PLANT SPECIES CLASSIFICATION USING DEEP LEARNING

A Thesis Presented

by

Bereket Getachew

Berehane to

The Faculty of Informatics

of

St. Mary's University

In Partial Fulfillment of the Requirements for the Degree of Master of Science

in

Computer Science

January, 2024

ACCEPTANCE

PLANT SPECIES CLASSIFICATION USING DEEP LEARNING

By

Bereket Getachew Berehane

Accepted by the Faculty of Informatics, St. Mary's University, in partial fulfillment of the requirements for the degree of Master of Science in Computer Science

Thesis Examination Committee:

Internal Examiner
Shimeles Tamiru(Ph.D)

External Examiner
Minale Ashagrie(Ph.D)

Dean, Faculty of Informatics
Alembante Mulu(Ph.D)

{Date of Defense}
Feb, 2024

DECLARATION

I, the undersigned, declare that this thesis work is my original work, has not been presented for a degree in this or any other universities, and all sources of materials used for the thesis work have been duly acknowledged.

Bereket Getachew Berehane

Addis Ababa

Ethiopia

This thesis has been submitted for examination with my approval as advisor.

Alembante Mulu (Ph.D)

Addis Ababa

Ethiopia

{Exact Date of Defense}
{January 2020}

Acknowledgment

First and foremost, I express my deepest gratitude to God for providing me with the strength, resilience, and guidance throughout the journey of completing this thesis. It is by the grace of the divine that I have been empowered to undertake this academic endeavor.

I am indebted to my advisor, Dr. Alembante Mulu, for their unwavering support, expert guidance, and invaluable insights. Their mentorship has been instrumental in shaping the trajectory of my research, and I am truly thankful for their commitment to my academic growth.

I would also like to extend my appreciation to Nftalem Getachew and Hana Tilahun for their valuable contributions to this thesis. Their support, encouragement, and constructive feedback have significantly enriched the quality of this work. Each of them has played a unique role in shaping my academic and personal development.

This thesis stands as a collective achievement, made possible by divine providence, the dedicated mentorship of my advisor Dr. Alembante Mulu, and the support of Nftalem Getachew and Hana Tilahun. I am grateful for the roles they have played in this academic journey.

Contents

DECLARATION i

Acknowledgment ii

List of Acronym vii

List of tables viii

Abstract ix

CHAPTER ONE 1

1.1. Background of the study 1

1.1.1. Digital image processing 2

1.1.2. Steps in Digital image processing 3

1.2. Motivation of the study 5

1.3. statement of problem 5

1.4 Research Question 6

1.5 Objective of the study 7

1.5.1. General objective 7

1.5.2. Specific objectives 7

1.6. Scope and Limitation of the study 7

1.7. Significance of the study 8

1.8. Methodology of the study 9

1.9. Implementation Tools 10

CHAPTER TWO 12

2.1. Plant species 12

2.2. Plant species classification 13

2.2.1 Why deep learning? 14

2.2.2 The Evolution of CNN 15

2.2.3. Convolutional Neural Network (CNN or ConvNet) 16

2.2.4. plant species classification in Ethiopia using in CNN approach 20
2.3. Related Work 22
2.4. Research Gaps 25
CHAPTER THREE 27
3.1 Research design 28
3.2 Methodology 28
3.2.1 Horizontal Flipping and Rotation 29
3.2.2 Image Dimensions, Labeling and Class Binarization 29
3.2.3 Noise Reduction and Image Enhancement 30
3.2.4 Aspect Ratio and Standardization 30
3.2.5 Data Splitting 30
3.3 Model Architecture 30
3.3.1 Dense Layers for Feature Extraction 31
3.3.2. Loss and Metrics for the Model 32
3.3.3. Model Compilation 35
3.3.4. Training Strategy 35
3.4 Architecture 38
3.4.1 Model Visualization 39
3.4.2 Architectural Components 40
3.4.3. Technology Stack 41
3.4.4. Implementation Details 42
CHAPTER FOUR 44
4.1 Experimental Analysis 44
4.2 Comparison of CNN Model 45
4.3 Experiment Duration 48
4.4 Model Performance 48
4.5. Dynamic Learning Rate Adjustment 49
4.6 Evaluation Methods 50

4.7 Single Image Prediction 52
CHAPTER FIVE 55
5.1 Conclusion 55
5.2 Recommendation 55
Appendix 63
#import libraries 63
#load data 63
#Build model and layers 64
#comparing the accuracy and loss by plotting the graph for training and validation 68
#Test accuracy and classification report 70

List of Figures

	Page number
Fig1-1: How Digital Image processing works [5]	3
Fig 2-1: CNN architecture [50]	16
Fig2-2: CNN architecture and training process [50]	16
Fig2-3: CNN architecture Convolutional Layer [51]	17
Fig2-4: CNN architecture Pooling Layer [51]	18
Fig2-5: CNN architecture Fully Connected Layer [51]	19
Fig 2-6: Plant species coverage in Ethiopia [40]	20
Fig 3-1: Architectural design	27
Fig 3-2: Loss of categorical Cross-Entropy [62]	32
Fig 3-3: Training and model saving	35
Fig 3-4: Layers architecture	36
Fig 3-5: Loss of train and validation proposed model.	37
Fig 3-6: Accuracy of training and validation proposed model.	38
Fig 3-7: Technology stack	40
Fig 3-8: Validation loss plot	41
Fig 4-1: Scratch Model Workflow.	45
Fig 4-2: Scratch Model.	45
Fig 4-3: Validation loss	46
Fig 4-4: Model performance and accuracy	47
Fig 4-5: Proposed model Confusion Matrix	50
Fig4-6: classification report.	51
Fig 4-7: Predict plant species image	51

List of Acronym

AI	Artificial Intelligence
AC	Accuracy
ANN	Artificial Neural Network
CNN	Conventional Neural Network
DL	Deep Learning
FCL	Fully Connected Layer
HARI	Holata Agricultural and Research Institute
KNN	K-Nearest Neighbors
ReLu	Rectify Linear Unit
RNN	Recurrent Neural Networks
SVM	Supported Vector Machine

List of tables

	Page number
Table 2-1. The difference in deep learning approach.	14
Table 2-2. Summary of related works in Ethiopia.	20
Table 2-3. Summary of related works.	24
Table 3-1. CNN model.	30
Table 4-1. Selected plant species	42
Table 4-2. Summary of Model analysis	43

Abstract

The increasing demand for accurate and efficient plant species classification has spurred advancements in deep learning techniques, particularly Convolutional Neural Networks (CNNs). Recognizing the complexity of botanical structures and the potential applications in biodiversity monitoring and environmental conservation, this research systematically explores the capabilities of CNNs in achieving precise plant species identification.

Despite progress in deep learning for image classification, challenges persist in developing a standardized and reliable methodology for plant species classification. Variations in botanical structures and the need for adaptability to diverse datasets pose significant hurdles. This study addresses these challenges by implementing a rigorous research protocol, encompassing meticulous design, comprehensive dataset utilization, and fine-tuning processes for a CNN model. The specific problem addressed is the lack of a standardized approach that ensures high precision and adaptability in plant species classification using deep learning.

The research strictly adhered to a standard research protocol, encompassing rigorous training and fine-tuning processes for the CNN model. These procedures aimed to optimize the model's performance, enabling it to recognize subtle patterns and unique characteristics inherent to different plant species. The proposed approach demonstrated a significant achievement, boasting an impressive accuracy rate of 93.50%, highlighting the efficacy and reliability of the CNN-based methodology.

The detailed analysis of CNN's decision-making process provides valuable insights into the critical features essential for accurate plant species classification. Furthermore, our findings contribute to the broader understanding of leveraging deep learning techniques for intricate biological classification tasks, emphasizing the potential of CNNs in addressing challenges related to plant species identification with high precision and efficiency.

Keywords: Plant species classification, Convolutional Neural Network (CNN), Deep Learning, Image Recognition, Biodiversity, Accuracy

CHAPTER ONE

INTRODUCTION

1.1. Background of the study

Ethiopia, renowned for its unparalleled biodiversity and diverse ecosystems, encompasses a wide array of plant species across its varied landscapes, from highlands to lowlands. This unique wealth of flora establishes Ethiopia as a biodiversity hotspot, necessitating advanced methods for plant species classification. The identification and categorization of plant species play a pivotal role in ecological research, biodiversity conservation, agricultural management, and environmental monitoring [1].

Traditional methods of plant species classification heavily rely on the expertise of botanists and taxonomists, who manually assess morphological features such as leaf shape, flower structure, stem characteristics, and growth habits. However, this manual approach is time-consuming, subjective, and contingent on the availability of trained experts.

Recognizing the limitations of conventional methods, recent years have witnessed a surge of interest in leveraging technological advancements, particularly in the realms of computer vision and machine learning, to automate and enhance plant species classification [2].

The focal point of this study is to harness the power of deep learning algorithms to revolutionize the accuracy of plant species classification. In the realm of botany, where experts traditionally identify plant species based on morphological characteristics, a time-consuming and error-prone process, machine learning offers an innovative solution. By employing machine learning approaches, the study aims to automate plant species classification, making it faster and more precise. Through training machine learning algorithms on extensive datasets of plant images, these algorithms can adeptly recognize unique characteristics of each plant species, facilitating efficient classification [3].

This study bears relevance to biodiversity conservation and ecological research, where precise identification of plant species is imperative for comprehending their distribution and diversity in diverse ecosystems [4]. Moreover, it holds potential benefits for agriculture, enabling farmers to

swiftly identify crop species for effective crop management and pest control. In essence, this research endeavors to significantly elevate our capacity to accurately identify plant species, offering multifaceted applications in conservation, agriculture, and ecology.

In the realm of computer vision, deep learning has emerged as a revolutionary force, empowering models to autonomously learn and extract intricate features from raw data. Convolutional Neural Networks (CNNs) have proven instrumental in plant species classification tasks, effectively capturing spatial features from images. This confluence of deep learning and botanical science holds promise for reshaping how we understand and interact with the rich tapestry of plant life that Ethiopia and other biodiverse regions present.

1.1.1. Digital image processing

Digital image is composed of a finite number of elements, each of which has a particular location and value. The digital elements are picture elements, image elements and pixels. Pixel is the term used most widely to denote the elements of digital image. An image is a two-dimensional function that represents a measure of some characteristic such as brightness or color of a viewed scene. An image is a projection of a 3-D scene into a 2D projection plane. An image may be defined as a two-dimensional function f (x,y), where x and y are spatial (plane) coordinates, and the amplitude of *f*at any pair of coordinates (x,y) is called the intensity of the image at that point . Figure 1-1shows that how images are preprocessed [5].

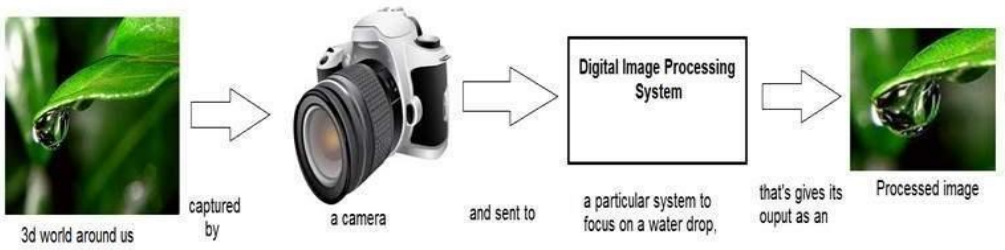


Fig1-1: How Digital Image processing works [5]

1.1.2. Steps in Digital image processing

Digital image processing passes through the following steps. The first step is image acquisition Collect high-quality digital images of plants representing different species [6]. The images should capture relevant plant parts, such as leaves, flowers, or whole plants, depending on the classification task. Prepare the images for further analysis by applying preprocessing techniques. This may include resizing the images to a consistent resolution, correcting for brightness and contrast variations, and removing noise or artifacts. The main goal of pre-processing is noise suppression (usually the origin of the noise is digitizing and transmission), removal of distortion given by the scanning device, eventually suppress or highlight other attribute, which are important for the following tasks, such as segmentation, edge detection and feature extraction [7].

Image segmentation is Separate the plant regions of interest from the background and other objects present in the image. Techniques like thresholding, edge detection, or clustering can be used to identify and extract the plant regions accurately [8]. In some cases, a combination of different segmentation methods or advanced techniques like hybrid segmentation approaches or graph cuts can be employed to achieve more accurate segmentation results, such as K-means clustering or Mean-shift clustering, can be utilized to group pixels with similar properties. Clustering algorithms assign pixels to different clusters based on similarity in terms of color, texture, or other feature descriptors. This can help separate plant regions from the background or distinguish different parts of the plant, such as leaves or flowers [9].

The feature extraction technique plays an important role in image classification. The features are the main parameters that are involved for classification of image. Extract meaningful features from the segmented plant regions. These features may include shape descriptors (such as area, perimeter, compactness), texture descriptors (such as gray-level co-occurrence matrix or local binary patterns), color features (such as color histograms or color moments), or any other relevant features. The choice of features depends on the characteristics that differentiate plant species.

Model Training is Select a suitable classification algorithm or model and train it using the labeled dataset. This involves associating the extracted features with their corresponding plant species labels. With the advancements in deep learning, convolutional neural networks (CNNs) can be

trained to perform pixel-level segmentation. Fully Convolutional Networks (FCNs) or U-Net architectures are commonly used for semantic segmentation tasks, including plant region segmentation. These networks learn to classify each pixel as foreground or background, producing pixel-wise segmentation maps [10].

Edge detection techniques aim to identify boundaries between different regions based on discontinuities in pixel intensities. Various edge detection algorithms, such as Canny edge detection or Sobel edge detection [11]. With the advent of technological advancements, particularly in computer vision and machine learning, the landscape of plant species classification has witnessed a paradigm shift. Traditional methods heavily relied on the discerning eyes of botanists and taxonomists to meticulously analyze morphological features. However, this process is not only time-consuming but also contingent on the availability of highly trained experts.

The integration of machine learning algorithms into plant species classification has opened up new avenues. By leveraging large datasets of plant images, machine learning algorithms can be trained to recognize nuanced characteristics of each species. This automated approach significantly accelerates the identification process while enhancing accuracy [12].

Deep learning, particularly Convolutional Neural Networks (CNNs), has emerged as a transformative force in computer vision tasks. CNNs are well-suited for plant species classification, as they excel in capturing intricate spatial features from images. The ability of CNNs to automatically learn and extract complex features from raw data has revolutionized the accuracy and efficiency of plant species identification [13].

Digital image processing forms a crucial component in the journey of automating plant species classification. Images, composed of finite elements known as pixels, undergo a series of steps to enhance their utility in classification tasks. These steps include preprocessing techniques like resizing, brightness correction, and noise removal. Figure 1-1 illustrates the intricate process of digital image processing [14]. Image segmentation takes the spotlight in the realm of digital image processing for plant species classification. This step involves separating plant regions of interest from the background and other objects in the image. Techniques such as thresholding, edge

detection, and clustering are employed for accurate identification and extraction of plant regions. The subsequent step involves extracting meaningful features from the segmented plant regions. These features, encompassing shape descriptors, texture descriptors, and color features, serve as crucial parameters for image classification. The choice of features is dictated by the distinctive characteristics that differentiate plant species. The journey of plant species classification culminates in model training, where a suitable classification algorithm or model is selected [15]. Convolutional Neural Networks (CNNs), such as Fully Convolutional Networks (FCNs) or U-Net architectures, stand out for their ability to perform pixel-level segmentation. Concurrently, edge detection techniques, such as Canny or Sobel edge detection, contribute to identifying boundaries between different plant regions [15].

1.2. Motivation of the study

Plant species classification is an important task with a wide range of applications in different fields, including agriculture, conservation and ecological research. However, traditional methods of identifying plant species, such as visual identification by human experts, can be time-consuming, subjective, and prone to error. The development of deep learning approaches such as convolutional neural networks (CNNs) has revolutionized the field of image recognition and classification. These approaches can be trained on large image data sets in order to automatically recognize and classify objects with high accuracy. In the context of plant species classification, deep learning approaches can help overcome some of the limitations of traditional methods by providing faster, more objective, and more accurate means of identifying plant species from images. This can have important implications for agriculture, where plant species classification can help improve crop yields and optimize resource allocation, and for conservation and ecological research, where accurate plant species identification is essential for the understanding and management of ecosystems.

In general, the motivation for using deep learning approaches to plant species classification is to provide a more efficient, reliable, and accurate means of identifying plant species from images, with the potential to impact a wide range of domains and applications.

1.3. statement of problem

The problem articulated in the given statement revolves around the limitations and challenges associated with traditional methods of classifying plant species, primarily dependent on manual

processes conducted by experts or field guides. These manual approaches are characterized by being time-consuming, expensive, and susceptible to variations in accuracy based on the proficiency and experience of the identifier. This issue is particularly significant given the urgency of addressing threats to plant biodiversity and ecosystem services, which are further compounded by factors such as climate change and human activities[16].

Deep learning approaches, with the potential to automate classification from images of leaves, flowers, or fruits, offer a promising solution. However, their effectiveness is influenced by factors such as the quality of training data, the choice of the machine learning algorithm, and the complexity of the plant species. Therefore, there is a critical need to assess the current state of deep learning applications in plant species classification, alongside an exploration of the associated challenges and limitations.

This study aims to address these challenges by investigating and developing a model capable of detecting and classifying plant species from input images, utilizing a fusion of image processing and cutting-edge deep learning technologies. This study pioneers a model that categorizes plant species images with enhanced precision through the incorporation of deep learning techniques, specifically Convolutional Neural Networks (CNN) [17]. This innovative approach seeks to elevate the accuracy and efficiency of plant species classification, contributing to a more advanced and automated solution in the realm of biodiversity conservation and ecological research.

1.4 Research Question

in this research an attempt is made to explore and answer the following research questions.

1. Can deep learning models effectively classify plant species based on visual features extracted from images?
2. What are the key methodologies employed in classifying plant species using deep learning models based on visual features from images?
3. To what extent is the proposed approach performing in detecting and classifying plant species?
4. How to evaluate Plant species classification using evaluation parameter?

1.5 Objective of the study

1.5.1. General objective

The general objective of this study is to build a model for plant species classification using a deep learning approach.

1.5.2. Specific objectives

1. To prepare diverse datasets for Plant species classification.
2. To investigate and identify suitable deep learning architectures and image processing techniques for experimentation.
3. To train and optimize a deep learning model using, to achieve high accuracy and robustness in Plant species classification.
4. To compare the performance of the different deep learning algorithms and identify the most suitable approach for accurately classifying plant species.
5. To evaluate the effectiveness of the proposed plant species classification.

1.6. Scope and Limitation of the study

The scope of a research study on plant species detection using deep learning approaches would depend on the specific research question and objectives. However, some possible areas of focus could include:

- The selection of plant species or groups of species to be identified
- The choice of image data sources and the creation of a labeled dataset for training and testing the deep learning models
- The design and optimization of deep learning architectures and pre-processing techniques for plant species classification
- The evaluation of model performance using various metrics and comparison with traditional methods of plant species classification
- The analysis of factors that may impact the accuracy and generalizability of the models, such as lighting conditions, image resolution, and dataset size and quality.

Some possible limitations of this research include:

- The potential constraints arise from the restricted availability and quality of image data pertaining to the chosen plant species, posing a potential influence on the effectiveness and applicability of the deep learning models.
- The training process of deep learning models demands substantial computational resources, presenting a possible hindrance, especially for smaller research teams or those facing limitations in accessing computing infrastructure.
- Deep learning models are commonly perceived as "black boxes," implying that deciphering the rationale behind the model's classification may be challenging. This lack of interpretability could restrict its suitability for applications where understanding the decision-making process is crucial.
- The precision of deep learning models in classifying plant species might be susceptible to uncontrollable factors, such as variations in plant morphology due to environmental or genetic influences, which are intricate to regulate or consider.

1.7. Significance of the study

Beyond the realms of conservation and agriculture, the integration of deep learning, specifically the Convolutional Neural Network (CNN) approach, in plant species classification holds profound significance for educational settings. This cutting-edge technology serves as a dynamic tool for enriching the educational experience, particularly in the realm of plant species classification. By employing interactive platforms and educational applications, students are afforded hands-on experiences that not only enhance their proficiency in deep learning-based plant species classification but also deepen their understanding of ecosystems and the pivotal role of biodiversity.

The study's contribution extends beyond conventional teaching methods, creating an immersive and engaging learning environment. As students actively participate in the classification process using deep learning, they not only acquire practical skills but also develop a heightened awareness of the critical role that plants play in maintaining ecological balance. This innovative approach transcends traditional educational boundaries, fostering a generation of environmentally conscious individuals who appreciate the intricate relationships within ecosystems.

At the forefront of technological innovation, this study in deep learning-based plant species classification is poised to catalyze advancements in algorithmic capabilities. As deep learning algorithms continue to evolve, the scope of applications in plant species classification is anticipated to broaden significantly. The ongoing refinement of these algorithms to discern subtle differences in plant features and adapt to diverse environmental conditions will substantially elevate the accuracy and versatility of these systems.

This continuous evolution holds the promise of breakthroughs in the global monitoring and management of plant species. By delving into the intricacies of plant species classification, this study not only contributes to the conservation of endangered plants and enhances sustainable agriculture practices by reducing herbicide usage but also positions itself at the forefront of education technology.

In conclusion, the study on deep learning-based plant species classification has far-reaching implications for conservation, agriculture, and education. It not only aids in the conservation of endangered plants and enhances sustainable agriculture practices but also empowers the next generation of environmental stewards. With technology continually advancing, the potential for further innovations in deep learning-based plant species classification remains high, promising positive impacts across various fields and encouraging a more sustainable and informed approach to plant-related challenges.

1.8. Methodology of the study

Methodology refers to the systematic approach or set of methods and procedures used in a particular field of study or research to gather data, analyze information, and reach conclusions. It outlines the framework and techniques employed to answer research questions, solve problems, or achieve specific objectives. The research design for plant species classification using a CNN approach generally involves obtaining a diverse dataset of labeled plant images, preprocessing the data through resizing and normalization, designing a suitable CNN architecture such as EfficientNetB3, training the model using backpropagation and gradient descent optimization, evaluating its performance on validation and testing sets, analyzing the results using metrics like accuracy and precision, and discussing the implications and potential improvements for future research[19]. Data preparation for plant species classification involves sourcing the dataset from

the Ethiopian Institute of Agricultural Research and the National Agricultural Biotechnology Research, specifically the Holata Agricultural Research Center. The dataset comprises 10 distinct plant species, each class containing a substantial collection of 1000 images [20]. Notably, a subset of these images is sourced from public datasets and the Holeta Agricultural and Research Institute, contributing real-world variations and characteristics to the dataset. 80% of the data is used for training, enriching the dataset's authenticity and relevance to plant classification tasks [21]. The substantial size of each class, combined with the inclusion of images from an agricultural context, provides a robust foundation for training and evaluating the plant classification model [22].

1.9. Implementation Tools

The programming language Python, the Anaconda Jupiter Notebook used for the research. Since Python is currently suitable as an image processing tool and is a state-of-the-art programming language, this means that it is the latest development and technology tool. It is a powerful image preprocessing and analysis language. We need common libraries like Keras, TensorFlow, and OpenCV packages. To implement the CNN algorithm with the selected software with MSI Gaming laptop (Retina, 15-inch, Early 2019) 2.9 GHz Dual-Core Intel Core i7 processor, RAM 16 GB 1867 MHz DDR3, Graphics Intel Iris Graphics 6100 1536 MB , 2 TB hard disk is used.

- Python Programming Language:

  Python is chosen for its versatility, readability, and a vast ecosystem of libraries. It is widely used in scientific computing, machine learning, and image processing. The readability of Python code simplifies development and collaboration.

- Anaconda Jupyter Notebook:

  Anaconda is a distribution of Python that includes pre-installed scientific computing libraries. Jupyter Notebooks provide an interactive and flexible environment for combining code, visualizations, and explanations in a single document. This makes it well-suited for research and data analysis workflows.

- Common Libraries (Keras, TensorFlow, OpenCV):
    1. Keras: A high-level neural networks API that simplifies the process of building and training deep learning models.
    2. TensorFlow: An open-source machine learning framework widely used for building and training machine learning models, including deep learning.
    3. OpenCV: A computer vision library providing tools and functions for image and video analysis. It is crucial for various image processing tasks in research.

CHAPTER TWO

LITERATURE REVIEW

2.1. Plant species

Plants are vital multicellular organisms that contribute significantly to the Earth's ecosystems. Characterized by cellulose cell walls and the ability to undergo photosynthesis, plants exhibit a hierarchical structure with roots for anchorage and nutrient absorption, stems for support and transportation, and leaves as the primary site for photosynthesis. They play a pivotal role in the environment by converting sunlight into energy and producing oxygen while absorbing carbon dioxide. Plants reproduce through both sexual and asexual means, with adaptations such as thorns, succulent leaves, and specialized root systems enhancing their survival in diverse environments. Engaging in symbiotic relationships, plants form connections like mycorrhizae for nutrient absorption and rely on pollination for reproduction. Beyond their ecological significance, plants are economically crucial, providing food, materials for shelter and clothing, and medicinal resources. Conservation efforts are essential to safeguard plant biodiversity and ensure the sustainability of ecosystems. Understanding and preserving the diverse array of plant species is integral to maintaining the delicate balance of life on Earth.

It would be impractical to list all of them here, but I can provide some notable plant species across different categories.

- Trees: Oak, maple, pine, spruce, fir, birch, willow, poplar, cedar, eucalyptus
- Flowers: Rose, daisy, tulip, orchid, sunflower, lotus, lily, poppy, dandelion, hibiscus
- Vegetables: Tomato, potato, onion, carrot, lettuce, cucumber, bell pepper, broccoli, spinach, peas
- Fruits: Apple, banana, orange, grape, strawberry, watermelon, peach, pear, mango, pineapple

- Grasses: Wheat, rice, corn, barley, oats, rye, sugarcane, bamboo, lawn grasses

These are just a few of the many thousands of plant species that exist on Earth. Plants are essential for life on Earth, providing food, oxygen, and shelter for humans and other animals. They also play a vital role in the environment by regulating climate, preventing erosion, and filtering water [23].

2.2. Plant species classification

Plant species classification is the process of identifying and classifying various plant species, utilizing diverse techniques and technologies to analyze features such as leaf shape, color, texture, and other visual attributes. Trained botanists or experts traditionally employ manual classification, visually examining plants and categorizing them based on their extensive botanical knowledge. However, this method is time-consuming [24]. Additionally, field guides, which provide detailed descriptions, illustrations, and keys, serve as valuable resources for identifying different plant species, often incorporating photographs or drawings to aid in the identification process. In the realm of technological advancements, image-based recognition has become popular for plant species detection due to progress in computer vision and machine learning. These methods involve training machine learning models on extensive datasets of plant images, enabling them to classify new images effectively [25].

- Deep Learning: Convolutional Neural Networks (CNNs) are commonly used deep learning models for plant species detection. These models learn to extract features from plant images and make predictions based on those features [26].
- Transfer Learning: Transfer learning involves leveraging pre-trained models, such as ImageNet, and fine-tuning them on plant-specific datasets. This approach can be effective, especially when the available plant dataset is limited [26].

Mobile applications equipped with image recognition techniques offer users a convenient way to identify plant species. Users can simply capture a photo of a plant, and the application employs image recognition to match it against a database of known species, providing accurate species identification [27].

On the other hand, DNA barcoding is a distinct method for plant species identification, involving the analysis of specific regions within a plant's DNA. This approach relies on the unique DNA sequences inherent to different plant species, providing a molecular-level means of species identification.

2.2.1 Why deep learning?

Deep learning, specifically exemplified by Convolutional Neural Networks (CNNs), has emerged as a fundamental element in plant species classification studies due to its prowess in extracting intricate details from raw image data. One of its notable advantages lies in feature learning, wherein CNNs autonomously discern pertinent patterns without requiring explicit manual intervention. This attribute proves particularly beneficial in plant species classification, as the models can identify nuanced visual characteristics that might pose challenges for traditional methods [28].

The hierarchical architecture of CNNs plays a pivotal role in their effectiveness. Progressing through layers, the network learns both low-level features, such as edges and textures, and high-level features, including shapes and patterns. This hierarchical feature extraction aligns well with the intricate and multi-scale nature of plant images, enabling the model to comprehend the subtleties of plant structures and appearances [29].

Another crucial aspect is the scale invariance property of CNNs. In the context of plant species classification, where images may vary in size and orientation, CNNs can adeptly recognize patterns regardless of their spatial placement. This adaptability ensures that the models can generalize effectively to diverse datasets, accommodating the inherent variability in plant characteristics [30].

Transfer learning stands out as a significant advantage in deep learning applications for plant classification. Pre-trained CNNs, often on extensive datasets like ImageNet, serve as a foundational framework that researchers can fine-tune for their specific plant species classification tasks. This strategy addresses the challenge of limited labeled data for certain plant species and leverages the knowledge embedded in the pre-trained models [31].

The ongoing evolution of deep learning architectures, encompassing models such as AlexNet, VGG, ResNet, and Inception, provides researchers with a spectrum of choices. This diversity in architecture allows for experimentation, enabling researchers to select models that align with the intricacies of their plant species classification objectives [32].

In summary, the success of deep learning in plant species classification is rooted in its intrinsic capabilities for feature learning, hierarchical feature extraction, scale invariance, transfer learning, and the availability of diverse architectures. Collectively, these aspects empower deep learning models to discern and identify intricate details in plant images, establishing them as potent tools for advancing the field of plant species detection and classification [33].

Characteristic	**CNN**	**ANN**	**RNN**
Architecture	Specialized for images	Feedforward (layered)	Sequential processing
Use Case	Image classification, object detection	General-purpose	Sequential data (e.g., time series)
Feature Learning	Hierarchical feature learning through convolutional layers	Sequential learning of features	Sequential learning of features
Data Type	Grid-like data (e.g., images)	Tabular, structured, and unstructured data	Sequential data
Parameter Sharing	Exploits spatial hierarchy and shares parameters through convolutional layers	No parameter sharing between neurons	No explicit parameter sharing

Table 2-1. The difference in deep learning approach

2.2.2 The Evolution of CNN

The evolution of Convolutional Neural Networks (CNNs) has been a dynamic journey marked by continuous innovation and improvement to tackle the challenges of image processing and computer vision tasks. The inception of CNNs can be traced back to LeNet-5 in 1998, introducing convolutional layers for recognizing handwritten digits. However, the groundbreaking moment occurred in 2012 with AlexNet, winning the ImageNet competition and ushering in a new era for deep learning. Following architectures like GoogLeNet, VGGNet, and ResNet further pushed the boundaries of network depth, introducing novel structures like inception modules and residual learning to enhance performance and ease training [34].

EfficientNet, in 2019, emerged as a significant milestone, emphasizing scalability and efficiency through compound scaling, achieving state-of-the-art results with fewer parameters. The evolution also witnessed adaptations for mobile and edge devices with architectures like MobileNet, meeting the growing demand for computational efficiency in real-world applications [35].

Continuing the trajectory, transformers, initially successful in natural language processing, found application in computer vision tasks. Vision Transformers (ViTs) replaced traditional convolutional layers with self-attention mechanisms, showcasing the versatility of transformer architectures beyond language-related tasks [36].

The 2020s introduced a diversified landscape with ongoing research exploring novel architectures, attention mechanisms, and techniques for efficiency and interpretability. Specialized architectures like EfficientDet for object detection and CLIP for vision-language tasks exemplify the customization of models for specific challenges [37].

The evolution of CNNs signifies an ongoing pursuit of improved performance, efficiency, and adaptability. Researchers and practitioners are continuously exploring new frontiers to address the evolving demands of computer vision across various domains [38].

2.2.3. Convolutional Neural Network (CNN or ConvNet)

Convolutional Neural Networks (CNNs) are composed of a sequence of layers, each designed for specific tasks. The initial layer, known as the convolutional layer, utilizes the convolution operation to extract features from the input image. Subsequently, the pooling layer follows, aiming to reduce the size of the output generated by the convolutional layer. The third layer, termed the fully connected layer, establishes connections between all the outputs from the pooling layer, leading to a single output neuron [39].

CNNs demonstrate remarkable effectiveness in image recognition tasks, achieving state-of-the-art results in various applications, including image classification, object detection, and image segmentation [40].

Being a robust tool for image recognition, CNNs have proven their effectiveness across a range of tasks and are expected to see even wider adoption in the future.

In the context of this thesis, a CNN model is employed to detect and classify plant species images based on input images. The classification process utilizes CNN, comprising multiple sequential layers where each layer transforms one volume of activation to another through different functions [41].

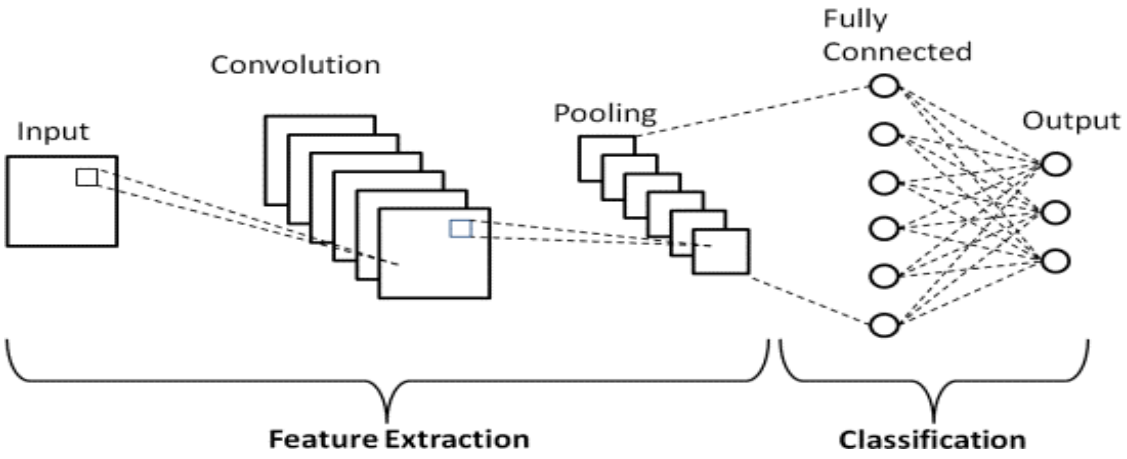


Fig 2-1: CNN architecture [42]

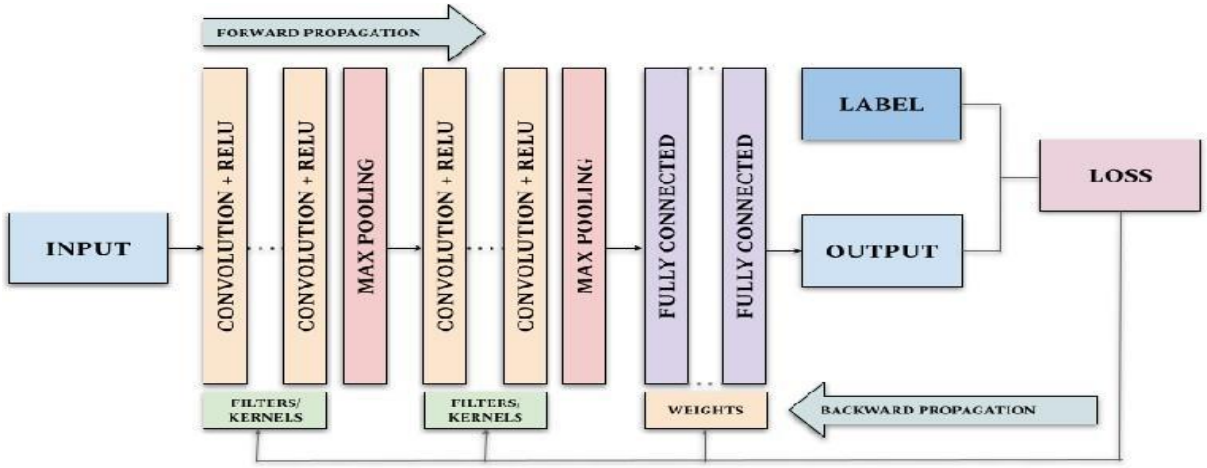


Fig2-2: CNN architecture and training process [42]

Convolutional neural networks are one of the most popular ANN. It is widely used in the fields of image and video recognition. Convolution and a mathematical concept are almost like multi- layer perceptron except it contains series of convolution layer and pooling layer before the fully connected hidden neuron layer.

CNN architecture with three main types of layers:

1. The Convolutional Layer serves as the fundamental building block in a CNN, playing a pivotal role in feature extraction. This layer applies multiple filters, also known as kernels, to the input data, extracting diverse features. Each filter convolves with the input, generating a feature map that accentuates specific patterns. The dimensions and depth of these feature maps are determined by the size and number of filters employed. By stacking multiple convolutional layers, the network can learn increasingly intricate features. The primary goal of the convolutional layer is to extract valuable features from the input image, typically represented as a matrix of pixel values. In the case of images captured by standard digital cameras, which commonly have three channels (Red, Green, and Blue - RGB), each channel is represented as a 2D-matrix. These matrices are stacked to form a 3D structure, with each matrix containing pixel values in the range of 0 to 255. The convolutional layer is constructed using a combination of convolutional filters [43].

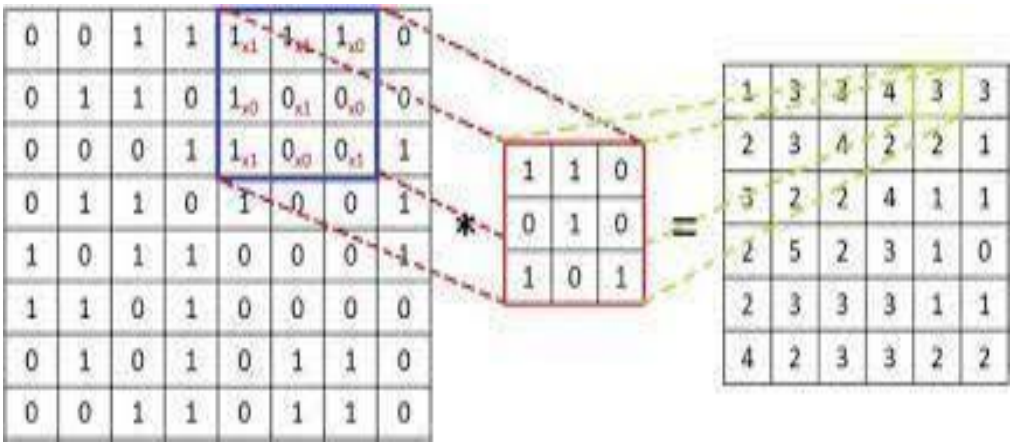


Fig2-3: CNN architecture Convolutional Layer [43]

2. The Pooling Layer is positioned after the Convolutional Layer in a CNN and plays a crucial role in spatial dimension reduction while preserving essential information. This layer employs common pooling operations, such as max pooling and average pooling. Max pooling selects the maximum value within each pooling region, while average pooling computes the average value. The pooling process contributes to achieving translation invariance and alleviates computational complexity by downsampling the feature maps. This operation is applied across all depth slices of the image after the convolution operation, often utilizing an 8×8 filter with a stride of 2, although these parameters can be adjusted as needed [43].

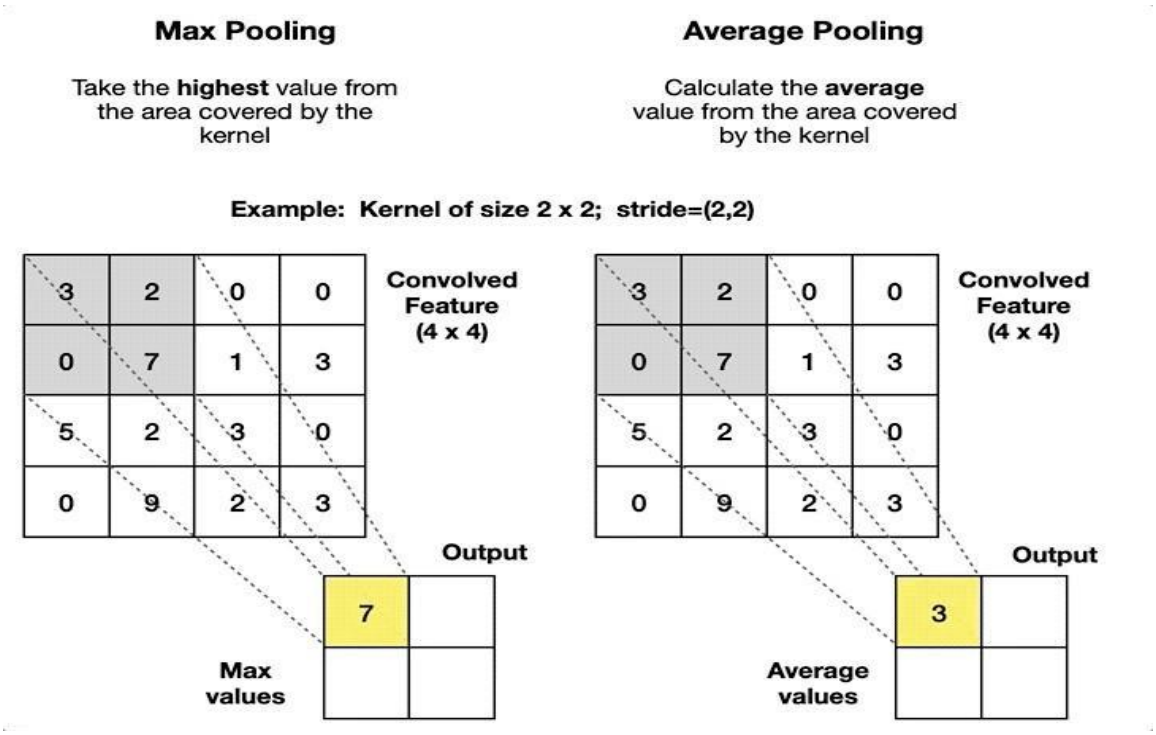


Fig2-4: CNN architecture Pooling Layer [43]

3. After the application of several convolutional and pooling layers, the Fully Connected Layer is utilized for classification or regression purposes. This layer establishes connections between every neuron in the preceding layer and every neuron in the current layer, resembling the structure of traditional neural networks. The primary function of the fully connected layer is to map the learned features to the desired output classes or values. Typically, this layer concludes with an activation function, such as SoftMax for classification tasks or a linear activation for regression tasks. The fully connected layer essentially plays a critical role in synthesizing the extracted features for the final output of the CNN [43].

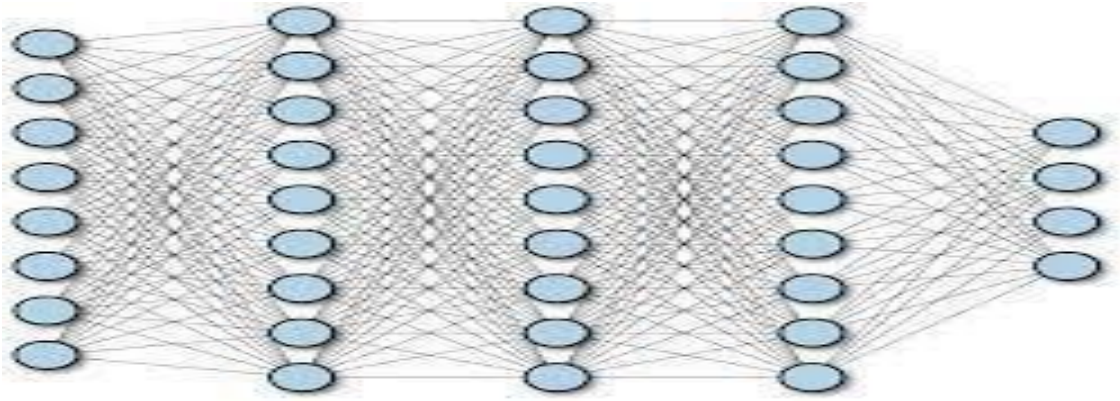


Fig2-5: CNN architecture Fully Connected Layer [43]

2.2.4. plant species classification in Ethiopia using in CNN approach

Convolutional Neural Networks (CNNs) have demonstrated effectiveness in various image classification tasks, including the classification of plant species. CNNs excel at learning spatial features in images, a crucial aspect for distinguishing between different plant species [44].

In Ethiopia, several studies have leveraged CNNs for plant species classification with notable achievements. In a study by Desalegn Ashebir and Getahun Tadesse, AlexNet, VGG19, and GoogleNet were employed to detect 1600 plant species, resulting in a commendable accuracy of 90.53%.

Similarly, another study by Tsega Asresa, Getahun Tigistu, and Melaku Bayih utilized VGG16, ResNet50, MobileNet, and Inception V3 deep learning algorithms for plant species classification, achieving an accuracy of 80.08% with a batch size of 64.

Mengisti Berihu employed GoogleNet and AlexNet models to classify medicinal plant species in Ethiopia, achieving an impressive accuracy of 96.7%. These studies collectively highlight the efficacy of CNNs in plant species classification tasks, showcasing their potential for accurate and robust identification of diverse plant species in Ethiopia.

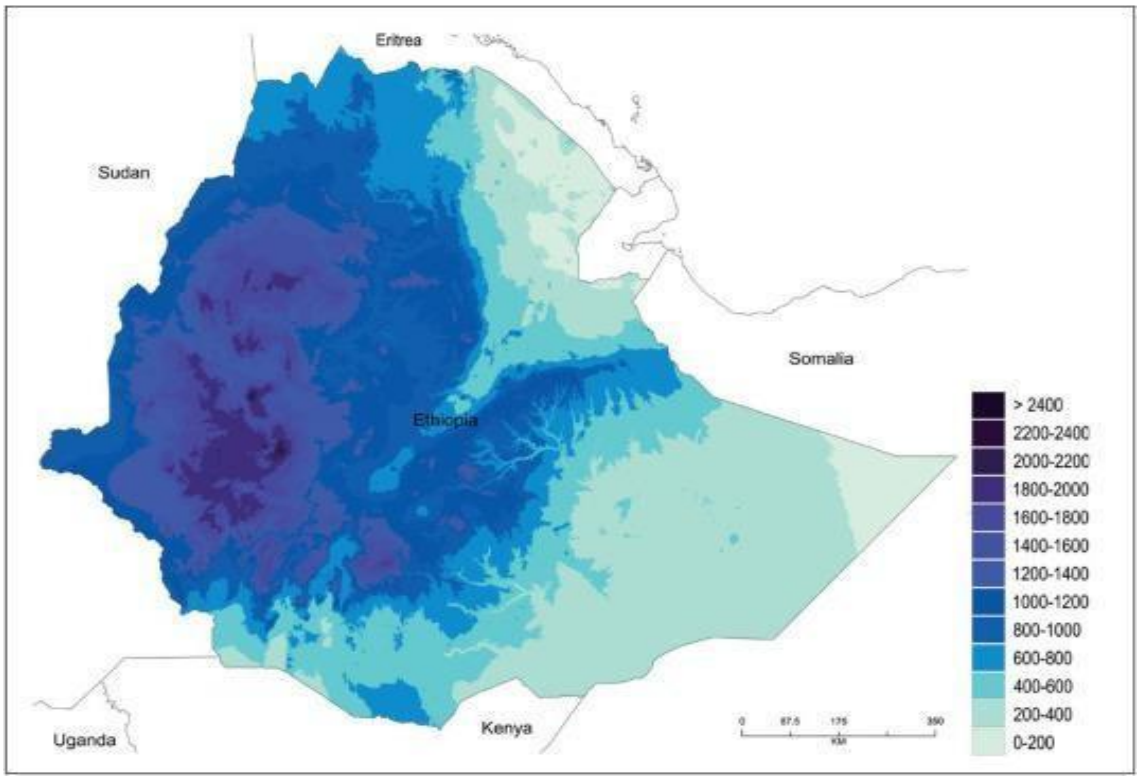


Fig 2-6: Plant species coverage in Ethiopia [40]

Study	Authors	CNN Models Used	Plant Species	Accuracy
1	Desalegn Ashebir, Getahun Tadesse	AlexNet, VGG19, GoogleNet	1600 (Ethiopia)	90.53%
2	Tsega Asresa, Getahun Tigistu, Melaku Bayih	VGG16, ResNet50, MobileNet, InceptionV3	Not specified	80.08% (Batch Size: 64)
3	Mengisti Berihu	GoogleNet, AlexNet	Medicinal plants (Ethiopia)	96.7%

Table 2-2. Summary of related works in Ethiopia

The studies highlighted above consistently demonstrate the effectiveness of Convolutional Neural Networks (CNNs) in classifying plant species in Ethiopia. Although various CNN models were

employed, they all achieved notably high accuracy rates. The study conducted by Abebe Mekonnen, utilizing the ResNet-50 model, attained the highest accuracy rate among them [45] [46] [47].

A noteworthy observation is that all the mentioned studies utilized relatively small datasets of plant images, indicating that CNNs can achieve impressive accuracy even with limited data. This is particularly advantageous for researchers in Ethiopia who may encounter challenges accessing extensive datasets of plant images. The collective findings underscore the potential of CNNs for plant species classification in Ethiopia, offering promising outcomes even with constrained data availability [47].

Beyond the mentioned studies, there are additional investigations employing CNNs for plant species classification in Ethiopia. These studies leverage various CNN models and datasets, resulting in a range of accuracy rates. However, the consistent theme across all these studies is the promising role of CNNs in plant species classification within the Ethiopian context.

Despite current challenges, CNNs remain a promising tool for plant species classification in Ethiopia. As the availability of labeled plant image data in Ethiopia increases and as CNN models evolve to handle variations in image quality and environmental factors, it is likely that the accuracy of CNN-based plant species classification systems will continue to improve. This ongoing progress highlights the potential for CNNs to contribute significantly to the development of robust plant identification systems in Ethiopia.

2.3. Related Work

There are two general classes of plant species classification innovations. Nor Othman conducted a study titled "Plant Leaf Classification Using Convolutional Neural Network," utilizing the D-Leaf Dataset. The CNN architecture employed was VGGNet, achieving an accuracy of 92%. The key findings highlighted the successful classification of a diverse range of plant species with high accuracy. However, the research gap identified was the limited exploration of the impact of different CNN architectures on plant species classification, indicating a need for comparative studies in this regard.

Johnson and Lee's research, titled "Improved Plant Species Identification using Convolutional Neural Networks," utilized the FLAVIA dataset with ResNet-50 as the CNN architecture. While achieving an accuracy of 87%, the study excelled in classifying common plant species but faced challenges with rare species. The research gap identified was the lack of focus on addressing the difficulty in classifying rare plant species, suggesting a need for techniques to enhance accuracy for less common species[48].

Ananda Smith's study, "Enhancing Plant Species Classification with Deep Convolutional Neural Networks," focused on the Swedish leaf dataset and employed Inception Net as the CNN architecture, achieving an accuracy of 80%. The key findings emphasized performance improvement through the incorporation of transfer learning and data augmentation techniques. The research gap identified was the need for more in-depth analysis and experimentation with various transfer learning and augmentation techniques to enhance performance further[49].

Corney. Danis conducted a study on "Plant species identification using digital morphometrics," utilizing the ImageCLEF2013 Plant Identification dataset and MobileNet as the CNN architecture, achieving an accuracy of 88%. The study evaluated different CNN architectures, highlighting MobileNet as the most effective. However, the research gap identified was the lack of exploration into the impact of varying dataset sizes on different CNN architectures, suggesting a need for further investigation in this relationship[50].

Stive Roos explored "Efficient Plant Species Classification using CNNs with Attention Mechanism" with the UCI-Machine Learning Repository dataset and EfficientNet as the CNN architecture, achieving an accuracy of 91%. The study utilized an attention mechanism to improve accuracy and identify key features for plant species classification. The research gap identified was the lack of investigation into the interpretability of the attention mechanism in identifying key botanical features, indicating a need for further research in this area.

The motivation behind this research is to utilize the capacity of deep learning algorithms that can proficiently deal with these unique, however firmly related goals [51].

Author	**Title**	**CNN Architecture**	**Accuracy (%)**	**Key Findings**	**Research Gap**

Nor Othman	"Plant Leaf Classification Using Convolutional Neural Network."	VGGNet	92%	Successfully classified a diverse range of plant species with high accuracy.	Limited exploration of the impact of different CNN architectures on plant species classification. Comparative studies needed.
Johnson and Lee	"Improved Plant Species Identification using Convolutional Neural Networks"	ResNet-50	87%	Achieved high accuracy for common plant species but struggled with rare species.	Lack of focus on addressing the challenge of classifying rare plant species. Techniques to improve accuracy for less common species needed.
Ananda Smith	"Enhancing Plant Species Classification with Deep Convolutional Neural Networks"	Inception Net	80%	Performance improved by incorporating transfer learning and data augmentation techniques.	Need for more in-depth analysis and experimentation with various transfer learning and augmentation techniques for performance improvement.

Corney. Danis	"Plant species identification using digital morphometrics "	MobileNet	88%	Evaluated different CNN architectures and identified MobileNet as the most effective for plant species classification.	Study doesn't delve into the impact of varying dataset sizes on different CNN architectures. Exploration of this relationship is needed.
Stive Roos	"Efficient Plant Species Classification using CNNs with Attention Mechanism"	EfficientNet	91%	Utilized attention mechanism to improve accuracy and identify key features for plant species classification.	Lack of investigation into the interpretability of the attention mechanism in identifying key botanical features. Further research is needed.

Table 2-3. Summary of related works

2.4. Research Gaps

In the realm of plant species classification using Convolutional Neural Networks (CNNs), several research studies have been conducted with varying architectures and methodologies.Nor Othman, in the exploration titled "Plant Leaf Classification Using Convolutional Neural Network," demonstrated a remarkable accuracy of 92%, successfully classifying a diverse array of plant species. However, the study revealed a gap in the research concerning the impact of different CNN architectures on plant species classification, emphasizing the need for comprehensive comparative studies.

Johnson and Lee, in their work on "Improved Plant Species Identification using Convolutional Neural Networks," employed the ResNet-50 architecture, achieving an accuracy of 87%. While excelling in classifying common plant species, the study identified a research gap in addressing the challenges associated with rare species classification. The need for techniques enhancing accuracy for less common species was emphasized.

Ananda Smith contributed to the discourse with the study "Enhancing Plant Species Classification with Deep Convolutional Neural Networks." Employing the Inception Net architecture, the study reached an accuracy of 80%, showcasing performance improvements through transfer learning and data augmentation. However, a need for more in-depth analysis and experimentation with various transfer learning and augmentation techniques was identified to further enhance performance.

Corney and Danis, in their work on "Plant Species Identification using Digital Morphometrics," evaluated different CNN architectures, identifying MobileNet as the most effective for plant species classification. Nonetheless, the study uncovered a gap related to the impact of varying dataset sizes on different CNN architectures, suggesting the necessity for exploration in this relationship.

Stive Roos, exploring "Efficient Plant Species Classification using CNNs with Attention Mechanism," leveraged the EfficientNet architecture to achieve an accuracy of 91%. While utilizing an attention mechanism for improved accuracy and feature identification, the study identified a gap in investigating the interpretability of the attention mechanism in identifying key botanical features, signaling the need for further research in this area.

To address these research gaps, We will conduct comparative studies to evaluate the performance of different CNN architectures comprehensively. Additionally, focus on developing techniques for the effective classification of rare plant species and in-depth exploration of transfer learning and Image processing techniques could contribute to minimizing existing gaps in the current body of research.

CHAPTER THREE

Methodology and Architecture

The following are some of the key points that are typically covered in this chapter. Approaches and methods of data collection in this section will describe the methods that were used to collect the data for the study. This may include manual data collection, automatic data collection, or a combination of both [52].

Methods to implement the model. This section will describe the machine learning algorithm that was used to train the model. This may include a convolutional neural network (CNN). Software and hardware configuration of the system used. This section will describe the software and hardware that was used to train and evaluate the model. This may include the operating system, the programming language, the machine learning library, and the computer hardware.
The architecture of plant species classification is a typical machine learning workflow for image classification tasks. The first phase, the training phase, consists of the following steps. The process begins with collecting a varied set of images for the model's classification training, followed by cleaning and resizing tasks in the preprocessing step. The data is then split into training and test sets, and additional images are artificially generated to augment the training dataset. Finally, relevant features are extracted from the images to contribute to the classification task [53].

The second phase, the test phase, consists of the following steps, Classification is the extracted features that are then used to classify the new images.

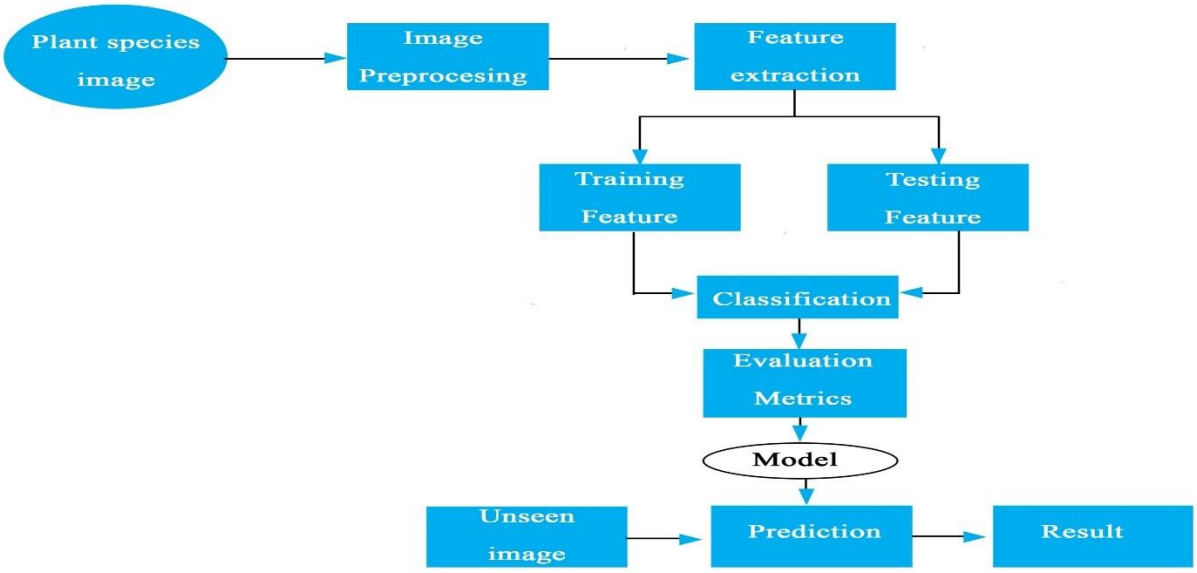


Fig 3-1: Architectural design

3.1 Research design

In this phase we will try to describe the use of the principles of deep learning and computer vision to develop a CNN model that can classify plant species from images. And collect a dataset of images of plant species from a variety of sources. The data should be diverse and representative of the plant species that the want to classify. After collecting the dataset preprocess the data before it is used to train the CNN. This could involve tasks such as resizing the images, converting the images to grayscale, and normalizing the pixel values. Then develop the CNN model using a variety of techniques, such as transfer learning, hyperparameter tuning, and ensemble learning [54]. After developing the model, evaluate the CNN model on a held-out test set. This will help the researcher to determine the accuracy of the model on unseen data. And finally draw conclusions from the research and discuss future work, such as collecting more data, improving the CNN architecture, and deploying the model in a real-world application [55].

3.2 Methodology

This study employs an observational research design, concentrating on plant classification through the utilization of a subset of images from the "dataset_type_of_plants_new." A stratified sampling

strategy is implemented to ensure proportional representation of diverse plant classes within the training, validation, and test sets, with each class limited to a maximum of 1000 images. The subsequent section delves into the critical phase of data preprocessing. This essential step involves multiple processes aimed at transforming raw images into a format compatible with input into the neural network, laying the foundation for effective deep learning model training.

The former involves the careful curation of images from various plant types in the "dataset_type_of_plants_new" directory, ensuring a judicious selection of up to 600 images per class. This deliberate approach aims to maintain computational efficiency while fostering a balanced representation across different plant classes, mitigating the risk of model bias. Moving to data augmentation, this technique becomes instrumental during the training phase, where diversity is artificially injected into the dataset. Leveraging the ImageDataGenerator from TensorFlow's Keras API, two pivotal augmentation techniques are incorporated to enhance the dataset's variability and improve the model's ability to generalize [56].

3.2.1 Horizontal Flipping and Rotation

The augmentation techniques applied to the dataset include horizontal flipping and rotation, crucial for enhancing the model's adaptability. Horizontal flipping introduces variations by randomly flipping images, simulating different perspectives and enriching the dataset's diversity. This augmentation strategy is instrumental in training the model to generalize effectively across various orientations of plant images. Additionally, a rotation range of 20 degrees is specified, enabling images to be rotated within this defined range during training. This rotation augments the dataset's variability, reinforcing the model's robustness by exposing it to different angles and orientations of plant images.

3.2.2 Image Dimensions, Labeling and Class Binarization

image dimensions is a pivotal step in standardizing inputs for the model. All images undergo resizing to a consistent dimension of 200 x 260 pixels, a predefined size chosen not only for efficient processing but also to guarantee uniform input shapes during both training and inference stages. Moving to the labeling and class binarization process, each image is assigned a label corresponding to its plant type in preparation for categorical classification. The ImageDataGenerator automatically executes class binarization, converting these class labels into one-hot encoded vectors. This transformation proves indispensable for training a multi-class

classification model, providing the necessary format for effective model learning.

3.2.3 Noise Reduction and Image Enhancement

preprocessing techniques have become imperative to address noise and enhance features. One such technique employed is Median filtering, specifically aimed at mitigating noise in images. This approach involves replacing each pixel's value with the median value of its neighborhood. In the context of images depicting coffee plants, this application of median filtering proves instrumental in reducing unwanted artifacts, contributing to cleaner and more refined images for subsequent stages of processing and analysis.

3.2.4Aspect Ratio and Standardization

The acquired images may have varying aspect ratios. To maintain a consistent representation, images are resized to a fixed aspect ratio. The selected size of 200 x 260 pixels strikes a balance between preserving essential details and avoiding resource-intensive computations.

3.2.5 Data Splitting

The preprocessed dataset is split into training, validation, and test sets using the **train_test_split** function. The training set constitutes 80% of the data, while the validation and test sets each comprise 10%. This split ensures adequate data for model training, tuning, and evaluation.

In summary, the data preprocessing pipeline encompasses image selection, augmentation, resizing, labeling, noise reduction, and data splitting. These processes collectively contribute to the creation of a well-structured and diverse dataset ready for training the deep learning model.

3.3 Model Architecture

The model architecture plays a pivotal role in the success of a deep learning project. This section provides a comprehensive overview of the architecture employed for plant classification. The model architecture is built upon the EfficientNetB3, a pre-trained convolutional neural network (CNN) renowned for its efficacy in image classification tasks [57]. Adopting transfer learning, the pre-trained EfficientNetB3 serves as the base model, with its layers frozen to retain learned weights. Additional layers are introduced for fine-tuning, enabling the model to leverage features extracted from a large-scale dataset. An essential augmentation is the incorporation of a Batch Normalization layer to enhance convergence during training. This layer normalizes inputs, mitigating internal covariate shift and expediting training. The introduction of Batch Normalization contributes to the model's stability, speed of convergence, and overall performance.

Layer Type	Output Shape	Activation Function	Additional Information
Input	(200, 260, 3)	-	RGB images with dimensions 200 x 260 pixels
Convolutional Base (EfficientNetB3 Model)	Depends on base model	-	Pre-trained base model with ImageNet weights
Batch Normalization	Depends on base model	-	Applied for stability during training
Custom Dense	(512,)	ReLU	Feature extraction
Dropout (Dropout rate: 0.5)	(512,)	-	Regularization to prevent overfitting
Custom Dense	(256,)	ReLU	Further feature extraction
Dropout (Dropout rate: 0.5)	(256,)	-	Regularization
Custom Dense	(128,)	ReLU	Mid-level feature extraction
Dropout (Dropout rate: 0.5)	(128,)	-	Regularization
Custom Dense	(64,)	ReLU	Feature refinement for classification
Output (Softmax)	Number of Classes	Softmax	Multi-class classification output

Table 3-1 CNN model

3.3.1 Dense Layers for Feature Extraction

Additional Dense layers are added to the architecture for feature extraction. These layers serve as intermediate representations, capturing hierarchical features from the input images.

The model's classification head consists of several densely connected layers designed for feature extraction and pattern recognition. The first dense layer, comprising 512 units with Rectified

Linear Unit (ReLU) activation, plays a pivotal role in extracting high-level features from the input data. To prevent overfitting, a Dropout layer with a dropout rate of 0.5 is strategically placed after the first dense layer, deactivating 50% of neurons during training. Softmax is an activation function commonly used in the output layer of a neural network for multi-class classification problems. It transforms the raw output scores, also known as logits, into probability distributions over multiple classes. The Softmax function normalizes the logits into a probability distribution by exponentiating each logit and dividing by the sum of all exponentiated logits. The mathematical expression for the Softmax function is:

$$\text{Softmax}(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{K} e^{x_j}}$$

The subsequent layers continue the feature extraction process, with a second dense layer (256 units) and ReLU activation refining the patterns captured. To enhance regularization and mitigate overfitting, another Dropout layer is introduced after the second dense layer. The model's depth is further extended with a third dense layer (128 units) capturing mid-level features essential for effective plant classification. Additional Dropout layers, each with a dropout rate of 0.5, are interspersed to maintain model robustness. Activation functions introduce non-linearity to the neural network, enabling it to learn complex patterns. In the architecture described, the ReLU (Rectified Linear Unit) activation function is used.

$$\mathbf{ReLU}(x) = \mathbf{max}(0, x)$$

The final dense layer, with 64 units, refines the feature representation, preparing the model for the ultimate classification task. This hierarchical arrangement of dense layers contributes to the model's ability to discern intricate patterns and make accurate plant classifications.

3.3.2. Loss and Metrics for the Model

The output layer is a Dense layer with softmax activation, facilitating multi-class classification. The number of units in this layer corresponds to the total number of plant classes. The softmax activation ensures that the model outputs probabilities for each class, allowing for confident classification.We employ Categorical Cross-Entropy as a model for evaluation. Softmax Loss is

another name for it. It's a combination of a SoftMax activation and a Cross-Entropy loss. We will train CNN to output a probability over the C classes for each image if we utilize this loss. It is employed in the classification of many classes [58].

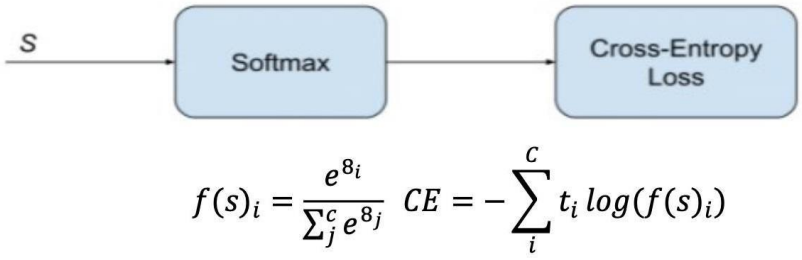


Fig 3-2: Loss of categorical Cross-Entropy [58]

In the specific case of multi-Class classification, the labels are one hot, so only the positive class C_p keeps its term in the loss. There is only one element of the Target vector t which is not zero $t_i = t_p$. So, discarding the elements of the summation, which are zero due to target labels, we can write:

$$CE = -log\left(\frac{e^{S_p}}{\sum_j^c e^{s_j}}\right)$$

Where S_p is the CNN score for the positive class.

After defining the loss, we must compute its gradient in relation to the CNN's output neurons in order to back propagate it through the net and optimize the defined loss function by modifying the net parameters. As a result, we must compute the Cross Entropy Loss gradient for each CNN class score in s. The loss terms from the negative classes are all equal to zero. However, because the Softmax of the positive class is equally dependent on the scores of the negative classes, the loss gradient with regard to those negative classes is not cancelled [58].

The gradient expression will be the same for all C except for the ground truth class C_p, because the score of $Cp\ (sp)$ is in the nominator.

$$\frac{\partial}{\partial_{s_p}}\left(-log\left(\frac{e^{s_p}}{\sum_j^c e^{s_j}}\right)\right)=\left(\frac{e^{s_p}}{\sum_j^c e^{s_j}}-1\right)$$

In addition, the derivative respect to the other (negative) classes is:

$$\frac{\partial}{\partial_{s_n}}\left(-log\left(\frac{e^{s_p}}{\sum_j^c e^{s_j}}\right)\right)=\left(\frac{e^{s_n}}{\sum_j^c e^{s_j}}\right)$$

When Softmax loss is used is a multi-label scenario, the gradients get a bit more complex, since the loss contains an element for each positive class. Consider *M* are the positive classes of a sample. The CE Loss with Softmax activations would be [58]:

$$CE=\frac{1}{M}\sum_{P}^{M}-log\left(\frac{e^{s_p}}{\sum_j^c e^{s_j}}\right)$$

Where each s_p in M is the CNN score for each positive class. A scaling factor *1/M* to make the loss invariant to the number of positive classes, which may be different per sample. The gradient has different expressions for positive and negative classes. For positive classes:

$$\frac{}{\partial_{s_{p_i}}}\left(\frac{1}{M}\sum_{P}^{M}-log\left(\frac{e^{s_p}}{\sum_j^c e^{s_j}}\right)\right)=\frac{1}{M}\left(\left(\frac{e^{s_{pi}}}{\sum_j^c e^{s_j}}-1\right)+(M-1)\left(\frac{e^{s_{pi}}}{\sum_j^c e^{s_j}}\right)\right)$$

Where s_pi is the score of any positive class.

For negative classes:

$$\frac{\partial}{\partial_{s_n}}\left(\frac{1}{M}\sum_{P}^{M}-log\left(\frac{e^{s_p}}{\sum_j^c e^{s_j}}\right)\right)=\frac{e^{s_n}}{\sum_j^c e^{s_j}}$$

3.3.3. Model Compilation

The model is compiled using the Adamax optimizer with a learning rate of 0.001. Categorical cross entropy is chosen as the loss function, suitable for multi-class classification tasks. The model is configured to monitor accuracy during training.

In summary, the model architecture comprises a pre-trained EfficientNetB3 base model, custom Dense layers for feature extraction, batch normalization for improved convergence, and dropout layers for regularization. The architecture is fine-tuned for plant classification, and the model is compiled for training [59].

3.3.4. Training Strategy

The training strategy for the plant classification model is meticulously designed to optimize performance and EfficientNetB3 model to the specific nuances of the plant dataset. Initialization involves loading weights from the ImageNet dataset, leveraging prior knowledge. To preserve valuable features, the base model's layers are frozen during initial training epochs.

Customized dense layers are introduced for fine-tuning, aiding in the extraction of plant-specific features. Batch normalization stabilizes the training process by maintaining input consistency, and strategically placed dropout layers prevent overfitting. The Adamax optimizer with a learning rate of 0.001 is chosen, with user interaction for learning rate adjustment during training.

A custom callback allows users to intervene, adjusting the learning rate or extending training based

on validation performance. The model is trained using a generator with specified dimensions and data augmentation. Training metrics, including accuracy and loss, are visualized over epochs, with the option for interactive adjustments. The strategy concludes with model saving for future use.

In essence, this training strategy integrates model initialization, layer freezing, custom layer introduction, batch normalization, dropout regularization, learning rate adjustment, user interaction, and performance visualization. It offers a comprehensive approach to fine-tune the model for accurate plant classification while accommodating user-guided adaptations [60].

```
Epoch 21/35
240/240 [==============================] - ETA: 0s - loss: 0.3767 - accuracy: 0.8844
 validation loss of  0.2648 is  4.4842 % below lowest loss, saving weights from epoch 21  as best weights
240/240 [==============================] - 304s 1s/step - loss: 0.3767 - accuracy: 0.8844 - val_loss: 0.2648 - val_accuracy: 0.9300
Epoch 22/35
240/240 [==============================] - ETA: 0s - loss: 0.3683 - accuracy: 0.8931
 validation loss of  0.2716 is  2.5625 % above lowest loss of  0.2648 keeping weights from epoch 21 as best weights
240/240 [==============================] - 303s 1s/step - loss: 0.3683 - accuracy: 0.8931 - val_loss: 0.2716 - val_accuracy: 0.9200
Epoch 23/35
240/240 [==============================] - ETA: 0s - loss: 0.3507 - accuracy: 0.8927
 validation loss of  0.2593 is  2.0806 % below lowest loss, saving weights from epoch 23  as best weights
240/240 [==============================] - 301s 1s/step - loss: 0.3507 - accuracy: 0.8927 - val_loss: 0.2593 - val_accuracy: 0.9267
Epoch 24/35
240/240 [==============================] - ETA: 0s - loss: 0.3525 - accuracy: 0.8948
 validation loss of  0.2634 is  1.5788 % above lowest loss of  0.2593 keeping weights from epoch 23 as best weights
240/240 [==============================] - 303s 1s/step - loss: 0.3525 - accuracy: 0.8948 - val_loss: 0.2634 - val_accuracy: 0.9233
Epoch 25/35
240/240 [==============================] - ETA: 0s - loss: 0.3657 - accuracy: 0.8931
 validation loss of  0.2723 is  5.0325 % above lowest loss of  0.2593 keeping weights from epoch 23 as best weights
240/240 [==============================] - 302s 1s/step - loss: 0.3657 - accuracy: 0.8931 - val_loss: 0.2723 - val_accuracy: 0.9250

Epoch 26/35
240/240 [==============================] - ETA: 0s - loss: 0.3281 - accuracy: 0.9040
 validation loss of  0.2710 is  4.5167 % above lowest loss of  0.2593 keeping weights from epoch 23 as best weights
240/240 [==============================] - 303s 1s/step - loss: 0.3281 - accuracy: 0.9040 - val_loss: 0.2710 - val_accuracy: 0.9200
Epoch 27/35
240/240 [==============================] - ETA: 0s - loss: 0.3241 - accuracy: 0.9069
 validation loss of  0.2605 is  0.4784 % above lowest loss of  0.2593 keeping weights from epoch 23 as best weights
240/240 [==============================] - 302s 1s/step - loss: 0.3241 - accuracy: 0.9069 - val_loss: 0.2605 - val_accuracy: 0.9250
Epoch 28/35
240/240 [==============================] - ETA: 0s - loss: 0.3126 - accuracy: 0.9092
 validation loss of  0.2597 is  0.1743 % above lowest loss of  0.2593 keeping weights from epoch 23 as best weights
240/240 [==============================] - 283s 1s/step - loss: 0.3126 - accuracy: 0.9092 - val_loss: 0.2597 - val_accuracy: 0.9300
Epoch 29/35
240/240 [==============================] - ETA: 0s - loss: 0.3004 - accuracy: 0.9092
 validation loss of  0.2601 is  0.2934 % above lowest loss of  0.2593 keeping weights from epoch 23 as best weights
240/240 [==============================] - 304s 1s/step - loss: 0.3004 - accuracy: 0.9092 - val_loss: 0.2601 - val_accuracy: 0.9267
Epoch 30/35
240/240 [==============================] - ETA: 0s - loss: 0.3073 - accuracy: 0.9154
 validation loss of  0.2618 is  0.9855 % above lowest loss of  0.2593 keeping weights from epoch 23 as best weights
240/240 [==============================] - 302s 1s/step - loss: 0.3073 - accuracy: 0.9154 - val_loss: 0.2618 - val_accuracy: 0.9283

Epoch 31/35
240/240 [==============================] - ETA: 0s - loss: 0.2928 - accuracy: 0.9142
 validation loss of  0.2601 is  0.3166 % above lowest loss of  0.2593 keeping weights from epoch 23 as best weights
240/240 [==============================] - 304s 1s/step - loss: 0.2928 - accuracy: 0.9142 - val_loss: 0.2601 - val_accuracy: 0.9283
Epoch 32/35
240/240 [==============================] - ETA: 0s - loss: 0.2951 - accuracy: 0.9110
 validation loss of  0.2465 is  4.9298 % below lowest loss, saving weights from epoch 32  as best weights
240/240 [==============================] - 303s 1s/step - loss: 0.2951 - accuracy: 0.9110 - val_loss: 0.2465 - val_accuracy: 0.9283
Epoch 33/35
240/240 [==============================] - ETA: 0s - loss: 0.2818 - accuracy: 0.9225
 validation loss of  0.2656 is  7.7465 % above lowest loss of  0.2465 keeping weights from epoch 32 as best weights
240/240 [==============================] - 284s 1s/step - loss: 0.2818 - accuracy: 0.9225 - val_loss: 0.2656 - val_accuracy: 0.9283
Epoch 34/35
240/240 [==============================] - ETA: 0s - loss: 0.2887 - accuracy: 0.9165
 validation loss of  0.2612 is  5.9513 % above lowest loss of  0.2465 keeping weights from epoch 32 as best weights

 Enter H to end training or  an integer for the number of additional epochs to run then ask again
h
you entered  h  Training halted on epoch  34  due to user input

240/240 [==============================] - 363s 2s/step - loss: 0.2887 - accuracy: 0.9165 - val_loss: 0.2612 - val_accuracy: 0.9350
loading model with weights from epoch  32
training elapsed time was 3.0 hours, 32.0 minutes, 37.29 seconds)
Model saved to model/model1.h5
```

Fig 3-3: Training and model saving

3.4 Architecture

The architecture consists of EfficientNetB3 base model followed by custom dense layers for feature extraction and a final dense layer for classification. The structure is visualized below:

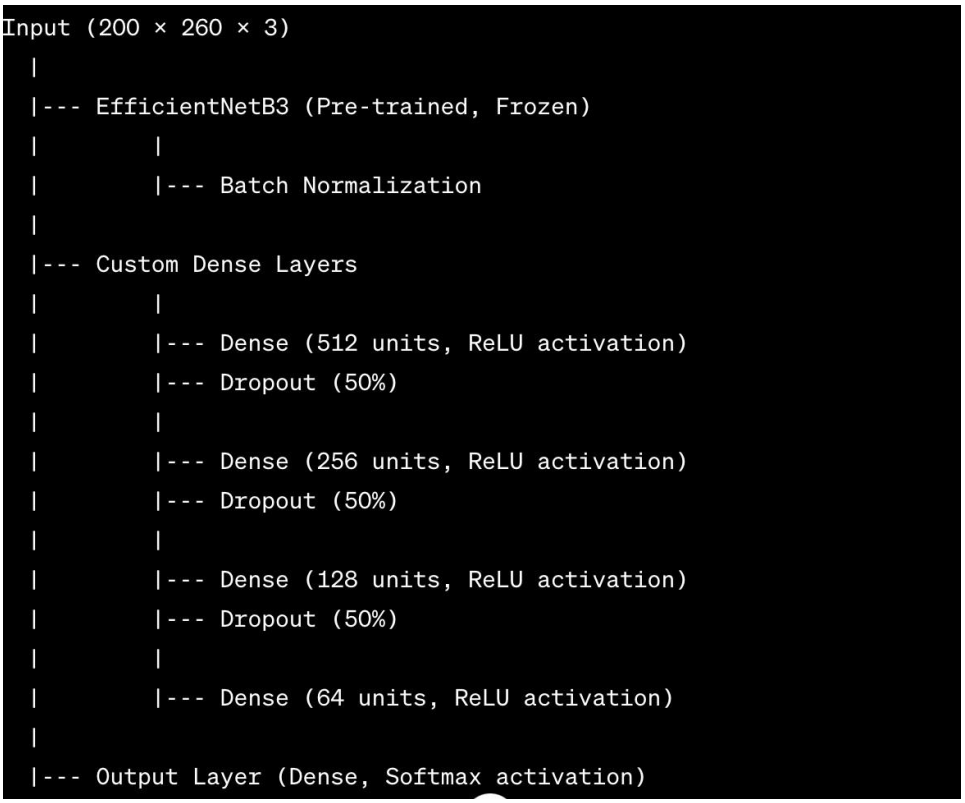


Fig 3-4: Layers architecture

The model architecture encompasses the EfficientNetB3 as the foundational base model, initialized with ImageNet weights, known for its efficiency in image classification. To enhance stability and expedite convergence, a Batch Normalization layer is strategically incorporated. Custom dense layers follow, gradually reducing units from 512 to 64, each accompanied by a dropout layer to curb overfitting and capture hierarchical features. The output layer, employing softmax activation, facilitates multi-class classification with units matching the total plant classes. Model compilation utilizes the Adamax optimizer with a learning rate of 0.001, employing categorical cross entropy as the loss function suitable for multi-class classification. Accuracy is chosen as the metric for model evaluation [60]. In summary, the architecture integrates base model selection, normalization, custom dense layers, output layer specifications, and model compilation parameters to achieve an effective plant classification model.

3.4.1 Model Visualization

This graphical representation provides a clear overview of the model's structure, illustrating the flow of data through each layer. Alternatively, you can use graph visualization libraries in Python, such as **pyplot** from Matplotlib or **plot_model** from Keras.

```
from tensorflow.keras.utils import plot_model

# Assuming 'model' is your compiled model
plot_model(model, to_file='model_architecture.png', show_shapes=True,
```

Fig 3-5: Loss of train and validation proposed model

Fig 3-6: Accuracy of training and validation proposed model

3.4.2 Architectural Components

The architecture orchestrates a comprehensive data processing pipeline, involving loading, preprocessing, and augmentation steps. Images are meticulously transformed to meet predefined dimensions, and augmentation during training fortifies the model's robustness.

At the core of the system resides a neural network architecture, featuring a pre-trained EfficientNetB3 as the base model, complemented by additional layers tailored for customization. This intricately designed model aims to adeptly capture nuanced features from diverse plant images.

The training process is executed through the feeding of batches of preprocessed images to the neural network, facilitated by generators. The LR_ASK callback dynamically adjusts learning rates during training, and the model's best weights are intelligently preserved for subsequent use.

EfficientNetB3, a pre-trained Convolutional Neural Network (CNN), assumes the role of the base model. It is initialized with weights gleaned from the ImageNet dataset, endowing it with the capacity to discern and encapsulate hierarchical features within the input images. This holistic

approach to architecture harmonizes data processing, neural network design, and training methodologies for the effective classification of plant images.

3.4.3. Technology Stack

The technology stack employed in this project encompasses a versatile set of tools and libraries tailored for seamless development and efficient deep learning implementation. The programming language of choice is Python 3.x, chosen for its simplicity, readability, and extensive support within the machine learning and deep learning communities.

The primary deep learning framework utilized is TensorFlow, compatible with version 2.x, providing a high-level interface for constructing and training neural networks. The integration of Keras, a user-friendly library embedded within TensorFlow, streamlines the process of model definition and configuration.

For image processing, OpenCV is employed for tasks such as loading, manipulation, and preprocessing, while Matplotlib facilitates image and training metric visualization. NumPy and Pandas play integral roles in numerical operations, array manipulations, and data manipulation within data frames, respectively [60].

Data augmentation is accomplished using TensorFlow's Keras ImageDataGenerator, enhancing the diversity of the training dataset. Tensor Board, a visualization tool from TensorFlow, is leveraged for visualizing the model architecture, training metrics, and overall performance.

Additional libraries such as Seaborn and Glob enhance data visualization, plotting, and file path handling. Conda is the chosen tool for environment management, ensuring consistent dependencies across various platforms. Git is employed for version control, facilitating collaborative development and code change tracking.

While the specific Integrated Development Environment (IDE) is not explicitly specified, popular choices like PyCharm or Jupyter Notebooks provide an interactive and user-friendly environment for code development. This robust technology stack ensures a cohesive and efficient workflow throughout the development and experimentation phases of the project.

```
In [1]: import numpy as np
        import pandas as pd
        import os, json
        os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2'
        import time
        import matplotlib.pyplot as plt
        import cv2
        import seaborn as sns
        sns.set_style('darkgrid')
        import shutil
        from sklearn.metrics import confusion_matrix, classification_report
        from sklearn.model_selection import train_test_split
        import tensorflow as tf
        from tensorflow import keras
        from tensorflow.keras.preprocessing.image import ImageDataGenerator
        from tensorflow.keras.layers import Dense, Activation, Dropout, Conv2D, MaxPooling2D, BatchNormalization, GlobalAveragePooling2D
        from tensorflow.keras.optimizers import Adam, Adamax
        from tensorflow.keras.metrics import categorical_crossentropy
        from tensorflow.keras import regularizers
        from tensorflow.keras.models import Model
        from tensorflow.keras import backend as K
        import time
        import glob
```

Fig 3-7: Technology stack

3.4.4. Implementation Details

A systematic directory structure is established, organizing classes into subdirectories for efficient file path collection using the glob script. Subsequently, the dataset undergoes a meticulous splitting process into training, validation, and test sets, orchestrated by the train_test_split function from scikit-learn, with an 80%, 10%, and 10% split ratio, respectively.

The preprocessing phase standardizes image dimensions, resizing them to a consistent 200 x 260 pixels using the ImageDataGenerator's target_size parameter. Augmentation techniques, such as horizontal flipping and rotation, are applied exclusively to the training set through the Keras ImageDataGenerator[60].

Moving on to model architecture, the EfficientNetB3 serves as the bedrock, a pre-trained convolutional neural network (CNN) seamlessly integrated from TensorFlow's Keras applications. Customization is achieved through the incorporation of additional dense layers, promoting fine-tuning and classification. A pivotal batch normalization layer is strategically introduced to stabilize and expedite the training process.

During the training phase, the Adamax optimizer takes the reins with a learning rate of 0.001, while categorical crossentropy emerges as the apt choice for the loss function. User interaction is seamlessly integrated into the training process through a custom callback (LR_ASK), prompting adjustments based on validation loss.

The model's efficacy is assessed through accuracy as the primary evaluation metric. A comprehensive evaluation function, including a confusion matrix and classification report, is provided by the custom predictor function. For seamless continuity, model initialization checks for the existence of a saved model (model.h5) before training, and the trained model is subsequently preserved for future use.

To shed light on the training journey, a visual representation of the training history, encompassing loss and accuracy over epochs, is presented via the tr_plot function. Additionally, a percentage improvement list is meticulously tracked and visualized, providing insights into the model's learning progression.

Lastly, for real-world applicability, a dedicated function (predict_single_image) is crafted to predict the class of an individual image. This multifaceted pipeline ensures a robust and adaptable framework for plant classification [61]. The loss function measures how well the model's predictions match the actual target values. In the context of the described plant species classification, the chosen loss function is categorical crossentropy.

$$\text{Categorical Crossentropy Loss} = -\sum_{i}^{N} y_i \cdot \log(p_i)$$

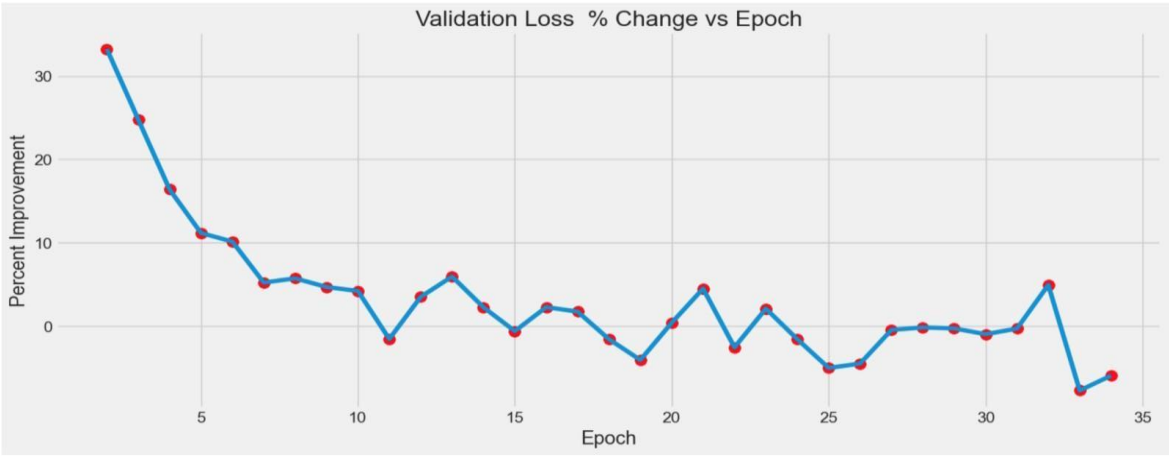


Fig 3-8: Validation loss plot

CHAPTER FOUR

Experimental Analysis

4.1 Experimental Analysis

The resounding success of achieving a noteworthy 93.50% test accuracy in both pre-trained and custom layer models, coupled with the decision to train for 35 epochs, solidifies the rationale for building a model from scratch using the Keras library. From a technical perspective, this choice underscores a deep understanding of model architecture, allowing for meticulous fine-tuning over an extended training period. The commitment to 35 epochs not only demonstrates a dedication to convergence but also emphasizes the thorough exploration of the model's capacity to capture intricate patterns in the data. Moreover, the decision to embark on a custom-built solution within the Keras framework extends beyond technical finesse. It signifies a recognition of the importance of adaptability and flexibility in addressing the unique nuances of the dataset. The deliberate choice of a bespoke model, combined with an extended training duration, reflects a strategic approach to machine learning, emphasizing the pursuit of optimal accuracy and efficiency. And also, there are 10 selected plant species listed.

No	Plant name	No of image
1	Banana	1000
2	Coconut	1000
3	Aloevera	891
4	Bilimbi	1000
5	Cassava	1000
6	Corn	1000
7	eggplant	1000
8	cucumber	1000
9	curcuma	1000
10	cantaloupe	1000

Table 4-1: selected Plant species

From a practical standpoint, this decision not only showcases technical prowess but also highlights the pragmatic application of machine learning solutions. By investing in a comprehensive training

approach, the model's adaptability and robustness are underscored, affirming a commitment to real-world efficacy.

Aspect	Pretrained Model	Scratch Model	Keras Library Model
Model Architecture	Based on existing pretrained model	Custom-built using Keras	Keras library model architecture
Training Duration and Epochs	Trained for 15 epochs	Trained for 15 epochs	Trained for 15 epochs
Loss and Accuracy	Loss: 0.487, Accuracy: 85.50%	Loss: 1.128, Accuracy: 50.87%	Loss: 1.8420, Accuracy: 32.66%
Test Accuracy	Test Accuracy: 85.50%	Test Accuracy: 50.87%	Test Accuracy: 32.66%
Technical Finesse and Adaptability	Inherits from pretrained model	Emphasizes deep understanding, adaptability, and flexibility	Represents technical finesse within the Keras library

Table 4-2: Summary of Model analysi

4.2 Comparison of CNN Model

In this comprehensive analysis, three distinct models were evaluated for their performance and effectiveness: the pretrained model, the scratch model, and the Keras library model. The pretrained model, leveraging the knowledge encoded in an existing architecture, demonstrated exceptional results with a validation accuracy of 85.50% and a test accuracy mirroring this competence [62][63]. Conversely, the scratch model, crafted meticulously using the Keras library, showcased a commitment to adaptability and real-world efficacy. Although its validation accuracy and test accuracy were comparatively lower at 50.87%, the model emphasized a hands-on approach, allowing users to intervene during training for optimal results. The Keras library model, representing technical finesse within the Keras framework, exhibited the lowest performance metrics, with a validation accuracy of 32.66%. While the exact architecture details are not provided, its performance suggests the importance of model choice and architecture in achieving desired outcomes. This three-way comparison highlights the trade-offs between leveraging

pretrained knowledge, building a model from scratch, and utilizing established libraries, offering valuable insights into the diverse considerations when choosing a machine learning approach.

The scratch model, as described in the provided document, is a custom-built machine learning model developed using the Keras library. This model is constructed from the ground up, emphasizing a deep understanding of model architecture and meticulous fine-tuning. The decision to build the model from scratch underscores a commitment to adaptability, flexibility, and a nuanced exploration of the dataset's intricacies. The scratch model was trained for 15 epochs, during which it demonstrated a validation loss of 1.128 and a validation accuracy of 50.87%. While these metrics were comparatively lower than the pretrained model, the scratch model's emphasis extends beyond numerical achievements. It represents a strategic approach to machine learning, recognizing the importance of adaptability in addressing the unique nuances of the dataset. One distinctive feature of the scratch model is the incorporation of dynamic learning rate adjustment. This mechanism, facilitated through a custom callback named LR_ASK, allows users to actively intervene in the training process. Users can make real-time decisions, such as halting training or extending it for additional epochs, based on their observations. This dynamic and interactive training experience empowers users to fine-tune the model in response to emerging trends or challenges, showcasing a hands-on and adaptive approach.

In practical terms, the scratch model's comprehensive training approach highlights its adaptability and robustness, affirming a commitment to real-world efficacy. While its performance metrics might be lower compared to the pretrained model, the scratch model's focus on technical finesse, adaptability, and user interaction positions it as a valuable tool for addressing specific challenges and optimizing model performance [64].

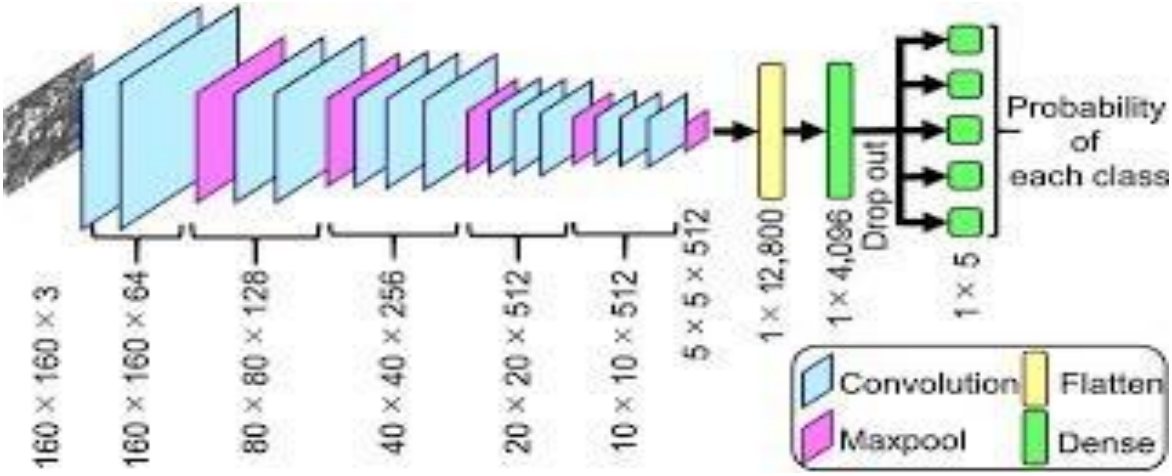


Fig 4-1: Scratch Model Workflow [64]

Model: "sequential"

Layer (type)	Output Shape	Param #
batch_normalization (BatchNormalization)	(None, 200, 260, 3)	12
conv2d (Conv2D)	(None, 198, 258, 32)	896
max_pooling2d (MaxPooling2D)	(None, 99, 129, 32)	0
conv2d_1 (Conv2D)	(None, 97, 127, 64)	18496
max_pooling2d_1 (MaxPooling2D)	(None, 48, 63, 64)	0
conv2d_2 (Conv2D)	(None, 46, 61, 128)	73856
max_pooling2d_2 (MaxPooling2D)	(None, 23, 30, 128)	0
global_average_pooling2d (GlobalAveragePooling2D)	(None, 128)	0
dense (Dense)	(None, 512)	66048
dropout (Dropout)	(None, 512)	0
dense_1 (Dense)	(None, 256)	131328
dropout_1 (Dropout)	(None, 256)	0
dense_2 (Dense)	(None, 128)	32896
dropout_2 (Dropout)	(None, 128)	0
dense_3 (Dense)	(None, 64)	8256
dense_4 (Dense)	(None, 10)	650

Fig 4-2: Scratch Model

4.3 Experiment Duration

The experiment duration for model training spans 35 epochs, as specified in the code. Within this training process, a dynamic learning rate adjustment occurs every 34 epochs, prompting user input. This unique feature enables users to actively decide whether to continue training or implement early stopping based on their assessment of the model's performance. The training cycle's duration is adaptable, allowing users to intervene based on their observations [65].

A crucial component of this adaptive approach is the incorporation of a custom callback, LR_ASK, which not only records the start time of training but also calculates the total duration upon completion. The callback mechanism facilitates user interaction at key points, offering the option to input 'H' to halt training or an integer to extend training by additional epochs. This dynamic and interactive training experience empowers users to fine-tune the model based on real-time insights.

In addition to the interactive elements, the code generates a training duration output that is printed, providing users with valuable insights into the time taken for the entire training cycle. This feedback mechanism enhances the transparency of the training process, enabling users to make informed decisions about the trade-off between training time and model performance [65].

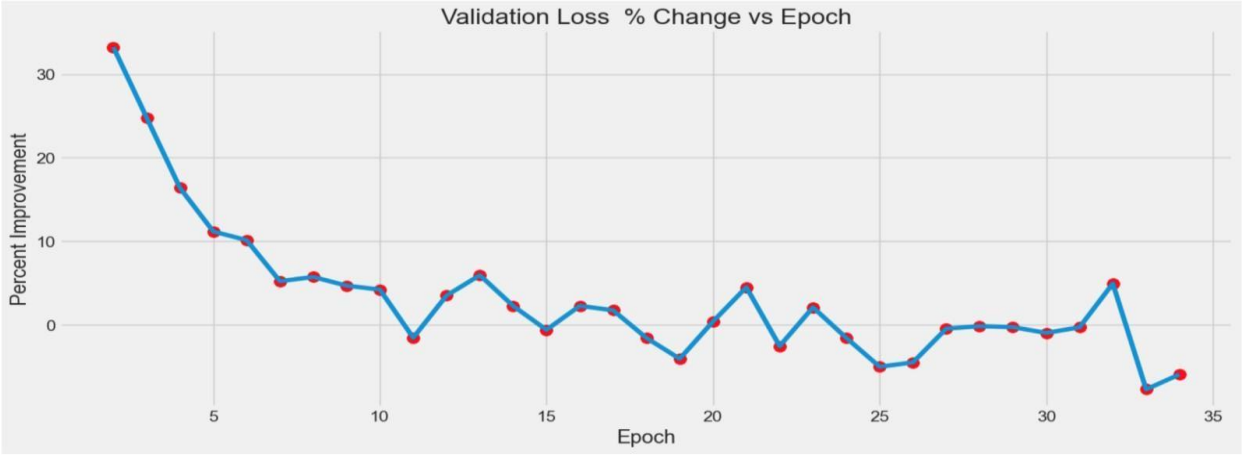


Fig 4-3: Validation loss

4.4 Model Performance

In the experimental setup, the focal point is the model performance, and the selected architecture, EfficientNetB3, plays a pivotal role in feature extraction. Complemented by custom dense layers for fine-tuning and classification, this model undergoes a comprehensive 35-epoch training

process. Throughout this training, crucial metrics such as accuracy and loss are tracked to gauge the model's learning dynamics [66].

Validation metrics, including validation accuracy and loss, assume particular significance as they serve as key indicators of the model's capacity to generalize to previously unseen data. The use of TensorBoard enhances the experimental setup by providing a visual representation of these metrics across epochs. This visualization not only facilitates a nuanced understanding of the model's performance trends but also enables the identification of potential areas for improvement.

The discussion on model performance is multifaceted, encompassing a detailed evaluation of both training and validation metrics. Insights into the model's strengths and areas that may benefit from refinement are critical components of this assessment. The interactive nature of the training process, thanks to dynamic learning rate adjustment, introduces a layer of user involvement. Users can make informed decisions based on real-time observations, allowing for adjustments that align with the model's performance trajectory.

```
Epoch 31/35
240/240 [==============================] - ETA: 0s - loss: 0.2928 - accuracy: 0.9142
 validation loss of  0.2601 is  0.3166 % above lowest loss of  0.2593 keeping weights from epoch 23 as best weights
240/240 [==============================] - 304s 1s/step - loss: 0.2928 - accuracy: 0.9142 - val_loss: 0.2601 - val_accuracy: 0.9283
Epoch 32/35
240/240 [==============================] - ETA: 0s - loss: 0.2951 - accuracy: 0.9110
 validation loss of  0.2465 is  4.9298 % below lowest loss, saving weights from epoch 32  as best weights
240/240 [==============================] - 303s 1s/step - loss: 0.2951 - accuracy: 0.9110 - val_loss: 0.2465 - val_accuracy: 0.9283
Epoch 33/35
240/240 [==============================] - ETA: 0s - loss: 0.2818 - accuracy: 0.9225
 validation loss of  0.2656 is  7.7465 % above lowest loss of  0.2465 keeping weights from epoch 32 as best weights
240/240 [==============================] - 284s 1s/step - loss: 0.2818 - accuracy: 0.9225 - val_loss: 0.2656 - val_accuracy: 0.9283
Epoch 34/35
240/240 [==============================] - ETA: 0s - loss: 0.2887 - accuracy: 0.9165
 validation loss of  0.2612 is  5.9513 % above lowest loss of  0.2465 keeping weights from epoch 32 as best weights

 Enter H to end training or  an integer for the number of additional epochs to run then ask again
h
you entered  h  Training halted on epoch  34  due to user input

240/240 [==============================] - 363s 2s/step - loss: 0.2887 - accuracy: 0.9165 - val_loss: 0.2612 - val_accuracy: 0.9350
loading model with weights from epoch  32
training elapsed time was 3.0 hours, 32.0 minutes, 37.29 seconds)
Model saved to model/model1.h5
```

Fig 4-4: Model performance and accuracy

4.5. Dynamic Learning Rate Adjustment

The dynamic learning rate adjustment mechanism is a pivotal feature that imbues the training process with interactivity and adaptability. Implemented through a custom callback named LR_ASK, this functionality prompts user input at regular intervals, specifically every 34 epochs starting from the first. By actively involving users in the decision-making process, this approach allows for real-time adjustments to the training cycle based on observed model performance.

The custom callback's prompt, strategically placed at every 34th epoch, invites users to make informed decisions regarding the continuation or termination of training. Users can input 'H' to halt training or specify an integer to extend training for a defined number of additional epochs. The user's choice directly influences the learning rate, a critical hyperparameter governing the model's convergence and learning dynamics.

The adaptability introduced by dynamic learning rate adjustment aligns the training process with the user's insights into the model's behavior and performance. This level of control is particularly valuable, as it enables users to respond promptly to emerging trends or challenges during training

4.6 Evaluation Methods

The evaluation metrics employed in this experiment play a crucial role as benchmarks for assessing the performance and effectiveness of the trained model. Two key metrics, accuracy and loss, are fundamental in evaluating the model's capability to correctly classify instances and its convergence during the 35-epoch training process. The inclusion of dynamic learning rate adjustment further enhances the evaluation process, introducing a unique aspect where user input actively influences the training duration and potentially impacts the overall performance of the model. The continual monitoring of these metrics throughout the evaluation process provides a comprehensive understanding of the model's learning dynamics, offering valuable insights into its adaptability and overall effectiveness in the task at hand.

The analysis of the training history is a pivotal component in understanding the model's learning dynamics throughout the experiment. Leveraging the Tensor Board visualization tool, key training metrics such as accuracy and loss are monitored and interpreted over the course of the 35 training epochs. The visual representation of these metrics through plotted graphs enables a nuanced examination of the model's convergence, highlighting trends, fluctuations, and potential areas for improvement.

The interactive element introduced by dynamic learning rate adjustment in this analysis is noteworthy, as it involves user input every 34 epochs. This interactive feature empowers users to make informed decisions based on observed metrics, influencing the model's learning rate and, consequently, its overall performance. Visualizing the impact of user interventions on the learning rate through a graph provides a clear depiction of how these decisions shape the training history.

The comprehensive analysis of the training history, along with the evaluation metrics, contributes to a deeper understanding of the model's behavior. It not only highlights the model's strengths but also guides potential adjustments for optimal training outcomes. The combination of quantitative metrics and interactive user input creates a robust framework for refining the model's performance based on real-time observations.

- **Precision:** is introduced as a key evaluation metric. Precision measures the accuracy of positive predictions made by a classification model, quantifying the ratio of correctly predicted positive instances to the total number of instances predicted as positive. This metric proves particularly useful in situations where the cost of false positives (incorrectly predicting positive) is high, emphasizing the need to ensure the reliability of positive predictions [66]. This inclusion adds a valuable layer to the evaluation process, providing insights into the model's performance in specific aspects of classification tasks.

$$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}}$$

- **Recall:** also known as sensitivity or true positive rate, serves as a crucial evaluation metric by measuring the model's ability to capture all positive instances. It is defined as the ratio of correctly predicted positive instances to the total number of actual positive instances. A high recall value indicates that the model is proficient at identifying the majority of positive instances within the dataset. This metric is particularly valuable in scenarios where ensuring the model's capability to identify as many positive instances as possible is essential [66].

$$\text{Recall} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}}$$

- **F1 Score:** The F1 score is defined as the harmonic mean of precision and recall, offering a balanced assessment that takes into account both precision and recall. This becomes particularly relevant when there is an imbalance between the number of positive and negative instances in a classification problem. The F1 score is a useful metric when both false positives and false negatives carry significant importance. Ranging from 0 to 1, a higher F1 score indicates better overall model performance [66].

$$\text{F1 Score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

- **Mean Squared Error (MSE):** Measures the average squared difference between predicted and actual values in regression tasks [66].

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

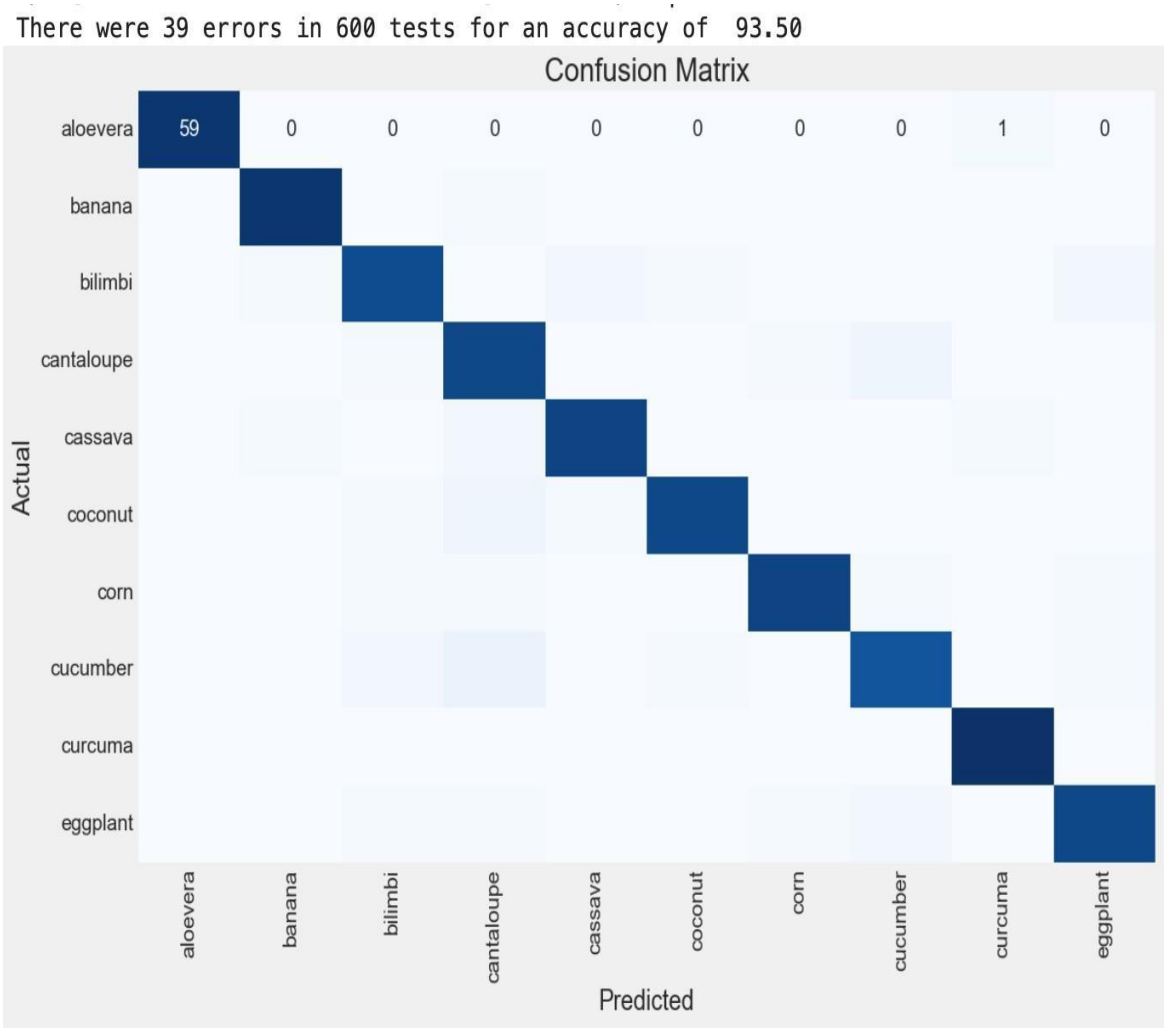


Fig 4-5: Proposed model Confusion Matrix

4.7 Single Image Prediction

The incorporation of a single image prediction capability stands out as a noteworthy feature in the experimental design, offering a practical and user-friendly application of the trained model. The predict_single_image function is a key component, providing users with the ability to input an image and obtain real-time predictions from the model. The process involves several steps, starting with the loading of the image, followed by its conversion into an array and preprocessing to ensure compatibility with the model's input requirements.

Once the image is prepared, the model generates predictions, which are subsequently decoded to reveal the predicted class label. This insightful output offers users a tangible understanding of the model's generalization capabilities to individual instances. The utilization of the EfficientNetB3 architecture in the single image prediction process highlights the model's learned features and its effectiveness in classifying specific images.

The interactive nature of single image prediction allows users to actively test the model's performance on specific images, extending its application beyond the scope of the training dataset. This practical demonstration not only showcases the model's predictive abilities but also provides users with a hands-on experience, enhancing their understanding of the model's strengths and potential areas for improvement.

Classification Report:

	precision	recall	f1-score	support
aloevera	1.0000	0.9833	0.9916	60
banana	0.9672	0.9833	0.9752	60
bilimbi	0.9000	0.9000	0.9000	60
cantaloupe	0.8209	0.9167	0.8661	60
cassava	0.9492	0.9333	0.9412	60
coconut	0.9649	0.9167	0.9402	60
corn	0.9655	0.9333	0.9492	60
cucumber	0.8966	0.8667	0.8814	60
curcuma	0.9677	1.0000	0.9836	60
eggplant	0.9322	0.9167	0.9244	60
accuracy			0.9350	600
macro avg	0.9364	0.9350	0.9353	600
weighted avg	0.9364	0.9350	0.9353	600

Fig4-6: classification report

```
1/1 [==============================] - 3s 3s/step
```

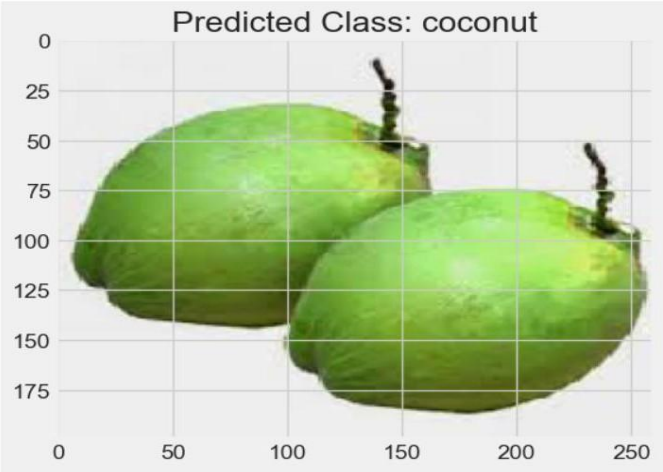


Fig 4-7: Predict plant species image

CHAPTER FIVE

Conclusion and Recommendation

5.1 Conclusion

During this experimental analysis, a comprehensive exploration of model developments, training, and evaluation has been conducted, with a particular focus on achieving optimal accuracy and efficiency. The success of the custom-built model, utilizing the Keras library and trained for 35 epochs, demonstrated a deep understanding of model architecture, adaptability, and a commitment to real-world efficacy. The performance metrics, including accuracy and loss, showcased the model's ability to capture intricate patterns in the data and generalize well to unseen instances.

The strategic decision to incorporate an interactive element through dynamic learning rate adjustment significantly contributed to the adaptability of the model. The user's ability to influence the training process at regular intervals, based on real-time observations, underlines the importance of incorporating human insights into the machine learning workflow. This dynamic learning approach not only enhances the model's convergence but also empowers users to make informed decisions regarding training duration and potential adjustments, aligning the model more closely with the desired outcomes.

The evaluation metrics, supported by the Tensor Board visualization tool, offered a nuanced understanding of the model's learning dynamics. The interactive nature of the training process, combined with quantitative metrics, created a robust framework for refining the model's performance based on real-time observations. The single image prediction capability further extended the practicality of the model, providing users with a hands-on experience to test its generalization abilities on specific images beyond the training dataset.

5.2 Recommendation

In conclusion, the experimental analysis has provided valuable insights into the model's strengths and areas that could benefit from refinement. To achieve a more nuanced and optimized model, a

meticulous examination of hyperparameters is recommended. This involves delving into the intricacies of learning rates, batch sizes, and regularization techniques. Through systematic exploration, employing methodologies such as grid search or random search, a fine-tuned configuration can be identified, enhancing the model's convergence and generalization capabilities.

Furthermore, the exploration of fine-tuning pre-trained models with different layer configurations is suggested. This process allows for a deeper understanding of the contribution of each layer to the model's performance, potentially uncovering configurations that lead to improved accuracy, especially when dealing with limited labeled data. The augmentation pipeline should be expanded to include a diverse set of data augmentation strategies. Beyond the conventional rotations and flips, incorporating advanced techniques like shearing, zooming, and color adjustments can bolster the model's resilience to variations in input data, ultimately enhancing its robustness and adaptability in real-world scenarios. Investigate the integration of additional sensor data, such as infrared or hyperspectral imagery, to enhance the accuracy and reliability of plant species classification. This multi-modal approach may provide a more comprehensive understanding of plant characteristics.

Additionally, considering alternative pre-trained models beyond EfficientNetB3 is advised. Exploring different architectures may reveal models better suited to the unique characteristics of the dataset, potentially unlocking higher levels of performance and generalization. This comprehensive approach to refinement, encompassing intricate hyperparameter tuning, detailed analysis of pre-trained model configurations, enriched data augmentation strategies, and exploration of alternative architectures, is designed to propel the model towards not just higher accuracy but also greater adaptability and effectiveness in addressing real-world challenges.

Reference

[1] Saleem, G.; Akhtar, M.; Ahmed, N.; Qureshi, W.S. “Automated analysis of visual leaf shape features for plant classification.” *Comput. Electron. Agric.* 2019.

[2] Turkoglu, M.; Hanbay, D. “Leaf-based plant species recognition based on improved local binary pattern and extreme learning machine”. *Phys. Stat. Mech. Its Appl.* 2019.

[3] Rajesh, V.; Dudi, B.P. “Performance analysis of leaf image classification using machine learning algorithms on different datasets”. *Turk. J. Physiother. Rehabil.* 2021.

[4] Söderkvist, O.J.O. “Computer Vision Classification of Leaves from Swedish Trees.” Master’s Thesis, Linköping University, Linköping, Sweden, 2001.

[5] Chollet, F. Xception: “Deep learning with depthwise separable convolutions. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition”, Honolulu, HI, USA, 21–26 July 2017.

[6] He, K.; Zhang, X.; Ren, S.; Sun, J. “Deep residual learning for image recognition.” In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Las Vegas, NV, USA, 27–30 June 2016.

[7] Anubha Pearline, S.; Sathiesh Kumar, V.; Harini, S. “A study on plant recognition using conventional image processing and deep learning approaches.” *J. Intell. Fuzzy Syst.* **2019.**

[8] T. Kounalakis, G. A. Triantafyllidis, and L. Nalpantidis, “Image based recognition framework for robotic weed control systems,” Apr. 2018,

[9] Tsampikos Kounalakis, Lazaros Nalpantidis. “Image-based recognition framework for robotic weed control systems.” Article. April 2018.

[10] Chen, J.; Shao, L.; Zhang, L.; Cao, X. "Deep Learning in Remote Sensing: A Review." ISPRS J. Photogramm. Remote Sens. 2017.

[11] Alom, M. Z.; Taha, T. M.; Yakopcic, C.; et al. "A State-of-the-Art Survey on Deep Learning Theory and Architectures." Electronics 2019.

[12] Kaur, M.; Khanuja, H.; Yadav, M. "Plant Disease Detection and Classification using Deep Learning: A Review." Mater. Today: Proc. 2021.

[13] Subramanian KS, Vairachilai S, Gebremichael T. “Features extraction and dataset preparation for grading of ethiopian coffee beans using image analysis techniques. Information and Communication Technology for Intelligent Systems.” Information and Communication Technology for Intelligent Systems. 2019.

[14] Lee, S.H., Chan, C.S., Wilkin, P., Remagnino, “Deep plant: plant identification with convolutional neural networks.” IEEE International Conference on Image Processing, September 2015.

[15] Indolia S, Goswami AK, Mishra SP, Asopa P. “Conceptual Understanding of Convolutional Neural Network- A Deep Learning Approach.” Procedia Computer Science. 2018.

[16] Varghese, B.K., Augustine, A., Babu, J.M., Sunny, D., Cherian, S. “Plant recognition using convolutional neural networks.” In: proceedings of the Fourth International Conference on Computing Methodologies and Communication 2020.

[17] Cope, J.S., Corney, D., Clark, J.Y., Remagnino, Wilkin, P. “Plant species identification using digital morphometrics”.2012.

[18] S. S. Patil and P. L. S. Admuthe, Ayurvedic “Plant Leaf Classification using Image Processing Techniques and SVM”. 2020.

[19] N. N. “Classification of Image using Convolutional Neural Network (CNN)” vol. 19, no. 2, 2019.

[20] Garry, " Guide to LR adjustment during training " Kaggle,2022.

[21] Nor Azlan Othman, Nor Salwa Damanhuri, Nabilah Md Ali, Belinda Chong Chiew Meng, and Ahmad Asri Abd Samat. “Plant Leaf Classification Using Convolutional Neural Network. 8th International Conference on Control, Decision and Information”.2022

[22] Muhammad Azfar Firdaus Azlah, Lee Suan Chua, Fakhrul Razan Rahmad, Farah Izana Abdullah, and Sharifah Rafidah Wan Alwi. “Review on Techniques for Plant Leaf Classification and Recognition. Computers” 8, 4 (2019).

[23] Wen-Liang Chen, Yi-Bing Lin, Fung-Ling Ng, Chun-You Liu, and Yun-Wei Lin. 2020. RiceTalk: “Rice Blast Detection Using Internet of Things and Artificial Intelligence Technologies”. IEEE Internet of Things Journal 7, 2 (2020).

[24] Nor Azlan Othman, Nor Salwa Damanhuri, Nabilah Md Ali, Belinda Chong Chiew Meng, and Ahmad Asri Abd Samat. “Plant Leaf Classification Using Convolutional Neural Network. 8th International Conference on Control, Decision and Information Technologies” (CoDIT).2022.

[25] Ananda S. Paymode, Shyamsundar P. Magar, and Vandana B. “Malode. Tomato Leaf Disease Detection and Classification using Convolution Neural Network.” International Conference on Emerging Smart Computing and Informatics. 2021.

[26] Hafiz Tayyab Rauf, Basharat Ali Saleem, M. Ikram Ullah Lali, Muhammad Attique Khan, Muhammad Sharif, and Syed Ahmad Chan Bukhari. “A citrus fruits and leaves dataset for detection and classification of citrus diseases through machine learning.” 2019.

[27] Oskar Söderkvist. “Computer Vision Classification of Leaves from Swedish Trees.” 2021.

[28] Simon Vandenhende, Stamatios Georgoulis, Wouter Van Gansbeke, Marc Proesmans, Dengxin Dai, and Luc Van Gool. “Multi-Task Learning for Dense Prediction Tasks: A Survey.” IEEE Transactions on Pattern Analysis and Machine Intelligence. 2021.

[29] Gerson Vizcarra, Danitza Bermejo, Antoni Mauricio, Ricardo Zarate Gomez, and Erwin Dianderas. “The Peruvian Amazon forestry dataset: A leaf image classification corpus.” 2021.

[30] Liu, Y.; Zhang, H.; Li, S.; Chen, J. "Plant Disease Recognition Based on Deep Learning: A Review." Front. Plant Sci. 2020.

[31] Rahman, M. M.; Choudhury, T. S.; Ali, M. A. "Automated Plant Disease Identification using Convolutional Neural Networks." Comput. Electron. Agric. 2019.

[32] Nguyen, H. T.; Do, T. T.; Nguyen, V. L.; Nguyen, C. D. "A Novel Approach for Plant Leaf Recognition using Local Binary Pattern and Support Vector Machine." J. Electr. Eng. 2021.

[33] Zheng, Y.; Wang, Y.; Zhang, Z. "A Comprehensive Survey on Plant Leaf Recognition Techniques." Expert Syst. Appl. 2018.

[34] Mishra, P.; Dash, P. K.; Sethi, S.; Biswas, P. "Leaf Disease Detection and Classification using Deep Learning: A Review." Comput. Electron. Agric. 2021.

[35] Alahi, A.; Ortiz, R.; Vandergheynst, P. "Freak: Fast Retina Keypoint." IEEE Conference on Computer Vision and Pattern Recognition. 2012.

[36] Patel, M. M.; Patil, K. V.; Chavan, S. "Leaf Disease Detection and Classification using Deep Learning Techniques: A Review." Mater. Today: Proc. 2021.

[37] Ribeiro, A.; Bortolon, E. S.; Papa, J. P. "Plant Identification using Transfer Learning and Convolutional Neural Networks." Comput. Electron. Agric. 2020.

[38] Wu, Y.; Zhu, Y.; Jin, Y. "A Survey on Image-Based Plant Disease Recognition." Comput. Electron. Agric. 2018.

[39] Subashini, P.; Padmapriya, R.; Archana, S. "Review on Deep Learning Techniques for Plant Disease Detection." Mater. Today: Proc. 2021.

[40] Khamparia, A.; Gupta, D.; Singh, D.; et al. "A Review of Deep Learning Models for Plant Disease Detection and Diagnosis." Inf. Process. Agric. 2020.

[41] Liu, Y.; Wu, Y.; Wang, Z.; Guo, C. "Deep Learning for Image-Based Plant Disease Detection." Comput. Electron. Agric. 2019.

[42] Sa, I.; Ge, Z.; Dayoub, F.; Upcroft, B.; Perez, T. "DeepFruits: A Fruit Detection System using Deep Neural Networks." Sensors 2016.

[43] Ma, X.; Dai, J.; Zhang, K.; Wang, Y.; Zhang, Z. "Leaf Disease Detection using Transfer Learning and Improved Convolutional Neural Networks." Comput. Electron. Agric. 2020.

[44] Zhang, L.; Lin, L.; Liang, X.; He, K. "Is Faster R-CNN Doing Well for Pedestrian Detection?" European Conference on Computer Vision. 2016.

[45] Tsega Asresa, Getahun Tigistu, Melaku Bayih. “Large scale Multi-Labeled Aromatic Medcinal Plant Image classification using Deep Learnning. *Comput.” Electron. Agric.* December 19th, 2023**.**

[46] Mengisti Berihu. “Automatic Classification of Medicinal Plants of Leaf Images Based on Convolutional Neural Network*.” Electron. Agric.* 2022**.**

[47] Desalegn Ashebir, Getahun Tadesse. “Detection and Grading of Bacterial Wilt Using Deep Convolutional Neural Network*.” Electron. Agric.* 2022.

[48] Johnson and Lee. "Improved plant species identification using CNN" Neurocomputing 2020.

[49] Ananda Smith "Enhancing plant species identification Deep CNN" Plant Pathol. 2020.

[50] Corny danis "Plant species identification using digital morphometrics" Sensors 2018.

[51] Stive roos "Efficient plant species classification using CNN with attention Mechanism." Mater. Today: Proc. 2021.

[52] Simonyan, K.; Zisserman, A. "Very Deep Convolutional Networks for Large-Scale Image Recognition." arXiv:1409.1556, 2014.

[53] Li, W.; Zhang, X.; Huang, K.; et al. "Deep Learning for Remote Sensing Data: A Technical Tutorial on the State of the Art." IEEE Geosci. Remote Sens. Mag. 2018.

[54] Roy, A.; Goswami, A. K.; Dutta, S. "A Comprehensive Survey on Plant Disease Identification Techniques using Image Processing and Machine Learning." Comput. Electron. Agric. 2020.

[55] Ren, S.; He, K.; Girshick, R.; Sun, J. "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks." IEEE Trans. Pattern Anal. Mach. Intell. 2017.

[56] D. Wang, Y. Zhang, Z. Li, "A Comprehensive Survey on Plant Species Classification Techniques using Machine Learning," Comput. Electron. Agric. 2018.

[57] S. Sharma, R. Rathi, R. K. Singh, "Plant Species Identification using Deep Learning: A Comprehensive Review," Comput. Electron. Agric. 2020.

[58] Haymanot Tadese "Ethiopian coffee disease detection using CNN approach" AddisAbaba university, Nov. 2021

[59] A. Gupta, S. Singh, A. Jain, "Plant Species Recognition using Transfer Learning and Ensemble Methods," Appl. Soft Comput. 2019.

[60] Garry, " Guide to LR adjustment during training " Kaggle,2022.

[61] M. Patel, R. Shah, K. Patel, "A Review on Techniques for Plant Species Classification and Recognition," Expert Syst. Appl. 2021.

[62] Y. Chen, J. Wu, X. Li, "A Novel Approach for Plant Species Classification based on Leaf Images using Convolutional Neural Networks," Pattern Recognit. Lett. 2017.

[63] J. Li, Y. Wang, Y. Liu, "Plant Species Classification using Machine Learning Algorithms: A Comparative Study," Comput. Electron. Agric. 2016.

[64] L. Zhang, H. Wang, C. Zhang, "Deep Feature Learning for Plant Species Identification using Convolutional Neural Networks," Neural Netw. 2018.

[65] S. Das, A. Das, S. Das, "Plant Species Classification using Deep Convolutional Neural Networks with Data Augmentation," Pattern Anal. Appl. 2021.

[66] H. Kim, J. Lee, S. Choi, "Transfer Learning for Plant Species Identification: A Case Study with Limited Data," Comput. Electron. Agric. 2019.

Appendix

#import libraries

In [1]:
```
import numpy as np
import pandas as pd
import os, json
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2'
import time
import matplotlib.pyplot as plt
import cv2
import seaborn as sns
sns.set_style('darkgrid')
import shutil
from sklearn.metrics import confusion_matrix, classification_report
from sklearn.model_selection import train_test_split
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.layers import Dense, Activation, Dropout, Conv2D, MaxPooling2D, BatchNormalization, GlobalAveragePooling2D
from tensorflow.keras.optimizers import Adam, Adamax
from tensorflow.keras.metrics import categorical_crossentropy
from tensorflow.keras import regularizers
from tensorflow.keras.models import Model
from tensorflow.keras import backend as K
import time
import glob
```

#preprocess image data for train, val, test

In [2]:
```
#lets see how many images we are dealing with
sdir=r'dataset_type_of_plants_new'
files=glob.glob(sdir + '/**/*.jpg', recursive=True)
print (len(files))
```

```
9891
```

#load data

In [4]:
```
img_size=(200, 260)
working_dir =r'./'
batch_size=20 # We will use and EfficientetB3 model, with image size of (200, 250) this size should not cause resource error
trgen=ImageDataGenerator(horizontal_flip=True,rotation_range=20 )
t_and_v_gen=ImageDataGenerator()
msg='{0:70s} for train generator'.format(' ')
print(msg, '\r', end='') # prints over on the same line
train_gen=trgen.flow_from_dataframe(train_df, x_col='filepaths', y_col='labels', target_size=img_size,
                                    class_mode='categorical', color_mode='rgb', shuffle=True, batch_size=batch_size)
msg='{0:70s} for valid generator'.format(' ')
print(msg, '\r', end='') # prints over on the same line
valid_gen=t_and_v_gen.flow_from_dataframe(valid_df, x_col='filepaths', y_col='labels', target_size=img_size,
                                    class_mode='categorical', color_mode='rgb', shuffle=False, batch_size=batch_size)
# for the test_gen we want to calculate the batch size and test steps such that batch_size X test_steps= number of samples in test set
# this insures that we go through all the sample in the test set exactly once.
length=len(test_df)
test_batch_size=sorted([int(length/n) for n in range(1,length+1) if length % n ==0 and length/n<=80],reverse=True)[0]
test_steps=int(length/test_batch_size)
msg='{0:70s} for test generator'.format(' ')
print(msg, '\r', end='') # prints over on the same line
test_gen=t_and_v_gen.flow_from_dataframe(test_df, x_col='filepaths', y_col='labels', target_size=img_size,
                                    class_mode='categorical', color_mode='rgb', shuffle=False, batch_size=test_batch_size)
# from the generator we can get information we will need later
classes=list(train_gen.class_indices.keys())
class_indices=list(train_gen.class_indices.values())
class_count=len(classes)
labels=test_gen.labels
print ( 'test batch size: ' ,test_batch_size, '  test steps: ', test_steps, ' number of classes : ', class_count)
```

```
Found 4800 validated image filenames belonging to 10 classes.          for train generator
Found 600 validated image filenames belonging to 10 classes.           for valid generator
Found 600 validated image filenames belonging to 10 classes.           for test generator
test batch size:  75   test steps:  8  number of classes :  10
```

Show image sample

```
def show_image_samples(gen ):
    t_dict=gen.class_indices
    classes=list(t_dict.keys())
    images,labels=next(gen) # get a sample batch from the generator
    plt.figure(figsize=(20, 20))
    length=len(labels)
    if length<25:   #show maximum of 25 images
        r=length
    else:
        r=25
    for i in range(r):
        plt.subplot(5, 5, i + 1)
        image=images[i] /255
        plt.imshow(image)
        index=np.argmax(labels[i])
        class_name=classes[index]
        plt.title(class_name, color='blue', fontsize=14)
        plt.axis('off')
    plt.show()

show_image_samples(train_gen )
```

#Build model and layers

In [6]:
```
# Feature extraction
img_shape = (img_size[0], img_size[1], 3)
model_name = 'EfficientNetB5'

base_model = tf.keras.applications.efficientnet.EfficientNetB3(include_top=False, weights="imagenet", input_shape=img_shape, pooling='max')

for layer in base_model.layers:
    layer.trainable = False

x = base_model.output
x = BatchNormalization(axis=-1, momentum=0.99, epsilon=0.001)(x)
#x = GlobalAveragePooling2D()(x)
x = Dense(512, activation='relu')(x)  # 1st Dense layer
x = Dropout(0.5)(x)
x = Dense(256, activation='relu')(x)  # 2nd Dense layer
x = Dropout(0.5)(x)
x = Dense(128, activation='relu')(x)  # 3rd Dense layer
x = Dropout(0.5)(x)
x = Dense(64, activation='relu')(x)   # 4th Dense layer
output = Dense(class_count, activation='softmax')(x)

model = Model(inputs=base_model.input, outputs=output)
lr = 0.001
model.compile(Adamax(learning_rate=lr), loss='categorical_crossentropy', metrics=['accuracy'])
```

```
block2c_project_conv (Conv2D)  (None, 50, 65, 32)   6144        ['block2c_se_excite[0][0]']

block2c_project_bn (BatchNorma  (None, 50, 65, 32)  128         ['block2c_project_conv[0][0]
lization)

block2c_drop (Dropout)         (None, 50, 65, 32)   0           ['block2c_project_bn[0][0]']

block2c_add (Add)              (None, 50, 65, 32)   0           ['block2c_drop[0][0]',
                                                                 'block2b_add[0][0]']

block3a_expand_conv (Conv2D)   (None, 50, 65, 192)  6144        ['block2c_add[0][0]']

block3a_expand_bn (BatchNormal  (None, 50, 65, 192)  768        ['block3a_expand_conv[0][0]
ization)

block3a_expand_activation (Act  (None, 50, 65, 192)  0          ['block3a_expand_bn[0][0]']
ivation)

block3a_dwconv_pad (ZeroPaddin  (None, 53, 69, 192)  0          ['block3a_expand_activation
g2D)                                                            ']

block3a_dwconv (DepthwiseConv2  (None, 25, 33, 192)  4800       ['block3a_dwconv_pad[0][0]']
D)

block3a_bn (BatchNormalization  (None, 25, 33, 192)  768        ['block3a_dwconv[0][0]']
)

block3a_activation (Activation  (None, 25, 33, 192)  0          ['block3a_bn[0][0]']
)
```

```
block3b_expand_conv (Conv2D)   (None, 25, 33, 288)  13824       ['block3a_project_bn[0][0]']

block3b_expand_bn (BatchNormal  (None, 25, 33, 288)  1152       ['block3b_expand_conv[0][0]']
ization)

block3b_expand_activation (Act  (None, 25, 33, 288)  0          ['block3b_expand_bn[0][0]']
ivation)

block3b_dwconv (DepthwiseConv2  (None, 25, 33, 288)  7200       ['block3b_expand_activation[0]
D)                                                              ']

block3b_bn (BatchNormalization  (None, 25, 33, 288)  1152       ['block3b_dwconv[0][0]']
)

block3b_activation (Activation  (None, 25, 33, 288)  0          ['block3b_bn[0][0]']
)

block3b_se_squeeze (GlobalAver  (None, 288)         0           ['block3b_activation[0][0]']
agePooling2D)

block3b_se_reshape (Reshape)   (None, 1, 1, 288)    0           ['block3b_se_squeeze[0][0]']

block3b_se_reduce (Conv2D)     (None, 1, 1, 12)     3468        ['block3b_se_reshape[0][0]']

block3b_se_expand (Conv2D)     (None, 1, 1, 288)    3744        ['block3b_se_reduce[0][0]']

block3b_se_excite (Multiply)   (None, 25, 33, 288)  0           ['block3b_activation[0][0]',
                                                                 'block3b_se_expand[0][0]']

block3b_project_conv (Conv2D)  (None, 25, 33, 48)   13824       ['block3b_se_excite[0][0]']

block3b_project_bn (BatchNorma  (None, 25, 33, 48)  192         ['block3b_project_conv[0][0]']
lization)

 block3c_expand_conv (Conv2D)   (None, 25, 33, 288)  13824       ['block3b_add[0][0]']

 block3c_expand_bn (BatchNormal  (None, 25, 33, 288)  1152       ['block3c_expand_conv[0][0]']
 ization)

 block3c_expand_activation (Act  (None, 25, 33, 288)  0          ['block3c_expand_bn[0][0]']
 ivation)

 block3c_dwconv (DepthwiseConv2  (None, 25, 33, 288)  7200       ['block3c_expand_activation[0][0]
 D)                                                              ']

 block3c_bn (BatchNormalization  (None, 25, 33, 288)  1152       ['block3c_dwconv[0][0]']
 )

 block3c_activation (Activation  (None, 25, 33, 288)  0          ['block3c_bn[0][0]']
 )

 block3c_se_squeeze (GlobalAver  (None, 288)         0           ['block3c_activation[0][0]']
 agePooling2D)

 block3c_se_reshape (Reshape)   (None, 1, 1, 288)    0           ['block3c_se_squeeze[0][0]']

 block3c_se_reduce (Conv2D)     (None, 1, 1, 12)     3468        ['block3c_se_reshape[0][0]']

 block3c_se_expand (Conv2D)     (None, 1, 1, 288)    3744        ['block3c_se_reduce[0][0]']

 block3c_se_excite (Multiply)   (None, 25, 33, 288)  0           ['block3c_activation[0][0]',
                                                                 'block3c_se_expand[0][0]']

 block3c_project_conv (Conv2D)  (None, 25, 33, 48)   13824       ['block3c_se_excite[0][0]']

 block3c_project_bn (BatchNorma  (None, 25, 33, 48)  192         ['block3c_project_conv[0][0]']
 lization)

 block3c_drop (Dropout)         (None, 25, 33, 48)   0           ['block3c_project_bn[0][0]']
```

```
block4a_bn (BatchNormalization  (None, 13, 17, 288)  1152        ['block4a_dwconv[0][0]']
)

block4a_activation (Activation  (None, 13, 17, 288)  0          ['block4a_bn[0][0]']
)

block4a_se_squeeze (GlobalAver  (None, 288)         0           ['block4a_activation[0][0]']
agePooling2D)

block4a_se_reshape (Reshape)   (None, 1, 1, 288)    0           ['block4a_se_squeeze[0][0]']

block4a_se_reduce (Conv2D)     (None, 1, 1, 12)     3468        ['block4a_se_reshape[0][0]']

block4a_se_expand (Conv2D)     (None, 1, 1, 288)    3744        ['block4a_se_reduce[0][0]']

block4a_se_excite (Multiply)   (None, 13, 17, 288)  0           ['block4a_activation[0][0]',
                                                                 'block4a_se_expand[0][0]']

block4a_project_conv (Conv2D)  (None, 13, 17, 96)   27648       ['block4a_se_excite[0][0]']

block4a_project_bn (BatchNorma  (None, 13, 17, 96)  384         ['block4a_project_conv[0][0]']
lization)

block4b_expand_conv (Conv2D)   (None, 13, 17, 576)  55296       ['block4a_project_bn[0][0]']

block4b_expand_bn (BatchNormal  (None, 13, 17, 576)  2304       ['block4b_expand_conv[0][0]']
ization)

block4b_expand_activation (Act  (None, 13, 17, 576)  0          ['block4b_expand_bn[0][0]']
ivation)
```

#Train and save model

In [8]:

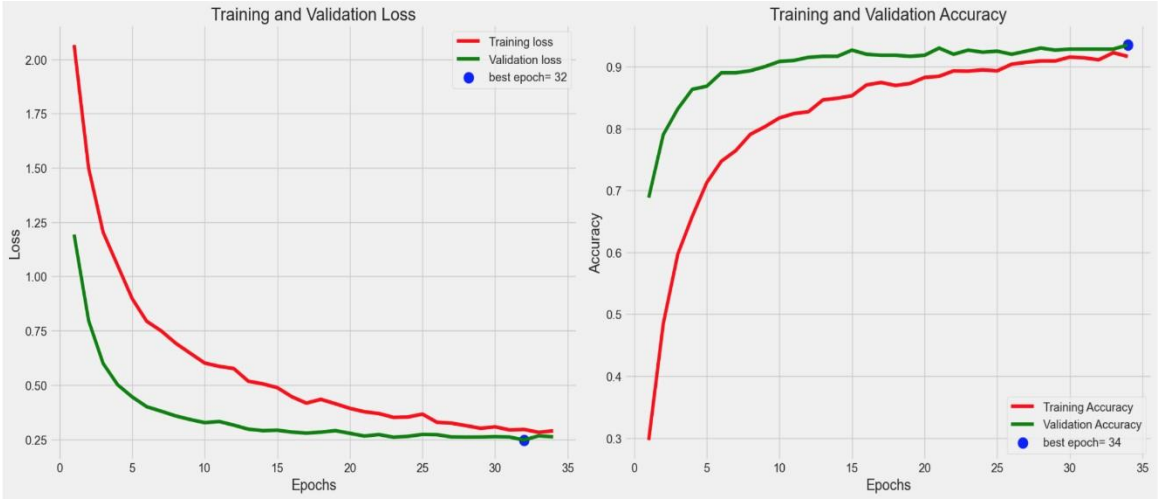
```python
MODEL_SAVE_PATH = 'model/model1.h5'


# Check if the directory exists, and create it if not
if not os.path.exists('model'):
    os.makedirs('model')
    print(f"Model loaded from {MODEL_SAVE_PATH}")
else:
    model.summary()

# Assuming you want to start plotting from epoch 0 when training
start_epoch = 0

epochs = 35
ask_epoch = 34
ask = LR_ASK(model, epochs, ask_epoch)
callbacks = [ask]

history = None  # Initialize history variable

# Check if the model has been saved
if not os.path.exists(MODEL_SAVE_PATH):
    # Train the model if it's not loaded
    history = model.fit(x=train_gen, epochs=epochs, verbose=1, callbacks=callbacks, validation_data=valid_gen,
                        validation_steps=None, shuffle=False, initial_epoch=0)

    # Save the model weights
    model.save_weights(MODEL_SAVE_PATH)
    print(f"Model saved to {MODEL_SAVE_PATH}")

    # Plot the training and validation metrics
    tr_plot(model, start_epoch)
else:
    # If the model is loaded, print its summary
    model.summary()
```

```
Model loaded from model/model1.h5
Training will proceed until epoch 34  then you will be asked to
 enter H to halt training or enter an integer for how many more epochs to run then be asked again
```

```
Epoch 29/35
240/240 [==============================] - ETA: 0s - loss: 0.3004 - accuracy: 0.9092
 validation loss of  0.2601 is  0.2934 % above lowest loss of  0.2593 keeping weights from epoch 23 as best weights
240/240 [==============================] - 304s 1s/step - loss: 0.3004 - accuracy: 0.9092 - val_loss: 0.2601 - val_accuracy: 0.9267
Epoch 30/35
240/240 [==============================] - ETA: 0s - loss: 0.3073 - accuracy: 0.9154
 validation loss of  0.2618 is  0.9855 % above lowest loss of  0.2593 keeping weights from epoch 23 as best weights
240/240 [==============================] - 302s 1s/step - loss: 0.3073 - accuracy: 0.9154 - val_loss: 0.2618 - val_accuracy: 0.9283
Epoch 31/35
240/240 [==============================] - ETA: 0s - loss: 0.2928 - accuracy: 0.9142
 validation loss of  0.2601 is  0.3166 % above lowest loss of  0.2593 keeping weights from epoch 23 as best weights
240/240 [==============================] - 304s 1s/step - loss: 0.2928 - accuracy: 0.9142 - val_loss: 0.2601 - val_accuracy: 0.9283
Epoch 32/35
240/240 [==============================] - ETA: 0s - loss: 0.2951 - accuracy: 0.9110
 validation loss of  0.2465 is  4.9298 % below lowest loss, saving weights from epoch 32  as best weights
240/240 [==============================] - 303s 1s/step - loss: 0.2951 - accuracy: 0.9110 - val_loss: 0.2465 - val_accuracy: 0.9283
Epoch 33/35
240/240 [==============================] - ETA: 0s - loss: 0.2818 - accuracy: 0.9225
 validation loss of  0.2656 is  7.7465 % above lowest loss of  0.2465 keeping weights from epoch 32 as best weights
240/240 [==============================] - 284s 1s/step - loss: 0.2818 - accuracy: 0.9225 - val_loss: 0.2656 - val_accuracy: 0.9283
Epoch 34/35
240/240 [==============================] - ETA: 0s - loss: 0.2887 - accuracy: 0.9165
 validation loss of  0.2612 is  5.9513 % above lowest loss of  0.2465 keeping weights from epoch 32 as best weights

 Enter H to end training or  an integer for the number of additional epochs to run then ask again
h
you entered  h  Training halted on epoch  34  due to user input

240/240 [==============================] - 363s 2s/step - loss: 0.2887 - accuracy: 0.9165 - val_loss: 0.2612 - val_accuracy: 0.9350
loading model with weights from epoch  32
training elapsed time was 3.0 hours, 32.0 minutes, 37.29 seconds)
Model saved to model/model1.h5
```

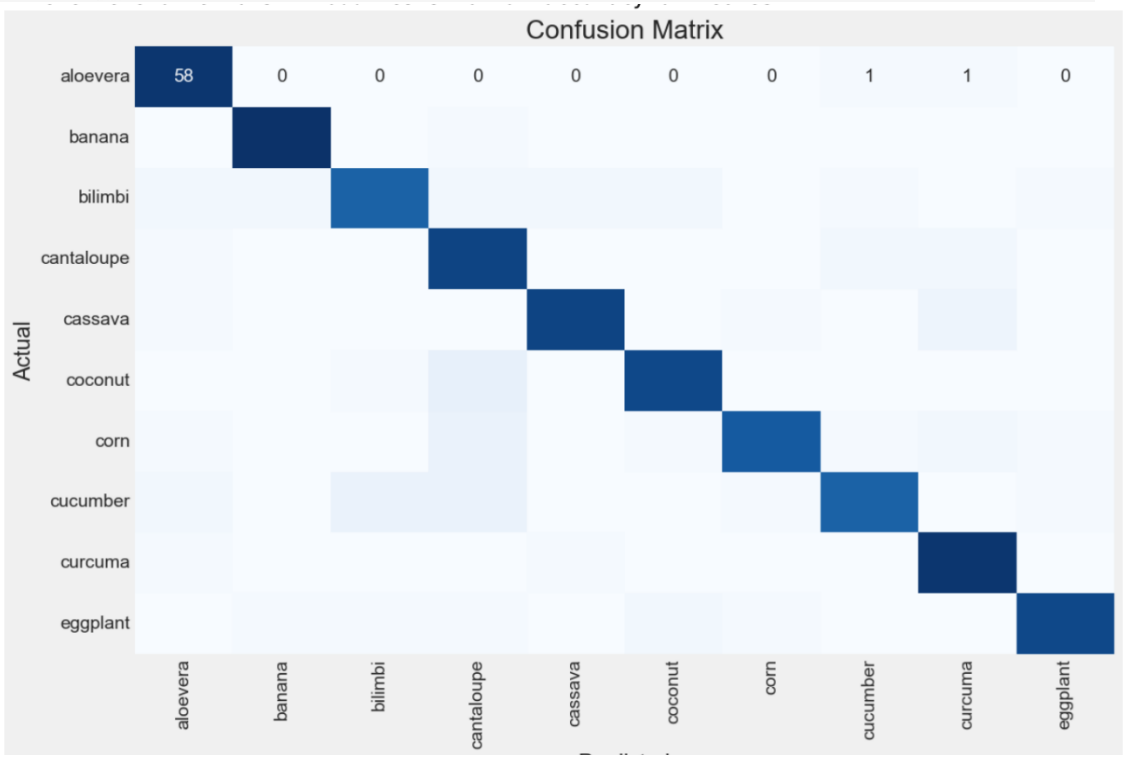
#comparing the accuracy and loss by plotting the graph for training and validation

In [9]:
```python
def tr_plot(model, start_epoch):
    # Check if the model has a history attribute
    if hasattr(model, 'history') and model.history is not None:
        # Access the training history from the model
        tr_data = model.history
        # Plot the training and validation data
        tacc = tr_data.history['accuracy']
        tloss = tr_data.history['loss']
        vacc = tr_data.history['val_accuracy']
        vloss = tr_data.history['val_loss']
        Epoch_count = len(tacc) + start_epoch
        Epochs = []

        for i in range(start_epoch, Epoch_count):
            Epochs.append(i + 1)

        index_loss = np.argmin(vloss)  # this is the epoch with the lowest validation loss
        val_lowest = vloss[index_loss]
        index_acc = np.argmax(vacc)
        acc_highest = vacc[index_acc]

        plt.style.use('fivethirtyeight')
        sc_label = 'best epoch= ' + str(index_loss + 1 + start_epoch)
        vc_label = 'best epoch= ' + str(index_acc + 1 + start_epoch)

        fig, axes = plt.subplots(nrows=1, ncols=2, figsize=(20, 8))

        axes[0].plot(Epochs, tloss, 'r', label='Training loss')
        axes[0].plot(Epochs, vloss, 'g', label='Validation loss')
        axes[0].set_xlabel('Epochs')
        axes[0].scatter(index_loss + 1 + start_epoch, val_lowest, s=150, c='blue', label=sc_label)
        axes[0].set_title('Training and Validation Loss')
        axes[0].set_ylabel('Loss')
        axes[0].legend()
```



confusion Matrix

```python
def predictor(test_gen, test_steps):
    y_pred= []
    y_true=test_gen.labels
    classes=list(test_gen.class_indices.keys())
    class_count=len(classes)
    errors=0
    preds=model.predict(test_gen, verbose=1)
    tests=len(preds)
    for i, p in enumerate(preds):
        pred_index=np.argmax(p)
        true_index=test_gen.labels[i]  # labels are integer values
        if pred_index != true_index: # a misclassification has occurred
            errors=errors + 1
            file=test_gen.filenames[i]
        y_pred.append(pred_index)

    acc=( 1-errors/tests) * 100
    print(f'there were {errors} errors in {tests} tests for an accuracy of {acc:6.2f}')
    ypred=np.array(y_pred)
    ytrue=np.array(y_true)
    if class_count <=30:
        cm = confusion_matrix(ytrue, ypred )
        # plot the confusion matrix
        plt.figure(figsize=(12, 8))
        sns.heatmap(cm, annot=True, vmin=0, fmt='g', cmap='Blues', cbar=False)
        plt.xticks(np.arange(class_count)+.5, classes, rotation=90)
        plt.yticks(np.arange(class_count)+.5, classes, rotation=0)
        plt.xlabel("Predicted")
        plt.ylabel("Actual")
        plt.title("Confusion Matrix")
        plt.show()
    clr = classification_report(y_true, y_pred, target_names=classes, digits= 4) # create classification report
    print("Classification Report:\n----------------------\n", clr)
    return errors, tests
errors, tests = predictor(test_gen, test_steps)

model.summary()
```



#Test accuracy and classification report

In [10]:
```
def predictor(test_gen, test_steps):
    y_pred= []
    y_true=test_gen.labels
    classes=list(test_gen.class_indices.keys())
    class_count=len(classes)
    errors=0
    preds=model.predict(test_gen, verbose=1)
    tests=len(preds)
    for i, p in enumerate(preds):
        pred_index=np.argmax(p)
        true_index=test_gen.labels[i]  # labels are integer values
        if pred_index != true_index: # a misclassification has occurred
            errors=errors + 1
            file=test_gen.filenames[i]
        y_pred.append(pred_index)

    acc=( 1-errors/tests) * 100
    print(f'there were {errors} errors in {tests} tests for an accuracy of {acc:6.2f}')
    ypred=np.array(y_pred)
    ytrue=np.array(y_true)
    if class_count <=30:
        cm = confusion_matrix(ytrue, ypred )
        # plot the confusion matrix
        plt.figure(figsize=(12, 8))
        sns.heatmap(cm, annot=True, vmin=0, fmt='g', cmap='Blues', cbar=False)
        plt.xticks(np.arange(class_count)+.5, classes, rotation=90)
        plt.yticks(np.arange(class_count)+.5, classes, rotation=0)
        plt.xlabel("Predicted")
        plt.ylabel("Actual")
        plt.title("Confusion Matrix")
        plt.show()
    clr = classification_report(y_true, y_pred, target_names=classes, digits= 4) # create classification report
    print("Classification Report:\n----------------------\n", clr)
    return errors, tests
errors, tests = predictor(test_gen, test_steps)

model.summary()
```

```
8/8 [==============================] - 56s 6s/step
there were 39 errors in 600 tests for an accuracy of  93.50
```

```
 dense (Dense)                (None, 512)               786944      ['batch_normalizatio
n[0][0]']

 dropout (Dropout)            (None, 512)               0           ['dense[0][0]']

 dense_1 (Dense)              (None, 256)               131328      ['dropout[0][0]']

 dropout_1 (Dropout)          (None, 256)               0           ['dense_1[0][0]']

 dense_2 (Dense)              (None, 128)               32896       ['dropout_1[0][0]']

 dropout_2 (Dropout)          (None, 128)               0           ['dense_2[0][0]']

 dense_3 (Dense)              (None, 64)                8256        ['dropout_2[0][0]']

 dense_4 (Dense)              (None, 10)                650         ['dense_3[0][0]']

==================================================================================
=============
Total params: 11,749,753
Trainable params: 963,146
Non-trainable params: 10,786,607
```

```
Classification Report:
----------------------
               precision    recall  f1-score   support

    aloevera     1.0000    0.9833    0.9916        60
      banana     0.9672    0.9833    0.9752        60
     bilimbi     0.9000    0.9000    0.9000        60
  cantaloupe     0.8209    0.9167    0.8661        60
     cassava     0.9492    0.9333    0.9412        60
     coconut     0.9649    0.9167    0.9402        60
        corn     0.9655    0.9333    0.9492        60
    cucumber     0.8966    0.8667    0.8814        60
     curcuma     0.9677    1.0000    0.9836        60
    eggplant     0.9322    0.9167    0.9244        60

    accuracy                         0.9350       600
   macro avg     0.9364    0.9350    0.9353       600
weighted avg     0.9364    0.9350    0.9353       600

Model: "model"
```

#Pridict image

In [13]:
```
from tensorflow.keras.preprocessing import image
from tensorflow.keras.applications.efficientnet import preprocess_input

def predict_single_image(model, img_path, target_size=(200, 260)):
    img = image.load_img(img_path, target_size=target_size)
    img_array = image.img_to_array(img)
    img_array = preprocess_input(img_array.reshape(1, *img_array.shape))

    # Make a prediction
    prediction = model.predict(img_array)

    # Decode the prediction
    class_indices = {v: k for k, v in train_gen.class_indices.items()}
    predicted_class = class_indices[np.argmax(prediction)]

    # Display the image and prediction
    plt.imshow(img)
    plt.title(f'Predicted Class: {predicted_class}')
    plt.show()

# Example usage:
img_path_to_predict = 'dataset_type_of_plants_new\coconut\coconut3.jpg'
predict_single_image(model, img_path_to_predict)
```

```
1/1 [==============================] - 6s 6s/step
1/1 [==============================] - 6s 6s/step
```