PLANT DISEASE CLASSIFICATION USING DEEP LEARNING

A PROJECT REPORT

Submitted by

ALLA ROHITH REDDY [RA1911003010361]
CHITRALEKHA CHEDURUPALLI [RA1911003010387]

Under the guidance of

Dr. K.R. JANSI

Assistant Professor, Department of Computing Technologies

in partial fulfillment for the award of the degree of

BACHELOR OF TECHNOLOGY
in
COMPUTER SCIENCE & ENGINEERING

DEPARTMENT OF COMPUTING TECHNOLOGIES

COLLEGE OF ENGINEERING AND TECHNOLOGY

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

KATTANKULATHUR – 603203

MAY 2023

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

KATTANKULATHUR – 603203

BONAFIDE CERTIFICATE

Certified that 18CSP109L project report titled "**PLANT DISEASE CLASSIFICATION USING DEEP LEARNING**" is the bonafide work of **ALLA ROHITH REDDY [RA1911003010361], CHITRALEKHA CHEDURUPALLI [RA1911003010387]** who carried out the project work under my supervision. Certified further, that to the best of my knowledge the work reported here in does not form part of any other thesis or dissertation on the basis of which a degree or award was conferred on an earlier occasion for this or any other candidate.

Dr. K.R. JANSI
SUPERVISOR
Assistant Professor
Department of Computing Technologies

Dr. K.R. JANSI
PANEL HEAD
Assistant Professor
Department of Computing Technologies

Dr. M. PUSHPALATHA
HEAD OF THE DEPARTMENT
Department of Computing Technologies

INTERNAL EXAMINER

EXTERNAL EXAMINER

Department of Computing Technologies
SRM Institute of Science and Technology
Own Work Declaration Form

Degree/ Course	: B.Tech in Computer Science and Engineering
Student Names	: Alla Rohith Reddy, Chitralekha Chedurupalli
Registration Number	: RA1911003010361,RA1911003010387
Title of Work	: Plant Disease Classification Using Deep Learning

We hereby certify that this assessment compiles with the University's Rules and Regulations relating to Academic misconduct and plagiarism, as listed in the University Website, Regulations, and the Education Committee guidelines.
We confirm that all the work contained in this assessment is our own except where indicated, and that we have met the following conditions:

- Clearly references / listed all sources as appropriate.
- Referenced and put in inverted commas all quoted text (from books, web, etc.)
- Given the sources of all pictures, data etc. that are not my own
- Not made any use of the report(s) or essay(s) of any other student(s) either past or present
- Acknowledged in appropriate places any help that I have received from others (e.g., fellow students, technicians, statisticians, external sources)
- Compiled with any other plagiarism criteria specified in the Course handbook / University website.

We understand that any false claim for this work will be penalized in accordance with the University policies and regulations.

DECLARATION:
We are aware of and understand the University's policy on Academic misconduct and plagiarism and we certify that this assessment is our own work, except where indicated by referring, and that we have followed the good academic practices noted above. **Student 1 Signature:** **Student 2 Signature:** **Date:**
If you are working in a group, please write your registration numbers and sign with the date for every student in your group.

ACKNOWLEDGEMENT

We express our humble gratitude to **Dr. C. Muthamizhchelvan**, Vice-Chancellor, SRM Institute of Science and Technology, for the facilities extended for the project work and his continued support.

We extend our sincere thanks to Dean-CET, SRM Institute of Science and Technology, **Dr. T. V. Gopal**, for his invaluable support.

We wish to thank **Dr. Revathi Venkataraman**, Professor and Chairperson, School of Computing, SRM Institute of Science and Technology, for her support throughout the project work.

We are incredibly grateful to our Head of the Department, **Dr. M. Pushpalatha,** Professor, Department of Computing Technologies, SRM Institute of Science and Technology, for her suggestions and encouragement at all the stages of the project work.

We want to convey our thanks to our Project Coordinators, **Dr. M. Kanchana**, **Dr. G. Usha, Dr. R. Yamini, and Dr. K. Geetha**, Panel Head, **Dr. K.R. Jansi**, Assistant Professor and Panel members, **Dr.D. Malathi**, Professor, **Dr. J. Dorathi Jayaseeli,** Associate Professor and **Dr. S. Nithiya,** Assistant Professor, Department of Computing Technologies, SRM Institute of Science and Technology, for their inputs during the project reviews and support.

We register our immeasurable thanks to our Faculty Advisor, **Dr. M. Kanchana**, Associate Professor and **Dr. G. Usha,** Associate Professor, Department of Computing Technologies, SRM Institute of Science and Technology, for leading and helping us to complete our course.

Our inexpressible respect and thanks to our guide, **Dr. K.R. Jansi**, Assistant Professor, Department of Computing Technologies, SRM Institute of Science and Technology, for providing us with an opportunity to pursue our project under her mentorship. She provided us with the freedom and support to explore the research topics of our interest. Her passion for solving problems and making a difference in the world has always been inspiring.

We sincerely thank all the staff and students of Computing Technologies Department, School of Computing, S.R.M Institute of Science and Technology, for their help during our project. Finally, we would like to thank our parents, family members, and friends for their unconditional love, constant support, and encouragement.

Alla Rohith Reddy [RA1911003010361]

Chitralekha Chedurupalli [RA1911003010387]

ABSTRACT

The role of agriculture in human life is crucial. In recent times almost 60% of the population seems to have some involvement in agriculture. Recent years have seen a dramatic increase in agricultural difficulties due to a combination of factors, including sudden climatic shifts and a deficiency of crop immunity. Farmers are not motivated to increase their agricultural productivity day by day because of the agricultural environment. Identification and treatment of the diseases caused in plants have become extremely difficult due to the rapid expansion in the diversity of conditions and very little knowledge of growers. Many approaches have been put forth in recent years to address this significant deep learning field. The conventional approach has the capability to identify illnesses in various crops. Innovative techniques were utilized to boost the accuracy and precision of the outcomes. Convolutional neural networks have demonstrated to be very efficient for a variety of computer vision tasks, including the detection of plant diseases and image classification. The texture and visual similarities of the leaves contribute to the classification of complaint type. This project suggests a methodology for categorizing plant disease symptoms using convolutional neural networks. To increase the sample size, data augmentation is applied to two plant disease datasets, which are then used to train and evaluate the model. The proposed model utilizes multiple convolution and pooling layers in a CNN architecture and is trained on the Plant Village dataset. Following training with 80% of the sample, the model is properly tested using 10% sample to validate the results, using the remaining 10% sample from the Plant Village dataset containing images of both healthy and diseased plants. The proposed CNN model achieved a testing accuracy of 96.5% in detecting and recognizing the plant variety and the type of diseases the plant was infected with.

TABLE OF CONTENTS

LIST OF FIGURES

ABBREVIATIONS

CNN	Convolutional Neural Network
PD	Plant Disease
DL	Deep Learning
ML	Machine Learning
CV	Cross Validation
ANN	Artificial Neural Network
SVM	Support Vector Machine
KNN	K-Nearest Neighbor
API	Application Programming Interface
DAE	Denoising auto encoder

CHAPTER 1

INTRODUCTION

1.1 Overview

It is no secret that a large portion of India's population relies on agriculture and agricultural produce for a living. A decrease in both the quality and quantity of agricultural goods has been linked to the plant leaf disease. That is why it's crucial for farmers to be able to recognize symptoms of disease early on. Early detection of diseases can prevent widespread crop failure.

Farmers in underdeveloped nations like India have a hard time identifying plant leaf diseases just by looking at them, and consulting a scientist or specialist is prohibitively expensive. The latest developments in computer vision developed Advancements in deep learning have cleared the way to identify and treat plant illnesses by utilizing camera-captured images as the foundation. for differentiating between various plant diseases. The present research provides a useful technique for fast identifying and recognizing the various type of diseases and illnesses Across a diverse range of plant species available in our agriculture crops. The system's architecture was designed to detect and recognize several plant types, including apple, corn, grapes, potatoes, tomatoes, sugarcane, and others. Furthermore, the technology has the capability to identify various plant diseases.

Investigators such as research, scientists and many other people Successfully trained.
deep learning models(algorithms) to detect, recognize diseases in plants by using around 6000 combined image datasets containing both healthy and diseased plant leaves. The trained system/model or algorithm was able to register up to 100% efficiency in picking out and detecting the plant variety and sorting the diseases by which the plant is sick with, and the trained model has obtained an accuracy rate of 96.5%.

1.2 Plant Disease Overview

Agriculture has been an inseparable part of the Indian economy since ancient times. In fact, it is the backbone of the country's economy, providing employment to nearly half of its population. India's agricultural prowess is evident from the fact that it is the largest producer of pulses, rice, wheat, and various spices and their products, globally. However, the growth of the farmers' economic status is closely tied to the quality of their produce, which is heavily influenced by plant growth and yield. Consequently, in the agricultural domain, the ability to identify and treat plant diseases assumes great significance. The reason being, plants are highly vulnerable to diseases that impede their growth and development, ultimately affecting the farmer's ecosystem in unimaginable ways.

Advancements in technology have transformed the agricultural landscape, and automated disease detection techniques have emerged as a game-changer in identifying plant diseases at an early stage. Unlike manual diagnosis, which can take a considerable amount of time and effort, automated methods can swiftly and accurately detect the presence of disease in various parts of the plant, including the leaves. It's a well-known fact that diagnosing plant diseases using leaf photographs is a time-consuming process, requiring expert intervention. Hence, the development of computational methods has become the need of the hour to automate the integration of technology and agriculture has enabled farmers to keep a close eye on their crops, detect any disease outbreak, and take necessary measures promptly. A growing number of diseases have made it difficult to keep up with, and farmers lack the knowledge necessary to properly diagnose and treat them. Similarities in leaf structure and appearance help researchers pinpoint the illness.

There is a detrimental effect on agricultural output due to the prevalence of plant infection. Increasing food insecurity is possible if plant diseases are not identified in a timely order. To effectively manage and make decisions on agricultural production, early detection of plant disease is essential. Specifying plant disease has become increasingly important in recent years. Applying modern techniques like the recognition rates and accuracy of the findings had both improved thanks to the use of machine learning(ml) and deep learning algorithms(dl) algorithms like Convolutional neural network, (CNN) etc. have all been studied for their potential in plant disease identification and diagnosis.

What causes Plant diseases?

Crop disease has been traditionally classified into two categories: abiotic and biotic. Abiotic diseases are non-infectious, often caused by unfavorable environmental conditions such as extreme temperatures, lack or excess of moisture, or harmful air contaminants emitted by nearby chemical or metallurgical plants. Moreover, unhealthy physicochemical soil composition is another factor that can cause abiotic diseases, often resulting from poor-quality herbicide treatment of fields. The prevalence of such diseases highlights the importance of sustainable agriculture not just for environmental protection but also for business profitability. Even light regimes can have an adverse impact on crops, particularly on those grown in greenhouses. Biotic diseases, on the other hand, are infectious, and their causes can range from toxins released by some embryophytes to harmful fungi. The early detection and management of crop diseases are critical to minimizing losses and ensuring optimum yields. Therefore, sustainable agricultural practices that promote a healthy plant-soil-environment nexus are crucial to maintain the health and productivity of our crops. Plant diseases can cause significant economic losses and have serious impacts on food security, making them an important area of research and management.

Common plant diseases include:

1. **Fungal diseases** - These include diseases such as powdery mildew, downy mildew, and rust, which can cause damage to leave, stems, and fruits.
2. **Bacterial diseases** - These include diseases such as bacterial wilt and fire blight, which can cause wilting, leaf spots, and necrosis.
3. **Viral diseases** - These include diseases such as mosaic viruses and yellowing viruses, which can cause stunted growth and deformation of leaves and fruit.
4. **Nematode diseases** - These include diseases such as root-knot nematodes, which can cause root damage and reduced yield.
5. **Abiotic diseases** - These include diseases caused by factors such as nutrient deficiencies, water stress, and temperature extremes, which can cause stunted growth and reduced yield.

When do farmers know about plant diseases?

Plant diseases can have a range of observable symptoms, including discernible changes in color, function, or shape as the plant responds to infection. For example, the fungus Verticillium wilt is characterized by leaf wilting, while common bacterial blight causes brown necrotic lesions surrounded by a vivid yellow halo on bean plants. These symptoms are not caused by the pathogen itself but rather by the plant's response to the infection. Fungal infections often result in local or general necrosis, abnormal growth. Bacterial infections can cause symptoms such as ooze, water-soaked lesions, or bacterial streaming in water from a cut stem. Viral infections can cause malformations, necrosis, wilting, dwarfism, and discoloration, including yellowing and vein clearing. Identifying the signs and symptoms of crop diseases is essential for prompt diagnosis and effective management, which can help minimize losses and maintain crop health.

Plant disease treatment

In recent years, the progress of artificial intelligence has revolutionized various industries, including agriculture. With the development of deep learning algorithms, it has become possible to automatically identify plant diseases from raw images. This has enabled farmers to detect diseases in their crops at an early stage, which can help prevent extensive crop damage and losses.

One of the most popular deep learning models used in disease identification is the Convolutional Neural Network (CNN). CNN is a multi-layer feed-forward neural network that can extract features from photos automatically. By training the neural network to extract features from photos of diseased plants, it can learn to identify various types of diseases accurately.

Farmers' livelihoods are closely tied to the quality of their crops, and plant diseases can significantly impact their economic growth. The use of automated disease detection techniques, such as those based on deep learning, is advantageous in detecting diseases at an early stage. Plant diseases manifest in various parts of the plant, such as the leaves, and manual diagnosis using leaf photographs can be time-consuming.To address this issue, the proposed framework utilizes low-cost, open-source software to detect plant diseases accurately. The framework employs segmentation, feature extraction, and classification techniques to detect and identify plant diseases.

Digital images of leaves are taken, and the proposed framework uses CNN and deep neural networks to classify the affected region in the leaves accurately.
In conclusion, the development of deep learning algorithms has provided a promising approach for detecting plant diseases automatically. The proposed framework is an excellent example of the use of AI technology in agriculture and offers a low-cost solution for farmers to detect plant diseases accurately.

1.3 Scope

Agriculture is an integral part of the Indian economy. The Indian agriculture sector employs nearly half of the country's workforce. India holds the title of the world's largest producer of pulses, rice, wheat, spices, and spice products. In recent times, significant climate changes and lack of immunity in crops have produced a substantial increase in the proliferation of crop diseases. This leads to widespread crop destruction, hampers farmers' ability to grow food, and ultimately costs them money. It is quite tough for farmers to identify various diseases in various plants since it demands good experience in plant infection detection and farmers couldn’t pay a person to attain this issue.

The projected annual crop losses due to plant disease worldwide is roughly 14.1% of crop loss is responsible because of the plant disease. The old tools and techniques are not very useful since it takes up loads of time and manual labor. Hence, we need a system that focuses on disease detection and recognition which is useful for decision making. The suggested system consists of four main phases.

1.4 Problem statement

In the current era, technology has revolutionized various industries, including agriculture. With the advent of artificial intelligence, detecting plant diseases has become easier and more accurate. However, the traditional approach of detecting plant diseases with the naked eye observation by plant experts is still prevalent. This method is effective but requires skilled personnel, and it is time-consuming.

In countries where farmers lack access to experts or facilities, the traditional method may not be the best approach. This is where the suggested technique for tracking vast fields of crops becomes useful. By using automated disease detection techniques, farmers can quickly and easily identify plant diseases and take appropriate measures to prevent the disease from spreading.

The proposed technique for detecting plant diseases using artificial intelligence offers several benefits, such as reducing the cost of expert consultation, reducing the time required for disease identification, and increasing the accuracy of disease detection. It is an innovative approach that can help farmers detect diseases at an early stage and take appropriate measures to ensure healthy plant growth and yield.. Identifies the relationships between infections, hosts, and their environments. As a means of determining the nature of plant ailments. Using algorithms for precise outcomes, we can lessen the prevalence of disease outbreaks. By using efficient methods for reducing file size, we can ensure that image quality is not severely diminished while still saving space. By building upon the work of the writers, we can improve our ability to diagnose diseases with greater precision and obtain more reliable results. The primary goal is to use ML algorithms to diagnose plant diseases.

Among the primary aims are:
• Develop a method for more reliable disease detection in plants.
• Obtaining the right data collection for training and ensuring its continued high quality.
• Create a user-friendly interface.

1.5 System requirement specifications:

Software requirements:

Operating Systems: Linux and Windows.
Anaconda simulation tool with Jupyter notebook.

Python 3.8.

Hardware specifications:

Pentium IV/III processor.
Hard disc: at least 80 GB.
RAM should be at least 8 GB.

CHAPTER 2

LITERATURE SURVEY

2.1 General

A literature review is an essential component of academic research and serves as acritical tool for synthesizing and analyzing existing knowledge and understanding about a particular topic. By analyzing previously published research, the literature review identifies gaps in the existing knowledge, provides an overview of the current state of research, and highlights areas for further exploration. The review process typically involves locating and evaluating relevant sources, such as books, academic articles, and other scholarly publications. The sources are then organized into a coherent and logical framework that highlights the key themes, concepts, and debates surrounding the topic. The structure of a literature review can vary, but it typically begins with an introduction that provides background information on the topic and outlines the purpose and scope of the review. This is followed by a summary of the main findings and research methods employed in the previous studies, highlighting the strengths and weaknesses of each. The review then proceeds to a synthesis of the material, which involves combining and reorganizing information from the sources to identify patterns and trends in the literature. Overall, a well-constructed literature review is an important part of the research process, as it provides a comprehensive understanding of the existing knowledge in a field, highlights key areas for future research, and contributes to the development of new ideas and perspectives.

2.2 Review of Literature Survey:

Various procedures must be undertaken to determine if the leaf is diseased or healthy. Classification, Feature Extraction, Preprocessing, and Classifier Training. Ramesh et al [1] paper's merits are the suggestion of an RFC-based machine learning strategy for distinguishing between good and ill papaya leaves. However, it also has several drawbacks, including poor accuracy when additional plants were added to the dataset, which made it less precise. as well as, distinguishing between various diseases, a process that may be simplified with the use of a reliable descriptor and ML models.

S.P Mohanty [2] first provides the example leaf image. The leaf image's color channels are separated after the green pixels in the original image are masked. Using CNN, it gets more recall rate than other classifier techniques. provided an illustration using a photograph. The leaf image's color channels are then separated, and the original image's green pixels are hidden. I think this is the biggest problem with this article.

“Detection Grading of disease Diseases Using Computer Vision” [3] K-means clustering is used to divide the defective area; To extract textural information, GLCM is employed, and fuzzy logic is used to grade, the severity of the condition. They used an ANN as a classifier, which how seriously the sick leaf is affected.

S.S. Harakannanavar [4] proposed Removing the vein so it may be the image from each leaf, RGB photos are turned into white and subsequently into a grey-level image. Here offers a method for accurately diagnosing okra leaf samples for YVMV. The system reads aloud the topic and the content of the related message. It provides a voice base mailing service where the visually disabled person could read and send corresponding mail by their own without the help of others. It requires introductory information about keyboard keys. System has excluded all these and overcome all difficulties faced by the blind people.

SVM-based “Multiple Classifier System for Disease Identification Using Wheat Leaves,” [5] Color Characteristics are encoded in RGB to HIS using parameter for the GLCM's shape taken from there are seven stable states. They employed the use of a Multiclass SVM classifier, used Wheat plant disease detection while working offline.

J Eunice [6] has proposed a technique Bhattacharya's similarity formula that, when compared to 100 healthy photos, can be used to detect paddy plant illness. It is advantageous because it necessitates a lot of monitoring effort in large crop farms and because it identifies disease indications on plant leaves at an extremely early stage. It is insufficient to identify or categorize sickness. Training data cannot be separated linearly.

T.S. Xian [7] proposed, the diseased zone of the leaf is divided using the Indices Based Histogram method. The challenges of slice segmentation, polygon approximation, and mean-shift segmentation are all ones that the authors have successfully navigated. According to Kaleem et al., pre-processing entails resizing the photos to 300*300, eliminating background noise, boosting brightness, and adjusting contrast. SVM classifier is used to categorize leaf diseases, while K-means clustering is employed for segmentation and the extraction of important matter.

G Geetha introduced a "Pattern Recognition Techniques for Cotton Leaf Disease Identification" [8] This use a method based on slicing up snakes; in this instance, Hu's moments are used as a distinguishing feature. The BPNN classifier addresses the various class issues while the active contour model is referred to as employed to restrict the life force present inside the infected region. It is discovered that the average categorization is 85.52% (IJCER).

"Diagnosis of Banana Bacterial Wilt and Black Sigatoka Disease Using Vision-Based Automation" [9]. Area under the curve analysis, peak components, its five shape characteristics are used to generate max trees from color histograms after they have been extracted and converted from RGB to HSV and are all used to convert RGB to L*a*b. Classifiers were employed. Randomized trees produce a very high score in seven classifiers, offer real-time data, and give the application flexibility.

Shima Ramesh, P.V. Vinod, [10] suggested a series of procedures to detect whether a leaf is sick or healthy. These steps include data cleansing, feature extraction, classifier training, and classification. They suggested an RFC-based ML strategy for distinguishing between healthy and diseased papaya leaves, which is a significant step toward creating a reliable system. However, the accuracy is poor because they applied the method to more plants, which led to a larger and less reliable dataset in addition to improving disease classification with a robust descriptor and ML models.

Dr D. C. Joy A. Ms. Wise, [11] First gave picture of a leaf as an example. The image of the leaf is then broken down into its constituent color channels, and the green pixels are removed using a mask. There is a noticeable increase in recall rate when compared to other classifier methods while using CNN. They employed spectroscopic methods to identify back then. The main disadvantage, according to this paper, is that these methods are extremely costly and can only be used by skilled professionals.

Dhiman Mondal and Dipak Kumar Kole, [12] They produced a successful technique for identifying and categorizing YVMV in okra leaves using grayscale conversion of RGB images so that the image of the vein from each leaf can be extracted.

Aruna Chakraborty and D. Dutta Majumder [13] used Bhattacharya's similarity calculation to compare visually sick paddy plants to healthy ones. 100 reference photographs of healthy plants. It is helpful because it cuts down on the amount of time spent keeping an eye on crops on a vast farm, and it can see the first signs of illness right where they show up on the leaves. Not enough information to diagnose or categorize sickness. the inability to linearly divide training data.

Nanjesh B.R, Tejoindhi M.R, Ashwin Geet D'sa and Jagadeesh Gujanuru Math, “Indices Based Histogram” [14] is a technique presented by for isolating diseased areas of leaves on a plant. In the authors' opinion segmentation by slicing, approximating polygons, and shifting the image by a certain amount to get a segmentation by means of a mean shift all fall short. For Kaleem et al. to take them into consideration, images were reduced in size to 300x300 pixels, noise was eliminated, brightness was raised, and contrast was turned up to 11.

A. Blessie and Dr D.C. Joy Winnie Wise, " Prediction and classification of leaf diseases using digital image processing," [15] Innovations in Information, Communication Systems and Embedded Systems: IEEE International Conference (2021).

2.3 Existing system

The current approach for detecting plant disease is simple naked eye observation by plant experts, which can be used to detect and identify plant diseases. In these circumstances, the suggested technique is useful for tracking vast fields of crops. Furthermore, in some nations, farmers lack adequate facilities or are unaware that they can contact experts. As a result, consulting experts is not only more expensive but also more time consuming. In those circumstances, the suggested technique for tracking many plants would be useful. To prevent further damage to a plant's growth, early detection of a plant disease is crucial. Despite the widespread use of Machine Learning (ML) models for this purpose, the recent advances in a particular subset of ML known as Deep Learning (DL) have sparked renewed interest in this field of study because of the promising prospects for improved accuracy it offers. To detect and categorize plant illnesses, a wide variety of created and customized DL architectures are employed alongside several visualization techniques. In addition, several different performance indicators are employed in the analysis of these architectural styles and methods. In this article, we'll go through how different DL models can be used to depict plant diseases. Additionally, several study gaps are recognized to acquire higher clarity for diagnosing plant diseases before any outward signs appear manifest clearly, developing.

2.4 Disadvantages:

- Only humans can predict diseases.
- The procedure is extremely slow.
- Consumption of time, space and price is also very high.
- The use of CNN as a classifier has been overlooked.
- Unfortunately, the planned deployment of the model did not take place.

CHAPTER 3

SYSTEM ARCHITECTURE AND DESIGN

3.1 Deep learning model

The field of plant disease diagnosis is rapidly evolving, and cutting-edge technology and deep learning techniques may soon revolutionize how neurologists diagnose various plant diseases in a noninvasive manner. Deep learning's convolutional neural network (CNN) has shown remarkable promise in the realm of image categorization, making it a powerful tool for plant disease diagnosis.
To train the CNN, a database of plant leaf images was collected, with each image labeled as either healthy or diseased. Unlike other approaches, no preparation of plant disease data was required as the CNN was fed unaltered images as input. The database consisted of many photographs from various categories, including "deceased" and "healthy," which were used as samples to train the CNN.

The DL method of CNN was used to predict various plant disease outcomes in this study. The CNN is a powerful tool for image recognition and can be trained to recognize specific patterns or features in images, making it ideal for detecting the visual symptoms of plant diseases. By using CNN and additional feature extraction techniques, it may be possible to improve the accuracy and reliability of plant disease diagnosis.

Overall, this study highlights the potential of deep learning and CNNs in revolutionizing the field of plant disease diagnosis. With further advancements in technology and increased availability of plant leaf image databases, it may soon be possible to diagnose and treat plant diseases with greater accuracy and speed, helping to mitigate the significant economic losses caused by plant diseases.

Advantages:

1. To identify various plant diseases easily and reduce the workload of farmers.
2. In terms of deep learning techniques, it is the best model to identify various plant diseases quickly. CNN aids in the efficient analysis of images and videos.
3. Detects related images with a low-cost camera.

Application:

To Detect diseases on leaves of plants and recommend the pesticide as per the types of disease to Farmer". To find the interaction between the disease-causing agents and host plant in relation to the overall environment. To identify various diseases in plants. To implement a method for preventing the diseases and providing management for reducing the losses/damages caused by diseases. Prevent diseases on plants for farmers.

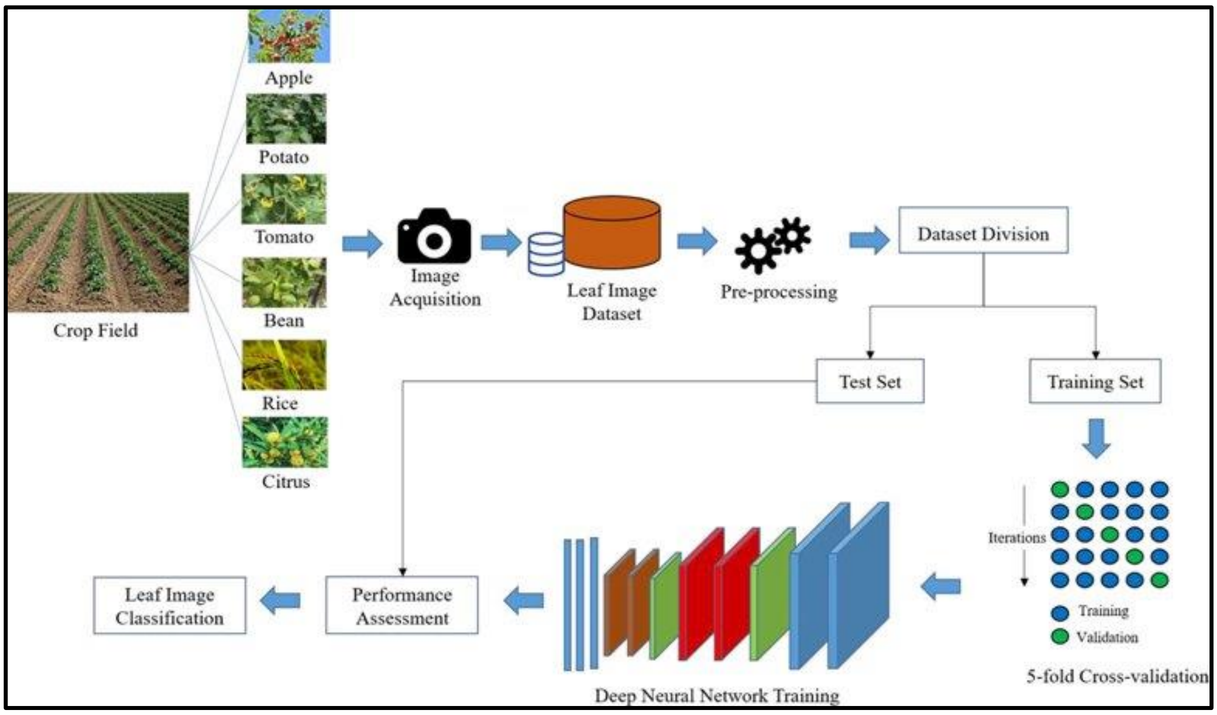


Figure 3.1.1 Architecture Diagram

Firstly, upload an image of a plant which needs to be tested and then the image gets processed and sent to a classifier. The classifier compares the uploaded image's characteristics to those of the photos already stored in the database; if a match is found, the corresponding illness is returned, otherwise, a message stating that the leaf is healthy is shown.

The process of plant disease detection using deep learning can be divided into two phases: training and testing. In the training phase, the user uploads an image of a leaf that may be healthy or diseased. The RGB image of the leaf will then be prepared for feature extraction. This is typically done by preprocessing the image to resize and normalize it. The next step is featuring extraction, which involves extracting relevant features from the leaf image.

Different algorithms can be used for feature extraction, such as convolutional neural networks (CNNs) or support vector machines (SVMs). Among the extracted features, some important features are selected to be sent to the classifier. In the final step of the training phase, the selected features are fed to the classifier, which identifies the disease based on the extracted features and labels it with the corresponding disease name. The classifier can be a machine learning algorithm such as a decision tree, random forest, or neural network.

In the testing phase, a new image of a leaf is uploaded, and the process of feature extraction is repeated. The extracted features are then sent to the classifier, which compares them with the features in the trained dataset to detect the plant disease. If the features match with a particular disease, the classifier will label the disease with the corresponding name. Overall, the process of plant disease detection using deep learning involves extracting relevant features from leaf images, selecting important features, and using a classifier to identify the disease. In the testing phase, the classifier compares the extracted features with the trained dataset to detect the disease.

3.2 Flow chart

A flowchart is a graphical representation of a process, system or algorithm using symbols and arrows to show the sequence of steps or actions in a logical order. It is a visual tool that helps to understand, analyse, and improve a process or system. In a flowchart, each step or action is represented by a symbol, such as a rectangle, circle, or diamond. Arrows or lines connect the symbols to show the sequence of steps or actions in the process. As seen in figure 3.2.1 first step is to collect the data and split it into training and testing dataset, then the images are pre-processed and are feeded to the CNN model which in turn classifies various plant diseases.

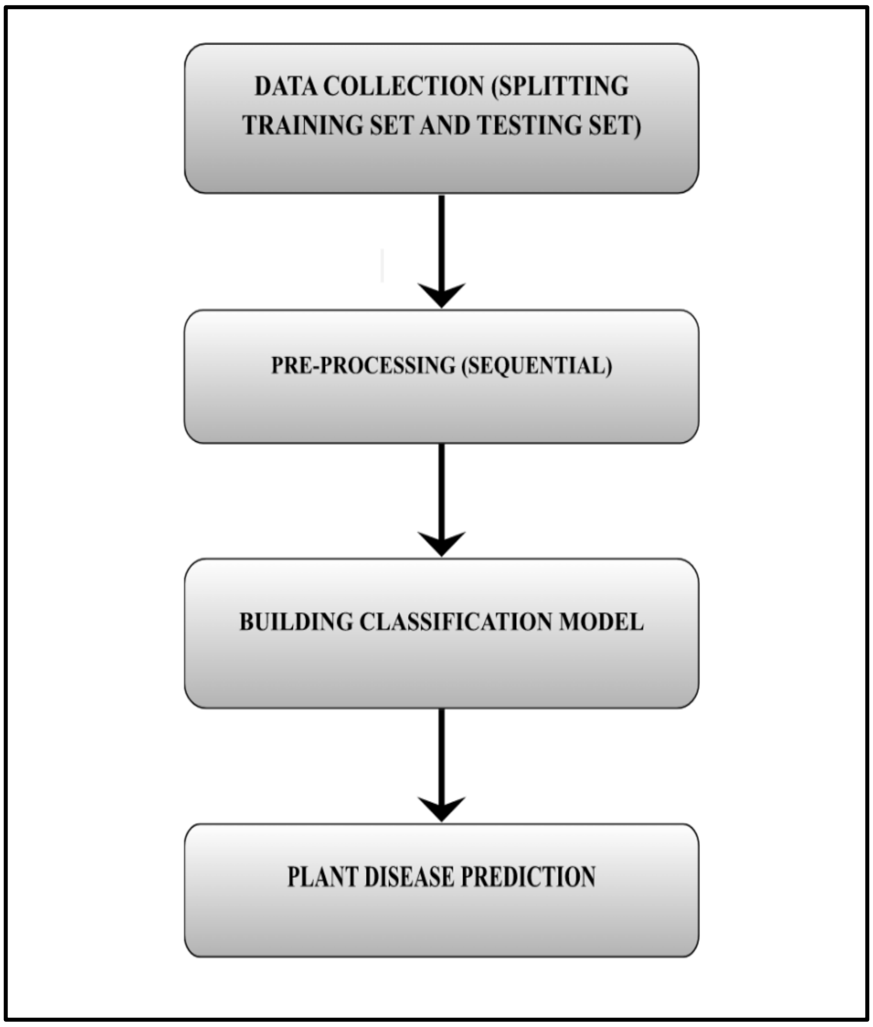

DATA COLLECTION (SPLITTING TRAINING SET AND TESTING SET)
PRE-PROCESSING (SEQUENTIAL)
BUILDING CLASSIFICATION MODEL
PLANT DISEASE PREDICTION

Figure 3.2.1 Flow Chart Diagram

3.3 Use case Diagram

Use case diagrams can be determined as the organized manner of system functionalities. In Figure 3.3.1 the functionalities required are the collection of the dataset then after the model is used, training the data using the model, testing the data after the training, then the deep learning algorithm is used and finally the disease classification as output will be displayed to the user with the help of the interface.

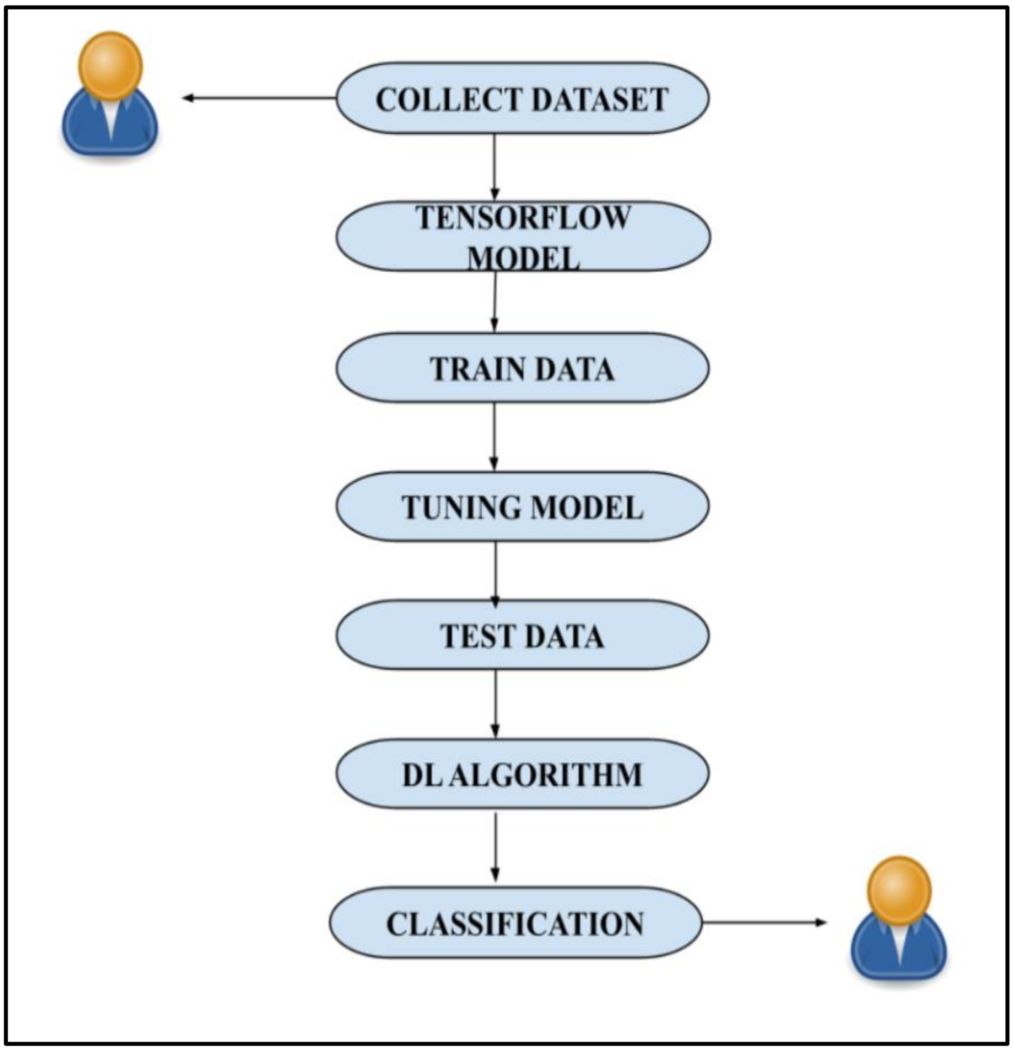


Figure 3.3.1 Use Case Diagram

3.4 Class Diagram

Class diagrams generally represent the static view of the system and represent different aspects of the application. Using notes whenever required to describe some aspects of the diagram and at the end of the drawing, it should be understandable to the end user. Finally, before making the final version, the diagram should be made on plain paper and rechecked as many times as possible to make it correct. As shown in figure 3.4.1, Firstly load the image and then feed it to the CNN model, then the model extract features from the image and classifies the disease.

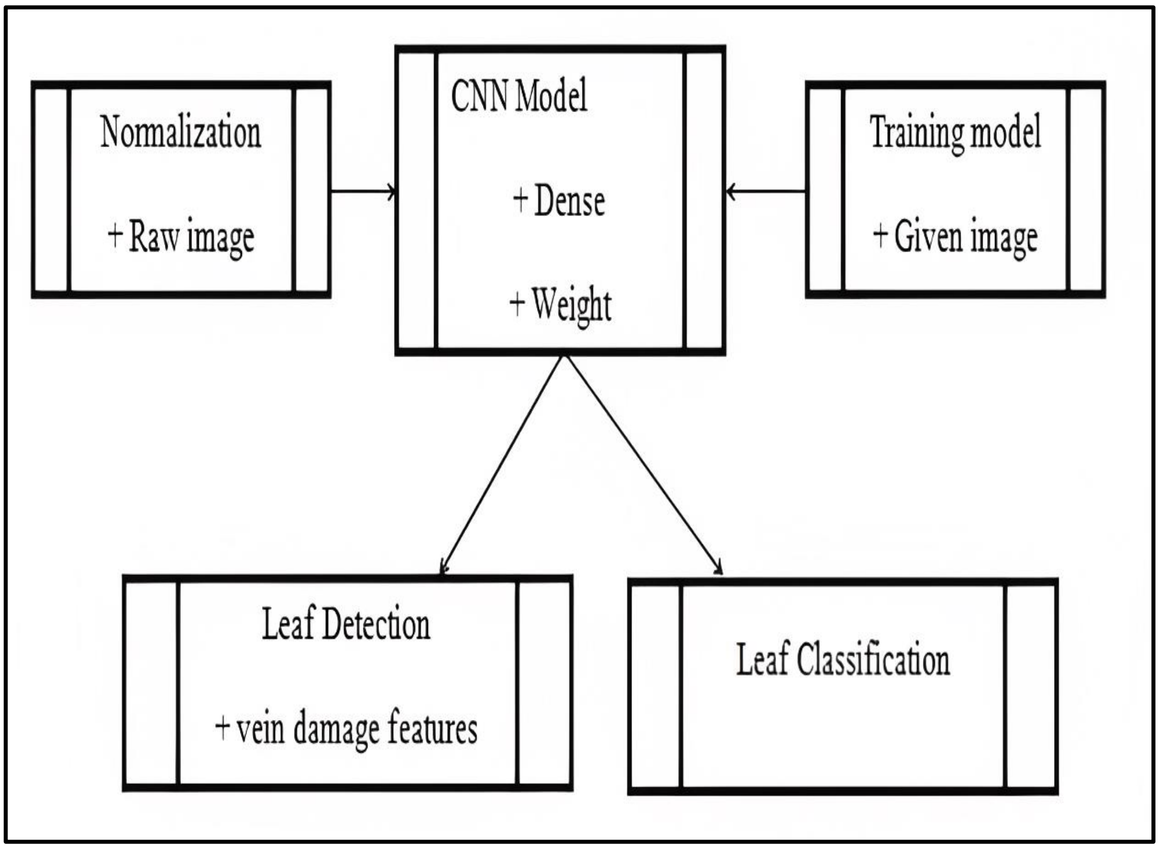


Figure 3.4.1 Class Diagram

3.5 Sequence Diagram

The flow of logic within the system in a visualized manner is known as a sequence diagram. In the above-mentioned Figure 3.5 image is dispatched as a report and then the past data is validated with the validated data. Tuning of the model is done after that disease classification, then the detailed report is generated by the deep learning model and then displayed to the user. As seen in figure 3.5.1 first step is to collect the data and split it into training and testing dataset, then the images are pre-processed and are feeded to the CNN model which in turn classifies various plant diseases.

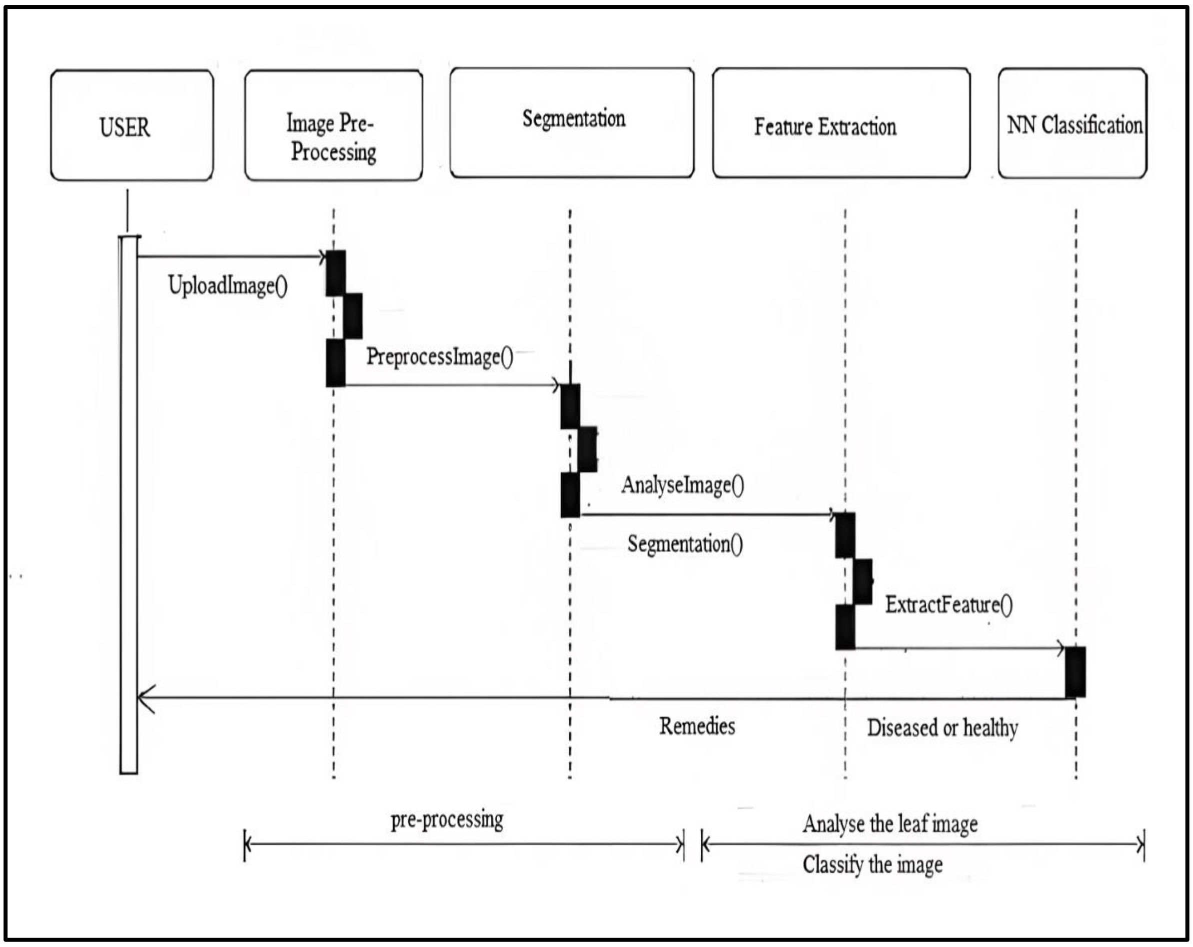


Figure 3.5.1 Sequence Diagram

3.6 Activity Diagram

The activity diagram starts with the "Start" node, which represents the beginning of the process. The first activity is "Capture Image," which represents taking a picture of a plant that might be diseased. The next activity is "Preprocess Image," which includes resizing, cropping, and normalizing the image to prepare it for input into the machine learning model. The next activity is "Extract Features," which involves using computer vision techniques to identify important features in the image that can help the machine learning model detect diseases. These features might include color, texture, shape, or other visual characteristics of the plant.

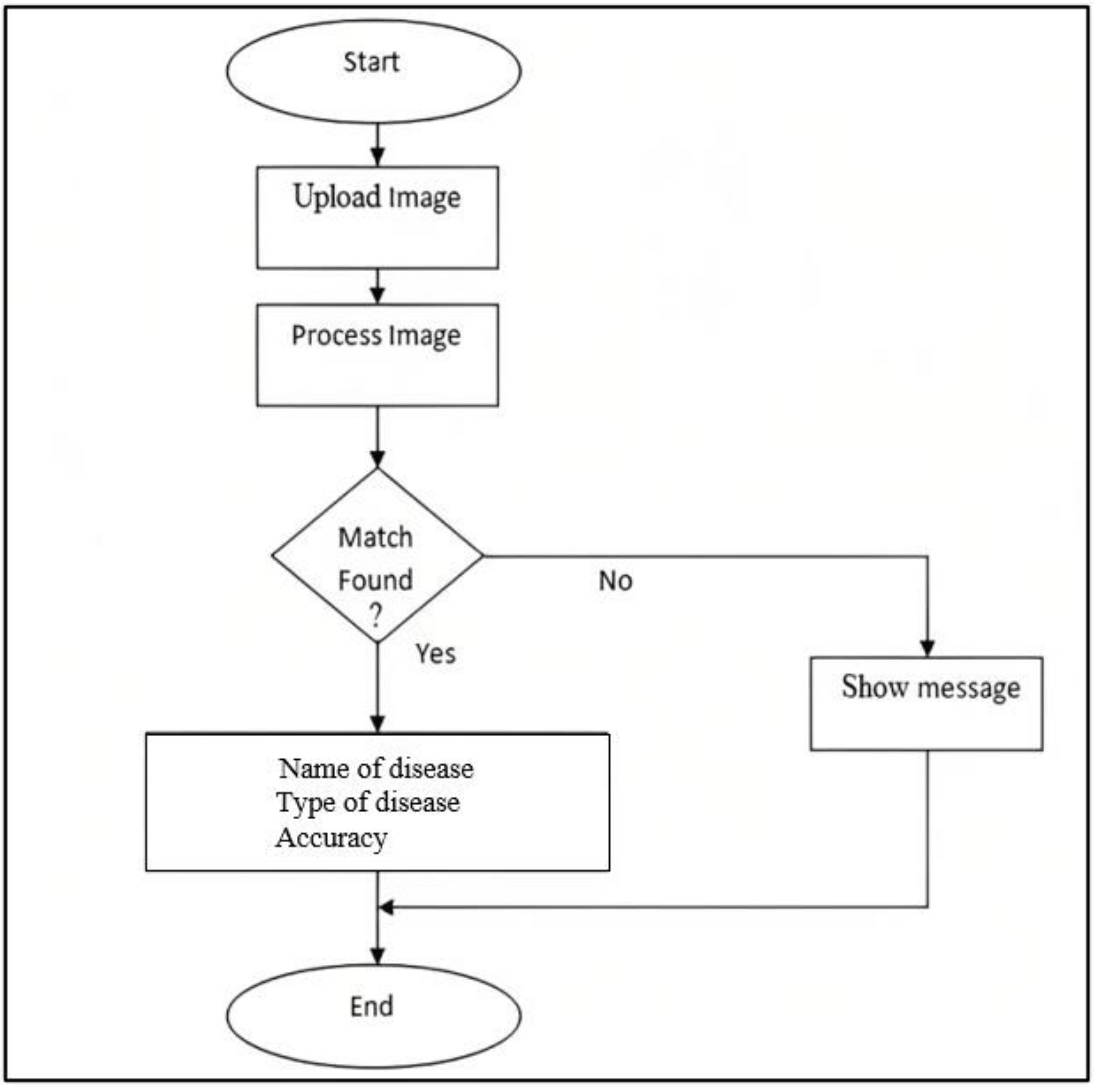


Figure 3.6.1 Activity Diagram

3.7 Block Diagram

In Figure 3.7.1 A block diagram is a graphical representation of a system, process or concept using blocks connected by lines or arrows to show the relationship between them. It is a way of visualizing the components of a system and their interactions.

In a block diagram, each block represents a component or function of the system, and the lines or arrows indicate the flow or direction of data or signals between the blocks. The blocks can be represented in various shapes and sizes, depending on the nature of the component or function they represent. As seen in figure 3.7.1 first step is to collect the data and split it into training and testing dataset, then the images are pre-processed and are feeded to the CNN model which in turn classifies various plant diseases.

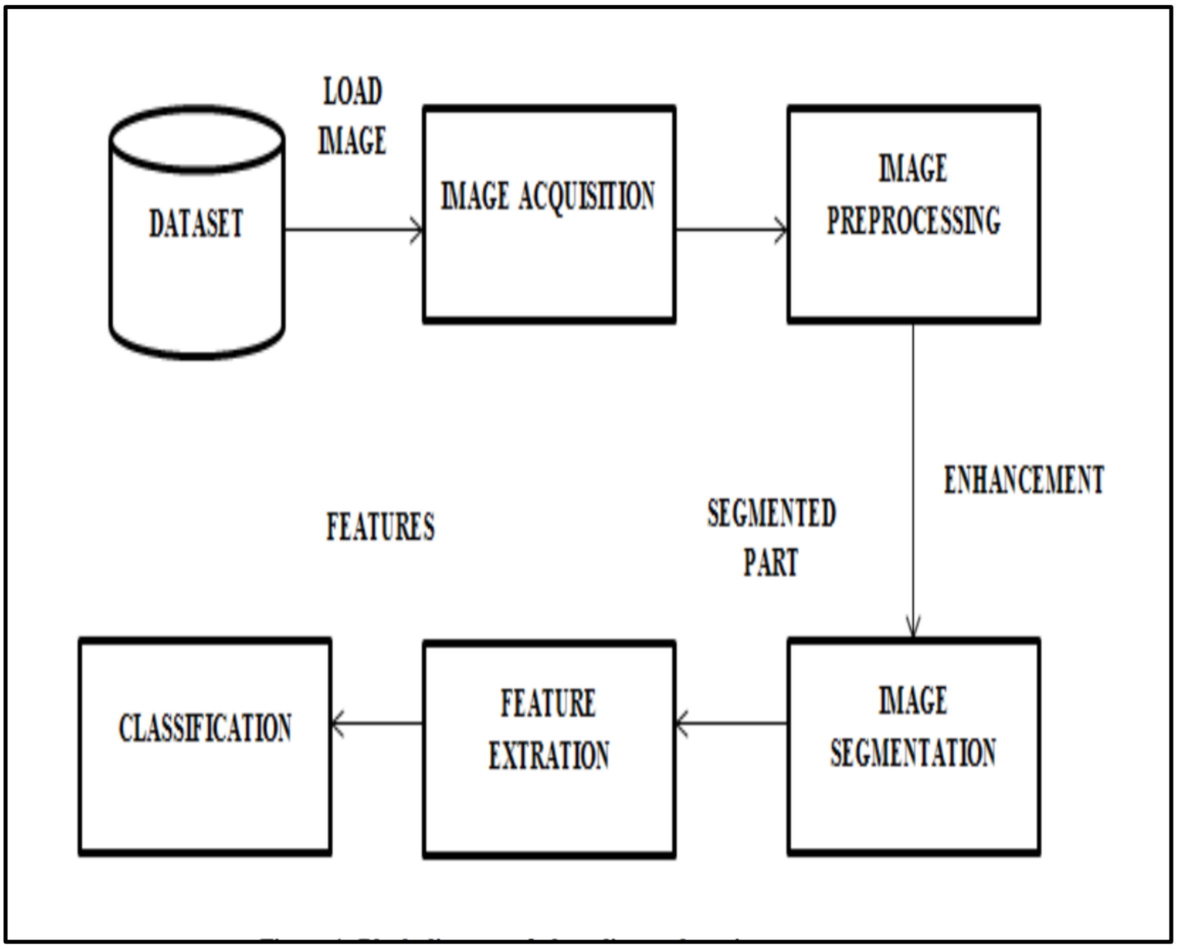


Figure 3.7.1 Block Diagram

3.8 CNN working process.

To train a model for detecting plant diseases using a Convolutional Neural Network (CNN), several steps are involved. First, the dataset of plant images needs to be preprocessed. This includes resizing and cropping the images, as well as converting them to an array format that can be used by the CNN. The same preprocessing steps are applied to the test image as well. The dataset used for training the CNN should include many images of various plant diseases, with hundreds of images available for each disease. These images serve as test cases for the program, and help the model learn to identify the specific features that indicate the presence of a particular disease.

The CNN network is composed of several layers, including Dense, Dropout, Activation, Flatten, Convolution2D, and MaxPooling2D. Each of these layers performs a specific function, such as reducing the size of the input image or extracting important features from the image. Once the model has been successfully trained on the dataset, it can be used to identify various plant diseases in new images. The test image is compared to the learned model to determine the presence of any specific disease. By following these steps, a CNN can accurately identify plant diseases, providing a valuable tool for farmers and others involved in agriculture. Convolutional neural networks (CNNs) have revolutionized the field of image analysis by providing an automated solution for complex image processing, classification, and segmentation tasks. CNNs use multiple layers of neurons that perform convolution operations, which allow them to extract and identify specific features in images. These features can then be used to classify or segment images, making CNNs extremely valuable for a variety of applications.

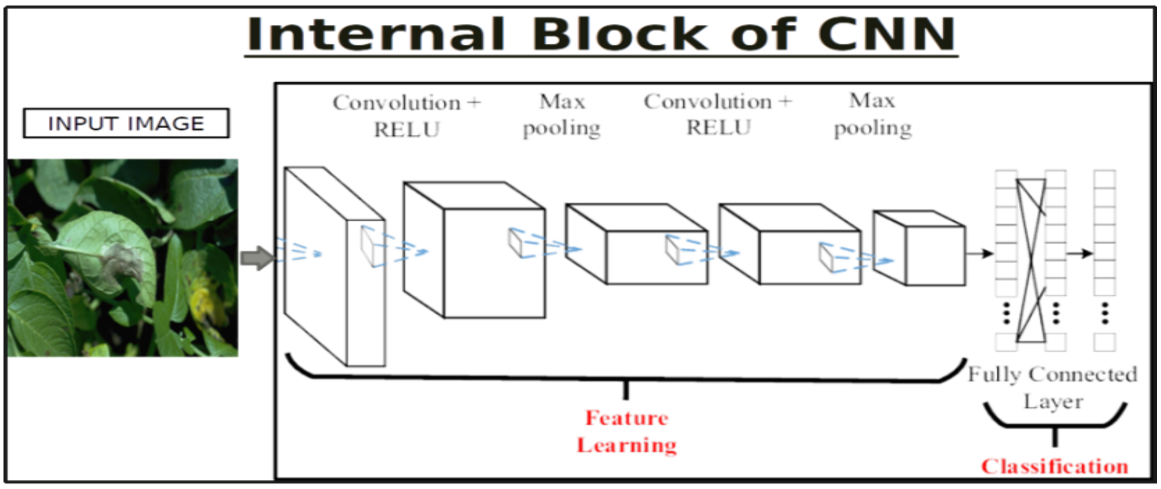


Figure 3.8.1 working of CNN.

- **Convolution**: We feed images here, apply convolutions and perform
- **Pooling**: After feature maps are extracted, we try to reduce the size of feature maps, that is compress them, which is down sampling.
- **Flattening**: After all the feature maps are pooled, we flatten them. Here, we consider the 2D pixels and convert into 1D and then feed them to our dense fully connected artificial neural network and train the model.
- **Padding:**

  There are a few ways to deal with the pixels around the edges:
- Inadequate resolution due to missing edge pixels
- Inflating by using pixels with no value
- Protective cushioning against reflections

By far the most effective method is reflection padding, which involves copying pixels from the image's edges and pasting them onto the image's periphery so that the convolutional kernel has enough room to analyze the edge pixels. With a 3x3 kernel, an extra pixel should be placed around the perimeter.

The dimension has been half and adjusted to the number of pixels added to each side. Many research articles follow a common practice of simply ignoring the edge pixels, which results in a loss of data (which is exacerbated by the presence of multiple deep convolutional layers). Due to this, location of any suitable graphics that could effectively express some of the arguments made here without being deceptive or conflating stride 1 and stride 2 cvds.

- **Normalization:**

  The term "normalization" refers to the process of removing the mean and subtracting the standard deviation. In a process known as Min-Max scaling, the range of the dataset is transformed to be between -1 and 1, standardizing the data on the same scale.

  Normalization of input features is a typical technique for achieving data standardization by means of the mean being subtracted & scale to random values. It is often crucial that the input features have the same order and variance and are centered around zero. Images, for example, have their data scaled so that the values range from 0 to 1 by dividing each pixel's value by 255.

- **Rectified Linear Unit:**

  A Rectified Linear Unit is used as a non-linear activation function. A ReLU says if the value is less than zero, round it up to zero.
- **Strides:**

  If the input has width and height and the kernel iterates over each pixel in turn, the resulting output will also have width & height. The output channel/feature map size can be reduced by half by using a stride two convolution instead of a stride one convolution, in which the convolutional kernel takes strides of one pixel. Since the kernel only generates a single, aggregated output for each stride, the maximum value that could be produced input with width w, height h, and depth 3 with padding would be width w/2, height h/2, and depth 1.
- **Batch Normalization:**

  Training times can be cut in half and accuracy can be improved by a magnitude when using batch normalization to stabilize the predictions made by a network.

  By removing the average batch's activations and dividing the standard variation of the batch's activations. However, even after normalizing the input, some activations may still be larger than average, rendering the network less stable. By applying a modification called batch normalization, we can keep the return confidence interval around 1 and the mean output close to 0. It's important to note that the way batch normalization functions vary between the training and inference phases.
- **Input shape:**

  In the context of deep learning, the input shape refers to the size and dimensions of the data that is fed into the model for training. For images, the input shape typically consists of the number of pixels in the image and the number of color channels.
- **Output shape:**

  The output shape of a deep learning model refers to the size and dimensions of the data that is produced by the model after processing the input data. For image classification tasks, the output shape typically consists of the number of classes or categories that the model is trained to recognize.
- **Epochs:**

  In deep learning, an epoch refers to a single iteration through the entire training dataset. The number of epochs is a hyperparameter that is set by the user and determines how many times the model will be trained on the entire dataset.

- **Image Data Generator:**

  An image data generator is a tool in the Keras library that is used to generate batches of image data for training deep learning models. It can be used to apply data augmentation techniques such as resizing, shearing, zooming, and flipping to the input images to increase the size of the training dataset and improve the model's ability to generalize to new images.

- **Training Process:**

  The training process involves iteratively feeding batches of input data into the model and adjusting the model's weights and biases in response to the errors between the model's predictions and the ground truth labels. This is done using an optimization algorithm such as stochastic gradient descent.

- **Validation process:**

  The validation process involves evaluating the performance of the model on a separate dataset that is not used for training. This is done to assess the model's ability to generalize to new data and avoid overfitting to the training dataset. The validation dataset is typically a subset of the overall dataset, and the validation steps parameter specifies how many batches of data to process before evaluating the model's performance.

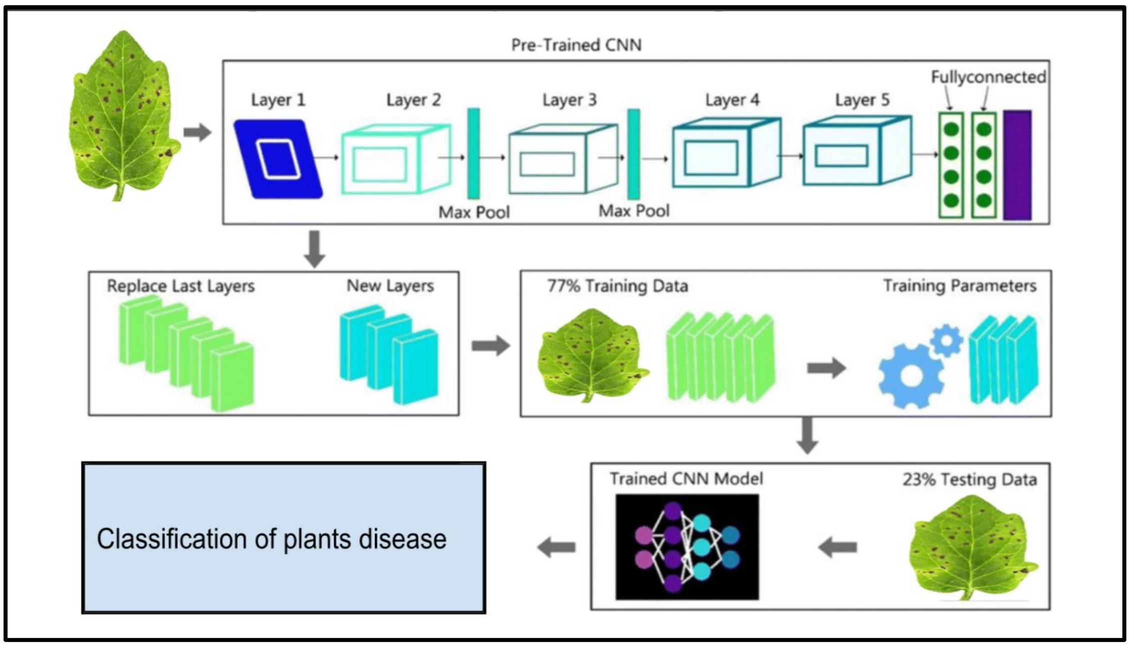


Figure 3.8.2 Layers of CNN model

3.9 Keras model:

While the sequential model may be simple to use, it can be limiting in terms of its layer stack structure with only one input and output. This is where the Functional API comes into play, providing a versatile model design. In fact, for most users and purposes, this method is highly recommended. Keras is known for its industry-strength models, made possible through the technique of model subclassing. By creating your own implementation of the model, you can tailor it to your specific needs and achieve even greater levels of customization and control. With the power of the Functional API and Keras model subclassing, the possibilities are endless.

Types of Keras Models

1. Sequential Model in Keras

Through this method, we can build up models in a methodical fashion. But we can't build models with multiple inputs and/or outputs because of this limitation. It works well with one-input, one-output layers in a simple stack. The presence of many inputs or outputs in any given tier of the stack renders this paradigm inappropriate. It is inappropriate even if non-linear topology is what we seek.

2. Functional API in Keras

To build a model as well as add layers in keras, it gives you greater freedom. Using a multi-input/multi-output paradigm is made possible by the API's functional programming interface. In addition, this facilitates the dissemination of these levels. Using Keras functional API, we can construct layer graphs. Since a functional API is just a data model, it can be stored as a single file and used to recreate the same model even if you lose the source code. The graph can be modeled with little effort, and its nodes can be accessed without any hassle.

Model Subclassing in Keras

Developing models using a sequential approach is restrictive. The flexibility of a fully functional API is limited. However, you may make your own complete model in Keras. To do this, a call method must be subclassed from the Model class. A Keras tensor is created with the help of the input () function.

kernels:

Inside each convolutional layer is a filter known as a convolutional kernel. The integer matrix that is of the same size as that of the kernel is used as the filter, and it is applied to a subset of an input image pixel. For lucidity's sake, in the final channel/feature map, we multiply each pixel by its level equivalent within the kernel and sum the products. Each convolution is a special case of an affine function, making these changes linear in nature. A common form of input for computer vision is an RGB image, which has three color channels. To keep things simple, let's use a grayscale image with a single channel (a two-dimensional matrix) and a convolutional kernel of size three.

The kernel scans the first few rows of the input matrix, which contains the image's pixel values, by moving horizontally over the columns of integers. The kernel then descends in a vertical fashion to the succeeding rows. Please take note that the filter may skip over a single pixel or multiple pixels at once, as will be explained in further depth below.

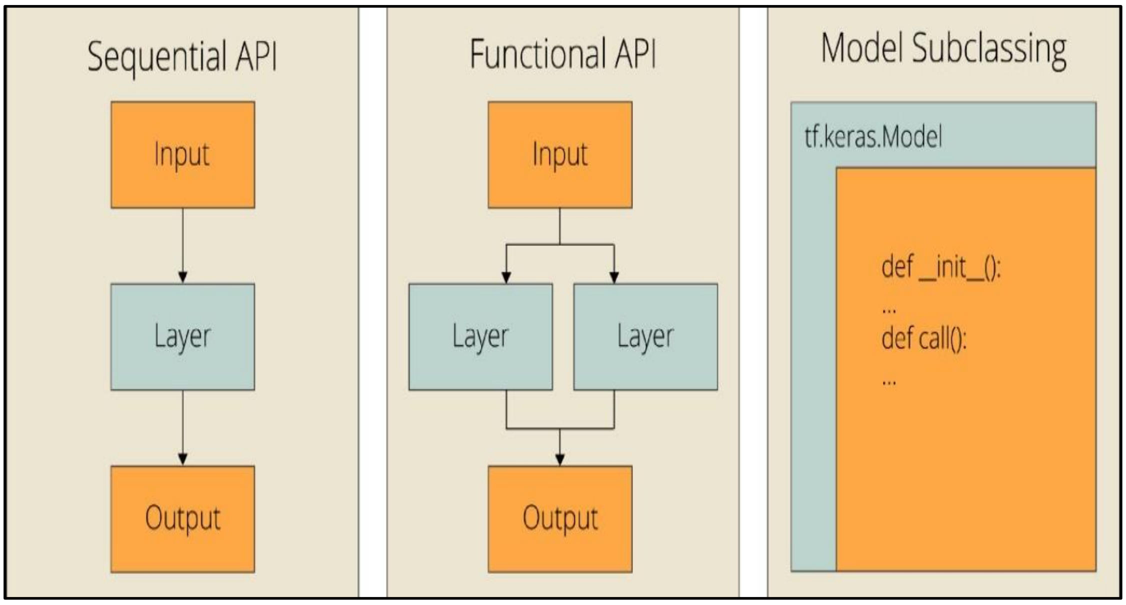


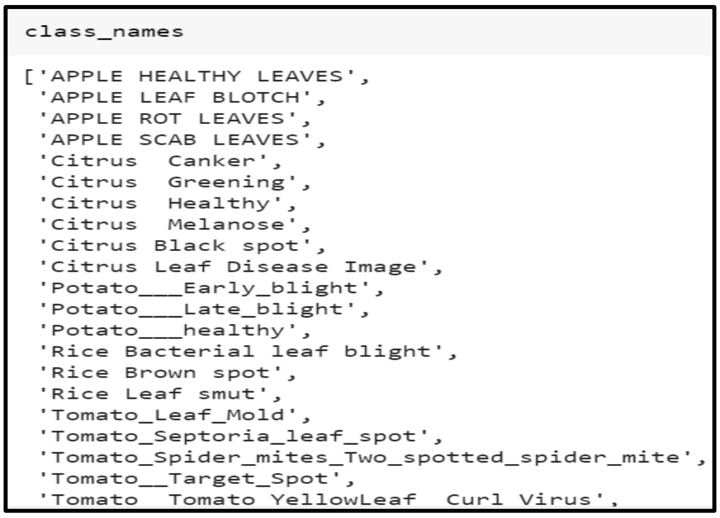
Figure 3.9.1 Keras Model

3.10 Preparing Dataset

This dataset contains approximately 6237 images which are further classified into 4865 for training images and 1372 for testing images, which were then classified into 21 Classes as shown in Figure:

1. Early Blight: The plant which is at starting stage of the disease.
2. Late Blight: The plant which is at ending stage of the disease.
3. Healthy: The plant which is healthy and have no symptoms of disease.
4. Apple leaf blotch: Apple leaf contains dark circular green patches which is caused by bacteria.
5. Apple Scab leaves: Apple scab is a common fungal disease that affects apple trees. It causes dark, scabby lesions to form on the leaves, which can ultimately cause the leaves to yellow, wither, and drop prematurely.
6. Apple rot leaves: Apple rot is a disease that affects apple trees, causing the leaves to rot and decay. This disease is typically caused by fungal pathogens that can infect the leaves, as well as the fruit and bark of the tree.
7. Citrus scab: affects citrus trees, including oranges, lemons, limes, grapefruits, and tangerines. The disease is caused by the fungus Elsinoe fawcettii, which infects the leaves, fruit, and twigs of the tree.
8. Citrus Canker: is a bacterial disease that affects citrus trees. It causes characteristic lesions on the leaves, stems, and fruit of infected trees.
9. Citrus tristeza: Citrus tristeza is a viral disease that affects citrus trees and is caused by the Citrus tristeza virus (CTV).
10. Citrus greening: is a bacterial disease that affects citrus trees, including oranges, lemons, limes, grapefruits, and tangerines. The disease is caused by the bacteria Candidatus Liberibacter.
11. Rice bacterial leaf blight is a bacterial disease that affects rice plants. Symptoms of the disease include water-soaked, yellowish-green, or grayish-white spots on the leaves, which can turn brown and necrotic as the disease progresses.
12. Rice brown: Symptoms of the disease include circular to oval-shaped lesions on the leaves that are brown or reddish-brown in color& yellow halos around them.
13. Rice leaf smut: Symptoms of the disease include elongated, gray-green, or yellow-green lesions on the leaves that can expand and merge over time. Infected leaves may also become twisted or distorted.

14. Tomato bacterial spot: Symptoms of the disease include small, water-soaked lesions on the leaves, stems, and fruit that can turn brown or black and necrotic. Infected fruit may also exhibit scarring or cracking.
15. Tomato leaf mold: Symptoms of the disease include yellowing and wilting of the lower leaves, followed by the appearance of velvety, grayish-brown mold on the upper surface of the leaves.
16. Tomato Septoria leaf spot: Symptoms of the disease include small, circular lesions on the leaves that are grayish brown in color and have a yellow halo around them.
17. Tomato spider mites: Symptoms of the infestation include yellowing or bronzing of the leaves, webbing on the undersides of the leaves, and a decline in plant health.
18. Tomato target spot: Symptoms of the disease include circular lesions on the leaves that are grayish brown in color with concentric rings or a "target" pattern.
19. Tomato mosaic virus: Symptoms of the disease include mosaic-like patterns of yellowing and discoloration on the leaves, as well as stunted growth, distorted fruit, and reduced yields.
20. Bean angular leaf spot: Bean angular leaf spot is a bacterial disease that affects bean plants. Symptoms of the disease include angular, water-soaked lesions on the leaves that can turn brown and necrotic over time.
21. Bean rust: Bean rust is a fungal disease that affects bean plants. Symptoms of the disease include reddish-brown pustules on the undersides of the leaves that can release spores.



```
class_names
```

```
['APPLE HEALTHY LEAVES',
 'APPLE LEAF BLOTCH',
 'APPLE ROT LEAVES',
 'APPLE SCAB LEAVES',
 'Citrus  Canker',
 'Citrus  Greening',
 'Citrus  Healthy',
 'Citrus  Melanose',
 'Citrus Black spot',
 'Citrus Leaf Disease Image',
 'Potato___Early_blight',
 'Potato___Late_blight',
 'Potato___healthy',
 'Rice Bacterial leaf blight',
 'Rice Brown spot',
 'Rice Leaf smut',
 'Tomato_Leaf_Mold',
 'Tomato_Septoria_leaf_spot',
 'Tomato_Spider_mites_Two_spotted_spider_mite',
 'Tomato__Target_Spot',
 'Tomato  Tomato YellowLeaf  Curl Virus',
```

Figure 3.10.1 Plant Disease Classes

Dataset:

Our dataset is in the format of .jpg in which we divided them into early blight, late blight, and healthy images.

Training: 4865 images.

Testing: 1372 images

Batch Length: 78

Batch Size: 80

Epochs: 100.

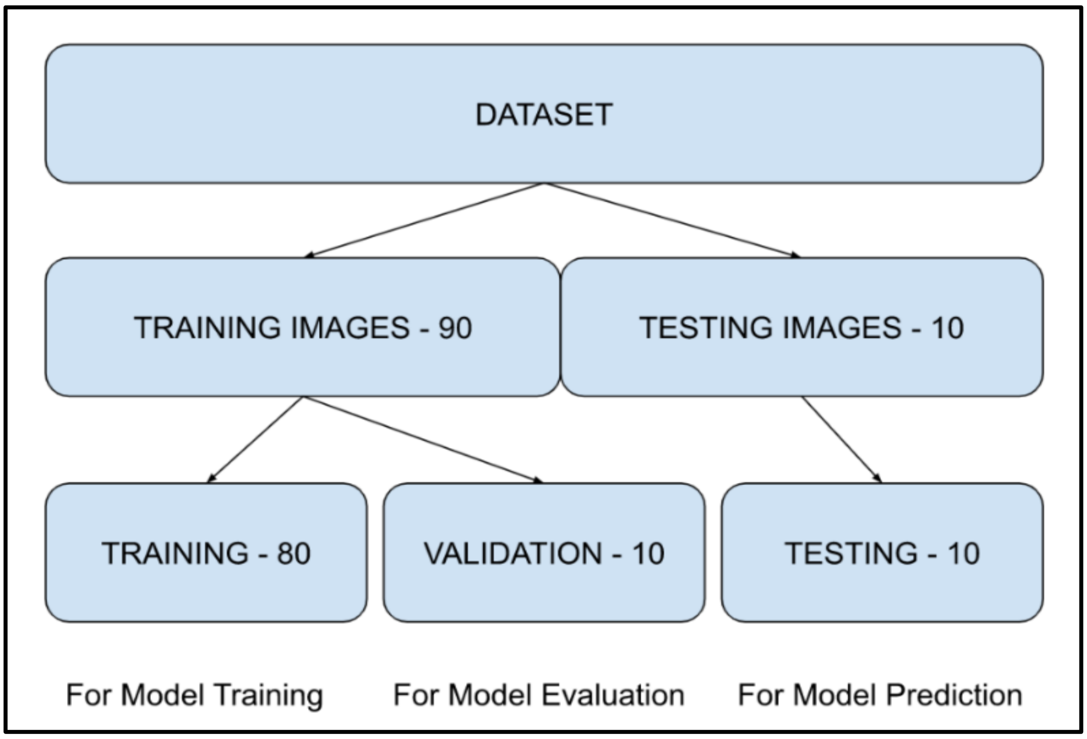
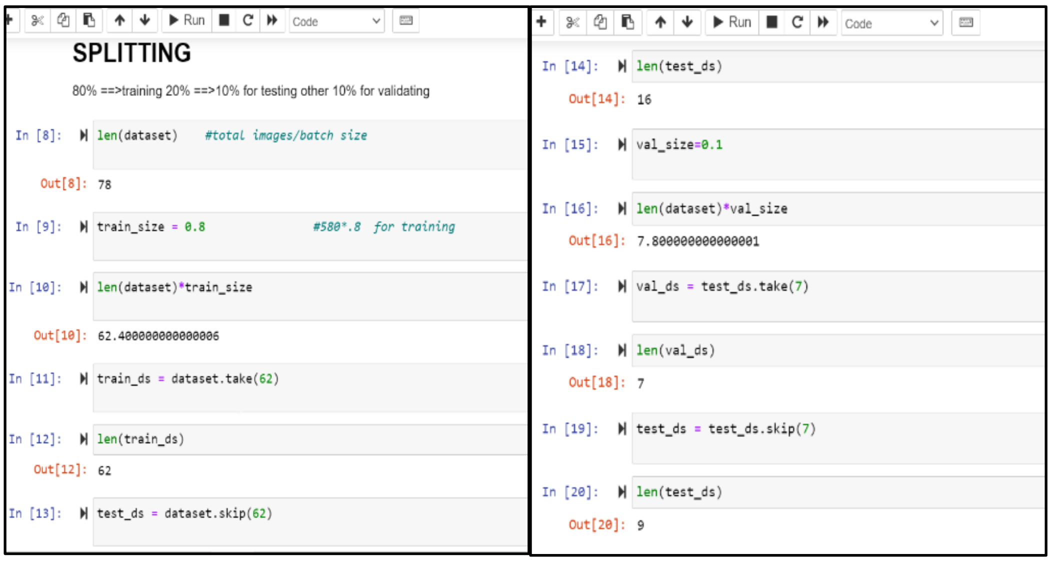


Figure 3.10.2 Splitting of datasets

The images are pre-processed, and the samples are split into three parts naming them as a test, train, and validation. This splitting is done in different percentages i.e., 69 images to train the dataset in which 62 images are divided into training set and 7 images into validation set, and finally the remaining 9 images is for testing set. (All values are in percentage.)



SPLITTING

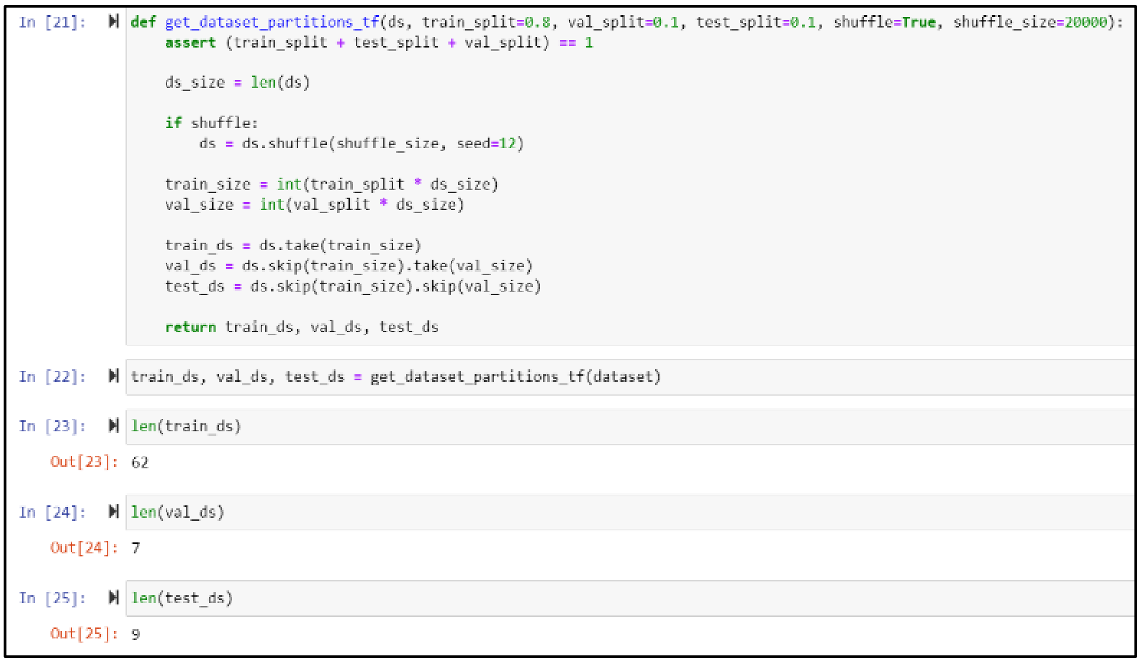
80% ==>training 20% ==>10% for testing other 10% for validating

```
In [8]:  len(dataset)     #total images/batch size

Out[8]:  78

In [9]:  train_size = 0.8            #580*.8  for training

In [10]: len(dataset)*train_size

Out[10]: 62.400000000000006

In [11]: train_ds = dataset.take(62)

In [12]: len(train_ds)
Out[12]: 62

In [13]: test_ds = dataset.skip(62)
```

```
In [14]: len(test_ds)
Out[14]: 16

In [15]: val_size=0.1

In [16]: len(dataset)*val_size
Out[16]: 7.800000000000001

In [17]: val_ds = test_ds.take(7)

In [18]: len(val_ds)
Out[18]: 7

In [19]: test_ds = test_ds.skip(7)

In [20]: len(test_ds)
Out[20]: 9
```

Figure 3.10.3 Dataset splitting

As mentioned above in the figure 3.10.3 The code snippet for dataset splitting is shown, it is as follows 80% for training 10% for testing and 10% for validation.



```
In [21]: def get_dataset_partitions_tf(ds, train_split=0.8, val_split=0.1, test_split=0.1, shuffle=True, shuffle_size=20000):
             assert (train_split + test_split + val_split) == 1

             ds_size = len(ds)

             if shuffle:
                 ds = ds.shuffle(shuffle_size, seed=12)

             train_size = int(train_split * ds_size)
             val_size = int(val_split * ds_size)

             train_ds = ds.take(train_size)
             val_ds = ds.skip(train_size).take(val_size)
             test_ds = ds.skip(train_size).skip(val_size)

             return train_ds, val_ds, test_ds

In [22]: train_ds, val_ds, test_ds = get_dataset_partitions_tf(dataset)

In [23]: len(train_ds)
Out[23]: 62

In [24]: len(val_ds)
Out[24]: 7

In [25]: len(test_ds)
Out[25]: 9
```

Figure 3.10.4 Output of splitting

As mentioned in the figure 3.10.4 This code snippet is for the output of splitting. 62 images for training, 7 images for validation and 9 images for testing whereas the batch size is 78.

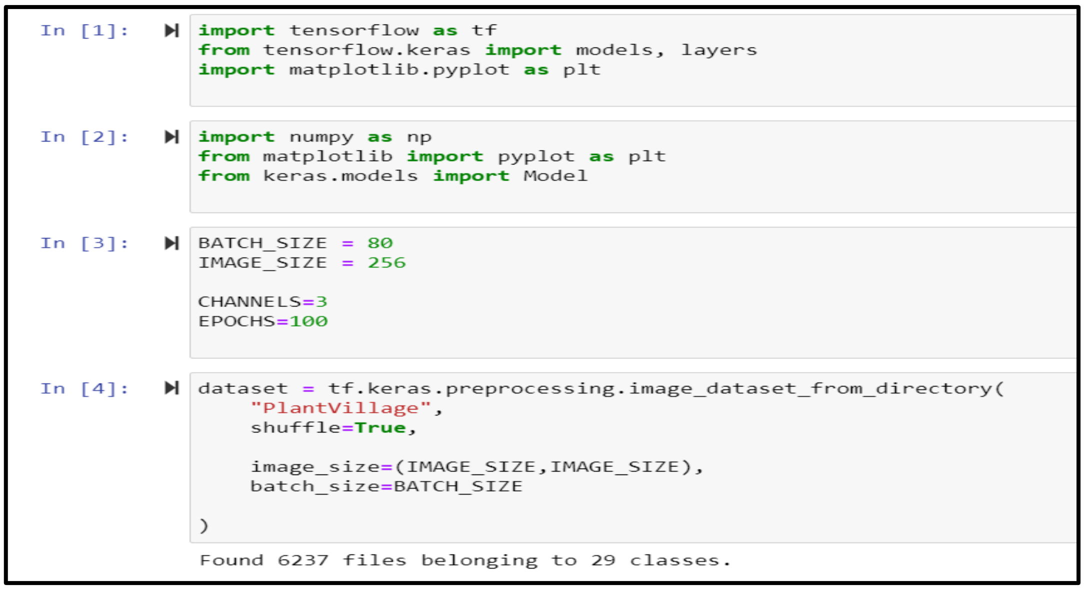
CHAPTER 4

METHODOLOGY

4.1 Image Acquisition:

Plant disease detection is carried by using two different data sets. The first data set has 15 categories, whereas the second has 38. Multiple photographs of each plant are included in both databases. The number of photos in the first batch is 2952. This work concludes on the village plant dataset, which includes 38 types of plants.

The dataset of diseased plant images has been collected from the website Kaggle and this dataset is used for the training of the model for the classification of diseases on plants. The dataset includes diseased plants of potato leaves, citrus, apple leaves and tomato leaves. Using the keras preprocessing image data generator function, we must first import our data set before we can generate its dimensions, rescale it, determine its range, zoom in and out, and flip it horizontally. Next, we use the data generator to load our image dataset from a local folder.

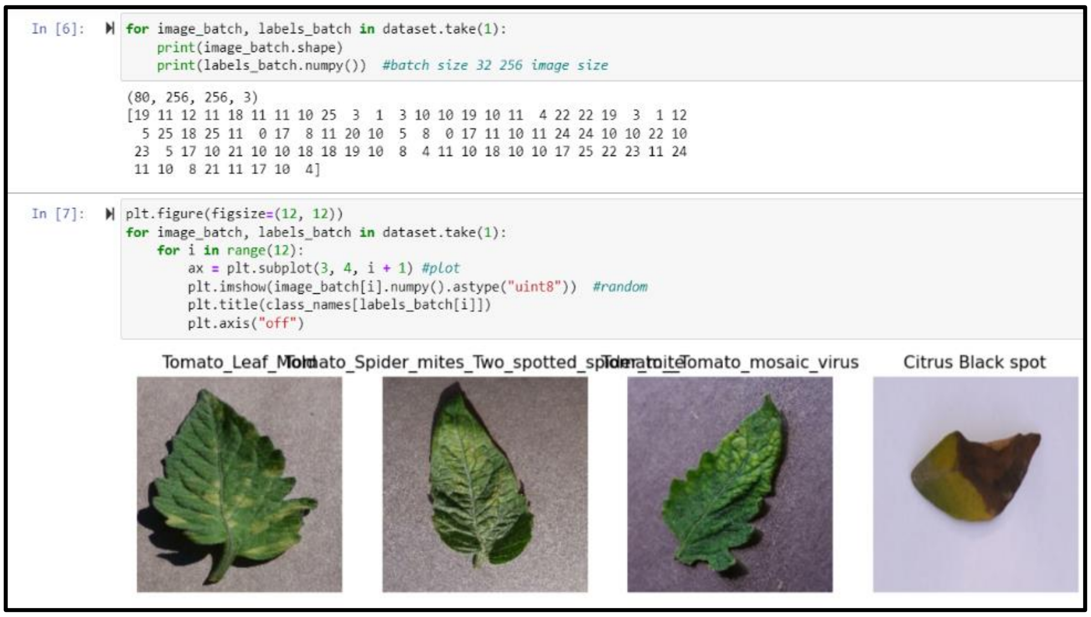


```
In [1]:  import tensorflow as tf
         from tensorflow.keras import models, layers
         import matplotlib.pyplot as plt

In [2]:  import numpy as np
         from matplotlib import pyplot as plt
         from keras.models import Model

In [3]:  BATCH_SIZE = 80
         IMAGE_SIZE = 256

         CHANNELS=3
         EPOCHS=100

In [4]:  dataset = tf.keras.preprocessing.image_dataset_from_directory(
             "PlantVillage",
             shuffle=True,

             image_size=(IMAGE_SIZE,IMAGE_SIZE),
             batch_size=BATCH_SIZE
         )
         Found 6237 files belonging to 29 classes.
```

Figure 4.1.1 Importing Datasets

4.2 Preprocessing:

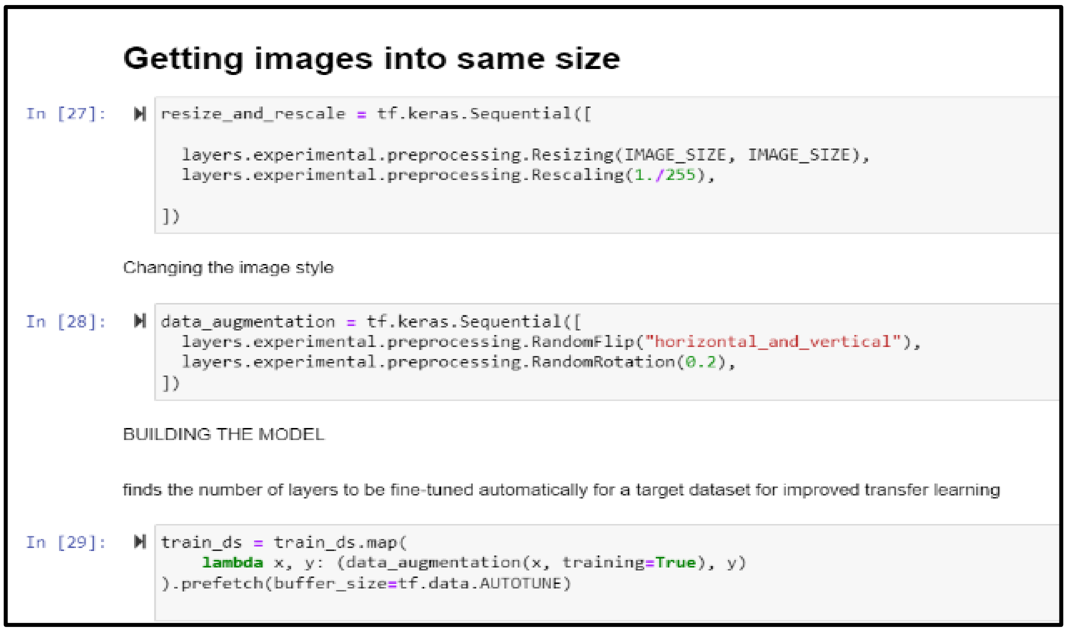
In this module, we specify the conditions under which the network should be trained, including the types of data to be used, the number of training iterations, the size of each training batch, and the training mode. Preprocessing involves techniques such as resizing, rescaling, rotation, normalization, and data augmentation to improve the efficiency of the model. In data augmentation the training images undergo flipping, rotating, brightness, cropping and contrast. And increase the dataset without adding more images to the dataset.



```
In [6]:  for image_batch, labels_batch in dataset.take(1):
             print(image_batch.shape)
             print(labels_batch.numpy())  #batch size 32 256 image size

         (80, 256, 256, 3)
         [19 11 12 11 18 11 11 10 25  3  1  3 10 10 19 10 11  4 22 22 19  3  1 12
           5 25 18 25 11  0 17  8 11 20 10  5  8  0 17 11 10 11 24 24 10 10 22 10
          23  5 17 10 21 10 10 18 18 19 10  8  4 11 10 18 10 10 17 25 22 23 11 24
          11 10  8 21 11 17 10  4]

In [7]:  plt.figure(figsize=(12, 12))
         for image_batch, labels_batch in dataset.take(1):
             for i in range(12):
                 ax = plt.subplot(3, 4, i + 1) #plot
                 plt.imshow(image_batch[i].numpy().astype("uint8"))  #random
                 plt.title(class_names[labels_batch[i]])
                 plt.axis("off")
```

Tomato_Leaf_MoTomato_Spider_mites_Two_spotted_spTomato_iteTomato_mosaic_virus
Citrus Black spot

Figure 4.2.1 Preprocessing



Getting images into same size

```
In [27]:  resize_and_rescale = tf.keras.Sequential([

            layers.experimental.preprocessing.Resizing(IMAGE_SIZE, IMAGE_SIZE),
            layers.experimental.preprocessing.Rescaling(1./255),

          ])
```

Changing the image style

```
In [28]:  data_augmentation = tf.keras.Sequential([
            layers.experimental.preprocessing.RandomFlip("horizontal_and_vertical"),
            layers.experimental.preprocessing.RandomRotation(0.2),
          ])
```

BUILDING THE MODEL

finds the number of layers to be fine-tuned automatically for a target dataset for improved transfer learning

```
In [29]:  train_ds = train_ds.map(
              lambda x, y: (data_augmentation(x, training=True), y)
          ).prefetch(buffer_size=tf.data.AUTOTUNE)
```

Figure 4.2.2 Preprocessing techniques

4.3 Image Enhancement:

The null values in the data set should be removed before feeding it to our Machine Learning Algorithm. Missing Values should not be feeded to the Model It makes the Model not to work properly. If there are any null values in the Dataset need to process the data with methods like Dropping or Imputation. Common techniques for image enhancement are removing noise, adjusting contrast and brightness. The technique used for these processes is DAE – Denoising auto encoder.

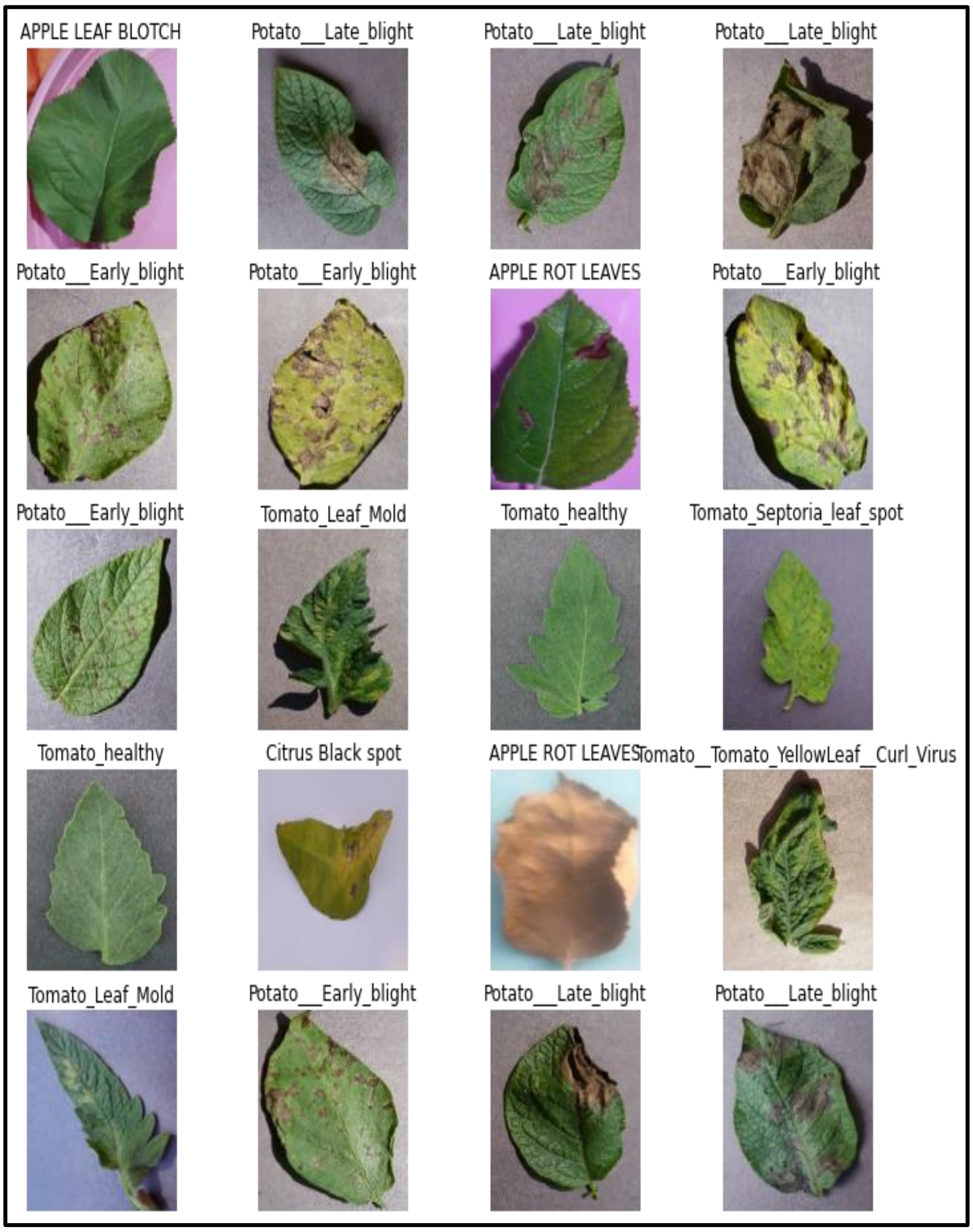


Figure 4.3.1 Image Enhancement

4.4 Feature Extraction:

This module is responsible for extracting relevant features from the input images. In the realm of image analysis, CNN stands out as a powerful feature extraction tool due to its ability to capture intricate patterns and structures within images. In the realm of machine learning, feature extraction is a critical part of dimensionality reduction, which involves reducing a vast amount of raw data into smaller, more manageable sets. When dealing with an enormous amount of data, it's crucial to minimize resource consumption and avoid errors, which is where feature extraction comes in. By selecting and combining variables into functions, feature extraction helps to identify the most pertinent features from large data sets, making it an essential tool for any data scientist or machine learning practitioner.

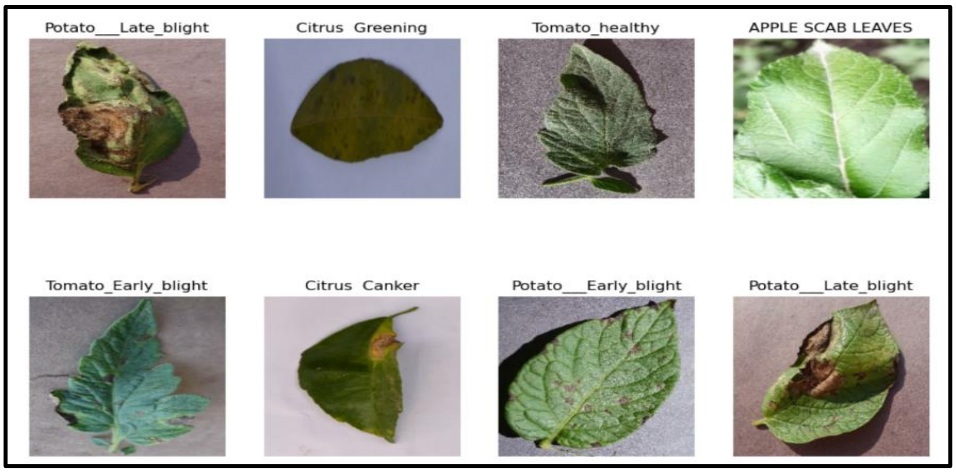


Figure 4.4.1 Feature Extraction

Figure 4.4.2 Obtained Features

4.5 Disease Classification:

- In our proposed work, to identify and categorize plant diseases using CNN.
- CNN is an ANN type based on the principle of hidden layers with several applications in the fields of image processing and recognition.
- The basic framework or flow of our project consists of 5 Acquiring images, processing them, cutting them up into pieces, extracting features from them, and finally classifying them are all stages in the process.

Image acquisition:

- The dataset which is selected to build the model plays a very important role in the result observed from it. In image-oriented datasets, the images must be properly labeled, and it must be made sure that no such images exist which do not have labels. For our CNN model we have made use of the Plant Village dataset.
- This dataset has a collection of 3 directories of healthy and diseased plant leaves of Potato. With a total of 2,147 images, this dataset houses the following directories of plant leaves.

Image Pre-processing:

- The images which are acquired cannot be directly fed into the Convolutional Neural Network. Since, the computer does not understand anything other than binary numbers, the images also must be converted into machine understandable format. To do this, the pictures are changed to an array.
- The images of the dataset are converted into 256 x 256 size arrays where each block represents the pixel value. The acquired images may have noise and other discolorations which are not ideal for image processing.
- All these ambiguities must be handled. This is carried out using the process of image segmentation, where the images are divided into smaller segments to map out the areas of interest and to avoid the areas which are less important, and which may cause disruptions to the accuracy of the system.

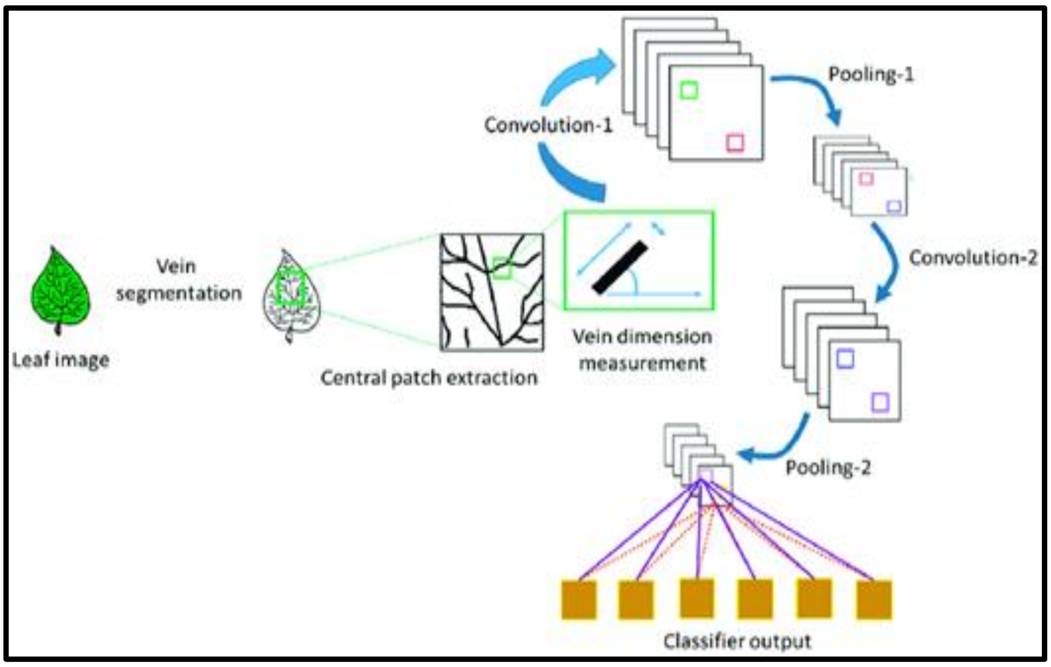


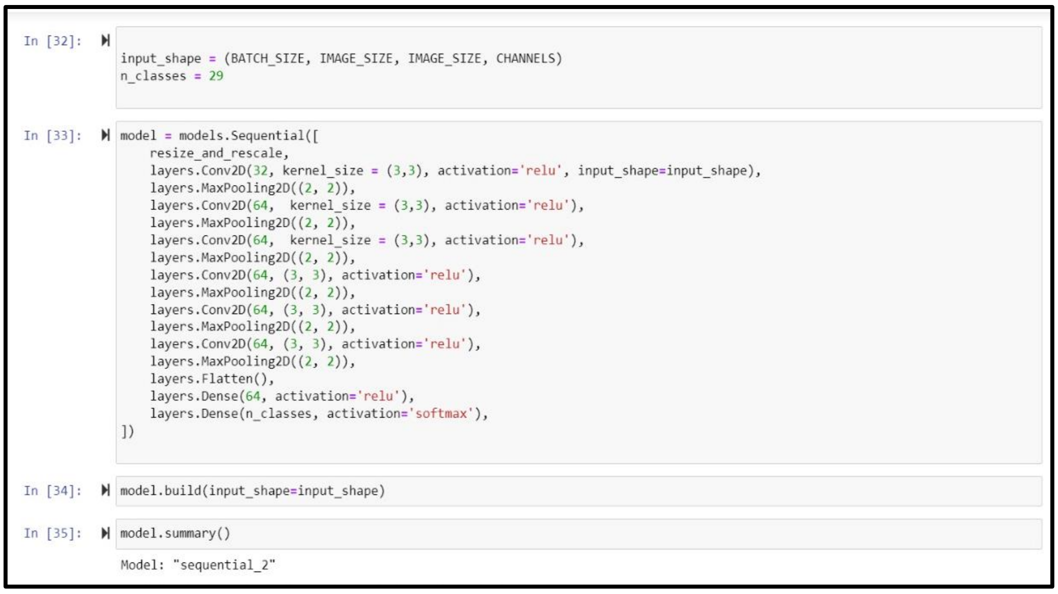
Figure 4.5.1 CNN Working Model

Feature Extraction:

- After the images are processed into proper format and divided into segments, the next feature extraction is a crucial process.
- In Convolutional Neural Networks, this is accomplished with the aid of activation function and then max-pooling layers.
- This method helps in reducing the size of data from the dataset.
- It also helps in building the model with minimal machine effort and improves the learning speed in the machine learning process.

A. Activation Function:

The main purpose of using an activation function in a CNN is to decide which combination of weights and input will fire the next neuron. The activation function used in our CNN model is the Rectified Linear Unit activation function, often known as ReLU. If the input is higher than zero, the function returns the input; otherwise, it returns zero. Where x is an input and y are the output, this can be written as y = max (0, x).

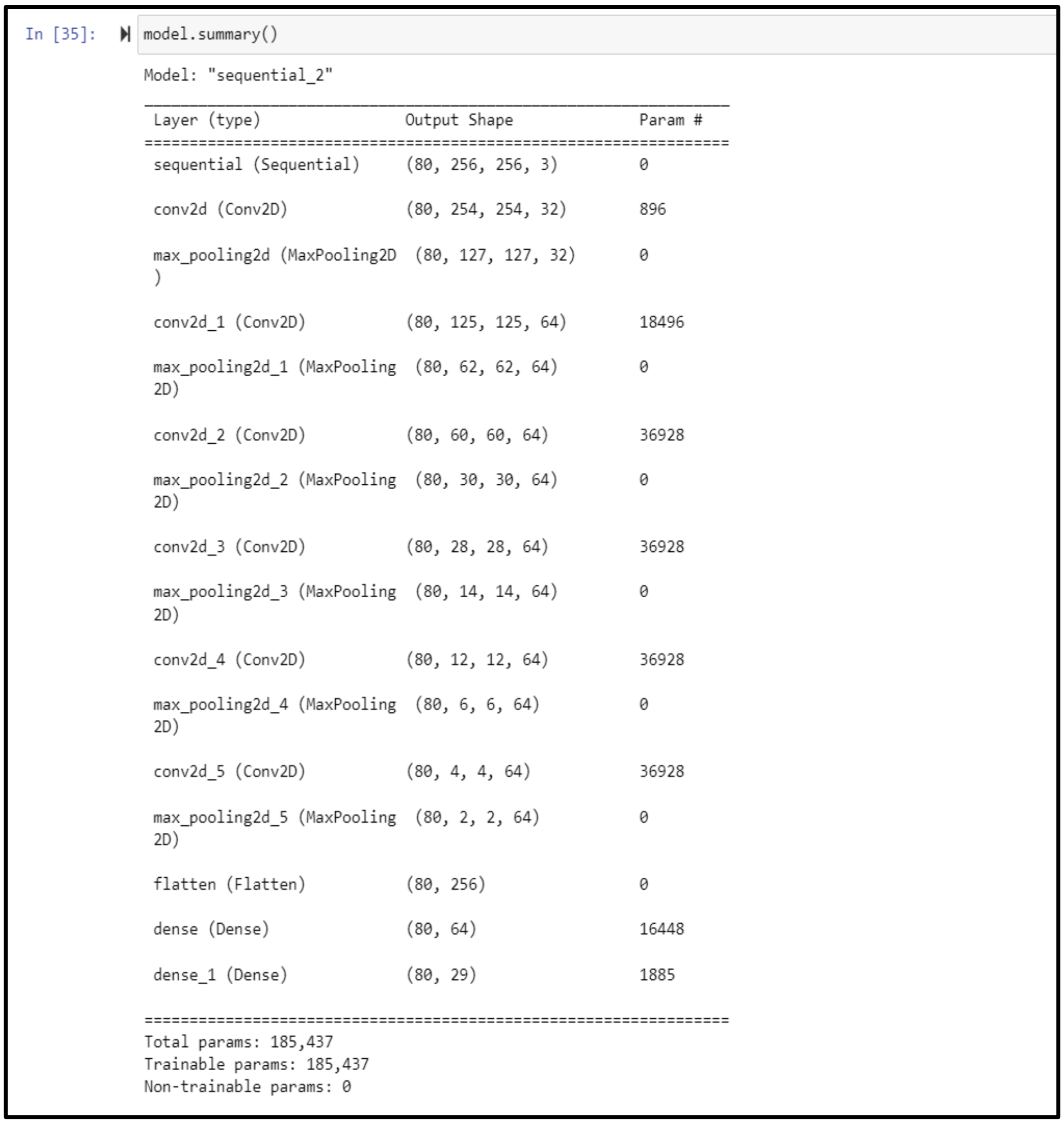


```
In [32]:  input_shape = (BATCH_SIZE, IMAGE_SIZE, IMAGE_SIZE, CHANNELS)
          n_classes = 29

In [33]:  model = models.Sequential([
              resize_and_rescale,
              layers.Conv2D(32, kernel_size = (3,3), activation='relu', input_shape=input_shape),
              layers.MaxPooling2D((2, 2)),
              layers.Conv2D(64,  kernel_size = (3,3), activation='relu'),
              layers.MaxPooling2D((2, 2)),
              layers.Conv2D(64,  kernel_size = (3,3), activation='relu'),
              layers.MaxPooling2D((2, 2)),
              layers.Conv2D(64, (3, 3), activation='relu'),
              layers.MaxPooling2D((2, 2)),
              layers.Conv2D(64, (3, 3), activation='relu'),
              layers.MaxPooling2D((2, 2)),
              layers.Conv2D(64, (3, 3), activation='relu'),
              layers.MaxPooling2D((2, 2)),
              layers.Flatten(),
              layers.Dense(64, activation='relu'),
              layers.Dense(n_classes, activation='softmax'),
          ])

In [34]:  model.build(input_shape=input_shape)

In [35]:  model.summary()

          Model: "sequential_2"
```

Figure 4.5.2 Activation function

B. Multi-Maximum Pooling-Layer Structures:

All CNNs have pooling layers after their activation function layers. Pooling layers are used to reduce the dimensions of feature maps. They are 3 Different pooling methods including averaging, maximizing, and global pooling. We have made use of max-pooling technique because it enhances the features of the images by taking the maximum of the values present in that region. At the end of this operation, we obtain the most prominent features of the image. This step helps in avoiding overfitting and reduces noise distortion. Other than these 2 techniques used for feature extraction of an image, a dropout layer is used to reduce overfitting of the model, where the neurons are dropped at random. Flatten layer is used to convert the output into one long feature vector which is given as input into the next layer.



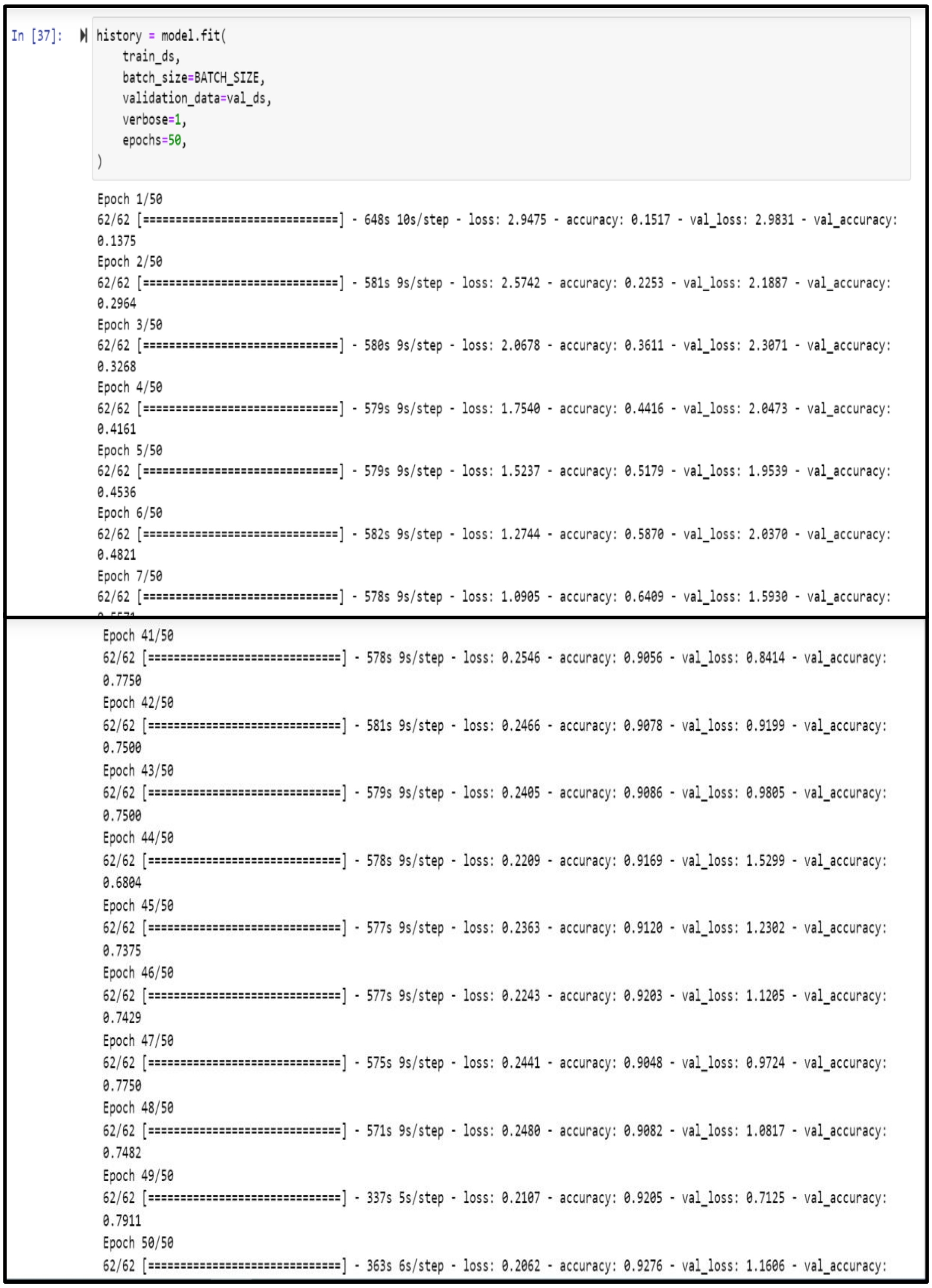
```
In [35]:   model.summary()

           Model: "sequential_2"
           _________________________________________________________________
            Layer (type)                Output Shape              Param #
           =================================================================
            sequential (Sequential)     (80, 256, 256, 3)         0

            conv2d (Conv2D)             (80, 254, 254, 32)        896

            max_pooling2d (MaxPooling2D  (80, 127, 127, 32)       0
            )

            conv2d_1 (Conv2D)           (80, 125, 125, 64)        18496

            max_pooling2d_1 (MaxPooling  (80, 62, 62, 64)         0
            2D)

            conv2d_2 (Conv2D)           (80, 60, 60, 64)          36928

            max_pooling2d_2 (MaxPooling  (80, 30, 30, 64)         0
            2D)

            conv2d_3 (Conv2D)           (80, 28, 28, 64)          36928

            max_pooling2d_3 (MaxPooling  (80, 14, 14, 64)         0
            2D)

            conv2d_4 (Conv2D)           (80, 12, 12, 64)          36928

            max_pooling2d_4 (MaxPooling  (80, 6, 6, 64)           0
            2D)

            conv2d_5 (Conv2D)           (80, 4, 4, 64)            36928

            max_pooling2d_5 (MaxPooling  (80, 2, 2, 64)           0
            2D)

            flatten (Flatten)           (80, 256)                 0

            dense (Dense)               (80, 64)                  16448

            dense_1 (Dense)             (80, 29)                  1885

           =================================================================
           Total params: 185,437
           Trainable params: 185,437
           Non-trainable params: 0
```

Figure 4.5.3 Multi-Maximum Pooling-Layer Structures

Classification:

After the layers of the CNN are decided, we compile the model. The dataset is divided into training and test datasets, we have used 10% of the dataset for validation. We have an epoch value of 50, where epoch value may be thought of as the number of times the model iteratively examines the data. A variety of visual attributes are recorded with the use of real-time image processing tools. Once the model is trained, it is tested on the validation dataset.

The model is trained using CNN algorithm with around 6000 images consisting in the dataset which are in format of jpg images and then they are trained and tested and ran over 50 epochs because higher the number pf epochs higher the accuracy. The model gone through the whole process attained an accuracy between 90-93 percentage.



```
In [37]:  history = model.fit(
              train_ds,
              batch_size=BATCH_SIZE,
              validation_data=val_ds,
              verbose=1,
              epochs=50,
          )
```

```
Epoch 1/50
62/62 [==============================] - 648s 10s/step - loss: 2.9475 - accuracy: 0.1517 - val_loss: 2.9831 - val_accuracy:
0.1375
Epoch 2/50
62/62 [==============================] - 581s 9s/step - loss: 2.5742 - accuracy: 0.2253 - val_loss: 2.1887 - val_accuracy:
0.2964
Epoch 3/50
62/62 [==============================] - 580s 9s/step - loss: 2.0678 - accuracy: 0.3611 - val_loss: 2.3071 - val_accuracy:
0.3268
Epoch 4/50
62/62 [==============================] - 579s 9s/step - loss: 1.7540 - accuracy: 0.4416 - val_loss: 2.0473 - val_accuracy:
0.4161
Epoch 5/50
62/62 [==============================] - 579s 9s/step - loss: 1.5237 - accuracy: 0.5179 - val_loss: 1.9539 - val_accuracy:
0.4536
Epoch 6/50
62/62 [==============================] - 582s 9s/step - loss: 1.2744 - accuracy: 0.5870 - val_loss: 2.0370 - val_accuracy:
0.4821
Epoch 7/50
62/62 [==============================] - 578s 9s/step - loss: 1.0905 - accuracy: 0.6409 - val_loss: 1.5930 - val_accuracy:
```

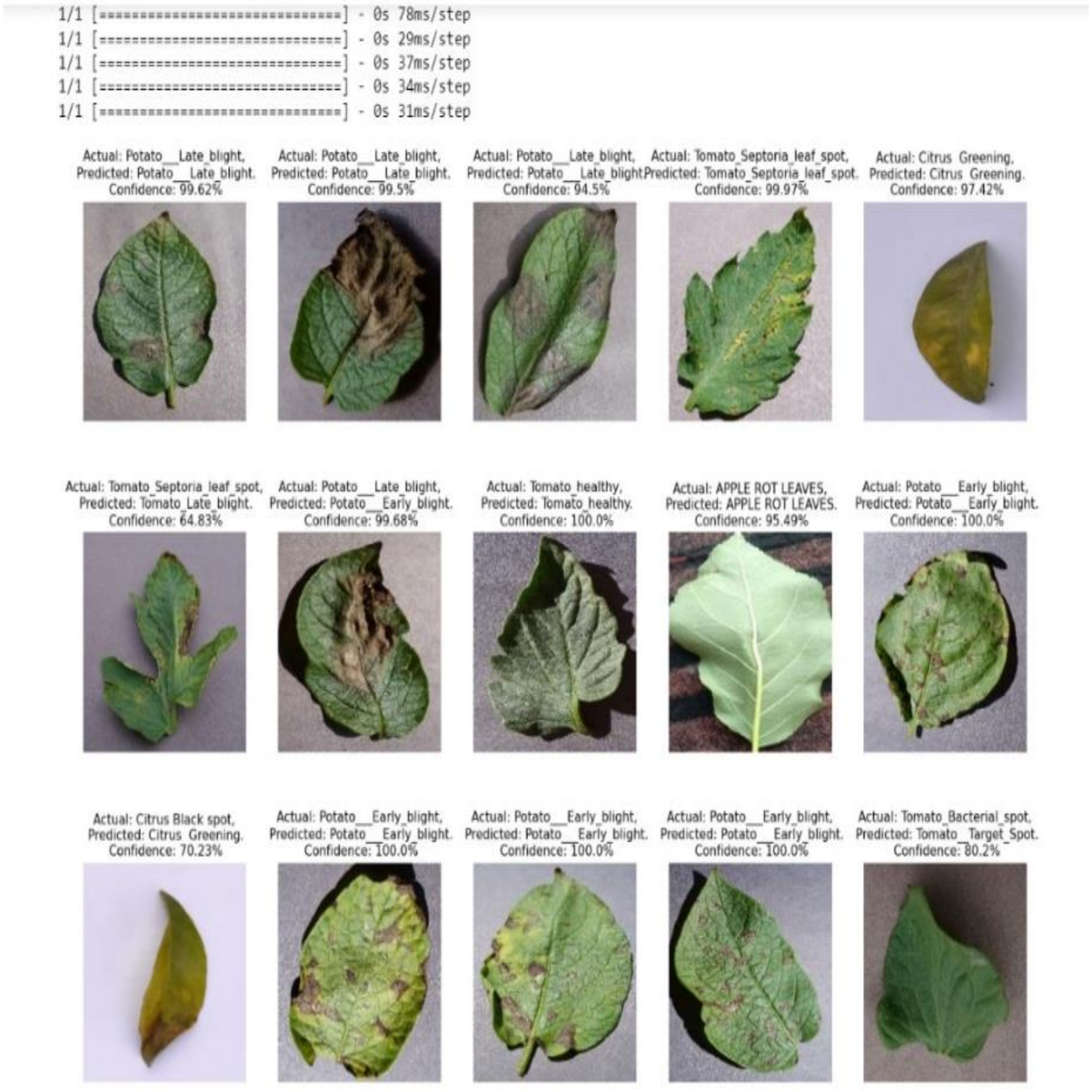
```
Epoch 41/50
62/62 [==============================] - 578s 9s/step - loss: 0.2546 - accuracy: 0.9056 - val_loss: 0.8414 - val_accuracy:
0.7750
Epoch 42/50
62/62 [==============================] - 581s 9s/step - loss: 0.2466 - accuracy: 0.9078 - val_loss: 0.9199 - val_accuracy:
0.7500
Epoch 43/50
62/62 [==============================] - 579s 9s/step - loss: 0.2405 - accuracy: 0.9086 - val_loss: 0.9805 - val_accuracy:
0.7500
Epoch 44/50
62/62 [==============================] - 578s 9s/step - loss: 0.2209 - accuracy: 0.9169 - val_loss: 1.5299 - val_accuracy:
0.6804
Epoch 45/50
62/62 [==============================] - 577s 9s/step - loss: 0.2363 - accuracy: 0.9120 - val_loss: 1.2302 - val_accuracy:
0.7375
Epoch 46/50
62/62 [==============================] - 577s 9s/step - loss: 0.2243 - accuracy: 0.9203 - val_loss: 1.1205 - val_accuracy:
0.7429
Epoch 47/50
62/62 [==============================] - 575s 9s/step - loss: 0.2441 - accuracy: 0.9048 - val_loss: 0.9724 - val_accuracy:
0.7750
Epoch 48/50
62/62 [==============================] - 571s 9s/step - loss: 0.2480 - accuracy: 0.9082 - val_loss: 1.0817 - val_accuracy:
0.7482
Epoch 49/50
62/62 [==============================] - 337s 5s/step - loss: 0.2107 - accuracy: 0.9205 - val_loss: 0.7125 - val_accuracy:
0.7911
Epoch 50/50
62/62 [==============================] - 363s 6s/step - loss: 0.2062 - accuracy: 0.9276 - val_loss: 1.1606 - val_accuracy:
```

Figure 4.5.4 Epochs

Testing:

1. Based on the results observed after implementing our Convolutional Neural Network model, it can be safe to say that our model is able to identify and classify the various plant diseases present in the dataset with good accuracy.
2. The accuracy of the built model is observed to be 96.83%

```
scores = model. evaluate(test_ds)

9/9 [==============================] - 56s 1s/step - loss: 1.4111 - accuracy: 0.9292
```



```
1/1 [==============================] - 0s 78ms/step
1/1 [==============================] - 0s 29ms/step
1/1 [==============================] - 0s 37ms/step
1/1 [==============================] - 0s 34ms/step
1/1 [==============================] - 0s 31ms/step
```

Actual: Potato__Late_blight, Predicted: Potato__Late_blight. Confidence: 99.62%
Actual: Potato__Late_blight, Predicted: Potato__Late_blight. Confidence: 99.5%
Actual: Potato__Late_blight, Predicted: Potato__Late_blight Confidence: 94.5%
Actual: Tomato_Septoria_leaf_spot, Predicted: Tomato_Septoria_leaf_spot. Confidence: 99.97%
Actual: Citrus Greening, Predicted: Citrus Greening. Confidence: 97.42%
Actual: Tomato_Septoria_leaf_spot, Predicted: Tomato_Late_blight. Confidence: 64.83%
Actual: Potato__Late_blight, Predicted: Potato__Early_blight. Confidence: 99.68%
Actual: Tomato_healthy, Predicted: Tomato_healthy. Confidence: 100.0%
Actual: APPLE ROT LEAVES, Predicted: APPLE ROT LEAVES. Confidence: 95.49%
Actual: Potato__Early_blight, Predicted: Potato__Early_blight. Confidence: 100.0%
Actual: Citrus Black spot, Predicted: Citrus Greening. Confidence: 70.23%
Actual: Potato__Early_blight, Predicted: Potato__Early_blight. Confidence: 100.0%
Actual: Potato__Early_blight, Predicted: Potato__Early_blight. Confidence: 100.0%
Actual: Potato__Early_blight, Predicted: Potato__Early_blight. Confidence: 100.0%
Actual: Tomato_Bacterial_spot, Predicted: Tomato_Target_Spot. Confidence: 80.2%

Figure 4.5.5 Final Output

CHAPTER 5

RESULTS AND DISCUSSIONS

5.1 Experimental Setup

The experimental setup for plant disease classification using deep learning typically involves the following steps:

- **Data collection:** The first step is to collect a dataset of images of diseased and healthy plant leaves. The dataset should be diverse and representative of the types of diseases that the model will be trained to recognize.

- **Data preprocessing**: The images in the dataset need to be preprocessed before training the model. This may involve resizing the images, normalizing the pixel values, and augmenting the dataset with variations of the images.

- **Model selection:** The next step is to select an appropriate deep learning model architecture for the task. This may involve experimenting with different models or using a pre-trained model and fine-tuning it for the specific task.

- **Model training:** The model is trained on the preprocessed dataset using an appropriate loss function and optimization algorithm. The number of epochs and batch size are also set during this step.

- **Model evaluation:** Once the model is trained, it is evaluated on a validation dataset to measure its performance. The metrics used for evaluation may include accuracy, precision, recall, F1-score, etc.

- **Model testing:** Finally, the model is tested on a separate, unseen test dataset to assess its generalization performance. The performance of the model on the test dataset is used to determine its effectiveness in classifying plant diseases.

5.2 Performance and Analysis

Accuracy: Accuracy is calculated as the ratio of the number of correctly classified samples to the total number of samples in the dataset.

Accuracy = (TP + TN) / (TP + TN + FP + FN) (Eq 5.2.1)

where TP (true positives) is the number of correctly classified samples from the positive class, TN (true negatives) is the number of correctly classified samples from the negative class, FP (false positives) is the number of samples incorrectly classified as positive, and FN (false negatives) is the number of samples incorrectly classified as negative.

Precision: Precision is calculated as the ratio of true positives to the total number of samples predicted as positive.

Precision = TP / (TP + FP) (Eq 5.2.2)

Recall: Recall is calculated as the ratio of true positives to the total number of samples that belong to the positive class.

Recall = TP / (TP + FN) (Eq 5.2.3)

F1-score: F1-score is the harmonic mean of precision and recall and provides a balanced measure of the model's performance.

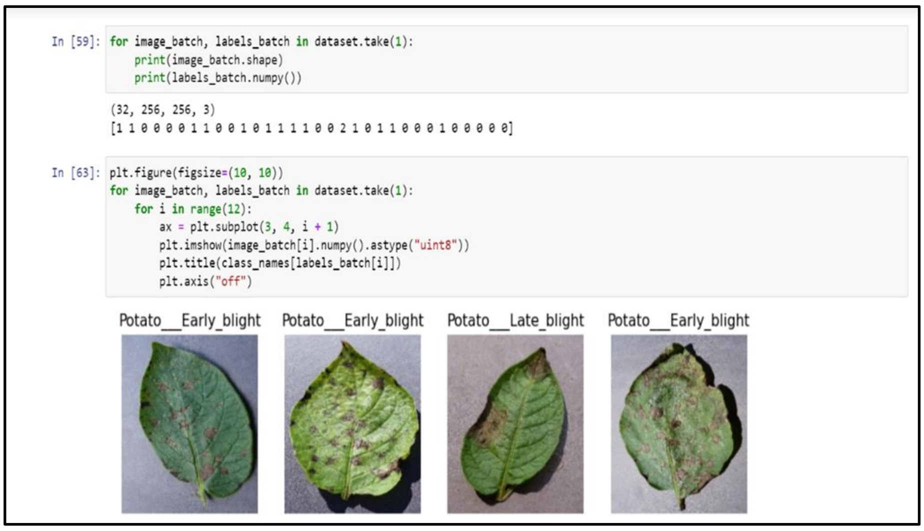
F1-score = 2 * (Precision * Recall) / (Precision + Recall) (Eq 5.2.4)

Note that the formulas above assume a binary classification problem, where the model is trained to classify samples as either healthy or diseased. For multi-class classification problems, the formulas may need to be adapted accordingly.

In order to determine the effectiveness of our plant disease identification system, we utilized a range of performance metrics, including accuracy, precision, and loss. By testing the system over numerous epochs, we were able to track the accuracy of the model as it improved over time.

To measure accuracy, we examined how effectively our system was able to identify healthy plants and exclude them from disease diagnosis. Similarly, we used precision to determine how many diseased plants were correctly identified. Finally, recall provided a weighted average of both metrics, giving us a comprehensive understanding of our system's performance.

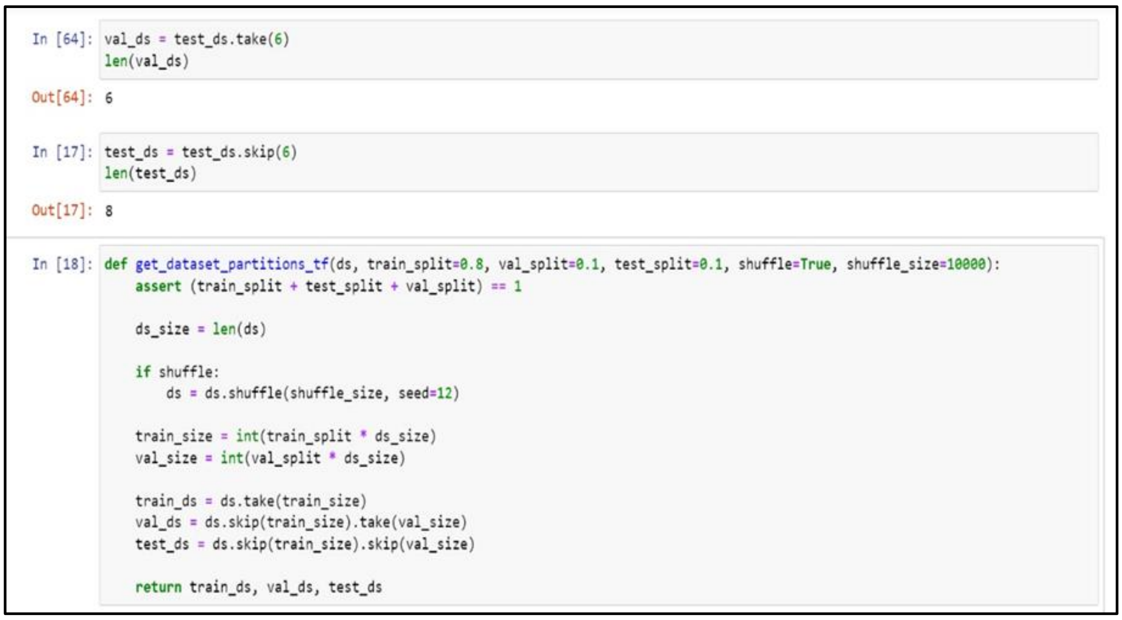
Ultimately, farmers receive a detailed report outlining the disease stage of their plants based on the test results. This information is crucial, as it has better understandable progression of the disease and take required/necessary action. To carry out these experiments, we utilized a range of environments, tools, and libraries to ensure the accuracy and reliability of our system. After compiling our model and training it on the train data with an epoch value of 50 and then plotting the accuracies of the training and validation accuracy's and losses, the following graphs can be observed. From these graphs, it can be concluded that at the end of the 50 epochs, both the accuracy and loss of the training and validation dataset tend to converge at one point. Some libraries will be importing for our better usage and here importing tensor flow to make machine learning and developing neural networks faster and easier. We are using 50 of Epochs and considering the plant village dataset. Next, we will be giving the resolution of the images in datasets. The resolution is given in the form of pixels and our model will be detecting the three different types of diseases of potato. The image here appears in the size according to our given resolution dimensions.



```
In [59]: for image_batch, labels_batch in dataset.take(1):
             print(image_batch.shape)
             print(labels_batch.numpy())

         (32, 256, 256, 3)
         [1 1 0 0 0 0 1 1 0 0 1 0 1 1 1 1 0 0 2 1 0 1 1 0 0 0 1 0 0 0 0 0]

In [63]: plt.figure(figsize=(10, 10))
         for image_batch, labels_batch in dataset.take(1):
             for i in range(12):
                 ax = plt.subplot(3, 4, i + 1)
                 plt.imshow(image_batch[i].numpy().astype("uint8"))
                 plt.title(class_names[labels_batch[i]])
                 plt.axis("off")
```

Figure.5.2.1 Importing Dataset

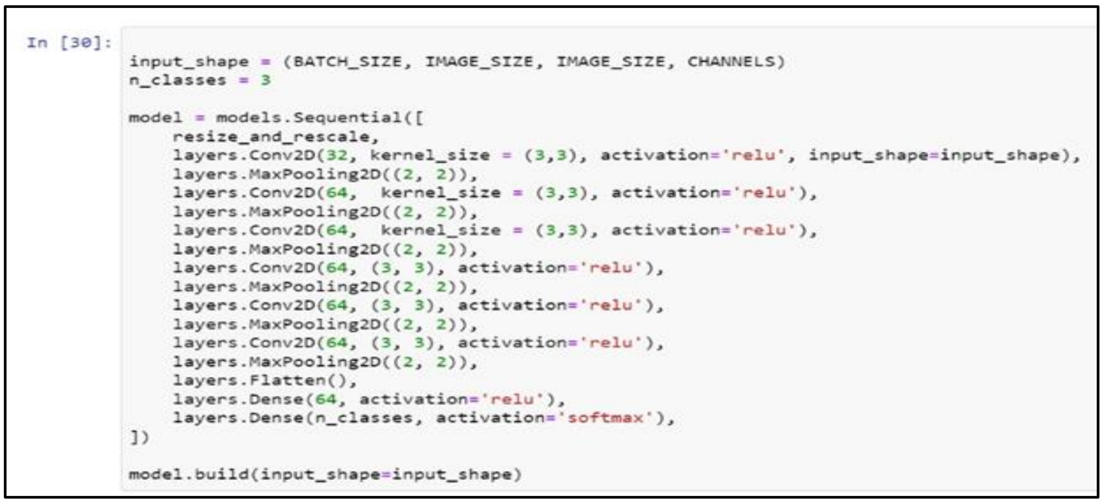
This is the total number of images of the dataset which we can get from the len(dataset). From that 80 percent of the images will be used for training the model .so, train_size = 0.8 will be assigning 80 percent of images from total dataset to the Training Phase. The remaining 20 percent of the images will be assigned for the testing phase.



```
In [64]: val_ds = test_ds.take(6)
         len(val_ds)

Out[64]: 6

In [17]: test_ds = test_ds.skip(6)
         len(test_ds)

Out[17]: 8

In [18]: def get_dataset_partitions_tf(ds, train_split=0.8, val_split=0.1, test_split=0.1, shuffle=True, shuffle_size=10000):
             assert (train_split + test_split + val_split) == 1

             ds_size = len(ds)

             if shuffle:
                 ds = ds.shuffle(shuffle_size, seed=12)

             train_size = int(train_split * ds_size)
             val_size = int(val_split * ds_size)

             train_ds = ds.take(train_size)
             val_ds = ds.skip(train_size).take(val_size)
             test_ds = ds.skip(train_size).skip(val_size)

             return train_ds, val_ds, test_ds
```

Figure.5.2.2 Dataset Partition

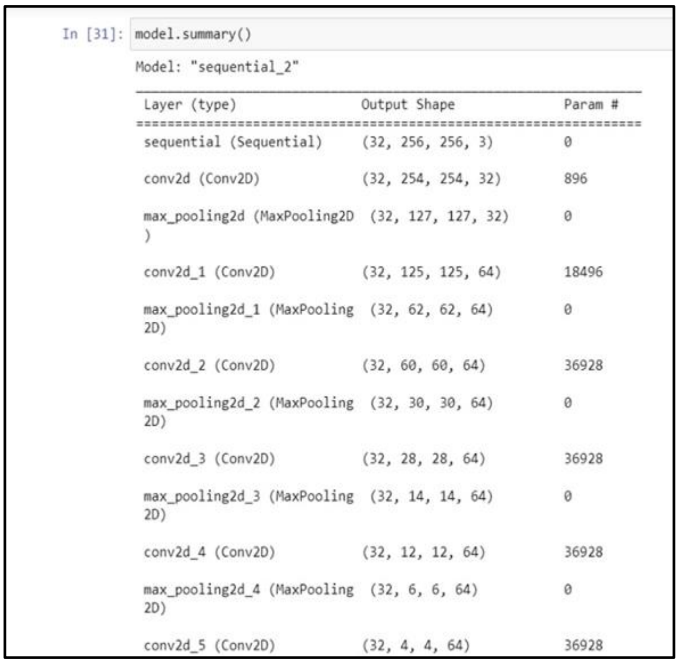
As shown in the figure 5.2.2 the code line 26 will be training the model with the training dataset.



```
In [30]:
         input_shape = (BATCH_SIZE, IMAGE_SIZE, IMAGE_SIZE, CHANNELS)
         n_classes = 3

         model = models.Sequential([
             resize_and_rescale,
             layers.Conv2D(32, kernel_size = (3,3), activation='relu', input_shape=input_shape),
             layers.MaxPooling2D((2, 2)),
             layers.Conv2D(64,  kernel_size = (3,3), activation='relu'),
             layers.MaxPooling2D((2, 2)),
             layers.Conv2D(64,  kernel_size = (3,3), activation='relu'),
             layers.MaxPooling2D((2, 2)),
             layers.Conv2D(64, (3, 3), activation='relu'),
             layers.MaxPooling2D((2, 2)),
             layers.Conv2D(64, (3, 3), activation='relu'),
             layers.MaxPooling2D((2, 2)),
             layers.Conv2D(64, (3, 3), activation='relu'),
             layers.MaxPooling2D((2, 2)),
             layers.Flatten(),
             layers.Dense(64, activation='relu'),
             layers.Dense(n_classes, activation='softmax'),
         ])

         model.build(input_shape=input_shape)
```

Figure 5.2.3 Max Pooling and Activation functions

Model Summary () is a method in the Keras deep learning library that prints a summary of the neural network model. The summary includes information about the layers in the model, such as the type of layer, the output shape, the number of parameters, and the number of trainable parameters. The summary can be useful for checking the structure of the neural network and making sure it matches the intended design. It can also help to identify any issues with the network, such as overfitting or underfitting, by examining the number of parameters and the complexity of the model.



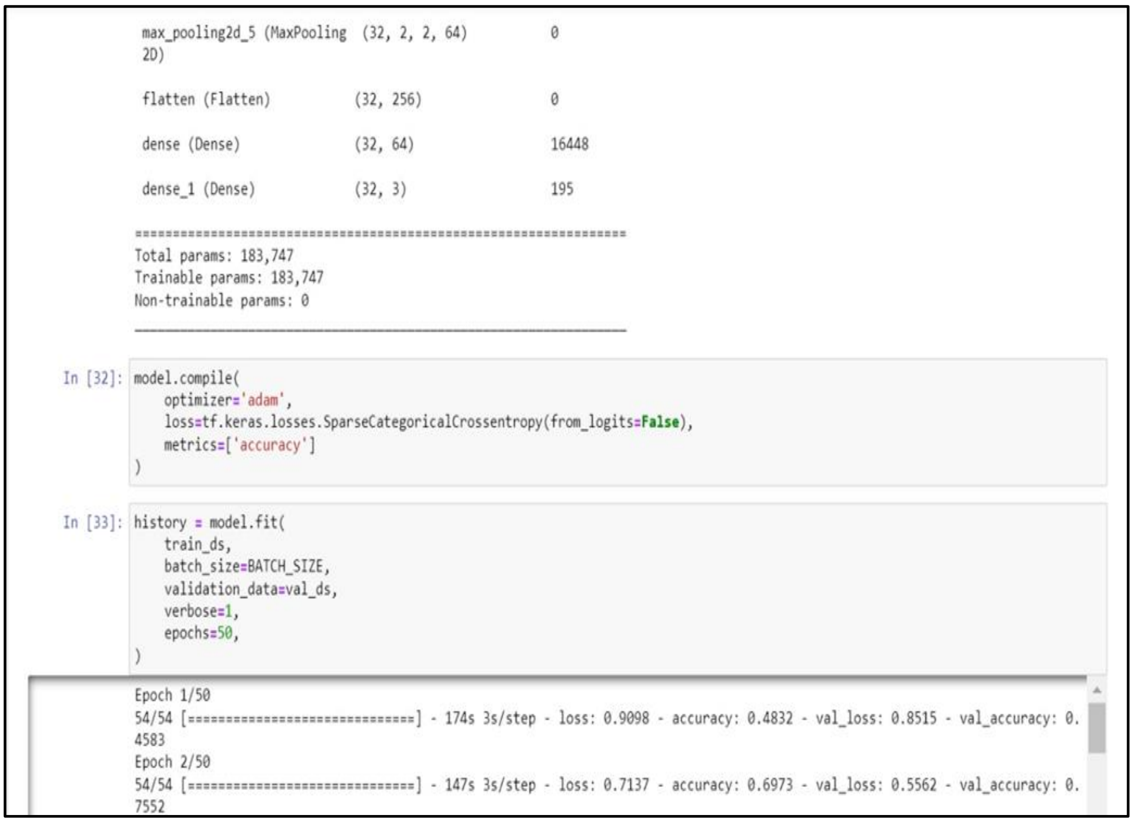
```
In [31]: model.summary()

Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 sequential (Sequential)     (32, 256, 256, 3)         0

 conv2d (Conv2D)             (32, 254, 254, 32)        896

 max_pooling2d (MaxPooling2D  (32, 127, 127, 32)       0
 )

 conv2d_1 (Conv2D)           (32, 125, 125, 64)        18496

 max_pooling2d_1 (MaxPooling  (32, 62, 62, 64)         0
 2D)

 conv2d_2 (Conv2D)           (32, 60, 60, 64)          36928

 max_pooling2d_2 (MaxPooling  (32, 30, 30, 64)         0
 2D)

 conv2d_3 (Conv2D)           (32, 28, 28, 64)          36928

 max_pooling2d_3 (MaxPooling  (32, 14, 14, 64)         0
 2D)

 conv2d_4 (Conv2D)           (32, 12, 12, 64)          36928

 max_pooling2d_4 (MaxPooling  (32, 6, 6, 64)           0
 2D)

 conv2d_5 (Conv2D)           (32, 4, 4, 64)            36928
```

Figure 5.2.4 Summary

refers to the number of times that a machine learning algorithm, such as neural networks, will be trained on the entire training dataset. Each epoch consists of multiple iterations, or steps, where the algorithm updates the model's parameters based on the loss function and the optimization algorithm.

In CNNs (Convolutional Neural Networks), epochs refer to the number of times the entire training dataset is passed through the neural network during the training process. In each epoch, the CNN receives the training dataset as input, processes it through its layers, and updates its weights based on the error between the predicted outputs and the actual outputs.

The number of epochs used during training is an important hyperparameter that can significantly affect the performance of the CNN. If the number of epochs is too low, the CNN may not have enough opportunities to learn the underlying patterns in the data and may underfit the training dataset. On the other hand, if the number of epochs is too high, the CNN may overfit the training dataset, meaning it becomes too specialized to the training data and may not generalize well to new, unseen data. Typically, the optimal number of epochs for a CNN is determined through experimentation and tuning. The training process can be monitored through metrics such as accuracy and loss, which can help determine when the model has converged, and further training is unlikely to improve its performance. In practice, it is common to use techniques such as early stopping or learning rate decay to prevent overfitting and improve the generalization performance of the CNN.



```
 max_pooling2d_5 (MaxPooling  (32, 2, 2, 64)         0
 2D)

 flatten (Flatten)           (32, 256)               0

 dense (Dense)               (32, 64)                16448

 dense_1 (Dense)             (32, 3)                 195

=================================================================
Total params: 183,747
Trainable params: 183,747
Non-trainable params: 0
_________________________________________________________________
```

```
In [32]: model.compile(
             optimizer='adam',
             loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=False),
             metrics=['accuracy']
         )

In [33]: history = model.fit(
             train_ds,
             batch_size=BATCH_SIZE,
             validation_data=val_ds,
             verbose=1,
             epochs=50,
         )
```

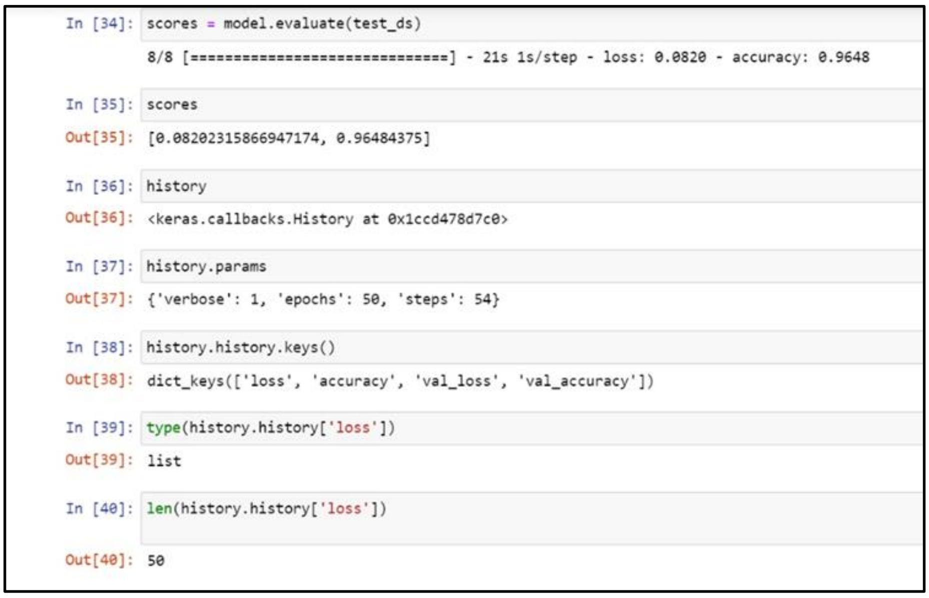
```
Epoch 1/50
54/54 [==============================] - 174s 3s/step - loss: 0.9098 - accuracy: 0.4832 - val_loss: 0.8515 - val_accuracy: 0.
4583
Epoch 2/50
54/54 [==============================] - 147s 3s/step - loss: 0.7137 - accuracy: 0.6973 - val_loss: 0.5562 - val_accuracy: 0.
7552
```

Figure 5.2.5 Epochs

In the realm of Convolutional Neural Networks (CNNs), the evaluation score plays a crucial role in determining the effectiveness of the model in identifying and classifying samples in a test dataset. This score is usually expressed as a single numerical value that summarizes the overall performance of the model with respect to its ability to accurately classify the test samples. The most used evaluation scores in CNNs include accuracy, precision, recall, F1-score, and area under the ROC curve (AUC). These metrics offer varying perspectives on the performance of the model and are often used in different contexts and applications. Ultimately, the evaluation score serves as a crucial indicator of the effectiveness of a CNN in identifying and classifying samples in a test dataset and should be given careful consideration during the development and fine-tuning of the model.

- The model is trained using CNN algorithm and KNN algorithm with around 2147 images complete dataset.
- Below is the graphical representation of the accuracy of detecting diseases using CNN and KNN algorithms.
- As we use a split model the accuracy may vary but the maximum value recorded is 98.83 percent.

After training the model we will be testing the model with the testing dataset and the output shows an actual label which is an actual image which says whether it is diseased or not and shows predicted label which is the actual output the model detects.



```
In [34]: scores = model.evaluate(test_ds)
         8/8 [==============================] - 21s 1s/step - loss: 0.0820 - accuracy: 0.9648

In [35]: scores
Out[35]: [0.08202315866947174, 0.96484375]

In [36]: history
Out[36]: <keras.callbacks.History at 0x1ccd478d7c0>

In [37]: history.params
Out[37]: {'verbose': 1, 'epochs': 50, 'steps': 54}

In [38]: history.history.keys()
Out[38]: dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])

In [39]: type(history.history['loss'])
Out[39]: list

In [40]: len(history.history['loss'])

Out[40]: 50
```

Figure 5.2.6 Evaluation

5.3 Graphs

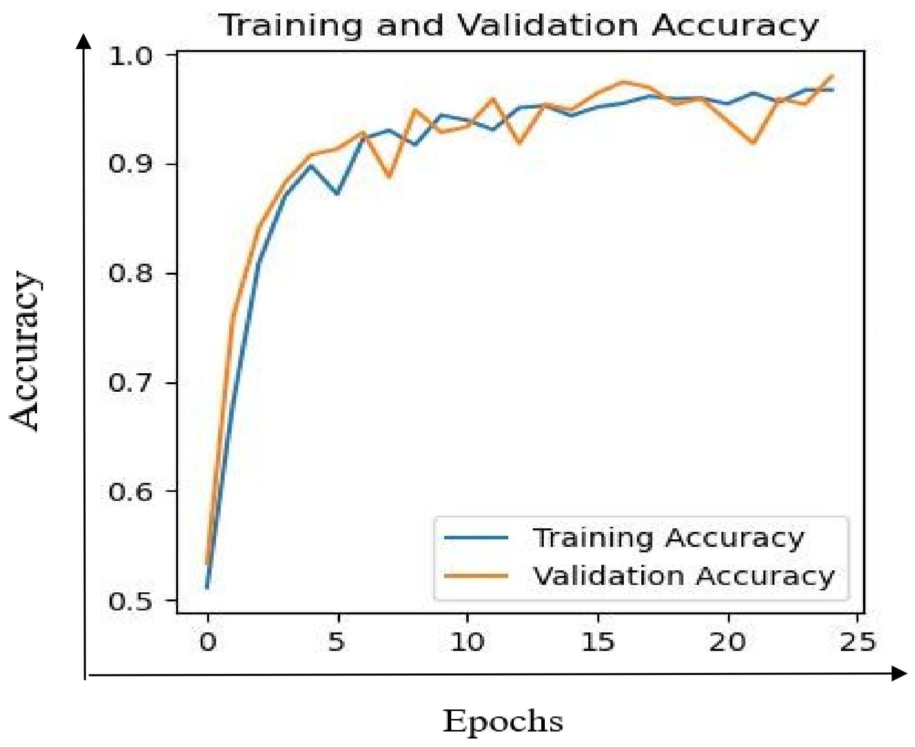
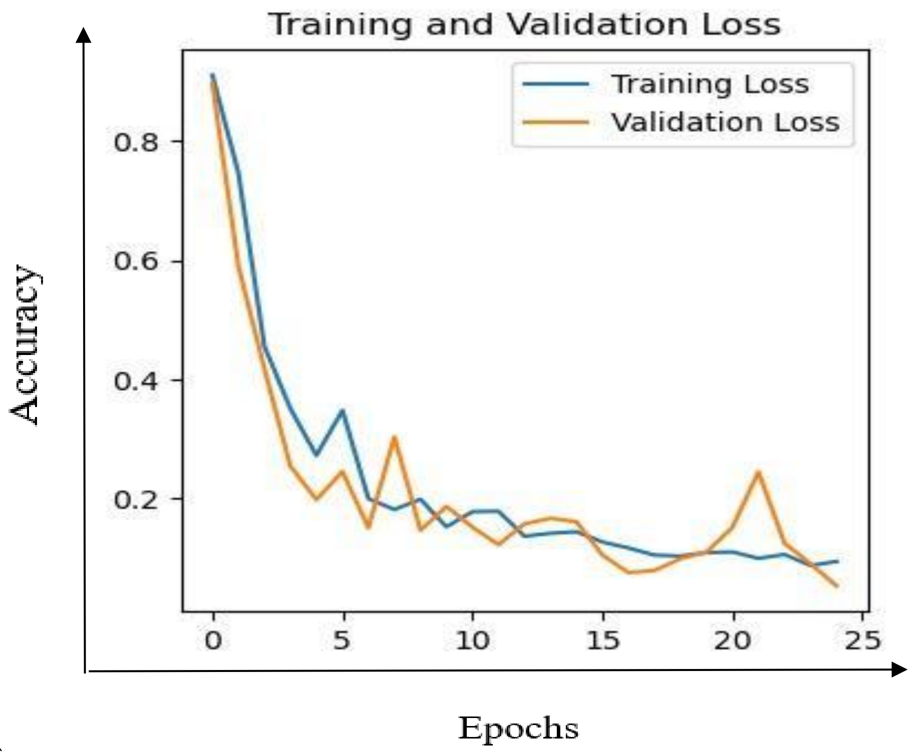


Figure 5.3.1 Training Accuracy vs Validation Accuracy

The training vs validation accuracy graph provides a clear and insightful view of how well a model is performing during the training and validation stages. It is a dynamic representation that showcases the accuracy of the model as it trains on the training dataset and evaluates on the validation dataset.



`

Figure 5.3.2 Training Loss vs Validation Loss

The training vs validation loss graph provides a clear and insightful view of how much loss occurs to the model during the training and validation stages. It is a dynamic representation that showcases the loss of the model as it trains on the training dataset and evaluate on the validation dataset.

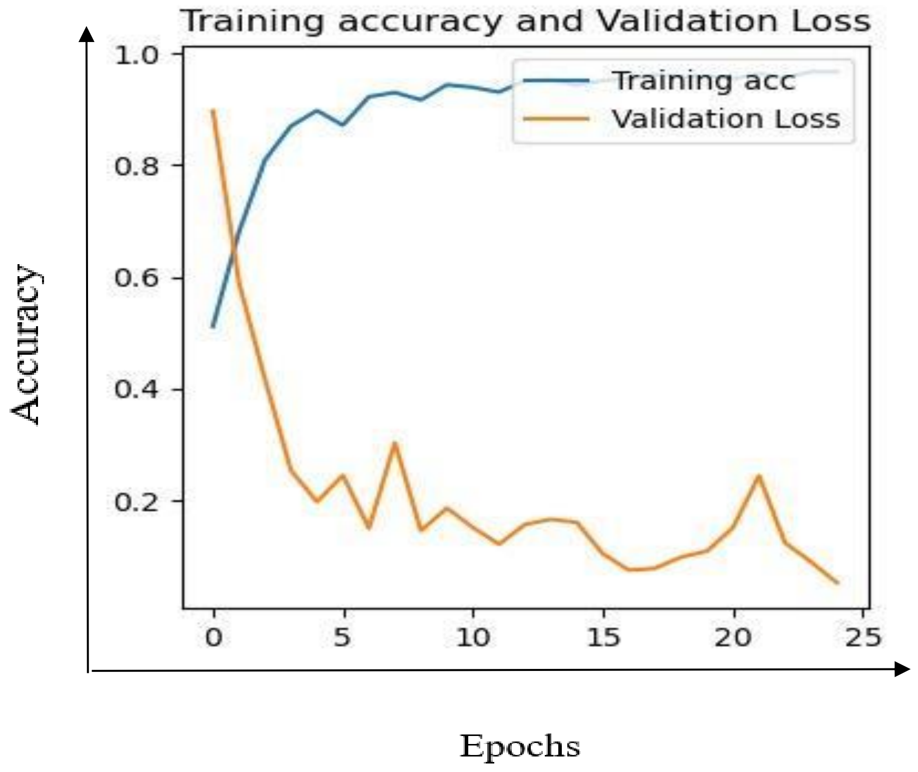


Figure 5.3.3 Training Accuracy vs Validation Loss

The training accuracy vs validation loss graph provides a clear and insightful view of how much loss occurs to the model during the validation stage and how much accuracy during the training stage. It is a dynamic representation that showcases the loss and accuracy of the model as it trains on the training dataset and evaluate on the validation dataset.

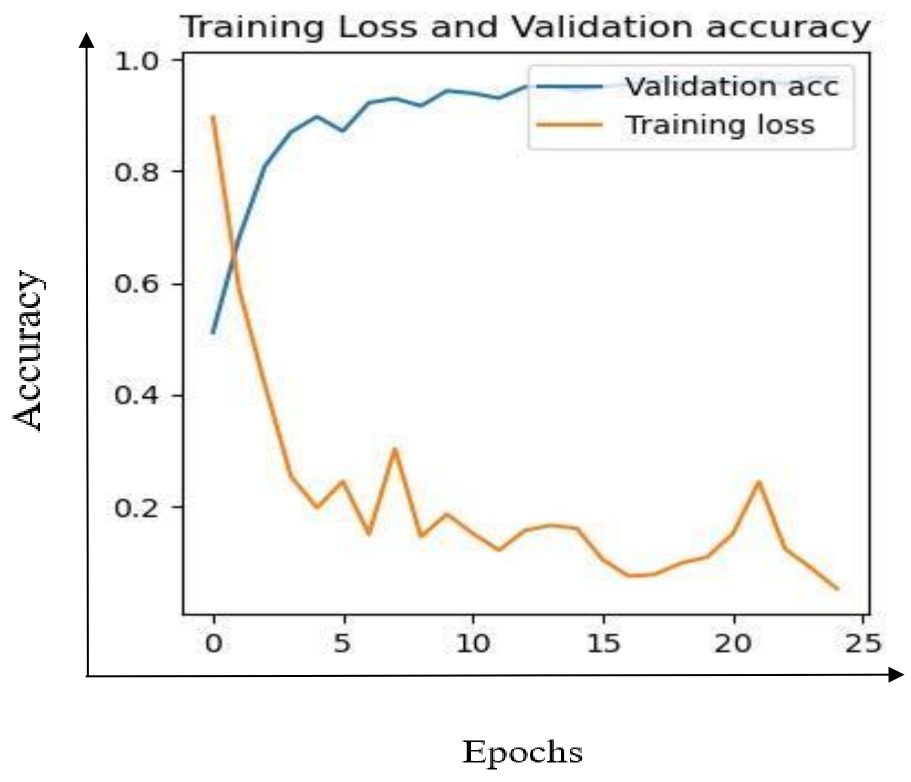


Figure 5.3.4 Training Loss vs Validation Accuracy

The training accuracy vs validation loss graph provides a clear and insightful view of how much loss occurs to the model during the validation stage and how much accuracy during the training stage. It is a dynamic representation that showcases the loss and accuracy of the model as it trains on the training dataset and evaluate on the validation dataset.

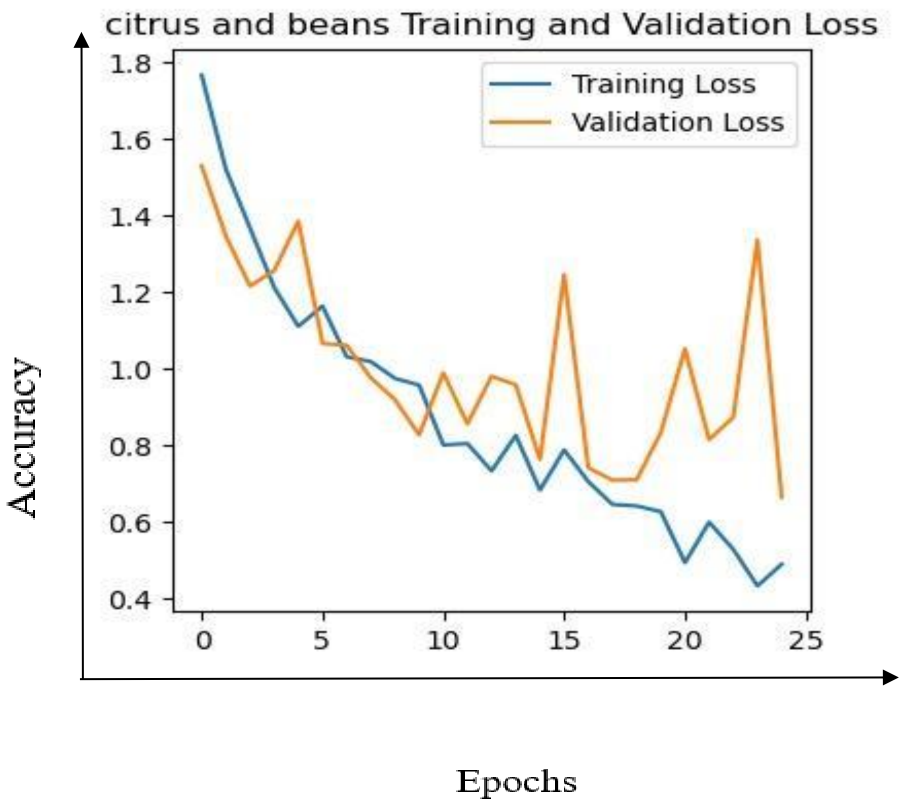


Figure 5.3.5 Training Loss vs Validation Loss

The training vs validation loss graph provides a clear and insightful view of how much loss occurs to the model during the training and validation stages. It is a dynamic representation that showcases the loss of the model as it trains on the training dataset and evaluate on the validation dataset.

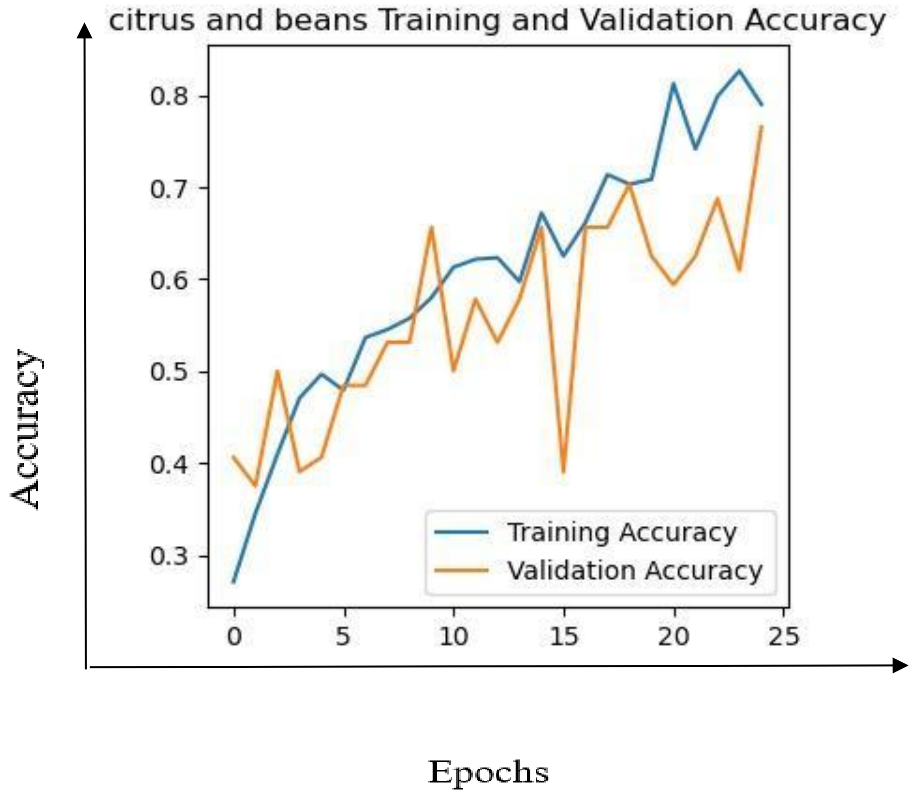


Figure 5.3.6 Training Accuracy vs Validation Accuracy

The training vs validation accuracy graph provides a clear and insightful view of how well a model is performing during the training and validation stages. It is a dynamic representation that showcases the accuracy of the model as it trains on the training dataset and evaluates on the validation dataset.

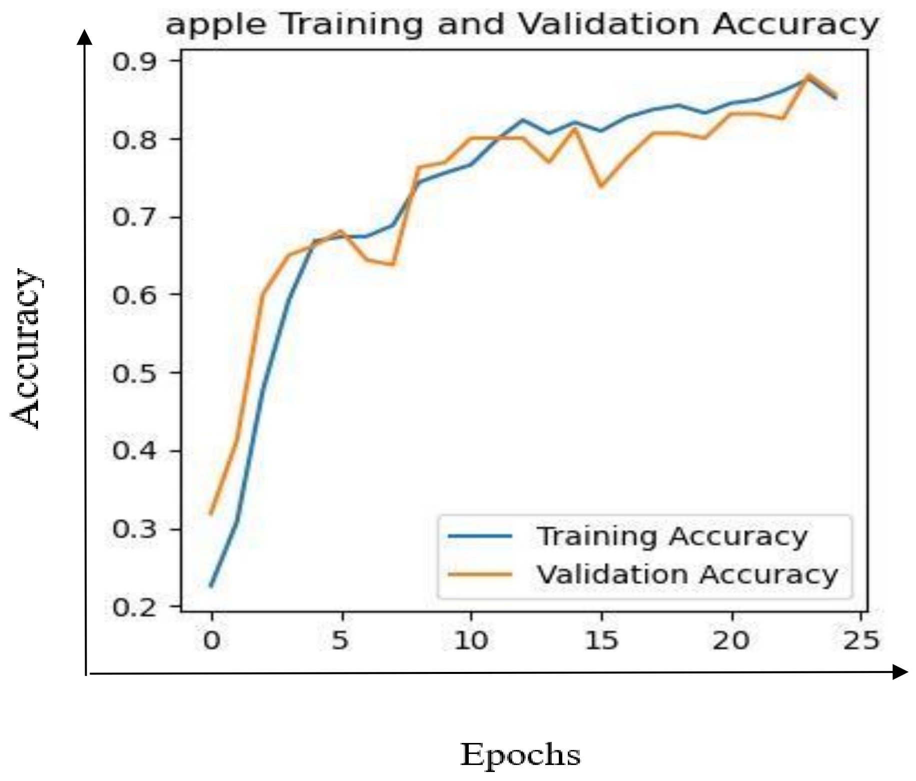


Figure 5.3.7 Training Accuracy vs Validation Accuracy

The training vs validation accuracy graph provides a clear and insightful view of how well a model is performing during the training and validation stages. It is a dynamic representation that showcases the accuracy of the model as it trains on the training dataset and evaluates on the validation dataset.

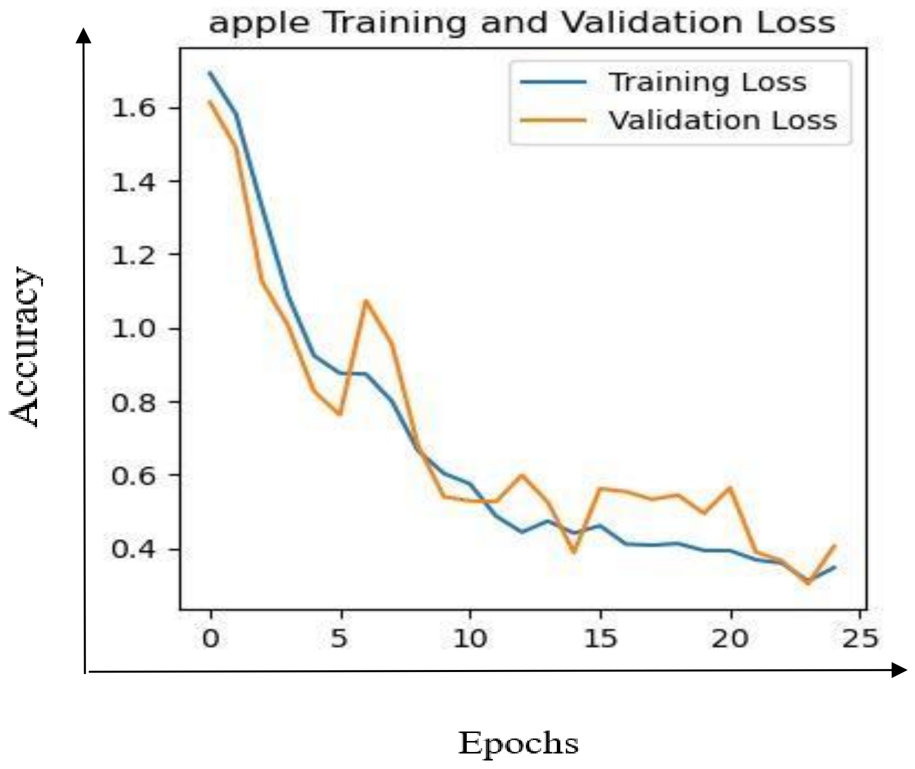


Figure 5.3.8 Training Loss vs Validation Loss

The training vs validation loss graph provides a clear and insightful view of how much loss occurs to the model during the training and validation stages. It is a dynamic representation that showcases the loss of the model as it trains on the training dataset and evaluate on the validation dataset.

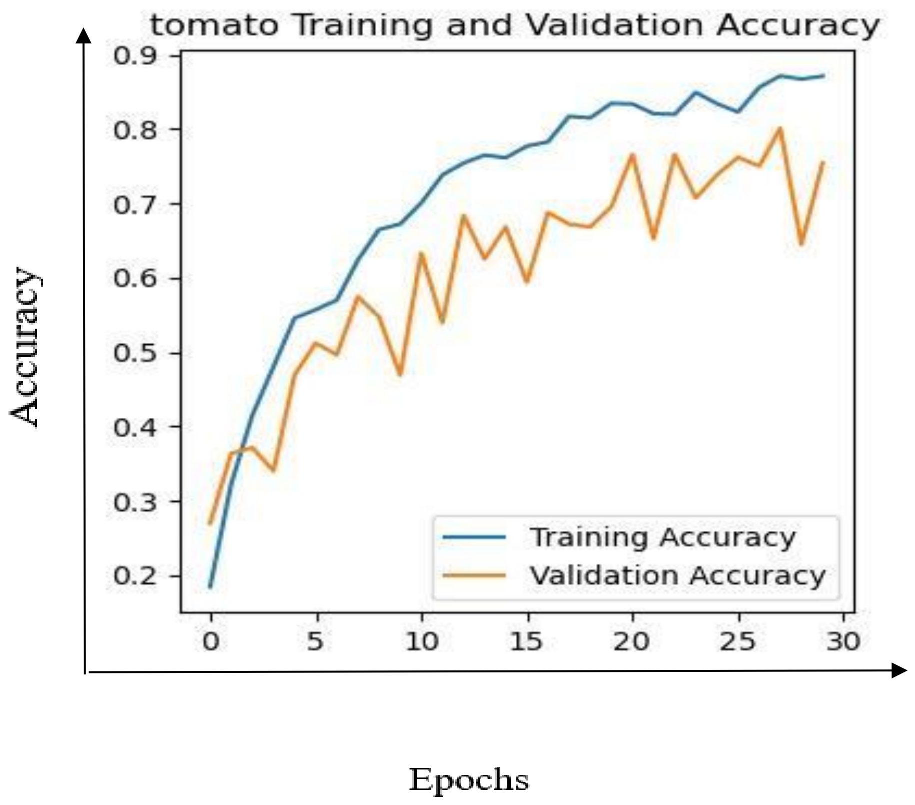


Figure 5.3.9 Training Accuracy vs Validation Accuracy

The training vs validation accuracy graph provides a clear and insightful view of how well a model is performing during the training and validation stages. It is a dynamic representation that showcases the accuracy of the model as it trains on the training dataset and evaluates on the validation dataset.

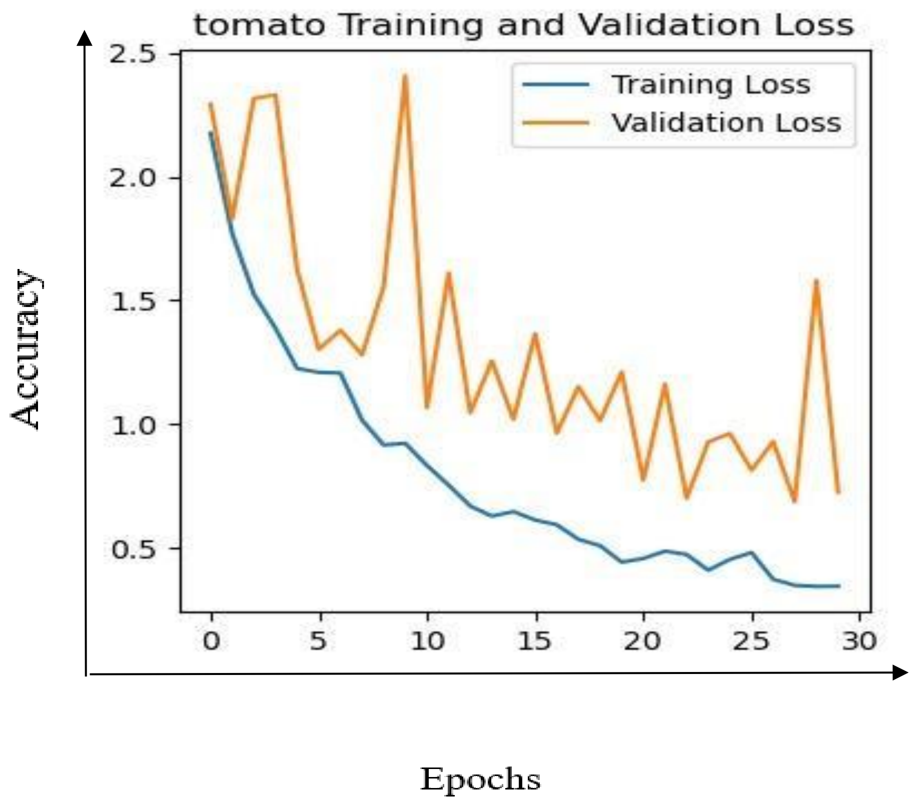


Figure 5.3.10 Training Loss vs Validation Loss

The training vs validation loss graph provides a clear and insightful view of how much loss occurs to the model during the training and validation stages. It is a dynamic representation that showcases the loss of the model as it trains on the training dataset and evaluate on the validation dataset.

CHAPTER 6

CONCLUSION & FUTURE SCOPE

The proposed model uses Convolutional Neural Network (CNN) and employs a variety of computer vision techniques, including RGB to gray conversion to extract informative features of the leaf samples. All these features collectively help in classification of various diseases of plant leaves. This project aims to provide an overview of the field by discussing several approaches used to analyze leaf images for identifying plant diseases. The research employs data from various sources to address significant challenges in applying machine learning and deep learning for diagnosing plant illnesses. In summary, the field of plant disease detection using deep learning CNNs is a rapidly advancing area with numerous opportunities for future research and development. Findings from this project can guide the development of more precise diagnostic tools in the future, which could improve the detection and management of plant diseases, leading to more effective and sustainable agricultural practices. Future work on this model may explore the use of fusion techniques to extract more significant features and evaluate its performance on other leaf samples from datasets. Predictive models can be developed using feature selection and extraction techniques from the plant village dataset, which can help detect plant diseases early on.

FUTURE SCOPE:

The field of plant disease detection using deep learning CNNs offers several promising avenues for future exploration and development. One direction is to broaden the scope of the project by including a wider variety of plant species and diseases. Collecting and labeling more images of healthy and diseased plants could enhance the accuracy and robustness of the model. Another future direction is to incorporate additional data sources, such as environmental and weather data, into the model. This would require more advanced models capable of integrating multiple data sources to provide more comprehensive insights into plant health. Furthermore, transfer learning could be explored to fine-tune pre-trained CNN models on new datasets for disease detection. This approach could potentially reduce the amount of training data required and increase the efficiency of the model. Additionally, user-friendly mobile or web-based applications could be developed to facilitate quick and easy identification of plant diseases using deep learning models. These applications could incorporate image recognition capabilities and provide real-time feedback on plant health. In summary, the field of plant disease detection using deep learning CNNs is a rapidly advancing area with numerous opportunities for future research and development.

REFERENCES

[1] I. Sutskever, A. Krichevsky, and G. H. E. Hinton, Image Classification with Deep Convolutional Neural Networks, in Neural Information Processing Systems, (2012).

[2] Aruna Chakraborty, D. Dutta Majumder. Used a technique called Bhattacharya’s similarity calculation that can be “used for detecting paddy plant disease” International Journal of Applied Information Systems. (2014).

[3] J. S. Eun , H. Park and S. H. Kim, Disease image-based diagnosing and predicting of the crops with the deep learning mechanism, In Information and Communication Technology Convergence (ICTC), IEEE 2017 International Conference on, (2017).

[4] Zhang & Dai, “Picture of Plant Disease Segmentation Model Using Shuffle Frog Leap Algorithm and Pulse Coupled Neural Networks.”, (2018).

[5] Fuentes,Yoon,J. Lee, “High-performance deep neural network-based tomato plant diseases and pests' diagnosis system with refinement filter bank”, Frontiers Plant Sci., vol. 9, p. 1162. (2018).

[6] Ritika Arora, Aakanksha Rastogi, and Shanu Sharma. Disease severity detection using k-means clustering,” As a classifier, artificial neural network (ANN) was utilised to determine how severely infected a leaf was.(2018).

[7] V. S. Rajpurohit and S. S. Sannakki proposed a “Classification of Pomegranate in Diseases Based on Back Propagation Neural Network”, colour and texture are employed as characteristics to segment the area where the defect is most apparent. The categorization in this case was performed by a neural network classifier. (2019).

[8] P. R. Rothe and R. V. Kshirsagar. “Pattern Recognition Techniques for Detection of Cotton Leaf Disease". (2020).

[9] Kumar, Madhav & Sachin, “Coffee Plant Disease Identification Using Convolutional Neural Network”, (2020).

[10] Andre, Ferreira “Detecting plant diseases on photos with convolutional neural networks”, (2020).

[11] uan Tian, Shenglian Lu, Xinyu Guo and Chunjiang Zhao, "SVM Multiple Classification System for the Prediction of Wheat Leaves Diseases," Colour features are changed to HIS from RGB, via GLCM, with seven various moments used as parameters. Specifically, they made use of an SVM classifier with MCS to detect wheat plant diseases in a non-real time setting. (2020).

[12] Ernest Mwebaze, John A. Quinn, and James. They suggested " Automatic Vision-Based Diagnosis of Black Sigatoka Disease and Banana Bacterial Wilt Disease " by the RGB, HSV, and L*a*b colour spaces are all represented by histograms, which are then extracted and modified. To classify data, we employ area under the curve analysis and a maximum tree constructed from peak components. Remarkably high scores may be obtained using randomised trees in seven different classifiers, and the application benefits from the real-time data and adaptability they offer. (2020).

[13] Shima Ramesh, Vinod, “Cucumber leaf disease detection with multiclass support vector machines”. IEEE International Conference on the topic Signal Processing, Wireless Communications and Networking (2021).

[14] Dhiman Mondal, Dipak Kumar Kole, “Leaves Diseases Prediction and Grading using Fuzzy Logic & Computer Vision Technology”, second International Conference on the topic of Integrated Networks and Signal Processing (2021).

[15] A. Blessy, Dr. Joy Winnie Wise “Digital image processing for the prediction and segregation of leaf diseases”. IEEE International Conference on the topic Innovations in Information, Embedded and Communication Systems (2021).

APPENDIX 1

This section contains details on the language, software and packages used in our project.

This project is developed in Python, which is a general-purpose interpreted, interactive, object oriented and high-level programming language. It offers concise and readable code. Despite being highly complex with versatile workflows, the AI and ML algorithms, when written in Python, can help the developers create robust and reliable machine intelligent systems. The list of Python packages used in our project are:

- **Convolution**: We feed images here, apply convolutions and perform
- **Pooling**: After feature maps are extracted, we try to reduce the size of feature maps, that is compress them, which is down sampling.
- **Flattening**: After all the feature maps are pooled, we flatten them. Here, we consider the 2D pixels and convert into 1D and then feed them to our dense fully connected artificial neural network and train the model.
- **Padding:**

There are a few ways to deal with the pixels around the edges:

1. Inadequate resolution due to missing edge pixels
2. Inflating by using pixels with no value
3. Protective cushioning against reflections

By far the most effective method is reflection padding, which involves copying pixels from the image's edges and pasting them onto the image's periphery so that the convolutional kernel has enough room to analyze the edge pixels. With a 3x3 kernel, an extra pixel should be placed around the perimeter. The dimension has been half and adjusted to the number of pixels added to each side. Many research articles follow a common practice of simply ignoring the edge pixels, which results in a loss of data (which is exacerbated by the presence of multiple deep convolutional layers). Due to this, I was unable to locate any suitable graphics that could effectively express some of the arguments made here without being deceptive or conflating stride 1 and stride 2 cvds.

- **Strides:**

  If the input has width and height and the kernel iterates over each pixel in turn, the resulting output will also have width & height. The output channel/feature map size can be reduced by half by using a stride two convolution instead of a stride one convolution, in which the convolutional kernel takes strides of one pixel. Since the kernel only generates a single, aggregated output for each stride, the maximum value that could be produced input with width w, height h, and depth 3 with padding would be width w/2, height h/2, and depth 1.

- **Rectified Linear Unit:**

  A Rectified Linear Unit is used as a non-linear activation function. A ReLU says if the value is less than zero, round it up to zero.

- **Normalization:**

  The term "normalization" refers to the process of removing the mean and subtracting the standard deviation. In a process known as Min-Max scaling, the range of the dataset is transformed to be between -1 and 1, standardizing the data on the same scale.

  Normalization of input features is a typical technique for achieving data standardization by means of the mean being subtracted & scale to random values. It is often crucial that the input features have the same order and variance and are centered around zero. Images, for example, have their data scaled so that the values range from 0 to 1 by dividing each pixel's value by 255.

- **Activation Function:**

  The main purpose of using an activation function in a CNN is to decide which combination of weights and input will fire the next neuron. The activation function used in our CNN model is the Rectified Linear Unit activation function, often known as ReLU. If the input is higher than zero, the function returns the input; otherwise, it returns zero. Where x is an input and y are the output, this can be written as y = max (0, x).

- **Matplotlib:** It is a python library. It can cooperate to various visual styles like, make interactive Figures that can zoom, pan, update, customize visual style and layout, export to many file formats, off between preserving global structure and local structure.

- **Batch Normalization:**

  Training times can be cut in half and accuracy can be improved by a magnitude when using batch normalization to stabilize the predictions made by a network. By removing the average batch's activations and dividing the standard variation of the batch's activations. However, even after normalizing the input, some activations may still be larger than average, rendering the network less stable. By applying a modification called batch normalization, we can keep the return confidence interval around 1 and the mean output close to 0. It's important to note that the way batch normalization functions vary between the training and inference phases.

  The deep learning model takes in an N-dimensional tensor with a specific shape for input and output. Typically, for a 2D input, the shape of the input tensor is (batch size, input dimensions) and the output tensor is in N dimensions with the shape (size of batch, units).

  To prepare the input data, the Image Data Generator performs various transformations such as resizing, shearing, zooming, and flipping. All possible image orientations can be represented using this generator.

  During the training process, the data in the train dataset directory can be preprocessed using the train datagen. flow from directory function, with the intended image size being set using the target size parameter. Similarly, for model testing, the Test catagen. flow from a directory function can be used. The steps per epoch variable indicates the number of times the model will be run on the training data, and the fit generator is used to fit the data into the model.

  The number of epochs determines how many iterations of forward and backward training the model will undergo. To validate the model's performance, the validation and test data are taken from the validation data, with the quantity of validation/test samples being denoted by validation steps. These processes ensure that the deep learning model is accurately trained and tested.

- **Numpy**: It is a Python library that provides a multidimensional array object, various derived objects , and a plan for fast operations on arrays, including logical, shape manipulation, mathematical, sorting, selecting, basic linear algebra, I/O, discrete Fourier transforms, basic statistical operations, random simulation, etc. It is the elementary library for scientific problems in Python.

- **Pandas:** It is a Python package that provides fast, flexible, and expressive data structures designed to make working with "relational" or "labeled" data both easy and flexible. It targets to be the elementary hierarchy level for doing practical and real-world data analysis in Python. Additionally, it has the goal of becoming the most powerful and flexible open-source data analysis / manipulation tool available in any language.

- **Multi-Maximum Pooling-Layer Structures:**

  All CNNs have pooling layers after their activation function layers. Pooling layers are used to reduce the dimensions of feature maps. They are 3 Different pooling methods including averaging, maximizing, and global pooling. We have made use of max-pooling technique because it enhances the features of the images by taking the maximum of the values present in that region. At the end of this operation, we obtain the most prominent features of the image. This step helps in avoiding overfitting and reduces noise distortion. Other than these 2 techniques used for feature extraction of an image, a dropout layer is used to reduce overfitting of the model, where the neurons are dropped at random. Flatten layer is used to convert the output into one long feature vector which is given as input into the next layer.

- **Convolutional Neural Network:**

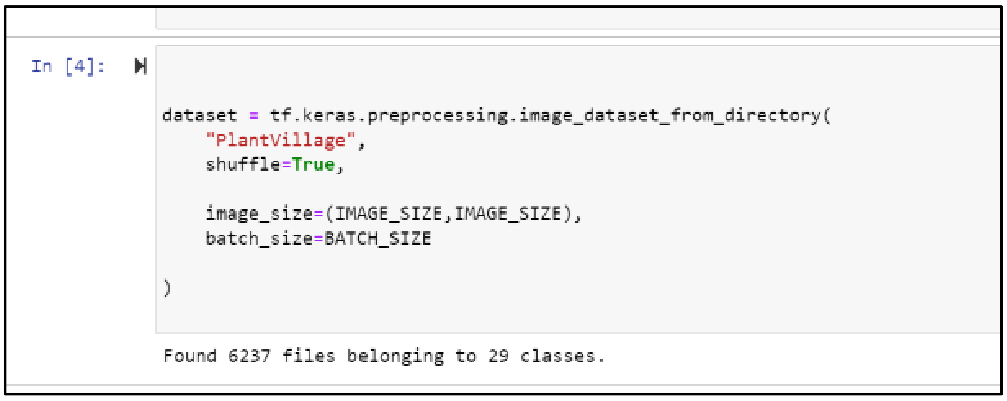
  Convolutional Neural Networks (CNN's) are the backbone of image classification, a deep learning phenomenon that takes an image and assigns it a class and a label that makes it unique. A convolutional neural network (CNN) is an efficient recognition algorithm which is widely used in pattern recognition and image processing.

  It has many features such as simple structure, less training parameters and adaptability which means that it can read 2D images by applying filters that other conventional algorithms cannot. It has become a hot topic in voice analysis and image recognition.

  The reason why CNN is so popular is that it requires very little pre-processing, which means that it can read 2D images by applying filters that other conventional algorithms cannot. CNNs are equipped with an input layer, an output layer, and hidden layers, all of which help process and classify images. The hidden layers comprise convolutional layers, ReLU layers, pooling layers, and fully connected layers, all of which play a crucial role

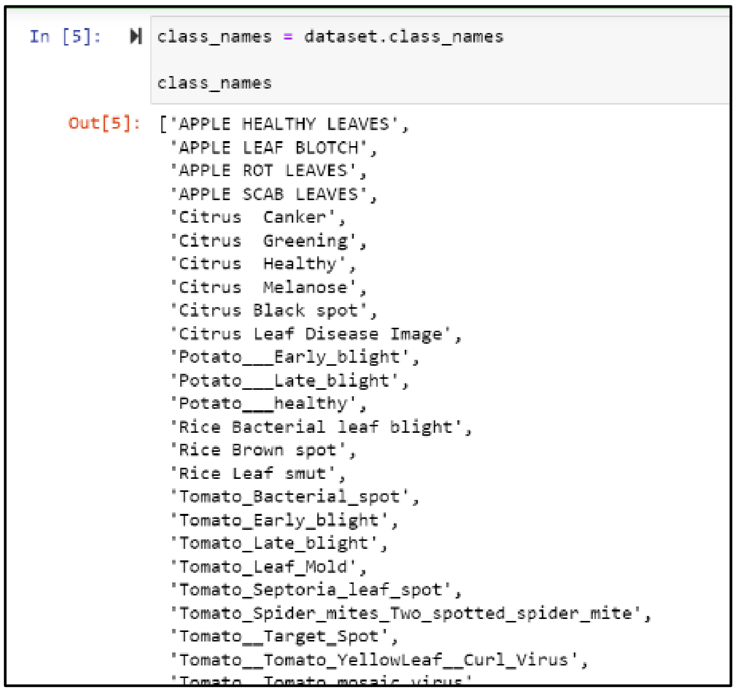
APPENDIX 2

This section contains the code for training the model and developing algorithm used in this project.



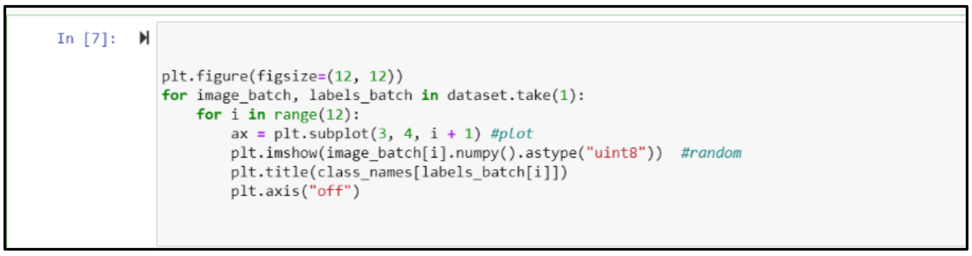
```
In [4]:

dataset = tf.keras.preprocessing.image_dataset_from_directory(
    "PlantVillage",
    shuffle=True,

    image_size=(IMAGE_SIZE,IMAGE_SIZE),
    batch_size=BATCH_SIZE

)

Found 6237 files belonging to 29 classes.
```

Figure 1: Importing Dataset



```
In [5]:  class_names = dataset.class_names

         class_names

Out[5]:  ['APPLE HEALTHY LEAVES',
          'APPLE LEAF BLOTCH',
          'APPLE ROT LEAVES',
          'APPLE SCAB LEAVES',
          'Citrus  Canker',
          'Citrus  Greening',
          'Citrus  Healthy',
          'Citrus  Melanose',
          'Citrus Black spot',
          'Citrus Leaf Disease Image',
          'Potato___Early_blight',
          'Potato___Late_blight',
          'Potato___healthy',
          'Rice Bacterial leaf blight',
          'Rice Brown spot',
          'Rice Leaf smut',
          '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',
```

Figure 2: Disease Classes



```
In [7]:

plt.figure(figsize=(12, 12))
for image_batch, labels_batch in dataset.take(1):
    for i in range(12):
        ax = plt.subplot(3, 4, i + 1) #plot
        plt.imshow(image_batch[i].numpy().astype("uint8"))  #random
        plt.title(class_names[labels_batch[i]])
        plt.axis("off")
```

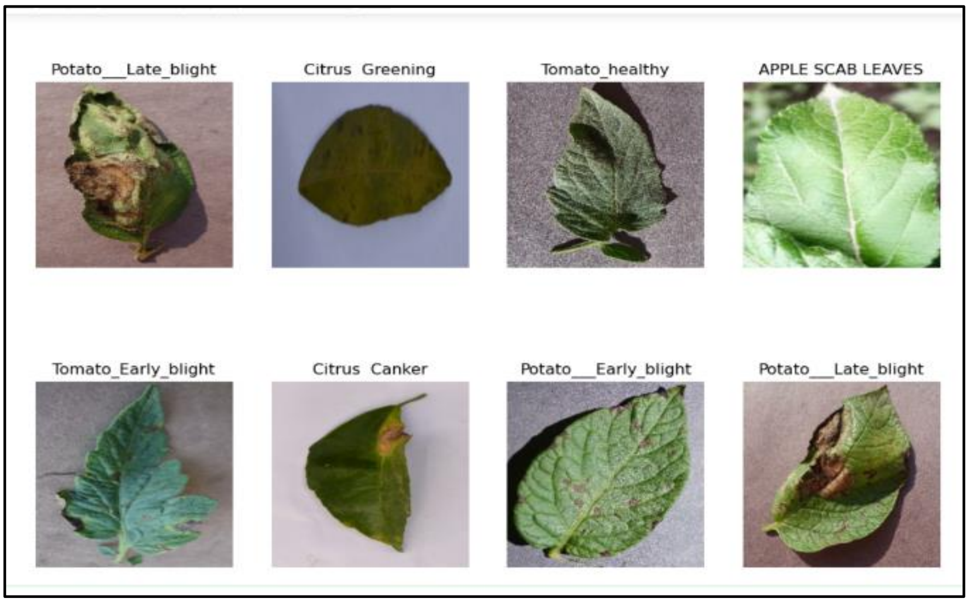
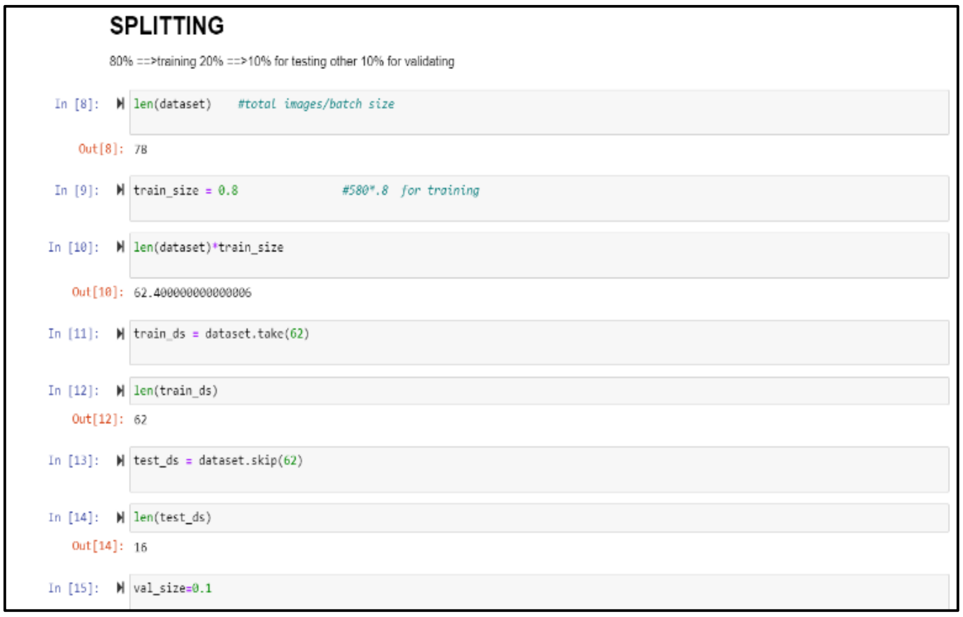


Figure 3: Input of dataset

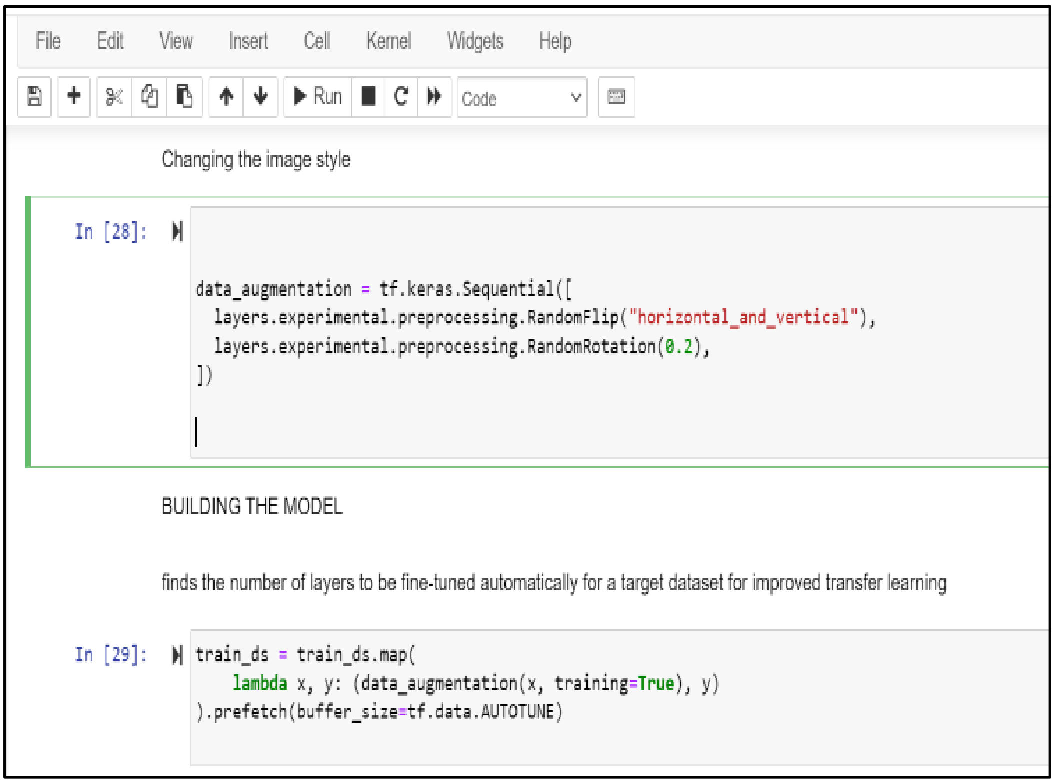


SPLITTING

80% ==>training 20% ==>10% for testing other 10% for validating

```
In [8]:  len(dataset)     #total images/batch size

Out[8]:  78

In [9]:  train_size = 0.8              #580*.8  for training

In [10]: len(dataset)*train_size

Out[10]: 62.400000000000006

In [11]: train_ds = dataset.take(62)

In [12]: len(train_ds)
Out[12]: 62

In [13]: test_ds = dataset.skip(62)

In [14]: len(test_ds)
Out[14]: 16

In [15]: val_size=0.1
```

Figure 4: Splitting dataset.



Changing the image style

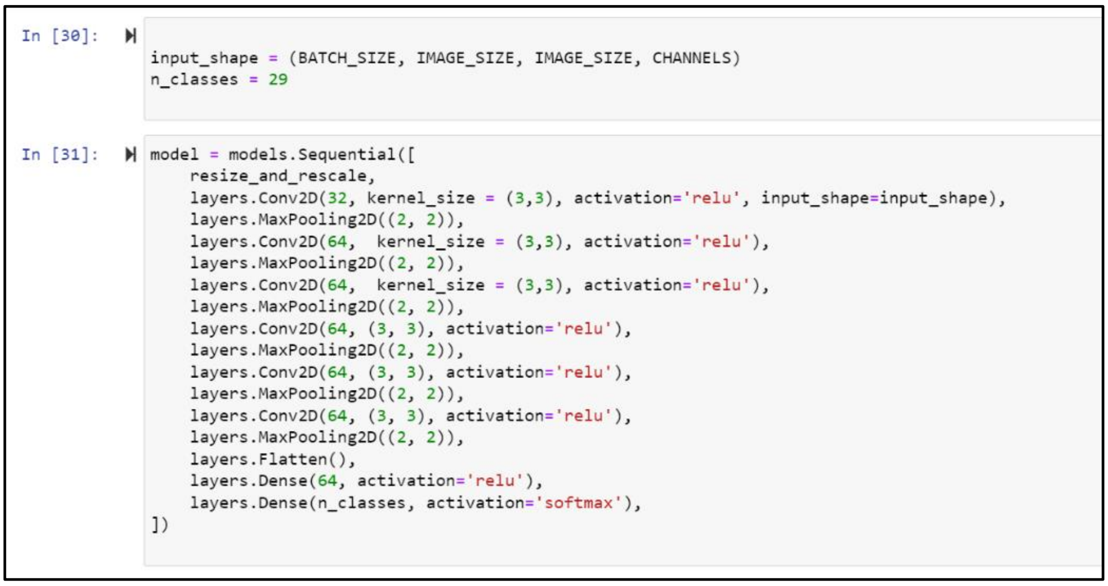
```
In [28]:

data_augmentation = tf.keras.Sequential([
  layers.experimental.preprocessing.RandomFlip("horizontal_and_vertical"),
  layers.experimental.preprocessing.RandomRotation(0.2),
])
```

BUILDING THE MODEL

finds the number of layers to be fine-tuned automatically for a target dataset for improved transfer learning

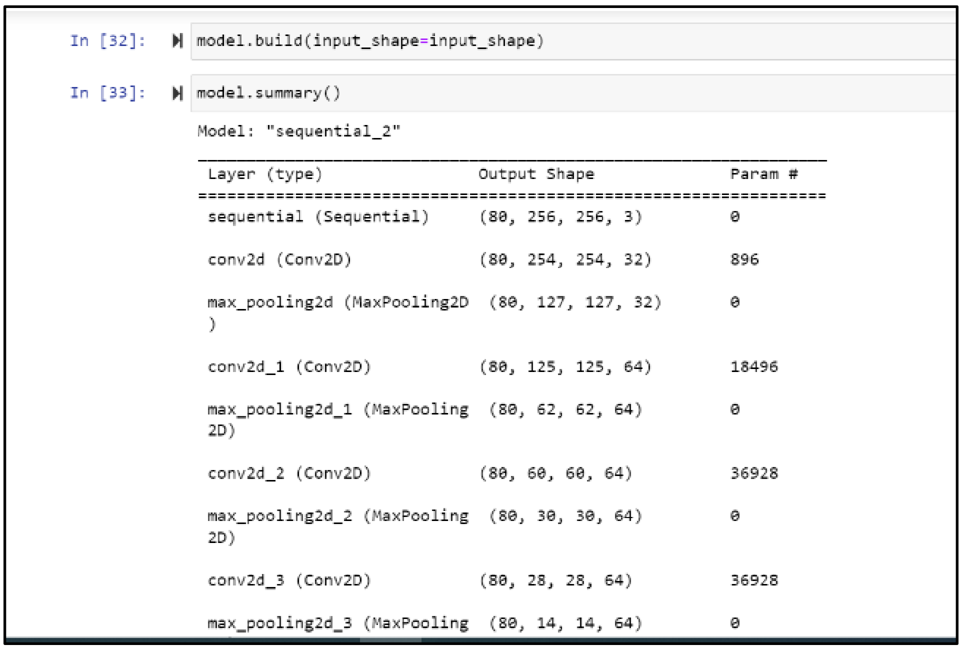
```
In [29]: train_ds = train_ds.map(
             lambda x, y: (data_augmentation(x, training=True), y)
         ).prefetch(buffer_size=tf.data.AUTOTUNE)
```

Figure 5: Data augmentation



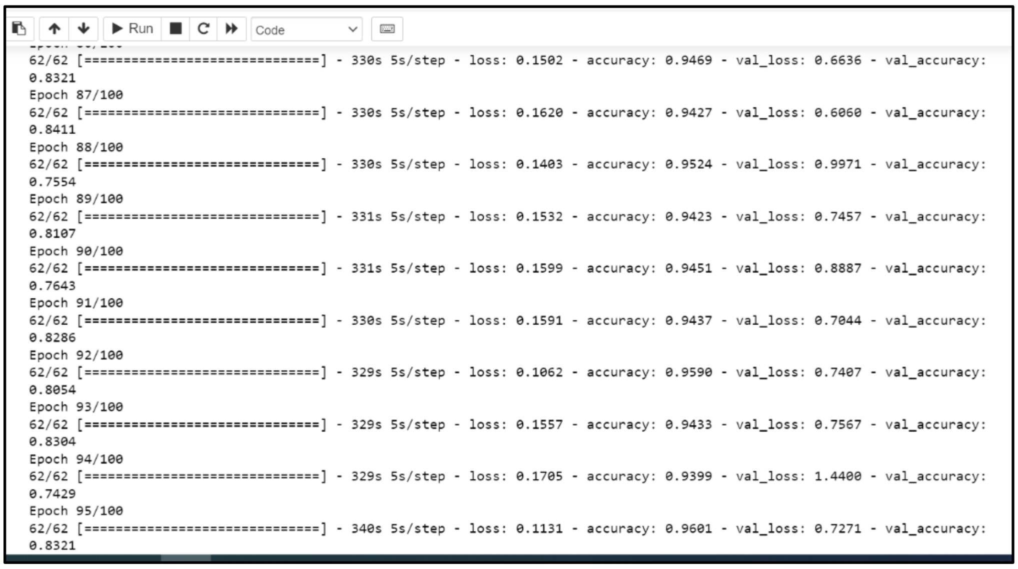
```
In [30]:
input_shape = (BATCH_SIZE, IMAGE_SIZE, IMAGE_SIZE, CHANNELS)
n_classes = 29

In [31]: model = models.Sequential([
    resize_and_rescale,
    layers.Conv2D(32, kernel_size = (3,3), activation='relu', input_shape=input_shape),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64,  kernel_size = (3,3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64,  kernel_size = (3,3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Flatten(),
    layers.Dense(64, activation='relu'),
    layers.Dense(n_classes, activation='softmax'),
])
```

Figure 6: Layers of CNN



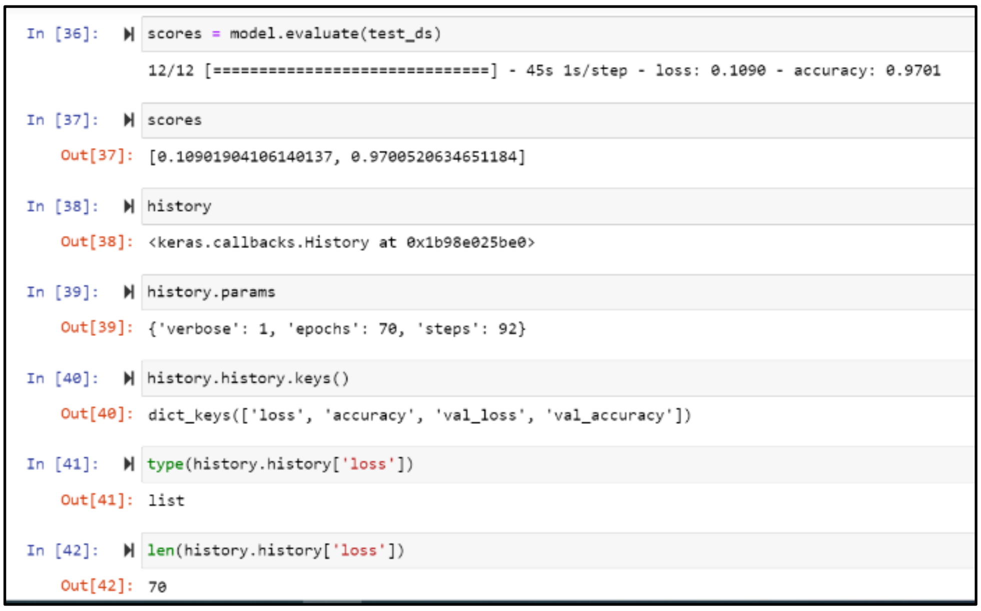
```
In [32]:   model.build(input_shape=input_shape)

In [33]:   model.summary()

           Model: "sequential_2"
           _________________________________________________________________
            Layer (type)                Output Shape              Param #
           =================================================================
            sequential (Sequential)     (80, 256, 256, 3)         0

            conv2d (Conv2D)             (80, 254, 254, 32)        896

            max_pooling2d (MaxPooling2D  (80, 127, 127, 32)       0
            )

            conv2d_1 (Conv2D)           (80, 125, 125, 64)        18496

            max_pooling2d_1 (MaxPooling  (80, 62, 62, 64)         0
            2D)

            conv2d_2 (Conv2D)           (80, 60, 60, 64)          36928

            max_pooling2d_2 (MaxPooling  (80, 30, 30, 64)         0
            2D)

            conv2d_3 (Conv2D)           (80, 28, 28, 64)          36928

            max_pooling2d_3 (MaxPooling  (80, 14, 14, 64)         0
```

Figure 7: Model Summary



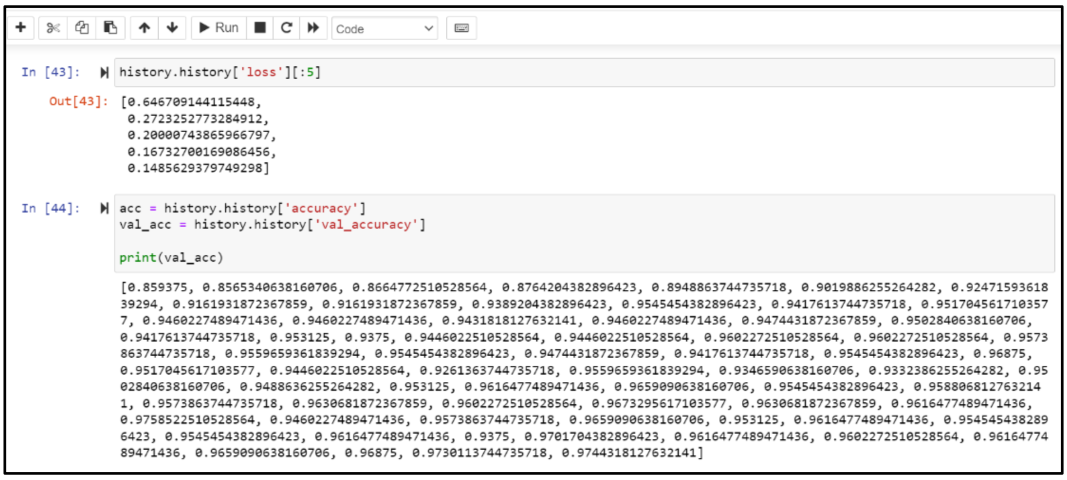
```
62/62 [==============================] - 330s 5s/step - loss: 0.1502 - accuracy: 0.9469 - val_loss: 0.6636 - val_accuracy:
0.8321
Epoch 87/100
62/62 [==============================] - 330s 5s/step - loss: 0.1620 - accuracy: 0.9427 - val_loss: 0.6060 - val_accuracy:
0.8411
Epoch 88/100
62/62 [==============================] - 330s 5s/step - loss: 0.1403 - accuracy: 0.9524 - val_loss: 0.9971 - val_accuracy:
0.7554
Epoch 89/100
62/62 [==============================] - 331s 5s/step - loss: 0.1532 - accuracy: 0.9423 - val_loss: 0.7457 - val_accuracy:
0.8107
Epoch 90/100
62/62 [==============================] - 331s 5s/step - loss: 0.1599 - accuracy: 0.9451 - val_loss: 0.8887 - val_accuracy:
0.7643
Epoch 91/100
62/62 [==============================] - 330s 5s/step - loss: 0.1591 - accuracy: 0.9437 - val_loss: 0.7044 - val_accuracy:
0.8286
Epoch 92/100
62/62 [==============================] - 329s 5s/step - loss: 0.1062 - accuracy: 0.9590 - val_loss: 0.7407 - val_accuracy:
0.8054
Epoch 93/100
62/62 [==============================] - 329s 5s/step - loss: 0.1557 - accuracy: 0.9433 - val_loss: 0.7567 - val_accuracy:
0.8304
Epoch 94/100
62/62 [==============================] - 329s 5s/step - loss: 0.1705 - accuracy: 0.9399 - val_loss: 1.4400 - val_accuracy:
0.7429
Epoch 95/100
62/62 [==============================] - 340s 5s/step - loss: 0.1131 - accuracy: 0.9601 - val_loss: 0.7271 - val_accuracy:
0.8321
```

Figure 8: Epochs



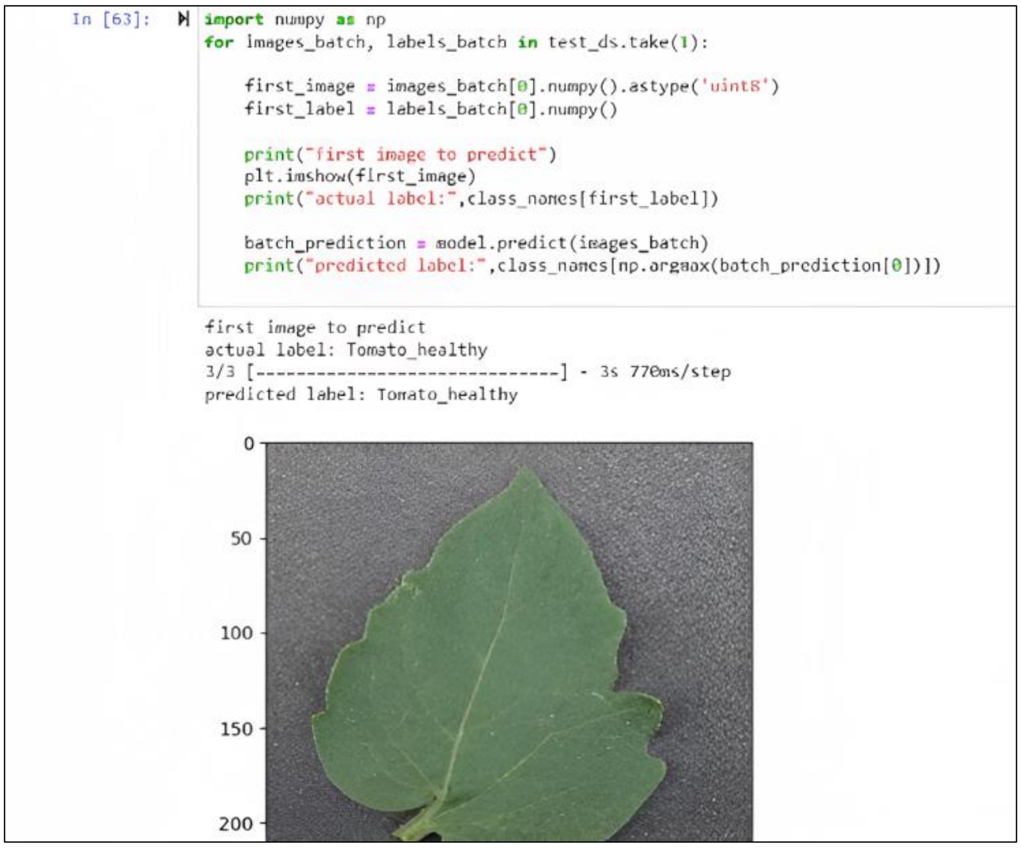
```
In [36]:  scores = model.evaluate(test_ds)

          12/12 [==============================] - 45s 1s/step - loss: 0.1090 - accuracy: 0.9701

In [37]:  scores

Out[37]:  [0.10901904106140137, 0.9700520634651184]

In [38]:  history

Out[38]:  <keras.callbacks.History at 0x1b98e025be0>

In [39]:  history.params

Out[39]:  {'verbose': 1, 'epochs': 70, 'steps': 92}

In [40]:  history.history.keys()

Out[40]:  dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])

In [41]:  type(history.history['loss'])

Out[41]:  list

In [42]:  len(history.history['loss'])

Out[42]:  70
```

Figure 9: Scores



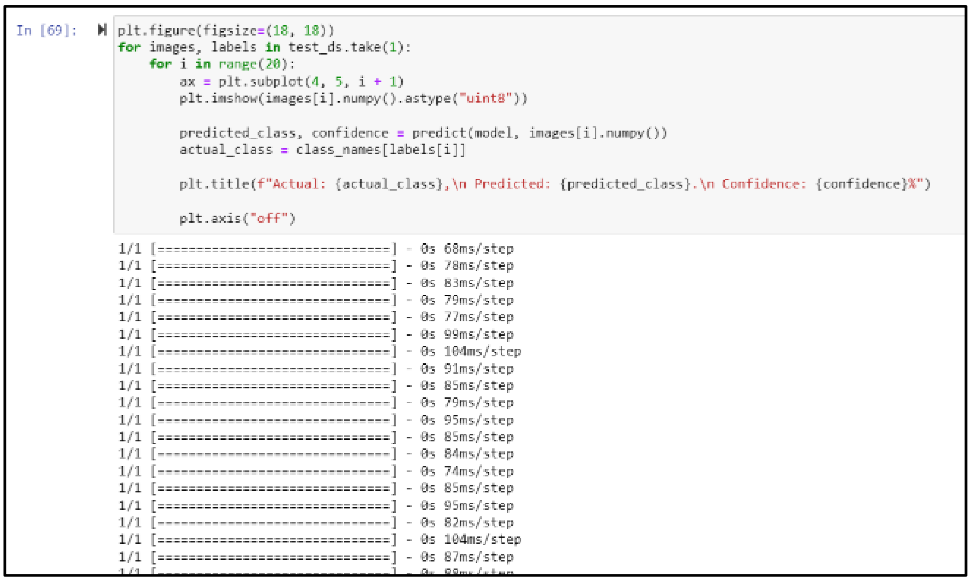
```
In [43]:  history.history['loss'][:5]

Out[43]:  [0.646709144115448,
           0.2723252773284912,
           0.20000743865966797,
           0.16732700169086456,
           0.1485629379749298]

In [44]:  acc = history.history['accuracy']
          val_acc = history.history['val_accuracy']

          print(val_acc)

          [0.859375, 0.8565340638160706, 0.8664772510528564, 0.8764204382896423, 0.8948863744735718, 0.9019886255264282, 0.92471593618
          39294, 0.9161931872367859, 0.9161931872367859, 0.9389204382896423, 0.9545454382896423, 0.9417613744735718, 0.951704561710357
          7, 0.9460227489471436, 0.9460227489471436, 0.9431818127632141, 0.9460227489471436, 0.9474431872367859, 0.9502840638160706,
          0.9417613744735718, 0.953125, 0.9375, 0.9446022510528564, 0.9446022510528564, 0.9602272510528564, 0.9602272510528564, 0.9573
          863744735718, 0.9559659361839294, 0.9545454382896423, 0.9474431872367859, 0.9417613744735718, 0.9545454382896423, 0.96875,
          0.9517045617103577, 0.9446022510528564, 0.9261363744735718, 0.9559659361839294, 0.9346590638160706, 0.9332386255264282, 0.95
          02840638160706, 0.9488636255264282, 0.953125, 0.9616477489471436, 0.9659090638160706, 0.9545454382896423, 0.958806812763214
          1, 0.9573863744735718, 0.9630681872367859, 0.9602272510528564, 0.9673295617103577, 0.9630681872367859, 0.9616477489471436,
          0.9758522510528564, 0.9460227489471436, 0.9573863744735718, 0.9659090638160706, 0.953125, 0.9616477489471436, 0.954545438289
          6423, 0.9545454382896423, 0.9616477489471436, 0.9375, 0.9701704382896423, 0.9616477489471436, 0.9602272510528564, 0.96164774
          89471436, 0.9659090638160706, 0.96875, 0.9730113744735718, 0.9744318127632141]
```

Figure 10: Accuracy



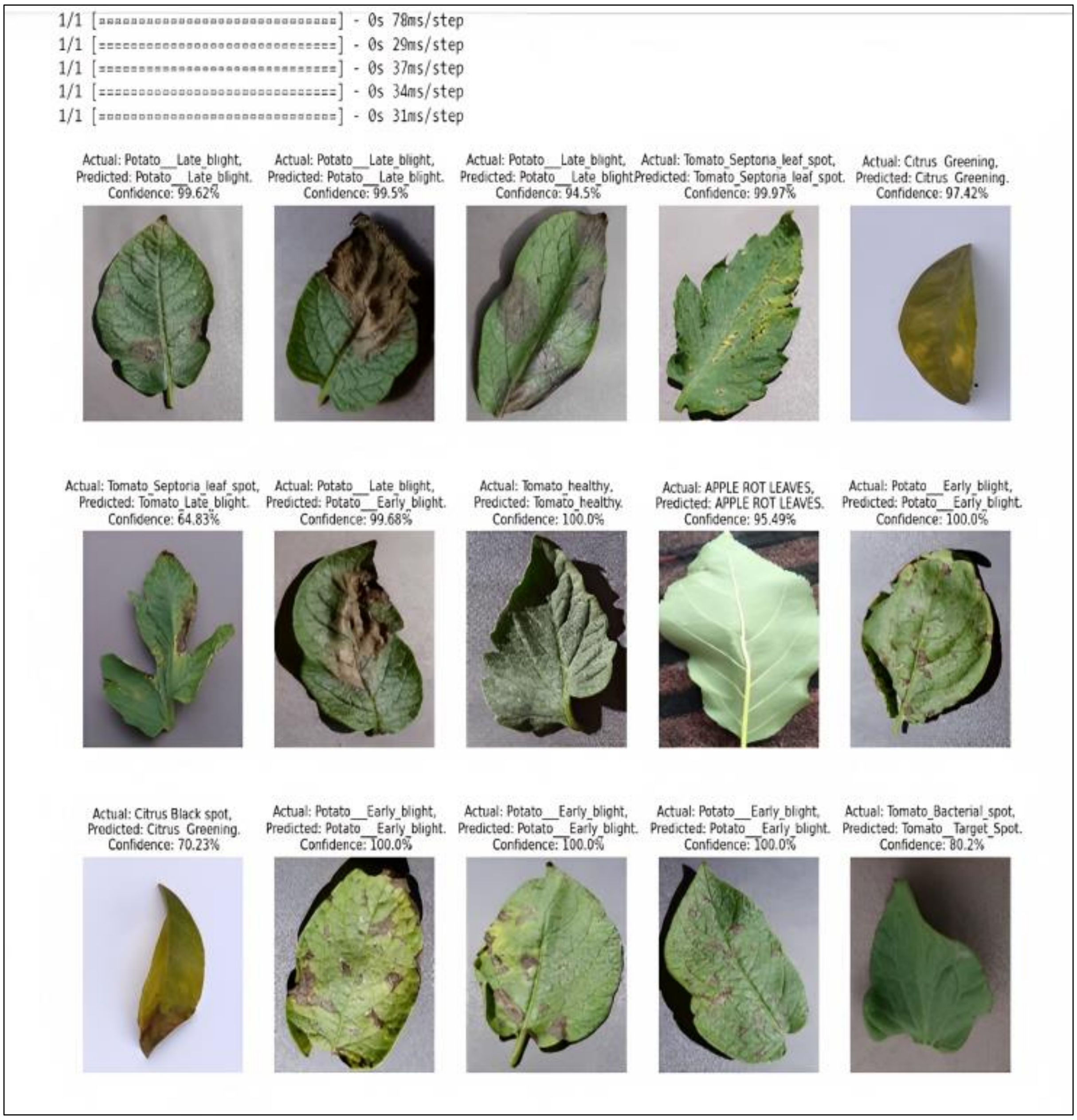
```
In [63]:  import numpy as np
          for images_batch, labels_batch in test_ds.take(1):

              first_image = images_batch[0].numpy().astype('uint8')
              first_label = labels_batch[0].numpy()

              print("first image to predict")
              plt.imshow(first_image)
              print("actual label:",class_names[first_label])

              batch_prediction = model.predict(images_batch)
              print("predicted label:",class_names[np.argmax(batch_prediction[0])])

          first image to predict
          actual label: Tomato_healthy
          3/3 [==============================] - 3s 770ms/step
          predicted label: Tomato_healthy
```

Figure 11: Output Prediction



```
In [69]:  plt.figure(figsize=(18, 18))
          for images, labels in test_ds.take(1):
              for i in range(20):
                  ax = plt.subplot(4, 5, i + 1)
                  plt.imshow(images[i].numpy().astype("uint8"))

                  predicted_class, confidence = predict(model, images[i].numpy())
                  actual_class = class_names[labels[i]]

                  plt.title(f"Actual: {actual_class},\n Predicted: {predicted_class}.\n Confidence: {confidence}%")

                  plt.axis("off")

          1/1 [==============================] - 0s 68ms/step
          1/1 [==============================] - 0s 78ms/step
          1/1 [==============================] - 0s 83ms/step
          1/1 [==============================] - 0s 79ms/step
          1/1 [==============================] - 0s 77ms/step
          1/1 [==============================] - 0s 99ms/step
          1/1 [==============================] - 0s 104ms/step
          1/1 [==============================] - 0s 91ms/step
          1/1 [==============================] - 0s 85ms/step
          1/1 [==============================] - 0s 79ms/step
          1/1 [==============================] - 0s 95ms/step
          1/1 [==============================] - 0s 85ms/step
          1/1 [==============================] - 0s 84ms/step
          1/1 [==============================] - 0s 74ms/step
          1/1 [==============================] - 0s 85ms/step
          1/1 [==============================] - 0s 95ms/step
          1/1 [==============================] - 0s 82ms/step
          1/1 [==============================] - 0s 104ms/step
          1/1 [==============================] - 0s 87ms/step
```

Figure 12: Overall Prediction



```
1/1 [==============================] - 0s 78ms/step
1/1 [==============================] - 0s 29ms/step
1/1 [==============================] - 0s 37ms/step
1/1 [==============================] - 0s 34ms/step
1/1 [==============================] - 0s 31ms/step
```

Actual: Potato___Late_blight,
Predicted: Potato___Late_blight.
Confidence: 99.62%

Actual: Potato___Late_blight,
Predicted: Potato___Late_blight.
Confidence: 99.5%

Actual: Potato___Late_blight,
Predicted: Potato___Late_blight
Confidence: 94.5%

Actual: Tomato_Septoria_leaf_spot,
Predicted: Tomato_Septoria_leaf_spot.
Confidence: 99.97%

Actual: Citrus_Greening,
Predicted: Citrus_Greening.
Confidence: 97.42%

Actual: Tomato_Septoria_leaf_spot,
Predicted: Tomato_Late_blight.
Confidence: 64.83%

Actual: Potato___Late_blight,
Predicted: Potato___Early_blight.
Confidence: 99.68%

Actual: Tomato_healthy,
Predicted: Tomato_healthy.
Confidence: 100.0%

Actual: APPLE ROT LEAVES,
Predicted: APPLE ROT LEAVES.
Confidence: 95.49%

Actual: Potato___Early_blight,
Predicted: Potato___Early_blight.
Confidence: 100.0%

Actual: Citrus Black spot,
Predicted: Citrus_Greening.
Confidence: 70.23%

Actual: Potato___Early_blight,
Predicted: Potato___Early_blight.
Confidence: 100.0%

Actual: Potato___Early_blight,
Predicted: Potato___Early_blight.
Confidence: 100.0%

Actual: Potato___Early_blight,
Predicted: Potato___Early_blight.
Confidence: 100.0%

Actual: Tomato_Bacterial_spot,
Predicted: Tomato__Target_Spot.
Confidence: 80.2%

Figure 13: Final Output

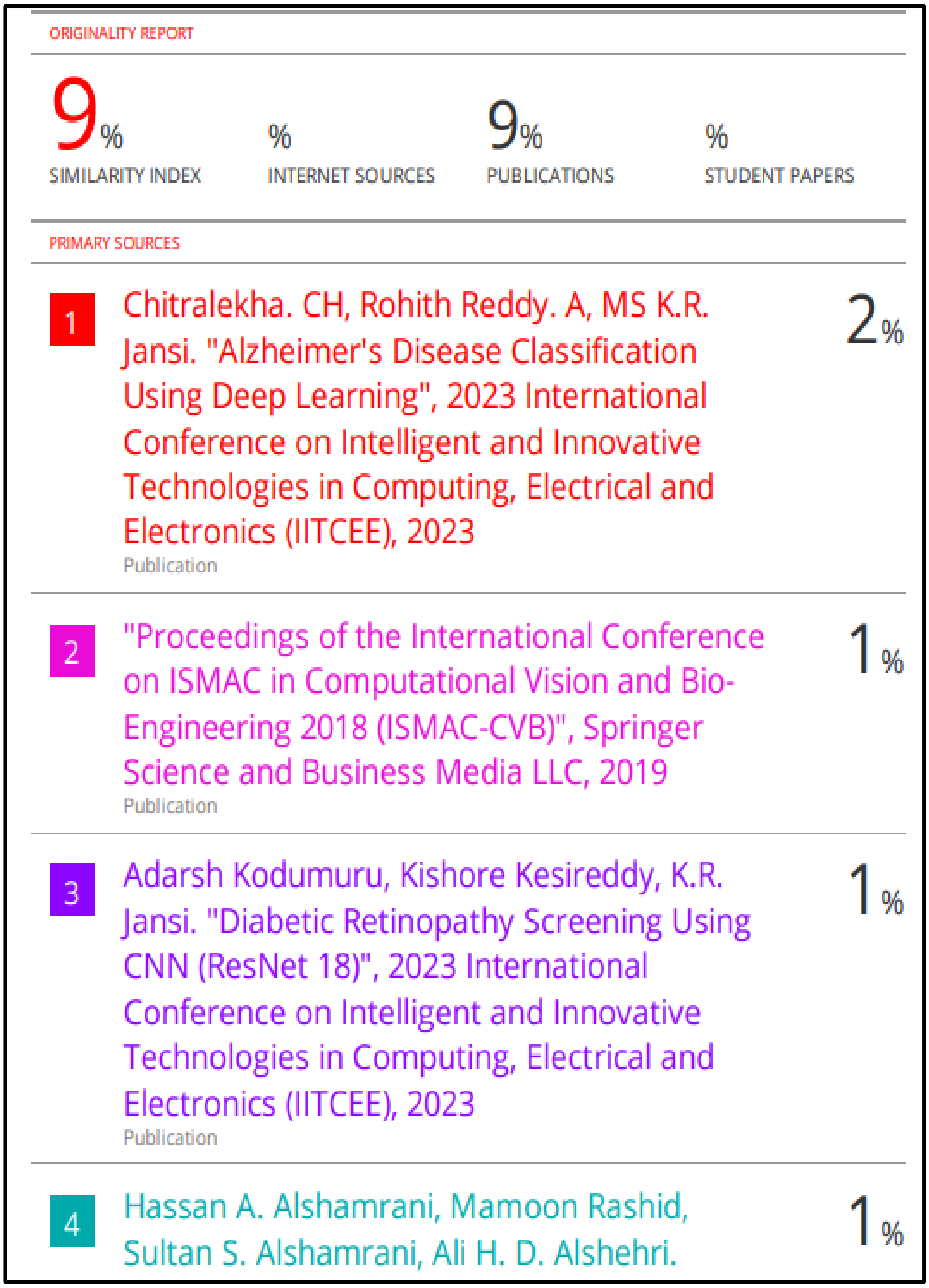
Format - I

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY **(Deemed to be University u/s 3 of UGC Act, 1956)**		
Office of Controller of Examinations		
REPORT FOR PLAGIARISM CHECK ON THE DISSERTATION/PROJECT REPORTS FOR UG PROGRAMMES **(To be attached in the dissertation/ project report)**		
1	Name of the Candidate **(IN BLOCK LETTERS)**	ALLA ROHITH REDDY CHITRALEKHA CHEDURUPALLI
2	Address of the Candidate	D/NO: 6-219/2, 201B, SAI SPOORTHI HOUSING COMPLEX, KAKATIYA NAGAR, HYDERABAD Pin Code: 500054 **Mobile Number:** 7981849470 D/NO: 40-6-13, COAL SHOP, REVENUE COLONY, MOGHALRAJAPURAM, VIJAYAWADA, ANDHRA PRADESH. Pin Code: 520010 Mobile Number: 9985777403
3	Registration Number	RA1911003010361 RA1911003010387
4	Date of Birth	3 Dec 2001 26 March 2001
5	Department	Computer Science and Engineering
6	Faculty	Engineering and Technology, School of Computing
7	Title of the Dissertation/Project	Plant Disease Detection using Deep learning techniques
8	Whether the above project /dissertation is done by	~~Individual~~ or Group : (Strike whichever is not applicable) a) If the project/ dissertation is done in group, then how many students together completed the project:02. Mention the Name & Register number of other candidates: Ch. Chitralekha –RA1911003010381 Alla Rohith Reddy-RA1911003010361
9	Name and address of the Supervisor / Guide	Dr. K.R Jansi Associate Professor Department of Computer Science and Engineering SRM Institute of Science and Technology Kattankulatur - 603203. **Mail ID:** jansik@srmist.edu.in Mobile Number: 9600082712

10	Name and address of Co-Supervisor / Co- Guide (if any)	NIL **Mail ID: Mobile Number:**

11	Software Used	Turnitin		
12	Date of Verification	10 – May -2023		
13	**Plagiarism Details: (to attach the final report from the software)**			
Chapter	**Title of the Chapter**	**Percentage of similarity index (including self citation)**	**Percentage of similarity index (Excluding self-citation)**	**% of plagiarism after excluding Quotes, Bibliography, etc.,**
1	INTRODUCTION	1%	0%	<1%
2	LITERATURE SURVEY	<1%	0%	<1%
3	SYSTEM ARCHITECTURE	2%	1%	<1%
4	METHODOLOGY	1%	<1%	1%
5	RESULTS	1%	1%	<1%
6	CONCLUSION	<1%	<1%	<1%
7	REFERENCES	<1%	1%	<1%
Appendices		1%	1%	<1%
We declare that the above information have been verified and found true to the best of our knowledge.				
Signature of the Candidate		**Name & Signature of the Staff (Who uses the plagiarism check software)**		
Name & Signature of the Supervisor/ Guide		**Name & Signature of the Co-Supervisor/Co-Guide**		
Name & Signature of the HOD				

PLAIGIARISM REPORT



ORIGINALITY REPORT

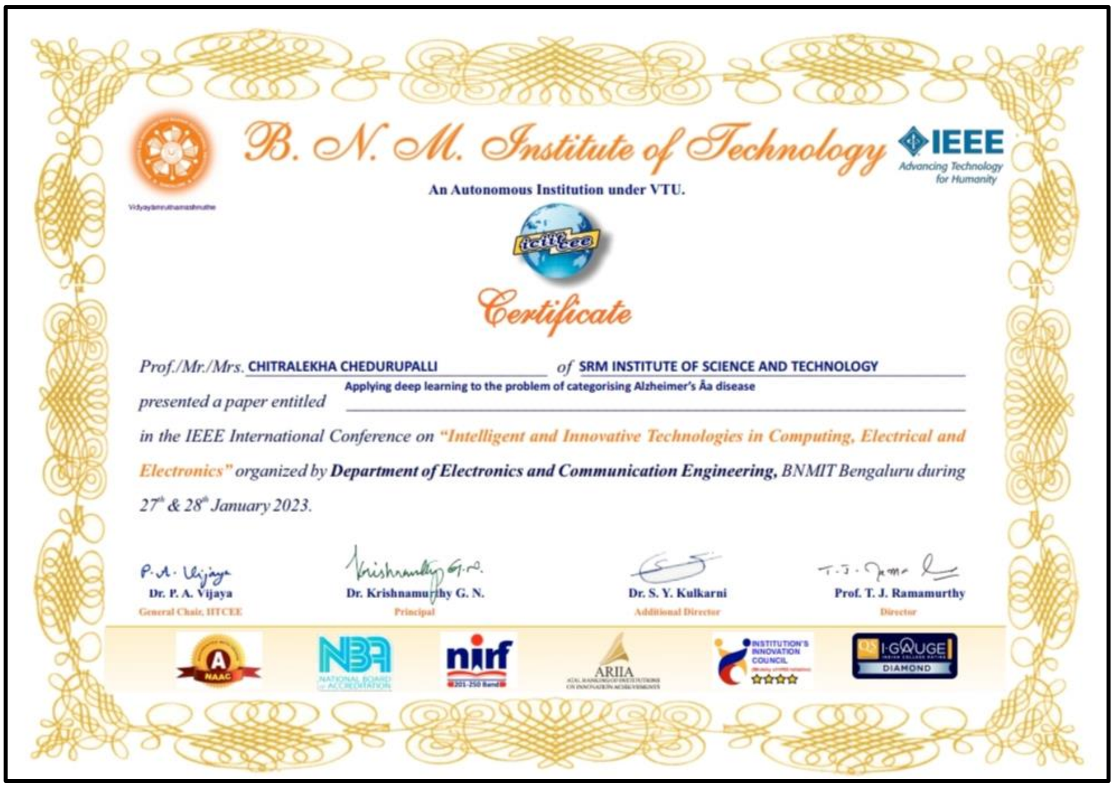
9%	%	9%	%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	Chitralekha. CH, Rohith Reddy. A, MS K.R. Jansi. "Alzheimer's Disease Classification Using Deep Learning", 2023 International Conference on Intelligent and Innovative Technologies in Computing, Electrical and Electronics (IITCEE), 2023 Publication	2%
2	"Proceedings of the International Conference on ISMAC in Computational Vision and Bio-Engineering 2018 (ISMAC-CVB)", Springer Science and Business Media LLC, 2019 Publication	1%
3	Adarsh Kodumuru, Kishore Kesireddy, K.R. Jansi. "Diabetic Retinopathy Screening Using CNN (ResNet 18)", 2023 International Conference on Intelligent and Innovative Technologies in Computing, Electrical and Electronics (IITCEE), 2023 Publication	1%
4	Hassan A. Alshamrani, Mamoon Rashid, Sultan S. Alshamrani, Ali H. D. Alshehri.	1%

PUBLICATION STATUS

MINOR PROJECT:



B. N. M. Institute of Technology

An Autonomous Institution under VTU.

IEEE Advancing Technology for Humanity

Vidyayamruthamashnuthe

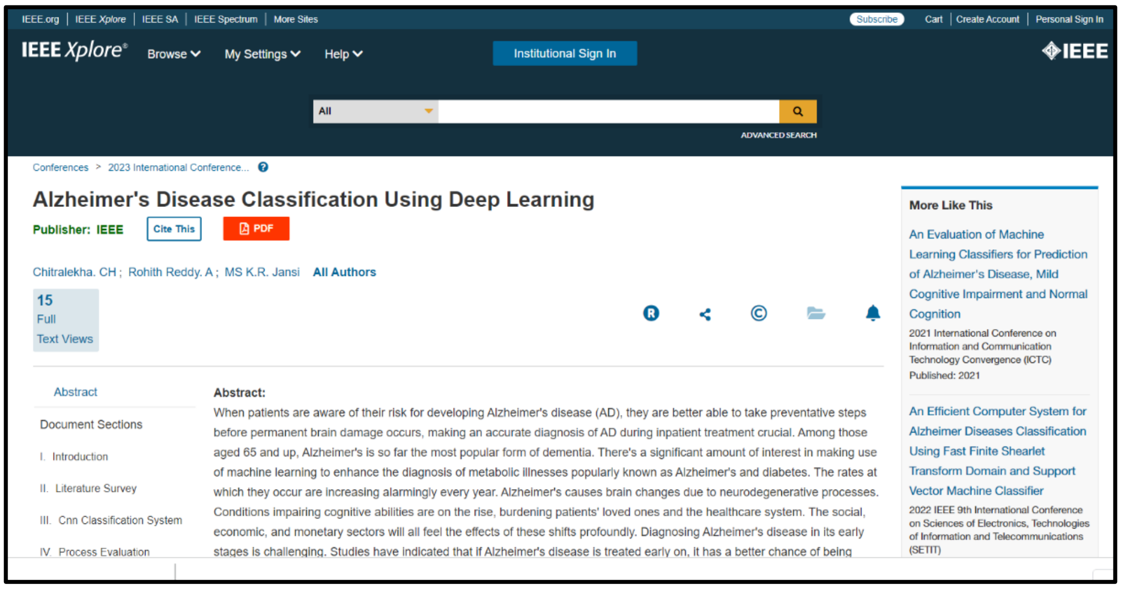
Certificate

Prof./Mr./Mrs. CHITRALEKHA CHEDURUPALLI of SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

presented a paper entitled Applying deep learning to the problem of categorising Alzheimer's Âa disease

in the IEEE International Conference on "Intelligent and Innovative Technologies in Computing, Electrical and Electronics" organized by Department of Electronics and Communication Engineering, BNMIT Bengaluru during 27th & 28th January 2023.

Dr. P. A. Vijaya	Dr. Krishnamurthy G. N.	Dr. S. Y. Kulkarni	Prof. T. J. Ramamurthy
General Chair, IITCEE	Principal	Additional Director	Director



IEEE.org | IEEE Xplore | IEEE SA | IEEE Spectrum | More Sites
Subscribe
Cart | Create Account | Personal Sign In
IEEE Xplore
Browse
My Settings
Help
Institutional Sign In
IEEE
All
ADVANCED SEARCH

Conferences > 2023 International Conference...

Alzheimer's Disease Classification Using Deep Learning

Publisher: IEEE Cite This PDF

Chitralekha. CH ; Rohith Reddy. A ; MS K.R. Jansi All Authors

15 Full Text Views

Abstract:

When patients are aware of their risk for developing Alzheimer's disease (AD), they are better able to take preventative steps before permanent brain damage occurs, making an accurate diagnosis of AD during inpatient treatment crucial. Among those aged 65 and up, Alzheimer's is so far the most popular form of dementia. There's a significant amount of interest in making use of machine learning to enhance the diagnosis of metabolic illnesses popularly known as Alzheimer's and diabetes. The rates at which they occur are increasing alarmingly every year. Alzheimer's causes brain changes due to neurodegenerative processes. Conditions impairing cognitive abilities are on the rise, burdening patients' loved ones and the healthcare system. The social, economic, and monetary sectors will all feel the effects of these shifts profoundly. Diagnosing Alzheimer's disease in its early stages is challenging. Studies have indicated that if Alzheimer's disease is treated early on, it has a better chance of being

More Like This

An Evaluation of Machine Learning Classifiers for Prediction of Alzheimer's Disease, Mild Cognitive Impairment and Normal Cognition

2021 International Conference on Information and Communication Technology Convergence (ICTC)

Published: 2021

An Efficient Computer System for Alzheimer Diseases Classification Using Fast Finite Shearlet Transform Domain and Support Vector Machine Classifier

2022 IEEE 9th International Conference on Sciences of Electronics, Technologies of Information and Telecommunications (SETIT)

MAJOR PROJECT:

Submitted invention disclosure form and paper is to be submitted for book chapter.