A

**Capstone Project Report**

On

**Title of the project**

Submitted during 4th semester in partial fulfilment of the requirements for the award of degree of

**Bachelor of Technology**

in

**Computer Engineering**

by

**Kunwar Uttkarsh Singh (22001003069)**

Under supervision of

**Ms. Dimple**



**Department of Computer Engineering**

**FACULTY OF INFORMATICS & COMPUTING**

**J.C. Bose University of Science & Technology, YMCA**

**Faridabad – 121006**

**May 2024**

**CANDIDATE’S DECLARATION**

I hereby certify that the work being presented in this Project titled **“FINE IMAGE CLASSIFICATION USING ML**”in partial fulfilment of the requirements for the degree of Bachelor of Technology in Computer Engineering and being submitted to “J.C. Bose University of Science and Technology, YMCA, Faridabad”, is an authentic record of my own work carried out under the supervision of Dr. Amita Arora.

The work contained in this project has not been submitted to any other University or Institute for the award of any degree or diploma by me.

**Student’s Signature**

**Student’s Name**

**Supervisor Name and Signature**

**TABLE OF CONTENTS**

**Candidate’s Declaration**

**Chapter 1: Introduction**

1.1 Brief of the Project

1.2 Technologies Used

**Chapter 2: Proposed Work**

3.1 Problem Formulation

3.2 Objectives

3.3. Methodology

**Chapter 3: Result Analysis and Future scope**

**References**

**Chapter 1: Introduction**

* 1. **Brief of the Project**

The Plant Disease Recognition System is an innovative solution designed to revolutionize the process of identifying and managing plant diseases. With the goal of safeguarding crops and ensuring optimal agricultural productivity, this system leverages advanced technologies such as machine learning, image processing, and web development to provide accurate and efficient disease detection capabilities.

Using a comprehensive dataset comprising approximately 87,000 RGB images of healthy and diseased crop leaves, categorized into 38 different classes, the system employs state-of-the-art convolutional neural networks (CNNs) trained on TensorFlow and Keras to analyze and classify plant images. By harnessing the power of deep learning algorithms, the system can accurately identify signs of diseases in plants, enabling farmers and agricultural practitioners to take timely and appropriate action to mitigate crop losses.

The user-friendly web interface, developed using Streamlit, allows users to upload images of plants suspected of having diseases and receive instant analysis and recommendations. With a simple and intuitive interface, users can navigate the system seamlessly, accessing results within seconds and facilitating prompt decision-making in agricultural management practices.

The Plant Disease Recognition System stands as a testament to the potential of technology in addressing critical challenges in agriculture. By providing farmers with a reliable and efficient tool for disease detection, the system empowers them to proactively manage and protect their crops, ultimately contributing to food security and sustainability on a global scale.

**1.2 Technologies Used**

The Plant Disease Recognition System employs a sophisticated blend of technologies, each contributing to its effectiveness and efficiency:

Python: As a versatile and widely-used programming language, Python serves as the backbone of the system's backend logic and algorithm implementation. Its extensive libraries and frameworks, such as NumPy, Pandas, and Scikit-learn, facilitate data manipulation, machine learning model development, and integration with other technologies.

Visual Studio Code (VS Code): A powerful and intuitive integrated development environment (IDE), VS Code provides developers with a robust set of tools for writing, debugging, and testing code. Its rich ecosystem of extensions enhances productivity and streamlines the development workflow, ensuring code quality and maintainability.

Convolutional Neural Network (CNN): CNNs are a class of deep learning models specifically designed for processing structured grid-like data, such as images. By leveraging convolutional layers, pooling layers, and fully connected layers, CNNs can automatically learn hierarchical representations of image features, enabling accurate classification and recognition of plant diseases.

TensorFlow: Developed by Google Brain, TensorFlow is an open-source deep learning framework widely used for building and training neural network models. Its flexible architecture supports both high-level and low-level APIs, making it suitable for a wide range of machine learning tasks. TensorFlow's efficient computation capabilities, distributed training support, and extensive ecosystem of tools and libraries make it an ideal choice for developing complex CNN models.

Keras: Built on top of TensorFlow, Keras provides a user-friendly interface for constructing and training deep learning models with minimal code. Its high-level abstractions and modular design enable rapid prototyping and experimentation, allowing developers to focus on model architecture and experimentation rather than low-level implementation details. Keras seamlessly integrates with TensorFlow, providing access to its powerful backend while offering a simplified and intuitive API for building CNN architectures.

Machine Learning: Machine learning techniques, including supervised learning, unsupervised learning, and transfer learning, play a crucial role in developing the disease recognition algorithms. Supervised learning algorithms, such as support vector machines (SVMs) and random forests, are used for training classifiers on labeled image data, while unsupervised learning techniques, such as clustering, may be employed for data exploration and feature extraction. Transfer learning, which involves leveraging pre-trained CNN models and fine-tuning them on domain-specific datasets, enables the efficient transfer of knowledge from related tasks, improving model performance with limited labeled data.

Image Processing: Image processing techniques are utilized to preprocess raw image data before feeding it into the machine learning models. Common preprocessing steps include resizing, normalization, and augmentation, which help standardize the input data and enhance the model's ability to learn relevant features. Image enhancement techniques, such as contrast adjustment and noise reduction, may also be applied to improve image quality and clarity, leading to more accurate disease detection results.

Streamlit: A Python library for building interactive web applications, Streamlit simplifies the development of data-driven applications by enabling developers to create and deploy web interfaces directly from Python scripts. With Streamlit, developers can easily integrate machine learning models, visualizations, and user inputs into web applications, allowing for rapid prototyping and deployment. Streamlit's intuitive API and automatic layout management streamline the development process, making it an ideal choice for creating the user interface of the Plant Disease Recognition System.

**Chapter 2: Proposed Work**

**2.1 Problem Formulation**

The Plant Disease Prediction Playlist aims to tackle the pressing issue of timely and accurate identification of plant diseases, which significantly impacts crop yield and agricultural productivity. The lack of efficient disease detection methods often results in substantial crop losses, affecting farmers' livelihoods and food security. By developing a robust deep learning model for plant disease recognition, this project seeks to empower farmers and agricultural practitioners with a tool that can aid in early disease detection, prompt intervention, and effective crop management strategies.

**2.2 Objectives**

**2.2.1 Dataset Exploration and Preprocessing:**

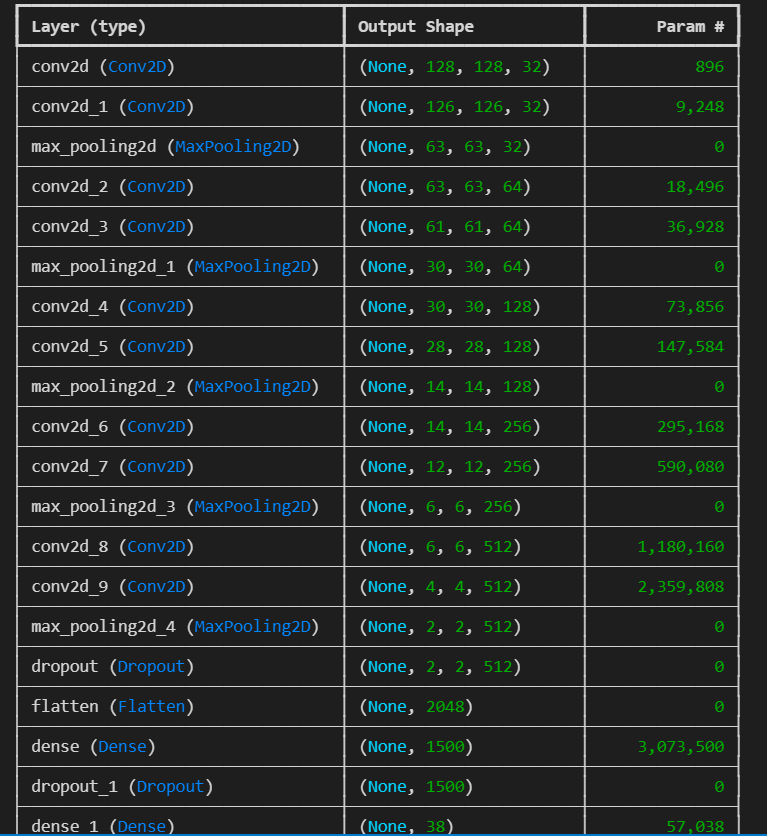
Comprehensive Dataset Analysis: Conducting a thorough exploration of the dataset to understand its structure, class distribution, and image characteristics. This includes analyzing the distribution of healthy and diseased samples across different classes and assessing potential biases or imbalances.

Image Preprocessing: Preprocessing the images to standardize their format, size, and quality. Common preprocessing techniques may include resizing, normalization, and noise reduction to ensure consistency and compatibility for model training.

**2.2.2 Model Architecture Design:**

Custom CNN Architecture: Designing a convolutional neural network (CNN) architecture tailored to the plant disease recognition task. The architecture will comprise multiple layers, including convolutional layers for feature extraction and pooling layers for spatial downsampling. Additional layers such as dropout and batch normalization may be incorporated to improve model generalization and prevent overfitting.

Hyperparameter Tuning: Experimenting with different hyperparameters, such as learning rate, batch size, and optimizer choice, to optimize model performance. Hyperparameter tuning involves systematically exploring the parameter space to identify the combination that yields the best results.



**2.2.3 Model Training and Evaluation:**

Training Process: Training the CNN model using the curated dataset, with a focus on monitoring training progress and performance metrics. This involves iteratively updating the model parameters using backpropagation and gradient descent optimization techniques.

Evaluation Metrics: Evaluating the trained model on both training and validation datasets to assess its performance. Common evaluation metrics include accuracy, precision, recall, and F1-score, which provide insights into the model's ability to correctly classify healthy and diseased plant images.

**2.2.4 Performance Analysis and Optimization:**

Model Performance Analysis: Analyzing the model's performance and identifying potential challenges such as underfitting, overfitting, and loss function overshooting. This involves visualizing training and validation metrics over epochs to diagnose performance issues and guide optimization efforts.

Optimization Strategies: Implementing strategies to address performance challenges and optimize model performance. These strategies may include architectural adjustments (e.g., adding or removing layers), regularizationtechniques (e.g., dropout, L2regularization), and learning rate scheduling to improve convergence and stability during training.

**2.2.5 Model Testing and Deployment:**

Testing Process: Testing the trained model on a separate set of test images to evaluate its real-world performance and generalization to unseen data. This involves assessing the model's ability to accurately classify images outside the training and validation datasets.

Deployment as a Web Application: Deploying the trained model as a user-friendly web application for plant disease prediction, enabling easy access and usability for end-users. The web application will allow users to upload images of plant leaves and receive real-time predictions of potential diseases, facilitating prompt intervention and crop management decisions.

**2.3 Methodology**

**2.3.1 Data Collection and Preprocessing:**

Curating Dataset: Collecting a comprehensive dataset of plant images, including healthy and diseased samples across multiple classes. The dataset will be carefully curated to ensure diversity and representativeness of plant species and disease types.

Image Preprocessing: Preprocessing the images to enhance quality and standardize input data for model training. This may involve resizing, normalization, and augmentation techniques to improve model robustness and performance.

**2.3.2 Model Development and Training:**

CNN Architecture Design: Designing a custom CNN architecture optimized for the plant disease recognition task. The architecture will be tailored to handle the complexity and variability of plant images, incorporating appropriate layers and activation functions.

Model Training: Training the CNN model using the curated dataset, with a focus on optimizing model parameters and minimizing training loss. This involves iteratively adjusting model weights through backpropagation and gradient descent optimization algorithms.

**2.3.3 Model Evaluation and Performance Analysis:**

Evaluation Metrics: Evaluating the trained model's performance on separate training and validation datasets using appropriate evaluation metrics. This includes calculating accuracy, precision, recall, and F1-score to assess model effectiveness in disease classification.

Performance Visualization: Visualizing training and validation metrics over epochs to analyze model convergence and identify performance trends. Visualization techniques such as learning curves, confusion matrices, and precision-recall curves will be employed to gain insights into model behavior.

**2.3.4 Model Optimization and Fine-Tuning:**

Hyperparameter Tuning: Conducting hyperparameter tuning experiments to optimize model performance. This involves systematically varying hyperparameters such as learning rate, batch size, and optimizer choice to identify the configuration that yields the best results.

Regularization Techniques: Applying regularization techniques such as dropout, batch normalization, and weight decay to prevent overfitting and improve model generalization. These techniques help to reduce model complexity and improve its ability to generalize to unseen data.

**2.3.5 Model Deployment and Usage:**

Web Application Development: Developing a user-friendly web application for plant disease prediction, enabling easy access and usability for end-users. The web application will feature an intuitive interface for uploading images and receiving real-time predictions.

**Chapter 3: Result Analysis and Future Scope**

**3.1 Result Analysis**

**3.1.1 Evaluation Metrics Overview**

Precision Analysis:

Precision scores provide valuable insights into the model's ability to accurately classify positive instances for each class. It indicates the proportion of correctly predicted positive instances out of all instances classified as positive. Analyzing precision scores across classes helps identify classes where the model exhibits high precision, indicating minimal false positive predictions, and classes where precision is lower, suggesting potential areas for improvement in minimizing misclassifications.

Recall Analysis:

Recall scores evaluate the model's effectiveness in capturing positive instances for each class. It measures the proportion of correctly identified positive instances out of all actual positive instances. High recall scores indicate that the model successfully captures most positive instances, while lower recall scores may indicate missed positive instances, leading to false negatives. Analyzing recall scores across classes helps identify classes where the model excels in capturing positive instances and classes where recall is lower, indicating potential areas for improvement in sensitivity to positive instances.

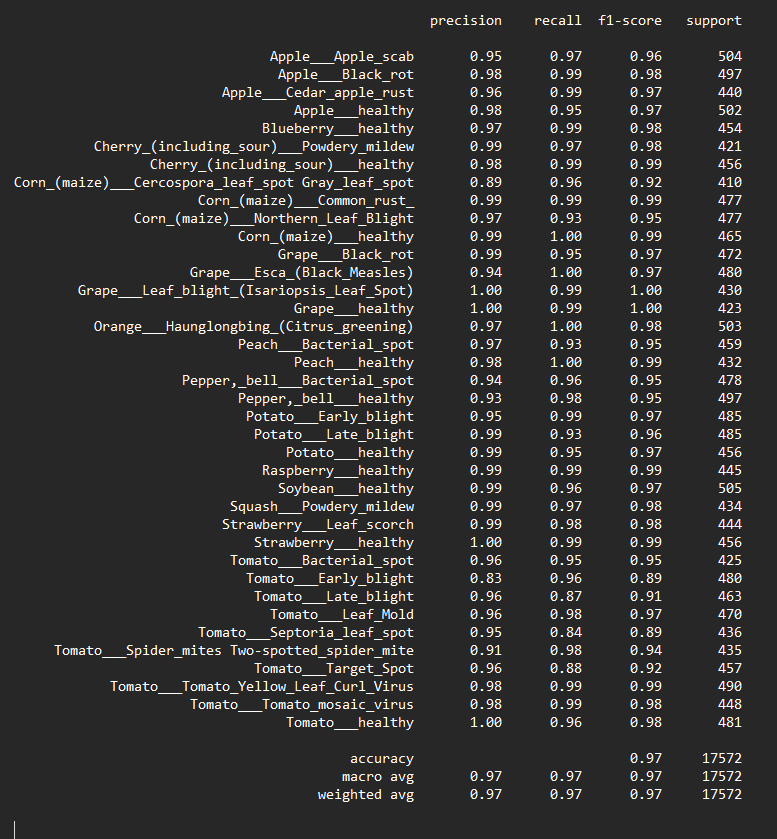
F1-Score Analysis:

The F1-score, the harmonic mean of precision and recall, offers a balanced measure of the model's performance. It considers both precision and recall, providing a single metric to evaluate the model's effectiveness in binary classification tasks. High F1-scores indicate a balanced trade-off between precision and recall, reflecting a well-performing model. Analyzing F1-scores across classes helps identify classes where the model achieves a harmonious balance between precision and recall and classes where there may be a trade-off between the two metrics.

Support Analysis:

Support metrics reveal the distribution of actual occurrences of each class in the dataset, providing context for evaluation metrics. It indicates the number of instances available for each class, influencing the reliability and significance of precision, recall, and F1-score metrics. Balanced support across classes ensures representative evaluation, while imbalanced support may affect the interpretation of evaluation metrics, particularly for minority classes.





**3.1.2 Performance Insights**

High-Performing Classes:

Classes with high precision, recall, and F1-scores demonstrate the model's effectiveness in accurately identifying diseases. Analysis of these classes can reveal successful patterns and strategies employed by the model, such as distinctive visual features or robust training data representation. Understanding the factors contributing to high performance in these classes can inform model development and optimization strategies.

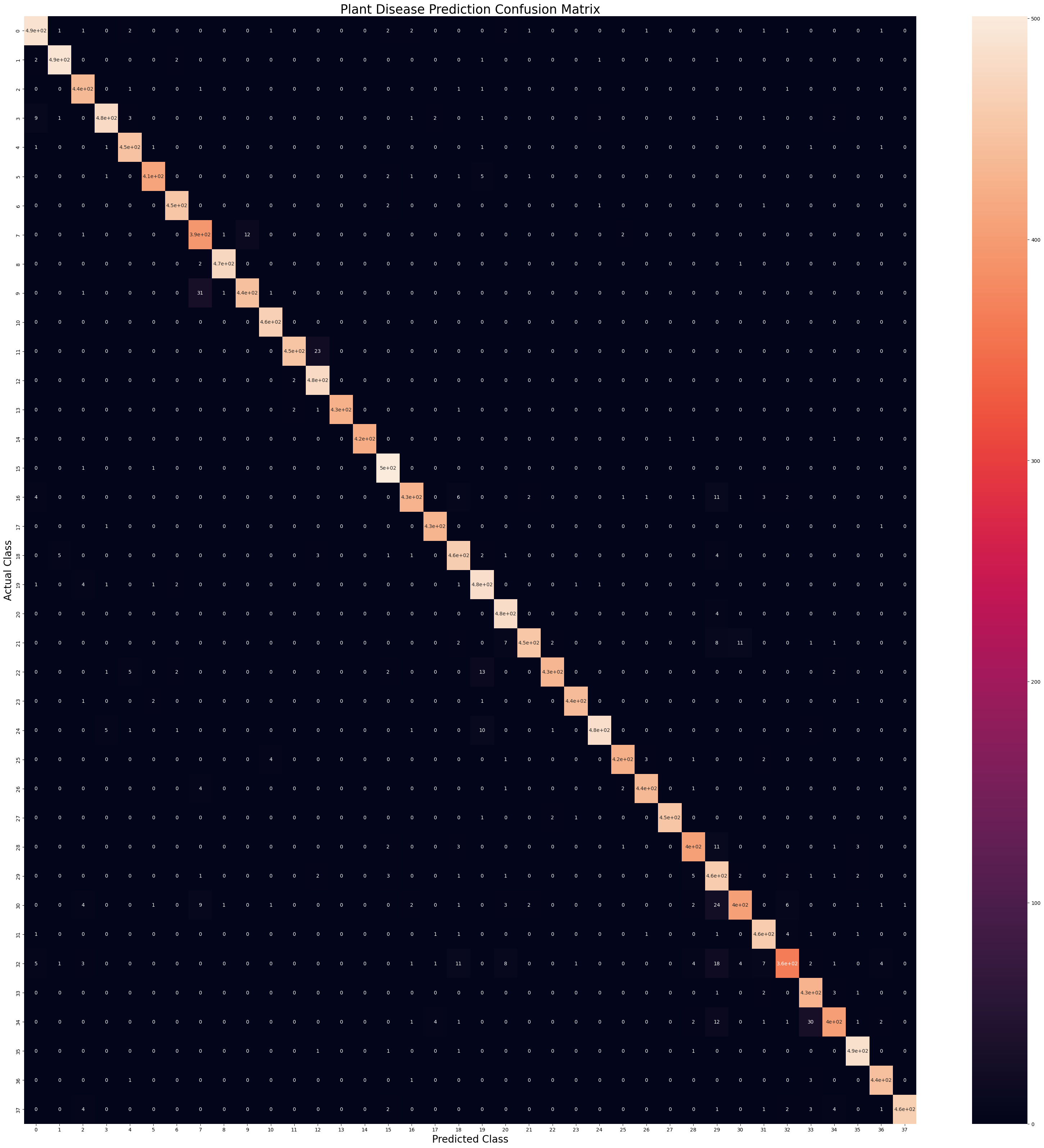
Challenging Classes:

Classes with lower precision, recall, or F1-scores highlight areas where the model may struggle or require improvement. These classes pose challenges due to factors such as visual ambiguity, class imbalance, or inadequate representation in the training data. Analyzing the challenges faced by the model in these classes helps identify specific areas for model refinement, data augmentation, or architectural adjustments to enhance performance.

**3.1.3 Model Interpretation**

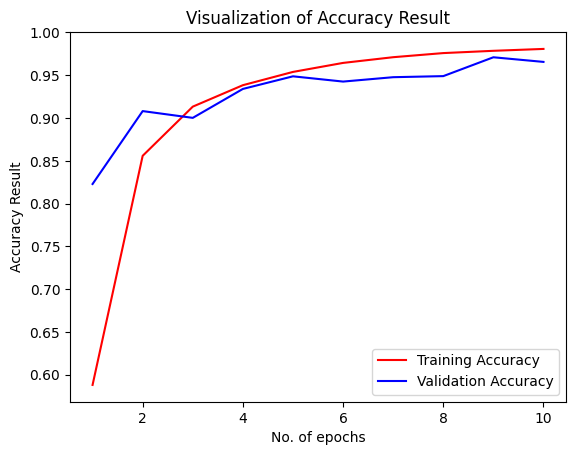
Confusion Matrix Analysis:

The confusion matrix provides a detailed breakdown of the model's predictions and misclassifications across different classes. It visualizes the true positive, false positive, true negative, and false negative predictions, offering insights into the model's classification patterns and areas of confusion. Analyzing the confusion matrix helps identify common sources of misclassification, class overlap, or ambiguity, guiding efforts to improve model accuracy and robustness.



Visualization Techniques:

Visualizing precision-recall curves, confusion matrices, and other performance metrics enhances the interpretation of model results and facilitates comparison across classes. Visualization techniques provide intuitive insights into model performance, enabling stakeholders to identify trends, outliers, or anomalies that may require further investigation or intervention. Interactive visualization tools empower stakeholders to explore model behavior and performance metrics dynamically, fostering deeper understanding and engagement with model outcomes.



**3.2 Future Scope**

**3.2.1 Model Improvement Strategies**

Fine-Tuning Hyperparameters:

Further experimentation with hyperparameters, such as learning rate, batch size, and optimizer choice, can optimize model performance and convergence. Fine-tuning hyperparameters enables iterative refinement of model training strategies, leading to enhanced performance and stability across diverse datasets and tasks.

Data Augmentation:

Increasing the diversity and size of the dataset through data augmentation techniques can enhance model generalization and robustness. Data augmentation introduces variations in training samples, such as rotations, translations, and transformations, enriching the training data and improving the model's ability to capture diverse visual patterns and features.

Architecture Refinement:

Iterative refinement of the CNN architecture, including layer adjustments, feature extraction techniques, and model complexity management, may improve the model's ability to capture complex patterns in plant images. Architectural refinements aim to enhance model expressiveness, adaptability, and efficiency, leading to superior performance and scalability across different datasets and applications.

**3.2.2 Domain-Specific Enhancements**

Transfer Learning:

Leveraging pre-trained models and transfer learning techniques tailored to the domain of plant disease recognition can accelerate model training and improve performance. Transfer learning enables the transfer of knowledge and feature representations learned from large-scale datasets or related tasks, enhancing the model's ability to generalize and adapt to new data distributions and challenges.

Multi-Modal Fusion:

Integrating additional data sources, such as sensor data, spectral imaging, or environmental factors, alongside visual images, can provide complementary information for disease detection and classification. Multi-modal fusion techniques combine information from diverse sources to enhance model robustness, interpretability, and reliability, enabling more comprehensive and accurate plant disease diagnosis in real-world settings.

Continuous Monitoring:

Implementing a system for continuous monitoring and updating of the model based on new data, feedback from users, and evolving challenges can ensure ongoing improvement and adaptation. Continuous monitoring enables model refinement, performance validation, and quality assurance, fostering a dynamic and responsive model ecosystem that meets the evolving needs of stakeholders and end-users.

**3.2.3 Application Expansion**

Crop Diversity:

Extending the scope of the model to cover a broader range of crops, plant species, and agricultural contexts can broaden its applicability and utility. Addressing diverse agricultural challenges requires tailoring the model to specific crops, diseases, and environmental conditions, enabling more targeted

**REFERENCES**

1. Saha, A., et al. (2019). Plant Leaf Disease Detection using Image Processing Techniques: A Review. \*International Journal of Advanced Computer Science and Applications (IJACSA), 10\*(11), 466-474. Retrieved from https://thesai.org/Downloads/Volume10No11/Paper\_63-Plant\_Leaf\_Disease\_Detection.pdf

2. Sharma, P., et al. (2020). A Review on Methods for Detection of Plant Leaf Diseases using Image Processing Techniques. \*International Journal of Computer Applications, 169\*(7), 1-5. Retrieved from https://www.ijcaonline.org/archives/volume169/number7/sharma-2020-ijca-922575.pdf

3. Patel, R., et al. (2020). A Comprehensive Review on Automated Diagnosis of Plant Leaf Diseases using Image Processing Techniques. \*International Journal of Computer Sciences and Engineering, 8\*(10), 41-46. Retrieved from http://www.ijcseonline.org/ncrtccsp-20/v8i10p6.html

4. "Plant Disease Detection & Recognition using Image Processing" by The Artificial Intelligence Hub. (YouTube). Retrieved from [https://www.youtube.com/watch?v=6yXV6XJyGzk](https://www.youtube.com/watch?v=6yXV6XJyGzk)