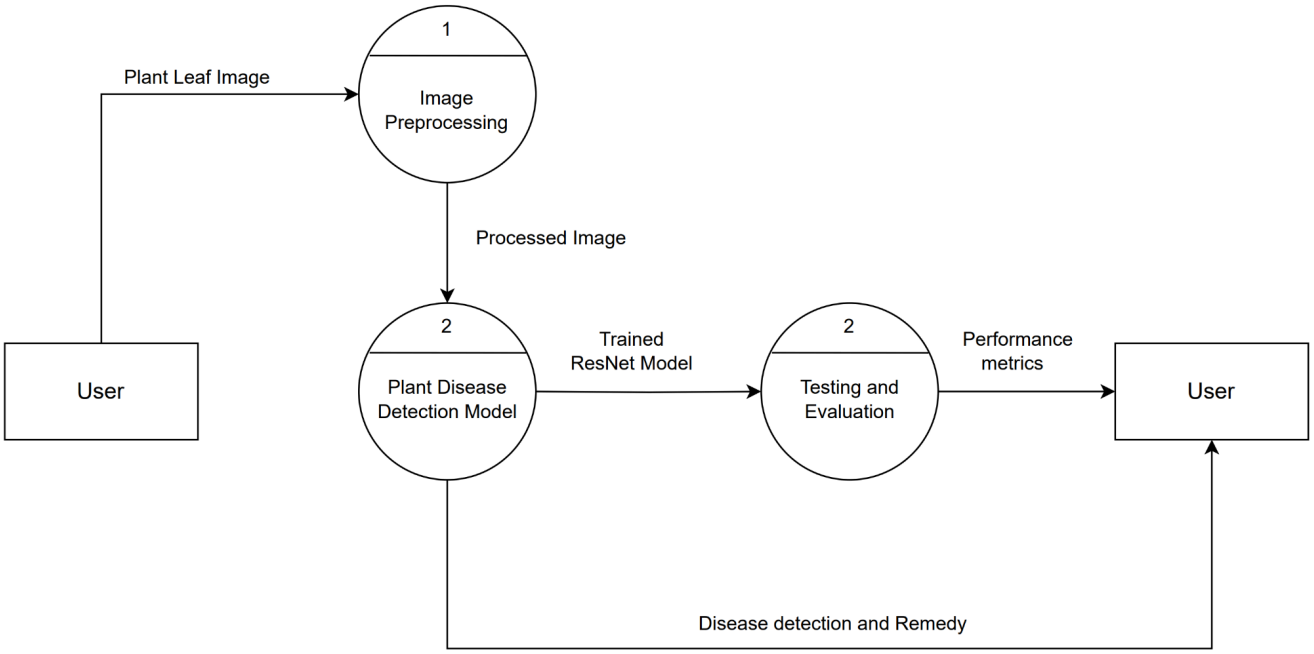

Figure 4.7 Data Flow Diagram Level 1

Level 1 Expands the single process into multiple subprocesses :

1.Subprocesses:

1. **Image Preprocessing**: Handles tasks like resizing, normalization, and data augmentation of input images.
2. **Disease Detection and Classification**: Uses the ResNet-50 model to predict whether the plant is healthy or diseased.
3. **Result Generation**: Generates and formats the final output, including disease type and suggested remedies.

2.Data Stores:

- **Image Database**: Temporary storage for uploaded and preprocessed images.
- **Model Logs**: Stores performance metrics and other logs related to the MLFlow tracking system.

3.Data flows:

- Image flows from the user to preprocessing, then to the disease detection model.
- The model outputs predictions, which flow to the result generation process.
- Final results are sent to the user, completing the data flow.

4.3 Description of the CNN ResNet -50 Architecture

The ResNet-50 architecture, a highly popular convolutional neural network (CNN), is employed to implement the deep learning solution for plant disease detection. ResNet-50 is well-known for its ability to handle complex image classification tasks while minimizing the vanishing gradient problem, thanks to its residual learning framework.

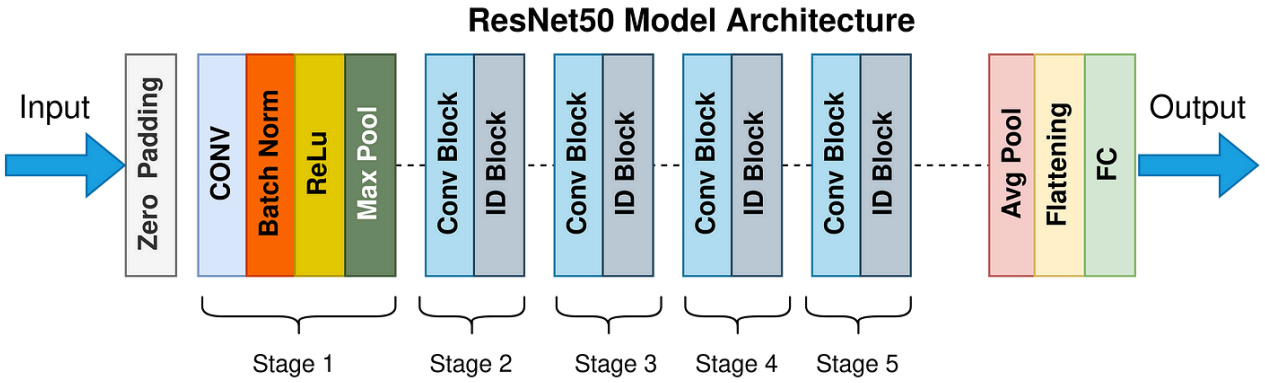

Figure 4.8 CNN ResNet Architecture [21]

Key Features of ResNet-50:

1. **Residual Learning Framework**:
   ResNet-50 is built with residual blocks, which allow the network to learn identity mappings. This ensures that deep layers focus on learning residual functions, improving the overall accuracy of the model while maintaining computational efficiency.
2. **Deep Architecture**:
   ResNet-50 contains 50 layers, enabling it to extract high-level and low-level features from input images. This is particularly important for identifying subtle differences between healthy and diseased leaves.
3. **Pretrained Weights**:
   The model utilizes pretrained weights on the ImageNet dataset, which significantly reduces the computational cost and time required for training. By leveraging transfer learning, the model can effectively adapt to the plant disease classification task with minimal data and training.

Implementation Details:

- **Input Layer**:
  The network takes input images resized to 224x224 pixels with three color channels (RGB). Images are normalized to ensure consistent performance.
- **Base Model**:
  The ResNet-50 architecture was used as the base model with its top layers removed. This allowed customization to fit the specific requirements of the project.
- **Custom Layers**:
  After the base model, additional layers were added, including:
  - **Global Average Pooling Layer**: Reduces feature map dimensions while retaining spatial information.
  - **Dropout Layer**: Introduced to prevent overfitting by randomly disabling certain neurons during training.
  - **Dense Layers**: Includes a fully connected layer with 256 neurons (ReLU activation) for feature extraction and a final dense layer with the softmax activation function to classify images into the corresponding plant diseases.
- **Training Details**:
  The model was trained using the Adam optimizer, which adapts the learning rate for efficient convergence. Cross-entropy was used as the loss function to handle the multi-class classification task. The dataset was split into training and validation subsets to ensure reliable evaluation of the model's performance during training.

Benefits of Using ResNet-50:

- **Accuracy and Efficiency**: Its residual framework ensures high classification accuracy without requiring excessive computational resources.
- **Transfer Learning**: Leveraging pretrained weights enables faster and more efficient training with limited data.
- **Scalability**: The model can be extended to include additional crops and diseases by fine-tuning on new datasets.

By integrating the ResNet-50 architecture into the project, the system achieves robust performance in identifying plant diseases, ensuring that farmers and agricultural stakeholders can benefit from an accurate and reliable solution.

Chapter 5: Implementation

The design and implementation involved the systematic development of a deep learning-based solution for plant disease detection using the ResNet-50 architecture. The code is divided into distinct sections, each addressing specific tasks required to preprocess the dataset, build and train the model, and evaluate its performance. The code is written in VS code in the format of an ipynb file. The notebook is connected to a python environment on the system where all the necessary libraries and packages were installed.

5.1 Code Snippets

1. Importing the libraries

```
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.applications import ResNet50
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Dense, Flatten, GlobalAveragePooling2D, Dropout
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import ModelCheckpoint
import mlflow
import mlflow.tensorflow
import os
```

Code 5.1:Imported Libraries

This piece of code imports several libraries and modules that are essential for building, training, and evaluating the deep learning model for plant disease detection.

TensorFlow and Keras Imports

- **tensorflow as tf**: TensorFlow is an open-source machine learning framework widely used for deep learning tasks. It provides a comprehensive ecosystem for training and deploying machine learning models. The alias `tf` is commonly used as shorthand for referencing TensorFlow functionality throughout the code.
- **tensorflow.keras.preprocessing.image.ImageDataGenerator**: This module is used for real-time data augmentation and image preprocessing. As discussed earlier, it allows transformations such as rotating, zooming, and flipping images to increase the diversity of the training data and reduce overfitting by helping the model generalize better to new, unseen data.

- **tensorflow.keras.applications.ResNet50**: This imports the ResNet50 model, a pre-trained convolutional neural network (CNN) model. ResNet50 is widely used for transfer learning, where a pre-trained model on a large dataset like ImageNet is fine-tuned for a specific task. In this case, ResNet50 will be used to detect plant diseases by leveraging its powerful feature extraction capabilities.
- **tensorflow.keras.models.Model**: This is the base class for building Keras models. It is used to create both sequential and functional models in Keras. The `Model` class provides the necessary tools to define the layers and architectures of neural networks.
- **tensorflow.keras.layers.Dense, Flatten, GlobalAveragePooling2D, Dropout**: These are essential building blocks for constructing the neural network layers:
    - **Dense**: Fully connected layers that apply weights to the input to produce an output. The Dense layers will be used to add classification layers on top of the base ResNet50 model.
    - **Flatten**: This layer is used to convert multi-dimensional input (like the output of the convolutional layers) into a one-dimensional vector, which can be passed to the fully connected layers.
    - **GlobalAveragePooling2D**: A pooling layer that averages the spatial dimensions of the input. This layer reduces the size of the output from the convolutional layers, making the model more computationally efficient and less prone to overfitting.
    - **Dropout**: A regularization technique that randomly sets a fraction of input units to 0 during training, helping prevent overfitting and improving the model's ability to generalize to new data.
- **tensorflow.keras.optimizers.Adam**: Adam is an optimization algorithm that adjusts the learning rate during training to minimize the loss function. It is widely used due to its efficiency and adaptability, especially for tasks involving deep learning. The optimizer helps the model adjust weights during training to learn from the data.
- **tensorflow.keras.callbacks.ModelCheckpoint**: This callback is used to save the best model during training, based on validation performance. This ensures that the model with the best generalization ability is preserved, avoiding the problem of overfitting to the training data. The model will be saved every time an improvement in validation accuracy is achieved.

MLFlow Imports

- **mlflow**: MLFlow is an open-source platform designed to manage the machine learning lifecycle, including tracking experiments, model versioning, and deployment. The mlflow module is used

to track the model's training process and manage different versions of the model as experiments are run.

- **mlflow.tensorflow**: This module integrates TensorFlow with MLFlow, enabling seamless tracking of TensorFlow-based models. It automatically logs model parameters, metrics, and artifacts (like the trained model) during training, which helps keep track of experiments and facilitates easy model deployment.

The code imports the necessary components for building and training a plant disease detection model using a deep learning approach. The model uses ResNet50 as a base, applying transfer learning for feature extraction, and then fine-tuning the network to classify different plant diseases.

2. ML Flow Experiment Tracking

```
DATASET_DIR = "./Data"
IMG_SIZE = (224, 224)
BATCH_SIZE = 32
EPOCHS = 3
LEARNING_RATE = 0.0001
MLFLOW_TRACKING_URI = "http://127.0.0.1:8082"

mlflow.set_tracking_uri(MLFLOW_TRACKING_URI)
mlflow.tensorflow.autolog()
```

Code 5.2 ML Flow Experiment tracking

In this section, the primary focus is on configuring the dataset path, setting parameters for training, and enabling experiment tracking using MLFlow.

- **Dataset Directory (DATASET_DIR)**: This variable defines the path to the folder containing the dataset. The images in this folder should be organized into subfolders, each representing a different class (e.g., a specific plant disease). By specifying the **DATASET_DIR**, we can easily load and preprocess the images from the correct location.
- **Image Size (IMG_SIZE)**: The images are resized to 224x224 pixels to match the input size required by the ResNet-50 model. This ensures that all images passed to the model have consistent dimensions, enabling optimal performance during training.
- **Batch Size (BATCH_SIZE)**: This defines how many images the model will process in one iteration (batch). A batch size of 32 means that the model will process 32 images before updating its weights. This is a common batch size for training deep learning models and strikes a balance between computational efficiency and model performance.

- **Number of Epochs (EPOCHS)**: The number of epochs determines how many times the model will iterate over the entire dataset. Set to 3 here for testing purposes, this can be increased for more extensive training to improve the model's accuracy.
- **Learning Rate (LEARNING_RATE)**: The learning rate controls the speed at which the model's weights are updated during training. A smaller learning rate, such as 0.0001, helps ensure the model learns gradually and avoids overshooting optimal solutions. This is especially crucial when using pre-trained models like ResNet-50, as fine-tuning requires careful adjustments.
- **MLFlow Tracking URI (MLFLOW_TRACKING_URI)**: This points to the MLFlow server that will handle experiment logging. The local server URL **(http://127.0.0.1:8082)** is used for tracking experiments during training, where parameters, metrics, and models are logged for easy review and comparison.

Additionally, the **mlflow.set_tracking_uri()** function is called to set the connection to the tracking server, while **mlflow.tensorflow.autolog()** automatically logs the training process, including metrics like loss and accuracy, as well as the model architecture. This integration with MLFlow allows for comprehensive tracking of the model's performance.

3. Data Augmentation

```
train_datagen = ImageDataGenerator(
    rescale=1.0/255,
    rotation_range=20,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    validation_split=0.2
)

train_generator = train_datagen.flow_from_directory(
    DATASET_DIR,
    target_size=IMG_SIZE,
    batch_size=BATCH_SIZE,
    class_mode='categorical',
    subset='training'
)

val_generator = train_datagen.flow_from_directory(
    DATASET_DIR,
    target_size=IMG_SIZE,
    batch_size=BATCH_SIZE,
    class_mode='categorical',
    subset='validation'
)
```

Code 5.3 Data Augmentation

The definition of the data augmentation strategies and prepares the training and validation datasets for the model.

- **ImageDataGenerator (train_datagen)**: This function is used to perform data augmentation, which improves the model's ability to generalize by artificially expanding the dataset. The parameters inside **ImageDataGenerator** include:
    - **rescale**: Normalizes pixel values by scaling them from a range of 0–255 to 0–1, making them easier to process by the model.
    - **rotation_range, width_shift_range, height_shift_range**: These introduce random rotations and shifts in the images to simulate different orientations and positions, improving robustness.
    - **shear_range, zoom_range**: Randomly applies shear and zoom effects to make the model resilient to image distortions.
    - **horizontal_flip**: Flips images horizontally, helping the model learn features invariant to orientation.
    - **validation_split**: Reserves 20% of the data for validation, ensuring a separate subset for model evaluation.
- **train_generator**: The **train_datagen.flow_from_directory()** method is used to load and augment images for training. It:
    - Loads images from the specified **DATASET_DIR**, resizing them to the target size (**IMG_SIZE**).
    - Sets the batch size (**BATCH_SIZE**) and ensures the labels are one-hot encoded (**class_mode = 'categorical'**).
    - Specifies that this generator is for training data using the `subset='training'` parameter.
- **val_generator**: Similar to `train_generator`, the `val_generator` loads images for validation. It uses the same augmentation parameters but focuses on validation data (`subset='validation'`), allowing the model to be evaluated on unseen data during training.

The **ImageDataGenerator** is used to apply data augmentation techniques, which are crucial for improving the model's robustness and generalization ability. By rescaling pixel values to a range of 0 to 1, we standardize the input data, making it more efficient for the model to process. Augmentations such as random rotation, width and height shifting, shearing, and zooming increase the diversity of the training data. This ensures that the model is exposed to a variety of plant image conditions, such as

different angles, scales, and orientations, which helps prevent overfitting and improves its ability to generalize to new, unseen images. The horizontal flip also makes the model invariant to the horizontal positioning of plants. Additionally, reserving 20% of the dataset for validation ensures that the model's performance is assessed on data it hasn't been trained on, providing an unbiased evaluation and further preventing overfitting. These steps collectively help the model become more capable of accurately identifying plant diseases under real-world conditions, where images can vary in multiple ways.

4. Model loading and Training

```
base_model = ResNet50(weights="imagenet", include_top=False, input_shape=(224, 224, 3))

x = base_model.output
x = GlobalAveragePooling2D()(x)
x = Dropout(0.5)(x)
x = Dense(256, activation='relu')(x)
predictions = Dense(len(CLASS_NAMES), activation='softmax')(x)

model = Model(inputs=base_model.input, outputs=predictions)

model.compile(optimizer=Adam(learning_rate=LEARNING_RATE), loss='categorical_crossentropy', metrics=['accuracy'])
```

Code 5.4 Model Loading and Training

The code utilizes **ResNet50**, a pre-trained deep learning model developed on the ImageNet dataset. This model is advantageous because it has already learned fundamental visual features like edges and textures, which are transferable to the plant disease detection task. By setting **include_top = False**, the final classification layers of ResNet50 are excluded, allowing for the addition of custom layers suited for the specific problem at hand.

The following steps are performed:

1. **GlobalAveragePooling2D**: This layer reduces the dimensionality of the data by averaging the feature maps, transforming them into a single vector, which helps retain important information while reducing complexity.
2. **Dropout(0.5)**: This technique randomly deactivated 50% of the neurons during training, promoting generalization and mitigating overfitting by preventing the model from relying too heavily on specific neurons.
3. **Dense(256, activation='relu')**: This fully connected layer allows the model to capture more complex patterns in the data, with the ReLU activation introducing non-linearity to improve learning capabilities.

4. **Dense(len(CLASS_NAMES), activation='softmax')**: This output layer makes the final predictions by converting the output into probabilities, each representing the likelihood of the image belonging to a particular class of plant disease.

The model is compiled using the **Adam optimizer**, which dynamically adjusts the learning rate during training for faster convergence, and **categorical cross entropy** to measure prediction accuracy. The performance is evaluated using **accuracy** as the metric to track model effectiveness.

Leveraging a pre-trained model like ResNet50 enhances the training efficiency by building upon extensive prior learning, ultimately improving the model's capacity to detect plant diseases more accurately.

5. Best model Selection

```
mlflow.start_run()

checkpoint_path = "./best_model.h5"
checkpoint = ModelCheckpoint(filepath=checkpoint_path, monitor='val_accuracy', save_best_only=True, mode='max')

history = model.fit(
    train_generator,
    validation_data=val_generator,
    epochs=EPOCHS,
    callbacks=[checkpoint]
)

mlflow.log_artifact(checkpoint_path)  # Log the best model
mlflow.end_run()
```

Code 5.5 Selecting the best model

The code initiates the training process for the plant disease detection model while simultaneously tracking the experiment using **MLFlow**. The flow of the code is as follows:

1. **mlflow.start_run()**: This command begins a new MLFlow run, enabling the tracking of parameters, metrics, and model artifacts throughout the training process.
2. **ModelCheckpoint**: The `checkpoint` callback is created to save the model's weights at the file path `"./best_model.h5"`. The callback monitors the validation accuracy (`'val_accuracy'`) during training, ensuring that only the best-performing model (with the highest validation accuracy) is saved. This helps in preserving the model state that performs best on unseen data.
3. **model.fit()**: The model is trained on the training data provided by the `train_generator` and validated on the `val_generator`. The training runs for a set number of epochs (3 for testing purposes), with the `checkpoint` callback monitoring the model's performance. The

`train_generator` handles data augmentation and prepares the images for model input, while the `val_generator` provides validation data for performance evaluation after each epoch.

4. **mlflow.log_artifact(checkpoint_path)**: After the training is complete, the best model, stored at the checkpoint path, is logged in MLFlow as an artifact. This allows for easy retrieval and comparison of models in future experiments or deployments.
5. **mlflow.end_run()**: Finally, the MLFlow run is concluded, and all logged data, including metrics and model artifacts, are saved for analysis and future reference.

This approach enables model training with continuous performance monitoring and ensures the best model is preserved for further evaluation or deployment, all while leveraging MLFlow's experiment tracking capabilities.

6. Epochs

```
Epoch 1/3
419/419 ━━━━━━━━━━━━━━━━━━━━ 0s 8s/step - accuracy: 0.7728 - loss: 0.7714
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. T
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. T
419/419 ━━━━━━━━━━━━━━━━━━━━ 3504s 8s/step - accuracy: 0.7731 - loss: 0.7704 - val_accuracy: 0.1138 - val_loss:
Epoch 2/3
419/419 ━━━━━━━━━━━━━━━━━━━━ 0s 8s/step - accuracy: 0.9665 - loss: 0.1046
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. T
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. T
419/419 ━━━━━━━━━━━━━━━━━━━━ 3744s 9s/step - accuracy: 0.9665 - loss: 0.1046 - val_accuracy: 0.3986 - val_loss:
Epoch 3/3
419/419 ━━━━━━━━━━━━━━━━━━━━ 0s 8s/step - accuracy: 0.9746 - loss: 0.0805
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. T
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. T
419/419 ━━━━━━━━━━━━━━━━━━━━ 3885s 9s/step - accuracy: 0.9746 - loss: 0.0805 - val_accuracy: 0.9836 - val_loss:
2025/01/14 18:14:59 WARNING mlflow.tensorflow: Failed to infer model signature: could not sample data to infer m
2025/01/14 18:14:59 WARNING mlflow.tensorflow: You are saving a TensorFlow Core model or Keras model without a s
2025/01/14 18:15:20 WARNING mlflow.models.model: Model logged without a signature and input example. Please set
🏃 View run tasteful-stag-438 at: http://127.0.0.1:8082/#/experiments/0/runs/207f34c70b4e4cf3ba84a6752e886a00
🧪 View experiment at: http://127.0.0.1:8082/#/experiments/0
```

Figure 5.1: The training process

The code is running the training process for the plant disease detection model, with the following steps:

1. **Epochs 1-3**: The model is trained for three epochs, progressively improving in accuracy. In the first epoch, the model achieves an accuracy of 77.28% on the training data, with a validation accuracy of 11.38%. This is typical for the early stages of training, where the model starts to recognize basic patterns. By the second epoch, the accuracy improves to 96.65%, with validation accuracy increasing to 39.86%. By the third epoch, the model reaches an accuracy of 97.46% on the training set and 98.36% on the validation set, indicating it is effectively learning and generalizing.

2. **Saving the Best Model**: The model is being saved in HDF5 format (`.h5` file), but there are warnings suggesting that saving in the newer Keras format (`.keras`) is recommended. Despite this, the model is still being saved for future use.
3. **MLFlow Warnings**: There are a few warnings related to logging and model signature inference with MLFlow. These do not affect the training process but indicate that some features of MLFlow, such as automatic inference of the model's signature and support for inference with pandas DataFrames, are not available due to missing model signature information.
4. **Run and Experiment Tracking**: MLFlow is tracking the experiment, logging details such as the accuracy and the best model saved during training. The user can view the results of the experiment on the MLFlow UI at `http://127.0.0.1:8082`.

Overall, the model is making good progress, and the best model is being saved for later use. Despite the warnings, the core training and tracking process is functioning as expected.

All the modules of the project can be explained with the help of code snippets and explained

5.2 Results and Discussions

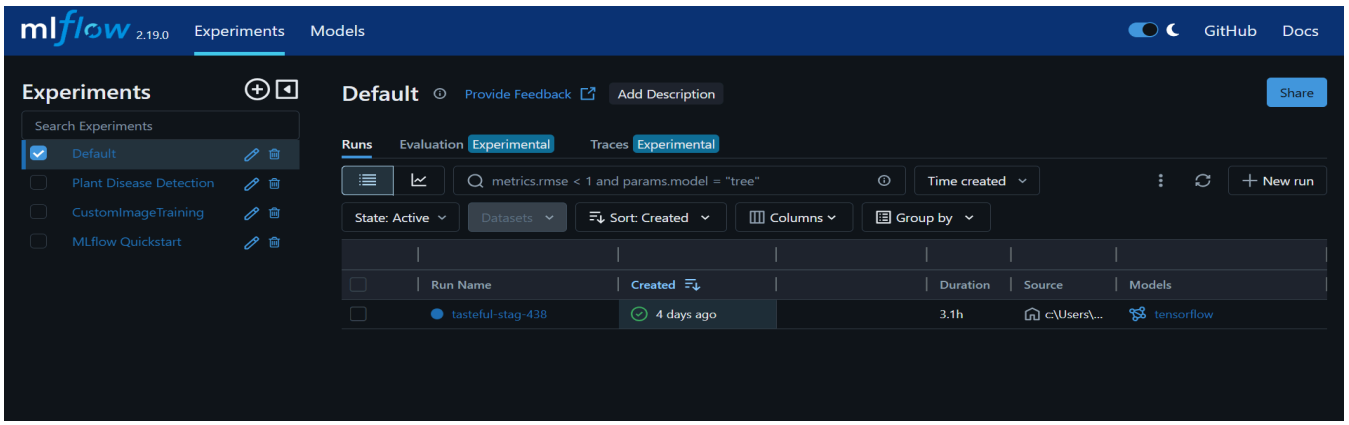

Figure 5.2 ML Flow Dashboard

The entire process of the model being trained is logged and tracked using MLFlow, an MLOps tool. The dashboard shows the run where the model was trained and a broad overview as to the details of the software and the different runs that can be tracked. MLFlow gives the advantage of reusability, versioning and tracking that allows for smarter management of the entire ML workflow.

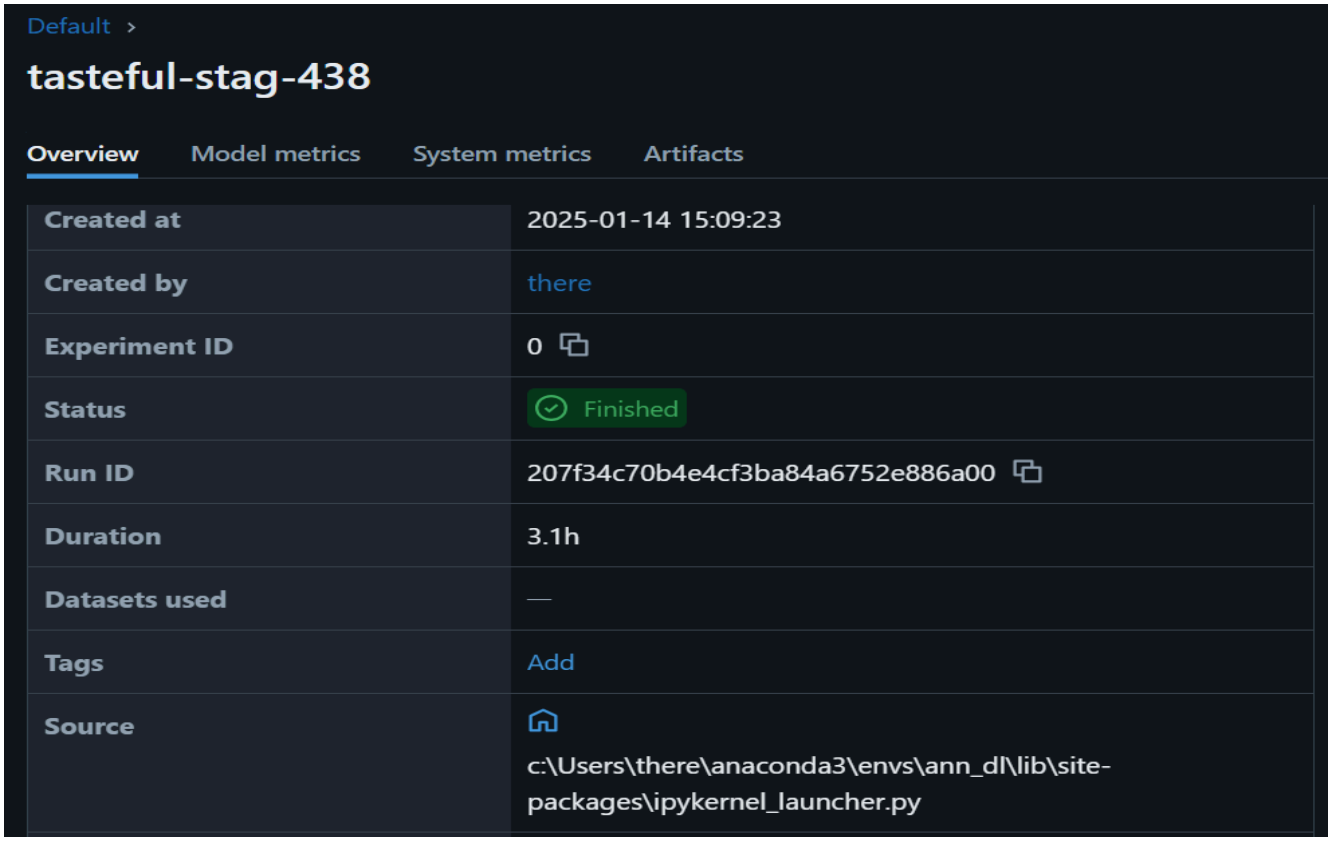

Figure 5.3 Overview of the ML flow model

Here, the details of the specific run where the model was trained are given. The date of creation, the user and the status of the run are given. The time taken to train the model is also logged here.

Snapshots of the Implementation:

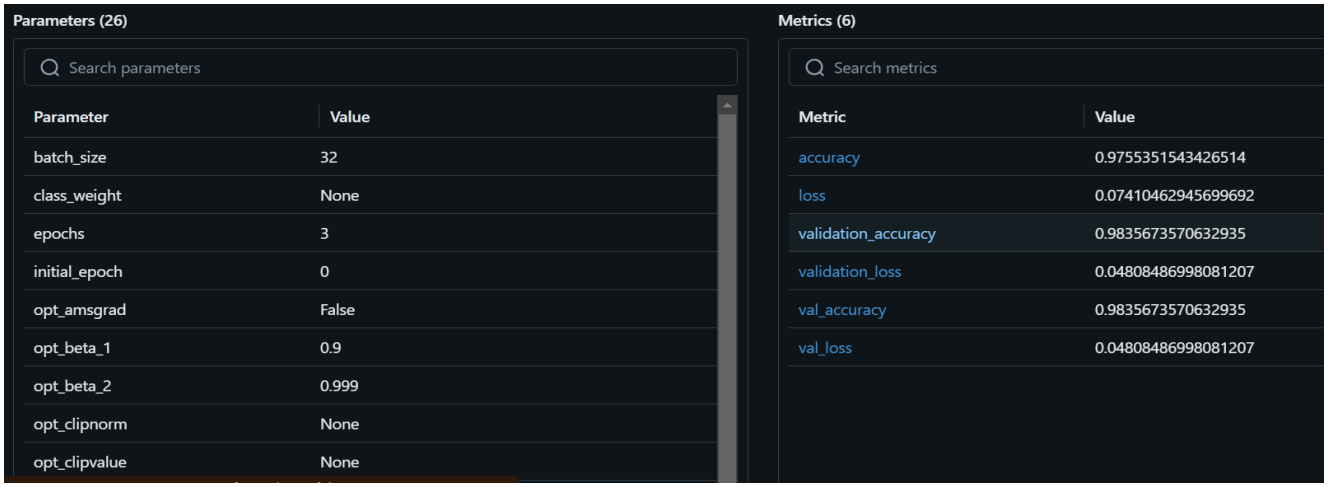

Figure 5.4 Model evaluation Parameters

Here, the various hyperparameters related to the model are given, these include the number of training epochs, batch size and various others. There are a total of 26 hyperparameters used. On the right side of the panel, the various model metrics are given. These include the accuracy, loss, validation accuracy and validation loss.

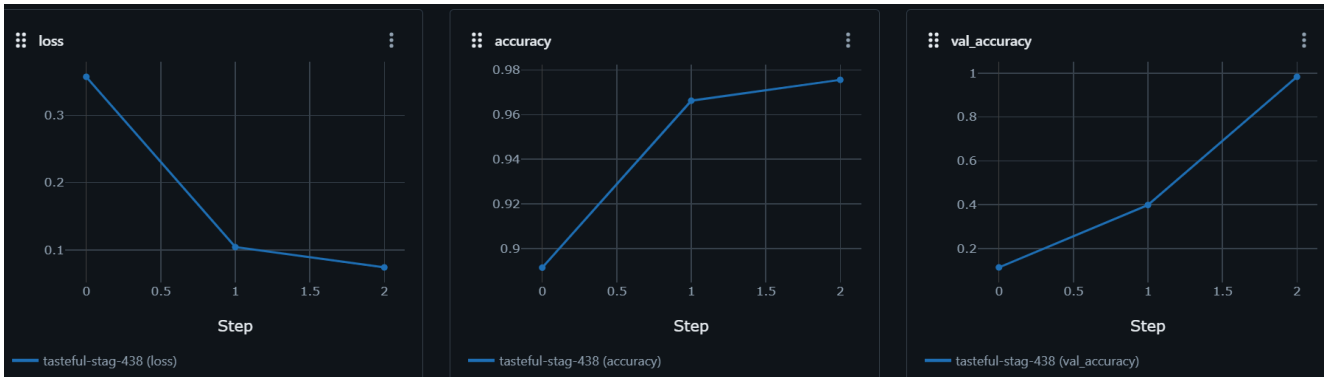

Figure 5.5 Visualization of model metrics

The above is a visual representation of the change in model metrics with each epoch. As three epochs were run, we see three discrete values in all the different metrics. It's clear that the model's performance improves drastically from the first epoch evidenced by the change in all three metrics. The loss reduces from 0.77 till 0.08. The accuracy improves from 0.77 to 0.97.

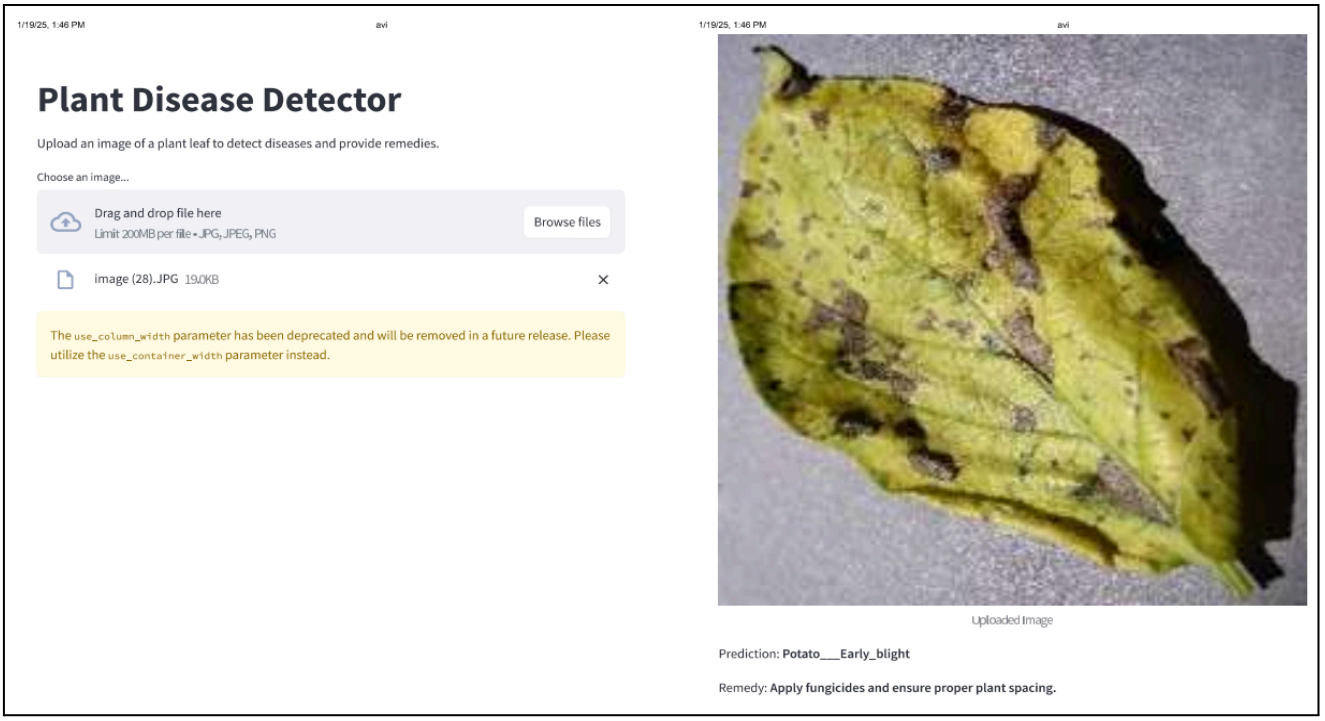

Figure 5.6 Final Output

Here the GUI displays a section where an image can be either dragged and dropped or uploaded by browsing files on the system. The uploaded image is then displayed along with the predicted

disease below which there is the proposed remedy. This GUI is a very simple and user-friendly interface that makes use of the model and connects it to the end user in a visual and hence interpretable format.

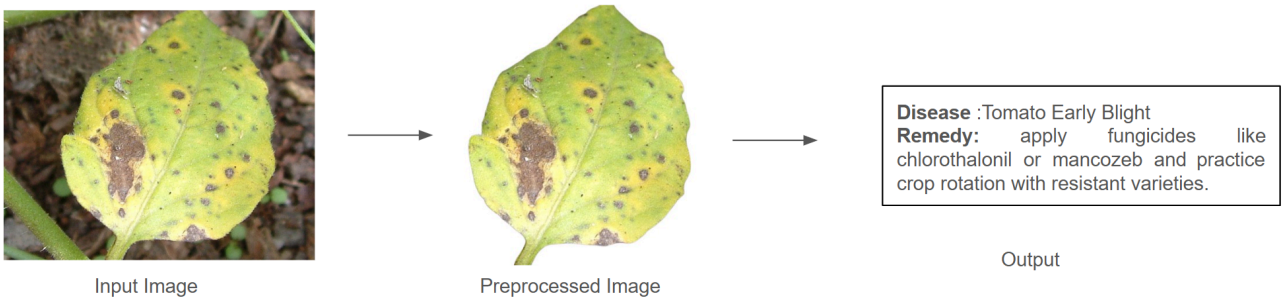

Figure 5.7 Plant disease detection Workflow

This shows the workflow of the system, starting with the input image, where a clear image of a tomato leaf is uploaded to the system. This image undergoes preprocessing, which involves resizing, noise reduction, and normalization to ensure uniformity and compatibility with the deep learning model. The preprocessed image is then passed through the trained model, which analyzes it to identify any disease present on the leaf. Finally, the output is displayed, which includes the detected disease is early blight and its suggested remedy is to apply chlorothalonil or adopting crop rotation practices. This end-to-end process ensures accurate disease identification and actionable recommendations for treatment.

Therefore the implementation section outlines the workflow, from input image preprocessing to disease detection using a trained model, while the results demonstrate accurate identification of diseases like early blight and provide corresponding remedies. This approach ensures practical solutions for effective crop management.

Chapter 6: Conclusion

Conclusion

The plant disease detection project showcases the transformative role of deep learning in addressing key agricultural challenges, particularly the early identification of plant diseases. Leveraging the pre-trained ResNet-50 model fine-tuned for this task, the system achieved impressive accuracy levels of 97.46% on training data and 98.36% on validation data, demonstrating its ability to effectively distinguish between healthy and diseased plant leaves. Advanced preprocessing steps, including resizing, normalization, and data augmentation, played a pivotal role in enhancing model robustness and preventing overfitting. These measures, combined with the use of the PlantVillage dataset, ensured that the system could generalize effectively across diverse scenarios, addressing the limitations of many existing models.This project successfully satisfies all its primary objectives.

The project also integrates MLFlow, which streamlined the experiment tracking process, providing valuable insights into model performance, metrics, and artifacts. This ensured that the best-performing model was saved and optimized for deployment. Compared to existing solutions, this system stands out for its high accuracy, enhanced robustness, and efficient tracking mechanisms, making it a practical and scalable tool for real-world agricultural applications.

The practical benefits of this system are significant. It provides a user-friendly and accessible solution for farmers to detect diseases early, thereby mitigating crop losses and improving yield. Additionally, by offering the potential for future integration of treatment recommendations, the system could revolutionize plant disease management by not only identifying the issue but also guiding farmers on the next steps.

In the broader context, this project illustrates the growing role of artificial intelligence and machine learning in transforming agriculture. It sets the foundation for smarter farming practices, where technology minimizes the labor and resource intensity of traditional methods. As more data becomes available and models continue to evolve, the potential for AI-driven agricultural tools to enhance food security and sustainability becomes even clearer. This project is a significant step toward that future, making farming more efficient, productive, and resilient to challenges like plant disease outbreaks.

Chapter 7: Future Enhancements

To further improve the plant disease detection system, several key enhancements can be implemented.

- **Model Improvement**

  Exploring more advanced models like DenseNet or EfficientNet could improve performance, especially in distinguishing subtle plant diseases. Fine-tuning pretrained models on plant-specific datasets or transferring learning from agricultural-related domains would help the system better recognize plant diseases.
- **Dataset Expansion and Augmentation**

  Expanding the dataset to include additional crop types and varying environmental conditions would increase the model's generalization capability. Incorporating diverse lighting, weather, and seasonal data can enhance robustness. Additionally, addressing class imbalances by oversampling or adjusting class weights during training could improve fairness and accuracy.
- **Real-Time Performance and Optimization**

  Optimizing the system for deployment on mobile devices and edge hardware is essential for real-time disease detection. Converting the model to TensorFlow Lite and using model compression techniques could reduce latency and resource requirements. Faster inference times can be achieved by exploring model distillation techniques that preserve accuracy while reducing the model size.
- **Deployment and User Interface**

  A mobile app could be developed to enable farmers to take pictures and receive instant disease diagnoses, along with treatment suggestions. Integrating the system with IoT devices like drones or sensors would allow for large-scale, real-time monitoring of crops. Furthermore, adding a feedback loop for continuous learning from user inputs could help improve the system over time.
- **Cross-Platform Support**

  Deploying the model on the cloud would allow for greater scalability and accessibility, enabling farmers to use the system remotely. Multilingual support would ensure the system is accessible to a global audience, broadening its impact across different regions.

These enhancements would make the plant disease detection system more accurate, scalable, and user-friendly, ultimately improving its effectiveness in real-world applications and supporting sustainable farming practices.

Bibliography

[1] Computer Vision-Based Automated Detection and Severity Grading of Rice Spot Disease for Precision Management, M. Kumar and I. Sethi, IEEE International Conference on Interdisciplinary Approaches in Technology and Management for Social Innovation (IATMSI), Gwalior, India, 2024, pp. 1-4, doi: 10.1109/IATMSI60426.2024.10503168

[2] Vision Based Plant Leaf Disease Detection and Recognition Model Using Machine Learning Techniques (R. Sathya, S. Senthil Vadivu, S. Ananthi, V. C. Bharathi and G. Revathy) 7th International Conference on Electronics, Communication and Aerospace Technology (ICECA), Coimbatore, India, 2023, pp. 458-464, doi: 10.1109/ICECA58529.2023.10395620

[3] A Bibliometric and Word Cloud Analysis on the Role of IoT in Agricultural Plant Disease Detection (Rutuja Rajendra Patil, Sumit Kumar, Ruchi Rani, Poorva Agrawal, Sanjeev Kumar Pippal) Applied System Innovation, vol. 6, no. 1, pp. 27, 2023. DOI: 10.3390/asi6010027

[4] FieldPlant: A Dataset of Field Plant Images for Plant Disease Detection and Classification With Deep Learning (Emmanuel Moupojou, Appolinaire Tagne, Florent Retraint, Anicet Tadonkemwa, Dongmo Wilfried, Hyppolite Tapamo, Marcellin Nkenlifack) IEEE Access (2023), this study introduces FieldPlant, a dataset for realistic plant disease detection in field conditions. DOI: 10.1109/ACCESS.2023.3263042

[5] Apple Leaf Disease Detection Using Collaborative ML/DL and Artificial Intelligence Methods: Scientometric Analysis (Anupam Bonkra, Pramod Kumar Bhatt, Joanna Rosak-Szyrocka) International Journal of Environmental Research and Public Health (2023), this paper reviews AI-driven methods for detecting apple leaf diseases. DOI: 10.3390/ijerph20043222

[6] Pepper bell leaf disease detection and classification using optimized convolutional neural network (Hassan Mustafa, Muhammad Umer, Umair Hafeez, Ahmad Hameed, Ahmed Sohaib, Saleem Ullah & Hamza Ahmad Madni) Multimed Tools Appl 82, 12065–12080 (2023). https://doi.org/10.1007/s11042-022-13737-8

[7] A Systematic Review on the Detection and Classification of Plant Diseases Using Machine Learning (Deepkiran Munjal, Laxman Singh, Mrinal Pandey, Sachin Lakra) IJSI vol.11, no.1 2023: pp.1-25. https://doi.org/10.4018/IJSI.315657

[8] PLDD—A Deep Learning-Based Plant Leaf Disease Detection (R. K. Lakshmi and N. Savarimuthu) IEEE Consumer Electronics Magazine, vol. 11, no. 3, pp. 44-49, 1 May 2022, doi: 10.1109/MCE.2021.3083976

[9] Vision-Based Wilted Plant Detection (J. Madake, S. Shinde, S. Singh, S. Talwekar, S. Bhatlawande and S. Shilaskar) IEEE Conference on Interdisciplinary Approaches in Technology and Management for Social Innovation (IATMSI), Gwalior, India, 2022, pp. 1-6, doi: 10.1109/IATMSI56455.2022.10119325.

[10] Detection and classification of chilli leaf disease using a squeeze-and-excitation-based CNN model (B.Nageswararao Naik, R.Malmathanraj, P. Palanisamy) Ecological Informatics 69(6):101663 DOI:10.1016/j.ecoinf.2022.101663

[11] Plant Disease Detection Using Deep Convolutional Neural Network (J. Arun Pandian, V. Dhilip Kumar, Oana Geman, Mihaela Hnatiuc, Muhammad Arif, and K. Kanchana Devi) Applied Sciences, vol. 12, no. 14, pp. 6982, 2022. DOI: 10.3390/app12146982

[12] Plant Disease Detection using AI based VGG-16 Model (A. A. Alatawi, S. M. Alomani, N. I. Alhawiti, and M. Ayaz) International Journal of Advanced Computer Science and Applications (IJACSA), vol. 13, no. 4, pp. 477–482, 2022. DOI: 10.14569/IJACSA.2022.0130484.

[13] Investigation on Object Detection Models for Plant Disease Detection Framework (K. L. R and N. Savarimuthu) IEEE 6th International Conference on Computing, Communication and Automation (ICCCA), Arad, Romania, 2021, pp. 214-218, doi: 10.1109/ICCCA52192.2021.9666441.

[14] Disease Detection in Apple Leaves Using Deep Convolutional Neural Network (Prakhar Bansal, Rahul Kumar and Somesh Kumar) Bansal, P.; Kumar, R.; Kumar, S. Disease Detection in Apple Leaves Using Deep Convolutional Neural Network. Agriculture 2021, 11, 617. https://doi.org/10.3390/agriculture11070617

[15] Paddy Leaf Disease Detection Using an Optimized Deep Neural Network (Sankaranarayanan Nalini) Published in Computers, Materials & Continua (CMC), Volume 68, Issue 1, February 2021, with DOI 10.32604/cmc.2021.012431

[16] Identification of disease using deep learning and evaluation of bacteriosis in peach leaf (Saumya Yadav, Neha Sengar, Akriti Singh, Anushikha Singh, Malay Kishore Dutta) Ecological Informatics Volume 61, March 2021, 101247,doi.org/10.1016/j.ecoinf.2021.101247

[17] Identification of Pathological Disease in Plants using Deep Neural Networks (R. Biswas, A. Basu, A. Nandy, A. Deb, R. Chowdhury and D. Chanda) Taiwan 2nd International Conference on Computing, Analytics and Networks (Indo-Taiwan ICAN), Rajpura, India, 2020, pp. 45-48, doi: 10.1109/Indo-TaiwanICAN48429.2020.9181339.

[18] SVM-Based Detection of Tomato Leaves Diseases (Usama Mokhtar, Nashwa El Bendary, Aboul Ella Hassenian, E. Emary, Mahmoud A. Mahmoud, Hesham Hefny & Mohamed F. Tolba) In: Filev, D., et al. Intelligent Systems 2019. Advances in Intelligent Systems and Computing, vol 323. Springer, Cham. https://doi.org/10.1007/978-3-319-11310-4_55

[19] A Computer Vision System for Guava Disease Detection and Recommended Curative Solution Using Deep Learning Approach (A. S. M. Farhan Al Haque, R. Hafiz, M. A. Hakim and G. M. Rafiqul Islam) 22nd International Conference on Computer and Information Technology (ICCIT), Dhaka, Bangladesh, 2019, pp. 1-6, doi: 10.1109/ICCIT48885.2019.9038598.

[20] Plant Disease Detection and Classification by Deep Learning (Muhammad Hammad Saleem, Johan Potgieter and Khalid Mahmood Arif) Plants (Basel). 2019 Oct 31;8(11):468. doi: 10.3390/plants8110468. PMID: 31683734; PMCID: PMC6918394

[21] Suvaditya Mukherjee “ResNet 50 Model Architecture” Source:Towards Data Science August 18 , 2022 Available: https://towardsdatascience.com/the-annotated-resnet-50-a6c536034758

[22] D. P. Hughes and M. Salathé, "PlantVillage Dataset," Kaggle, 2018. Available: https://www.kaggle.com/datasets/emmarex/plantdisease

[23] https://www.datacamp.com/tutorial/introduction-to-convolutional-neural-networks-cnns

[24] https://www.ibm.com/think/topics/machine-learning

[25] https://mlflow.org/docs/latest/index.html

[26] https://www.geeksforgeeks.org/opencv-overview

[27] https://builtin.com/data-science/pytorch-vs-tensorflow