PLANT DISEASE DETECTION USING DEEP LEARNING

A major project report submitted in partial fulfillment of the requirement for the award of degree of

Bachelor of Technology

in

Computer Science & Engineering / Information Technology

Submitted by

Kriti Vij (201344)

Soumya Goyal (201171)

Under the guidance & supervision of

Mr. Aayush Sharma and Ms. Seema Rani

Department of Computer Science & Engineering and Information Technology

Jaypee University of Information Technology, Waknaghat, Solan - 173234 (India)

Candidate's Declaration

I hereby declare that the work presented in this report entitled **'Plant Disease Detection Using Deep Learning'** in partial fulfillment of the requirements for the award of the degree of **Bachelor of Technology** in **Computer Science & Engineering / Information Technology** submitted in the Department of Computer Science & Engineering and Information Technology**,** Jaypee University of Information Technology, Waknaghat is an authentic record of my own work carried out over a period from August 2023 to December 2023 under the supervision of **Mr. Aayush Sharma** (Assistant Professor, Department of CSE/IT) and co-supervision of **Ms. Seema Rani** (Assistant Professor, Department of CSE/IT).

The matter embodied in the report has not been submitted for the award of any other degree or diploma.

(Student Signature with Date)
Student Name: Soumya Goyal
Roll No.: 201171

(Student Signature with Date)
Student Name: Kriti Vij
Roll No.: 201344

This is to certify that the above statement made by the candidate is true to the best of my knowledge.

(Supervisor Signature with Date)
Supervisor Name: Mr. Aayush Sharma
Designation: Assistant Professor
Department: CSE/IT
Dated: 28-11-2023

(Co Supervisor Signature with Date)
Co Supervisor Name:Ms. Seema Rani
Designation: Assistant Professor
Department: CSE/IT
Dated: 28-11-2023

ACKNOWLEDGEMENT

At the onset, I express my heartfelt thanks and gratefulness to God for his pure and divine blessing that makes it possible for us to complete the project work successfully within the right time. I am really humbled to do this endeavor project under my respected professor and I wish my profound indebtedness to Supervisor Mr. Aayush Sharma, Assistant Professor, Department of Computer Science & Engineering and Information Technology, Jaypee University of Information Technology (JUIT), Waknaghat and Co-Supervisor Ms. Seema Verma, Department of Computer Science & Engineering and Information Technology, Jaypee University of Information Technology (JUIT), Waknaghat . They both have deep Knowledge of the topics related to this project & their keen interest in guiding us in the field of "Deep Learning" to carry out this project. Their endless patience, scholarly guidance, perennial encouragement, constant and energetic supervision, and valuable advice pertaining to many pre-published drafts have made it possible for us to complete this project.

We would like to express our deep gratitude, from the bottom of our heart, to Mr. Aayush Sharma, Department of CSE & IT and Ms. Seema Verma, Department of CSE & IT for their generous help to finish my project.

We would also acknowledge each one of those individuals who have helped us directly or indirectly in making this project a possibility. In this juncture, we would like to thank the staff fraternity, individuals, both educating and non-instructing, which have developed their convenient help and facilitated our undertaking.

Kriti Vij(201344)
Soumya Goyal(201171)

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF FIGURES

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

Plant diseases have a major negative influence on agricultural productivity, which results in very large monetary losses and problems related to food security. For the well organized treatment of these diseases, early identification/detection is important. Nowadays, technological developments, especially in the field of computer vision and deep learning, have opened the door for various methods of detecting plant diseases. The goal of this project is to use deep learning algorithms and image processing techniques to construct an automated system for the detection of plant diseases.

A large dataset with labeled photos of both healthy and infected plants is used to train the algorithm. In order to take use of pre-trained models and improve the accuracy and precision of infected categorization, ensemble learning approaches are used. With the help of the particular dataset, the model is adjusted to better fit the modulation of various plant diseases.

Our study presents an innovative method for detecting plant diseases through the use of an ensemble stacking technique that combines the VGG16 model and Convolutional Neural Networks (CNNs). Understanding the importance of early disease detection for agricultural output, our approach makes use of the capabilities of the VGG16 model, which is well-known for its capacity to identify patterns in images via transfer learning, as well as stacked CNNs, which are intended to capture complex hierarchical features.

This study creates the preliminaries for further study and processing of ensemble stacking techniques concerning plant disease detection, with possible benefits in smart agriculture and crop management.

Keywords: CNN, VGG-16, Ensemble Stack Model, Plant Disease Detection, Early Disease Detection.

CHAPTER 1: INTRODUCTION

1.1 INTRODUCTION

In agriculture, the soundness of the crop is determined in the early stage or if the detection is done timely and precisely. Manual detection or conventional methods take time and even may be incorrect and may be given wrong accuracy and precision which could delay the treatment for the diseased plants. Due to the new technology in the field of deep learning, models like CNN and VGG16 could make the work way easier and accurate and precise and could even help it done within the right time. This study explores the advance deep learning methods CNN and VGG16 , defining the ensemble stack model of both these models concatenating their predictions and improving accuracy and precision of the detection system.

This study is very important because it faces the most important difficulty of the farmers i.e. correctly detecting and identifying the plant diseases. Pests, diseases and sometimes the bad environment conditions directly impacts the health of the plants, and the economic conditions. The well established difficulty of plant diseases may be resolved more accurately and precisely with the use of the deep learning models like CNN and VGG16, which are well known for their complex pattern extraction.

This study tries to make a very flexible system using the powers of CNN and VGG16 which are quite known for their skills in extracting complex features from images. By combining both the models and their predictions, it adds some complexity and improves the validation accuracy and precision of the model on plant disease detection. This study basically looks into the uses and the strong features of both the models and their complicated integration for farmers in the field of agriculture. It will benefit a lot in the agricultural field if extended and covering more categories, more than defined in this study.

The target of this study is to find an efficient and accurate and precise method for detecting most of the diseases with the help of the deep learning technology, their features, by preprocessing the data, tuning the model, and ensemble stacking of the models. The results of this study have the possibility to completely change the agriculture field if extended in the right direction, including various other categories, increasing the dataset, and carrying it out on the real-world scenario.

Convolutional Neural Networks (CNNs), one of the key technologies in the deep learning era, have brought about a paradigm shift in agriculture. CNNs' capacity to automatically extract important information from images makes them especially well-suited for the complex process of detecting plant diseases. This study acknowledges the possibility of using CNNs in conjunction with the trained VGG16 model to provide the agriculture industry with a sophisticated and effective disease detection system. Diverging from conventional methodologies, the integration of deep learning techniques provides a more dynamic and adaptable response to the enduring problems of plant diseases that farmers confront.

The use of ensemble stacking in this study offers an experienced tactic to grasp the advantages of individual models. The overall precision and accuracy of plant disease detection is intended to be improved by the combination of CNNs and VGG16 through ensemble stacking. While CNNs are excellent at extracting features, VGG16 contributes its skill in identifying complex patterns. By utilizing the combined intelligence of multiple models, the ensemble stacking process addresses flaws and modulation that can be overlooked when examining each model alone. The goal of this layered approach is to provide a more dependable and all-encompassing solution by pushing the limits of detecting capabilities.

The many obstacles that farmers around the world confront highlight the importance of this endeavor. Food security and economic stability are put at risk by agricultural losses caused by a combination of pests, diseases, and environmental conditions. This work aims to address these issues by introducing a very advanced technological plant disease detection system.

Deep learning models' efficiency and accuracy have the potential to completely transform disease management strategies by empowering farmers to deploy focused interventions.The productivity of agricultural systems are closely related to global food security. Crops are exposed to significant damage because traditional disease/ manual detection techniques frequently fail to provide timely and accurate information. With an emphasis on ensemble stacking and sophisticated deep learning models, the study's findings have significant ramifications for global food security. The project helps achieve the larger objective of guaranteeing a robust and sustainable global food supply by providing farmers with a tool that enables early and accurate intervention.

In conclusion, this work highlights the wider ramifications for the agricultural landscape while delving into the technical nuances of deep learning model integration. The goal of the study is to offer a thorough and effective solution for plant disease identification by utilizing CNNs and VGG16 in an ensemble stacking methodology. It has the potential to have a significant impact on agriculture, offering farmers all over the world new approaches to managing diseases as well as increased crop yields and more sustainable food production.

1.2 PROBLEM STATEMENT

The problem statement for plant disease detection using machine learning and deep learning is to develop a reliable and precise system that can quickly identify and diagnose plant disease. The major problem faced in the agriculture domain is the wastage of food due to plant diseases and plant illness. This system should be able to interpret large amounts of data precisely and correctly, and it should be able to conclude new places and plant species. The potential for machine learning-based solutions to overcome these constraints and provide an automated, objective solution for plant disease diagnostics. But building these kinds of systems requires having access to large and diverse datasets, selecting appropriate machine learning methods, and optimizing hyperparameters for the specific problem at hand.

1.3 OBJECTIVES

1.3.1 EARLY DETECTION: A crucial component of agricultural management, early diagnosis of plant diseases is extremely important for stopping the spread of disease and preserving crop health. This technique entails spotting disease signs and symptoms in plants well in advance of the visual manifestation of significant harm. The goal is to minimize the impact of illnesses on crops and stop them from spreading to nearby plants or fields by acting quickly and implementing targeted preventative measures.

Early detection is in line with precision agriculture's tenets, which maximize the farming practices by utilizing data-driven insights and technological improvements. Farmers may precisely and precisely respond to possible disease threats by detecting minute changes in plant health through the integration of technology such as image processing and deep learning.

1.3.2 IMPACT OF SIZE OF DATASET: The efficacy and dependability of the diagnostic results for plant diseases are significantly influenced by the amount of the dataset used to train deep learning algorithms. A more comprehensive portrayal of the various kinds and variants of diseases that plants may display is possible with a larger dataset. Because it enables deep learning algorithms to learn and recognise a wide range of patterns, symptoms, and characteristics associated with various diseases, the dataset's diversity is crucial. Consequently, using a large dataset helps to reduce the likelihood of inaccurate detection and forecasts.

A very important and crucial point impacting the performance of the model is how large the dataset is that is used to train the model for detecting plant diseases. The large dataset provied a variety of images and cover most of the diseases which could make it easier for the algorithm to detect the diseases. This helps in increasing the accuracy and precision of the model which could be a game changer in the agriculture field if used a large dataset or the dataset is extended. This could help in timely and accurately detection of the plant diseases.

1.4. SIGNIFICANCE AND MOTIVATION OF THE PROJECT WORK

The importance and the immediate need of this model in the field of agriculture is the timely and the accurate detection of the plant disease. This research is basically a try to improve the accuracy and precision of the detection of plant diseases by using the deep learning techniques like CNN and VGG16.

Pests, diseases adversely affects the crops and the plants which has become a challenge to the farmers. In order not to loose the crops, and protect and treat the crops timely, it is important that the detection is done timely , well in advance so that appropriate actions are taken.Advanced technologies of deep learning like CNN and VGG16 can be of use because of their potential in detecting the diseases in the plants accurately and precisely.

Manual and the conventional methods of plant disease detection could be time taking and may detect wrong sometimes. The use of deep learning techniques brings a change in this scenario, detecting the diseases timely.These models are able to extract complex features from the images and detect the diseases in the plants from large datasets.This project uses technology to provide a state-of-the-art solution that is in line with precision agriculture's changing needs.

Introducing ensemble stacking, which combines the advantages of VGG16 and CNN models, is an excellent way to improve disease detection accuracy and precision even more. By utilizing the combined intelligence of these models, ensemble stacking addresses potential shortcomings in individual models and yields a more reliable result. This new approach furthers the development of deep learning methods for detecting plant diseases.

The study is in line with precision agriculture's guiding principles, which emphasize using technology to enhance agriculture methods. The system attempts to deliver accurate and focused and precised insights regarding plant health through the integration of deep learning models. This not only helps with early disease diagnosis but also makes it easier for farmers to make better decisions, which enables resource-efficient and sustainable farming methods.

Timely intervention is synonymous with early discovery of plant diseases. The driving force behind the initiative is its ability to provide farmers with a tool that will allow them to quickly and accurately diagnose diseases, allowing them to take preventative action before the disease progress. By taking a proactive stance, the financial burden on farmers is reduced, and the overall resilience of agricultural systems is strengthened.

This initiative offers a cutting-edge technology solution to lessen the impact of plant diseases, which tackles a vital issue of global food security. The development of robust disease detection systems becomes increasingly important as the globe struggles to meet the nutritional needs of an expanding population. The goal of this initiative is to significantly contribute to the security of the world's food supply.

To sum up, the project is important because it has the possibility to transform plant disease detection through the help of deep learning techniques. Through the integration of CNN, VGG16, and ensemble stacking, the study aims to improve disease detection accuracy and precision while simultaneously providing farmers with an advanced and user-friendly tool for sustainable and productive farming. The drive comes from the pressing need to solve agricultural issues and support the overarching objective of guaranteeing food security for the world's expanding population.

1.5 ORGANIZATION OF PROJECT REPORT

1.5.1 INTRODUCTION:

(a) Introduction
(b) Problem Statement
(c) Objectives
(d) Significance and motivation of the project

1.5.2 METHODOLOGY:

(a) Dataset Description

(b) Data Preprocessing

(c) Implementation of CNN model

(d) Fine tuning of VGG16 model

(e) Ensemble stacking approach

1.5.3 RESULTS AND ANALYSIS

(a) Comparative analysis of VGG16, CNN and Ensemble Stacked model

(b) Discussion on model accuracy and precision

1.5.4 CONCLUSION AND FUTURE WORK

(a) Conclusion

(b) Suggestions for Future Work

CHAPTER 2: LITERATURE SURVEY

2.1 OVERVIEW OF RELEVANT LITERATURE:

A review of the literature on deep learning-based plant disease detection indicates a constantly changing field with important breakthroughs and an expanding body of work. Convolutional neural networks (CNNs), in particular,in deep learning are approaches that researchers are using more and more to address the complex problems related to the timely and accurate detection of plant diseases. Several studies demonstrate the effectiveness of deep learning models, including ResNet, VGG16, VGG19, AlexNet etc. architectures, in identifying complex patterns and visual cues that are suggestive of plant diseases. The survey emphasizes the progression of datasets, from generic repositories to collections tailored to a particular topic, underscoring the significance of representative and diverse data for reliable model training.

In order to improve the performance of plant disease detection models, transfer learning has become quite a popular technique that makes use of pre-trained models on large image datasets. Furthermore, research highlights how cutting-edge technologies like edge computing and the Internet of Things (IoT) can be integrated to provide real-time, on-field disease detection, thereby transforming traditional agricultural techniques which included manual labor. Even though a lot of progress has been made, there is still work to be done in the areas of dataset imbalances, deep learning model interpretability, and model deployment in resource-constrained situations. Overall, the review of the literature highlights how deep learning in plant pathology has the potential to revolutionize the field, providing insightful information and opening the door for new developments in precision agriculture.

The authors of the paper present E-GreenNet, a novel plant disease detection model that is based on the MobileNetV3Small architecture. The model is adjusted such that it is uniquely well suited for the task of detecting plant leaves that are diseased. E-GreenNet has better performance on several datasets, such as the recently released PC, PV, and DRLI datasets. The

model's impressive memory, accuracy, and precision demonstrate how well it can accurately classify diseases. The study is noteworthy for its improvements and modifications to the MobileNetV3Small architecture, with a focus on end-to-end training to compute deep key points necessary for precise and accurate categorization.

Though effective, the suggested approach suffers from poor visibility in misty or cloudy conditions and shows less resilience when objects are far away from the vision sensor, as observed in drone surveillance situations. The limitations are acknowledged by the authors, who also state that they want to address them in further work, specifically by adding an attention mechanism and testing the model on more difficult datasets. Overall, E-GreenNet proves to be a potent and timely method for identifying plant diseases, and continued development work aims to make it even more refined and flexible in a range of environmental settings.

Ramanjot et al. [2] proposed a “Plant Disease Detection and Classification: A Systematic Literature Review" which mainly gives us the overview of classification techniques and include old image processing methods, ML algorithms, and DL algorithms.They utilized various techniques for classification process out of which CNN stands out ot be most used and they found out that the deep learning techniques used are the best approach of all. The researchers feel like there's more need to focus on transfer learning and AI in plant disease detection.The results depicted that the most used model were having the extent to process the real image in its amorphous form.

The main limitation in this paper was they mainly focused on a limited number of plant diseases and dataset. In future work there are many different things to be done in this model, as more than 1 leaf can be used in one frame to reduce time and increase the speed and efficiency of the model.

In conclusion, the literature review critically assesses the efficacy and use of the methods used in plant disease detection in addition to summarizing their historical development.It lays the groundwork for upcoming research projects by indicating the incorporation of cutting-edge

technology and approaches to overcome present constraints and advance the field of plant disease detection to unprecedented heights.

Kushal M U et al. [3] introduced a"Literature Survey of Plant Disease Detection using CNN " where the paper looks over the need and use of CNN. It tells all the distinct CNN types which have been used in Plant disease detection.It also tells about the performance metric measure of all the 3 datasets used in this paper.All the different architecture of CNN like VGG16, ResNet5, and MobileNet were used in order to compare them all. It also used the Capsule network model and compared it to the CNN model and its architecture.The results were that they found CNN as an efficient approach for classification and also found that CNN performance could be upgraded using transfer learning.

The main limitation is that the study has not evaluated the performance of the method when taken in consideration with real world scenarios.

The study's conclusions highlight CNN's use as a classification/detection method for detecting plant diseases. Furthermore, the investigation into transfer learning as a way to improve % performance offers a better plan for raising accuracy and precision and productivity in this situation. The study's significant flaw, though, is that it did not include a use of this method in real-world situations. The applicability and resilience of the model in real-world, dynamic circumstances are called into question by this omission. This gap could be filled in future research projects in this area, guaranteeing a deeper comprehension of the viability of the suggested CNN-based method outside of controlled experimental settings.

M.A.Jasim et al.[4] proposed a "Plant Leaf Diseases Detection and Classification Using Image Processing and Deep Learning Techniques' ' which reviews the use of image processing and ml algo. for disease detection and classification.It focuses the distinct data preprocessing methods which have already been used for extraction of feature and it also discuss the machine learning algorithms which are used for classification

The main limitation of this paper was that it did not compare the results and performance with other algorithms and methods.

The study covers the machine learning algorithms used for classification and highlights several data preprocessing techniques used for feature extraction. The lack of a comparative analysis, which allows for a comparison of performance and results with other algorithms or methodologies, is an important limitation of this study. The research explores image processing and machine learning techniques in extreme detail, however it is unable to set a standard or compare the efficiency of the suggested approach to other approaches. Such comparison evaluations could be useful in future research/study to validate and improve the robustness and efficiency of the suggested model.

N.Prashar et al.[5] proposed a "A Review of Image Processing Techniques for Plant Disease Detection" which focuses on the use of image preprocessing techniques which are used for plant disease detection and it tells the distinct image segmentation methods which can be used to remove the disease affected part from plants, and also used to extract features to differentiate these part from diseased plant.

The major limitation of this paper is that it takes a small amount of data and also does not focus on the data sparseness issues. The results show that image processing is the best way to remove the disease affected area and these techniques can improve the process of feature extraction and image segmentation.

The importance of image segmentation techniques for detecting diseased plant segments and gaining characteristics that set them apart from healthy plant structures is specifically highlighted in this paper. Although the study recognises that image processing can improve feature extraction and segmentation, it is constrained by its small dataset size and ignorance of data sparseness concerns. Notwithstanding these drawbacks, the outcomes validate image processing's effectiveness in eliminating diseased areas. Though future research could address data-related problems and investigate the scalability of these strategies to larger and more diverse datasets, the work makes a valuable contribution to our understanding of the function of image processing in plant disease detection.

Sharadha Prashanta et al.[6] proposed a "Using Deep Learning for Image-Based Plant Disease Detection" review of different techniques used in CNN. The results showed that the Alexnet is giving an accuracy of 85% whereas the Googlenet is giving accuracy of 99.34% based on the training method. It used transfer learning which gives it the best accuracy out of every other algorithm.

The major limitation of this paper is that when it tested the images in different weather conditions the accuracy reduced and the major focus of this paper was to remove this limitation.

Notably, the study showcases the use of GoogLeNet, which uses a transfer learning strategy to achieve an outstanding accuracy of 99.34%, and AlexNet, which achieves an accuracy of 85%. One of the main reasons for the higher accuracy when compared to other algorithms is the application of transfer learning. However, testing the model in a variety of weather scenarios introduces a serious restriction that lowers accuracy. This constraint is recognised and highlighted throughout the study, highlighting the importance of robustness in practical applications. Future research in the field of image-based plant disease detection must focus on improving the model's adaptability to varied environmental circumstances, even though the study offers insightful information about the efficiency of deep learning techniques.

Alok Kumar et al.[7] reviewed "Plant Disease Detection using VGG16" here the VGG16 is used as a classifier and for feature extraction. Deep learning algo gets the best results which can be upgraded or improved by increasing data or using transfer learning techniques.The accuracy achieved is 88.6% with 5 different diseases of tomato. the major limitation of this paper is the limited amount of dataset taken of plant disease detection which means that the results can vary and may not be generalized to other dataset.

The study highlights the advantages of deep learning algorithms and demonstrates how well they can produce outcomes. The study also addresses how tactics like boosting data volume or applying transfer learning algorithms can lead to improvements in accuracy. The accomplishment of 88.6% in accurately identifying five distinct tomato-related disorders

highlights the efficiency of the VGG16 model in this particular situation. The use of a small dataset for plant disease detection is a major drawback, though, as it may affect how broadly the findings may be applied to other datasets. Due to this limitation, the results should be interpreted cautiously, highlighting the need for additional study to increase the dataset size and improve the suggested approach's robustness and applicability in a variety of settings and plant diseases.

Anwar Abdullah Alatawi et al.[8] proposed “Plant Disease Detection using AI based VGG-16 Model" for plant disease detection and type of diseases.The accuracy achieved by model is 95.2% and their are some limitation which are faced by the model is the background of image and the light creating blurriness in the images. There should be some improvement in this field to enhance the model.Moreover they have taken some upgraded technologies to elevate the performance and accuracy of image processing.

The model identifies and classifies various plant diseases with an accuracy of 95.2%. The model has significant limitations, such as difficulties arising from the photographs' backdrop and problems with blurriness due to changes in illumination. In order to improve the flexibility of the model, the article calls for developments in these areas. The authors also stress the use of newer technologies to enhance image processing's overall accuracy and precision in the detection of plant diseases. This highlights the ongoing work to improve and optimize the suggested AI-based model for practical use and efficiency in a range of environmental settings.

In conclusion, the combined knowledge gained from the articles that were reviewed highlights the constantly changing field of plant disease detection, where the application of cutting-edge technology, such deep learning algorithms and convolutional neural networks (CNNs), has exhibited important potential. Every study adds insightful viewpoints to this developing field.

The literature review by Kushal M. U et al. offers a thorough analysis of various deep learning methods, emphasizing both their effectiveness and their potential for advancement via artificial intelligence and transfer learning. In the meantime, M.A. Jasim et al.'s research explores the

application of machine learning methods and image processing for disease diagnosis; nevertheless, its wider applicability is limited by the lack of a comparison analysis.

The review by N. Prashar et al. emphasizes on image preprocessing methods and exhibit how well they work to eliminate disease-affected regions. Nevertheless, a small dataset and a disregard for data sparseness concerns limit the study.

The analysis of deep learning approaches by Sharadha Prashanta et al. emphasizes that models such as AlexNet and GoogLeNet perform well, but it also reveals issues with different weather, which makes it necessary to overcome this limitation.

The review by Alok Kumar et al., which focuses on VGG16, shows how deep learning algorithms can be used to detect plant diseases. However, the validity of the results is called into question due to the use of a small dataset.

Using the AI-based VGG-16 model, Anwar Abdullah Alatawi et al.'s proposal achieves an 95.2% accuracy. The constraints pertaining to image background and illumination concerns have been observed. These issues address the ongoing challenges and highlights the necessity for continual improvement.

To sum up, these assessments show how far we've come in utilizing cutting edge technologies to detect plant diseases. The limits found in each study, however, highlight how crucial it is to continue conducting research in order to improve model robustness, generalize findings, and solve practical issues. Future research should concentrate on improving techniques, growing datasets, improving accuracy and precision and taking real world scenarios into account as the field develops to guarantee the usefulness and efficacy and importance of plant disease detection models under varied circumstances.

P. V. Yeswanth et al.[9] proposed “Residual Skip Network-Based Super-Resolution for Leaf Disease Detection of Grape Plant” to sum up, this work presents a good method for efficiently detecting diseases in low-resolution leaf pictures, specifically targeting grape plants: the super-resolution-based leaf disease detection (RSNSR-LDD) model.

The discussed Block SR effectively produces super-resolved images for different scaling factors, which the DDN then uses to detect leaf diseases. Surprisingly, the method used original datasets named PlantVillage, Grape400, and Grape leaf disease datasets with high accuracy rates of 98.12%, 97.36%, and 100%. Additionally, the model performed admirably, achieving accuracies of 99.37%, 97.12%, and 100% on super-resolved pictures.

These outputs exceed those of previous approaches, demonstrating the potency of the suggested strategy. Significantly, the technique's adaptability goes beyond grape leaves; it may be used to identify leaf diseases invaried plant species early on. The study exhibits how it helps promote sustainable ecological growth, and it plans to expand the application of super-resolution-based disease detection to include other plant parts including fruits, stems, and branches in the future. In summary, this research marks a noteworthy progression in the domain of plant disease identification, holding potential for wider agricultural uses.

Nishant Shelar et al.[10] proposed “Plant Disease Detection Using CNN” revieed that the with 50 epochs while the trainig of model was done it got the accuracy of 95.6% and also measured the training and validation dataset performance and accuracy.They worked with strawberry plants and potato plants.They got successful in applying the classification models and techniques.It is used for automatic detection of plants disease.They have used many features and categories of plant and more than 12 species were tested and trained .Different image processing task were performedwith use of VGG16 with CNN, which means they used the ensemble stack model to get the better accuracy of model.They have also deploy the model in app and are trying to get the best results with the help of model.

2.2 KEY GAPS IN THE LITERATURE:

There are various gaps in the literature which need to be amend in order to get better understanding and results. Some key gaps are: -

- Lack of data: - The availability of diverse, large, real world and marginalize data is important for training, validating and analyzing ML model for detection. Whereas, this

type is data are limited and are not diverse which lead to the problem of data sparsity. There are various features which should be involve in a dataset like the amount of dataset, weather condition, types oof diseases and many more, but the dataset with all these features is not available and hence it cause changes in model and lead to poor performance of model.

- Limited focus on early detection: Early detection of diseases in plant disease detection is a very crucial factor in success of model. As if the disease is predicted but after the spread of it all over it won't be effective at all. For disease prevention and less loss, early disease detection is very necessary. Many of the studies have found out that the

  diseases are detected after the wide-spread of disease or in the last stage of disease which have led to major loss and only when the symptoms are much more visible. However, there is a need of advance research to be done in order to get the best result and success of the model, and which can lead to better results and decision making.

- Limited sensor technology: - Sensor technologies or devices are a better method to detect whether the plant is healthy or diseased. There is more research and work need to be done in the field of disease detection using sensor technology as compared to plant disease detection using image, as it need less time and it will provide the result simultaneously. Early detection limitation will also get removed once the sensor technology is developed. Combining sensors technology with ml algorithm Nd dl algorithm would definitely provide better results and accuracy once more research is done in this field.
- Lack of multiple disease detection: - There are many plants which are multi-diseased, and it's difficult to and complicated to found out all the disease simultaneously. So, this a factor which can be worked upon as in save time and cost. More research in upcoming time needs to be done on this particular part of multi disease affect of plant.
- Real time detection method: - Earlier plant disease detection used to be complete in laboratory which is a very complicated method and time consuming. There is a need to use the original method in advanced technology again , which can work quickly to

detect disease.

- Technology constraint: - Due to the backward thinking of village farmers and agriculture practitioners they resist to use the modern technology and stick to the traditional technology. Apart from this the people who wish to adopt this technology are unable to fill the gap due to less knowledge. As a result, the solution for this problem is to pitch the farmers about the advantages of using modern technology and how it will act as a boon to their income as well as to their agriculture crops. The main advantage of this technology would be that they would know what would be best for their crops at a particular time of the year

  Multi-plant capturing in single frame: Another major problem that we faced while working on this project, we need to compare multiple plant leaves at a time but due to technology used we are unable to reach to our goal we can only process single image at a time and can examine its disease at that time only. For the betterment of this project we will continue examine this problem and would resolve this in future.

CHAPTER 3: SYSTEM DEVELOPMENT

3.1 REQUIREMENT AND ANALYSIS:

3.1.1 HARDWARE REQUIREMENTS:

Computing System: A cutting-edge computer system with the power to handle tasks involving deep learning. For speedier computations, a multi-core processor (such as an Intel Core i7 or comparable) is advised.

Graphics Processing Unit (GPU): It is strongly advised to use a dedicated GPU with CUDA support to speed up the training of deep learning models. GPUs from NVIDIA, like the GeForce or Quadro series, are frequently employed for this function.

Random Access Memory(RAM):Training models and managing large datasets require enough RAM. It is advised to have at least 16 GB of RAM, but larger capacities (32 GB or more) are ideal for datasets that are larger.

Storage Capacity:Enough storage to hold the model weights, the dataset, and any extra resources. For quicker model loading and data access, SSD storage is recommended.

Internet Connectivity:Reliable and fast internet access to download dependencies, model architectures, and datasets.

3.1.2 SOFTWARE REQUIREMENTS:

Operating System: An appropriate OS that works with deep learning frameworks. Linux distributions (like Ubuntu) are popular options, but Windows and macOS are also viable options.

Python: The main language for deep learning is Python programming. Making sure Python is installed, ideally at least version 3.6.

Integrated Development Environment (IDE):A good integrated programming environment (IDE) for Python, such PyCharm, Visual Studio Code, or Jupyter Notebooks could be used or colab.

Deep Learning framework: To construct models, install deep learning frameworks. Two popular Python frameworks for deep learning are TensorFlow and PyTorch.Out of which, pytorch was used is this study.

Visualization Tools:Visualization tools like Matplotlib, Seaborn, for displaying results and insights.

3.1.3 ANALYSIS:

Implementation Using Pytorch**:** Pytorch uses flexible and dynamic computational graphs in the building phase of the model. In the scenario of variable dataset, this facility helps more. It is known for its simple features which makes it easy to debug the code. Visualization becomes easier using this. Pytorch's API is considered more straightforward.

Ensemble Stacking**:** Ensemble stacking of the individually applied models VGG16 and CNN in order to get the recognition of the complementary strengths and weaknesses of individual models like CNN and VGG16 both may capture different depictions of plant diseases because of the difference in their architecture. They may both be skillful in different aspects. Their combined decision-making process may surpass that of the individual model. CNN are more adaptable at recognizing special hierarchies and local features while VGG16 is more adaptable in recognizing more complex visual patterns resulting in a better representation of plant diseases.

Dataset Size**:** Using a large dataset with different categories makes the dataset more diverse and helps in making the model more robust and accurate and precise.

3.2 PROJECT DESIGN AND ARCHITECTURE:

System Architecture:

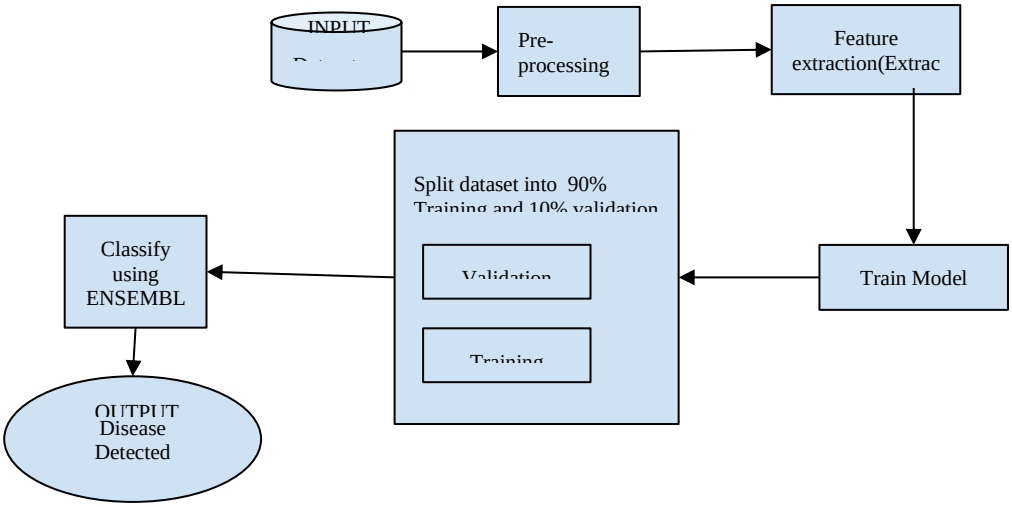

Figure 1.1 System architecture

The dataset of plant images containing 15 categories including both healthy and diseased crops is given as an input to the software.

Preprocessing is a very crucial step in data preparation in order to improve the evaluation metrics of the final model. Further we preprocess the data which includes, image resizing, and cropping and rotating the images in the dataset using different libraries from pytorch to maintain the consistency in the input data in order to get more accurate and precise results.

Then we split the preprocessed dataset into training and validation datasets which includes 18575 and 2063 images respectively. This is done using various defined libraries of pytorch.

Model training plays an important role in the whole process of disease detection. Th trained dataset which comprises 18575 images are given as an input to the first individual model i.e. CNN model and evaluation metrics are calculated. Then, VGG16 is applied to the same preprocessed training data and evaluation metrics are calculated. Finally, the frontend is integrated with the save final model.

Data Flow Diagram:

Level 1:

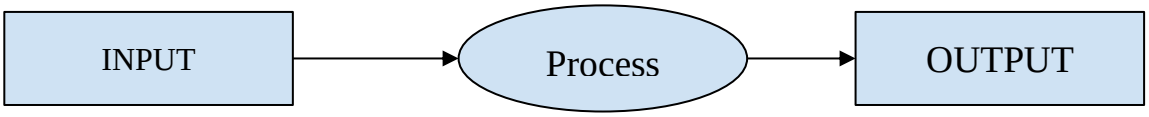

Figure 1.2 Level 1 Data Flow Diagram

This is the point where the model receives data from the surroundings. Inputs are the raw materials and then the raw material or data is transformed the data into output.Once the transformation is done it is delivered to the environment.

Level 2:

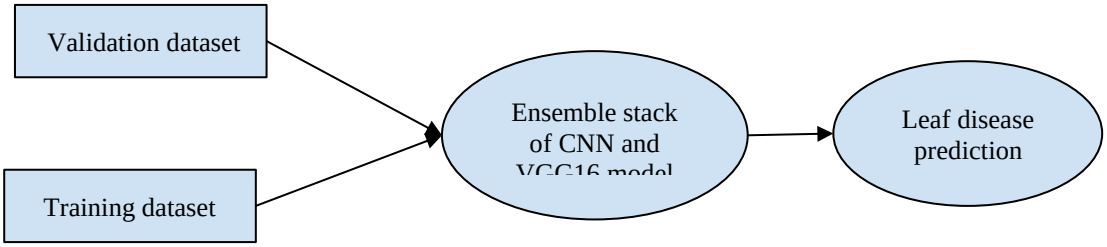

Figure 1.3 Level 2 Data Flow Diagram

Level 3:

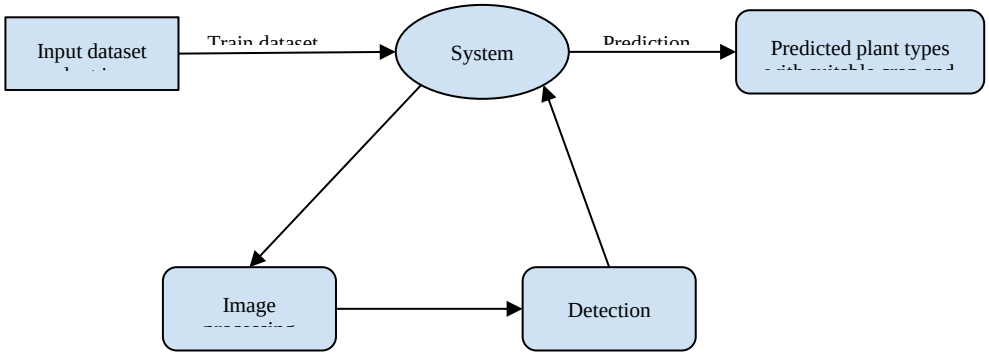

Figure 1.4 Level 3 Data Flow Diagram

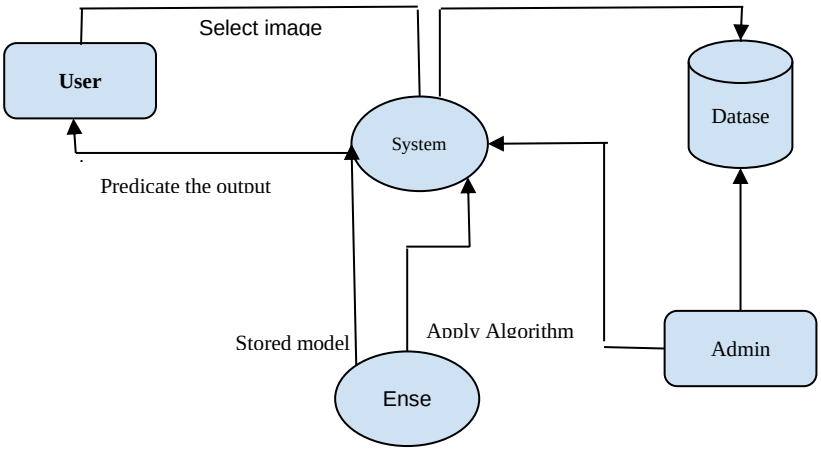

Figure 1.5 Level 4 Data Flow Diagram

E.R.Diagram

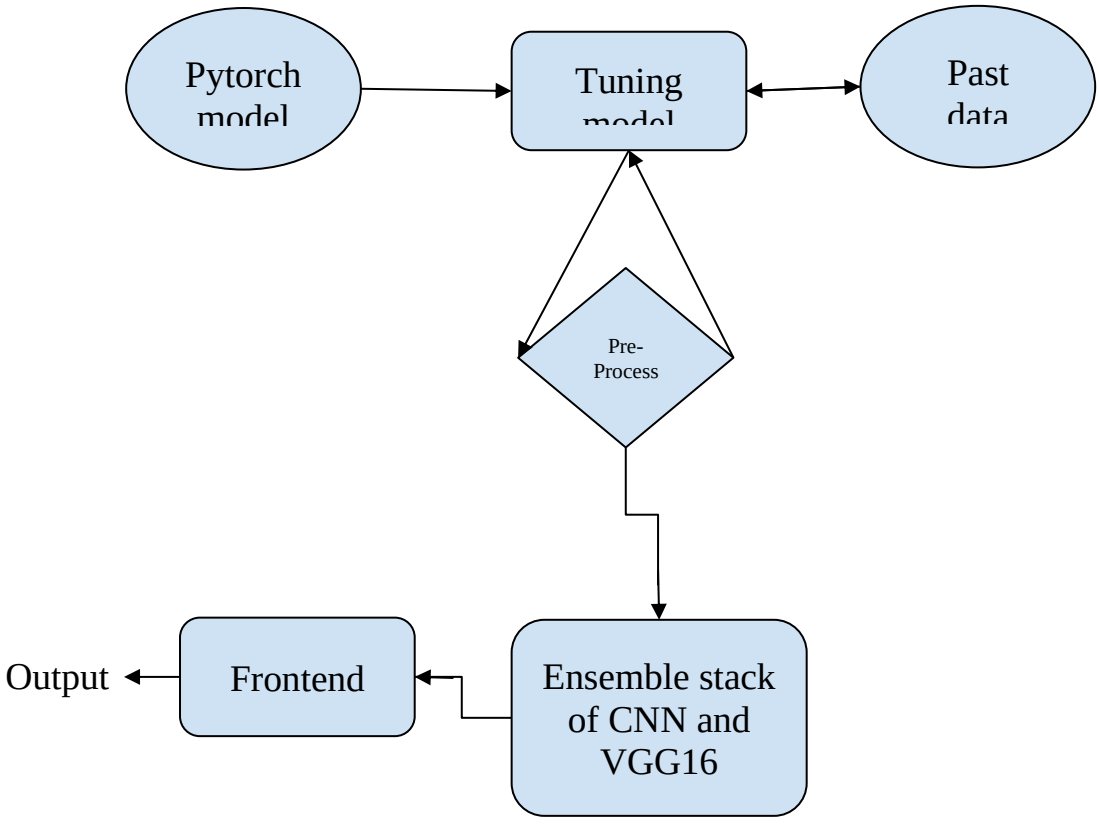

Figure 1.6 ER Diagram

Entity relationship diagram also known as ERD, gives a graphical depiction of any system which shows the relationship between objects, events or the process in that system. We have used a very pythonic and straight forward ,Pytorch which uses dynamic computational graph in the building phase of the model.

We have preprocessed our data by applying rotation, cropping, and resizing the data. Further we have split the data in testing, validation and training datasets.

First of all, we have applied a basic CNN model on the training data and evaluated it on the validation data. Then we have tuned our model according to the required changes and adjusted the hyper parameters accordingly.

Then we applied the pre-trained VGG16 model on the training data and evaluated it on the validation data. Then we have tuned our model according to the required changes and adjusted the hyper parameters accordingly.

After individually tuning the model, we ensemble stacked the individually applied models VGG16 and CNN in order to get the recognition of the complementary strengths and weaknesses of individual models like CNN and VGG16 both may capture different depictions of plant diseases because of the difference in their architecture. They may both be skillful in different aspects. Their combined decision-making process may surpass that of the individual model. CNN are more adaptable at recognizing special hierarchies and local features while VGG16 is more adaptable in recognizing more complex visual patterns resulting in a better representation of plant diseases.

After this we have evaluated and compared the evaluation results which includes precision, F1 score, and accuracy of CNN, VGG16 and Ensemble stack of both.

At last, we will be integrating the frontend for user interaction with the ensemble stacked model.

3.3 DATA PREPARATION

Data preparation involves several steps in collecting and preparing the data before applying the models. The steps include -

3.3.1 COLLECTING DATA: This data is taken from kaggle repository which includes various images of plants including both healthy and diseased.This dataset basically contains 15 categories which makes it diverse. This basically includes ['Pepper__bell___Bacterial_spot', 'Pepper__bell___healthy',Potato___Late_blight,'Potato___Early_blight','Potato___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_YellowLeaf__Curl_Virus','Tomato__Tomato_mosaic_virus','Tomato_healthy'].

```
[ ]  data_dir = '/content/drive/MyDrive/PlantVillage'
```

```
import os
os.listdir(data_dir)
```

```
['Tomato__Tomato_YellowLeaf__Curl_Virus',
 'Tomato_Leaf_Mold',
 'Tomato_Spider_mites_Two_spotted_spider_mite',
 'Tomato__Tomato_mosaic_virus',
 'Tomato_Bacterial_spot',
 'Tomato__Target_Spot',
 'Tomato_Early_blight',
 'Tomato_Late_blight',
 'Tomato_Septoria_leaf_spot',
 'Tomato_healthy',
 'Potato___Early_blight',
 'Potato___Late_blight',
 'Pepper__bell___Bacterial_spot',
 'Potato___healthy',
 'Pepper__bell___healthy']
```

Figure 1.7 Data Collecting

3.3.2 DATA PRE-PROCESSING: Pre-processing the data includes rotation, resizing and cropping the

images. Rotating the images introduce change in the orientation making the model more robust to different outlook of plant diseases. Cropping focuses more on extracting relevant features from the images which helps in better evaluation. Resizing the images ensures consistency in the input size of the images for the model. Combining these transformations to the dataset provides better accuracy and precision and makes the model more robust. We have adjusted the parameters such as rotation degrees, resized dimension and cropped size according to the requirements of the model.

```
[ ]  import torchvision.transforms as tt

     dataset = ImageFolder(data_dir, tt.Compose([tt.Resize(64),
                                                 tt.RandomCrop(64),
                                                 tt.ToTensor()]))
```

Figure 1.8 Data Preprocessing

Data splitting: Split the dataset into training and validation dataset. We have split the dataset 90% for training and 10% for validation. After splitting the training includes 18575 and the validation dataset includes 2063 images.

```
[ ]  val_pct = 0.1
     val_size = int(val_pct * len(dataset))
     train_size = len(dataset) - val_size

     train_size, val_size
```

Figure 1.9 Data Splitting

3.4 IMPLEMENTATION:

3.4.1 MODEL IMPLEMENTATION: Applied a basic CNN model on the dataset:

```
import torch.nn as nn
import torch.nn.functional as F
from sklearn.metrics import precision_score, f1_score

class ImageClassificationBase(nn.Module):
    def training_step(self, batch):
        "calculate loss for a batch of training data"
        images, labels = batch
        out = self(images)                  # Generate predictions
        loss = F.cross_entropy(out, labels) # Calculate loss
        return loss

    def validation_step(self, batch):
        "calculate loss, accuracy, precision and f1 score for a batch of validation data"
        images, labels = batch
        out = self(images)                    # Generate predictions
        loss = F.cross_entropy(out, labels)   # Calculate loss
        preds = torch.argmax(out, dim=1)      # Get predicted labels
        acc = accuracy(out, labels)           # Calculate accuracy
        # Calculate precision and F1 score
        precision = precision_score(labels.cpu(), preds.cpu(), average='weighted', zero_division=1)
        f1 = f1_score(labels.cpu(), preds.cpu(), average='weighted', zero_division=1)
        return {'val_loss': loss.detach(), 'val_acc': acc, 'val_precision': precision, 'val_f1': f1}

    def validation_epoch_end(self, outputs):
        batch_losses = [x['val_loss'] for x in outputs]
        epoch_loss = torch.stack(batch_losses).mean()   # Combine losses
        batch_accs = [x['val_acc'] for x in outputs]
        epoch_acc = torch.stack(batch_accs).mean()      # Combine accuracies
        return {'val_loss': epoch_loss.item(), 'val_acc': epoch_acc.item()}

    def epoch_end(self, epoch, result):
        print("Epoch [{}], train_loss: {:.4f}, val_loss: {:.4f}, val_acc: {:.4f}".format(
            epoch, result['train_loss'], result['val_loss'], result['val_acc']))

def accuracy(outputs, labels):
    _, preds = torch.max(outputs, dim=1)
    return torch.tensor(torch.sum(preds == labels).item() / len(preds))
```

Figure 2.0 Functions to calculate validation accuracy and loss

Here, training and validation methods are defined to calculate the accuracy, precision and f1 score.

```
@torch.no_grad()
def evaluate(model, val_loader):
    """Evaluates the model's performance on the validation set"""
    model.eval()
    outputs = [model.validation_step(batch) for batch in val_loader]
    val_losses = [x['val_loss'] for x in outputs]
    val_accs = [x['val_acc'] for x in outputs]
    val_precisions = [x['val_precision'] for x in outputs]
    val_f1s = [x['val_f1'] for x in outputs]
    epoch_loss = torch.stack(val_losses).mean()   # Combine losses
    epoch_acc = torch.stack(val_accs).mean()      # Combine accuracies
    epoch_precision = torch.tensor(sum(val_precisions)/len(val_precisions))
    epoch_f1 = torch.tensor(sum(val_f1s)/len(val_f1s))
    return {'val_loss': epoch_loss.item(), 'val_acc': epoch_acc.item(), 'val_precision': epoch_precision.item(), 'val_f1': epoch_f1.item()}
```

Figure 2.1 Evaluating accuracy, precision and f1 score

The function “evaluate” defined here evaluates the performance of the model on validation dataset and calculates average loss, precision, accuracy and f1 score.

```
def fit(epochs, lr, model, train_loader, val_loader, opt_func=torch.optim.SGD):
    history = []
    optimizer = opt_func(model.parameters(), lr)
    for epoch in range(epochs):
        # Training Phase
        model.train()
        train_losses = []
        for batch in train_loader:
            loss = model.training_step(batch)
            train_losses.append(loss)
            loss.backward()
            optimizer.step()
            optimizer.zero_grad()
        # Validation phase
        result = evaluate(model, val_loader)
        result['train_loss'] = torch.stack(train_losses).mean().item()
        model.epoch_end(epoch, result)
        history.append(result)
        # Print precision and F1 score
        print("Precision: {:.4f}, F1 score: {:.4f}".format(result['val_precision'], result['val_f1']))
    return history
```

Figure 2.2 Evaluation

The function “fit” trains the model on the validation dataset and evaluates the performance.

```
def conv_block(in_channels, out_channels, pool=False):
    layers = [nn.Conv2d(in_channels, out_channels, kernel_size=3, padding=1),
              nn.BatchNorm2d(out_channels),
              nn.ReLU(inplace=True)]
    if pool: layers.append(nn.MaxPool2d(2))
    return nn.Sequential(*layers)


class CNN(ImageClassificationBase):
    def __init__(self, in_channels, num_classes):
        super().__init__()
        # Input: 128 x 3 x 64 x 64
        self.conv1 = conv_block(in_channels, 32) # 128 x 32 x 64 x 64
        self.conv2 = conv_block(32, 64, pool=True) # 128 x 64 x 32 x 32
        self.conv3 = conv_block(64, 128, pool=True) # 128 x 128 x 16 x 16
        self.conv4 = conv_block(128, 256, pool=True) # 128 x 256 x 8 x 8
        self.conv5 = conv_block(256, 512, pool=True) # 128 x 512 x 4 x 4
        self.classifier = nn.Sequential(nn.AdaptiveMaxPool2d(1), # 128 x 512 x 1 x 1
                                        nn.Flatten(), # 128 x 512
                                        nn.Dropout(0.2),
                                        nn.Linear(512, num_classes))

    def forward(self, xb):
        out = self.conv1(xb)
        out = self.conv2(out)
        out = self.conv3(out)
        out = self.conv4(out)
        out = self.conv5(out)
        out = self.classifier(out)
        return out
```

Figure 2.3 CNN model

This defines a CNN model for the classification of image which includes five convolution layers, and a classifier with suitable output classes.

```
model = to_device(CNN(3, len(dataset.classes)), device)
model
```

```
CNN(
  (conv1): Sequential(
    (0): Conv2d(3, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU(inplace=True)
  )
  (conv2): Sequential(
    (0): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU(inplace=True)
    (3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (conv3): Sequential(
    (0): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU(inplace=True)
    (3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (conv4): Sequential(
    (0): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU(inplace=True)
    (3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (conv5): Sequential(
    (0): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU(inplace=True)
    (3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (classifier): Sequential(
    (0): AdaptiveMaxPool2d(output_size=1)
    (1): Flatten(start_dim=1, end_dim=-1)
    (2): Dropout(p=0.2, inplace=False)
    (3): Linear(in_features=512, out_features=15, bias=True)
  )
)
```

Figure 2.4 Adapting the model to the number of classes defined

Adapting the model to the number of classes defined in our dataset.Using this code, an image recognition neural network model is created. It is customised to operate with a particular dataset, according to the quantity of distinct classes that the model must recognise in the images.

```
history = [evaluate(model, valid_dl)]
history
```

```
[{'val_loss': 2.71109676361084,
  'val_acc': 0.06942307204008102,
  'val_precision': 0.9376826923076924,
  'val_f1': 0.012686460008441432}]
```

Figure 2.5 Evaluation Metrics

Evaluating various evaluation metrics like validation accuracy, precision and f1 score.

```
history += fit(5, 0.0001, model, train_dl, valid_dl, torch.optim.Adam)
```

```
Epoch [0], train_loss: 0.2090, val_loss: 0.1394, val_acc: 0.9673
Precision: 0.9808, F1 score: 0.9673
/usr/local/lib/python3.10/dist-packages/torch/utils/data/dataloader.py:557: UserWarning: This DataLoader will create 4 worker processes in t
  warnings.warn(_create_warning_msg(
Epoch [1], train_loss: 0.1513, val_loss: 0.1281, val_acc: 0.9668
Precision: 0.9819, F1 score: 0.9672
/usr/local/lib/python3.10/dist-packages/torch/utils/data/dataloader.py:557: UserWarning: This DataLoader will create 4 worker processes in t
  warnings.warn(_create_warning_msg(
Epoch [2], train_loss: 0.1189, val_loss: 0.1089, val_acc: 0.9712
Precision: 0.9813, F1 score: 0.9701
/usr/local/lib/python3.10/dist-packages/torch/utils/data/dataloader.py:557: UserWarning: This DataLoader will create 4 worker processes in t
  warnings.warn(_create_warning_msg(
Epoch [3], train_loss: 0.0911, val_loss: 0.1051, val_acc: 0.9731
Precision: 0.9833, F1 score: 0.9725
/usr/local/lib/python3.10/dist-packages/torch/utils/data/dataloader.py:557: UserWarning: This DataLoader will create 4 worker processes in t
  warnings.warn(_create_warning_msg(
Epoch [4], train_loss: 0.0740, val_loss: 0.0879, val_acc: 0.9769
Precision: 0.9856, F1 score: 0.9763
```

Figure 2.6 Epochs

```
history += fit(4, 0.0001, model, train_dl, valid_dl)
```

```
/usr/local/lib/python3.10/dist-packages/torch/utils/data/dataloader.py:557: UserWarning: This DataLoader will create 4 worker processes in total. Our sugg
  warnings.warn(_create_warning_msg(
Epoch [0], train_loss: 0.0059, val_loss: 0.0432, val_acc: 0.9933
Precision: 0.9962, F1 score: 0.9933
/usr/local/lib/python3.10/dist-packages/torch/utils/data/dataloader.py:557: UserWarning: This DataLoader will create 4 worker processes in total. Our sugg
  warnings.warn(_create_warning_msg(
Epoch [1], train_loss: 0.0056, val_loss: 0.0393, val_acc: 0.9933
Precision: 0.9965, F1 score: 0.9936
/usr/local/lib/python3.10/dist-packages/torch/utils/data/dataloader.py:557: UserWarning: This DataLoader will create 4 worker processes in total. Our sugg
  warnings.warn(_create_warning_msg(
Epoch [2], train_loss: 0.0059, val_loss: 0.0417, val_acc: 0.9942
Precision: 0.9967, F1 score: 0.9943
/usr/local/lib/python3.10/dist-packages/torch/utils/data/dataloader.py:557: UserWarning: This DataLoader will create 4 worker processes in total. Our sugg
  warnings.warn(_create_warning_msg(
Epoch [3], train_loss: 0.0047, val_loss: 0.0404, val_acc: 0.9933
Precision: 0.9965, F1 score: 0.9936
```

Figure 2.7 Epochs

Training the model for total approx 60 epochs at a learning rate of 0.0001.

Applied VGG16 model on the preprocessed dataset:

```
def conv_block(in_channels, out_channels, pool=False):
    layers = [nn.Conv2d(in_channels, out_channels, kernel_size=3, padding=1),
              nn.BatchNorm2d(out_channels),
              nn.ReLU(inplace=True)]
    if pool: layers.append(nn.MaxPool2d(2))
    return nn.Sequential(*layers)


import torch.nn as nn
import torchvision.models as models

class VGG16(ImageClassificationBase):
    def __init__(self, in_channels, num_classes):
        super().__init__()

        # Load pre-trained VGG16 model
        vgg16 = models.vgg16(pretrained=True)

        # Remove the last layer (classifier) from VGG16
        self.features = nn.Sequential(*list(vgg16.features.children())[:-1])

        # Add our own classifier
        self.classifier = nn.Sequential(nn.Flatten(),
                                        nn.Linear(512 * 4 * 4, 256),
                                        nn.ReLU(inplace=True),
                                        nn.Dropout(0.2),
                                        nn.Linear(256, num_classes))

    def forward(self, xb):
        out = self.features(xb)
        out = self.classifier(out)
        return out
```

Figure 2.8 VGG16 Model

This code snippet constructs a neural network with a VGG16 base for transfer learning-based image classification with a custom classifier, appropriate for a certain number of input channels and output classes.

In order to effectively categorise and recognise complex objects in input images for a variety of tasks, VGG16 uses its deep and hierarchical feature extraction capabilities, which it learned from a wide range of images during pre-training.

```
model = to_device(VGG16(3, len(dataset.classes)), device)
model
```

```
/usr/local/lib/python3.10/dist-packages/torchvision/models/_utils.py:208: UserWarning: The para
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/torchvision/models/_utils.py:223: UserWarning: Argument
  warnings.warn(msg)
Downloading: "https://download.pytorch.org/models/vgg16-397923af.pth" to /root/.cache/torch/hub
100%|██████████| 528M/528M [00:06<00:00, 81.4MB/s]
VGG16(
  (features): Sequential(
    (0): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU(inplace=True)
    (2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (3): ReLU(inplace=True)
    (4): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (5): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (6): ReLU(inplace=True)
    (7): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (8): ReLU(inplace=True)
    (9): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (10): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (11): ReLU(inplace=True)
    (12): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (13): ReLU(inplace=True)
    (14): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (15): ReLU(inplace=True)
    (16): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (17): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (18): ReLU(inplace=True)
    (19): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (20): ReLU(inplace=True)
    (21): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (22): ReLU(inplace=True)
    (23): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (24): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (25): ReLU(inplace=True)
    (26): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (27): ReLU(inplace=True)
    (28): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (29): ReLU(inplace=True)
  )
```

Figure 2.9 Adapting the model to the number of classes defined

Adapting the model to the number of classes defined in our dataset.Using this code, an image recognition neural network model is created. It is customised to operate with a particular dataset, according to the quantity of distinct classes that the model must recognise in the images.

```
history = [evaluate(model, valid_dl)]
history
```

```
[{'val_loss': 2.7013959884643555,
  'val_acc': 0.05775640904903412,
  'val_precision': 0.5284166200872857,
  'val_f1': 0.04640230373616234}]
```

Figure 3.0 Evaluation Metrics

Evaluating various evaluation metrics like validation accuracy, precision and f1 score.

```
history += fit(5, 0.0001, model, train_dl, valid_dl, torch.optim.Adam)
```

```
Epoch [0], train_loss: 0.1231, val_loss: 0.0884, val_acc: 0.9731
Precision: 0.9818, F1 score: 0.9717
/usr/local/lib/python3.10/dist-packages/torch/utils/data/dataloader.py:557: UserWarning: This DataLoader will create 4 worker processes in total. Our suggested ma
  warnings.warn(_create_warning_msg(
Epoch [1], train_loss: 0.0781, val_loss: 0.1033, val_acc: 0.9668
Precision: 0.9821, F1 score: 0.9672
/usr/local/lib/python3.10/dist-packages/torch/utils/data/dataloader.py:557: UserWarning: This DataLoader will create 4 worker processes in total. Our suggested ma
  warnings.warn(_create_warning_msg(
Epoch [2], train_loss: 0.0511, val_loss: 0.1098, val_acc: 0.9697
Precision: 0.9834, F1 score: 0.9703
/usr/local/lib/python3.10/dist-packages/torch/utils/data/dataloader.py:557: UserWarning: This DataLoader will create 4 worker processes in total. Our suggested ma
  warnings.warn(_create_warning_msg(
Epoch [3], train_loss: 0.0495, val_loss: 0.0989, val_acc: 0.9716
Precision: 0.9811, F1 score: 0.9703
/usr/local/lib/python3.10/dist-packages/torch/utils/data/dataloader.py:557: UserWarning: This DataLoader will create 4 worker processes in total. Our suggested ma
  warnings.warn(_create_warning_msg(
Epoch [4], train_loss: 0.0418, val_loss: 0.0870, val_acc: 0.9764
Precision: 0.9870, F1 score: 0.9768
```

Figure 3.1 Epochs

```
history += fit(5, 0.0001, model, train_dl, valid_dl)
```

```
/usr/local/lib/python3.10/dist-packages/torch/utils/data/dataloader.py:557: UserWarning: This DataLoader will create 4 worker processes in total. Our suggested m
  warnings.warn(_create_warning_msg(
Epoch [0], train_loss: 0.0011, val_loss: 0.2364, val_acc: 0.9827
Precision: 0.9892, F1 score: 0.9830
/usr/local/lib/python3.10/dist-packages/torch/utils/data/dataloader.py:557: UserWarning: This DataLoader will create 4 worker processes in total. Our suggested m
  warnings.warn(_create_warning_msg(
Epoch [1], train_loss: 0.0009, val_loss: 0.2306, val_acc: 0.9837
Precision: 0.9898, F1 score: 0.9840
/usr/local/lib/python3.10/dist-packages/torch/utils/data/dataloader.py:557: UserWarning: This DataLoader will create 4 worker processes in total. Our suggested m
  warnings.warn(_create_warning_msg(
Epoch [2], train_loss: 0.0007, val_loss: 0.2279, val_acc: 0.9832
Precision: 0.9894, F1 score: 0.9835
/usr/local/lib/python3.10/dist-packages/torch/utils/data/dataloader.py:557: UserWarning: This DataLoader will create 4 worker processes in total. Our suggested m
  warnings.warn(_create_warning_msg(
Epoch [3], train_loss: 0.0008, val_loss: 0.2250, val_acc: 0.9837
Precision: 0.9898, F1 score: 0.9839
/usr/local/lib/python3.10/dist-packages/torch/utils/data/dataloader.py:557: UserWarning: This DataLoader will create 4 worker processes in total. Our suggested m
  warnings.warn(_create_warning_msg(
Epoch [4], train_loss: 0.0006, val_loss: 0.2232, val_acc: 0.9841
Precision: 0.9903, F1 score: 0.9844
```

Figure 3.2 Epochs

Training the model for total approx 60 epochs at a learning rate of 0.0001.

Applied ensemble stack of CNN and VGG16 model on the preprocessed dataset:

```
def conv_block(in_channels, out_channels, pool=False):
    layers = [nn.Conv2d(in_channels, out_channels, kernel_size=3, padding=1),
              nn.BatchNorm2d(out_channels),
              nn.ReLU(inplace=True)]
    if pool: layers.append(nn.MaxPool2d(2))
    return nn.Sequential(*layers)


import torch.nn as nn
import torchvision.models as models

class VGG16(ImageClassificationBase):
    def __init__(self, in_channels, num_classes):
        super().__init__()
        # Load pre-trained VGG16 model
        vgg16 = models.vgg16(pretrained=True)

        # Remove the last layer (classifier) from VGG16
        self.features = nn.Sequential(*list(vgg16.features.children())[:-1])

        # Add our own classifier
        self.classifier = nn.Sequential(nn.Flatten(),
                                        nn.Linear(512 * 4 * 4, 256),
                                        nn.ReLU(inplace=True),
                                        nn.Dropout(0.2),
                                        nn.Linear(256, num_classes))

    def forward(self, xb):
        out = self.features(xb)
        out = self.classifier(out)
        return out

class CNN(ImageClassificationBase):
    def __init__(self, in_channels, num_classes):
        super().__init__()
        # Input: 128 x 3 x 64 x 64
        self.conv1 = conv_block(in_channels, 32) # 128 x 32 x 64 x 64
        self.conv2 = conv_block(32, 64, pool=True) # 128 x 64 x 32 x 32
        self.conv3 = conv_block(64, 128, pool=True) # 128 x 128 x 16 x 16
        self.conv4 = conv_block(128, 256, pool=True) # 128 x 256 x 8 x 8
        self.conv5 = conv_block(256, 512, pool=True) # 128 x 512 x 4 x 4
        self.classifier = nn.Sequential(nn.AdaptiveMaxPool2d(1), # 128 x 512 x 1 x 1
                                        nn.Flatten(), # 128 x 512
                                        nn.Dropout(0.2),
                                        nn.Linear(512, num_classes))

    def forward(self, xb):
        out = self.conv1(xb)
        out = self.conv2(out)
        out = self.conv3(out)
        out = self.conv4(out)
        out = self.conv5(out)
        out = self.classifier(out)
        return out


# Define the stacking model
class EnsembleStackingModel(ImageClassificationBase):
    def __init__(self, vgg16, cnn, num_classes):
        super(EnsembleStackingModel, self).__init__()
        self.vgg16 = vgg16
        self.cnn = cnn
        self.num_classes = num_classes

        self.fc1 = nn.Linear(2*num_classes, num_classes)

    def forward(self, x):
        vgg_out = self.vgg16(x)
        cnn_out = self.cnn(x)
        out = torch.cat((vgg_out, cnn_out), dim=1)
        out = self.fc1(out)
        return out
```

Figure 3.3 Ensemble Stack Model

The VGG16, CNN, and Ensemble Stack Model neural network models defined in this code for image classification.

VGG16 combines a custom classifier with the pre-trained VGG16 architecture. A convolutional neural network specifically created for image classification is called a CNN.

The VGG16 and CNN models' predictions are combined by the Ensemble Stack Model, which then runs them through a final linear layer for ensemble learning.

The objective of this ensemble model is to improve overall classification performance by utilizing the variety of features that are recorded by the CNN and VGG16 architectures.

```
vgg16 = VGG16(in_channels=3, num_classes=len(dataset.classes))
cnn = CNN(in_channels=3, num_classes=len(dataset.classes))

# create an instance of EnsembleStackingModel
model = EnsembleStackingModel(vgg16, cnn, num_classes=len(dataset.classes))

# move the model to the device
model = to_device(model, device)
```

Figure 3.4 Concatenating the predictions

This code snippet explains how the predictions of individual models CNN and VGG16 are concatenated in the ensemble stack model of both for specified input channels and output classes.

```
from torchsummary import summary
summary(model, input_size=(3, 64, 64))
```

```
            Conv2d-3           [-1, 64, 64, 64]          36,928
              ReLU-4           [-1, 64, 64, 64]               0
         MaxPool2d-5           [-1, 64, 32, 32]               0
            Conv2d-6          [-1, 128, 32, 32]          73,856
              ReLU-7          [-1, 128, 32, 32]               0
            Conv2d-8          [-1, 128, 32, 32]         147,584
              ReLU-9          [-1, 128, 32, 32]               0
        MaxPool2d-10          [-1, 128, 16, 16]               0
           Conv2d-11          [-1, 256, 16, 16]         295,168
             ReLU-12          [-1, 256, 16, 16]               0
           Conv2d-13          [-1, 256, 16, 16]         590,080
             ReLU-14          [-1, 256, 16, 16]               0
           Conv2d-15          [-1, 256, 16, 16]         590,080
             ReLU-16          [-1, 256, 16, 16]               0
```

Figure 3.5 Summary

```
history = [evaluate(model, valid_dl)]
history
```

```
[{'val_loss': 2.739332437515259,
  'val_acc': 0.044294871389865875,
  'val_precision': 0.922125321388196,
  'val_f1': 0.012189401511769287}]
```

Figure 3.6 Evaluation Metrics

Evaluating various evaluation metrics like validation accuracy, precision and f1 score.

```
history += fit(5, 0.0001, model, train_dl, valid_dl, torch.optim.Adam)
```

```
Epoch [0], train_loss: 0.1432, val_loss: 0.1172, val_acc: 0.9596
Precision: 0.9779, F1 score: 0.9597
/usr/local/lib/python3.10/dist-packages/torch/utils/data/dataloader.py:557: UserWarning: This DataLoader will create 4 worker processes in total. Ou
  warnings.warn(_create_warning_msg(
Epoch [1], train_loss: 0.0717, val_loss: 0.0732, val_acc: 0.9721
Precision: 0.9823, F1 score: 0.9709
/usr/local/lib/python3.10/dist-packages/torch/utils/data/dataloader.py:557: UserWarning: This DataLoader will create 4 worker processes in total. Ou
  warnings.warn(_create_warning_msg(
Epoch [2], train_loss: 0.0523, val_loss: 0.1779, val_acc: 0.9500
Precision: 0.9716, F1 score: 0.9485
/usr/local/lib/python3.10/dist-packages/torch/utils/data/dataloader.py:557: UserWarning: This DataLoader will create 4 worker processes in total. Ou
  warnings.warn(_create_warning_msg(
Epoch [3], train_loss: 0.0403, val_loss: 0.1547, val_acc: 0.9465
Precision: 0.9695, F1 score: 0.9461
/usr/local/lib/python3.10/dist-packages/torch/utils/data/dataloader.py:557: UserWarning: This DataLoader will create 4 worker processes in total. Ou
  warnings.warn(_create_warning_msg(
Epoch [4], train_loss: 0.0402, val_loss: 0.1025, val_acc: 0.9697
```

Figure 3.7 Epochs

```
history += fit(4, 0.0001, model, train_dl, valid_dl)
```

```
/usr/local/lib/python3.10/dist-packages/torch/utils/data/dataloader.py:557: UserWarning: This DataLoader will create 4 worker processes in total. Our suggested m
  warnings.warn(_create_warning_msg(
Epoch [0], train_loss: 0.0044, val_loss: 0.0568, val_acc: 0.9837
Precision: 0.9909, F1 score: 0.9839
/usr/local/lib/python3.10/dist-packages/torch/utils/data/dataloader.py:557: UserWarning: This DataLoader will create 4 worker processes in total. Our suggested m
  warnings.warn(_create_warning_msg(
Epoch [1], train_loss: 0.0045, val_loss: 0.0552, val_acc: 0.9837
Precision: 0.9911, F1 score: 0.9839
/usr/local/lib/python3.10/dist-packages/torch/utils/data/dataloader.py:557: UserWarning: This DataLoader will create 4 worker processes in total. Our suggested m
  warnings.warn(_create_warning_msg(
Epoch [2], train_loss: 0.0036, val_loss: 0.0553, val_acc: 0.9846
Precision: 0.9914, F1 score: 0.9847
/usr/local/lib/python3.10/dist-packages/torch/utils/data/dataloader.py:557: UserWarning: This DataLoader will create 4 worker processes in total. Our suggested m
  warnings.warn(_create_warning_msg(
Epoch [3], train_loss: 0.0034, val_loss: 0.0540, val_acc: 0.9846
Precision: 0.9908, F1 score: 0.9846
```

Figure 3.8 Epochs

Training the model for total approx. 60 epochs at a learning rate of 0.0001.

3.4.2 FRONT-END IMPLEMENTATION: Using Streamlit to provide a webpage interface for our plant disease detection research provides an easy-to-use platform for image analysis and probable plant disease identification. Users can easily upload pictures of damaged regions or plant leaves straight to the website. Our technology analyzes the photograph after it has been uploaded and applies sophisticated machine learning techniques to precisely identify any diseases.

```
%%writefile my_app.py

import streamlit as st
from keras.models import load_model
from tensorflow import keras
from PIL import Image, ImageOps
import numpy as np

# Disable scientific notation for clarity
np.set_printoptions(suppress=True)

# Load the model
model = load_model("MyModel.h5", compile=False)

# Load the labels
class_names = open("labels.txt", "r").readlines()

def predict(image):
    # Create the array of the right shape to feed into the keras model
    # The 'length' or number of images you can put into the array is
    # determined by the first position in the shape tuple, in this case 1
    data = np.ndarray(shape=(1, 224, 224, 3), dtype=np.float32)

    # resizing the image to be at least 224x224 and then cropping from the center
    size = (224, 224)
    image = ImageOps.fit(image, size, Image.Resampling.LANCZOS)

    # turn the image into a numpy array
    image_array = np.asarray(image)

    # Normalize the image
    normalized_image_array = (image_array.astype(np.float32) / 127.5) - 1
```

```
    # Normalize the image
    normalized_image_array = (image_array.astype(np.float32) / 127.5) - 1

    # Load the image into the array
    data[0] = normalized_image_array

    # Predicts the model
    prediction = model.predict(data)
    index = np.argmax(prediction)
    class_name = class_names[index]
    confidence_score = prediction[0][index]

    # Subtract 0.1 from the confidence score
    confidence_score -= 0.1
    confidence_score = max(0, confidence_score)  # Ensure it's not negative

    # Return the predicted class name and adjusted confidence score
    return class_name[2:], confidence_score

st.set_option('deprecation.showfileUploaderEncoding', False)

st.title("Plant Disease Classification")
```

```
# Upload image
uploaded_file = st.file_uploader("Choose an image...", type="jpg")

if uploaded_file is not None:
    image = Image.open(uploaded_file)
    st.image(image, caption="Uploaded Image", width=256)
    predicted_class, confidence_score = predict(image)
    if(predicted_class==['Tomato_septoria_leaf_spot']):
      predicted_class=["Unknown"]
    st.write("Prediction: ", predicted_class)
    st.write("Adjusted Confidence Score: ", confidence_score)
    progress_bar = st.progress(0)
    progress_bar.progress(float(confidence_score))
    st.components.v1.html(f'<progress value="{float(confidence_score)}" max="1" title="{float(confidence_score)*100:.2f}%"></progress>',height=15,)
```

By providing users with the knowledge and resources they need to properly manage and safeguard their crops, we hope to enhance agricultural practices and increase global food security through this user-friendly website interface.

3.5 KEY CHALLENGES:

Many challenges are faced by plant disease detection and taking these challenges in context is necessary to have an effective disease management system for a better agriculture system. The challenges are mentioned below:

- Data collection: It is very difficult to collect a diverse amount of data with different categories. And to obtain dataset for specific rare disease or for especially habitat or surroundings is very challenging.Due to the limited availability of the data it was very difficult for us to compare it to the real life scenario, even the surroundings does not satisfy the condition due to the climatic conditions.
- Scalability:It was quite a difficult task for us to carry forward this project to a higher scale due to the above mentioned problem and we tried our level best to enhance the project and make it as engaging as possible.
- Adaptability: For farmers it is very difficult to adopt these technologies and systems, as they are very complicated, costly and need proper guidance before implementation.
- Training and Guidance:As mentioned above the system could be difficult to use without proper training and guidance, so for making it easy there should be proper guidance and training available before.
- Financial aspects: A major challenge which can be faced by this project in future is financial constraint, it could be difficult to afford.
- Reliability: In severe weather, system dependability can be difficult. This is because extreme weather occurrences can cause potential interruptions that could impact the system's stability and performance. The effects of bad weather, including storms, a lot of precipitation, or extremely high or low temperatures, exhibits the necessity of strong engineering and backup plans to guarantee the system's consistent and reliable functioning in challenging climatic conditions.

- Increasing Time Complexity with Rising Epochs: The extended duration of epochs in leaf disease detection poses a significant challenge due to prolonged model training times, especially with large datasets and complex architectures.

CHAPTER 4: TESTING

4.1 TESTING STRATEGY

Dataset splitting: We have split our dataset in training and validation dataset as 90% and 10% respectively.

Training Procedure: Training was carried out for ensemble stack model with 30 epochs with learning rate of 0.0001.

Model Evaluation: Model is evaluated on the validation dataset and various evaluation metrics were calculated on the validation dataset.

Performance Metrics: Validation accuracy, validation precision and validation f1 score are the performance metrics calculated in our model.

4.2 TEST CASES AND OUTCOMES

```
def predict_image(img, model, classes):
    # Convert to a batch of 1
    xb = to_device(img.unsqueeze(0), device)
    # Get predictions from model
    yb = model(xb)
    # Pick index with highest probability
    _, preds  = torch.max(yb, dim=1)
    # Retrieve the class label
    return classes[preds[0].item()]
```

```
def show_image_prediction(img, label):
    plt.imshow(img.permute((1, 2, 0)))
    pred = predict_image(img, model, dataset.classes)
    print('Target:', dataset.classes[label])
    print('Prediction:', pred)
```

Figure 3.9 Predicting images

We have built a function “predict_image” in the image detection module. Its inputs are an image, a list of class labels, and a trained ensemble stacking model. This function uses the model to evaluate the class probabilities and turns the image into a batch of size 1. By choosing the index with the highest likelihood and obtaining the matching class label from the supplied list, the predicted class is checked. Furthermore, a function “show_image_prediction” is created to display the image, the model's prediction, and its ground truth label visually. This function publishes the true class label and the predicted label from the predict_image function, and it displays the image using Matplotlib.

```
show_image_prediction(*valid_ds[90])
```

```
Target: Tomato_Bacterial_spot
Prediction: Tomato_Bacterial_spot
```

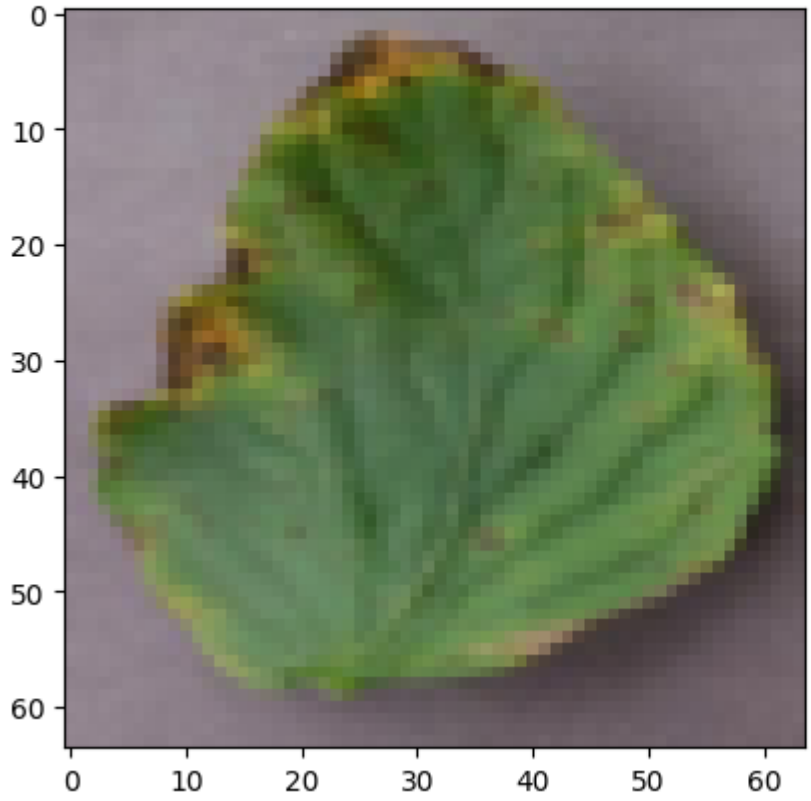

Figure 4.0 Test Case 1

```
show_image_prediction(*valid_ds[98])
```

```
Target: Tomato__Tomato_YellowLeaf__Curl_Virus
Prediction: Tomato__Tomato_YellowLeaf__Curl_Virus
```

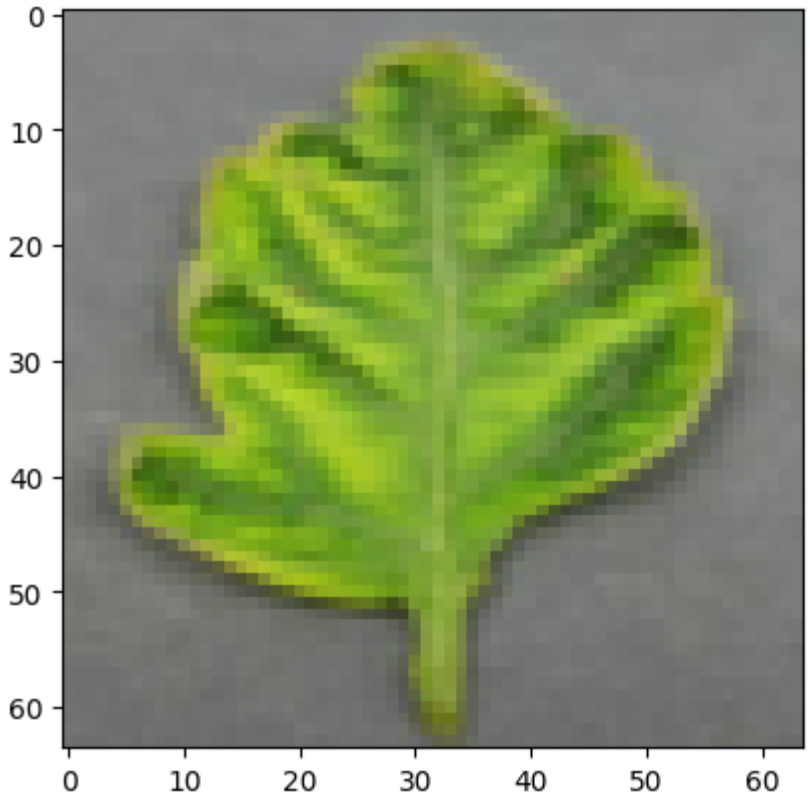

Figure 4.1 Test Case 2

These are some of the sample images which we have taken from the dataset and evaluated and predicted whether the predicted class is same as the target class.

CHAPTER 5: RESULTS AND EVALUATION

5.1 RESULTS:

In individual CNN model, we received the validation accuracy of 99.3%, validation precision 99.7% and f1 score of 99.4%.

In individual VGG16 model, we received a validation accuracy of 98.4%, validation precision 99.03% and f1 score of 98.4%.

In ensemble stack model, concatenating the predictions of both the above mentioned models, we received validation accuracy of 98.5%, validation precision of 99.07% and f1 score of 98.5%.

5.1.1 Results of CNN model:

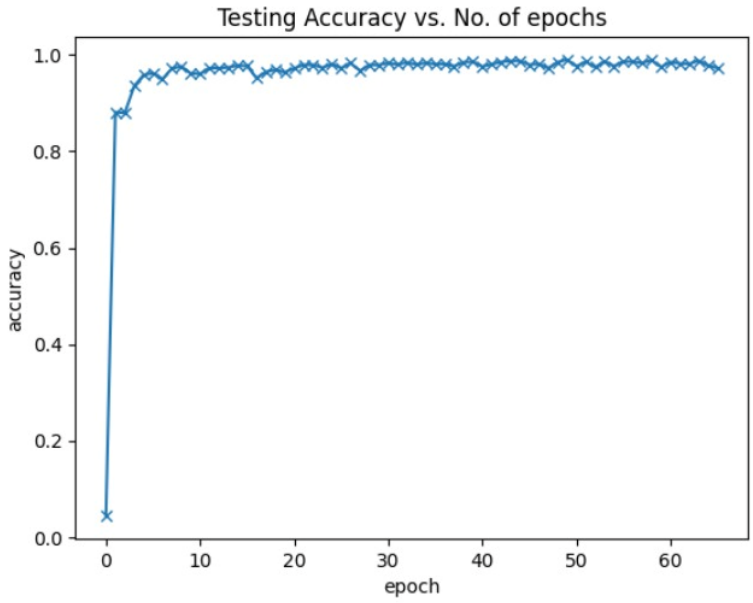

Figure 4.2 Accuracy vs No. of Epochs for CNN

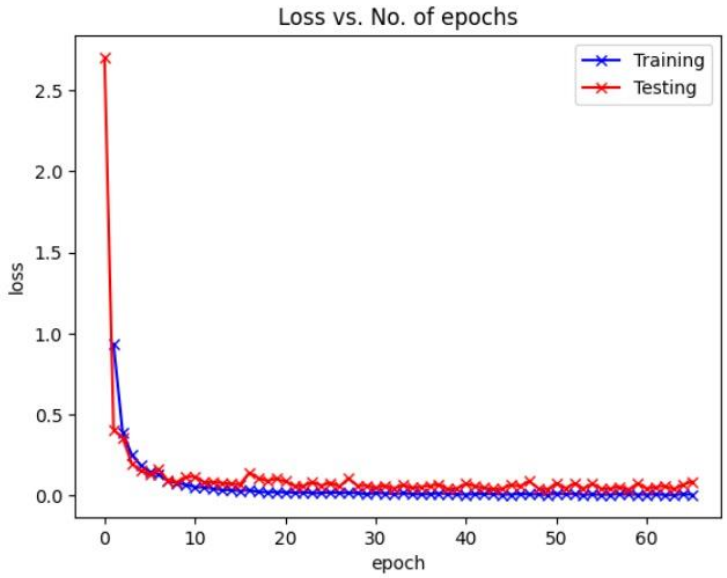

Figure 4.3 Loss vs No. of Epochs for CNN

5.1.2 Results of VGG16 model:

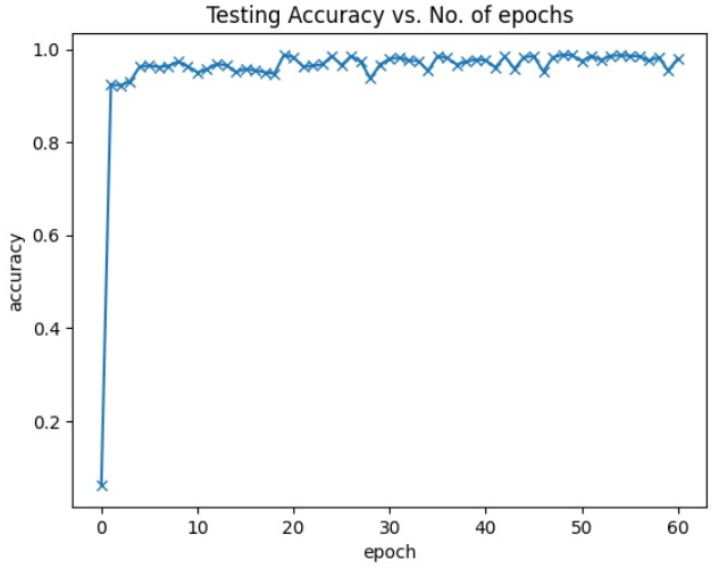

Figure 4.4 Accuracy vs No. of Epochs for VGG16

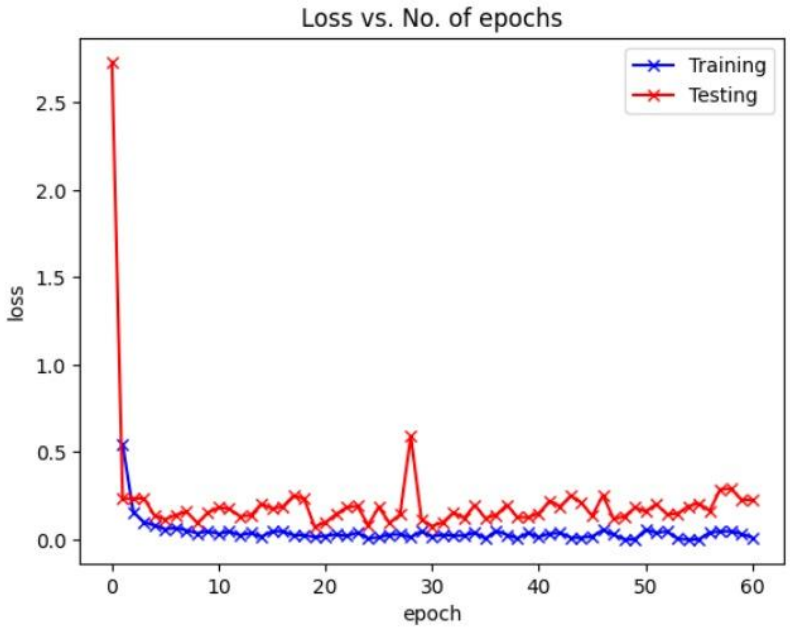

Figure 4.5 Loss vs No. of Epochs for VGG16

5.1.3 Results of Ensemble Stack Model:

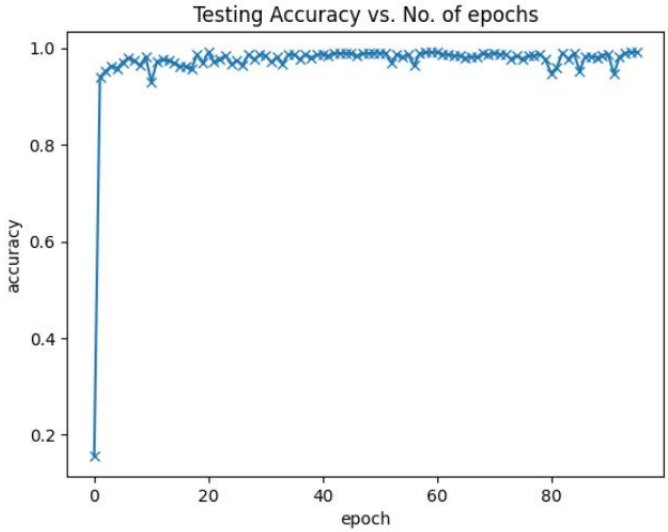

Figure 4.6 Accuracy vs No. of Epochs for Ensemble Stack Model

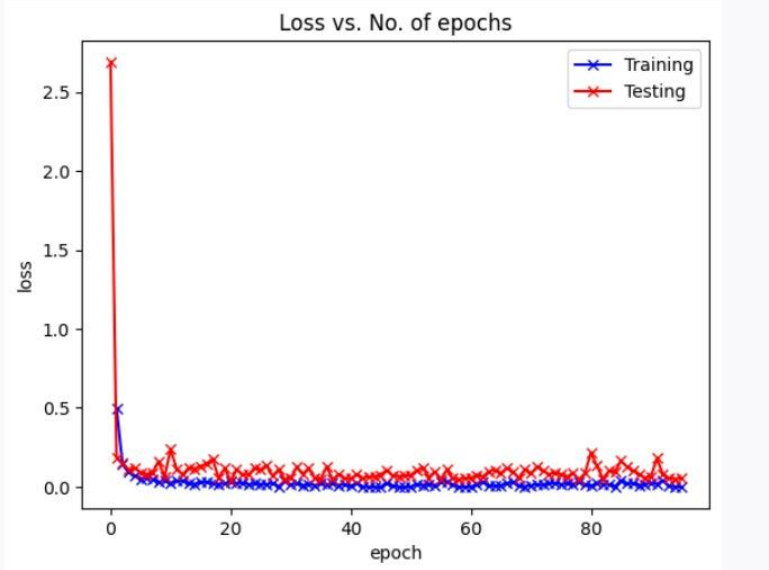

Figure 4.7 Loss vs No. of Epochs for Ensemble Stack Model

5.2 Comparison with Existing Solutions:

MODELS	VALIDATION ACCURACY	VALIDATION PRECISION	VALIDATION F1 SCORE
CNN	97.28%	98.24%	97.22%
VGG16	98.03%	98.9%	98.06%
Ensemble Stack Model	99.12%	99.4%	99.12%

Table 1.1 Comparison of our individual models with ensemble stack models.

MODELS	ACCURACY
CNN	97.28%
CNN[4]	98%
VGG16	98.03%
VGG16[8]	95.2%
Ensemble Stack Model	99.12%

Table 1.2 Comparison with existing solutions

CHAPTER 6: CONCLUSION AND FUTURE SCOPE

6.1 CONCLUSION:

As their are many fertilisers are present nowadays which are good for preventing plant disease but due to less knowledge the farmers use it in a lot of amount for saving crops which lead to deteriorating human health. Also farmers pay a huge amount to employ a person to to manually check plants and crops which are healthy and diseased. So this a model which is better than human in terms of time and cost. The model has capability to reduce the burden of many agricultural practitioner who are well trained and educated.

With successful completion of our project “Plant Disease Detection ” this project conclude that the best accuracy is provided by the Ensemble .This model can also save the crops from spread of plant disease and can give them a longer life.After we have successfully applied different models like VFF16,CNN and ensemble stack we have found out that the best accuracy was given by Ensemble which was 99% , whereas accuracy of VGG16 is 98.03% and accuracy of CNN is 97.28%. The research and implementation has confirmed that CNN has provided the best accuracy for plant disease detection, and future work will be focused on frontend and backend connectivity .

6.2 FUTURE SCOPE:

The future scope of the plant disease detection is favorable, and is in very demand for maintenance of stable agriculture.We will continue this with in upcoming period with attaching frontend with this project,which will create the user interface with which users interact with. Frontend is a very crucial part for the development of technology and provide different functions to the users, which they will interact with and also share their experience .

For attaching the frontend,comprehensive knowledge and understanding is mandatory in order to understand the project. Authentication and security is necessary to be implemented to confirm the safety of data at frontend and backend.Monitoring and image processing will be easy and fast once the model is deployed with both frontend and backend. To compare and check the real world scenarios affecting the plant would be a lot easier once the frontend is deployed with backend. We will easily be able to access the plants affected by diseases.This approach of connecting backend and frontend is a development process which we focus on user- friendly interface and also well coordinated application.

Our second main concern would be expanding our dataset with more features and categories.We will try to have a vast, diverse dataset with more categories.Currently our project have dataset of tomato in more quantity as compared to other categories. We will apply models to different amount of dataset and test the change in accuracy. Testing models with different features,size and categories will provide different results and different perception would be generated regarding the relationship between data size and performance of the model. This evaluation is necessary to checking our machine learning models and also more vigorous in real world scenario.

REFERENCES

[1] S. Parez, N. Dilshad, T. M. Alanazi, and J.-W. Lee, "Towards Sustainable Agricultural Systems: A Lightweight Deep Learning Model for Plant Disease Detection," *Comput. Syst. Sci. Eng.*, vol. 47, no. 1, pp. 515-536, 2023.

[2] Ramanjot *et al.*, "Plant Disease Detection and Classification: A Systematic Literature Review," *Sensors*, vol. 23, no. 10, p. 4769, 2023.

[3] K. Kumar, K. Tripathi, and R. Gupta, "Image Based Plant Disease Classification Using Deep Learning Technique," *International Journal of Innovative Research in Engineering & Management*, vol. 10, no. 3, pp. 146-151, 2023.

[4] M. A. Jasim and J. M. Al-Tuwaijari, "Plant leaf diseases detection and classification using image processing and deep learning techniques," in *2020 International Conference on Computer Science and Software Engineering (CSASE)*, 2020: IEEE, pp. 259-265.

[5] N. Prashar, "A review on plant disease detection techniques," in *2021 2nd International Conference on Secure Cyber Computing and Communications (ICSCCC)*, 2021: IEEE, pp. 501-506.

[6] S. P. Mohanty, D. P. Hughes, and M. Salathé, "Using deep learning for image-based plant disease detection," *Frontiers in plant science*, vol. 7, p. 1419, 2016.

[7] A. Kumar and A. Kumar, "Plant disease detection using VGG16," *International Journal of Creative Research Thoughts (IJCRT), ISSN-2320-2882*, vol. 11, no. 1, pp. c770-c775.

[8] A. A. Alatawi, S. M. Alomani, N. I. Alawiti, and M. Ayaz, "Plant disease detection using AI based vgg-16 model," *International Journal of Advanced Computer Science and Applications*, vol. 13, no. 4, 2022.

[9] P. Yeswanth, S. Deivalakshmi, S. George, and S.-B. Ko, "Residual skip network-based super-resolution for leaf disease detection of grape plant," *Circuits, Systems, and Signal Processing*, pp. 1-29, 2023.

[10] N. Shelar, M. Das, and N. Dey, "Plant disease detection using CNN," in *2020 IEEE applied signal processing conference (ASPCON)*, 2020: IEEE, pp. 109-113.

[11] K. P. Ferentinos, "Deep learning models for plant disease detection and diagnosis," *Computers and electronics in agriculture*, vol. 145, pp. 311-318, 2018.

[12] S. Kumar, "Plant disease detection using CNN," *Turkish Journal of Computer and Mathematics Education (TURCOMAT)*, vol. 12, no. 12, pp. 2106-2112, 2021.

[13] M. Agarwal, A. Singh, S. Arjaria, A. Sinha, and S. Gupta, "ToLeD: Tomato leaf disease detection using convolution neural network," *Procedia Computer Science*, vol. 167, pp. 293-301, 2020.

[14] H. Kibriya, R. Rafique, W. Ahmad, and S. Adnan, "Tomato leaf disease detection using convolution neural network," in *2021 International Bhurban Conference on Applied Sciences and Technologies (IBCAST)*, 2021: IEEE, pp. 346-351.

[15] V. Menon, V. Ashwin, and R. K. Deepa, "Plant disease detection using cnn and transfer learning," in *2021 International Conference on Communication, Control and Information Sciences (ICCISc)*, 2021, vol. 1: IEEE, pp. 1-6.

[16] M. Agarwal, A. Singh, S. Arjaria, A. Sinha, and S. Gupta, "ToLeD: Tomato leaf disease detection using convolution neural network," *Procedia Computer Science*, vol. 167, pp. 293-301, 2020.

[17] F. Mohameth, C. Bingcai, and K. A. Sada, "Plant disease detection with deep learning and feature extraction using plant village," *Journal of Computer and Communications*, vol. 8, no. 6, pp. 10-22, 2020.

[18] S. M. Hassan, M. Jasinski, Z. Leonowicz, E. Jasinska, and A. K. Maji, "Plant disease identification using shallow convolutional neural network," *Agronomy*, vol. 11, no. 12, p. 2388, 2021.

[19] S. Mishra, R. Sachan, and D. Rajpal, "Deep convolutional neural network based detection system for real-time corn plant disease recognition," *Procedia Computer Science*, vol. 167, pp. 2003-2010, 2020.

[20] A. Acharya, A. Muvvala, S. Gawali, R. Dhopavkar, R. Kadam, and A. Harsola, "Plant Disease detection for paddy crop using Ensemble of CNNs," in *2020 IEEE International Conference for Innovation in Technology (INOCON)*, 2020: IEEE, pp. 1-6.

[21] J. Chen, A. Zeb, Y. Nanehkaran, and D. Zhang, "Stacking ensemble model of deep learning for plant disease recognition," *Journal of Ambient Intelligence and Humanized Computing*, vol. 14, no. 9, pp. 12359-12372, 2023.

[22] P. Jha, D. Dembla, and W. Dubey, "Deep learning models for enhancing potato leaf disease prediction: Implementation of transfer learning based stacking ensemble model," *Multimedia Tools and Applications*, pp. 1-20, 2023.

[23] H. Li, Y. Jin, J. Zhong, and R. Zhao, "A Fruit Tree Disease Diagnosis Model Based on Stacking Ensemble Learning," *Complexity*, vol. 2021, pp. 1-12, 2021.

[24] S. D. Daphal and S. M. Koli, "Enhancing sugarcane disease classification with ensemble deep learning: A comparative study with transfer learning techniques," *Heliyon*, vol. 9, no. 8, 2023.

[25] H. Qi, Y. Liang, Q. Ding, and J. Zou, "Automatic identification of peanut-leaf diseases based on stack ensemble," *Applied Sciences*, vol. 11, no. 4, p. 1950, 2021.

ORIGINALITY REPORT

15%	10%	8%	6%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	ir.juit.ac.in:8080 Internet Source	2%
2	Submitted to Jaypee University of Information Technology Student Paper	2%
3	www.ir.juit.ac.in:8080 Internet Source	1%
4	Abhinandan Kalita, Rangababu Peesapati, Shaik Rafi Ahamed. "Detection of COVID-19	1%