AgriGuard: Empowering Farmers with Cutting-Edge Plant Disease Detection Technology

Submitted in partial fulfillment of the requirements for the degree of

Bachelor of Technology

in

B. Tech

Computer Science and Engineering

by

Ujjwal Tiwari (20BCE0415)

Utkarsh Singh (20BCI0284)

Tarang Gupta (20BCI0310)

Under the guidance of

Dr. Naresh K

School of Computer Science and Engineering,

VIT, Vellore.

May, 2024

DECLARATION

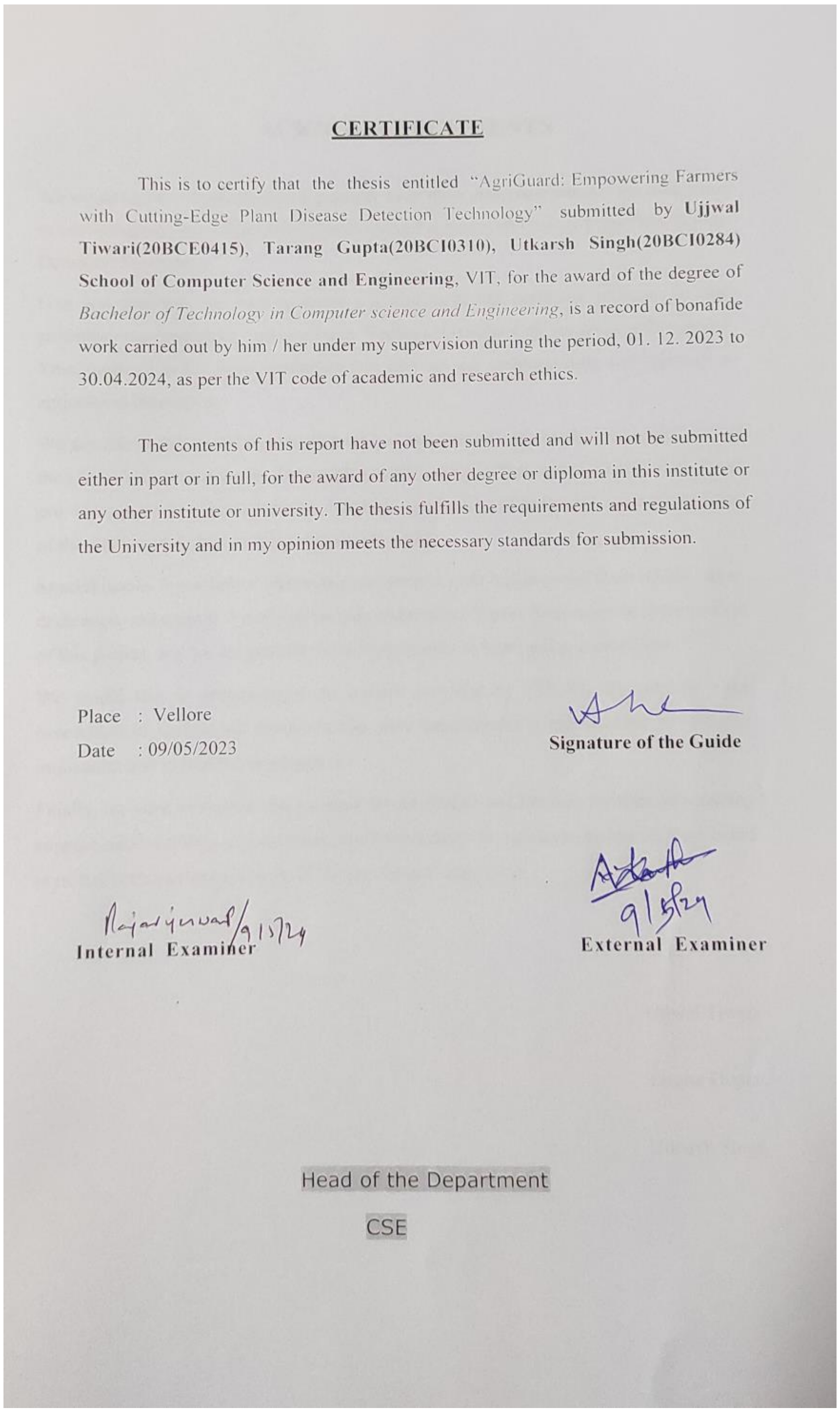
I hereby declare that the thesis entitled "AgriGuard: Empowering Farmers with Cutting-Edge Plant Disease Detection Technology" submitted by me, for the award of the degree of *Bachelor of Technology in Computer science and Engineering* to VIT is a record of bonafide work carried out by me under the supervision of Dr. Naresh K.

I further declare that the work reported in this thesis has not been submitted and will not be submitted, either in part or in full, for the award of any other degree or diploma in this institute or any other institute or university.

Place : Vellore
Date : 09/05/2024

Signature of the Candidate

[illegible]



CERTIFICATE

This is to certify that the thesis entitled "AgriGuard: Empowering Farmers with Cutting-Edge Plant Disease Detection Technology" submitted by **Ujjwal Tiwari(20BCE0415)**, **Tarang Gupta(20BCI0310)**, **Utkarsh Singh(20BCI0284)** **School of Computer Science and Engineering**, VIT, for the award of the degree of *Bachelor of Technology in Computer science and Engineering*, is a record of bonafide work carried out by him / her under my supervision during the period, 01. 12. 2023 to 30.04.2024, as per the VIT code of academic and research ethics.

The contents of this report have not been submitted and will not be submitted either in part or in full, for the award of any other degree or diploma in this institute or any other institute or university. The thesis fulfills the requirements and regulations of the University and in my opinion meets the necessary standards for submission.

Place : Vellore

Date : 09/05/2023

[illegible]

Signature of the Guide

[illegible] 9/5/24

Internal Examiner

[illegible] 9/5/24

External Examiner

Head of the Department

CSE

ACKNOWLEDGEMENTS

We would like to extend our sincere gratitude to all those who contributed to the completion of our college project, "AgriGuard: Empowering Farmers with Cutting-Edge Plant Disease Detection Technology."

First and foremost, we want to thank our professors and academic advisors for their guidance, encouragement, and invaluable feedback throughout the duration of this project. Your mentorship has been instrumental in shaping our understanding and approach to agricultural innovation.

We are also deeply grateful to the farmers and agricultural experts who generously shared their knowledge, experiences, and insights with us. Your willingness to engage with us and provide real-world perspectives has enriched our project and deepened our understanding of the challenges facing the agricultural community.

Special thanks to our fellow classmates and project team members for their collaboration, dedication, and support. Your contributions and teamwork have been essential to the success of this project, and we are grateful for the opportunity to learn and grow together.

We would like to acknowledge the support provided by VIT for this project. Your investment in student-led initiatives like ours demonstrates a commitment to fostering innovation and excellence in education.

Finally, we want to express our gratitude to our friends and families for their unwavering support, understanding, and encouragement throughout this academic endeavor. Your belief in us has been a constant source of motivation and inspiration.

Student Name

Ujjwal Tiwari
Tarang Gupta
Utkarsh Singh

Executive Summary

The internet and telecommunications technology have transformed the agricultural landscape, providing farmers with a range of resources to improve their operations. One of the significant challenges in agriculture is plant diseases, which can significantly reduce crop yields and economic returns.

In our study, we developed a highly effective plant disease detection model using the Plant Village dataset, comprising 54,000 images of 14 crop species. Leveraging a finely tuned ResNet50 architecture, our model surpassed other popular models like Normal CNN, Inception V3, and VGGnet in accurately identifying crop illnesses across various forms. With the proliferation of internet and telecommunications technology, such advancements are crucial in agriculture, where diseases can significantly diminish yields and economic returns. Our model offers farmers a tool for early detection, enabling timely interventions and minimizing losses. This underscores the pivotal role of plant disease detection models in enhancing agricultural productivity and economic outcomes.

Our study showcases the efficacy of a finely tuned ResNet50 model in detecting plant diseases using the extensive Plant Village dataset. Outperforming other models, it empowers farmers to identify crop illnesses early, enabling timely interventions and ultimately enhancing agricultural productivity and economic returns.

List of Figures

List of Tables

1. INTRODUCTION

1.1. OBJECTIVE

Agriculture has long been the backbone of human civilization, providing sustenance and nourishment to billions of people around the world. In fact, over 90% of the global population is in some way associated with farming, making it a crucial industry for our survival. However, the very foundation of our food security is now under threat from a range of factors, such as climate change, the decline of pollinators, and the emergence of plant diseases.

The goal is to create a deep learning model that uses native datasets to identify illnesses in apple plants. The main objective is to lessen the negative effects of pests and diseases on agricultural products, with a special emphasis on the widespread problem of scab disease, which is harming Kashmir's apple production.
Early diagnosis and intervention will be possible because to the model's precise identification of disease indicators in apple plants using picture inputs. By enabling prompt diagnosis, farmers may take proactive steps to stop the spread of illnesses, which will eventually reduce crop losses and lessen the hardship experienced by fruit growers.

1.2. MOTIVATION

AgriGuard is a ground-breaking technology that has the potential to completely change the agriculture industry. It provides farmers with a proactive approach to plant disease identification by utilising state-of-the-art technologies. Our technology quickly detects possible agricultural risks using artificial intelligence, data analytics, and sophisticated sensors, enabling farmers to take preventative measures before irreversible harm is done.

AgriGuard was founded with the straightforward but significant goal of preserving farmers' livelihoods and guaranteeing food security for next generations. It gives farmers real-time information about plant health so they can use resources optimally, make well-informed choices, and reduce losses from disease outbreaks.

It also represents a dedication to environmental management and sustainability. Our technology maximises yields and profitability while promoting environmentally responsible agricultural practices by minimising crop losses and eliminating the need for chemical treatments.

1.3. BACKGROUND

AgriGuard was founded out of a deep-seated concern for the difficulties that farmers encounter all across the world. Plant diseases may be detected using traditional methods, but these approaches are often insufficient, leaving farmers susceptible to unplanned outbreaks that might wipe out whole crops and livelihoods. A varied group of agricultural professionals, engineers, and data scientists got together with the common goal of revolutionising the way farmers approach disease control after seeing the urgent need for a better solution.

Moreover, a strong feeling of duty to the farming community and the environment drove the creation of AgriGuard in addition to technological innovation. Understanding how important agriculture is to the world's food security and sustainability, the team made it a top priority to develop a solution that will support ecological balance, resource efficiency, and productivity gains in addition to increased output.

2. PROJECT DESCRIPTION AND GOALS

2.1. SURVEY ON EXISTING SYSTEM

Recognition of plant diseases has been a topic of discussion for many years. With the use of machine learning techniques, several researchers have created numerous appropriate architectures and contributed their ideas to the detection of plant diseases.

Yatoo, A. A., & Sharma, A. (2021) [5]. In this paper, the performance of a Fine-tuned CNN-based model using the optimizer Stochastic Gradient Descent (SGD) and VGG16 was compared to identify the diseases, scab and cedar apple rust, the models were trained using publicly available The Plant Pathology Data Set 2020 was used which comprises a total of 3,651 high-quality, real-life symptom leaf photos. The results of the experiments show how the recommended model works better than VGG16 with 85% accuracy. The authors of [6] presents a study on image classification using a 5-layer deep convolutional neural network (CNN) for recognizing four types of plant diseases. The dataset used in the study included images of anthracnose, downy mildew, powdery mildew, and target leaf spots collected from online sources. The study evaluated the performance of the deep CNN using accuracy measurements and compared it with the conventional classifiers and AlexNet. The results showed an accuracy of 93.4% on the unbalanced dataset and 92.2% on the balanced dataset.

According to the study [7], an effective deep learning model with three parts—a backbone network, a bi-FPN network, and a class/box prediction network—can be used to identify plant leaf diseases. Images of four distinct plant species with five distinct illnesses are included in the dataset used to train and test the CNN. Using a number of criteria, including accuracy, precision, recall, and F1 score, the proposed system is assessed and contrasted with the current plant disease detection methods. Using pre-trained convolution neural network (CNN) models for the extraction and support vector machine (SVM) for classification, Agarwal, P., & Reddy, M. J. (2019) [8] underlined the significance of early identification of plant diseases for averting large crop losses. Using a collection of plant photos, including those with healthy plants and those with various illnesses, such as bacterial blight and powdery mildew, they trained and tested their system, achieving 96% accuracy.

Syed-Ab-Rahman, S.F., Hesamian, M.H. & Prasad, M. (2022) [9], This paper describes a deep convolutional neural network based on Faster R-CNN, which has four components, used to classify citrus leaves into four categories: black spot, Huanglongbing, canker, and healthy. The network was trained on a Kaggle dataset consisting of 477 training samples and 127 test samples, with images in PNG format and size of 256x256. The results showed high performance accuracy, with 86.2% for black spot, 97.2% for canker, and 94.6% for Huanglongbing. M. Kathiravan, K. H. Priya, S. Sreesubha, A. Irumporai, V. S. Kumar, and V. V. Reddy [10], This paper uses OpenCV to preprocess images and a convolutional neural network in TensorFlow for deep learning to classify different diseases in a dataset of paddy, guava, and cotton called Kaggle Fruits 360. The dataset used for experimentation is 90k, and the proposed model achieves an accuracy of 93.3%, which is better than other ML methods such as SVM, KNN, and ANN.

In [11] The paper proposed Two-channel Convolutional Neural Network was built using elements from VGG and ResNet models to identify instances of maize leaf disease. Three datasets representing different types of leaf diseases (big spot, gray leaf spot, and rust) were obtained from the "kaggle" website. The network's validation accuracy ultimately reached 98.33%, and the loss incurred during testing was 0.07628. Mondal, Joyanta & Islam, Md & Zabeen, Sarah & Islam, A. B. M. Alim Al & Noor, Jannatun. (2022) [12] The paper presents a new Deep CNN architecture that is lightweight and does not use transfer learning. The method is tested on the Plant Village dataset which contains 54,303 leaf images divided into 38 categories based on species and disease. The proposed method achieves an accuracy of 98.18%, which is higher than VGG-16 However, it should be noted that this method is limited to leaf disease detection and a smaller number of categories.

Ullah, F., Khan, R. U., Khan, K., Albattah, W., & Qamar, A. M. (2021). [13] The paper proposed a CNN model that combines computer vision (CV) to categorize various layers of damaged leaves. The study employed the Plantvillage dataset, which included 54303 images of unhealthy and healthy leaves, classified into 38 categories by disease and species. The proposed approach achieved an average final accuracy of 0.938. However, the study identified some limitations such as database availability and size, feature extraction methods, classification module difficulties, and the constraints of existing systems. Jiang, F., Lu, Y., Chen, Y., Cai, D., & Li, G. (2020) [14]. The paper present how to use deep learning and support vector machine (SVM) techniques to precisely detect four "common rice leaf diseases" (rice blast, bacterial leaf blight, brown spot, and leaf streak). Four different diseases—rice blast, rice bacterial spot, rice streak leaf spot, and rice sheath blight—were present in a collection of 8911 images of rice leaves. These images were split into training and testing sets, and a deep learning approach based on CNN was used to extract features from the images before SVM was used to classify the images into various disease categories. The findings indicated that the suggested approach outperformed numerous other methods that were examined in the study, achieving an accuracy of 96.83% on the testing set.

2.2. RESEARCH GAP

Table 1. Comparison

RefNo.	Methodology	Datasets	Performance Measures	Limitation
[5]	Fine-tuned CNN, Stochastic Gradient Descent (SGD), VGG16.	The Plant Pathology Dataset 2020	Recall- 0.85, Accuracy- 0.85, F1-Score- 0.85, Average Score 735, Accuracy = (85%)	A software application is needed, The model could not generalize to other forms of plant diseases.
[6]	5 Layers CNN, DCNNs, AlexNet and conventional classifiers.	Plant Village Dataset, and Forestry Images Dataset	Precision- 92.18%, Accuracy- 93.4% and 92.2%	Limited Dataset, Not able to generalize to different types leaf symptoms or diseases.
[7]	Efficient De deep learning model, Backbone Network, BiFPN Network, and Class/ Box Prediction Network	PlantVillage Dataset	Precision- 88.13%, Recall- 94.06% and F1-score is 90.99%, Accuracy- 74.10%	The crop species which are trained are less.
[8]	YOLOv3, Simple Classifier, VGGNet, Transfer Learning with ResNet.	PlantVillage Dataset	Accuracy- 96%	It does not provide specific details about the type of model used for leaf classification or the dataset it was trained on.
[9]	Faster R-CNN. (RPN), (ROI) pooling and classifier.	Citrus leave prepared Dataset	F1-score -(90, 97.9, 93.0), Precision- (93.8, 99.0, 91.0), Recall- (87, 95.8, 94.6), Accuracy- (86.18, 97.2, 94.64) for blackspot, Canker, Huanglogbing, respectively.	Improvement on robustness , reduce the standard deviation.
[10]	convolutional neural network using TensorFlow for deep learning.	The Kaggle Fruits 360 Dataset	Accuracy: 93.3% The SVM scored 89.1%.	Not many diseases are detected in this system.
[11]	Two-channel CNN: VGG and ResNet model	Leaf disease Dataset	VGG validation accuracy: 93.33%, loss: 0.2385, ResNet validation accuracy: 97.75%, loss: 0.1276, Final validation accuracy: 98.33%, loss: 0.07628	Good RGB pixel camera is needed, updating in leaf disease over a period need to be done.
[12]	Deep CNN.	Plant Village Dataset	Accuracy: 98.18% Training Accuracy: 0.9836 Validation Accuracy: 0.9806	Limited to leaves dataset only, fewer categories of just 38 by species.

			Precision: 0.9831 Recall: 0.9816 F1-Score: 0.9816	
[13]	CNN model with the integration of computer vision (CV).	PlantVillage Dataset	Average final accuracy - 93.8% Precision - 0.991, Recall - 0.959, F-measure- 0.975.	Available Databases and Size Issue, Issues with Available Feature Extraction Methods, Difficulties in Classification Module, limitations of available systems.
[14]	CNN and SVM model.	Rice leaf disease Dataset	Average accuracy correct recognition rate 96.8%	Small Sample size, Lack of diversity in images, only rely on color features which lead to less accurate

2.3. PROBLEM STATEMENT

According to recent estimations, agricultural yields across numesystemrous crops are expected to fall significantly due to bad weather conditions and the proliferation of pests and diseases, which are causing concern among farmers throughout the world. Without crop insurance and government aid, notably in the form of minimum price guarantees, farmers, particularly those in poor communities, face significant losses.

To address these challenges and increase global agricultural resilience, a deep learning model based on local datasets is proposed. The approach aims to identify and mitigate the impact of pests and diseases on agricultural crops by detecting illness in plants, regardless of crop type. The programme uses photo inputs to identify illnesses and intends to provide early and accurate diagnosis, offering farmers actionable knowledge.
Implement targeted interventions and management strategies to maintain crop health and increase overall agricultural yield.

Output: Detection of agricultural diseases across various crops, facilitating proactive measures for disease control and crop health management.

3. TECHNICAL SPECIFICATION

3.1 System Requirements

Hardware Requirements

- Minimum multi-core processor (e.g., Intel Core i5 or equivalent)
- Recommended: Multi-core processor with higher clock speed
- Minimum 8GB, Recommended 16GB or more

Software Requirements

- Image processing libraries (e.g., Keras,OpenCV) for preprocessing uploaded images.
- Dependency management tools like pip for Python packages.
- TensorFlow for implementing the deep learning model.

3.1.1 Functional Requirements

- **Image Upload Functionality:**
  Users should be able to upload images of diseased plants.
- **Disease Detection Algorithm:**
  Implement a machine learning algorithm to detect diseases in plants based on uploaded images.
- **Output Presentation:**
  Display the results of disease detection to users in a clear and understandable format.
  Provide information on the identified disease, its symptoms, and recommended actions for mitigation.
- **Data Management:**
  Maintain a database of labeled images for training and validation of the disease detection model.
  Ensure data privacy and security measures are in place to protect user data and uploaded images.

3.1.2 Non-Functional Requirements

- **Accuracy:**
  The disease detection algorithm should achieve high accuracy in identifying various forms of crop illnesses to minimize misdiagnosis.
- **Reliability:**
  The system should be highly reliable, ensuring minimal downtime and interruptions in service.
- **Usability:**
  The user interface should be intuitive and user-friendly, catering to users with varying levels of technical expertise.
- **Scalability:**
  Design the system to be scalable, capable of accommodating growth in user base and increasing data volume over time.

3.2 Feasibility Study

3.2.1 Technical Feasibility

- Evaluate the availability and suitability of technologies like machine learning frameworks (e.g., TensorFlow), web development frameworks (e.g., Django, Flask), and image processing libraries.
- Assess the feasibility of processing and analyzing large volumes of image data for disease detection using available computational resources.
- Determine the feasibility of integrating the disease detection model with web and mobile platforms for seamless access by users.
- Evaluate whether the system architecture can scale to accommodate an increasing number of users and data volume over time.

3.2.2 Economical Feasibility

- Estimate the expenses associated with software development, including hiring developers, acquiring licenses for software tools, and infrastructure costs.
- Assess ongoing expenses such as server maintenance, database hosting, and cloud storage fees.
- Explore potential revenue streams, such as subscription fees from users, partnerships with agricultural organizations, or government grants.
- Analyze the demand for the AgriEasy platform among farmers and agricultural experts, considering factors like market size, competition, and willingness to pay for disease detection services.

3.2.3 Social Feasibility

- Evaluate farmers' willingness to adopt the AgriEasy platform and integrate it into their apple farming practices.
- Assess the potential benefits of early disease detection in agriculture, such as increased crop yields, reduced pesticide use, and improved food security.
- Ensure the platform is accessible to farmers in rural areas with limited internet connectivity or technical expertise.
- Provide resources and training to help users understand how to use the platform effectively for disease detection and management.

3.3 System Specifications

3.3.1 Hardware Specifications

- Ascertain the system's hardware needs, including the amount of processing power needed for data processing and model inference.

- Take into account the performance and scalability needs to handle future rises in user traffic and data volume.
- Choose appropriate cloud computing services or hardware components according to the requirements of the system and your financial limits.

3.3.2 Software Specifications

- Determine which software tools and frameworks—such as database management systems, machine learning libraries, and programming languages like Python and TensorFlow—are needed to construct the system.
- Define the development environment and version control procedures to guarantee the system code's repeatability and maintainability.
- Describe the system's modular components, such as user interface modules, machine learning models, and data processing pipelines, and the software architecture.

4. DESIGN APPROACH AND DETAILS

4.1. System Architecture

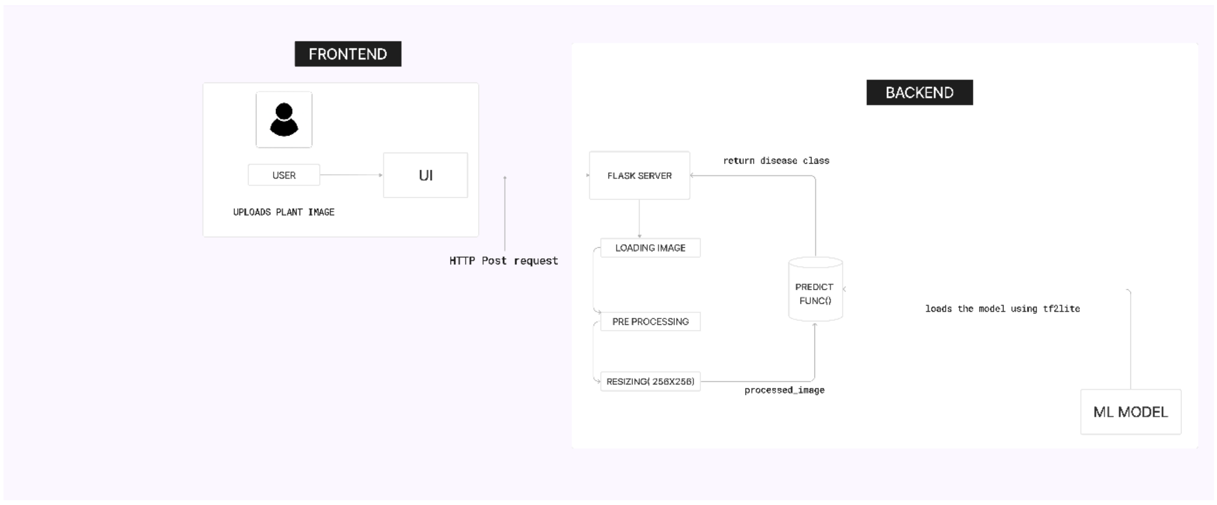


Fig 4.1.1 – Architecture

4.2. Design

4.2.1. Data Flow Diagram

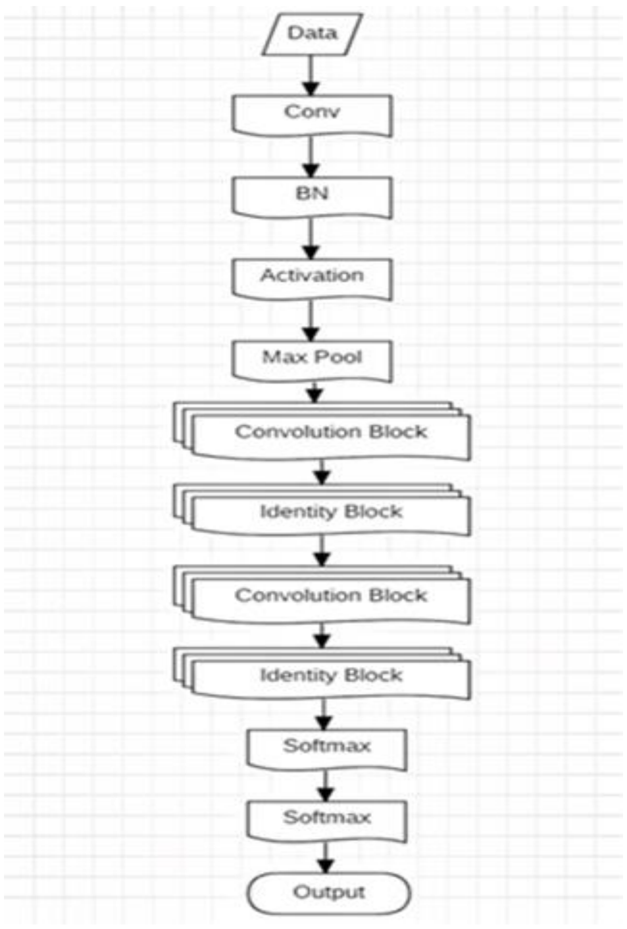


Fig 4.2.1 – Data flow

4.2.2. Use Case Diagram

Description: The system displays the results of disease detection to the farmer, providing information on the identified disease and recommended actions for mitigation.

This Use Case Diagram illustrates the primary interactions between the actors (farmers) and the AgriEasy system, highlighting the key functionalities of the platform for disease detection in agriculture.

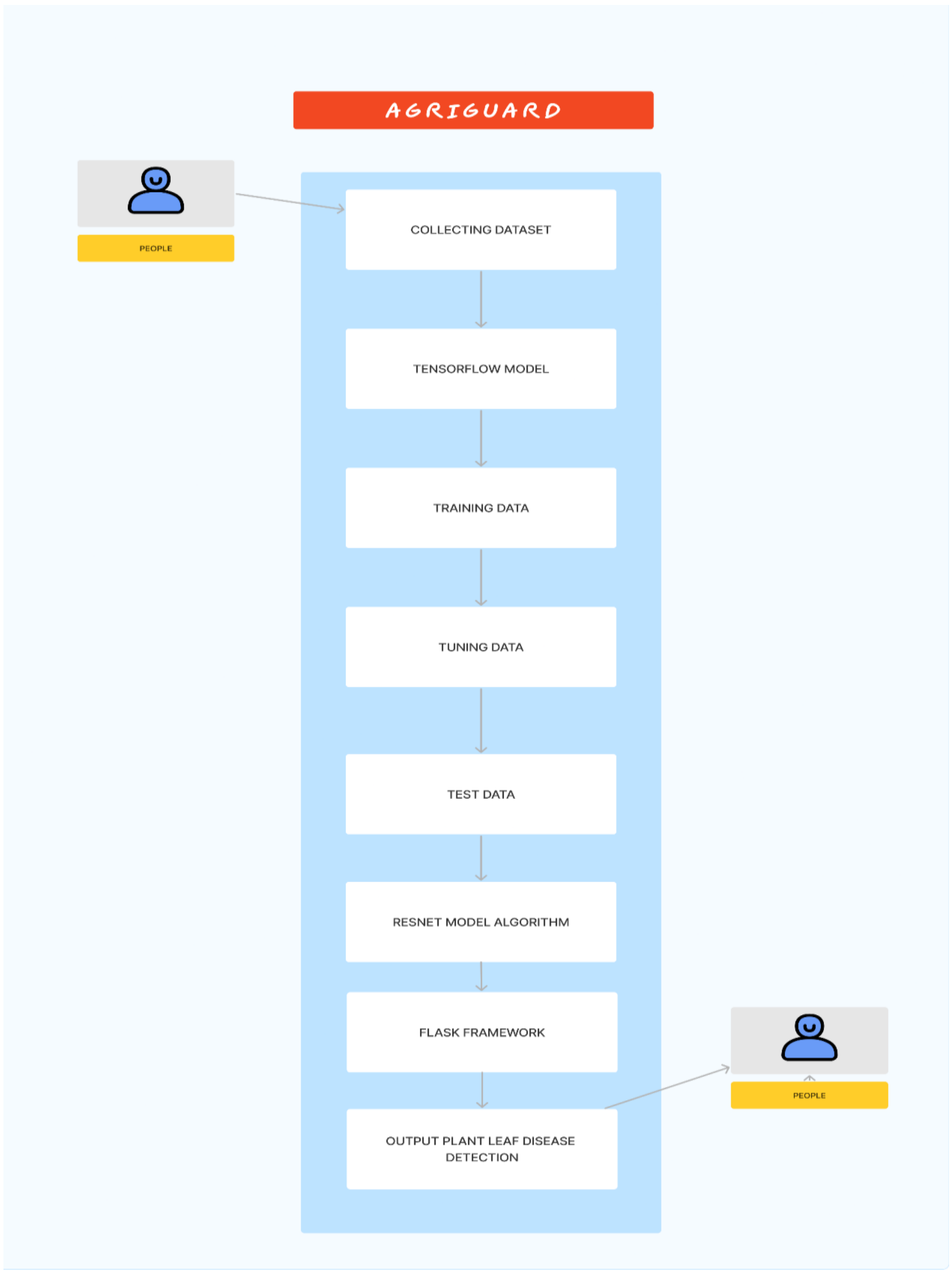


Fig 4.2.2 – Use Case

4.2.3. Class Diagram

A class diagram is essentially a graphical depiction of the system's static view that shows various application features. Therefore, the entire system is represented by a set of class diagrams. The class diagram's name ought to be significant way to characterize the system's components.

Every component and how it relates to the others should be determined beforehand. The minimal amount of properties for each class should be defined, as extra properties would confuse the diagram. Each class's responsibility (attributes and methods) should be clearly identified.

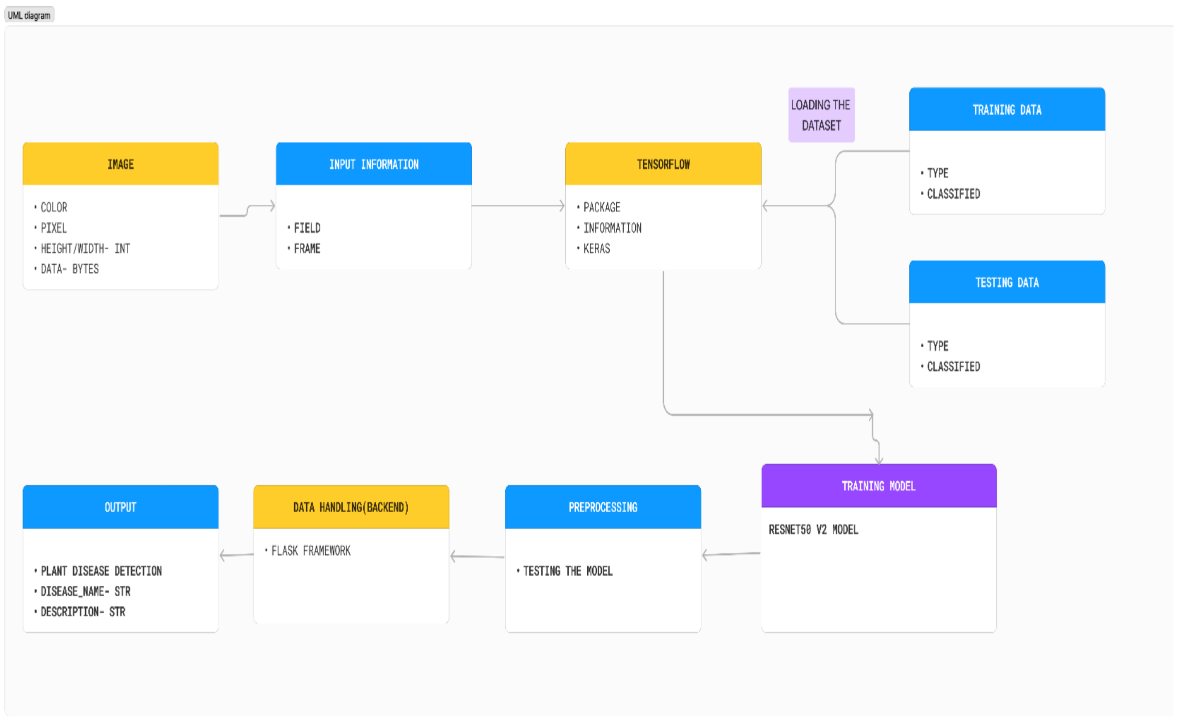


Fig 4.2.3 – Class Diagram

4.2.4. Sequence Diagram

Sequence diagrams are often used for both analysis and design reasons. They provide a visual representation of the logic flow within your system, allowing you to evaluate and record your logic. The most widely used UML artifact for dynamic modeling, which focuses on determining the behavior inside your system, is a sequence diagram.

Activity diagramming, communication diagramming, time diagramming, and interaction overview diagramming are further dynamic modeling methodologies. In the context of developing modern business applications, sequence diagrams are among the most crucial design-level models, along with class diagrams and physical data models.

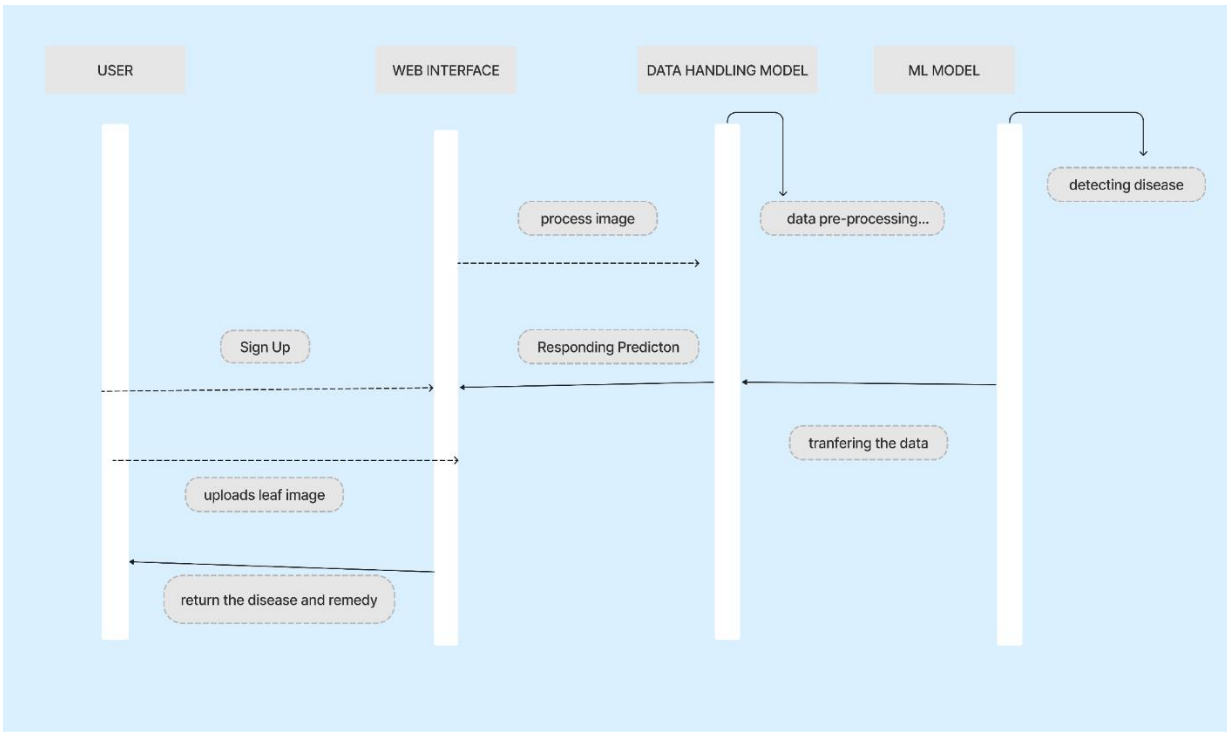


Fig 4.2.4 – Sequence Diagram

4.3. Constraints, Alternatives and Tradeoffs

Constraint:

- Dataset Quality: The plant disease detection model relies substantially on a diverse and high-quality dataset. Constraints may develop if the dataset is insufficiently representative of diverse crop species and disease kinds, resulting in limited model generalisation.
- Deep learning models like ResNet-50 require significant computational resources, including high-performance GPUs and enough storage. Limited access to these resources may impede model development and optimisation.
- Model Interpretability: Although deep learning models like ResNet-50 are highly accurate in disease identification, their complicated topologies might make it difficult to understand the decision-making process. This lack of interpretability may complicate conveying model predictions to end users, such as farmers and agricultural professionals.

Alternatives:

- 1.Transfer Learning: Rather than training the model from scratch, transfer learning can be used to fine-tune pre-trained models using the Plant Village dataset. This method may speed up model training and use fewer data samples, hence addressing constraints associated to dataset size and processing resources. Ensemble Learning .
- Techniques: Model averaging or boosting can be used to aggregate predictions from various base models, such as ResNet-50, Normal CNN, Inception V3, and VGGnet. Ensemble approaches frequently produce more robust and accurate forecasts by combining the strengths of various models.
- 3.Hybrid Approaches: Combining machine learning approaches and classical image processing algorithms can provide complementing benefits in illness identification. Hybrid techniques may improve model interpretability and make it easier to extract domain-specific information related to crop diseases.

Tradeoffs:

- Complexity vs. Interpretability: ResNet-50, a deep learning model, has great accuracy but limited interpretability due to its inherent complexity. It is critical to strike a balance between model complexity and interpretability, as overly complex models might impede end-user comprehension and confidence.
- Computational Efficiency vs. Model Performance: While complicated deep learning architectures can improve model performance, they frequently need large computational resources and lengthy training cycles. Tradeoffs between computational efficiency and model performance must be evaluated for practical deployment in resource-constrained contexts.
- Overfitting vs. Generalisation: Achieving robust performance requires a model that can generalise to new data without overfitting. Regularisation approaches, data augmentation, and cross-validation tactics can help achieve a balance between model generalisation and training data fit.

5. PROPOSED METHODOLOGY

5.1. Overview

In this project, the concept of transfer learning is used to achieve classification. By adjusting a model that has already been trained for a problem of a similar nature, transfer learning allows us to utilize it. The main benefit of transfer learning in deep learning is that it facilitates the reuse of previously acquired information, which reduces the amount of effort needed and prevents having to start from scratch. Pre-trained models are used together with the transfer learning principle for picture classification. Our proposed methodology for plant disease detection using ResNet50 as a base architecture involves using transfer learning to leverage the features learned by the model on the ImageNet dataset. After Pre-Processing and augmenting the Plant Village dataset, we will fine-tune ResNet50 to classify healthy and diseased plants. We will then evaluate the performance of our model and compare it with InceptionNet, and VGG16. This will allow us to determine which model is most effective for plant disease detection and provide insights into the potential applications of transfer learning for image classification tasks.

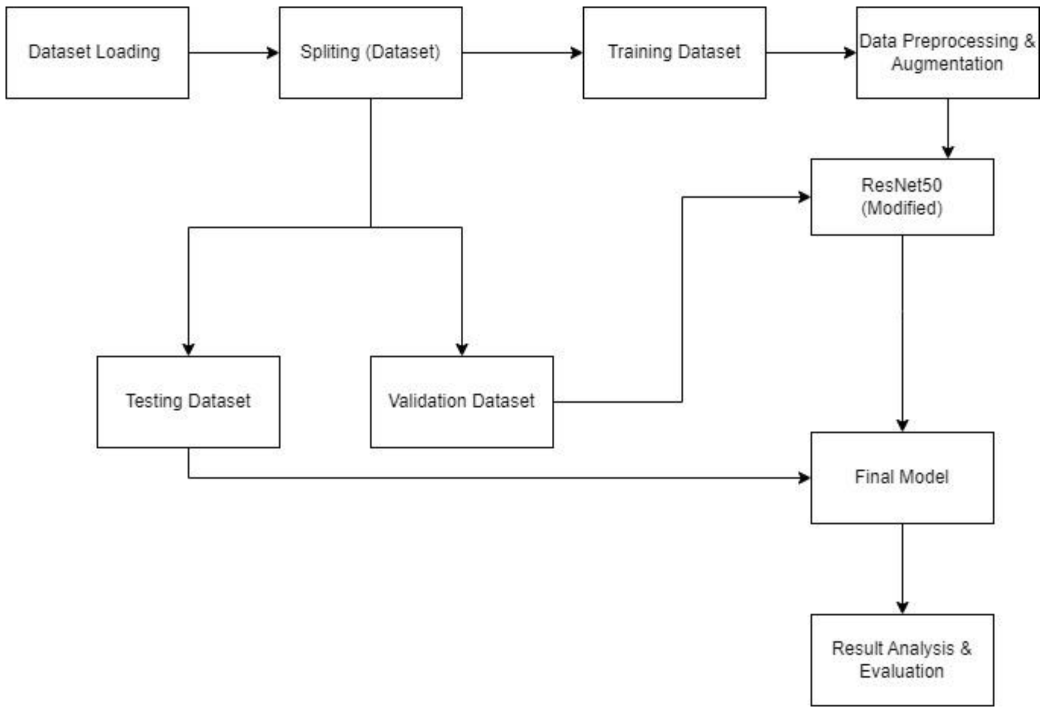


Fig 5.1 – Block Diagram

5.2. Data Preparation

5.2.1. Dataset

In order to classify and identify disease on images that the model has never encountered before, deep learning models were tested and trained on images of plant leaves. We are using the "PlantVillage dataset,"[15] which is a publicly available dataset containing images of diseased and healthy plant leaves. The dataset includes images of 38 different plant species, each with multiple categories of disease, as well as images of healthy leaves. The dataset was created to help with the development of machine learning models for plant disease detection and has been widely used in research.

TABLE II. Dataset details

Attribute	**Value**
Number of classes	38 (14 plants)
Total number of images	54,306
Image size	256x256 pixels
Image format	JPG
Source	Publicly available

5.2.2. Data Splitting

The data is initially divided into three parts: training, validation, and testing, with a percentage ratio of 80%, 10%, and 10%, respectively. This split ratio was selected based on a study [16].

The training set comprises 80% of the data, which is 34,727 samples. The validation set is

10% of the data, which is 8,702 samples, and the testing set is also 10% of the data, which is 10,876 samples. The testing set is used for model evaluation and prediction, while the training set is further split into training and validation sets with ratios of 80% and 20%, respectively, to assess whether the model is overfitting.

The Plant Village training dataset consists of 38,400 RGB images of 14 crop species, with 2,400 images per class. Each image is labelled as "healthy" or "diseased" and belongs to a specific crop species. The images have varying resolutions and aspect ratios, ranging from 256 x 256 to 2,560 x 1,920, and captured in a controlled environment. The dataset also includes metadata files containing information about the crop species, type of disease, and severity of the disease, which can help in developing more complex models.

The validation set of the Plant Village dataset contains 16,906 labelled images of 14 crop species, with 1,066 images per class. It is a subset of the training set, with some images from the training set also present in the validation set. The validation set is used to evaluate the model's performance during training and to test its ability to generalize to new, unseen data, which helps to prevent overfitting.

5.2.3. Data Pre-Processing

Image Pre-Processing enhances the picture data required for image categorization. There are geometric alterations of photographs used in Pre-Processing techniques, such image translation, picture scaling, and rotation. In the Pre-Processing processes, we have scaled down all the photos' resolutions to 256*256 pixels. It must guarantee that all photos are of the same resolution. Then data rescaling can normalize the pixel values of images from the range **[0, 255]** to **[0, 1]**, making it easier for the model to learn and reducing the impact of outliers.

5.2.4. Data Augmentation

In data augmentation by applying random transformations to images, we can create new images that are similar to the originals, but not identical. For example, we can flip an image horizontally to create a mirror image or rotate an image to simulate a different viewpoint. By doing this, we can increase the diversity of the training data and help the model to learn more generalizable features.

We first define a set of data augmentation parameters and specify a range of random transformations to apply to the input images, including rotations, translations, shears, zooms, and flips. We also specify the fill mode to use when creating new pixels to fill in any empty space that may be created by the transformations.

5.3. ResNet50 (Base Architecture)

In transfer learning, models such as AlexNet, AlexNetOWTBn, GooLeNet, Overfeat, and VGG are commonly used due to their stacked convolutional layers. However, deep CNN networks present several challenges, including difficulties in network optimization, the vanishing gradient problem, and degradation problems. To address these challenges, the Residual network (ResNet) introduced a new idea that has proven effective in solving complex tasks and increasing detection accuracy. ResNet aims to overcome the training difficulties encountered in deep CNN networks, such as saturation and accuracy degradation.

In this paper, we utilized the ResNet50 architecture, which consists of 50 layers of residual networks, to address these challenges and improve performance. ResNet-50 is a powerful pre-trained convolutional neural network architecture that has already learned to recognize a wide range of features in natural images. By leveraging the knowledge that ResNet-50 has learned from the ImageNet dataset, we can use transfer learning to develop a plant disease detection model that performs well on a new dataset of plant images with diseases.

To accomplish this, we first remove the fully connected layer(s) from the pre-trained ResNet-50 model and add our own fully connected layer(s) on top of the convolutional layers. We then freeze the weights of the pre-trained ResNet-50 model to ensure that we do not lose the knowledge it has already learned. We train the new fully connected layer(s) on our own plant disease dataset while keeping the convolutional layers of the ResNet-50 model fixed. Finally, we fine-tune the weights of the ResNet-50 model by unfreezing some of its layers and continuing to train the entire network on our own dataset. This transfer learning approach can save a lot of time and effort while providing excellent results for plant disease detection.

5.4. Fine Tuning

We are using fine-tuning and freezing layers to leverage the features learned by the model on the ImageNet dataset for plant disease classification. Fine-tuning involves adapting a pre-trained model that has already been trained for a problem of a similar nature, which allows us to reuse previously acquired information and reduce the amount of effort needed. In addition, by freezing the layers before the 154th layer of ResNet50, we can leverage the pre-trained weights of these layers, which have learned generic features that are likely to be useful for the new task.

We have set specific parameters for fine-tuning, including a batch size of 32, 20 epochs, 1 validation step, an Adam optimizer with a learning rate of 0.00001. These values may need to be adjusted based on the performance of the model during training. By fine-tuning the remaining trainable layers in ResNet50, we can classify healthy and diseased plants and evaluate the performance of the model to determine the most effective model for plant disease detection.

This approach provides insights into the potential applications of transfer learning and freezing layers for image classification tasks, where we can leverage pre-trained models and fix the weights of some layers to improve performance and reduce overfitting. Overall, by combining fine-tuning and freezing layers, we are striking a balance between leveraging the pre-trained weights of the model and allowing it to adapt to the new task, which is a common strategy in transfer learning.

TABLE III. Fine Tuning Parameters

Parameter	**Value**
Batch size	32
Epoch	20
Validation steps	1
Optimizer	Adam (adaptive moment estimation)
Learning rate	0.00001
Loss function	Sparse categorical entropy

5.5. Training:

During the training phase, the model is trained on batches of augmented images, and the parameters of the model are iteratively adjusted to minimize a loss function. At each epoch, the training data is divided into batches, and the model is fed a batch of images. For each image, the model predicts the probability of each class (healthy or diseased), and these predictions are compared to the actual class label of the image. The difference between the predicted and actual labels is quantified using a loss function, such as sparse categorical cross-entropy.

The backpropagation algorithm is then used to calculate the gradients of the loss function with respect to each of the model's parameters. These gradients are then used to adjust the parameters in the opposite direction of the gradients, which updates the loss function. This process is repeated over all epochs, and the model is validated on a separate set of images called the validation set. The goal of training is to find the set of model parameters that minimize the loss function on the training data. However, it's important to note that overly complex models with too many parameters can lead to overfitting. To prevent overfitting, early stopping and regularization techniques are used.

During training, the batch size, the number of steps per epoch, and the number of epochs are all important hyperparameters that need to be set appropriately. The Early Stopping and Model Checkpoint call backs are two techniques used to optimize the training process by adjusting the number of epochs and saving the best model based on the validation loss. Once the training is complete, the trained model can be used to classify new images into their respective classes. The performance of the model can be evaluated using various metrics such as accuracy, precision, and recall which measure the model's ability to correctly classify images. The best model is usually chosen based on the performance on the testing set, which is a set of images not seen by the model during training or validation. In the testing stage, the test dataset which was earlier generated by splitting is used with the final CNN model to find out the performance of our model using various performance metrics such as accuracy, precision, recall, F1 score, and confusion matrix.

6. RESNET MODEL ARCHITECTURE

Kaiming He et al. [17] presented the residual network (ResNet) in 2015 as a deep convolutional neural network architecture that was created to address the issue of vanishing gradients that occurs in very deep neural networks. One of the most often used deep learning architectures is ResNet, which comes in 50-, 101-, and 152-layer variations. The skip connections are made feasible by residual blocks, which have two or three convolutional layers and a skip connection that multiplies the input by the block's output. This architecture allows the network to learn residual functions rather than the desired input-output mapping.

ResNet helps to tackle challenging problems and improves detection precision. ResNet aims to address the saturation and accuracy degradation that occur during the deep CNN training process. We used the ResNet50 with our output layers , architecture in this paper. 50 layers of residual networks made up ResNet50. ResNet50 architecture is depicted in Figure below.

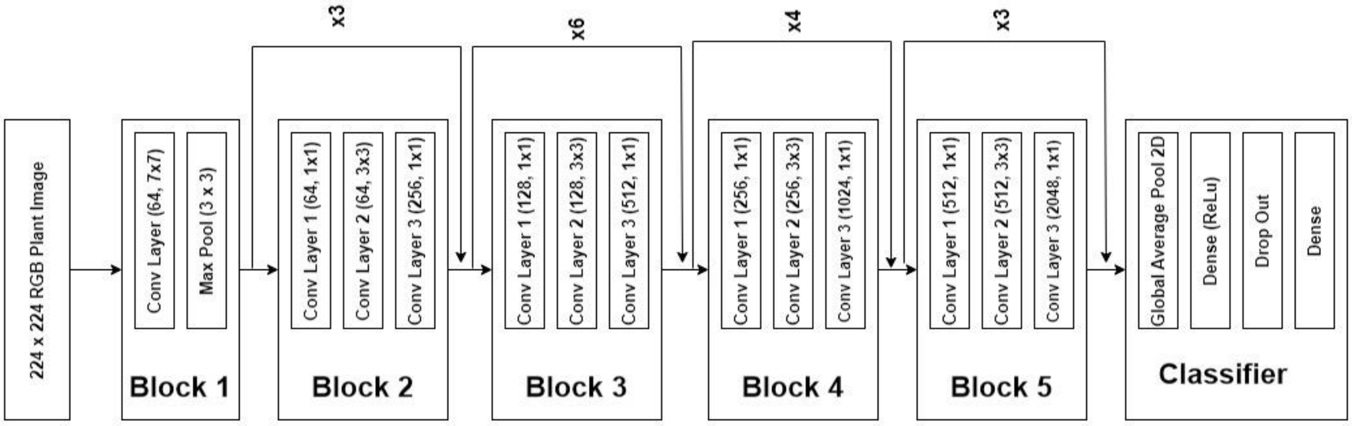


Figure 6.1: Generic architecture of Resnet50

ResNet50 is a deep convolutional neural network architecture used for image classification, object detection, and other computer vision tasks. It consists of 50 convolutional layers, and it builds upon the original ResNet architecture by introducing several improvements, including bottleneck layers and skip connections.

Here is a brief explanation of each layer in ResNet50:

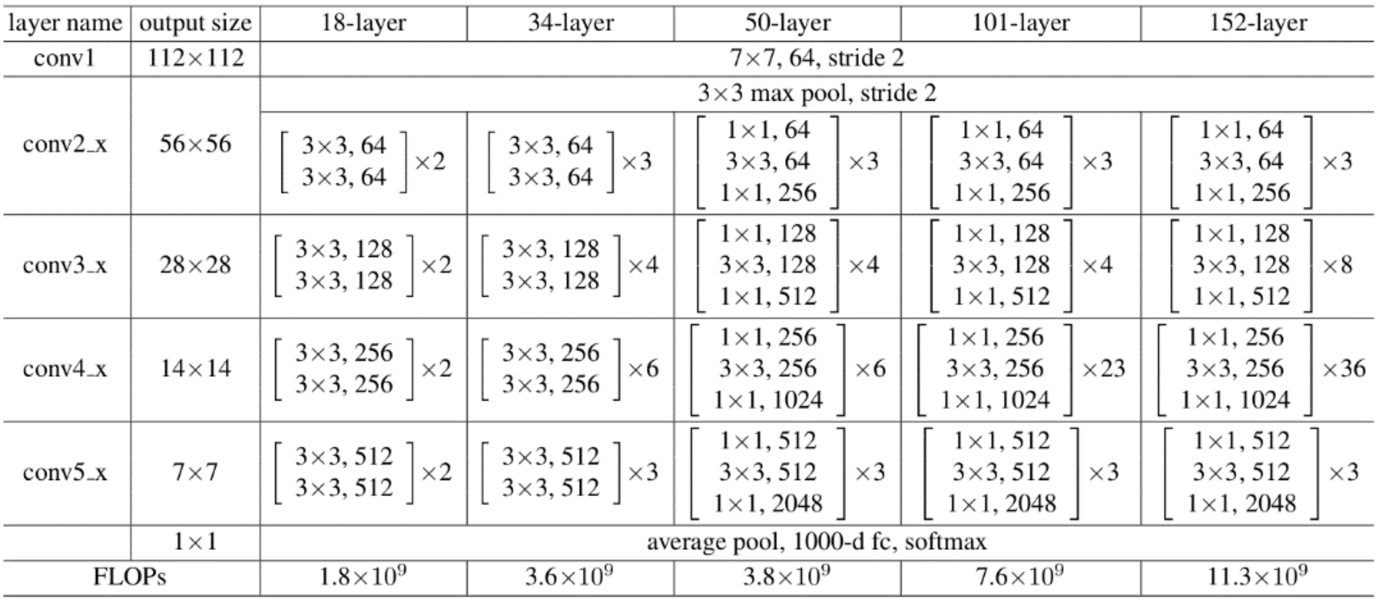
1. **Input layer:** The input layer takes the raw image data as input and prepares it for processing by the rest of the network.

2. **Convolutional layer:** The first layer applies a convolution operation to the input image, which applies a set of filters to the input to extract features from it. This layer has 64 filters with a kernel size of 7x7.
3. **Batch normalization layer:** The output of the convolutional layer is passed through a batch normalization layer, which normalizes the output to improve the stability and performance of the network.
4. **ReLU activation layer:** The output of the batch normalization layer is passed through a rectified linear unit (ReLU) activation layer, which introduces non-linearity into the network.
5. **Max pooling layer:** The output of the ReLU activation layer is then passed through a max pooling layer, which reduces the spatial size of the feature maps and helps to make the network more robust to variations in the input.
6. **Bottleneck layers:** The majority of the ResNet50 V2 network consists of bottleneck layers, which are designed to reduce the computational cost of the network while maintaining its accuracy. Each bottleneck layer consists of three sub-layers: a 1x1 convolutional layer, a 3x3 convolutional layer, and another 1x1 convolutional layer. These layers are used to reduce the number of channels in the input, apply a non-linear activation function, and then increase the number of channels again.
7. **Output layer:** The output layer is a fully connected layer that produces the final output of the network. For image classification, this layer typically has a softmax activation function that produces a probability distribution over the possible classes.

The bottleneck building block is used in the 50-layer ResNet. A bottleneck residual block, also referred to as a "bottleneck", uses 1*1 convolutions to cut down on the number of parameters and matrix multiplications. This makes each layer's training significantly faster. Instead of using a stack of two levels, it employs three layers. As listed in the table below, the 50-layer ResNet design consists of the following components:

- A 7×7 kernel convolution alongside 64 other kernels with a 2-sized stride.
- A max pooling layer with a 2-sized stride.
- 9 more layers 3×3,64 kernel convolution, another with 1×1,64 kernels, and a third with 1×1,256 kernels. These 3 layers are repeated 3 times.
- 12 more layers with 1×1,128 kernels, 3×3,128 kernels, and 1×1,512 kernels, iterated 4 times.
- 18 more layers with 1×1,256 cores, and 2 cores 3×3,256 and 1×1,1024, iterated 6 times.
- 9 more layers with 1×1,512 cores, 3×3,512 cores, and 1×1,2048 cores iterated 3 times (up to this point the network has 50 layers)

Average pooling, followed by a fully connected layer with 1000 nodes, using the softmax activation function.



layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix}3\times3, 64\\3\times3, 64\end{bmatrix}\times2$	$\begin{bmatrix}3\times3, 64\\3\times3, 64\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 64\\3\times3, 64\\1\times1, 256\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 64\\3\times3, 64\\1\times1, 256\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 64\\3\times3, 64\\1\times1, 256\end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix}3\times3, 128\\3\times3, 128\end{bmatrix}\times2$	$\begin{bmatrix}3\times3, 128\\3\times3, 128\end{bmatrix}\times4$	$\begin{bmatrix}1\times1, 128\\3\times3, 128\\1\times1, 512\end{bmatrix}\times4$	$\begin{bmatrix}1\times1, 128\\3\times3, 128\\1\times1, 512\end{bmatrix}\times4$	$\begin{bmatrix}1\times1, 128\\3\times3, 128\\1\times1, 512\end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix}3\times3, 256\\3\times3, 256\end{bmatrix}\times2$	$\begin{bmatrix}3\times3, 256\\3\times3, 256\end{bmatrix}\times6$	$\begin{bmatrix}1\times1, 256\\3\times3, 256\\1\times1, 1024\end{bmatrix}\times6$	$\begin{bmatrix}1\times1, 256\\3\times3, 256\\1\times1, 1024\end{bmatrix}\times23$	$\begin{bmatrix}1\times1, 256\\3\times3, 256\\1\times1, 1024\end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix}3\times3, 512\\3\times3, 512\end{bmatrix}\times2$	$\begin{bmatrix}3\times3, 512\\3\times3, 512\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 512\\3\times3, 512\\1\times1, 2048\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 512\\3\times3, 512\\1\times1, 2048\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 512\\3\times3, 512\\1\times1, 2048\end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

Figure 6.2: Average Pooling

The architecture, which uses 33 filters, is comparable to the VGG net [18], however ResNet is nearly 8 times deeper. The ResNet-50 model with recurrent connections as residual blocks and transfer learning is deployed using 50 parameterized layers.

A network-in-network (NIN) architecture called ResNet is relying on a large number of stacked residual units. The network was built using this set of residual units as its building pieces. Building blocks that make up the ResNet Architecture are made up of a group of residual units. The convolution, pooling, and layer components make up the residual units.

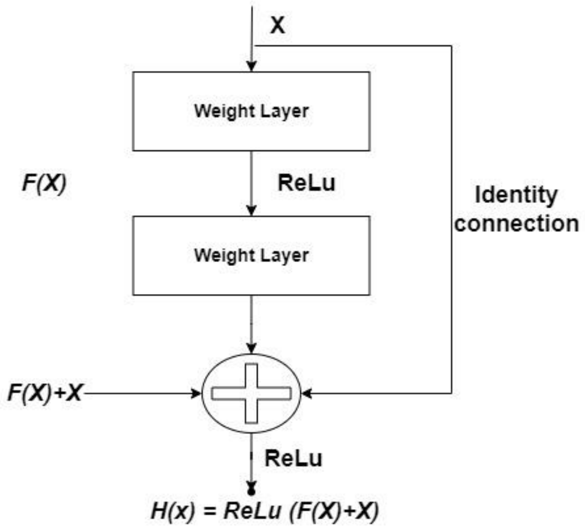


Figure 6.3: Residual Block of Deep Residual Network [17]

Here, each unit can be expressed as,

$$H(x) = ReLU(F(x) + x) \tag{1}$$

where, $H(x)$is an identifying mapping of ReLU.

x are the inputs to the first of layers.

$H(x) - x$is assuming the input and output are of same dimensions.

$F(x) + x$ is the original function.

Now for nth units,

$$y_n = H(x_n) + f(x_n, w) \tag{2}$$

$$x_{n+1} = f(y_n) \tag{3}$$

$$H(x_n) = x_n \tag{4}$$

Where, x_n is the input,

x_{n+1} is the output of nth units,

f Is the residual function of nth units.

$H(x_n)$is an identifying mapping of ReLU

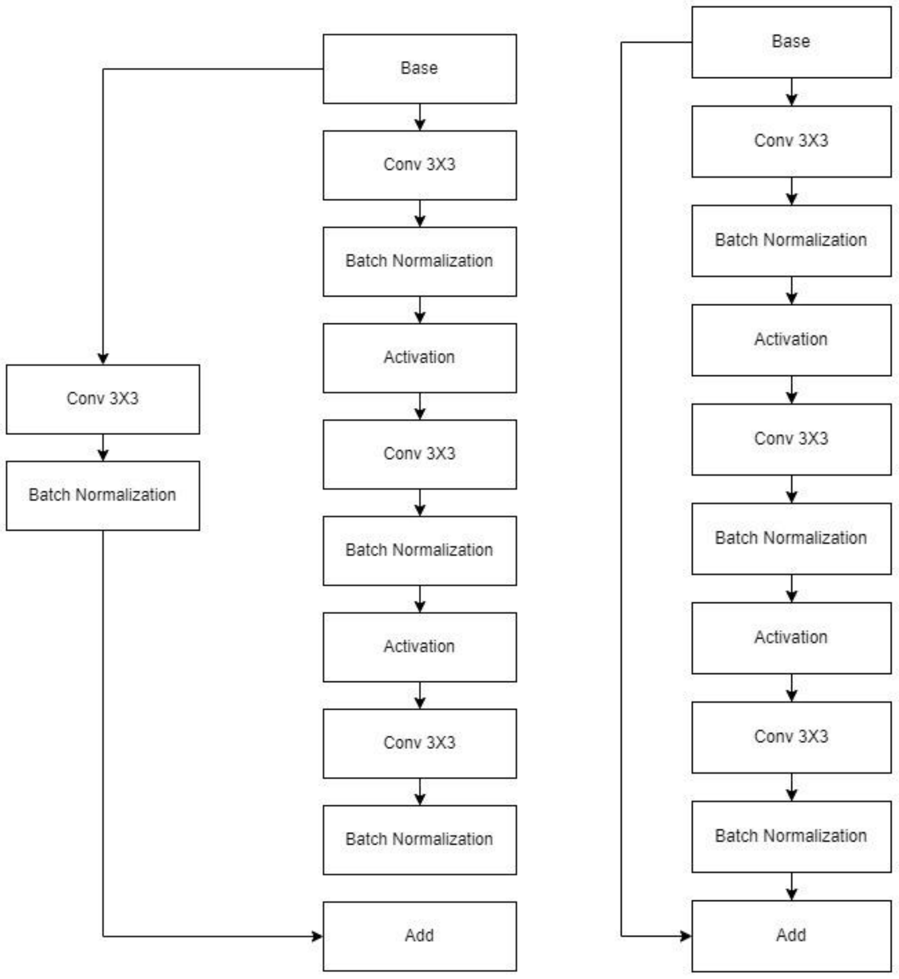


Figure 6.4: Convolution Block and Identity Block

There are two types of blocks in ResNet-50, Identity Block and Convolution Block

If and only if the value of "x" is added to the output layer,

$$input\ size\ ==\ output\ size \tag{6}$$

If not, we include a "convolutional block" in the shortcut path to make the input and output sizes equal.

There are two approaches to equalise the input and output sizes: Conducting 1*1 convolutions and padding the input volume.

The size of the output layer is determined using,

$$\left[\left\{\frac{n+2p-f}{s}\right\}+1\right]^2 \tag{7}$$

Where, $n =$ input image size,
$p =$ padding,
$s =$ stride,
$f =$ number of filters.

For, 1*1 convolutional layer, size of output layer,

$$(\frac{n}{2})^2 \tag{8}$$

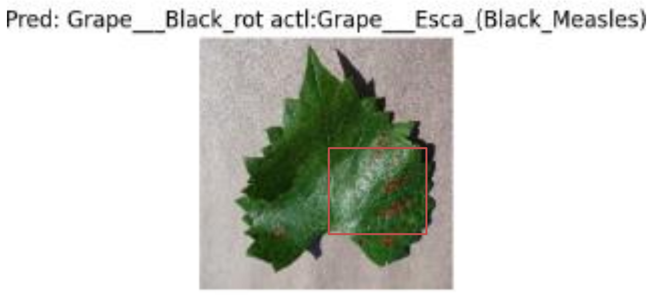
Given, the input size is ‘n.’

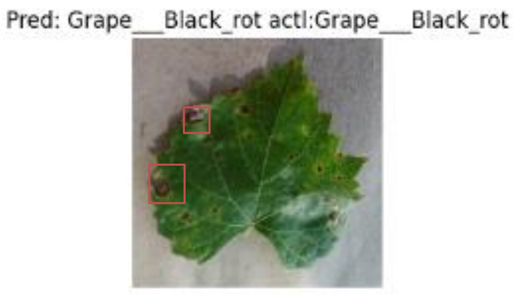
7. RESULTS

1.1. Subjective Result

1.1.1.ResNet50

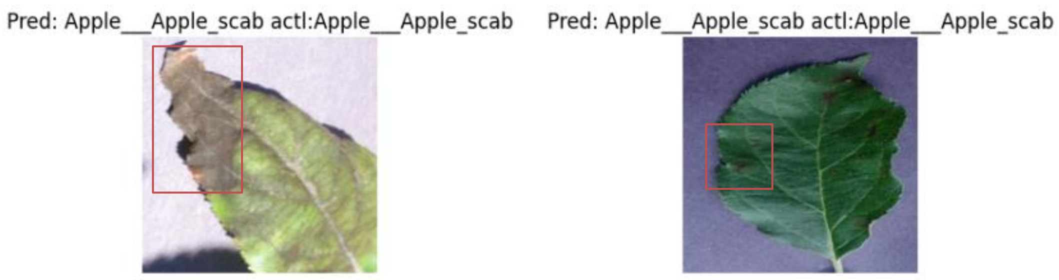
ResNet50 has the highest accuracy among the models listed, with a score of 95.68%. Its precision and recall values are also very high, indicating that it is able to correctly classify both diseased and healthy plants with a high degree of accuracy. This is likely due to its more complex architecture, which allows it to learn more complex representations of the input data. Overall, ResNet50 is a strong choice for plant disease detection tasks.

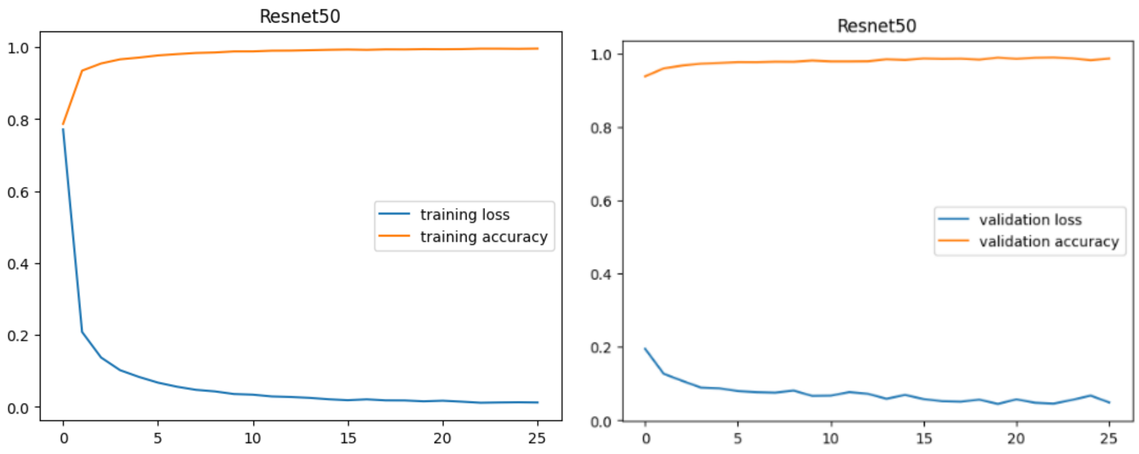




1.1.2.Modified Resnet50

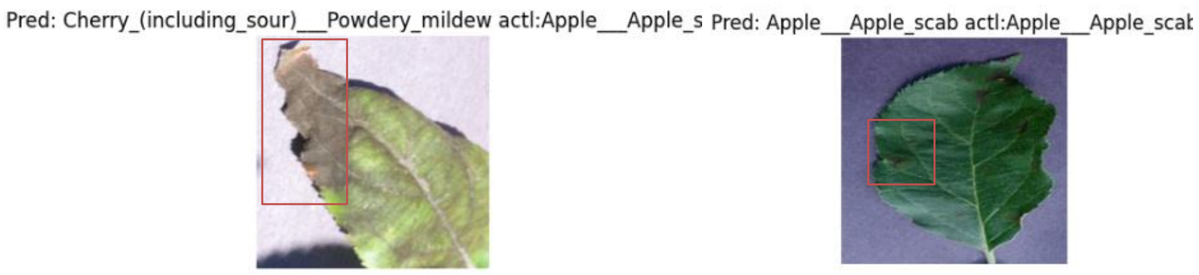
Resnet50 using transfer learning is a variant of Resnet50 that has been fine-tuned on the plant disease detection task, resulting in an even higher accuracy of 98.72%. Its precision and recall values are also very high, indicating that it is able to correctly classify both diseased and healthy plants with a high degree of accuracy. This model is a good choice for tasks where high accuracy is essential, but it may require more computational resources and training time due to its complexity.


Pred: Apple___Apple_scab actl:Apple___Apple_scab
Pred: Apple___Apple_scab actl:Apple___Apple_scab



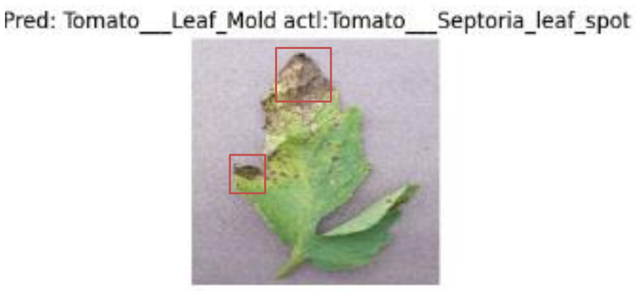
1.1.3. InceptionV3

InceptionV3 has a more complex architecture compared to VGG16, but slightly lower accuracy at 91.68%. However, its precision and recall values are high, indicating that it is able to correctly classify both diseased and healthy plants with a high degree of accuracy. InceptionV3 is designed to handle input images of various sizes, which may make it a good choice for datasets with a large variation in image sizes. This model is a good choice for tasks where high accuracy is essential, but a simpler architecture is preferred over ResNet50.


Pred: Cherry_(including_sour)___Powdery_mildew actl:Apple___Apple_s
Pred: Apple___Apple_scab actl:Apple___Apple_scab

1.1.4. VGG16

VGG16 is a simpler model architecture compared to ResNet50 and InceptionV3, which may explain why its accuracy is slightly lower at 90.33%. However, its precision value is higher than the other models, indicating that it is better at correctly identifying diseased plants. The recall value is also lower, suggesting that it may struggle to correctly identify healthy plants. VGG16 is a good choice for tasks where computational resources are limited, or when simplicity is preferred over high accuracy.



1.2. Objective Result

S.N	Model	Testing Accuracy	Precision	Recall
1.	Inception V3	91.68%	91.95%	91.68%
2.	Vgg16	90.33%	91.49%	90.33%
3.	Resnet 50	95.68%	95.98%	95.68%
4.	Resnet50 (Fine-tuned)	98.72%	98.78%	98.72%

From the metrics table we observed that Resnet50 using Transfer Learning gives the best accuracy.

ResNet50 has ability to effectively learn and extract features from complex image datasets. By leveraging its deep learning capabilities, ResNet50 is able to identify patterns and textures that are indicative of specific diseases. Hence it is good for plant disease detection.

1.3. Comparison of Proposed method with Existing Methods

S.N	Model	Accuracy	Precision	Recall
1.	**Base paper** model -Resnet50 Using Transfer Learning [8]	96%	-	-
2.	**Proposed** model -Modified Resnet50 using transfer Learning	**98.72%**	98.78%	98.72%
3.	Efficient Deep learning model [7]	74.10%	88.13%	94.06%
4.	Deep CNN [12]	98.18%	98.31%	98.16%
5.	Efficient De deep learning model, Backbone Network, BiFPN Network [7]	74.10%	88.13%	94.06%
6.	CNN model with the integration of computer vision (CV) [13]	93.8%	99.1%	95.9%

8. CONCLUSION

In conclusion, the early detection of plant diseases is vital for ensuring sustainable agriculture and food security. In recent years, deep learning-based approaches have shown great promise in this area. Our study proposes a modified ResNet50 using transfer learning approach for plant disease detection, which we compared to existing methods.

The results of our experiments demonstrated that the proposed method outperformed the base paper model and other existing methods, exhibiting superior accuracy, precision, and recall. This finding indicates that the modified ResNet50 using transfer learning approach is an exceptionally effective and efficient method for detecting plant diseases. Additionally, our proposed approach can be incorporated into mobile applications, allowing for increased accessibility for farmers in remote areas. The development and deployment of this model has the potential to significantly contribute to sustainable agriculture and food security, both of which are critical issues faced by the world today. Future research can focus on improving the model's performance by incorporating larger datasets, refining the transfer learning approach, and exploring other deep learning architectures.

In conclusion, our proposed modified ResNet50 using transfer learning approach is a reliable and effective tool for the early detection and management of plant diseases, offering a promising solution to the challenges faced in agriculture and food production.

9.References

[1] Adeleke, B. S., & Babalola, O. O. (2022). Roles of plant endosphere microbes in agriculture-a review. *Journal of Plant Growth Regulation*, *41*(4), 1411-1428.

[2] Khakimov, A., Salakhutdinov, I., Omolikov, A., & Utaganov, S. (2022). Traditional and current-prospective methods of agricultural plant diseases detection: A review. In *IOP Conference series: earth and environmental science* (Vol. 951, No. 1, p. 012002). IOP Publishing.

[3] Talreja, R., Jawrani, V., Watwani, B., Sengupta, S., Rohera, P., & Raghuwanshi, K. S. (2022, May). AgriCare: An Android Application for Detection of Paddy Diseases. In *2022 3rd International Conference for Emerging Technology (INCET)* (pp. 1-6). IEEE.

[4] Pantazi, X. E., Moshou, D., & Tamouridou, A. A. (2019). Automated leaf disease detection in different crop species through image features analysis and One Class Classifiers. *Computers and electronics in agriculture*, *156*, 96-104.

[5] Yatoo, A. A., & Sharma, A. (2021, November). A Novel Model for Automatic Crop Disease Detection. In *2021 Sixth International Conference on Image Information Processing (ICIIP)* (Vol. 6, pp. 310-313). IEEE.

[6] Ma, J., Du, K., Zheng, F., Zhang, L., Gong, Z., & Sun, Z. (2018). A recognition method for cucumber diseases using leaf symptom images based on deep convolutional neural network. *Computers and electronics in agriculture*, *154*, 18-24.

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

[8] Venkataramanan, Aravindhan & Agarwal, Pooja. (2019). Plant Disease Detection and Classification Using Deep Neural Networks.

[9] Syed-Ab-Rahman, S.F., Hesamian, M.H. & Prasad, M. Citrus disease detection and classification using end-to-end anchor-based deep learning model. *Appl Intell* **52**, 927–938 (2022).

[10] M. Kathiravan, K. H. Priya, S. Sreesubha, A. Irumporai, V. S. Kumar, and V. V. Reddy, "ML Algorithm-Based Detection of Leaf Diseases," 2022 4th International Conference on Smart Systems and Inventive Technology (ICSSIT), Tirunelveli,

India, 2022, pp. 1396-1400, doi: 10.1109/ICSSIT53264.2022.9716430.

[11] Wu, Y. (2021). Identification of maize leaf diseases based on convolutional neural network. In *Journal of physics: Conference series* (Vol. 1748, No. 3, p. 032004). IOP Publishing.

[12] Mondal, Joyanta & Islam, Md & Zabeen, Sarah & Islam, A. B. M. Alim Al & Noor, Jannatun. (2022). Plant Leaf Disease Network (PLeaD-Net): Identifying Plant Leaf Diseases through Leveraging Limited-Resource Deep Convolutional Neural Network.

[13] Ullah, F., Khan, R. U., Khan, K., Albattah, W., & Qamar, A. M. (2021). Image-based detection of plant diseases: From classical machine learning to deep learning journey. Wireless Communications and Mobile Computing, 2021, 5541859.

[14] Jiang, F., Lu, Y., Chen, Y., Cai, D., & Li, G. (2020). Image recognition of four rice leaf diseases based on deep learning and support vector machine. *Computers and Electronics in Agriculture*, *179*, 105824.

[15] Hughes, D., & Salathé, M. (2015). An open access repository of images on plant health to enable the development of mobile disease diagnostics. *arXiv preprint arXiv:1511.08060*.

[16] Mohanty, S. P., Hughes, D. P., & Salathé, M. (2016). Using deep learning for image-based plant disease detection. *Frontiers in plant science*, *7*, 1419.

[17] He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 770-778).

[18] Simonyan, K., & Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*.

[19] Dataset link- https://github.com/spMohanty/PlantVillage-Dataset/tree/master/raw/color

APPENDIX A – SAMPLE CODE

Pre-Processing and Data Augmentation

```
import os
import urllib.request
import zipfile

url = 'https://github.com/spMohanty/PlantVillage-Dataset/archive/master.zip'

data_dir = '/content/dataset'

if not os.path.exists(data_dir):
    os.makedirs(data_dir)

print("Downloading dataset...")
zip_path, _ = urllib.request.urlretrieve(url)


print("Extracting dataset...")
with zipfile.ZipFile(zip_path, 'r') as zip_ref:
    zip_ref.extractall(data_dir)

dataset_dir = os.path.join(data_dir, 'PlantVillage-Dataset-master')

print("Dataset directories:")
for dir_name in os.listdir(dataset_dir):
    if os.path.isdir(os.path.join(dataset_dir, dir_name)):
        print("- " + dir_name)

import tensorflow as tf
from tensorflow.keras import layers
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

tf.__version__

input_folder = "/content/dataset/PlantVillage-Dataset-master/raw/color"
output_folder = "/content/dataset/PlantVillage-Dataset-master/split_data"
splitfolders.ratio(input_folder, output=output_folder, seed=42, ratio=(0.8, 0.1, 0.1))

train_ds = tf.keras.preprocessing.image_dataset_from_directory(
    directory=os.path.join(output_folder, "train"),
    batch_size=batch_size,
    color_mode="rgb",
    image_size=(img_height, img_width),
    shuffle=True,
    seed=42,
)

val_ds = tf.keras.preprocessing.image_dataset_from_directory(
    directory=os.path.join(output_folder, "val"),
    batch_size=batch_size,
    color_mode="rgb",
    image_size=(img_height, img_width),
    shuffle=True,
    seed=42,
)

test_ds = tf.keras.preprocessing.image_dataset_from_directory(
    directory=os.path.join(output_folder, "test"),
    batch_size=batch_size,
    color_mode="rgb",
    image_size=(img_height, img_width),
    shuffle=False,
    seed=42,
)

class_names = train_ds.class_names
print(class_names)

preprocess_layer = tf.keras.Sequential([
    layers.Rescaling(1./255),
    layers.Resizing(224,224)
   ])

normalized_ds = train_ds.map(lambda x, y: (preprocess_layer(x), y))
image_batch, labels_batch = next(iter(normalized_ds))
first_image = image_batch[0]
print(np.min(first_image), np.max(first_image))

data_augmentation = tf.keras.Sequential([
  tf.keras.layers.RandomFlip('horizontal'),
  tf.keras.layers.RandomRotation(0.2),
])
```

Fine Tuned Model Training

```
import tensorflow as tf
from tensorflow.keras import layers
resnet_model= tf.keras.applications.ResNet50V2(include_top=False,
                   input_shape=(224,224,3),
                   weights='imagenet',
                   )

resnet_model.trainable = True

import pandas as pd

layerss = [(layer, layer.name, layer.trainable) for layer in resnet_model.layers]
pd.DataFrame(layerss, columns=['Layer Type', 'Layer Name', 'Layer Trainable'])

fine_tune_at = 154 #143

for layer in resnet_model.layers[:fine_tune_at]:
  layer.trainable = False

resnet_model.summary()

inputs = layers.Input(shape=(256, 256, 3))
x = preprocess_layer(inputs)
x=data_augmentation(x)

x = resnet_model(x, training=False)

x = layers.GlobalAveragePooling2D()(x)
x = layers.Dense(512, activation='relu')(x)
x = layers.Dropout(0.2)(x)
outputs = layers.Dense(38, activation='softmax')(x)
```

```
tf2_model = tf.keras.Model(inputs, outputs)

tf2_model.summary()

#very low learning rate
tf2_model.compile(optimizer=tf.keras.optimizers.Adam(1e-5),
loss='sparse_categorical_crossentropy', metrics=['accuracy'])

loss0, accuracy0 = tf2_model.evaluate(val_ds)

from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint
early_stopping = EarlyStopping(monitor='val_loss', patience=6, verbose=1,mode="min",
restore_best_weights=True)
model_checkpoint = ModelCheckpoint('best_resnet_model.h5', save_best_only=True)

tf2_history = tf2_model.fit(train_ds, epochs=30, validation_data=val_ds, callbacks=[early_stopping,
model_checkpoint])

plt.plot(tf2_history.history['loss'], label = 'training loss')
plt.plot(tf2_history.history['accuracy'], label = 'training accuracy')
plt.legend()
plt.title('Resnet50')
```

FRONTEND CODE

INDEX.HTML

```
{% block body %}
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
    <!-- css files -->
    <link rel="stylesheet" href="{{ url_for('static', filename='css/bootstrap.css') }}">
    <link rel="stylesheet" href="{{ url_for('static', filename='css/font-awesome.min.css') }}">
    <link rel="stylesheet" href="{{ url_for('static', filename='css/style.css') }}">
    <style>
        header {
            background-color: rgb(140, 10, 40);
            margin-top: 0rem;
            display: block;
        }
    </style>
</head>
<body>
<nav class="navbar navbar-expand-lg navbar-dark bg-black static-top">
    <div class="container">
        <a class="navbar-brand" href="{{ url_for('home') }}">
            <img src="{{ url_for('static', filename='images/AGRIGUARD.png') }}" style="width: 90px;
height: 55px" alt="">
        </a>
        <button class="navbar-toggler" type="button" data-toggle="collapse" data-
target="#navbarResponsive"
                aria-controls="navbarResponsive" aria-expanded="false" aria-label="Toggle navigation">
            <span class="navbar-toggler-icon"></span>
        </button>
        <div class="collapse navbar-collapse" id="navbarResponsive">
            <ul class="navbar-nav ml-auto">
                <li class="nav-item active">
                    <a class="nav-link" href="{{ url_for('home') }}">Home
                        <span class="sr-only">(current)</span>
                    </a>
                </li>
                <li class="nav-item">
                    <a class="nav-link" href="FAQ.html">FAQs</a>
                </li>
            </ul>
        </div>
    </div>
</nav>

<section class="banner_w3lspvt" id="home">
    <div class="csslider infinity" id="slider1">
        <div class="banner-top">
            <div class="overlay">
                <div class="container">
                    <div class="w3layouts-banner-info text-center">
                        <h3 class="text-wh">AGRIGUARD</h3>
                        <h4 class="text-wh mx-auto my-4"><b>Empowering Farmers with Cutting-Edge
Plant Disease Detection Technology</b></h4>
                    </div>
                </div>
            </div>
        </div>
    </div>
</section>

<section class="blog py-5">
    <div class="container py-md-5">
        <h3 class="heading mb-sm-5 mb-4 text-center"> Our Services</h3>
        <div class="row blog-grids" style="justify-content: center;">
            <div class="col-lg-4 col-md-6 blog-middle mb-lg-0 mb-sm-5 pb-lg-0 pb-5 text-center">
```

```
                <img src="{{ url_for('static', filename='images/s4.jpg') }}" class="img-fluid" alt="" />
                <a href="{{ url_for('disease') }}">
                    <div class="blog-info">
                        <h4>Plant Detection</h4>
                        <p class="mt-2">Predicting the name and causes of crop disease and suggestions to
cure it</p>
                    </div>
                </a>
            </div>
        </div>
    </div>
</section>

<footer class="text-center py-5">
    <div class="container py-md-3">
        <!-- logo -->
        <h2 class="logo2 text-center">
            <a href="{{ url_for('home') }}">
                CAPSTONE PROJECT
            </a>
        </h2>
        <!-- //logo -->
        <!-- address -->
        <div class="contact-left-footer mt-4">
        </div>
        <div class="w3l-copy text-center">
            <p class="text-da">
                <ol>UJJWAL TIWARI</ol>
                <ol>TARNG GUPTA</ol>
                <ol>UTKARSH SINGH </ol>
                <br>
            </p>
        </div>
    </div>
</footer>
</body>
</html>
{% endblock %}
```

DISEASE.HTML

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

<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Plant Disease Prediction</title>
    <style>
        body {
            background-color: pink;
            font-family: Arial, sans-serif;
            margin: 0;
            padding: 0;
            display: flex;
            justify-content: center;
            align-items: center;
            height: 100vh;
            flex-direction: column;
        }

        h1 {
            margin-bottom: 20px;
        }

        input[type="file"] {
            margin-bottom: 10px;
        }
    </style>
</head>

<body>
    <h1>Upload an image of a plant leaf</h1>
    <form action="/predict" method="post" enctype="multipart/form-data">
        <input type="file" name="image" accept="image/*">
        <input type="submit" value="Predict">
    </form>
</body>

</html>
```

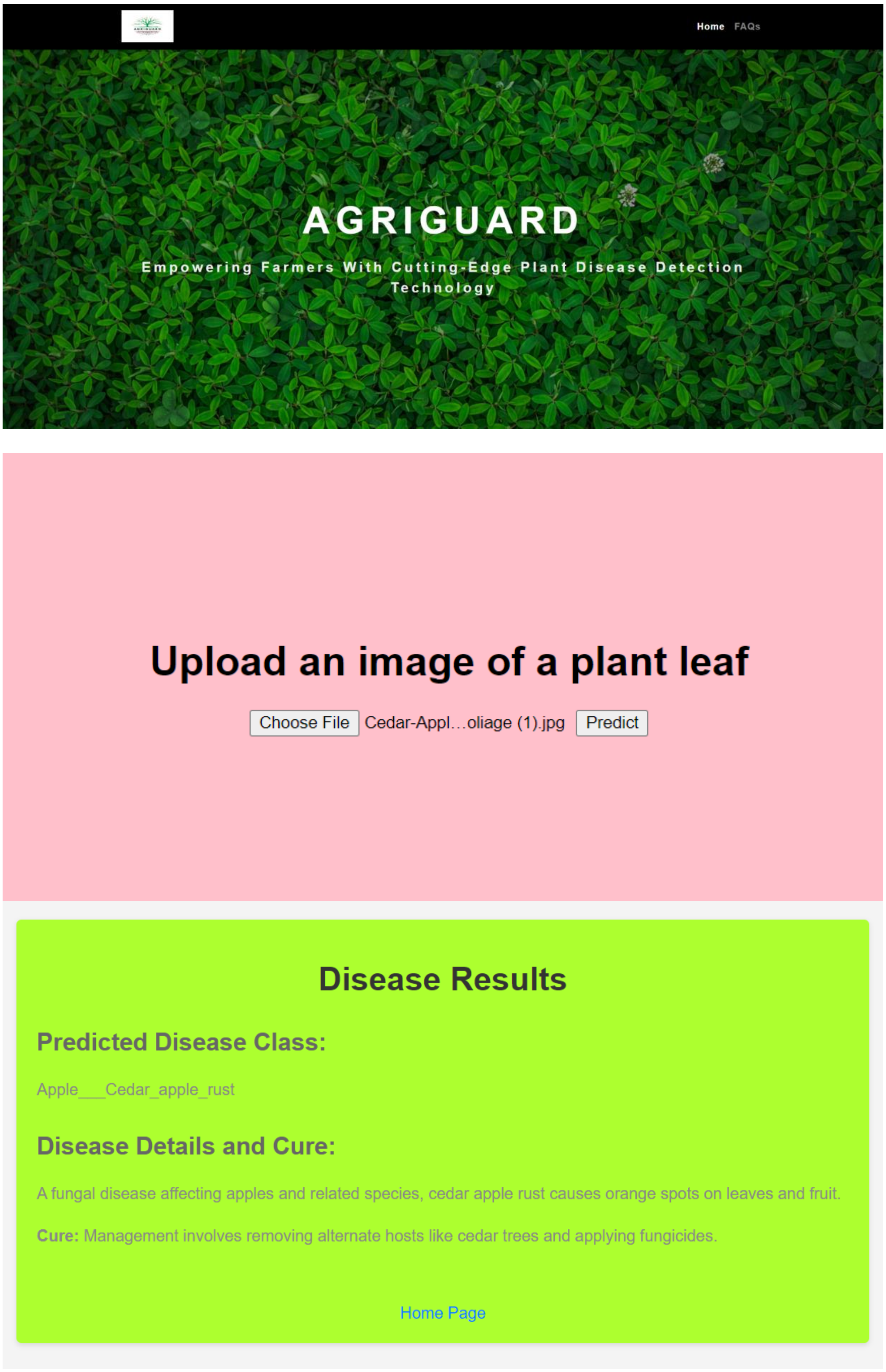
RESULT.HTML

```
<!DOCTYPE html>
<html>

<head>
    <title>Agriguard - Results</title>
    <style>
        body {
            font-family: Arial, sans-serif;
            background-color: #f4f4f4;
            margin: 0;
            padding: 0;
        }
        .container {
            max-width: 800px;
            margin: 50px auto;
            padding: 20px;
            background-color: greenyellow;
            border-radius: 5px;
            box-shadow: 0 2px 5px rgba(0, 0, 0, 0.1);
        }

        h1 {
            color: #333;
            text-align: center;
        }

        h2 {
            color: #666;
            margin-top: 30px;
        }

        p {
            color: #888;
            line-height: 1.6;
        }

        a {
            display: block;
            text-align: center;
            margin-top: 20px;
            text-decoration: none;
            color: #007bff;
        }

        a:hover {
            text-decoration: underline;
        }
    </style>
</head>

<body>
    <div class="container">
        <h1>Disease Results</h1>
        {% if prediction %}
        <h2>Predicted Disease Class:</h2>
        <p>{{ prediction }}</p>
        <h2>Disease Details and Cure:</h2>
        {% if prediction == 'Apple___Apple_scab' %}
        <p>This fungal disease affects apple trees, causing dark lesions on leaves and fruit, leading to
reduced yield and
            quality.</p>
        <p><strong>Cure:</strong> Management involves cultural practices like pruning to improve
air circulation, and
            fungicide applications.</p>
        {% elif prediction == 'Apple___Black_rot' %}
        <p>Another fungal disease of apple trees, black rot causes brown lesions on fruit and foliage,
leading to fruit
            decay.</p>
        <p><strong>Cure:</strong> Cultural practices like sanitation and fungicide applications are
key for management.</p>
        {% elif prediction == 'Apple___Cedar_apple_rust' %}
        <p>A fungal disease affecting apples and related species, cedar apple rust causes orange spots
on leaves and fruit.
        </p>
        <p><strong>Cure:</strong> Management involves removing alternate hosts like cedar trees
and applying fungicides.</p>
        {% elif prediction == 'Apple___healthy' %}
        <p>The apple tree appears healthy with no signs of disease.</p>
        {% elif prediction == 'Blueberry___healthy' %}
        <p>The blueberry plant appears healthy with no signs of disease.</p>
        {% elif prediction == 'Cherry_(including_sour)___Powdery_mildew' %}
        <p>This fungal disease affects cherry trees, causing white powdery patches on leaves and
fruit.</p>
        <p><strong>Cure:</strong> Management includes fungicide applications and pruning to
improve air circulation.</p>
        {% elif prediction == 'Cherry_(including_sour)___healthy' %}
        <p>The cherry tree appears healthy with no signs of disease.</p>
        {% elif prediction == 'Corn_(maize)___Cercospora_leaf_spot Gray_leaf_spot' %}
        <p>These fungal diseases affect corn, causing leaf lesions which can reduce photosynthesis and
yield.</p>
        <p><strong>Cure:</strong> Crop rotation and fungicide applications are common management
strategies.</p>
        {% elif prediction == 'Corn_(maize)___Common_rust_' %}
        <p>These fungal diseases also affect corn, causing characteristic rust-colored lesions on
leaves.</p>
        <p><strong>Cure:</strong> Management involves resistant crop varieties and fungicide
applications.</p>
        {% elif prediction == 'Corn_(maize)___Northern_Leaf_Blight' %}
        <p>Another fungal disease of corn, northern leaf blight causes cigar-shaped lesions on leaves,
leading to yield
            loss.</p>
        <p><strong>Cure:</strong> Management involves resistant crop varieties and fungicide
applications.</p>
        {% elif prediction == 'Corn_(maize)___healthy' %}
        <p>The corn plant appears healthy with no signs of disease.</p>
        {% elif prediction == 'Grape___Black_rot' %}
        <p>This fungal disease affects grapes, causing brown lesions on leaves and fruit, leading to
reduced yield and
            quality.</p>
        <p><strong>Cure:</strong> Management includes cultural practices like pruning and fungicide
applications.</p>
        {% elif prediction == 'Grape___Esca_(Black_Measles)' %}
        <p>A fungal disease affecting grapevines, causing leaf spots and vine decline.</p>
        <p><strong>Cure:</strong> Management involves pruning, cultural practices, and fungicide
applications.</p>
        {% elif prediction == 'Grape___Leaf_blight_(Isariopsis_Leaf_Spot)' %}
        <p>Another fungal disease of grapevines, leaf blight causes spots on leaves, leading to reduced
yield.</p>
        <p><strong>Cure:</strong> Management includes fungicide applications and cultural
practices.</p>
        {% elif prediction == 'Grape___healthy' %}
        <p>The grapevine appears healthy with no signs of disease.</p>
        {% elif prediction == 'Orange___Haunglongbing_(Citrus_greening)' %}
        <p>A bacterial disease affecting citrus trees, citrus greening causes yellowing and decline of
leaves and fruit.</p>
        <p><strong>Cure:</strong> Management includes removing infected trees and controlling
vector insects.</p>
        {% elif prediction == 'Peach___Bacterial_spot' %}
        <p>This bacterial disease affects peach trees, causing dark lesions on leaves and fruit, leading
to reduced yield
            and quality.</p>
        <p><strong>Cure:</strong> Management involves cultural practices and copper-based
fungicides.</p>
        {% elif prediction == 'Peach___healthy' %}
        <p>The peach tree appears healthy with no signs of disease.</p>
        {% elif prediction == 'Pepper,_bell___Bacterial_spot' %}
        <p>This bacterial disease affects bell peppers, causing dark lesions on leaves and fruit.</p>
        <p><strong>Cure:</strong> Management includes sanitation and copper-based fungicides.</p>
        {% elif prediction == 'Pepper,_bell___healthy' %}
        <p>The bell pepper plant appears healthy with no signs of disease.</p>
        {% elif prediction == 'Potato___Early_blight' %}
        <p>This fungal disease affects potatoes, causing dark lesions on leaves and tubers, leading to
reduced yield and
            quality.</p>
        <p><strong>Cure:</strong> Management involves crop rotation, resistant varieties, and
fungicide applications.</p>
        {% elif prediction == 'Potato___Late_blight' %}
        <p>Another fungal disease of potatoes, late blight causes dark lesions on leaves and stems,
leading to rapid plant
            decline.</p>
        <p><strong>Cure:</strong> Management includes cultural practices and fungicide
applications.</p>
        {% elif prediction == 'Potato___healthy' %}
        <p>The potato plant appears healthy with no signs of disease.</p>
        {% elif prediction == 'Raspberry___healthy' %}
        <p>The raspberry plant appears healthy with no signs of disease.</p>
        {% elif prediction == 'Soybean___healthy' %}
        <p>The soybean plant appears healthy with no signs of disease.</p>
        {% elif prediction == 'Squash___Powdery_mildew' %}
        <p>This fungal disease affects squash, causing white powdery patches on leaves and fruit.</p>
        <p><strong>Cure:</strong> Management includes cultural practices and fungicide
applications.</p>
        {% elif prediction == 'Strawberry___Leaf_scorch' %}
        <p>This disease causes browning and death of leaf tissue in strawberry plants.</p>
        <p><strong>Cure:</strong> Management involves sanitation and cultural practices.</p>
        {% elif prediction == 'Strawberry___healthy' %}
        <p>The strawberry plant appears healthy with no signs of disease.</p>
        {% elif prediction == 'Tomato___Bacterial_spot' %}
        <p>This bacterial disease affects tomatoes, causing dark, water-soaked lesions on leaves and
fruit.</p>
        <p><strong>Cure:</strong> Management involves crop rotation, sanitation, and copper-based
bactericides.</p>
        {% elif prediction == 'Tomato___Early_blight' %}
        <p>This fungal disease of tomatoes is characterized by dark concentric lesions on lower
leaves.</p>
        <p><strong>Cure:</strong> Management includes crop rotation, fungicide applications, and
removing infected plant
            debris.</p>
        {% elif prediction == 'Tomato___Late_blight' %}
        <p>Late blight is a fungal disease causing dark, water-soaked lesions on leaves, stems, and
fruit.</p>
        <p><strong>Cure:</strong> Management involves fungicide applications, cultural practices,
and removing infected
            plant material.</p>
        {% elif prediction == 'Tomato___Leaf_Mold' %}
        <p>Leaf mold is a fungal disease causing yellowing and fuzzy growth on the undersides of
leaves.</p>
        <p><strong>Cure:</strong> Management includes improving air circulation, avoiding
overhead watering, and fungicide
            applications.</p>
        {% elif prediction == 'Tomato___Septoria_leaf_spot' %}
        <p>Septoria leaf spot is a fungal disease characterized by small, dark spots with yellow halos
on leaves.</p>
        <p><strong>Cure:</strong> Management involves crop rotation, sanitation, and fungicide
applications.</p>
        {% elif prediction == 'Tomato___Spider_mites Two-spotted_spider_mite' %}
        <p>Spider mites are tiny pests that feed on tomato leaves, causing stippling, webbing, and leaf
yellowing.</p>
```

```
        <p><strong>Cure:</strong> Management involves cultural practices and applying miticides if
infestation levels are
            high.</p>
        {% elif prediction == 'Tomato___Target_Spot' %}
        <p>Target spot is a fungal disease causing dark, concentric lesions on leaves and fruit.</p>
        <p><strong>Cure:</strong> Management includes crop rotation, sanitation, and fungicide
applications.</p>
        {% elif prediction == 'Tomato___Tomato_Yellow_Leaf_Curl_Virus' %}
        <p>Yellow leaf curl virus is a viral disease causing yellowing, curling, and stunting of
leaves.</p>
        <p><strong>Cure:</strong> Management involves using virus-resistant tomato varieties,
controlling whiteflies, and
            removing infected plants.</p>
        {% elif prediction == 'Tomato___Tomato_mosaic_virus' %}
        <p>Mosaic virus causes mottling, distortion, and stunting of leaves.</p>
        <p><strong>Cure:</strong> Management involves using virus-free seed, controlling aphid
vectors, and removing
            infected plants promptly.</p>
        {% elif prediction == 'Tomato___healthy' %}
        <p>The tomato plant appears healthy with no signs of disease.</p>
        {% endif %}
        {% endif %}
        <br>
        <a href="/">Home Page</a>
    </div>
</body>

</html>
```

FLASK SERVER (APP.py)

```
import os
from flask import Flask, render_template, request
from werkzeug.utils import secure_filename
import keras.utils as Image
import numpy as np
import tensorflow as tf
from keras.applications.resnet import decode_predictions, preprocess_input

os.environ['TF_ENABLE_ONEDNN_OPTS'] = '0'

app = Flask(__name__)

interpreter = tf.lite.Interpreter(model_path="model/tf2_lite_model.tflite")
interpreter.allocate_tensors()

disease_classes = ['Apple___Apple_scab', 'Apple___Black_rot', 'Apple___Cedar_apple_rust',
'Apple___healthy', 'Blueberry___healthy', 'Cherry_(including_sour)___Powdery_mildew',
'Cherry_(including_sour)___healthy', 'Corn_(maize)___Cercospora_leaf_spot Gray_leaf_spot',
'Corn_(maize)___Common_rust_', 'Corn_(maize)___Northern_Leaf_Blight',
'Corn_(maize)___healthy', 'Grape___Black_rot', 'Grape___Esca_(Black_Measles)',
'Grape___Leaf_blight_(Isariopsis_Leaf_Spot)', 'Grape___healthy',
'Orange___Haunglongbing_(Citrus_greening)', 'Peach___Bacterial_spot', 'Peach___healthy',
'Pepper,_bell___Bacterial_spot', 'Pepper,_bell___healthy', 'Potato___Early_blight',
'Potato___Late_blight', 'Potato___healthy', 'Raspberry___healthy', 'Soybean___healthy',
'Squash___Powdery_mildew', 'Strawberry___Leaf_scorch', 'Strawberry___healthy',
'Tomato___Bacterial_spot', 'Tomato___Early_blight', 'Tomato___Late_blight',
'Tomato___Leaf_Mold', 'Tomato___Septoria_leaf_spot', 'Tomato___Spider_mites Two-
spotted_spider_mite', 'Tomato___Target_Spot', 'Tomato___Tomato_Yellow_Leaf_Curl_Virus',
'Tomato___Tomato_mosaic_virus', 'Tomato___healthy']

def predict(image):
    input_details = interpreter.get_input_details()
    output_details = interpreter.get_output_details()
    interpreter.set_tensor(input_details[0]['index'], image)
    interpreter.invoke()
    output_data = interpreter.get_tensor(output_details[0]['index'])

    predicted_class_index = np.argmax(output_data)
    predicted_class = disease_classes[predicted_class_index]
    return predicted_class

@app.route('/')
def home():
    return render_template('index.html')

@app.route('/disease')
def disease():
    return render_template('disease.html')

@app.route('/predict', methods=['POST'])
def make_prediction():
    if request.method == "POST":
        file = request.files["image"]
        if file:
            filename = secure_filename(file.filename)
            file.save(filename)
            img = Image.load_img(filename, target_size=(256, 256))
            x = Image.img_to_array(img)
            x = np.expand_dims(x, axis=0)
            x = preprocess_input(x)
            predicted_class = predict(x)
            return render_template('result.html', prediction=predicted_class)
    return "Error"

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


Home FAQs

AGRIGUARD

Empowering Farmers With Cutting-Edge Plant Disease Detection Technology

Upload an image of a plant leaf

Choose File Cedar-Appl...oliage (1).jpg Predict

Disease Results

Predicted Disease Class:

Apple___Cedar_apple_rust

Disease Details and Cure:

A fungal disease affecting apples and related species, cedar apple rust causes orange spots on leaves and fruit.

Cure: Management involves removing alternate hosts like cedar trees and applying fungicides.

Home Page

AgriGuard: Plant Disease Detection Technology

Ujjwal Tiwari | Utkarsh Singh | Tarang Gupta |Dr.Naresh K | SCOPE

Introduction

Early disease detection is critical for effective control measures in crop production Utilizing image recognition and machine learning, a CNN model aids in identifying common plant diseases promptly. By analyzing patterns and features in images, this technology enables farmers to address potential issues promptly, minimizing crop losses and preventing the spread of diseases to nearby plants.

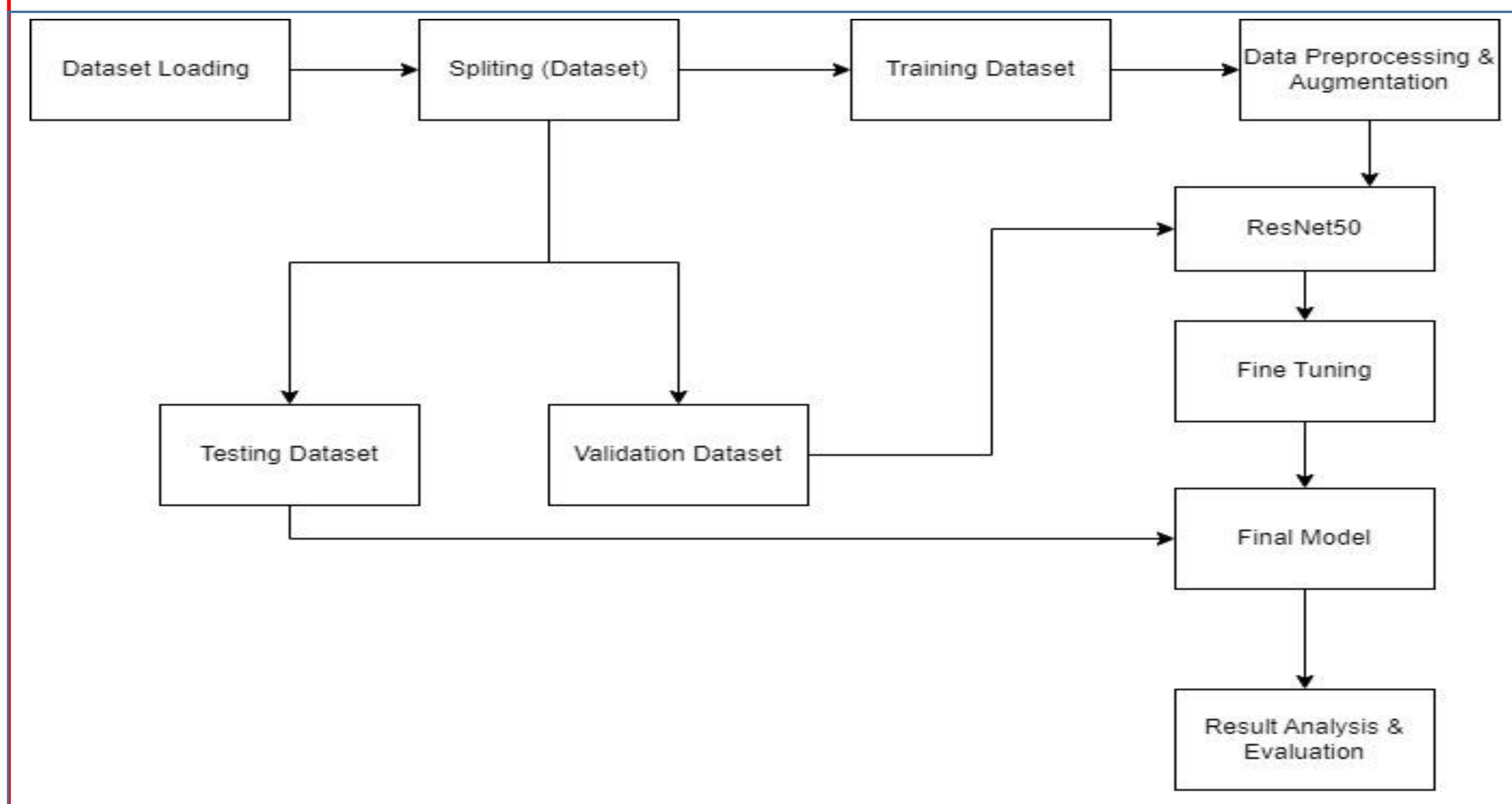
Motivation

Leveraging cutting-edge technology like image recognition and machine learning, our project endeavors to provide farmers with efficient tools to swiftly identify and combat common plant diseases. By enabling proactive intervention, we aim to minimize crop losses and contribute to a more resilient and sustainable agricultural ecosystem.

Scope of the Project

The project scope includes collecting and curating indigenous apple plant image datasets, developing a deep learning model for disease detection, training and validating the model, deploying it for farmer use, assessing its impact on disease management and agricultural productivity, and ensuring continuous improvement over time.

Methodology



As shown above flowchart, we have outlined a process in our framework that is divided into multiple steps.

1.Dataset: Utilized the PlantVillage dataset comprising 54,306 images of 38 plant species, categorized into healthy and diseased classes, to develop a machine learning model for plant disease detection.

2.Data Splitting : Partitioned the dataset into training (80%), validation (10%), and testing (10%) sets to ensure model evaluation, prevent overfitting, and assess generalization ability.

3.Data Pre-Processing: Standardized image resolutions to 256x256 pixels and normalized pixel values to [0, 1] range, enhancing uniformity and facilitating model learning.

4.Data Augmentation :Applied random transformations such as rotations, translations, flips, and shears to diversify training data and improve model robustness against variations in plant leaf images.

5.ResNet 50: Leveraged the ResNet50 architecture, comprising 50 layers of residual networks, for its effectiveness in handling complex tasks and mitigating training difficulties encountered in deep CNN networks.

6.Fine Tuning: Employed fine-tuning by adapting pre-trained ResNet50 weights for plant disease classification, freezing layers up to the 154th layer, and optimizing hyperparameters like batch size, epochs, and learning rate.

The main equation of the Resnet model

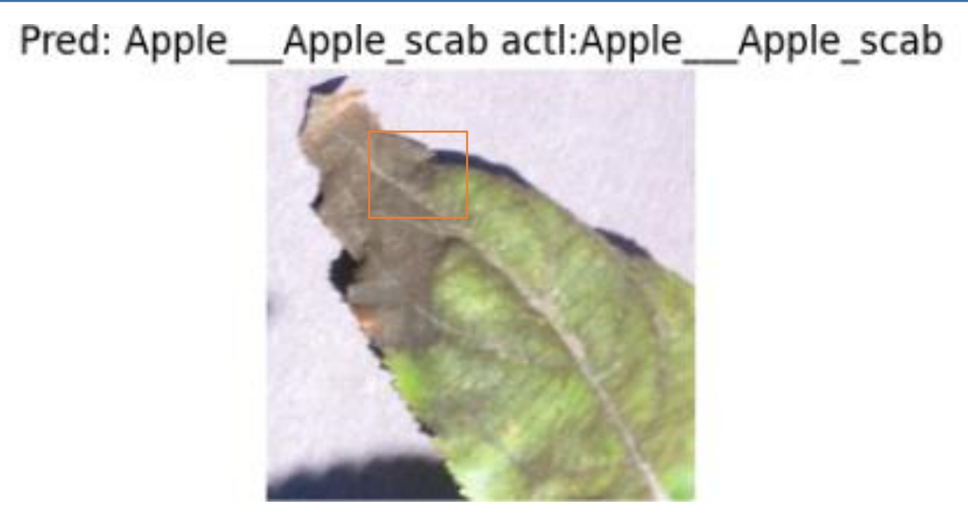
$$\boldsymbol{H(x) = ReLU(F(x) + x)}$$

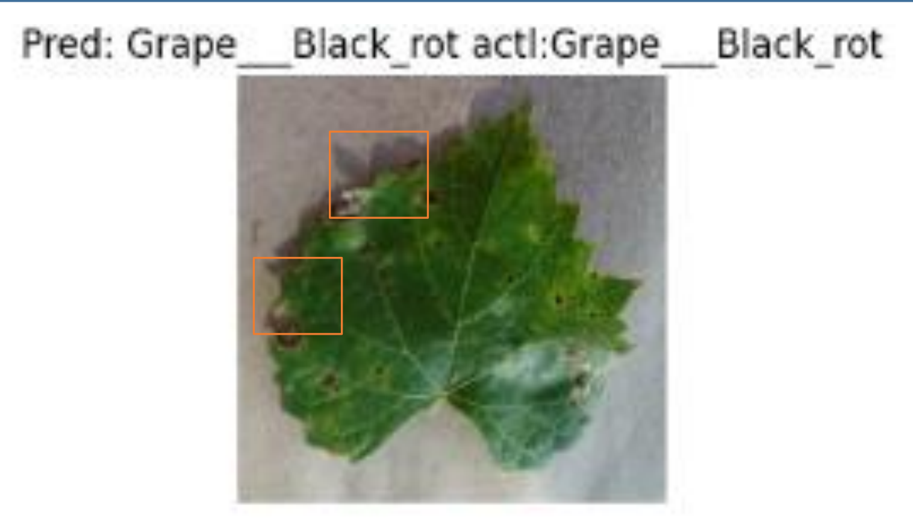
where, $H(x)$is an identifying mapping of ReLU. x are the inputs to the first of layers.

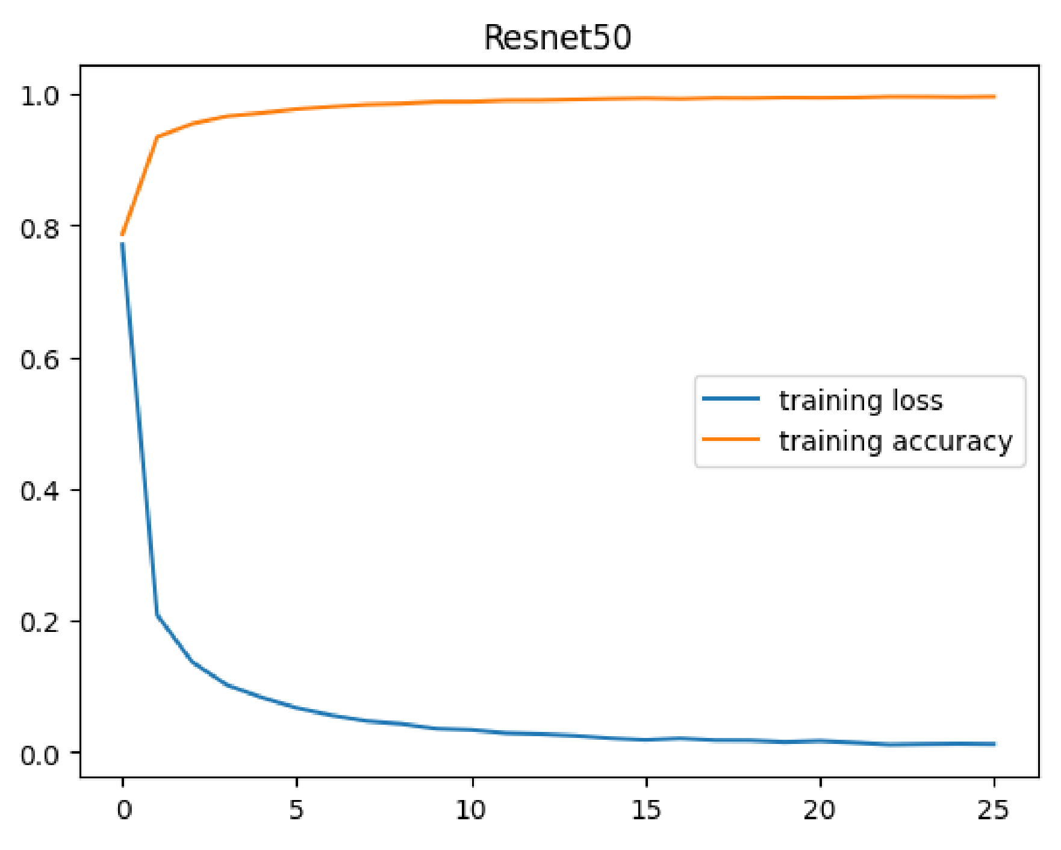
$H(x) - x$ is assuming the input and output are of same dimensions.

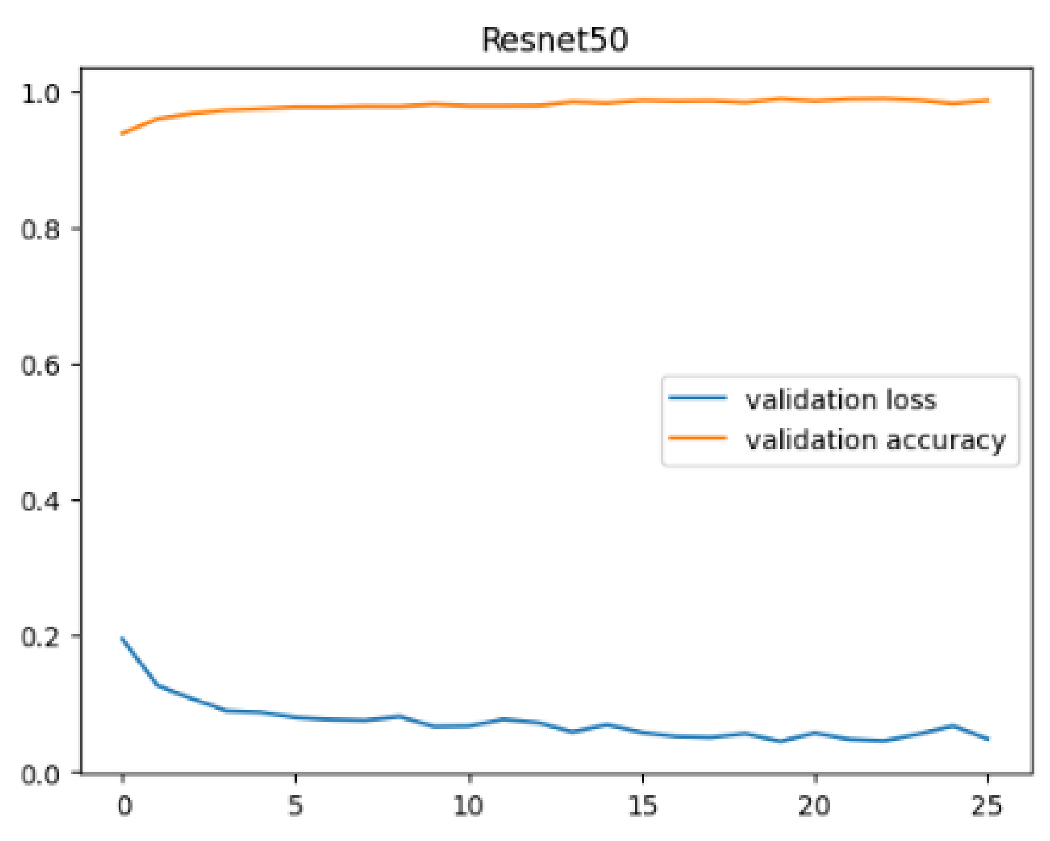
$F(x) + x$ is the original function.

Results









Resnet50 using transfer learning is a variant of Resnet50 that has been fine-tuned on the plant disease detection task, resulting in an even higher accuracy of 98.72%. Its precision and recall values are also very high, indicating that it is able to correctly classify both diseased and healthy plants with a high degree of accuracy.

S.N	Model	Testing Accuracy	Precision	Recall
1.	Inception V3	91.68%	91.95%	91.68%
2.	Vgg16	90.33%	91.49%	90.33%
3.	Resnet 50	95.68%	95.98%	95.68%
4.	Resnet50 (Fine-tuned)	98.72%	98.78%	98.72%

From the metrics table we observed that Resnet50 using Transfer Learning gives the best accuracy.esNet50 has ability to effectively learn and extract features from complex image datasets. By leveraging its deep learning capabilities, ResNet50 is able to identify patterns and textures that are indicative of specific diseases. The results of our experiments demonstrated that the proposed method outperformed the base paper model and other existing methods, exhibiting superior accuracy, precision, and recall .Hence it is good for plant disease detection.

Conclusion

In conclusion, early detection of plant diseases is essential for sustainable agriculture. Our study introduces a modified ResNet50 with transfer learning, outperforming existing methods in accuracy and efficiency. This approach, suitable for mobile applications, enhances accessibility for farmers, particularly in remote areas. Deployment of this model holds potential to significantly contribute to addressing global challenges in agriculture. Overall, our proposed method stands as a reliable and effective solution, offering promise for improved management of plant diseases and agricultural sustainability.

References

Adeleke, B. S., & Babalola, O. O. (2022). Roles of plant endosphere microbes in agriculture-a review. *Journal of Plant Growth Regulation*, *41*(4), 1411-1428.

Simonyan, K., & Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*.

Mohanty, S. P., Hughes, D. P., & Salathé, M. (2016). Using deep learning for image-based plant disease detection. *Frontiers in plant science*, *7*, 1419.

Dataset link- https://github.com/spMohanty/PlantVillage-Dataset/tree/master/raw/color