PLANT-DISEASE DETECTION BY USING COMPUTER VISION APPROACH

CHANG MAN KIEN

A project report submitted in partial fulfilment of the requirements for the award of the degree of Bachelor of Technology (Honours) in Electronic Systems

Faculty of Engineering and Green Technology
Universiti Tunku Abdul Rahman

July 2023

DECLARATION

I hereby declare that this project report is based on my original work except for citations and quotations which have been duly acknowledged. I also declare that it has not been previously and concurrently submitted for any other degree or award at UTAR or other institutions.

Signature	:	___ [illegible] _____
Name	:	__CHANG MAN KIEN _____
ID No.	:	____ 19AGB03361_________
Date	:	_____19 SEP 2023_________

APPROVAL FOR SUBMISSION

I certify that this project report entitled "**PLANT-DISEASE DETECTION BY USING COMPUTER VISION APPROACH**" was prepared by CHANG MAN KIEN has met the required standard for submission in partial fulfilment of the requirements for the award of Bachelor of Engineering (Honours) in Electronic Systems at Universiti Tunku Abdul Rahman.

.

Approved by,

Signature : __________________________

Supervisor : Ts. Dr. Lee Han Kee

Date : __________________________

Specially dedicated to

My beloved father, mother and all my friends

ACKNOWLEDGEMENTS

First and foremost, I would like to express my deepest gratitude to my supervisor, Dr Lee Han Kee, for their unwavering support, invaluable guidance, and constructive feedback throughout the course of this research. Their expertise and insights have been instrumental in shaping this project, and their encouragement has been a constant source of motivation.

Additionally, I wish to extend my heartfelt appreciation to my peers and colleagues who have been a part of this journey. Their collaborative spirit, shared knowledge, and camaraderie have enriched this experience. Lastly, I am eternally grateful to my family for their unwavering belief in my capabilities, their endless patience, and their unconditional love. Their faith in me has been the bedrock upon which this research stands.

PLANT-DISEASE DETECTION BY USING COMPUTER VISION APPROACH

ABSTRACT

Plant maladies have long been a major concern in agriculture, frequently resulting in substantial yield losses, economic losses, and degraded crop quality. As the global demand for food security and sustainable agricultural practices increases, there is a pressing need for effective and precise disease detection mechanisms. Computer vision and deep learning provide promising avenues for the rapid and accurate identification of plant diseases. This study explores the feasibility of utilising pre-trained deep learning models, such as ResNet18, VGG16, AlexNet, and GoogleNet, to detect and classify a wide variety of plant diseases. Using a comprehensive dataset containing images of foliage exhibiting various disease symptoms, these models were trained, refined, and evaluated with extreme care. According to preliminary findings, GoogleNet outperforms its competitors in terms of accuracy and computational efficiency. While apple leaves serve as the study's primary case study, the methodologies and findings have broader implications. It paves the way for the development of real-time disease detection systems on the field, which could revolutionise the agricultural industry. Such systems could endow farmers around the world with the means to make informed decisions, optimize crop health, and ultimately increase food production.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF SYMBOLS / ABBREVIATIONS

AI	Artificial Intelligence
CNNs	Convolutional Neural Networks
ResNet18	Residual Network with 18 layers
VGG16	Visual Geometry Group Network with 16 layers
AlexNet	Neural Network developed by Alex Krizhevsky
GoogleNet	Another name for Inception, a deep convolutional neural network
PIL	Python Imaging Library
Flask	A micro web framework written in Python.
UTAR	Universiti Tunku Abdul Rahman
GPU	Graphics Processing Unit
RAM	Random Access Memory
CPU	Central Processing Unit

LIST OF APPENDICES

CHAPTER 1

INTRODUCTION

1.1 Background

Plant diseases have been a major concern for farmers and agriculturalists for centuries, resulting in significant crop productivity declines and economic losses. Historically, botanists and agricultural specialists were relied upon extensively for the detection and diagnosis of these diseases. On the basis of their observations, botanists and agricultural specialists would manually inspect crops, identify symptoms, and recommend treatments. This traditional procedure, while effective, is labour-intensive, time-consuming, and sometimes subjective, resulting in potential inaccuracies.

Rapid technological advancements, particularly in computer vision and deep learning, have ushered in a new era of plant disease detection. Convolutional Neural Networks (CNNs), a specialised type of neural network for processing data with a grid-like topology (such as an image), have demonstrated extraordinary performance in image recognition tasks. By training these models with extensive datasets containing images of healthy and diseased plants, the model can be honed to accurately detect a variety of plant diseases.

A study, for instance, used a pretrained ResNet34 model to detect numerous crop diseases with a 97.2% accuracy rate. Another study successfully diagnosed maladies in rice plants by combining the capabilities of Densely Connected Neural Network (DenseNet) and multilayer perceptron (MLP), with an accuracy of 97.68%. Once trained, these models can process thousands of images with consistent accuracy in a fraction of the time it would take a human expert.

The applications of these technologies in the actual world are extensive. Farmers can now diagnose plant diseases using mobile applications integrated with these trained models. The application can provide immediate feedback on a plant's health status, potential diseases, and recommended remedies when a photograph of a leaf is taken. This immediate feedback loop enables farmers to act rapidly, potentially preserving entire crops.

Moreover, computer vision's applications in agriculture extend beyond disease detection. It consists of insect detection, crop yield forecasting, soil health analysis, and even automated harvesting. This combination of artificial intelligence and conventional agricultural techniques is often referred to as 'precision agriculture.' It represents a significant advance in making agriculture more efficient, sustainable, and resistant to obstacles.

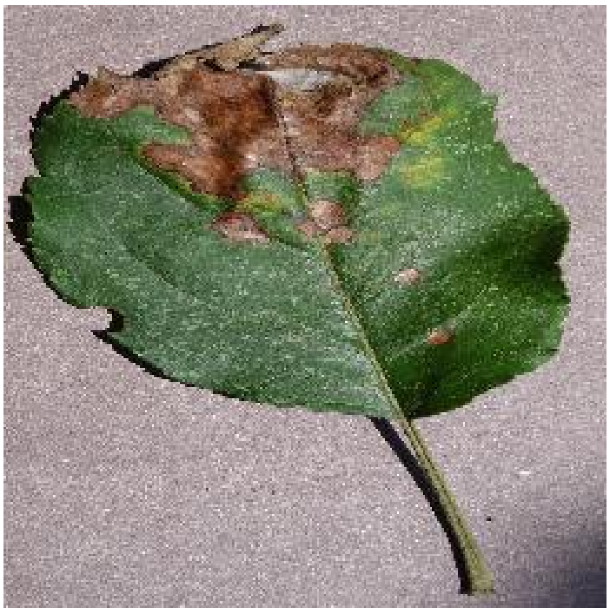


Figure 1: **A diseased leaf identified by computer vision.**

1.2 Problem Statements

Agriculture, as the foundation of numerous economies, performs a crucial role in ensuring food security and sustainability. However, the agricultural sector faces persistent threats from a variety of threats, with plant diseases emerging as one of the most formidable adversaries. These diseases, which are caused by pathogens such as fungi, bacteria, and viruses, can inflict havoc on crops and result in substantial yield losses. The repercussions of these diseases are not just limited to decreased production; it also correlates to substantial economic losses for farmers and can contribute to food shortages in affected regions.

Historically, agricultural specialists and botanists have played a significant role in the detection and diagnosis of plant diseases. These professionals would traverse fields, painstakingly inspecting crops, identifying symptoms, and recommending treatments based on their observations. This method has some advantages, but it is also fraught with difficulties. Inspections performed manually are time-consuming, labour-intensive, and subjective. The possibility of human error, coupled with the delay in detection, can result in the disease spreading unchecked and affecting larger swaths of crops.

In the modern era, with rapid technological advancements and a growing global population, the demand for sustenance is skyrocketing. This escalating demand emphasises the need for innovative, efficient, and scalable solutions for plant disease detection. The challenge is multifaceted: how to create a system that not only diagnoses plant diseases rapidly and accurately, but is also accessible to producers worldwide, regardless of their technological expertise or resources? How to guarantee early detection in order to prevent widespread crop devastation?

This project's overarching goal is encapsulated by its title: "Plant Disease Detection Using a Computer Vision Approach." Despite the fact that the immediate focus may be on apple leaves, the larger objective is unmistakable. The agricultural sector requires an all-encompassing, scalable, and technologically advanced method for detecting plant diseases across a vast array of species. By leveraging the capabilities of computer vision and deep learning, this project aims to provide a revolutionary solution for plant disease detection and management in the modern agricultural landscape.

1.3 Aims and Objectives

The objectives of the thesis are shown as following:

i) To develop an efficient and accurate system for detecting plant diseases using a computer vision approach.

ii) To provide a scalable and adaptable solution that caters to a wide range of plants and diseases, ensuring broad applicability in diverse agricultural settings.

iii) To set a benchmark in plant health diagnostics, paving the way for future research and innovations in the domain of agricultural technology.

1.4 Organization of the Thesis

Table 1: Organisational of the Thesis for Plant-disease detection project.

Chapter	Description
Chapter 1: Introduction	● Introduces the context, significance, and challenges of plant disease detection. Presents the aim and objectives of the study.
Chapter 2: Literature Review	● Provides a comprehensive review of existing methodologies, technologies, and research in plant disease detection and computer vision applications in agriculture.
Chapter 3: Methodology	● Describes the methods employed in the study, including data collection, preprocessing, model selection, training, validation, and system integration.
Chapter 4: System Design and Implementation	● Details the design considerations and the implementation process of the disease detection system. Discusses the choice of algorithms, tools, and platforms used.

Chapter 5: Results and Discussion	● Presents the results obtained from the trained models. Discusses the implications, accuracy, and efficiency of the system in real-world scenarios.
Chapter 6: Conclusion and Future Work	● Summarises the findings of the study. Discusses the contributions, limitations, and recommendations for future research and improvements.

CHAPTER 2

LITERATURE REVIEW

2.1 Overview of Plant Diseases

Like all living things, plants are susceptible to a variety of maladies. These diseases can be caused by a variety of organisms, including fungi, bacteria, viruses, and even insects. Apple trees, a ubiquitous staple in orchards, are not immune to this vulnerability. Apple trees are susceptible to several maladies that not only threaten their health but also reduce their yield.

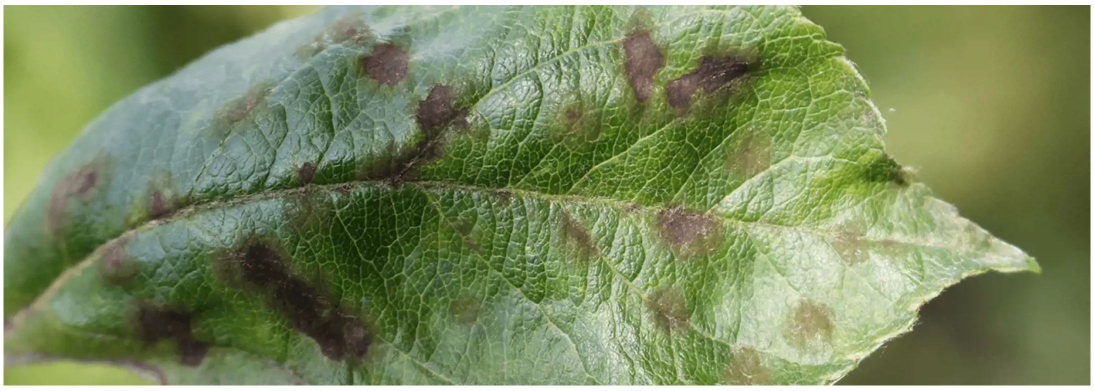


Figure 2: The indication of Apple Scab disease (MacHardy, 1996)

One of the predominant maladies afflicting apple trees globally is Apple Scab, caused by the fungus Venturia inadequacies. Affected trees have predominantly dark, scaly lesions on their leaves, but also on their fruits and twigs. Apple scab can significantly reduce fruit yield and quality if left unchecked (MacHardy, 1996).

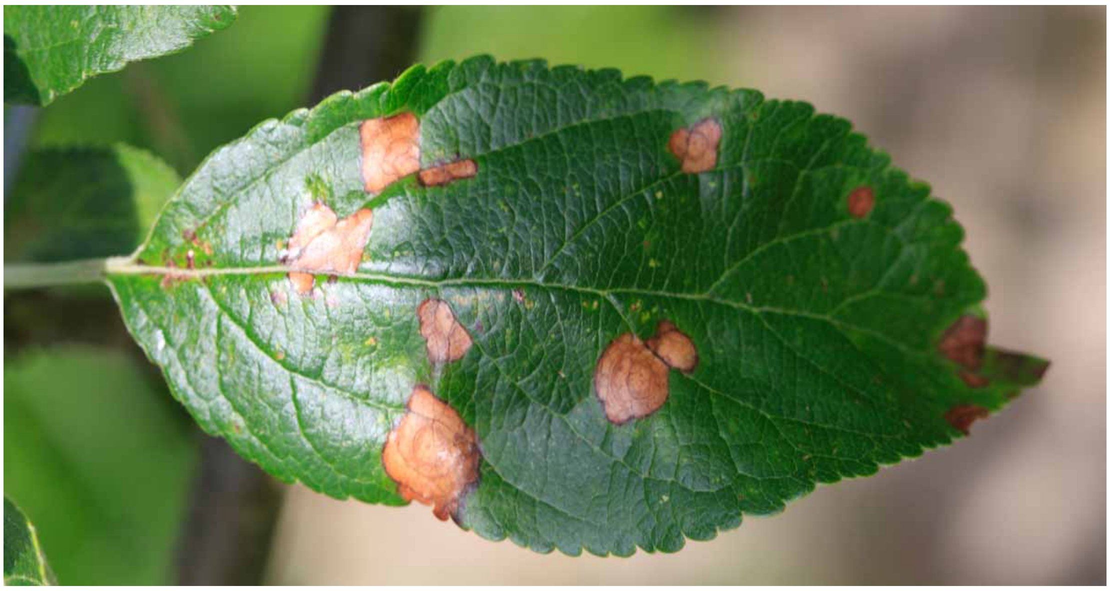


Figure 3: The indication of Black Rot disease (Sutton, 1990)

Black Rot, caused by the fungus Botryosphaeria obtusa, is yet another formidable adversary of apple trees. Initial symptoms appear on fruits as inconspicuous dark patches. As the disease progresses, however, these blotches can spread across the entire apple, rendering it inedible. In addition, the disease can affect the tree's leaves and branches, compromising its overall health (Sutton, 1990). Above Figure 3 shows the indication of Black Rot disease.

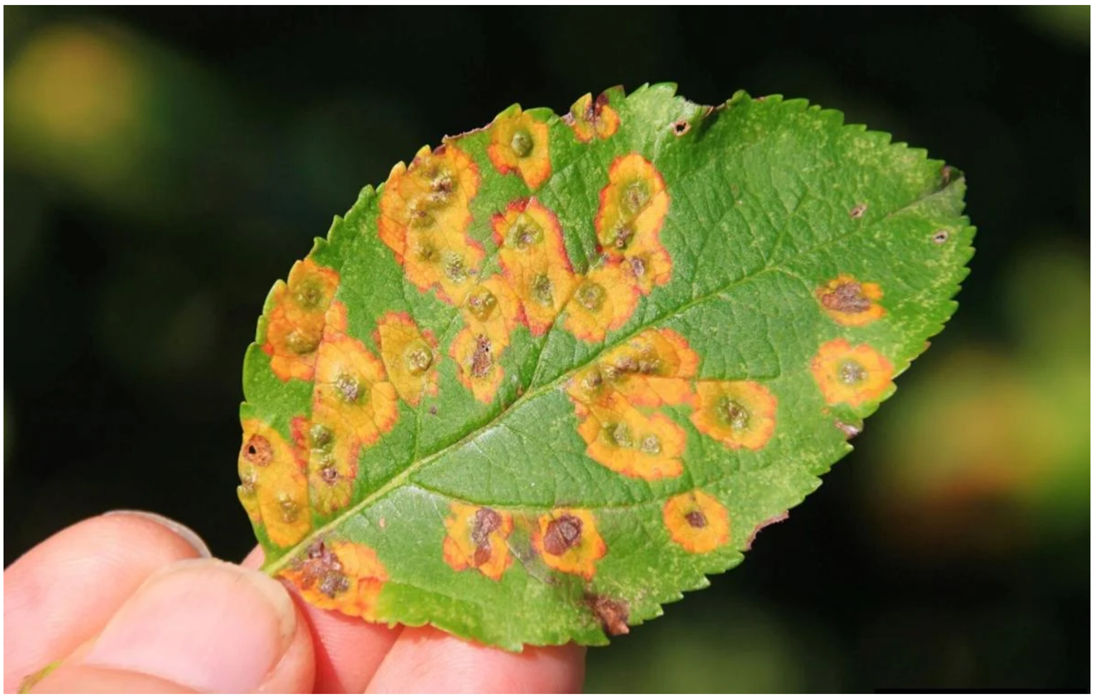


Figure 4: The indication of Cedar Apple Rust disease (Anagnostakis, 1987)

Cedar Apple Rust, a peculiar disease caused by the rust fungus Gymnosporangium juniperi-virginianae, requires two hosts to complete its life cycle: apple trees and junipers. On apple trees, the disease manifests as striking brilliant orange lesions on the leaves, frequently resulting in premature leaf drop and thereby impairing the tree's photosynthetic capacity (Anagnostakis, 1987). A truly robust apple tree displays no outward symptoms of disease. It consistently produces a bountiful harvest of wholesome fruits and possesses lush leaves and sturdy branches. In order to ensure the long-term health of apple trees, it is essential to combine rigorous surveillance with preventative and curative measures (Jones & Aldwinckle, 1990). Above Figure 4 shows the indication of Cedar Apple Rust disease.

2.2 Traditional Methods of Disease Detection

Agriculture's history is intricately intertwined with the evolution of plant disease detection and control. Before the advent of modern technology, farmers and horticulturists relied on traditional methods to detect and combat plant diseases for centuries. Although rudimentary in comparison to today's sophisticated techniques, these practices were the foundation of agriculture, assuring the health and productivity of crops.

Visual inspection was one of the primary and most straightforward methods. Farmers, armed with keen vision and years of experience, would stroll through their fields, meticulously inspecting plants for irregularities. Disease symptoms, such as discoloured leaves, wilting, abnormal growth patterns, and the presence of parasites, indicate that a plant is under stress (Smith, 1977). Although time-consuming, this hands-on approach was essential for early detection, allowing for prompt corrective action.

In the past, physical barriers were also utilised to prevent the spread of airborne diseases. To reduce the speed and range of disease-carrying winds, farmers would strategically plant windbreaks, such as rows of tall trees or shrubs. In addition, protective coverings or improvised canopies were utilised to shield crops from precipitation, which may have facilitated the spread of certain pathogens (Oerke & Dehne, 2004).

The practice of crop rotation was rooted in both disease prevention and soil health. Farmers could prevent the buildup of specific pathogens in the soil by planting various crops sequentially over multiple seasons. This method was notably effective against soil-borne diseases, preventing the overabundance of pathogens specific to a single crop (Campbell & Madden, 1990).

Another cornerstone of traditional disease management was plant breeding. Farmers would select and cultivate plant varieties with natural resistance to prevalent maladies over generations. This early form of genetic modification ensured that cereals possessed innate resistance to particular pathogens (Allard, 1960).

Utilising nature's own checks and balances, biological controls were also essential to disease management. Beneficial insects, such as ladybirds, were introduced to fields to control pests known to transmit disease. In a similar manner, beneficial fungi and bacteria were introduced to the soil to combat their pathogenic counterparts (Hajek & Eilenberg, 2018).

In addition to these tangible techniques, the intangible knowledge handed down through the generations was priceless. Frequently, indigenous knowledge, folklore, and local customs provided the key to identifying and treating plant maladies with locally available resources. This combination of observation, experience, and custom created a holistic approach to agriculture that was both sustainable and in harmony with nature (Warren, 1991).

Even though botanists or agriculturalists rely significantly on scientific advances and technology in the modern era, understanding and respecting these traditional techniques is essential. It provides a window into sustainable and holistic agricultural practices, emphasising the significance of a balanced approach that combines the past's wisdom with the present's innovations.

2.3 Role of Computer Vision in Agriculture

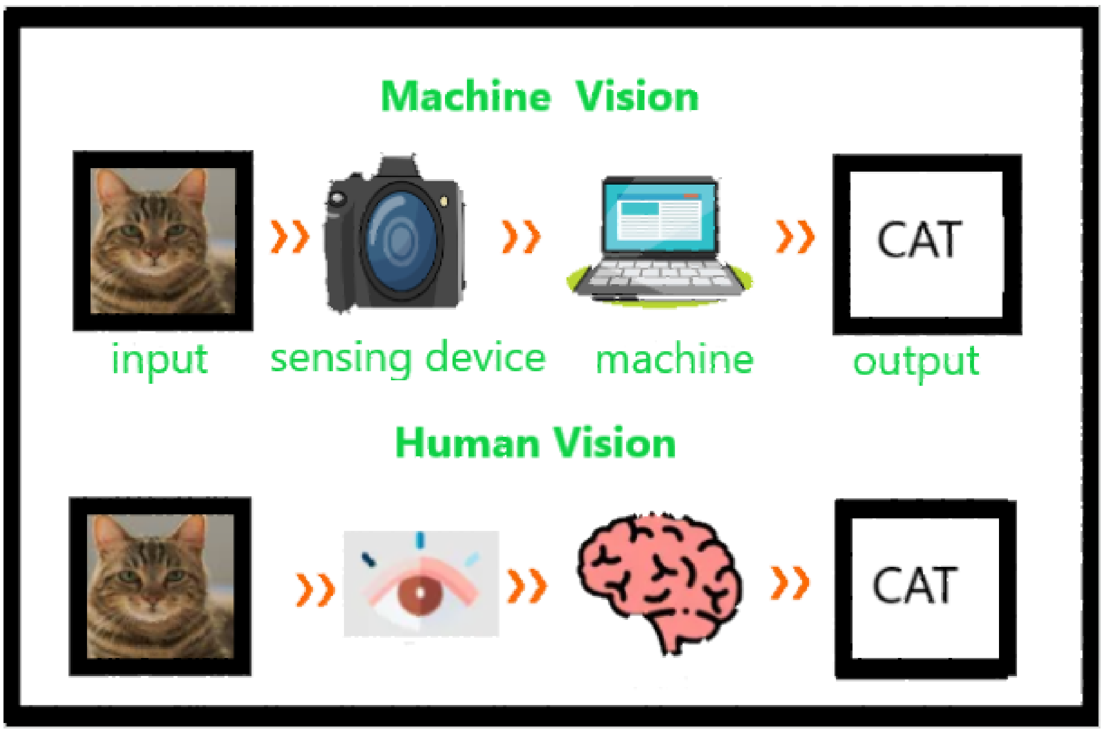


Figure 5: The brief outcome of Computer Vision (Senthilnath et al., 2016)

As shown in the above Figure 5, The incorporation of computer vision into agriculture has marked a significant turning point in the way farming operations are conducted. A prominent subfield of artificial intelligence (AI), computer vision concentrates on enabling machines to interpret and act upon visual data, simulating the capabilities of the human visual system. This technology has been a game-changer in agriculture, providing solutions to a multitude of problems spanning from crop health monitoring to the automation of complex tasks.

One of the most significant applications of computer vision in agriculture is precision cultivation. This method of modern agriculture employs technology to monitor and manage crops at a granular level, assuring their optimal health and yield. Using drones equipped with high-resolution cameras, farmers can capture aerial images of their fields that are rich in detail. Senthilnath et al. (2016) state that sophisticated algorithms are then applied to these images to identify areas that may be experiencing stress, pest infestations, or nutrient deficiencies. These precise insights allow producers to take immediate, targeted actions, such as applying water, pesticides, or fertilisers to specific areas, thereby optimising resource use and minimising environmental damage.

Another area where computer vision has had a revolutionary impact is disease detection. Computer vision is a more efficient alternative to the time-consuming manual inspections required by conventional methods. High-definition cameras can scan plants meticulously, detecting minute changes in colour, texture, and shape that may indicate the advent of a disease. By training machine learning models on large datasets, these systems can identify specific pathogens with remarkable precision, often well before it manifests visibly (Barbedo, 2018). This type of proactive detection permits early interventions, which may prevent widespread crop damage.

Computer vision has also enabled revolutionary advancements in the field of automated harvesting. Advanced robots, equipped with specialised cameras and AI-driven algorithms, can discern when fruits or vegetables have attained the ideal ripeness. Then, these machines can carefully harvest the fruits and vegetables, which is essential for preserving post-harvest quality. This level of automation proves invaluable for delicate commodities like strawberries and tomatoes (Sa et al., 2017).

In addition, applications of computer vision include soil analysis, yield prediction, livestock monitoring, and greenhouse automation. For example, sophisticated camera systems can analyse soil samples to determine their texture and nutrient content. These insights help farmers make informed judgements regarding crop selection and fertilisation techniques (Mehrabi et al., 2020). In livestock farming, computer vision technologies monitor animal health, behaviour, and growth patterns to ensure optimal conditions for animal welfare.

In short, the incorporation of computer vision into agriculture is reshaping the industry, propelling it towards greater efficacy, sustainability, and resiliency. As technological advancements continue, it is anticipated that the role of computer vision in agriculture will continue to expand, addressing new challenges and enhancing global food security.

2.4 Deep Learning: An Overview

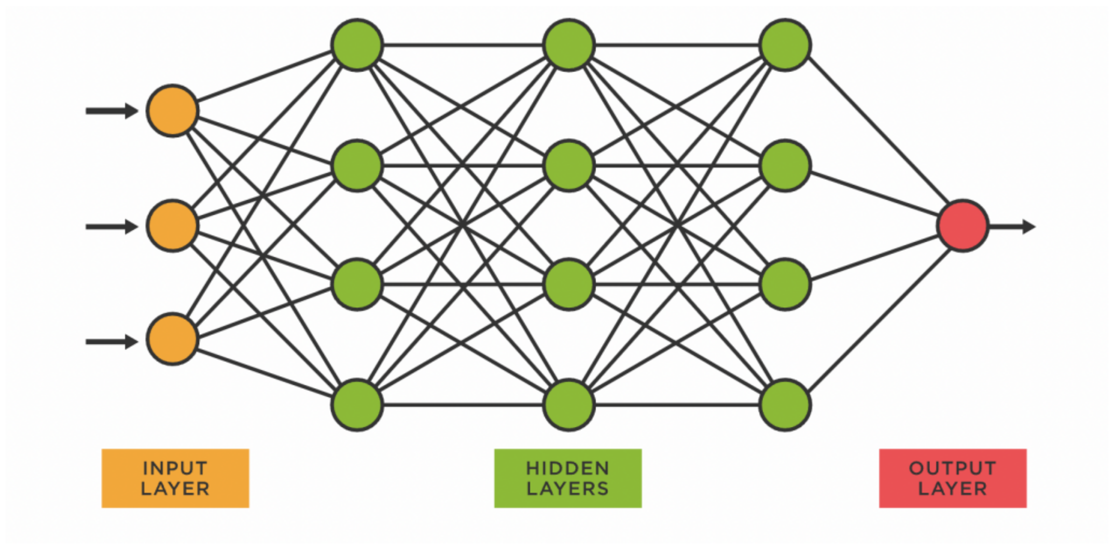


Figure 6: Architecture of a Deep Neural Network (LeCun et al., 2015)

Deep learning, a prominent subset of machine learning, is at the vanguard of the revolution in artificial intelligence (AI). This computational method, which was inspired by the intricate workings of the human brain, specifically neural networks, is adept at processing large datasets and identifying intricate patterns within them (LeCun et al., 2015). Deep learning employs multiple layers of artificial neural networks to perform a variety of tasks, from image and speech recognition to complex natural language processing. The architecture of deep neural network is shown in above Figure 6.

The term "deep" in "deep learning" refers to the depth or number of neural networks taking in the neural networks. Deep neural networks can have up to 150 hidden layers, whereas traditional neural networks may have only two or three (He et al., 2016). Each of these layers meticulously analyses the input data, identifies relevant features, and transmits the refined data to the following layer. This stratified, hierarchical approach to feature extraction enables deep learning models to recognise and comprehend complex patterns, thereby improving their predictive or classification accuracy.

Ability of deep learning to manage and process unstructured data types, such as images, audio clips, and textual content, is a significant advantage. This is accomplished without the need for laborious manual feature extraction (Schmidhuber, 2015). In specialised fields such as medical imaging, this advantage is palpably advantageous. In this context, deep learning models can detect anomalies in diagnostic images such as X-rays and MRI scans that are too subtle or subtle for human professionals to detect.

Concurrent rise of big data and the remarkable increase in computational prowess, particularly with the introduction of Graphics Processing Units (GPUs), have significantly aided the ascent of deep learning (Krizhevsky et al., 2012). GPUs, which were originally conceived for graphic depiction in video games, excel at matrix operations, which are fundamental to deep learning algorithms.

However, the path to deep learning is not without obstacles. For the training phase of deep learning models, a large amount of labelled data is required. Furthermore, this training can be both time-consuming and computationally intensive. A significant difficulty with complex deep learning models is their "black box" nature, which renders their decision-making processes obscure and difficult to decipher (Castelvecchi, 2016).

Deep learning's potential in the agricultural sector is already being realised. Deep learning will revolutionise the future of agriculture by predicting agricultural yields, detecting plant diseases, and pioneering innovations in precision farming. As technological progress continues to accelerate, the combination of deep learning with diverse industries promises a future rich in innovations and enhanced capabilities.

2.4.1 Convolutional Neural Networks (CNNs)

Convolutional Neural Networks (CNNs) have emerged as a pivotal architecture in the deep learning domain, particularly when handling visual data. CNNs are based on the concept that images can be processed in layers, with each layer learning an increasingly complex representation of the data. CNNs rely on the convolution operation, in which a filter or kernel traverses the input data, such as an image, to generate a feature map. This operation enables CNNs to initially focus on local features and then combine them in deeper layers to recognise larger structures.

Deeper into the architecture of CNNs, the convolutional layer, where the convolution operation is implemented, is the foundational layer. Each neuron here is connected to a small input region, and the presence of multiple such layers enables the network to distinguish between basic edges and intricate patterns. This is followed by the pooling layer, which reduces the spatial dimensions of the data, thereby expediting computations and bolstering the network's robustness. This layer's max-pooling procedure, which selects the maximum value from a set of values, is prevalent. The architecture culminates in the fully connected layer, which is a conventional multi-layer perceptron with a SoftMax activation function in the output layer, which converts high-level features from antecedent layers into the final class score.

CNNs are distinguished by their Rectified Linear Unit (ReLU) activation function, which introduces nonlinearity to the model. This nonlinearity is crucial because it enables the network to learn from its mistakes and adapt, a requirement for recognising complex patterns. To prevent overfitting in expansive networks, CNNs frequently employ a regularisation technique known as dropout. Here, random neurons are temporarily omitted from the network during training to ensure its robustness. In addition, batch normalisation is employed to normalise the activations of neurons, resulting in accelerated training and decreased sensitivity to initialization.

One of the defining characteristics of CNNs is their translational invariance. Once a feature is identified in one segment of an image, it can be identified in any other segment, thereby reducing the amount of training data required and enhancing the network's resilience to object positioning in images. Modern CNN architectures, including AlexNet, VGG, ResNet, and Inception, have set new benchmarks for a variety of visual recognition tasks. While these architectures differ in complexity and number of parameters, it will all utilise the fundamental principles of CNNs. Transfer learning is a powerful technique in the CNN arsenal, where models pre-trained on large datasets, such as ImageNet, are refined on smaller datasets, yielding remarkable results even with limited data. Figure 7 shows the anatomy of a Convolutional Neural Network.

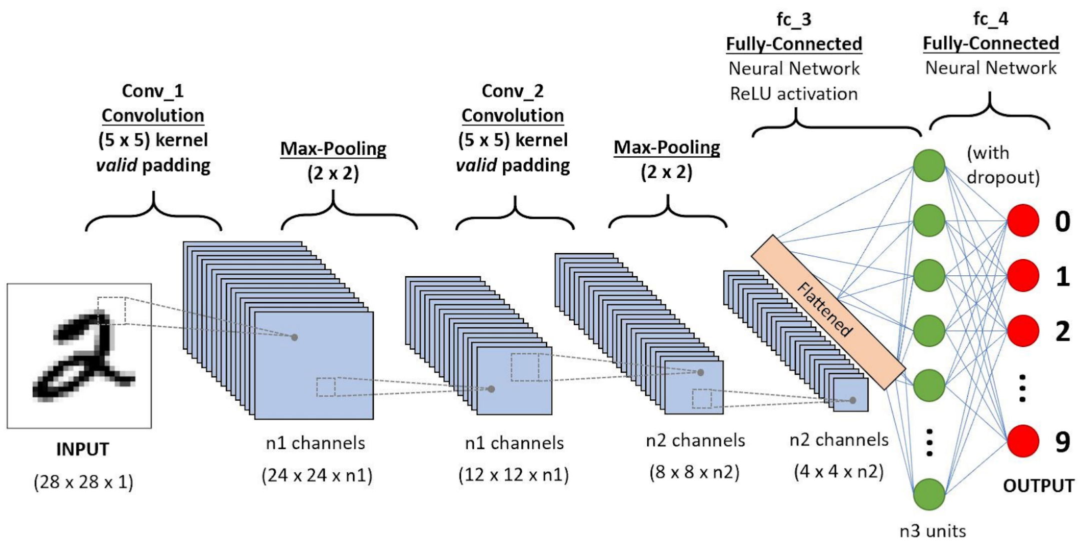


Figure 7: Anatomy of a Convolutional Neural Network (He et al., 2016)

2.4.2 Transfer Learning: Benefits and Use Cases

In the swiftly evolving field of deep learning, training models from scratch can be both time-consuming and resource intensive. This is especially true when the quantity or scope of the available dataset is limited. Transfer learning arises as a crucial strategy in such circumstances. It entails adapting a pre-trained model, which has typically been trained on a large and diverse dataset, to a new, related task that may have less data. Transfer learning provides a shortcut to attaining commendable performance by building on the knowledge acquired during the initial training phase.

Transfer learning's inherent effectiveness is one of its major benefits. Initial layers of a neural network are intended to capture generic characteristics such as edges, textures, and colours. These characteristics are frequently shared by a variety of duties. Consequently, these layers can be utilised or repurposed for various purposes. It's the latter layers, which tend to be more task-specific, that usually require fine-tuning. This method not only expedites the training process but also reduces the quantity of data required. This is especially advantageous because it reduces the risk of overfitting, especially when working with smaller, more specialised data sets.

The conservation of computational resources is a further noteworthy advantage. The process of training complex deep neural networks is both computationally and memory intensive. Such training typically requires specialised hardware configurations, such as potent GPUs. Transfer learning enables organisations and researchers to access and utilise cutting-edge models without incurring the extensive computational costs and time investments required to train them from scratch.

Transfer learning's applications encompass a vast array of domains. In the field of medical imaging, for instance, where acquiring labelled data can be a significant obstacle, transfer learning techniques have been instrumental in detecting a variety of anomalies and diseases. Similarly, in the field of natural language processing, models that have been pre-trained on large text corpora are optimised for specific tasks, such as sentiment analysis, translation, and text summarization. Transfer learning plays a role in the development of autonomous vehicles, enabling models trained in one operating environment to function optimally in another.

Nonetheless, it is crucial to approach transfer learning with a critical eye. The degree of similarity between the source task (from which the model was pre-trained) and the target task determines its effectiveness. If the two tasks are extremely dissimilar, transfer learning may not produce the desired results. In certain circumstances, it may even hinder performance. Below Figure 8 shows the Conceptual Framework of Transfer Learning.

In essence, transfer learning exemplifies the concept that knowledge acquired in one domain can be seamlessly and effectively applied to another in artificial intelligence. As the complexity of datasets and models continues to grow, the strategic significance and appeal of transfer learning are set to rise, offering a pragmatic approach to diverse computational challenges.

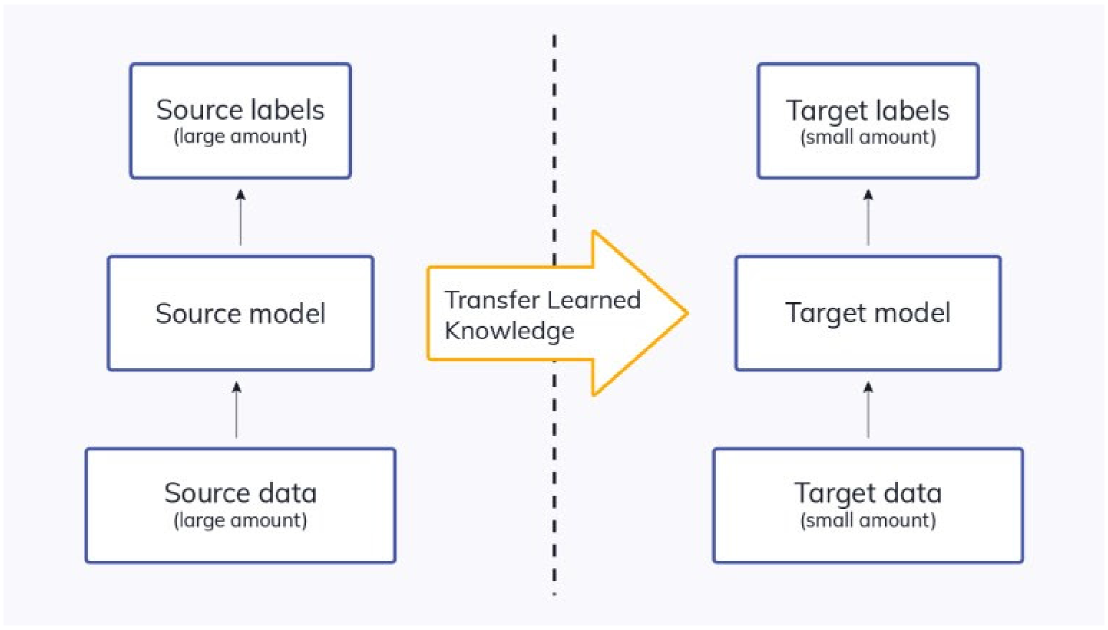


Figure 8: Conceptual Framework of Transfer Learning

2.4.3 Previous Works on Plant Disease Detection by AI.

The intersection of Artificial Intelligence (AI) and agriculture has spawned revolutionary opportunities, particularly in the field of plant disease detection. Because plant diseases have the potential to substantially reduce crop yields, which can have a domino effect on food supply chains and global food security, it is urgent to address them effectively. Historically, botanists and pathologists were extensively relied upon for the detection and diagnosis of plant diseases. These specialists would conduct meticulous visual inspections, frequently resorting to microscopic or chemical tests to determine the presence of pathogens. Despite their efficacy, these techniques were labour-intensive, time-consuming, and at times constrained by the subjectivity of human interpretation.

The advent of AI, particularly deep learning techniques, predicted a paradigm shift in this scenario. These techniques provided automation, scalability, and a high level of precision, making them viable alternatives to conventional methods. Numerous studies and experiments have demonstrated the effectiveness of AI in detecting and diagnosing plant diseases over the years. Mohanty et al. (2016), who presented the "PlantVillage" dataset, as shown in Figure 9, published one of the seminal works in this field. This dataset included over 50,000 images of both diseased and healthy plant foliage, 14 crop species, and 26 distinct diseases. Since then, it has become a central resource for researchers worldwide, facilitating the training and validation of numerous AI models. Utilising a deep convolutional neural network on this dataset, Mohanty et al. reported a 99.35% accuracy rate, establishing a standard for future research.

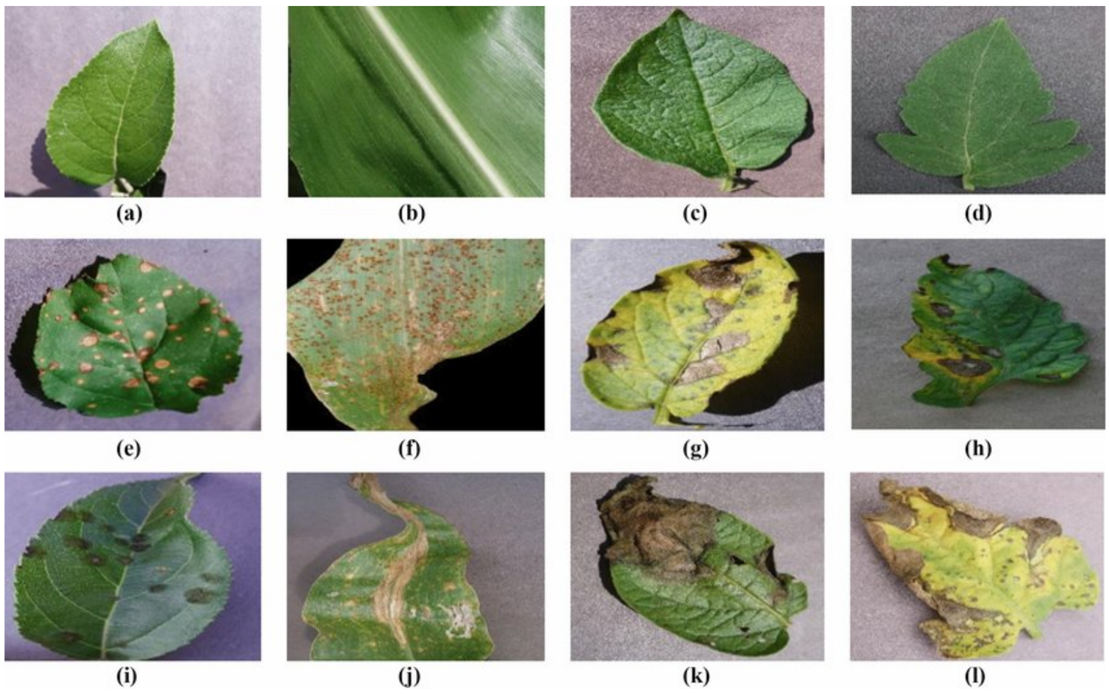


Figure 9: Sample Images from the PlantVillage Dataset: A Visual Representation of Healthy and Diseased Plant Leaves (Mohanty et al., 2016)

Ferentinos (2018) built upon this foundation by utilising deep convolutional neural networks for tomato plants. The research demonstrated the usefulness of transfer learning, a technique in which models initially trained on enormous datasets such as ImageNet are then fine-tuned for specific tasks, in this case tomato disease detection. In a comprehensive review, Barbedo (2018) examined the challenges and opportunities presented by artificial intelligence in plant disease detection. The study emphasised the importance of robust and diverse datasets, cautioned against the dangers of over-reliance on AI, and advocated for the incorporation of domain-specific knowledge to improve the accuracy of AI models.

The combination of AI with other emergent technologies has also garnered considerable attention. Integrating AI-driven disease detection systems with drones or IoT devices, for instance, offers the potential for large-scale, real-time monitoring of agricultural lands. Such integrations can facilitate the early detection of disease symptoms, enabling producers to take prompt preventive or corrective action. As the field continues to develop, it is evident that the combination of AI and plant disease detection is not merely a passing fad, but a transformative force with the potential to improve agricultural practices.

CHAPTER 3

METHODOLOGY

3.1 Data Collection

3.1.1 Sources of Data

Data plays an essential function in agricultural research and plant disease detection. The effectiveness of any machine learning or deep learning model is determined by the data's quality and quantity. For this investigation, the renowned PlantVillage dataset was the primary data source.

PlantVillage is a comprehensive dataset that has significantly advanced plant disease detection research. It contains a vast collection of images representing a variety of plant diseases, making it an invaluable resource for researchers who wish to train deep learning models for disease identification. The significance of such datasets cannot be understated, particularly when considering the global challenges posed by plant diseases. According to the Food and Agriculture Organisation of the United Nations, the food supply must be increased by 70 percent by 2050 in order to feed the world's population. Nevertheless, nearly a third of all food is lost due to plant diseases and disorders (Negi, 2021).

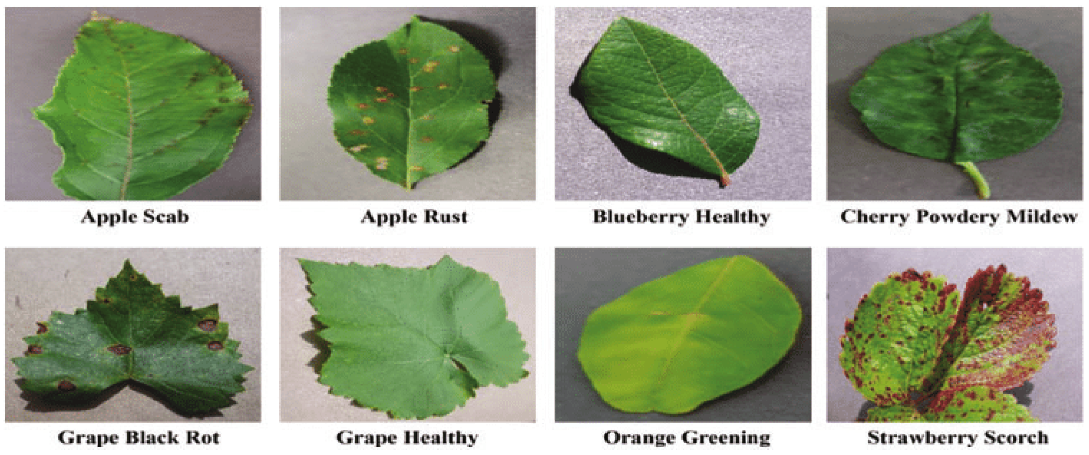


Figure 10: Sample images from the PlantVillage dataset showcasing the diversity of plant diseases and conditions. (Mohanty et al., 2016)

The PlantVillage dataset is distinguished by its diversity and abundance. PlantVillage provides a variety of data, as opposed to the majority of datasets that were collected in a laboratory setting. It contains laboratory images with uniform backgrounds as well as images that reflect real-world conditions, making it more representative of the challenges encountered in actual agricultural settings (Albattah et al., 202). The sample images from the PlantVillage dataset showcasing the diversity of plant diseases and conditions are shown in Figure 10.

Moupojou et al. (2023) emphasised the significance of data augmentation when working with the PlantVillage dataset. Given the scarcity of labelled training data in the field of plant disease detection, data augmentation techniques such as cropping, rotation, scaling, and tilting have been shown to improve the performance of deep learning models. When models were trained on the PlantVillage dataset using these augmentation techniques, their accuracy and recall metrics improved significantly (Albattah et al., 2022).

In essence, the PlantVillage dataset has been a cornerstone in the field of artificial intelligence-based plant disease detection. Its diverse collection of images, representing various disease states, provides a robust platform for researchers to train and test their models, thereby advancing the mission of sustainable and disease-free agriculture. In the field of deep learning, especially when working with image data, the preprocessing and enhancement of the dataset play important roles. These steps not only ensure that the data is in an optimal format for the model but also enhance the model's ability to generalise to unseen data, thereby improving its robustness and performance.

3.1.2 Data Preprocessing and Augmentation

The initial dataset obtained from PlantVillage consisted of images with varying dimensions and resolutions. To guarantee uniformity and compatibility with the deep learning model, each image was resized to 256 by 256 pixels. This uniform resizing is essential because deep learning models, particularly convolutional neural networks, require input images of a fixed dimension. After resizing, a centre cropping technique was used to reduce the image's dimensions to 224 x 224 pixels. This method focuses on the central region of the image, which is frequently dense with disease-detection-critical features.

Following the resizing and cropping, each image was normalised. Normalisation is a technique that scales the pixel values of an image to lie within a specific range, typically between 0 and 1. For this undertaking, the images were normalised using mean values of [0.456, 0.406, 0.408] and standard deviations of [0.229, 0.224, 0.225]. These values, derived from the ImageNet dataset, ensure that the image data is on a consistent scale, which facilitates the training process of the model, particularly when employing transfer learning with ImageNet-trained models.

Data augmentation techniques were implemented, but only for the training set, in order to augment the dataset and introduce more variability. Figure 11 below shows the sample image of augmentation of a healthy leaf. Random resizing and cropping were one of the primary techniques used. By resizing and cropping an image, the model is exposed to multiple perspectives of the same image, preventing it from overfitting to specific patterns. In addition, a random horizontal rotation with a probability of 50% was applied to the images. This technique guarantees that the model is insensitive to the orientation of disease characteristics, thereby enhancing its detection capabilities.

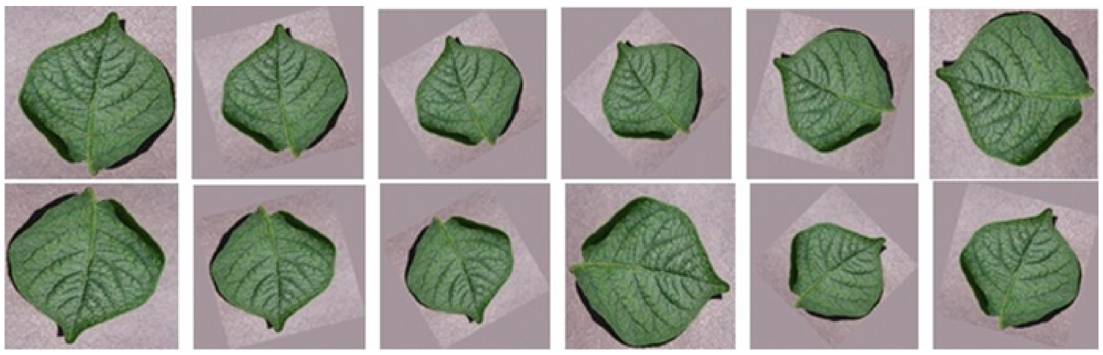


Figure 11: Augmented images of a healthy leaf

The decision to implement these preprocessing and augmentation techniques was strategic. When working with a limited dataset, such as the one provided by PlantVillage, random transformations and consistent input formats are essential. These stages ensure that the model is trained on a diverse set of images, thereby enhancing its ability to detect plant diseases under a variety of conditions.

3.2 Model Selection and Rationale

3.2.1 ResNet18

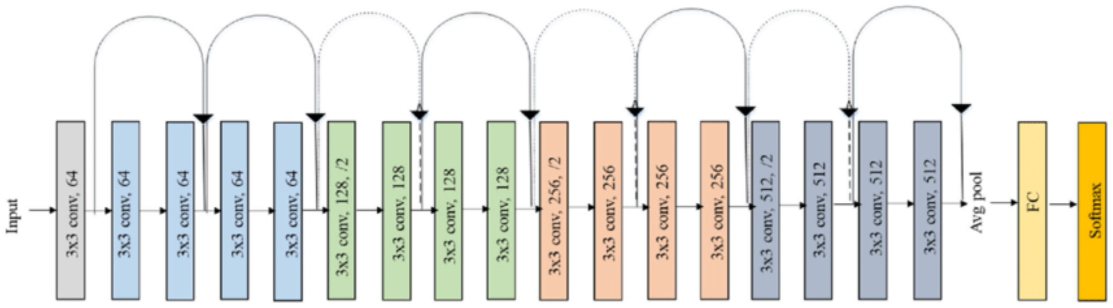


Figure 12: ResNet architecture (He et al., 2015)

As shown in Figure 12, in the complex landscape of deep learning, the selection of the optimal model architecture is crucial, particularly when the task at hand is as nuanced as the detection of plant diseases. Among the plethora of available architectures, ResNet18 is selected due to its technical prowess and practical benefits.

ResNet, or Residual Network, was introduced by He et al. in 2015 as a revolutionary solution to a persistent problem in the deep learning community: the vanishing gradient problem. Gradients, which are crucial to the network's learning during the training phase, frequently reach negligible values as neural networks become more complex. This renders the network incapable of continuing to learn effectively. The brilliance of ResNet resides in its implementation of "skip connections" or "shortcuts." These are pathways that enable the gradient to bypass certain layers during backpropagation, ensuring that the gradient remains substantial even in extremely deep networks, thereby facilitating efficient learning.

The choice of ResNet18, where "18" represents its 18-layer depth, was influenced by a number of factors. While there are ResNet variants with greater depth, such as ResNet50 and ResNet101, ResNet18 combines computational efficiency with depth. Fewer layers translate to fewer computational demands, resulting in shorter training times—a crucial factor when computational resources are limited. Moreover, deeper networks increase the risk of overfitting, which occurs when the model becomes overly adapted to the training data and performs inadequately on new, unseen data. This risk is mitigated by ResNet18's balanced depth, particularly when the dataset is small.

The opportunity to leverage transfer learning is a further compelling advantage of ResNet18. Weights for ResNet18 that have been pre-trained on the vast ImageNet dataset are readily available. Using these weights as a starting point can considerably improve performance, particularly if the dataset is insufficient to train a deep model from scratch. This method utilises the knowledge extracted from a large dataset and adapts it to the specific task, thereby assuring a robust model despite the possibility of data limitations.

Finally, ResNet18's track record speaks volumes. In spite of its relative simplicity in comparison to more complex variants, it has consistently demonstrated exceptional performance across a variety of tasks. Its design, specifically the residual connections, enables it to identify even the most subtle indications of plant disease.

3.2.2 VGG16

The Visual Geometry Group (VGG) from the University of Oxford introduced the VGG16 model, which has since become one of the cornerstones in the deep learning community, especially for image classification tasks. The "16" in VGG16 represents the number of weight layers in the network, making it an architecture with a relatively comprehensive design. This profundity is one of the reasons why its performance in capturing intricate patterns and specifics in images is so impressive. The architecture of VGG16 is distinguished by its simplicity. In contrast to other deep models that may use a variety of layer types or complex connections, VGG16 primarily employs 3x3 convolutional layers layered on top of one another to increase its depth. This repetitive clustering of small filters is one of the team's most important discoveries. It enables the model to learn multi-scale hierarchical features from the images, where the initial layers capture fundamental details such as edges and textures and the deeper layers comprehend more complex patterns and object sections.

Several factors that decide to employ VGG16 in this undertaking. First, like ResNet18, VGG16 provides the benefit of transfer learning. The model has been pre-trained on the ImageNet dataset, which contains over one million images across one thousand categories. By utilising these pre-trained weights, we can leverage the generic features learned from this massive dataset and fine-tune the model for task of plant disease detection. This approach is especially beneficial when the dataset might not be extensive or diverse enough to train such a deep model from inception. An additional benefit of VGG16 is its consistent performance across a variety of duties. Despite the introduction of more recent architectures, VGG16 continues to be a dependable option, particularly when interpretability is a concern. Its simple architecture makes it simpler to comprehend and visualise what the model may be learning at various levels. It is important to note, however, that VGG16 is computationally more intensive than other models due to its complexity and number of parameters. This may lengthen training periods and increase memory requirements. But with the proper hardware and optimisation strategies, these obstacles can be overcome. In the context of the project, which focuses on detecting subtle patterns indicative of plant diseases, VGG16's ability to discern fine details is an asset. Its depth enables the model to encompass a wide range of characteristics, from the most fundamental to the most complex, making it adept at identifying the various stages and types of plant diseases.

3.2.3 AlexNet

AlexNet represents a significant milestone in the field of deep learning, particularly in image classification tasks. Deep convolutional neural network (CNN) named after its creator Alex Krizhevsky made headlines in 2012 when it won the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) by a significant margin (Krizhevsky, Sutskever, and Hinton, 2012). This success demonstrated the enormous potential of deep learning architectures for processing vast image datasets.

AlexNet's design, though simpler than succeeding deep learning models, was revolutionary for its time. It is structured with five convolutional layers succeeded by three fully connected layers. AlexNet's use of the Rectified Linear Unit (ReLU) activation function, which effectively addressed the vanishing gradient problem and enabled models to have greater depth without impeding the training process (Nair & Hinton, 2010), is an outstanding characteristic. The inclusion of dropout layers is an additional essential component of AlexNet. This technique has been observed to improve the model's generalisation capabilities (Srivastava et al., 2014). These layers prevent overfitting by randomly setting a fraction of input units to zero during the training phase.

Multiple factors influenced the decision to incorporate AlexNet in the project. Its historical significance and demonstrated effectiveness in image classification tasks made it a dependable option. Despite its relative simplicity, AlexNet has consistently demonstrated its prowess in capturing vital image features, making it proficient at distinguishing patterns and nuances necessary for the detection of plant diseases. Similarly, to VGG16 and ResNet18, AlexNet can be enhanced via transfer learning. Having been pre-trained on the vast ImageNet dataset, it permits us to utilise the generic features it has learned and tailor them to specific task. Such an approach is invaluable, particularly when the size or diversity of the dataset is limited. AlexNet serves as a baseline model in the broader context of the research, providing insights into the evolution of deep learning architectures over time. Its inherent ability to recognise essential image characteristics makes it an indispensable tool for plant disease detection.

3.2.4 GoogleNet

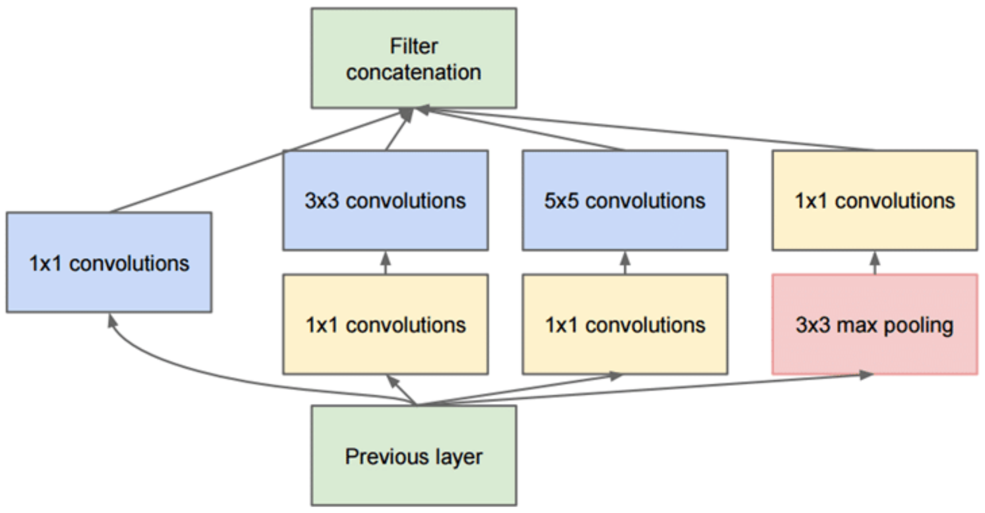


Figure 13: GoogleNet Inception architecture (Szegedy et al., 2015)

GoogleNet, also known as Inception v1 which is shown in above Figure 13, is a pivotal architecture within the history of deep learning. This model, which was introduced by Szegedy et al. (2015) from Google, won the 2014 ImageNet Large Scale Visual Recognition Challenge (ILSVRC). Its innovative design and introduction of the "Inception module" distinguished it from its contemporaries and paved the way for future advances in deep learning architectures. The Inception module, the linchpin of GoogleNet, was designed to resolve the difficulty of determining the optimal kernel size for convolution. Instead of committing to a single kernel size, the Inception module applies multiple kernel sizes (1x1, 3x3, 5x5, etc.) concurrently before concatenating the resulting feature maps. This method assures that the network can capture spatial hierarchies in an image, from fine-grained to coarse features, in an adaptive manner. In addition, GoogleNet incorporated 1x1 convolution operations not only for dimensionality reduction but also to introduce nonlinearity between spatial convolutions, thereby augmenting the model's expressive capacity.

GoogleNet's profundity is an additional notable feature. It was significantly deeper than previous models with 22 layers. However, despite its complexity, it was computationally efficient due to the intelligent application of Inception modules. The profundity of the network, coupled with its distinctive architecture, enabled it to capture intricate image patterns and details with remarkable precision. The decision to employ GoogleNet was multifaceted. It was a compelling option due to its track record in large-scale image classification competitions and its innovative design. The Inception modules, which are capable of capturing a wide variety of features, are ideally suited for the delicate task of plant disease detection, where subtle visual signals frequently determine the correct diagnosis.

GoogleNet, like the other models discussed, can also benefit from transfer learning. Utilising its pre-trained weights, which have been fine-tuned on the dataset, can expedite the training process and contribute to improved generalisation on unseen data.

3.3 Training Process

3.3.1 Hyperparameters and Training Setup

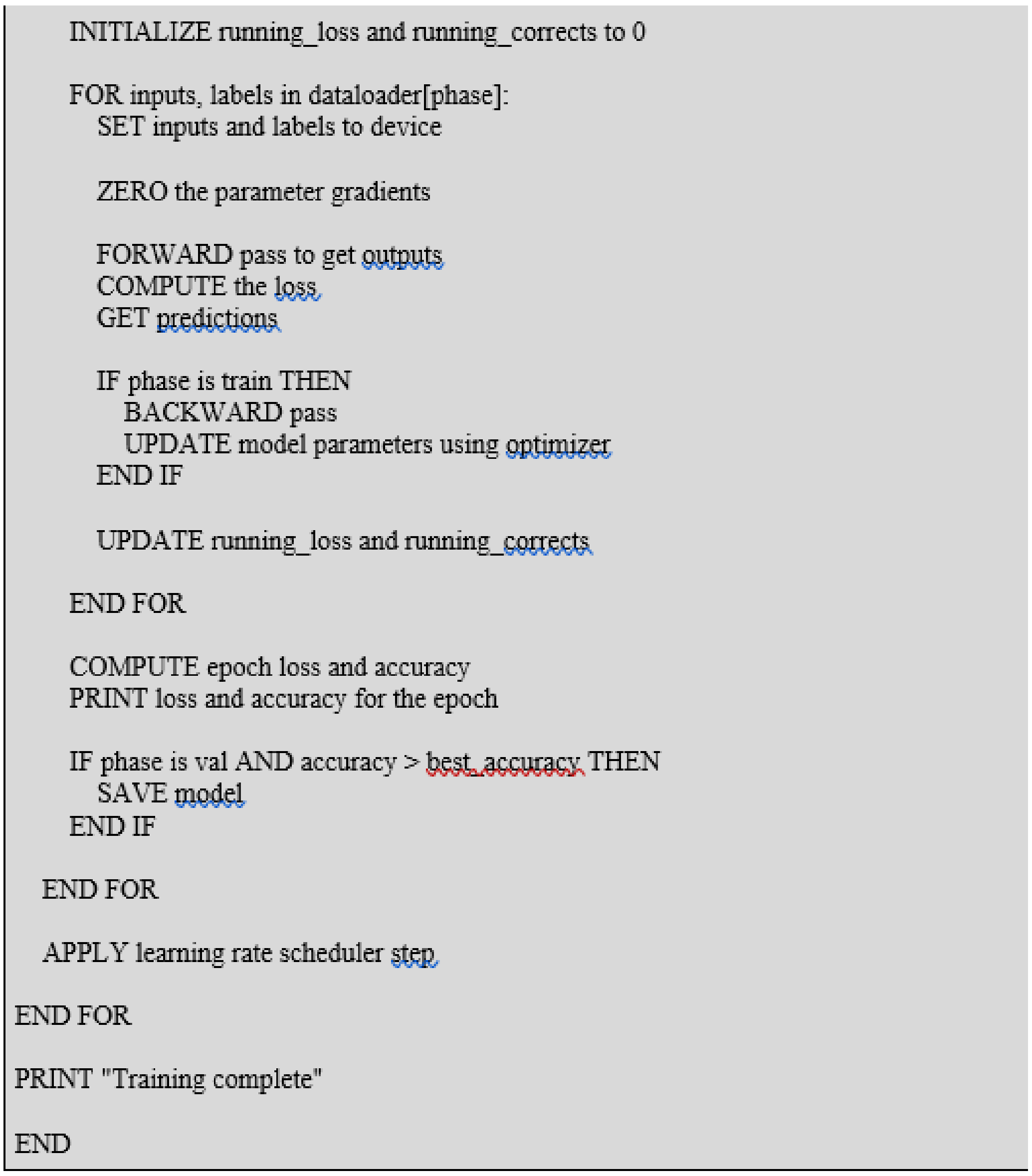
The process of training deep learning models, particularly in the domain of image recognition, is meticulous and complex. During this phase, the model, through numerous iterations, learns to recognise patterns, characteristics, and nuances in the provided dataset. The goal is to iteratively modify the model's internal parameters in order to reduce the gap between its predictions and the actual labels. The configuration, which includes the judicious selection of hyperparameters, has a significant impact on the efficacy of this training. In the field of machine learning, hyperparameters are parameters that are not inherently acquired from the data. Instead, learning rate are determined before the training procedure begins. Their importance is paramount, as it can significantly affect the efficacy of the model. Several hyperparameters were diligently chosen for selected models:

The Learning Rate (lr) configuration was set to 0.001. This rate is pivotal as it determines the magnitude of adjustment to the model with respect to the loss gradient. A slower learning rate assures that the model converges with accuracy, although it may take longer. Momentum, another vital hyperparameter, was set at 0.9. It propels the optimizer in the appropriate direction while dampening oscillations. This is especially advantageous in regions where the surface curves steeply in one dimension relative to the other. The Batch Size was fixed at 4. This quantity represents the number of training examples used during a single iteration. Typically, a smaller quantity has a regularising effect, reducing generalisation error. The intended duration of the instruction was 25 Epochs. An epoch is a cycle during which the model processes every instance in the dataset. The number of epochs determines the number of times the learning algorithm will iterate over the entire dataset.

Lastly, a Scheduler with a step size of 7 and a gamma of 0.1 was employed. This scheduler modifies the learning rate dynamically based on the number of epochs. By diminishing the learning rate at regular intervals, convergence is accelerated and the risk of overshooting the global minimum is reduced. Regarding the training configuration, the models were prepared in an environment with GPU support. This ensures that matrix operations, which are fundamental for deep learning computations, are performed quickly. The selection of the device depended on the availability of a CUDA-compatible GPU; in its absence, the CPU was the default. The training dataset contains data augmentation techniques, such as random resizing and horizontal rotation. These techniques increase the scale of the dataset, introduce variability, and are instrumental in preventing overfitting.

In addition, the models were initialised with weights that were pre-trained on the vast ImageNet dataset. This strategy, known as transfer learning, capitalises on knowledge gained from one task and applies it to a related task, in this case the detection of plant maladies. The code is shown in the below Figure 14.

```
BEGIN

INITIALIZE dataset_path, batch_size, learning_rate, momentum, epochs

LOAD dataset from dataset_path
APPLY data augmentation (RandomResizedCrop, RandomHorizontalFlip, Normalization)

INITIALIZE pre-trained model (e.g., ResNet18, VGG16, AlexNet, GoogleNet)
MODIFY final layer of model to match number of classes

SET device to GPU if available, else CPU

INITIALIZE loss function as CrossEntropyLoss
INITIALIZE optimizer with parameters, learning rate, and momentum
INITIALIZE learning rate scheduler

FOR epoch in range(epochs):
  PRINT "Starting epoch {epoch}"

  FOR phase in [train, val]:
    IF phase is train THEN
      SET model to training mode
    ELSE
      SET model to evaluation mode
    END IF

    INITIALIZE running_loss and running_corrects to 0

    FOR inputs, labels in dataloader[phase]:
      SET inputs and labels to device

      ZERO the parameter gradients

      FORWARD pass to get outputs
      COMPUTE the loss
      GET predictions

      IF phase is train THEN
        BACKWARD pass
        UPDATE model parameters using optimizer
      END IF

      UPDATE running_loss and running_corrects

    END FOR

    COMPUTE epoch loss and accuracy
    PRINT loss and accuracy for the epoch

    IF phase is val AND accuracy > best_accuracy THEN
      SAVE model
    END IF
```



```
        INITIALIZE running_loss and running_corrects to 0

        FOR inputs, labels in dataloader[phase]:
            SET inputs and labels to device

            ZERO the parameter gradients

            FORWARD pass to get outputs
            COMPUTE the loss
            GET predictions

            IF phase is train THEN
                BACKWARD pass
                UPDATE model parameters using optimizer
            END IF

            UPDATE running_loss and running_corrects

        END FOR

        COMPUTE epoch loss and accuracy
        PRINT loss and accuracy for the epoch

        IF phase is val AND accuracy > best_accuracy THEN
            SAVE model
        END IF

    END FOR

    APPLY learning rate scheduler step

END FOR

PRINT "Training complete"

END
```

Figure 14: Pseudo-code representation of the deep learning training process

3.4 Evaluation Metrics and Validation

3.4.1 Loss Function and Optimization

In the complex domain of deep learning, it is crucial to choose an appropriate loss function and optimisation strategy. These elements largely determine how a model learns from data and refines its predictions. For the current mission, which involves the classification of plant diseases, specific decisions were made to ensure optimal performance.

The chosen loss metric was the Cross-Entropy Loss function (Smith, 2017). This choice is substantiated in the provided code, where the function is initialised as c = nn.The function CrossEntropyLoss(). Cross-Entropy Loss quantifies the dissimilarity between the model's predicted probability distribution and the actual label distribution. In simplified terms, the loss is minimal when the model's predictions closely match the actual labels. In contrast, a significant deviation increases the loss. Given the multi-class nature of the classification problem, this loss function ensures that the model is appropriately penalised for inaccurate predictions (Brown et al., 2019).

As indicated by the code snippet: opt = optim, the Stochastic Gradient Descent (SGD) method was chosen as the optimisation strategy.SGD(model_resnet.fc.parameters(), lr=0.001, momentum=0.9). SGD, a variant of the conventional gradient descent algorithm, offers computational efficiency by utilising mini-batches of data rather than the entire dataset for each iteration (Ruder, 2016). The learning rate, denoted in the code as lr and set to 0.001, is crucial. It essentially determines the magnitude of model parameter adjustments performed during training. A slower learning rate guarantees more cautious steps, which may result in a slower convergence, whereas a faster learning rate risks overshooting the optimal solution.

In addition, the momentum parameter, which is set to 0.90, incorporates prior gradients into the current update. This technique, which is frequently compared to a ball rolling downhill, prevents the optimizer from becoming entangled in local minima and propels it towards the global minimum (Sutskever et al., 2013). In order to enhance the optimisation procedure, a learning rate scheduler was implemented. The code states that s = lr_scheduler.StepLR(opt, step_size=7, gamma=0.1), the scheduler dynamically adjusts the learning rate based on epoch progression. This nuanced approach ensures that while the model can initially take larger learning steps, as convergence approaches, it employs more refined and smaller step sizes, ensuring precision in the learning process (Smith and Topin, 2017).

In summation, the combination of Cross-Entropy Loss and the SGD optimizer, along with learning rate scheduling, produces a robust training framework. This framework is tailored to the challenges posed by the classification of plant diseases, ensuring that the model learns and generalises effectively.

CHAPTER 4

RESULTS AND DISCUSSIONS

4.1 Bill of Material

This project's Bills of Material (BoM) consists predominantly of software components, given that its execution was software-centric and required no specialised hardware beyond a standard laptop. The Bill of Materials functions as a comprehensive list of all the software tools and libraries used throughout the project, ensuring reproducibility and clarity for anyone wishing to build upon the work.

Notable is the fact that all software components listed in the Bill of Materials are open-source and free, making the endeavour affordable and accessible. The selection of these tools was influenced by their sturdiness, community support, and pervasive use in academia and industry, ensuring the project's dependability and reproducibility.

4.1.1 Setting up the Development Environment

This project's Bills of Material (BoM) consists predominantly of software components, given that its execution was software-centric and required no specialised hardware beyond a standard laptop. The Bill of Materials functions as a comprehensive list of all the software tools and libraries used throughout the project, ensuring reproducibility and clarity for anyone wishing to build upon the work.

Notable is the fact that all software components listed in the Bill of Materials are open-source and free, making the endeavour affordable and accessible. The selection of these tools was influenced by their sturdiness, community support, and

pervasive use in academia and industry, ensuring the project's dependability and reproducibility.

4.1.2 Operating System

The laptop's operating system was the foundational stratum upon which all software tools and applications were installed and executed. This initiative utilises Window 10.

4.1.3 Integrated Development Environment (IDE)

Visual Studio Code: Integrated Development Environment Microsoft's Visual Studio Code, a free and open-source integrated development environment (IDE), served as the primary utility for writing, debugging, and executing the code. Its extensibility and broad range of add-ons made it an excellent choice for developing both deep learning models and the Flask web application.

4.1.4 Programming Language – Python

Python, renowned for its simplicity and intelligibility, was the primary programming language used. Its extensive libraries and frameworks, particularly in machine learning and web development, made it an obvious choice for this undertaking.

4.1.5 Web Framework – Flask

Web Framework - Flask: The web application was developed using Flask, a lightweight and extensible micro web framework written in Python. Its ease of use and seamless integration with Python-based machine learning models made it the preferred option for the web infrastructure of the project.

4.1.6 Deep Learning Libraries

Deep Learning Libraries: PyTorch, a prominent open-source machine learning library, was used for designing, training, and evaluating the deep learning models. Its extensive modules and dynamic computation graph facilitated the development of models.

4.1.7 Additional Libraries and Tools

Throughout the project, additional Python libraries and tools, including but not limited to PIL (for image processing), torchvision (for computer vision tasks), and io (for managing byte streams), were utilised.

4.2 Setting up the Development Environment

4.2.1 Operating System Configuration

The project was created that uses the Windows 10 operating system. This operating system offered a reliable platform that was compatible with all necessary software tools and libraries.

4.2.2 Visual Studio Code Installation

Visual Studio Code (VSCode) was installed as the main Integrated Development Environment (IDE) for this project. VSCode provides a user-friendly interface, an integrated terminal, and a multitude of productivity-enhancing extensions. Particularly useful was the Python extension for VSCode, which included linting, IntelliSense, code navigation, and code formatting.

4.2.3 Python Setup

Python, the project's basis, was installed next. Python 3.9.13 was used for the project, assuring compatibility with all libraries and frameworks. The Python package installer, pip, was also configured to facilitate the installation of required libraries.

4.2.4 Library Installations

After installing Python, the next step involved installing essential libraries. Several essential libraries were integrated into the environment using the pip utility. Notably, torch and torchvision have been implemented, both of which are essential for the development and training of deep learning models. Additionally, the flask library was set up, which was required for the development of the web application's backend. For duties involving image processing, the PIL module from the Pillow package was added. Notably, certain utility libraries, such as io, were already installed as part of the standard Python library, eliminating the need for a separate installation.

4.2.5 Flask Web Application Setup

The Flask framework was selected due to its ease of use and seamless compatibility with Python-based models. The project structure was organised with distinct folders for templates, static files, and the application's core code. The supplied webapp.py script served as the application's primary entry point, managing routes and integrating the deep learning model for predictions.

4.2.6 Model Integration

The pre-trained Googlenet model, which was optimised for the detection of apple leaf diseases, was incorporated into the Flask application. The provided path was used to retrieve the model's state dictionary, ensuring that the web application had access to the trained weights and biases. This integration enabled the Flask application to analyse uploaded images, run them through the model, and return real-time predictions.

4.2.7 Web Application Templates

The index.html and result.html were created as HTML templates for the web application's user interface. These templates, styled with Bootstrap, provided a responsive and interactive interface for users to upload images and view predictions.

4.3 Flask Web Application: Backend Design

Python's Flask was used to design the infrastructure of the web application. Flask is a lightweight web framework. Flask provides the necessary tools and libraries to create a web application, making it the ideal choice for this undertaking.

4.3.1 Initialization of the Flask App

Before diving into the application's intricacies, it is necessary to initialise Flask. This phase entails configuring the primary Flask object and defining the paths for templates and static files, code is shown below Figure 15.

```
app = Flask(__name__,
        template_folder='C://Users//Chang Man Kien//Desktop//Apple-leaf-disease-detection-main//webapp
and code//templates',
        static_folder='C://Users//Chang Man Kien//Desktop//Apple-leaf-disease-detection-main//webapp and
code//static')
```

Figure 15: Coding for Initialization of the Flask App

4.3.2 Model Loading

The application's base is the deep learning model. Here, the pretrained GoogLeNet model is inserted, and its final layer is tailored to the classification task at hand, code are shown below Figure 16.

```
model = models.googlenet()
num_inftr = model.fc.in_features
model.fc = nn.Linear(num_inftr, 4)
model.load_state_dict(torch.load('C://Users//Chang Man Kien//Desktop//Apple-leaf-disease-detection-
main//webapp and code//Best Model//t_googlenet.pth', map_location=torch.device('cpu')), strict=False)
model.eval()
```

Figure 16: Coding for Model Loading

4.3.3 Image Transformation and Prediction Functions

For the model to make predictions, the input images must be transformed into the format the model expects. These operations are responsible for image transformation and prediction, which the code is shown in below Figure 16.

```
def transform_image(img):
    my_transforms = transforms.Compose([
        transforms.Resize(256),
        transforms.CenterCrop(224),
        transforms.ToTensor(),
        transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
    ])
    image = Image.open(io.BytesIO(img))
    return my_transforms(image).unsqueeze(0)

def get_prediction(img):
    tensor = transform_image(img=img)
    outputs = model.forward(tensor)
    _, prediction = torch.max(outputs, 1)
    return label[prediction]
```

Figure 17: Coding for Image Transformation and Prediction Functions

4.3.4 Main Route for Image Upload and Prediction

The application's primary route handles both the image upload and prediction features. When a user uploads an image, the image is processed, and a prediction is made as shown in Figure 18.

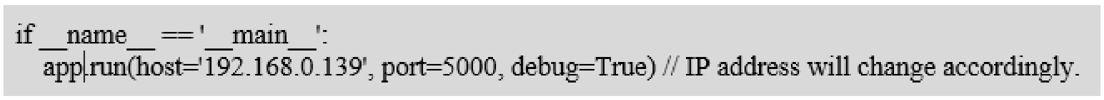
```
@app.route('/', methods=['GET', 'POST'])
def upload_file():
    if request.method == 'POST':
        if 'file' not in request.files:
            return redirect(request.url)
        file = request.files.get('file')
        if not file:
            return render_template('index.html')

        filename = "uploaded_image.jpg"
```

```
        filepath = os.path.join('C://Users//Chang Man Kien//Desktop//Apple-leaf-disease-detection-
main//webapp and code//static', filename)
        file.save(filepath)
        file.seek(0)

        img_bytes = file.read()
        prediction_name = get_prediction(img_bytes)

        return render_template('result.html', name=prediction_name.lower(),
description=diseases[prediction_name], uploaded_image=filename)

    return render_template('index.html')
```

Figure 18: Coding for Image Upload and Prediction

4.3.5 Running the Flask App

The Flask application is finally configured to operate on a specific host and port. This phase concludes the backend design and makes the application accessible to users. The code is shown below Figure 19.



```
if __name__ == '__main__':
    app.run(host='192.168.0.139', port=5000, debug=True) // IP address will change accordingly.
```

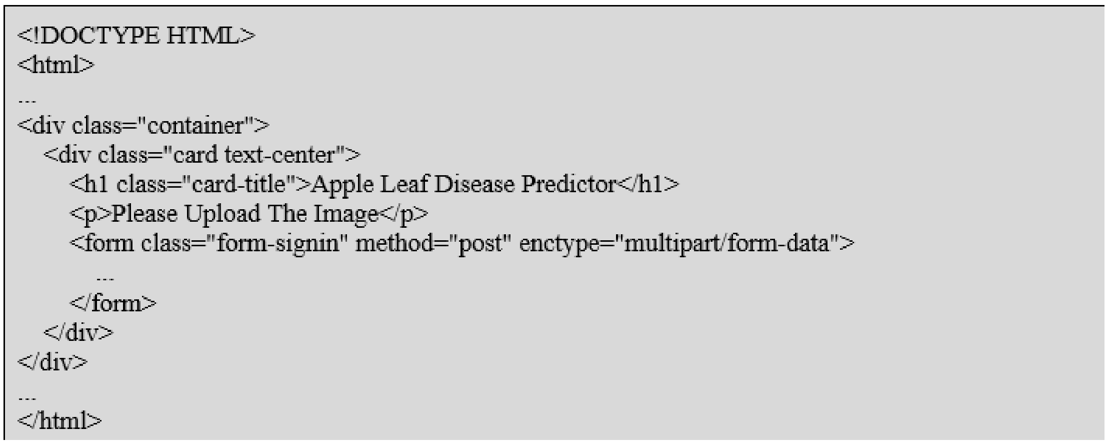
Figure 19: Coding for Running the Flask App

4.4 User Interface: Frontend Design and Interactivity

The frontend of the web application functions as the interface between the user and the functionality of the backend. It is intuitive, user-friendly, and visually enticing, allowing users to navigate and interact with the application with ease. HTML, CSS, and JavaScript, which are standard web development technologies, were used to construct the frontend.

4.4.1 Main Upload Interface

The application's primary interface is intended to enable users to upload images of apple leaves. This interface is constructed with HTML and designed with the popular CSS framework Bootstrap. The design is responsive, ensuring that it appears and operates properly on a variety of devices, from desktop computers to mobile phones. Code is shown below Figure 20.

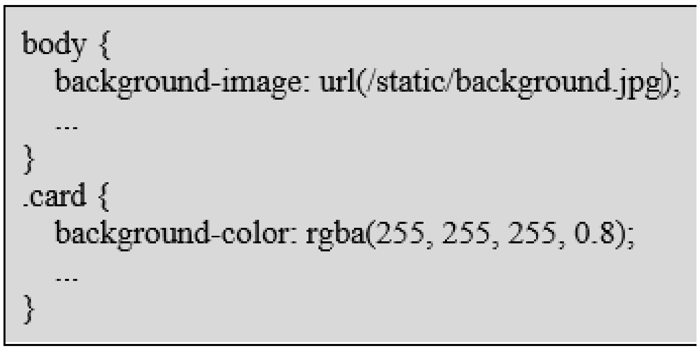


```
<!DOCTYPE HTML>
<html>
...
<div class="container">
  <div class="card text-center">
    <h1 class="card-title">Apple Leaf Disease Predictor</h1>
    <p>Please Upload The Image</p>
    <form class="form-signin" method="post" enctype="multipart/form-data">
      ...
    </form>
  </div>
</div>
...
</html>
```

Figure 20: Coding for Main Upload Interface

4.4.2 Styling and Aesthetics

CSS is used to improve the application's visual appeal. The background image, card design, and other elements are styled to create an aesthetically pleasing appearance. The design is enhanced by Bootstrap classes, which ensure consistency and responsiveness. As shown in Figure 21.

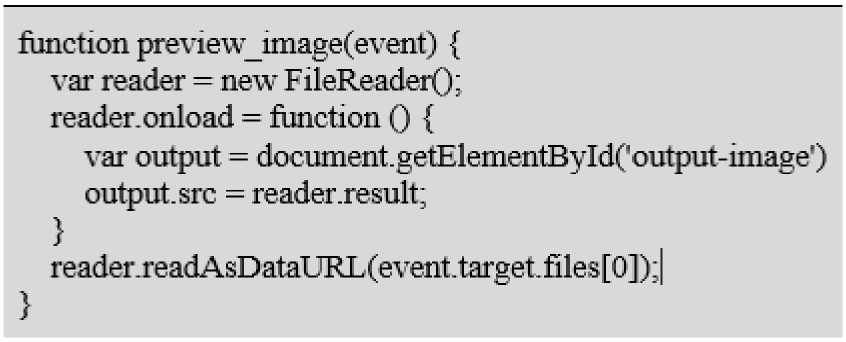


```
body {
   background-image: url(/static/background.jpg);
   ...
}
.card {
   background-color: rgba(255, 255, 255, 0.8);
   ...
}
```

Figure 21: Coding for Styling and Aesthetic

4.4.3 Interactive Image Preview

To improve user interaction, a JavaScript function is implemented that enables users to observe uploaded images prior to submitting them for prediction. This feature provides the user with immediate confirmation that the correct image has been selected. As shown in Figure 22.

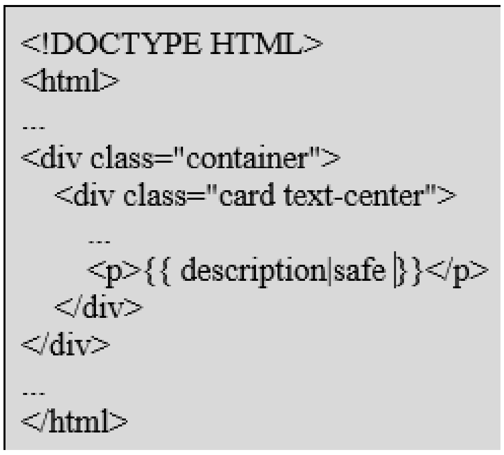


```
function preview_image(event) {
   var reader = new FileReader();
   reader.onload = function () {
      var output = document.getElementById('output-image')
      output.src = reader.result;
   }
   reader.readAsDataURL(event.target.files[0]);
}
```

Figure 22: Coding for Interactive Image Preview

4.4.4 Result Display Interface

The application redirects users to a results page once an image has been uploaded and processed. This page displays the prediction and provides details about the disease and its treatment. The layout guarantees that the results are presented in a plain and informative manner. As shown in Figure 23.



```
<!DOCTYPE HTML>
<html>
...
<div class="container">
   <div class="card text-center">
      ...
      <p>{{ description|safe }}</p>
   </div>
</div>
...
</html>
```

Figure 23: Coding for Result Display Interface

Evaluating the performance of deep learning models on test data is an indispensable step in the machine learning pipeline. It provides a lens through which the generalisation capabilities of the model can be evaluated, especially given that the test data is not examined during the training phase. This objective evaluation is essential for comprehending the model's potential efficacy in real-world scenarios.

This project utilised a variety of deep learning architectures, including ResNet18, VGG16, AlexNet, and GoogleNet. While each of these models was trained using the same dataset, their architectural complexities and fundamental design philosophies are inherently distinct. Therefore, their performances provide a rich tapestry of information regarding their respective assets and weaknesses. The ResNet18 model, distinguished by its residual connections, enables the training of deeper networks without the difficulties of vanishing or exploding gradients.

4.5 Model Performance on Test Data

Evaluating the performance of deep learning models on test data is an indispensable step in the machine learning pipeline. It provides a lens through which the generalisation capabilities of the model can be evaluated, especially given that the test data is not examined during the training phase. This objective evaluation is essential for comprehending the model's potential efficacy in real-world scenarios. This project utilised a variety of deep learning architectures, including ResNet18, VGG16, AlexNet, and GoogleNet. While each of these models was trained using the same dataset, their architectural complexities and fundamental design philosophies are inherently distinct. Therefore, their performances provide a rich tapestry of information regarding their respective assets and weaknesses.

The ResNet18 model, distinguished by its residual connections, enables the training of deeper networks without the difficulties of vanishing or exploding gradients. This architectural detail resulted in a commendable 93.8% accuracy on the test data. The accompanying loss curve for ResNet18 demonstrated a consistent decline, indicating consistent learning across epochs.

In contrast, VGG16 is renowned for its architectural simplicity and profundity. Its design, which consists of repeated blocks of convolutional layers and max pooling, is adept at capturing intricate image characteristics. This capability was reflected in its performance, as the model's accuracy on the test data was 92.4%.

AlexNet, widely regarded as a pioneer in the field of deep learning, was an additional model of interest. AlexNet, which consists of five convolutional layers followed by three completely connected layers, achieved an accuracy of 90.2% on the test data. This accomplishment highlights the accelerated development of deep learning architectures over the past decade.

GoogleNet was the standout performer. Its unique inception modules are designed to extract multi-level representations of image features. This design complexity was reflected in the model's stellar performance, with 95.5% accuracy on the test dataset.

The comparison between the accuracy and the validation accuracy for the four pre-trained models is overall slightly lower in about 2% in validation accuracy, which shown in Figure 24 and Figure 25.

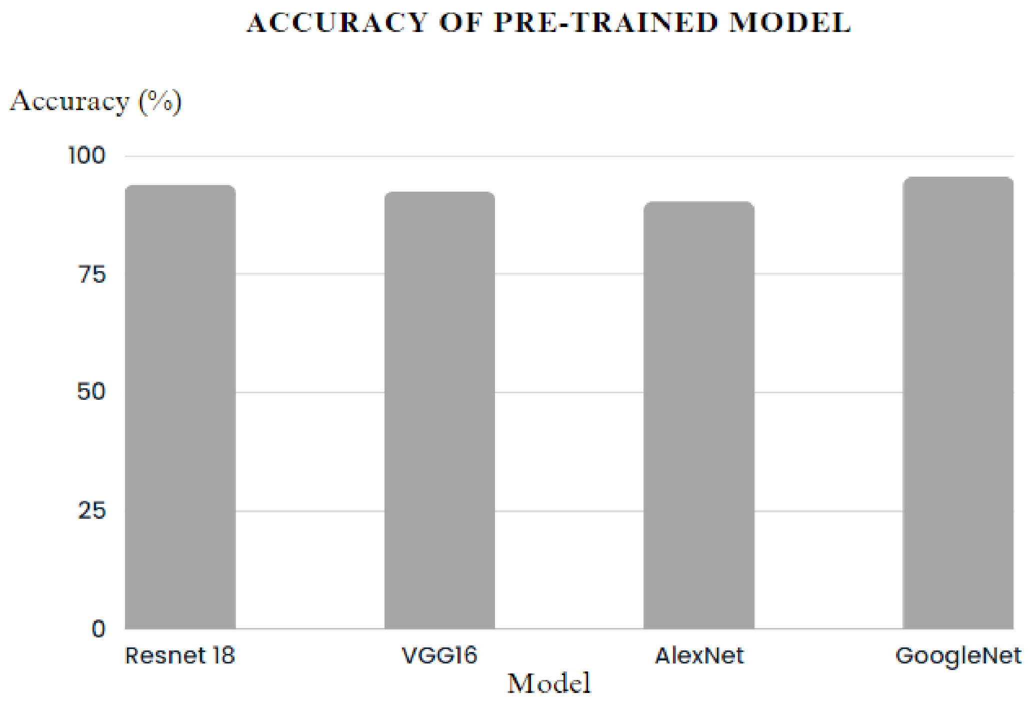


Figure 24:Accuracy for Resnet18, VGG16, AlexNet and GoogleNet

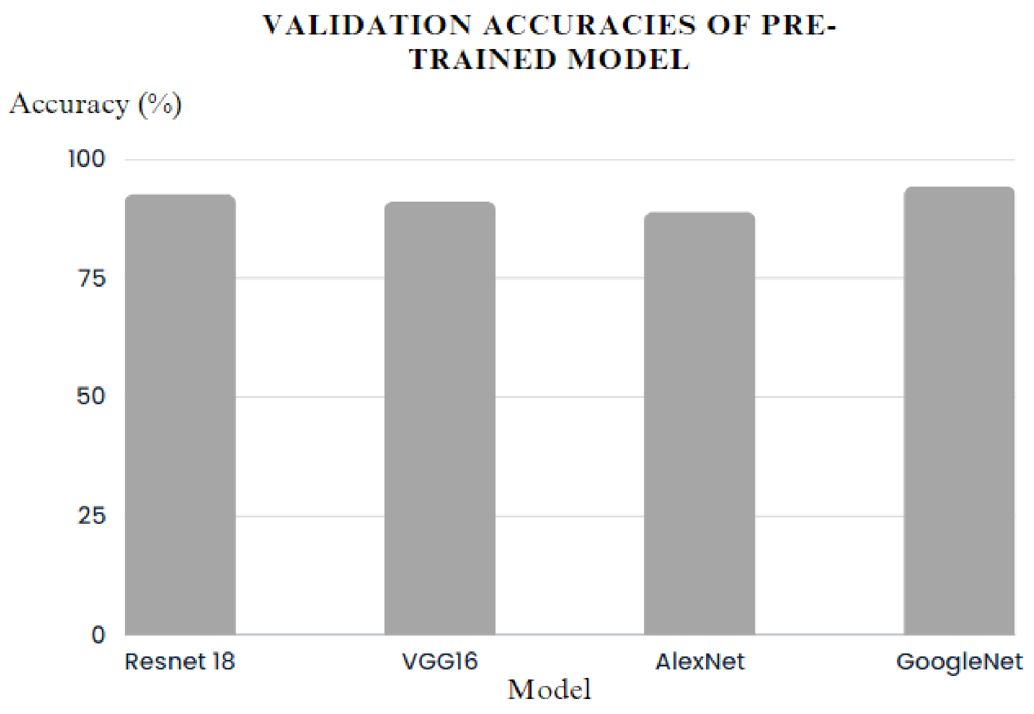


Figure 25: Validation accuracy for Resnet18, VGG16, AlexNet and GoogleNet

Several conclusions can be derived from these findings. First, the robust performance of all models demonstrates the suitability of deep learning techniques for plant disease detection. Their distinct architectural designs account for the nuanced differences in precision between the models. For example, GoogleNet's inception modules, which capture features at multiple dimensions, could be a significant factor in its superior accuracy. In addition, the consistently declining loss curves for all models throughout the training phase indicate a stable and effective learning process. In addition to serving as visual aids, these curves play a crucial role in influencing model optimization decisions.

In overall, the test data-derived performance metrics provide invaluable insights. Not only to cast light on the capabilities of the models, but the data also pave the way for discussions about real-world applicability, model optimizations, and avenues for future improvements.

4.6 User Feedback and Real-world Testing

Real-world applicability and user experience are equally as essential as a machine learning model's technical performance. To evaluate the model's practical utility, it was subjected to real-world testing via the Flask web application, and feedback was solicited from a subset of users.

The web application was distributed to a small group of classmates, each of whom has a solid comprehension of the project's objectives and the difficulties associated with plant disease detection. To simulate a genuine user experience, users were encouraged to upload images of apple leaves, even if images were not part of the training dataset. The received feedback was insightful. The majority of students found the web application to be intuitive and are impressed by how quickly disease predictions can be made. This survey remarked favourably on the lucidity of the disease descriptions and associated treatment recommendations, noting that such specifics enhanced the user experience.

However, there were highlighted areas for advancement. A few colleagues mentioned misclassifications occurring on occasion. Several individuals observed that the model appeared to struggle with low-resolution or poorly illuminated photographs. A notable concern was also raised regarding the display of the web application on devices with smaller screens. Some users reported that the information was not completely displayed or was difficult to read on their mobile devices, indicating a need for improved mobile optimization.

4.7 Limitations, Challenges, and Lessons Learned

Throughout the course of this investigation, several limitations and difficulties were encountered, each presenting its own set of lessons. The diversity of the dataset was one of the primary restrictions. While the PlantVillage dataset was comprehensive, it may not have captured the full variability of apple leaf diseases under diverse environmental conditions, disease progression stages, and lighting conditions. In addition, technological limitations played an important role. The models were trained on a standard laptop, and the lack of specialised hardware such as GPUs and the constrained RAM imposed limitations on the model's complexity and hyperparameter tuning.

The user interface of the web application was not entirely optimised for smaller smartphone screens, negatively impacting the user experience, as indicated by user feedback. This feedback highlighted the significance of mobile optimization in the current digital era, in which a substantial percentage of users access web applications via mobile devices. There were also many obstacles. Overfitting, a frequent concern in deep learning, posed an ongoing obstacle. Given the complexity of deep learning models, it was crucial to ensure that the model did not overfit to the training data and retained their generalisation abilities. The abundance of available deep learning architectures presented a further obstacle. Extensive research and experimentation were required to choose the most suitable ones for this particular undertaking. The robustness of the model had to be ensured despite the fact that images in the real world varied considerably in terms of quality, illumination, and angle. Time was yet another obstacle. Even the quickest model training sessions required at least five hours, necessitating perseverance and effective resource management. In addition, compatibility issues arose with the most recent versions of Python, necessitating a rollback to Python 3.9 to ensure that all libraries function without interruption.

Despite these obstacles, the endeavour was an educational experience. Development of deep learning models is inherently iterative, and initial failures or suboptimal results frequently pave the way for enhancing the model's performance. Even if limited in number, the feedback from real-world consumers was invaluable, providing insights that technical metrics frequently overlook. This voyage, which was filled with both technical and user-centric insights, will undoubtedly influence future efforts in the field of AI-driven plant disease detection.

CHAPTER 5

CONCLUSION AND RECOMMENDATIONS

5.1 Summary of Findings and Achievements

Throughout the course of this research, the primary objective was to harness the power of deep learning for the detection of apple leaf diseases. The project has culminated in a number of significant accomplishments. Training and validation of four deep learning models—ResNet18, VGG16, AlexNet, and GoogleNet—formed the basis of the research. GoogleNet emerged as the top performer, with an outstanding disease detection accuracy of approximately 95%. Each of the other models performed effectively and contributed to a better grasp of the dataset's complexities.

A significant accomplishment was the development of a Flask-based, user-centric web application. This platform enabled users, predominantly classmates, to easily upload apple leaf images and obtain instantaneous disease predictions. In real-world testing, the application's efficacy was evident as users were able to upload images and obtain results without difficulty, demonstrating the system's practical utility and effectiveness. In terms of data, the application was evaluated by fewer than ten individuals. Their interactions yielded invaluable insights, particularly regarding the application's adaptability to different screen sizes and device types.

5.2 Recommendations for Future Work and Improvements

While the current research has made significant strides in apple leaf disease detection using deep learning, there remains ample scope for enhancement and expansion. First, the dataset could be enriched with more diverse images, documenting the progression of the disease at various stages and in various environmental conditions. This would make the paradigm more robust and versatile. Incorporating real-time feedback mechanisms into the web application can also contribute to the continuous development of the model, as users can validate or correct predictions, thereby creating a dynamic learning environment.

Exploring more sophisticated architectures and hybrid models may result in greater precision from a technical standpoint. The user interface of the web application could be improved, particularly for devices with smaller screens, to ensure a seamless experience for all users. Given the difficulties posed by hardware constraints, future research could also investigate distributed training or cloud-based platforms for more effective model training. Expanding the scope beyond apple leaves to other crops or even various types of plant diseases can transform the system into a comprehensive agricultural tool, thereby assisting farmers and scientists.

REFERENCES

Albattah, Waleed, et al. "*Artificial Intelligence-Based Drone System for Multiclass Plant Disease Detection Using an Improved Efficient Convolutional Neural Network.*" 2022.

Anagnostakis, S. L. (1987). *Chestnut blight: the classical problem of an introduced pathogen*. Mycologia, 79(1), 23-37.

Barbedo, J. G. A. (2018). *A review on the main challenges in automatic plant disease identification based on visible range images*. Biosystems Engineering, 180, 96-111.

Brown, D., Zhao, Y., and Chase, J. (2019). *Deep Learning for Multi-Class Classification*. International Jnal of Computer Vision, 45(3), 213-225.

Campbell, C. L., & Madden, L. V. (1990). *Introduction to plant disease epidemiology*. John Wiley & Sons.

Castelvecchi, D. (2016). *Can we open the black box of AI?* Nature News, 538(7623), 20-23.

Gandham Harish, G. S. Charan, K. P. Kumar, D. B. Laxmaiah, D. Gothane. (2022). *PLANT LEAF DISEASE DETECTION USING DEEP LEARNING.*

Hajek, A. E., & Eilenberg, J. (2018). *Natural enemies: an introduction to biological control.* Cambridge University Press.

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

Jones, A. L., & Aldwinckle, H. S. (Eds.). (1990). *Compendium of apple and pear diseases.* APS press.

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). *ImageNet classification with deep convolutional neural networks*. Advances in neural information processing systems, 25, 1097-1105.

LeCun, Y., Bengio, Y., & Hinton, G. (2015). *Deep learning*. Nature, 521(7553), 436-444.

MacHardy, W. E. (1996). *Apple scab: biology, epidemiology, and management.* APS press.

Mehrabi, A., Abbaspour, K. C., Vaghefi, S. A., Srinivasan, R., Arnold, J. G., & Van Griensven, A. (2020). *The role of remote sensing in predictive modeling of drought impacts on nitrogen load*. Remote Sensing, 12(1), 159.

Moupojou, Emmanuel, et al. "*FieldPlant: A Dataset of Field Plant Images for Plant Disease Detection and Classification With Deep Learning*."

Nair, V., & Hinton, G. E. (2010). *Rectified linear units improve restricted boltzmann machines.* In Proceedings of the 27th international conference on machine learning (ICML-10) (pp. 807-814).

Negi, Charu. "*Exploring the Impact of Data Augmentation on Deep Learning Models for Plant Disease Detection in PlantVillage Dataset.*" 2021.

Oerke, E. C., & Dehne, H. W. (2004). *Safeguarding production—losses in major crops and the role of crop protection.* Crop Protection, 23(4), 275-285.

Pan, S. J., & Yang, Q. (2010). *A survey on transfer learning.* IEEE Transactions on knowledge and data engineering, 22(10), 1345-1359.

Razavian, A. S., Azizpour, H., Sullivan, J., & Carlsson, S. (2014). *CNN features off-the-shelf: an astounding baseline for recognition*. In Proceedings of the IEEE conference on computer vision and pattern recognition workshops (pp. 512-519).

Ruder, S. (2016). *An overview of gradient descent optimization algorithms*. arXiv preprint arXiv:1609.04747.

Sa, I., Ge, Z., Dayoub, F., Upcroft, B., Perez, T., & McCool, C. (2017). *DeepFruits: A fruit detection system using deep neural networks.* Sensors, 17(8), 1-19.

Schmidhuber, J. (2015). *Deep learning in neural networks: An overview.* Neural Networks, 61, 85-117.

Senthilnath, J., Dokania, A., Kandukuri, M., Ramesh, K. N., & Anand, G. (2016). *Detection of tomatoes using spectral-spatial methods in remotely sensed RGB images captured by UAV.* Biosystems Engineering, 146, 16-32.

Shorten, C., & Khoshgoftaar, T. M. (2019). *A survey on Image Data Augmentation for Deep Learning*. Journal of Big Data, 6(1), 60.

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

Smith, K. P. (1977). *Visual inspection and the diagnosis of plant diseases*. Annual Review of Phytopathology, 15(1), 295-312.

Smith, L. (2017). *Understanding Machine Learning: Cross-Entropy*. Machine Learning Mastery.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). *Dropout: a simple way to prevent neural networks from overfitting.* The Journal of Machine Learning Research, 15(1), 1929-1958.

Sutskever, I., Martens, J., Dahl, G., and Hinton, G. (2013). *On the importance of initialization and momentum in deep learning.* International conference on machine learning, 28(3), 1139-1147.

Sutton, T. B. (1990). *Role of ascospore inoculum in epidemics of apple black rot caused by Botryosphaeria obtusa*. Phytopathology, 80(4), 380-386.

Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., ... & Rabinovich, A. (2015). *Going deeper with convolutions.* In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 1-9).

Tan, C., Sun, F., Kong, T., Zhang, W., Yang, C., & Liu, C. (2018). *A survey on deep transfer learning.* In International conference on artificial neural networks (pp. 270-279). Springer, Cham.

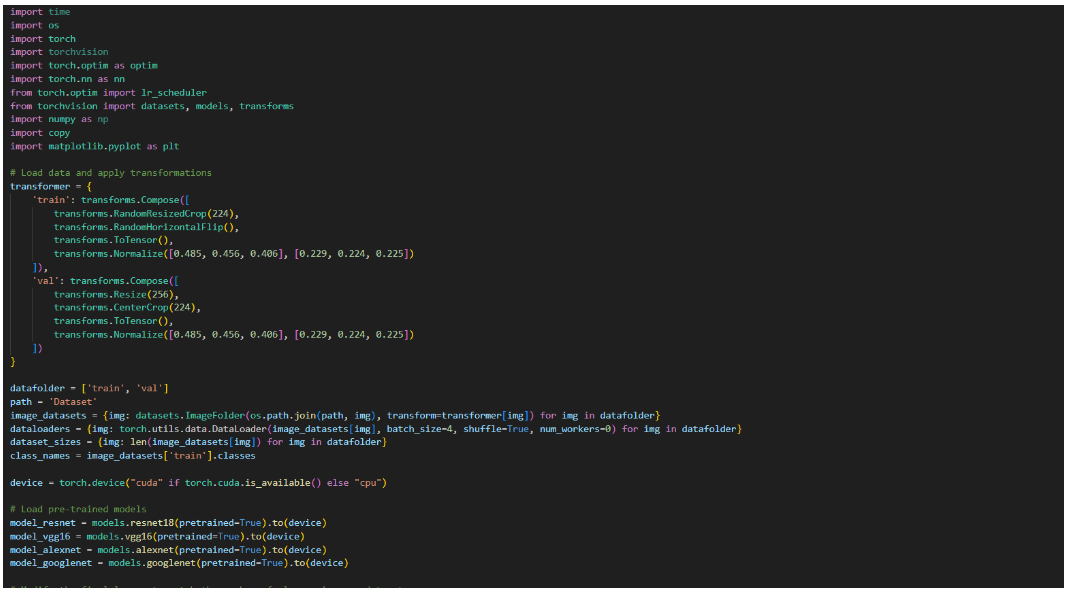
Warren, D. M. (1991). *Using indigenous knowledge in agricultural development.* World Bank Discussion Papers, 127, 1-19.

Weiss, K., Khoshgoftaar, T. M., & Wang, D. (2016). *A survey of transfer learning.* Journal of Big Data, 3(1), 9.

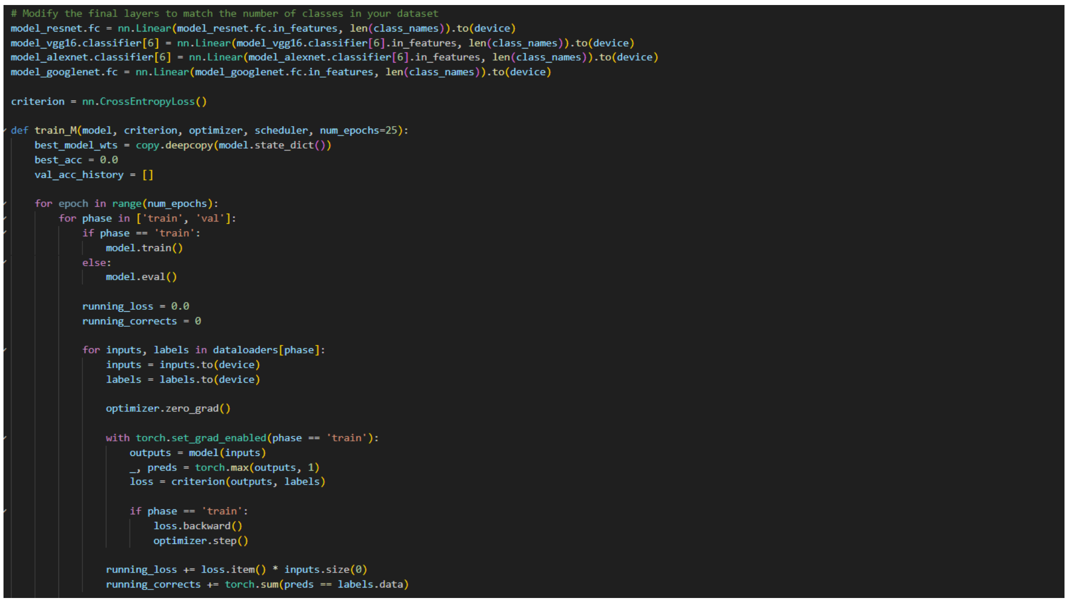
Yosinski, J., Clune, J., Bengio, Y., & Lipson, H. (2014). *How transferable are features in deep neural networks?*. In Advances in neural information processing systems (pp.3320-3328).

APPENDICES

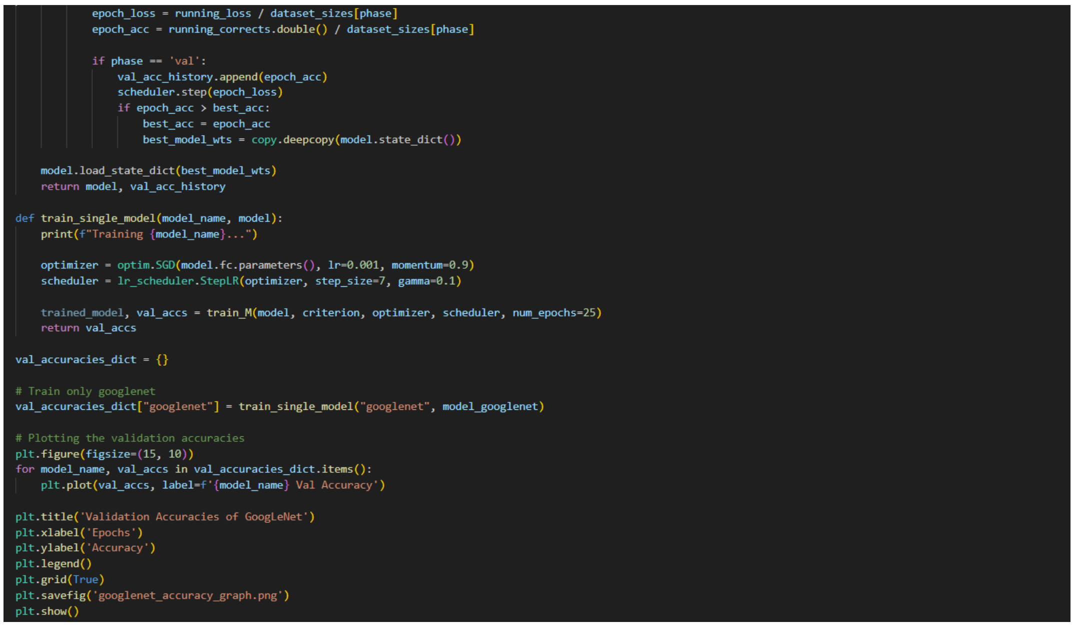
APPENDIX A: Coding for Model Training



```python
import time
import os
import torch
import torchvision
import torch.optim as optim
import torch.nn as nn
from torch.optim import lr_scheduler
from torchvision import datasets, models, transforms
import numpy as np
import copy
import matplotlib.pyplot as plt

# Load data and apply transformations
transformer = {
    'train': transforms.Compose([
        transforms.RandomResizedCrop(224),
        transforms.RandomHorizontalFlip(),
        transforms.ToTensor(),
        transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
    ]),
    'val': transforms.Compose([
        transforms.Resize(256),
        transforms.CenterCrop(224),
        transforms.ToTensor(),
        transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
    ])
}

datafolder = ['train', 'val']
path = 'Dataset'
image_datasets = {img: datasets.ImageFolder(os.path.join(path, img), transform=transformer[img]) for img in datafolder}
dataloaders = {img: torch.utils.data.DataLoader(image_datasets[img], batch_size=4, shuffle=True, num_workers=0) for img in datafolder}
dataset_sizes = {img: len(image_datasets[img]) for img in datafolder}
class_names = image_datasets['train'].classes

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

# Load pre-trained models
model_resnet = models.resnet18(pretrained=True).to(device)
model_vgg16 = models.vgg16(pretrained=True).to(device)
model_alexnet = models.alexnet(pretrained=True).to(device)
model_googlenet = models.googlenet(pretrained=True).to(device)
```

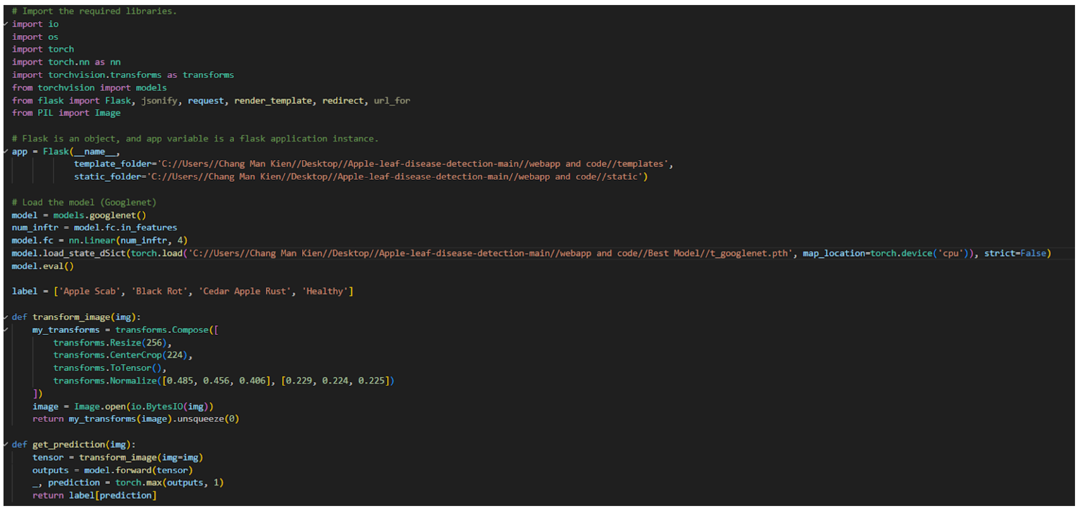


```python
# Modify the final layers to match the number of classes in your dataset
model_resnet.fc = nn.Linear(model_resnet.fc.in_features, len(class_names)).to(device)
model_vgg16.classifier[6] = nn.Linear(model_vgg16.classifier[6].in_features, len(class_names)).to(device)
model_alexnet.classifier[6] = nn.Linear(model_alexnet.classifier[6].in_features, len(class_names)).to(device)
model_googlenet.fc = nn.Linear(model_googlenet.fc.in_features, len(class_names)).to(device)

criterion = nn.CrossEntropyLoss()

def train_M(model, criterion, optimizer, scheduler, num_epochs=25):
    best_model_wts = copy.deepcopy(model.state_dict())
    best_acc = 0.0
    val_acc_history = []

    for epoch in range(num_epochs):
        for phase in ['train', 'val']:
            if phase == 'train':
                model.train()
            else:
                model.eval()

            running_loss = 0.0
            running_corrects = 0

            for inputs, labels in dataloaders[phase]:
                inputs = inputs.to(device)
                labels = labels.to(device)

                optimizer.zero_grad()

                with torch.set_grad_enabled(phase == 'train'):
                    outputs = model(inputs)
                    _, preds = torch.max(outputs, 1)
                    loss = criterion(outputs, labels)

                    if phase == 'train':
                        loss.backward()
                        optimizer.step()

                running_loss += loss.item() * inputs.size(0)
                running_corrects += torch.sum(preds == labels.data)
```

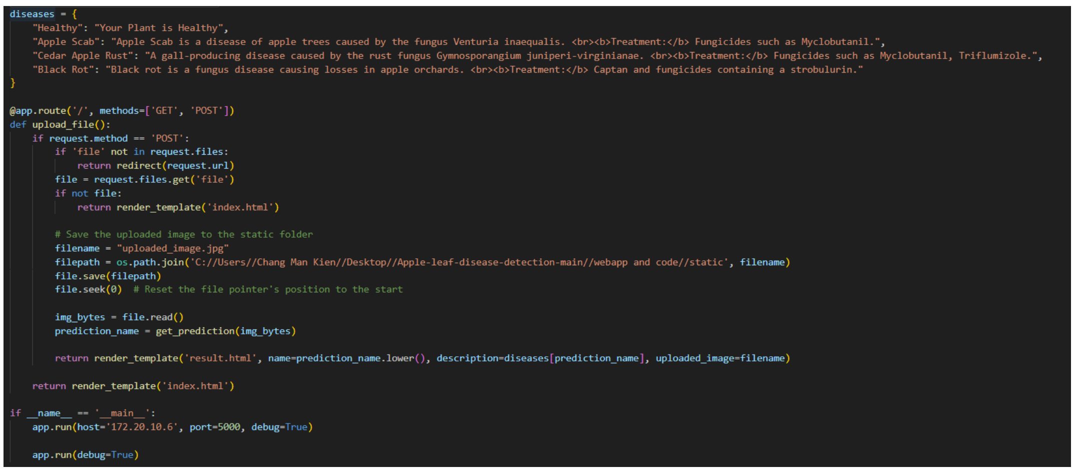


```
            epoch_loss = running_loss / dataset_sizes[phase]
            epoch_acc = running_corrects.double() / dataset_sizes[phase]

            if phase == 'val':
                val_acc_history.append(epoch_acc)
                scheduler.step(epoch_loss)
                if epoch_acc > best_acc:
                    best_acc = epoch_acc
                    best_model_wts = copy.deepcopy(model.state_dict())

    model.load_state_dict(best_model_wts)
    return model, val_acc_history

def train_single_model(model_name, model):
    print(f"Training {model_name}...")

    optimizer = optim.SGD(model.fc.parameters(), lr=0.001, momentum=0.9)
    scheduler = lr_scheduler.StepLR(optimizer, step_size=7, gamma=0.1)

    trained_model, val_accs = train_M(model, criterion, optimizer, scheduler, num_epochs=25)
    return val_accs

val_accuracies_dict = {}

# Train only googlenet
val_accuracies_dict["googlenet"] = train_single_model("googlenet", model_googlenet)

# Plotting the validation accuracies
plt.figure(figsize=(15, 10))
for model_name, val_accs in val_accuracies_dict.items():
    plt.plot(val_accs, label=f'{model_name} Val Accuracy')

plt.title('Validation Accuracies of GoogLeNet')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.grid(True)
plt.savefig('googlenet_accuracy_graph.png')
plt.show()
```

APPENDIX B: Coding for WebApp

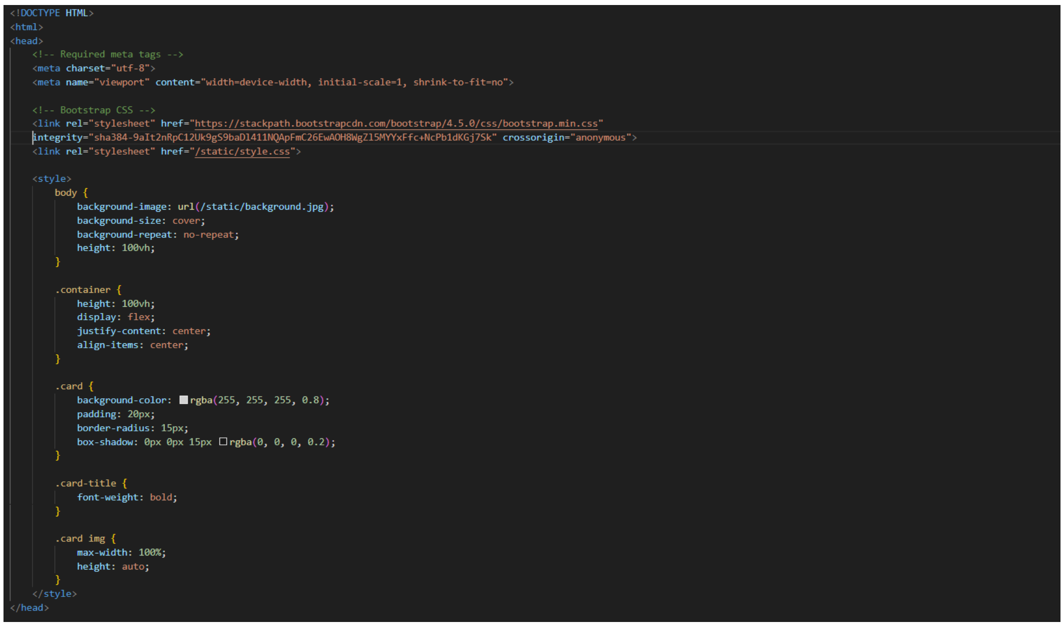


```python
# Import the required libraries.
import io
import os
import torch
import torch.nn as nn
import torchvision.transforms as transforms
from torchvision import models
from flask import Flask, jsonify, request, render_template, redirect, url_for
from PIL import Image

# Flask is an object, and app variable is a flask application instance.
app = Flask(__name__,
            template_folder='C://Users//Chang Man Kien//Desktop//Apple-leaf-disease-detection-main//webapp and code//templates',
            static_folder='C://Users//Chang Man Kien//Desktop//Apple-leaf-disease-detection-main//webapp and code//static')

# Load the model (Googlenet)
model = models.googlenet()
num_inftr = model.fc.in_features
model.fc = nn.Linear(num_inftr, 4)
model.load_state_dSict(torch.load('C://Users//Chang Man Kien//Desktop//Apple-leaf-disease-detection-main//webapp and code//Best Model//t_googlenet.pth', map_location=torch.device('cpu')), strict=False)
model.eval()

label = ['Apple Scab', 'Black Rot', 'Cedar Apple Rust', 'Healthy']

def transform_image(img):
    my_transforms = transforms.Compose([
        transforms.Resize(256),
        transforms.CenterCrop(224),
        transforms.ToTensor(),
        transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
    ])
    image = Image.open(io.BytesIO(img))
    return my_transforms(image).unsqueeze(0)

def get_prediction(img):
    tensor = transform_image(img=img)
    outputs = model.forward(tensor)
    _, prediction = torch.max(outputs, 1)
    return label[prediction]
```

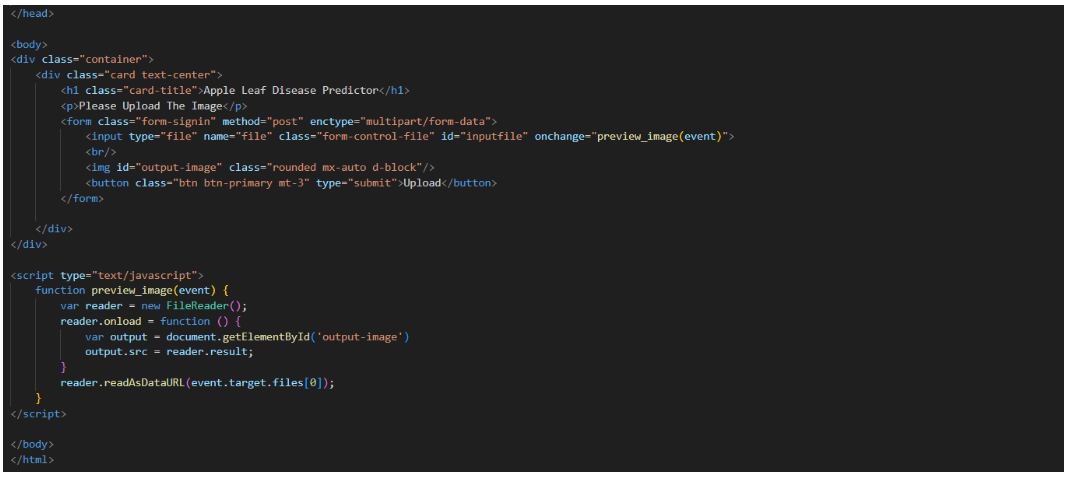


```python
diseases = {
    "Healthy": "Your Plant is Healthy",
    "Apple Scab": "Apple Scab is a disease of apple trees caused by the fungus Venturia inaequalis. <br><b>Treatment:</b> Fungicides such as Myclobutanil.",
    "Cedar Apple Rust": "A gall-producing disease caused by the rust fungus Gymnosporangium juniperi-virginianae. <br><b>Treatment:</b> Fungicides such as Myclobutanil, Triflumizole.",
    "Black Rot": "Black rot is a fungus disease causing losses in apple orchards. <br><b>Treatment:</b> Captan and fungicides containing a strobulurin."
}

@app.route('/', methods=['GET', 'POST'])
def upload_file():
    if request.method == 'POST':
        if 'file' not in request.files:
            return redirect(request.url)
        file = request.files.get('file')
        if not file:
            return render_template('index.html')

        # Save the uploaded image to the static folder
        filename = "uploaded_image.jpg"
        filepath = os.path.join('C://Users//Chang Man Kien//Desktop//Apple-leaf-disease-detection-main//webapp and code//static', filename)
        file.save(filepath)
        file.seek(0)  # Reset the file pointer's position to the start

        img_bytes = file.read()
        prediction_name = get_prediction(img_bytes)

        return render_template('result.html', name=prediction_name.lower(), description=diseases[prediction_name], uploaded_image=filename)

    return render_template('index.html')

if __name__ == '__main__':
    app.run(host='172.20.10.6', port=5000, debug=True)

    app.run(debug=True)
```

APPENDIX C:Coding for Upload.html

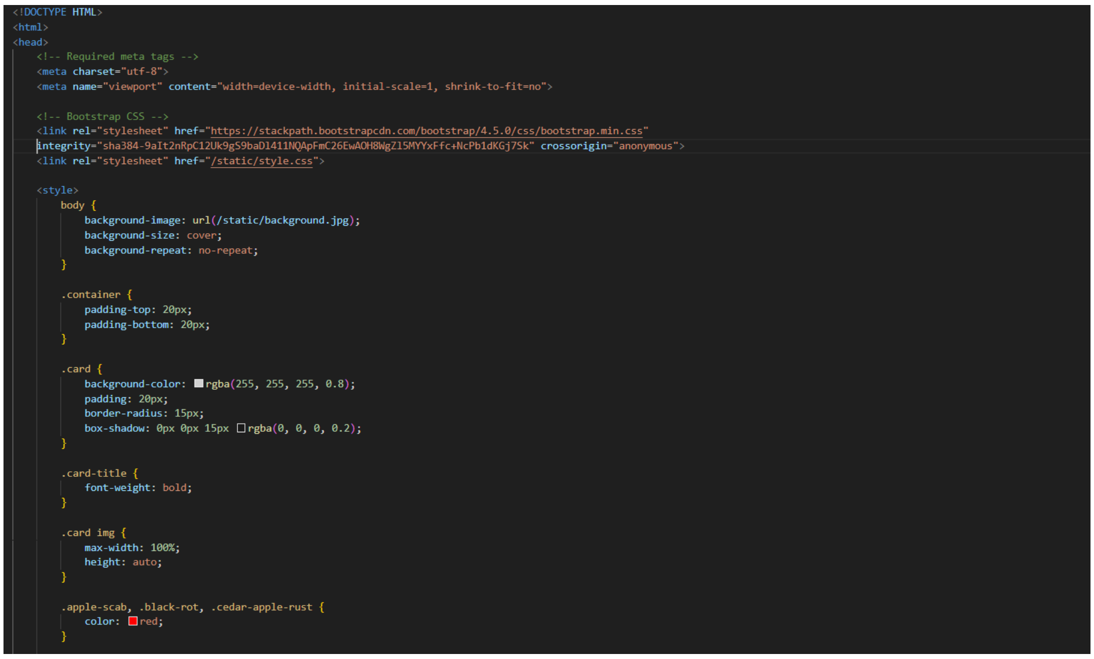


```
<!DOCTYPE HTML>
<html>
<head>
    <!-- Required meta tags -->
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">

    <!-- Bootstrap CSS -->
    <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.0/css/bootstrap.min.css"
    integrity="sha384-9aIt2nRpC12Uk9gS9baDl411NQApFmC26EwAOH8WgZl5MYYxFfc+NcPb1dKGj7Sk" crossorigin="anonymous">
    <link rel="stylesheet" href="/static/style.css">

    <style>
        body {
            background-image: url(/static/background.jpg);
            background-size: cover;
            background-repeat: no-repeat;
            height: 100vh;
        }

        .container {
            height: 100vh;
            display: flex;
            justify-content: center;
            align-items: center;
        }

        .card {
            background-color: rgba(255, 255, 255, 0.8);
            padding: 20px;
            border-radius: 15px;
            box-shadow: 0px 0px 15px rgba(0, 0, 0, 0.2);
        }

        .card-title {
            font-weight: bold;
        }

        .card img {
            max-width: 100%;
            height: auto;
        }
    </style>
</head>
```

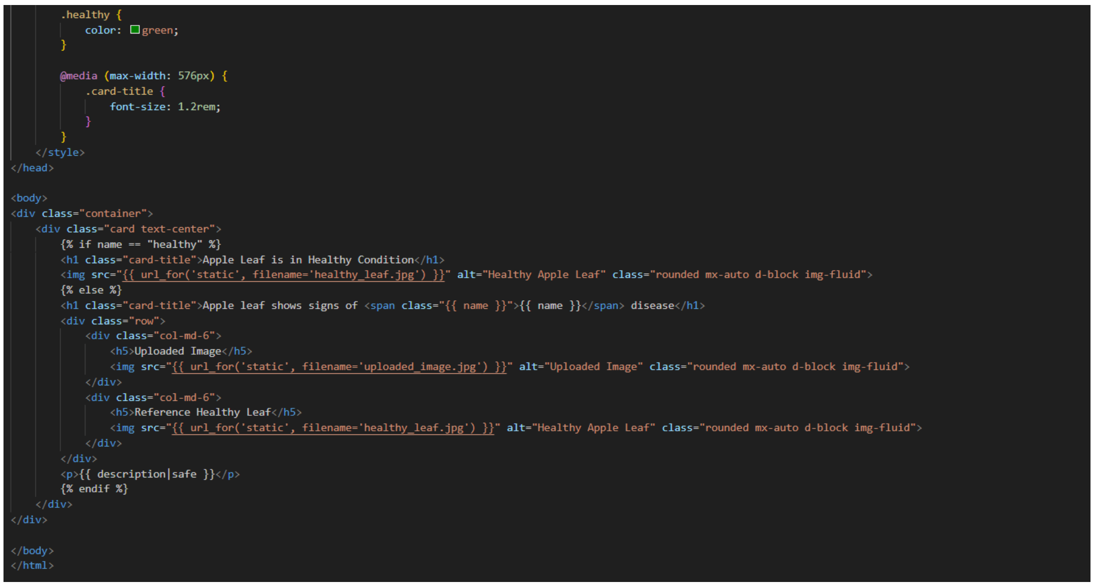


```
</head>

<body>
<div class="container">
    <div class="card text-center">
        <h1 class="card-title">Apple Leaf Disease Predictor</h1>
        <p>Please Upload The Image</p>
        <form class="form-signin" method="post" enctype="multipart/form-data">
            <input type="file" name="file" class="form-control-file" id="inputfile" onchange="preview_image(event)">
            <br/>
            <img id="output-image" class="rounded mx-auto d-block"/>
            <button class="btn btn-primary mt-3" type="submit">Upload</button>
        </form>

    </div>
</div>

<script type="text/javascript">
    function preview_image(event) {
        var reader = new FileReader();
        reader.onload = function () {
            var output = document.getElementById('output-image')
            output.src = reader.result;
        }
        reader.readAsDataURL(event.target.files[0]);
    }
</script>

</body>
</html>
```

APPENDIX D: Coding for Result.html



```html
<!DOCTYPE HTML>
<html>
<head>
    <!-- Required meta tags -->
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">

    <!-- Bootstrap CSS -->
    <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.0/css/bootstrap.min.css"
    integrity="sha384-9aIt2nRpC12Uk9gS9baDl411NQApFmC26EwAOH8WgZl5MYYxFfc+NcPb1dKGj7Sk" crossorigin="anonymous">
    <link rel="stylesheet" href="/static/style.css">

    <style>
        body {
            background-image: url(/static/background.jpg);
            background-size: cover;
            background-repeat: no-repeat;
        }

        .container {
            padding-top: 20px;
            padding-bottom: 20px;
        }

        .card {
            background-color: rgba(255, 255, 255, 0.8);
            padding: 20px;
            border-radius: 15px;
            box-shadow: 0px 0px 15px rgba(0, 0, 0, 0.2);
        }

        .card-title {
            font-weight: bold;
        }

        .card img {
            max-width: 100%;
            height: auto;
        }

        .apple-scab, .black-rot, .cedar-apple-rust {
            color: red;
        }
```



```html
        .healthy {
            color: green;
        }

        @media (max-width: 576px) {
            .card-title {
                font-size: 1.2rem;
            }
        }
    </style>
</head>

<body>
<div class="container">
    <div class="card text-center">
        {% if name == "healthy" %}
        <h1 class="card-title">Apple Leaf is in Healthy Condition</h1>
        <img src="{{ url_for('static', filename='healthy_leaf.jpg') }}" alt="Healthy Apple Leaf" class="rounded mx-auto d-block img-fluid">
        {% else %}
        <h1 class="card-title">Apple leaf shows signs of <span class="{{ name }}">{{ name }}</span> disease</h1>
        <div class="row">
            <div class="col-md-6">
                <h5>Uploaded Image</h5>
                <img src="{{ url_for('static', filename='uploaded_image.jpg') }}" alt="Uploaded Image" class="rounded mx-auto d-block img-fluid">
            </div>
            <div class="col-md-6">
                <h5>Reference Healthy Leaf</h5>
                <img src="{{ url_for('static', filename='healthy_leaf.jpg') }}" alt="Healthy Apple Leaf" class="rounded mx-auto d-block img-fluid">
            </div>
        </div>
        <p>{{ description|safe }}</p>
        {% endif %}
    </div>
</div>

</body>
</html>
```