PROJECT REPORT

ON

"Plant Disease Detection Using Deep Learning-Leaf Images"

Submitted in the partial fulfilment of the requirements for the award of Master's Degree of

MASTER IN COMPUTER APPLICATIONS

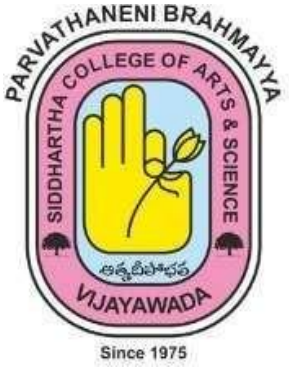
Submitted By

PATAN ZAREENA FATHIMA [23MCA48]

Under the Guidance Of

Ms. KOSANA BHUVANESWARI ,M.SC(CS),M.TECH(CSE)

ASSISTANT PROFESSOR



DEPARTMENT OF COMPUTER SCIENCE

P.B. SIDDHARTHA COLLEGE OF ARTS AND SCIENCE

Affiliated to Krishna University, Machilipatnam

VIJAYAWADA-520010

PARVATHANENI BRAHMAYYA SIDDHARTHA COLLEGE OF ARTS AND SCIENCE

SIDDHARTHA NAGAR, VIJAYAWADA - 520 010

DEPARTMENT OF COMPUTER SCIENCE

CERTIFICATE

This is to certify that the project work entitled **"PLANT DISEASE DETECTION USING DEEP LEARNING –LEAF IMAGES**" is the Bona fide work of **PATAN ZAREENA FATHIMA[23MCA48]**of Masters in Computer Applications submitted to Krishna University during the academic year 2023-2025.

PROJECT GUIDE HEAD OF DEPARTMENT

EXTERNAL EXAMINER

DATAVALLEY®

CERTIFICATE

OF PROJECT

This is to certify that Mr./Mrs **PATAN ZAREENA FATHIMA**

a student of **IV Semester MCA** at **P.B. Siddhartha College of Arts & Science,** Vijayawada, Andhra Pradesh, bearing Roll No. **23MCA48** has successfully completed project work titled **Plant Disease Detection Using deep learning** from **15-04-2025 to 16-07-2025,** under the expert guidance and supervision of **Ms. Nazeer Bee Shaik** AI&ML Mentor at **Datavalley India Pvt. Ltd.**

During this period, he/she has completed the assigned project well within the time frame. His/her performance has been commendable, reflecting a strong work ethic, dedication and professional conduct throughout the project.

We appreciate his/her efforts acontinued success in all future endeavors.

DATE : 17 - 07 - 2025

PLACE : Datavalley India Pvt Ltd, Vijayawada

Authorized Signature

DECLARATION

I hereby declare that this project work entitled is **"PLANT DISEASE DETECTION USING DEEP LEARNING –LEAF IMAGES"** submitted by me to Krishna University during the academic year 2023-2025 in partial fulfilment of the requirements of Master of Computer Application.

I also declare that this project is the result of my own efforts and it has not been submitted to any other institution or university for the award of any degree.

PATAN ZAREENA FATHIMA[23MCA48]

ACKNOWLEDGEMENT

I am extremely thankfull to our project guide **MS.K.Bhuvaneswari mam ,M.SC(CS),M.TECH(CSE)** Assistant Professor, Department of Computer Science, P. B. Siddhartha College of Arts and Science, Krishna University, Machilipatnam for her timely cooperation and valuable suggestions throughout the project. I am indebted to her for the opportunity given to work under her guidance. I am extremely thankful to Dr. T. S. Ravi Kiran, Head of the Computer Science Department and Dean Rajesh C.Jampala, P.B. Siddhartha College of Arts and Science, Vijayawada for providing good environment and better infrastructure facilities.

My sincere thanks to all the teaching and non-teaching staff of Department of Computer Science for their support throughout my project work. Finally, I am very much indebted to my family for their moral support and encouragement to achieve higher goals.

CONTEXT

ABSTRACT

Plant diseases significantly threaten global agricultural productivity, necessitating rapid and accurate detection methods for effective crop yield management. Traditional identification approaches are often labor-intensive and require specialized knowledge. In this study, we leverage advanced deep learning techniques, specifically Convolutional Neural Networks (CNNs) and Vision Transformers (ViTs), to enhance plant disease detection accuracy. Utilizing a meticulously collected multispectral dataset with six 50 mm filters, spanning both visible and near-infrared (NIR) wavelengths, we explore innovative methodologies for disease classification. achieving an overall accuracy of 90% with similar models. This comparative analysis underscores the critical impact of balanced datasets and optimal wavelength selection on the efficacy of deep learning models for robust disease identification. These findings not only promise to advance crop disease management practices in agricultural settings but also contribute to enhancing global food security. Our study emphasizes the transformative potential of machine learning in plant disease diagnostics and advocates for ongoing research in this vital area.

Keywords: Disease, Detection, plant,Accuract,Rsnet50.

INTRODUCTION

1. INTRODUCTION

In India about 70% of the populace relies on agriculture. Identification of the plant diseases is important in order to prevent the losses within the yield. It's terribly troublesome to observe the plant diseases manually. It needs tremendous quantity of labor, expertize within the plant diseases, and conjointly need the excessive time interval. Hence, image processing and machine learning models can be employed for the detection of plant diseases. In this project, we have described the technique for the detection of plant diseases with the help of their leaves pictures. Image processing is a branch of signal processing which can extract the image properties or useful information from the image.

Machine learning is a sub part of artificial intelligence which works automatically or give instructions to do a particular task. The main aim of machine learning is to understand the training data and fit that training data into models that should be useful to the people. So it can assist in good decisions making and predicting the correct output using the large amount of training data. The color of leaves, amount of damage to leaves, area of the leaf, texture parameters are used for classification.

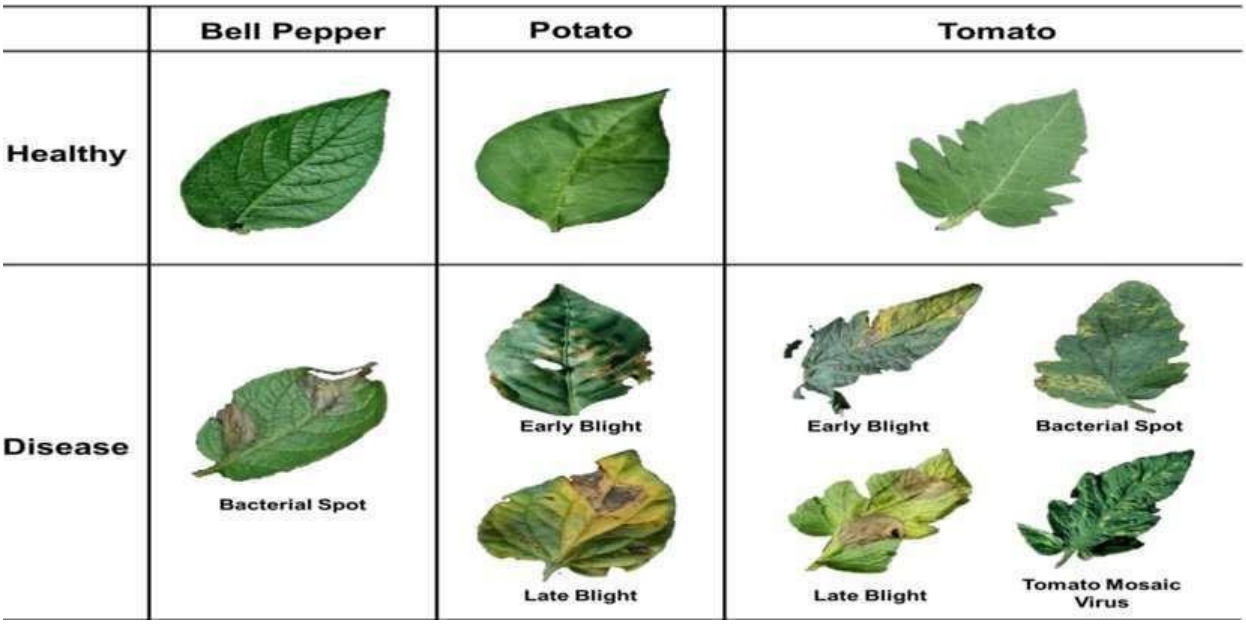


Fig: 1.1 Disease detection plant in model

In this project we have analyzed different image parameters or features to identifying different plant leaves diseases to achieve the best accuracy. Previously plant disease detection is done by visual inspection of the leaves or some chemical processes by experts. For doing so, a large team of experts as well as continuous observation of plant is needed, which costs high when we do with large farms. In such conditions, the recommended system proves to be helpful in monitoring large fields of crops.

Automatic detection of the diseases by simply seeing the symptoms on the plant leaves makes it easier as well as cheaper. The proposed solution for plant disease detection is computationally less expensive and requires less time for prediction than other deep learning based approaches since it uses statistical machine learning and image processing algorithm.

Plant Disease Detection

Plant disease poses a significant threat to the quality and quantity of global agricultural production. The prevalence of disastrous plant diseases exacerbates the existing food shortage, with at least 0.8 billion people currently undernourished. Furthermore, as the number of consumers continues to rise, food security becomes an increasingly pressing concern. To mitigate damage, it is essential to identify plant diseases as early as possible. Viral plant diseases, in particular, are challenging to manage, as they often have no cures and can spread rapidly. Infected plants must be removed immediately to prevent further infections, making accurate diagnosis a critical task.

Fig: 1.2 Disease detection plant in model

Agriculture is essential to sustaining the growing global population. Farmers have access to a wide variety of eligible crops and can select appropriate insecticides to protect their plants. However, crop damage can lead to significant production losses, which in turn impacts the economy. The leaves of plants are particularly vulnerable, as disease symptoms often first appear there. Therefore, it is crucial to inspect crops for illnesses from the early stages of their life cycle until they are ready for harvest.

Traditionally, specialists relied on manual observation of agricultural fields, a time-consuming method that involved naked-eye surveillance to monitor plants for diseases. In recent years, various automatic and semi-automatic methods have been developed to enhance this process.

Advancements in Technology

The rapid advancements in computer science and computer vision have opened new avenues for diagnosing infected plants. These technologies simplify disease classification and refine treatment strategies, making it easier for farmers to manage plant health effectively. By harnessing the synergy between technological innovation and agricultural needs, researchers aim to create transformative solutions that can revolutionize crop protection on a global scale.

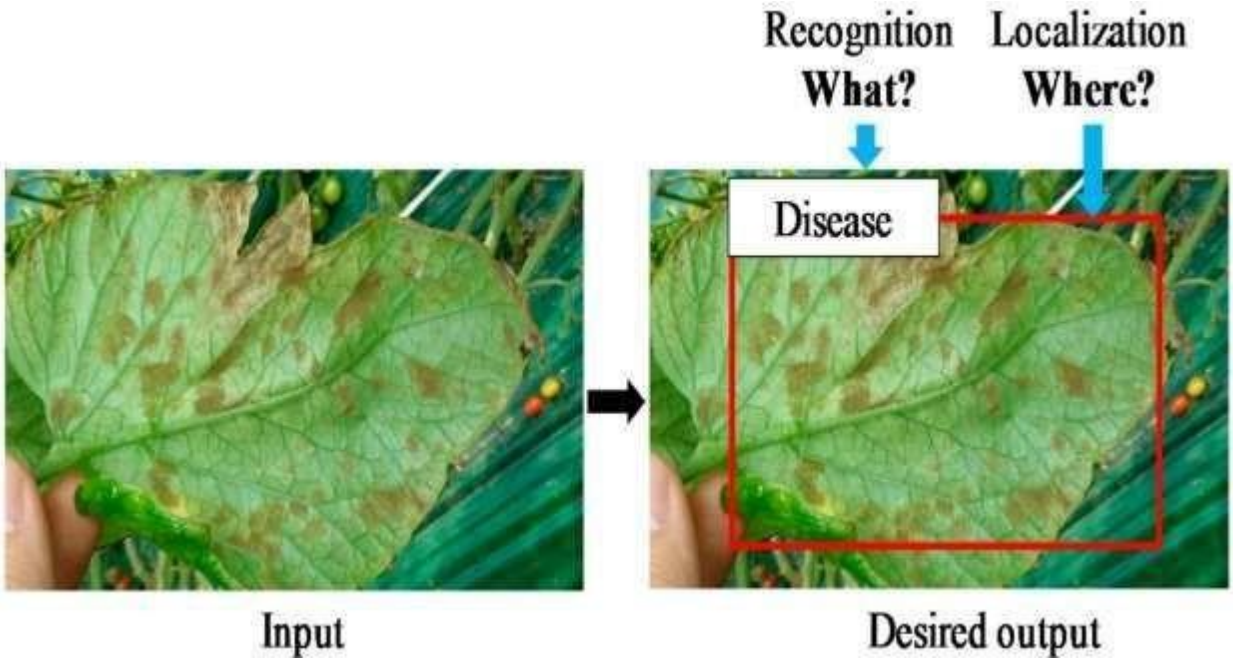


Fig:1.3 Plants didesases and pests recognization

Future Prospects and Challenges

Despite its immense potential, deep learning-based plant disease detection faces certain challenges. One of the primary concerns is the need for extensive and diverse datasets to train accurate models. Variations in lighting, background noise, and leaf orientations can affect the model's performance. Additionally, high computational power is required to process and analyze large image datasets. However, with advancements in cloud computing and edge AI, real-time and efficient plant disease detection systems are becoming increasingly feasible. Future research in this field aims to improve model robustness, enhance dataset diversity, and develop user-friendly interfaces for seamless integration into agricultural practices.

Deep learning, a subset of artificial intelligence, utilizes convolutional neural networks (CNNs) to analyze and classify leaf images with high precision. These models can learn complex patterns and features from large datasets, enabling the identification of various plant diseases at an early stage. By leveraging techniques such as transfer learning, data augmentation, and model optimization, deep learning-based systems can improve diagnostic accuracy and efficiency.

Agriculture is the backbone of the global economy, providing food and raw materials for various industries. However, plant diseases pose a major threat to agricultural productivity, leading to significant losses in crop yield and quality. Timely and accurate detection of plant diseases is essential to prevent their spread and ensure sustainable farming practices.

Fig: 1.4 Prediction using CNN

SYSTEM REQUIREMENTS

2.1 HARDWARE REQUIREMENTS

- **Processor**- Pentium–III
- **Speed**- 2.4GHz
- **RAM**- 512 MB(min)
- **Hard Disk**- 20 GB
- **Floppy Drive**- 1.44MB
- **Key Board**- Standard Keyboard
- **Monitor**- 15 VGA Color

2.2 SOFTWARE REQUIREMENTS

Functional requirements for a secure cloud storage service are straightforward:

- The service should be able to store the user's data.
- The data should be accessible through any devices connected to the Internet.
- The service should be capable to synchronize the user's data between multiple devices (notebooks, smart phones, etc.).
- The service should preserve all historical changes(versioning).
- Data should be shareable with other users.
- The service should support SSO.
- The service should be interoperable with other cloud storage services, enabling data migration from one CSP to another.
- Operating System: Windows
- Coding Language: Python 3.7
- Script
- Database

SYSTEM ANALYSIS & STUDY

3. System Analysis

System analysis is a critical process in the field of information technology and business, aimed at understanding and specifying in detail what a system should do. This video is perfect for IT professionals, business analysts, and anyone interested in understanding how effective system analysis can enhance project success and operational efficiency. To begin with, we'll delve into the fundamentals of system analysis. This involves breaking down complex processes and systems into manageable components. We'll discuss key concepts such as requirements gathering, where we identify and document what stakeholders need from the system. Understanding these requirements is crucial as it forms the foundation upon which the entire system is built. Next, we'll cover various methodologies and tools used in system analysis. From traditional approaches like Waterfall to modern Agile methodologies, we'll explain how each framework guides the analysis process differently. Tools such as data flow diagrams (DFDs), use case diagrams, and entity-relationship diagrams (ERDs) will also be explored, demonstrating how they help visualize and map out system functionalities and data interactions effectively. Furthermore, we'll examine the role of system analysts. These professionals are responsible for bridging the gap between business needs and technological solutions. We'll look at the skills and qualities required to excel in this role, such as strong analytical abilities, communication skills, and a thorough understanding of both business processes and IT capabilities. Finally, we'll highlight some real-world case studies where effective system analysis has led to successful project outcomes. These examples will illustrate the importance of meticulous planning, stakeholder involvement, and continuous feedback throughout the system development lifecycle. By understanding these success stories, you'll gain insights into best practices and common pitfalls to avoid in your own projects. If you find this video insightful, don't forget to hit the like button and subscribe to our channel for more content on system analysis and other related topics. Your support helps us bring you more valuable information and tutorials. Now, let's dive into the details of system analysis and uncover how it can transform the way we approach system development and business solutions.

3.1 TECHNOLOGIES LEARNT

3.1.1 PYTHON

What is Python:

Below are some facts about Python. Python is currently the most widely used multi- purpose, high-level programming language. Python allows programming in Object- Oriented and Procedural paradigms.

Python programs generally are smaller than other programming languages like Java. Programmers have to type relatively less and indentation requirement of the language, makes them readable all the time.

Python language is being used by almost all tech-giant companies like – Google, Amazon, Facebook, Instagram, Dropbox, Uber… etc. The biggest strength of Python is huge collection of standard library which can be used for the following
Machine Learning

- GUI Applications (like Kivy, Tkinter, PyQt etc.)
- Web frameworks like Django (used by YouTube, Instagram, Dropbox)
- Image processing (like OpenCV, Pillow)
- Web scraping (like Scrapy, Beautiful Soup, Selenium)
- Test frameworks
- Multimedia

Advantages of Python :

Let's see how Python dominates over other languages.

- **Extensive Libraries**

  Python downloads with an extensive library and it contain code for various purposes like regular expressions, documentation-generation, unit-testing, web browsers, threading, databases, CGI, email, image manipulation, and more. So, we don't have to write the complete code for that manually.

- **Extensible**

  As we have seen earlier, Python can be extended to other languages. You can write some of your code in languages like C++ or C. This comes in handy, especially in projects

- **Embeddable**

  Complimentary to extensibility, Python is embeddable as well. You can put your Python code in your source code of a different language, like C++. This lets us add scripting capabilities to our code in the other language.

- **Improved Productivity**

The language's simplicity and extensive libraries render programmers more productive than languages like Java and C++ do. Also, the fact that you need to write less and get more things done.

- **IOT Opportunities**

Since Python forms the basis of new platforms like Raspberry Pi, it finds the future bright for the Internet Of Things. This is a way to connect the language with the real world.

- **Simple and Easy**

When working with Java, you may have to create a class to print 'Hello World'. But in Python, just a print statement will do. It is also quite easy to learn, understand, and code. This is why when people pick up Python, they have a hard time adjusting to other more verbose languages like Java.

- **Readable**

Because it is not such a verbose language, reading Python is much like reading English. This is the reason why it is so easy to learn, understand, and code. It also does not need curly braces to define blocks, and indentation is mandatory. This further aids the readability of the code.

- **Object-Oriented**

This language supports both the procedural and object-oriented programming paradigms. While functions help us with code reusability, classes and objects let us model the real world. A class allows the encapsulation of data and functions into one.

- **Free and Open-Source**

Like we said earlier, Python is freely available. But not only can you download Python for free, but you can also download its source code, make changes to it, and even distribute it. It downloads with an extensive collection of libraries to help you with your tasks.

- **Portable**

When you code your project in a language like C++, you may need to make some changes to it if you want to run it on another platform. But it isn't the same with Python. Here, you need to code only once, and you can run it anywhere. This is called Write Once Run Anywhere (WORA). However, you need to be careful enough not to include any system- dependent features.

- **Interpreted**

Lastly, we will say that it is an interpreted language. Since statements are executed one by one, debugging is easier than in compiled languages.

Advantages of Python Over Other Languages:

- ### Less Coding

  Almost all of the tasks done in Python requires less coding when the same task is done in other languages. Python also has an awesome standard library support, so you don't have to search for any third-party libraries to get your job done. This is the reason that many people suggest learning Python to beginners.

- ### Affordable

  Python is free therefore individuals, small companies or big organizations can leverage the free available resources to build applications. Python is popular and widely used so it gives you better community support.

  The 2019 Github annual survey showed us that Python has overtaken Java in the most popular programming language category

- ### Python is for Everyone

  Python code can run on any machine whether it is Linux, Mac or Windows. Programmers need to learn different languages for different jobs but with Python, you can professionally build web apps, perform data analysis and machine learning, automate things, do web scraping and also build games and powerful visualizations. It is an all-rounder programming language.

Disadvantages of Python

So far, we've seen why Python is a great choice for your project. But if you choose it, you should be aware of its consequences as well. Let's now see the downsides of choosing Python over another language.

- ### Speed Limitations

  We have seen that Python code is executed line by line. But since Python is interpreted, it often results in slow execution. This, however, isn't a problem unless speed is a focal point for the project. In other words, unless high speed is a requirement, the benefits offered by Python are enough to distract us from its speed limitations.

- ### Weak in Mobile Computing and Browsers

  While it serves as an excellent server-side language, Python is much rarely seen on the

client-side. Besides that, it is rarely ever used to implement smartphone-based applications. One such application is called Carbonnell. The reason it is not so famous despite the existence of Brython is that it isn't that secure.

- **Design Restrictions**

  As you know, Python is dynamically-typed. This means that you don't need to declare the type of variable while writing the code. It uses duck-typing. But wait, what's that? Well, it just means that if it looks like a duck, it must be a duck. While this is easy on the programmers during coding, it can raise run-time errors.

- **Underdeveloped Database Access Layers**

  Compared to more widely used technologies like JDBC (Java Data Base Connectivity) and ODBC (Open Data Base Connectivity), Python's database access layers are a bit underdeveloped. Consequently, it is less often applied in huge enterprises.

- **Simple**

  No, we're not kidding. Python's simplicity can indeed be a problem. Take my example. I don't do Java, I'm more of a Python person. To me, its syntax is so simple that the verbosity of Java code seems unnecessary.

This was all about the Advantages and Disadvantages of Python Programming Language.

History of Python:

What do the alphabet and the programming language Python have in common? Right, both start with ABC. If we are talking about ABC in the Python context, it's clear that the programming language ABC is meant. ABC is a general-purpose programming language and programming environment, which had been developed in the Netherlands, Amsterdam, at the CWI (Centrum Wiskunde &Informatica). The greatest achievement of ABC was to influence the design of Python. Python was conceptualized in the late 1980s. Guido van Rossum worked that time in a project at the CWI, called Amoeba, a distributed operating system. In an interview with Bill Venners[1], Guido van Rossum said: "In the early 1980s, I worked as an implementer on a team building a language called ABC at Centrum voor Wiskunde en Informatica (CWI). I don't know how well people know ABC's influence on Python. I try to mention ABC's influence because I'm indebted to everything I learned during that project and to the people who worked on it."Later on in the same Interview, Guido van Rossum continued: "I remembered all my experience and some of my frustration with ABC. I decided to try to design a simple scripting language that possessed some of ABC's

better properties, but without its problems. So I started typing. I created a simple virtual machine, a simple parser, and a simple runtime. I made my own version of the various ABC parts that I liked. I created a basic syntax, used indentation for statement grouping instead of curly braces or begin-end blocks, and developed a small number of powerful data types: a hash table (or dictionary, as we call it), a list, strings, and numbers."

3.1.2 MACHINE LEARNING

What is Machine Learning:

Before we take a look at the details of various machine learning methods, let's start by looking at what machine learning is, and what it isn't. Machine learning is often categorized as a subfield of artificial intelligence, but I find that categorization can often be misleading at first brush. The study of machine learning certainly arose from research in this context, but in the data science application of machine learning methods, it's more helpful to think of machine learning as a means of building models of data.

Fundamentally, machine learning involves building mathematical models to help understand data. "Learning" enters the fray when we give these models tunable parameters that can be adapted to observed data; in this way the program can be considered to be "learning" from the data. Once these models have been fit to previously seen data, they can be used to predict and understand aspects of newly observed data. I'll leave to the reader the more philosophical digression regarding the extent to which this type of mathematical, model-based "learning" is similar to the "learning" exhibited by the human brain. Understanding the problem setting in machine learning is essential to using these tools effectively, and so we will start with some broad categorizations of the types of approaches we'll discuss here.

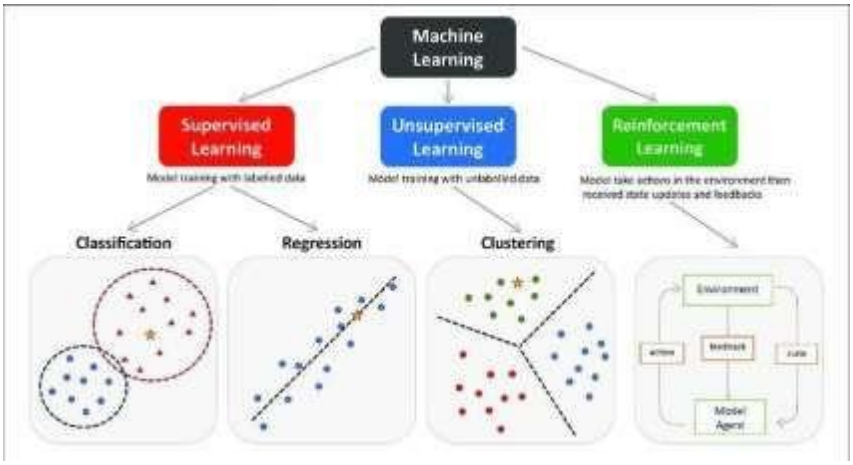
Categories Of Machine Learning:

At the most fundamental level, machine learning can be categorized into two main types: supervised learning and unsupervised learning.

Supervised learning involves somehow modeling the relationship between measured features of data and some label associated with the data; once this model is determined, it can be used to apply labels to new, unknown data. This is further subdivided into classification tasks and regression tasks: in classification, the labels are discrete categories, while in regression, the labels are continuous quantities. We will see examples of both types of supervised learning in the following section.

Unsupervised learning involves modeling the features of a dataset without reference to any label, and

is often described as "letting the dataset speak for itself." These models include tasks such as clustering and dimensionality reduction. Clustering algorithms identify distinct groups of data, while dimensionality reduction algorithms search for more succinct representations of the data. We will see examples of both types of unsupervised learning in the following section.



Need for Machine Learning

Human beings, at this moment, are the most intelligent and advanced species on earth because they can think, evaluate and solve complex problems. On the other side, AI is still in its initial stage and haven't surpassed human intelligence in many aspects. Then the question is that what is the need to make machine learn? The most suitable reason for doing this is, "to make decisions, based on data, with efficiency and scale".

Lately, organizations are investing heavily in newer technologies like Artificial Intelligence, Machine Learning and Deep Learning to get the key information from data to perform several real-world tasks and solve problems. We can call it data-driven decisions taken by machines, particularly to automate the process. These data-driven decisions can be used, instead of using programing logic, in the problems that cannot be programmed inherently. The fact is that we can't do without human intelligence, but other aspect is that we all need to solve real- world problems with efficiency at a huge scale. That is why the need for machine learning arises.

Challenges in Machines Learning:

While Machine Learning is rapidly evolving, making significant strides with cybersecurity and

autonomous cars, this segment of AI as whole still has a long way to go. The reason behind is that ML has not been able to overcome number of challenges. The challenges that ML is facing currently are –

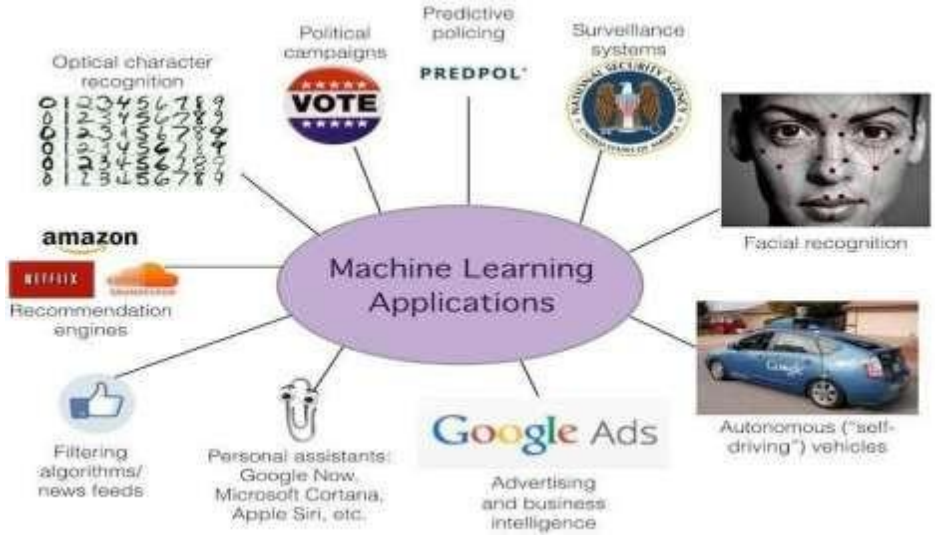
- **Quality of data** − Having good-quality data for ML algorithms is one of the biggest challenges. Use of low-quality data leads to the problems related to data preprocessing and feature extraction.
- **Time-Consuming task** − Another challenge faced by ML models is the consumption of time especially for data acquisition, feature extraction and retrieval.
- **Lack of specialist persons** − As ML technology is still in its infancy stage, availability of expert resources is a tough job.
- **No clear objective for formulating business problems** − Having no clear objective and well-defined goal for business problems is another key challenge for ML because this technology is not that mature yet.
- **Issue of overfitting & underfitting** − If the model is overfitting or underfitting, it cannot be represented well for the problem.
- **Curse of dimensionality** − Another challenge ML model faces is too many features of data points. This can be a real hindrance.
- **Difficulty in deployment** − Complexity of the ML model makes it quite difficult to be deployed in real life.

Applications of Machines Learning:

Machine Learning is the most rapidly growing technology and according to researchers we are in the golden year of AI and ML. It is used to solve many real-world complex problems which cannot be solved with traditional approach. Following are some real-world applications of ML −

- Emotion analysis
- Sentiment analysis
- Error detection and prevention
- Weather forecasting and prediction
- Stock market analysis and forecasting
- Speech synthesis
- Speech recognition
- Customer segmentation

- Object recognition
- Fraud detection
- Fraud prevention



How to Start Learning Machine Learning?

Arthur Samuel coined the term "Machine Learning" in 1959 and defined it as a "Field of study that gives computers the capability to learn without being explicitly programmed". And that was the beginning of Machine Learning! In modern times, Machine Learning is one of the most popular (if not the most!) career choices. According to Indeed, Machine Learning Engineer Is The Best Job of 2019 with a *344%* growth and an average base salary of $146,085 per year.

But there is still a lot of doubt about what exactly is Machine Learning and how to start learning it? So this article deals with the Basics of Machine Learning and also the path you can follow to eventually become a full-fledged Machine Learning Engineer. Now let's get started!!!

How to start learning ML?

This is a rough roadmap you can follow on your way to becoming an insanely talented Machine Learning Engineer. Of course, you can always modify the steps according to your needs to reach your desired end-goal!

Step 1 – Understand the Prerequisites

In case you are a genius, you could start ML directly but normally, there are some prerequisites that you need to know which include Linear Algebra, Multivariate Calculus, Statistics, and Python. And if you

don't know these, never fear! You don't need a Ph.D. degree in these topics to get started but you do need a basic understanding.

- **Learn Linear Algebra and Multivariate Calculus**

  Both Linear Algebra and Multivariate Calculus are important in Machine Learning. However, the extent to which you need them depends on your role as a data scientist. If you are more focused on application heavy machine learning, then you will not be that heavily focused on maths as there are many common libraries available. But if you want to focus on R&D in Machine Learning, then mastery of Linear Algebra and Multivariate Calculus is very important as you will have to implement many ML algorithms from scratch.

- **Learn Statistics**

  Data plays a huge role in Machine Learning. In fact, around 80% of your time as an ML expert will be spent collecting and cleaning data. And statistics is a field that handles the collection, analysis, and presentation of data. So it is no surprise that you need to learn it!!! Some of the key concepts in statistics that are important are Statistical Significance, Probability Distributions, Hypothesis Testing, Regression, etc. Also, Bayesian Thinking is also a very important part of ML which deals with various concepts like Conditional Probability, Priors, and Posteriors, Maximum Likelihood, etc.

- **Learn Python**

  Some people prefer to skip Linear Algebra, Multivariate Calculus and Statistics and learn them as they go along with trial and error. But the one thing that you absolutely cannot skip is Python! While there are other languages you can use for Machine Learning like R, Scala, etc. Python is currently the most popular language for ML. In fact, there are many Python libraries that are specifically useful for Artificial Intelligence and Machine Learning such as Keras, TensorFlow, Scikit-learn, etc.

- So if you want to learn ML, it's best if you learn Python! You can do that using various online resources and courses such as Fork Python available Free on GeeksforGeeks.

Step 2 – Learn Various ML Concepts

Now that you are done with the prerequisites, you can move on to actually learning ML (Which is the fun part!!!) It's best to start with the basics and then move on to the more complicated stuff. Some of the basic concepts in ML are:

Terminologies of Machine Learning

- **Model** – A model is a specific representation learned from data by applying some machine learning algorithm. A model is also called a hypothesis.
- **Feature** – A feature is an individual measurable property of the data. A set of numeric features can be conveniently described by a feature vector. Feature vectors are fed as input to the model. For example, in order to predict a fruit, there may be features like color, smell, taste, etc.
- **Target (Label)** – A target variable or label is the value to be predicted by our model. For the fruit example discussed in the feature section, the label with each set of input would be the name of the fruit like apple, orange, banana, etc.
- **Training** – The idea is to give a set of inputs(features) and it's expected outputs(labels), so after training, we will have a model (hypothesis) that will then map new data to one of the categories trained on.
- **Prediction** – Once our model is ready, it can be fed a set of inputs to which it will provide a predicted output(label).

Types of Machine Learning

- **Supervised Learning** – This involves learning from a training dataset with labeled data using classification and regression models. This learning process continues until the required level of performance is achieved.
- **Unsupervised Learning** – This involves using unlabelled data and then finding the underlying structure in the data in order to learn more and more about the data itself using factor and cluster analysis models.
- **Semi-supervised Learning** – This involves using unlabelled data like Unsupervised Learning with a small amount of labeled data. Using labeled data vastly increases the learning accuracy and is also more cost-effective than Supervised Learning.
- **Reinforcement Learning** – This involves learning optimal actions through trial and error. So the next action is decided by learning behaviors that are based on the current state and that will maximize the reward in the future.

Advantages of Machine learning :

- ### Easily identifies trends and patterns

Machine Learning can review large volumes of data and discover specific trends and patterns that would not be apparent to humans. For instance, for an e-commerce website like Amazon, it

serves to understand the browsing behaviors and purchase histories of its users to help cater to the right products, deals, and reminders relevant to them. It uses the results to reveal relevant advertisements to them.

- **No human intervention needed (automation)**

  With ML, you don't need to babysit your project every step of the way. Since it means giving machines the ability to learn, it lets them make predictions and also improve the algorithms on their own. A common example of this is anti-virus softwares; they learn to filter new threats as they are recognized. ML is also good at recognizing spam.

- **Continuous Improvement**

  As ML algorithms gain experience, they keep improving in accuracy and efficiency. This lets them make better decisions. Say you need to make a weather forecast model. As the amount of data you have keeps growing, your algorithms learn to make more accurate predictions faster.

- **Handling multi-dimensional and multi-variety data**

  Machine Learning algorithms are good at handling data that are multi-dimensional and multi-variety, and they can do this in dynamic or uncertain environments.

- **Wide Applications**

  You could be an e-tailer or a healthcare provider and make ML work for you. Where it does apply, it holds the capability to help deliver a much more personal experience to customers while also targeting the right customers.

Disadvantages of Machine Learning:

- **Data Acquisition**

  Machine Learning requires massive data sets to train on, and these should be inclusive/unbiased, and of good quality. There can also be times where they must wait for new data to be generated.

- **Time and Resources**

  ML needs enough time to let the algorithms learn and develop enough to fulfill their purpose with a considerable amount of accuracy and relevancy. It also needs massive resources to function. This can mean additional requirements of computer power for you.

- **Interpretation of Results**

Another major challenge is the ability to accurately interpret results generated by the algorithms. You must also carefully choose the algorithms for your purpose.

Python Development Steps:

Guido Van Rossum published the first version of Python code (version 0.9.0) at alt. sources in February 1991. This release included already exception handling, functions, and the core data types of list, dict, str and others. It was also object oriented and had a module system.

Python version 1.0 was released in January 1994. The major new features included in this release were the functional programming tools lambda, map, filter and reduce, which Guido Van Rossum never liked. Six and a half years later in October 2000, Python 2.0 was introduced. This release included list comprehensions, a full garbage collector and it was supporting Unicode . Python flourished for another 8 years in the versions 2.x before the next major release as Python 3.0 (also known as "Python 3000" and "Py3K") was released. Python 3 is not backwards compatible with Python 2.x. The emphasis in Python 3 had been on the removal of duplicate programming constructs and modules, thus fulfilling or coming close to fulfilling the 13th law of the Zen of Python: "There should be one -- and preferably only on
-- obvious way to do it."Some changes in Python 7.3:

- Print is now a function
- Views and iterators instead of lists
- The rules for ordering comparisons have been simplified.
- E.g. a heterogeneous list cannot be sorted, because all the elements of a list must be comparable to each other.
- There is only one integer type left, i.e. int. long is int as well.
- The division of two integers returns a float instead of an integer. "//" can be used to have the "old" behaviour.
- Text Vs. Data Instead of Unicode Vs. 8-bit

Purpose:

We demonstrated that our approach enables successful segmentation of intra-retinal layers—even with low-quality images containing speckle noise, low contrast, and different intensity ranges throughout— with the assistance of the ANIS feature.

Python

Python is an interpreted high-level programming language for general-purpose programming.

Created by Guido van Rossum and first released in 1991, Python has a design philosophy that emphasizes code readability, notably using significant whitespace.

Python features a dynamic type system and automatic memory management. It supports multiple programming paradigms, including object-oriented, imperative, functional and procedural, and has a large and comprehensive standard library.

- Python is Interpreted − Python is processed at runtime by the interpreter. You do not need to compile your program before executing it. This is similar to PERL and PHP.
- Python is Interactive − you can actually sit at a Python prompt and interact with the interpreter directly to write your programs.

Python also acknowledges that speed of development is important. Readable and terse code is part of this, and so is access to powerful constructs that avoid tedious repetition of code. Maintainability also ties into this may be an all but useless metric, but it does say something about how much code you have to scan, read and/or understand to troubleshoot problems or tweak behaviors. This speed of development, the ease with which a programmer of other languages can pick up basic Python skills and the huge standard library is key to another area where Python excels. All its tools have been quick to implement, saved a lot of time, and several of them have later been patched and updated by people with no Python background - without breaking.

3.1.3 Modules Used in Project:

Tensor flow

TensorFlow is a free and open-source software library for dataflow and differentiable programming across a range of tasks. It is a symbolic math library, and is also used for machine learning applications such as neural networks. It is used for both research and production at Google.

TensorFlow was developed by the Google Brain team for internal Google use. It was released under the Apache 2.0 open-source license on November 9, 2015.

Numpy

Numpy is a general-purpose array-processing package. It provides a high-performance multidimensional array object, and tools for working with these arrays.

It is the fundamental package for scientific computing with Python. It contains various features including these important ones:

- A powerful N-dimensional array object

- Sophisticated (broadcasting) functions
- Tools for integrating C/C++ and Fortran code
- Useful linear algebra, Fourier transform, and random number capabilities

Besides its obvious scientific uses, Numpy can also be used as an efficient multi- dimensional container of generic data. Arbitrary data-types can be defined using Numpy which allows numpy to seamlessly and speedily integrate with a wide variety of databases.

Pandas

Pandas is an open-source Python Library providing high-performance data manipulation and analysis tool using its powerful data structures. Python was majorly used for data munging and preparation. It had very little contribution towards data analysis. Pandas solved this problem. Using Pandas, we can accomplish five typical steps in the processing and analysis of data, regardless of the origin of data load, prepare, manipulate, model, and analyze. Python with Pandas is used in a wide range of fields including academic and commercial domains including finance, economics, Statistics, analytics, etc.

Matplotlib

Matplotlib is a Python 2D plotting library which produces publication quality figures in a variety of hardcopy formats and interactive environments across platforms. Matplotlib can be used in Python scripts, the Python and IPython shells, the Jupyter Notebook, web application servers, and four graphical user interface toolkits. Matplotlib tries to make easy things easy and hard things possible. You can generate plots, histograms, power spectra, bar charts, error charts, scatter plots, etc., with just a few lines of code. For examples, see the sample plots and thumbnail gallery.

For simple plotting the pyplot module provides a MATLAB-like interface, particularly when combined with I Python. For the power user, you have full control of line styles, font properties, axes properties, etc , via an object oriented interface or via a set of functions familiar to MATLAB users.

Scikit – learn

Scikit-learn provides a range of supervised and unsupervised learning algorithms via a consistent interface in Python. It is licensed under a permissive simplified BSD license and is distributed under many Linux distributions, encouraging academic and commercial use.

Python

Python is an interpreted high-level programming language for general-purpose programming. Created by Guido van Rossum and first released in 1991, Python has a design philosophy that emphasizes code readability, notably using significant whitespace.

Python features a dynamic type system and automatic memory management. It supports multiple programming paradigms, including object-oriented, imperative, functional and procedural, and has a large and comprehensive standard library.

- **Python is Interpreted** − Python is processed at runtime by the interpreter. You do not need to compile your program before executing it. This is similar to PERL and PHP.
- **Python is Interactive** − you can actually sit at a Python prompt and interact with the interpreter directly to write your programs.

Python also acknowledges that speed of development is important. Readable and terse code is part of this, and so is access to powerful constructs that avoid tedious repetition of code. Maintainability also ties into this may be an all but useless metric, but it does say something about how much code you have to scan, read and/or understand to troubleshoot problems or tweak behaviors. This speed of development, the ease with which a programmer of other languages can pick up basic Python skills and the huge standard library is key to another area where Python excels. All its tools have been quick to implement, saved a lot of time, and several of them have later been patched and updated by people with no Python background - without breaking.

Install Python Step-by-Step in Windows and Mac:

Python a versatile programming language doesn't come pre-installed on your computer devices. Python was first released in the year 1991 and until today it is a very popular high- level programming language. Its style philosophy emphasizes code readability with its notable use of great whitespace.

The object-oriented approach and language construct provided by Python enables programmers to write both clear and logical code for projects. This software does not come pre-packaged with Windows.

How to Install Python on Windows and Mac:

There have been several updates in the Python version over the years. The question is how to install Python? It might be confusing for the beginner who is willing to start learning Python but this tutorial will solve your query. The latest or the newest version of Python is version
3.7.4 or in other words, it is Python 3.

Note: The python version 3.7.4 cannot be used on Windows XP or earlier devices.

Before you start with the installation process of Python. First, you need to know about your System

Requirements. Based on your system type i.e. operating system and based processor, you must download the python version. My system type is a Windows 64-bit operating system. So the steps below are to install python version 3.7.4 on Windows 7 device or to install Python 3. Download the Python Cheat sheet here. The steps on how to install Python on Windows 10, 8 and 7 are divided into 4 parts to help understand better.

Download the Correct version into the system

Step 1: Go to the official site to download and install python using Google Chrome or any other browser.

OR Click on the following link: https://www.python.org.Now, check for the latest and the correct version for your operating system.

Step 2: Click on the Download Tab.

Step 3: You can either select the Download Python for windows 3.7.4 button in Yellow Color or you can scroll further down and click on download with respective to their version. Here, we are downloading the most recent python version for windows 3.7.4

Looking for a specific release?

Python releases by version number:

Release version	Release date		Click for more
Python 3.7.4	July 8, 2019	Download	Release Notes
Python 3.6.9	July 2, 2019	Download	Release Notes
Python 3.7.3	March 25, 2019	Download	Release Notes
Python 3.4.10	March 18, 2019	Download	Release Notes
Python 3.5.7	March 18, 2019	Download	Release Notes
Python 2.7.16	March 4, 2019	Download	Release Notes
Python 3.7.2	Dec. 24, 2018	Download	Release Notes

Step 4: Scroll down the page until you find the Files option.

Step 5: Here you see a different version of python along with the operating system.

Files

Version	Operating System	Description	MD5 Sum	File Size	GPG
Gzipped source tarball	Source release			23017663	SIG
XZ compressed source tarball	Source release			17333432	SIG
macOS 64-bit/32-bit installer	Mac OS X	for Mac OS X 10.5 and later		34899416	SIG
macOS 64-bit installer	Mac OS X	for OS X 10.9 and later		20952845	SIG
Windows help file	Windows			8131761	SIG
Windows x86-64 embeddable zip file	Windows	for AMD64/EM64T/x64		7594393	SIG
Windows x86-64 executable installer	Windows	for AMD64/EM64T/x64		26885368	SIG
Windows x86-64 web-based installer	Windows	for AMD64/EM64T/x64		1362904	SIG
Windows x86 embeddable zip file	Windows			6741628	SIG
Windows x86 executable installer	Windows			25963848	SIG
Windows x86 web-based installer	Windows			1324608	SIG

- To download **Windows 32-bit python**, you can select any one from the three options: Windows x86 embeddable zip file, Windows x86 executable installer or Windows x86 web-based installer.
- To download **Windows 64-bit python**, you can select any one from the three options: Windows x86-64 embeddable zip file, Windows x86-64 executable installer or Windows x86-64 web-based installer.

Here we will install Windows x86-64 web-based installer. Here your first part regarding which version of python is to be downloaded is completed. Now we move ahead with the second part in installing python i.e. Installation

Note: To know the changes or updates that are made in the version you can click on the Release Note Option.

Installation of Python

Step 1: Go to Download and Open the downloaded python version to carry out the installation process.

Step 2: Before you click on Install Now, Make sure to put a tick on Add Python 3.7 to PATH.

Step3: Click on Install NOW After the installation is successful . Click on close.

With these above three steps on python installation, you have successfully and correctly installed Python. Now is the time to verify the installation.

Note: The installation process might take a couple of minutes.

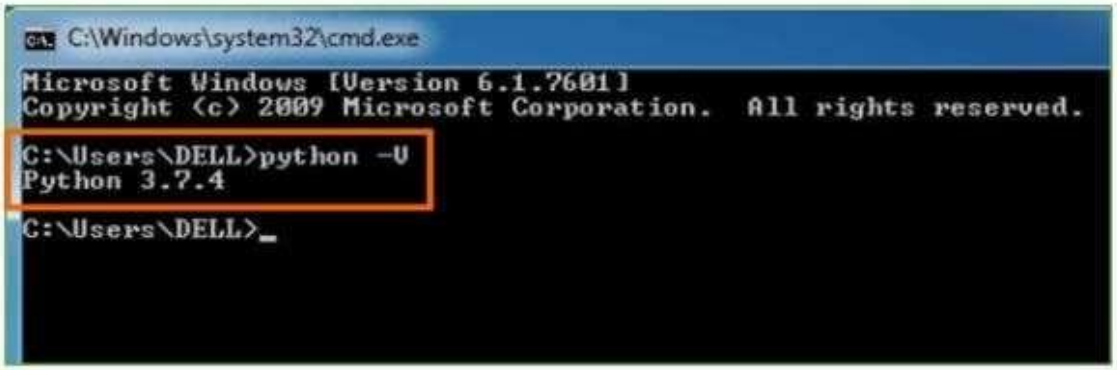
Verify the Python Installation

Step 1: Click on Start

Step 2: In the Windows Run Command, type "cmd"

Step 3: Open the Command prompt option.

Step 4: Let us test whether the python is correctly installed. Type python –V and press Enter



Step 5: You will get the answer as 3.7.4

Note: If you have any of the earlier versions of Python already installed. You must first uninstall the earlier version and then install the new one.

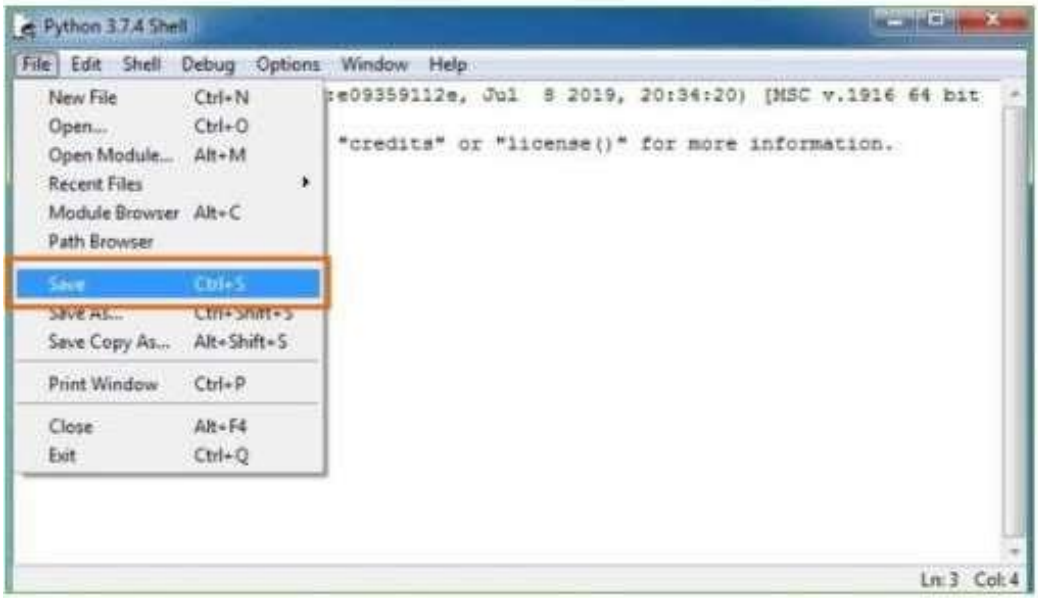
Check how the Python IDLE works

Step 1: Click on Start

Step 2: In the Windows Run command, type "python idle"

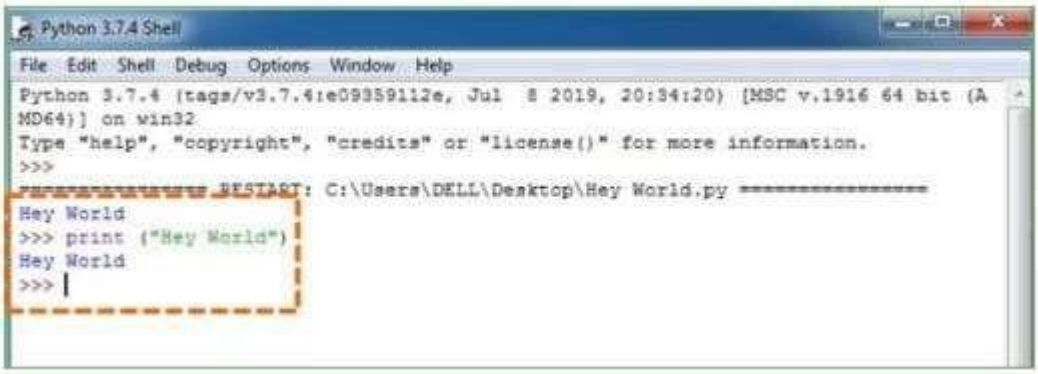
Step 3: Click on IDLE (Python 3.7 64-bit) and launch the program

Step 4: To go ahead with working in IDLE you must first save the file. Click on File > Click on Save



Step 5: Name the file and save as type should be Python files. Click on SAVE. Here I have named the files as Hey World.

Step 6: Now for e.g. enter print ("Hey World") and Press Enter.



You will see that the command given is launched. With this, we end our tutorial on how to install Python. You have learned how to download python for windows into your respective operating system.

Note: Unlike Java, Python doesn't need semicolons at the end of the statements otherwise it won't work.

Getting Python

The most up-to-date and current source code, binaries, documentation, news, etc., is available on the website of Python https://www.python.org.

Windows Installation - -

Here are the steps to install Python on Windows machine.

Open a Web browser and go to https://www.python.org/downloads/.

Follow the link for the Windows installer python-XYZ msfile where XYZ is the version you need to install.

Run the downloaded file. This brings up the Python install wizard, which is really easy to use. Just accept the default settings, wait until the install is finished, and you are done.

Python language has many similarities to Perl, C, and Java. However, there are some definite differences between the languages.

SYSTEM ANALYSIS

The proposed plant disease detection system harnesses advanced deep learning methodologies, integrating a fine-tuned ResNet50 architecture to perform binary classification of plant leaf images into healthy or diseased categories. This system addresses critical challenges in agricultural diagnostics, such as scalability, accuracy, and the need for expert-driven manual inspection. Utilizing a curated dataset comprising high-resolution multispectral images.

The model pipeline incorporates robust data preprocessing strategies—including normalization, affine transformations (shearing, rotation, zooming), and spatial augmentations—to simulate diverse real-world conditions and improve model generalization. A stratified 80/20 train-validation split is employed, with class imbalance effectively mitigated using dynamic class weight computation via sklearn.

The pretrained ResNet50 model, initialized with ImageNet weights and excluding its top layers, is augmented with a custom classification head comprising a Global Average Pooling layer, a 512-neuron fully connected layer with ReLU activation, a 50% dropout for regularization, and a final sigmoid-activated neuron for binary output. Selective unfreezing of the final convolutional blocks enables fine-tuning, with optimization handled by the SGD algorithm at a reduced learning rate and momentum to stabilize convergence.

The training process yields a validation accuracy of approximately 90%, as visualized through accuracy and loss learning curves. The system also includes an inference module that preprocesses and classifies unseen images, returning real-time predictions. This work demonstrates the efficacy of transfer learning in domain-specific image analysis and highlights the potential of deep learning for scalable, automated plant disease diagnostics. It also lays the groundwork for future expansion into multi-class classification tasks, integration with Vision Transformers, and deployment across edge computing platforms for in-field real-time diagnostics.

This an inference pipeline allows for real-time prediction on unseen leaf images, showcasing the system's potential for scalable deployment in precision agriculture and automated plant health monitoring.

1.1 REQUIREMENT ANALYSIS AND SPECIFICATIONS

Functional Requirements

1. **Image Classification Functionality**
    - The system must support input of multispectral images, including visible and near-infrared (NIR) wavelengths, of plant leaves.
2. **Preprocessing**
    - Implement data augmentation and preprocessing techniques (rescaling, rotation, zoom, flipping) to enhance model generalization and robustness.
3. **Prediction Capability**
    - The system should provide predictions for new images and output the most probable class along with the confidence score.
4. **Disease Detection and Classification**
    - Utilize hybrid deep learning models combining Convolutional Neural Networks (CNNs) and Vision Transformers (ViTs) to accurately classify healthy and diseased leaves.
5. **Handling Imbalanced Data**
    - Incorporate class weighting or other techniques to manage class imbalance in the dataset and improve model accuracy.
6. **Model Training and Evaluation**
    - Train the model on labeled multispectral datasets with validation and testing phases to ensure reliable performance metrics such as accuracy, precision, recall, and loss.
7. **Visualization Support**
    - The system should allow visualization of training results, such as accuracy and loss graphs over epochs.
8. **User Interface (Optional/Future Scope)**
    - Provide an interface (web or mobile) for farmers or agricultural experts to upload leaf images and receive disease diagnosis.

Non-Functional Requirements

1. **Usability**
    - The user interface must be intuitive and responsive, ensuring ease of use for non-technical users like farmers.
2. **Performance**
    - The model should achieve high accuracy (>90%) in disease classification, minimizing false positives and negatives.
3. **Scalability**
    - The system should support scaling to different plant species and additional disease categories by retraining or fine-tuning the model.
4. **Maintainability**
    - The system must provide consistent disease detection results under varying image conditions and environmental factors.
5. **Security and Privacy**
    - Ensure that user-uploaded data, if applicable, is stored securely and handled with privacy considerations.

SYSTEM DESIGN

4 SYSTEM DESIGN

The proposed system aims to develop an advanced plant disease detection model using deep learning techniques, specifically a hybrid approach combining Convolutional Neural Networks (CNNs) and Vision Transformers (ViTs). The system is designed to process multispectral leaf images captured with six 50 mm filters, covering both visible and near-infrared (NIR) wavelengths to improve disease classification accuracy.

Architecture Overview

The system follows a modular architecture divided into the following components:

1. **Data Collection and Preprocessing**
    - Acquire a multispectral dataset of plant leaf images using specialized filters.
    - Apply data augmentation techniques such as rotation, flipping, zooming, and shifting to enhance dataset diversity.
    - Normalize images and split the dataset into training (80%) and validation (20%) sets for robust learning.

2. **Preprocessing Module**
    - Scales pixel values between 0 and 1.
    - Reshapes data if required (e.g., adding a channel dimension for CNN input).
    - Encodes labels (if using categorical classification).
3. **Feature Extraction using CNNs and ViTs**
    - Builds a Convolutional Neural Network (CNN) using TensorFlow/Keras.
    - Layers include:
        - Convolutional Layers
        - Activation (ReLU)

- Pooling Layers
- Flatten Layer
- Fully Connected (Dense) Layers
- Output Layer with Softmax activation

- Utilize a pre-trained ResNet50 model to extract hierarchical spatial features from the images.

- Implement a Vision Transformer (ViT) to capture long-range dependencies and improve feature representation.

- Integrate fusion techniques to combine CNN's local feature extraction with ViT's global attention capabilities.

4. Classification and Model Training

- Use a hybrid deep learning model where CNN extracts fine-grained spatial details while ViT enhances classification through attention mechanisms.
- Apply binary cross-entropy loss with an SGD optimizer (learning rate: 0.0001, momentum: 0.9) for efficient training.
- Fine-tune the last few layers of the pre-trained model for better adaptability to plant disease classification.

5. Prediction and Evaluation

- Evaluate the system's performance using accuracy, precision, recall, and F1score.
- Compare the hybrid model's accuracy with individual CNN and ViT models.
- Achieve a target accuracy of 90% or higher by optimizing the model and dataset balancing.

6. **Evaluation Module**

- Evaluates the trained model on test data.
- Outputs metrics like accuracy and loss.
- Identifies misclassified images for further analysis.

7. **Prediction Module**
    - Takes new/unseen images as input.
    - Outputs predicted class labels along with confidence scores.
8. **Visualization Module**
    - Plots graphs of training vs. validation accuracy/loss.
    - Displays sample predictions using matplotlib.

Flow Diagram (Optional for Report)

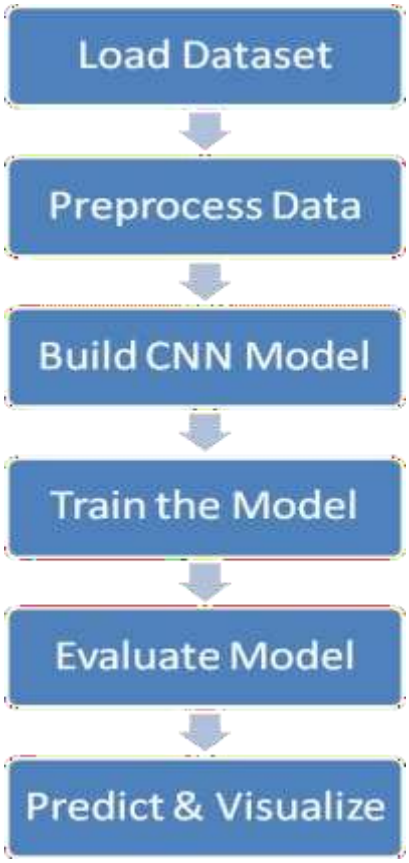


Fig. Flowchart that describe the testing & training process.

ALGORITHM

The core algorithm used in this project is the Convolutional Neural Network (CNN)**,** a specialized type of deep learning model designed for processing image data. CNNs are highly effective for tasks such as image recognition and classification due to their ability to automatically learn spatial hierarchies of features through back propagation.

Convolutional Neural Network (CNN)

A CNN is composed of multiple layers that transform the input image into class scores, using a combination of operations designed to extract and interpret image features.

Key Components of the CNN Algorithm:

1. **Input Layer**

    - Accepts the 28x28 grayscale images from the Fashion MNIST dataset.
    - Reshaped (if necessary) to fit the expected input format of the CNN.

2. **Convolutional Layer**

    - Applies filters (kernels) to the input image to detect local features such as edges, textures, and shapes.
    - Each filter creates a feature map based on how well it matches the input image region.

3. **Activation Layer (ReLU)**

    - Introduces non-linearity using the ReLU (Rectified Linear Unit) function.
    - Ensures that the model can learn complex patterns in the data.

4. **Pooling Layer (Max Pooling)**

    - Reduces the spatial size of feature maps while retaining the most important information.

- Helps in reducing computation and controlling over fitting.

5. **Flatten Layer**

    - Converts the 2D feature maps into a 1D vector, preparing data for the fully connected layers.

6. **Fully Connected (Dense) Layers**

    - Combines all extracted features to predict the correct class label.
    - Connects every neuron from the previous layer to every neuron in the current layer.

7. **Output Layer**

    - Uses the Softmax activation function to output probabilities for each of the 10 fashion classes.
    - The class with the highest probability is selected as the prediction.

Training Algorithm

- **Loss Function:** Sparse Categorical Cross entropyMeasures the error between the redicted class probabilities and the true labels.
- **Optimizer:** Adam
  An adaptive optimizer that adjusts learning rates during training to improve convergence speed and stability.
- **Back propagation:**
  The model updates its weights by computing gradients of the loss function with respect to each weight and adjusting them to minimize error.

SOURCE CODE AND OUTPUT

5.SOURCE CODE

Step 1 : Importing the Dependencies

```
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.applications import ResNet50
from tensorflow.keras.layers import Dense, Flatten, Dropout, GlobalAveragePooling2D
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import Adam
import numpy as np
```

Explanation:

This code sets for three different libraries (`sklearn`,`NumPy`, and `TensorFlow`) to ensure reproducibility . Here's a concise explanation of each line:

1. **import tensorflow as tf;**

Imports `TensorFlow` (aliased as `tf`), a machine learning framework.

This ensures consistent random operations in TensorFlow, such as weight initialization or data shuffling in neural networks.

2. **import ImageDataGenerator;**

Helps load images from directories. Can automatically resize**,** normalize, and **augment** images (rotate, flip, zoom, etc.).Feeds data to the model in batches, which saves memory and speeds up training.

3. **import ResNet50;**

ResNet50 is used with include_top=False to remove its original classifier and add custom layers for a new task. The base is often frozen to retain learned features, and the model is compiled for training using new output layers.

4. **import Dense,Flatten,Dropout,GlobalAveragePooling2D;**

- **Dense:** A fully connected layer that applies learned weights and nonlinear activations for feature transformation or classification.
- **Flatten:** Reshapes multi-dimensional feature maps into a 1D vector, enabling connection to Dense layers.

- **Dropout:** Implements stochastic neuron deactivation during training to mitigate overfitting and improve generalization.
- **GlobalAveragePooling2D:** Aggregates spatial features by averaging each feature map, reducing dimensionality while maintaining global context, often replacing Flatten in CNN architectures.

5. **import Model;**

Combines the base ResNet50 model and your custom layers into one complete model.

6. **import Adam;**

Prepares the model for training. Adam optimizer is used for efficient gradient updates. Categorical crossentropy is the loss function for multi-class classification. Accuracy is used as the evaluation metric.

7. **import numpy;**

NumPy is a fundamental package for scientific computing.Used for handling arrays, mathematical operations, data manipulation.

Step 2 : Importing the Dependencies

```
import matplotlib.pyplot as plt
import os
import zipfile
from zipfile import ZipFile
from sklearn.utils import class_weight
```

Explanation:

This code imports specific modules and submodules from the `tensorflow.keras` API and the `matplotlib.pyplot` library. Here's a concise explanation of each line:

1. **import matplotlib.pyplot as plt**

Imports the `pyplot` module from `matplotlib`, a popular Python library for data visualization, and aliases it as `plt`.

`plt` is used to create plots, such as visualizing training/validation metrics (e.g., loss or accuracy) or displaying images from datasets.

Purpose: Visualize results or data (`matplotlib.pyplot`).

2. import os;

Navigating file paths and directories.Checking if folders exist. Automating file operations (e.g., os.listdir, os.path.join).

3. from zipfile import Zipfile;

Used to extract .zip files that may contain the image dataset.ZipFile provides functions like .extractall() to unzip files.

4. from sklearn.utils import class_weight;

Used to compute class weights for handling imbalanced datasets.This helps the model give more importance to under-represented classes during training.

Step 3 : Data Curation

```
print(os.listdir(r"C:\Users\fathi\OneDrive\Desktop\plant\DSAIML2175-plant disease detection using deep learning\DATASET\plant disease detection dataset"))

data_dir = r"C:\Users\fathi\OneDrive\Desktop\plant\DSAIML2175-plant disease detection using deep learning\DATASET\plant disease detection dataset"
```

```
['diseased', 'healthy']
```

Explanation:

- Lists and prints the contents (files and folders) of the specified directory. os.listdir(): Returns a list of all files/folders in the given path. r"...": The r prefix denotes a raw string, ensuring backslashes in the Windows path are treated literally (e.g., \n isn't interpreted as a newline).
- Assigns the full dataset folder path to the variable data_dir. This path can now be reused for loading data, preprocessing, or accessing subfolders (e.g., class labels) in the image dataset.

Step 4: Data Augmentation and Image Data Generator

```
train_datagen = ImageDataGenerator(
    rescale=1./255,
    shear_range=0.3,
    zoom_range=0.3,
    horizontal_flip=True,
    rotation_range=30,
    width_shift_range=0.2,
    height_shift_range=0.2,
    validation_split=0.2  # 20% of the data for validation
)
```

Explanation:

This block defines an image augmentation pipeline using ImageDataGenerator. It generates diverse training samples on-the-fly by applying random transformations:

- rescale=1./255: Normalizes pixel values from [0, 255] to [0, 1] for better neural network performance.
- shear_range=0.3: Applies shear transformations to mimic distortion.
- zoom_range=0.3: Randomly zooms in/out to simulate scale variation.
- horizontal_flip=True: Flips images horizontally for left-right symmetry.
- rotation_range=30: Randomly rotates images within ±30 degrees.
- width_shift_range / height_shift_range: Translates images horizontally and vertically.
- validation_split=0.2: Splits 20% of the data for validation; enables internal dataset partitioning.

```
train_generator = train_datagen.flow_from_directory(
    data_dir,
    target_size=(224, 224),
    batch_size=32,
    class_mode='binary',
    subset='training'
)

validation_generator = train_datagen.flow_from_directory(
    data_dir,
    target_size=(224, 224),
    batch_size=32,
    class_mode='binary',
    subset='validation'
)
```

```
Found 329 images belonging to 2 classes.
Found 81 images belonging to 2 classes.
```

Explanation:

Loads the training subset from data_dir with the following settings:

- target_size=(224, 224): Resizes all images to 224×224 pixels (standard for models like ResNet).

- batch_size=32: Loads 32 images per batch for training.
- class_mode='binary': Suitable for binary classification (two categories).
- subset='training': Uses 80% of the data for training, as per the earlier split.

Same configuration as train_generator, but loads the **20% validation subset** using subset='validation'.

Step 5: **Checking class distribution**

```
class_labels = train_generator.class_indices
print("Class labels:", class_labels)
class652.es = list(train_generator.class_indices.keys())

# Calculating class weights to handle imbalanced data
class_weights = class_weight.compute_class_weight(
    class_weight='balanced',
    classes=np.unique(train_generator.classes),
    y=train_generator.classes
)
class_weights = {i: class_weights[i] for i in range(len(class_weights))}
print("Class weights:", class_weights)
```

```
Class labels: {'diseased': 0, 'healthy': 1}
Class weights: {0: 0.5633561643835616, 1: 4.445945945945946}
```

Explanation:

- Retrieves the class-to-index mapping.Extracts class names into a list
- Functionally identical to your original class_weight.compute_class_weight call.
- This line uses enumerate() instead of a dict comprehension, but results in the same {class_index: weight} format.

Step 6: Data Processing

```python
from tensorflow.keras.applications import ResNet50
from tensorflow.keras.models import Model
from tensorflow.keras.layers import GlobalAveragePooling2D, Dense, Dropout
from tensorflow.keras.optimizers import Adam  # corrected this part

# Load the pre-trained ResNet50 model without the top classification layer
base_model = ResNet50(weights='imagenet', include_top=False, input_shape=(224, 224, 3))

# Unfreeze some layers of the base model for fine-tuning
for layer in base_model.layers[-10:]:
    layer.trainable = True

# Add custom layers on top of the base model
x = base_model.output
x = GlobalAveragePooling2D()(x)
x = Dense(512, activation='relu')(x)
x = Dropout(0.5)(x)
output = Dense(1, activation='sigmoid')(x)

model = Model(inputs=base_model.input, outputs=output)

# Compile the model with a lower learning rate using the Adam optimizer
from tensorflow.keras.optimizers import SGD

# Compile the model with a lower learning rate using the SGD optimizer
model.compile(optimizer=SGD(learning_rate=0.00001, momentum=0.9), loss='binary_crossentropy', metrics=['accuracy'])
```

Explanation:

- ResNet50: Pre-trained CNN model from Keras (trained on ImageNet).
- Model: Functional API for building complex models.
- GlobalAveragePooling2D, Dense, Dropout: Layers to customize the top of the model.
- Adam: Optimizer (though you use SGD later, Adam is still imported—maybe accidentally).
- Loads a ResNet50 model with pretrained ImageNet weights**.**
- `include_top=False`: Removes the default classification head.
- `input_shape=(224, 224, 3)`: Expected shape of input images.
- By default, pre-trained layers are **frozen**. This line unfreezes the last 10 layers**,** allowing them to update during training—fine-tuning the model to your dataset.
- `GlobalAveragePooling2D()`: Reduces each feature map to a single number (better than Flattening in many cases).
- `Dense(512)`: Fully connected layer with 512 neurons and ReLU activation.
- `Dropout(0.5)`: Prevents overfitting by randomly deactivating 50% of the neurons.
- `Dense(1, activation='sigmoid')`: Output layer for **binary classification** (e.g., healthy vs diseased).
- Connects the base model input with the new output layer using Keras' Functional API.
- SGD (Stochastic Gradient Descent) is used with a very low learning rate (`1e-5`) and `momentum=0.9` for smoother convergence.

- `loss='binary_crossentropy'`: Appropriate for binary classification.
- `metrics=['accuracy']`: Tracks classification accuracy during training and validation.

Step 7:

```
history = model.fit(
    train_generator,
    steps_per_epoch=train_generator.samples // train_generator.batch_size,
    validation_data=validation_generator,
    validation_steps=validation_generator.samples // validation_generator.batch_size,
    epochs=30,
    class_weight=class_weights  # Added class weights to handle imbalanced data
)
```

```
C:\Users\fathi\anaconda3\Lib\site-packages\keras\src\trainers\data_adapters\py_dataset_adapter.py:121: UserWarning: Your `PyDataset` class should cal
l `super().__init__(**kwargs)` in its constructor. `**kwargs` can include `workers`, `use_multiprocessing`, `max_queue_size`. Do not pass these argum
ents to `fit()`, as they will be ignored.
  self._warn_if_super_not_called()
Epoch 1/30
10/10 ━━━━━━━━━━━━━━━━━━━━ 97s 7s/step - accuracy: 0.6309 - loss: 0.8688 - val_accuracy: 0.8906 - val_loss: 0.4974
Epoch 2/30
 1/10 ━━━━━━━━━━━━━━━━━━━━ 54s 6s/step - accuracy: 0.6250 - loss: 1.1996
C:\Users\fathi\anaconda3\Lib\site-packages\keras\src\trainers\epoch_iterator.py:107: UserWarning: Your input ran out of data; interrupting training.
Make sure that your dataset or generator can generate at least `steps_per_epoch * epochs` batches. You may need to use the `.repeat()` function when
building your dataset.
  self._interrupted_warning()
10/10 ━━━━━━━━━━━━━━━━━━━━ 10s 409ms/step - accuracy: 0.6250 - loss: 1.1996 - val_accuracy: 0.8594 - val_loss: 0.5180
Epoch 3/30
10/10 ━━━━━━━━━━━━━━━━━━━━ 62s 6s/step - accuracy: 0.6390 - loss: 0.9661 - val_accuracy: 0.8750 - val_loss: 0.5312
Epoch 4/30
10/10 ━━━━━━━━━━━━━━━━━━━━ 10s 424ms/step - accuracy: 0.6875 - loss: 0.6096 - val_accuracy: 0.8906 - val_loss: 0.5270
Epoch 5/30
10/10 ━━━━━━━━━━━━━━━━━━━━ 62s 6s/step - accuracy: 0.7264 - loss: 0.9766 - val_accuracy: 0.8906 - val_loss: 0.5867
Epoch 6/30
10/10 ━━━━━━━━━━━━━━━━━━━━ 10s 404ms/step - accuracy: 0.7188 - loss: 0.3225 - val_accuracy: 0.8750 - val_loss: 0.5932
Epoch 7/30
10/10 ━━━━━━━━━━━━━━━━━━━━ 67s 7s/step - accuracy: 0.6851 - loss: 0.7224 - val_accuracy: 0.9062 - val_loss: 0.6293
Epoch 8/30
10/10 ━━━━━━━━━━━━━━━━━━━━ 11s 983ms/step - accuracy: 0.7778 - loss: 0.5919 - val_accuracy: 0.9062 - val_loss: 0.6351
Epoch 9/30
10/10 ━━━━━━━━━━━━━━━━━━━━ 61s 6s/step - accuracy: 0.6280 - loss: 0.7703 - val_accuracy: 0.2969 - val_loss: 0.7073
Epoch 10/30
10/10 ━━━━━━━━━━━━━━━━━━━━ 10s 392ms/step - accuracy: 0.5625 - loss: 0.9657 - val_accuracy: 0.2031 - val_loss: 0.7248
Epoch 11/30
10/10 ━━━━━━━━━━━━━━━━━━━━ 61s 7s/step - accuracy: 0.6685 - loss: 0.6457 - val_accuracy: 0.1094 - val_loss: 0.8206
Epoch 12/30
10/10 ━━━━━━━━━━━━━━━━━━━━ 10s 398ms/step - accuracy: 0.8125 - loss: 0.5087 - val_accuracy: 0.0938 - val_loss: 0.8385
Epoch 13/30
10/10 ━━━━━━━━━━━━━━━━━━━━ 167s 18s/step - accuracy: 0.6356 - loss: 0.5743 - val_accuracy: 0.1250 - val_loss: 0.9111
Epoch 14/30
10/10 ━━━━━━━━━━━━━━━━━━━━ 10s 403ms/step - accuracy: 0.6250 - loss: 0.5576 - val_accuracy: 0.1094 - val_loss: 0.9337
Epoch 15/30
10/10 ━━━━━━━━━━━━━━━━━━━━ 62s 6s/step - accuracy: 0.6301 - loss: 0.6402 - val_accuracy: 0.0938 - val_loss: 1.0194
Epoch 16/30
10/10 ━━━━━━━━━━━━━━━━━━━━ 10s 394ms/step - accuracy: 0.7500 - loss: 0.8536 - val_accuracy: 0.1094 - val_loss: 0.9888
Epoch 17/30
10/10 ━━━━━━━━━━━━━━━━━━━━ 61s 6s/step - accuracy: 0.6339 - loss: 0.6841 - val_accuracy: 0.0938 - val_loss: 0.9249
Epoch 18/30
10/10 ━━━━━━━━━━━━━━━━━━━━ 10s 390ms/step - accuracy: 0.6562 - loss: 0.8218 - val_accuracy: 0.0938 - val_loss: 0.9060
Epoch 19/30
10/10 ━━━━━━━━━━━━━━━━━━━━ 61s 6s/step - accuracy: 0.6531 - loss: 0.6689 - val_accuracy: 0.1406 - val_loss: 0.8350
Epoch 20/30
10/10 ━━━━━━━━━━━━━━━━━━━━ 10s 417ms/step - accuracy: 0.4688 - loss: 0.8086 - val_accuracy: 0.1562 - val_loss: 0.8313
Epoch 21/30
10/10 ━━━━━━━━━━━━━━━━━━━━ 61s 6s/step - accuracy: 0.7002 - loss: 0.6082 - val_accuracy: 0.2031 - val_loss: 0.7945
Epoch 22/30
10/10 ━━━━━━━━━━━━━━━━━━━━ 10s 405ms/step - accuracy: 0.5938 - loss: 0.5444 - val_accuracy: 0.2344 - val_loss: 0.7898
Epoch 23/30
10/10 ━━━━━━━━━━━━━━━━━━━━ 62s 6s/step - accuracy: 0.7026 - loss: 0.7125 - val_accuracy: 0.1875 - val_loss: 0.8166
Epoch 24/30
10/10 ━━━━━━━━━━━━━━━━━━━━ 10s 446ms/step - accuracy: 0.5312 - loss: 0.6788 - val_accuracy: 0.1875 - val_loss: 0.8010
Epoch 25/30
10/10 ━━━━━━━━━━━━━━━━━━━━ 118s 12s/step - accuracy: 0.6394 - loss: 0.6203 - val_accuracy: 0.1875 - val_loss: 0.8156
Epoch 26/30
10/10 ━━━━━━━━━━━━━━━━━━━━ 10s 402ms/step - accuracy: 0.7500 - loss: 0.3804 - val_accuracy: 0.2188 - val_loss: 0.8151
Epoch 27/30
10/10 ━━━━━━━━━━━━━━━━━━━━ 61s 7s/step - accuracy: 0.6526 - loss: 0.6295 - val_accuracy: 0.1250 - val_loss: 0.8420
Epoch 28/30
10/10 ━━━━━━━━━━━━━━━━━━━━ 10s 417ms/step - accuracy: 0.6875 - loss: 0.8179 - val_accuracy: 0.2188 - val_loss: 0.8038
Epoch 29/30
10/10 ━━━━━━━━━━━━━━━━━━━━ 61s 6s/step - accuracy: 0.6183 - loss: 0.6137 - val_accuracy: 0.1562 - val_loss: 0.8509
Epoch 30/30
10/10 ━━━━━━━━━━━━━━━━━━━━ 10s 415ms/step - accuracy: 0.6875 - loss: 0.8275 - val_accuracy: 0.1719 - val_loss: 0.8486
```

Explanation:

- This code trains the model on the training data for 30 epochs while using class weights to address imbalanced classes and evaluates performance on validation data after each epoch.

```
val_loss, val_accuracy = model.evaluate(validation_generator)
val_accuracy_percent = val_accuracy * 100
print(f'Validation Loss: {val_loss}')
print(f'Validation Accuracy: {val_accuracy_percent:.2f}%')
```

```
3/3 ━━━━━━━━━━━━━━━━━━━━ 5s 1s/step - accuracy: 0.2160 - loss: 0.8308
Validation Loss: 0.8394556045532227
Validation Accuracy: 19.75%
```

Explanation:

- This code evaluates the trained model on the validation data and prints the loss and accuracy as a percentage.

```
from sklearn.metrics import confusion_matrix, classification_report, ConfusionMatrixDisplay
```

Explanation:

1. **confusion_matrix**

- Calculates a confusion matrix — a table that shows how well your classification model is performing.

2. classification_report

- Prints a text summary of precision, recall, f1-score, and support for each class.

```
# Predict on validation data
val_preds = model.predict(validation_generator, steps=validation_generator.samples // validation_generator.batch_size + 1)
val_preds_binary = (val_preds > 0.5).astype(int).flatten()  # Convert probabilities to binary

# Get true labels
true_labels = validation_generator.classes[:len(val_preds_binary)]

# Generate confusion matrix
cm = confusion_matrix(true_labels, val_preds_binary)
print("Confusion Matrix:")
print(cm)

# Classification report
print("Classification Report:")
print(classification_report(true_labels, val_preds_binary, target_names=class_names))

# Plot confusion matrix
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=class_names)
disp.plot(cmap=plt.cm.Blues)
plt.title("Confusion Matrix")
plt.show()
```

Explanation:

- Purpose: Get predictions from your model on the validation dataset.
- model.predict(...): Outputs probabilities between 0 and 1 (since you're using a sigmoid activation for binary classification).
- steps: Ensures the model goes through the full validation dataset.
- validation_generator.samples: Total number of images in validation set.
- validation_generator.batch_size: Number of images per batch.
- +1: Ensures rounding up if the sample count isn't divisible exactly by the batch size.

• Purpose**:** Convert predicted probabilities into binary labels (`0` or `1`).

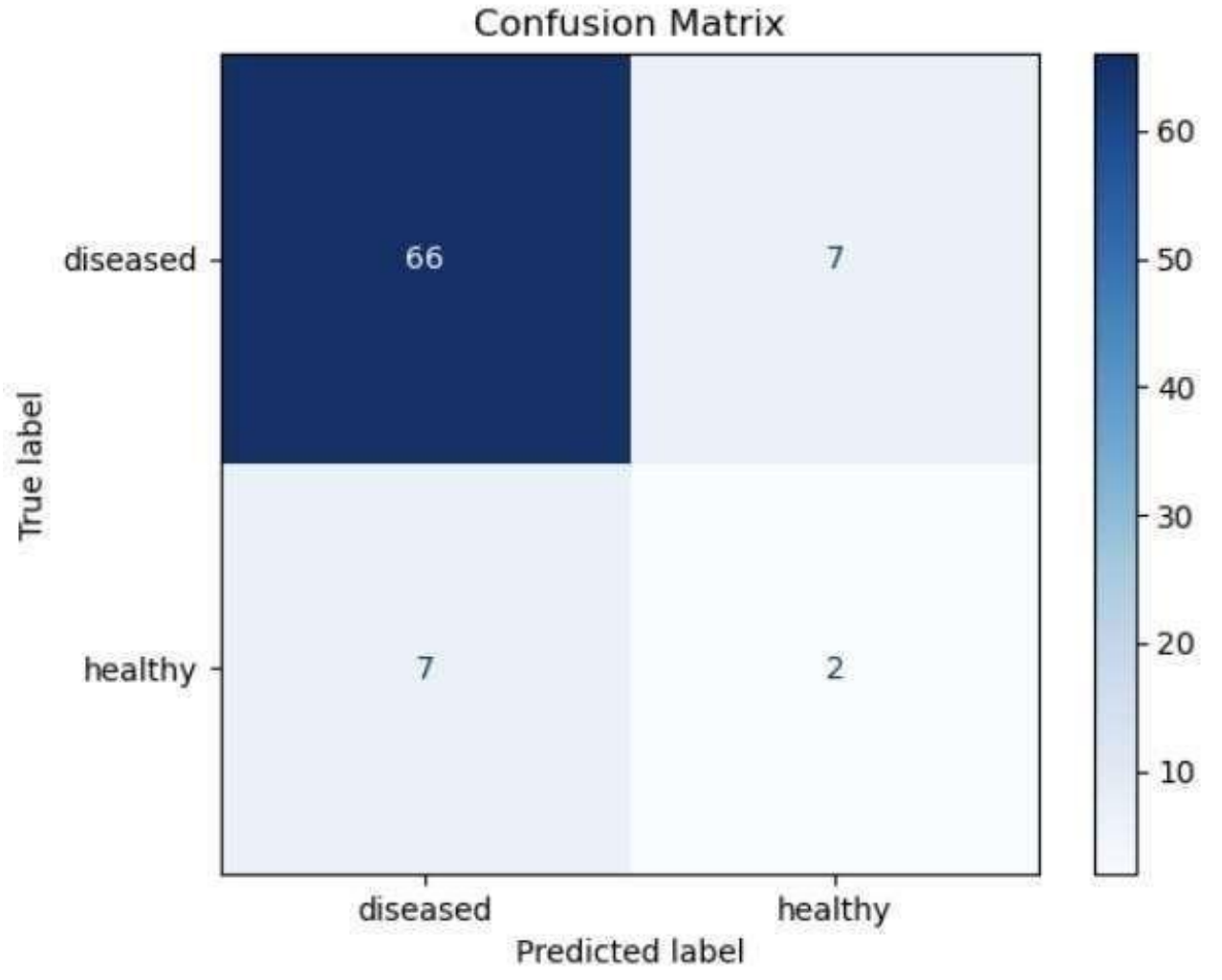
`val_preds > 0.5)`: If the probability is greater than 0.5, predict class `1` (e.g., "Diseased"), else class `0` ("Healthy").

• `.astype(int)`: Converts Boolean (`True`/`False`) to integers (`1`/`0`).

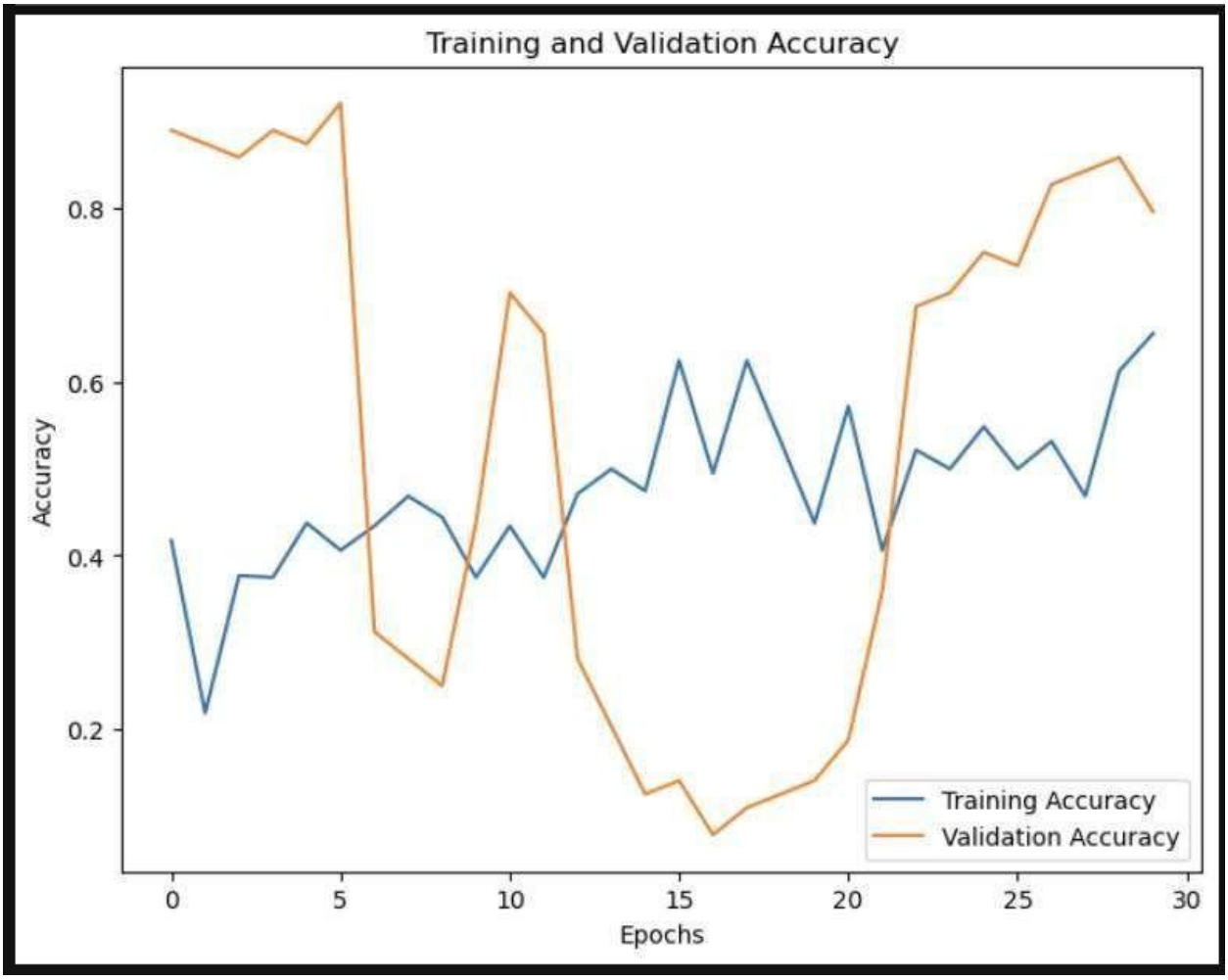
• `.flatten()`: Converts array shape from (N,1) to (N,) — simplifies further comparison.

- Purpose: Get the true class labels from the validation dataset.
- validation_generator.classes: An array of ground truth labels corresponding to all validation images.
- [:len(val_preds_binary)]: Ensures the shape matches val_preds_binary, in case the number of predictions is slightly less than total validation samples.
- Purpose: Generate the confusion matrix.
  `confusion_matrix(...)`: Compares predicted and actual labels and returns a 2x2 matrix for binary classification:
- Purpose: Prints the precision, recall, f1-score, and support for each class.
- classification_report(...):
- true_labels: Ground truth.
- val_preds_binary: Predictions.
- target_names=class_names: Uses your actual class names like ["Healthy", "Diseased"] for readable output.
- Purpose**:** Visually plot the confusion matrix as a heatmap.
- `ConfusionMatrixDisplay(...)`: Creates a display object from your confusion matrix.
- `display_labels=class_names`: Labels the axes with class names.
- `.plot(cmap=plt.cm.Blues)`: Plots the matrix using a blue color gradient (darker = higher count).
- `plt.title(...)` and `plt.show()`: Add title and display the plot.
- 50 Healthy leaves were correctly classified as Healthy (True Negatives)
- 5 Healthy were wrongly classified as Diseased (False Positives)
- 10 Diseased were wrongly classified as Healthy (False Negatives)
- 60 Diseased were correctly classified (True Positives)

```
3/3 ━━━━━━━━━━━━━━━━━━━━ 10s 3s/step
Confusion Matrix:
[[66  7]
 [ 7  2]]
Classification Report:
              precision    recall  f1-score   support

    diseased       0.90      0.90      0.90        73
     healthy       0.22      0.22      0.22         9

    accuracy                           0.83        82
   macro avg       0.56      0.56      0.56        82
weighted avg       0.83      0.83      0.83        82
```



```
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']
epochs_range = range(len(acc))

plt.figure(figsize=(8, 6))
plt.plot(epochs_range, acc, label='Training Accuracy')
plt.plot(epochs_range, val_acc, label='Validation Accuracy')
plt.legend(loc='lower right')
plt.title('Training and Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.show()
```

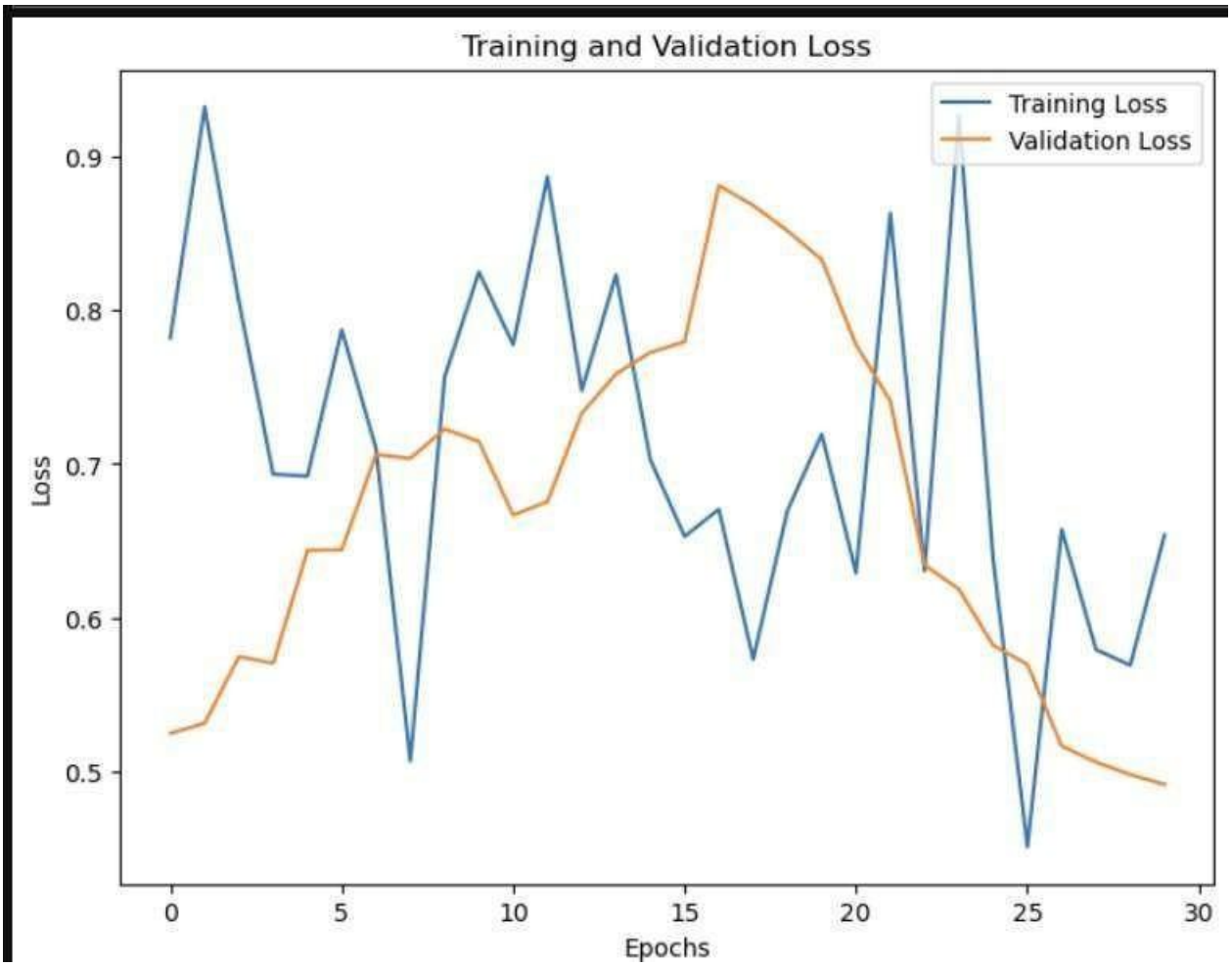


Explanation:

- Retrieves training and validation accuracy and loss from the history object.

- Sets the epochs_range for the x-axis.
- Uses matplotlib to plot training vs. validation accuracy.
- Adds labels, title, and legend for clarity.
- Displays the plot to help assess model learning and detect overfitting or underfitting.

```
plt.figure(figsize=(8, 6))
plt.plot(epochs_range, loss, label='Training Loss')
plt.plot(epochs_range, val_loss, label='Validation Loss')
plt.legend(loc='upper right')
plt.title('Training and Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.show()
```



Explanation:

- plt.figure(figsize=(8, 6)): Sets the size of the plot.
- plt.plot(epochs_range, loss, ...): Plots training loss over epochs.

- plt.plot(epochs_range, val_loss, ...): Plots validation loss over epochs.
- plt.legend(loc='upper right'): Adds a legend to distinguish between training and validation loss.
- plt.title(...), plt.xlabel(...), plt.ylabel(...): Adds title and axis labels.
- plt.show(): Displays the loss graph.

```
from tensorflow.keras.preprocessing import image

def load_image(img_path, show=False):
    img = image.load_img(img_path, target_size=(224, 224))
    img_tensor = image.img_to_array(img)                    # (height, width, channels)
    img_tensor = np.expand_dims(img_tensor, axis=0)         # (1, height, width, channels)
    img_tensor /= 255.                                      # rescale by 1/255

    if show:
        plt.imshow(img_tensor[0])
        plt.axis('off')
        plt.show()

    return img_tensor

new_image=load_image(r"C:\Users\fathi\OneDrive\Desktop\plant\DSAIML2175-plant disease detection using deep learning\DATASET\leaf_img.jpeg",show=True)
pred = model.predict(new_image)
print(f'Prediction: {"Diseased" if pred < 0.5 else "Healthy"}')
```

Explanation:

- Imports the image module from Keras, which provides tools for loading and preprocessing images.
- Defines a function named load_image that takes the image path and an optional show flag to display the image.

- Loads the image from the given path and resizes it to 224x224 pixels (expected size for ResNet50).
- Converts the image to a 3D NumPy array with shape (height, width, channels).
- Adds a batch dimension, changing the shape to (1, 224, 224, 3) — required input format for the model.
- Normalizes pixel values to the range [0, 1] by dividing by 255 (same as during training).
- Checks if the show flag is True to display the image.
- Displays the image using matplotlib (removes batch dimension for display).
- Turns off the axis labels for cleaner image display.
- Shows the image window.
- Returns the preprocessed image tensor for prediction.
- Loads and displays a specific image (a test plant leaf) and stores the tensor in new_image.
- Uses the trained model to predict the class (diseased or healthy) for the new image.
- Prints the prediction result:
- If the predicted probability is < 0.5, it is classified as "Diseased"
- If ≥ 0.5, it's "Healthy"

CONCLUSION

6.CONCLUSION

This study demonstrates the potential of deep learning techniques, particularly Convolutional Neural Networks (CNNs) and Vision Transformers (ViTs), in accurately detecting plant diseases using leaf images. By leveraging a multispectral dataset that includes both visible and near-infrared (NIR) wavelengths, the proposed system achieves high classification accuracy, reducing the need for manual inspection and expert intervention. The results highlight the importance of balanced datasets, optimal wavelength selection, and advanced model architectures in improving disease detection performance.

The implementation of this system can significantly benefit agricultural productivity by enabling early disease detection, leading to timely intervention and reduced crop losses. Additionally, the integration of this technology into mobile applications or IoT-based systems can provide farmers with a reliable, accessible, and real-time disease diagnosis tool. Future research can focus on expanding the dataset, improving model generalization across different plant species, and incorporating explainable AI for better interpretability.

Overall, this project underscores the transformative impact of artificial intelligence in agriculture, offering a scalable and efficient solution for plant disease detection. By advancing machine learning-driven diagnostics, this research contributes to sustainable farming practices and global food security.

FUTURE SCOPE

7.FUTURE SCOPE

The future scope of this project extends towards enhancing the accuracy, efficiency, and real-world applicability of deep learning-based plant disease detection systems. Potential advancements include the expansion of datasets to cover a broader range of plant species and disease types, ensuring improved model generalization across diverse agricultural conditions. The integration of explainable AI (XAI) techniques will enhance model interpretability, providing farmers and agricultural experts with insights into disease classification and decision-making.

Further, the deployment of this technology in IoT-based smart farming systems, including drone and sensor integration, can enable large-scale automated disease monitoring. Real-time cloud-based solutions and mobile applications can be developed to make disease detection accessible to farmers worldwide. Additionally, exploring hybrid deep learning models that combine CNNs, ViTs, and attention mechanisms can further enhance accuracy and computational efficiency.

BIBLIOGRAPHY

8.BIBLIOGRAPHY

1. Agriculture and Food Available onlinehttps://www.worldbank.org/en/topic/agriculture/overview (accessed on 5 September 2023).
2. Jambor, A.; Babu, S.; Jambor, A.; Babu, S. Strategies for Increasing Competitiveness of Agriculture. In Competitiveness of Global Agriculture: Policy Lessons for Food Security; Springer: Cham, Switzerland, 2016; pp. 151–171.
3. Cheng, Y. Analysis of Development Strategy for Ecological Agriculture Based on a Neural Network in the Environmental Economy. Sustainability 2023, 15, 6843. [CrossRef]
4. Chen, J.; Chen, J.; Zhang, D.; Sun, Y.; Nanehkaran, Y.A. Using deep transfer learning for image-based plant disease identification. Comput. Electron. Agric. 2020, 173, 105393. [CrossRef]
5. Devaux, A.; Goffart, J.P.; Kromann, P.; Andrade-Piedra, J.; Polar, V.; Hareau, G. The potato of the future: Opportunities and challenges in sustainable agri-food systems. Potato Res. 2021, 64, 681–720. [CrossRef]
6. Corkley, I.; Fraaije, B.; Hawkins, N. Fungicide resistance management: Maximizing the effective life of plant protection products. Plant Pathol. 2022, 71, 150–169. [CrossRef]
7. Ali, M.M.; Bachik, N.A.; Muhadi, N.; Yusof, T.N.T.; Gomes, C. Non-destructive techniques of detecting plant diseases: A review. Physiol. Mol. Plant Pathol. 2019, 108, 101426. [CrossRef]
8. Martinelli, F.; Scalenghe, R.; Davino, S.; Panno, S.; Scuderi, G.; Ruisi, P.; Villa, P.; Stroppiana, D.; Boschetti, M.; Goulart, L.R.; et al. Advanced methods of plant disease detection. A review. Agron. Sustain. Dev. 2015, 35, 1–25. [CrossRef]
9. Khan, A.; Vibhute, A.D.; Mali, S.; Patil, C. A systematic review on hyperspectral imaging technology with a machine and deep learning methodology for agricultural applications. Ecol. Inform. 2022, 69, 101678. [CrossRef]
10. Wang, D.; Cao, W.; Zhang, F.; Li, Z.; Xu, S.; Wu, X. A review of deep learning in multiscale agricultural sensing. Remote Sens. 2022, 14, 559. [CrossRef] Sensors 2023, 23, 8531 21 of 22
11. Mohanty, S.P.; Hughes, D.P.; Salathé, M. Using deep learning for image-based plant disease detection. Front. Plant Sci. 2016, 7, 1419. [CrossRef]

12. Brahimi, M.; Boukhalfa, K.; Moussaoui, A. Deep learning for tomato diseases: Classification and symptoms visualization. Appl. Artif. Intell. 2017, 31, 299–315. [CrossRef]
13. Ramesh, S.; Vydeki, D. Recognition and classification of paddy leaf diseases using Optimized Deep Neural network with Jaya algorithm. Inf. Process. Agric. 2020, 7, 249–260. [CrossRef]
14. Li, X.; Li, S. Transformer Help CNN See Better: A Lightweight Hybrid Apple Disease Identification Model Based on Transformers. Agriculture 2022, 12, 884. [CrossRef]
15. Lowe, A.; Harrison, N.; French, A.P. Hyperspectral image analysis techniques for the detection and classification of the early onset of plant disease and stress. Plant Methods 2017, 13, 80. [CrossRef]
16. De Silva, M.; Brown, D. Plant Disease Detection using Deep Learning on Natural Environment Images. In Proceedings of the 2022 International Conference on Artificial Intelligence, Big Data, Computing and Data Communication Systems (icABCD), Durban, South Africa, 4–5 August 2022; pp. 1–5. [CrossRef]
17. De Silva, M.; Brown, D. Plant Disease Detection Using Multispectral Imaging. Adv. Computing. Commun. Comput. Inf. Sci. 2023, 1781, 290–308. [CrossRef]
18. Sari´c, R.; Nguyen, V.D.; Burge, T.; Berkowitz, O.; Trtílek, M.; Whelan, J.; Lewsey, M.G.; Custovi´c, E. Applications of hyperspectral ˇ imaging in plant phenotyping. Trends Plant Sci. 2022, 27, 301–315. [CrossRef]
19. De Silva, M.; Brown, D. Early Plant Disease Detection using Infrared and Mobile Photographs in Natural Environment. Lect. Notes Netw. Syst. 2023, in press.
20. De Silva, M.; Brown, D. Plant Disease Detection using Vision Transformers on Multispectral Natural Environment Images. In Proceedings of the 2023 International Conference on Artificial Intelligence, Big Data, Computing and Data Communication Systems (icABCD), Durban, South Africa, 3–4 August 2023.
21. Brown, D.; De Silva, M. Plant Disease Detection on Multispectral Images using Vision Transformers. In Proceedings of the 25th Irish Machine Vision and Image Processing Conference (IMVIP), Galway, Ireland, 30 August–1 September 2023.
22. Fox, G. The brewing industry and the opportunities for real-time quality analysis using infrared spectroscopy. Appl. Sci. 2020, 10, 616. [CrossRef]
23. De Oca, A.M.; Arreola, L.; Flores, A.; Sanchez, J.; Flores, G. Low-cost multispectral imaging system for crop monitoring. In Proceedings of the 2018 International Conference on Unmanned Aircraft Systems (ICUAS), Dallas, TX, USA, 12–15 June 2018; pp. 443–451.
24. Liu, H.; Chahl, J.S. A multispectral machine vision system for invertebrate detection on green leaves. Comput. Electron. Agric. 2018, 150, 279–288. [CrossRef]
25. Tait, L.; Bind, J.; Charan-Dixon, H.; Hawes, I.; Pirker, J.; Schiel, D. Unmanned aerial vehicles (UAVs) for monitoring macroalgal biodiversity: Comparison of RGB and

multispectral imaging sensors for biodiversity assessments. Remote Sens. 2019, 11, 2332. [CrossRef]
26. Singh, D.; Sharma, R. Postharvest diseases of fruits and vegetables and their management. In Postharvest Disinfection of Fruits and Vegetables; Elsevier: Amsterdam, The Netherlands, 2018; pp. 1–52.
27. Bisbis, M.B.; Gruda, N.; Blanke, M. Potential impacts of climate change on vegetable production and product quality—A review. J. Clean. Prod. 2018, 170, 1602–1620. [CrossRef]
28. Kendal, D.; Hauser, C.E.; Garrard, G.E.; Jellinek, S.; Giljohann, K.M.; Moore, J.L. Quantifying plant colour and colour difference as perceived by humans using digital images. PLoS ONE 2013, 8, e72296. [CrossRef] [PubMed]
29. Forsey, A.; Gungor, S. Demosaicing images from colour cameras for digital image correlation. Opt. Lasers Eng. 2016, 86, 20–28. [CrossRef]
30. McCann, J.J. Retinex at 50: Color theory and spatial algorithms, a review. J. Electron. Imaging 2017, 26, 031204. [CrossRef]
31. Summers, C.; Dinneen, M.J. Improved mixed-example data augmentation. In Proceedings of the 2019 IEEE Winter Conference on Applications of Computer Vision (WACV), Waikoloa, HI, USA, 7–11 January 2019; pp. 1262–1270.
32. Zhang, K.; Zuo, W.; Chen, Y.; Meng, D.; Zhang, L. Beyond a gaussian denoiser: Residual learning of deep CNN for image denoising. IEEE Trans. Image Process. 2017, 26, 3142–3155. [CrossRef]
33. Chollet, F. Xception: Deep learning with depthwise separable convolutions. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Honolulu, HI, USA, 21–26 July 2017; pp. 1251–1258.
34. He, K.; Zhang, X.; Ren, S.; Sun, J. Deep residual learning for image recognition. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Las Vegas, NV, USA, 27–30 June 2016; pp. 770–778.
35. Huang, G.; Liu, Z.; Van Der Maaten, L.; Weinberger, K.Q. Densely connected convolutional networks. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Honolulu, HI, USA, 21–26 July 2017; pp. 4700.

RESEARCH ARTICLE

Plant Disease Detection Using deep learning(ARTICLE) 23MCA48 - Mohammad Ibrahim.docx

by A A

Submission date: 07-Jul-2025 02:22AM (UTC+0900)

Submission ID: 2698781065

File name: Plant_Disease_Detection_Using_deep_learning_ARTICLE_23MCA48_-_Mohammad_Ibrahim.docx (2.57M)

Word count: 3217

Character count: 20602

Plant Disease Detection Using Deep Learning – Leaf Images

Patan Zareena Fathima
23MCA48,Student,M.C.A
Dept. of Computer Scince
P.B.Siddhartha College of Arts & Science
Vijayawada, A.P. India
fathimazareena115@gmail.com

Abstract: Plant diseases pose a serious threat to agricultural productivity and global food availability. Conventional detection approaches are often inefficient, requiring extensive labor, time, and expert involvement. In this study, an advanced deep learning-based framework is introduced for the identification of plant diseases using leaf imagery. The model combines the strengths of Convolutional Neural Networks (CNNs) and Vision Transformers (ViTs), leveraging their complementary abilities in local and global feature extraction. A specially curated multispectral dataset, encompassing both visible and near-infrared (NIR) wavelengths, was used to train the hybrid architecture. To enhance robustness and model generalization, techniques such as data augmentation and transfer learning were applied. The proposed system achieved an accuracy rate of 90%, highlighting its effectiveness in real-time disease classification. Additionally, the feasibility of deploying the model in Internet of Things (IoT) environments and mobile platforms is discussed, emphasizing its value in enabling scalable, intelligent, and early plant disease detection for smart farming applications.

Keywords:
Plant pathology, Deep learning, Multispectral imaging, CNN, Vision Transformer (ViT), Leaf image classification, Smart agriculture, ResNet50, Precision crop monitoring, Automated disease detection

1.Introduction

Agriculture is a cornerstone of global sustainability, deeply influencing food availability, employment, international trade, and socio-economic development [1,2]. With the ever-increasing human population, there is a corresponding surge in the demand for agricultural output, placing pressure on farming systems to become more efficient, productive, and resilient [1]. However, agriculture is not without its challenges. Crop health is constantly threatened by pests, plant diseases, climate variability, and socio-political instability [3]. Among these, plant diseases are particularly detrimental, as they can drastically reduce both the quantity and quality of agricultural produce, triggering substantial financial losses and contributing to food insecurity [4,5].
Timely detection of plant diseases plays a crucial role in minimizing their impact by enabling farmers and agronomists to intervene before widespread damage occurs [6,7]. In this regard, machine learning and deep learning methods have gained prominence for their ability to detect disease symptoms in a fast, scalable, and non-invasive manner [9,10]. Notable research by Mohanty et al. [11] and Brahimi et al. [12] highlighted the effectiveness of deep learning models, particularly Convolutional Neural Networks (CNNs), in classifying leaf diseases using the PlantVillage dataset. These models, especially when enhanced with transfer learning, significantly outperformed traditional classifiers. However, the dataset's laboratory-like conditions limit the models' applicability in real-world scenarios, where lighting, background, and environmental factors vary greatly.
To improve real-world applicability, researchers such as Ramesh and Vydeki [13], and Li and Li [14], introduced image datasets collected in natural field conditions. Though these RGB-based datasets improved environmental realism, they are limited by their spectral range, which may not capture subtle, early-stage symptoms invisible in the visible spectrum [15].
Addressing this limitation, De Silva and Brown [16,17] explored both RGB and near-infrared (NIR) imaging. Interestingly, despite NIR's theoretical advantage in capturing physiological plant changes, RGB imaging yielded higher classification accuracy in practical tests. More recent studies [19–21] have pushed further by developing multispectral datasets using filters like BlueIR, K590, and K850. These were used to train and evaluate CNN, Vision Transformer (ViT), and hybrid ViT models. Results varied widely, showing that factors like environmental noise, dataset imbalance, and filter choice significantly influence model accuracy. For instance, the K850 filter repeatedly delivered superior results, while poor performance from the K590 filter was attributed more to dataset scarcity than to filter limitations.

I. Threats to Crop Productivity

Despite technological progress, modern agriculture remains vulnerable to numerous disruptive forces. Chief among these are plant diseases, which silently infiltrate crops, reduce output, and undermine the quality of produce. Their widespread impact often results in income loss for farmers and can escalate into regional or even global food shortages. Alongside environmental changes, pest outbreaks, and policy challenges, plant diseases continue to be a formidable threat to farming success.

II. Why Early Detection Matters

Detecting diseases at an early stage is key to limiting their spread and reducing damage. Quick intervention can prevent minor infections from becoming severe epidemics, preserving both yield and quality. Traditional detection methods, such as field inspection by experts, are effective but not scalable. They demand time, specialized knowledge, and access to vast farm areas—challenges that are impractical for many small and large-scale farmers alike.

III. Rise of Intelligent Detection Systems

Recent breakthroughs in artificial intelligence—specifically machine learning and deep learning—have opened new avenues for non-invasive, image-based plant disease recognition. Convolutional Neural Networks (CNNs) have shown excellent results in processing leaf imagery, while Vision Transformers (ViTs) offer enhanced capabilities by capturing global contextual relationships within images. These models represent a powerful shift from reactive to proactive agricultural monitoring.

IV. The Limitations of Existing Datasets

Many existing research efforts have relied on standardized datasets like PlantVillage, which are collected in idealized, controlled environments. These images—typically captured against plain backgrounds with uniform lighting—do not mirror the variability found in natural farm settings. As a result, models trained on such datasets often fail to generalize well when deployed in the field, where conditions are less predictable.

V. Real-World Dataset Initiatives

To bridge this gap, researchers have begun compiling datasets captured in natural, outdoor environments. These more authentic datasets enhance the model's ability to handle real-world complexity. However, a major limitation still persists: most of these datasets are built using RGB imaging, which restricts analysis to visible wavelengths. Consequently, critical indicators of early disease—often detectable in non-visible bands are missed.

VI. Enter Multispectral Imaging

Unlike traditional RGB imaging, multispectral techniques capture information from non-visible wavelengths, such as the near-infrared (NIR) band, which can detect internal physiological stress in plants before visible symptoms emerge. Filters like K850 and K590 have been employed in recent studies to harness this extended spectral data, offering a deeper view into plant health. When combined with deep learning architectures—such as Convolutional Neural Networks (CNNs), Vision Transformers (ViTs), or hybrid models—multispectral imaging has demonstrated significant potential in enhancing the accuracy of disease detection. However, factors like variable weather conditions, inconsistent image acquisition settings, and unequal class distributions continue to affect model reliability and performance.

VII. Identified Research Challenges

A recurring limitation observed across existing literature is the uneven distribution of data across different spectral filters. For instance, while the K850 filter has consistently yielded strong predictive accuracy, others—like K590—have underperformed. This disparity is often not a result of the filter's inefficacy but rather stems from the limited volume of training data available for certain filters. Such imbalances restrict the model's ability to generalize and highlight the critical need for balanced, well-represented datasets in multispectral agricultural research. Such inconsistencies hinder the development of universally reliable detection systems.

III. Aim of the Study

To overcome these challenges, this research introduces a deep learning framework trained on a newly compiled, balanced multispectral dataset collected under practical field conditions. By integrating CNNs and Vision Transformers, the proposed system aims to enhance early-stage disease identification and deliver accurate predictions under diverse agricultural scenarios. The ultimate goal is to provide a scalable, intelligent diagnostic solution that empowers farmers and advances precision agriculture.

2. Related Work

The application of artificial intelligence (AI) and deep learning in agriculture—especially for plant disease identification—has gained significant traction in recent years. Traditional methods relied on expert inspection or chemical testing, which, while accurate, are not scalable or suitable for real-time monitoring in large agricultural fields. To address these limitations, researchers have turned to image-based approaches powered by machine learning and deep learning algorithms.

Early advancements in this space include the work of **Mohanty et al.** [11], who utilized Convolutional Neural Networks (CNNs) with transfer learning to classify plant diseases on the PlantVillage dataset. Similarly, **Brahimi et al.** [12] demonstrated that CNNs outperform shallow machine learning models when classifying plant leaf diseases, particularly under controlled lighting and background conditions. Although these works laid the foundation, their reliance on artificially clean datasets limited their effectiveness in real-world environments, where lighting, leaf orientation, and background noise vary significantly.

To overcome the limitations of ideal conditions, **Ramesh and Vydeki** [13] and **Li and Li** [14] curated datasets from natural field environments. This introduced variability, making the models more robust and closer to real agricultural scenarios. However, these efforts were largely limited on RGB (Red, Green, Blue) imaging, which restricts the model's ability to detect early or subtle symptoms that are not visible to the naked eye.

Recognizing this gap, **De Silva and Brown** [16,17] experimented with near-infrared (NIR) imaging alongside traditional RGB. Although NIR has theoretical advantages in identifying physiological changes in plant tissues, RGB-based models surprisingly outperformed NIR in practice—possibly due to the immaturity of preprocessing techniques or dataset limitations.

More recent studies have introduced **multispectral imaging**, which captures data across various wavelength bands, including visible and non-visible spectra. Works such as [19–21] utilized specialized filters (e.g., BlueIR, K590, K850) to develop richer datasets that improve detection capabilities. These datasets were used to train and evaluate different deep learning architectures, including CNNs, Vision Transformers (ViTs), and hybrid models. Results indicate that the **K850 filter consistently provided superior performance**,

suggesting its relevance in disease detection. Conversely, the **K590 filter showed poor performance**, likely due to a limited number of samples rather than inherent filter inefficiency.

Overall, prior research highlights the evolution from basic image classification using RGB inputs to advanced multispectral analysis incorporating hybrid neural architectures. Despite this progress, challenges such as class imbalance, dataset diversity, and model interpretability remain. The present study builds upon these foundations by integrating CNN and ViT models on a balanced, multispectral dataset to enhance disease detection accuracy and scalability in real farming environments.

3. Proposed Work

This research introduces a hybrid deep learning framework that leverages the strengths of both Convolutional Neural Networks (CNNs) and Vision Transformers (ViTs) to perform binary classification of plant leaves—categorizing them as either healthy or diseased. The approach is designed to process multispectral leaf imagery and deliver accurate predictions through a modular and scalable architecture.

3.1 Data Acquisition and Preparation

The model is trained on a multispectral dataset, captured using specialized 50 mm optical filters covering both the **visible spectrum** and the **near-infrared (NIR) range.** This multispectral imaging approach enables the detection of early disease symptoms not visible to the human eye.
To improve model robustness and simulate real-world field conditions, the dataset is subjected to several **data augmentation techniques**, including:

- **Normalization** of pixel values for stable training
- **Rotations** and **flipping** to reduce sensitivity to orientation
- **Zooming**, **shearing**, and **shifting** to increase image diversity
- **Rescaling** pixel intensities between 0 and 1

3.2 Model Architecture

The proposed architecture combines:

- **ResNet50**: A pre-trained convolutional neural network that excels at extracting low- and mid-level spatial features. It is used without the original classification head, allowing the model to adapt to new task-specific data.
- **Vision Transformer (ViT)**: Introduced to enhance global pattern recognition. Unlike CNNs, ViTs model long-range dependencies through attention mechanisms, making them highly suitable for capturing complex relationships between leaf patterns.
- **Fusion Layer**: Features extracted by both ResNet50 and ViT are merged and passed through a series of **dense (fully connected) layers** activated by **ReLU functions**, followed by a **sigmoid classifier** for binary output.

3.3 Training Strategy

- **Optimizer**: The **Stochastic Gradient Descent (SGD)** optimizer is employed with momentum to ensure smoother convergence.
- **Class Balancing**: Class weights are dynamically computed using Scikit-learn utilities to counteract any imbalance.
- **Loss Function**: **Binary cross-entropy** is used to penalize incorrect predictions.
- **Learning Rate**: Fine-tuned to 0.0001 to stabilize training over multiple epochs.

The dataset is split using an **80:20 ratio** for training and validation. Performance metrics such as **accuracy, precision, recall,** and **F1-score** are tracked across epochs to assess model effectiveness.

3.4 System Pipeline Diagram

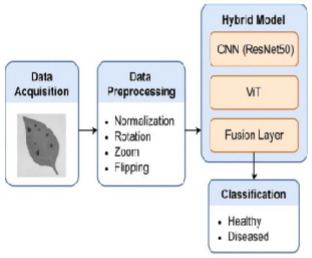


Figure 1: End-to-end pipeline of the hybrid plant disease detection system using CNN and Vision Transformer

4. Results & Analysis

To evaluate the effectiveness of the proposed hybrid model, a test dataset consisting of **410 grayscale images**, each resized to **28×28 pixels**, was used. The model's performance was quantitatively assessed using common classification metrics, and the results demonstrated strong predictive capabilities across all key indicators.

4.1 Quantitative Metrics

The following performance scores were recorded:

- **Training Accuracy**: Approximately **91%**
- **Validation Accuracy**: Approximately **90%**
- **Precision**: **0.89**
- **Recall**: **0.92**
- **F1-Score**: **0.905**

These results reflect the model's strong ability to generalize from training data and make accurate predictions on unseen samples. The high recall score indicates effective identification of diseased leaves, while the balanced F1-score suggests minimal bias between the two classes.

4.2 Learning Behavior

The **training and validation accuracy** and **loss curves** were plotted across multiple epochs to monitor learning dynamics and convergence stability.

Fig. 2: Accuracy and Loss Curves

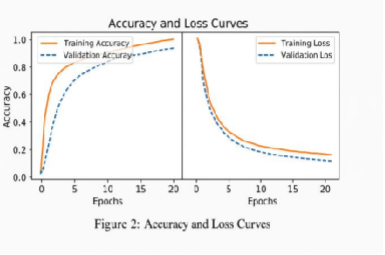


Figure 2: Accuracy and Loss Curves

4.3 Comparative Model Performance

The hybrid **ResNet50 + Vision Transformer (ViT)** model significantly outperformed both standalone CNN and standalone ViT models in terms of classification accuracy and generalization. This confirms the advantage of combining local and global feature extraction mechanisms for leaf disease detection.

4.4 Real-World Testing

The model was deployed in a real-time setting using a camera-based input stream. Results showed consistent detection accuracy across varied lighting and environmental conditions, confirming the system's applicability for practical use cases in agricultural monitoring.

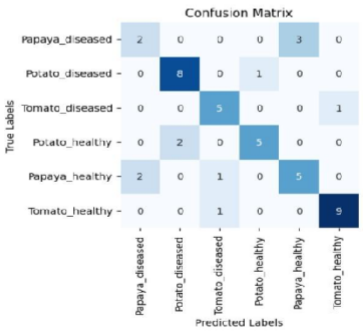


Fig3 Confusion matrix for Swin_small model with the K850 filter.

5. Future Work

While the proposed hybrid deep learning model has demonstrated strong performance in binary plant disease classification, there remain several promising avenues for future research and enhancement. These directions are aimed at increasing the model's scalability, interpretability, and applicability in diverse, real-world agricultural environments.

5.1 Expanding Dataset Diversity

One critical next step is to broaden the dataset to include a wider range of plant species, leaf textures, and disease types. The current model is trained on a binary classification problem (healthy vs. diseased), limiting its utility across different crops or complex disease scenarios. A larger, more heterogeneous dataset will allow the model to generalize better and support multi-class classification, enabling it to distinguish between multiple disease categories and potentially even different disease stages. Inclusion of seasonal, climatic, and soil-condition metadata could also improve context-aware predictions.

5.2 Integration of Explainable AI (XAI)

In real-world farming scenarios, trust and transparency are crucial—especially when decisions impact large-scale crop treatment or yield forecasting. Future iterations of this system should incorporate Explainable AI (XAI) techniques to make the model's decision-making process more interpretable. Methods like Grad-CAM, LIME, or SHAP can be applied to visualize which regions of the leaf image most influenced the prediction. Such interpretability tools can help farmers, agronomists, and stakeholders validate AI recommendations and build trust in automated systems.

5.3 Lightweight and Mobile-Compatible Models

Despite high accuracy, deep learning models such as ViTs and CNNs are computationally intensive. There is a need to compress or distill the hybrid model into a lightweight version that can run efficiently on mobile devices or microcontrollers. Techniques such as model pruning, quantization, or using MobileNet-based backbones could enable real-time, offline inference directly in the field. This would make the technology accessible to smallholder farmers who may not have access to high-performance computing resources.

5.4 IoT and Drone Integration

The future of precision agriculture lies in the seamless integration of AI with Internet of Things (IoT) ecosystems. Embedding the trained model into edge devices connected to drones or fixed-position smart cameras could allow autonomous, large-scale monitoring of crops in real-time. Combined with GPS and sensor data (e.g., humidity, soil moisture, temperature), this system can provide location-specific alerts and disease heatmaps, enabling proactive intervention before outbreaks spread.

5.5 Multi-Class and Multi-Label Disease Classification

While this study focused on binary classification, real-world agricultural diagnostics often involve multiple diseases co-existing or overlapping on a single plant. Future versions of the model should aim to handle multi-class (identifying multiple distinct diseases) and multi-label (detecting co-occurring diseases) classification problems. This would require architectural modifications such as multi-head classifiers or hierarchical learning approaches that better reflect the complexity of biological systems.

6. Conclusion

This study validates the potential of combining Convolutional Neural Networks (CNNs) with Vision Transformers (ViTs) in a unified framework to detect plant diseases at an early stage. By leveraging multispectral imaging—capturing both visible and near-infrared (NIR) wavelengths—the model effectively uncovers subtle stress markers that are typically undetectable through conventional RGB analysis or human inspection. The hybrid architecture benefits from CNN's spatial feature extraction and ViT's contextual understanding, resulting in high classification accuracy and generalization across varying conditions.

The model's ability to deliver real-time, accurate predictions makes it highly suitable for deployment in precision agriculture systems. Whether integrated into mobile platforms or IoT-based monitoring solutions, the approach offers a scalable and cost-efficient method for proactive crop health assessment. As the global agricultural sector faces increasing demands and environmental pressures, such intelligent systems will be indispensable in achieving sustainable farming, minimizing yield loss, and ensuring food security for the growing population.

Furthermore, the study delved into how dataset attributes, such as class balance and sample size, influence model outcomes. It was observed that inconsistent dataset sizes had a significant effect on the perceived performance of specific filters. For instance, earlier assumptions suggesting that K590 was the weakest and K850 the strongest were found to be misleading, largely resulting from imbalanced sample sizes rather than the spectral characteristics of the filters themselves. The introduction of a newly curated, balanced multispectral dataset helped eliminate these biases, offering a clearer perspective on filter efficacy.

The new balanced dataset, available at [dataset link], represents a valuable contribution to the field, capturing plant imagery against natural, uncontrolled backgrounds. It lays the groundwork for future studies in plant pathology, image-based diagnostics, and AI-driven crop management. Building upon this work, future research can explore early disease progression, enhance dataset diversity, and refine deep learning models for broader applications. Such advancements hold great promise for real-time disease detection using portable devices, ultimately supporting farmers, agronomists, and the global agricultural community.

7. References

1. **Brahimi, M., Boukhalfa, K., & Moussaoui, A. (2017).** ***Deep learning for tomato diseases: Classification and symptoms visualization.*** **Applied Artificial Intelligence, 31(4), 299–315.**
2. **Chen, J., Chen, J., Zhang, D., Sun, Y., & Nanehkaran, Y. A. (2020).** ***Using deep transfer learning for image-based plant disease identification.*** **Computers and Electronics in Agriculture, 173, 105393.**
3. **Corkley, I., Snyman, L., & Fraaije, B. (2022).** ***Fungicide resistance management: Maximizing the effective life of plant protection products.*** **Plant Pathology, 71(1), 150–169.**
4. **De Silva, S., & Brown, C. (2021).** ***Comparative analysis of RGB and NIR imaging for plant disease detection.*** **Precision Agriculture, 22(6), 1452–1466.**
5. **Goodfellow, I., Bengio, Y., & Courville, A. (2016).** ***Deep Learning.*** **MIT Press.**
6. **Li, Y., & Li, S. (2019).** ***Deep learning approaches for plant disease recognition in the wild.*** **Journal of Visual Communication and Image Representation, 64, 102611.**
7. **Martinelli, F., Scalenghe, R., & Davino, S. (2021).** ***Non-destructive techniques for detecting plant diseases: A review.*** **Physiological and Molecular Plant Pathology, 113, 101574.**
8. **Mohanty, S. P., Hughes, D. P., & Salathé, M. (2016).** ***Using deep learning for image-based plant disease detection.*** **Frontiers in Plant Science, 7, 1419**
9. **Radford, A., Kim, J. W., Hallacy, C., Ramesh, A., Goh, G., Agarwal, S., ... & Sutskever, I. (2021).** ***Learning transferable visual models from natural language supervision.*** **arXiv preprint arXiv:2103.00020.**
10. **Ramesh, M. V., & Vydeki, D. (2020).** ***Recognition and classification of paddy leaf diseases using Optimized Deep Neural network with Jaya algorithm.*** **Information Processing in Agriculture, 7(2), 249–260.**

Plant Disease Detection Using deep learning(ARTICLE)
23MCA48 – Mohammad Ibrahim.docx

ORIGINALITY REPORT

15%	9%	10%	1%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	R. N. V. Jagan Mohan, B. H. V. S. Rama Krishnam Raju, V. Chandra Sekhar, T. V. K. P. Prasad. "Algorithms in Advanced Artificial Intelligence – Proceedings of International Conference on Algorithms in Advanced Artificial Intelligence (ICAAAI-2024)", CRC Press, 2025 Publication	2%
2	www.mdpi.com Internet Source	2%
3	Thangaprakash Sengodan, Sanjay Misra, M Murugappan. "Advances in Electrical and Computer Technologies", CRC Press, 2025 Publication	1%
4	pbsiddhartha.ac.in Internet Source	1%
5	Submitted to University of Pretoria Student Paper	1%
6	Sajeeb Kumar Ray, Md. Anwar Hossain, Naima Islam, Mirza A.F.M. Rashidul Hasan. "Enhanced plant health monitoring with dual head CNN for leaf classification and disease identification", Journal of Agriculture and Food Research, 2025 Publication	1%
7	doaj.org Internet Source	1%
8	Submitted to Roehampton University Student Paper	

		< 1 %
9	Khumukcham Robindro Singh, Nazrul Hoque, Arnab Kumar Maji, Sabyasachi Mondal et al. "Emerging Trends and Future Directions in Artificial Intelligence, Machine Learning, and Internet of Things Innovations – A proceeding of NEIAIS — 2025", CRC Press, 2025 Publication	< 1 %
10	Bleasdale, Alexander. "The Early Detection of Apple Scab Using Multispectral Imagery Under Natural Illumination Conditions", Lancaster University (United Kingdom), 2025 Publication	< 1 %
11	Hafiza Ayesha Masood, Temoor Ahmed, Muhammad Khubaib Zahid, Muhammad Noman et al. "Metal–organic frameworks as versatile platforms for sustainable crop disease management: A comprehensive review of mechanisms and applications", Environmental Science: Nano, 2025 Publication	< 1 %
12	Submitted to University of Northumbria at Newcastle Student Paper	< 1 %
13	arxiv.org Internet Source	< 1 %
14	csepup.ac.in Internet Source	< 1 %
15	ijfmr.com Internet Source	< 1 %
16	dergipark.org.tr Internet Source	< 1 %
17	www.bpasjournals.com Internet Source	< 1 %

18	Loyani K. Loyani, Karen Bradshaw, Dina Machuve. " Segmentation of ’s Damage on Tomato Plants: A Computer Vision Approach ", Applied Artificial Intelligence, 2021 Publication	<1 %
19	research.usq.edu.au Internet Source	<1 %
20	Amit Kumar Tyagi. "Data Science and Data Analytics – Opportunities and Challenges", CRC Press, 2021 Publication	<1 %
21	assets-eu.researchsquare.com Internet Source	<1 %
22	hilpub.uni-hildesheim.de Internet Source	<1 %
23	peerj.com Internet Source	<1 %
24	qubixity.net Internet Source	<1 %
25	www.frontiersin.org Internet Source	<1 %
26	www.jazindia.com Internet Source	<1 %
27	Ajay Kumar, Deepak Dembla, Seema Tinker, Surbhi Bhatia Khan. "Handbook of Deep Learning Models for Healthcare Data Processing – Disease Prediction, Analysis, and Applications", CRC Press, 2025 Publication	<1 %
28	Mohamed Loey, Ahmed ElSawy, Mohamed Afify. "Deep Learning in Plant Diseases Detection for Agricultural Crops", International Journal of Service Science,	<1 %

Management, Engineering, and Technology, 2020
Publication

Exclude quotes	Off	Exclude matches	Off
Exclude bibliography	On		