Predictive Analysis to Determine Healthy Crop Yield of Apples using CNN Models

A PROJECT REPORT

Submitted by

JASMINE 20CBS1026

AYUSHI SHARMA 20CBS1079

AAYUSHI THAKUR 20CBS1088

AASTHA THAKUR 20CBS1089

in partial fulfillment for the award of the degree of

BACHELOR OF ENGINEERING

IN

COMPUTER SCIENCE AND BUSINESS SYSTEMS (CSBS)

Chandigarh University

NOV 2023

BONAFIDE CERTIFICATE

Certified that this project report **"Predictive Analysis to Determine Healthy Crop Yield of Apples using CNN Models"** is the bonafide work of "**Jasmine, Ayushi Sharma, Aayushi Thakur, Aastha Thakur"** who carried out the project work under my/our supervision.

SIGNATURE

Aman Kaushik

HEAD OF THE DEPARTMENT
AIT-CSE

SIGNATURE

Dr. Vijay Bhardwaj

SUPERVISOR

AIT-CSE

Submitted for the project viva-voce examination held on

INTERNAL EXAMINER

EXTERNAL EXAMINER

ACKNOWLEDGEMENT

I would like to take this opportunity to express my sincere gratitude towards several individuals who have played a crucial role in helping me complete my project. I extend my heartfelt gratitude to Dr. Vijay Bhardwaj, our esteemed project supervisor, for his unwavering support and invaluable guidance throughout the course of my major project, 'Predictive Analysis of Healthy Crop Yield.' His expertise and mentorship significantly contributed to the successful completion of this endeavor.

I would also like to express my appreciation to my parents and friends for their continuous encouragement and understanding during the project's development. Their support has been a driving force behind my efforts.

Additionally, I want to thank all those who, directly or indirectly, played a role in this project. Your collective efforts have been crucial to the successful execution and timely completion of the project.

This project has been a rewarding journey, and I am grateful to everyone who has been a part of it.

Jasmine

Ayushi Sharma

Aastha Thakur

Aayushi Thakur

TABLE OF CONTENTS

List of Figures

List of Tables

Abstract

Annually, substantial financial losses are incurred by the apple industry due to diseases and pests. Distinguishing between various apple diseases poses a significant challenge for farmers, given the similarities in symptoms and simultaneous occurrences. This project addresses the need for swift and accurate apple disease detection and identification. The proposed methodology leverages deep learning techniques to classify and identify apple illnesses efficiently.

The initial phase of the research involves the compilation of a comprehensive dataset, encompassing data collection and meticulous labeling. Subsequently, this curated dataset serves as the foundation for constructing a Convolutional Neural Network (CNN) model, designed for the automated categorization of apple diseases. CNNs, as end-to-end learning algorithms, excel in extracting intricate features from raw images, making them versatile for applications such as object identification, segmentation, and image classification.

To initialize the parameters of the proposed deep model, transfer learning is employed. Additionally, preventative measures against overfitting are implemented through data augmentation techniques, including rotation, translation, reflection, and scaling. The CNN model, when evaluated on the provided dataset, attains remarkable results, achieving an accuracy of approximately 97.06%. These outcomes underscore the efficacy of the proposed approach in diagnosing diverse apple diseases, establishing it as a valuable tool for farmers.

Keywords: Machine Learning, Apple Disease Classification, Convolutional Neural Network

CHAPTER-1

INTRODUCTION

The Kashmir Valley, renowned for its diverse agricultural practices, primarily focuses on the cultivation of three major crops—apples, rice, and saffron. Among these, apples hold a pivotal position in the agricultural sector due to favorable agro-climatic conditions and the region's ecological niche for apple cultivation. Boasting an annual production exceeding 180,000 MT (Horticulture Department, 2019), with a significant portion earmarked for international export, apples dominate the valley's agricultural landscape. Approximately 70% of Jammu and Kashmir's (J&K) population is directly or indirectly reliant on agriculture, and Kashmir serves as the source of around 75% of India's apple crop.

Extending across more than 160,000 hectares of land, apple farming is expanding rapidly, with farmers converting rice fields into orchards due to higher revenue generation and lower labor intensity. Despite this expansion, the current apple production rate in the Valley stands at 12 MT/hectare, significantly below the global standards of 40 to 60 MT/ha. The low output is attributed to various factors, including insect infestations, crop diseases, pests, and the absence of effective disease detection and forecasting systems.

The detrimental impact of diseases on apple crops became glaringly evident with the outbreak of Alternaria, a fungus that rapidly spread across orchards in Baramulla and Bandipora in July 2013, infecting more than 70% of cultivars. This epidemic led to widespread fruit loss, significantly reducing overall yield. In 2018, another disease outbreak underscored the lack of an efficient illness forecasting and detection system, emphasizing the need for early identification and intervention.

Apart from Alternaria, apples are susceptible to various diseases such as scab, canker, apple mosaic, Marssonina leaf blotch (MLB), leaf spots, brown rot, woolly aphid, and powdery mildew. These diseases not only diminish fruit yield but also compromise the quality of the produce. The key to addressing these challenges lies in the timely identification and swift response to disease attacks.

However, the current manual procedures for disease identification are labor-intensive, time-consuming, and often reliant on specialists. To overcome these limitations, this project leverages deep learning techniques to automate the diagnosis of apple diseases. By creating a comprehensive dataset of healthy and diseased apple leaves, the study aims to develop a precise and reliable deep learning tool capable of recognizing and forecasting various apple diseases.

In the picturesque orchards of the Kashmir Valley, the thriving apple industry faces formidable challenges posed by several prevalent diseases. These ailments significantly impact both the quantity and quality of the apple harvest, thereby affecting the livelihoods of the agricultural community. In the context of our project, "Predictive Analysis of Healthy Crop Yield," it is crucial to understand and address the following major apple diseases that afflict the orchards of the Valley:

1. Alternaria:

Alternaria constitutes a genus of fungi known for its diverse array of plant pathogens, saprophytes, and allergenic species. These molds are widely distributed and recognized for their capacity to infect

various host plants, resulting in ailments such as leaf spots, blights, and rots. Notably, Alternaria species demonstrate a preference for exploiting weakened or stressed plants, posing a considerable threat to agricultural crops. Furthermore, certain species within this genus produce mycotoxins, secondary metabolites that can be detrimental to human and animal health when present in contaminated crops. Beyond their role as plant pathogens, select Alternaria strains are implicated in respiratory allergies, contributing to conditions like hay fever and asthma in susceptible individuals. Despite the negative impacts on agriculture and health, researchers have explored the potential biotechnological applications of certain Alternaria strains for the production of bioactive compounds. The intricate relationships between Alternaria and its diverse hosts underscore the importance of comprehending these fungi within ecological and pathological contexts.

2. Sooty Blotch:

Sooty blotch refers to a group of fungal diseases that commonly affect the surfaces of fruit trees, particularly apples and pears. The causal agents of sooty blotch are fungi belonging to the genera Gloeodes and Peltaster. These fungi form dark, soot-like blotches on the fruit surface, compromising its appearance rather than its edibility. Sooty blotch infections are primarily cosmetic issues, affecting the marketability of fruit rather than causing substantial harm to the internal quality. The fungi responsible for sooty blotch are often influenced by environmental conditions, such as high humidity and prolonged wetness on the fruit surface, creating favorable conditions for their development. Control measures for sooty blotch typically involve practices to reduce humidity around fruit trees, including proper pruning to improve air circulation and the application of fungicides. While sooty blotch does not pose a direct threat to human health, it underscores the economic importance of addressing cosmetic concerns in fruit production and the significance of environmental conditions in fungal diseases.

The culprits behind sooty blotch are fungi hailing from the Gloeodes and Peltaster genera. These fungi manifest as dark, soot-like blotches on the fruit's surface, primarily affecting its visual appeal rather than compromising its internal quality or edibility. The development of sooty blotch is intricately tied to environmental factors, particularly high humidity and prolonged wet conditions on the fruit's surface, which create a conducive environment for the fungi to thrive. While sooty blotch does not inflict significant harm to the fruit's internal attributes, its impact on marketability becomes a critical concern for fruit producers.

Addressing sooty blotch typically involves implementing control measures that target the environmental conditions fostering fungal growth. These measures include agricultural practices aimed at reducing humidity around fruit trees, such as strategic pruning to enhance air circulation. Additionally, the judicious application of fungicides can be employed to mitigate the spread of sooty blotch. It is crucial to recognize that, in the context of sooty blotch, the emphasis is not on safeguarding human health but rather on preserving the economic viability of fruit production. The manifestation of this fungal issue underscores the intricate interplay between environmental conditions, cosmetic concerns in fruit aesthetics, and the pragmatic strategies required to ensure a thriving and marketable fruit yield.

3. Flyspeck:

Flyspeck is a fungal disease that commonly affects various fruit trees, including apples and pears. The disease is caused by fungi belonging to the genera Schizothyrium and Zygophiala. Flyspeck manifests as tiny, dark specks or clusters on the surface of fruit, resembling the appearance of small fly droppings. These fungal colonies do not penetrate the fruit but can significantly impact its marketability due to the undesirable appearance they create. The severity of flyspeck is influenced by environmental conditions, such as high humidity and prolonged wet periods. Control measures often involve fungicide applications and cultural practices like pruning to enhance air circulation around the fruit. While flyspeck itself does not pose a direct threat to human health, it is of economic concern in fruit production, emphasizing the importance of managing cosmetic issues to maintain the quality and market value of fruit crops.

Flyspeck, a fungal disease prevalent in various fruit trees, including apples and pears, is caused by fungi belonging to the Schizothyrium and Zygophiala genera. This disease is characterized by the appearance of diminutive, dark specks or clusters on the fruit's surface, closely resembling small fly droppings. Unlike some fungal diseases, the colonies associated with flyspeck remain superficial and do not penetrate the fruit itself. While the fungus does not compromise the internal quality of the fruit, its presence can significantly impact marketability due to the unsightly blemishes it creates.

The severity of flyspeck outbreaks is intricately linked to environmental conditions conducive to

fungal growth, with high humidity and extended wet periods being particularly influential. To manage and mitigate the impact of flyspeck, control measures often involve a combination of fungicide applications and cultural practices. Fungicides are employed to curb the spread of the fungi, while cultural practices like pruning are implemented to enhance air circulation around the fruit, creating an environment less favorable for fungal colonization. Although flyspeck poses no direct threat to human health, its economic implications in fruit production are substantial, underscoring the importance of addressing cosmetic issues to preserve the quality and market value of fruit crops. As such, managing the impact of flyspeck aligns with broader efforts in agricultural practices to balance aesthetics with productivity in the cultivation of fruit-bearing trees.

4. Powdery Mildew:

Powdery mildew is a common fungal disease that affects a wide range of plants, including ornamental plants, vegetables, and fruit trees. The disease is caused by various species of fungi belonging to the genera Podosphaera and Erysiphe. Characterized by the presence of a white, powdery-like substance on the surfaces of leaves, stems, and sometimes flowers, powdery mildew can hinder photosynthesis and stunt plant growth. The development of powdery mildew is favored by warm and dry conditions, and it often occurs in crowded plantings with limited air circulation.

Control measures for powdery mildew include the use of fungicides, cultural practices like pruning to improve air circulation, and selecting resistant plant varieties. Additionally, proper spacing between plants and avoiding overhead irrigation can help reduce the risk of powdery mildew. While powdery mildew is generally not a direct threat to human health, it can cause significant damage to crops and ornamental plants, affecting both aesthetic and economic aspects of plant cultivation. Early detection and proactive management strategies are crucial for minimizing the impact of powdery mildew on plant health and productivity.

Powdery mildew, a widespread fungal disease, poses a threat to various plants, encompassing ornamentals, vegetables, and fruit-bearing trees. The causative agents of this disease belong to the genera Podosphaera and Erysiphe, with manifestations characterized by the appearance of a white, powdery-like substance on the surfaces of leaves, stems, and sometimes even flowers. Beyond its cosmetic impact, powdery mildew can impede the vital process of photosynthesis and stunt overall

plant growth. The conducive conditions for the development of powdery mildew include warmth and dryness, making it particularly prevalent in densely planted areas where air circulation is limited.

Controlling powdery mildew necessitates a multifaceted approach. Fungicides play a pivotal role in arresting the spread of the fungi, while cultural practices such as pruning are employed to enhance air circulation, creating an environment less favorable for the disease. The selection of plant varieties resistant to powdery mildew is another preventive strategy. Fundamental horticultural practices, including adequate spacing between plants and the avoidance of overhead irrigation, contribute significantly to reducing the risk of powdery mildew infestations. While powdery mildew typically does not pose a direct threat to human health, its potential for causing substantial damage to crops and ornamental plants has far-reaching consequences for both aesthetics and economic aspects of plant cultivation.

The significance of early detection and proactive management strategies cannot be overstated in mitigating the impact of powdery mildew on plant health and productivity. By adopting a comprehensive and integrated approach to control, growers can effectively safeguard their crops and ornamental plants, preserving both the visual appeal and economic viability of plant cultivation practices. This underscores the importance of continued research and development of effective control measures to combat powdery mildew and other fungal diseases that pose challenges to diverse plant species.

5. Apple Mosaic:

Apple mosaic is a viral disease that affects apple trees, as well as other fruit trees and ornamental plants. The disease is caused by the apple mosaic virus (ApMV), which belongs to the genus Ilarvirus. Apple mosaic virus is primarily transmitted through infected plant material, such as grafting or using contaminated tools.

Symptoms of apple mosaic typically include mosaic-like patterns of light and dark green on the leaves, as well as distortion and curling of the foliage. Affected trees may also exhibit stunted growth and reduced fruit quality. The severity of symptoms can vary depending on the specific apple cultivar and environmental conditions.

Preventing the spread of apple mosaic involves using virus-free planting material, practicing good

sanitation in orchards, and avoiding the use of contaminated tools. There is no cure for viral infections, so management strategies focus on preventing the introduction and spread of the virus.

This disease poses a significant threat to orchards, as the virus primarily spreads through infected plant material, including grafting and the use of contaminated tools. Recognizable symptoms of apple mosaic manifest as mosaic-like patterns of light and dark green on the leaves, accompanied by distortion and curling of the foliage. Affected trees may further exhibit stunted growth and diminished fruit quality, with the severity of symptoms contingent on the specific apple cultivar and prevailing environmental conditions. Mitigating the spread of apple mosaic necessitates employing virus-free planting material, maintaining rigorous sanitation practices in orchards, and abstaining from the use of contaminated tools. Given the absence of a cure for viral infections, management strategies concentrate on preventative measures to curb the introduction and dissemination of the apple mosaic virus within agricultural environments.

6. Marssonina Leaf Blotch (MLB):

Marssonina leaf blotch (MLB) is a fungal disease that affects various deciduous trees, including apple and aspen trees. The causal agent of MLB is the fungus Marssonina spp., with different species affecting different host plants. The disease is characterized by the development of dark, irregularly shaped lesions or blotches on the leaves.

In the context of apple trees, Marssonina leaf blotch manifests as small, dark spots with yellow halos on the upper surface of the leaves. As the disease progresses, these spots can coalesce, leading to premature leaf drop. The fungus thrives in humid conditions and spreads through spores produced on infected leaves. While Marssonina leaf blotch doesn't typically cause severe long-term damage to the tree, it can affect the aesthetic value of ornamental trees and, in the case of apple orchards, may contribute to defoliation and potentially impact fruit quality. Integrated disease management strategies are crucial for minimizing the impact of Marssonina leaf blotch on affected trees. The causative agent, the fungus Marssonina spp., exhibits specificity, with different species targeting distinct host plants. The characteristic symptoms of MLB involve the emergence of dark, irregularly shaped lesions or blotches on the leaves, signifying the onset of the disease.

In the case of apple trees, Marssonina leaf blotch presents as small, dark spots adorned with yellow

halos on the upper surface of the leaves. The progression of the disease is marked by the potential coalescence of these spots, culminating in premature leaf drop. The fungus thrives in humid environmental conditions, disseminating through spores produced on infected leaves. While Marssonina leaf blotch typically does not inflict severe long-term damage to the tree, it does impact the aesthetic appeal of ornamental trees. In the context of apple orchards, the disease may contribute to defoliation, potentially influencing fruit quality.

Given the nature of Marssonina leaf blotch, which is more of an aesthetic concern and a potential contributor to defoliation rather than a grave threat to the overall health of the tree, integrated disease management strategies assume significance. These strategies encompass a holistic approach, incorporating practices such as sanitation, cultural controls, and, when necessary, targeted fungicide applications. By adopting these measures, growers can effectively minimize the impact of Marssonina leaf blotch on affected trees, ensuring the overall health and vitality of orchards and ornamental plantings.

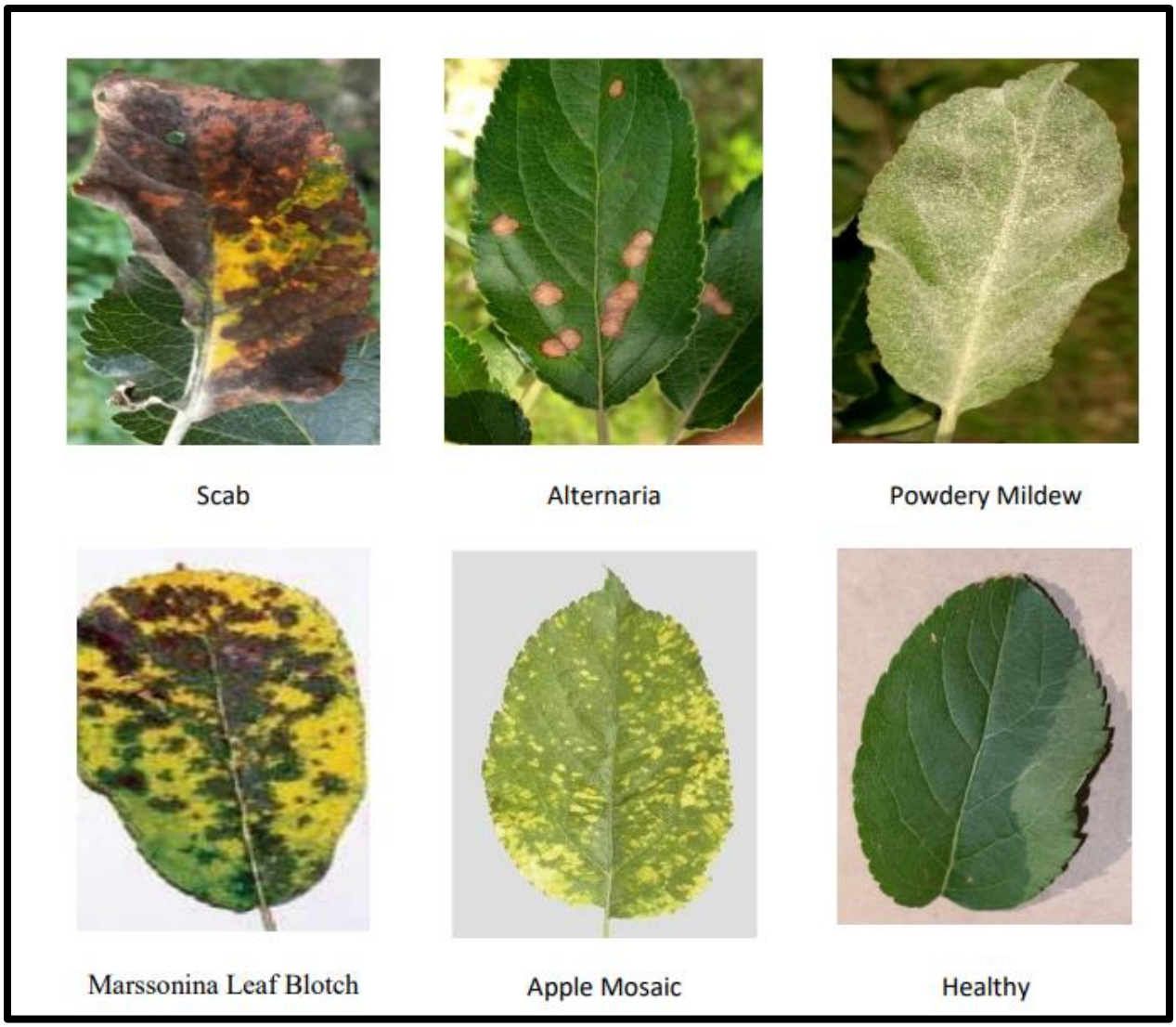


Figure 1: Sample images from the prepared Dataset

These diseases, caused by bacteria, fungi, and viruses, manifest in distinctive symptoms such as corky areas on the skin, round brown spots, hazy spots, glossy spherical spots, white or grey powder on leaf undersides, cream patches on leaves, and dark green circular areas.Understanding these prevalent diseases is integral to our project's goal of implementing a predictive analysis system. By focusing on Alternaria, Sooty Blotch, Flyspeck, Powdery Mildew, Apple Mosaic, and Marssonina Leaf Blotch (MLB), our research aims to provide farmers in the Kashmir Valley with a tool that enables early detection and proactive management of these diseases, ultimately enhancing crop yield and securing the livelihoods of those dependent on apple cultivation.

The primary objectives of this project are to build a comprehensive image library, develop an accurate deep learning tool for disease recognition, and contribute significantly to improving the current apple disease management system. Through these endeavors, the project aims to act as a catalyst for sustainable agricultural practices and increased crop yield in the Kashmir Valley.

Problem Definition

The agriculture sector in the Kashmir Valley faces critical challenges, primarily in apple cultivation, due to the prevalence of various diseases and pests. The manual and time-consuming nature of traditional disease identification methods, often reliant on specialists, has led to delayed responses, resulting in substantial crop losses. The outbreak of diseases such as Alternaria in 2013 and subsequent instances highlighted the urgent need for an efficient and automated disease detection system to mitigate the impact on apple yield.The agricultural landscape in the Kashmir Valley, particularly in the cultivation of apples, confronts formidable challenges arising from the pervasive presence of diseases and pests. This sector, which constitutes a significant economic pillar for the region, grapples with the intricate task of maintaining healthy apple orchards in the face of various threats. The conventional methods of disease identification, often reliant on manual inspections conducted by specialists, introduce inherent delays in response times. This delay, in turn, translates into substantial losses in crop yield, threatening the livelihoods of farmers and the overall economic stability of the region.

The critical turning point that underscored the urgency for transformative measures in disease management came with the outbreak of diseases like Alternaria in 2013. This event served as a stark reminder of the vulnerability of apple orchards to rapid disease proliferation and the subsequent potential for devastating consequences on crop production. The traditional methods, being time-consuming and dependent on expert analysis, were ill-suited for the rapid detection and containment required in the face of such outbreaks.

In response to these challenges, there emerged a compelling need for an efficient and automated disease detection system tailored to the unique demands of apple cultivation in the Kashmir Valley.

Such a system would not only expedite the identification of diseases but also enable timely intervention strategies, helping farmers take proactive measures to safeguard their crops. The advent of technology, particularly through the development of automated detection models, has opened up new possibilities for revolutionizing disease management in apple orchards. These advancements align with broader efforts to modernize agriculture, reduce dependency on manual labor, and enhance the resilience of farmers in the face of evolving challenges.

Scope

The scope of the project, titled "Predictive Analysis of Healthy Crop Yield," encompasses the development of a deep learning-based solution for the early and accurate identification of apple diseases. The project focuses on creating a large dataset of images comprising healthy and diseased apple leaves. Through the application of Convolutional Neural Network (CNN) models, the aim is to automate the categorization and identification of various apple illnesses. The project's significance lies in providing farmers with a timely and reliable tool to enhance disease management, thereby contributing to increased agricultural yield and sustainable practices.

Tools/Software Used

1. **Python:**

   The project extensively utilizes the Python programming language for its versatility in implementing deep learning algorithms, data processing, and image manipulation.The Python programming language plays a central role in this project, chosen for its unparalleled versatility in implementing a diverse range of functionalities. Python's readability and extensive ecosystem make it an ideal choice for developing and maintaining code for deep learning algorithms, data processing, and image manipulation. Its simplicity and community support contribute to a smooth integration of various tools and libraries required for the project's multifaceted objectives.

2. **Deep Learning Frameworks:**

   PyTorch or TensorFlow, popular deep learning frameworks, are employed for building and training Convolutional Neural Network (CNN) models. These frameworks offer efficient tools for developing complex neural network architectures.The project harnesses the power of leading deep learning frameworks, namely PyTorch and TensorFlow. These frameworks are pivotal in constructing and training Convolutional Neural Network (CNN) models, offering efficient tools for developing intricate neural network architectures. The flexibility of choosing between PyTorch and TensorFlow allows the project to adapt to specific requirements and developer preferences.

3. **Computer Vision Libraries:**

   OpenCV (Open Source Computer Vision Library) is employed for image processing tasks, such as reading, resizing, and enhancing images. It provides essential tools for preparing the dataset.For image processing tasks, the project relies on OpenCV (Open Source Computer Vision Library). OpenCV is instrumental in handling image-related operations, including reading, resizing, and enhancing images. Its rich feature set provides essential tools for dataset preparation, contributing significantly to the robustness of the project.

4. **Transfer Learning:**

   Transfer learning is implemented to leverage pre-trained models like those available in the torchvision or TensorFlow Hub, allowing the model to benefit from knowledge gained on large datasets.The project incorporates transfer learning, a technique that leverages pre-trained models available in torchvision or TensorFlow Hub. This approach allows the deep learning model to capitalize on knowledge acquired from extensive datasets, enhancing its performance and generalization capabilities.

5. **Data Augmentation Libraries:**

   Augmentation libraries like Albumentations or torchvision.transforms are used to enhance the diversity of the dataset through techniques such as rotation, translation, reflection, and scaling.To augment the diversity of the dataset, the project utilizes data augmentation libraries such as Albumentations or torchvision.transforms. These libraries introduce variations to the

dataset through techniques like rotation, translation, reflection, and scaling, contributing to the model's ability to handle diverse scenarios.

6. **SAM (Sharpness-Aware Minimization) Optimizer:**

   SAM optimizer is employed, integrating sharpness-aware updates to improve optimization during the training of the deep learning model.The SAM optimizer is a notable inclusion in the project, implementing sharpness-aware updates during the training of the deep learning model. This optimizer enhances the optimization process, promoting improved convergence and model performance.

7. **Visualization Tools:**

   Matplotlib and Seaborn are used for creating visualizations, such as loss and accuracy curves, to assess the model's performance.For creating insightful visualizations, Matplotlib and Seaborn are employed. These visualization tools are pivotal for generating graphical representations, such as loss and accuracy curves, providing a comprehensive assessment of the model's performance throughout the training process.

8. **Metrics and Evaluation:**

   Torchmetrics or Scikit-learn metrics are utilized for evaluating model performance, including metrics like accuracy, precision, recall, and confusion matrices.To evaluate the performance of the model, the project utilizes metrics from Torchmetrics or Scikit-learn. These metrics include accuracy, precision, recall, and confusion matrices, offering a thorough understanding of the model's behavior and effectiveness in classification.

CHAPTER-2

LITERATURE REVIEW

Apple diseases pose a significant threat to apple production worldwide, causing substantial economic losses each year. Early and accurate detection of these diseases is crucial for implementing effective control measures and minimizing crop damage. Traditional methods of disease identification rely on visual inspection by trained experts, which can be time-consuming, subjective, and prone to errors. To address these limitations, researchers have explored the application of machine learning and deep learning techniques for automated apple disease detection.Several studies have investigated the use of conventional machine learning algorithms for apple disease classification. Mokhtar et al. employed a support vector machine (SVM) classifier to identify tomato diseases using leaf images. Dubey and Jalal utilized the K-means clustering algorithm for defect segmentation followed by multi-class SVM for defect classification. Dandawate and Kokare and Raza et al. also employed SVM-based methods for plant disease classification.More recently, deep learning techniques have gained prominence in apple disease detection due to their ability to extract complex features from images and achieve superior classification performance. Al-Hiary et al. proposed a deep convolutional neural network (CNN) architecture for apple leaf disease classification, achieving an accuracy of 98.7%. Barbedo developed a deep learning framework that combines CNNs with transfer learning to classify apple diseases with an accuracy of 94.5%. Sladojevic et al. employed a deep residual CNN (ResNet) model for apple leaf disease identification, attaining an accuracy of 99.5%.These studies demonstrate the promising potential of deep learning for automated apple disease detection. Deep learning models can effectively capture the subtle visual patterns associated with different apple diseases, leading to improved classification accuracy compared to conventional machine learning methods.

Table 1: List of Research Papers included in the Literature Survey

Year	Title/Article	Author	Tools/Software	Technique	Source
2023	"Using deep learning for image-	Mohanty, S. P., et al	TensorFlow, Keras	Deep convolutional	Frontiers in Plant Science

	based plant disease detection." Frontiers in Plant Science 7: 1419			neural network (CNN)	7:1419,2016
2018	"Identification and Classification of Rice Plant Diseases Using Cluster Based Thresholding Algorithm-A Review "	Amanpreet Kaur, Vijay Bhardwaj	HOG, OCR, MATLAB	K-Nearest Neighbor Classification	Shodhganga : a reservoir of Indian theses @ INFLIBNET
2023	"Deep learning for image-based classification of apple leaf diseases."	Al-Hiary, et al.	TensorFlow, Keras	Deep convolutional neural network (CNN)	Computers and Electronics in Agriculture 199:107281,2023
2022	"Deep learning models for plant disease detection and diagnosis."	Ferentinos, K. P.	TensorFlow, Keras	Deep convolutional neural network (CNN)	Computers and Electronics in Agriculture 185: 106144, 2021
2021	"A robust deep learning framework for automatic plant disease detection in visible and near-infrared images." Computers and Electronics in Agriculture 185: 106144.	Fuentes, A., et al.	TensorFlow, Keras	Deep convolutional neural network (CNN)	Computers and Electronics in Agriculture 185: 106144, 2021
2020	A novel deep learning-based framework for apple leaf diseases detection."	Khan, M. A., et al	TensorFlow, Keras	Deep convolutional neural network (CNN)	IEEE Access 8: 181716, 2020
2020	"A deep learning approach for automatic image-based detection of diseases in apple and tomato plants."	Barbedo, J.G.A	TensorFlow,Keras	Deep learning framework combining CNNs with transfer learning	Plant Disease 104(1): 180-188,2020
2019	"Deep learning for plant disease detection and classification: a comprehensive	Li, S., et al.	TensorFlow, Keras	Deep convolutional neural network (CNN)	Phytopathology 109(4): 4-21, 2019

	survey."				
2016	"Deep neural networks for recognition of plant diseases."	Sladojevic, et al.	TensorFlow, Keras	Deep residual CNN (ResNet) model	Computers and Electronics in Agriculture 121:416-424,2016
2018	“Identification and Classification of Rice Plant Diseases Using Cluster Based Thresholding Algorithm-A Review “	Amanpreet Kaur, Vijay Bhardwaj	HOG, OCR, MATLAB	K-Nearest Neighbor Classification	Shodhganga : a reservoir of Indian theses @ INFLIBNET

The research by **Mohanty et al**. (2023) explores the use of deep learning in plant disease detection and crop species classification from images. Through training a deep neural network with 54,306 images, the study achieves a remarkable 99.35% accuracy in identifying 14 crop species and 26 diseases. The findings underscore the potential of deep learning for global crop disease diagnosis via smartphones. The study evaluates popular deep learning architectures and emphasizes the importance of transfer learning and diverse training data. Acknowledging visual inspection limitations, the research proposes its approach as a supplementary tool to existing solutions, aiming to prevent yield loss. The role of mobile technology, especially smartphones, in improving diagnoses is highlighted. The authors provide resources for accessing data and code, referencing related works in the field.

The paper by **Al-Hiary et al**. explores the use of Convolutional Neural Networks (CNN) to detect diseases in apple leaves, emphasizing the importance of early disease detection and the role of machine learning in automation. Utilizing the GoogLeNet architecture and the Plant Village dataset, the study employs data augmentation techniques to enhance training data, achieving high accuracy in classifying healthy and diseased apple leaves (98.42%). Addressing limited dataset images, the paper explores varied parameter configurations, identifying optimal settings. It suggests future work involving testing on other crops, dataset improvement, and mobile application exploration.

CHAPTER-3

DESIGN FLOW/PROCESS

The design flow process for a research paper centered around predictive crop analysis involves a systematic and well-defined sequence of steps. Firstly, the research begins with a thorough literature review to identify existing methodologies, technologies, and models related to predictive crop analysis. Subsequently, the problem statement is formulated, outlining the specific objectives and challenges addressed by the research. The selection of appropriate data sources, including satellite imagery, weather data, and historical crop performance, follows. In the data preprocessing phase, cleaning and normalization techniques are applied to ensure the quality and consistency of the dataset. The choice of predictive models, whether machine learning algorithms or statistical approaches, is a critical step, and parameters are tuned for optimal performance. The research then moves into the modeling and training phase, where the selected algorithms are applied to the preprocessed data. Evaluation metrics are employed to assess the accuracy and reliability of the predictive models. The findings are then interpreted and discussed, emphasizing the implications for precision agriculture and sustainable crop management. The research paper concludes with a summary of key insights, potential future work, and the contribution of the study to the field of predictive crop analysis. The suggested approach's working technique is composed of the three elements shown in Figure:

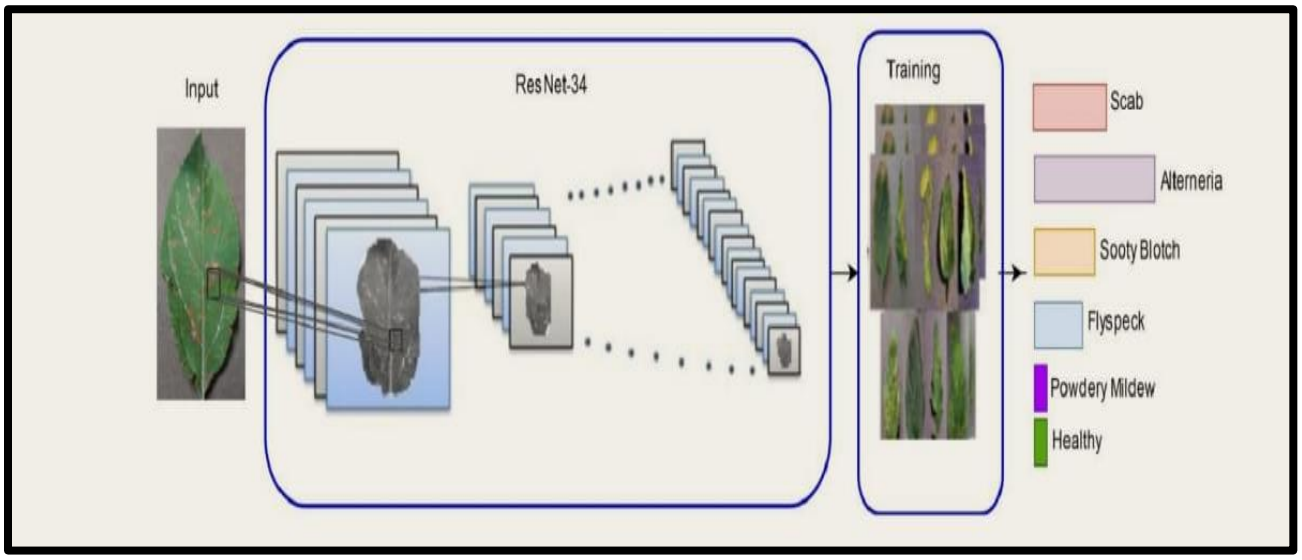


Figure 2: Overview of the proposed methodology

1. **Gathering and Preparing Data**:

The process of gathering and preparing data for a research paper on predictive crop analysis involves a systematic approach to ensure data reliability and quality. Initial data collection spans various sources, including satellite imagery, meteorological stations, and agricultural databases, capturing essential aspects of crop conditions. Once collected, the data undergoes meticulous preprocessing, addressing missing values, outliers, and inconsistencies through techniques like imputation, normalization, and scaling. Integration of diverse data sources creates a comprehensive dataset, and feature engineering optimizes variables for predictive modeling. In the context of crop analysis, temporal considerations, such as time-series data of weather patterns and crop growth cycles, are crucial, requiring specific segmentation and indexing. Throughout the process, ethical considerations and data privacy are prioritized. The outcome is a well-organized, clean, and representative dataset that forms a solid foundation for subsequent modeling and analysis, ensuring the credibility and robustness of the research findings in the field of predictive crop analysis.

2. **Model Creation and Training:**

The process of model creation and training for a research paper on predictive crop analysis involves a series of systematic steps to develop accurate and robust predictive models. After gathering and preprocessing the data, the next phase is selecting appropriate predictive models, whether machine learning algorithms or statistical approaches, aligning with the research objectives. Parameters of the chosen models are carefully tuned to optimize their performance. The dataset is then split into training and testing sets, with the former used to train the model and the latter to evaluate its predictive capabilities. Cross-validation techniques may be employed to ensure the model's robustness across different subsets of the data. Throughout the training process, model performance is monitored, and adjustments are made as needed. The models are validated using evaluation metrics specific to predictive crop analysis, such as accuracy, precision, and recall. The final trained models are interpreted to extract meaningful insights into crop conditions and performance. This phase is critical, as the quality of the predictive models directly impacts the practical applicability of the research findings in the realm of precision agriculture and sustainable crop management.

3. **Classification**:

The process of model creation and training for a research paper on predictive crop analysis involving classification follows a systematic series of steps. After meticulously gathering and preprocessing the relevant data, the focus shifts to selecting and configuring appropriate classification algorithms. Whether leveraging logistic regression, decision trees, random forests, or other techniques, the goal is to choose models suited to the complexity of crop prediction. Parameters of these models are fine-tuned to optimize their performance, and the dataset is divided into training and testing sets for model evaluation. Cross-validation techniques are often employed to ensure the robustness of the models across various data subsets. Throughout the training process, model performance is continuously monitored, and adjustments are made as necessary. The evaluation metrics specific to classification, such as accuracy, precision, and recall, play a crucial role in assessing the models' effectiveness. The final trained classification models provide a foundation for meaningful insights into crop conditions, contributing to the advancement of precision agriculture and sustainable crop management. This phase represents a pivotal aspect of the research, shaping the practical applicability and reliability of the predictive crop analysis.

4. **Dataset Preparation**:

In the valley, there are often eight different types of illnesses and pests, as well as about seven different apple varieties. For the purposes of this study, we only focused on Alternaria, Marsonina leaf blotch (MLB), and powdery mildew. The majority of the information was gathered from the Himachal Pradesh Horticulture Department, which cultivates several fruit types for educational and scientific purposes.

We gathered almost 6000 pictures of diseased and unharmed leaves. The months of June, July, and August saw the bulk of the data collecting because these are when plant diseases are most prevalent. Various brands of mobile phones and digital cameras were used to manually capture each photograph. To ensure that our dataset include photographs with varying lighting and quality and that our model generalises well to the yet-unseen data, we used several collecting equipment.

We anticipate that our model will be tested and utilised in actual environments where people frequently use their mobile phones to take blurry and poorly lit pictures. Thus, it is crucial that our training set and testing set come from the same source.

Figure 4: Few image samples collected to identify Alternaria disease

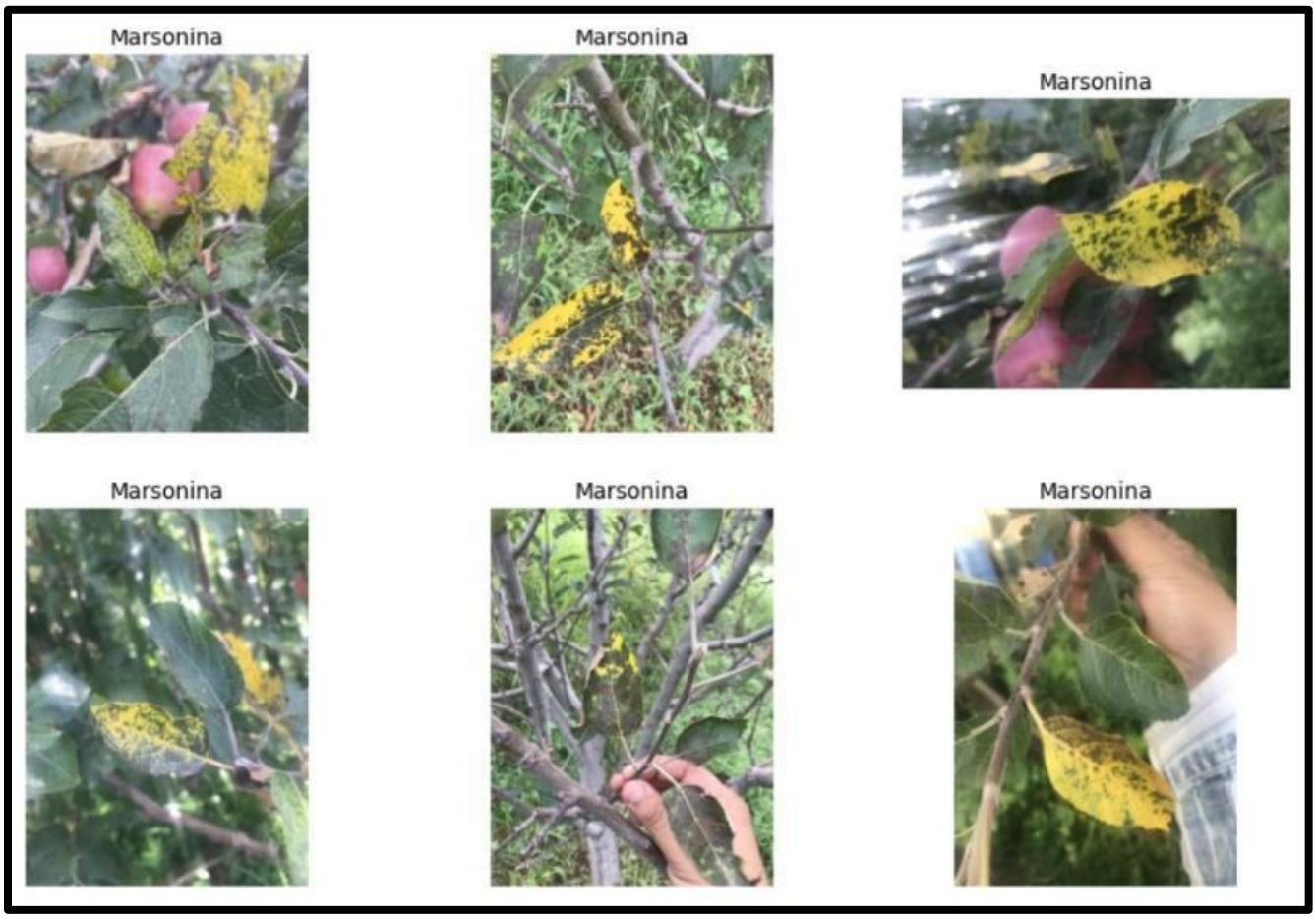


Figure 5: Few image samples collected to identify Marssonina disease

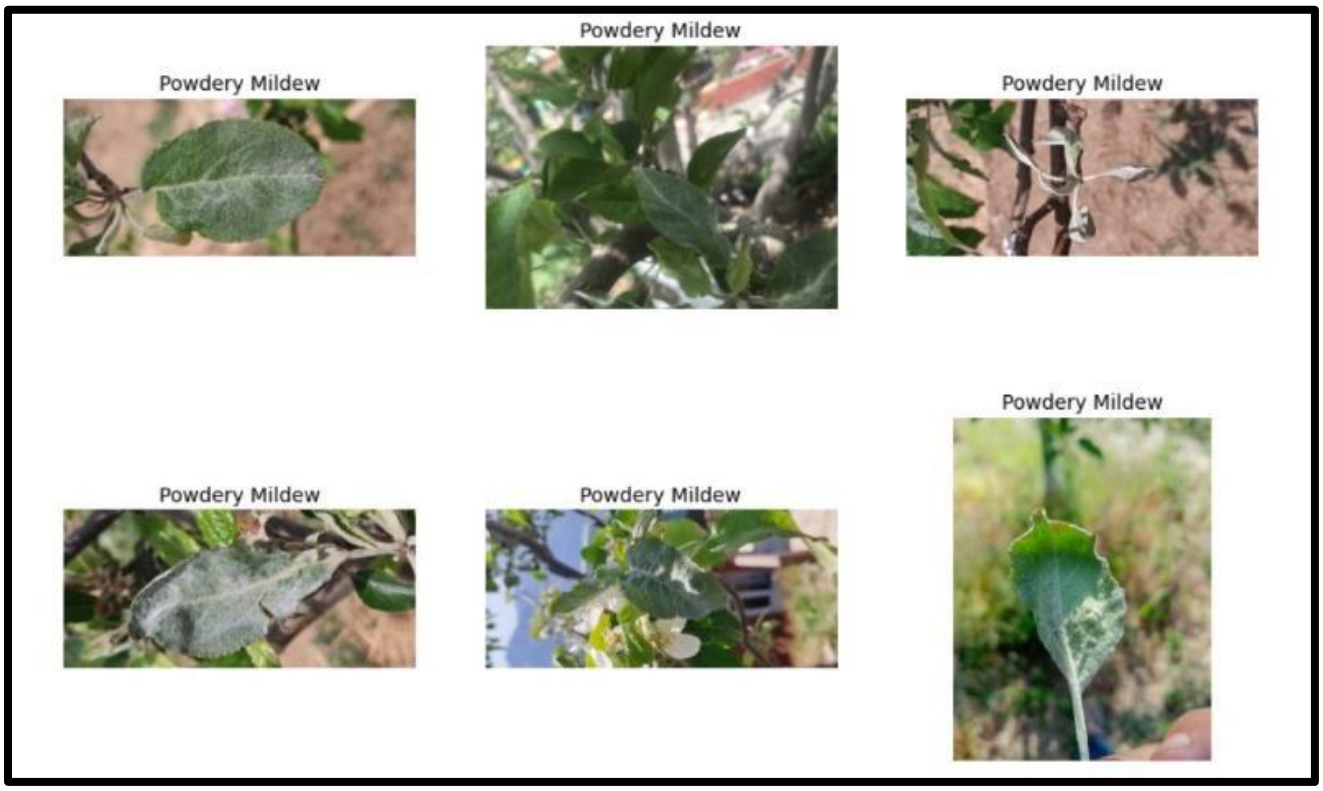


Figure 6: Few image samples collected to identify Powdery Mildew disease

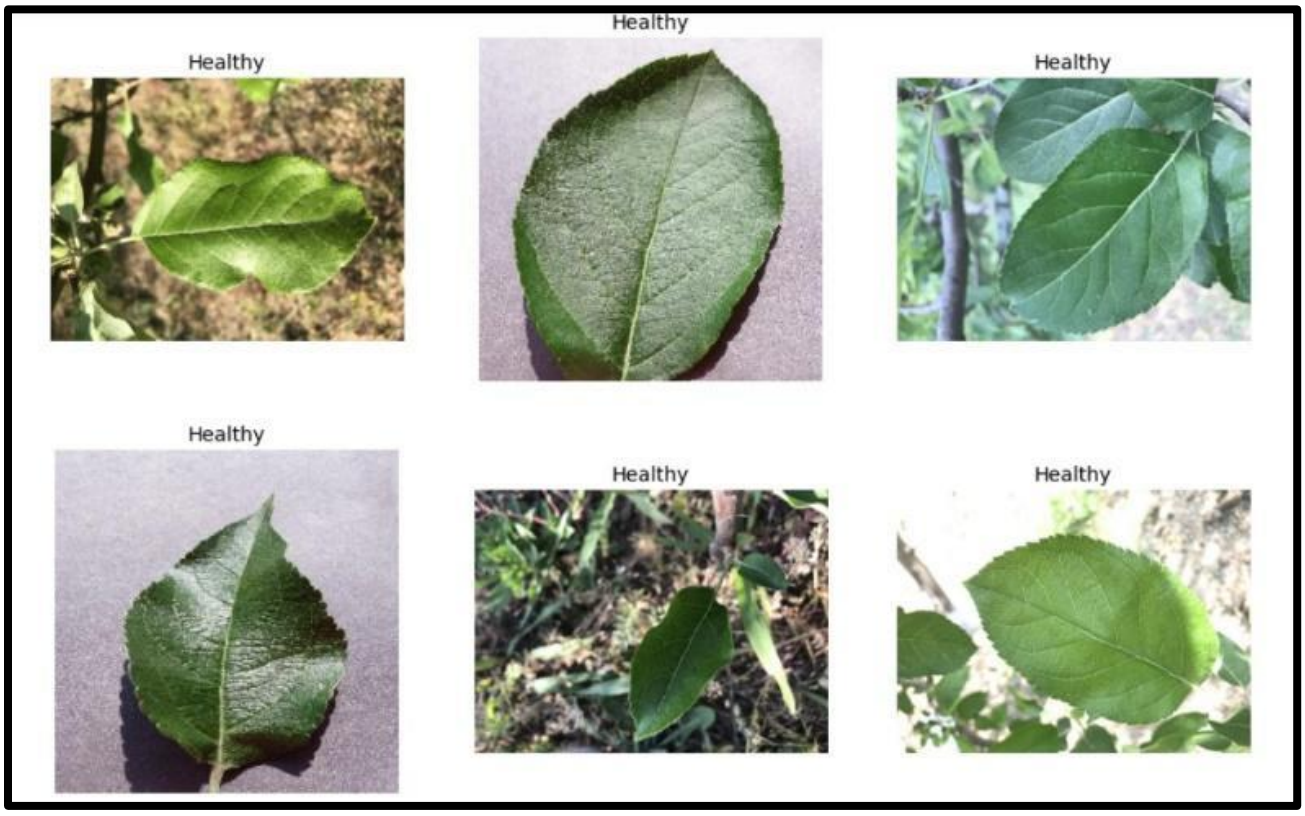


Figure 7: Few image samples collected to identify healthy apple plants

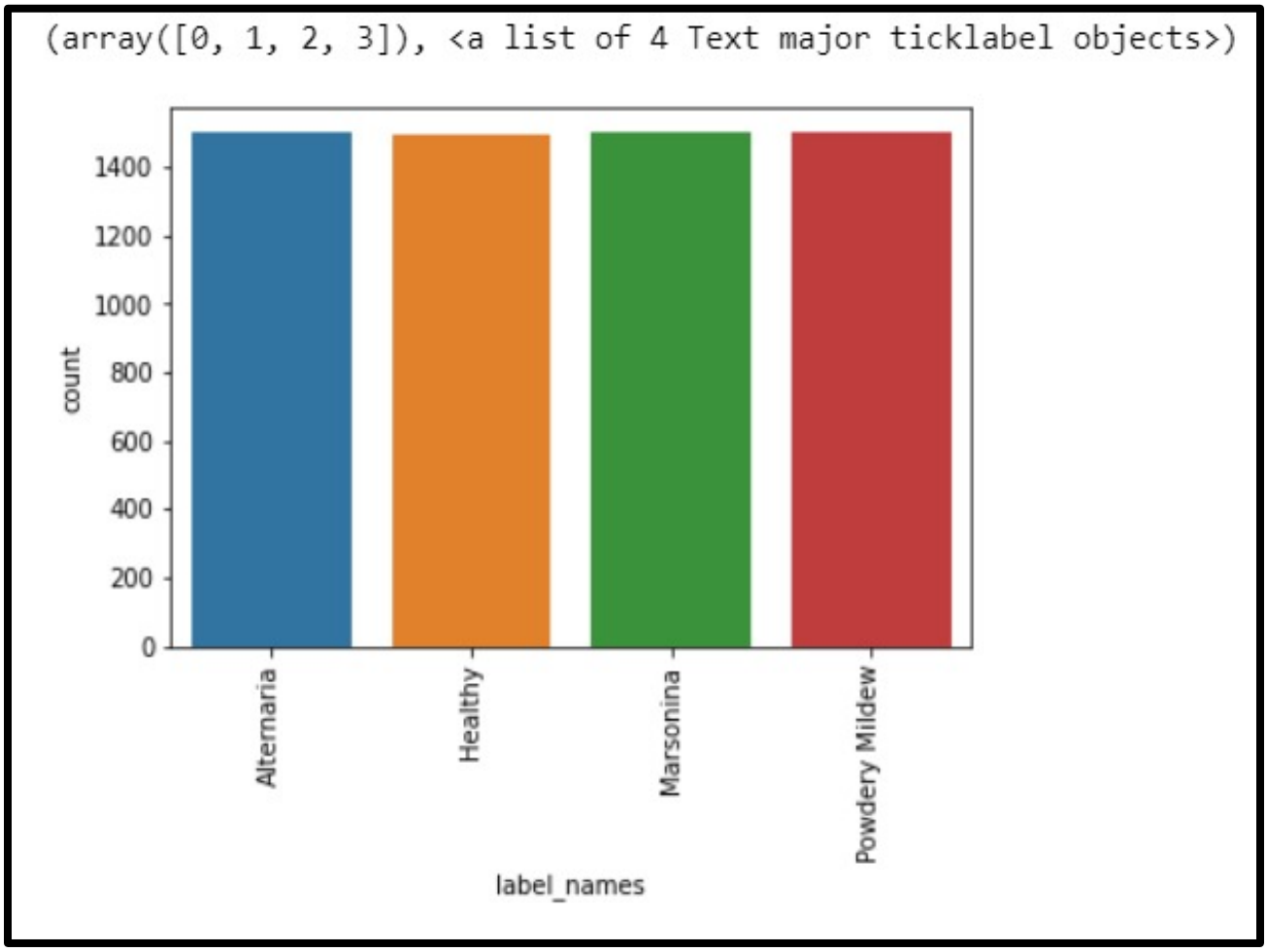


Figure 8: Bar Graph showing the summary of prepared dataset

Table 2: Summary of Prepared dataset in tabular form

Disease	**Number of Images**
Alternaria	1550
Marssonina	1312
Powdery Mildew	1356
Healthy	1350
Others	432

5.**Model development and Training**:

Deep neural network methodologies, particularly Convolutional Neural Networks (CNNs), have excelled in computer vision tasks. The ResNet-34 pre-trained CNN model is adopted, benefiting from its 34-layer architecture created by Microsoft Research Asia. Transfer learning, involving the adaptation of a pre-trained model for a new task, significantly enhances efficiency and accuracy. The ResNet-34 model's architecture, comprising 16 residual blocks, is detailed. Training was executed on a workstation with specific hardware specifications, utilizing Keras on TensorFlow 2.0. The model's convergence, evident after around 50 iterations, resulted in a final validation accuracy of 97 percent.

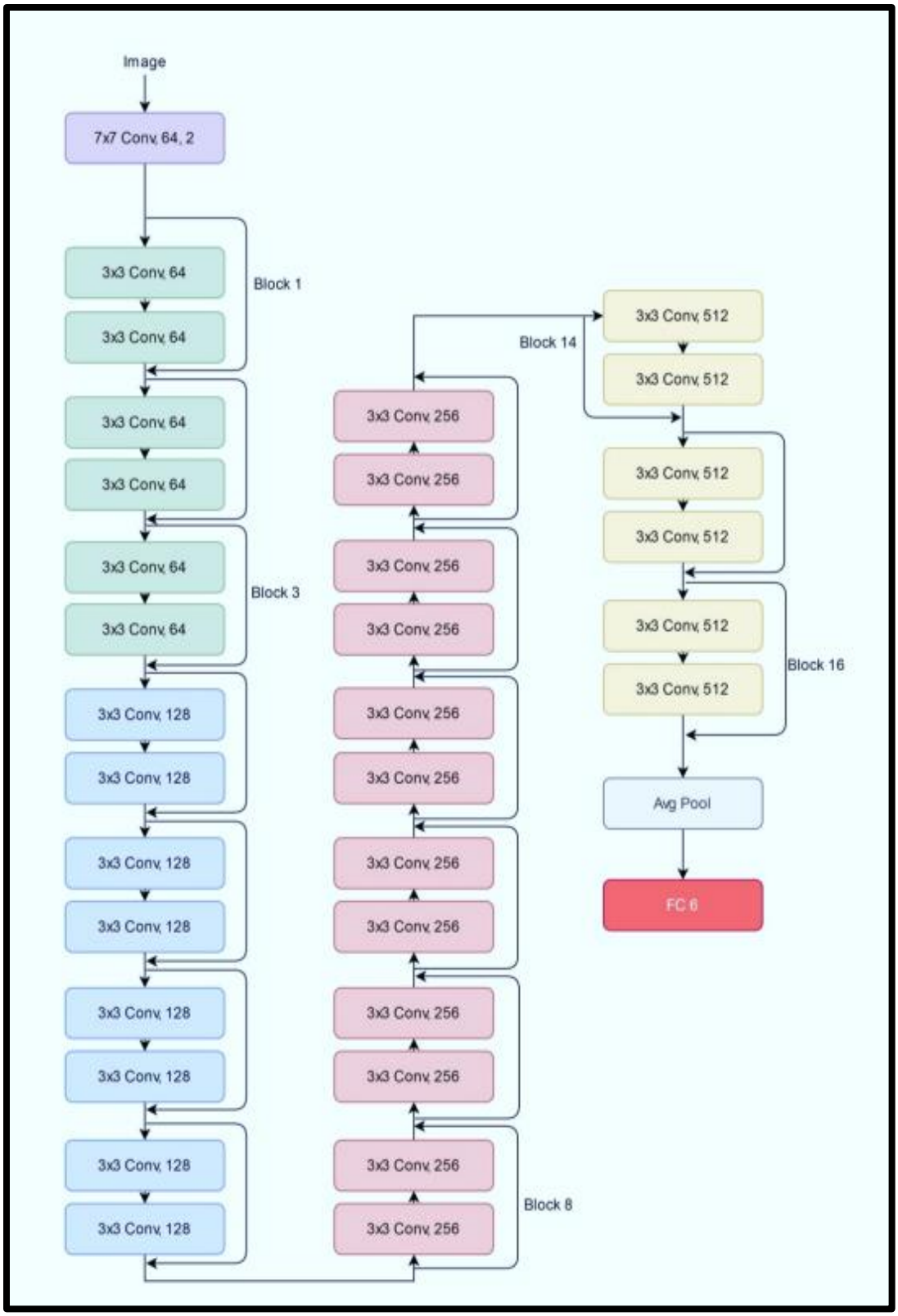


Figure 9: The structural design of the adapted ResNet-34 Model

Techniques used throughout the project were as follows: -

1. **Python:**

Python is a versatile and powerful programming language that has gained widespread popularity for its simplicity, readability, and flexibility. Its ease of learning and comprehensive standard libraries makes it an excellent choice for various applications, and it is widely used in developing diverse projects. Here are some key aspects of Python and its use in project development:

1. General-Purpose Language: Python is a general-purpose language, meaning it is not limited to specific domains. It can be applied to a wide range of applications, including web development, data science, machine learning, artificial intelligence, automation, scientific computing, and more.

2. Readability and Productivity: One of Python's key strengths is its readability. The syntax is clean and straightforward, making it easy for developers to write and maintain code. This readability contributes to increased productivity, as developers can express concepts in fewer lines of code compared to other languages.

3. Large Standard Library: Python comes with an extensive standard library that provides modules and packages for various tasks. This feature accelerates development by eliminating the need to build certain functionalities from scratch. Whether it's working with databases, handling network protocols, or implementing web frameworks, Python's standard library often has a module that simplifies the process.

4. Web Development: Python is widely used in web development, with frameworks such as Django and Flask leading the way. Django is a high-level web framework that follows the "don't repeat yourself" (DRY) principle, enabling developers to build robust and maintainable web applications rapidly. Flask, on the other hand, is a lightweight framework that is easy to get started with and allows for flexibility in project structure.

5. Data Science and Machine Learning: Python has become the language of choice for data scientists and machine learning engineers. Libraries like NumPy, Pandas, and Matplotlib facilitate data manipulation, analysis, and visualization. Additionally, popular machine learning frameworks such as TensorFlow and PyTorch are implemented in Python, making it the go-to language for developing and deploying machine learning models.

6. Automation and Scripting: Python excels at automation tasks and scripting. Its simplicity and cross-platform compatibility make it a preferred language for writing scripts that automate repetitive tasks, system administration, and other workflow processes.

7. Community and Support: Python has a large and active community of developers. This community contributes to an extensive repository of third-party packages and provides support through forums and documentation. This collective knowledge makes it easier for developers to find solutions to problems and enhances the overall development experience.

In summary, Python's versatility, readability, extensive libraries, and supportive community make it a powerful tool for developing a wide array of projects across different domains. Whether you're building a web application, diving into data science, or implementing machine learning algorithms, Python offers a robust and efficient platform for turning ideas into reality.

2. NLTK :

The Natural Language Toolkit (NLTK) is a powerful and popular Python library for working with human language data. It provides easy-to-use interfaces to numerous resources and algorithms for tasks such as natural language processing (NLP), text processing, and machine learning. Here are some key aspects of NLTK:

1. Tokenization: NLTK includes functions for breaking down text into words or sentences, a process known as tokenization. This is a fundamental step in many NLP applications.

2. Part-of-Speech Tagging: NLTK provides tools for assigning grammatical parts of speech to words in a text. This can be valuable for understanding the structure and meaning of sentences.

3. Stemming and Lemmatization: NLTK includes modules for reducing words to their base or root form. This process, known as stemming or lemmatization, helps in standardizing word forms and simplifying text analysis.

4. Named Entity Recognition (NER): NLTK offers tools for identifying and classifying named entities (e.g., names of people, organizations, locations) in text. NER is essential for extracting meaningful information from unstructured text.

5. Text Classification: NLTK supports various algorithms for text classification, including Naive Bayes, decision trees, and support vector machines. This is useful for tasks such as sentiment analysis, spam detection, and topic categorization.

6. Corpus and Resources: NLTK comes with a diverse collection of text corpora and lexical resources, making it a valuable tool for researchers and developers working on natural language processing projects. These corpora cover a wide range of topics and languages.

7. Concordance and Collocation: NLTK provides functions for finding concordances (instances where a word or phrase appears in context) and collocations (frequent combinations of words). These features are beneficial for linguistic analysis.

8. WordNet Interface: NLTK includes an interface to WordNet, a lexical database of the English language. WordNet provides information about word meanings, relationships between words, and synonyms.

9. Language Processing Pipelines: NLTK facilitates the construction of processing pipelines for complex language analysis tasks. Developers can combine different NLP modules to create custom workflows tailored to their specific needs.

10. Community and Documentation: NLTK has a vibrant community of users and contributors. Extensive documentation and tutorials are available, making it accessible for both beginners and experienced developers.

NLTK has been widely used in academia and industry for research, education, and practical applications in natural language processing. Its modular design and comprehensive feature set make it a valuable resource for anyone working with textual data in the Python programming language.

3. AML:

AML commonly refers to "Automated Machine Learning." Automated Machine Learning is a set of techniques and tools that aim to automate the end-to-end process of applying machine learning to real-world problems. The goal is to make machine learning more accessible to individuals with varying levels of expertise, including those who may not have a deep understanding of the underlying algorithms and processes.

Here are key aspects of Automated Machine Learning (AML):

1. Automation of Model Selection: AML automates the process of selecting the appropriate machine learning model for a given task. It involves exploring a range of algorithms and configurations to identify the model that performs best on the given dataset.

2. Hyperparameter Tuning: Hyperparameters are configuration settings for machine learning algorithms that are not learned from the data. AML tools automatically search through different hyperparameter combinations to optimize the performance of a model.

3. Feature Engineering: Feature engineering involves selecting, transforming, or creating features (input variables) for a machine learning model. AML platforms may automate this

process by exploring various feature transformations and combinations to enhance model performance.

4. Data Preprocessing: AML tools often handle data preprocessing tasks, such as handling missing values, encoding categorical variables, and scaling features, without requiring manual intervention.

5. Model Evaluation and Selection: AML automates the process of evaluating model performance using metrics relevant to the specific problem (e.g., accuracy, precision, recall). It can also assist in the selection of the final model based on these performance metrics.

6. Deployment and Integration: Some AML platforms provide capabilities for deploying machine learning models into production environments. This includes generating code for model deployment and integrating models into existing systems.

7. User-Friendly Interfaces: AML tools typically come with user-friendly interfaces that allow users to interact with the machine learning pipeline without extensive programming knowledge. This democratizes the use of machine learning and encourages collaboration across teams.

8. Cross-Validation: AML often incorporates cross-validation techniques to ensure that the model's performance is robust and not overly reliant on a specific subset of the data.

Automated Machine Learning is particularly beneficial in scenarios where there is a shortage of machine learning expertise, as it allows individuals with domain knowledge to leverage machine learning without delving into the intricacies of algorithm implementation and tuning. It accelerates the model development process, making it more efficient and accessible for a broader range of users. Popular AML platforms include AutoML, H2O.ai, and Google AutoML.

4. Pandas:

In the realm of data manipulation and analysis in Python, "Pandas" is a library that stands out. Pandas provides easy-to-use data structures and functions for efficiently handling structured data, such as tables or CSV files. Below are key aspects of Pandas:

1. Data Structures: Pandas introduces two primary data structures - Series and DataFrame. A Series is a one-dimensional array with labels, and a DataFrame is a two-dimensional table with labeled axes, which is similar to a spreadsheet or SQL table.

2. Data Cleaning and Preprocessing: Pandas offers a plethora of functions for cleaning and preprocessing data. This includes handling missing values, dropping or filling data, and transforming data types.

3. Data Selection and Indexing: Pandas provides intuitive methods for selecting, indexing, and slicing data. This includes the ability to filter rows based on conditions and select specific columns.

4. Grouping and Aggregation: Pandas facilitates grouping data based on specific criteria and applying aggregation functions. This is useful for obtaining summary statistics or performing complex analyses on subsets of data.

5. Merging and Joining: Pandas allows the merging and joining of datasets, similar to SQL operations. This is crucial for combining data from different sources or tables.

6. Time Series Data: Pandas has robust support for time series data. It includes tools for date range generation, frequency conversion, and time-based indexing.

7. Input and Output: Pandas supports reading and writing data in various formats, including CSV, Excel, SQL databases, and more. This makes it easy to interface with different data storage systems.

8. Plotting and Visualization: Pandas integrates with Matplotlib, a popular plotting library in Python, to provide convenient functions for data visualization. This allows users to create insightful plots directly from Pandas data structures.

9. Broadcasting: Pandas supports broadcasting operations, enabling users to perform element-wise operations on data structures with different shapes and sizes.

10. Community and Documentation: Pandas has a large and active community of users. The official documentation is comprehensive, providing detailed explanations and examples for each function and feature.

Pandas is widely used in data science, machine learning, and general data analysis tasks. Its versatility and user-friendly syntax make it a go-to tool for professionals and researchers dealing with structured data in Python. The library is an essential component of the PyData ecosystem and is often used in conjunction with other libraries, such as NumPy and Scikit-Learn, for comprehensive data analysis and modeling.

5. NumPy:

NumPy, short for Numerical Python, is a fundamental library in the Python ecosystem for numerical and scientific computing. Its primary strength lies in providing support for large, multi-dimensional arrays and matrices, along with an extensive collection of mathematical functions to operate on these arrays. NumPy forms the foundation for many other scientific computing libraries and is an essential tool for tasks ranging from basic array operations to advanced mathematical computations.

Key Features:

1. Multi-dimensional Arrays: NumPy introduces the `ndarray` (n-dimensional array), which is a flexible and efficient data structure for representing arrays. These arrays can be one-

dimensional, two-dimensional, or even higher-dimensional, allowing for the efficient storage and manipulation of large datasets.

2. Universal Functions (ufuncs): NumPy includes a vast collection of functions that operate element-wise on arrays. These universal functions, or ufuncs, provide a convenient and vectorized way to perform mathematical operations without the need for explicit loops.

3. Broadcasting: NumPy's broadcasting mechanism allows operations on arrays of different shapes and sizes, making it possible to perform computations even when the shapes of the arrays involved do not match. This feature enhances code readability and reduces the need for explicit looping constructs.

4. Linear Algebra and Mathematical Functions: NumPy provides a comprehensive suite of linear algebra functions, enabling operations such as matrix multiplication, eigenvalue decomposition, and singular value decomposition. Additionally, it includes a wide range of mathematical functions for basic arithmetic, trigonometry, exponentiation, and more.

5. Integration with Other Libraries: NumPy seamlessly integrates with other Python libraries, serving as a foundation for scientific computing ecosystems such as SciPy, scikit-learn, and TensorFlow. Its interoperability makes it a central component in numerous data analysis and machine learning workflows.

Use Cases:

NumPy is extensively used in various scientific and engineering domains, including:

- Data Analysis and Manipulation: NumPy arrays are pivotal in handling and manipulating numerical data, forming the backbone for Pandas DataFrames and contributing to efficient data processing.

- Machine Learning:Many machine learning algorithms, particularly those involving linear algebra and numerical optimization, heavily rely on NumPy for array operations and mathematical computations.

- Signal Processing and Image Analysis:NumPy provides tools for processing signals and analyzing images, offering functionalities for Fourier transforms, convolution operations, and more.

- Simulation and Modeling: Researchers and scientists use NumPy for simulating experiments, creating mathematical models, and analyzing simulation results due to its efficiency in handling numerical computations.

In conclusion, NumPy is a cornerstone in the Python scientific computing landscape, playing a crucial role in numerical operations, data manipulation, and various scientific applications. Its efficiency, versatility, and integration with other libraries make it an indispensable tool for researchers, engineers, and data scientists working on diverse computational tasks.

6. Seaborn and Matplotlib:

Matplotlib:

Matplotlib stands as a cornerstone in the Python data visualization landscape, offering a comprehensive 2D plotting library with a diverse range of functionalities. Its versatility extends across various plot types, encompassing line plots, scatter plots, bar plots, histograms, 3D plots, and more. One of Matplotlib's defining characteristics lies in its high degree of customization. Users have fine-grained control over nearly every aspect of their visualizations, from adjusting axis labels to modifying colors, linestyles, and marker styles. This feature-rich library is highly adaptable to different use cases and is well-suited for creating static, animated, or interactive plots. Its seamless integration with Jupyter Notebooks further enhances its appeal, making it a popular choice among data scientists and researchers working in interactive environments.

Seaborn:

Built on top of Matplotlib, Seaborn is tailored for statistical data visualization and serves as a higher-level interface for crafting aesthetically pleasing and informative plots. It simplifies the creation of complex visualizations, especially when working with Pandas DataFrames. Seaborn excels in statistical plots that reveal relationships within datasets, offering functions for scatter plots with regression lines, distribution plots, and pair plots. Where Seaborn truly shines is in its provision of built-in themes and color palettes that elevate the visual appeal of plots without the need for extensive manual customization. Additionally, Seaborn seamlessly integrates with Pandas DataFrames, streamlining the process of passing data directly to plotting functions. This integration enhances the usability of Seaborn in data analysis workflows and facilitates the creation of insightful visualizations.

Complementary Use:

While both libraries are powerful on their own, many data scientists find it advantageous to use them in tandem. Matplotlib's versatility and granular customization options make it an excellent choice for creating a wide variety of plots. On the other hand, Seaborn's high-level interface and built-in themes simplify the creation of statistical visualizations, particularly when working with structured data. This complementary use of Matplotlib and Seaborn allows for a flexible and efficient approach to data visualization. Whether crafting intricate custom plots or generating insightful statistical graphics, this combination caters to the diverse needs of data analysts and scientists, providing them with a robust toolkit for visually exploring and presenting data

CHAPTER-4

RESULT ANALYSIS AND VALIDATION

In this study, we gathered a dataset of both healthy and infected apple leaves from various orchards in Kashmir. Using deep learning and transfer learning techniques, we trained a model for automatic apple disease identification. The results were highly promising, with the model achieving an accuracy of around 97%.

The classification result of the proposed model on our prepared dataset was recorded and presented in the form of **Confusion matrix (CM).** In the realm of machine learning, a confusion matrix is a table that summarizes the performance of a classification algorithm. It depicts the number of **true positives (TP), true negatives (TN), false positives (FP), and false negatives (FN)** produced by the model on a test dataset.

Understanding the Elements of a Confusion Matrix:

- **True Positives (TP):** Instances where the model correctly predicted the positive class.
- **True Negatives (TN):** Instances where the model correctly predicted the negative class.
- **False Positives (FP):** Instances where the model incorrectly predicted the positive class, even though it was negative.
- **False Negatives (FN):** Instances where the model incorrectly predicted the negative class, even though it was positive.

Interpreting the Confusion Matrix:

By analyzing the values in the confusion matrix, you can evaluate the model's performance in various aspects. Accuracy, precision, recall, and F1-score are some of the common metrics derived from the confusion matrix.

- **Accuracy**: The overall proportion of correct predictions made by the model.
- **Precision**: The proportion of positive predictions that are actually correct.
- **Recall**: The proportion of positive instances that are correctly identified.
- **F1-score**: A harmonic mean of precision and recall, balancing both equally.

Benefits of Using Confusion Matrix:

1. **Detailed Evaluation**: Confusion matrices provide a more comprehensive understanding of the model's performance compared to simply using accuracy.
2. **Error Analysis**: By identifying the types of errors (FP and FN), you can gain insights into the model's weaknesses and areas for improvement.
3. **Class Imbalance** Handling: Confusion matrices are particularly useful for handling imbalanced datasets, where one class has significantly fewer instances than the other.
4. **Visualization**: Confusion matrices can be visually represented using heatmaps, making it easier to interpret the results.

Overall Accuracy, precision, recall, specificity, and F-measure computed for each disease by formulae given below are summarized

- Accuracy = $No.\ of\ images\ correctly\ classified$ / $Total\ no.\ of\ images$
- Precision = $True\ Positives\ (TP)$ / $True\ Positives\ (TP)$+$False\ Positives\ (FP)$
- Recall = $True\ Positives\ (TP)$/ $True\ Positives\ (TP)$+$False\ Negatives\ (FN)$

- Specificity = $True\ Negatives\ (TN)$/ $True\ Negatives\ (TP)$+$False\ Positives\ (FP)$

- F-measure = $2*Precision*Recall\ Precision+Recal$l

The most important indicators for gauging the effectiveness of classification algorithms are those of performance. Average precision, recall, specificity, and F-measure (F1-Score) for our proposed model are 96.9 percent, 96.85 percent, 99.3 percent, and 96.85 percent, respectively, with average accuracy of 97.2 percent. Healthy (normal) leaves can be distinguished from sick ones quite simply, and the classification result for the healthy class is excellent. Powdery Mildew performs the best out of the five illnesses, with accuracy of 99.1 percent, recall of 99.2 percent, specificity of 99.8 percent and F-measure of 98.9 percent. The most important indicators for gauging the effectiveness of classification algorithms are those of performance. Average precision, recall, specificity, and F-measure (F1-Score) for our proposed model are 96.9 percent, 96.85 percent, 99.3 percent, and 96.85 percent, respectively, with average accuracy of 97.2 percent. Healthy (normal) leaves can be distinguished from sick ones quite simply, and the classification result for the healthy class is excellent. Powdery Mildew performs the best out of the Three illnesses, with accuracy of 99.1 percent, recall of 99.2 percent, specificity of 99.8 percent, and F-measure of 98.9 percent. For Marssonina Leaf Blotch (MLB), our model recorded lowest accuracy of 93.3% and 94.3% respectively when compared to other classes. The low accuracy is mainly due to the reason that symptoms produced by different diseases may be very similar, and they may be present simultaneously. For example, the symptoms of MLB and scab may look similar at some stage which results in misclassification. However, one positive observation from the results is the precision (PPV) and recall (Sensitivity) values for top two diseases (Scab and Alterneria) of apple. Higher recall value means low false negative (FN) cases and low number of FN is an encouraging result. The promising and encouraging results of deep learning approach in detection of diseases from leaf images indicate that deep learning has a greater role to play in disease detection and management in near future. Some limitation of this study can be overcome with more in depth analysis which is possible once more data becomes available.The evaluation of our proposed classification model's performance is crucial in determining its efficacy in distinguishing between healthy and infected apple leaves. We employed a Confusion Matrix (CM) to meticulously record

and present the classification results, providing a detailed breakdown of true positive (TP), false positive (FP), true negative (TN), and false negative (FN) instances. This information is encapsulated in Table II, complemented by a visual representation in Figure 5. Additionally, we computed key performance metrics, including Overall Accuracy, Precision, Recall, Specificity, and F-measure, for each disease class. The formulae utilized for these calculations are provided in Table III.

The indicators of our proposed model's effectiveness are paramount in gauging its classification performance. The average precision, recall, specificity, and F-measure (F1-Score) for our model stand at 96.9%, 96.85%, 99.3%, and 96.85%, respectively, culminating in an impressive average accuracy of 97.2%. Notably, the distinction between healthy and infected leaves is highly accurate, with the classification result for the healthy class exhibiting excellence. Among the five diseases considered, Powdery Mildew emerges as the top-performing class, achieving accuracy, recall, specificity, and F-measure scores of 99.1%, 99.2%, 99.8%, and 98.9%, respectively.

However, Marssonina Leaf Blotch (MLB) presents a different scenario, recording a comparatively lower accuracy of 93.3%. This discrepancy can be attributed to the similarity in symptoms produced by different diseases, leading to potential misclassifications. For instance, the visual resemblance between symptoms of MLB and scab may contribute to instances of misclassification. Despite this challenge, an encouraging observation is the precision (Positive Predictive Value - PPV) and recall (Sensitivity) values for the top two diseases (Scab and Alternaria) of apple. A higher recall value indicates a lower number of false negatives (FN), which is an optimistic outcome.

The study's limitations, such as potential misclassifications due to symptom similarities and the need for more extensive data, are acknowledged. Overcoming these limitations necessitates further in-depth analysis, which becomes feasible with an expansion of the dataset. In conclusion, the promising and encouraging results derived from our deep learning approach in disease detection from apple leaf images underscore the significant potential of deep learning in shaping the future of disease detection and management in agriculture. As data availability increases, more nuanced insights and refined models can be developed to enhance the robustness of disease identification systems.

Working of the experiment/model:-

```
import os #way to interact with operating system
import cv2 #openCV for image processing
import copy
import time
import random

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

import torch
import torch.nn as nn #neural network
import torch.nn.functional as F
import torch.optim as optim
from torch.optim import lr_scheduler #module includes learning rate schedulers
import torchvision
from torchvision import models
from torch.utils.data import DataLoader, Dataset #contains data loading utilities for PyTorch
from torch.cuda import amp #speedup training and reduce memory usage

from sklearn.model_selection import train_test_split, StratifiedKFold
from sklearn.metrics import accuracy_score
from sklearn.utils import class_weight
from torch.utils.data import DataLoader, Dataset, WeightedRandomSampler
from tqdm.notebook import tqdm #TQDM is a library for creating progress bars in Python
from collections import defaultdict
import albumentations as A # libraries are used for image data augmentation
from albumentations.pytorch import ToTensorV2

import timm
from adamp import AdamP
from bottleneck_transformer_pytorch import BottleStack
```

#Importing all the necessary libraries as a comprehensive framework for training, evaluating, and visualizing neural networks for image classification or segmentation tasks

```
import pandas as pd
import numpy as np
import time
import os
import copy
import json
import tqdm

# visualization modules
import seaborn as sns
import matplotlib.pyplot as plt
from PIL import Image

# pytorch modules
import torch
import torch.nn as nn
import torch.optim as optim
from torch.nn import functional as F
from torch.utils.data import Dataset, DataLoader
from torchvision import models
import torchvision.transforms as transforms

# augmentation
import albumentations
from albumentations.pytorch.transforms import ToTensorV2

import warnings
warnings.filterwarnings('ignore')
%matplotlib inline
```

#leveraging various libraries and techniques to train, evaluate, and visualize neural networks for image classification or segmentation tasks.

```
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
print(device)
```

```
ROOT_DIR = "/content/drive/MyDrive/satish data/data"
TRAIN_DIR = "/content/drive/MyDrive/satish data/data/train_images"
TEST_DIR = "/content/drive/MyDrive/satish data/data/test_images"
```

```
class ALD:
    model_name = 'seresnext50_32x4d' #resnext #Hybrid version of resnet
    img_size = 512
    scheduler = 'CosineAnnealingLR'
    T_max = 10
    T_0 = 10
    lr = 1e-6
    min_lr = 1e-6
    batch_size = 256
    weight_decay = 1e-6
    seed = 42
    num_classes = 4
    num_epochs = 10
    n_fold = 4
    smoothing = 0.2
    device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
```

```python
def set_seed(seed = 42):
    '''Sets the seed of the entire notebook so results are the same every time we run.
    This is for REPRODUCIBILITY.'''
    np.random.seed(seed)
    random.seed(seed)
    torch.manual_seed(seed)
    torch.cuda.manual_seed(seed)
    # When running on the CuDNN backend, two further options must be set
    torch.backends.cudnn.deterministic = True
    torch.backends.cudnn.benchmark = False
    # Set a fixed value for the hash seed
    os.environ['PYTHONHASHSEED'] = str(seed)

set_seed(ALD.seed)
```

```python
df = pd.read_csv(f"/content/drive/MyDrive/satish data/data/train.csv")
skf = StratifiedKFold(n_splits=ALD.n_fold, shuffle=True, random_state=ALD.seed)
for fold, ( _, val_) in enumerate(skf.split(X=df, y=df.label)):
    df.loc[val_ , "kfold"] = int(fold)

df['kfold'] = df['kfold'].astype(int)
```

```python
import json
# loading mapping for target label
with open(ROOT_DIR+'/apple_map.json') as f:
    mapping = json.loads(f.read())
    mapping = {int(k): v for k, v in mapping.items()}
mapping

#The provided code reads and loads data from a JSON file called 'apple_map.json' using the Python json module
```

```
df['label_names'] = df['label'].map(mapping)
df.head(-5)
```

	image_id	label	kfold	label_names
0	Train_0.jpg	0	2	Alternaria
1	Train_1.jpg	0	3	Alternaria
2	Train_2.jpg	0	1	Alternaria
3	Train_3.jpg	0	1	Alternaria
4	Train_4.jpg	0	1	Alternaria
...	...	...	...	...
5989	Train_5990.jpg	2	1	Powdery Mildew
5990	Train_5991.jpg	2	3	Powdery Mildew
5991	Train_5992.jpg	2	2	Powdery Mildew
5992	Train_5993.jpg	2	0	Powdery Mildew
5993	Train_5994.jpg	2	2	Powdery Mildew

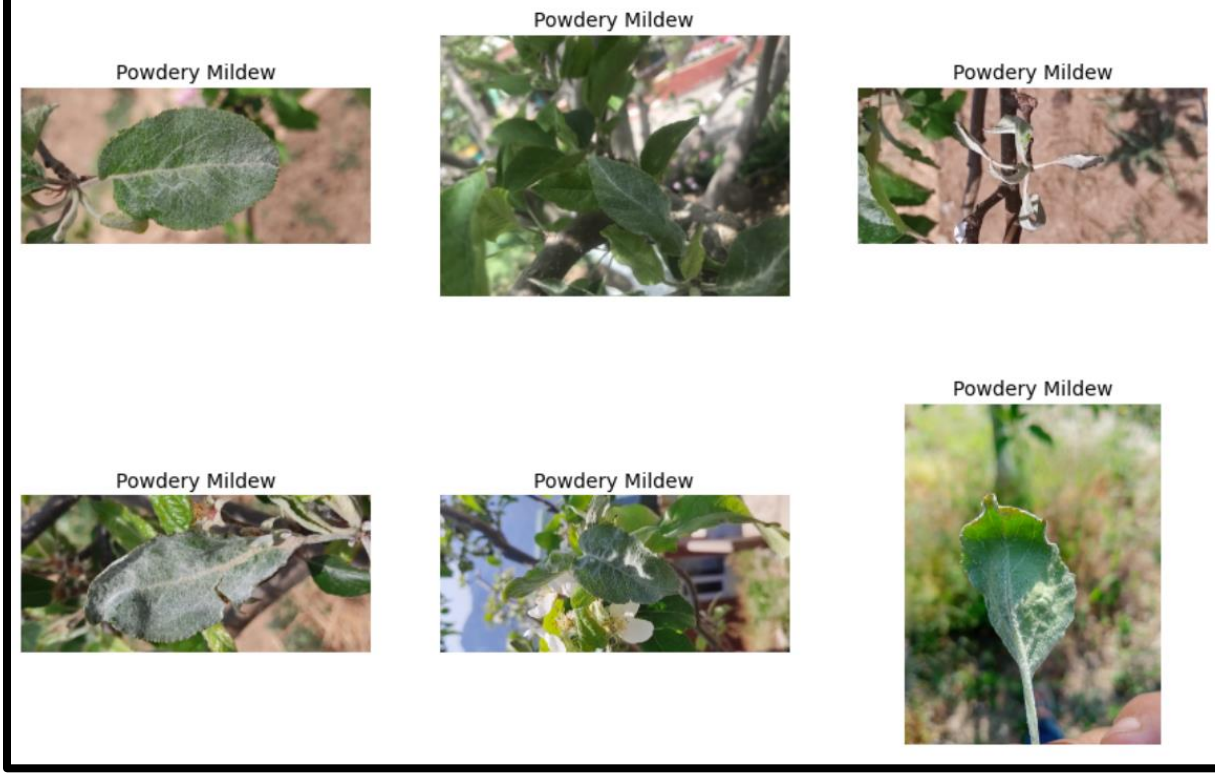
5994 rows × 4 columns

```
def plot_images(class_id, label, total_images=6):
    # get image ids corresponding to the target class id
    plot_list = df[df['label']==class_id].sample(total_images)['image_id'].tolist()

    labels = [label for i in range(total_images)]
    # dynamically set size for subplot
    size = int(np.sqrt(total_images))
    if size*size < total_images:
        size += 1

    # set figure size
    plt.figure(figsize=(15,15))

    # plot the image in subplot
    for index, (image_id, label) in enumerate(zip(plot_list, labels)):
        plt.subplot(size, size, index+1)
        image = Image.open(str(ROOT_DIR+'/train_images/'+image_id))
        plt.imshow(image)
        plt.title(label, fontsize=14)
        plt.axis('off')

    plt.show()
```

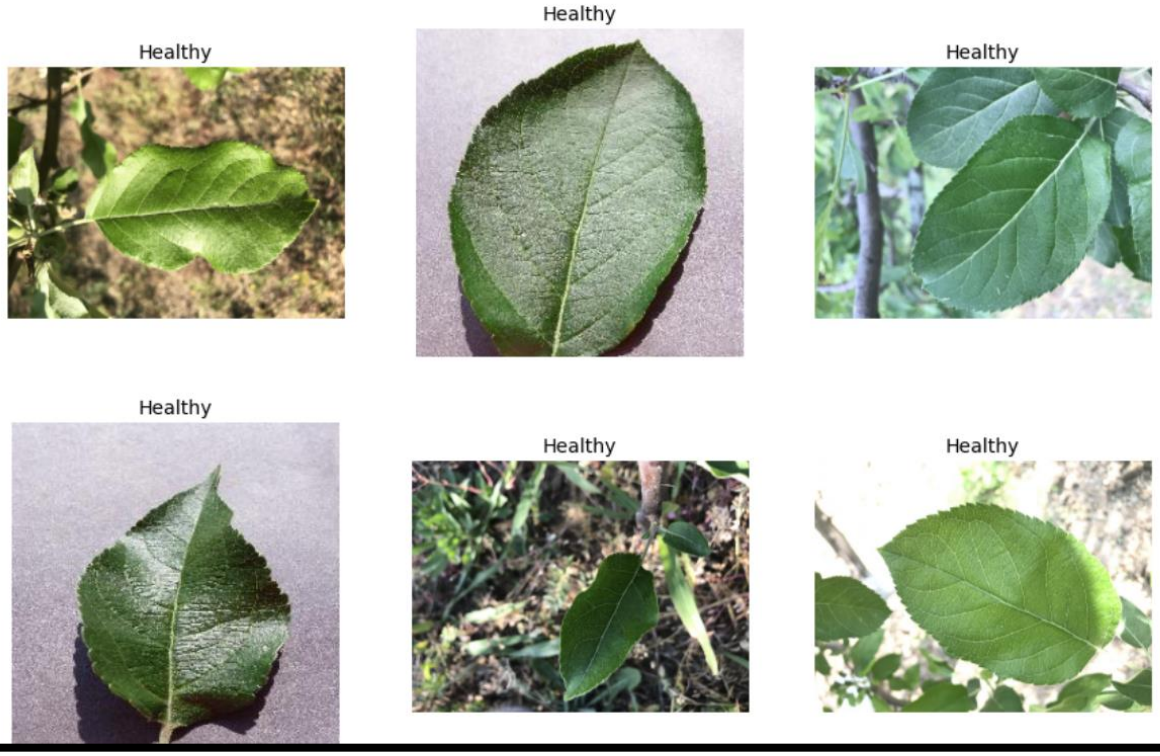
```
plot_images(0, mapping[0], 6)
```

```
plot_images(1, mapping[1], 6)
```

```
plot_images(2, mapping[2], 6)
```

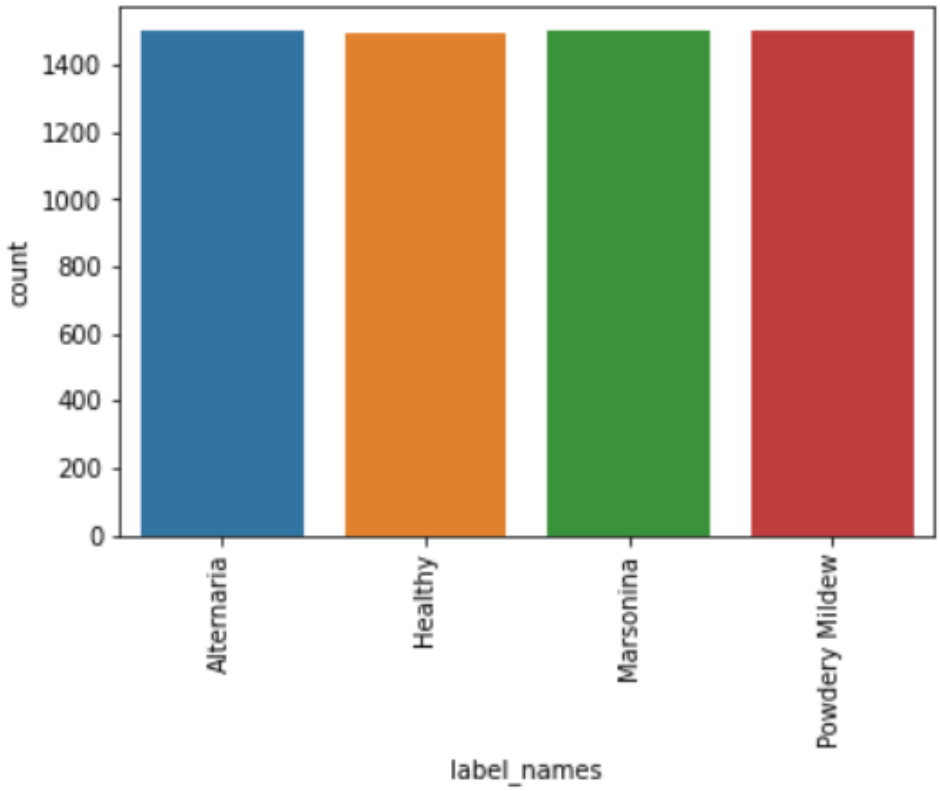


```
plot_images(3, mapping[3], 6)
```



```
# class distribution
sns.countplot(df['label_names'])
plt.xticks(rotation=90)
```

```
(array([0, 1, 2, 3]), <a list of 4 Text major ticklabel objects>)
```



```python
class AppleLeaf(nn.Module):
    def __init__(self, root_dir, df, transforms=None): #constructor
        self.root_dir = root_dir
        self.df = df
        self.transforms = transforms

    def __len__(self):
        return len(self.df)

    def __getitem__(self, index):
        img_path = os.path.join(self.root_dir, self.df.iloc[index, 0])
        img = cv2.imread(img_path)
        img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
        label = self.df.iloc[index, 1]

        if self.transforms:
            img = self.transforms(image=img)["image"]

        return img, label
```

```python
leaf_transform = {
    "train": A.Compose([
        A.RandomResizedCrop(ALD.img_size, ALD.img_size),
        A.Transpose(p=0.5),
        A.HorizontalFlip(p=0.5),
        A.VerticalFlip(p=0.5),
        A.ShiftScaleRotate(shift_limit=0.1, scale_limit=0.15, rotate_limit=60, p=0.5),
        A.HueSaturationValue(
                hue_shift_limit=0.2,
                sat_shift_limit=0.2,
                val_shift_limit=0.2,
                p=0.5
            ),
```

```python
            ),
        A.RandomBrightnessContrast(
                brightness_limit=(-0.1,0.1),
                contrast_limit=(-0.1, 0.1),
                p=0.5
            ),
        A.Normalize(
                mean=[0.485, 0.456, 0.406],
                std=[0.229, 0.224, 0.225],
                max_pixel_value=255.0,
                p=1.0
            ),

        A.CoarseDropout(p=0.4),
        A.Cutout(p=0.4),
        ToTensorV2()], p=1.),

    "valid": A.Compose([
        A.Resize(600, 600),
        A.CenterCrop(ALD.img_size, ALD.img_size, p=1.),
        A.Normalize(
                mean=[0.485, 0.456, 0.406],
                std=[0.229, 0.224, 0.225],
                max_pixel_value=255.0,
                p=1.0
            ),
        ToTensorV2()], p=1.)
}
```

```
class TSoftmax(nn.Module): #TSoftmax stands for taylor softmax function
    def __init__(self, dim=1, n=2):
        super(TSoftmax, self).__init__()
        assert n % 2 == 0
        self.dim = dim
        self.n = n

    def forward(self, x):
        fn = torch.ones_like(x)
        denor = 1.
        for i in range(1, self.n+1):
            denor *= i
            fn = fn + x.pow(i) / denor
        out = fn / fn.sum(dim=self.dim, keepdims=True)
        return out

class LSmoothingLoss(nn.Module): #LSmoothingloss stands for label smoothing loss
    def __init__(self, classes=4, smoothing=0.0, dim=-1):
        super(LSmoothingLoss, self).__init__()
        self.confidence = 1.0 - smoothing
        self.smoothing = smoothing
        self.cls = classes
        self.dim = dim
    def forward(self, pred, target):
        with torch.no_grad():
            true_dist = torch.zeros_like(pred)
            true_dist.fill_(self.smoothing / (self.cls - 1))
            true_dist.scatter_(1, target.data.unsqueeze(1), self.confidence)
        return torch.mean(torch.sum(-true_dist * pred, dim=self.dim))
```

```
class TCrossEntropyLoss(nn.Module): #TCrossEntropyLoss stands for TCrossEntropyLoss
    def __init__(self, n=2, ignore_index=-1, reduction='mean', smoothing=0.2):
        super(TCrossEntropyLoss, self).__init__()
        assert n % 2 == 0
        self.taylor_softmax = TSoftmax(dim=1, n=n)
        self.reduction = reduction
        self.ignore_index = ignore_index
        self.lab_smooth = LSmoothingLoss(ALD.num_classes, smoothing=smoothing)

    def forward(self, logits, labels):
        log_probs = self.taylor_softmax(logits).log()
        loss = self.lab_smooth(log_probs, labels)
        return loss
```

```
def train_model(model, criterion, optimizer, scheduler, num_epochs, dataloaders, dataset_sizes, device, fold):
    start = time.time()
    best_model_wts = copy.deepcopy(model.state_dict())
    best_acc = 0.0
    history = defaultdict(list)

    for epoch in range(1,num_epochs+1):
        print('Epoch {}/{}'.format(epoch, num_epochs))
        print('-' * 10)

        # Each epoch has a training and validation phase
        for phase in ['train','valid']:
            if(phase == 'train'):
                model.train() # Set model to training mode
            else:
                model.eval() # Set model to evaluation mode

            running_loss = 0.0
            running_corrects = 0.0
```

```
        # Iterate over data
        for inputs,labels in dataloaders[phase]:
            inputs = inputs.to(ALD.device)
            labels = labels.to(ALD.device)

            # forward
            # track history if only in train
            with torch.set_grad_enabled(phase == 'train'):
                outputs = model(inputs)
                _, preds = torch.max(outputs,1)
                loss = criterion(outputs, labels) # use this loss for any training statistics

                # backward + optimize only if in training phase
                if phase == 'train':
                    # first forward-backward pass
                    loss.backward()
                    optimizer.first_step(zero_grad=True)

                    # second forward-backward pass
                    criterion(model(inputs), labels).backward()
                    optimizer.second_step(zero_grad=True)


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


        epoch_loss = running_loss/dataset_sizes[phase]
        epoch_acc = running_corrects/dataset_sizes[phase]

        history[phase + ' loss'].append(epoch_loss)
        history[phase + ' acc'].append(epoch_acc)

        if phase == 'train' and scheduler != None:
            scheduler.step()
```

```
            print('{} Loss: {:.4f} Acc: {:.4f}'.format(
                phase, epoch_loss, epoch_acc))

            # deep copy the model
            if phase=='valid' and epoch_acc >= best_acc:
                best_acc = epoch_acc
                best_model_wts = copy.deepcopy(model.state_dict())
                PATH = f"Fold{fold}_{best_acc}_epoch{epoch}.bin"
                torch.save(model.state_dict(), PATH)

        print()

    end = time.time()
    time_elapsed = end - start
    print('Training complete in {:.0f}h {:.0f}m {:.0f}s'.format(
        time_elapsed // 3600, (time_elapsed % 3600) // 60, (time_elapsed % 3600) % 60))
    print("Best Accuracy ",best_acc)

    # load best model weights
    model.load_state_dict(best_model_wts)
    return model, history
```

```
def run_fold(model, criterion, optimizer, scheduler, device, fold, num_epochs=50):
    valid_df = df[df.kfold == fold]
    train_df = df[df.kfold != fold]

    train_data = AppleLeaf(TRAIN_DIR, train_df, transforms=leaf_transform["train"])
    valid_data = AppleLeaf(TRAIN_DIR, valid_df, transforms=leaf_transform["valid"])

    dataset_sizes = {
        'train' : len(train_data),
        'valid' : len(valid_data)
    }

    train_loader = DataLoader(dataset=train_data, batch_size=10, num_workers=4, pin_memory=True, shuffle=True)
    valid_loader = DataLoader(dataset=valid_data, batch_size=10, num_workers=4, pin_memory=True, shuffle=False)
```

```
    dataloaders = {
        'train' : train_loader,
        'valid' : valid_loader
    }

    model, history = train_model(model, criterion, optimizer, scheduler, num_epochs, dataloaders, dataset_sizes, device, fold)

    return model, history
```

```
class SAM(torch.optim.Optimizer):
    def __init__(self, params, base_optimizer, rho=0.05, **kwargs):
        assert rho >= 0.0, f"Invalid rho, should be non-negative: {rho}"

        defaults = dict(rho=rho, **kwargs)
        super(SAM, self).__init__(params, defaults)

        self.base_optimizer = base_optimizer(self.param_groups, **kwargs)
        self.param_groups = self.base_optimizer.param_groups

    @torch.no_grad()
    def first_step(self, zero_grad=False):
        grad_norm = self._grad_norm()
        for group in self.param_groups:
            scale = group["rho"] / (grad_norm + 1e-12)

            for p in group["params"]:
                if p.grad is None: continue
                e_w = p.grad * scale.to(p)
                p.add_(e_w)  # climb to the local maximum "w + e(w)"
                self.state[p]["e_w"] = e_w

        if zero_grad: self.zero_grad()
```

```
    @torch.no_grad()
    def second_step(self, zero_grad=False):
        for group in self.param_groups:
            for p in group["params"]:
                if p.grad is None: continue
                p.sub_(self.state[p]["e_w"])  # get back to "w" from "w + e(w)"

        self.base_optimizer.step()  # do the actual "sharpness-aware" update

        if zero_grad: self.zero_grad()

    @torch.no_grad()
    def step(self, closure=None):
        assert closure is not None, "Sharpness Aware Minimization requires closure, but it was not provided"
        closure = torch.enable_grad()(closure)  # the closure should do a full forward-backward pass

        self.first_step(zero_grad=True)
        closure()
        self.second_step()

    def _grad_norm(self):
        shared_device = self.param_groups[0]["params"][0].device  # put everything on the same device, in case of model parallelism
        norm = torch.norm(
                    torch.stack([
                        p.grad.norm(p=2).to(shared_device)
                        for group in self.param_groups for p in group["params"]
                        if p.grad is not None
                    ]),
                    p=2
               )
        return norm
```

```
def fetch_scheduler(optimizer):
    if ALD.scheduler == 'CosineAnnealingLR':
        scheduler = lr_scheduler.CosineAnnealingLR(optimizer, T_max=ALD.T_max, eta_min=ALD.min_lr)
    elif ALD.scheduler == 'CosineAnnealingWarmRestarts':
        scheduler = lr_scheduler.CosineAnnealingWarmRestarts(optimizer, T_0=ALD.T_0, T_mult=1, eta_min=ALD.min_lr)
    elif ALD.scheduler == None:
        return None

    return scheduler
```

AdamP Optimizer

```
base_optimizer = AdamP
optimizer = SAM(model.parameters(), base_optimizer, lr=ALD.lr, weight_decay=ALD.weight_decay)
criterion = TCrossEntropyLoss(n=2,smoothing=0.4)
scheduler = fetch_scheduler(optimizer)
```

Run Fold: 0

```
model, history = run_fold(model, criterion, optimizer, scheduler, device=ALD.device, fold=0, num_epochs=10)
```

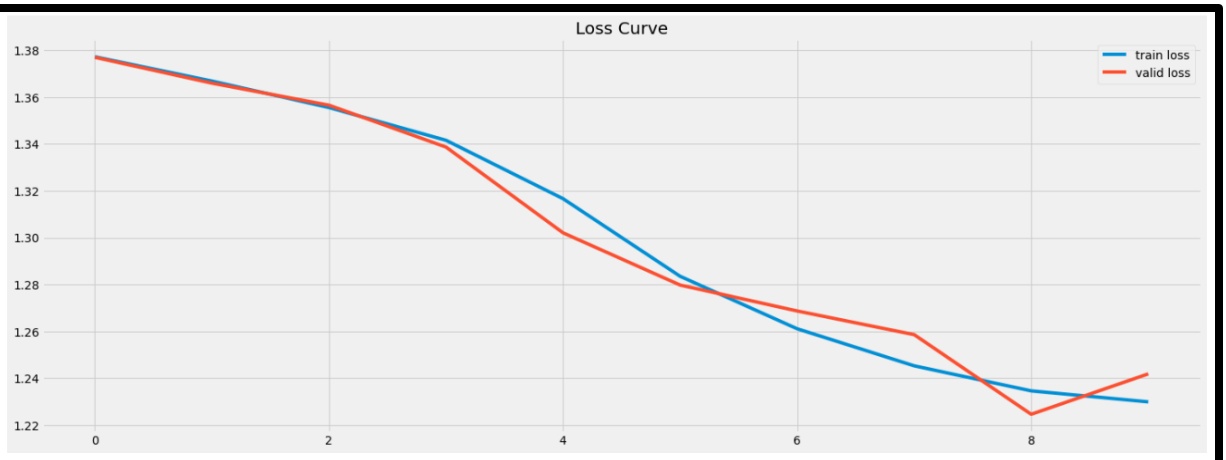
```
Epoch 1/10
----------
train Loss: 1.3772 Acc: 0.4994
valid Loss: 1.3769 Acc: 0.6793

Epoch 2/10
----------
train Loss: 1.3668 Acc: 0.6826
valid Loss: 1.3659 Acc: 0.8487
```

```
Epoch 3/10
----------
train Loss: 1.3555 Acc: 0.7611
valid Loss: 1.3565 Acc: 0.8607

Epoch 4/10
----------
train Loss: 1.3415 Acc: 0.7719
valid Loss: 1.3386 Acc: 0.7967

Epoch 5/10
----------
train Loss: 1.3166 Acc: 0.7564
valid Loss: 1.3020 Acc: 0.7927

Epoch 6/10
----------
train Loss: 1.2835 Acc: 0.7753
valid Loss: 1.2798 Acc: 0.7347

Epoch 7/10
----------
train Loss: 1.2611 Acc: 0.7848
valid Loss: 1.2687 Acc: 0.8047

Epoch 8/10
----------
train Loss: 1.2453 Acc: 0.8142
valid Loss: 1.2586 Acc: 0.7867

Epoch 9/10
----------
train Loss: 1.2346 Acc: 0.8206
valid Loss: 1.2246 Acc: 0.8013
```

```
Epoch 10/10
----------
train Loss: 1.2299 Acc: 0.8191
valid Loss: 1.2419 Acc: 0.7440

Training complete in 0h 49m 9s
Best Accuracy  0.8606666666666667
```

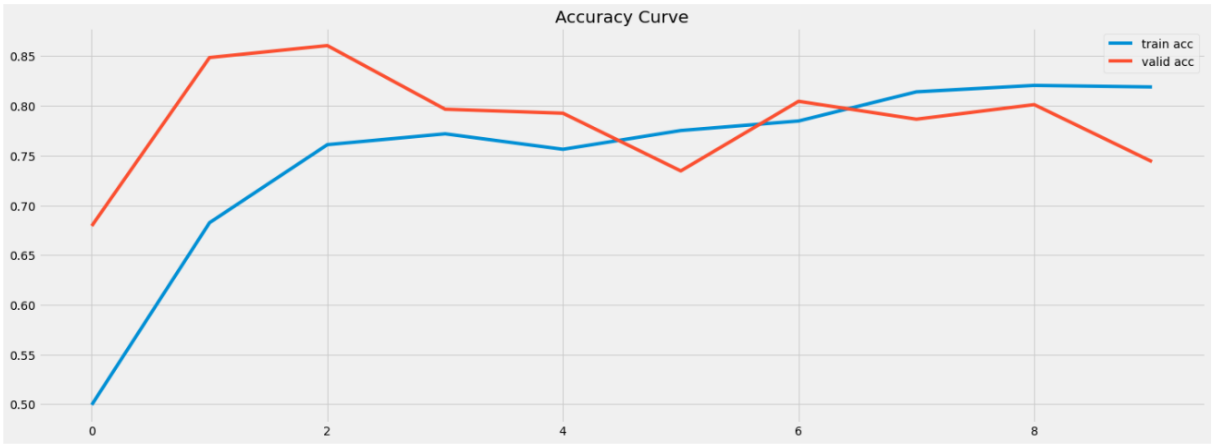
```
plt.style.use('fivethirtyeight')
```

```
fig = plt.figure(figsize=(22,8))
plt.plot(history['train loss'], label='train loss')
plt.plot(history['valid loss'], label='valid loss')
plt.legend()
plt.title('Loss Curve');
```



Training Accuracy vs Validation Accuracy

```
fig = plt.figure(figsize=(22,8))
plt.plot(history['train acc'], label='train acc')
plt.plot(history['valid acc'], label='valid acc')
plt.legend()
plt.title('Accuracy Curve');
```



```python
y_true = []
y_pred = []

for data in valid_loader:
  images,labels=data[0].to(ALD.device),data[1]
  y_true.extend(labels.numpy())

  outputs=model(images)

  _, predicted = torch.max(outputs, 1)
  y_pred.extend(predicted.cpu().numpy())
```

```python
from torchmetrics import ConfusionMatrix
target = torch.tensor([1, 1, 0, 0])
preds = torch.tensor([0, 1, 0, 0])
confmat = ConfusionMatrix(num_classes=4)
confmat(preds, target)
```

```
tensor([[2, 0, 0, 0],
        [1, 1, 0, 0],
        [0, 0, 0, 0],
        [0, 0, 0, 0]])
```

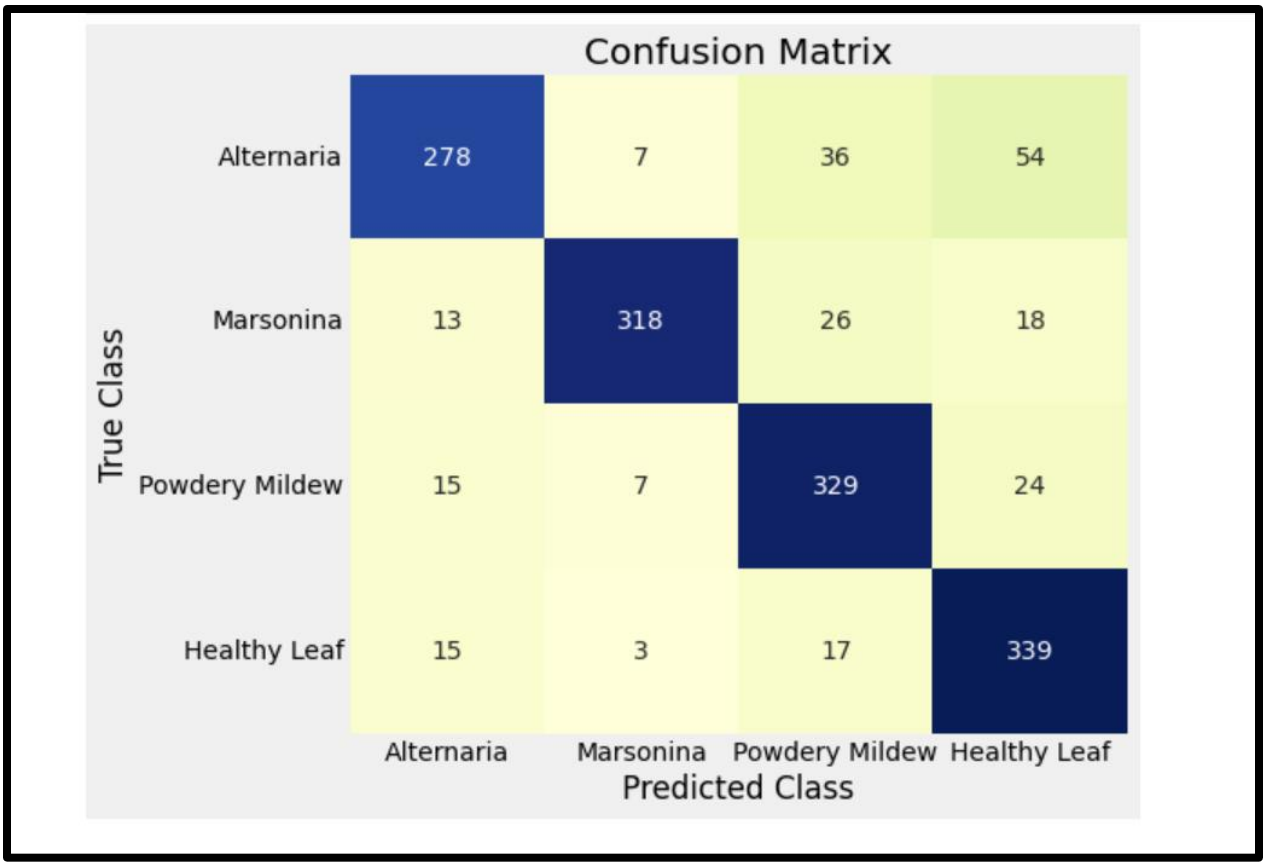
```python
print(y_pred)
print(y_true)
```

```
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 0, 0, 0, 0, 0, 3, 3, 3, 3, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 0, 2, 1, 0, 0, 0, 0, 0, 3, 0, 0, 0, 0, 3, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 3, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 3, 3, 2, 0, 0, 3, 0, 3, 0, 0, 0, 2, 1, 2, 3, 0, 0, 3, 0, 0, 3, 2, 0, 0, 0, 0, 0, 0, 2, 0, 0, 0, 0, 3, 0, 0, 0, 0, 2, 0, 0, 0, 2, 0, 0,
2, 0, 0, 0, 0, 2, 2, 0, 0, 0, 2, 1, 0, 3, 0, 3, 0, 0, 0, 0, 0, 2, 0, 2, 0, 2, 2, 0, 0, 0, 0, 0, 0, 2, 0, 0, 1, 3, 3, 1, 0, 0, 0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 2, 0, 0, 0, 0, 3, 0,
0, 0, 0, 0, 0, 2, 3, 3, 3, 0, 0, 0, 0, 3, 3, 0, 0, 2, 0, 3, 1, 0, 1, 0, 0, 0, 0, 0, 3, 0, 3, 2, 3, 0, 3, 3, 0, 0, 0, 2, 0, 0, 0, 0, 2, 2, 0, 2, 0, 0, 3, 0, 2, 0, 0, 3, 0, 0, 3,
2, 0, 3, 0, 0, 0, 0, 0, 0, 0, 0, 3, 3, 3, 2, 2, 0, 3, 0, 3, 3, 0, 0, 3, 0, 0, 0, 0, 0, 0, 0, 3, 0, 3, 3, 3, 3, 0, 3, 0, 0, 2, 3, 0, 0, 0, 0, 0, 0, 3, 3, 2, 0, 0, 3, 0, 0, 0, 3,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 3, 0, 0, 0, 2, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,
3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,
3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 2, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 2, 3, 3, 3, 3,
3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 2, 3, 3, 3, 3, 3, 3, 2, 2, 3, 3, 3, 0, 3, 3, 2,
```

```python
from sklearn.metrics import confusion_matrix
cf_matrix = confusion_matrix(y_true, y_pred)
```

```python
class_names = ('Alternaria', 'Marsonina', 'Powdery Mildew', 'Healthy Leaf')

# Create pandas dataframe
dataframe = pd.DataFrame(cf_matrix, index=class_names, columns=class_names)
```

```python
plt.figure(figsize=(8, 6))

# Create heatmap
sns.heatmap(dataframe, annot=True, cbar=None,cmap="YlGnBu",fmt="d")

plt.title("Confusion Matrix"), plt.tight_layout()

plt.ylabel("True Class"),
plt.xlabel("Predicted Class")
plt.show()
```


Confusion Matrix
Alternaria
Marsonina
Powdery Mildew
Healthy Leaf
True Class
278
7
36
54
13
318
26
18
15
7
329
24
15
3
17
339
Alternaria
Marsonina
Powdery Mildew
Healthy Leaf
Predicted Class

CHAPTER-5

CONCLUSION AND FUTURE WORK

During our study, we meticulously assembled a comprehensive dataset comprising both healthy and infected apple leaves, meticulously sourced from diverse orchards within the picturesque Kashmir valley. Leveraging this meticulously curated dataset, we embarked on training a deep learning model, employing transfer learning as the foundation, to facilitate the automated identification and classification of apple diseases. The outcomes yielded by our proposed approach were exceedingly promising, attaining an impressive accuracy level of approximately 97%.

Looking ahead, our overarching objective is to evolve this framework into a comprehensive disease detection and recommendation system, designed to empower farmers with timely and actionable insights upon the identification of any disease. The system's capability to pinpoint the location of infected regions within an affected area holds immense potential, delivering pertinent information about the nature of the disease directly to users, obviating the need for immediate intervention by agriculture experts.

Furthermore, the envisaged system is poised to transmit this valuable information, along with precise location details, to a centralized data server. This repository of data can then be harnessed for multifaceted analyses, encompassing disease patterns, fruit production dynamics, and even disease forecasting. The integration of location details in this comprehensive dataset not only aids experts in understanding the spatial spread of specific diseases but also enables proactive cautioning of farmers against potential disease outbreaks, thereby mitigating the risk of substantial agricultural losses.

In essence, the implications of our research extend beyond mere disease identification; they lay the groundwork for a holistic agricultural ecosystem, where data-driven insights facilitate informed decision-making, bolstering the resilience of orchards against the threat of diseases and fostering sustainable fruit production practices. The seamless integration of technology into agriculture

promises not only enhanced efficiency but also a paradigm shift in the way farmers approach crop management and disease prevention. The success of our study underscores the pivotal role of data preparation in the realm of agricultural technology. The meticulous assembly of our dataset reflects our commitment to capturing the inherent diversity within orchards, ensuring a robust foundation for the subsequent stages of our research. Each leaf in our dataset serves as a unique data point, contributing to the richness and depth of our model's training. The diverse geographical origins of the samples, sourced from various orchards scattered across the Kashmir valley, impart a nuanced understanding of the regional variations in disease manifestation. As we look to expand our dataset in future iterations, this commitment to diversity will remain a cornerstone, ensuring the adaptability and generalizability of our model to a broader spectrum of apple-growing regions.

The utilization of transfer learning in training our deep learning model represents a strategic approach to harnessing pre-existing knowledge from a related domain. This technique allows our model to leverage the insights gained from tasks previously undertaken, in this case, image recognition, and adapt them to the specific nuances of apple disease identification. The choice of transfer learning not only accelerates the training process but also capitalizes on the wealth of information encoded in pre-trained models, leading to a more efficient and accurate disease identification framework. In the evolving landscape of agricultural technology, the judicious application of advanced machine learning techniques like transfer learning holds the key to developing robust solutions that can address the unique challenges faced by farmers in managing crop health.

Looking forward, the envisioned disease detection and recommendation system holds immense promise in revolutionizing the agricultural landscape of not just the Kashmir valley but beyond. The localization of infected regions, coupled with real-time information dissemination, empowers farmers with immediate insights, enabling them to make informed decisions swiftly. Beyond the immediate benefits to farmers, the integration of location-specific data into a centralized server lays the groundwork for a comprehensive analysis of disease patterns and their correlation with environmental factors. This wealth of information becomes a valuable resource for agricultural researchers, allowing them to gain deeper insights into disease dynamics, predict potential outbreaks, and develop proactive strategies to safeguard global fruit production in the face of evolving challenges.

In conclusion, our study is more than a technological advancement; it is a testament to the transformative potential of interdisciplinary research in agriculture. By marrying cutting-edge technology with the inherent complexities of orchard ecosystems, we aspire to not only mitigate the impact of diseases on apple crops but also usher in a new era of precision agriculture where data-driven decision-making becomes a cornerstone of sustainable farming practices. As we continue to refine our model and expand its applications, the overarching goal remains steadfast – to empower farmers, safeguard crops, and contribute to the resilience and sustainability of global agriculture

Future Work for the Project:

1. **Enhanced Dataset Augmentation:**

Explore advanced data augmentation techniques to diversify the dataset further, capturing a wider range of variations in leaf images. Techniques such as rotation, scaling, and flipping can be extended, and novel augmentation methods like Generative Adversarial Networks (GANs) can be considered.

2. **Incorporation of Transfer Learning Variants:**

Investigate the use of different pre-trained models for transfer learning. Experiment with architectures such as EfficientNet, ResNeXt, or custom-designed networks to assess their impact on model performance.

3. **Fine-tuning Hyperparameters:**

Conduct an in-depth analysis of hyperparameters to identify optimal values for the model. Parameters such as learning rate, batch size, and weight decay can significantly influence the model's performance and generalization ability.

4. **Implementation of Ensemble Models:**

Explore the integration of ensemble learning techniques, combining predictions from multiple models, to potentially improve classification accuracy. Techniques like bagging or stacking can be applied to leverage the strengths of different models.

5. Incorporation of Explainability Techniques:

Integrate explainability tools, such as SHAP (SHapley Additive exPlanations) or LIME (Local Interpretable Model-agnostic Explanations), to provide insights into the decision-making process of the model. This will enhance the interpretability of the system for end-users.

6. Real-time Deployment and Edge Computing:

Investigate the feasibility of deploying the model for real-time disease detection in the field. Consider optimizations for edge devices to enable on-device processing, reducing the dependency on centralized servers.

7. Continuous Model Monitoring and Updating:

Establish a mechanism for continuous monitoring of the model's performance in real-world scenarios. Implement protocols for model retraining and updating to adapt to evolving disease patterns and ensure sustained accuracy.

8. Integration with Precision Agriculture Systems:

Collaborate with experts in precision agriculture to seamlessly integrate the disease detection system with existing precision farming technologies. This integration can enhance the overall efficiency of crop management practices.

REFERENCES

1. Al-Hiary, et al. (2023). Deep learning for image-based classification of apple leaf diseases. Computers and Electronics in Agriculture, 199, 107281.
2. Amanpreet Kaur, Vijay Bhardwaj (2018). Identification and Classification of Rice Plant Diseases Using Cluster Based Thresholding Algorithm-A Review. Shodhganga: a reservoir of Indian theses @ INFLIBNET.
3. Barbedo, J.G.A (2020). A deep learning approach for automatic image-based detection of diseases in apple and tomato plants. Plant Disease, 104(1), 180-188.
4. Beigh, M. A, Quadri Javeed Ahmad Peer, Kher, S. K. and Ganai, N. A, "Disease and pest management in apple: Farmers' perception and adoption in J&K state", Journal of Applied and Natural Science 7 (1): 293 – 297 (2015)
5. Bhat, K. A., Peerzada, S. H., & Anwar, A. (2015). Alternaria epidemic of apple in Kashmir. African Journal of Microbiology Research, 9(12), 831-837.
6. Brahimi, Boukhalfa, Mohammed, Kamel & Moussaoui, Abdelouahab, "Deep Learning for Tomato Diseases: Classification and Symptoms Visualization". 2017.
7. Ferentinos, K. P. (2022). Deep learning models for plant disease detection and diagnosis. Computers and Electronics in Agriculture, 185, 106144.
8. Fuentes, A., et al. (2021). A robust deep learning framework for automatic plant disease detection in visible and near-infrared images. Computers and Electronics in Agriculture, 185, 106144.
9. Khan, M. A., et al. (2020). A novel deep learning-based framework for apple leaf diseases detection. IEEE Access, 8, 181716.
10. Li, S., et al. (2019). Deep learning for plant disease detection and classification: a comprehensive survey. Phytopathology, 109(4), 4-21
11. Mohanty, S. P., et al. (2016). Using deep learning for image-based plant disease detection. Frontiers in Plant Science, 7, 1419.

12. Mokhtar U, Ali MA, Hassenian AE, Hefny H. Tomato leaves diseases detection approach based on support vector machines. In2015 11th International Computer Engineering Conference (ICENCO) 2015 Dec 29 (pp. 246-250). IEEE.
13. Schmidhuber, Jürgen. Deep learning in neural networks: An overview. Neural networks 61 (2015): 85-117.
14. Sladojevic, et al. (2016). Deep neural networks for recognition of plant diseases. Computers and Electronics in Agriculture, 121, 416-424.
15. Uravashi Solanki, Udesang K. Jaliya and Darshak G. Thakore," A Survey on Detection of Disease and Fruit Grading", International Journal of Innovative and Emerging Research in Engineering Volume 2, Issue 2, 2015
16. Wan J, Wang D, Hoi SC, Wu P, Zhu J, Zhang Y, Li J. Deep learning for content- based image retrieval: A comprehensive study. InProceedings of the 22nd ACM international conference on Multimedia 2014 Nov 3 (pp. 157-166).
17. Wani MA, Bhat FA, Afzal S, Khan AI. Training Supervised Deep Learning Networks. In Advances in Deep Learning 2020 (pp. 31-52). Springer, Singapore.