Plant Disease Detection

Major project report submitted in partial fulfilment of the requirement for the degree of **Bachelor of Technology**

in

Computer Science and Engineering & Information Technology

by

Sanya Kanwar(181465)

Stuti Gupta(181467)

Under the supervision of

Dr. Ruchi Verma

Department of Computer Science Engineering and Information Technology
Jaypee University of Information Technology, Waknaghat, 173234,

Himachal Pradesh, INDIA

CERTIFICATE

This is to certify that the work which is being presented in the project report titled **Plant Disease Detection** in partial fulfilment of the requirements for the award of the degree of **Bachelor of Technology in Computer Science and Engineering & Information Technology**, Jaypee University of Information Technology, Waknaghat is an authentic record of work carried out by Sanya Kanwar and Stuti Gupta during the period from Janaury 2022 to May 2022 under the supervision of **Dr. Ruchi Verma**, Department of Computer Science and Engineering & Information Technology, Jaypee University of Information Technology, Waknaghat.

Sanya Kanwar(181465)

Stuti Gupta(181467)

The above statement made is correct to the best of my knowledge.

Supervisor Name: Dr. Ruchi Verma

Designation: Assistant Professor (Grade-II)

Department Name: Computer Science and Engineering

ACKNOWLEDGEMENT

Firstly, I express my heartiest thanks and gratefulness to almighty God for his divine blessing that made it possible to complete the project work successfully.

I am quite grateful to my supervisor, Dr. Ruchi Verma, Asst. Prof. Senior Grade, Department of CSE Jaypee University of Information Technology, Waknaghat, for her assistance. To complete this assignment, my supervisor has extensive knowledge and a deep interest in the subject of Machine Learning. His never-ending patience, intellectual direction, constant encouragement, constant and energetic supervision, constructive criticism, good suggestions, and reading many poor versions and fixing them at all stages made it possible to finish this job.

I'd like to thank Dr. Ruchi Verma, Department of CSE, for her invaluable assistance in completing my project.

I would also like to express my gratitude to everyone who has directly or indirectly assisted me in making this project a success. In this unique scenario, I'd want to appreciate the different staff members, both teaching and non-teaching, who have developed their helpful assistance and facilitated my project. Finally, I must express my gratitude for my parents' unwavering support and patience.

Sanya Kanwar(181465)

Stuti Gupta(181467)

Table of Contents

ABSTRACT

Plant disease detection is crucial in agriculture, as farmers must frequently make decisions such as whether or not the crop is sufficient. These can produce major system difficulties that impair each product's quality, quantity, or productivity, and they should be treated carefully. Plant diseases produce frequent disease outbreaks, result in large numbers of deaths, and have significant economic implications. These issues must be addressed quickly in order to save lives and money. Plant disease categorization by machine is a hot research issue. Because it is critical to be able to recognize the disease symptoms that occur on the plant's leaves when monitoring a vast region and at an early stage. This enables image-based automated inspection using computer vision techniques.

Manual plant disease identification, on the other hand, is a time-consuming, incorrect operation that can only be done in a small area at a time. Plant illnesses can be detected early this way, and pest and infection management techniques can be utilized to solve pest problems while reducing hazards to humans and the environment, or so they believed. We describe the current trends and difficulties of plant leaf diseases using extensive research and modern imaging techniques. We hope that, for all intents and purposes, this research will be of enormous use to researchers studying plant and insect illnesses, or so they thought. Also, we talked about some of the most pressing concerns and issues that needed to be handled, or so they believed.

Chapter 01-Introduction 1.1 Introduction:

For the world's health and well-being, accurate identification of plant diseases is critical. It is critical to recognise illnesses, including early prevention, in this ever-changing environment to avoid problems that may arise in other ways. Some of these issues, such as worldwide food shortages, could have catastrophic effects for humanity. In order to respond to climate change and achieve a better lifestyle, it is critical to avoid wasteful waste of financial resources from an environmental standpoint. Humans find it difficult to notice the effects of several plant diseases with the naked eye. It is tedious and ineffective to repeat this process over and over.

Plant pathologists must have outstanding observation skills in order to identify distinctive symptoms and detect plant diseases accurately. Plant disease detection techniques that are based on the look and visual symptoms of the plant can be quite useful. It may be used to automate the entire pipeline in the agricultural sector. Because these redundant jobs boost machine performance over humans, this not only improves efficiency but also operational productivity. Using deep learning and computer vision technology, we tackle the aforementioned problem of automating the classification of plant diseases.

With the benefits of automated learning and feature extraction in recent years, there has undoubtedly been considerable worry in both academic and industrial circles. It's utilised a lot in image and video processing, as well as voice and natural language processing. Simultaneously, it has essentially transformed into a hotspot of research in the field of agricultural protection, such as the detection of plant diseases and pest control, etc. In-depth understanding can surely prevent major flaws created by plantbased selection when it comes to identifying plant illnesses. Dot pathogenesis enhances the effectiveness of research and the speed of technological progress by making plant disease more purposeful.

This paper examines the considerable advancement of in-depth research technologies in the field of leaf spot recognition in recent years. Plant disease outbreaks have a significant detrimental influence on agricultural production. Malnutrition will rise if plant diseases are not detected early. Early identification is essential for effective plant disease prevention and control, and it plays a vital role in agricultural production management and decision-making. Plant disease identification has become a big challenge in recent years, albeit in a subtle way.

Agriculture is a sector that has a significant impact on human existence and economic status. Agriculture is the primary source of income for approximately 58 percent of India's population. In terms of farm yields, India is ranked second in the world. Agriculture provided employment to more than 50 percent of the workforce in 2018, adding 18 – 20 percent to the country's GDP.

As a result, India has established itself as one of the leading countries in terms of agricultural yield and productivity. Given that agriculture employs the majority of the people, it is critical to understand the issues that this sector faces. Inefficient farming strategies and procedures, poor use of compost, manures, and fertilisers, insufficient water supplies, and many diseases attacking plants are only some of the issues that the agriculture area faces. Diseases are extremely destructive to plants' health, which has an impact on their growth. The attack of these many diseases on plants causes a significant reduction in yield performance, both in terms of quality and quantity.

Diseased plants account for 20-30% of all crop losses. As a result, detecting plant diseases is critical in order to avoid major losses in productivity, performance, and agricultural output. Because manual recognition takes a long time and is prone to errors, it results in incorrect treatment. Recent technological advancements and evolution have paved the road for plant disease detection and identification, as well as better treatment of damaged plants.

The current advancement in technology, as well as its evolution, has made it conceivable and practicable to detect and identify plant diseases, as well as to contribute to better plant treatment in the event of diseased conditions. The proposed plant leaf disease recognition system focuses on 14 plant kinds, including apple, blueberry , cherry, maize, grape, orange, peach, pepper, potato, raspberry,ksoybean, squash , strawberry, and tomato. This method is based on Deep Learning ideas, with convolutional neural networks (CNN) being used to create a statistical model that is applied to the input image and modifies it to identify output tags.

Plants' leaves, which are the most vulnerable, display disease symptoms first. From the beginning of their life cycle until they are ready to be harvested, the crops must be monitored for illnesses. The traditional naked eye observation approach was initially used to monitor the plants for diseases. This is a time-consuming strategy that requires professionals to manually check the crop fields. A variety of strategies have been used in recent years to produce automatic and semi-automatic plant disease detection systems, and automatic disease detection by simply looking at the symptoms on the plant leaves makes it easier and cheaper.

1.2 Objective:

This project seeks to address the flaws in existing disease diagnosis diagnostic papers by offering a review of the most recent research in the field of leaf disease diagnosis utilising image processing and Deep Learning study methodologies.

Automatic detection of plant disease is an important study topic because it might help monitor large fields of crops and, as a result, detect disease symptoms as soon as they occur on plant leaves.

Plants are susceptible to disease for a variety of reasons, including fertilisers, cultural practises, environmental conditions, and so on. These diseases reduce agricultural yield and, as a result, the economy.

Plant pathologists need good observation skills to detect plant diseases accurately. Because these redundant jobs boost machine performance over humans, this not only improves efficiency, but also operational productivity. Early identification is the foundation for effective plant disease prevention and control, and it plays a vital role in agricultural production management and decision-making.

We used the concept of transfer learning for classification. The main advantage of transfer learning is that it does not start the learning process from scratch, but rather applies patterns gained when solving a comparable problem. As a result, rather of beginning from scratch, the model takes advantage of past knowledge. A pre-trained model is widely used to represent transfer learning in image classification. A pre-trained model is one that has been trained to solve a problem similar to ours on a large benchmark dataset. We employed five models for pre-trained weights, including Inception v3.

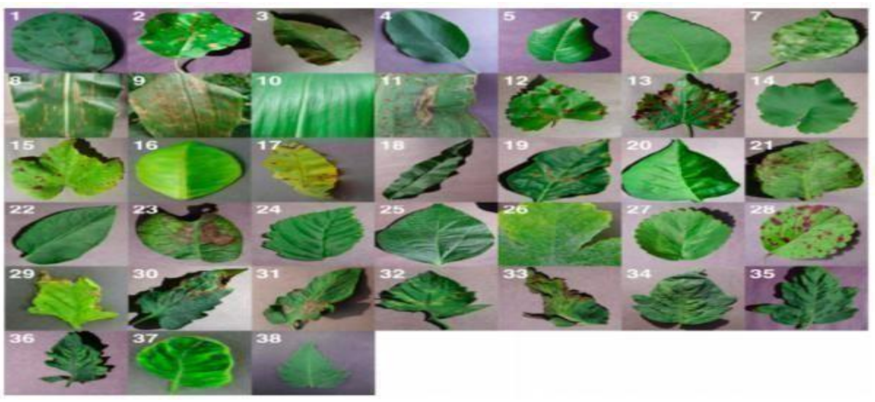
1.3 Problem Statement:

Agriculture is one field that has a significant impact on human lives and economic condition. Agricultural products are lost due to poor management. Diseases harm the health of the plant, which has an impact on its growth. It is critical to monitor the progress of the farmed crop to ensure minimal loss. Convolutional Neural Networks are a type of Deep Learning network that is commonly used for image classification, as well as other common tasks like image segmentation and signal processing. The major goal of the proposed work is to discover a solution to the challenge of detecting 38 distinct types of plant illnesses using the simplest approach while employing the fewest computational resources possible to obtain better outcomes than previous models.

1.4 Methodology:

DATASET:

- Plant village Dataset consists of Images of various plant leaves named Apple, Blueberry, Cherry, Corn, Grape, Orange, Peach, Pepper, Potato, Soybean, Squash, Strawberry and Tomato.
- The Plant Village dataset contains 54303 healthy and unhealthy leaf images, which are divided into 38 species and disease categories. We looked at over 50,000 images of plant leaves with labels from 38 different classes in order to predict disease classes. On this compressed image, we perform optimization and model predictions after resizing the image to 256 × 256 pixels.
- We looked at illnesses in tomato and potato plants, which had nine and two diseased subclasses, respectively, and one healthy subclass.
- The complete collection contains almost 20,000 photos of various diseasedAleaves of various kinds.
- Through the current web portal PlantVillage, we are announcing the availability of nearly 50,000 skillfully chosen photographs on healthy and sick leaves of crops plants. Both the data and the platform are described. These data are the start of a crowdsourcing project to enable computer vision technologies to help solve the problem of crop plant production losses caused by viral illnesses.



Data Processing and Augmentation

Building a good image classifier requires a lot of picture augmentation. Despite the fact that datasets can contain hundreds to thousands of training samples, the variety may not be adequate to develop an appropriate model. Flipping the image vertically/horizontally, rotating the image via various angles, and scaling the image are just a few of the many image augmentation choices. These enhancements help to boost the amount of meaningful data in a dataset. Each image in the Plant Village dataset has a resolution of 256 by 256 pixels. The Keras deep-learning framework is used for data processing and picture enhancement.

The following are the training enhancement options:

- Rotation - Rotate a training image in a random direction.
- Brightness - Assists the model in adapting to changes in lighting while training by feeding images of varied brightness.
- Shear - Change the angle of shearing.

SYSTEM OVERVIEW:

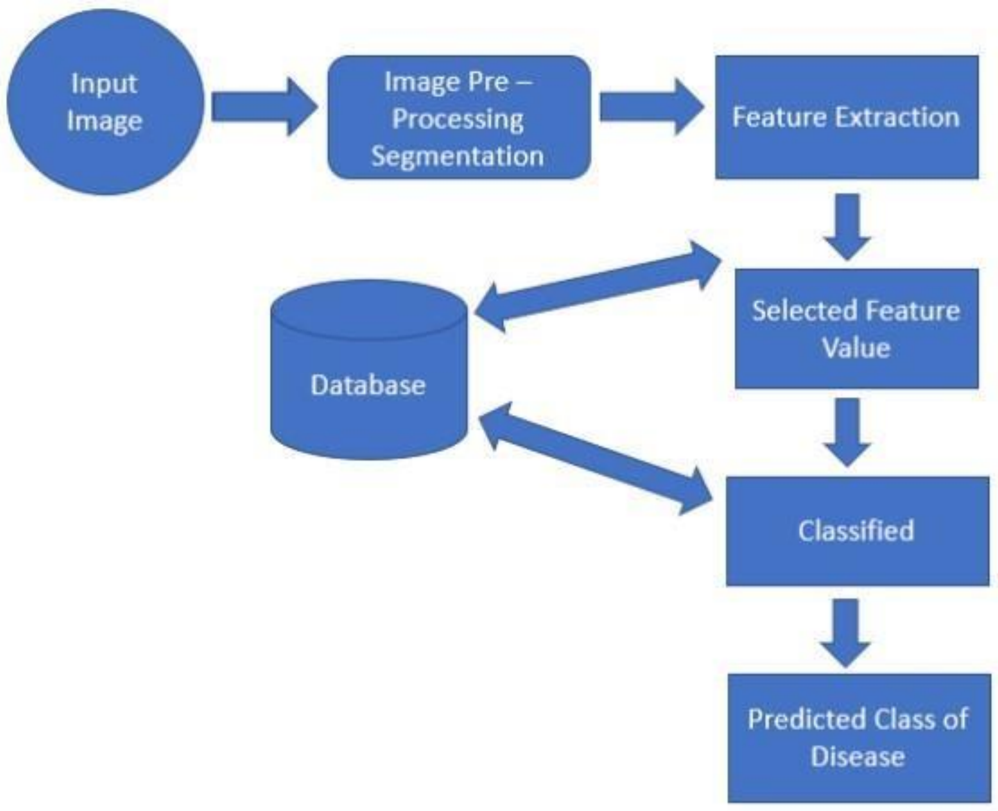


Image processing steps for detecting plant illnesses

The entire procedure is broken down into three stages:

1.The images are uploaded from the dataset, provided from Kaggle Plant Village Dataset.

2.Picture segmentation, image enhancement, and colour space conversion are all examples of pre-processing. First, a filter is applied to the digital picture of the image. Then, for each image, create an array. Each image name is converted to a binary field using the scientific term for Binarizes Diseases.

3. CNN classifiers are taught to recognise illnesses in different plant classes. The Level 2 results are used to activate a classifier that has been trained to classify various plant illnesses. The leaves are considered healthy if they are not present.

FLOW CHART:

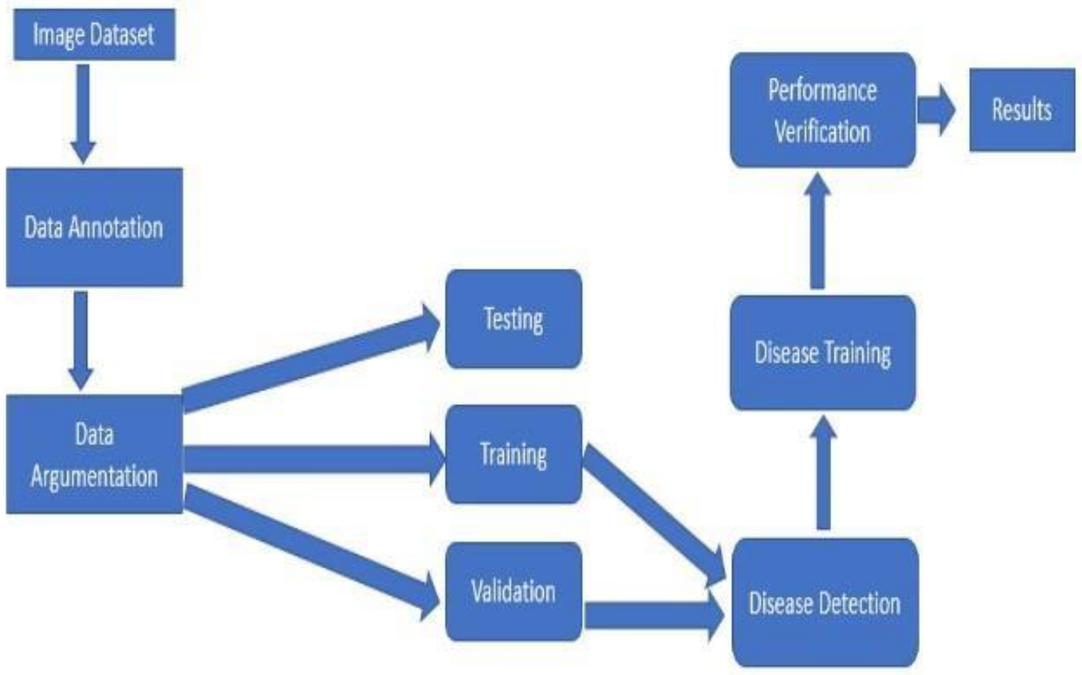


Image Dataset
Data Annotation
Data Argumentation
Testing
Training
Validation
Disease Detection
Disease Training
Performance Verification
Results

1.5 Organization:

The project report is broken down into 5 sections. The first chapter covers the background and motivation for the proposed application, the problem statement and aims to answer the issue statement, the recommended technique or research, and the highlighting of successful proposed applications. Chapter 2 illustrates the literature survey of the project from which we took the references. The system development chapter includes the site map, use case diagram, activity diagram, and system wireframe, which is the proposed application's user interface. Software design approach, tools, requirements, system performance specifications, and timescales are discussed in Chapter 4. The fifth chapter concludes the implementation, project evaluation, benefits and future scope of the project.

Chapter 02- Literature Survey

Authors	Year	Description	Outcomes
D.A. Bashish, et.al	2010	The author divided the leaf image into four clusters using k-means clustering, expecting that at least one of the clusters would have sick pixels. A CCM approach based on Spatial Gray-level Dependence Matrices was developed for feature extraction (SGDM). This strategy provides us with textural properties. The feed forward back propagation approach was used to classify the data using a neural network (BPNN).	The total disease detection and categorization accuracy of the system was determined to be around 93%. Using HS colour characteristics in BPNN , however, the best overall average accuracy of 99.66 percent was attained. Angular moment, mean intensity level, variance, correlation, product moment, contrast , and entropy were discovered to be texture oriented properties.
M.Bhange et.al	2015	When applied to large datasets, k-means clustering is more efficient, hence it was employed for segmentation in this work. The sum of squared distances were used to compare the colour histograms of two photographs. Morphology can be used to extract boundaries or the shape vector. When comparing photos with spatial information, CCV is used. SVM is utilised for classification in this study.	The authors were able to attain an overall system accuracy of 82 percent using SVM. Support vector machines take a long time to train, yet they are quite precise. As a consequence, three feature vectors, one for colour histogram, morphology, and CCV, were obtained.

V. Singh et.al	2016	To perform clustering, the genetic algorithm's search capacity was employed to divide unlabeled N-dimension points into K clusters. The texture and colour of an image have been evaluated in the CCM approach. Two techniques, one using kmeans clustering and the other with the Genetic algorithm , used the minimum distance criterion.	Local homogeneity, contrast , cluster shadow, energy , and cluster prominence are texture properties computed for the H picture. The minimum distance criterion has an accuracy of 86.54 percent with k-mean clustering and 95.71 percent with SVM. The accuracy was enhanced to 93.63 percent by combining the Genetic Algorithm with g the minimum distance criterion.
E.Kiani et.al	2017	There are five inputs and two outputs in the algorithm. Two of the inputs are about iron deficiency, while the other is about fungal infection. If the leaf is infected, the outputs refer to the two diseases.	Using the proposed approach, the authors of this paper were able to attain an overall system accuracy of 96 percent.
H. Ali et.al	2017	The distance between two colours is calculated using the delta E colour difference algorithm. For feature extraction, LBP (local binary patterns), RGB histogram, and HSV histogram features are used.	Rotation invariance is obtained from the RGB histogram. The lighting invariance induced by varied lightning circumstances is captured in the HSV histogram as a feature.
G. Saradhambal, et.al	2018	ThekOtsu technique assumes that the image has two types of pixels , forming a bimodal histogram with foreground and background pixels. Shape and texture oriented features were employed for feature extraction.	

Chapter 03 - System Development

LANGUAGE USED

The language used in this project is Python

PLATFORM USED:

Google colaboratory

LIBRARIES USED

- **Numpy :** Python's The NumPy library is the backbone of scientific computing. It's a Python library that includes a multidimensional array object, a number of derived objects (such as masked arrays and matrices), and a number of routines for performing fast array operations like mathematical, logical, shape manipulation, sorting, selecting, I/O, discrete Fourier transforms, basic linear algebra, basic statistical operations, random simulation, and more. In Python, we have lists that act like arrays, but they are slow to parse.

  NumPy promises a 50-fold faster array object than standard Python lists. The array array object in NumPy provides with a variety of auxiliary functions .

- **Pickle:** For Python object structures, the pickle module provides binary serialisation and de-serialization methods. Pickling converts a Python object hierarchy into a byte stream, whereas unpickling converts a byte stream (from a binary file or bytes-like object) back into an object hierarchy. Pickling (and unpickling) is also known as "serialisation," "marshalling," or "flattening," while the terms "pickling" and "unpickling" are used here to avoid misunderstanding. For Python object structures, the pickle module supports binary serialisation and de-serialization methods.

The process of transforming a Python object hierarchy into a byte stream is known as pickling.

The converse of Pickling is Unpickling, which is the process of transforming a byte stream into an object hierarchy.

- **CV2:** OpenCV is a free and open-source toolbox for computer vision, machine learning, and image processing. OpenCV supports a variety of programming languages, including Python, C++, Java, and others. It can recognise items, persons, and even human handwriting in photographs and videos. When used in conjunction with other libraries, such as Numpy, a very efficient library for numerical operations, the number of weapons in your arsenal increases, as every operation that Numpy can perform may be combined with OpenCV.

- This OpenCV tutorial will show you how to use a range of OpenCV programmes and projects to conduct image processing operations on photos and videos.

- **LabelBinarizer:** Labels should be binarized in a one-to-all basis.

Scikit-learn includes a number of regression and binary classification techniques. The so-called one-vs-all approach is a straightforward way to extend these algorithms to the multi-class classification problem.

This basically entails learning one regressor or binary classifier per class throughout learning time. To do so, multi-class labels must be converted to binary labels (belong or does not belong to the class). The convert method in LabelBinarizer makes this operation simple.

When it comes to prediction, one allocates the class to which the associated model provided the most confidence. The inverse transform function in LabelBinarizer makes this simple.

- **Keras:** Keras is a Python-based deep learning API that runs on top of the TensorFlow machine learning framework.It was created with the goal of allowing quick experimentation. It is critical to be able to move quickly from idea to outcome when conducting research.

Keras is:

Simple-Simple, yet far from simplistic. Keras relieves developer cognitive strain, allowing you to focus on the most important aspects of the problem.

Flexible-Simple processes should be quick and straightforward, but arbitrarily sophisticated workflows should be feasible via a clear route that builds on what you've already learned .

Keras is a deep learning model-level library that provides high-level building elements. Keras does not execute low-level calculations like Tensor products or convolutions; instead, it uses a specialised tensor manipulation library that is finetuned to operate as a backend engine. Keras has handled it so well that instead of adopting a single tensor library and executing operations on that library, it now allows alternative backend engines to be plugged into Keras.

- **Sklearn:** In Python The most useful and robust machine learning package in Python is Scikit-learn (Sklearn). It provides a set of quick machine learning and statistical modelling capabilities, including as classification, regression, clustering, and dimensionality reduction, using a Python consistency interface. This Python-based toolbox heavily relies on NumPy, SciPy, and Matplotlib.

- Scikit-learn is a free supervised and unsupervised machine learning software. It also has functions for model fitting, data preparation, model selection, and model assessment

- **OS:** Python Operating System module with functions for creating and deleting directories, retrieving their contents, modifying and defining the current directory, and more. It can also automate a variety of operating system processes.

- **Listdir:** In Python, the listdir() function is used to acquire a list of all files and folders in a given directory. If no directory is specified, the current working directory's list of files and folders will be returned.
  Python technique listdir() provides a list of the names of the entries in the path named directory. The order of the items is random. Even if they are present in the directory, it excludes the special entries '.' and '..'.

- **Matplotlib:** Matplotlib is a Python low-level graph charting toolkit that acts as a visualisation tool .John D. Hunter designed Matplotlib. Matplotlib is a Python charting toolkit, as well as its numerical mathematics extension NumPy. It provides an object-oriented API for incorporating plots into programmes that use general-purpose GUI toolkits such as Tkinter, wxPython, Qt, or GTK . There is also a procedural "pylab" interface based on a state machine (similar to OpenGL) that is intended to be similar to MATLAB, albeit its use is discouraged. Matplotlib is used by SciPy. He created Matplotlib at the beginning. It has had an active development community since then and is offered under a BSDstyle licence. Soon after John Hunter's death in August 2012. Michael Droettboom was selected as matplotlib's principal developer, and he was joined by Thomas Caswell. Matplotlib is a Num FOCUS-supported project. Matplotlib 2 .0.x is compatible with Python versions 2.7 through 3.10. Matplotlib 1.2 introduced Python 3 support .
  Matplotlib 1.4 is the most recent release that supports Python 2.6. By signing the Python 3 Statement, Matplotlib has vowed not to support Python 2 after 2020.Matplotlib is open source and free to use. Matplotlib is primarily written in Python, with a few pieces in C, Objective-C, and Javascript for platform compatibility .

METHODS:

There are four stages to the plant disease detection system procedure. Images are collected in the first phase using a digital camera, a cell phone, or the internet. The second phase divides the image into many clusters, each of which can be treated differently. The next part is about feature extraction methods, and the final phase is about illness classification.

Image capturing

Images of plant leaves are captured with digital media such as cameras , mobile phones, and other devices that have the required resolution and size. Images are also available for download on the internet.

The construction of an image database is exclusively the responsibility of the application system developer. The improved efficiency of the classifier in the detection system's final phase is due to the photo database.

Image Segmentation

This phase seeks to make an image's representation more intelligible and easier to examine by simplifying it . This phase is the foundation of feature extraction and the primary method to image processing. Images can be segmented using a variety of methods, including k-means clustering, Otsu's algorithm , and thresholding , among others. The k-means clustering classifies objects or pixels into K number of classes based on a collection of features . The classification is performed by minimising the sum of squares of distances between items and their respective groups.

The area of interest is the result of segmentation to this point. As a result, the characteristics from this area of interest must be retrieved in this stage. These characteristics are required to interpret a sample image. Color, shape, and texture can all be used to create features [14]. The majority of researchers are now planning to employ textural traits to detect plant illnesses. Gray-level co-occurrence matrix (GLCM), colour cooccurrence method , spatial grey-level dependence matrix, and histogram based feature extraction are some of the feature extraction approaches that can be used to construct the system. The GLCM method is a texture categorization statistical method.

Classification

The classification phase entails determining whether or not the input image is healthy. If a diseased image is discovered, several existing works have categorised it into a number of disorders. A software procedure, often known as a classifier, must be written in MATLAB for classification. k-nearest neighbour (KNN), support vector machines (SVM) , artificial neural network (ANN) , back propagation neural network (BPNN) , Naive Bayes , and Decision tree classifiers have all been employed by researchers in recent years. SVM is proven to be the most widely used classifier. Though each classifier has advantages and weaknesses, SVM is a straightforward and reliable technique .

Algorithm :

Deep Learning

Deep learning is a type of machine learning technique that use multiple layers to extract higher levels of information from raw data. Deep learning is a type of machine learning that instructs a computer to filter data across layers. Deep learning exemplifies the way the human brain filters information. Neural network architectures are used in many deep learning approaches. The term "deep" refers to the hidden layers that make up a neural network. Unlike traditional neural networks, which have two to three hidden layers, deep neural networks can have up to one hundred and fifty .

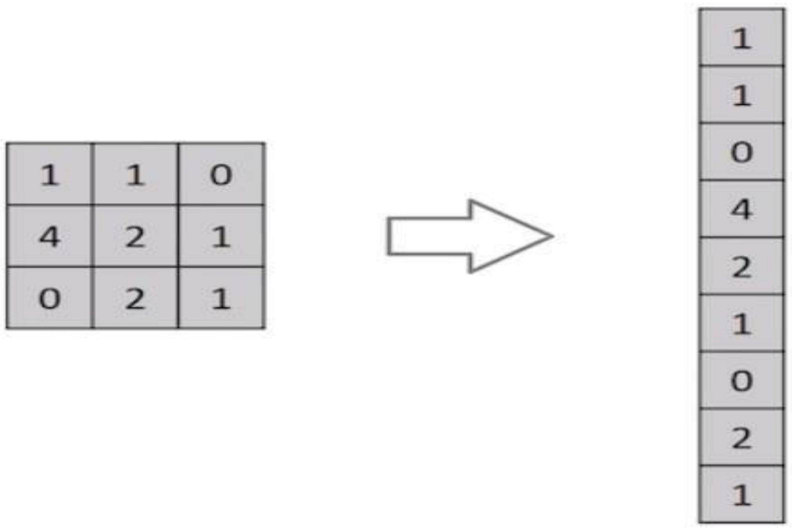
Convolutional neural networks

Convolutional neural networks are a subtype of deep neural networks (CNN) . A CNN mixes well-read characteristics with input data before applying 2D convolutional layers, making it more suitable for processing 2D data like as images. CNNs eliminate the need for human feature extraction and removal for picture classification. CNN's proprietary model extracts features directly from photos. The retrieved features are not pre-trained; they are well-read when the network is trained on a small number of image groups. The Convolutional Neural Network (CNN) model contains a number of layers that do image processing, including an input layer, an output layer, a convolutional layer, a fully convolutional layer, a soft-max layer, a connected layer, and a pooling layer.

A Convolutional Neural Network (ConvNet/CNN) is a Deep Learning method that can take an input image and give importance (learnable weights and biases) to various aspects/objects in the image, as well as identify one from the other. ConvNet requires significantly less pre-processing as compared to other classification algorithms. ConvNets can learn these filters/characteristics with enough training, whereas primitive techniques require hand-engineering of filters.

The design of a ConvNet is inspired by the Visual Cortex's organisation, which is analogous to the linking pattern of neurons in the human brain . Individual neurons respond to stimuli exclusively in the Receptive Field, a tiny region of the visual field .

Why ConvNets over Feed-Forward Neural Nets ?

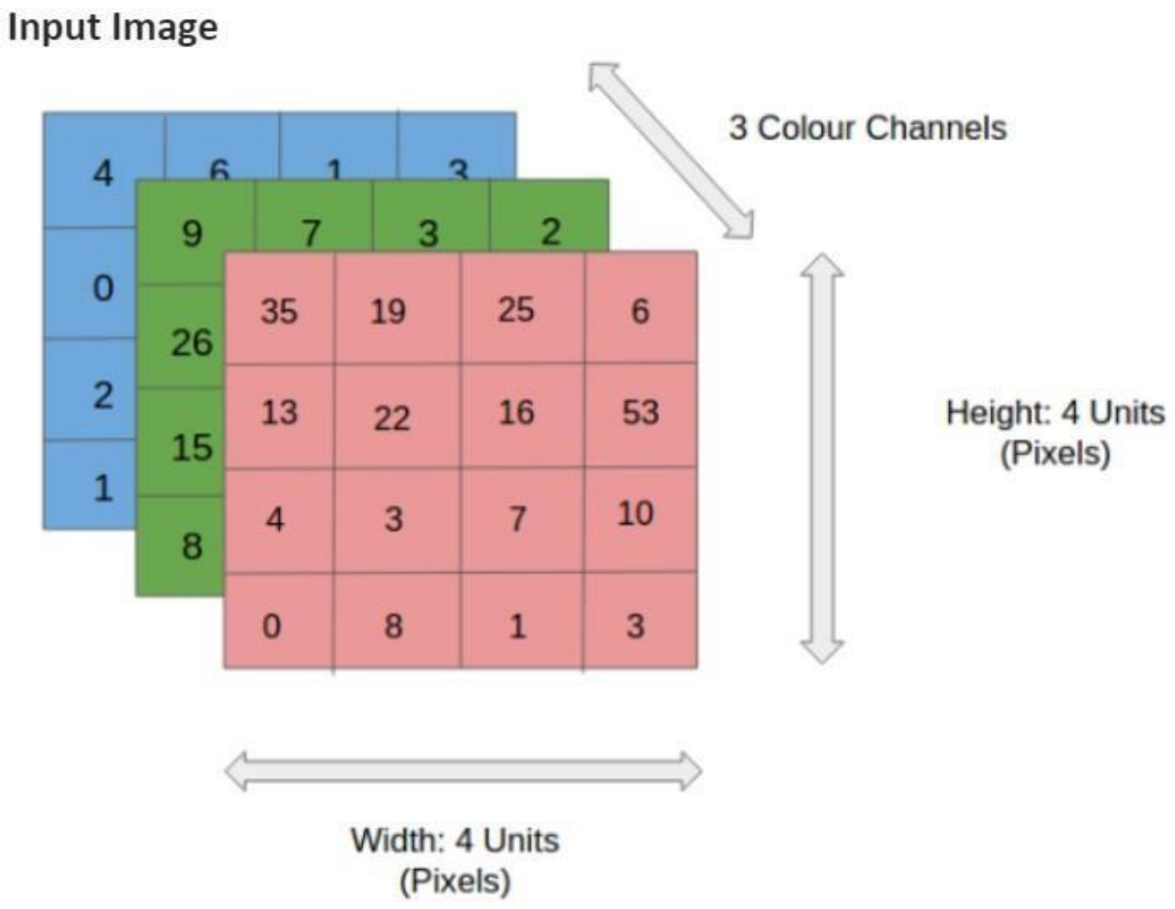


Flattening of a 3x3 image matrix into a 9x1 vector

Isn't a picture just a matrix of pixel values? Why not simply flatten the picture (for example, a 3x3 image matrix into a 9x1 vector) and pass it to a Multi-Level Perceptron for classification ? No, not really.

In the case of exceedingly simple binary pictures, the approach may display an average precision score when doing class prediction, but it will have little to no accuracy when dealing with complicated images with pixel dependencies throughout.

A ConvNet may successfully capture the Spatial and Temporal links in a picture by using appropriate filters. The architecture provides better fitting to the picture dataset because to the reduced number of parameters involved and the reusability of weights.



4x4x3 RGB Image

The picture in the graphic is an RGB image divided into three colour planes: red, green , and blue. Images can exist in a variety of colour spaces , including grayscale, RGB , HSV , CMYK, and others.

You can imagine how computationally demanding things would be once the photos reached, say, 8K (76804320). The ConvNet's function is to compress the pictures into a format that is easier to process while retaining elements that are important for generating a decent prediction. This is critical for designing an architecture that is not just effective at learning features but also scalable to large datasets.

Convolution Layer — The Kernel

Convoluting a 5x5x1 image with a 3x3x1 kernel to get a 3x3x1 convolved feature

5 (Height) x 5 (Breadth) x 1 (Image Dimensions) (Number of channels, eg. RGB)

The green portion in the preceding illustration is similar to our 5x5x1 input picture, I. The Kernel/Filter, K, represents the element involved in carrying out the convolution operation in the initial half of a Convolutional Layer. K has been chosen as a 3x3x1 matrix.

The Kernel shifts 9 times because Stride Length = 1 (Non-Strided), each time performing a matrix multiplication operation between K and the region P of the picture over which the kernel is hovering.

Movement of the Kernel

The filter moves to the right with a given Stride Value until it has processed the entire width . It then returns to the beginning of the image (left) with the same Stride Value and repeats the method until the full image has been traversed.

Convolution operation on a MxNx3 image matrix with a 3x3x3 Kernel

Kernel will have the exact depth as we have for the input picture for the cases having many channels (e.g., RGB). Matrix Multiplication is conducted between the Kn. In stacks ([K1, I1], [K2, I2], and [K3, I3], and the results are combined with the bias to produce a flattened one-depth channel Convoluted Feature Output.

Convolution Operation with Stride Length = 2

The purpose of the Convolution Operation is to extract high-level properties such as edges from the input image. Convolutional Networks don't have to be limited to a single Convolutional Layer. The first ConvLayer is traditionally responsible for gathering LowLevel data such as edges, colour, gradient direction, and so on. The design adapts to the High-Level features as more layers are added, giving us a network that knows the photographs in the dataset as well as we do.

The process yields two types of results: one in which the convolved feature's dimensionality is lowered in contrast to the input, and another in which the dimensionality is either raised or constant.

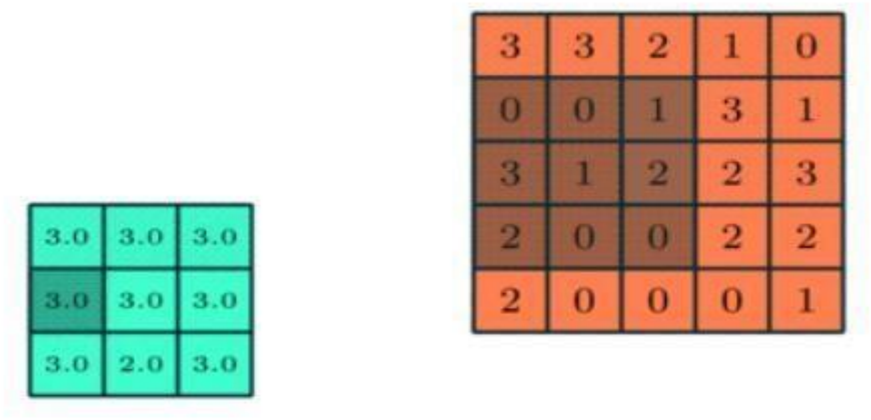
SAME padding: 5x5x1 image is padded with 0s to create a 6x6x1 image

When we augment the 5x5x1 picture into a 6x6x1 image and the apply the 3x3x1 kernel to it, the convolved matrix has dimensions of 5x5x1. As a result, Same Padding was born.

If we execute the same procedure without padding, we are presented with a matrix with the same dimensions as the Kernel (3x3x1) — Valid Padding.

The following repository has numerous such GIFs that will help you understand how Padding and Stride Length work together to generate outcomes that are relevant to our requirements .

Pooling Layer



3x3 pooling over 5x5 convolved feature

Like the Convolutional Layer, the Pooling Layer is responsible for reducing the Convolved Feature's spatial size. The computer power required to process the data is lowered because to dimensionality reduction. It's also useful for extracting rotational and positional invariant dominant traits, which makes it easier to train the model.

Maximum pooling and average pooling are the two forms of pooling. The largest value from the Kernel-covered region of the image is returned by Max Pooling. The average of all the data from the region of the picture covered by the Kernel is returned by Average Pooling.

Max Pooling can also be used to reduce noise. It filters out all noisy activations and does both denoising and dimensionality reduction. Average Pooling, on the other hand, is only a noise suppression approach that reduces dimensionality. As a consequence, we may infer that Max Pooling performs better than Average Pooling.

Pooling variety

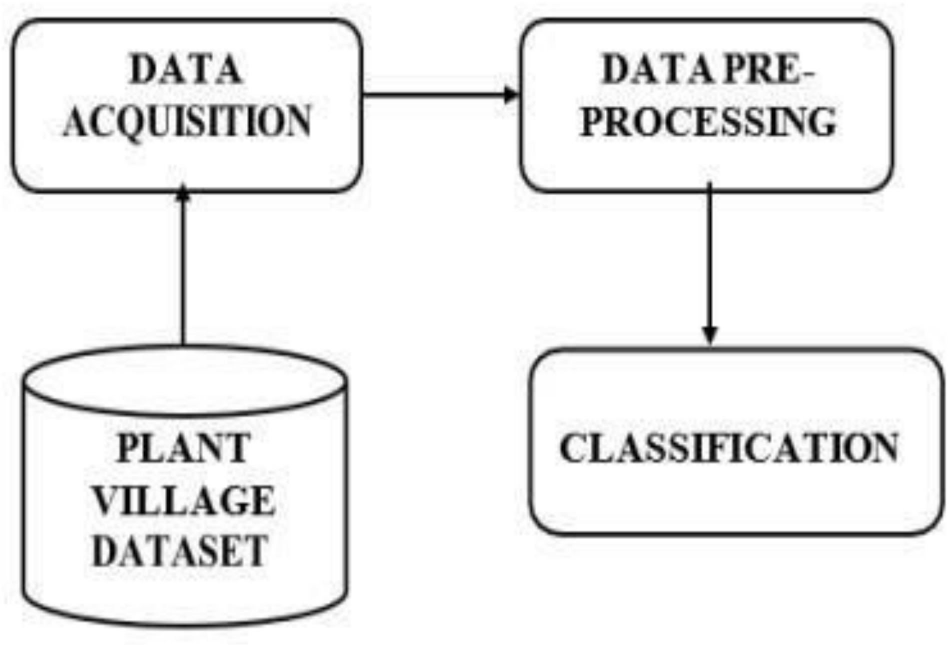
The Convolutional Layer and the Pooling Layer make up the i-th layer of a Convolutional Neural Network. Depending on the photo's complexity, the number of such layers may be increased to collect even more low-level information, but at the cost of more processing power.

After using the aforementioned strategy, we were able to effectively enable the model to comprehend the characteristics. After that, we'll flatten the final output before feeding it to a conventional Neural Network for classification.

System Architecture

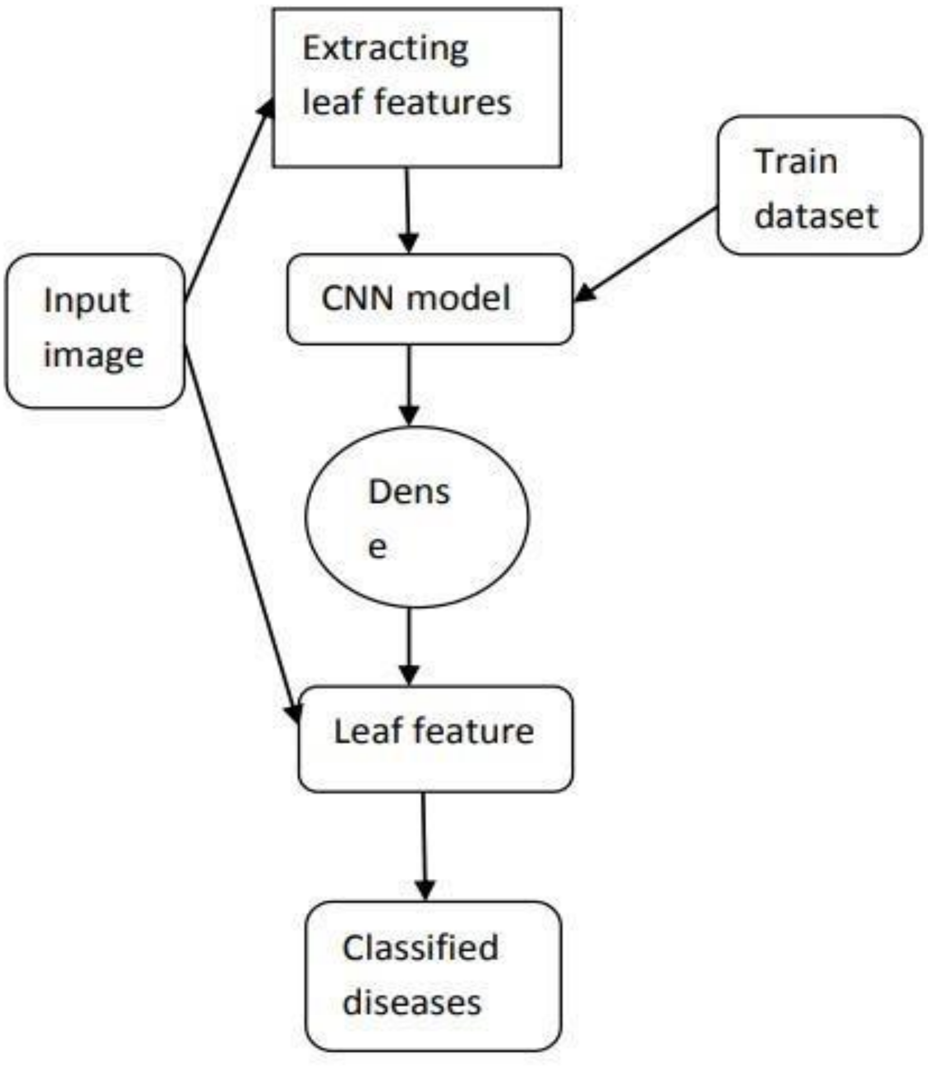
The suggested system architecture includes data collecting from a large dataset, processing at multiple convolutional layers, and then plant disease classification, which determines if a plant image is healthy or diseased.

To create an image processing system to automate the detection and classification of leaf batches into distinct illnesses in order to determine the cause of the symptom using an automatic tool. The system is made up of three basic pieces, as depicted in the diagram above: Image Analyzer, FeaturehDatabase, and Classifier [9]. The suggested processing for these blocks is divided into two parts as follows: offline Phase 1: A picture analyzer is used to extract anomalous features from a large number of defective photographs.



Use Case Diagram

The module extracts the leaf features initially when we supply a new input image. The CNN model is then applied. It then compares the features to a dataset that has already been trained. After that, it goes via dense CNN, which extracts the leaf features separately. The module will then determine whether or not the plant leaf is infected with disease. It displays the result from one of the 38 specified and trained classes. After that, the output will be in a textual format.



DATA FLOW DIAGRAM

DFDs (Data Flow Diagrams) depict the processes of data transfer from the input to the prediction of the related output.

1. The DFD Level 0 shows the users entering a picture of plant leaves. In turn, the system identifies and recognises plant leaf disease.

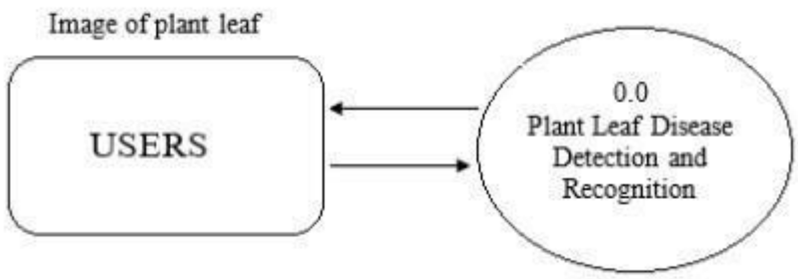


Fig1. Data flow diagram level 0

2.The DFD Level 1 is depicted in Figure 1, where the CNN model uses an image from the training dataset to forecast the type of leaf disease.

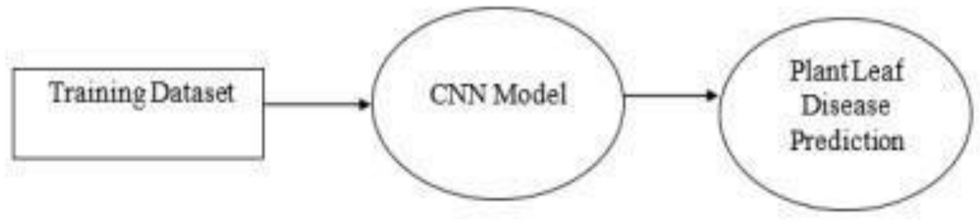


Fig 2. Data flow diagram level 1

3. DFD Level 2 delves even deeper into aspects of DFD Level 1.
   It can be used to design or record all of the specific/necessary information regarding how the system works.

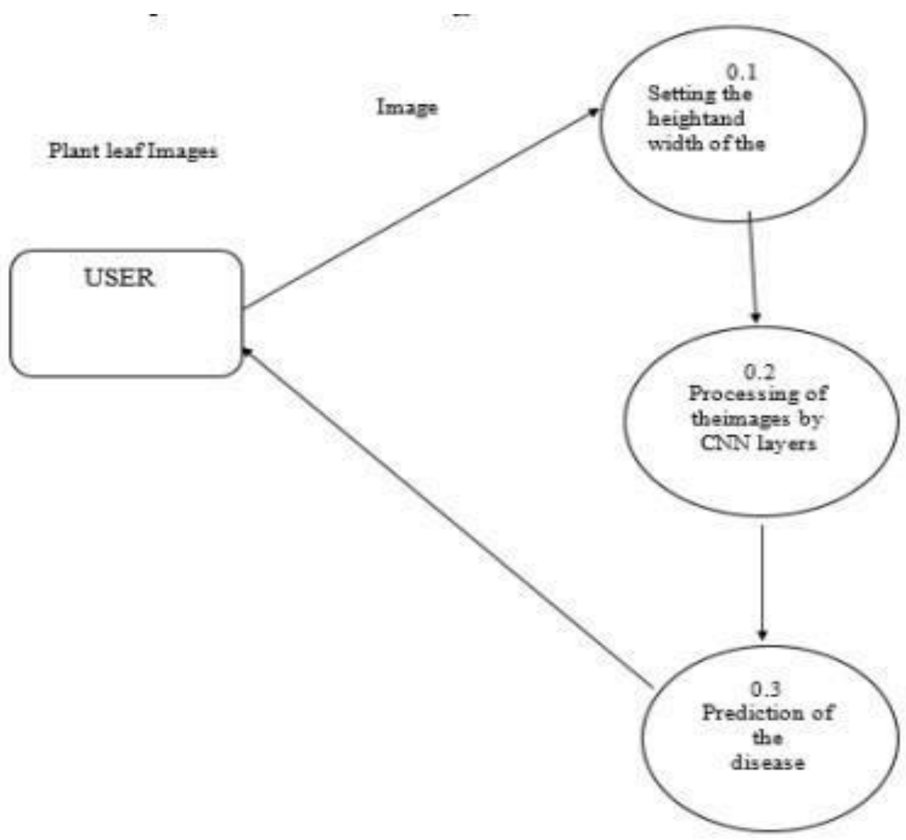

Plant leaf Images
Image
USER
0.1
Setting the heightand width of the
0.2
Processing of theimages by CNN layers
0.3
Prediction of the disease

Fig 3. Data flow diagram level 2

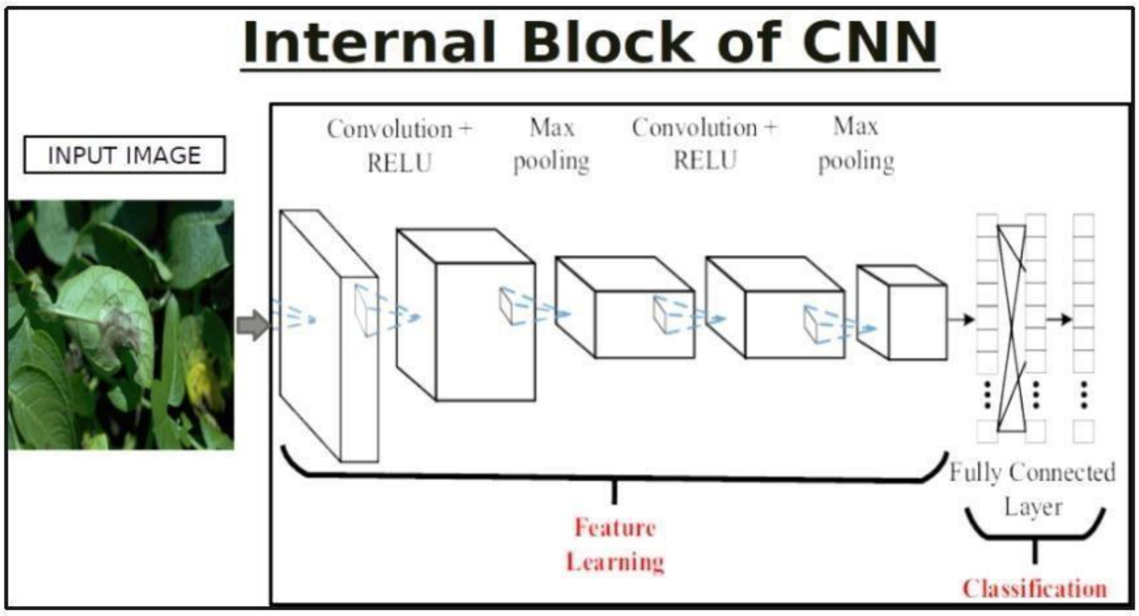
Chapter 4 – Performance Analysis

Data Analysis

The Plant Village dataset, which was downloaded from the Kaggle website and consists of images of diseased and healthy plant leaf images, was used in this proposed system project.

We discovered that the dataset had no missing values after further investigation. The data was further examined to learn more about the many spices and diseases that affect plant leaves. There were 14 different plant varieties in the sample. There are 54305 pictures in total in the training dataset.

CNN MODEL



CNN Architecture.

Batch Normalization:

Batch normalisation is a technique for normalising data between layers of a Neural Network rather than in the raw data. Instead of using the entire data set, minibatches are used. It facilitates learning by speeding up training and utilising higher learning rates.

Batch normalisation is a transformation that keeps the mean output close to 0 and the standard deviation of the output close to 1.

Batch normalisation standardises the inputs to a layer for each mini-batch while training very deep neural networks. This stabilises the learning process and reduces the number of training epochs required to build deep networks considerably.

During training and inference, batch normalisation works differently. In the sense that these are impossible, Batch Normalization (BN) does not prevent the vanishing or exploding gradient problem. Rather, it lowers the chances of these happening.

Conv2D:

This is the layer that convolves the image into numerous images and activates them. This layer generates a tensor of outputs by convolving the layer input with a convolution kernel . A bias vector is constructed and appended to the outputs if use bias is True. Finally, if activation is not None, the outputs are activated as well.

Provide the keyword parameter input shape (tuple of integers or None, does not include the sample axis) when using this layer as the first layer in a model, e.g. input shape=(128, 128, 3) for 128x128 RGB photos in data format="channels last." When a dimension's size is changeable, you can use None. **Arguments**

filters: integer, the output space's dimensionality (i.e. the number of output filters in the convolution).

The height and breadth of the 2D convolution window are specified by kernel size, which is an integer or a tuple/list of two numbers. To express the same value for all spatial dimensions, a single integer can be used.

strides: An integer or a tuple/list of two numbers indicating the convolution's height and breadth strides. To express the same value for all spatial dimensions, a single integer can be used. Any stride value less than 1 is incompatible with any dilation rate value less than 1. bias_regularizer: The bias vector was subjected to the regularizer function (see keras.regularizers).

activity_regularizer:The layer's output (its "activation") is subjected to a regularizer function

(see keras.regularizers). kernel_constraint: Constraint function applied to the kernel

matrix

(see keras.constraints). bias_constraint: Constraint function applied to the bias vector

(see keras.constraints).

Kernel: In image processing, a kernel is a convolution matrix or masks that can be used to blur , sharpen , emboss , identify edges, and more by convolutioning a kernel with an image .

Activation Functions:

The valuesothat represent the image are processed via an activation function or activation layer once the image's feature map has been constructed. Because images are non-linear, the activation function takes values that represent theoimage, which are in a linear form (i.e. just a list of numbers) thank to the convolutional layer , and amplifies their nonlinearity .

A Rectified Linear Unit (ReLU) is the most common activation function used for this, but other activation functions are also infrequently utilised.

In a convolutional neural network, a non-linearityolayer is made up of an activation function that takes the feature map generated by the convolutional layer and outputs the activation map.

An activation function in a neural network explains how the weighted sum of input is transformed into an output from a node or nodes in a layer. A neuron's activation status is determined by its Activation Function. During the prediction phase, it will employ simpler mathematical operations to decide if the neuron's input to the network is necessary or not. The activation function is a node that is placed at the end or in the middle of a Neural Network . They aid in determining whether or not the hneuron will fire.

The activation function is a nonlinear transformation of the inputosignal. The following layer of neurons receives this altered output as input.

There are various types of activation functions; however, for the sake of this essay, I will concentrate on Rectified Linear Units (ReLU).

The ReLU function is the most often used activation function in today's neural networks. ReLU has a number of benefits over other activation functions, including the fact that it does not excite all neurons at the same time. The ReLU function above, converts all negative inputs to zero and does not activate the neuron . It is extremely computationally efficient since just a few neurons are activated at a time. The positive zone does not reach a saturation point. In practise, ReLU converges six times quicker than tanh and sigmoid activation functions. One disadvantage of ReLU is that it is saturated in the negative regiong, implying that the gradient there is zero.When the gradient is 0 during backpropagation , all of the weight are not updated; to fix this, we use Leaky ReLU . ReLU functions are also not zero-centered. This means it'll have to follow a zig-zag path to get to its optimal location, which might take a long time.

Pooling Layers:

The data is then transmitted through a pooling layer after it has been triggered. Pooling "downsamples" an image, which means it compresses the information that makes up the image, making it smaller. The network becomes more flexible and adept at recognising objects/imageshbased on relevant attributes thanks to the pooling process.

When we look at a photograph, we are usually just interested in the aspects we care about, such as people or animals, and not the background information.

A pooling layer in a CNN, similarly, will abstract away the unneeded bits of the image, preserving just thehparts it believes arehrelevant, ashdetermined by the pooling layer's size.

The hope is that the network will only learn the bits of the image that really depict the object in question since it must make judgements about the most relevant sections of the image. This prevents overfitting, which occurs when the network learns too much about the training case and fails to generalise to fresh data.

In a CNN design, the Pooling layer is visible between the Convolution layers. This layer essentially minimises the number of parameters and computations in thejnetwork, preventing overfitting by graduall shrinking the network's spatial size.

This layer has two operations: average pooling and maximum pooling. In this post, just Maxpooling will be discussed.

Max-pooling as the name implies, will only take the maximum amount fromha pool. This is accomplished by sliding filters through the input, with the maximum parameter being removed and the remainder being discarded at each step. This actually reduces the network's resolution.

The pooling layer, unlike the convolution layer, does not affect the network's depth; the depth dimension remains unchanged.

MaxPooling2D:

This procedure is repeated for the next two levels to max pool the value from a certain sizehmatrix. By obtaining the maximum value for each channel of the input over an input window, downsamples the input along its spatial dimensions (height and width) (of size determined by pool size). Strides are used to adjust the window's dimensions. The following spatial form (number of rows or columns) is obtained when using the "valid" padding option: output shape = math.floor((input shape - pool size) / strides) + 1 (when input shape >= pool size). When using the "same" padding option, the output shape is: output shape = math.floor((input shape - 1) / strides) + 1. For instance, if strides = (1, 1) and padding ="valid" are used.

Arguments

- **pool_size**: window size, which might be an integer or a tuple of two numbers. Over a 2x2 pooling window, (2, 2) will take the maximum value . If only one integer is supplied, the window length for both dimensions will be the same.

- **strides:** None, integer, tuple of two integers Values of strides For each pooling step, specifies how far the pooling window travels. If none is specified,hpool size is used byhdefault.

- **padding:** "Valid" or "same" are two options (case-insensitive). "Valid" denotes the absence of padding. "same" pads the input uniformly to the left and right, or up and down, so that the output has the same height and width dimensions as the input.

- **data_ format**: One of channels last (default) or channels first (optional). The dimensions in the inputs are ordered. Inputs having shape (batch, height, width, channels) are assigned to channels last, while inputs without shape are assigned to channels first (batch, channels, height, width). The image data format setting in your Keras config file at /.keras/keras.json is used by default. If you don't set it, it defaults to "channels last."

Flatten:

This function flattens the dimensionality of a picture after it has been convolved. Dense: This is the hidden layer that is used to make this a fully connected model. Dropout is employed to avoid overfitting on the dataset, whereas dense means that the output layer only has one neuron that determines which category each image belongs to. The input is flattened. The batch size is unaffected . Flattening adds an extra channel dimension and output shape is affected if inputs are shaped (batch) without a feature axis (batch, 1).
Arguments

data format: A string with one of the following values: channels last (default) or channels first. The dimensions in the inputs are ordered. Inputs of shape (batch,...) correspond to channels last, while inputs with shape (batch,...) belong to channels first (batch, channels, ...). The image data format setting in your Keras config file at / .keras/ keras.json is used by default. If you don't set it, it defaults to "channels last."

Dropout:

Dropout is applied to the input.

The Dropout layer, which helps minimise overfitting, sets input units to 0 at random with a rate frequency at each step during training time. Inputs that are not set to 0 are scaled up by 1/(1 - rate) such that the total sum remains unaltered.

The Dropout layer only applies when the training parameter is set to True, which means no data are dropped during inference. When using model.fit, training is automatically set to True, and in other cases, you can manually set the kwarg to True when calling the layer.

(This is in contrast to a Dropout layer with trainable=False.) Because Dropout has no variables or weights that may be changed, trainable has no effect on the layer's behavior.

Arguments

- **rate:** Float the value between 0 and 1. The input unit fraction will decrease.
- **noise_shape:** The geometry of the binary dropout mask that will be multiplied with the input is represented by a 1D integer tensor. Use noise shape=(batch size, 1, features) if your inputs have shape (batch size, timesteps, features) and you want the dropout mask to be the same for all imesteps.
- **seed:** A random seed is a Python integer.

Dense:

It's just another densely connected NN layer.

Dense implements the operation output = activation(dot(input, kernel) + bias), where activation is the element-wise activation function supplied as the activation parameter, kernel is the layer's weights matrix, and bias is the layer's bias vector (only applicable if use bias is True). Dense has all of these characteristics.

Dense computes the dot product between the inputs and the kernel along the last axis of the inputs and axis 0 of the kernel if the input to the layer has a rank greater than 2. (using tf.tensordot). For example, if the input has dimensions (batch size, d0, d1), we generate a kernel with shape (d1, units) that works on every sub-tensor of shape (1, 1, d1) (there are batch size * d0 such sub-tensors) along axis 2 of the input. In this scenario, the output will be shape (batch size, d0, units).

Furthermore, once a layer has been called, its properties cannot be changed (except the trainable attribute). Keras will generate an input layer to insert before the current layer if a popular kwarg input shape is given. This can be regarded as the same as defining an InputLayer directly.

Image Data Generator

It rescales the image, applies shear in a certain range, zooms the image, and flips it horizontally. This Image Data Generator includes every possible image orientation. The Image Data Generator class in Keras makes it simple to enhance your photographs. It offers a variety of augmentation options, including standardisation, rotation, shifts, flips, brightness changes, and more. Image data augmentation is used to increase the size of the training dataset in order to improve the model's performance and generalisation capacity. The Image Data Generator class in the Keras deep learning toolkit supports image data augmentation.

To improve the model's performance and generalisation ability, image data augmentation is employed to expand the training dataset. The Image Data Generator class in Keras' deep learning library allows you to supplement image data.

The Image Data Generator will then generate 10 images in each training iteration. An iteration is defined as the totalfnumber of samples divided by batch size. In the example above, each training period will have 100 iterations.

Training Process:

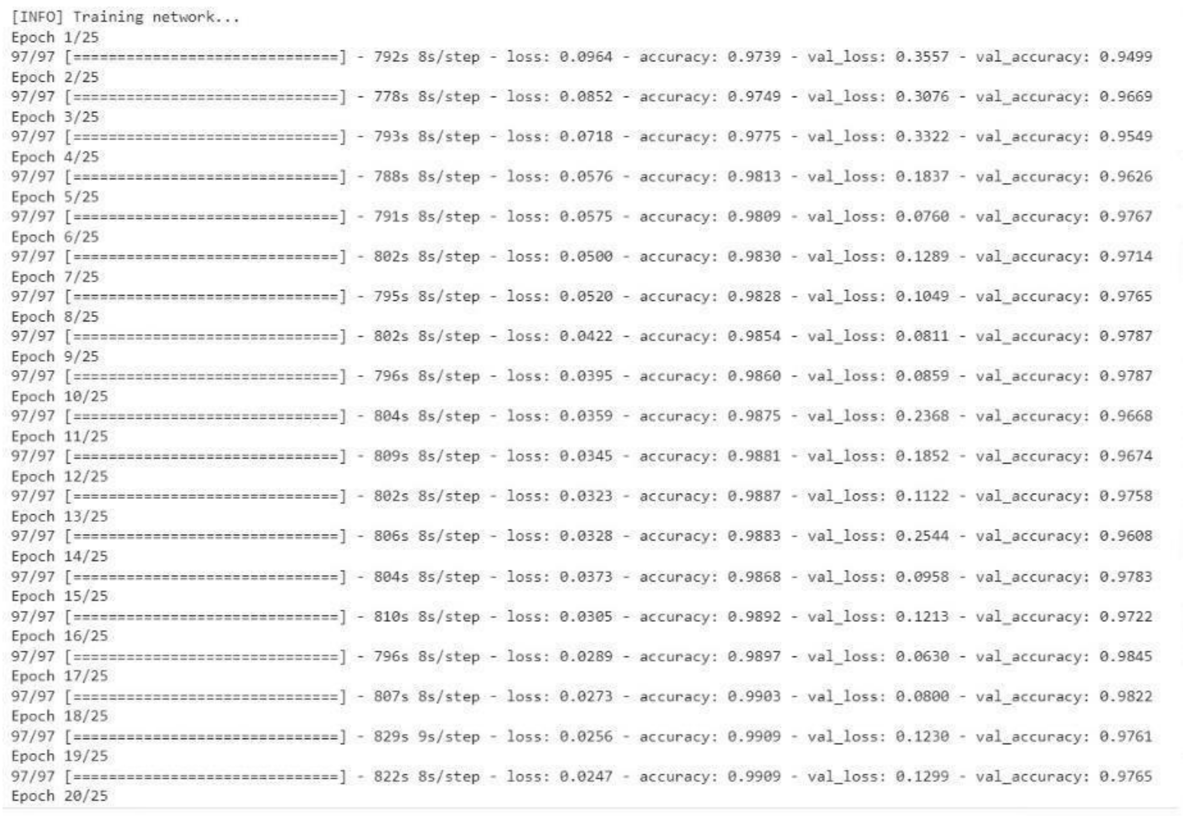
The function train datagen, flow from directory is used to prepare data from the train dataset directory. The image's target size is specified by target size. The flow from directory function is used to prepare test data for the model, and everything is the same as before.

Fit generator is used to fit the data into the model created above; additional factors include steps per epochs, which tells us how many times the model will run for the training data.

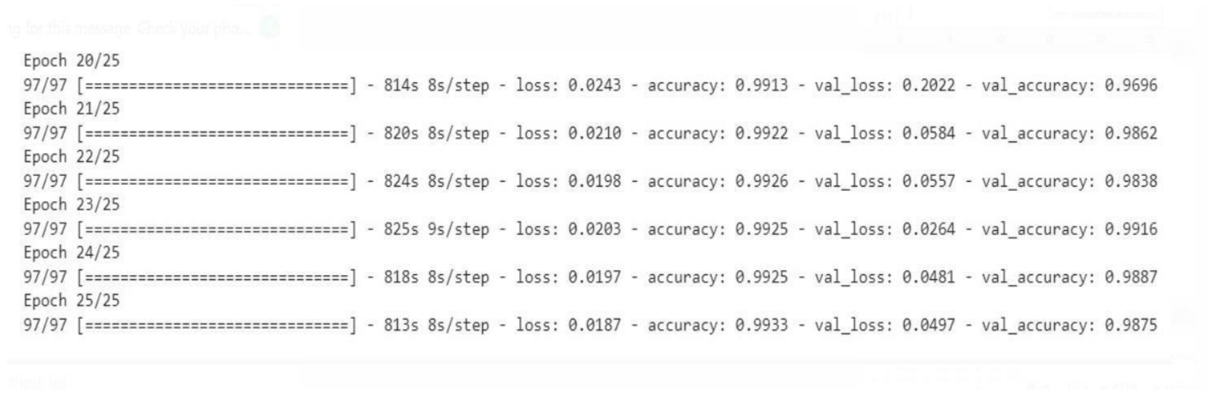
Epochs:

This indicates how many times the model will be trained in both forward and backward passes. When an entire dataset is only transported forward and backward through the neural network once, it is called an Epoch. We divide one epoch into many smaller batches since it is too large to provide to the computer all at once. An   epoch is a word used in machine learning that refers to the number of passes the machine learning algorithm has made across the full training dataset. Batches are commonly used to group data sets (especially when the amount of data is very large)

Steps per epoch is the number of samples to train per epoch. It specifies how many batches of samples should be used in a single epoch. It's used to signal the end of one epoch and the start of the next. You can ignore it if you have a fixed-size training set.



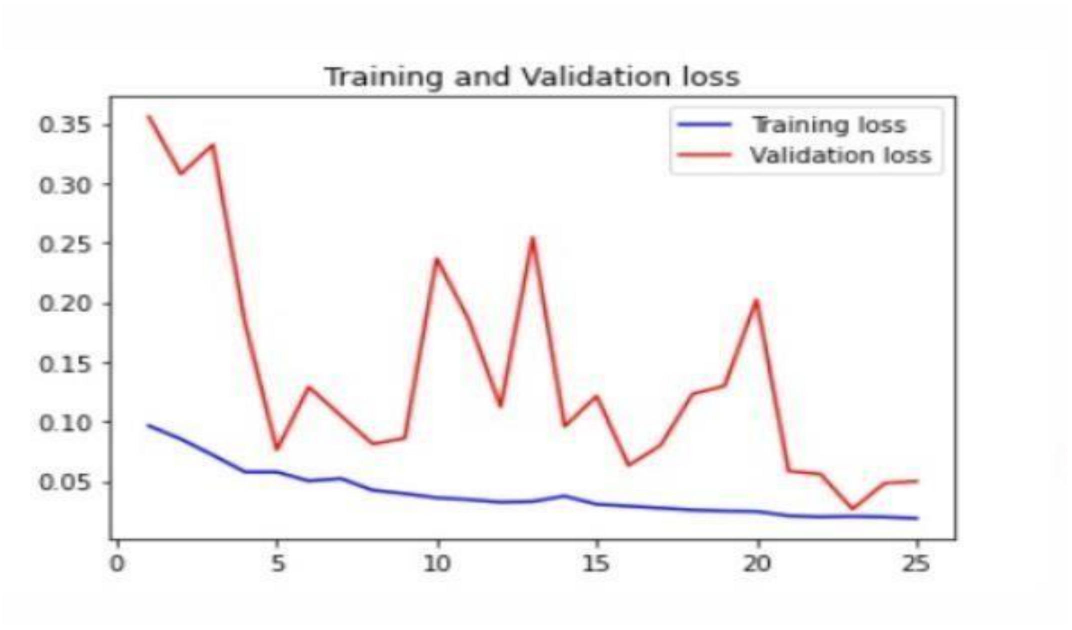
```
[INFO] Training network...
Epoch 1/25
97/97 [==============================] - 792s 8s/step - loss: 0.0964 - accuracy: 0.9739 - val_loss: 0.3557 - val_accuracy: 0.9499
Epoch 2/25
97/97 [==============================] - 778s 8s/step - loss: 0.0852 - accuracy: 0.9749 - val_loss: 0.3076 - val_accuracy: 0.9669
Epoch 3/25
97/97 [==============================] - 793s 8s/step - loss: 0.0718 - accuracy: 0.9775 - val_loss: 0.3322 - val_accuracy: 0.9549
Epoch 4/25
97/97 [==============================] - 788s 8s/step - loss: 0.0576 - accuracy: 0.9813 - val_loss: 0.1837 - val_accuracy: 0.9626
Epoch 5/25
97/97 [==============================] - 791s 8s/step - loss: 0.0575 - accuracy: 0.9809 - val_loss: 0.0760 - val_accuracy: 0.9767
Epoch 6/25
97/97 [==============================] - 802s 8s/step - loss: 0.0500 - accuracy: 0.9830 - val_loss: 0.1289 - val_accuracy: 0.9714
Epoch 7/25
97/97 [==============================] - 795s 8s/step - loss: 0.0520 - accuracy: 0.9828 - val_loss: 0.1049 - val_accuracy: 0.9765
Epoch 8/25
97/97 [==============================] - 802s 8s/step - loss: 0.0422 - accuracy: 0.9854 - val_loss: 0.0811 - val_accuracy: 0.9787
Epoch 9/25
97/97 [==============================] - 796s 8s/step - loss: 0.0395 - accuracy: 0.9860 - val_loss: 0.0859 - val_accuracy: 0.9787
Epoch 10/25
97/97 [==============================] - 804s 8s/step - loss: 0.0359 - accuracy: 0.9875 - val_loss: 0.2368 - val_accuracy: 0.9668
Epoch 11/25
97/97 [==============================] - 809s 8s/step - loss: 0.0345 - accuracy: 0.9881 - val_loss: 0.1852 - val_accuracy: 0.9674
Epoch 12/25
97/97 [==============================] - 802s 8s/step - loss: 0.0323 - accuracy: 0.9887 - val_loss: 0.1122 - val_accuracy: 0.9758
Epoch 13/25
97/97 [==============================] - 806s 8s/step - loss: 0.0328 - accuracy: 0.9883 - val_loss: 0.2544 - val_accuracy: 0.9608
Epoch 14/25
97/97 [==============================] - 804s 8s/step - loss: 0.0373 - accuracy: 0.9868 - val_loss: 0.0958 - val_accuracy: 0.9783
Epoch 15/25
97/97 [==============================] - 810s 8s/step - loss: 0.0305 - accuracy: 0.9892 - val_loss: 0.1213 - val_accuracy: 0.9722
Epoch 16/25
97/97 [==============================] - 796s 8s/step - loss: 0.0289 - accuracy: 0.9897 - val_loss: 0.0630 - val_accuracy: 0.9845
Epoch 17/25
97/97 [==============================] - 807s 8s/step - loss: 0.0273 - accuracy: 0.9903 - val_loss: 0.0800 - val_accuracy: 0.9822
Epoch 18/25
97/97 [==============================] - 829s 9s/step - loss: 0.0256 - accuracy: 0.9909 - val_loss: 0.1230 - val_accuracy: 0.9761
Epoch 19/25
97/97 [==============================] - 822s 8s/step - loss: 0.0247 - accuracy: 0.9909 - val_loss: 0.1299 - val_accuracy: 0.9765
Epoch 20/25
```



```
Epoch 20/25
97/97 [==============================] - 814s 8s/step - loss: 0.0243 - accuracy: 0.9913 - val_loss: 0.2022 - val_accuracy: 0.9696
Epoch 21/25
97/97 [==============================] - 820s 8s/step - loss: 0.0210 - accuracy: 0.9922 - val_loss: 0.0584 - val_accuracy: 0.9862
Epoch 22/25
97/97 [==============================] - 824s 8s/step - loss: 0.0198 - accuracy: 0.9926 - val_loss: 0.0557 - val_accuracy: 0.9838
Epoch 23/25
97/97 [==============================] - 825s 9s/step - loss: 0.0203 - accuracy: 0.9925 - val_loss: 0.0264 - val_accuracy: 0.9916
Epoch 24/25
97/97 [==============================] - 818s 8s/step - loss: 0.0197 - accuracy: 0.9925 - val_loss: 0.0481 - val_accuracy: 0.9887
Epoch 25/25
97/97 [==============================] - 813s 8s/step - loss: 0.0187 - accuracy: 0.9933 - val_loss: 0.0497 - val_accuracy: 0.9875
```

Process of Validation:

Validation/test data is fed into the model using validation data. The number of validation/test samples is indicated by validation steps.



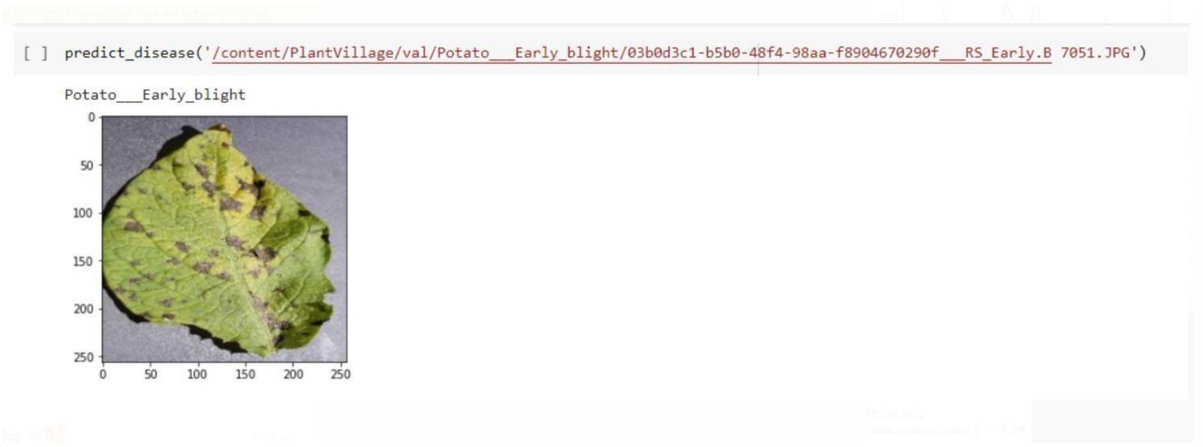
MODEL FOR TRAINING AND TESTING

The dataset is pre-processed, including image reshaping, resizing, and array conversion. The test image undergoes similar processing. A dataset of around 38 different plant leaf diseases is obtained, from which every image can be used as a software test image.

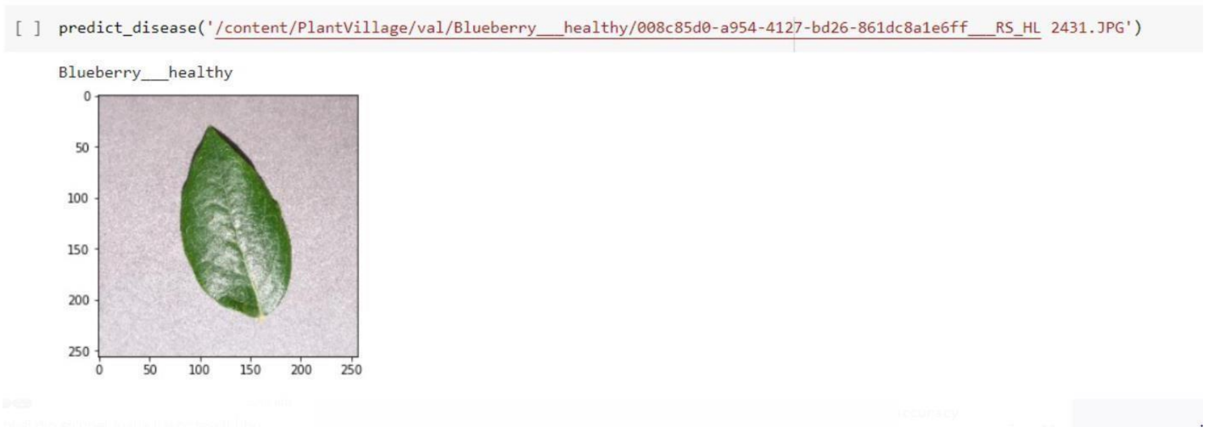
The train dataset is used to train the CNN model so that it can recognise the testpimage and the disease it is associated with.

Dense, Dropout, Activation, Flatten, Convolution2D, and MaxPooling2D are some of the layers in CNN.

If the plant species is present in the dataset, the software can identify the illness after the model has been successfully trained. Following successful training and pre processing, the test image and trained model are compared in order to forecast the disease.



```
[ ]  predict_disease('/content/PlantVillage/val/Potato___Early_blight/03b0d3c1-b5b0-48f4-98aa-f8904670290f___RS_Early.B 7051.JPG')

     Potato___Early_blight
```



```
[ ]  predict_disease('/content/PlantVillage/val/Blueberry___healthy/008c85d0-a954-4127-bd26-861dc8a1e6ff___RS_HL 2431.JPG')

     Blueberry___healthy
```

CHAPTER 5 – CONCLUSION

Discussion on previous Results achieved:

Because convolutional networks are known to be ready to learn features when trained on larger datasets, the results obtained when trained with only original photos will not be investigated. An overall accuracy of 88 percent was reached after fine-tuning the network's parameters. In addition, the trained model was tested on each class separately.
Every image from the validation collection was put to the test.

These methods are used to determine if the leaves are sick or healthy. The automation of the detecting system employing complicated photos acquired in outdoor lightning and extreme climatic circumstances is one of the obstacles in this approach. Despite significant drawbacks, this review paper indicates that theseodisease detection algorithms are efficient and accurate enoughpto runpthe systemlbuilt for the detection of leaf diseases. As a result, there is still much that may be done to improve existing works in this sector.

It looked at how images from a specific dataset (training dataset) in the field and previous data sets were utilised to forecast plant disease patterns using a CNN model. This leads to the following conclusions on plant leaf disease prediction. Because this system covers the most sorts of plant leaves, farmers may learn about leaves that have never been farmed before. It also lists all conceivable plant leaves, which aids farmers in deciding which crop to produce. Furthermore, this system takes into account previous data production, allowing the farmer to gain insight into market demand and costs for specific plants.

Because agriculture is so important to India's economy, it's necessary to detect and recognise leaf diseases that cause losses. This research uses a deep learning technique known as CNN to create a system that can identify, detect, and recognise 13 different plant leaf diseases. The disorders of seven classes were identified using a minimal set of layers in this method. Plant Village data was used to train the neural network.
The user can choose any image from the collection, which is then loaded, and the disease prediction is displayed on the User Interface. With rare exceptions, a convolutional neural network trained forpidentifying and recognising plant leaf disease could properly categorise and predict diseases for almost all photos.

The produced results should be compared to some other results, as advised by good practise standards. Furthermore, no commercial solutions exist, with the exception of those that handle plant species recognition based on leaf photos. The use of a deep learning system to automatically categorise and detect plant diseases from leaf photos was investigated in this research. The complete technique was presented, from gathering the images required for training and validation to image pre-processing and augmentation, and finally coaching and fine-tuning the deep CNN. Various tests were carried out to assess the performance of the newly constructed model. There was nog comparison with similar outcomes using thepprecise procedure because the proposed methodohas notpbeen used in the field of disease recognition, as far as we all know.
We've included a test image of a tomato leaf with Septoria leaf spot.

RESULT:

Because convolutional networks are known to be ready toplearn features when trained on larger datasets , the results obtained when trained with only original photos will not be investigated. An overall accuracy of 98 percent was reached after fine-tuning the network's parameters. In addition, the trained model was tested on each class separatelyd. Every image from the validation collection was put to the test. The produced results should bepcompared to some other results, as advised by good practise standards. Furthermore, no commercial solutions exist, with the exception of those that handle plant species recognition based on leaf photos. The use of a deep learningpsystem to automatically categorise and detect plant diseases from leaf photos was investigated in this research**.**

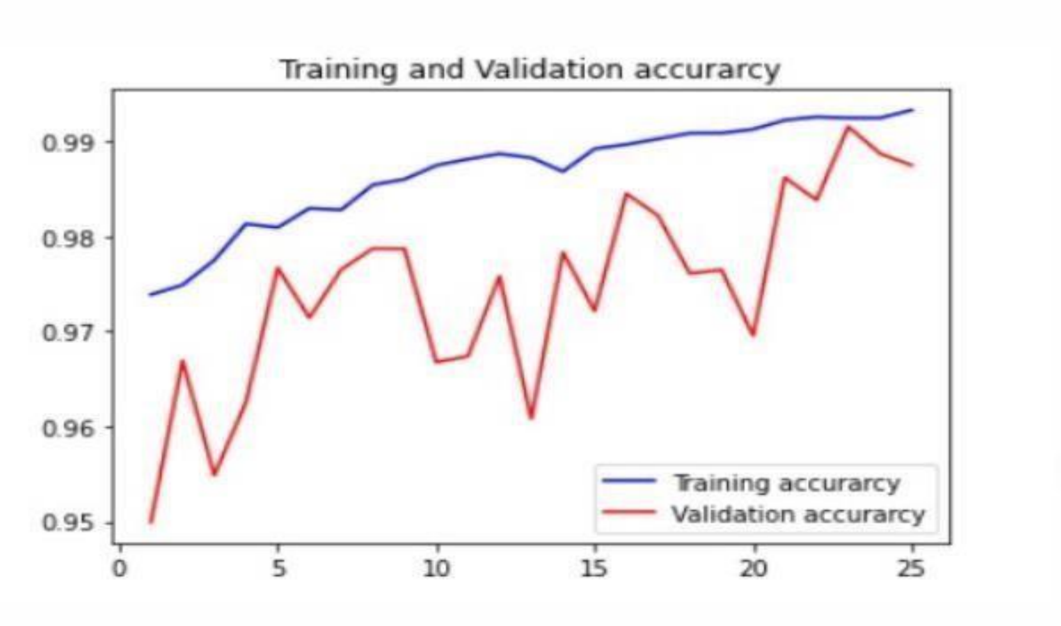
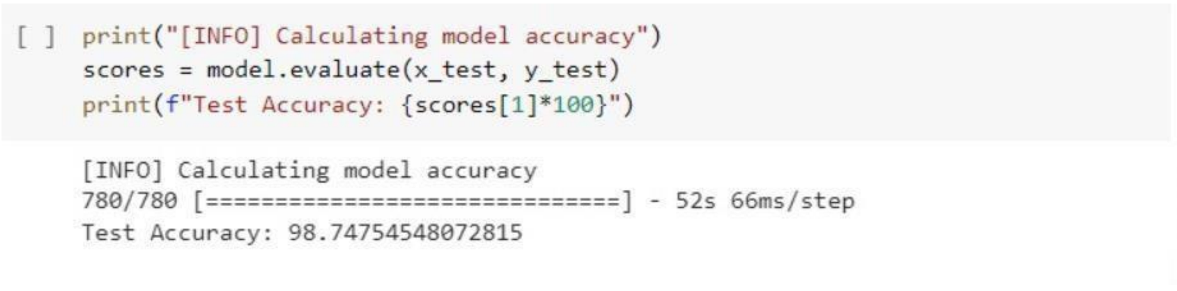


Fig.The training and validation accuracy is shown above

The accuracy that came out to be was pretty good which is 98.74%.



```
[ ]  print("[INFO] Calculating model accuracy")
     scores = model.evaluate(x_test, y_test)
     print(f"Test Accuracy: {scores[1]*100}")

     [INFO] Calculating model accuracy
     780/780 [==============================] - 52s 66ms/step
     Test Accuracy: 98.74754548072815
```

FUTURE WORK

The agricultural department seeks to automate the process of detecting yield crops (real time). This method can be automated by displaying the prediction result in a web or desktop application. To optimise the labour for Artificial Intelligence implementation.

To detect the sickness of that crop, this device just looks at the plant's leaf. Other sections of the crop, such as roots, stems, and branches, would be more convenient to identify than the existing method. Additionally, picture classification will be performed to determine whether or not the supplied leaf belongs to the selected category. If a model is given an input that is not a leaf image, it will display a disease name.

LIMITATIONS

- In some circumstances, the implementation still lacks accuracy in terms of results. More tweaking is required.
- For segmentation, prior information is required.
- In order to get more accuracy, a database extension is required.
- There have been very few diseases covered. As a result, work must be expanded to include more disorders.
- The following are some of the possible causes of misclassification: Disease symptoms differ from plant to plant, hence feature optimization and more training samples are required to cover more cases and more correctly forecast disease.

ADVANTAGES:

The following are the benefits of the suggested algorithm:

- Estimators are used to automatically initialise cluster centres, eliminating the requirement for human input during segmentation.

- The proposed technique improves detection accuracy.

- Existing approaches require user input to select the optimum segmentation of the input image, but the proposed method is completely automated.

- It also offers environmentally friendly treatment options for the condition.

References:

1. Eftekhar Hossain, Md. Farhad Hossain, and Mohammad Anisur Rahaman, "A Color and Texture Based Approach for the Detection and Classification of Plant Leaf Disease Using KNN Classifier," International Conference on Electrical, Computer, and Communication Engineering (ECCE), Cox's Bazaar, Bangladesh, 2019.

2. 2. Sammy V. Militante, Bobby D. Gerardo, and Nanette V. Dionisio, "Plant Leaf Detection and Disease Recognition Using Deep Learning," IEEE Eurasia Conference on IOT, Communication, and Engineering (ECICE), Yunlin, Taiwan, 2019, pages 579-582.

3. 3. Ch. Usha Kumari, S. Jeevan Prasad, and G. Mounika, "Leaf Disease Detection: Feature Extraction with K-means Clustering and Classification with ANN," Proceedings of the 3rd International Conference on Computing Methodologies and Communication (ICCMC), Erode, India, 2019, pp. 1095-1098.

4. Mercelin Francis, C. Deisy, "Disease Detection and Classification in Agricultural Plants Using Convolutional Neural Networks — A Visual Understanding", Proceeding of the 6 th International Conference on Signal processing and Integrated network, Noida, India, 2019, pp 1063-1068.

5. Jiayue Zhao, Jianhua Qu, "A Detection Method for Tomato Fruit Common Physiological Diseases Based on YOLOv2 "Proceeding of the 10th International Conference on Information Technology in Medicine and Education", Qingdao, China, China, 2019

6. Robert G. de Luna, Elmer P. Dadios, Argel A. Bandala, “Automated Image Capturing System for Deep Learning-based Tomato Plant Leaf Disease Detection and Recognition” Proceeding of the IEEE Region 10 Conference, Jeju, South Korea, 2018, pp 1414-1419.

7. Halil Durmus, Ece Olcay Gunes, “Disease detection on the leaves of the tomatoplants by using deep learning” Proceeding of the 6 th International Conference oftheAgroGeoinformatics, 2016, Fairfax, VA, USA.

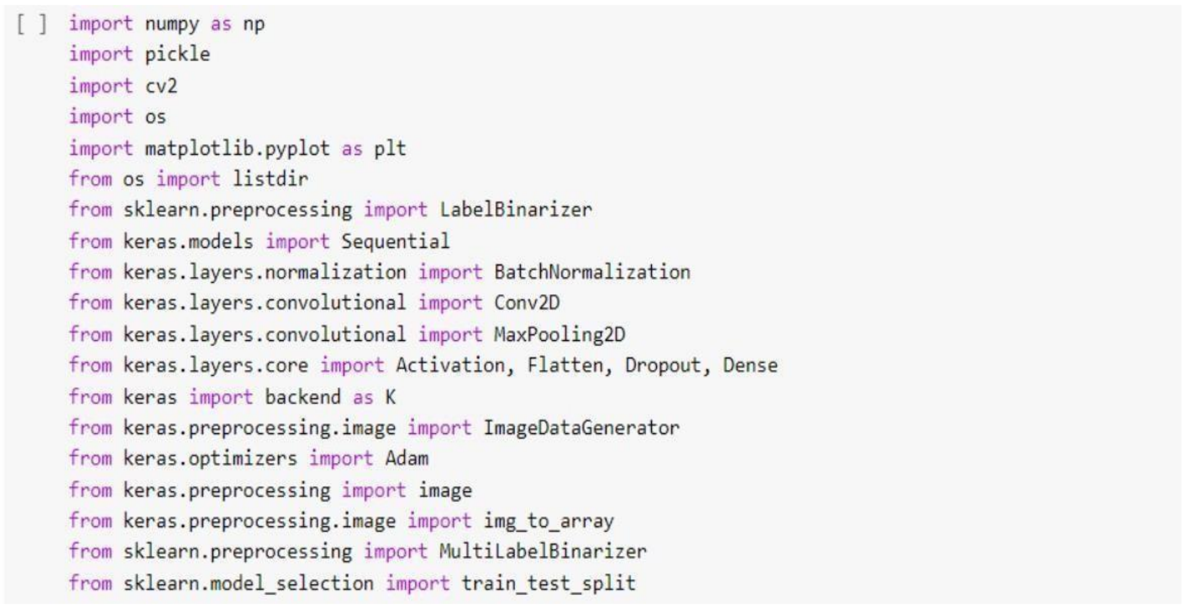
8. Husin, Zulkifli Bin, Ali Yeon Bin Md Shakaff, Abdul Hallis Bin Abdul Aziz, and Rohani Binti S. Mohamed Farook. ”Feasibility study on plant chili disease detection using image processing techniques.” In 2012 Third International Conference on Intelligent Systems Modelling and Simulation, pp. 291-296. IEEE, 2012.

9. Y. Dandawate and R. Kokare, “An automated approach for classification of plant diseases towards development of futuristic decision support system in indian perspective,” Proceedings of the International Conference Advances in Computing, Communications and Informatics (ICACCI), 2015, pp. 794-799.

10. Monika Jhuria, Ashwani Kumar, and Rushikesh Borse. ”Image processingfor smart farming: Detection of disease and fruit grading.” In 2013 IEEE Second International Conference on Image Information Processing (ICIIP-2013), pp. 521-526. IEEE, 2013

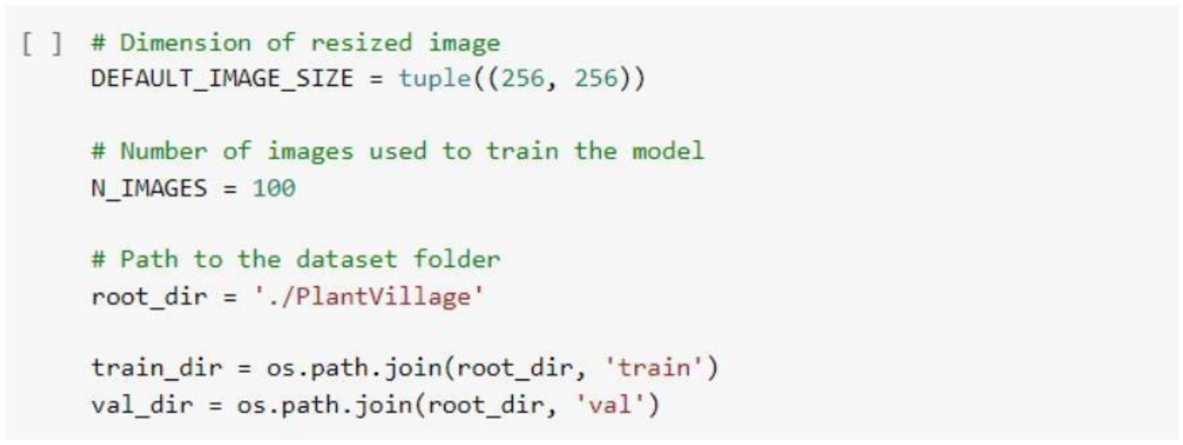
APPENDICES:

Importing required libraries:



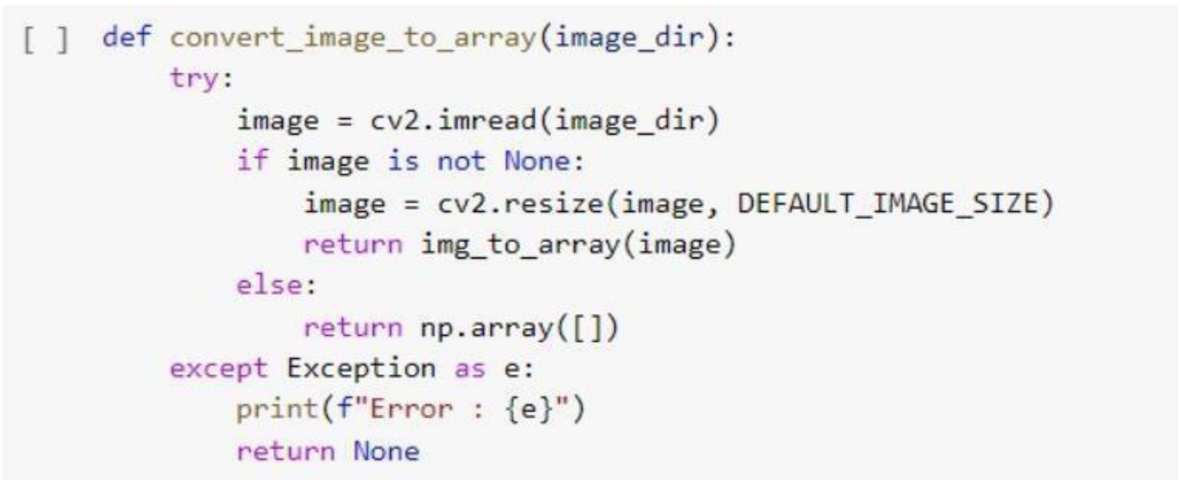
```
import numpy as np
import pickle
import cv2
import os
import matplotlib.pyplot as plt
from os import listdir
from sklearn.preprocessing import LabelBinarizer
from keras.models import Sequential
from keras.layers.normalization import BatchNormalization
from keras.layers.convolutional import Conv2D
from keras.layers.convolutional import MaxPooling2D
from keras.layers.core import Activation, Flatten, Dropout, Dense
from keras import backend as K
from keras.preprocessing.image import ImageDataGenerator
from keras.optimizers import Adam
from keras.preprocessing import image
from keras.preprocessing.image import img_to_array
from sklearn.preprocessing import MultiLabelBinarizer
from sklearn.model_selection import train_test_split
```

Loading the dataset:



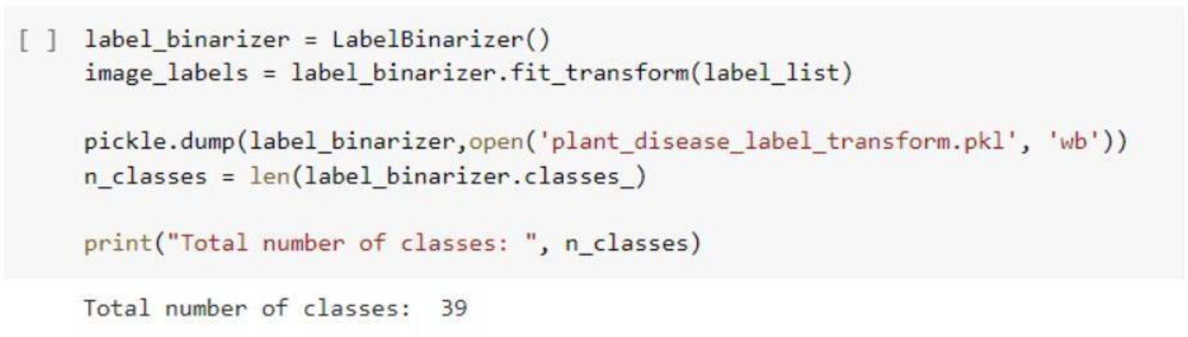
```
# Dimension of resized image
DEFAULT_IMAGE_SIZE = tuple((256, 256))

# Number of images used to train the model
N_IMAGES = 100

# Path to the dataset folder
root_dir = './PlantVillage'

train_dir = os.path.join(root_dir, 'train')
val_dir = os.path.join(root_dir, 'val')
```

Resizing of image:



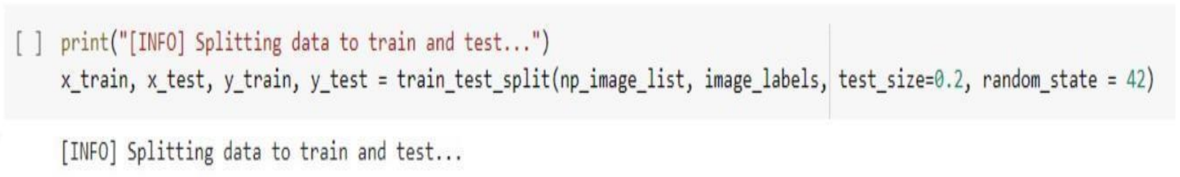
```
[ ]  def convert_image_to_array(image_dir):
         try:
             image = cv2.imread(image_dir)
             if image is not None:
                 image = cv2.resize(image, DEFAULT_IMAGE_SIZE)
                 return img_to_array(image)
             else:
                 return np.array([])
         except Exception as e:
             print(f"Error : {e}")
             return None
```

Examine Label or classes in training dataset



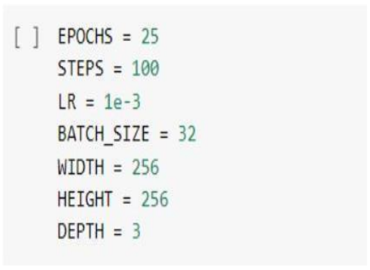
```
[ ]  label_binarizer = LabelBinarizer()
     image_labels = label_binarizer.fit_transform(label_list)

     pickle.dump(label_binarizer,open('plant_disease_label_transform.pkl', 'wb'))
     n_classes = len(label_binarizer.classes_)

     print("Total number of classes: ", n_classes)

     Total number of classes:  39
```

Splitting the data into training and testing database

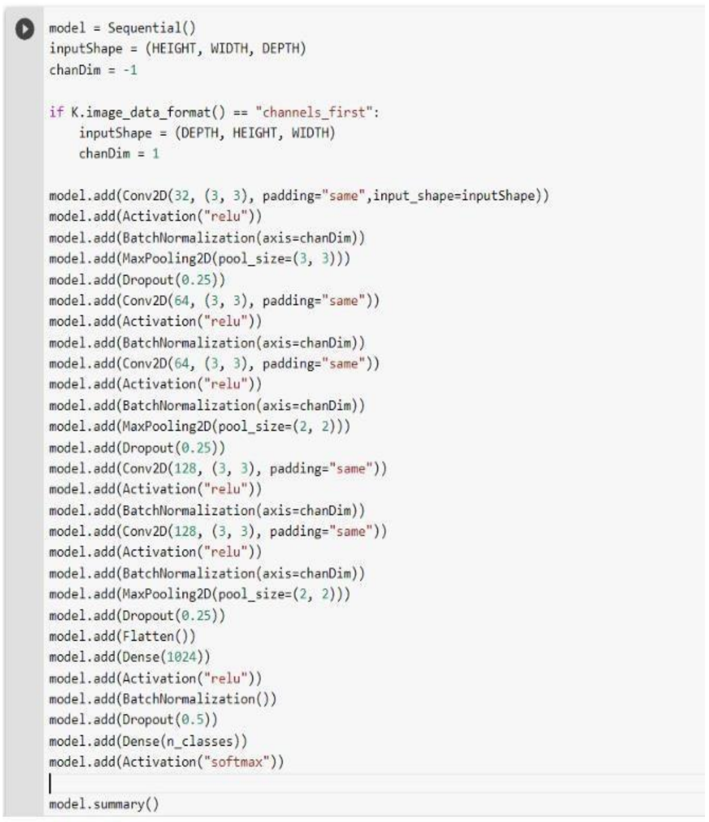


```
[ ]  print("[INFO] Splitting data to train and test...")
     x_train, x_test, y_train, y_test = train_test_split(np_image_list, image_labels, test_size=0.2, random_state = 42)

     [INFO] Splitting data to train and test...
```

Building the model



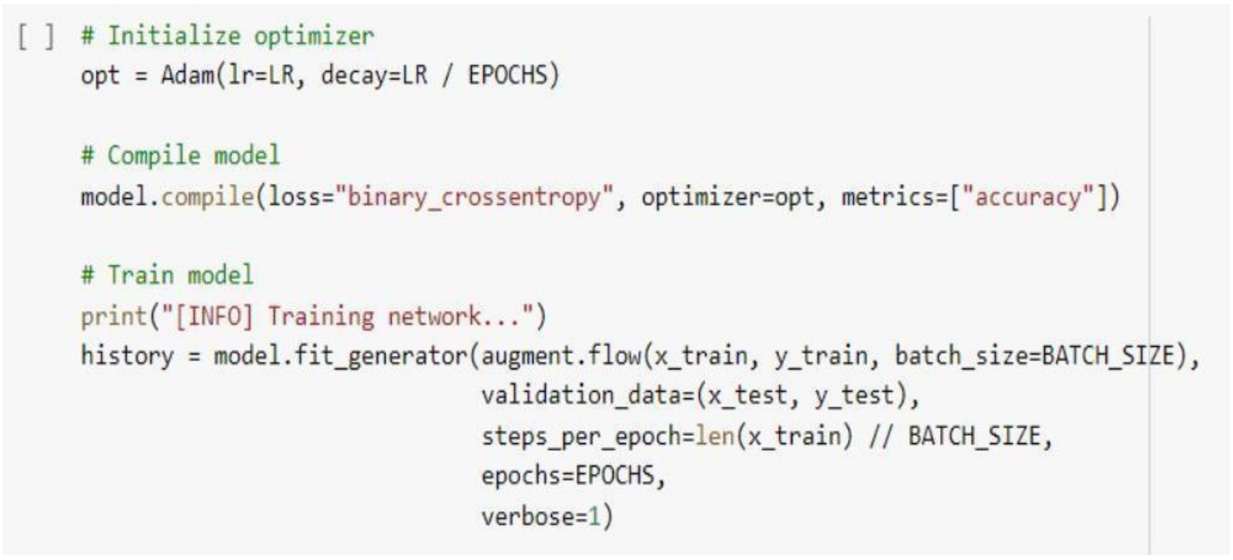
```
[ ] EPOCHS = 25
    STEPS = 100
    LR = 1e-3
    BATCH_SIZE = 32
    WIDTH = 256
    HEIGHT = 256
    DEPTH = 3
```



```
model = Sequential()
inputShape = (HEIGHT, WIDTH, DEPTH)
chanDim = -1

if K.image_data_format() == "channels_first":
    inputShape = (DEPTH, HEIGHT, WIDTH)
    chanDim = 1

model.add(Conv2D(32, (3, 3), padding="same",input_shape=inputShape))
model.add(Activation("relu"))
model.add(BatchNormalization(axis=chanDim))
model.add(MaxPooling2D(pool_size=(3, 3)))
model.add(Dropout(0.25))
model.add(Conv2D(64, (3, 3), padding="same"))
model.add(Activation("relu"))
model.add(BatchNormalization(axis=chanDim))
model.add(Conv2D(64, (3, 3), padding="same"))
model.add(Activation("relu"))
model.add(BatchNormalization(axis=chanDim))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Conv2D(128, (3, 3), padding="same"))
model.add(Activation("relu"))
model.add(BatchNormalization(axis=chanDim))
model.add(Conv2D(128, (3, 3), padding="same"))
model.add(Activation("relu"))
model.add(BatchNormalization(axis=chanDim))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(1024))
model.add(Activation("relu"))
model.add(BatchNormalization())
model.add(Dropout(0.5))
model.add(Dense(n_classes))
model.add(Activation("softmax"))

model.summary()
```

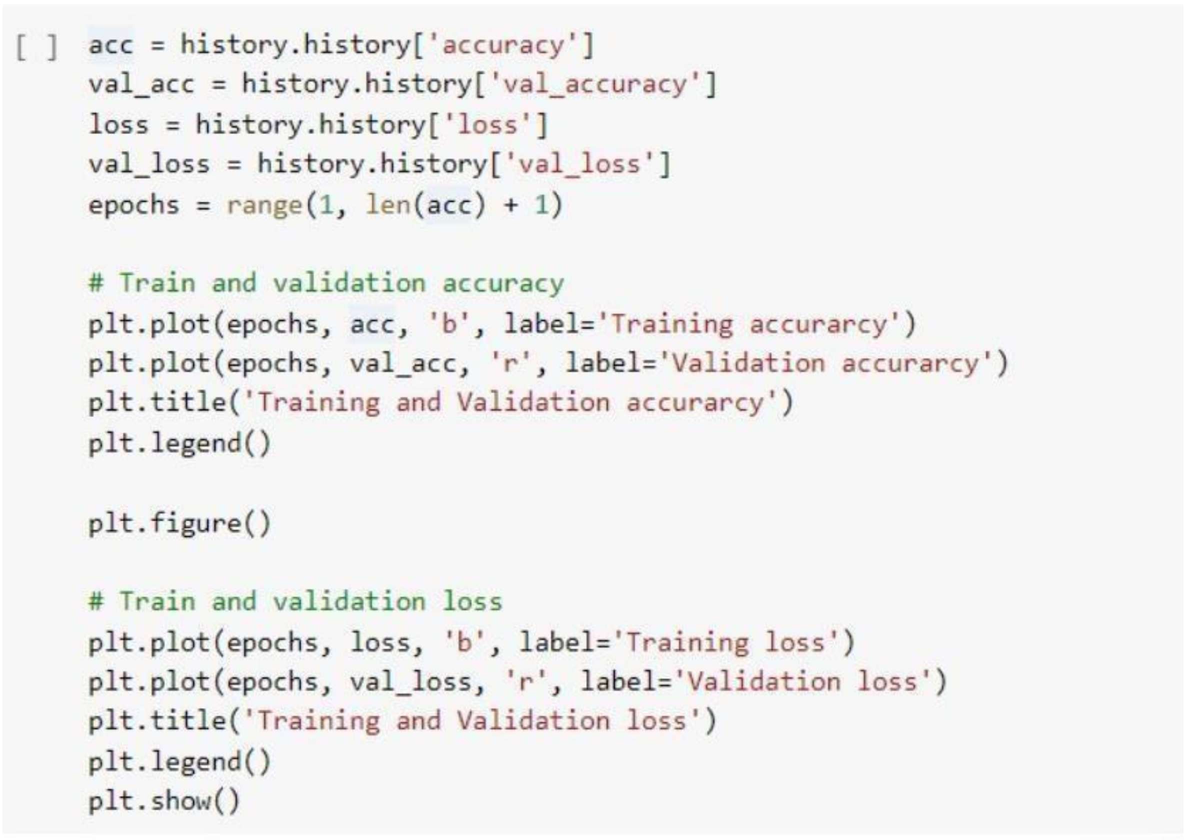
\

Train Model



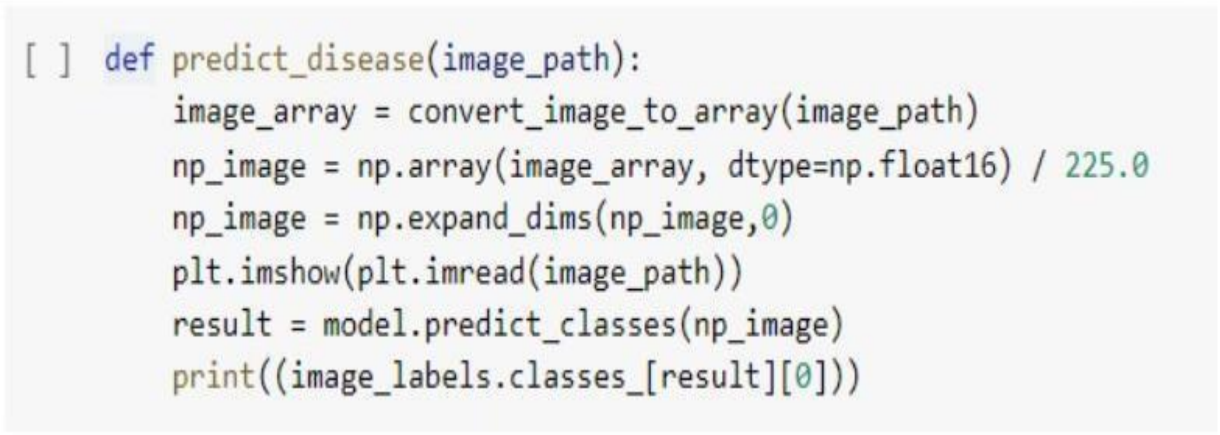
```
[ ] # Initialize optimizer
    opt = Adam(lr=LR, decay=LR / EPOCHS)

    # Compile model
    model.compile(loss="binary_crossentropy", optimizer=opt, metrics=["accuracy"])

    # Train model
    print("[INFO] Training network...")
    history = model.fit_generator(augment.flow(x_train, y_train, batch_size=BATCH_SIZE),
                                  validation_data=(x_test, y_test),
                                  steps_per_epoch=len(x_train) // BATCH_SIZE,
                                  epochs=EPOCHS,
                                  verbose=1)
```

Evaluate Model



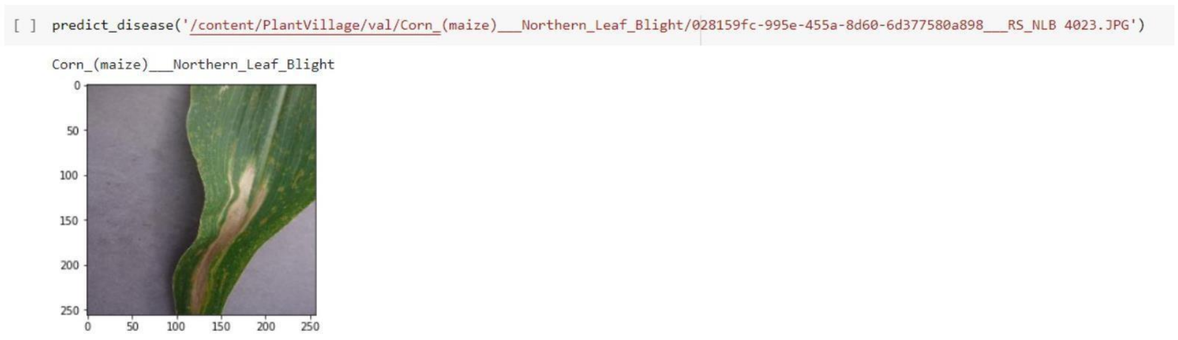
```
[ ]  acc = history.history['accuracy']
     val_acc = history.history['val_accuracy']
     loss = history.history['loss']
     val_loss = history.history['val_loss']
     epochs = range(1, len(acc) + 1)

     # Train and validation accuracy
     plt.plot(epochs, acc, 'b', label='Training accurarcy')
     plt.plot(epochs, val_acc, 'r', label='Validation accurarcy')
     plt.title('Training and Validation accurarcy')
     plt.legend()

     plt.figure()

     # Train and validation loss
     plt.plot(epochs, loss, 'b', label='Training loss')
     plt.plot(epochs, val_loss, 'r', label='Validation loss')
     plt.title('Training and Validation loss')
     plt.legend()
     plt.show()
```

Test Model



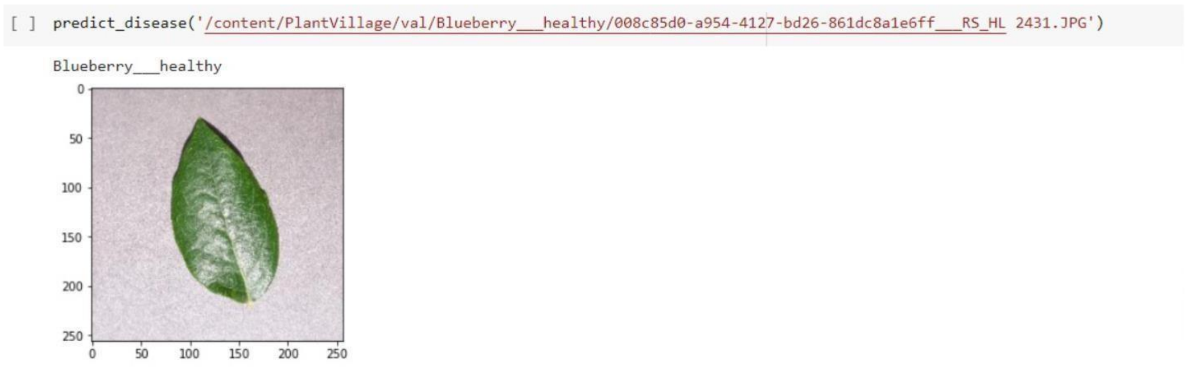
```
[ ]  def predict_disease(image_path):
         image_array = convert_image_to_array(image_path)
         np_image = np.array(image_array, dtype=np.float16) / 225.0
         np_image = np.expand_dims(np_image,0)
         plt.imshow(plt.imread(image_path))
         result = model.predict_classes(np_image)
         print((image_labels.classes_[result][0]))
```

Classification



```
[ ]  predict_disease('/content/PlantVillage/val/Corn_(maize)___Northern_Leaf_Blight/028159fc-995e-455a-8d60-6d377580a898___RS_NLB 4023.JPG')
```

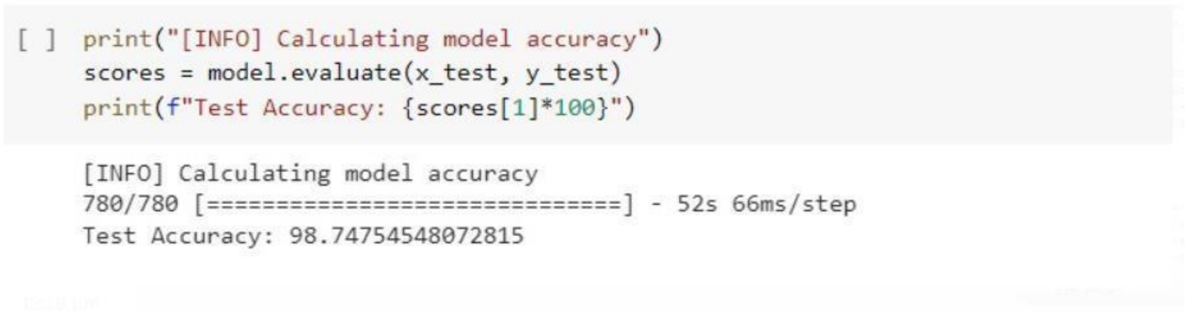
```
Corn_(maize)___Northern_Leaf_Blight
```



```
[ ]  predict_disease('/content/PlantVillage/val/Blueberry___healthy/008c85d0-a954-4127-bd26-861dc8a1e6ff___RS_HL 2431.JPG')
```

```
Blueberry___healthy
```

Accuracy



```
[ ]  print("[INFO] Calculating model accuracy")
     scores = model.evaluate(x_test, y_test)
     print(f"Test Accuracy: {scores[1]*100}")
```

```
[INFO] Calculating model accuracy
780/780 [==============================] - 52s 66ms/step
Test Accuracy: 98.74754548072815
```