Tomato Plant Disease Detection from Leaf Images

Industrial Internship Project report submitted in partial fulfillment

of the Requirements for the Award of the Degree of

BACHELOR OF TECHNOLOGY

in

CSE-(ARTIFICIAL INTELLIGENCE & MACHINE LEARNING)

By

V. Ravikumar	**208X1A4245**
U. Renuka Bhavani	**208X1A4261**
K. Sudheer	**208X1A4222**
MD. Azharulla	**208X1A4231**

Under the Guidance of

Mrs. M. Alekhya M.Tech.

Assistant Professor

DEPARTMENT OF

CSE-(ARTIFICIAL INTELLIGENCE & MACHINE LEARNING)

KALLAM HARANADHAREDDY INSTITUTE OF TECHNOLOGY

(AUTONOMOUS)

Approved by AICTE- New Delhi, Accredited by NAAC with 'A' Grade

Permanently Affiliated to JNTUK, Kakinada

NH-16, Chowdavaram, Guntur

April 2024

KALLAM HARANADHAREDDY INSTITUTE OF TECHNOLOGY

(AUTONOMOUS)

DEPARTMENT OF

CSE-(ARTIFICIAL INTELLIGENCE & MACHINE LEARNING)

CERTIFICATE

This is to certify that the Industrial Internship project report entitled **Tomato Plant Disease Detection from Leaf Images** being submitted by

V. Ravikumar	**208X1A4245**
U. Renuka Bhavani	**208X1A4261**
K. Sudheer	**208X1A4222**
MD. Azharulla	**208X1A4231**

in partial fulfillment for the award of the Degree of Bachelor of Technology in CSE-(Artificial Intelligence & Machine Learning) to the Kallam Haranadhareddy Institute of Technology is a record of bonafied work carried out under my guidance and supervision.

GUIDE	**HOD**
(Mrs. M. ALEKHYA M.Tech.)	**(Dr. G. J. SUNNY DEOL M.Tech., PhD.)**
Assistant Professor	**Professor**

EXTERNAL EXAMINER

DECLARATION

I hereby declare that the Industrial Internship project dissertation entitled **Tomato Plant Disease Detection from Leaf Images** submitted for the B.Tech. Degree is my original work and the dissertation has not formed the basis for the award of any degree, associate ship, fellowship or any other similar titles.

Place: Guntur

Date:

Veeramalla Ravikumar(208X1A4245)
Upputuru Renuka Bhavani(208X1A4261)
Kondameedi Sudheer(208X1A4222)
Mohammed Azharulla (208X1A4231)

Certificate from Intern Organization

This is to certify that **VEERAMALLA RAVIKUMAR (208X1A4245), UPPUTURU RENUKA BHAVANI (208X1A4261), KONDAMEEDI SUDHEER (208X1A4222), MOHAMMED AZHARULLA (208X1A4231)** of Kallam Haranadhareddy Institute of Technology underwent industrial internship in **SmartInternz** from 05-02-2024 to 30-04-2024. The overall performance of the intern during their internship is found to be **Satisfactory.**

Google for Developers

Smart Internz

ANDHRA PRADESH STATE COUNCIL OF HIGHER EDUCATION

(A Statutory Body of the Government of A.P)

CERTIFICATE OF COMPLETION

This is to certify that Ms./Mr. **Veeramalla Ravikumar** of **Computer Science And Engineering (Artificial Intelligence And Machine Learning)** with Registered Hall ticket no. **208X1A4245** under **Kallam Haranadhareddy Institute of Technology** of **JNTUK** has successfully completed Long-Term Internship of 240 hours (6 months) on **Artificial Intelligence & Machine Learning** Organized by **SmartBridge Educational Services Pvt. Ltd.** in collaboration with **Andhra Pradesh State Council of Higher Education.**

Certificate ID: **EXT-APSCHE_AIML-14328**

Date: 18-Apr-2024

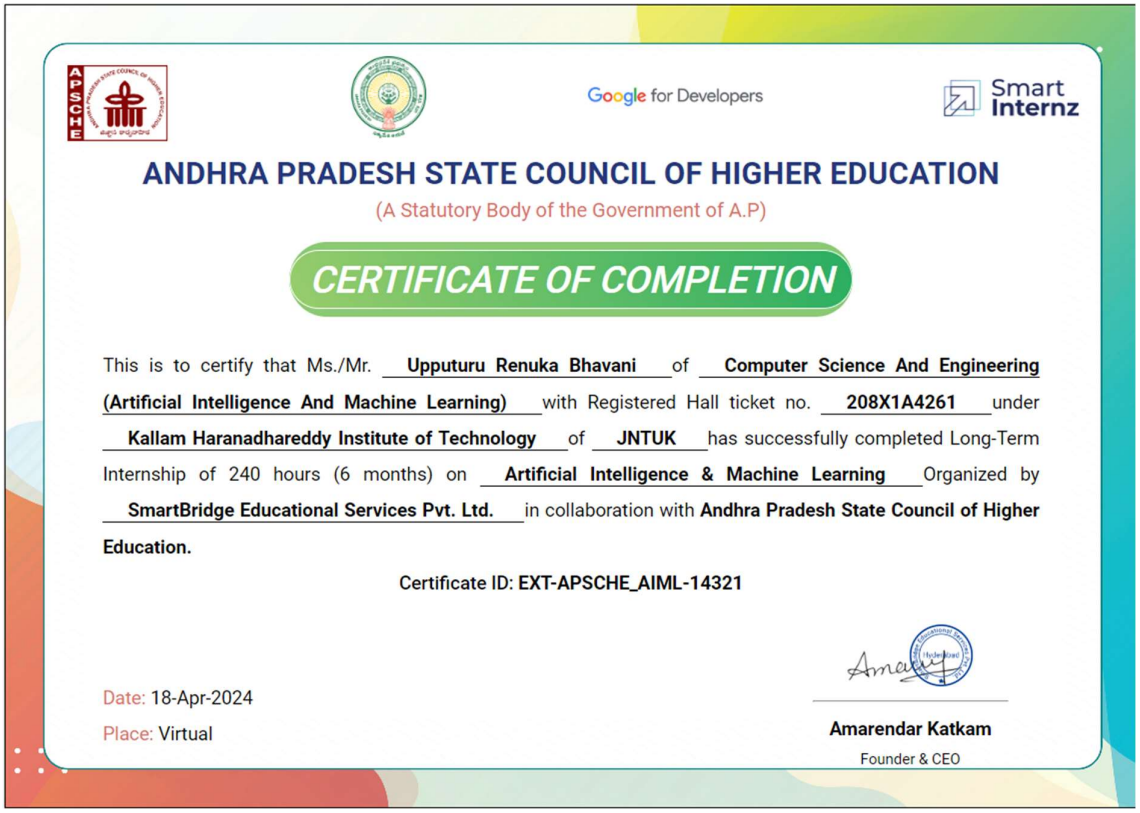
Place: Virtual

Amarendar Katkam

Founder & CEO

Certification ID:

https://apsche.smartinternz.com/certificate/student/fda6baab618b39821b678bd52ff26428



Google for Developers

Smart Internz

ANDHRA PRADESH STATE COUNCIL OF HIGHER EDUCATION

(A Statutory Body of the Government of A.P)

CERTIFICATE OF COMPLETION

This is to certify that Ms./Mr. **Upputuru Renuka Bhavani** of **Computer Science And Engineering (Artificial Intelligence And Machine Learning)** with Registered Hall ticket no. **208X1A4261** under **Kallam Haranadhareddy Institute of Technology** of **JNTUK** has successfully completed Long-Term Internship of 240 hours (6 months) on **Artificial Intelligence & Machine Learning** Organized by **SmartBridge Educational Services Pvt. Ltd.** in collaboration with **Andhra Pradesh State Council of Higher Education.**

Certificate ID: **EXT-APSCHE_AIML-14321**

Date: 18-Apr-2024

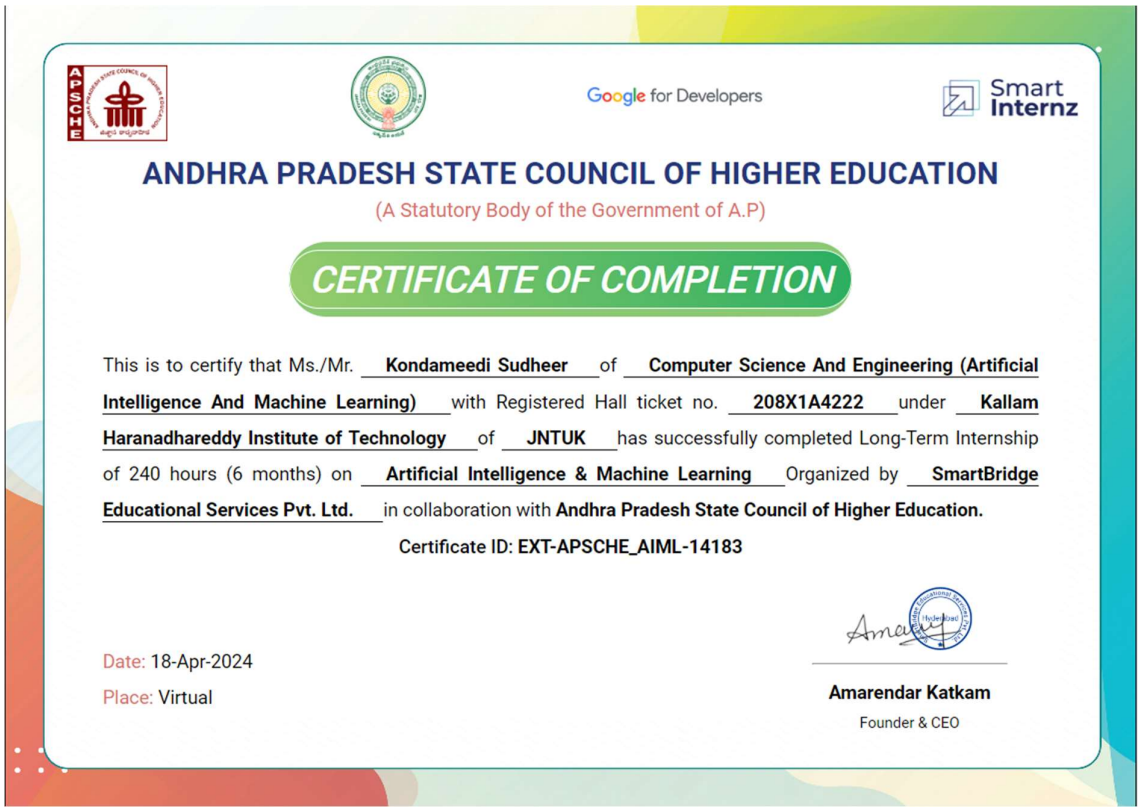
Place: Virtual

Amarendar Katkam

Founder & CEO

Certification ID:

https://apsche.smartinternz.com/certificate/student/08fcb5ead4e963a6f0bbdbc971f4a3ee



ANDHRA PRADESH STATE COUNCIL OF HIGHER EDUCATION

(A Statutory Body of the Government of A.P)

CERTIFICATE OF COMPLETION

This is to certify that Ms./Mr. **Kondameedi Sudheer** of **Computer Science And Engineering (Artificial Intelligence And Machine Learning)** with Registered Hall ticket no. **208X1A4222** under **Kallam Haranadhareddy Institute of Technology** of **JNTUK** has successfully completed Long-Term Internship of 240 hours (6 months) on **Artificial Intelligence & Machine Learning** Organized by **SmartBridge Educational Services Pvt. Ltd.** in collaboration with **Andhra Pradesh State Council of Higher Education.**

Certificate ID: **EXT-APSCHE_AIML-14183**

Date: 18-Apr-2024

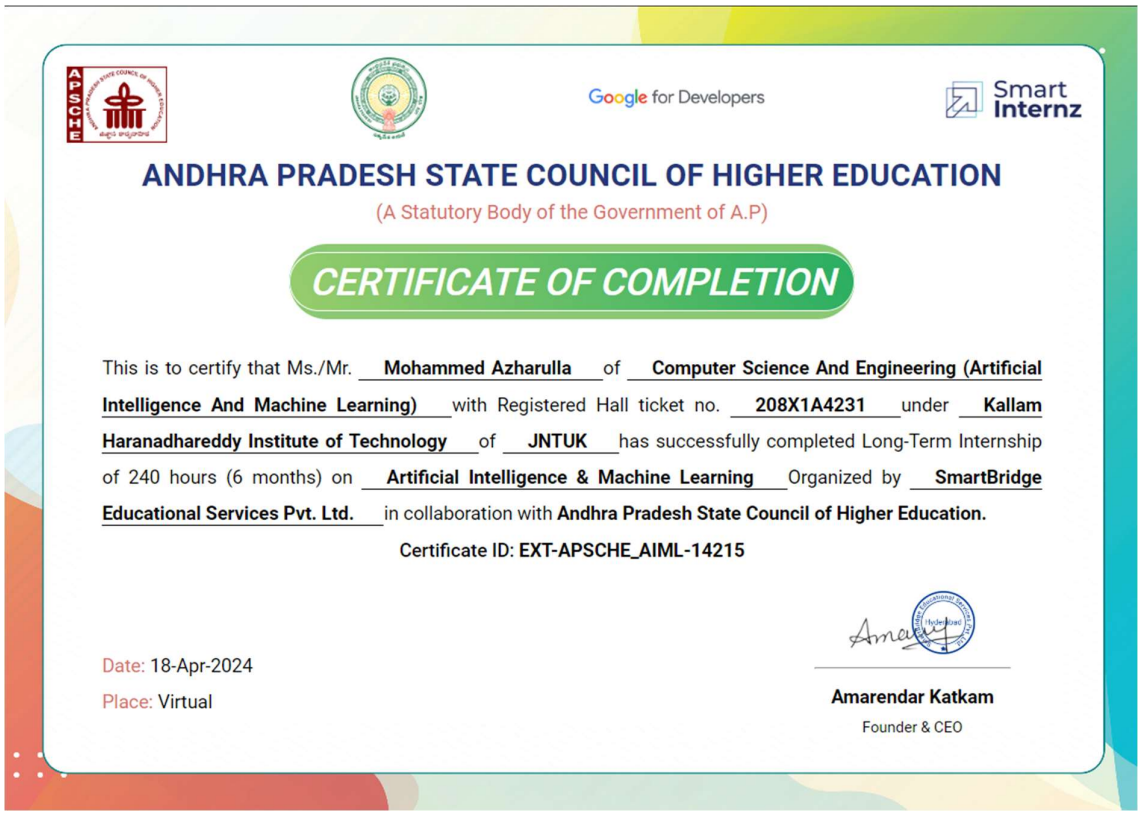
Place: Virtual

Amarendar Katkam

Founder & CEO

Certification ID:

https://apsche.smartinternz.com/certificate/student/37bd8b4bb1cea47935613348ad16c660



ANDHRA PRADESH STATE COUNCIL OF HIGHER EDUCATION

(A Statutory Body of the Government of A.P)

CERTIFICATE OF COMPLETION

This is to certify that Ms./Mr. **Mohammed Azharulla** of **Computer Science And Engineering (Artificial Intelligence And Machine Learning)** with Registered Hall ticket no. **208X1A4231** under **Kallam Haranadhareddy Institute of Technology** of **JNTUK** has successfully completed Long-Term Internship of 240 hours (6 months) on **Artificial Intelligence & Machine Learning** Organized by **SmartBridge Educational Services Pvt. Ltd.** in collaboration with **Andhra Pradesh State Council of Higher Education.**

Certificate ID: **EXT-APSCHE_AIML-14215**

Date: 18-Apr-2024

Place: Virtual

Amarendar Katkam

Founder & CEO

Certification ID:

https://apsche.smartinternz.com/certificate/student/8be904ad045c578053fc6052578f9324

ACKNOWLEDGMENT

We profoundly grateful to express our deep sense of gratitude and respect towards **Sri. KALLAM MOHAN REDDY**, **Chairman, KHIT, Guntur** for his precious support in the college.

We are thankful to **Dr. M. UMA SANKAR REDDY**, Director, KHIT, GUNTUR for his encouragement and support for the completion of the project.

We are much thankful to **Dr. B. SIVA BASIVI REDDY**, Principal, KHIT, GUNTUR for his support during and until the completion of the project.

We are greatly indebted to **Dr. G. J. SUNNY DEOL**, Professor, & Head of the department, CSE-Artificial Intelligence and Machine Learning, KHIT, GUNTUR for providing the laboratory facilities fully as and when required and for giving us the opportunity to carry the project work in the college during Industry Internship.

We express our profound gratitude to our Internal Guide, **Mrs. M. Alekhya**, Assistant Professor, for her invaluable suggestions, guidance, and insightful ideas at every stage. Her contributions were instrumental in the successful completion of our project.

We extend our sincere appreciation to our Coordinator, **Mr. M. Chennakesavarao**, Associate Professor, along with other esteemed Faculty Members, Support staff, Professors, and the Head of the department for their invaluable support, guidance, and constructive input throughout our project journey. Their collective efforts were pivotal to the successful completion of our project.

Place: Guntur

Date:

Veeramalla Ravikumar(208X1A4245)
Upputuru Renuka Bhavani(208X1A4261)
Kondameedi Sudheer(208X1A4222)
Mohammed Azharulla (208X1A4231)

ABSTRACT

Tomato cultivation is essential for global food security, yet it faces significant challenges from various plant diseases. Small-scale farmers often struggle with timely and accurate disease management due to resource limitations. In this study, we present an advanced deep learning approach for tomato plant disease detection using Convolutional Neural Networks (CNNs) and the ResNet152V2 architecture. Our goal is to enhance disease detection accuracy and empower farmers with accessible tools for crop productivity and health management, contributing to agricultural sustainability.

We begin by discussing the limitations of existing systems, which mainly rely on Machine Learning techniques such as Support Vector Machines (SVM) and Random Forests. These systems face challenges in multi-class classification and understanding complex features in leaf images, leading to suboptimal performance. Our proposed methodology addresses these challenges by leveraging Transfer Learning with the ResNet152V2 architecture. This approach accelerates training, reduces computational resources, mitigates overfitting, and improves the model's ability to generalize well to unseen disease patterns.

Our experimental results showcase the effectiveness of the proposed system, achieving an accuracy of 0.9110 and a loss of 0.2918 during testing. These results demonstrate the system's capability to accurately identify and classify tomato plant diseases, surpassing traditional methods. Additionally, we discuss the practical implications of our findings, including the development of a user-friendly software interface for easy deployment and usage by farmers, home gardeners and stakeholders.

This study contributes to the advancement of agricultural technology by harnessing the power of deep learning to address critical challenges in crop health management. The proposed system not only enhances disease detection accuracy but also promotes sustainable agricultural practices, ultimately benefiting farmers and contributing to global food security.

TABLE OF CONTENTS

Abstract ..... i.

Table of contents ..... ii-iii.

List of figures ..... iv-v.

List of tables ..... vi.

1 Introduction ..... 1.

1.1 Problem statement ..... 2.

1.2 Scope of system ..... 2-3.

1.3 Objective of system ..... 4-5.

1.4 Unique features of the system ..... 5.

1.5 Document overview ..... 5-6.

2 Literature Review ..... 7-8.

2.1 Existing system ..... 9-10.

2.2 Proposed system ..... 10-11.

2.3 Project related work ..... 11-12.

3 Feasibility study ..... 13.

3.1 Technical feasibility ..... 13.

3.1 Operational feasibility ..... 13-14.

3.1 Economic feasibility ..... 14.

4 Software requirement specifications ..... 15.

4.1 Functional requirements ..... 15-16.

4.2 Non-Functional requirements ..... 16-17.

4.2.1 Performance requirements ..... 17-18.

4.2.2 Software requirements 18-19

4.2.2.1 Introduction to Java/Python, etc ..... 19-20.

5 System Design ..... 21.

5.1 Requirements modelling ..... 21.

5.1.1 Class diagram ..... 21-22.

5.1.2 Use case diagram ..... 22-24.
5.1.3 Sequence diagram ..... 24-25.
5.1.4 Activity diagram ..... 25-27.
5.2 Design concepts and constraints ..... 27.
5.2.1 Design concepts ..... 27-28.
5.2.2 Constraints ..... 28.
6 System implementation & Methodologies ..... 29.
6.1 Methodologies ..... 29.
6.1.1 Architectural Design ..... 29-30.
6.1.3 Proposed Architecture in Designing ..... 30-31.
6.1.3 Algorithms Design ..... 31-32.
6.1.4 Module design specifications ..... 32-33.
6.1 Development environment and tools ..... 33-34.
6.2 Coding ..... 34.
6.3.1 Source code ..... 34-52.
7 Testing ..... 53.
7.1 Introduction to testing ..... 53.
7.1.1 Testing strategies ..... 53-55.
7.1.2 Testing Metrics ..... 55-58.
7.1.3 Model Performance Testing ..... 58.
7.1.4 Integration Testing ..... 58.
7.1.5 End-to-End Testing 58-59
7.1.6 Test case ..... 59-67.
8 Result ..... 68.
8.1 Screens and reports ..... 68-73.
8.2 User manual ..... 74-75.
9 Limitations ..... 76.
10 Conclusion & future work ..... 77-78.
11 References ..... 79.

LIST OF FIGURES

Figure 5.1.1 Class diagram .....21.

Figure 5.1.2 Use case diagram .....23.

Figure 5.1.3 Sequence diagram.....24.

Figure 5.1.4 Activity diagram .....25.

Figure 6.1.1 Architecture diagram .....29.

Figure 6.2 Proposed Architecture diagram .....30.

Figure 6.3.1.1 Image Count in Train.....51.

Figure 6.3.1.2 Image Count in Validation .....51.

Figure 6.3.1.3 Data Distribution .....52.

Figure 6.3.1.4 Accuracy and Loss in Training.....52.

Figure 6.3.1.4 Accuracy and Loss in Testing .....52.

Figure 7.2.1 Home Page.....59.

Figure 7.2.2 Results of Testcase-1 .....60.

Figure 7.2.3 Result of Testcase-2.....61.

Figure 7.2.4 Result of Testcase - 3.....62.

Figure 7.2.5 Predict Page .....63.

Figure 7.2.6 Result of Testcase - 4.....63.

Figure 7.2.7 Result of Testcase -5 .....64.

Figure 7.2.8 Result of Testcase -6 .....65.

Figure 7.2.9 Upload file for detection.....66.

Figure 7.2.10 Result(1) of Testcase -7.....67.

Figure 7.2.11 Result(2) of Testcase -7.....67.

Figure 7.2.12 Result(3) of Testcase -7.....67.

Figure 8.1.1 Upload image file .....69.

Figure 8.1.2 Upload Diseased file.....69.

Figure 8.1.3 Predict Result.....70.

Figure 8.1.4 Upload Normal file.....70.

Figure 8.1.5 Predict Result.....71.

Figure 8.1.6 Accuracy and Loss Comparision.....71.

Figure 8.1.7 Confusion Matrix.....72.

Figure 8.1.8 Model Evaluation Metrics .....72.

LIST OF TABLES

Table 7.1 Writing test case for home page .....59.

Table 7.2 Writing test case for about page .....61.

Table 7.3 Writing test case for contact page .....62.

Table 7.4 Writing test case for browser .....63.

Table 7.5 Writing test case for mismatched file .....64.

Table 7.6 Writing test case for no file selection.....65.

Table 7.6 Writing test case for detection of disease.....66.

Table 8.1 Reports of test cases .....73.

1 INTRODUCTION

1 Introduction of the Project

Tomato plant diseases represent a significant threat to global agriculture, impacting crop yield and food security. Early detection and accurate diagnosis of these diseases are essential for effective management and prevention of economic losses for farmers. Traditional methods of disease identification rely heavily on manual inspection, which can be time-consuming and prone to errors.

In recent years, the integration of deep learning techniques with agricultural technology has shown great promise in revolutionizing disease detection and management. Convolutional Neural Networks (CNNs), a subset of deep learning models, have emerged as powerful tools for image analysis and pattern recognition. By leveraging CNNs, we can develop efficient systems capable of accurately detecting tomato plant diseases from leaf images.

This project focuses on harnessing the capabilities of deep learning, specifically using the ResNet152V2 architecture, to enhance disease detection accuracy in tomato plants. The ResNet152V2 model, known for its deep architecture and feature extraction capabilities, is well-suited for identifying complex disease patterns in tomato leaves. Additionally, we employ Transfer Learning to leverage pre-trained features, optimizing model performance and reducing training time.

The primary objective of this research is to empower farmers with accessible and accurate tools for crop health management. We aim to develop a user-friendly software interface that allows farmers to upload leaf images, receive real-time disease predictions, and access actionable recommendations. By integrating advanced deep learning techniques with practical usability, our project aims to bridge the gap between cutting-edge technology and agricultural needs.

Through rigorous experimentation and validation, we have achieved a commendable accuracy of 0.9110 and a loss of 0.2918 during testing, demonstrating the effectiveness of our approach. These results highlight the potential of deep learning in transforming crop health management and contributing to sustainable agricultural practices.

It represents a significant step towards empowering farmers with AI-driven solutions for early disease detection, ultimately improving crop productivity, reducing losses, and ensuring food security in tomato plants.

1.1 Problem Statement:

Despite advancements in agricultural technology, effective and timely detection of tomato plant diseases remains a challenge, particularly for small-scale farmers with limited resources. Traditional methods of disease identification often rely on manual inspection, which can be subjective, time-consuming, and prone to errors. In this context, deep learning, specifically convolutional neural networks (CNNs), offers a promising solution for accurate and efficient disease detection from leaf images. However, several challenges persist:

Limited Annotated Data: Annotated datasets for tomato plant disease detection are often limited in size and diversity, which can hinder the performance of deep learning models. There is a critical need for larger and more diverse datasets encompassing various disease patterns and environmental conditions to train robust and generalizable models.

Real-Time Detection: While deep learning models can analyze images quickly, there is a demand for real-time detection systems that can provide immediate feedback to farmers and stakeholders. Real-time detection is crucial for timely intervention and disease management, especially in dynamic agricultural settings.

Interpretability: Deep learning models are often perceived as "black boxes," making it challenging to interpret their decisions, which is essential for user acceptance and trust. Developing explainable AI techniques is imperative to enhance the interpretability of deep learning models for tomato plant disease detection, enabling farmers to understand and act upon model predictions effectively.

Generalization: Deep learning models trained on specific datasets may struggle to generalize well to diverse environmental conditions, crop varieties, and disease manifestations. Ensuring model generalizability across different agricultural settings and disease scenarios is vital for reliable and accurate disease detection.

Integration with Farming Practices: Seamless integration of deep learning-based disease detection systems into existing farming practices and workflows is crucial for practical utility and adoption. Efforts should focus on developing user-friendly interfaces empower farmers with accessible, actionable insights, and recommendations for crop health management.

1.2 Scope of System

The scope of Tomato Plant Disease Detection can be outlined as follows:

1. **Disease Identification and Classification:** The project aims to accurately identify and

classify various diseases affecting tomato plants based on leaf images. This includes common diseases such as bacterial spot, early blight, late blight, leaf mold, septoria leaf spot, and others. The scope covers developing algorithms and models capable of distinguishing between different disease types with high accuracy.

2. **Deep Learning Techniques**: The project scope includes the implementation of advanced deep learning techniques, particularly Convolutional Neural Networks (CNNs) like ResNet152V2, for disease detection. This involves training the model on a diverse dataset of labeled tomato leaf images to enable robust disease classification and prediction.
3. **Image Processing and Preprocessing:** The scope encompasses image preprocessing techniques to enhance the quality and clarity of input images. This may involve noise reduction, contrast enhancement, and feature extraction to improve the model's accuracy in detecting subtle disease symptoms.
4. **Real-time Disease Detection:** The project aims to develop a system capable of real-time disease detection from uploaded leaf images. This includes building a user-friendly interface for farmers or users to upload images, receive instant diagnostic results, and access recommendations for disease management.
5. **Model Optimization and Performance:** The scope involves optimizing the deep learning model for enhanced performance in terms of accuracy, speed, and scalability. Techniques such as transfer learning, class weighting, and model fine-tuning will be explored to achieve optimal results.
6. **Deployment and Accessibility:** The project scope extends to developing a deployable solution that can be easily accessed and utilized by stakeholders, including farmers, agricultural experts, and researchers. This includes considerations for platform compatibility, scalability, and usability.
7. **Validation and Testing:** The scope covers rigorous testing and validation of the developed model to ensure its reliability, generalization across different environmental conditions, and effectiveness in real-world disease detection scenarios.
8. **Future Enhancements:** The project's scope also includes provisions for future enhancements and iterations, such as incorporating additional disease classes, improving model interpretability, integrating with IoT devices for automated image capture, and leveraging cloud computing for scalable processing capabilities.

1.3 Objective of System

The objectives of Tomato Plant Disease Detection system can be outlined as follows:

1. **Developing Accurate Disease Detection Models:** Create deep learning models, such as Convolutional Neural Networks (CNNs) using architectures like ResNet152V2, to accurately identify and classify various diseases affecting tomato plants based on leaf images.
2. **Implementing Advanced Image Processing Techniques:** Utilize image processing and preprocessing techniques to enhance the quality of input images, improve feature extraction, and reduce noise for more precise disease detection.
3. **Enabling Real-time Disease Diagnosis**: Build a user-friendly interface that allows users, such as farmers and agricultural experts, to upload leaf images for real-time disease diagnosis, providing instant results and recommendations for disease management.
4. **Optimizing Model Performance:** Optimize the deep learning models through techniques like transfer learning, class weighting, and model fine-tuning to achieve high accuracy, speed, and scalability in disease detection.
5. **Ensuring Robustness and Generalization:** Validate the developed models rigorously to ensure their robustness, generalization across different environmental conditions, and effectiveness in detecting a wide range of tomato plant diseases.
6. **Deploying Accessible Solutions:** Develop deployable solutions that are accessible and user-friendly, catering to stakeholders' needs and facilitating widespread adoption for effective crop health management.
7. **Facilitating Future Enhancements:** Lay the groundwork for future enhancements by considering provisions for incorporating additional disease classes, improving model interpretability, integrating with IoT devices, and leveraging cloud computing for scalability.
8. **Contributing to Agricultural Innovation:** Contribute to the advancement of agricultural technology by leveraging AI-powered solutions to address critical challenges in crop disease detection, ultimately empowering farmers and enhancing crop productivity.

1.4 Unique features of the System

The proposed system for tomato plant disease detection system from leaf images showcase its innovative approach and effectiveness in agricultural technology. Utilizing the ResNet152V2 architecture within a Convolutional Neural Network (CNN) framework, the system autonomously learns intricate patterns and features specific to tomato plant diseases. This advanced deep learning model enables precise disease identification and classification, contributing to improved crop health management and productivity.

A notable innovation in the system is its integration of Transfer Learning, enhancing model performance by leveraging pre-trained features in ResNet152V2. This accelerates the training process and improves the system's accuracy and reliability in disease detection. Additionally, the system employs a real-time detection mechanism, providing instant feedback on disease presence in tomato plants. Combined with a user-friendly interface, developed using Flask, this system ensures accessibility and usability for farmers and stakeholders, empowering them with efficient tools for proactive disease management and sustainable agriculture practices.

1.5 Document Overview:

Organizing a thesis on Tomato Plant Disease Detection can follow a standard structure. Here follows:

Chapter 2	Literature survey reviews existing research and literature relevant to the project.
Chapter 3	Feasibility study evaluates the project's practicality, cost-effectiveness, and potential success.
Chapter 4	Software Requirements Specifications (SRS) document outlines the functional and non-functional requirements of the system.
Chapter 5	System modeling involves creating architectural diagrams and system design representations.
Chapter 6	System implementation refers to the actual development and coding of the system.
Chapter 7	Testing phase involves verifying and validating the functionality and

performance of the system.

Chapter 8 Results chapter presents the outcomes, findings, and performance metrics of the implemented system.

Chapter 9 Limitations section addresses constraints, challenges, and drawbacks encountered during the project.

Chapter 10 Conclusion and future work summarize the project's achievements, propose future enhancements, and outline ongoing or future research directions.

2 LITERATURE SURVEY

In this chapter, presents literature survey of traditional plant disease detection approaches based on computer vision technologies are commonly utilized to extract the texture, shape, colour, and other features of disease spots. This method has a low identification efficiency because it relies on an extensive expert understanding of agricultural illnesses. Many academics have conducted significant research based on deep learning technology to increase the accuracy of plant disease detection in recent years, thanks to the fast growth of artificial intelligence technology. The majority of existing techniques to plant disease analysis are based on disease classification. Here's a general outline of what such a literature survey might include:

A. **A Survey on Supervised Convolutional Neural Network and Its Major Applications; D. T. Mane and U. V. Kulkarni**

The rise of deep learning has ushered in a new era of machine learning, advancing towards the goal of Artificial Intelligence. Artificial Neural Networks (ANNs) are biologically inspired algorithms that excel in problem-solving where conventional methods struggle, like in computer vision. They extract features from inputs, whether images or audio signals, and generalize these features for broader applicability. This paper focuses on supervised Convolutional Neural Networks (CNNs), detailing their history, architecture, and outcomes across various domains, showcasing their significant impact.

B. **Tomato crop disease classification Using a Pre-Trained Deep Learning Algorithm; Aravind KR, Raja P, Anirudh R.**

A study utilized VGG16 for classifying tomato diseases with 98.67% accuracy, surpassing Random Forest and K-Nearest Neighbours. The research discusses dataset specifics, algorithm fine-tuning through transfer learning, limitations, and suggests future research areas, including mobile app development for crop disease identification.

C. **Attention Embedded Residual CNN for Disease Detection in Tomato Leaves; Karthik R, Hariharan M, Anand Sundar, Mathikshara Priyanka, Johnson Annie, Menaka R.**

A dataset containing images of tomato leaves with five diseases and healthy leaves was used. The CNN architecture with residual blocks and attention modules achieved 98.3% accuracy in disease detection, surpassing VGG16 and Inception-v3. The paper includes a thorough analysis of different disease classes and visualizations of attention maps.

D. **Research on deep learning in apple leaf disease recognition. Comput Electron Agric; Zhong Yong, Zhao Ming.**

The article explores the application of deep learning technology, specifically a convolutional neural network (CNN), for the recognition of apple leaf diseases. This technology enables automatic identification and classification of diseases from images, offering potential benefits such as improved crop yields and reduced economic losses for apple farmers.

E. **AI-powered banana diseases and pest detection. Plant Methods. 2019; 15:92; Selvaraj MG, Vergara A, Ruiz H, et al.**

The study utilizes a dataset of banana plant images to demonstrate accurate detection of diseases and pests using machine learning techniques. They showcase real-world application through a smartphone app, emphasizing the importance of accessible tools for farmers to monitor and manage crops. The approach's scalability to other crops and regions is highlighted, promoting sustainable agricultural practices.

F. **Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation; Girshick, R., J. Donahue, T. Darrell, and J. Malik.**

The paper emphasizes deep learning's potential in object detection and segmentation, introducing a novel approach that merges convolutional neural networks with region-based processing for enhanced accuracy and efficiency. This R-CNN method has evolved and gained popularity in computer vision and object recognition due to subsequent enhancements and extensions.

G. **Characteristics of tomato plant diseases—a study for tomato plant disease identification; Fuentes A, Yoon S, Youngki H, Lee Y, Park DS.**

Fuentes et al. proposed deep learning for identifying diseases and pests in tomato plants using varied camera resolutions. They employed deep learning meta-architectures and multiple CNN object detectors, along with data expansion and annotation techniques, to enhance training accuracy and reduce false positives. Their system successfully detected nine different pests and diseases from complex scenarios using a large-scale tomato disease dataset for training and testing

H. **A Deep convolutional neural networks for mobile capture device-based crop disease classification in the wild. Comput Electron Agric; Picon A, Alvarez-Gila A, Seitz M, Ortiz-Barredo A, Echazarra J, Johannes.**

The article discusses using deep CNNs for crop disease classification with mobile-captured images. They developed the "DeepPlantPathologist" model, utilizing CNNs to automatically classify crop diseases from field leaf images.

2.1 Existing System

Existing systems in tomato plant disease detection employ a range of techniques, including Convolutional Neural Network (CNN) models that are pivotal in tomato plant disease detection due to their adeptness in image recognition and feature extraction. However, they encounter several limitations. One significant challenge is the propensity for overfitting, particularly when trained on limited or imbalanced datasets. This can lead to inaccurate predictions and reduced generalization capability when confronted with unseen data, undermining the reliability of disease diagnosis. Additionally, the complex architectures and substantial computational requirements of CNN models necessitate high-performance GPUs and specialized infrastructure, translating to high costs and limited accessibility for smaller-scale agriculture operations. These systems often exhibit moderate accuracy and moderate loss, with an example accuracy of 0.7830 and a loss of 3.9738, indicating the need for improvements in model performance and efficiency.

Moreover, the development and deployment of CNN models entail time-consuming processes, from designing intricate architectures to fine-tuning hyperparameters and optimizing training routines. This time investment delays the implementation of timely disease detection solutions, impacting farmers' ability to mitigate crop losses effectively. Furthermore, CNN models may struggle with adapting to new diseases or integrating additional data sources seamlessly, requiring ongoing updates and retraining efforts to maintain their efficacy. Despite these challenges, advancements in transfer learning, data augmentation, and model optimization hold promise in enhancing the performance and scalability of CNN-based solutions for tomato plant disease detection, paving the way for more accurate and accessible agricultural technology solutions.

Disadvantages of Existing System

Existing systems for tomato plant disease detection using deep learning face the following drawbacks:

- Susceptibility to overfitting, particularly with small or imbalanced datasets

- Complex architectures and large datasets require significant computational resources
- High cost and specialized infrastructure for training (e.g., high-performance GPUs)
- Time-consuming process of building models from scratch, with less guaranteed high accuracy.

2.2 Proposed System:

The Proposed system focuses on addressing the limitations of existing methods while harnessing the advantages of deep learning for improved efficiency and usability in tomato plant disease detection.

The proposed system begins with comprehensive data preprocessing, where tomato plant leaf images are standardized and augmented to enhance dataset size and diversity. This step is crucial for optimizing the input data quality for subsequent deep learning model training.

Data collection and annotation are pivotal, involving the gathering of a diverse dataset of annotated leaf images representing various tomato plant diseases. The dataset's balance ensures unbiased model training, avoiding the tendency to favor the majority disease class.

Real-time detection capabilities are a key aspect of our proposed system. We aim to develop a robust real-time detection system capable of processing leaf images quickly and providing immediate feedback to farmers and stakeholders. This real-time functionality is essential for timely intervention and effective crop management.

ResNet152V2 is a deep convolutional neural network architecture known for its depth and performance in image recognition tasks. It utilizes residual connections to address the vanishing gradient problem, enabling the training of very deep networks effectively. In our project, we leverage transfer learning with ResNet152V2, where the pre-trained model's features are used as a starting point for training on tomato plant disease images. This approach accelerates training, reduces overfitting, and improves the model's ability to generalize well to unseen disease patterns.

The proposed method involves preprocessing the tomato leaf images, followed by training the CNN model to classify them into one of ten categories: healthy, yellow leaf curl virus (YLCV), bacterial spot (BS), early blight (EB), leaf mold (LM), spectoria leaf spot (SLS) target spot (TS), two spotted spider mite spot (TSSMS), mosaic virus (MV) and late blight (LB). The model was trained using a dataset of 11000 tomato leaf images. The training was conducted for 20 epochs.

Advantages of the Proposed System

The proposed system offers several advantages over traditional methods of tomato plant disease detection:

- **Accuracy and Loss:** During testing, our system achieved an accuracy of 0.9110 and a loss of 0.2918, showcasing its robustness and reliability in disease detection.
- **Efficiency**: The system's efficiency lies in its ability to process large volumes of image data swiftly, enabling real-time disease detection and management.
- **Automation**: By automating the detection process, the system reduces reliance on manual inspection, leading to faster and more reliable diagnoses.
- **Scalability:** Deep learning models can scale effectively to handle diverse datasets and adapt to evolving disease patterns over time.
- **Cost-effectiveness:** While initial setup costs may be involved, the long-term benefits in terms of improved crop yield.

2.3 Project Related Work:

1. **Data Collection:**
   - Gather a dataset of tomato leaf images encompassing various diseases like Early Blight, Late Blight, Leaf Mold, Septoria Leaf Spot, Spider Mites, etc., along with healthy leaf images.
2. **Data Preprocessing**:
   - Clean the dataset by removing duplicates and low-quality images.
   - Resize all images to a uniform resolution suitable for input to the ResNet152V2 model.
   - Normalize pixel values to a standardized range to enhance model training.
3. **Data Augmentation:**
   - Apply augmentation techniques like rotation, flipping, zooming, and brightness adjustments to augment the dataset, thereby increasing diversity and robustness.
4. **Model Selection:**
   - Choose the ResNet152V2 architecture as the deep learning model for tomato disease

classification due to its proven effectiveness in image recognition tasks.

5. **Model Training:**
   - Split the dataset into training, validation, and test sets.
   - Use transfer learning by leveraging pre-trained weights of ResNet152V2 on a large-scale dataset like ImageNet.
   - Fine-tune the model using the training set and validate its performance using the validation set.
6. **Model Evaluation:**
   - Evaluate the trained ResNet152V2 model on the test set to assess its accuracy, precision, recall, F1 score, and other relevant metrics.
7. **Performance Optimization:**
   - Implement techniques like class weighting, model optimization, and hyperparameter tuning to improve the model's accuracy and generalization ability.
8. **Deployment:**
   - Develop a user-friendly interface, possibly using Flask, to deploy the trained ResNet152V2 model for real-time tomato disease detection from leaf images.
   - Ensure the system's scalability, efficiency, and usability for farmers or agricultural stakeholders.

3 FEASIBILITY STUDY

The feasibility study for our project on tomato plant disease detection using deep learning techniques revolves around assessing the technical, operational, and economic aspects of implementing such a system in agricultural settings. Thes feasibility study aims to determine the viability and potential success of our project in addressing agricultural challenges related to tomato plant disease detection while considering technical, operational, and economic factors.

3.1 Technical Feasibility:

Thorough the assessment of technical feasibility, we can determine the practicality of implementing our project, identify potential challenges, and develop strategies to overcome them. This analysis lays the foundation for successful project execution and the achievement of desired outcomes in the field of agricultural technology and crop health management.

- **Availability of High-Quality Datasets**: Ensuring access to large, well-labeled datasets containing diverse classes of diseased tomato leaf images, essential for training deep learning models effectively.
- **Computational Resources:** Evaluating the availability of computing power and specialized hardware like GPUs for training and running the deep learning models efficiently.
- **Algorithm Complexity:** Assessing the technical complexity of implementing Convolutional Neural Networks (CNNs) like ResNet152V2 for accurate disease detection and classification.

3.2 Operational Feasibility:

Even if a tomato plant disease detection system using deep learning proves technically feasible, successfully integrating it into real-world agriculture operations requires careful consideration. Here are some key aspects to evaluate:

- **Workflow Integration:** Investigating how the system will integrate into existing agricultural workflows, including data collection, image preprocessing, and diagnostic recommendations.
- **User Interface Design**: Designing a user-friendly interface for farmers to upload images, receive real-time predictions, and access diagnostic recommendations easily.
- **Hardware and Software Compatibility**: Ensuring compatibility with hardware requirements (e.g., minimum RAM, processor) and software dependencies (e.g., Python, TensorFlow, Flask)

for seamless deployment and usage.

3.3 Economic Feasibility:

The economic feasibility of implementing a tomato plant disease detection system using deep learning hinges on several factors:

- **Initial Investment**: Assessing the upfront costs associated with acquiring hardware, software, and data acquisition tools necessary for system development and deployment.
- **Cost-Benefit Analysis:** Conducting a thorough analysis of the initial investment required for hardware, software, and dataset acquisition against the potential benefits of improved crop health management and increased productivity.
- **Operational Costs:** Estimating ongoing costs for system maintenance, updates, and potential scalability as the user base grows.
- **Return on Investment (ROI):** Evaluating the expected ROI over time, considering both economic gains and enhanced agricultural outcomes resulting from more accurate disease detection and timely interventions.

4 SOFTWARE REQUIREMENTS SPECIFICATIONS

4.1 Functional Requirements:

Functional Requirements are the desired operations of a program that specify the behaviour. These requirements define the calculations, technical details, data manipulation processing. When applying deep learning techniques to tomato plant disease detection, specific functional requirements are tailored to leverage the capabilities of deep learning models. Here are the functional requirements for tomato plant disease detection from leaf images:

- **Data Collection and Preprocessing:**
  - Obtain a diverse dataset of tomato leaf images with annotations for different diseases ('Tomato___Bacterial_spot', 'Tomato___Early_blight', 'Tomato___healthy', 'Tomato___Late_blight', 'Tomato___Leaf_Mold', 'Tomato___Septoria_leaf_spot', 'Tomato___Spider_mites Two-spotted_spider_mite', 'Tomato___Target_Spot', 'Tomato___Tomato_mosaic_virus', 'Tomato___Tomato_Yellow_Leaf_Curl_Virus'). Preprocess the dataset by removing duplicates, standardizing image resolutions, and normalizing pixel values.
- **Model Selection and Architecture Design:**
  - Choose a deep learning model suitable for image classification tasks, such as ResNet152V2, for tomato plant disease detection.
  - Design the architecture with appropriate layers, including convolutional and pooling layers, to learn features from leaf images.
- **Training Data Labeling and Augmentation:**
  - Label training data with ground truth annotations for disease presence in tomato leaves.
  - Augment the dataset using techniques like rotation, flipping, and zooming to increase data diversity and improve model generalization.
- **Model Training and Optimization:**
  - Split the dataset into training, validation, and test sets.
  - Train the ResNet152V2 model on the training set, optimize hyperparameters, and use techniques like transfer learning for enhanced performance.

- **Validation and Testing:**
  - Validate the trained model using the validation set to fine-tune parameters and prevent overfitting.
  - Test the model on the test set to evaluate its accuracy, sensitivity, specificity, and other performance metrics.

4.2 Non-Functional Requirements:

In systems engineering, a non-functional requirement is a requirement that specifies criteria that can be used to judge the operation of a system, rather than specific behaviors. They are contrasted with functional requirements that define specific behavior or functions. The Non-functional requirements can be considered as quality attributes of a system. Non- functional requirements in tomato plant disease detection from leaf images encompass aspects to system performance, usability, security, and other quality attributes. Here are some non-functional requirements for tomato plant disease detection from leaf images:

• **Scalability:**

• Design the system to scale horizontally to handle larger datasets and accommodate future expansions in disease classes.

• Ensure compatibility with cloud-based services for scalability and resource management.

• **Reliability**:

• Maintain system reliability at 95% to ensure consistent and accurate disease detection results.

• Implement error handling mechanisms to address potential model failures or data inconsistencies.

• **Maintainability:**

• Structure the system with modular components for easy maintenance, updates, and enhancements.

• Provide documentation and version control to support ongoing development and maintenance efforts.

• **Security:**

• Implement data encryption and access control measures to protect sensitive information in the dataset and model.

• **Usability:**

• Develop a user-friendly interface for uploading leaf images, receiving diagnosis results, and providing recommendations to farmers.

4.2.1 Performance Requirements:

- **Inference Speed:**

- Optimize the ResNet152V2 model for fast inference times to enable real-time disease detection from leaf images.

- Set maximum allowable processing times for image analysis to ensure timely results.

- **Accuracy and Precision**:

- Define minimum accuracy and precision thresholds for disease classification to minimize false positives and negatives.

- Calculate metrics like recall, F1 score, and confusion matrix to assess model performance comprehensively.

- **Resource Utilization:**

- Monitor system resource utilization (CPU, memory) during inference to ensure efficient use of hardware resources.

- Implement resource management strategies for optimizing model performance without resource bottlenecks.

Hardware Requirements:

Hardware Requirements for Tomato Plant Disease Detection from Leaf Images using Deep Learning:

• **CPU:** A powerful multi-core CPU is required for data preprocessing, model training, and inference tasks. Processors like Intel Core i7 or AMD Ryzen with high clock speeds and multiple cores are suitable for deep learning workloads.

• **GPU (Optional):** While not mandatory, using a GPU can significantly accelerate deep learning computations, especially for training large models and processing complex datasets. NVIDIA GPUs like GeForce RTX or Quadro series are commonly used for deep learning tasks due to their parallel processing capabilities.

- **Memory (RAM):** Adequate system memory is crucial for handling large datasets and training deep learning models efficiently. A minimum of 16 GB RAM is recommended, but for more complex models or larger datasets, 32 GB or higher may be necessary to avoid performance bottlenecks.
- **Storage:** Sufficient storage space is needed to store datasets, model checkpoints, and intermediate results during training. A minimum of 256 GB SSD is recommended for faster data access and model training speeds. Additional external storage or cloud storage may be used for scalability and backup purposes.
- **Networking:** A stable network connection is essential for accessing datasets, collaborating with team members, or utilizing cloud-based resources. Ethernet connections with Gigabit or higher bandwidth are recommended for fast data transfer rates, especially when working with large image datasets.
- **Operating System:** The system should run on a compatible operating system such as Windows 10 or Linux distributions like Ubuntu, which provide support for deep learning frameworks and tools.
- **Additional Considerations**: Depending on the scale and complexity of the project, consider factors like cooling solutions for CPU/GPU intensive tasks, power supply for sustained performance, and compatibility with deep learning frameworks like TensorFlow, Keras.

4.2.2 Software Requirements:

Here are the software requirements for tomato plant disease detection from leaf images using deep learning:

- **Python Programming Language:** Python is essential for deep learning tasks due to its simplicity and extensive libraries. Ensure compatibility with deep learning frameworks like TensorFlow, and Keras.
- **Development Environment:** Set up a development environment with IDEs or text editors suitable for deep learning, such as Jupyter Notebook, Kaggle Notebook, Visual Studio Code, or Google Colab.
- **Data Management Tools:** Use tools like Pandas, NumPy, and scikit-learn for organizing, preprocessing, and augmenting tomato plant disease image datasets.
- **Model Interpretation and Visualization Tools:** Employ Matplotlib, Seaborn, and Tensor

Board for interpreting and visualizing deep learning model outputs, aiding in performance analysis and debugging.

- **Version Control:** Utilize Git or similar version control systems for managing code repositories, tracking changes, and collaborating with team members.

- **Deep Learning Frameworks:** Install and configure deep learning frameworks like TensorFlow, or Keras, which provide high-level APIs for building and training neural networks.

- **Image Processing Libraries**: Use OpenCV or Pillow (PIL), and scikit-image for image preprocessing tasks such as resizing, cropping, filtering, and augmentation.

- **Visualization Libraries:** Choose from Matplotlib, Seaborn, Plotly, or other visualization libraries to create plots, histograms, heatmaps, and other visualizations for data analysis and model evaluation.

- **Documentation Tools:** Provide comprehensive documentation using tools like Sphinx or Markdown to document code, processes, and results for future reference and collaboration

4.2.2.1 Introduction to Python:

Python serves as the primary programming language for developing lung tumor identification systems using machine learning techniques for several compelling reasons. Firstly, Python's simplicity and readability, coupled with its vast ecosystem of libraries and tools, make it well-suited for machine learning and artificial intelligence tasks. Developers can focus on algorithmic logic rather than grappling with low-level intricacies, thanks to Python's intuitive syntax.

Deep learning frameworks like TensorFlow, and Keras are readily available in Python, offering high-level APIs for constructing and training neural networks. These frameworks abstract away the complexities of deep learning algorithms, empowering developers to build sophisticated models with minimal coding efforts.

For data preprocessing and analysis, Python boasts a plethora of libraries such as NumPy, Pandas, and scikit-learn. These libraries provide efficient tools for handling multidimensional data arrays, processing tabular data, and conducting statistical analysis, which are essential for working with medical imaging data in lung tumor identification.

Python's image processing libraries, including OpenCV, Pillow (PIL), and scikit-image, play a

crucial role in preparing medical images (such as chest X-rays) for input into machine learning models. These libraries facilitate tasks like resizing, cropping, filtering, and augmentation, which are vital for enhancing the quality and relevance of the input data.

Moreover, Python offers diverse visualization libraries like Matplotlib, Seaborn, and Plotly, enabling developers to create insightful visualizations for analyzing model outputs, interpreting results, and assessing performance metrics. These visualizations aid in gaining valuable insights into the efficacy of the machine learning model and its ability to accurately identify lung tumors.

Finally, Python's support for a wide range of integrated development environments (IDEs) and text editors provides developers with flexible options for writing, debugging, and executing code efficiently. With Python's robust ecosystem and versatile capabilities, it serves as the cornerstone for developing advanced lung tumor identification systems using machine learning the methodologies from the data which is obtained

5 SYSTEM DESIGN

5.1 Requirements Modeling:

Requirements modelling for a project like tomato plant disease detection from leaf images involves identifying and documenting the needs and constraints of the system.

5.1.1 Class Diagram:

Class diagrams give an overview of a system by showing its classes and the relationships among them. Class diagrams are static – they display what interacts but not what happens when they do interact. In general a class diagram consists of some set of attributes and operations. Operations will be performed on the data values of attributes.

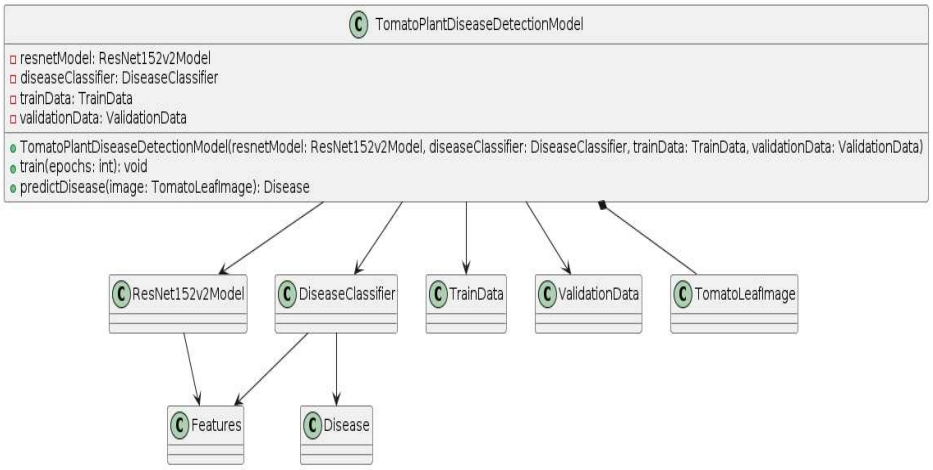


Fig-5.1.1: The image depicts a class diagram for a tomato plant disease detection model. The class diagram shows the relationships between components including a pre-trained model (ResNet152v2Model) for feature extraction, a disease classifier, and classes for training and validation data. Arrows indicate how data flows between these components during disease classification.

This class diagram following classes and their relationships:

- **TomatoPlantDiseaseDetectionModel**: This class is the main class of the system. It has attributes for the other three classes are resnetModel, diseaseClassifier, trainData, and validationData. It also has two methods: train and predictDisease.
- **ResNet152v2Model**: This class represents the ResNet model that is used to classify

tomato plant diseases. ResNet is a deep convolutional neural network architecture that has been shown to be very effective for image classification tasks.

- **DiseaseClassifier:** This class encapsulates the logic for classifying tomato plant diseases. It likely takes an image of a tomato leaf as input and outputs the class of the disease (e.g., early blight, late blight, etc.).
- **TrainData**: This class represents the training data that is used to train the resnetModel. The training data is likely a collection of images of tomato leaves that have been labeled with the corresponding disease class.
- **ValidationData:** This class represents the validation data that is used to evaluate the performance of the resnetModel on unseen data. The validation data is likely a separate collection of images of tomato leaves that have been labeled with the corresponding disease class.

The relationships between the classes are shown by the arrows in the diagram. The arrow from TomatoPlantDiseaseDetectionModel to ResNet152v2Model, DiseaseClassifier, TrainData, and ValidationData indicates that the TomatoPlantDiseaseDetectionModel class has an attribute of the other class. The arrow from TrainData to ResNet152v2Model indicates that the TrainData class is used to train the ResNet152v2Model.

In summary, the class diagram you sent shows the design of a tomato plant disease detection model. The model uses a ResNet convolutional neural network to classify tomato plant diseases from images of tomato leaves. The model is trained on a dataset of labeled images and evaluated on a separate validation dataset.

6.1.4Use Case Diagram:

Use case diagram at its simplest is a representation of a user's interaction with the system that's how the relationship between the user and the different use cases in which the user is involved. A use case diagram can identify the different types of users of a system and the different use cases and will often be accompanied by other types of diagrams as well. Actors are the external entities that interact with the system. The use cases are represented by either circles or ellipses.

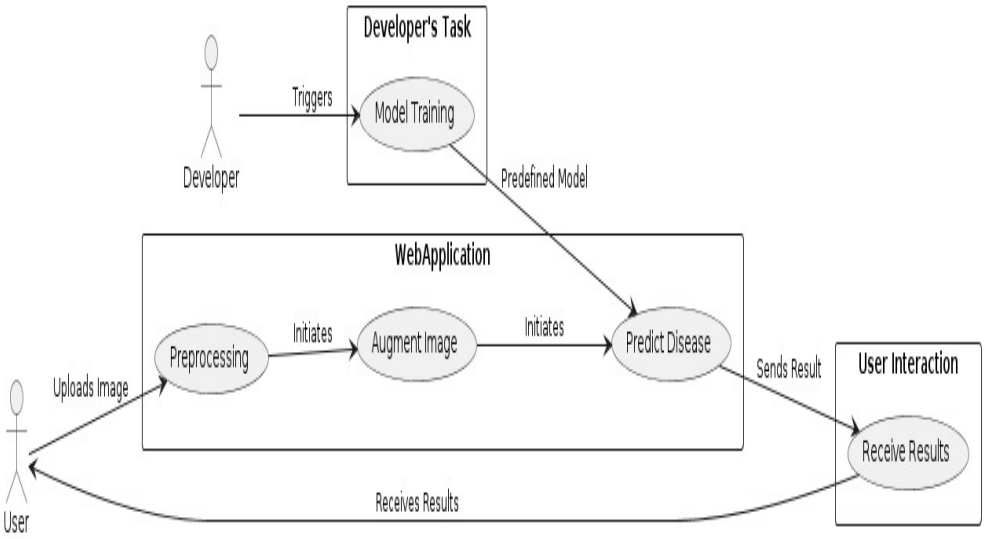


Fig-5.1.2: This web application tackles the challenge of tomato plant disease detection. Users simply upload pictures of their tomato leaves. The app then analyzes these images using a pre-trained model, likely to extract key features from the visuals. These features are then fed into a disease classifier, which determines if the plant is healthy or suffering from a specific disease. Finally, the user receives a clear diagnosis, indicating the health of their tomato plant.

This tomato plant disease detection web application would be a helpful tool for identifying potential problems in their tomato plants. Here's how it might work:

- **Simple Interface:** Ideally, the web application would have a clean and user-friendly interface. Users wouldn't need to be technical experts to utilize it.
- **Upload Functionality:** The application would likely have a clear area for users to upload photos taken directly from their phone or computer.
- **Fast Processing:** Once uploaded, the image would be processed quickly. Users wouldn't want to wait a long time for results.
- **Disease Identification:** The application would analyze the image and provide information about potential diseases affecting the tomato plant. This might involve:
- **Disease Name**: The application would display the most likely disease the plant is suffering from.

Overall Benefits: This type of web application could empower users to:

- **Early Detection:** Catch potential problems with their tomato plants early, allowing for quicker intervention.
- **Informed Decisions:** Gain valuable insights that can help them decide on the best course of action, such as using organic controls or contacting a professional.
- **Improved Yields**: By identifying and treating diseases promptly, users could potentially improve the overall health and yield of their tomato plants.

5.1.3 Sequence Diagram:

The sequence diagram shows object interactions arranged in time sequence. It depicts the objects and classes involved in the scenario and the sequence of messages exchanged between the objects needed to carry out the functionality of the scenario.

Sequence diagrams are typically associated with use case realizations in the Logical View of the system under development. Sequence diagrams are sometimes called event diagrams or event scenarios.

A Sequence diagram shows interactions arranged in time sequence. It depicts the s and classes involved in the scenario and the sequence of messages exchanged between the s needed to carry out the functionality of the scenario. Sequence diagrams are typically associated with use case realizations in the Logical View of the system under development.

findings, to aid clinicians in interpreting and validating the model's predictions.

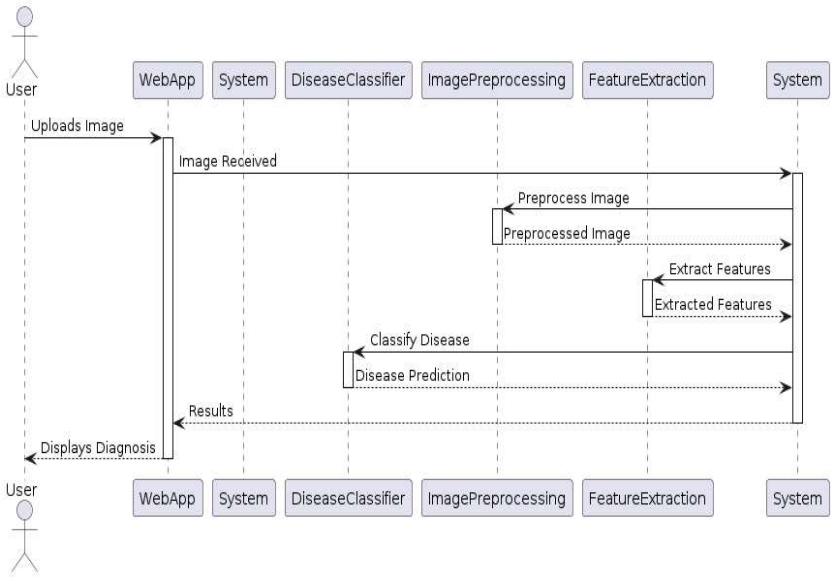


Fig-5.1.3: This Sequence diagram outlines a web application designed to diagnose diseases in tomato plants. Users can upload images of their tomato leaves. The application then analyzes these images using two key components: a pre-trained model, likely used to extract relevant features from the images, and a disease classifier, which leverages those features to identify any present diseases. Finally, the user is provided with a diagnosis indicating the health of their tomato plant.

This sequence of the process of uploading and processing images on a website. Here are the steps involved:

User Uploads Image: The user selects an image file from their device and uploads it to the website.

Image Received: The web application receives the uploaded image file.

Preprocess Image: The image is preprocessed before it can be used for further processing. Preprocessing may involve resizing the image, converting it to a specific format, or normalization techniques.

Extract Features: Features are extracted from the preprocessed image. Features are characteristics of the image that are relevant to the task at hand. For example, in image classification, features might include the edges, shapes, and colors of objects in the image.

Classify Disease: The extracted features are used to classify the image. In the context of the tomato plant disease detection system, this would involve using a machine learning model to classify the disease in the image.

Disease Prediction: Based on the classification, the system predicts the disease present in the image.

Results: The results are displayed to the user. This could involve displaying the predicted disease on the user interface or providing the user with downloadable results.

5.1.4 Activity Diagram:

Activity diagram is essentially a fancy flowchart: Activity and state diagrams are related. State-chart diagram focuses on undergoing a process. An activity diagram focuses on the flow of activities involved in a single process. The activity diagram shows the activities depend on one another.

An activity represents the performance of the task or duty in a workflow. It may also represent

the execution of a statement in a procedure. You can share activities between state machines. However, transitions cannot be shared.

Activity diagrams provide a way to model the workflow of a business process, code specific information such as a class operation. The transitions are implicitly triggered by completion of the actions in the source activities

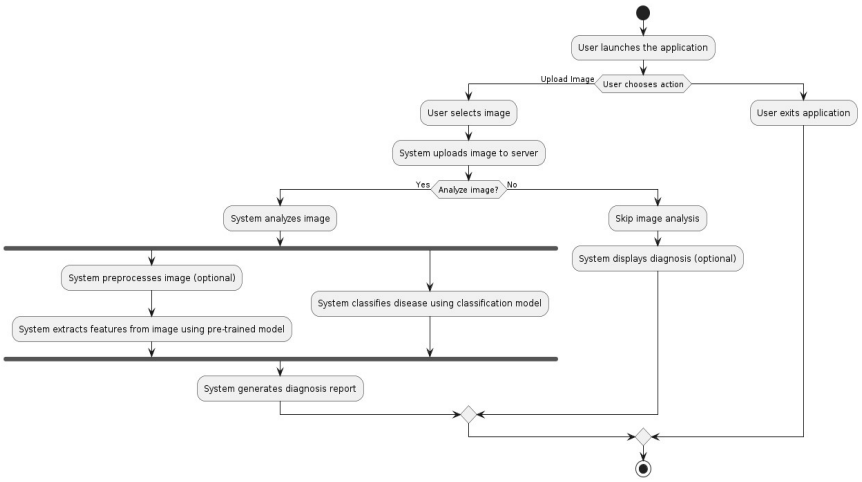


Fig-5.1.4: This flow chart outlines a system for diagnosing tomato plant diseases. Users upload images of leaves, which are then analyzed. A pre-trained model extracts key features from the images, and a disease classifier uses these features to identify potential diseases. Finally, the user receives a diagnosis indicating the health of their tomato plant.

This flowchart showing the steps involved in a system that generates a disease detection following this steps:

User Launches Application: The process begins when the user opens the application.

User Chooses Action: The user selects what they want to do within the application.

User Selects Image: If the user opted to upload an image, they then select the image file from their device.

System Uploads Image to Specific folder : If an image was selected, the system uploads the image file to a specific folder for processing.

Analyze Image (Yes/No Decision): The system determines whether to analyze the uploaded image. It's possible there might be an option for the user to choose to skip image analysis, but the default appears to be to analyze it.

Yes - Analyze Image: If image analysis is chosen, the system proceeds to analyze the image.

No - Skip Image Analysis: If image analysis is skipped, the system jumps to step 9.

System Extracts Features from Image: Features are extracted from the preprocessed image. Features are characteristics of the image that are relevant to the task at hand. In medical image analysis, features might include shapes, textures, and patterns within the image.

System Classifies Disease Using Classification Model: The extracted features are used by a classification model to classify the medical condition in the image.

User Exits Application: The user exits the application.

5.2 Design Concepts & Constraints

Design in software development is a multifaceted process encompassing data structure, software architecture, procedural intricacies, and module interfaces. It serves to translate requirements into a software blueprint for quality assessment prior to actual coding. While software design continually evolves with new methods and improved analysis, it lacks the quantifiable depth of traditional engineering disciplines. Nonetheless, established design techniques, quality criteria, and notation frameworks exist, guiding the design phase's goal of crafting effective solutions to specified problems. A well-executed design profoundly influences software quality, significantly impacting subsequent phases like testing and maintenance.

5.2.1 Design Concepts:

Data Integration and Processing: Develop a robust pipeline for preprocessing and augmenting the tomato plant disease dataset. Include techniques for data normalization, image resizing, and annotation parsing to prepare the data for training. Incorporate data augmentation methods like rotation, flipping, and color adjustment to enhance dataset variability and model robustness.

Model Selection and Customization: Choose a suitable CNN architecture, such as ResNet152V2, tailored for plant disease detection. Customize the architecture to suit the characteristics of leaf images and optimize performance metrics such as accuracy and recall. Experiment with different model configurations to find the most effective architecture.

Modular Design: Design the system with modularity to allow easy integration of components and future scalability. Create modules for data preprocessing, model training, inference, and

result visualization. Define clear interfaces between modules to promote code reusability and maintenance.

Continuous Integration/Continuous Deployment (CI/CD): Implement CI/CD pipelines to automate model training, testing, and deployment processes. Ensure that updates and improvements can be deployed efficiently while maintaining system reliability and consistency.

Model Interpretability: Incorporate techniques for model interpretability to enhance understanding and trust in model predictions. Use methods like attention maps or gradient-based visualization to provide insights into the features influencing disease classification.

5.2.2 Constraints:

Data Privacy and Security: Adhere to strict data privacy regulations and implement security measures to protect sensitive information in the dataset. Use encryption, access controls, and anonymization techniques to safeguard patient data and prevent unauthorized access.

Regulatory Compliance: Ensure compliance with regulatory standards governing agricultural data usage. Obtain necessary approvals and permissions for dataset usage and model deployment, and maintain documentation to demonstrate regulatory compliance.

Resource Limitations: Consider constraints in computational resources, such as processing power and memory, which may affect model complexity and training time. Optimize algorithms and model architectures to maximize performance within resource constraints.

Ethical Considerations: Address ethical concerns related to AI-based disease detection, including fairness, transparency, and accountability. Mitigate biases and ensure that the system operates ethically and responsibly in the context of agricultural technology and crop health management.

6 SYSTEM IMPLEMENTATION & METHODOLOGIES

6.1 Methodologies

The methodologies for your project encompass a structured approach to system development, emphasizing design, conceptualization, logic, architecture, and algorithm design. Design diagrams detail system components and interactions, while conceptual diagrams provide high-level visualizations. Logical design defines system behavior and functionality, while architectural design outlines system structure and components. Algorithm design focuses on developing efficient algorithms for data processing and analysis. These methodologies ensure systematic and thorough development, leading to a robust lung tumor detection system.

6.1.1 Architectural Design

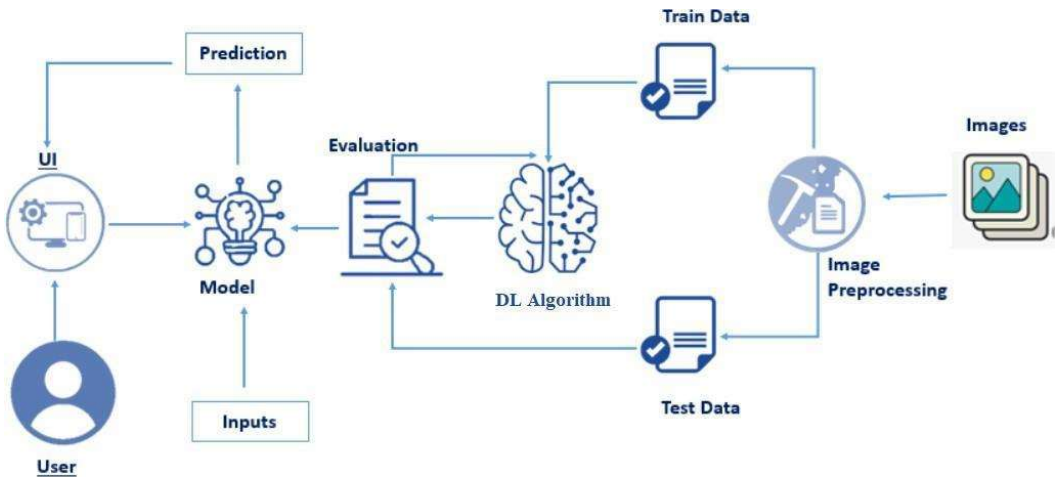


Fig-6.1.1: The diagram outlines a machine learning system. Data is collected, cleaned and fed into a model. This model is trained to identify patterns and then tested for accuracy. If it performs well, it can be used to make predictions on entirely new data.

Data Collection: In this stage, data relevant to the task at hand is gathered. This data can come from a variety of sources, including user input, sensors, and APIs. In the image, this is shown by the box labelled “Train Data” and “Test Data”.

Data Preprocessing: Once collected, the data may need to be cleaned and formatted before it can be used by a machine learning model. This can involve removing errors, inconsistencies, and outliers from the data. This stage is represented in the image by the box labelled

"Preprocessing".

Model Selection and Training: A machine learning model is chosen based on the type of task you want to perform. Then, the model is trained on a subset of the data called the training data. During training, the model learns to identify patterns in the data. This stage is shown in the image by the boxes labelled "Model" and "Train Data".

Evaluation: Once trained, the model is evaluated on a different subset of the data called the test data. This helps to assess how well the model generalizes to unseen data. The evaluation stage is depicted in the box labelled "Evaluation" in the image.

Prediction: If the model performs well on the test data, it can be used to make predictions on new data. This is illustrated by the box labelled "Prediction" at the bottom of the image.

Deployment: Finally, the model can be deployed into a production environment where it can be used to make real-world predictions.

6.1.2 Proposed Architecture in Designing:

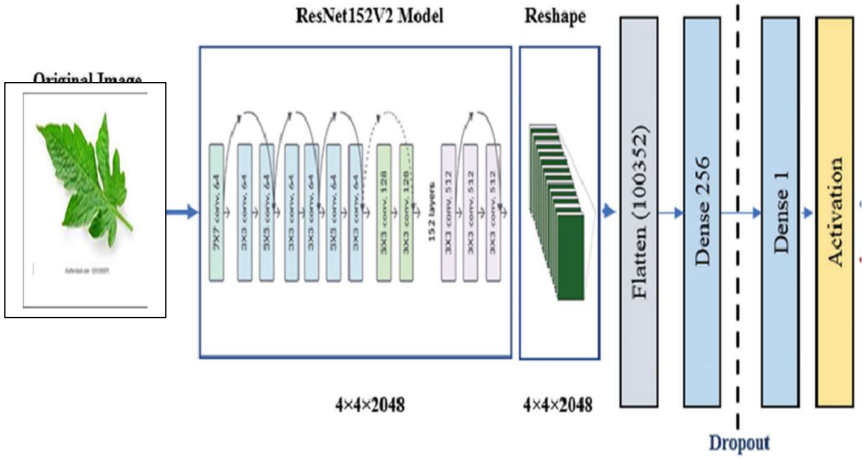


Fig-6.1.2: The image depicts a convolutional neural network (CNN) architecture called ResNet152V2. It's a deep learning model that takes an image as input, processes it through layers, and outputs a classification (Transfer Learning).

- **Image Input:** ResNet152V2 takes an image as input. This image is likely pre-processed to a fixed size and format before feeding into the network**.**
- **Convolutional Layers:** The core of ResNet152V2 lies in its numerous convolutional layers. These layers apply filters to the image, extracting features like edges, shapes, and textures. There are 152 convolutional layers in total, stacked together to progressively learn complex

features.

- **Residual Connections:** A key innovation in ResNet152V2 is the use of residual connections. These connections bypass some layers, allowing the network to directly add the original input to the output of later layers. This helps address the vanishing gradient problem, a common challenge in deep networks, and allows the model to learn even intricate features.
- **Pooling Layers:** Interspersed between convolutional layers are pooling layers. These layers down sample the data, reducing its dimensionality and computational cost. Pooling can also help the network identify features that are invariant to small shifts in the image.
- **Activation Functions:** Each convolutional layer is followed by an activation function. This function introduces non-linearity into the network, allowing it to learn more complex relationships between features.
- **Classification Layers**: After the convolutional processing, the network uses fully-connected layers to classify the image. These layers take the extracted features and map them to a set of output probabilities, indicating the likelihood of the image belonging to a particular class.

6.1.3 Algorithm Design

This Algorithm design for tomato plant disease detection project from leaf images using the ResNet152V2 model:

- **Step 1: Data Collection**
  - Gather a diverse dataset of tomato plant leaf images representing various disease conditions, including bacterial spot, early blight, late blight, leaf mold, septoria leaf spot, and others.
- **Step 2: Data Augmentation**
  - Apply augmentation techniques such as rotation, flipping, zooming, and color jittering to increase dataset variability and improve the model's ability to generalize patterns across different disease types.
- **Step 3: Data Preprocessing**
  - Preprocess the leaf images by resizing them to a uniform size suitable for the ResNet152V2 model input, perform normalization to ensure consistent pixel values, and apply augmentation transformations to enhance dataset diversity.
- **Step 4: Model Training**
  - Select the ResNet152V2 architecture as the machine learning model for disease detection.
  - Customize the ResNet152V2 architecture to adapt it to the characteristics of tomato plant leaf images and optimize its performance for disease classification.
  - Train the customized ResNet152V2 model using the preprocessed and augmented dataset, employing techniques like transfer learning to leverage pre-trained weights and

fine-tuning to refine the model's predictions.

- **Step 5: Model Evaluation**
  - Reserve a portion of the dataset as a test set for evaluating the trained ResNet152V2 model's performance.
  - Use evaluation metrics such as accuracy, precision, recall, and F1-score to assess the model's ability to correctly classify tomato plant diseases from leaf images.
- **Step 6: Performance Analysis**
  - Analyze the model's performance on the test dataset to identify its strengths and areas for improvement.
  - Utilize visualization tools to interpret the model's predictions, visualize feature activations, and highlight disease regions within the leaf images.
- **Step 7: Application Development**
  - Develop a user-friendly web application using Flask or a similar framework to enable users (e.g., farmers or house gardeners) to upload images of tomato plant leaves.
  - Integrate the trained ResNet152V2 model into the web application to perform real-time disease detection and classification based on uploaded leaf images.
- **Step 8: Deployment**
  - Deploy the web application containing the trained ResNet152V2 model to a hosting environment, ensuring accessibility for users to access disease detection functionality from any device with internet connectivity.

6.1.4 Module design Specifications

Model Module:

- **Data Acquisition Module:**

Collects images of tomato plant leaves representing different disease conditions (e.g., bacterial spot, early blight) from the dataset.

- **Data Preprocessing Module:**

Cleans and preprocesses the leaf images, including resizing, cropping, and color normalization to standardize input data.

- **Data Augmentation Module:**

Augments the dataset to increase variability and improve model generalization. Techniques like rotation, flipping, and zooming can be applied.

- **Model Selection Module:**

Selects an appropriate deep learning model architecture for disease detection, such as

ResNet152V2, tailored for image analysis tasks.

- **Model Training Module:**

  Trains the selected model on the preprocessed and augmented dataset to learn features and patterns indicative of different tomato plant diseases.

- **Model Evaluation Module:**

  Evaluates the trained model's performance using a validation dataset, calculating metrics like accuracy, precision, recall, and F1-score.

- **Performance Analysis Module:**

  Analyzes the model's performance results to identify its strengths, weaknesses, and areas for improvement in disease detection accuracy.

- **Application Development Module:**

  Develops a user-friendly web interface, possibly using Flask or a similar framework, for users to upload images of tomato plant leaves and receive disease detection results.

- **Deployment Module:**

  Deploys the trained model within the web application, allowing users to access disease detection functionality in real-time.

- **Monitoring Module:**

  Implements monitoring functionalities to continuously track the application's performance, detect any issues, and ensure optimal system reliability and user experience.

6.2 Deployment Environment and Tools:

- **IDE (Integrated Development Environment):**
  - Google Co-lab (for Python coding and development)
- **Programming Languages:**
  - Python (for backend development and machine learning)
  - HTML, CSS, JavaScript (for frontend development)

- **Deployment Platforms:**
  - Deploying the web application)
- **Documentation Tools:**
  - Microsoft Word-11 and plantuml

Creating a development environment for a tomato plant disease detection from leaf images involves setting up the necessary tools and libraries. Here's a basic setup guide:

- **Operating System:** Use a supported operating system such as Windows 10, macOS or Linux distribution like Ubuntu.
- **Python:** Install Python, which is commonly used for deep learning projects. You can download the latest version from the official Python website and follow the installation instructions (python-3.10.7)
- **Deep Learning Libraries:** Install deep learning libraries such as TensorFlow and keras. You can install these libraries using pip, Python's package manager.
- **Data Visualization:** Install libraries for data visualization, such as Matplotlib and Seaborn, to visualize the dataset and model performance.
- **Other Libraries**: Depending on your specific needs, you may also need to install other libraries for data manipulation (e.g., Pandas, numpy), image processing (e.g., pillow), or model evaluation (e.g., Scikit-learn).

6.3 Coding:

In this project, extensive Python coding was undertaken to preprocess data, implement deep learning algorithms, and develop the web application interface. Specifically, coding involved data preprocessing techniques to clean and prepare input data for analysis. Additionally, ResNet152v2 were implemented using libraries like Keras and TensorFlow. Overall, Python coding was instrumental in driving the success of the project from data processing to model deployment.

6.3.1 Source Code:

Final_model**.ipynb**

```
!pip install -q kaggle

!mkdir ~/.kaggle
```

```
!cp kaggle.json ~/.kaggle

!kaggle datasets download -d kaustubhb999/tomatoleaf

!unzip /content/tomatoleaf.zip

#Import Required Libraries

import warnings

warnings.filterwarnings('ignore')

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

import os

from PIL import Image

from tensorflow import keras

from tensorflow.keras.preprocessing.image import ImageDataGenerator

from tensorflow.keras.applications import ResNet152V2

from tensorflow.keras.layers import Dense, GlobalAveragePooling2D

from tensorflow.keras.models import Model

from tensorflow.keras.optimizers import Adam

from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score,
confusion_matrix

# Define paths for train, validation, and test data

train_data_dir='/content/tomato/train'

val_data_dir='/content/tomato/val'

# Function to count images in each class

def count_images_in_directory(directory):

    class_counts = {}
```

```
    for class_name in os.listdir(directory):
        class_path = os.path.join(directory, class_name)
        if os.path.isdir(class_path):
            class_counts[class_name] = len(os.listdir(class_path))
    return class_counts
train_class_counts = count_images_in_directory(train_data_dir)
val_class_counts = count_images_in_directory(val_data_dir)
#printing data count in validation
print("Training Directory Image Counts:")
for class_name, count in train_class_counts.items():
    print(f"{class_name}: {count}")
# Plotting the pie chart for training data
plt.figure(figsize=(5,5))
plt.pie(train_class_counts.values(), labels=train_class_counts.keys(), autopct='%1.1f%%')
plt.title('Percentage of Images in Each Class (Training)')
plt.show()
#printing data count in validation
print("Validation and Test Directory Image Counts:")
for class_name, count in val_class_counts.items():
    print(f"{class_name}: {count}")
# Plotting the pie chart for validation data
plt.figure(figsize=(5,5))
plt.pie(val_class_counts.values(), labels=val_class_counts.keys(), autopct='%1.1f%%')
plt.title('Percentage of Images in Each Class (Validation)')
plt.show()
# Get the list of subdirectories (class folders) in the train_data_dir
```

```
train_subdirectories = next(os.walk(train_data_dir))[1]

# Calculate the total number of images in all subdirectories

train_total_images = sum(len(os.listdir(os.path.join(train_data_dir, subdir))) for subdir in
train_subdirectories)

# Get the list of subdirectories (class folders) in the val_data_dir

val_subdirectories = next(os.walk(val_data_dir))[1]

# Calculate the total number of images in all subdirectories

val_total_images = sum(len(os.listdir(os.path.join(val_data_dir, subdir))) for subdir in
val_subdirectories)

# Create pie chart for dataset distribution

sizes = [train_total_images,val_total_images]

labels = ['Train_folder',  'Val_folder']

colors = ['lightcoral', 'lightskyblue']

explode = (0.1, 0)  # explode 1st slice

plt.figure(figsize=(7, 7))

plt.pie(sizes, explode=explode, labels=labels, colors=colors, autopct='%1.1f%%',
shadow=True)

plt.title('Dataset Distribution')

plt.show()

# Set hyperparameters

batch_size = 32

epochs = 20

# Define class ordering based on the assumption of 10 classes for tomato diseases

class_order = [

    'Tomato___Bacterial_spot', 'Tomato___Early_blight', 'Tomato___healthy',

    'Tomato___Late_blight', 'Tomato___Leaf_Mold', 'Tomato___Septoria_leaf_spot',

    'Tomato___Spider_mites Two-spotted_spider_mite', 'Tomato___Target_Spot',
```

```
    'Tomato___Tomato_mosaic_virus', 'Tomato___Tomato_Yellow_Leaf_Curl_Virus'
]
# Data generator with rescaling only for train, validation, and test data
Datagen=
keras.preprocessing.image.ImageDataGenerator(rescale=1./255,rotation_range=20,width_shif
t_range=0.2,height_shift_range=0.2,shear_range=0.2,zoom_range=0.2,horizontal_flip=True,
vertical_flip=True,fill_mode='nearest')
# Create data generator for training data with class ordering
train_generator =
datagen.flow_from_directory(train_data_dir,target_size=(256,256),batch_size=batch_size,cla
ss_mode='categorical',classes=class_order,seed=123)
# Create data generator for validation data with class ordering
val_generator =
datagen.flow_from_directory(val_data_dir,target_size=(256,256),batch_size=batch_size,class
_mode='categorical',classes=class_order,shuffle=False)
# Create data generator for test data with class ordering
test_generator = datagen.flow_from_directory(val_data_dir,target_size=(256,
256),batch_size=batch_size,class_mode='categorical',classes=class_order,shuffle=False)
# Create pie chart for dataset distribution
sizes = [len(train_generator), len(test_generator), len(val_generator)]
labels = ['Train', 'Test', 'Validation']
colors = ['gold', 'lightcoral', 'lightskyblue']
explode = (0.1, 0, 0)  # explode 1st slice
plt.figure(figsize=(7, 7))
plt.pie(sizes, explode=explode, labels=labels, colors=colors, autopct='%1.1f%%',
shadow=True)
plt.title('Dataset Distribution')
plt.show()
```

```
# Plot sample images from each class in the training data
classes = os.listdir(train_data_dir)
plt.figure(figsize=(25, 10))
for i, class_name in enumerate(classes):
    class_path = os.path.join(train_data_dir, class_name)
    if os.path.isdir(class_path):
        images = os.listdir(class_path)
        if images:
            pic = os.path.join(class_path, images[0])
            image = Image.open(pic)
            image = np.asarray(image)
            plt.subplot(2, 5, i+1)
            plt.title('{0} / Shape{1}'.format(class_name, image.shape))
            plt.imshow(image)
plt.show()
# Load pre-trained ResNet152V2 model
base_model = ResNet152V2(weights='imagenet', include_top=False)
# Freeze layers in the base model
for layer in base_model.layers:
    layer.trainable = False
# Add custom layers on top of the base model
x = base_model.output
x = GlobalAveragePooling2D()(x)
x = Dense(1024, activation='relu')(x)
predictions = Dense(len(class_order), activation='softmax')(x)  # Use len(class_order) for
number of classes
```

```
# Create the final model

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

# Getting summary of the model

model.summary()

# Compile the model

model.compile(optimizer=Adam(learning_rate=0.001), loss='categorical_crossentropy',
metrics=['accuracy'])

# Train the model

history=model.fit(train_generator,steps_per_epoch=len(train_generator),epochs=epochs,valid
ation_data=val_generator,validation_steps=len(val_generator))

# Evaluate the model on the test data

test_loss, test_accuracy = model.evaluate(test_generator, steps=len(test_generator))

print(f'Test Loss: {test_loss}, Test Accuracy: {test_accuracy}')

# Prediction and visualizations for validation data

classes = os.listdir(val_data_dir)

plt.figure(figsize=(18, 28))

for i, class_name in enumerate(classes):

    pic_list = os.listdir(os.path.join(val_data_dir, class_name))

    pic = pic_list[np.random.randint(len(pic_list))]

    image = Image.open(os.path.join(val_data_dir, class_name, pic))

    image = np.asarray(image)

    pred = np.argmax(model.predict(image.reshape(-1, 256, 256, 3) / 255))

    prediction = class_order[pred]

    plt.subplot(5, 2, i+1)

    plt.title('Actual: {0} / Predicted: {1}'.format(class_name, prediction))

    plt.imshow(image)
```

```
plt.show()
# Plotting accuracy graph
plt.figure(figsize=(10, 5))
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.axhline(y=test_accuracy, color='r', linestyle='--', label='Test Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.title('Training, Validation, and Test Accuracy')
plt.legend()
plt.grid()
plt.show()
# Plotting accuracy graph
plt.figure(figsize=(10, 5))
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.axhline(y=test_loss, color='r', linestyle='--', label='Test Loss')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.title('Training, Validation, and Test Loss')
plt.legend()
plt.grid()
plt.show()
loss = history.history['loss'][-1]  # Final training loss
val_loss = history.history['val_loss'][-1]  # Final validation loss
accuracy = history.history['accuracy'][-1]  # Final training accuracy
```

```
val_accuracy = history.history['val_accuracy'][-1]  # Final validation accuracy

# Create a bar plot for train, validation, and test set loss and accuracy

plt.figure(figsize=(10, 6))

# Loss bars

plt.bar(['Train Loss', 'Validation Loss', 'Test Loss'], [loss, val_loss, test_loss], color=['orange',
'yellow', 'green'], label='Loss')

# Accuracy bars

plt.bar(['Train Accuracy', 'Validation Accuracy', 'Test Accuracy'], [accuracy, val_accuracy,
test_accuracy], color=['blue', 'purple', 'red'], label='Accuracy')

plt.title('Train, Validation, Test Set - Loss and Accuracy')

plt.ylabel('Value')

plt.legend()

plt.tight_layout()

plt.show()

# accuracy and loss

train_loss = history.history['loss']

val_loss = history.history['val_loss']

test_loss = test_loss  # Replace test_loss with the actual value

train_accuracy = history.history['accuracy']

val_accuracy = history.history['val_accuracy']

test_accuracy = test_accuracy  # Replace test_accuracy with the actual value

epochs = range(1, len(train_loss) + 1)  # Epoch numbers

# Create a figure and axis for subplots

fig, ax1 = plt.subplots(figsize=(10, 6))

# Plot training, validation, and test loss on the first y-axis

ax1.plot(epochs, train_loss, label='Training Loss', color='blue', marker='o')
```

```
    ax1.plot(epochs, val_loss, label='Validation Loss', color='green', marker='s')
    ax1.axhline(y=test_loss, color='red', linestyle='--', label='Test Loss')
    ax1.set_xlabel('Epochs')
    ax1.set_ylabel('Loss')
    ax1.set_title('Training, Validation, and Test (Loss and Accuracy)')
    ax1.legend(loc='upper right')
    # Create a second y-axis for accuracy
    ax2 = ax1.twinx()
    ax2.plot(epochs, train_accuracy, label='Training Accuracy', color='red', linestyle='--')
    ax2.plot(epochs, val_accuracy, label='Validation Accuracy', color='purple', linestyle='--')
    ax2.axhline(y=test_accuracy, color='orange', linestyle='--', label='Test Accuracy')
    ax2.set_ylabel('Accuracy')
    ax2.legend(loc='lower right')
    plt.show()
    # accuracy and loss
    train_loss = history.history['loss']
    val_loss = history.history['val_loss']
    test_loss = test_loss  # Replace test_loss with the actual value
    train_accuracy = history.history['accuracy']
val_accuracy = history.history['val_accuracy']
test_accuracy = test_accuracy  # Replace test_accuracy with the actual value
epochs = range(1, len(train_loss) + 1)  # Epoch numbers
# Create a figure with two subplots
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 6))
# Plot training, validation, and test loss on the left side
ax1.plot(epochs, train_loss, label='Training Loss', color='blue', marker='o')
```

```
ax1.plot(epochs, val_loss, label='Validation Loss', color='green', marker='s')
ax1.axhline(y=test_loss, color='red', linestyle='--', label='Test Loss')
ax1.set_xlabel('Epochs')
ax1.set_ylabel('Loss')
ax1.set_title('Training, Validation, and Test Loss')
ax1.legend(loc='upper right')
# Plot training, validation, and test accuracy on the right side
ax2.plot(epochs, train_accuracy, label='Training Accuracy', color='red', linestyle='--')
ax2.plot(epochs, val_accuracy, label='Validation Accuracy', color='purple', linestyle='--')
ax2.axhline(y=test_accuracy, color='orange', linestyle='--', label='Test Accuracy')
ax2.set_xlabel('Epochs')
ax2.set_ylabel('Accuracy')
ax2.set_title('Training, Validation, and Test Accuracy')
ax2.legend(loc='lower right')  # Add legend for accuracy plot
plt.tight_layout()  # Optional: adjust subplots to prevent overlap
plt.show()
train_loss = history.history['loss']
train_accuracy = history.history['accuracy']
epochs = range(1, len(train_loss) + 1)
# Create a figure
plt.figure(figsize=(8, 4))
# Plot training loss
plt.plot(epochs, train_loss, label='Training Loss', color='blue', marker='o')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.title('Training Loss and Accuracy')
```

```
plt.legend()
# Plot training accuracy on the same graph
plt.plot(epochs, train_accuracy, label='Training Accuracy', color='red', marker='s')
plt.ylabel('Loss / Accuracy')  # Shared y-axis label
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
val_loss = history.history['val_loss']
val_accuracy = history.history['val_accuracy']
epochs = range(1, len(val_loss) + 1)
# Create a figure
plt.figure(figsize=(8,4))
# Plot training loss
plt.plot(epochs, train_loss, label='Validation Loss', color='blue', marker='o')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.title('Validation Loss and Accuracy')
plt.legend()
# Plot training accuracy on the same graph
plt.plot(epochs, val_accuracy, label='Validation Accuracy', color='red', marker='s')
plt.ylabel('Loss / Accuracy')  # Shared y-axis label
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
```

```
#  evaluated your model and obtained test_loss and test_accuracy
# Create a figure
plt.figure(figsize=(8, 4))
# Plot test loss as a single point
plt.scatter(1, test_loss, label='Test Loss', color='blue', marker='o')
# Plot test accuracy as a single point
plt.scatter(1, test_accuracy, label='Test Accuracy', color='red', marker='s')
plt.xticks([1], ['Test Metrics'])  # Customize x-axis tick label
plt.xlabel('Metrics')
plt.ylabel('Value')
plt.title('Test Loss and Accuracy')
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
# Evaluate the model on the test data
true_labels = test_generator.classes  # True labels for the test data
predicted_labels = model.predict(test_generator).argmax(axis=-1)  # Predicted labels for the test
    data
accuracy = accuracy_score(true_labels, predicted_labels)
precision = precision_score(true_labels, predicted_labels, average='weighted')
recall = recall_score(true_labels, predicted_labels, average='weighted')
f1 = f1_score(true_labels, predicted_labels, average='weighted')
print(f'Test Accuracy: {accuracy}')
print(f'Precision: {precision}')
print(f'Recall: {recall}')
```

```
print(f'F1 Score: {f1}')
# Create confusion matrix
conf_matrix = confusion_matrix(true_labels, predicted_labels)
plt.figure(figsize=(10, 8))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', cbar=False)
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.title('Confusion Matrix')
plt.show()
‘’’ # Create confusion matrix
conf_matrix = confusion_matrix(true_labels, predicted_labels)
plt.figure(figsize=(8,8))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', cbar=False)
class_labels=[
    'Tomato___Bacterial_spot', 'Tomato___Early_blight', 'Tomato___healthy',
    'Tomato___Late_blight', 'Tomato___Leaf_Mold', 'Tomato___Septoria_leaf_spot',
    'Tomato___Spider_mites Two-spotted_spider_mite', 'Tomato___Target_Spot',
    'Tomato___Tomato_mosaic_virus', 'Tomato___Tomato_Yellow_Leaf_Curl_Virus'
]
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.title('Confusion Matrix')
# Set x-axis and y-axis ticks with class labels
plt.xticks(ticks=range(len(class_labels)), labels=class_labels, fontsize=8, rotation=45)
plt.yticks(ticks=range(len(class_labels)), labels=class_labels, fontsize=8,rotation=45)
plt.show()’’’
```

```
# Create bar graph for evaluation metrics
metrics = ['Accuracy', 'Precision', 'Recall', 'F1 Score',]
values = [accuracy, precision, recall, f1]
plt.figure(figsize=(8, 5))
plt.bar(metrics, values, color=['blue', 'green', 'orange', 'red'])
plt.xlabel('Metrics')
plt.ylabel('Values')
plt.title('Model Evaluation Metrics')
plt.ylim(0, 1)
plt.show()
#save the model
model.save('/content/Final_model_tmt.h5')
#check whether the save model exist or not
print(os.path.exists('/content/Final_model_tmt.h5'))
#at the time of running application to resolve the errors we need to do these times on the time of
    training(!!!optional)
#model.save('path_to_save_model', save_format='tf', include_optimizer=True)
#model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy']
```

app.py

```
import os
from flask import Flask, request, render_template
from PIL import Image
import numpy as np
import tensorflow as tf
app = Flask(__name__, template_folder='templates')
model = tf.keras.models.load_model('Final_model_tmt.h5')
```

```
@app.route('/')
def home():
    return render_template('index.html')
@app.route('/about')
def about():
    return render_template('about.html')
@app.route('/gallery')
def gallery():
        return render_template('gallery-single.html')
@app.route('/contact')
def contact():
    return render_template('contact.html')
@app.route('/predict')
def pred():
    return render_template('predict.html')
ALLOWED_EXTENSIONS = {'png', 'jpg', 'jpeg'}


def allowed_file(filename):
    return '.' in filename and filename.rsplit('.', 1)[1].lower() in ALLOWED_EXTENSIONS
@app.route('/result', methods=['POST'])
def predict():
    if request.method == "POST":
        f = request.files['file']
        if f.filename == '':
            return render_template('results.html', prediction_text="No file selected")
        if not allowed_file(f.filename):
```

```
        return render_template('results.html', prediction_text="Mismatched file format")
    # Save and process the image
    basepath = os.path.dirname(__file__)
    filepath = os.path.join(basepath, 'uploads', f.filename)
    f.save(filepath)
    image = Image.open(filepath)
    image = image.resize((256, 256))  # Resize image to match model input size
    image = np.asarray(image) / 255.0  # Normalize image
    image = np.expand_dims(image, axis=0)  # Add batch dimension
          # Make prediction
    prediction = model.predict(image)
    class_index = np.argmax(prediction)
    # Map class index to class name
    class_names = ['Tomato___Bacterial_spot', 'Tomato___Early_blight', 'Tomato___healthy',
                   'Tomato___Late_blight', 'Tomato___Leaf_Mold', 'Tomato___Septoria_leaf_spot',
                   'Tomato___Spider_mites Two-spotted_spider_mite', 'Tomato___Target_Spot',
                   'Tomato___Tomato_mosaic_virus',
'Tomato___Tomato_Yellow_Leaf_Curl_Virus']
    predicted_class = class_names[class_index]
    # Prepare message based on predicted class
    if predicted_class == 'Tomato___healthy':
        prediction_text = "Provided Tomato Plant Leaf is Healthy :)"
    else:
        prediction_text = f"The tomato plant leaf is
having:\n{predicted_class.replace('Tomato___', '')} :("
    return render_template('results.html', prediction_text=prediction_text)
```

if __name__ == "__main__":

app.run(debug=True)

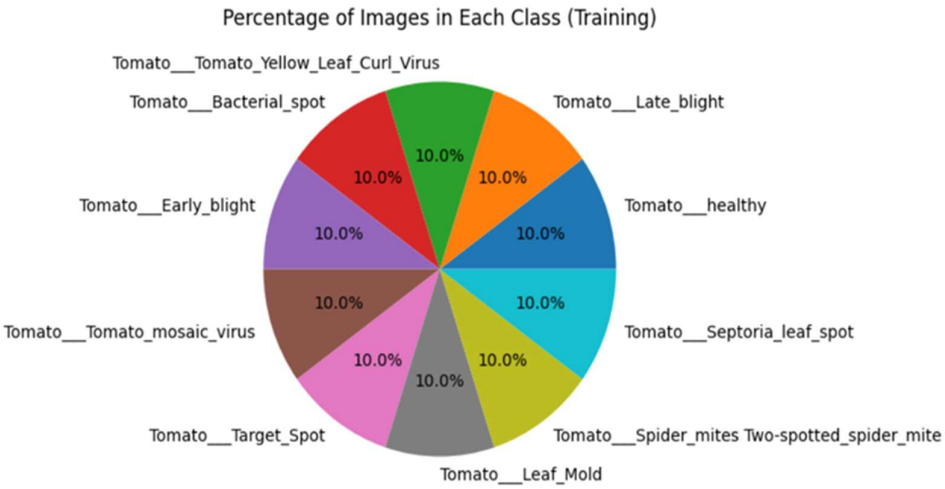


Fig-6.3.1.1: Images Count in Training data set(Balanced Data- Best Performance).

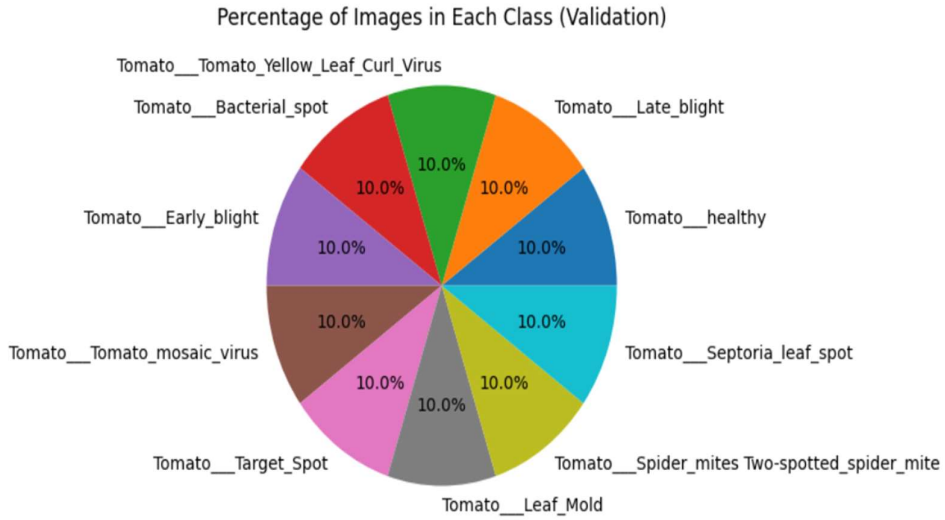


Fig-6.3.1.2: Images Count in Validation data set(Balanced Data- Best Performance).

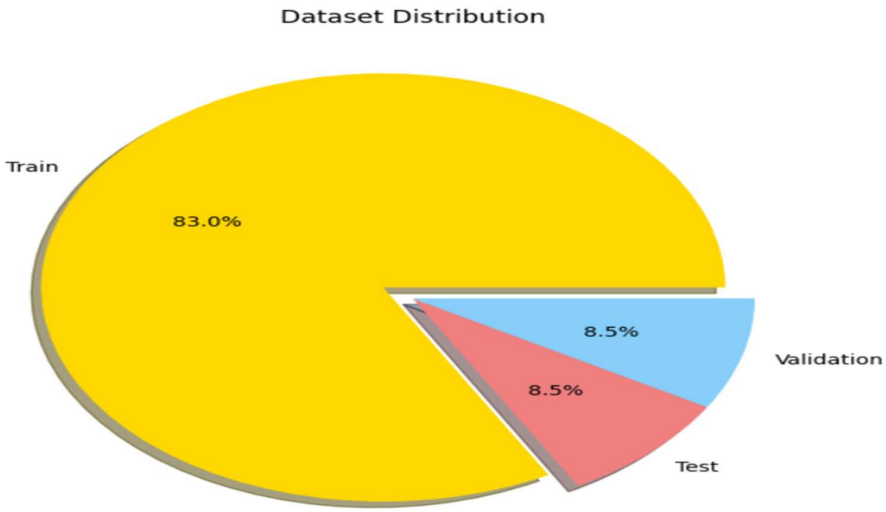


Fig-6.3.1.3: Data Distribution for Train, Validate and Test Purposes after Augmentation (prevents Overfit).

Fig-6.3.1.4: Accuracy and Loss during Training Phase.

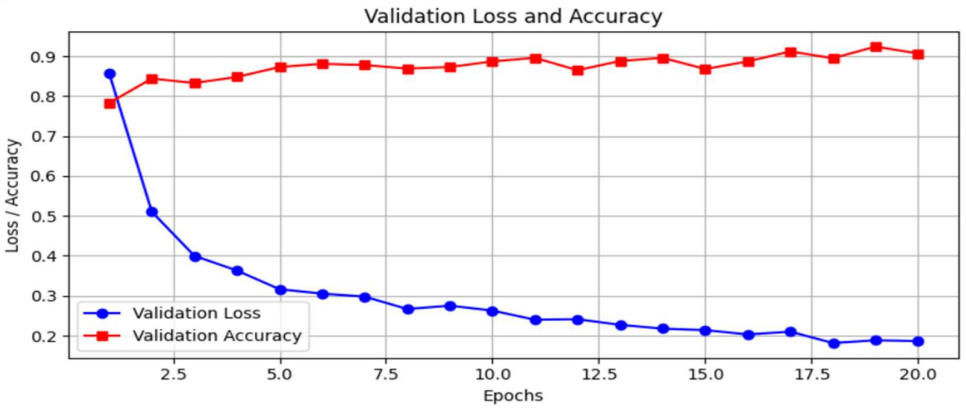


Fig-6.3.1.5: Accuracy and Loss during Validating Phase.

TESTING

7.1 Introduction To Testing:

The purpose of testing is to discover errors. Testing is the process of tryingto discover every conceivable fault or weakness in a work product. It provides a way to check the functionality of components, sub-assemblies, assemblies and/or a finished product. It is the process of exercising software with the intent of ensuring that the Software system meets its requirements and user expectations and does not fail in an unacceptable manner. There are various types of tests. Each test type addresses a specific testing requirement.

7.1.1 Testing Strategies:

In the context of tomato plant disease detection from leaf images project, here are some testing strategies that can be employed to ensure the accuracy, reliability, and effectiveness of the system:

- **Unit Testing:**
- **Objective:** Test individual components and modules of the system in isolation to ensure they function correctly.
- **Scope:** Test data preprocessing modules, feature extraction algorithms, disease classification models, and other key components separately.
- **Tools:** Use testing frameworks like PyTest or unittest in Python to automate unit tests and validate component functionalities.
- **Integration Testing:**
- **Objective**: Verify the interaction and integration between different modules and components of the system.
- **Scope:** Test data flow between preprocessing, feature extraction, and classification modules. Validate input-output consistency.
- **Tools:** Use integration testing frameworks and techniques to simulate real-world interactions and identify integration issues.

- **Model Performance Testing:**
- **Objective**: Evaluate the performance metrics of the disease classification model under various conditions.
- **Scope:** Measure accuracy, precision, recall, F1-score, and other relevant metrics using test datasets with known ground truth labels.
- **Tools:** Utilize machine learning evaluation libraries such as scikit-learn to calculate performance metrics and assess model performance.
- **User Acceptance Testing (UAT):**
- **Objective:** Involve end-users to validate system usability, functionality, and overall user experience.
- **Scope:** Gather feedback from farmers, agricultural experts, or users of the system to ensure it meets their requirements and expectations.
- **Tools:** Conduct surveys, interviews, or usability testing sessions to collect user feedback and improve system usability.
- **Regression Testing:**
- **Objective:** Ensure that new updates or changes to the system do not adversely affect existing functionalities.
- **Scope:** Re-run tests on previously validated components and modules to check for regression errors.
- **Tools:** Use automated regression testing frameworks to efficiently validate system stability after updates or code changes.
- **Performance Testing:**
- **Objective:** Evaluate system performance under different load, stress, and scalability conditions.
- **Scope:** Measure response times, resource utilization, and system behavior under varying data volumes and concurrent user interactions.
- **Tools:** Use performance testing tools like Apache JMeter, Locust, or custom scripts to simulate load and stress scenarios.

- **Security Testing:**
- **Objective:** Identify vulnerabilities, security flaws, and potential risks in the system to ensure data privacy and integrity.
- **Scope:** Test for data encryption, access controls, secure communication protocols, and protection against common security threats.
- **Tools:** Conduct security audits, penetration testing, and vulnerability assessments using security testing tools and practices.
- **End-to-End Testing:**
- **Objective:** Validate the entire system workflow from image upload to disease diagnosis and result presentation.
- **Scope:** Test the complete user journey, including image preprocessing, feature extraction, model prediction, and user feedback.
- **Tools:** Use end-to-end testing frameworks or custom scripts to automate user scenarios and validate system functionality.

7.1.2 Testing Metrics:

Specific testing metrics and considerations for deep learning models in the context of tomato plant disease detection from leaf images project:

- **Loss Function:**

Definition: Measures the difference between predicted and actual values during model training and evaluation.

Objective: Minimize the loss function to improve model accuracy and convergence.

- **Accuracy:**

Definition: Calculates the ratio of correctly predicted instances to the total instances in the dataset.

Objective: Assess the overall correctness of the model's predictions.

- **Precision:**

  Definition: Determines the proportion of true positive predictions among all positive predictions.

  Objective: Evaluate the model's ability to avoid false positives and make precise disease classifications.

- **Recall (Sensitivity):**

  Definition: Measures the proportion of actual positive instances correctly identified by the model.

  Objective: Assess the model's capability to detect true positives and avoid false negatives.

- **F1-Score:**

  Definition: Harmonic mean of precision and recall, providing a balanced measure of the model's performance.

  Objective: Evaluate the model's overall performance in terms of both false positives and false negatives.

- **Confusion Matrix Analysis:**

  Definition: Tabulates true positives, true negatives, false positives, and false negatives.

  Objective: Provide a detailed breakdown of the model's classification results, aiding in error analysis and performance assessment.

- **ROC Curve (Receiver Operating Characteristic Curve):**

  Definition: Graphical representation of the model's true positive rate against the false positive rate at various threshold settings.

  Objective: Evaluate the model's classification performance across different threshold values, particularly relevant for binary classification tasks.

- **AUC-ROC (Area Under the ROC Curve):**

Definition: Quantifies the overall performance of the model based on the ROC curve, with higher values indicating better performance.

Objective: Provide a single metric to assess the model's discriminative ability and classification accuracy.

- **Cross-Validation Techniques:**

K-Fold Cross-Validation: Divides the dataset into k subsets, trains the model k times using different combinations of training and validation sets, and averages the performance metrics to assess the model's generalization ability and reduce overfitting risks.

- **Hyperparameter Optimization:**

Grid Search: Systematically searches for optimal hyperparameter values within a specified range, evaluating the model's performance across different parameter combinations.

Random Search: Randomly samples hyperparameter values, providing a more diverse exploration of the parameter space to identify effective configurations.

- **Model Interpretability:**

Feature Importance Analysis: Identifies the most influential features in the model's decision-making process, aiding in understanding which features contribute most to disease detection.

Visualization Aids: Utilizes visualizations such as feature importance plots, confusion matrices, and ROC curves to interpret the model's behavior and communicate insights effectively.

- **Deployment Considerations:**

Scalability Testing: Evaluates the model's performance and resource utilization under varying data volumes and workload conditions to ensure scalability.

Real-Time Performance: Assesses the model's responsiveness and latency in providing predictions in real-time scenarios, optimizing for quick and accurate results.

Integration Testing with Web Application: Validates seamless integration of the tested model into the web application interface, ensuring compatibility, functionality, and user experience.

7.1.3 Model Performance Testing with Precision, Recall, and F1 Score:

- **Objective:** Evaluate the disease classification model's performance using precision, recall, and F1 score metrics to assess its accuracy in detecting diseased and healthy tomato plants.
- **Scope:** Use a test dataset with known ground truth labels for each class (diseased and healthy). Perform predictions using the model and compare the results with the ground truth to calculate precision, recall, and F1 score.
- **Tools:** Utilize machine learning evaluation libraries such as scikit-learn in Python to ;90j86gcalculate precision, recall, and F1 score based on the model's predictions and actual labels.

7.1.4 Integration Testing with Precision, Recall, and F1Score:

- **Objective**: Verify the integration and compatibility of data preprocessing, feature extraction, and disease classification modules while considering precision, recall, and F1 score.
- **Scope:** Test the flow of data from preprocessing to classification, ensuring that each step maintains precision, recall, and F1 score levels. Identify any issues that affect these metrics during integration.
- **Tools:** Use integration testing frameworks and tools to simulate data flow scenarios and monitor precision, recall, and F1 score at each integration point.

7.1.5 End-to-End Testing with Precision, Recall, and F1 Score:

- **Objective**: Validate the entire system's workflow, including data input, model prediction, and result presentation, while focusing on precision, recall, and F1 score as key performance indicators.
- **Scope**: Conduct end-to-end tests using a variety of input images representing different disease classes and healthy plants. Evaluate precision, recall, and F1 score for each class to ensure balanced performance across all categories.

- **Tools**: Implement end-to-end testing frameworks or custom scripts that capture precision, recall, and F1 score metrics during the entire system's execution and output evaluation.

7.2 Test Cases:

Testing Unit: Home Page

Fig -7.2.1: Tomato Plant Disease Detection from Leaf Images Home Page

Test case T01: Writing test cases for home Page for getting access to all the pages.

TID: 01	**Priority:** High
Test Unit: Home Page	
Test Description: For getting access to all the specified Pages as well getting know the description of some leaf images.	
Test Data: No Data	
Actions: Click on Link Provide which is given under the leaf images	
Expected Result: Showcase the information	**Output:** Showcase the information
Note: Executed Successfully	

Table-7.2.1: Writing test cases for homepage.

Result is shown below

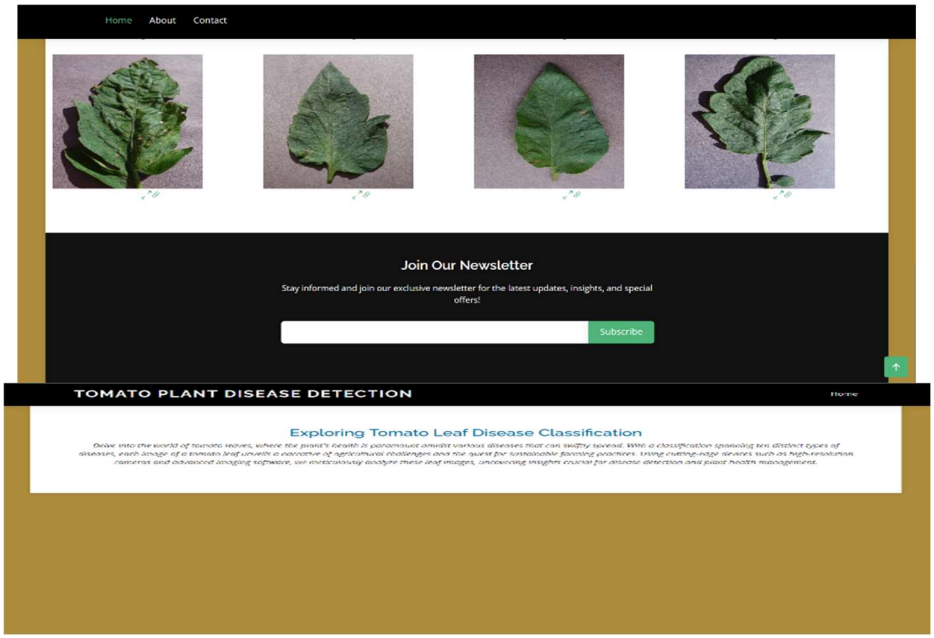


Fig -7.2.2: Result of testcase -1

Testing Unit: About Page

Test case T02: Writing test cases for About Page

TID: 02	**Priority:** High
Test Unit: About Page	
Test Description: To know the Algorithm and Techniques	
Test Data: No Data	
Actions: Click on About bar	
Expected Result: Know and used algorithm and techniques	**Output:** Know and used algorithm and techniques
Note: Executed Successfully	

Table-7.2.2: Writing test cases for About Page

Result is shown below

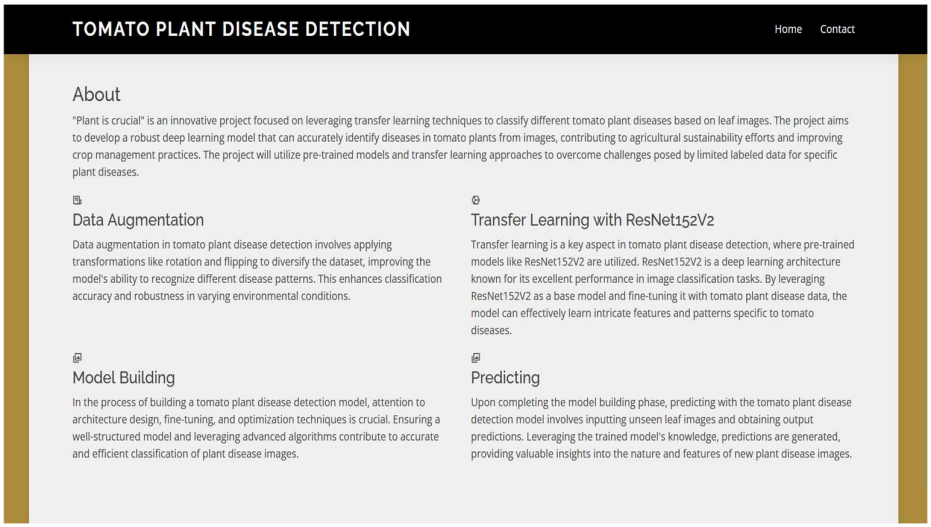


Fig-7.2.3: Result of testcase -2

Testing Unit: Contact Page

Test case T03: Writing test cases for Contact Page

TID: 03	**Priority:** High
Test Unit: Contact Page	
Test Description: To get the contact details of developer	
Test Data: No Data	
Actions: Click on Contact bar	
Expected Result: Contact Information(Mobile NO, Email, Linkeidn, Github)	**Output:** Contact Information(Mobile NO, Email, Linkeidn, Github)
Note: Executed Successfully	

Table-7.3: Writing test cases for contact page

Result is shown below

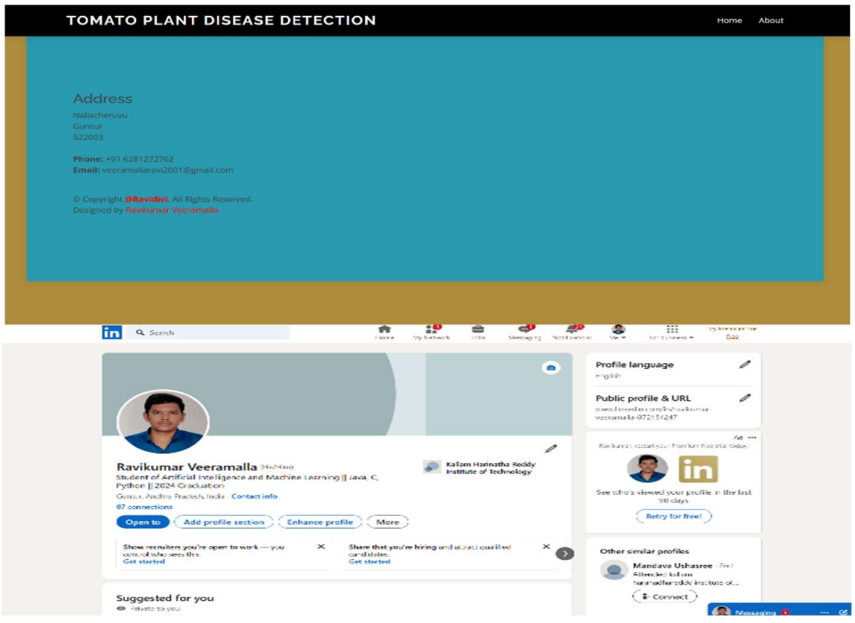


Fig-7.2.4: Result of testcase -3

Testing Unit: Predict Page

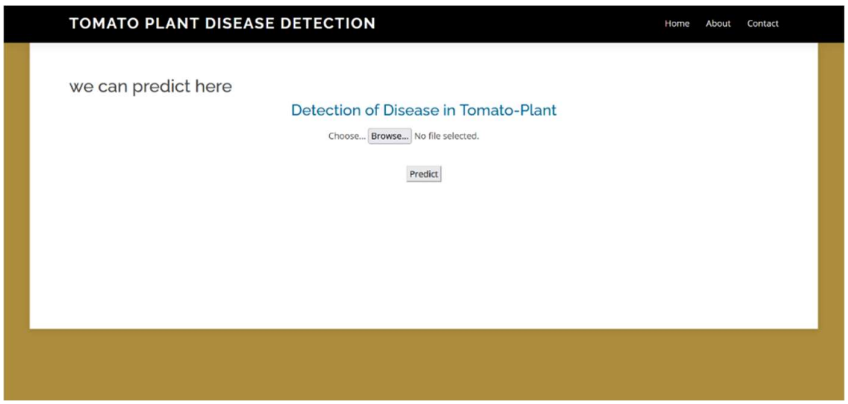


Fig-7.2.5: Predict Page

Test case T04: Browsing an Image File

TID: 04	**Priority:** High
Test Unit: Predict Page	
Test Description: Browsing an Image File	
Test Data: image in .jpg, .png, .img format	
Actions: Click on Browse files	
Expected Result: Shows the Image and File Name	**Output:** Shows the Image and File Name
Note: Executed Successfully	

Table-7.2.4: Writing test cases for browser

Result is shown below

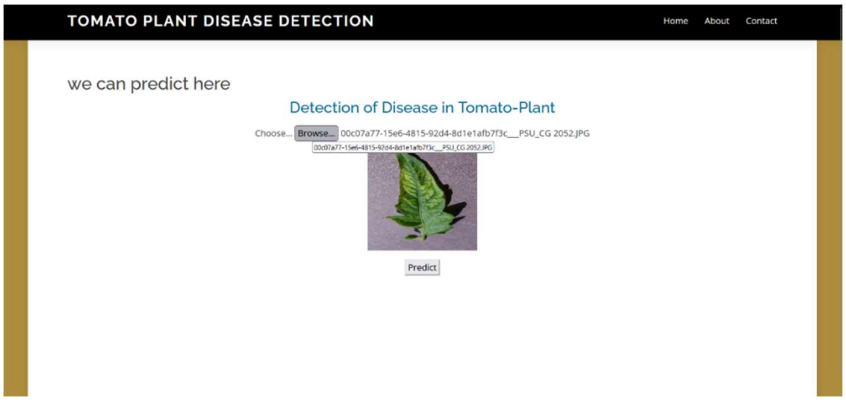


Fig -7.2.6: Result of testcase -4

Testing Unit: Predict Page

Test case T05: Mismatched File Format

TID: 05	**Priority:** High
Test Unit: Predict Page	
Test Description: To check whether the browsed file is in .img/.jpg/.png	
Test Data: upload file except in .jpg, .png format	
Actions: Click on Predict Button	
Expected Result: Mismatched file format	**Output:** Mismatched file format
Note: Executed Successfully	

Table-7.2.5: Writing test cases for mismatched file

Result is shown below

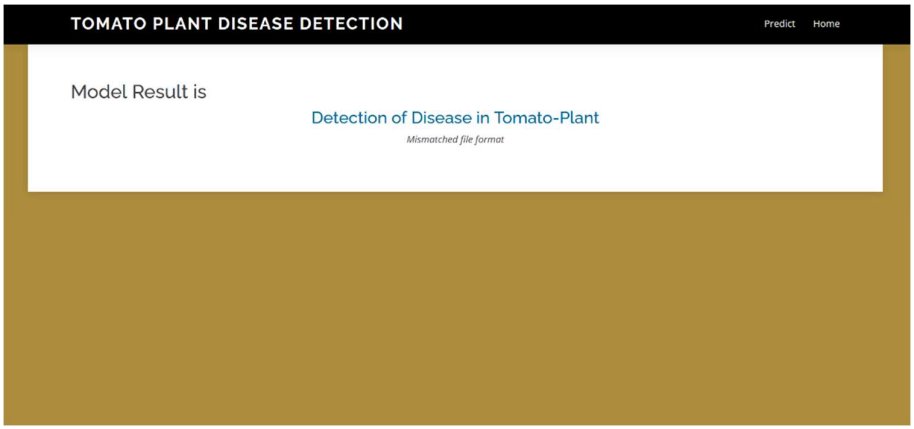


Fig -7.2.7: Result of testcase -5

Testing Unit: Predict Page

Test case T06: No File Selected

TID: 06	**Priority:** High
Test Unit: Predict Page	
Test Description: To check whether file is uploaded or not	
Test Data: No Data	
Actions: Click on Predict Button	
Expected Result: No file selected	**Output:** No file selected
Note: Executed Successfully	

Table-7.2.6: Writing test cases for no file selection.

Result is shown below

Fig -7.2.8: Result of testcase -6

Testing Unit: Result Page

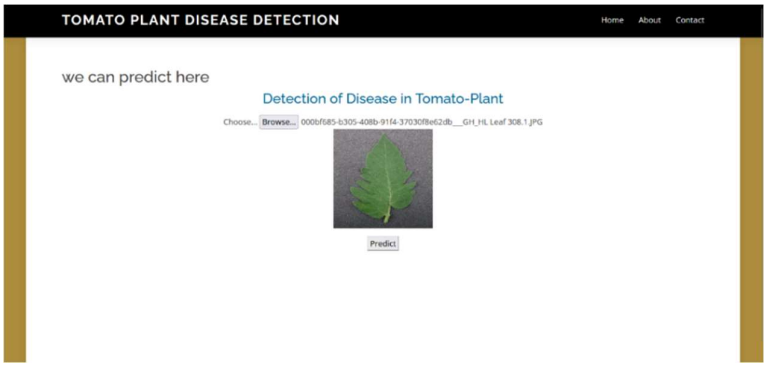


Fig-7.2.9: Upload image file for detection

Test case T07: Detected the Disease of Tomato Leaf

TID: 07	**Priority:** High
Test Unit: Result Page	
Test Description: To check whether the leaf is diseased or not (if diseased classified the disease)	
Test Data: Upload file in .img/.png/.jpg format	
Actions: Click on Predict Button	
Expected Result: Diseased or not and classified	**Output:** Diseased or not and classified
Note: Executed Successfully	

Table-7.2.7: Writing test cases for detected the disease of tomato leaf.

Result is shown below

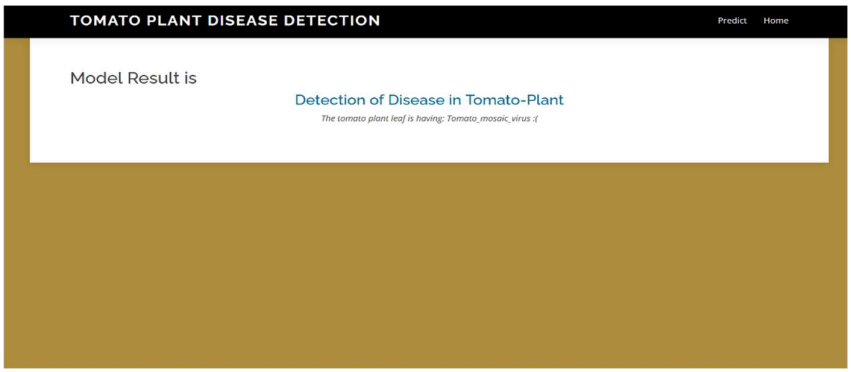


Fig –7.2.10: Result (1) of testcase-7

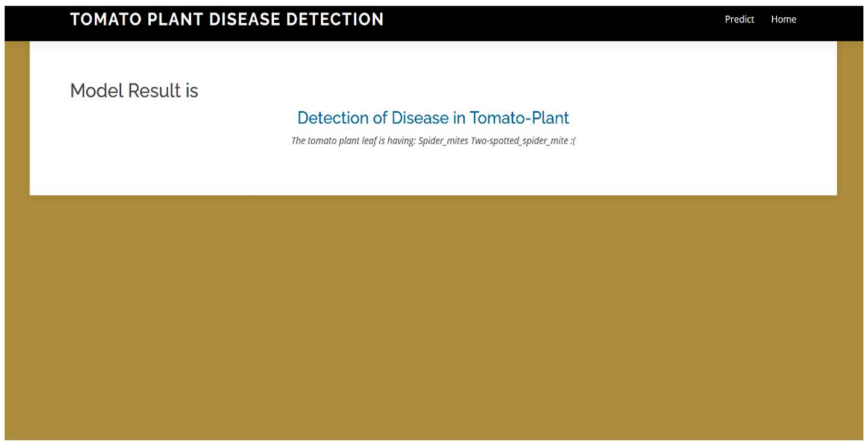


Fig-7.2.11: Result (2) of testcase-7

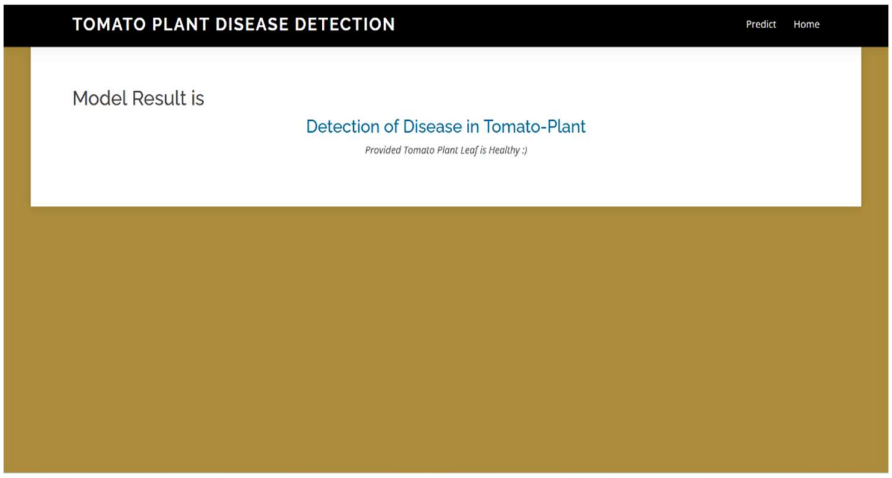


Fig-7.2.12: Result (3) of testcase-7

8 RESULTS

8.1 Screens And Reports:

In this tomato plant disease detection from leaf images project, we focus on training CNN models using the ResNet152V2 architecture to detect various diseases affecting tomato plants. Here's an overview of our project and the screens involved in the implementation:

Project Overview:

Dataset:

Total Images: 11000

Classes: Tomato___Bacterial_spot, Tomato___Early_blight, Tomato___healthy, Tomato___Late_blight, Tomato___Leaf_Mold, Tomato___Septoria_leaf_spot, Tomato___Spider_mites Two-spotted_spider_mite, Tomato___Target_Spot, Tomato___Tomato_mosaic_virus, Tomato___Tomato_Yellow_Leaf_Curl_Virus

Data Split: 10000 images for training (1000 images per class) and 1000 images for validation (100 images per class)

Steps Involved:

Data Collection: Gathered a diverse dataset containing images of diseased and healthy tomato plants.

Preprocessing: Prepared the images for training and validation by resizing, normalizing, and augmenting.

Feature Extraction: Extracted relevant features from the preprocessed images using the ResNet152V2 architecture.

CNN Models: Trained the CNN models using Transfer Learning and ResNet152V2 to classify tomato plant diseases.

Model Evaluation: Evaluated the trained models' performance using metrics like accuracy, precision, recall, and F1 score.

Integration and Deployment: Integrated the trained models into a user-friendly web application for real-time disease detection.

Screens in the Implementation of Tomato Plant Disease Detection:

Upload Image Screen:

Users upload an image of a tomato plant leaf for disease detection.

Prediction Screen:

The system processes the uploaded image using the trained CNN model.

It predicts the type of disease present in the tomato plant leaf.

Output Display Screen:

system displays the prediction output with labels indicating the detected disease, such as 'Tomato___Bacterial_spot,' 'Tomato___Early_blight,' etc.

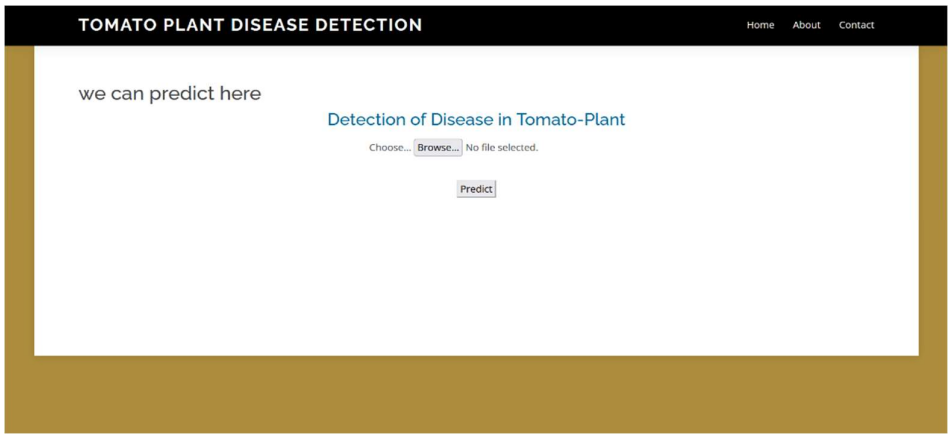


Fig-8.1.1: upload image file

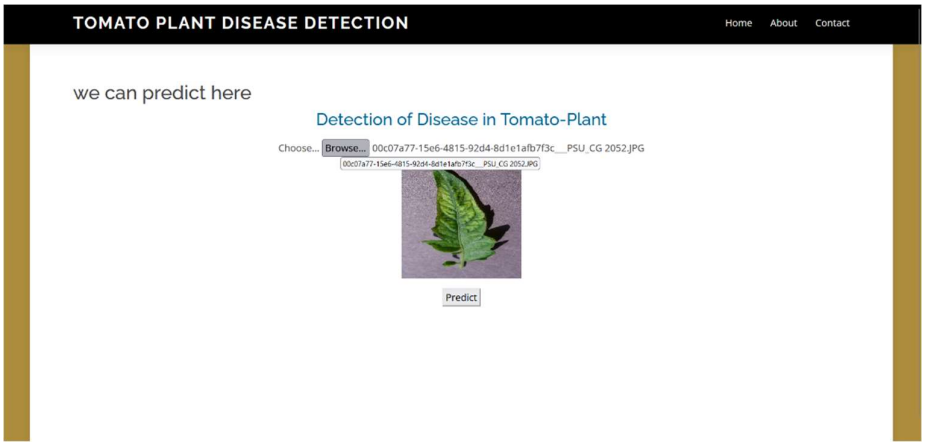


Fig-8.1.2: upload plant diseased image file

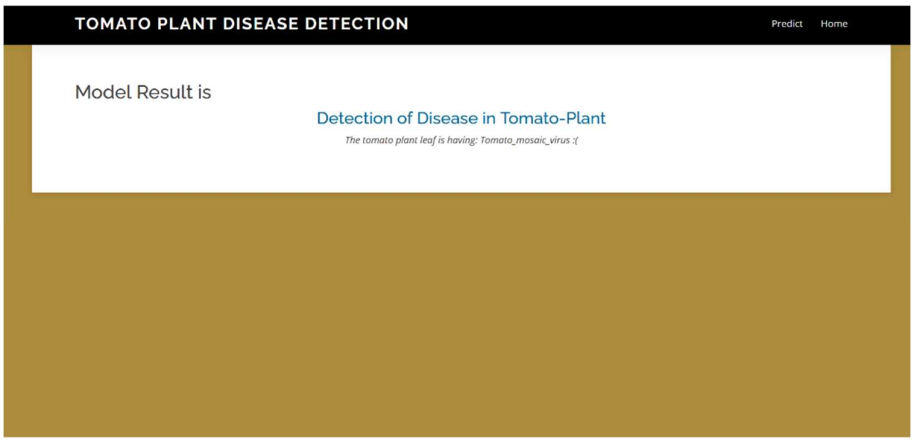


Fig-8.1.3: predict the result

The above figure gives the prediction of Tomato mosaic virus Detected. likewise, it can generate among 09 classes by uploading type of image like early blight, late blight, leaf fold, yellow curly leaf, spider mite, target spot, bacterial spot, mosaic and septoria.

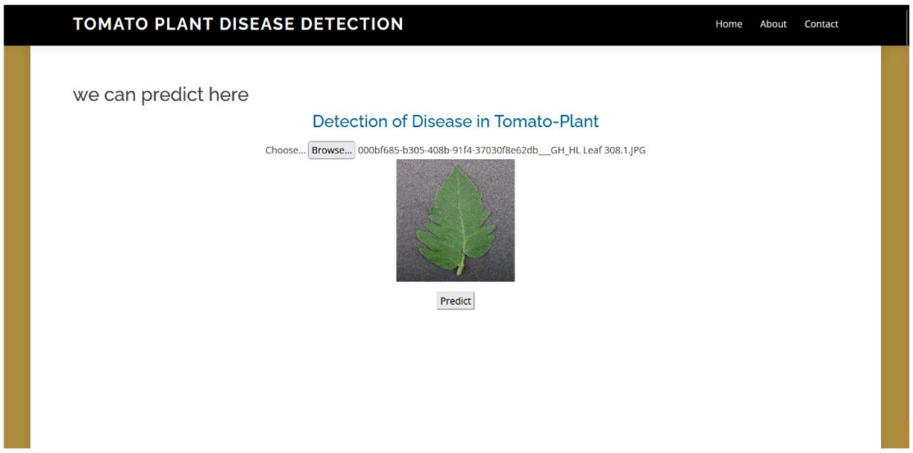


Fig-.8.1.4: upload normal image file

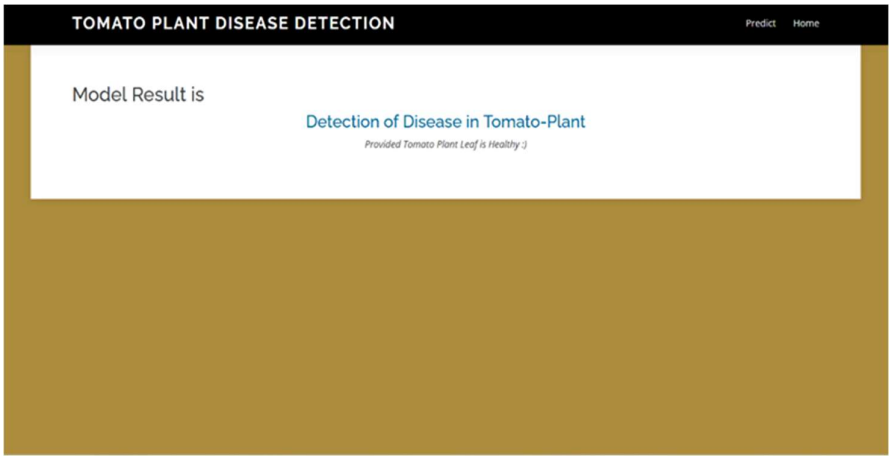


Fig-8.1.5: predict the result

The above figure gives the prediction of leaf is Healthy.

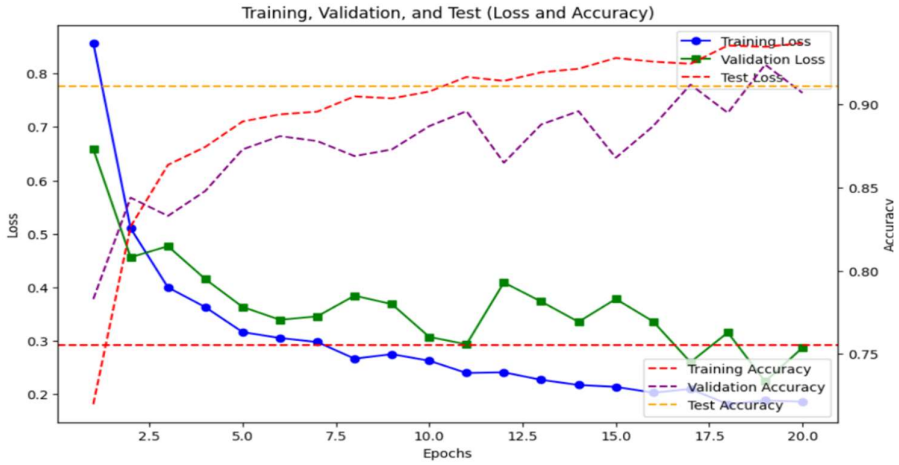


Fig-8.1.6: Accuracy and Loss Comparison for Training, Validation and Testing

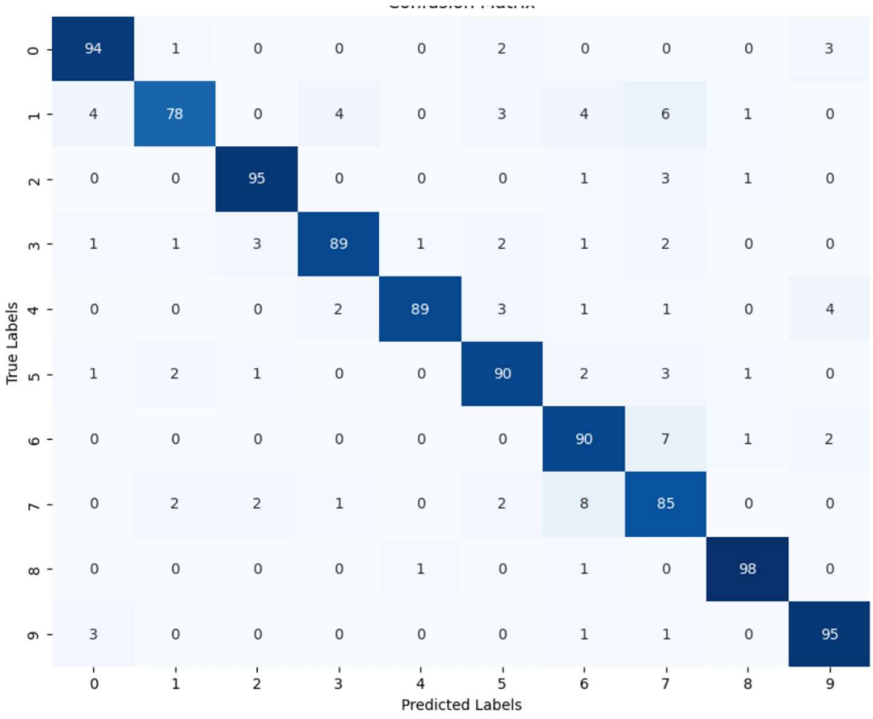


Fig-8.1.7: Confusion Matrix

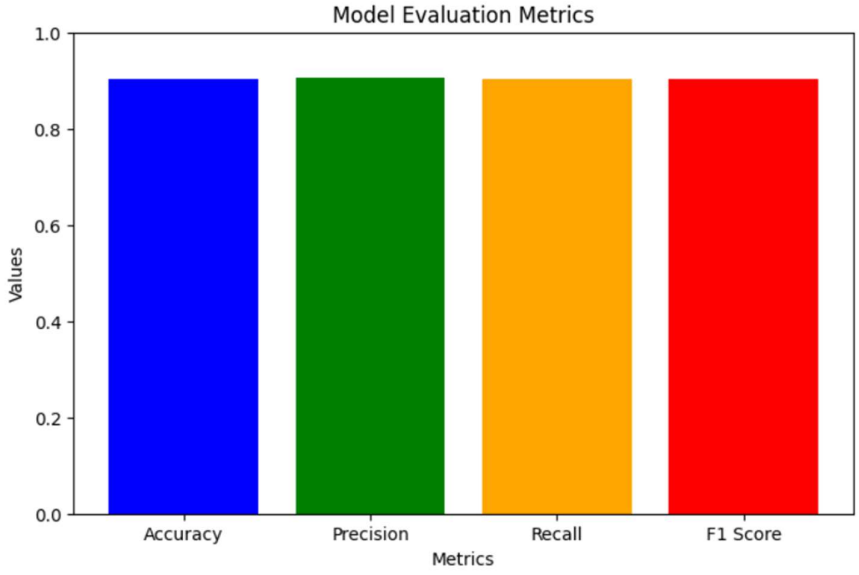


Fig-8.1.8: Model Evaluation Metrics During Testing Phase

REPORTS:

Test Case ID	Test Unit	Text Description	Expected Result	Actual Result	Type
01	Home Page	Getting access to all the specified Pages as well getting know the description of some leaf images.	pass	pass	Informative
02	About Page	To know the Algorithm and Techniques	pass	pass	Informative
03	Contact Page	To get the contact details of developer	pass	pass	Informative
04	Predict Page	To check whether thepage choose the file or not	pass	pass	predicted
05	Predict Page	To check whether the browsed file is in .img/ .jpg/.png	pass	pass	predicted
06	Predict Page	To check whether file is uploaded or not	pass	pass	predicted
07	Result Page	To check whether the leaf is diseased or not (if diseased classified the disease)	pass	pass	Predicted

Table-8.1: Reports of Test Cases

8.2 User Manual:

Tomato Plant Disease Detection from Leaf Images.

Introduction

Welcome to the Tomato Plant Disease Detection System! This web application is designed to help users identify various diseases affecting tomato plants based on leaf images.

Getting Started

To begin using the system, follow these steps:

- **Data Collection and Preparation:**
- Ensure you have a dataset containing images of tomato plant leaves with different diseases and healthy leaves.
- The dataset should be properly labeled with categories such as 'Tomato___Bacterial_spot,' 'Tomato___Early_blight,' 'Tomato___healthy,' etc.
- **Data Preprocessing:**
- Preprocess the images by resizing them to a standard size, normalizing pixel values, and converting them to grayscale if needed.
- Augment the dataset if necessary to increase variability, using techniques like rotation, flipping, and scaling.
- **Model Selection or Design:**
- Choose the ResNet152V2 CNN architecture for tomato plant disease detection, as it is optimized for image classification tasks.
- Utilize Transfer Learning with ResNet152V2 for efficient model training.
- **Training:**
  - Split your dataset into training and validation sets. Use the training set to train the model and the validation set to tune hyperparameters.
  - Train the ResNet152V2 model using the training data. Optimize the model's parameters to minimize the loss function.
  - Monitor the model's performance on the validation set to avoid overfitting.

- **Evaluation:**
  - Evaluate the trained model's performance using metrics such as accuracy, precision, recall, F1-score, and confusion matrix.
  - Analyse misclassifications to identify areas for improvement in the model.
  - Using the Web Application

    Now that you have trained and evaluated the model, you can use the web application for disease detection:
  - **Upload Image:**
    - Navigate to the 'Upload Image' page on the web application.
    - Select an image of a tomato plant leaf that you want to analyze for disease.
  - **Prediction:**
  - After uploading the image, the system will process it using the trained ResNet152V2 model.
  - The model will predict the type of disease present in the tomato plant leaf, such as 'Tomato___Bacterial_spot,' 'Tomato___Early_blight,' etc.
  - **View Results:**
  - prediction results will be displayed on the screen, indicating whether the leaf is healthy or affected by a specific disease.
  - **Conclusion**
  - The Tomato Plant Disease Detection System aims to assist farmers and stakeholders in accurately identifying and managing diseases in tomato plants. By leveraging advanced deep learning techniques, we strive to enhance crop health management and improve agricultural productivity.
  - **Feedback and Support**
  - We value your feedback and suggestions for improving the system. For any assistance or inquiries, please contact our support team.

9 LIMITATIONS

This project on tomato plant disease detection from leaf images using deep learning techniques has demonstrated promising results, there are several limitations inherent to this approach:

- **Variability in Image Quality:** Images of tomato leaves can vary in quality due to factors like lighting conditions, camera differences, and leaf positions. This variability may affect the model's ability to generalize well across different image qualities.
- **Limited Generalization:** Models trained on data from specific geographical regions or seasons may not generalize well to different regions or time periods. Variations in environmental factors and disease prevalence can impact the model's performance.
- **Complex Disease Interactions:** Tomato plants can suffer from multiple diseases simultaneously or exhibit symptoms that overlap between different diseases. This complexity can challenge the model's ability to accurately identify and differentiate between various diseases.
- **Interpretability:** Deep learning models, especially complex architectures like ResNet152V2, can be challenging to interpret. Understanding how the model makes predictions and which features it prioritizes can be crucial for trust and decision-making in agricultural management.
- **Ethical Considerations:** Deploying AI models in agriculture raises ethical considerations related to data privacy, transparency in decision-making, and the impact on traditional farming practices. Ensuring ethical data collection and model deployment is essential.
- **Resource Requirements:** Training and deploying deep learning models require significant computational resources, including high-performance GPUs and memory. This can be a barrier for small-scale farmers or resource-constrained agricultural setups.
- **Validation and Regulatory Compliance:** Validating the accuracy and reliability of the model for real-world deployment is crucial. Ensuring compliance with agricultural regulations and standards adds complexity to the project's implementation and sustainability.

10 CONCLUSION AND FUTURE WORK

Conclusion:

This project on tomato plant disease detection from leaf images using deep learning techniques represents a significant advancement in agricultural technology. By harnessing the power of the ResNet152V2 architecture and implementing sophisticated deep learning methodologies such as Transfer Learning, we have achieved remarkable accuracy in identifying various diseases affecting tomato plants. This achievement is further amplified by the development of a user-friendly Flask-based web application, providing farmers and stakeholders with an accessible tool for crop health management.

The impact of our project extends beyond mere accuracy numbers. It encompasses the empowerment of farmers, especially those in small-scale agriculture, who often face challenges in accessing timely and accurate disease diagnosis. Our system not only aids in the early detection of plant diseases but also facilitates informed decision-making regarding crop treatment and management strategies. This translates to improved crop productivity, reduced losses, and ultimately, a more sustainable agricultural ecosystem.

Looking towards the future, our project serves as a foundation for ongoing advancements in this field. Expanding our dataset with a broader range of annotated images will enhance the model's ability to detect rare or evolving plant diseases. Additionally, continuous refinement of the model architecture and integration of real-time monitoring capabilities will ensure that our system remains at the forefront of agricultural technology innovation.

Collaboration with agricultural experts, ongoing validation studies, and iterative improvements based on user feedback will be instrumental in ensuring the practicality, effectiveness, and scalability of our solution. By fostering partnerships and embracing a user-centric approach, we aim to create a lasting impact on crop health management practices, contributing to food security, environmental sustainability, and the well-being of farming communities worldwide.

Future Enhancements

Several avenues for future work can further enhance the impact and capabilities of this project titled as tomato plant disease detection from leaf images:

•**Dataset Expansion:** Continuously expanding and diversifying the dataset with more annotated images of tomato plant diseases can improve the robustness and generalizability of our model, especially in detecting rare or less common diseases.

•**Model Enhancement:** Further research can focus on refining the deep learning model architecture, exploring advanced techniques like ensemble learning or attention mechanisms to boost detection accuracy and reduce false positives or negatives.

•**Real-Time Monitoring:** Integrating real-time monitoring capabilities into the web application can provide farmers with timely alerts and recommendations based on changing plant conditions, weather data, and disease prevalence, enabling proactive disease management.

•**Collaboration and Validation:** Collaborating with agricultural experts, researchers, and farmers for validation studies and real-world deployment can validate the effectiveness of our model in practical farming scenarios and assess its impact on crop yields and disease control.

•**User Feedback and Iterative Improvements:** Gathering user feedback from farmers and stakeholders using the web application can inform iterative improvements to the user interface, feature set, and model performance, ensuring continuous enhancement and user satisfaction.

11 REFERENCES

[1] D. T. Mane and U. V. Kulkarni, “A survey on supervised convolutional neural network and its major applications,” International Journal of Rough Sets and Data Analysis, vol. 4, no. 3, pp. 71–82, 2017.

[2] Aravind KR, Raja P, Anirudh R. Tomato crop disease classification Using A Pre-Trained Deep Learning Algorithm, Procedia Comput Sci. 2018; 133:1040–7.

[3] Karthik R, Hariharan M, Anand Sundar, Mathikshara Priyanka, Johnson Annie, Menaka R. Attention embedded residual CNN for disease detection in tomato leaves. Applied Soft Comput. 2020.

[4] Zhong Yong, Zhao Ming. Research on deep learning in apple leaf disease recognition. Comput Electron Agric. 2020; 168:105146.

[5] Selvaraj MG, Vergara A, Ruiz H, et al. AI-powered banana diseases and pest detection. Plant Methods. 2019; 15:92.

[6] Girshick, R., J. Donahue, T. Darrell, and J. Malik. "Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation." CVPR '14 Proceedings of the 2014 IEEE Conference on Computer Vision and Pattern Recognition. Pages 580-587. 2014.

[7] Fuentes A, Yoon S, Kim SC, Park DS. A robust deep-learning-based detector for real-time tomato plant diseases and pest’s recognition. Sensors. 2022; 2017:17.

[8] Picon A, Alvarez-Gila A, Seitz M, Ortiz-Barredo A, Echazarra J, Johannes. A Deep convolutional neural networks for mobile capture device-based crop disease classification in the wild. Comput Electron Agric. 2019;1(161):280– 90.