

JIMMA UNIVERSITY
INSTITUTE OF TECHNOLOGY
FACULTY OF COMPUTING AND INFORMATICS
DEPARTMENT OF INFORMATION TECHNOLOGY

**Enset (*Enset ventricosum*) Plant Disease and pests Identification Using
Image Processing and Deep Convolutional Neural Network.**

BY: TSEGAYE YIBGETA BIZA

A Thesis Submitted to Faculty of Computing and Informatics of Jimma University
in Partial Fulfillment for the Degree of Master of Science in Information
Technology

Advisor: Million Meshesha (PhD)

Co Advisor: Muktar Bedaso (Msc)

Jimma, Ethiopia

Nov, 2021

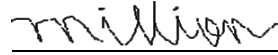
STUDENT NAME: TSEGAYE YIBGETA BIZA

ADVISOR: MILLION MESHESHA (PhD)

CO-ADVISOR: MUKHTAR BEDASO (MSC)

Approved by

1. Dr. Million Meshesha (PHD)



20/12/2021

Advisor

Signature

Date

4. Dr. Worku Jifara



20/12/2021

External Examiner

Signature

Date

DECLARATION

I declare that this thesis, "Enset (Ensete ventricosum) Plant Disease and Pests Identification System Using Image Processing and Deep Learning Approach," is my own work done under the supervision of my adviser. I have acknowledged and referred all materials used in this work in accordance with globally accepted practices, and this work has not been submitted, in whole or in part, for any other degree or professional qualification.

Approved by:

Advisor Name: Dr. Million Meshesha Signature: million Date: _____

Abstract

Enset is a monocarpic perennial crop which belongs to the Schistaminae order and the Musaceae family. Enset is a key food security crop in Southern Ethiopia, where almost 20 million people rely on it for survival. One of the most difficult aspects of Enset production is the necessity for precise and early diagnosis of its diseases. Plant leaf diseases and destructive pests are a major challenge in Enset production. Limited amount of research has been conducted to automate Enset disease detection. The studies conducted were concentrated on bacterial wilt disease detection, the detection of Enset mealybug pests is a forgotten subject but it is a major constraint on Enset production. This thesis looks into the use of deep learning to detect bacterial wilt disease and Enset mealybug pest, where data obtained is small and collected under minimally controlled conditions. We employed data augmentation to get over the limits of the dataset size.

The proposed approach is divided into four stages. The initial part entails gathering healthy and diseased Enset images with the support of agricultural experts, from various farms and research institutes. Then image processing tasks, such as resizing and segmentation are applied on the collected dataset in order to get an enhanced (simpler) image and to extract region of interest from the dataset images. The next step was to design a convolutional neural network that can categorize a given image as healthy, bacterial wilt or mealybug. Finally, using the collected dataset, the created CNN model was trained and evaluated, and it is compared to the state-of-the-art pre-trained convolutional neural network models: AlexNet, ResNet50, Inceptionv3, DenseNet201, VGG16 and EfficientNetB3.

The proposed approach was implemented using google Collaboratory or "Colab" for short. To detect and classify Enset diseases, the model has an accuracy of 99.68% for training and 98.87% for testing. We offer the most effective method for segmenting only the ROI section of Enset leaves. Sigatoka, leaf speckle, and cordana are some of the other diseases found in Enset. Due to a lack of data, we were only able to detect healthy, bacterial wilt, and root mealybug in this research. As a result, we urge that the aforesaid diseases be detected in future studies.

Keywords: - Enset Bacterial Wilt, Enset mealybug, Convolutional Neural Network, Deep Learning, Image processing

Acknowledgments

First and foremost, I want to express my gratitude to the Almighty God, who is the Source of Wisdom and the Origin of Knowledge. I praise and thank Him for all of His assistance during this project. I'd also like to express my gratitude to the Ever-Virgin, St. Mary, Mother of our Lord. Dr. Million Meshesha, my advisor, deserves my sincerest gratitude. I owe him a debt of gratitude for his advice and constructive comments. Conversations with him have always made me believe that anything is possible. I believe that one day I will be able to advise and guide my students as well as he has guided me.

Mr. Muktar Bedaso, my co-advisor, deserves special recognition for his unwavering support, enthusiasm, and vast expertise. I appreciate your taking the time to read and comment on this document's modifications.

My heartfelt gratitude also goes out to the employees of the Gurage Zone Agricultural Institute, the 'Cheha' District Agricultural Institute, and Wolkite University Agriculture College, who have provided invaluable insight into Enset diseases.

I'd want to express my gratitude to domain experts Mr. Gezahegn Alemu and Mr. Mekuria Haile (Domain Experts) for their professional advice provided throughout data gathering.

Contents

Abstract.....	iii
CHAPTER ONE	1
1.1 Background	1
1.2 Motivation.....	2
1.3 Statement of the Problem	2
1.4 Objective of the Study.....	4
1.4.1 General Objective	4
1.4.2 Specific Objectives	4
1.5 Scope and Limitation of the Study	5
1.6 Significance of the study	5
1.7 Organization of the thesis.....	6
CHAPTER TWO	8
2. LITERATURE REVIEW	8
2.1 Enset.....	8
2.2 Enset bacterial wilt	10
2.3 Enset root mealybug	11
2.4 Overview of Digital Image Processing	12
2.4.1 Image processing steps.....	13
2.6 Machine learning	17
2.7 Deep learning.....	19
2.7.1 Deep Feedforward Networks	19
2.7.1.1 Output Units.....	21
2.7.2 Regularization for Deep Learning.....	22
Dataset Augmentation	23
Early stopping	23
Drop out.....	24
2.7.3 Optimization for Training Deep Models.....	24
2.7.3.1 Algorithms with Adaptive Learning Rates	24
2.7.3.2 Optimization Strategies and Meta-Algorithms	25
2.7.3 Convolutional Networks.....	25
2.7.3.1 Convolutional Layer	30
2.7.3.2 Pooling Layer	32

2.7.3.3 Fully Connected (FC) Layer	33
2.8 Related works	37
2.9 Research gap.....	40
CHAPTER THREE	41
RESEARCH DESIGN AND METHODS	41
3.1 Methodology	41
3.1.1 Research Design	41
3.2 Implementation tools	41
3.3 Proposed Model Architecture.....	42
3.4 Image Data Preparation	45
3.4.1 Image Acquisition	45
3.4.2 Image Data Preprocessing.....	46
3.4.3 Segmentation	46
3.4.4 feature extraction	49
3.4.5 Data Augmentation	49
3.5 Classification	50
3.5.1 Training Phase	50
3.6 Evaluation Technique.....	54
CHAPTER FOUR	55
RESULTS AND DISCUSSION	55
4.1 Image Dataset.....	55
4.2 Experimental Results.....	55
4.2.1 Experimenting CNN algorithm by applying image segmentation.....	56
4.2.2 Experimental results of our model with out applying segmentation.....	58
4.2.3 Experimental results of our model without Data Augmentation.....	61
4.2.4 Changing Training and Testing Dataset Ratio.....	63
4.2.5 Changing optimizers	65
4.2.6 Comparing the proposed model with the state-of-the-art models	69
4.2.7 Comparing our model with machine learning classifiers.....	84
4.3 Summary of comparisons between proposed model and state-of-the-art models.....	86
4.4 Discussions	86
CHAPTER FIVE	88
CONCLUSION AND FUTURE WORKS	88

5.1 Conclusions	88
5.2 Contributions	88
5.3 future works	89
References	90

List of Figures

Figure 2. 1 Enset	9
Figure 2. 2 Enset bacterial wilt	11
Figure 2. 3 Enset root mealybug	12

Figure 2. 4 CNN architecture showing the features extraction and classification zones.....	17
Figure 2. 5 Rectified linear unit, or ReLU	20
Figure 2. 6 An example of 2-D convolution without kernel-flipping.....	26
Figure 2. 7 Parameter sharing	27
Figure 2. 8: 7-layer Architecture of CNN.....	28
Figure 2. 9 Conceptual model of CNN.....	28
Figure 2. 10 CNN backpropagation algorithm pseudo code.....	30
Figure 2. 11 Example of a 2×2 kernel	31
Figure 2. 12 A 4×4 Gray-Scale image	Figure 2. 13 A kernel of size 2×2
Figure 2. 14 Illustrating the first 5 steps of convolution operation.....	31
Figure 2. 15 The final feature map after the complete convolution operation	32
Figure 2. 16 max-pooling operation.....	33
Figure 2. 17 CNN architecture.....	33
Figure 2. 18 The Architecture of LeNet.....	34
Figure 2. 19 The Architecture of AlexNet	35
Figure 2. 20 The Architecture of VGGNet	35
Figure 2. 21 (1) Simple Inception Module, (2) Inception Module with dimensionality reduction.....	36
Figure 2. 22 (a) Mapping inside Residual block, (b) Simple direct mappings	36
Figure 2. 23 A DenseNet based model with three dense blocks.....	37
Figure 3. 1 Proposed Model Architecture.....	44
Figure 3. 2 Data Collection Chart.....	45
Figure 3. 3 segmentation of healthy, bacterial wilt and mealybug plant.	47
Figure 3. 4 Segmentation algorithm for segmenting ROI from image.....	48
Figure 3. 5 Summary of our model.....	52
Figure 3. 6 Convoluting a 3×3 kernel over a 5×5 input using 2×2 strides with no zero padding (padding="valid").....	52
Figure 3. 7 convoluting a 3×3 kernel over a 6×6 input padded with a 1×1 border of zeros using 2×2 strides (padding= "same")	52
Figure 3. 8 MaxPooling	53

Figure 4. 1 Classification accuracy and loss of our model	56
Figure 4. 2 training and validation accuracy of our model.....	57
Figure 4. 3 training and validation loss of our model.....	57
Figure 4. 4 confusion matrix of our model on test dataset.....	58
Figure 4. 5 training and validation accuracy and loss.....	59
Figure 4. 6 training and validation accuracy learning curves without segmentation.....	60
Figure 4. 7 training and validation loss without segmentation	60
Figure 4. 8 confusion matrix result on test data without segmentation	61
Figure 4. 9 Performance of our model without Data Augmentation	61
Figure 4. 10 Training and Validation accuracy of our model without Data Augmentation	62
Figure 4. 11 Training and Validation loss of our model without Data Augmentation	62
Figure 4. 12 Our model Prediction result without Data Augmentation	63
Figure 4. 13 training and validation accuracy and loss of 70/30 split	63
Figure 4. 14 training and validation accuracy of 70/30 split	64
Figure 4. 15 training and validation loss of 70/30 split	64
Figure 4. 16 confusion matrix of 70/30 split.....	65
Figure 4. 17 training and validation accuracy and loss by applying AdaGrad Optimizer.....	65
Figure 4. 18 training and validation accuracy graph of our model by applying AdaGrad optimizer ..	66
Figure 4. 19 training and validation loss graph of our model by applying AdaGrad optimizer ...	67
Figure 4. 20 Confusion matrix of our model after applying AdaGrad optimizer	67
Figure 4. 21 training and validation accuracy and loss with SGD.....	67
Figure 4. 22 training and validation accuracy graph with SGD	68
Figure 4. 23 training and validation loss with SGD.....	68
Figure 4. 24 confusion matrix after applying SGD.....	69
Figure 4. 25 AlexNet training and validation accuracy and loss	70
Figure 4. 26 AlexNet training and validation accuracy graph	71
Figure 4. 27 AlexNet training and validation loss graph	71
Figure 4. 28 AlexNet model Confusion matrix	72

Figure 4. 29 VGG16 training and validation accuracy and loss	73
Figure 4. 30 VGG16 training and validation accuracy graph.....	73
Figure 4. 31 VGG16 training and validation loss graph.....	74
Figure 4. 32 VGG16 confusion matrix	74
Figure 4. 33 ResNet model training and validation accuracy and loss.....	75
Figure 4. 34 ResNet model training and validation accuracy graph.....	76
Figure 4. 35 ResNet model training and validation loss graph.....	76
Figure 4. 36 ResNet model confusion matrix	77
Figure 4. 37 Inceptionv3 training and validation accuracy and loss.....	78
Figure 4. 38 Inceptionv3 training and validation accuracy graph	78
Figure 4. 39 Inceptionv3 training and validation loss graph	79
Figure 4. 40 Inceptionv3 confusion matrix.....	79
Figure 4. 41 DenseNet201 training and validation accuracy and loss	80
Figure 4. 42 DenseNet201 training and validation accuracy graph.....	81
Figure 4. 43 DenseNet training and validation loss graph.....	81
Figure 4. 44 DenseNet201 confusion matrix	82
Figure 4. 45 EfficientB3 training and validation accuracy and loss	83
Figure 4. 46 EfficientNetB3 training and validation accuracy graph	83
Figure 4. 47 EfficientNetB3 training and accuracy loss graph.....	83
Figure 4. 48 EfficientNetB3 confusion matrix	84
Figure 4. 49 Random Forest Classifier	85
Figure 4. 50 XBoost Classifier.....	85
Figure 4. 51 SVM classification	85

List of Tables

Table 2. 1 Frequently reported Enset production constraints (McKnight-CCRP, 2013).....	10
Table 2. 2 comparison between segmentation methods.....	15
Table 2. 3 Summary of related works	39

Table 3. 1 Augmentation techniques used	50
Table 4. 1 Summary of comparisons	86
Table 4. 2 summary of comparison between our model and other models from related works...	87

List of Abbreviations and Acronyms

ANN: Artificial Neural Network

BWE: bacterial wilt disease of Enset

CNN: Convolutional Neural Network

CSA: Central statistics Agency

DNN: Deep Neural Network)

FC: Fully Connected

GPU: Graphics Processing Unit

HOT: Histogram of Template

RGB: Red, Green and Blue

ReLU: Rectified Linear Unit

CHAPTER ONE

1.1 Background

Enset *ventricosum* (false banana) has high significance in the day-to-day life of more than 20 million Ethiopians as a food source, fiber, animal forage, construction materials, and medicines. [1]. Enset is mostly grown in south and south western region of Ethiopia. The corm and pseudo stem of Enset are the most important food sources, which are harvested as kocho (fermented starch obtained from decorticated (scraped) leaf sheaths and grated corms), bulla (a white powder obtained by dehydrating squeezed sap from scraped leaf sheaths and grated corms), and Amicho (boiled corm pieces of young Enset plants, prepared and consumed in a similar manner) [2].

According to Central Statistical Authority (CSA), about 123,479,334.00 Enset crops were harvested across the country in the 2016/17 agricultural season, yielding total production of 2,800,977.87 tons, 3,162,563.18 tons, and 1,10,060.62 tons in the form of Amicho, Kocho, and Bula, respectively. However, the production of Enset is hampered by biotic and abiotic agents such as insect pests, weeds, wild animals and soil nutrient depletion. Enset diseases are caused by several bacteria, fungi, viruses, and nematodes. The Enset bacterial wilt and mealybug are the major constraints in the production of Enset [3].

Despite the fact that many farmers in underdeveloped countries lack access to these advanced equipment's, internet and smartphone penetration have provided new tools for identifying in-field crop diseases. Global smartphone subscriptions are expected to exceed 10 billion by 2021, according to the Global System for Mobile Association (GMSA), with roughly 2 billion in Africa [4]. It is believed that digital Image Processing and machine learning are transforming agriculture and assisting farmers in forecasting their future.

The field of digital image processing refers to the use of a digital computer to process digital images. It's important to remember that a digital image is made up of a finite number of pieces, each of which has its own location and value. These are referred to as picture elements (pixels). There is no consensus among authors on where image processing ends and other related fields like image analysis and computer vision begin. Image processing is sometimes defined as a discipline in which images serve as both the input and output of a process. The ultimate objective of computer vision is to have computers mimic human vision, which includes learning and being able to draw

inferences and conduct actions based on visual inputs. This field is a subset of artificial intelligence (AI), with the goal of simulating human intelligence [5].

Convolutional neural networks (CNNs) are a type of neural network that processes input using a mathematical operation called convolution. CNN is an architecture which is being used for different computer vision tasks. Each layer of CNN learns to extract features from input images that will be used to classify the images in the end. The benefit of using a deep learning approach is that it reduces the number of image processing steps required when compared to typical machine learning methods. The aim of this thesis is to detect and classify Enset diseases and pests using image processing techniques and deep convolutional neural network.

1.2 Motivation

In Ethiopia, more than 20 million people rely on Enset as a source of sustenance. However, the bacterial wilt disease of Enset (BWE) has severely harmed Enset output in the country. The disease is widespread in the country's major Enset-growing regions, and it affects the crop at all phases of development, resulting in losses of up to 100% in severe cases. Currently, BWE has infected over 80% of Enset farms, and no Enset clone has been identified that is totally resistant to bacterial wilt [1].

In southern Ethiopia, the Enset root mealybug (*Cataenococcus Ensete* Williams and Matile Ferrero) has emerged as the most important insect pest of Enset (*Ensete ventricosum*) [6]. In 2005, farmers' fields in Yirgachefe, southern Ethiopia, were researched for the distribution of the Enset root mealybug on Enset roots and corms of the 'Genticha' clone. Each plant had an average of 87 mature Enset root mealybugs retrieved from its roots and corms [7].

The most significant aspect of plant disease management is detecting the disease as soon as it occurs on the farm, which may be accomplished by utilizing various computing infrastructures such as computer vision [8]. Deep learning simplifies the difficult feature extraction problem; hence it will be used to recognize images with associated features. We can now detect and classify several plant diseases using deep convolutional neural networks.

1.3 Statement of the Problem

The bacterial wilt and mealybug diseases of Enset are widespread in the country's major Enset growing regions causing losses of up to 100% in farm fields in extreme cases. Currently, BWE has

infected over 80% of Enset farms, and no Enset clone has been identified that is totally resistant to bacterial wilt [1]. Mealybug is another major disease of Enset, and about 37.6% of Enset plant is infected by these pests [9]. Mealybugs were recorded in Sidama, Gedeo, Gurage, Bench, Kembata Tembaro, Keffa, and Hadyia zones and Amaro and Yem districts [6].

Because the majority of Ethiopian farmers are uneducated and do not receive accurate and thorough information regarding Enset crop diseases, they have wrong concept about the symptoms and causes of the diseases. Hence, they require advice of experts such as pathologists. On the other hand, it is impossible for crop pathologists to visit every farm, and because even crop pathologists rely on manual eye inspection, the manual prediction approach is ineffective, time-consuming, and labor-intensive.

The shift in color of the leaf from green to yellow indicates bacterial wilt. It's also possible to see the yellowish mucus dripping from the pseudo stem. Another sign is that the corm of the plant appears to have yellow patches. Therefore, we can design a model that recognizes the disease in the crop based on the symptoms that are directly visible in the crop leaf. The other constraint is mealybugs, which are white-colored pests that attack the top root parts of Enset. Mealybug-infested Enset plants exhibit retarded growth, loss of vigor, dried lateral leaves but green central shoot and eventually plant death. By looking at the top root parts of the plant we can identify them by their whitish color.

Other minor but commonly reported diseases of Enset besides bacterial wilt and mealybug are black sigatoka and Corm rot [10]. To the best of our knowledge, only two studies on the detection of bacterial wilt of Enset have been undertaken, and no studies on the detection of mealybug have been conducted. These two diseases are the primary causes of Enset production declination. Hence, it is crucial to design a model that can automatically identify and classify the two major constraints of Enset production.

Identifying plant diseases using computer vision has been done for more than a decade and it has produced promising results. But only few researches have been conducted in Enset disease detection and classification.

Yidnekachew [11] has developed bacterial wilt detection model on Enset crop using a deep learning approach. However, due to a lack of computing resources, the largest and most well-

known CNN architectures, such as AlexNet and ResNet, have not been deployed in order to build a high-performing model for disease identification, Only the VGG16 and Inceptionv3 CNN architectures were considered cutting-edge. In this paper, we have looked into AlexNet, VGG16, ResNet, Iceptionv3, DenseNet, and EfficientNet pre-trained models. Furthermore, the study only considers bacterial wilt disease. As per our discussion with expert's, mealybug is also one of the most common Enset infections, although no automated identification of this disease has been undertaken.

Kibru and Getahun [12] has developed Enset Diseases Diagnosis Model Using Image Processing and Machine Learning Techniques. The researchers selected two Enset leaf diseases, Bacterial Wilt and Fusarium Wilt, and gathered 430 Enset leaf images from the Areka agricultural research center and a few selected sites in the SNNPR. CNNs have defied expectations and ascended to the throne as the most advanced computer vision technique [13]. In this thesis, the best deep learning algorithm for image classification was employed, which is the CNN model.

As a result, there is a need to develop an autonomous disease detection model that aids farmers and agricultural research institutes in detecting diseases earlier and with more precision. It is therefore the aim of this study to apply image processing and deep learning to detect the above mentioned major Enset plant diseases.

Finally, this thesis attempts to investigate and answer the following research questions.

1. Which state-of-the-art CNN model is suitable for detecting and classifying Enset plant diseases and pests?
2. Which machine learning algorithm is suitable for classifying Enset plant diseases and pests?
3. To what extent the proposed model works in detecting Enset diseases?

1.4 Objective of the Study

1.4.1 General Objective

The main objective of this thesis is to design and develop an automatic Enset diseases and pests' identification model by using image processing and deep CNN.

1.4.2 Specific Objectives

To achieve the general objective, the following specific objectives should be achieved:

- Review literature to identify major Enset diseases and pests, to identify best image classification algorithms
- To Collect datasets of Enset according to the three classes we are going to classify.
- To construct algorithms that are best suited for colour image segmentation.
- To make comparison between machine learning algorithms.
- To construct a CNN model for automating Enset disease detection
- Test and evaluate the proposed model with an unseen dataset.
- To give conclusion and recommendation for future studies

1.5 Scope and Limitation of the Study

The main focus of this research is designing, modeling and development of an automatic detection of Enset disease and pests by using deep convolutional neural networks. Two types of Enset plant diseases are considered, they are Bacterial wilt and mealybugs. Other types of minor Enset diseases like sigatoka, corm rot and leaf speckles are beyond the scope of this research. After identifying the disease of the Enset plant there may be a way to recommend the appropriate medicine for that disease, however recommending the appropriate treatment for the identified disease and estimating the severity of the detected disease is beyond the scope of this thesis work.

With the assistance of domain experts, image data was collected from the Wolkite University incubator center, Gurage zone agricultural institute, and local Enset farms. It is recommended to collect 1000 images per class to train a CNN model. But in this research, we have been able to collect only 987 images for all the three classes.

1.6 Significance of the study

The development of such a system has a great advantage for the Enset production industry. Some of the significances of this work include the following:

The developed model will help farmers, new investors, experts and others to identify Enset diseases and pests in fast, more accurate and simple way. This study also helps in early-stage Enset diseases and pests' identification: The development of this model will increase the effectiveness of disease controlling mechanism. The end product of this model helps farmers to take an effective protection approach by identifying the disease in its early stage.

The study offers the best strategy for segmenting only the ROI region of the plant leaves and mealybugs so that the model may learn the important attributes more quickly.

This study is a start for smart farming of Enset plant. It is also a benchmark for further studies on Enset plant disease and pests, based on which to develop disease identification systems for Enset and other agriculture products. On the other hand, the results of this research work have a great impact for the upcoming efforts to detect and classify Enset disease. The model developed in this study also has its own contribution towards overcoming the gaps observed in the current related works. In addition, this research work enables to optimize the amount of time taken (for both agricultural institutes and farmers) and money spent for diagnosis. The research work also enables early diagnosis and future works can be employed to suggest treatments accordingly with the severity of the diseases. Consequently, the production of Enset can increase. Due to this and the above reasons, we conduct this research work.

1.7 Organization of the thesis

The remainder of this thesis is organized in the following manner. The second chapter examines the many literatures that were reviewed in relation to Enset diseases and pests, digital image processing, and CNN. Furthermore, we have included a full overview of the CNN building blocks. Finally, the most successful CNN designs in the fields of image processing and pattern recognition are discussed.

The second chapter is also devoted to a discussion of related works on Enset plant disease detection. We reviewed studies on banana plant disease identifications because Enset is physically nearly identical to banana plants. Only those works are discussed whose contributions are relevant to our work. Furthermore, the prevalent gaps in the reviewed publications, as well as how we fill in the gaps, are thoroughly described.

A detailed explanation of the proposed system is provided in Chapter three. The system's components (preprocessing, segmentation, feature learning, and classification using CNN) are detailed in length, as well as the responsibilities of each component.

The experimental results of the suggested model for automatic Enset disease and pests' detection and classification is detailed in Chapter Four. The dataset used and the proposed model's

implementation are thoroughly discussed. Finally, the outcomes of the tests are compared to current state-of-the-art models.

The primary findings of this research endeavor will be summarized in Chapter Six. The major contributions of the proposed deep CNN model will also be discussed, as well as future development recommendations are included.

CHAPTER TWO

2. LITERATURE REVIEW

The field of image processing is becoming very important in everyday life. Digital image processing is a technology which is used in many research areas, such as information retrieval, biometric analysis, and agriculture.

This chapter provides an overview of Enset diseases and pests, image processing and its functionality, also a discussion of image processing steps followed, such as image preprocessing, image segmentation and feature extraction. A literature review is made concerning machine learning algorithms with specific consideration of deep learning models. Finally, related works review is provided to define the research gap this study attempts to fill.

2.1 Enset

Ensete ventricosum (Welw.) Cheesman, commonly known as Enset, is an Ethiopian monocarpic perennial plant of the Musaceae family. Enset is a multipurpose Ethiopian indigenous crop that provides year-round food security, as well as traditional medicine and fiber. The Enset cultivation strategy is both economically and agriculturally feasible in Ethiopia. In Enset culture, the most common organic amendments are cattle dung and household garbage. Every element of the plant can be utilized in some way. Farmers frequently remark that Enset provides them with food, clothing, shelter, bedding, cattle fodder, and a plate [22].



Figure 2. 1 Enset

Enset (a banana related plant) provides human food, fiber, animal feed, construction materials, and pharmaceuticals to more than 20% of Ethiopia's population [7]. However, a variety of problems endanger the long-term viability of Enset-based agriculture. Bacterial wilt, the Enset root mealybug, nematodes, fungi, and other vertebrate pests such as mole rats are the principal biotic stressors [6]. Table 2.1 depicts main Enset production constraints in Ethiopia's various zones.

Table 2. 1 Frequently reported Enset production constraints (McKnight-CCRP, 2013)

Major constraints in enset production	Zone							Total
	Silti	Gurage	Kembata	Sidamo	Dawro	Gedeo	Wolayta	
BWE	32.1	19.4	14.3	66.7	84.4	93.1	90.9	42.3
Enset root millibug	7.1	5.6	49.3	60.0	40.6	56.8	22.7	21.4
Leaf hoper	3.6	2.8	0.0	13.3	37.9	36.4	22.7	11.3
Mole rat	21.4	25.0	7.1	60.0	50.0	4.6	4.5	30.4
Porcupine	25.0	86.1	42.9	63.3	43.8	0.0	0.0	51.2
Swine	0.0	0.0	4.8	20.0	0.0	0.0	0.0	6.0
Corm rot	42.9	83.3	28.6	36.7	78.1	54.4	45.5	54.2
Drought	0.0	8.3	9.5	0.0	0.0	0.0	0.0	4.2

2.2 Enset bacterial wilt

Enset bacterial wilt is caused by *Xanthomonas campestris* pv. *Musacearum*. The disease is a major constraint of Enset production in Ethiopia. The bacterium is a motile, Gram-negative rod that is aerobic, has a single polar flagellum, and produces typical yellow, convex, mucoid colonies. The optimum temperature for growth is usually 25-30°C. *X. campestris* pv. *musacearum* is known to systemically invade all tissues of Enset and banana after infection. The pathogen enters the host through wounds on roots, pseudo stems, and leaves. Bacterial wilt attacks Enset at any stage of growth, including full maturity. The bacterium can live on Enset leaves and stem for at least four months, at least three months in soil and it can also live at least three or four days on farming equipment.

The initial symptoms of the disease occur on the central leaf and spread to all parts. Bacterial ooze exudes when non-dry part of the plant is removed. A typical bacterial wilt symptom in Enset plants older than two years is that the innermost leaf sheaths become yellowish and droop. In the leaf, Loss of turgor and wilting in the spear (youngest emerging leaf) are frequently the first symptoms, followed by yellowing and deformation, especially in young plants. Internally, vascular bundles have a cream, yellow, or pinkish discoloration that might affect the entire plant. The large air spaces within the leaf bases become filled with pockets of cream to pale yellow ooze and this characteristic appears to distinguish *Xanthomonas* wilt of Enset from other bacterial wilts of banana [23].

Research institutes have been working on producing superior Enset species with strong bacterial resistance. Agricultural institutes suggest that one of the ways to prevent the bacterial wilt is Sanitary control measure, which means keeping Enset manufacturing materials clean, which has yielded a good result in Gurage zone ‘Gumer’ and ‘Sodo’ districts [24].

The level of destruction could lead to 100% destruction of the crop field. Bacterium wilt can be transmitted from one plant to another in many ways like residues of an infected plant, farming utilities, birds, animals, infected soil and insects [25].



Figure 2. 2 Enset bacterial wilt

2.3 Enset root mealybug

Root mealybug is a generic term for a number of *Pseudococcidae* feeding on underground plant parts. Enset root mealybugs have an elongate-oval body covered with wax secretions on the dorsal and lateral sides. Cottony, spine-like projections form as a result of the wax secretions. Despite the fact that these waxy secretions are not a part of the mealybug's body, they are shed with each molt. Enset root mealybug, *Cataenococcus Ensete* is a major pest of Enset in Enset growing areas of southern Ethiopia. It has been reported from Wonago as a new record for Ethiopia [26]. The insect is known to attack Enset in Gedeo, Sidama, Gurage, Kembata Tembaro, Hadyia, Keffa and Bench zones and Amaro and Yem districts [6].

The pest attacks Enset plants at any age, with infestations being most serious on 2-4 year old plants. Mealybug-infested Enset plants exhibit retarded growth, loss of vigor, dried lateral leaves but green central shoot and eventually plant death [27]. Empirical data on Enset yield loss as a result of mealybug attack are scanty. According to Addis [27], more than 30% of the sampled Enset

farms were infested with Enset root mealybugs. According to Addis et al. [9], The distribution of the Enset root mealybug on Enset roots and corms of the ‘Genticha’ clone was studied in 2005 on farmers’ field at Yirgachefe, southern Ethiopia and an average of 87 adult Enset root mealybugs were collected from roots and corms per plant.

Generally, mealybugs are difficult to control with insecticides due to their cryptic nature, waxy-coat and life-style of forming dense colonies of multiple and overlapping generations [28].



Figure 2. 3 Enset root mealybug

2.4 Overview of Digital Image Processing

When photos were first transported by undersea cable between London and New York, one of the first applications of digital image was in the newspaper sector. The Bartlane cable picture transmission system, introduced in the early 1920s, cut the time it took to send a picture over the Atlantic in half. The selection of printing techniques and the distribution of intensity levels were two of the first issues in improving the visual quality of these early digital photos [14].

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 or grey level of the image at that point. When x, y and the intensity values of f are all finite, discrete quantities, we call the image a digital image. The field of digital image processing refers to processing digital images by means of digital computer. Note that a digital image is composed a finite number of elements, each of which has a particular location and value. These elements are called picture elements, image elements, pels and pixels. Pixel is the terms widely used to denote picture elements of a digital image [29].

2.4.1 Image processing steps

Image processing pass through image preprocessing, segmentation and feature extraction so as to prepare and extract image representations for the purpose of classification and identification of plant diseases including Enset.

2.4.1.1 Image preprocessing

The goal of pre-processing is to improve the image data by suppressing unwanted distortions or enhancing certain visual qualities that are useful for subsequent processing.

According to the size of the pixel neighborhood utilized for calculating a new pixel brightness, image pre-processing algorithms are divided into four groups. They are pixel brightness modifications, geometric transformations, the treated pixel's local neighborhood, and image restoration, which necessitates knowledge of the complete image [14].

Pixel brightness transformations

A brightness transformation alters the brightness of pixels, and the transformation is based on the attributes of the pixels themselves. Brightness corrections and grey scale transformations are the two types of pixel brightness adjustments. The brightness of a pixel is adjusted through brightness correction, which takes into account the pixel's initial brightness as well as its position in the image. Grey scale conversions alter the brightness of an image regardless of its location.

Geometric transformations

Geometric transforms are widely employed in computer graphics, and they're also frequently used in image analysis. Geometric transformations allow for the removal of geometric distortion that happens during the acquisition of a picture. Geometric modification may be required when attempting to match two different photographs of the same object. Geometric changes are only considered in 2D because this is sufficient for digital photos.

Local pre-processing

Pre-processing algorithms that employ a tiny neighborhood of a pixel in an input image to produce a new brightness value in the output image are the focus of this section. If signal processing terminology is utilized, such pre-processing processes are referred to as filtration (or filtering).

Image restoration

Image restoration refers to pre-processing procedures that try to prevent degradation by employing information of its nature. The majority of picture restoration techniques rely on convolution applied to the entire image.

Images can be degraded for a variety of reasons, including optical lens imperfections, electro-optical sensor non-linearity, graininess of the film material, relative motion between an object and the camera, incorrect focus, atmospheric turbulence in remote sensing or astronomy, and so on. The goal of picture restoration is to recreate the original image from a deteriorated copy.

2.4.1.2 Image Segmentation

Segmentation is the process of dividing a digital image into several parts in computer vision (sets of pixels, also known as super pixels) [30]. Image segmentation is frequently used to locate objects and boundaries (lines, curves, and so on) in images. In more technical terms, image segmentation is the process of assigning a label to each pixel in a picture so that pixels with the same label have similar visual characteristics. The output of image segmentation is a series of segments or contours taken from the image that collectively cover the full image. Each of the pixels in a region are similar with respect to some characteristic or computed property, such as color, intensity, or texture. Due to the necessity of image segmentation, a variety of algorithms have been developed, but the optimum technique should be chosen based on the input image.

The purpose of segmentation is to restructure and transform an image's representation into something more significant and less difficult to understand [31]. The result of image segmentation is a set of areas that, all things considered, cover the entire image, with each pixel in each zone being compared to some trademark or registered feature, such as color, intensity, or texture. With gray images of plant leaves, edge based and threshold segmentation techniques are commonly utilized, whereas color images of leaves are segmented using Region Based, Clustering, and Watershed segmentation approaches [30].

Edge detection

This method is used to locate the leaf borders inside a picture. It divides the image into sections by detecting the difference in brightness at the border. Some of the approaches used for edge detection include Sobel, Canny, Laplacian, and fuzzy logic [29].

Region Based

The basic goal of this strategy is to divide an image into several types of sections that are all the same. Pixels of the same type are detected and grouped together into areas of the same type. Region Growing, Region Splitting, and Region Merging are the three main methods of region-based segmentation [32]. A process for grouping pixels or subregions into bigger areas based on predetermined growth parameters is known as region growing. The main idea is to start with a set of "seed" points and generate areas from them by attaching nearby pixels with predetermined features comparable to the seed to each seed (such as ranges of intensity or color). The technique of region splitting and merging is to divide an image into a set of disjoint areas, then combine and/or split the regions.

1. Region segmentation using Clustering

Clustering is a segmentation technique in which a picture is first converted to a histogram, and then clustering is applied to it. K-means, which is designed for segmentation in textured images, is one of the most fundamental clustering techniques. It segments the image by clustering similar pixels [33].

2. Region segmentation using Watersheds

The topological analysis notion is used in the watershed-based method. The basins are represented by the intensity. The watershed approaches consider the image's gradient as a topographic surface. The pixels with the most gradient are depicted as unbroken boundaries [5].

Table 2. 2 comparison between segmentation methods [32]

<i>Segmentation Technique</i>	<i>Advantages</i>	<i>Disadvantages</i>
Edge Based	Good for images having better contrast between objects.	Not suitable for wrong detected or too many edges
Threshold	Does not require prior information of the image. Fast and simple for implementation	Depends on peaks, spatial details are not considered, Highly noise sensitive
Region Based	More immune to noise, useful when it is easy to define similarity criteria. Works well for images having good contrast between regions.	Time and memory requirement is more so expensive method.
Clustering	For small values of k, k-means is computationally faster. Eliminates noisy spots. More homogeneous regions are obtained.	Difficult to predict k with fixed number of clusters. Computationally expensive.
Watershed	Results are more stable, detected boundaries are	complex calculation of gradients

2.4.1.3 Feature Extraction

The technique of representing a raw image in a reduced form to aid decision making such as pattern detection, classification, or recognition is known as feature extraction [29]. Finding and extracting trustworthy and discriminative features is always an important part of the image recognition and computer vision process. Furthermore, as the number of application needs grows, a thorough examination and inquiry in the field of feature extraction becomes increasingly vital.

Feature Types

The features that will be implemented are determined by the sort of system in which they will be used. The most common categories employed in image processing are spectral, geometric, and texture features [34].

1. Spectral features

Color space is the most fundamental sort of spectral feature, and it is concerned with the color distribution over the image. The color histogram, which shows the joint probability of the intensities of color channels, is one of the primary techniques used to retrieve this information. The cumulative color histogram, which was designed to lower noise sensitivity, is an enhanced version of this approach. Additional approaches were used since the color histogram is particularly susceptible to quantization noise.

2. Shape Features

Shape identification is one of the most fundamental issues in image processing. While a human may easily locate shapes on an image, the existing system requires a lot of effort to obtain the same result. As a result, a variety of shape-based characteristics have been developed to aid with this process. It was discovered that dividing these features into two categories is critical. The first category includes features that are invariant to translation, rotation, and scaling, whereas the second category includes features that do not. The first collection of features is usually easier to extract and requires fewer steps.

3. Texture features

Texture characteristics are visual patterns with properties of homogeneity that aren't caused by the presence of only one color or intensity. These characteristics provide crucial information on the structural arrangement of surfaces and their relationship to their surroundings.

2.4.1.4 feature extraction in CNN

A convolutional neural network is a type of neural network that was created to analyze multi-dimensional data such as images and time series data (CNN). During the training phase, it includes feature extraction and weight computation. A convolution operator, which is beneficial for solving complex processes, is used to give these networks their names. The truth is that CNNs offer automatic feature extraction, which is their main benefit [35]. The given input data is first sent to a feature extraction network, and the retrieved features are then sent to a classifier network.

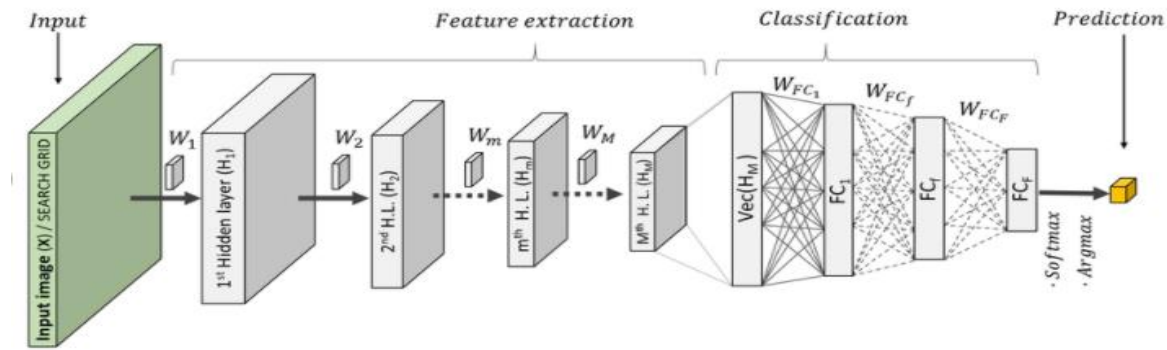


Figure 2. 4 CNN architecture showing the features extraction and classification zones [36].

2.6 Machine learning

machine learning as a set of methods that can automatically detect patterns in data, and then use the uncovered patterns to predict future data, or to perform other kinds of decision making under uncertainty (such as planning how to collect more data!).

A machine learning algorithm is an algorithm that is able to learn from data. But what do we mean by learning? Mitchell [37] provides the definition “A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P , if its performance at tasks in T , as measured by P , improves with experience E .”

The majority of machine learning problems are expressed in terms of how the system should deal with a specific case. In this situation, the example is a set of objectively measured features from an item or event that we want the machine learning system to process. The features of an image, for example, are usually the values of the pixels in the image.

To assess a machine learning algorithm’s capability, we must devise a quantifiable measure of its performance. This performance measure P is usually particular to the job T that the system is performing. We frequently assess the model's accuracy for tasks such as classification,

classification with missing inputs, and transcribing. The number of classifications a model successfully predicts divided by the total number of predictions made is known as model accuracy. We're usually interested in how well a machine learning algorithm works on data it hasn't seen before, because this affects how well it will function in the real world. As a result, we analyze these performance metrics using a distinct test set of data from the data used to train the machine learning system.

Machine learning algorithms can be broadly categorized as unsupervised or supervised by what kind of experience they are allowed to have during the learning process. A dataset is presented to the machine learning algorithms in order to be trained. A dataset is a collection of a large number of input data points gathered for the model's training (in our case, images).

Unsupervised learning methods encounter a dataset with a large number of features and subsequently learn relevant information about the dataset's structure. We normally wish to learn the whole probability distribution that generated a dataset in the context of deep learning, whether directly as in density estimation or implicitly for tasks like synthesis or denoising. Other unsupervised learning algorithms do additional tasks, such as clustering, which divides the dataset into groups of comparable samples.

A dataset with features is presented to supervised learning algorithms, but each sample is additionally tagged with a label or target. Our dataset, for example, includes both diseased and healthy Enset plants. Based on their measurements, **a supervised learning algorithm** can examine the dataset and learn to classify it into three separate groups (healthy, bacterial wilt, and mealybug). One of the major supervised algorithms is classification.

In this study supervised learning algorithms are used for image classification. Image classification is the work of assigning pixels in the image to categories or classes of interest [21]. In order to classify a set of data into different classes or categories, the relationship between the data and the classes into which they are classified must be well understood. To achieve this by computer, the computer must be trained.

Classification is the process of assigning spectral classes into information classes. Spectral classes are groups of pixels that are uniform with respect to their brightness values in the different spectral channels of data. Information classes are categories of interest that an analyst attempts to identify in the image on the basis of his knowledge and experience about the area [30].

2.7 Deep learning

Deep learning (DL) is becoming more and more significant in our daily lives. It has already had a significant impact in fields including cancer detection, precision medicine, self-driving cars, predictive forecasting, and speech recognition, among others. Traditional learning, classification, and pattern recognition algorithms require carefully built feature extractors that are not scalable for big data sets [32]. DNN (Deep Neural Network) is a type of neural network that employs several (deep) layers of units and highly optimized algorithms and structures. There are many deep learning architectures, for example, Deep convolution networks, deep residual networks, recurrent neural networks, reinforcement learning and variational autoencoders. Convolutional neural networks have a number of advantages in image classification and have performed well in a variety of object recognition tasks, owing to their network structure's ability to extract multi-level characteristics from images [39]. As a result, we'll be using CNN to develop our model.

2.7.1 Deep Feedforward Networks

The fundamental deep learning models are deep feedforward networks, often known as feedforward neural networks or multilayer perceptron's (MLPs). A feedforward network's purpose is to approximate a function f^* . For example, $y = f^*(x)$ transfers an input x to a category y in a classifier. A feedforward network learns the values of the parameters that result in the best function approximation by defining a mapping $y = f(x; \theta)$ [40].

Because information passes through the function being evaluated from x , the intermediate calculations necessary to define f , and finally to the output y , these models are referred to as feedforward models. There are no feedback links, therefore the model's outputs do not feed back into it. Recurrent neural networks are feedforward neural networks that have been extended to incorporate feedback connections. Convolutional networks are a type of feedforward networks that are used to recognize objects in pictures [41].

Because feedforward neural networks are often depicted by combining together several different functions, they are referred regarded as networks. For example, we might have three functions $f(1)$, $f(2)$, and $f(3)$ connected in a chain, to form $f(x) = f^{(3)}(f^{(2)}(f^{(1)}(x)))$ [40]. In this case, $f^{(1)}$ is called the first layer of the network, $f^{(2)}$ is called the second layer, and so on. The depth of the model is determined by the entire length of the chain. The phrase "deep learning" is derived from this terminology. The output layer is the last layer of a feedforward network.

We drive $f(x)$ to match $f^*(x)$ during neural network training. We can use the training data to generate noisy, approximation samples of $f(x)$ at various training points. A label $y \approx f^*(x)$ is attached to each sample x . The output layer must generate a value that is close to y at each point x , as specified in the training examples [40].

The training data does not directly influence the behavior of the other layers. The learning algorithm must figure out how to employ those layers to get the intended result, but the training data doesn't specify what each layer should do. Instead, the learning algorithm must figure out how to best employ these layers to approximate f . These layers are called hidden layers because the training data does not reflect the expected output for each of them [42].

Finally, these networks are known as neural networks because they are based on neuroscience. The network's hidden layers are usually vector-valued. The width of the model is determined by the dimensionality of these hidden layers. Each element of the vector can be thought of as having a neuron-like function. Rather than thinking of the layer as a single vector-to-vector function, we might think of it as a collection of units acting in concert, each representing a vector-to-scalar function. Each unit functions similarly to a neuron in that it gets input from a variety of other units and calculates its own activation value.

The concept of a hidden layer was introduced by feedforward networks, and this necessitates selecting the activation functions that would be used to compute the hidden layer values. The rectified linear unit, or ReLU, is the default activation function suggested in current neural networks [43]. When this function is applied to the output of a linear transformation, a nonlinear transformation is produced. However, in the sense that it is a piecewise linear function with two linear elements, the function remains very close to linear.

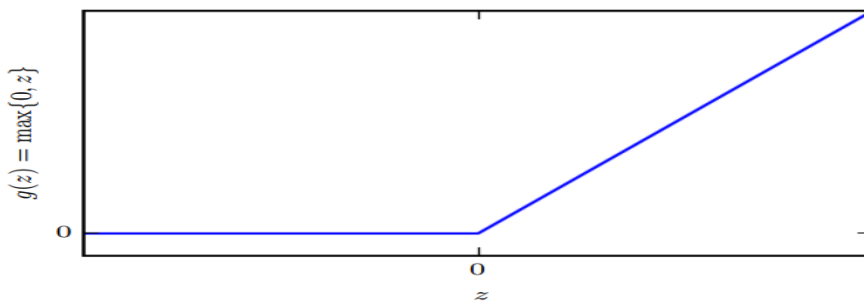


Figure 2. 5 Rectified linear unit, or ReLU [44]

2.7.1.1 Output Units

The output layer is in charge of generating the final product. Any unit that can be used as an output in a neural network can also be utilized as a hidden unit. Throughout this section, we suppose that the feedforward network provides a set of hidden features defined by $\mathbf{h} = \mathbf{f}(\mathbf{x}; \theta)$ [45]

Linear Units

Given features \mathbf{h} , a layer of linear output units produces a vector $\hat{\mathbf{y}} = \mathbf{W}^T \mathbf{h} + \mathbf{b}$. The mean of a conditional Gaussian distribution is frequently produced using linear output layers:

$$p(\mathbf{y} | \mathbf{x}) = \mathcal{N}(\mathbf{y}; \hat{\mathbf{y}}, \mathbf{I}).$$

Because linear units do not saturate, they are easy to employ with gradient-based optimization techniques and may be utilized with a wide range of algorithms [46].

Sigmoid Units

Predicting the value of a binary variable y is required for many jobs. This is how two-class classification problems can be expressed. For example, we could have used this output activation function if the outcome of our categorization was diseased or healthy.

The maximum-likelihood method involves defining a Bernoulli(binomial) distribution over y that is conditional on \mathbf{x} . The maximum-likelihood method involves defining a Bernoulli distribution over y that is conditional on \mathbf{x} ($P(y = 1 | \mathbf{x})$). For this number to be a valid probability, it must lie in the interval $[0, 1]$ [40].

A sigmoid output unit is defined by: [40]

$$\hat{y} = \sigma(\mathbf{w}^T \mathbf{h} + b)$$

where σ is the logistic sigmoid function [47]. The sigmoid output unit can be thought of as having two parts. First it computes using a linear layer to calculate $z = \mathbf{w}^T \mathbf{h} + b$. Next, it uses the sigmoid activation function to convert z into a probability

Softmax Units

The softmax function can be used to present a probability distribution over a discrete variable with n potential values at any moment. We now need to create a vector to generalize to the case of a discrete variable with n values $\hat{\mathbf{y}}$, with $\hat{y}_i = P(y = i | \mathbf{x})$. To depict a legitimate probability distribution, we require not only that each element of \hat{y}_i be between 0 and 1, but also that the entire

vector adds to 1. The multinoulli (categorical) distribution can be approached in the same way as the Bernoulli distribution.

First, a linear layer predicts unnormalized log probabilities: [48]

$$\mathbf{z} = \mathbf{W}^T \mathbf{h} + \mathbf{b}$$

Where, $z_i = \log P(y = i | \mathbf{x})$ The softmax function can then exponentiate and normalize \mathbf{z} to obtain the desired $\hat{\mathbf{y}}$. The softmax function is defined as follows: [48]

$$\text{softmax}(\mathbf{z})_i = \frac{\exp(z_i)}{\sum_j \exp(z_j)}$$

When training the softmax to produce a target value y using maximum log-likelihood, the exp function works very well, just as it does with the logistic sigmoid. We want to maximize $\log P(y = i; \mathbf{z}) = \log \text{softmax}(\mathbf{z})_i$ in this scenario. Defining the softmax in terms of exp is natural because the log in the log-likelihood can undo the exp of the softmax: [48]

$$\log \text{softmax}(\mathbf{z})_i = z_i - \log \sum_j \exp(z_j)$$

The first term of the equation indicates that the input z_i contributes directly to the cost function at all times. We know that learning can continue even if z_i 's contribution to the second part of the equation becomes very little because this term cannot saturate. The first phrase urges z_i to be pushed up, whereas the second term encourages all of \mathbf{z} to be pushed down when maximizing log probability.

2.7.2 Regularization for Deep Learning

One of the most difficult problems in machine learning is creating an algorithm that performs well not only on training data but also on new inputs. Many machine learning algorithms are expressly designed to minimize test error, which may come at the expense of higher training error. The term "regularization" refers to all of these procedures. "Any alteration we make to a learning algorithm that is meant to lower its generalization error but not its training error," according to regularization.

There are numerous ways for regularization. Some people apply extra constraints to a machine learning model, such as limiting parameter values. Ensemble methods [49] are; a type of regularization, use numerous hypotheses to explain the training data.

Dataset Augmentation

The greatest method to improve the generalization of a machine learning model is to train it on additional data [50]. Of course, the amount of data we have is restricted in practice. Making bogus data and adding it to the training set is one technique to get around this problem.

This method is the most straightforward for classification. A classifier must be able to summarize a complex, high-dimensional input x into a single category identity y . This suggests that a classifier's primary goal is to be invariant to a wide range of modifications. Simply by altering the x inputs in our training set, we can simply produce new (x, y) pairs [51].

For a specific classification task, object recognition, dataset augmentation has proven to be a very effective strategy [40]. Images are three-dimensional and include a large number of variables, many of which may be easily simulated. Even if the model has already been constructed to be somewhat translation invariant by applying the convolution and pooling techniques mentioned in CNN below, operations like translating the training images a few pixels in each direction can typically considerably improve generalization. Many other actions, such as rotating or resizing the image, have also shown to be quite useful.

It's important to avoid using transformations that modify the right class. Optical character recognition tasks, for example, necessitate distinguishing between the letters 'b' and 'd,' as well as the letters '6' and '9,' therefore horizontal flips and 180-degree rotations are not viable techniques of supplementing datasets for these tasks [52].

Early stopping

When we train large models with enough representational capacity to overfit the task, we frequently notice that training error drops over time, but validation set error tends to climb again [53].

This means that by returning to the parameter setting at the moment in time with the lowest validation set error, we can get a model with improved validation set error (and, ideally, better test set error). We save a copy of the model parameters every time the error on the validation set improves. We return these parameters rather than the most recent parameters when the training algorithm finishes. When no parameters have improved over the best recorded validation error after a predetermined number of repetitions, the algorithm quits.

Drop out

Dropout provides a computationally inexpensive but powerful method of regularizing a broad family of models [54]. Dropout is a new regularizer that has been proposed to reduce over-fitting [55].

It is a regularization method in which the activations of hidden units are stochastically adjusted to zero for each training case during training time. Because the dropped-out units can't impact other retained units, co-adaptations of feature detectors are broken apart. Another way to think about dropout is that it produces a particularly effective type of model averaging in which the number of trained models is exponentially more than the number of units, and all of the models have the same parameters. Using max-pooling dropout and fully-connected dropout, better results can be achieved [56].

2.7.3 Optimization for Training Deep Models

Optimization has been a critical component of neural network research for a long time [57]. The process of optimization can be broken down into three stages. The initial step is to start the algorithm and let it run until it finds a reasonable solution, such as a stationary point. The algorithm's second stage is to have it converge as quickly as possible. The algorithm must converge to a solution with a low objective value in the third stage (e.g., global minima).

2.7.3.1 Algorithms with Adaptive Learning Rates

Optimization algorithms are the foundation on which a machine can learn from its mistakes. They calculate gradients and try to reduce the loss function to the smallest possible value. Learning can be implemented in a variety of ways using various optimization techniques.

AdaGrad

The AdaGrad method adjusts the learning rates of all model parameters separately by scaling them inversely proportionate to the sum of all of their historical squared values. It is a way for determining the learning rate based on the circumstances [58]. Because the actual rate is set by factors, learning rates are adaptable. Parameters with a large gradient have a lower learning rate, while those with tiny gradients have a higher learning rate.

The AdaGrad algorithm has certain desired theoretical aspects in the context of convex optimization. However, it has been demonstrated empirically that the buildup of squared gradients from the start of training can result in a premature and disproportionate fall in the effective learning

rate for deep neural network models. AdaGrad works well with some deep learning models, but not all [40].

RMSProp

The AdaGrad is changed by RMSProp in the manner the gradient is accumulated. Gradients are added together to create an exponentially weighted average. RMSProp discards the history and keeps only the most recent gradient data [59].

Adam

Its name comes from the phrase "adaptive moments." It's a cross between RMSProp and momentum. Only the smooth form of the gradient is considered in the update operation, which also contains a bias correction technique [60].

2.7.3.2 Optimization Strategies and Meta-Algorithms

Many optimization techniques are more like general templates that can be specialized to provide algorithms or subroutines that can be incorporated into a variety of algorithms [40].

Batch Normalization

The composition of numerous functions or layers is required for very deep models. Under the premise that the other layers do not change, the gradient tells you how to update each parameter. In practice, we update all layers at the same time. When we make the update, unexpected consequences can occur when multiple functions that are linked together are altered at the same time, utilizing updates that were calculated assuming that the other functions would remain constant.

Batch normalization is a simple and elegant approach to re-parametrize nearly any deep network. The problem of coordinating updates across several layers is greatly reduced thanks to the reparameterization. Batch normalization can be applied to any input or hidden layer in a network [61].

2.7.3 Convolutional Networks

Convolutional networks, commonly known as convolutional neural networks or CNNs, are a type of neural network that is used to process data with a grid-like architecture [40]. The name "convolutional neural network" refers to the network's use of the convolutional mathematical procedure. Convolution is a type of linear operation that is specialized. Convolutional networks

are simple neural networks with at least one layer that uses convolution instead of ordinary matrix multiplication.

The first argument to the convolution is generally referred to as the input, while the second argument is often referred to as the kernel in convolutional network nomenclature. The feature map is a term used to describe the outcome.

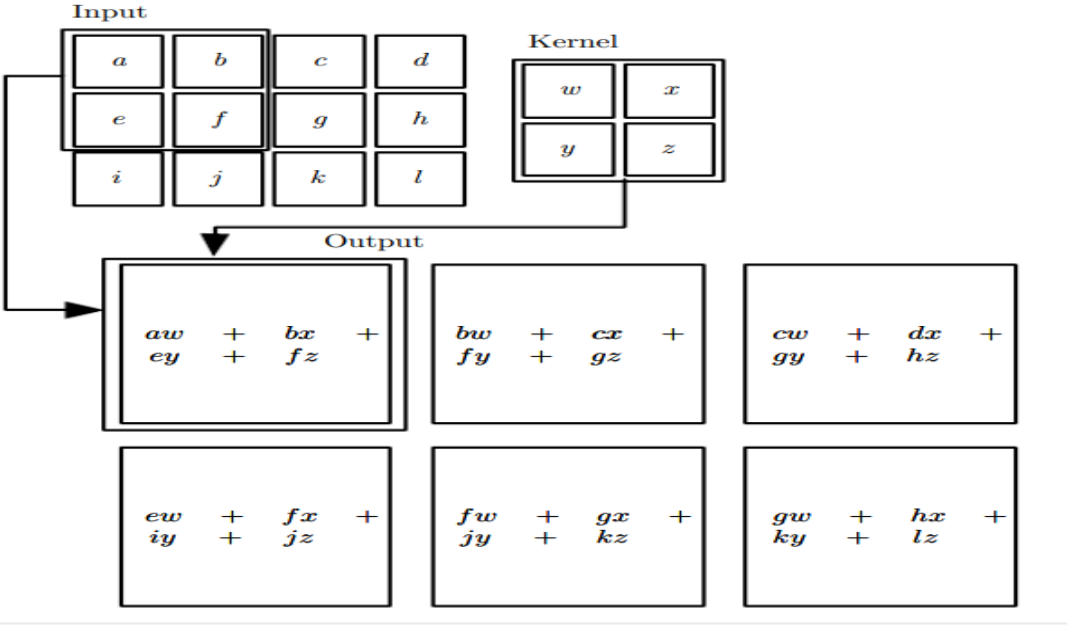


Figure 2. 6 An example of 2-D convolution without kernel-flipping [40]

Sparse interactions, parameter sharing, and equivariant representations are three fundamental notions that might help improve a machine learning system. Furthermore, convolution allows to work with inputs of varying sizes [40].

Sparse interactions (also known as sparse connectivity or sparse weights) are common in convolutional networks. Making the kernel smaller than the input does this. When processing a picture, for example, the input image may comprise thousands or millions of pixels, but we can detect small, important features like edges using kernels that are only tens or hundreds of pixels in size [62]. This means we may keep fewer parameters, which decreases the model's memory requirements while also increasing its statistical efficiency.

The use of the same parameter for many functions in a model is referred to as parameter sharing. When computing the output of a layer in a typical neural net, each member of the weight matrix is

used exactly once. It is multiplied by one of the input elements and then never looked at again. In a convolutional neural net, each member of the kernel is used at every position of the input.

The particular form of parameter sharing used in convolution gives the layer a trait termed equivariance to translation. As a function is said to be equivariant, it signifies that when the input changes, the output changes as well.

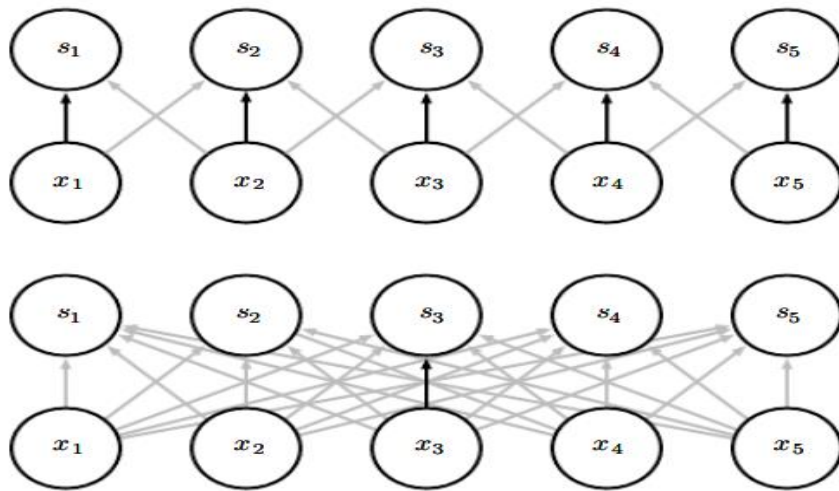


Figure 2. 7 Parameter sharing

As shown in Figure 2.6 explanation: Bold arrows represent connections between two models that use the same parameter. *(Top)* The bold arrows indicate uses of the central element of a 3-element kernel in a convolutional model. Due to parameter sharing, this single parameter is used at all input locations. *(Bottom)* In a fully connected model, the single bold arrow shows the use of the core element of the weight matrix. Because there is no parameter sharing in this paradigm, the parameter is only utilized once.

The particular form of parameter sharing used in convolution gives the layer a trait termed equivariance to translation. As a function is said to be equivariant, it signifies that when the input changes, the output changes as well. specifically, a function $f(x)$ is equivariant to a function g if $f(g(x)) = g(f(x))$ [40]. In the case of convolution, if we let g be any function that translates the input, i.e., shifts it, then the convolution function is equivariant to g [40].

A CNN is made up of several convolution and sub-sampling layers, as well as a fully linked layer and a normalization layer. Moving from input to output layers, the series of successive convolution layers performs progressively more sophisticated feature extraction at each layer. Following the convolution layers are fully connected layers that do categorization. Between each convolution layer, sub-sampling or pooling layers are frequently used. A 2D $n \times n$ pixelated image is fed into a CNN. Filters or kernels are groupings of 2D neurons that make up each layer.

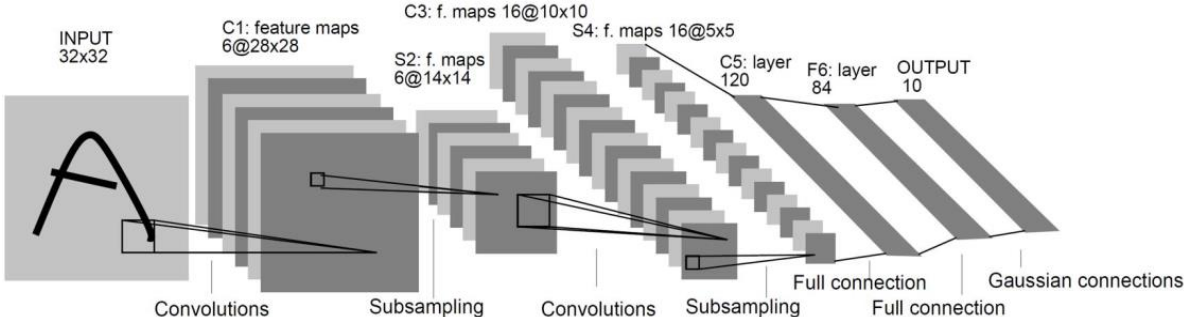


Figure 2. 8: 7-layer Architecture of CNN [63]

Convolutional Neural Network (CNN), also known as ConvNet, is a type of Artificial Neural Network (ANN) with a deep feed-forward architecture and amazing generalizing ability when compared to other networks with FC layers [64]. It can learn highly abstracted features of objects, especially spatial data, and can identify them more efficiently than other networks with FC layers.

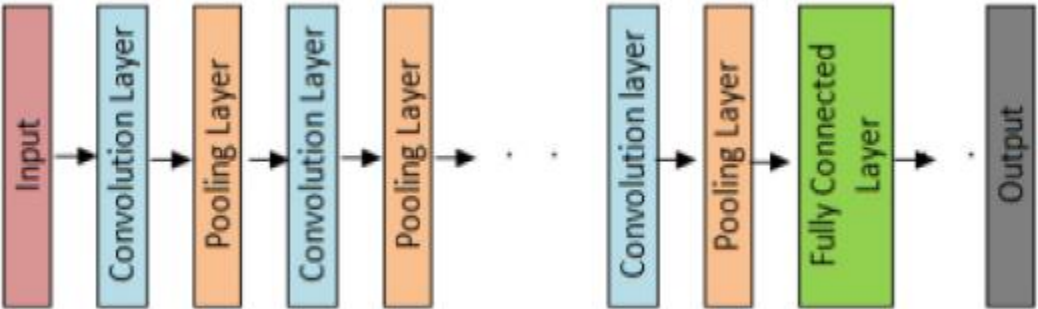


Figure 2. 9 Conceptual model of CNN [65]

Backpropagation is almost always utilized to train all parameters (weights and biases) in CNN. Here's a quick rundown of the algorithm. The cost function with respect to individual training example (x, y) in hidden layers can be defined as [40]:

$$J(W, b; x, y) = \frac{1}{2} \|h_{w,b}(x) - y\|^2$$

The equation for error term δ for layer l is given by:

$$\delta^{(l)} = \left((W^{(l)})^T \delta^{(l+1)} \right) \cdot f'(z^{(l)})$$

Where $\delta^{(l+1)}$ is the error for $(l + 1)$ th layer of a network whose cost function is $J(W, b; x, y)$. $f'(z^{(l)})$ represents the derivate of the activation function

$$\nabla_{w^{(l)}} J(W, b; x, y) = \delta^{(l+1)} (a^{(l+1)})^T$$

$$\nabla_{b^{(l)}} J(W, b; x, y) = \delta^{(l+1)}$$

Where a is the input, such that $a^{(1)}$ is the input for 1st layer (i.e., the actual input image) and $a^{(l)}$ is the input for $l - th$ layer.

Error for sub-sampling layer is calculated as:

$$\delta_k^{(l)} = \text{upsample} \left((W_k^{(l)})^T \delta_k^{(l+1)} \right) \cdot f'(z_k^{(l)})$$

Where k is the layer's filter number. The error must be cascaded in the opposite manner at the subsampling layer, for example, where mean pooling is utilized, up sample equally distributes the error to the preceding input unit. Finally, here is the gradient in relation to the feature map.

$$\nabla_{w_k^{(l)}} J(W, b; x, y) = \sum_{i=1}^m (a_i^{(l)}) * \text{rot } 90(\delta_k^{(l+1)}, 2)$$

$$\nabla_{b_k^{(l)}} J(W, b; x, y) = \sum_{a,b} (\delta_k^{(l+1)})_{a,b}$$

Where $(a_i^{(l)}) * \delta_k^{(l+1)}$ represents the convolution between error and the $i - th$ input in the $l - th$ layer with respect to the $k - th$ filter.

The flow of the backpropagation algorithm as utilized in a CNN is depicted in the Algorithm below, which proceeds through numerous epochs until either the maximum iterations are achieved or the cost function target is fulfilled.

```

1: Initialization weights to randomly generated value (small)
2: Set learning rate to a small value (positive)
3: Iteration  $n = 1$ ; Begin
4: for  $n$ 
  < max iteration OR Cost function criteria met, do
5: for image  $x_1$  to  $x_i$ , do
6:   a. Forward propagate through convolution, pooling and
7:   b. Derive Cost Function value for the image
8:   c. Calculate error term  $\delta^{(l)}$  with respect to weights for  $e$ 
9:     Note that the error gets propagated from layer to layer
10:    i. fully connected layer
11:    ii. pooling layer
12:    iii. convolution layer
13:   d. Calculate gradient  $\nabla_{w_k^{(l)}}$  and  $\nabla_{b_k^{(l)}}$  for weights  $w_k^{(l)}$  and
14:     Gradient calculated in the following sequence
15:    i. convolution layer
16:    ii. pooling layer
17:    iii. fully connected layer
18:   e. Update weights
19:     $w_{ji}^{(l)} \leftarrow w_{ji}^{(l)} + \Delta w_{ji}^{(l)}$ 
20:   f. Update bias
21:     $b_i^{(l)} \leftarrow b_i^{(l)} + \Delta b_i^{(l)}$ 

```

Figure 2. 10 CNN backpropagation algorithm pseudo code [40]

2.7.3.1 Convolutional Layer

The most fundamental component of any CNN architecture is the convolutional layer. It is made up of a set of convolutional kernels (also known as filters) that are convolved with the input picture (N-dimensional metrics) to produce an output feature map.

What is a kernel? A kernel can be thought of as a grid of discrete values or integers, with each value representing the kernel's weight. All of the weights of a kernel are assigned random numbers at the start of the CNN model's training process (different approaches are also available there for initializing the weights). The weights are then fine-tuned with each training epoch, and the kernel learns to extract significant information.

0	1
-1	2

Figure 2. 11 Example of a 2×2 kernel

What is a Convolution Operation, and how does it work? Before we proceed any further, it's important to understand CNN's input format. Unlike other traditional neural networks (which use a vector input), CNN takes a multi-channeled image (e.g., RGB images are 3 channels, while Gray-Scale images are single channels). Now, if we take a grayscale image with a 4×4 dimension and a 2×2 kernel with randomly initialized weights, we can understand the convolution procedure.

1	0	-2	1
-1	0	1	2
0	2	1	0
1	0	0	1

0	1
-1	2

Figure 2. 12 A 4×4 Gray-Scale image

Figure 2. 13 A kernel of size 2×2

Now, in the convolution operation, we take the 2×2 kernel and slide it horizontally and vertically over the entire 4×4 picture, taking the dot product between kernel and input image by multiplying their respective values and summing all values to obtain one scalar value in the output feature map [66].

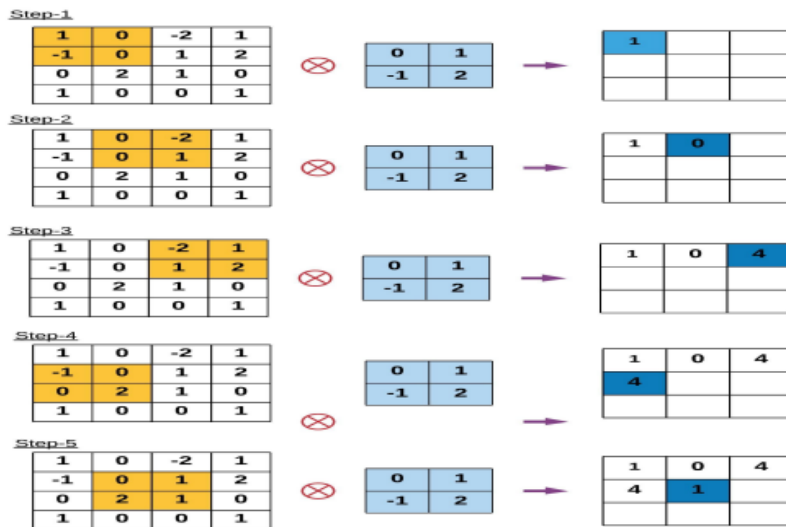


Figure 2. 14 Illustrating the first 5 steps of convolution operation [67]

After performing the complete convolution operation, the final output feature map is:

1	0	4
4	1	1
1	1	2

Figure 2. 15 The final feature map after the complete convolution operation [67]

In the preceding example, one can apply the convolution operation to the input picture with no padding and a stride (i.e., the size of the taken step along the horizontal or vertical position) of 1 to the kernel. In a convolution process, however, it is possible to choose a different stride value (other than 1) than 1. The fact that increasing the stride of the convolution process resulted in a lower-dimensional feature map is obvious.

2.7.3.2 Pooling Layer

The pooling layers are used to sub-sample the feature maps (created after convolution operations), i.e., it compresses the bigger feature maps to smaller feature maps. While shrinking the feature maps, the most prominent features (or information) in each pool stage are always preserved. Similar to convolution, the pooling process is conducted by defining the pooled region size and the stride of the operation. In different pooling layers, several types of pooling techniques are utilized, such as max pooling, min pooling, average pooling, gated pooling, tree pooling, and so on. The most common and often used method is max pooling [64].

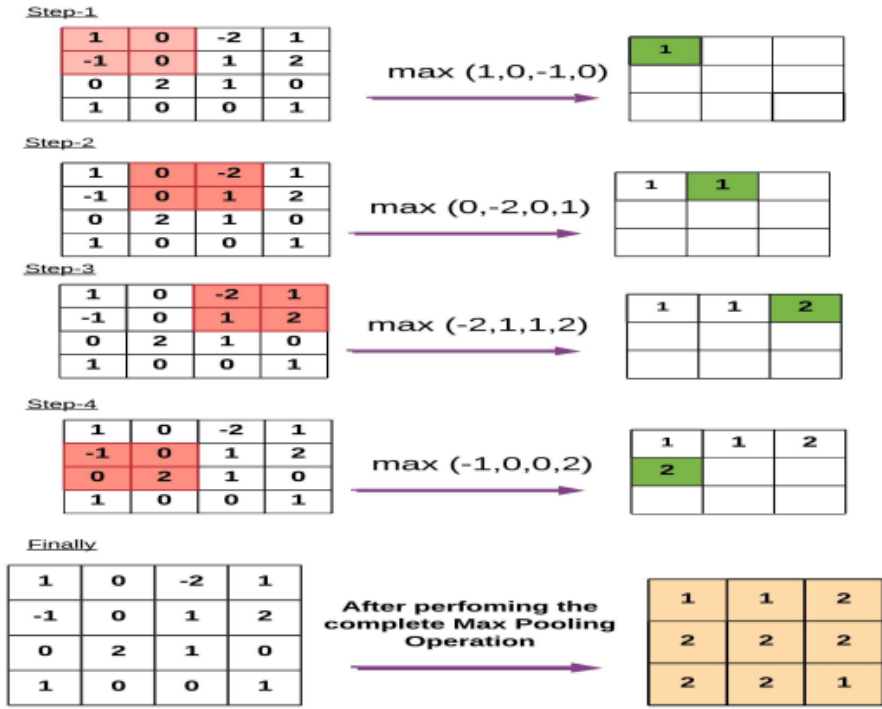


Figure 2. 16 max-pooling operation [40]

2.7.3.3 Fully Connected (FC) Layer

The final component (or layers) of the CNN architecture (used for classification) is usually made up of fully-connected layers, in which each neuron in one layer is coupled to every neuron in the preceding layer. The output layer (classifier) of the CNN architecture is the last layer of Fully-Connected layers [68].

The FC layers receive input from the last convolutional or pooling layer in the form of a set of metrics (feature maps), which are flattened to create a vector, which is then fed into the FC layer to construct CNN's final output [68].

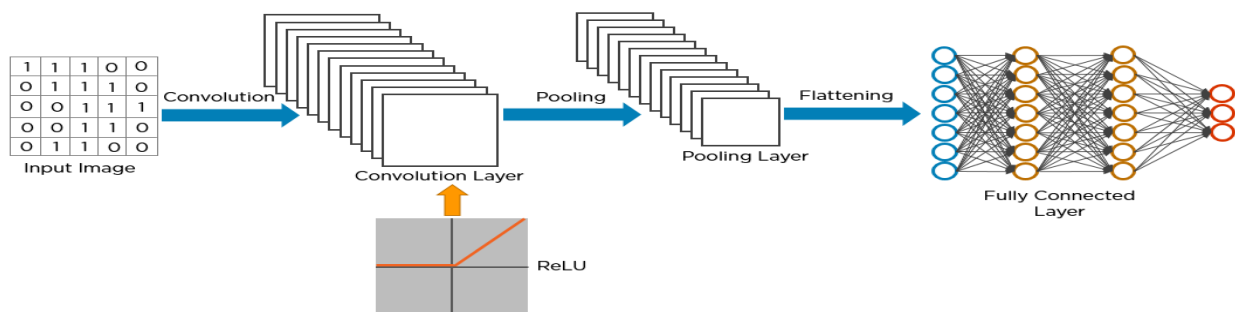


Figure 2. 17 CNN architecture [69]

The following processes are included in the CNN training process: data pre-processing and augmentation, parameter initialization, CNN regularization, and optimizer selection.

2.7.3.4 Image Classification

We assume that the input image comprises a single item in image classification, and then we use CNN models to classify the image into one of the pre-selected target classes. The following are a few of the most popular CNN architectures (models) for image classification [40]:

LeNet-5

The LeNet-5 was one of the first CNN architectures, and it was created to classify handwritten numbers [70]. LeCun et al. first proposed it in 1998. Three convolutional layers and two FC layers make up the LeNet-5's five weighted (trainable) layers. The first two convolution layers are followed by a max-pooling layer (to sub-sample the feature maps), and the last two convolution layers are followed by two fully linked layers. The classifier is the final layer of those completely connected layers, and it can categorize up to ten digits.

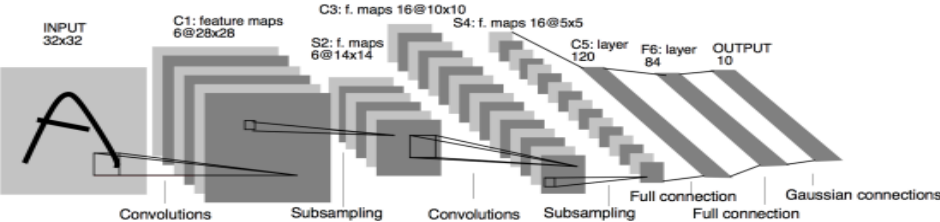


Figure 2. 18 The Architecture of LeNet [63]

AlexNet

Inspired from LeNet, Krizhevky et al. designed first large-scale CNN model, called AlexNet [71]. This is used to categorize ImageNet data. It is made up of eight weighted (learnable) layers, the first five of which are convolutional layers and the final three of which are fully linked layers. The last output layer, which was created for ImageNet data, uses 1,000 units to categorize the input photos into one of the thousand classes in the ImageNet dataset.

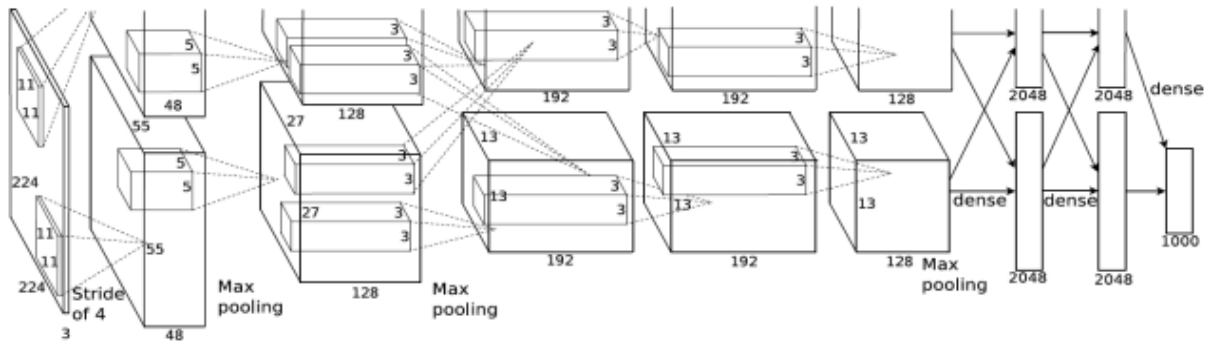


Figure 2. 19 The Architecture of AlexNet [69]

VGGNet

VGGNet [72], created by Simonyan and Zisserman in 2014, is one of the most prominent CNN architectures. The authors provided six distinct CNN configurations, the most successful of which are VGGNet-16 (configuration D) and VGGNet-19 (configuration E).



Figure 2. 20 The Architecture of VGGNet [73]

GoogLeNet

The GoogLeNet [74] architecture differs from all of the prior traditional CNN models in that it employs network branching rather than a single line sequential architecture. Szegedy et al. suggested the GoogLeNet in 2014. The GoogLeNet contains 22 weighted (learnable) layers, with the "Inception Module" serving as the network's basic building component. This module's processing occurs in parallel in the network, with each (basic) module processing 1x1, 3x3, and 5x5 filtered convolution layers in parallel before combining their output feature maps, resulting in very high-dimensional feature output.

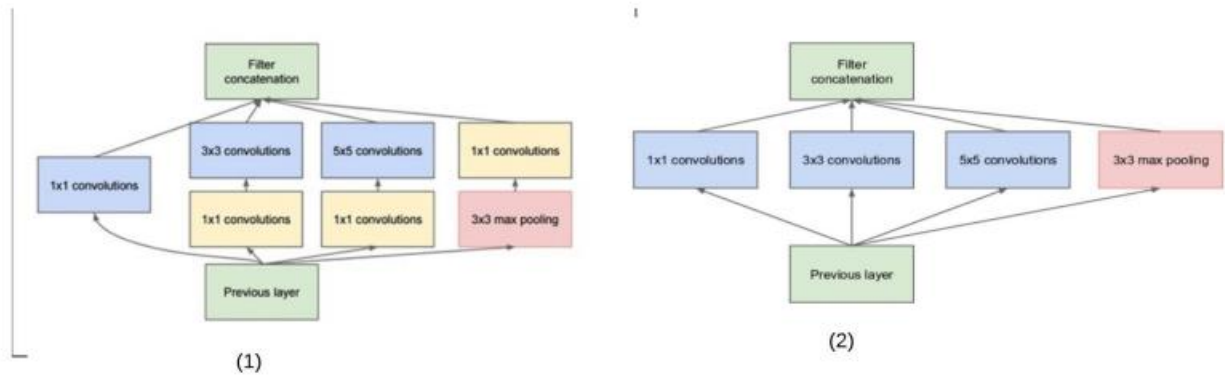


Figure 2. 21 (1) Simple Inception Module, (2) Inception Module with dimensionality reduction [69]

ResNet

Since a deep CNN model suffers from vanishing gradient problems, Microsoft proposed the ResNet model, which introduced the concept of "identity skip connection" to tackle the vanishing gradient problem [75]. Instead of learning a direct mapping ($H(x) = F(x)$), the ResNet's architecture uses residual mapping ($H(x) = F(x) + x$), and these blocks are known as residual blocks. Many residual blocks with three convolution layers make up the ResNet architecture.

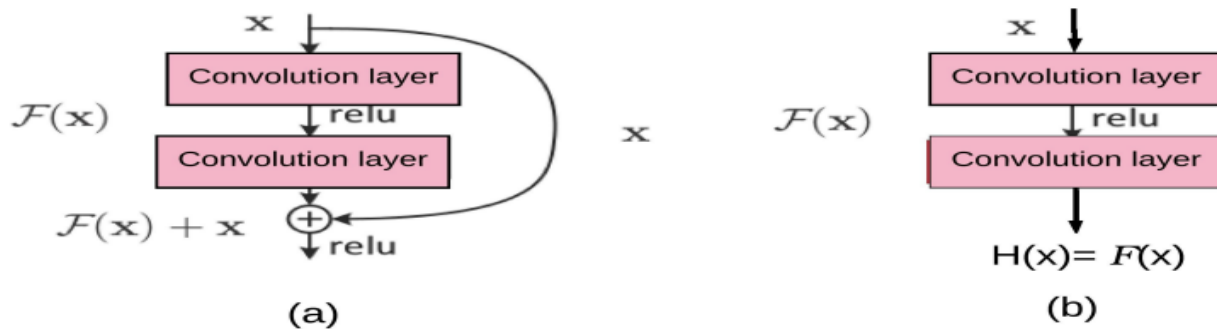


Figure 2. 22 (a) Mapping inside Residual block, (b) Simple direct mappings [63]

DenseNet

DenseNet [76] extends the concept of residual mapping by propagating the output of each block to all subsequent blocks inside each dense block in the network. It increases feature propagation ability and solves the vanishing gradient problem by propagating information in both forward and backward directions during model training.

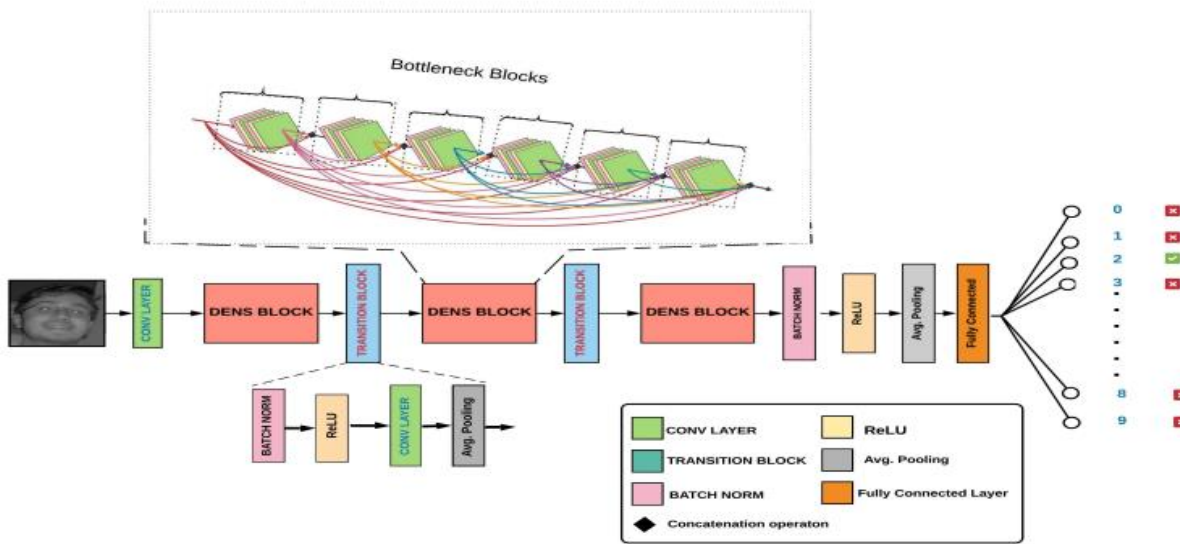


Figure 2. 23 A DenseNet based model with three dense blocks [69]

2.8 Related works

Banana is a plant which is very closely related to Enset, and because of that Enset is also called the false banana. Both Enset and banana can be infected by bacterial wilt. Banana Plant Disease Detection and Grading Using image processing has been developed by researchers Basavaraj Tigadi and Bhavana Sharma [77]. Banana leaf is highly exposed to diseases like Black Sigatoka, Yellow Sigatoka, Bunchy top, Panama Wilt, Streak virus. the proposed work uses Artificial Neural Network to classify the Banana plant diseases which are mentioned above. The proposed system involves several steps, which include- dataset creation, image pre-processing, HOT feature extraction and artificial neural network-based training and classification. Two kinds of features are extracted from the images of the leaves: Histogram of Template (HOT) features and Color features. To measure the severity of the disease, grading is assigned based on the affected pixel area.

Study on Banana Leaf Disease Identification Using Image Processing Methods has been accomplished by researchers D. Surya Prabh and Dr. J. Satheesh Kumar [78]. Major banana diseases that express the symptoms on leaves are panama disease, moko disease, sigatoka disease, black spot, banana bunchy top, infectious chlorosis, banana streak virus and banana bract mosaic virus disease, the model identifies this major disease of banana. The researchers used image processing techniques to prepare the image for training. The researchers extracted color feature and used the color feature for training. The researchers used thresholding, histogram and region

growing methods for segmentation. The researchers used SVM for classifying infected and healthy plants.

AI-powered banana diseases and pest detection has been accomplished by the researchers: Gomez Selvaraj, Alejandro Vergara, Henry Ruiz, Nancy Safari, Sivalingam Elayabalan, Walter Ocimati and Guy Blomme [79]. Large datasets (30,952 images) of expert-prescreened banana disease and pest symptom/damage images were collected from hotspots across Africa and Southern India. The researchers used a transfer learning strategy to retrain three different convolutional neural network (CNN) architectures in order to develop a detection model. Using images taken from various areas of the banana plant, six different models were built from 18 different classes (disease by plant parts). In majority of the models examined, banana disease and pest identification, was accomplished with an accuracy of more than 90%.

Yidnekachew K. [80] used a deep learning approach to detect and classify bacterial wilt disease of Enset. The number of original images collected from Enset farm was 4896. The researcher used CNN to develop the model. For the detection of BWE one feature parameter is used which is a color feature, this feature is considered because their visual color difference identifies whether the crop is infected by the disease or not by a human vision in the traditional system. The model has a total of 8 layers, five convolutions, and three dense layers. The proposed model is trained with a total of 111, 060 images after data augmentation during the training is applied in the dataset. The researcher analyzes that the mean percentage training accuracy of VGG16, InceptionV3, and the proposed CNN model is 96.6, 91.7, and 98.49 respectively. The mean percentage of validation accuracy for VGG16, InceptionV3, and proposed models is 96.8, 91.5, 98.48 respectively. By testing the models with unseen data, the researcher has obtained the test accuracy of VGG16 is 92.4%, InceptionV3 is 90.4%, and test accuracy of the proposed model is 97.86%.

Enset Diseases Diagnosis Model Using Image Processing and Machine Learning Techniques has been developed by researchers Kibru and Getahun [81]. The researcher selected two *Enset* leaf diseases viz. Bacterial Wilt and Fusarium Wilt disease and collected 430 *Enset* leaf images from Areka agricultural research center and some selected areas in SNNPR. Color features are extracted using color moments and HSV color space. In order to extract a texture feature of the *Enset* leaf image Gabor texture feature extraction technique is used. A total of three different morphological features that is regional descriptors of minor axis, major axis and circularity of the *Enset* image

pixels were measured and used as a shape feature. In order to classify Enset disease a multiclass support vector machine was used. The model has three classes bacterial wilt, fusarium wilt and healthy Enset. From all those diseases, a total of 460 Enset leaf images are collected from which 368 is used for training and 92 images are used for testing. The proposed model is experimented with four different kernels, and the overall result indicates that the RBF Kernel achieved the highest accuracy of 94.04% and 92.44% for bacterial wilt and fusarium wilt respectively.

Table 2. 3 Summary of related works

Author	Title	Methodology	Accuracy	Remark
Yidnekachew K.	Developing bacterialwilt detection model on Enset crop using a deep learning approach	The researcher used CNN algorithm for feature extraction and classification	97.86%	<ul style="list-style-type: none"> • Suitable for very large dataset • Modern deep learning approach was used • Color feature was used for classification
Kibru A. and Getahun T.	<i>Enset</i> Diseases Diagnosis Model Using Image Processing and Machine Learning Techniques	<ul style="list-style-type: none"> • Gabor texture feature extraction was used. • Multiclass Support vector machine was used for classification 	94.04% for BWE 92.44% for fusarium wilt	<ul style="list-style-type: none"> • The dataset was collected from agricultural research center • Image processing techniques were used to the images for training • The dataset collected is very small for deep learning
Basavaraj T. and Bhavana S.	Banana Plant Disease Detection and Grading Using Image processing	uses ANN to classify the Banana plant diseases	Not specified	<ul style="list-style-type: none"> • Histogram of Template (HOT) features and Color features were extracted. • Grading is assigned to measure the

				severity of the disease.
D. Surya Prabh and Dr. J. Satheesh Kumar	Banana Leaf Disease Identification Using Image Processing Methods	<ul style="list-style-type: none"> Used image processing techniques for image analysis used SVM for classification 	Not specified	Did not use CNN, which more effective in plant disease identification
G. Selvaraj, A. Vergara, H. Ruiz, N. Safari, S. Elayabalan, W. Ocimati and G. Blomme	AI-powered banana diseases and pest detection	<ul style="list-style-type: none"> used DCNN for classification 	ResNet50:97.36% InceptionV2: 97.3% MobileNetV1:93.64%	<ul style="list-style-type: none"> The model is able to detect the corm, leaves, pseudo stem, fruit diseases. Examined only three state-of-the-art models

2.9 Research gap

Computer vision has been used in the field of plant disease detection and has shown great improvement in modernizing field of agriculture. Machine learning algorithms like SVM, NN and CNN have been used in banana and Enset pant disease identification. The bacterial wilt and mealybug diseases can attack both Enset and banana plants. Many computer-vision researches have been conducted to identify banana diseases, but in the case of Enset, one research have been conducted to identify bacterial wilt disease of Enset using deep learning approach. But identifying mealybug of Enset plant have not been conducted yet. In this thesis, Enset mealybug detection is conducted together with bacterial wilt detection.

CHAPTER THREE

RESEARCH DESIGN AND METHODS

This chapter goes through the details of the suggested model and methods used for detecting and classifying Enset diseases. It takes a succession of stages, beginning with image preprocessing, ROI segmentation, feature extraction and learning, and ending with classification into specified classes. Deep learning is a branch of machine learning that is roughly modeled after the neural networks of the human brain, in which the computer learns which features are useful automatically. Deep learning has been successfully applied in a variety of fields, and it has lately entered the field of agriculture. The Convolution Neural Network is the foundation of our system. Experimental research approach is employed in this thesis, where one set of variables is kept constant while the other set of variables is measured as the subject of the experiment. Various tests are carried out with various dataset ratios. In addition, a variety of studies involving various activation functions and hyperparameters have been carried out.

3.1 Methodology

The procedural sequences of performing a study are known as methodology. A study's step-by-step procedures for performing a study are known as procedures. Methodological processes must be followed in order to make the research scientific, that is, unbiased and consistent, to perform studies objectively, to have a standard creative work, and to have a study that meets the standard.

3.1.1 Research Design

The experimental research design is used in this study. It consists of a hypothesis, a researcher-controllable variable, and variables that may be measured, calculated, and compared. Above all, experimental research is conducted in a controlled setting [12].

In this study, experimental method is used for data preparation, model building, training the model and testing the proposed model. To facilitate the experiment, image processing steps such as: image resizing and image segmentation are accomplished after acquiring images. Then designing the model, training the model and at last testing the model are the next phases.

3.2 Implementation tools

Experiments are based on a Google Colab and Anaconda prototype built with Keras (TensorFlow as a backend). Google Colab offers a free Intel(R) Xeon(R) CPU @ 2.30GHz with 2 CPUs, 8 GB

of RAM, 78 GB of storage, and 15 GB of GPU, so why not take advantage of it. With a batch size of 16, the model is trained for 100 epochs.

This days, Artificial neural network, Convolutional neural network and support vector machine are the best supervised machine learning algorithms [16]. For this thesis, we used CNN because many research papers suggest using CNN for computer vision tasks is the best choice [17] [18] [19] [20]. CNN is able to generalize among the images that are adaptable so that it can easily learn to recognize new images. neural networks have favorable properties that make them an excellent choice for object classification. The most important of these properties are generalization, expandability, representing multiple samples, and memory saving [21]. The CNN model is implemented by using Keras (TensorFlow as backend) on google Collaboratory(colab) platform. Python is used for writing the required source codes. Colab has a number of advantages over local processing, including faster GPUs, more memory, and longer runtimes. Python is simple to read, learn, and write. It has increased productivity and supports a vast group of libraries.

In order to detect and classify, classification entails two main processes.

1. Training phase: Convolution, activation, pooling, fully-connected, and dropout layers make up the training phase. The original resized and segmented image is fed into the convolution layer. A set of K learnable filters is used, each of which has a square width and height. The pooling layer is then used to lower the input volume's spatial size, reducing the number of parameters and the amount of processing in the network. Before using the Softmax classifier, fully connected layers are inserted at the end of the network. Dropout is also used to avoid overfitting by changing the network design during training.
2. Testing phase: the trained model has been given images that are not from the training and validation datasets to see how well it responds to new datasets.

3.3 Proposed Model Architecture

Preprocessing, segmentation, and classification are the three phases of the proposed system. We normalize the images to a standard size (256*256) during image preparation. The extraction of an area of interest (ROI) is done via image segmentation. We employ a convolutional neural network for classification. Training, validation, and testing are the three main processes of model

development. Finally, for grading into a certain class (Healthy, Bacterial wilt, and Mealybug), a Softmax activation was employed. Figure 4.1 depicts the proposed system architecture.

CNN uses convolution layers to extract feature maps from the images using various number of kernels. Each image's important features are retrieved and subsequently used for categorization within the CNN stacked layers. Different layers (mostly convolution, activation, and pooling) are built on top of each other in the training phase to learn the most typical or distinguishing features. After the features have been learned, the Softmax classifier is used to classify the data. After each convolution, several approaches such as batch normalizing and dropout (with decreased value) were employed at an early stage, followed by an activation layer. Using the validation dataset, the model's performance is measured during training. After reviewing the model's performance, the model that performed the best is saved and utilized as a predictive model.

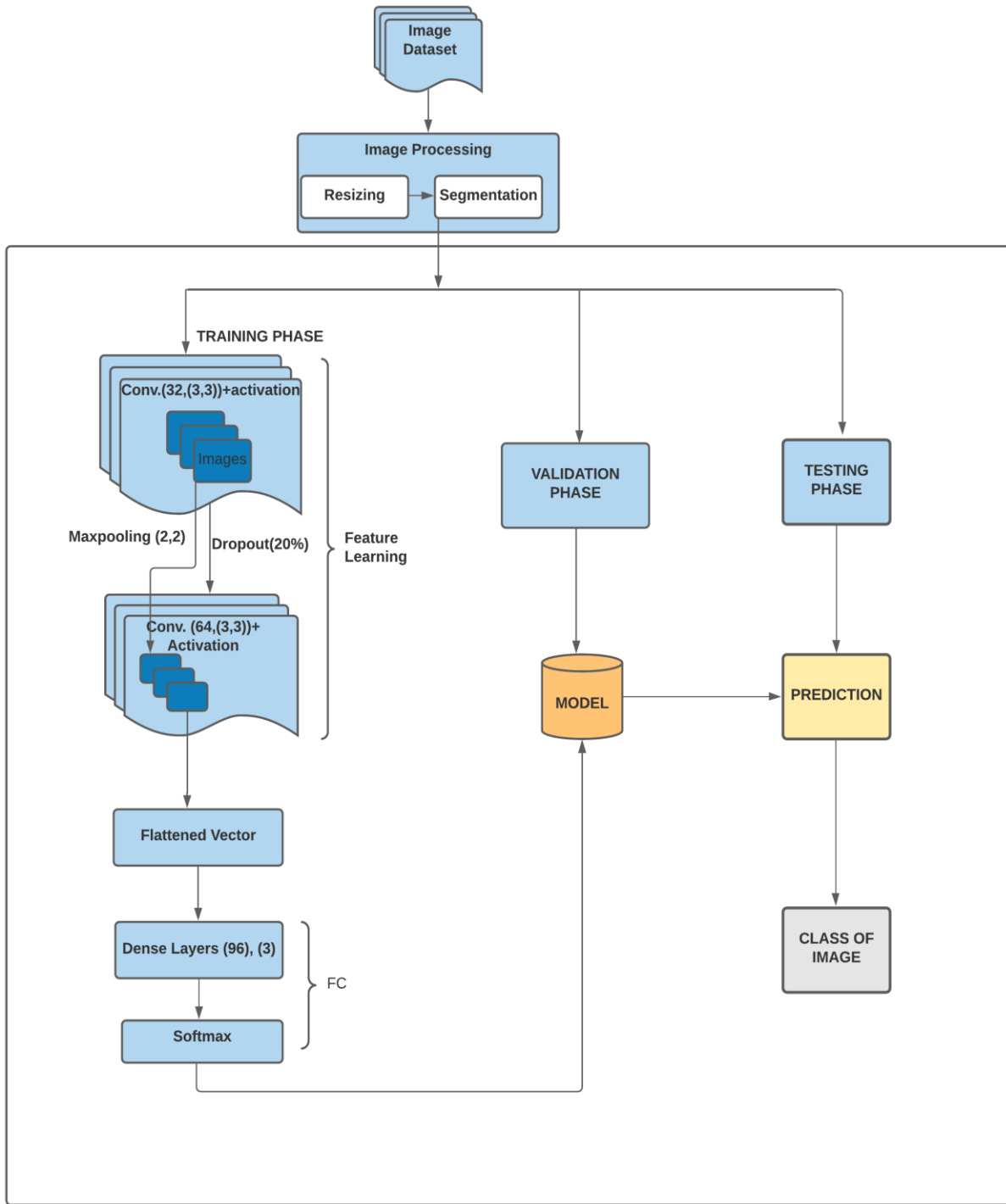


Figure 3. 1 Proposed Model Architecture

3.4 Image Data Preparation

3.4.1 Image Acquisition

Image data was collected from Wolkite university incubation center, Gurage zone ‘Cheha’ and ‘Eza’ district local Enset farms. Domain experts are involved during the collection, experimentation and evaluation of this proposed model.

Image acquisition process is the first work of image processing technique. The collected sample images of Enset for this study were taken with a Sonny VIXIA HF M50 HD camera, which features a 24 MP camera with a default resolution of 4248x5664 pixels, giving a total of (24,060,672 pixels or 24 MP). A high-quality image resolution was used to detect the details of each image. Moreover, the selection of background object should contrast the region of interest so, white background is used for healthy and bacterial wilt classes. Mealybugs are found at the root of the plant, so it impossible to capture them in a controlled environment, that is why we do segmentation to extract the region of interest from the images. We have three classes in our model, they are healthy, bacterial wilt and mealybug. Three hundred twenty-eight (328) for each healthy and mealybug classes, and 331 images are collected for bacterial wilt class, nine hundred eighty-seven (987) images are collected in total. It is divided using 80/20 data split for training and testing respectively. Allocating 80% of the dataset for training is close to optimal for reasonable sized datasets (greater than 100 images) [13]. Since the amount of data collected is limited, data augmentation is applied during training to artificially increase the amount of data.

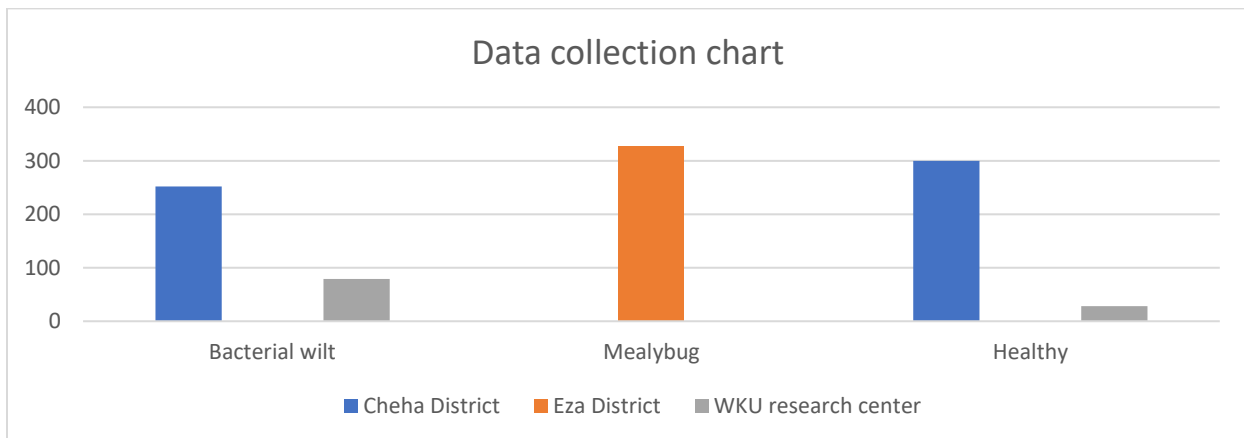


Figure 3. 2 Data Collection Chart

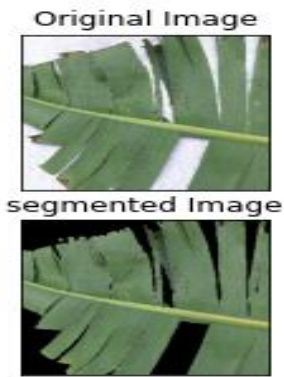
3.4.2 Image Data Preprocessing

Image preprocessing refers to the practice of upgrading data images in order to turn them into a usable format prior to computer processing. The dataset collected has a dimension of 5664X4248 pixels, so it takes too much time and resource to process. As a result, for efficient processing, we have scaled down the images to a uniform 256*256 dimension. OpenCV was used for image resizing and conversion into Numpy array. Multiple images in the same folder were sized to the appropriate dimension simultaneously by importing the opencv and glob libraries using the command `image=cv2.resize(img, (256,256))`.

3.4.3 Segmentation

Image segmentation is the technique of dividing or grouping a picture into various portions. Image segmentation can be accomplished in a variety of ways, ranging from simple thresholding to complex color image segmentation approaches. Because computers lack the ability to recognize objects intelligently, numerous different methods for segmenting images have been devised. The segmentation process is based on the image's numerous attributes. Color-based segmentation was used as segmentation method to extract the features. Extreme color detection In Images was accomplished by researchers Manuel, Julián and Sergio [82], but the algorithm detects only extreme colors from images. Our dataset contains shadows and noises so it is impossible to segment the ROI part of the region with extreme color detection. In order to segment the leaves from the images, otsu thresholding, background marker and color segmentation were used together. Our segmentation algorithm recognizes and extracts the green value pixel in the healthy image as foreground and the rest as background; similarly, it recognizes and extracts the yellow value pixels in the bacterial wilt image as foreground and the rest as background (makes the pixels equal to zero). The segmentation method treats the white value color pixels as background or sets them to zero because our photos were captured with a white background. The method recognizes bright (high pixels) areas of the image for mealybug segmentation and sets the rest of the image to pixel zero.

a. Healthy



b. Bacterial wilt



c. Mealybug



Figure 3. 3 segmentation of healthy, bacterial wilt and mealybug plant.

The non-ROI part of the image is given a value of zero to accomplish the segmentation. Because the pixel values outside the region of interest are similar, they are (set to zero), this allows the CNN algorithm to focus on learning the distinguishing traits.

Input: image
Output: Segmented Leaf
Begin:
Background_marker()
<pre> def remove_whites (image, marker) white_removal[image[:, :, 0] <= 200] = False # blue channel white_removal[image[:, :, 1] <= 220] = False # green channel white_removal[image[:, :, 2] <= 200] = False # red channel def color_index_marker(color_index_diff, marker): marker[color_index_diff <= -0.05] = False def otsu_color_index(excess_green, excess_red): return cv2.threshold(excess_green - excess_red, 0, 255, cv2.THRESH_BINARY + cv2.THRESH_OTSU) </pre>
<pre> Otsu_segmentation() def apply_marker(image, marker, background = 0, inverse = True): new_image[mask] = background return new_image def segment_with_otsu(image_file, background = 0): ret_val, marker = get_marker(image) segmented_image = apply_marker(image, marker, background) return ret_val, segmented_image </pre>
<pre> def generate_background_marker(file): original_image = read_image(file) marker = np.full((original_image.shape[0], original_image.shape[1]), True) color_index_marker(index_diff(original_image), marker) return original_image, marker </pre>
<pre> def segment_leaf(image_file, filling_mode, smooth_boundary, marker_intensity): original, marker = generate_background_marker(image_file) bin_image = np.zeros((original.shape[0], original.shape[1])) bin_image[marker] = 255 bin_image = bin_image.astype(np.uint8) largest_mask = (select_largest_obj(bin_image, fill_mode=filling_mode,smooth_boundary=smooth_boundary) if marker_intensity > 0: largest_mask[largest_mask != 0] = marker_intensity image = largest_mask else: image = original.copy() image[largest_mask == 0] = np.array([0, 0, 0]) return original, image </pre>
End

Figure 3. 4 Segmentation algorithm for segmenting ROI from image

3.4.4 feature extraction

Feature extraction is a way of representing images numerically for processing. The enormous number of features extraction in these large data sets necessitates a lot of computational resources to process which requires feature selection. Feature selection refers to strategies for selecting and/or combining variables into features in order to reduce the amount of data that needs to be processed while still accurately and thoroughly characterizing the original data set.

During feature extraction many sorts of features can be extracted, including texture, color, edges, shape, geometry, and so on. We went through the literature review for feature selection based on previous work and experiments in the initial stage of the project. The most essential and widely used criteria for plant disease classification are texture and color.

In this study, CNN is used for feature extraction. During the training phase, CNN includes feature extraction and weight computation. A convolution operator, which is beneficial for solving complex processes, is used to give these networks their names. The truth is that CNNs have the primary advantage of automatic feature extraction. The given input data is first sent to a feature extraction network, and the retrieved features are then sent to a classifier network.

3.4.5 Data Augmentation

Many Computer Vision tasks have shown that deep convolutional neural networks perform exceptionally well, However, these networks rely extensively on huge amount of input data. Data augmentation is used to artificially increase the quantity of training dataset. Data augmentation can operate as a regularizer in reducing overfitting and improving generalization in neural networks. It also increases performance in challenges involving unequal class sizes [83]. We ended up with a total of 13544 images after data augmentation.

Data Augmentation uses a set of techniques for increasing the size and quality of training datasets so that better Deep Learning models can be built with them. Some of the techniques are geometric transformations, color space transformations, kernel filters, mixing images, random erasing, feature space augmentation, adversarial training, GAN-based augmentation, neural style transfer, and meta-learning schemes [84]. The augmentation techniques used in this thesis are geometric transformations and color space transformations

Table 3. 1 Augmentation techniques used

parameters	values
Rotation_range	15
rescale	1./255
Shear_range	0.1
Zoom_range	0.2
Horizontal_flip	True
Width_shift_range	0.1
Height_shift_range	0.1

3.5 Classification

Based on the features extracted, classifiers are used to identify and categorize the diseases that affect plant leaves. K-nearest neighbors (K-NN), Support Vector Machines (SVM), Convolutional Neural Network (CNN), and Artificial Neural Network (ANN) are some of the classifiers that can be used to detect diseases in plants. Since CNNs have achieved significant achievements in the field of computer vision, deep convolutional-neural-network (CNN) models are used in this study to identify diseases in Enset plant. There are several CNN architectures to choose from, Alex Nets, GoogLeNet, and ResNet50 are the most popular convolutional neural networks for object detection and categorization from images [85], all of which are experimented with in this study.

CNN consists of 2 main parts: feature extraction and classification. Because CNN is a strong feature extractor, using it to identify Enset disease images can save time on feature engineering. This means that the network learns to optimize the filters (or kernels) by automatic learning, as opposed to hand-engineered filters in traditional techniques. For classification, features learned from the top layers of the CNN are used.

3.5.1 Training Phase

3.5.1.1 Feature Learning for Training and constructing model

Convolution layers

The input layer, hidden layers, and output layer make up a convolutional neural network. The hidden layers of a convolutional neural network include layers that perform convolution. This usually comprises a layer that does a dot product of the convolution kernel with the input matrix

of the layer. The convolution procedure generates a feature map as the convolution kernel slides along the input matrix for the layer, which then contributes to the input of the following layer. Other layers such as pooling layers, fully connected layers, and normalization layers are added after this.

Our CNN model has 12 layers: 1 input layer, 2 convolutional layers with stride and padding, 3 ReLU layers, 3 pooling layers, FCL, Softmax, and output layer. The first convolution layer receives a 256 x 256 x 3 picture as input. To extract features from the Enset plant, we employed 32 and 64 filter sizes, as well as a 3x3 kernel size. The number of pixels shifted over the input matrix is referred to as the stride. When the stride is set to 1, the filters are moved one pixel at a time. We shift the filters two pixels at a time when the stride is two, and so on. We have used stride size of two (2, 2) for this thesis. Filters do not always precisely fit the supplied image. There are two ways to handle this: To accommodate the picture, padding it with zeros (zero-padding) is necessary. Remove the portion of the image where the filter didn't work. This is known as valid padding, and it maintains only the image's legitimate parts. For this thesis, we chose "same" padding, which implies that if the stride size is one, the output is the same size as the input; otherwise, border elements are computed with zero padding to make the input and output same size.

Layer (type)	Output Shape	Param #
conv	(256, 256, 32)	896
max_pooling	(128, 128, 32)	0
dropout	(128, 128, 32)	0
conv	(128, 128, 64)	18496
max_pooling	(64, 64, 64)	0
dropout	(64, 64, 64)	0
flatten	(262144)	0
Dense (96)	(96)	25165920
Dense (3)	(3)	291
Total params: 25,185,603		

Trainable params: 25,185,603
 Non-trainable params: 0

Figure 3. 5 Summary of our model

For example, the following figure shows some part of an image. Let's say this is the first convolution layer with 3x3 kernel, if the CNN model has 64 filters it will do different kinds of 64 filters. The next convolution layers may have 2x2 kernel so the convolution will be a 2x2 metric.

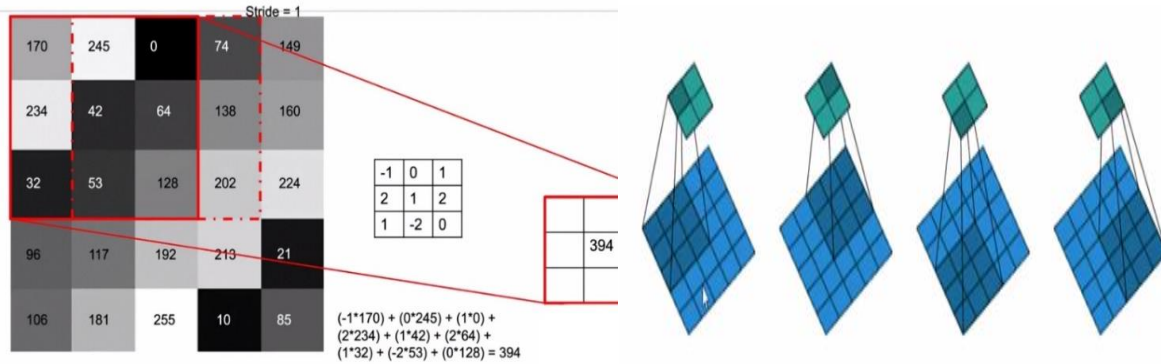


Figure 3. 6 Convolving a 3x3 kernel over a 5x5 input using 2x2 strides with no zero padding (padding="valid")

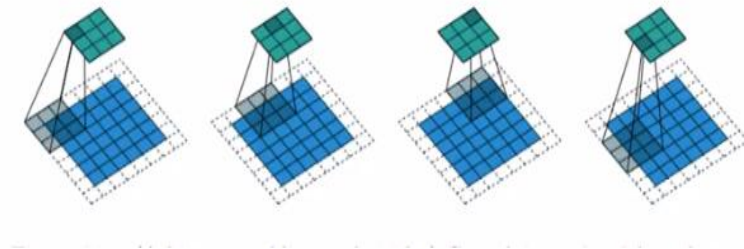


Figure 3. 7 convolving a 3x3 kernel over a 6x6 input padded with a 1x1 border of zeros using 2x2 strides (padding="same")

Pooling layer

Another layer of CNN is the pooling layer, which performs down sampling operations. The convolutional output is fed into this layer, which subsamples it to lower the in-plane dimensionality of the feature maps, the number of parameters, and the model's computational complexity. We used two pooling layers in our model and for both of the pooling layers, we used a 2x2 pool size with stride of 2x2.

There are two methods to do the pooling operation, they are max pooling and average pooling. When using the average pooling method, the image is smoothed out, and so the sharp features may not be visible. Max pooling chooses the image's brightest pixels. It's handy when the image's background is dark and we're just interested in the image's lighter pixels. For this thesis we have used max pooling because we are interested in the bright pixels of the images.

For example, the following figure shows max pooling being applied on part of an image. Max pooling takes the highest pixel value in the 2x2 kernel (pool size). i.e the kernel here kernel indicates the pool size, not the kernel we in used convolution layers.

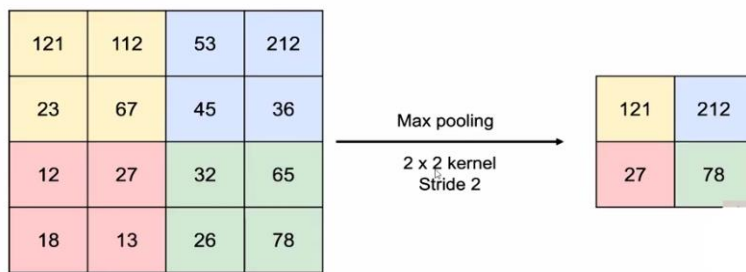


Figure 3. 8 MaxPooling

Activation Function

The activation function is used to determine the yes or no output of a neural network. The activation function in a neural network is responsible for converting the node's summed weighted input into the node's activation or output for that input. The linearity activation function Rectified Linear Unit (ReLU) was employed. The ReLU is a piecewise linear function that, if the input is positive, outputs directly; else, it outputs zero.

Dropout

Dropout is a method for decreasing or preventing overfitting on the training dataset by omitting some units or connections at random. After each Maxpooling layer and also after the flatten layer, we utilized a dropout layer with a dropping probability of 0.25 and 0.4.

3.5.1.2 Classification using the model

Fully connected layer (FC)

A fully connected layer (FC) is one in which all neurons in the first layer are connected to all neurons in the next layer. To compute the final output probabilities for each class, we used two fully connected layer. As we have three classes, the first layer has 96 neurons and the final(output)

layer, which is the model's output layer, has three neurons. The output of the flattening layer, which is a vector value, is accepted by the first FC layer.

3.6 Evaluation Technique

Contingency table is necessary for the performance metrics, which is also called the confusion matrix. A confusion matrix is a table that shows how well a classification model (or "classifier") performs on a set of test data for which the true values are known. This provides a comprehensive overview by summarizing the classification results. It shows the individual results for each of the categories by tabulating the predicted and actual categories. The performance metrics can be calculated by using the confusion matrix, which gives better idea.

CHAPTER FOUR

RESULTS AND DISCUSSION

This chapter describes in full the experimental evaluation of the suggested model for automatic identification and classification of Enset diseases. The proposed model or architecture is approved after an experimental evaluation. The training and validation test outcomes are compared. On our CNN model, the effect of segmentation is tested and compared. In addition, the proposed model's test results are shown and compared to those of other models.

4.1 Image Dataset

Healthy and diseased (bacterial wilt and mealybug) images are taken throughout the image acquisition phase. The Sony SLT-A65V 24 MP Digital camera was used to capture all of the images. All of the images have a pixel resolution of 4248 x 5664 because they were taken with a 24mp digital camera. There are 331 bacterial wilt images, 328 healthy images, and 328 mealybug images. There is no need to execute dataset balancing because each of our classes has almost the same amount of data. Our models accepts 256x256 color images and produces three classes of output. The suggested model performs a number of experiments by varying the ratio of training and testing datasets, using different epochs, and lastly utilizing different activation functions.

	Healthy	mealybug	Bacterial wilt	percent
Training data set	229	229	231	70%
Validation data set	34	34	66	20%
Test data set	65	65	34	10%

Since the amount of dataset is very limited, we have used data augmentation during training based on the parameters discussed in chapter three to artificially increase the number of dataset.

4.2 Experimental Results

In this study CNN is used for feature extraction and classification for constructing Enset disease detection model. The following parameters are used for performance evaluation of each model and the comparison with other state of the art models.

- A. Model accuracy: A model with a higher accuracy score is better for detecting and classifying Enset disease.

- B. Loss of the Model: A model with a lower loss value is more effective at detecting and grading new Enset diseasetypes (taking new images different from training and testing datasets)
- C. Time taken to train the model: a model with the shortest training time is preferred for classifying the testing data

4.2.1 Experimenting CNN algorithm by applying image segmentation

Experimental result of CNN after image segmentation registers a training accuracy of 99.68% and a validation accuracy of 98.87%. It has a training loss of 0.92% and a validation loss of 3.03%. This classification accuracy is obtained when the model is trained by applying segmentation, data augmentation, maxpooling, stride, dropouts and Adam optimizer.

```

Epoch 95/100
39/39 [=====] - 14s 350ms/step - loss: 0.0055 - accuracy: 1.0000 - val_loss: 0.0197 - val_accuracy: 0.9944
WARNING:tensorflow:Can save best model only with val_acc available, skipping.
Epoch 96/100
39/39 [=====] - 14s 349ms/step - loss: 0.0260 - accuracy: 0.9968 - val_loss: 0.0726 - val_accuracy: 0.9887
WARNING:tensorflow:Can save best model only with val_acc available, skipping.
Epoch 97/100
39/39 [=====] - 14s 345ms/step - loss: 0.0467 - accuracy: 0.9822 - val_loss: 0.0538 - val_accuracy: 0.9887
WARNING:tensorflow:Can save best model only with val_acc available, skipping.
Epoch 98/100
39/39 [=====] - 14s 347ms/step - loss: 0.0159 - accuracy: 0.9935 - val_loss: 0.0154 - val_accuracy: 0.9944
WARNING:tensorflow:Can save best model only with val_acc available, skipping.
Epoch 99/100
39/39 [=====] - 14s 349ms/step - loss: 0.0089 - accuracy: 0.9968 - val_loss: 0.0217 - val_accuracy: 0.9887
WARNING:tensorflow:Can save best model only with val_acc available, skipping.
Epoch 100/100
39/39 [=====] - 14s 349ms/step - loss: 0.0180 - accuracy: 0.9903 - val_loss: 0.1114 - val_accuracy: 0.9774
WARNING:tensorflow:Can save best model only with val_acc available, skipping.

```

Figure 4. 1 Classification accuracy and loss of our model

The model takes 8 minutes to train, with each epoch requiring an average of 4.8 seconds. Loading the image, serializing the network, and splitting the dataset into categories all require time. Because of its smaller size and the 12GB GPU (graphical processing unit) contribution from Colab, the model learns faster.

Figure 4.3 and figure 4.4 below show the learning curve of our model. A learning curve is a graph of model learning performance as a function of time or experience. On the x-axis is time or experience, and on the y-axis is learning or improvement.

Training and validation accuracy rises while training and validation loss reduces, as illustrated in the training loss and accuracy curve in figures 4.2 and 4.3. Our graph reveals no evidence of overfitting throughout all epochs because both training and validation accuracy, as well as training and validation loss, are nearly identical in the last epochs. Throughout the curve, the gaps are

relatively narrow. since our model has a stable accuracy and loss curve, we do not need to add batch normalization to our model

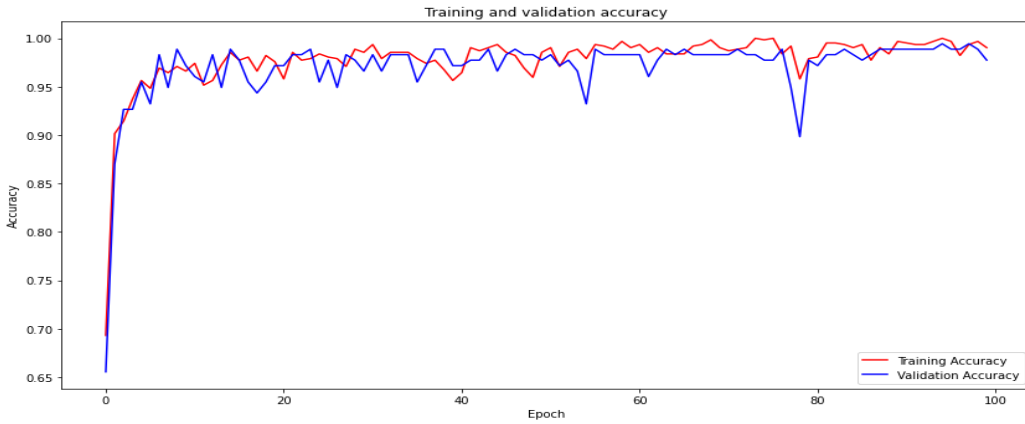


Figure 4. 2 training and validation accuracy of our model



Figure 4. 3 training and validation loss of our model

Once the model has been developed the test data set is used for confirming its accurate predictions. We tested our model's accuracy in predicting on this dataset. We have stated the performance of our model by the confusion matrix below in figure 4.4. On all of our test datasets, our model predicted properly with an accuracy of 99.97% and a loss of only 0.01%.

A confusion matrix is a summary of classification problem prediction outcomes. The number of right and incorrect predictions is broken down by class and summarized with count values. The true labels and predicated labels are plotted on the matrix. There are 34 mealybug class images, 34

bacterial wilt class images, and 32 healthy class images in our test dataset. Our model successfully predicted all of the mealybug images, all of the bacterial wilt images from 34 test images, and all of the healthy images from 32 images.

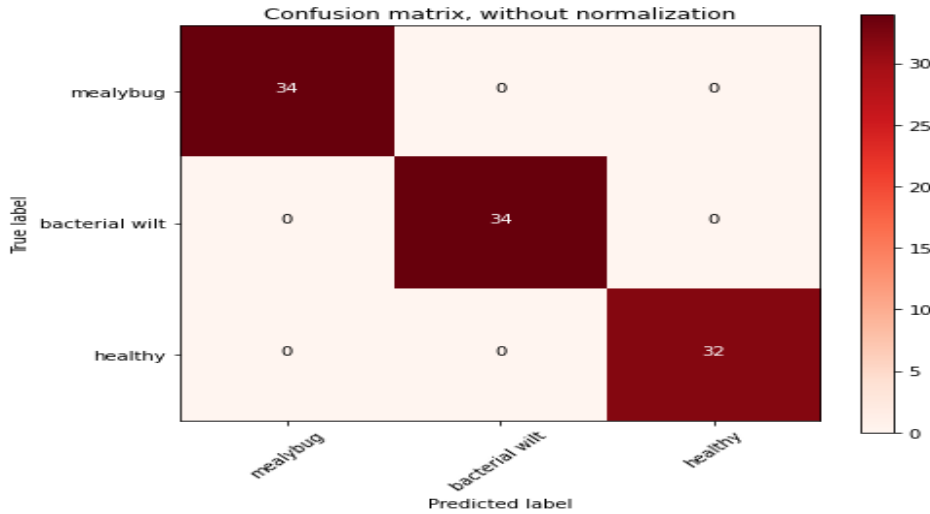


Figure 4. 4 confusion matrix of our model on test dataset

4.2.2 Experimental results of our model with out applying segmentation

In this experiment, we tested our model without applying segmentation. Our Enset disease detection model has a training accuracy of 99.26% and a validation accuracy of 97.93%. It has a training loss of 2.38% and a validation loss of 7.38%. This classification accuracy is obtained when the model is trained by applying data augmentation, maxpooling, stride and dropouts. It took 10.8 minutes to train the model, taking an average of 6.48 seconds for each epoch. It took more time than the model with segmentation because it has to learn all of the raw data given to the model.

```

Epoch 1/100
43/43 [=====] - 48s 380ms/step - loss: 2.4650 - accuracy: 0.5938 - val_loss: 0.5456 - val_accuracy: 0.7927
WARNING:tensorflow:Can save best model only with val_acc available, skipping.
Epoch 2/100
43/43 [=====] - 16s 361ms/step - loss: 0.4342 - accuracy: 0.8301 - val_loss: 0.3178 - val_accuracy: 0.9326
WARNING:tensorflow:Can save best model only with val_acc available, skipping.
Epoch 3/100
43/43 [=====] - 16s 362ms/step - loss: 0.2474 - accuracy: 0.9202 - val_loss: 0.2280 - val_accuracy: 0.9119
WARNING:tensorflow:Can save best model only with val_acc available, skipping.
Epoch 4/100
43/43 [=====] - 15s 358ms/step - loss: 0.2533 - accuracy: 0.9114 - val_loss: 0.4825 - val_accuracy: 0.7876
WARNING:tensorflow:Can save best model only with val_acc available, skipping.
Epoch 5/100
43/43 [=====] - 16s 359ms/step - loss: 0.4011 - accuracy: 0.8523 - val_loss: 0.2661 - val_accuracy: 0.9430
WARNING:tensorflow:Can save best model only with val_acc available, skipping.
- - - - -
Epoch 92/100
43/43 [=====] - 15s 347ms/step - loss: 0.0424 - accuracy: 0.9823 - val_loss: 0.0484 - val_accuracy: 0.9896
WARNING:tensorflow:Can save best model only with val_acc available, skipping.
Epoch 93/100
43/43 [=====] - 15s 351ms/step - loss: 0.0238 - accuracy: 0.9926 - val_loss: 0.0738 - val_accuracy: 0.9793
WARNING:tensorflow:Can save best model only with val_acc available, skipping.
Epoch 94/100
43/43 [=====] - 15s 348ms/step - loss: 0.0232 - accuracy: 0.9897 - val_loss: 0.0177 - val_accuracy: 0.9896
WARNING:tensorflow:Can save best model only with val_acc available, skipping.
Epoch 95/100
43/43 [=====] - 15s 352ms/step - loss: 0.0331 - accuracy: 0.9882 - val_loss: 0.0727 - val_accuracy: 0.9741
WARNING:tensorflow:Can save best model only with val_acc available, skipping.
Epoch 96/100
43/43 [=====] - 15s 351ms/step - loss: 0.1124 - accuracy: 0.9734 - val_loss: 0.0361 - val_accuracy: 0.9845
WARNING:tensorflow:Can save best model only with val_acc available, skipping.
Epoch 97/100
43/43 [=====] - 15s 352ms/step - loss: 0.0663 - accuracy: 0.9778 - val_loss: 0.0534 - val_accuracy: 0.9793
WARNING:tensorflow:Can save best model only with val_acc available, skipping.
Epoch 98/100
43/43 [=====] - 15s 349ms/step - loss: 0.0695 - accuracy: 0.9705 - val_loss: 0.1279 - val_accuracy: 0.9585
WARNING:tensorflow:Can save best model only with val_acc available, skipping.
Epoch 99/100
43/43 [=====] - 15s 353ms/step - loss: 0.0912 - accuracy: 0.9793 - val_loss: 0.0716 - val_accuracy: 0.9793
WARNING:tensorflow:Can save best model only with val_acc available, skipping.
Epoch 100/100
43/43 [=====] - 15s 355ms/step - loss: 0.0505 - accuracy: 0.9867 - val_loss: 0.0544 - val_accuracy: 0.9845
WARNING:tensorflow:Can save best model only with val_acc available, skipping.

```

Figure 4. 5 training and validation accuracy and loss

The graphs below indicate the training and validation learning curves. The learning curves indicate that our model performs well on unsegmented images too. The graphs indicate no sign of overfitting and underfitting.

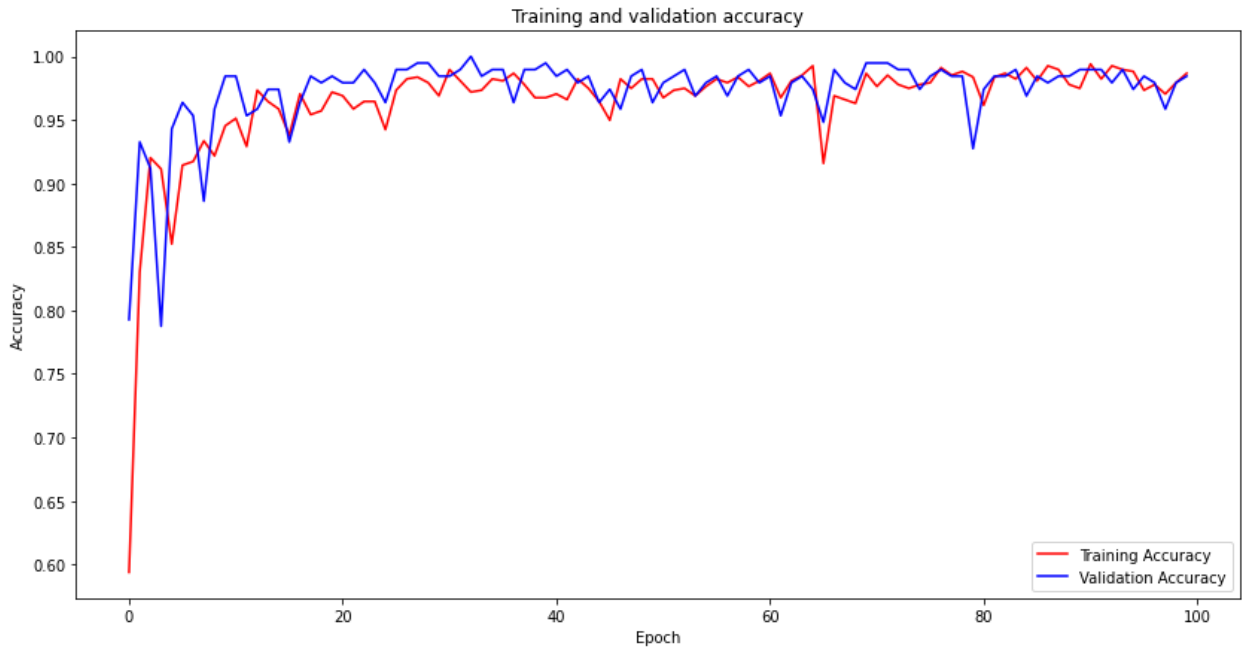


Figure 4. 6 training and validation accuracy learning curves without segmentation

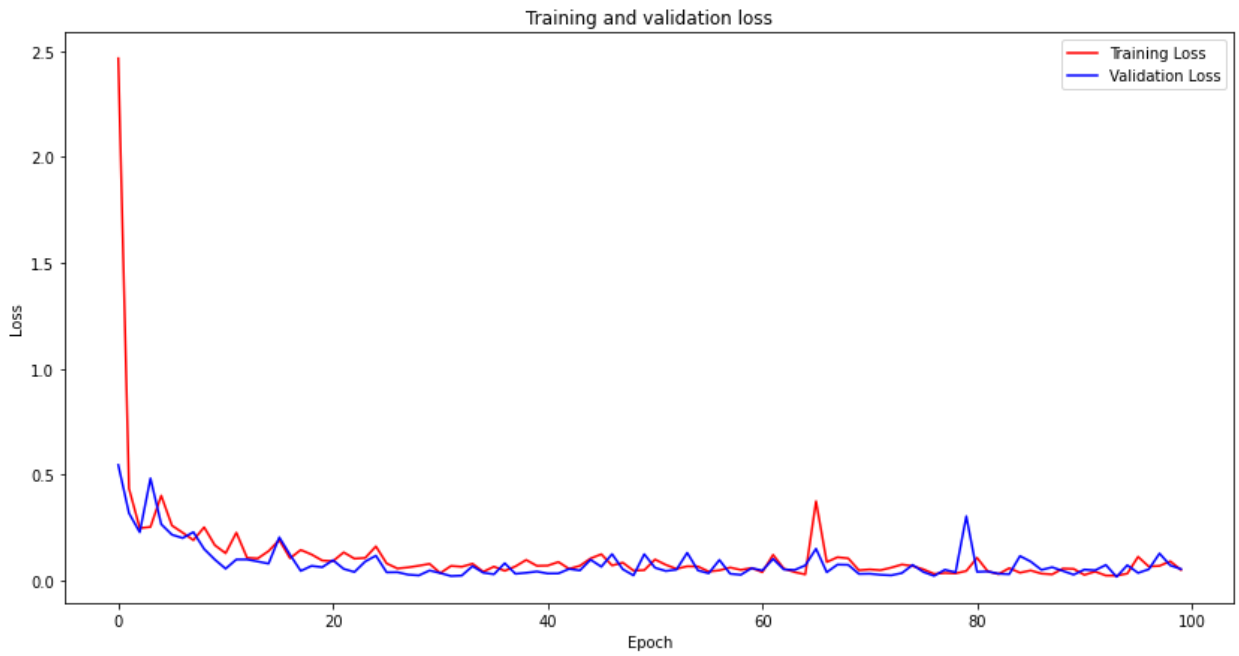


Figure 4. 7 training and validation loss without segmentation

As shown in the confusion matrix below, our model successfully predicted correctly on all of our test dataset with an accuracy of 99.70% and a loss of only 1.29%.

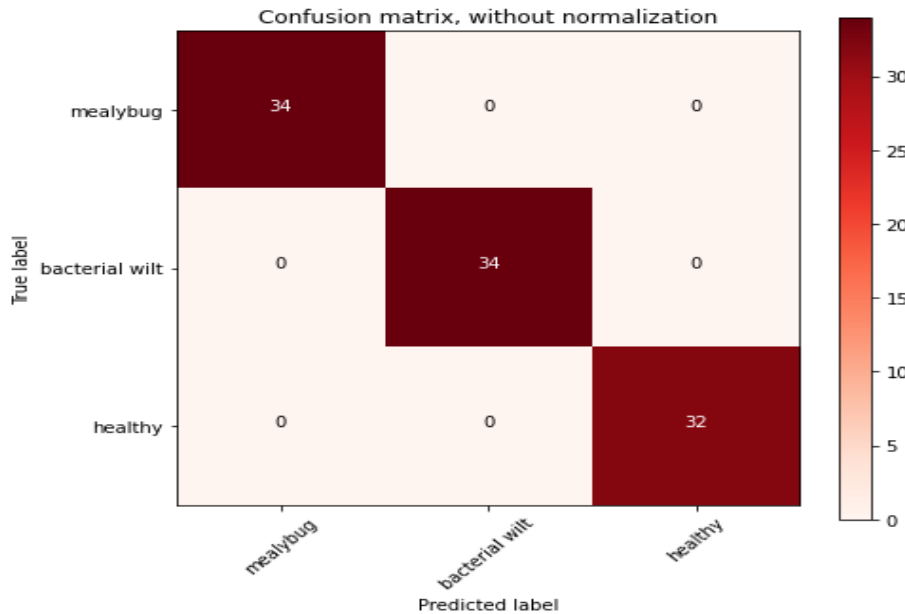


Figure 4. 8 confusion matrix result on test data without segmentation

4.2.3 Experimental results of our model without Data Augmentation

In this experiment, we tested our model without applying Data Augmentation. Our Enset disease detection model has a training accuracy of 98.70% and a validation accuracy of 37.76%. It has a training loss of 6.88% and a validation loss of 26.35%. When training our model, data augmentation was used as a regularizer to help reduce overfitting. Overfitting occurred when the model was trained without Data Augmentation, as indicated in figure 4.9 below.

```

Epoch 97/100
44/44 [=====] - ETA: 0s - loss: 0.0688 - accuracy: 0.9870WARNING:tensorflow:Can save best model only with val_acc available, skipping.
44/44 [=====] - 9s 193ms/step - loss: 0.0688 - accuracy: 0.9870 - val_loss: 2.6357 - val_accuracy: 0.3776
Epoch 98/100
44/44 [=====] - ETA: 0s - loss: 0.0215 - accuracy: 0.9942WARNING:tensorflow:Can save best model only with val_acc available, skipping.
44/44 [=====] - 9s 193ms/step - loss: 0.0215 - accuracy: 0.9942 - val_loss: 2.9680 - val_accuracy: 0.4337
Epoch 99/100
44/44 [=====] - ETA: 0s - loss: 0.0555 - accuracy: 0.9899WARNING:tensorflow:Can save best model only with val_acc available, skipping.
44/44 [=====] - 9s 193ms/step - loss: 0.0555 - accuracy: 0.9899 - val_loss: 2.5221 - val_accuracy: 0.3776
Epoch 100/100
44/44 [=====] - ETA: 0s - loss: 0.0535 - accuracy: 0.9870WARNING:tensorflow:Can save best model only with val_acc available, skipping.
44/44 [=====] - 9s 192ms/step - loss: 0.0535 - accuracy: 0.9870 - val_loss: 3.8292 - val_accuracy: 0.3520

```

Figure 4. 9 Performance of our model without Data Augmentation

Training accuracy rises while Validation accuracy reduces, as illustrated in the training loss and accuracy curve in figure 4.10 and 4.11. The graph reveals evidence of overfitting throughout all

epochs because both training and validation accuracy, as well as training and validation loss, are widely varied in most of the epochs.

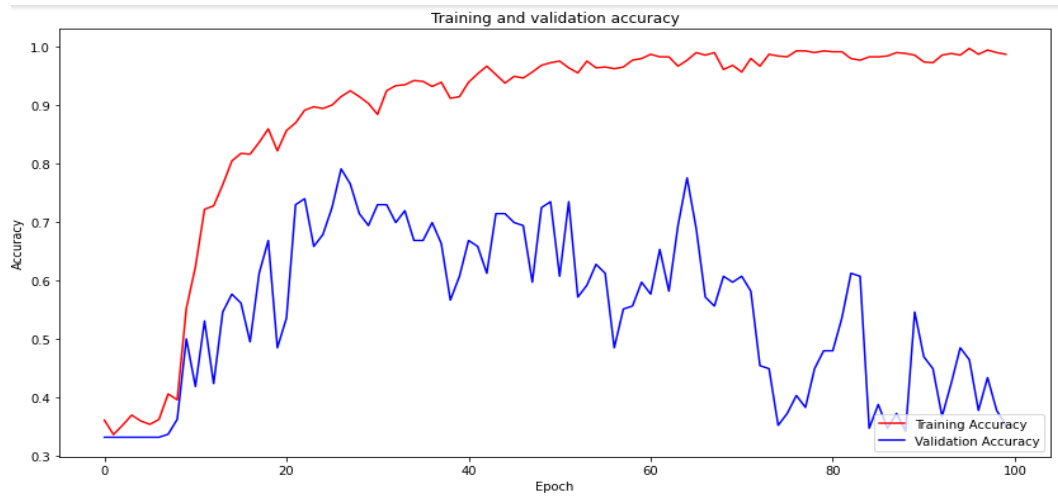


Figure 4. 10 Training and Validation accuracy of our model without Data Augmentation

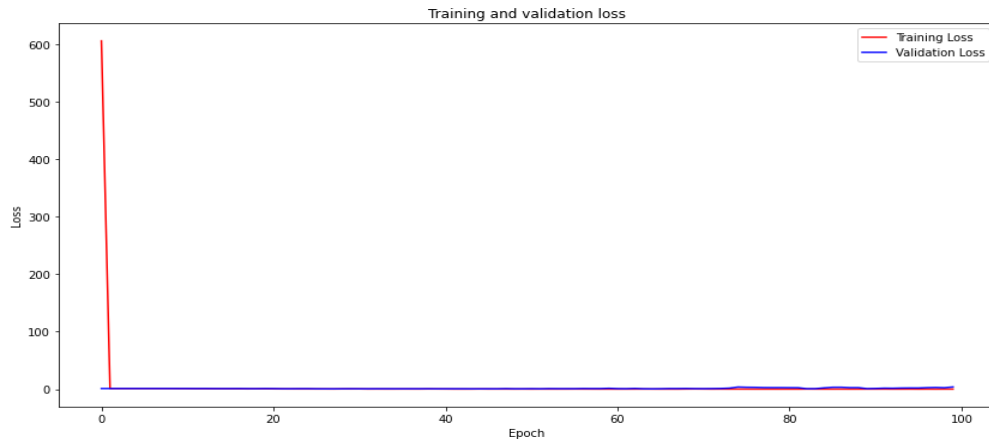


Figure 4. 11 Training and Validation loss of our model without Data Augmentation

As clearly depicted in figure 4.12 the confusion matrix shows the following results. There are 34 images for each of our class in our test dataset. From 34 mealybug images, the model predicted only 4 of them correctly but wrongly predicted 30 of them as bacterial wilt. From 34 images of healthy, the model all of them as bacterial wilt. The model predicted all the bacterial wilt class images correctly. Totally, the model predicted most of the images incorrectly with an accuracy of 37.25% and a loss of 38.23%.

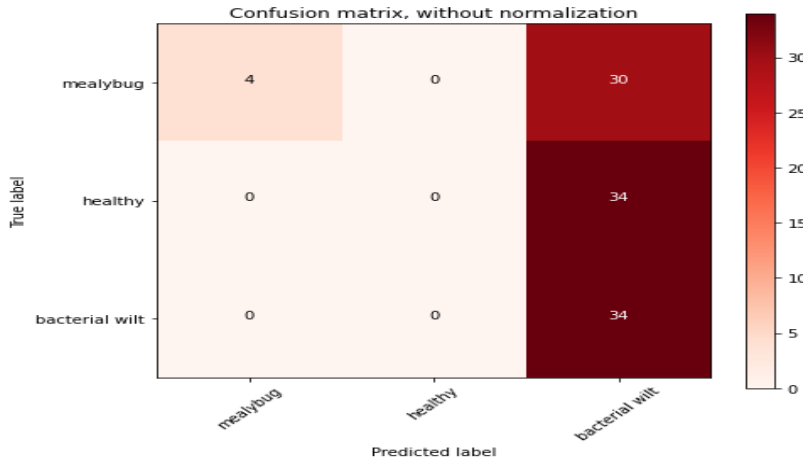


Figure 4. 12 Our model Prediction result without Data Augmentation

4.2.4 Changing Training and Testing Dataset Ratio

The outcome of the proposed model with a split ratio of 70/30% for training and testing dataset is examined. Our Enset disease detection model registered a training accuracy of 99.15% and a validation accuracy of 98.98%. It has a training loss of 2.19% and a validation loss of 5.13%. This classification accuracy is obtained when the model is trained by applying data augmentation, maxpooling, stride and dropouts. It took 11.7 minutes to train the model, taking an average of 7.02 seconds for each epoch.

```

Epoch 97/100
37/37 [=====] - ETA: 0s - loss: 0.0150 - accuracy: 0.9932WARNING:tensorflow:Can save best model only with val_acc available, skipping.
37/37 [=====] - 15s 390ms/step - loss: 0.0150 - accuracy: 0.9932 - val_loss: 0.0924 - val_accuracy: 0.9831
Epoch 98/100
37/37 [=====] - ETA: 0s - loss: 0.0135 - accuracy: 0.9966WARNING:tensorflow:Can save best model only with val_acc available, skipping.
37/37 [=====] - 14s 388ms/step - loss: 0.0135 - accuracy: 0.9966 - val_loss: 0.0575 - val_accuracy: 0.9763
Epoch 99/100
37/37 [=====] - ETA: 0s - loss: 0.0266 - accuracy: 0.9898WARNING:tensorflow:Can save best model only with val_acc available, skipping.
37/37 [=====] - 15s 391ms/step - loss: 0.0266 - accuracy: 0.9898 - val_loss: 0.0759 - val_accuracy: 0.9831
Epoch 100/100
37/37 [=====] - ETA: 0s - loss: 0.0219 - accuracy: 0.9915WARNING:tensorflow:Can save best model only with val_acc available, skipping.
37/37 [=====] - 14s 382ms/step - loss: 0.0219 - accuracy: 0.9915 - val_loss: 0.0513 - val_accuracy: 0.9898

```

Figure 4. 13 training and validation accuracy and loss of 70/30 split

Training and validation accuracy rises while training and validation loss reduces, as illustrated in the training loss and accuracy curve in figures 4.14 and 4.15. Our graph reveals no evidence of overfitting throughout all epochs because both training and validation accuracy, as well as training and validation loss, are nearly identical in the last epochs. Throughout the curve, the gaps are

relatively narrow. since our model has a stable accuracy and loss curve, we do not need to add batch normalization to our model



Figure 4. 14 training and validation accuracy of 70/30 split

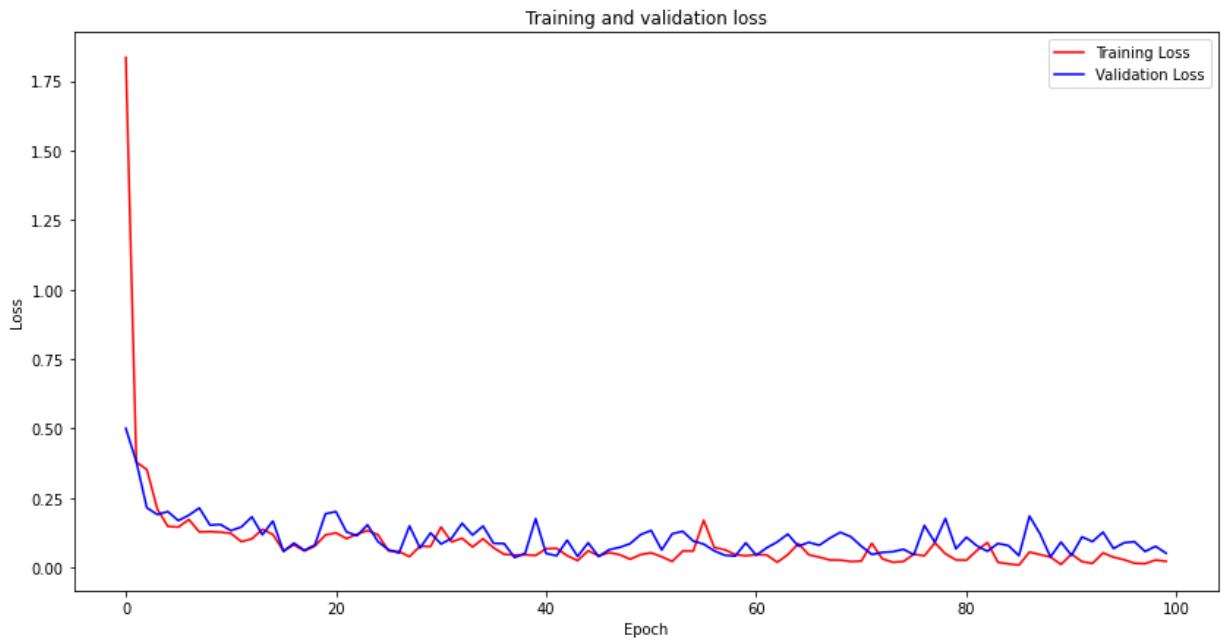


Figure 4. 15 training and validation loss of 70/30 split

As shown in the confusion matrix below, our model successfully predicted correctly on all of our test dataset with an accuracy of 99.95% and a loss of only 0.7%.

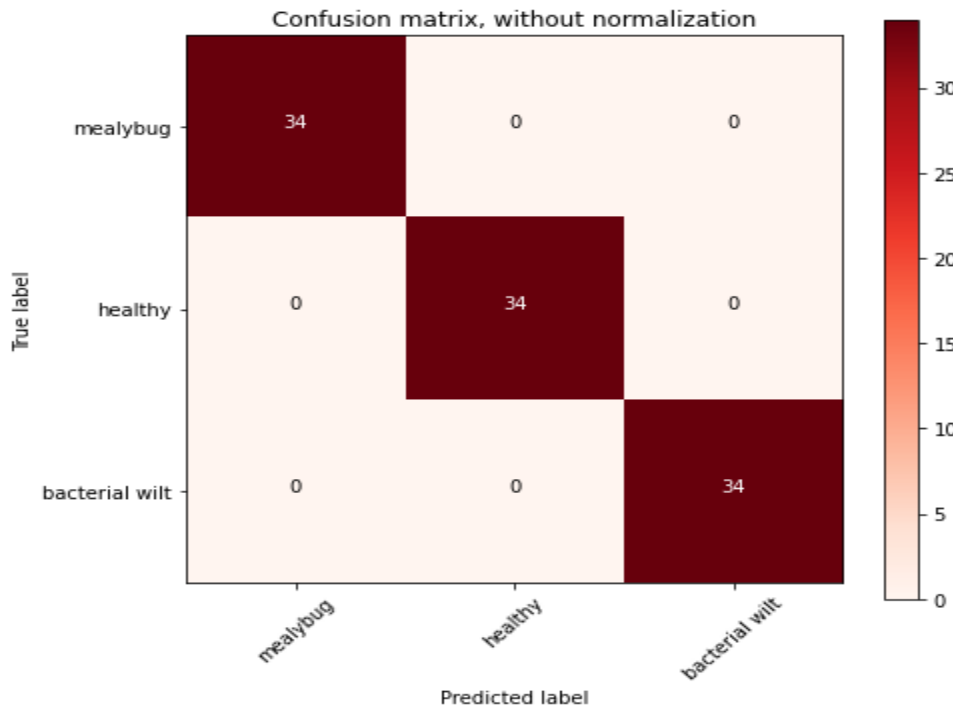


Figure 4. 16 confusion matrix of 70/30 split

4.2.5 Changing optimizers

4.2.5.1 Experimental results of our model by applying AdaGrad Optimizer

In this experiment, we tested our model by changing the optimizer from Adam to AdaGrad. Our model registered a training accuracy of 96.79% and a validation accuracy of 92.88%. It has a training loss of 8.84% and a validation loss of 26.19%. The model took 30 minutes to train, with each epoch requiring an average of 18 seconds.

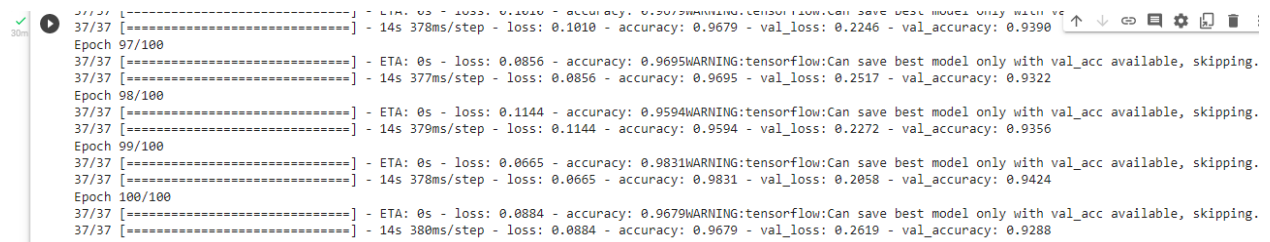


Figure 4. 17 training and validation accuracy and loss by applying AdaGrad Optimizer

The graphs below indicate the training and validation learning curves. The learning curves indicate that our model performs well by using AdaGrad optimizer too. The graphs indicate no sign of overfitting and underfitting.

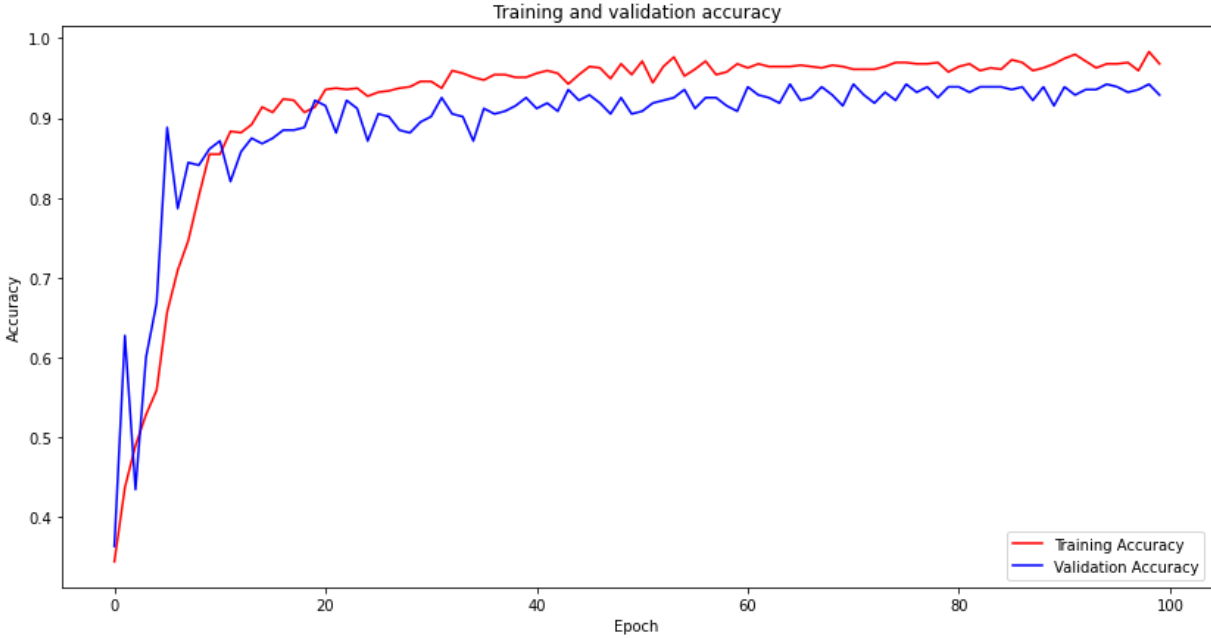


Figure 4. 18 training and validation accuracy graph of our model by applying AdaGrad optimizer

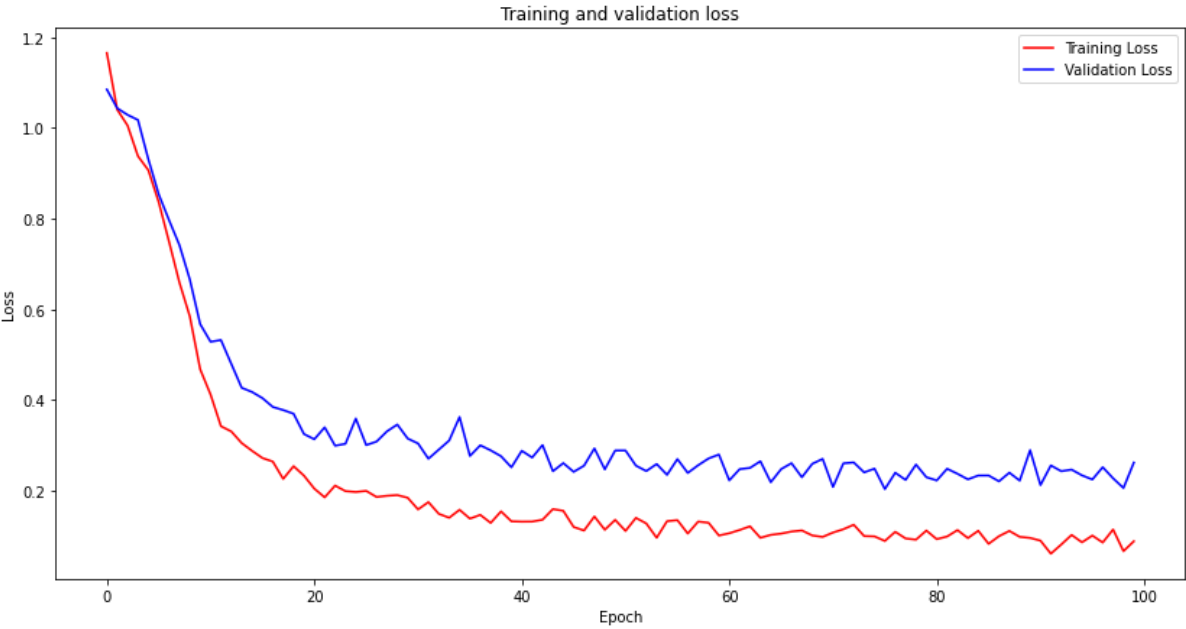


Figure 4. 19 training and validation loss graph of our model by applying AdaGrad optimizer

As depicted in figure 4.20 below, the confusion matrix shows the following results. There are 34 images for each of our class in our test dataset. From 34 mealybug images, the model predicted 30 of them correctly but wrongly predicted 3 as healthy and 1 image as bacterial wilt. From 34 images of healthy, the model predicted all of them correctly. From 34 bacterial wilt images, the model predicted 32 of them correctly but wrongly predicted 1 as healthy and 1 image as mealybug. Totally, the model predicted with an accuracy of 94.11% and a loss of 18.57%.

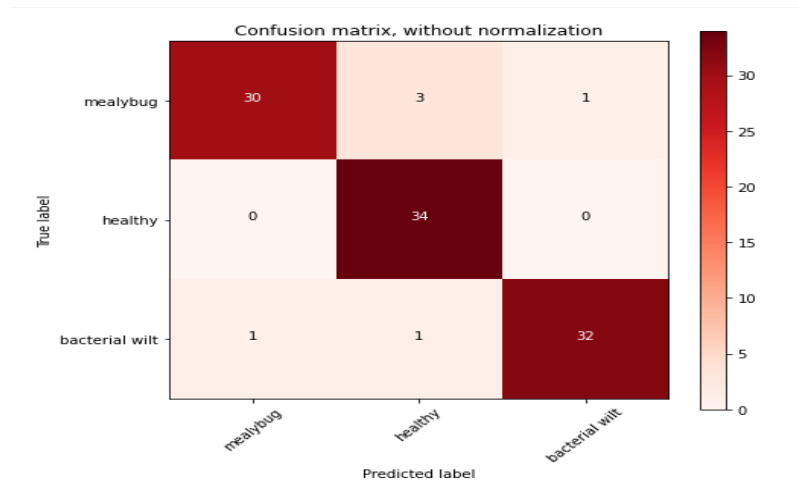


Figure 4. 20 Confusion matrix of our model after applying AdaGrad optimizer

4.2.5.2 Experimental result of our model by applying Stochastic Gradient Descent (SGD)

In this experiment, we tested our model by changing the optimizer from to SGD. Our model registered a training accuracy of 98.65% and a validation accuracy of 96.61%. It has a training loss of 3.98% and a validation loss of 14.26%. The model took 25 minutes to train, with each epoch requiring an average of 15 seconds.

```

[10] 37/37 [=====] - 14s 368ms/step - loss: 0.0471 - accuracy: 0.9882 - val_loss: 0.1700 - val_accuracy: 0.9492
Epoch 97/100
37/37 [=====] - ETA: 0s - loss: 0.0455 - accuracy: 0.9882WARNING:tensorflow:Can save best model only with val_acc available, skipping.
37/37 [=====] - 13s 361ms/step - loss: 0.0455 - accuracy: 0.9882 - val_loss: 0.1392 - val_accuracy: 0.9525
Epoch 98/100
37/37 [=====] - ETA: 0s - loss: 0.0504 - accuracy: 0.9882WARNING:tensorflow:Can save best model only with val_acc available, skipping.
37/37 [=====] - 14s 363ms/step - loss: 0.0504 - accuracy: 0.9882 - val_loss: 0.1854 - val_accuracy: 0.9458
Epoch 99/100
37/37 [=====] - ETA: 0s - loss: 0.0544 - accuracy: 0.9797WARNING:tensorflow:Can save best model only with val_acc available, skipping.
37/37 [=====] - 14s 366ms/step - loss: 0.0544 - accuracy: 0.9797 - val_loss: 0.1593 - val_accuracy: 0.9559
Epoch 100/100
37/37 [=====] - ETA: 0s - loss: 0.0398 - accuracy: 0.9865WARNING:tensorflow:Can save best model only with val_acc available, skipping.
37/37 [=====] - 14s 363ms/step - loss: 0.0398 - accuracy: 0.9865 - val_loss: 0.1426 - val_accuracy: 0.9661
    
```

Figure 4. 21 training and validation accuracy and loss with SGD

The graphs below indicate the training and validation learning curves. The learning curves indicate that our model performs well by using SGD optimizer too. The graphs indicate no sign of overfitting and underfitting.

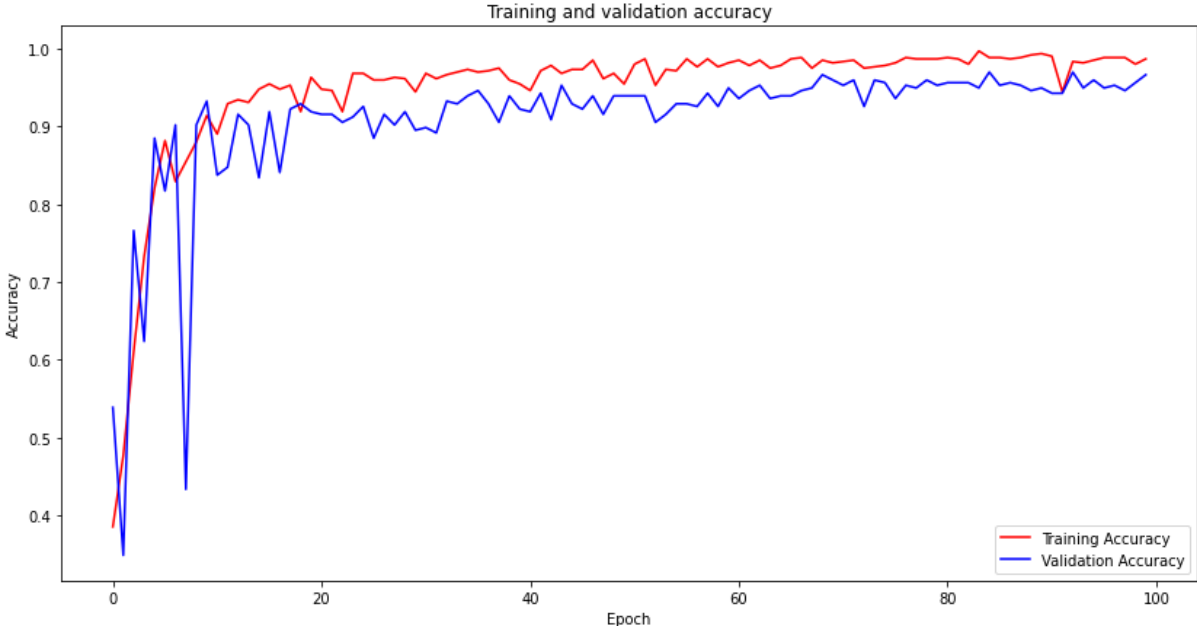


Figure 4. 22 training and validation accuracy graph with SGD

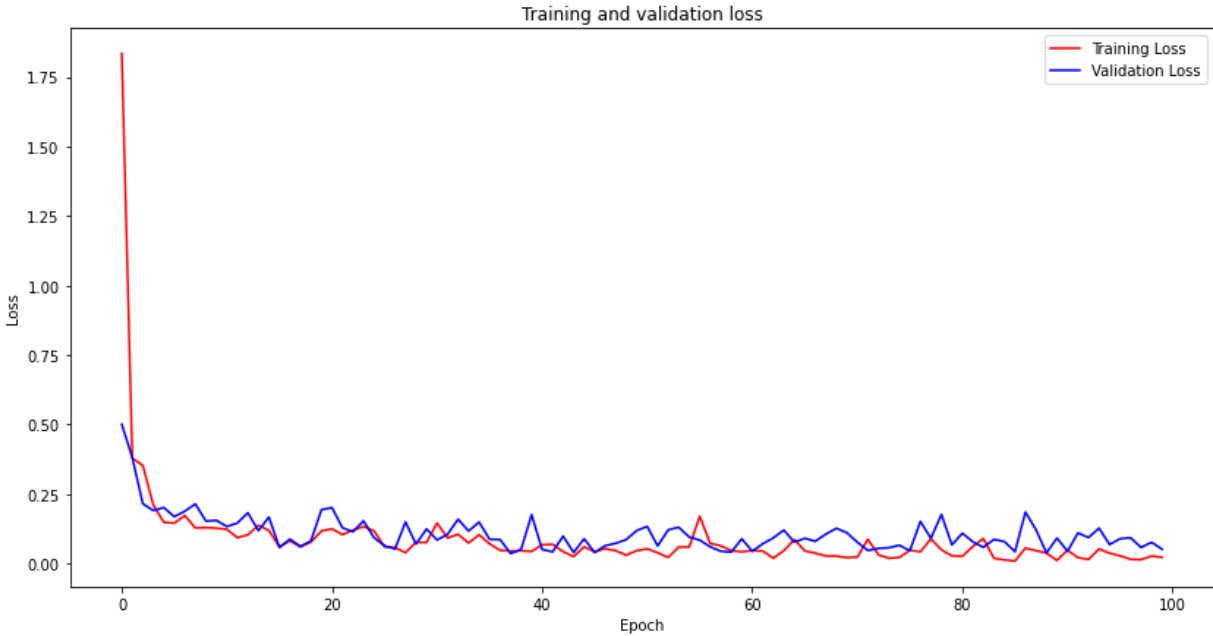


Figure 4. 23 training and validation loss with SGD

As depicted in figure 4.24 below, the confusion matrix shows the following results. There are 34 images for each of our class in our test dataset. From 34 mealybug images, the model predicted 32 of them correctly but wrongly predicted 2 as bacterial wilt. From 34 images of healthy, the model predicted all of them correctly. From 34 bacterial wilt images, the model predicted 33 of them correctly but wrongly predicted 1 image as mealybug. Totally, the model predicted with an accuracy of 97.05% and a loss of 12.88%.

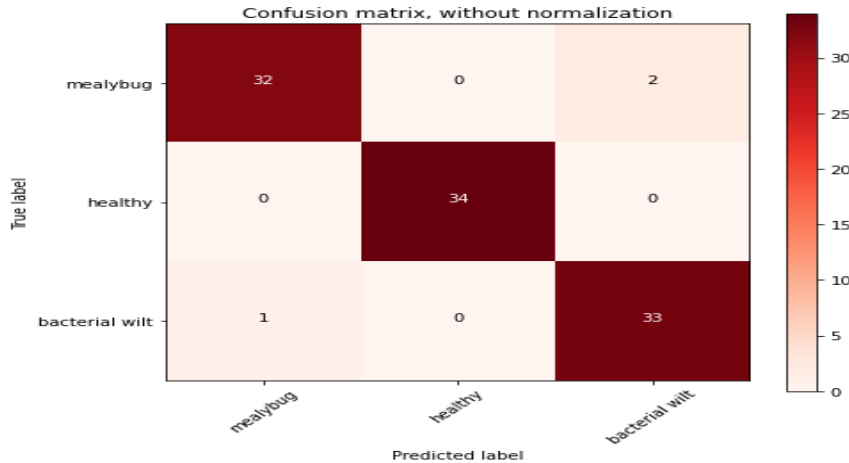


Figure 4. 24 confusion matrix after applying SGD

4.2.6 Comparing the proposed model with the state-of-the-art models

We compare our model's performance to that of the state-of-the-art models to assess its performance. Models that have won the ILSVRC (ImageNet Large Scale (Visual Recognition Challenges) in previous years are considered the state-of-the-art models. Such models include AlexNet, VGGNet, ResNet, Inceptionv3, DenseNet, and EfficientNet. We used the architecture from the keras docs to implement the model. We used our own dataset to train and evaluate each model. The following parameters are used to do the comparison.

4.2.6.1 Comparison with AlexNet model

The detail description of AlexNet model is explained in chapter two. The performance (accuracy and loss value) of AlexNet50 model is shown in figure 4.25. AlexNet50 achieves 97.34% training and 94.30% validation accuracy. It has a training loss of 8.54% and a validation loss of 21.17%. It takes 24 minutes to train the model, taking 14.4 seconds per epoch on average. The size of the model is also very huge (321 MB).

```

Epoch 1/100
43/43 [=====] - 243s 5s/step - loss: 1.0347 - accuracy: 0.7297 - val_loss: 28.8255 - val_accuracy: 0.3161
WARNING:tensorflow:Can save best model only with val_acc available, skipping.
Epoch 2/100
43/43 [=====] - 12s 281ms/step - loss: 0.7067 - accuracy: 0.8035 - val_loss: 5.9120 - val_accuracy: 0.4870
WARNING:tensorflow:Can save best model only with val_acc available, skipping.
Epoch 3/100
43/43 [=====] - 12s 281ms/step - loss: 0.4887 - accuracy: 0.8375 - val_loss: 4.3156 - val_accuracy: 0.6062
WARNING:tensorflow:Can save best model only with val_acc available, skipping.
Epoch 4/100
43/43 [=====] - 12s 280ms/step - loss: 0.5939 - accuracy: 0.7829 - val_loss: 2.6247 - val_accuracy: 0.7461
WARNING:tensorflow:Can save best model only with val_acc available, skipping.
Epoch 5/100
43/43 [=====] - 12s 279ms/step - loss: 0.4480 - accuracy: 0.8479 - val_loss: 1.0490 - val_accuracy: 0.7513
WARNING:tensorflow:Can save best model only with val_acc available, skipping.
Epoch 6/100
43/43 [=====] - 12s 276ms/step - loss: 0.4764 - accuracy: 0.8434 - val_loss: 2.7066 - val_accuracy: 0.5078
.....

Epoch 92/100
43/43 [=====] - 12s 281ms/step - loss: 0.1309 - accuracy: 0.9645 - val_loss: 0.9198 - val_accuracy: 0.8290
WARNING:tensorflow:Can save best model only with val_acc available, skipping.
Epoch 93/100
43/43 [=====] - 12s 274ms/step - loss: 0.1613 - accuracy: 0.9542 - val_loss: 0.2891 - val_accuracy: 0.9534
WARNING:tensorflow:Can save best model only with val_acc available, skipping.
Epoch 94/100
43/43 [=====] - 12s 277ms/step - loss: 0.1774 - accuracy: 0.9468 - val_loss: 0.1655 - val_accuracy: 0.9534
WARNING:tensorflow:Can save best model only with val_acc available, skipping.
Epoch 95/100
43/43 [=====] - 12s 276ms/step - loss: 0.0992 - accuracy: 0.9616 - val_loss: 0.2055 - val_accuracy: 0.9430
WARNING:tensorflow:Can save best model only with val_acc available, skipping.
Epoch 96/100
43/43 [=====] - 12s 276ms/step - loss: 0.1149 - accuracy: 0.9675 - val_loss: 0.1059 - val_accuracy: 0.9585
WARNING:tensorflow:Can save best model only with val_acc available, skipping.
Epoch 97/100
43/43 [=====] - 12s 277ms/step - loss: 0.0854 - accuracy: 0.9749 - val_loss: 0.2117 - val_accuracy: 0.9430
WARNING:tensorflow:Can save best model only with val_acc available, skipping.
Epoch 98/100
43/43 [=====] - 12s 277ms/step - loss: 0.0940 - accuracy: 0.9734 - val_loss: 0.4010 - val_accuracy: 0.8756
WARNING:tensorflow:Can save best model only with val_acc available, skipping.
Epoch 99/100
43/43 [=====] - 12s 276ms/step - loss: 0.0957 - accuracy: 0.9675 - val_loss: 0.1923 - val_accuracy: 0.9171
WARNING:tensorflow:Can save best model only with val_acc available, skipping.
Epoch 100/100
43/43 [=====] - 12s 278ms/step - loss: 0.1328 - accuracy: 0.9616 - val_loss: 0.2895 - val_accuracy: 0.9326
WARNING:tensorflow:Can save best model only with val_acc available, skipping.

```

Figure 4. 25 AlexNet training and validation accuracy and loss

Training and validation accuracy rises while training and validation loss reduces, as illustrated in the training loss and accuracy curve in figure 4.26 and 4.27. The graph reveals evidence of overfitting throughout all epochs because both training and validation accuracy, as well as training and validation loss, are widely varied in most of the epochs.

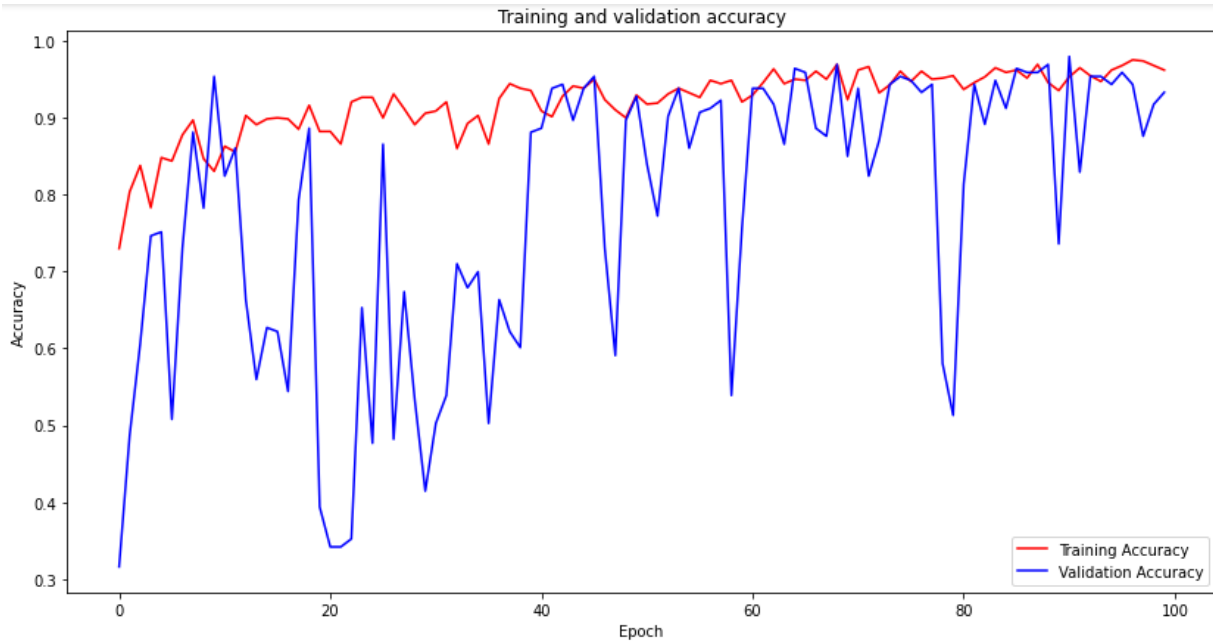


Figure 4. 26 AlexNet training and validation accuracy graph

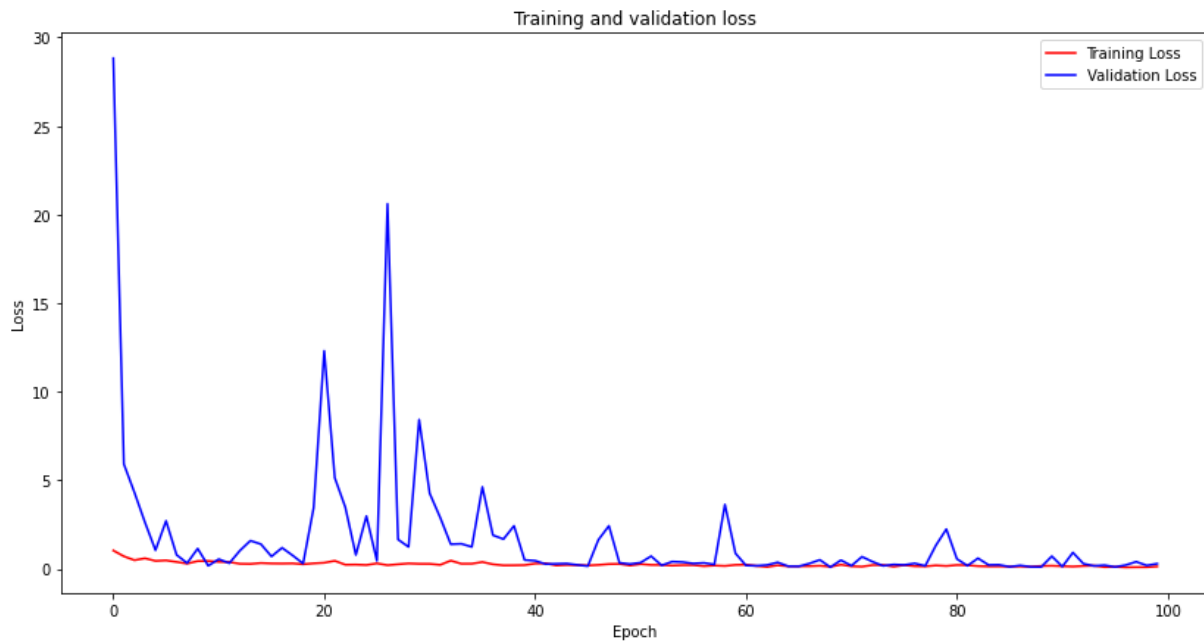


Figure 4. 27 AlexNet training and validation loss graph

As clearly depicted in figure 4.28 the confusion matrix shows the following results. There are 34 mealybug class images, 34 bacterial wilt class images, and 32 healthy class images in our test dataset. From 34 mealybug images Alexnet predicted 27 of them correctly but wrongly predicted

7 of them as healthy. From 34 images of bacterial wilt, Alexnet correctly predicted 33 of them and wrongly predicted 1 as healthy. Alexnet predicted all the healthy class images correctly. Totally, AlexNet predicted the images with an accuracy of 92% and a loss of 22.02%.

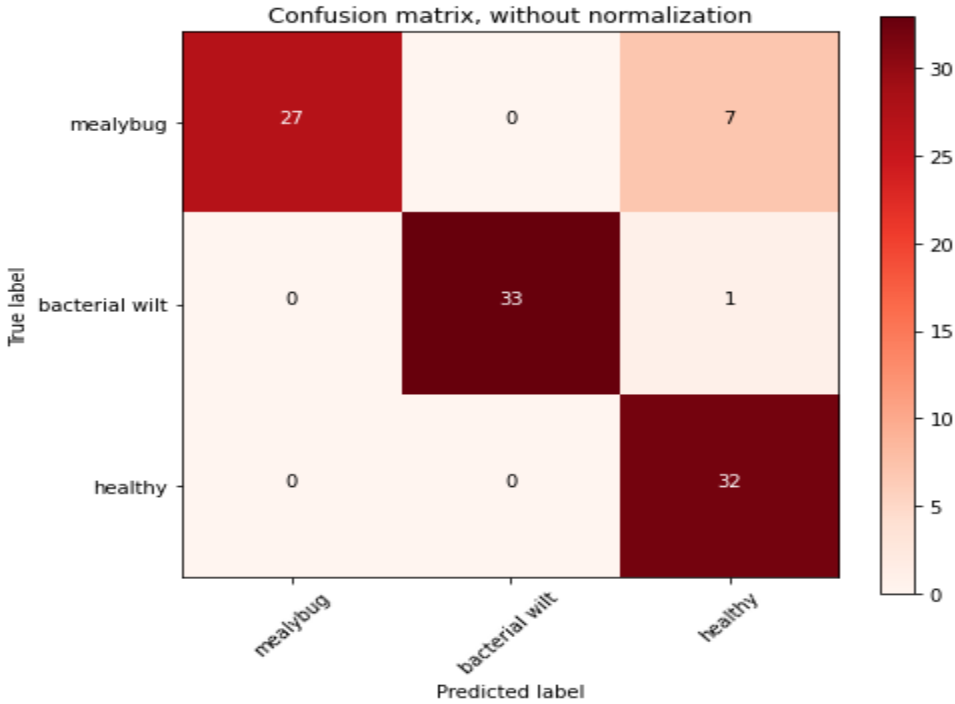


Figure 4. 28 AlexNet model Confusion matrix

4.2.6.2 Comparison with VGG16

The performance (accuracy and loss value) of VGG16 model is shown in figure 4.29, 4.30 and 4.31. VGG16 achieves 99.89% training and 98.96 validation accuracy on our data, as shown in figure 4.29. It has a training loss of 0.1% and a validation loss of 2.54%. It takes 20 minutes to train the model, taking 24 seconds per epoch on average. The size of the model is also very huge (57.1 MB). We training the VGG16 model with epochs number of 50 because the model already has highest accuracy starting from 1st epoch.

```

Epoch 1/50
22/22 [=====] - 209s 8s/step - loss: 0.3861 - accuracy: 0.8508 - val_loss: 0.1375 - val_accuracy: 0.9378
Epoch 2/50
22/22 [=====] - 12s 556ms/step - loss: 0.0537 - accuracy: 0.9823 - val_loss: 0.0495 - val_accuracy: 0.9896
Epoch 3/50
22/22 [=====] - 12s 559ms/step - loss: 0.0244 - accuracy: 0.9911 - val_loss: 0.0336 - val_accuracy: 0.9896
Epoch 4/50
22/22 [=====] - 12s 555ms/step - loss: 0.0186 - accuracy: 0.9956 - val_loss: 0.0303 - val_accuracy: 0.9896
Epoch 5/50
22/22 [=====] - 12s 552ms/step - loss: 0.0147 - accuracy: 0.9985 - val_loss: 0.0328 - val_accuracy: 0.9896
Epoch 6/50
22/22 [=====] - 12s 553ms/step - loss: 0.0141 - accuracy: 0.9970 - val_loss: 0.0320 - val_accuracy: 0.9896
Epoch 7/50
22/22 [=====] - 12s 553ms/step - loss: 0.0127 - accuracy: 0.9956 - val_loss: 0.0343 - val_accuracy: 0.9896
Epoch 8/50

```

```

Epoch 42/50
22/22 [=====] - 12s 551ms/step - loss: 5.8377e-04 - accuracy: 1.0000 - val_loss: 0.0272 - val_accuracy: 0.9845
Epoch 43/50
22/22 [=====] - 12s 555ms/step - loss: 5.4021e-04 - accuracy: 1.0000 - val_loss: 0.0273 - val_accuracy: 0.9896
Epoch 44/50
22/22 [=====] - 12s 556ms/step - loss: 5.6440e-04 - accuracy: 1.0000 - val_loss: 0.0280 - val_accuracy: 0.9896
Epoch 45/50
22/22 [=====] - 12s 554ms/step - loss: 9.1882e-04 - accuracy: 1.0000 - val_loss: 0.0243 - val_accuracy: 0.9845
Epoch 46/50
22/22 [=====] - 12s 552ms/step - loss: 5.2187e-04 - accuracy: 1.0000 - val_loss: 0.0262 - val_accuracy: 0.9896
Epoch 47/50
22/22 [=====] - 12s 555ms/step - loss: 9.4271e-04 - accuracy: 1.0000 - val_loss: 0.0233 - val_accuracy: 0.9845
Epoch 48/50
22/22 [=====] - 12s 558ms/step - loss: 4.9283e-04 - accuracy: 1.0000 - val_loss: 0.0249 - val_accuracy: 0.9896
Epoch 49/50
22/22 [=====] - 12s 556ms/step - loss: 4.2895e-04 - accuracy: 1.0000 - val_loss: 0.0258 - val_accuracy: 0.9896
Epoch 50/50
22/22 [=====] - 12s 554ms/step - loss: 4.5693e-04 - accuracy: 1.0000 - val_loss: 0.0254 - val_accuracy: 0.9896

```

Figure 4. 29 VGG16 training and validation accuracy and loss

The graphs below indicate the training and validation loss and accuracy curves of VGG16. Training and validation accuracy rises while training and validation loss reduces, as illustrated in the training loss and accuracy curve in figure 4.30 and 4.31. The graph reveals no evidence of overfitting throughout all epochs because both training and validation accuracy, as well as training and validation loss, are linearly going up and down.

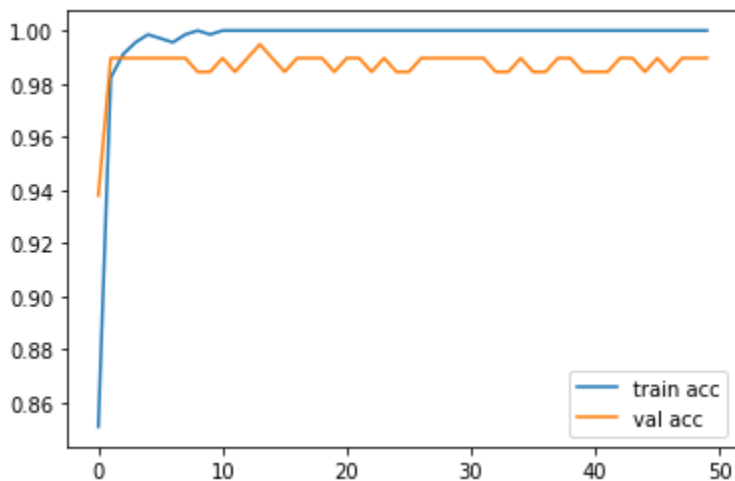


Figure 4. 30 VGG16 training and validation accuracy graph

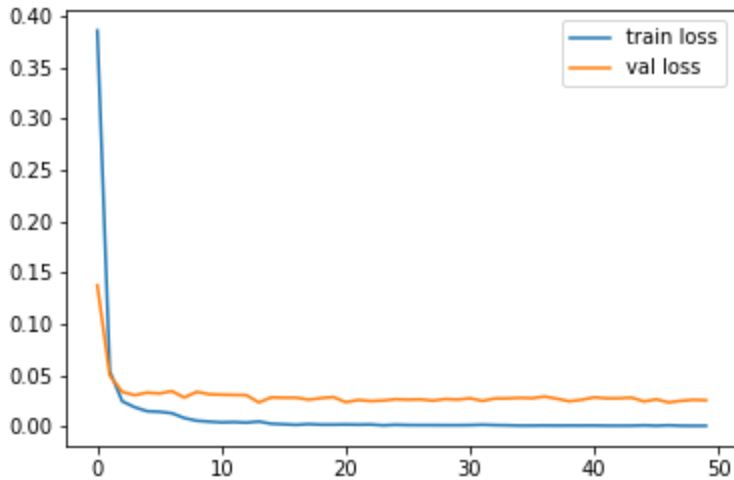


Figure 4. 31 VGG16 training and validation loss graph

As clearly depicted in figure 4.32 the confusion matrix shows the following results. There are 34 mealybug class images, 34 bacterial wilt class images, and 32 healthy class images in our test dataset. VGG16 predicted all of the test dataset images correctly with an accuracy of 99.98% and a loss of only 0.36%. The results of VGG16 model are astonishing, so we have used the VGG16 model to develop a more storage and time saving (which are important issues when working with low GPU) model.

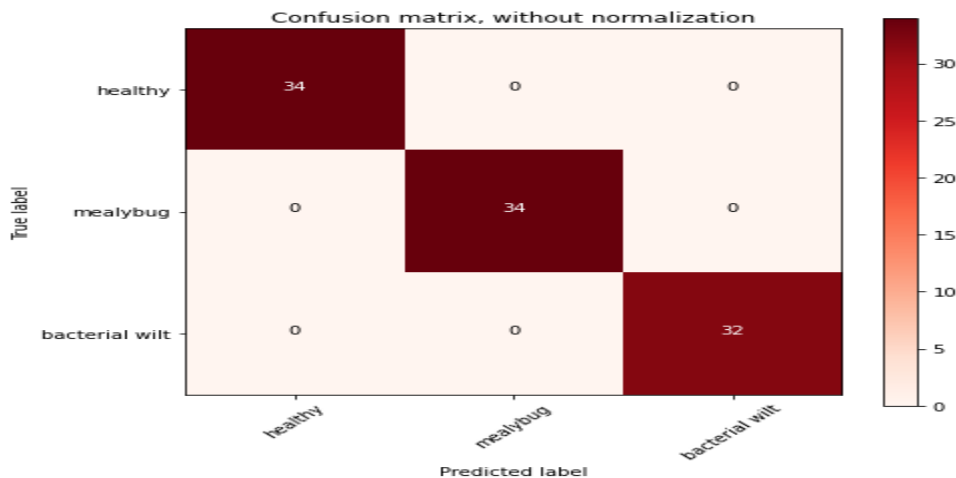


Figure 4. 32 VGG16 confusion matrix

4.2.6.3 Comparison with ResNet50 model

The performance (accuracy and loss value) of ResNet model is shown in figure 4.33, 4.34 and 4.35. ResNet achieves 95.72% training and 93.26 validation accuracy on our data. It has a training

loss of 14.16% and a validation loss of 20.26.% It takes 13 minutes to train the model, taking 15.6 seconds per epoch on average. The size of the model is also huge (93.9 MB).

```
Epoch 1/50
22/22 [=====] - 191s 7s/step - loss: 4.0699 - accuracy: 0.4225 - val_loss: 0.8671 - val_accuracy: 0.7098
Epoch 2/50
22/22 [=====] - 13s 570ms/step - loss: 1.0768 - accuracy: 0.6411 - val_loss: 0.5599 - val_accuracy: 0.8031
Epoch 3/50
22/22 [=====] - 13s 569ms/step - loss: 0.7615 - accuracy: 0.7001 - val_loss: 0.3733 - val_accuracy: 0.8497
Epoch 4/50
22/22 [=====] - 13s 567ms/step - loss: 0.6235 - accuracy: 0.7784 - val_loss: 0.4118 - val_accuracy: 0.8290
Epoch 5/50
22/22 [=====] - 13s 565ms/step - loss: 0.5434 - accuracy: 0.7917 - val_loss: 0.4227 - val_accuracy: 0.8394
Epoch 6/50
22/22 [=====] - 13s 565ms/step - loss: 0.3781 - accuracy: 0.8626 - val_loss: 0.3126 - val_accuracy: 0.8756
Epoch 7/50
22/22 [=====] - 13s 565ms/step - loss: 0.3883 - accuracy: 0.8479 - val_loss: 0.3183 - val_accuracy: 0.8860
Epoch 8/50
22/22 [=====] - 13s 567ms/step - loss: 0.3175 - accuracy: 0.8626 - val_loss: 0.2779 - val_accuracy: 0.9119

.
22/22 [=====] - 13s 565ms/step - loss: 0.1704 - accuracy: 0.9335 - val_loss: 0.2154 - val_accuracy: 0.9430
Epoch 44/50
22/22 [=====] - 13s 565ms/step - loss: 0.1416 - accuracy: 0.9572 - val_loss: 0.2026 - val_accuracy: 0.9326
Epoch 45/50
22/22 [=====] - 12s 563ms/step - loss: 0.3711 - accuracy: 0.8818 - val_loss: 0.9131 - val_accuracy: 0.7358
Epoch 46/50
22/22 [=====] - 13s 569ms/step - loss: 0.4825 - accuracy: 0.8479 - val_loss: 0.2602 - val_accuracy: 0.9016
Epoch 47/50
22/22 [=====] - 12s 562ms/step - loss: 0.1613 - accuracy: 0.9365 - val_loss: 0.2317 - val_accuracy: 0.9430
Epoch 48/50
22/22 [=====] - 13s 567ms/step - loss: 0.2659 - accuracy: 0.9129 - val_loss: 0.4891 - val_accuracy: 0.8705
Epoch 49/50
22/22 [=====] - 12s 562ms/step - loss: 0.2890 - accuracy: 0.9055 - val_loss: 0.2827 - val_accuracy: 0.9119
Epoch 50/50
22/22 [=====] - 13s 565ms/step - loss: 0.1970 - accuracy: 0.9261 - val_loss: 0.2134 - val_accuracy: 0.9430
```

Figure 4. 33 ResNet model training and validation accuracy and loss

The graphs below indicate the training and validation loss and accuracy curves of ResNet. Training and validation accuracy rises while training and validation loss reduces, as illustrated in the training loss and accuracy curve in figure 4.34 and 4.35. The graph reveals some evidence of overfitting throughout some epochs where the validation accuracy is better than training accuracy.

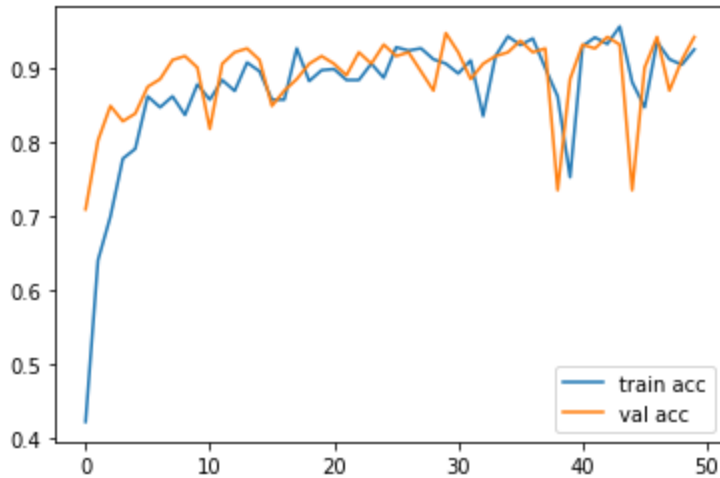


Figure 4. 34 ResNet model training and validation accuracy graph

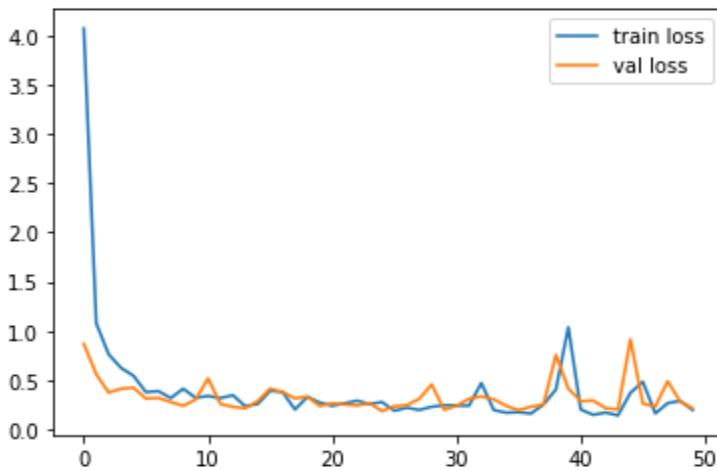


Figure 4. 35 ResNet model training and validation loss graph

As clearly depicted in figure 4.36 the confusion matrix shows the following results. There are 34 mealybug class images, 34 bacterial wilt class images, and 32 healthy class images in our test dataset. From 34 images found in healthy class it correctly predicted 31 of them incorrectly predicted 2 images as mealybug and 1 image as bacterial wilt. From 34 images found in mealybug class it correctly predicted all of the images. It also correctly predicted all of the images found in bacterial wilt class. Totally ResNet50 predicted well on our test dataset with an accuracy of 97% and a loss of 9.2%

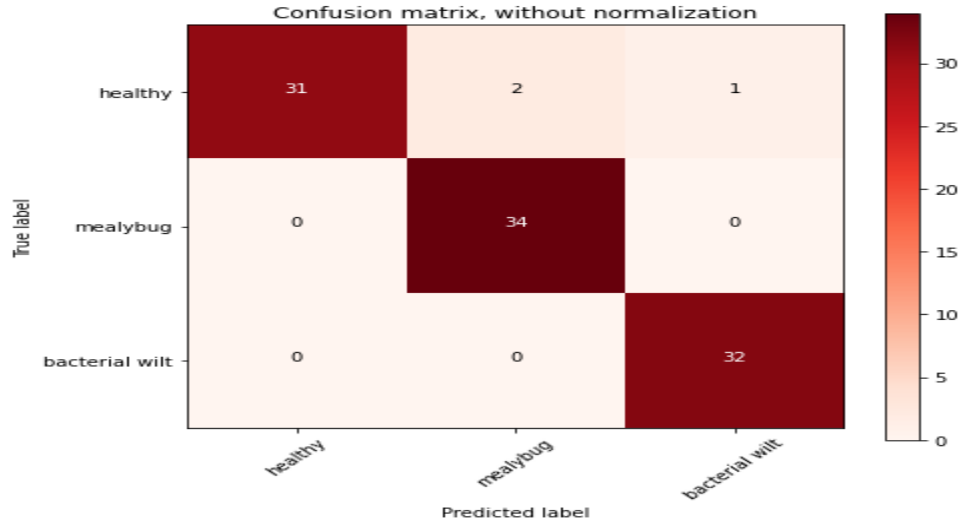


Figure 4. 36 ResNet model confusion matrix

4.2.6.4 Comparison with Inception model

The performance (accuracy and loss value) of Inceptionv3 model is shown in figure 4.37, 4.38 and 4.39. Inceptionv3 achieves 99.78% training and 99.48% validation accuracy on our data, as shown in figure 4.37. It has a training loss of 0.7% and a validation loss of 10.30%. It takes 23 minutes to train the model, taking 13.8 seconds per epoch on average. The size of the model is also huge (85.8 MB).

```

Epoch 1/100
22/22 [=====] - 182s 7s/step - loss: 3.2694 - accuracy: 0.7194 - val_loss: 0.1411 - val_accuracy: 0.9741
Epoch 2/100
22/22 [=====] - 12s 551ms/step - loss: 0.1157 - accuracy: 0.9808 - val_loss: 0.2560 - val_accuracy: 0.9637
Epoch 3/100
22/22 [=====] - 12s 547ms/step - loss: 0.0177 - accuracy: 0.9911 - val_loss: 0.1208 - val_accuracy: 0.9793
Epoch 4/100
22/22 [=====] - 12s 548ms/step - loss: 0.0075 - accuracy: 0.9970 - val_loss: 0.1179 - val_accuracy: 0.9845
Epoch 5/100
22/22 [=====] - 12s 548ms/step - loss: 0.0222 - accuracy: 0.9941 - val_loss: 0.1211 - val_accuracy: 0.9741
Epoch 6/100
22/22 [=====] - 12s 550ms/step - loss: 0.0273 - accuracy: 0.9911 - val_loss: 0.1003 - val_accuracy: 0.9896
Epoch 7/100
22/22 [=====] - 12s 549ms/step - loss: 0.0052 - accuracy: 0.9985 - val_loss: 0.0887 - val_accuracy: 0.9845
Epoch 8/100
22/22 [=====] - 12s 548ms/step - loss: 0.0235 - accuracy: 0.9911 - val_loss: 0.1241 - val_accuracy: 0.9741
Epoch 9/100

```

```

22/22 [=====] - 12s 553ms/step - loss: 0.0038 - accuracy: 0.9985 - val_loss: 0.2140 - val_accuracy: 0.9845
Epoch 92/100
22/22 [=====] - 12s 548ms/step - loss: 0.0177 - accuracy: 0.9970 - val_loss: 0.4483 - val_accuracy: 0.9793
Epoch 93/100
22/22 [=====] - 12s 547ms/step - loss: 0.0344 - accuracy: 0.9956 - val_loss: 0.7353 - val_accuracy: 0.9637
Epoch 94/100
22/22 [=====] - 12s 549ms/step - loss: 0.0787 - accuracy: 0.9941 - val_loss: 0.7587 - val_accuracy: 0.9689
Epoch 95/100
22/22 [=====] - 12s 546ms/step - loss: 9.5807e-04 - accuracy: 1.0000 - val_loss: 0.0432 - val_accuracy: 0.9948
Epoch 96/100
22/22 [=====] - 12s 545ms/step - loss: 2.8857e-06 - accuracy: 1.0000 - val_loss: 0.0516 - val_accuracy: 0.9896
Epoch 97/100
22/22 [=====] - 12s 545ms/step - loss: 0.0000e+00 - accuracy: 1.0000 - val_loss: 0.0563 - val_accuracy: 0.9896
Epoch 98/100
22/22 [=====] - 12s 545ms/step - loss: 3.5217e-09 - accuracy: 1.0000 - val_loss: 0.0568 - val_accuracy: 0.9896
Epoch 99/100
22/22 [=====] - 12s 548ms/step - loss: 0.0077 - accuracy: 0.9985 - val_loss: 0.0783 - val_accuracy: 0.9845
Epoch 100/100
22/22 [=====] - 12s 543ms/step - loss: 3.8739e-09 - accuracy: 1.0000 - val_loss: 0.1040 - val_accuracy: 0.9948

```

Figure 4. 37 Inceptionv3 training and validation accuracy and loss

The graphs below indicate the training and validation loss and accuracy curves of Inceptionv3. Training and validation accuracy rises while training and validation loss reduces, as illustrated in the training loss and accuracy curve in figure 4.38 and 4.39. The graph reveals that Inception performed very well on our dataset without overfitting and underfitting problems.

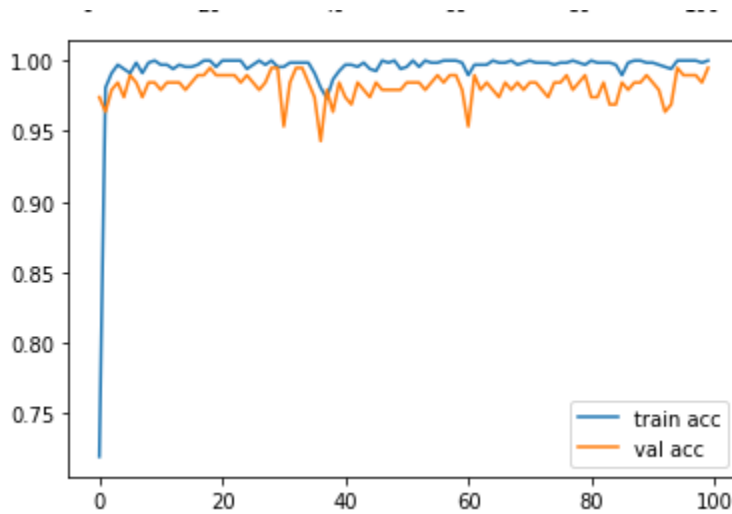


Figure 4. 38 Inceptionv3 training and validation accuracy graph

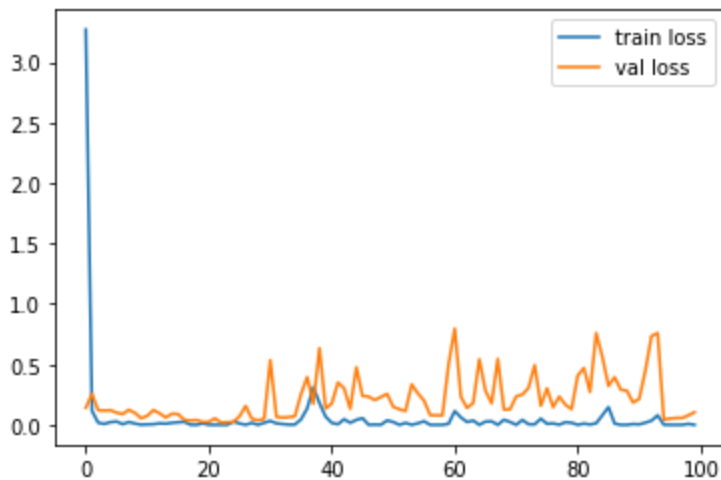


Figure 4. 39 Inceptionv3 training and validation loss graph

As clearly depicted in figure 4.40 the confusion matrix shows the following results. There are 34 mealybug class images, 34 bacterial wilt class images, and 32 healthy class images in our test dataset. From 34 images found in healthy class it correctly predicted 31 of them incorrectly predicted 2 images as mealybug and 1 image as bacterial wilt. From 34 images found in mealybug class it correctly predicted all of the images. It also correctly predicted all of the images found in bacterial wilt class. Totally Inceptionv3 predicted well on our test dataset with an accuracy of 97% and a loss of only 9.2%. As we can understand from the experimental results Inceptionv3 and ResNet50 almost have the same performance towards our test dataset.

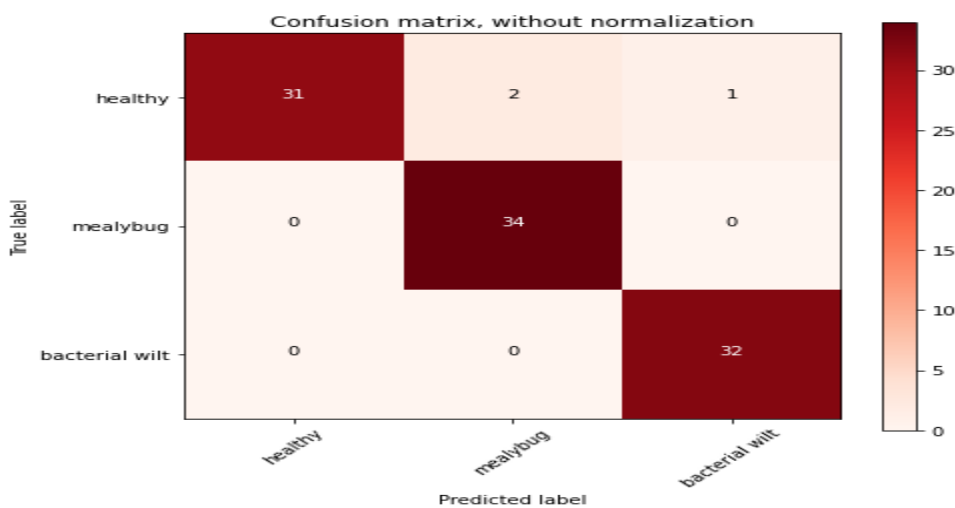


Figure 4. 40 Inceptionv3 confusion matrix

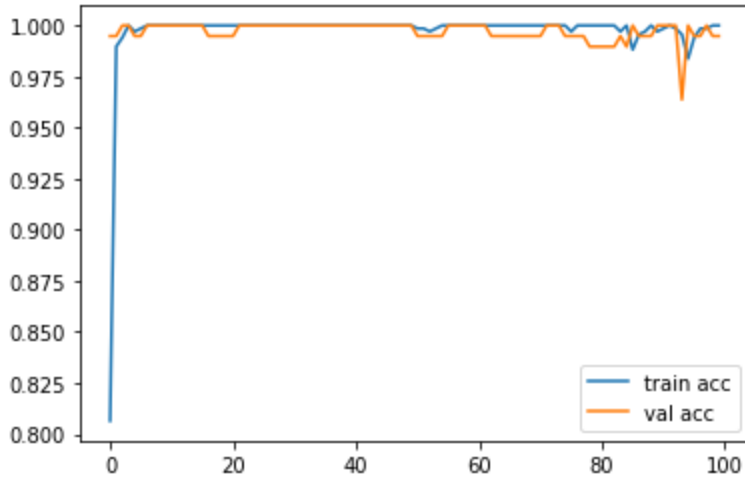


Figure 4. 42 DenseNet201 training and validation accuracy graph

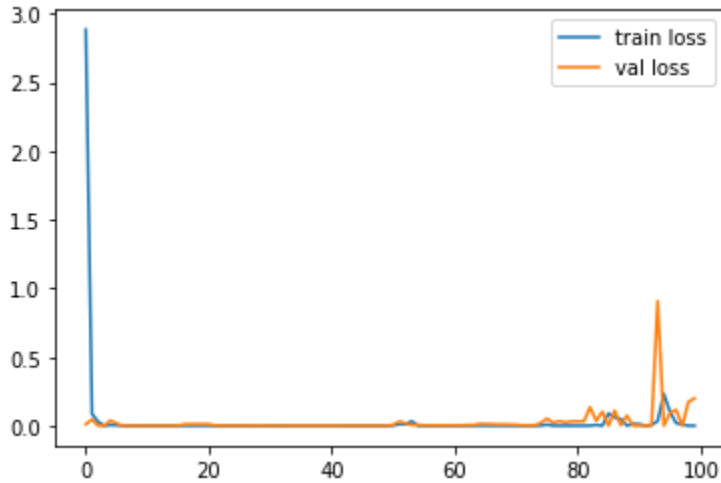


Figure 4. 43 DenseNet training and validation loss graph

As clearly depicted in figure 4.44 the confusion matrix shows the following results. There are 34 mealybug class images, 34 bacterial wilt class images, and 32 healthy class images in our test dataset. DenseNet21 predicted all of our images to correct classes. Totally DenseNet201 predicted perfectly on our test dataset with an accuracy of 100% and no loss at all, 0.0%. As we can understand from the experimental results DenseNet201 performed best towards our test dataset.

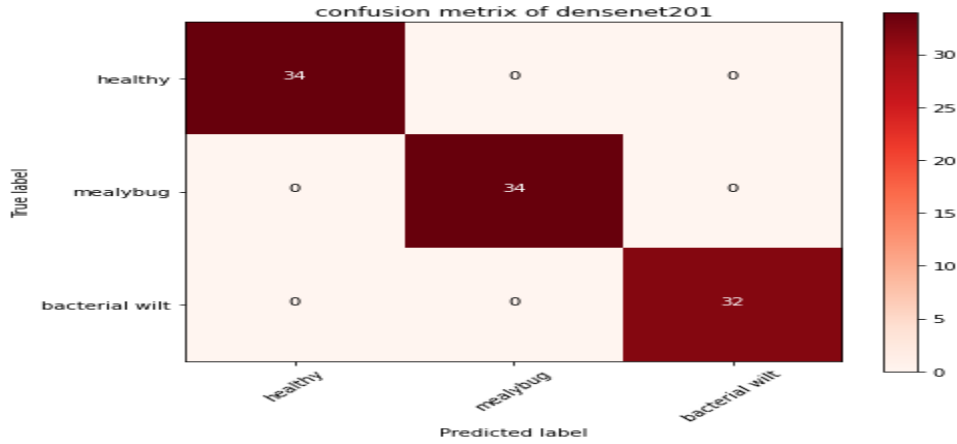


Figure 4. 44 DenseNet201 confusion matrix

4.2.6.7 Comparison with EfficientNet

The performance (accuracy and loss value) of EfficientNetB3 model is shown in figure 4.45, 4.46 and 4.47. DenseNet201 achieves 48.01% training and 52.01% validation accuracy on our data, as shown in figure 4.45. It has a training loss of 27.10% and a validation loss of 31.78%. It takes 41 minutes to train the model, taking 24.6 seconds per epoch on average.

```

Epoch 1/100
22/22 [=====] - 344s 14s/step - loss: 21.5459 - accuracy: 0.3781 - val_loss: 9.0878 - val_accuracy: 0.3834
Epoch 2/100
22/22 [=====] - 21s 966ms/step - loss: 9.4962 - accuracy: 0.3250 - val_loss: 11.7582 - val_accuracy: 0.4611
Epoch 3/100
22/22 [=====] - 21s 968ms/step - loss: 4.5758 - accuracy: 0.3840 - val_loss: 4.1375 - val_accuracy: 0.3368
Epoch 4/100
22/22 [=====] - 21s 968ms/step - loss: 3.1890 - accuracy: 0.3575 - val_loss: 4.8317 - val_accuracy: 0.3420
Epoch 5/100
22/22 [=====] - 21s 961ms/step - loss: 9.1579 - accuracy: 0.3412 - val_loss: 6.4376 - val_accuracy: 0.3368
Epoch 6/100
22/22 [=====] - 21s 959ms/step - loss: 4.5202 - accuracy: 0.3604 - val_loss: 3.6806 - val_accuracy: 0.5026
Epoch 7/100
22/22 [=====] - 21s 960ms/step - loss: 5.6652 - accuracy: 0.3264 - val_loss: 6.7752 - val_accuracy: 0.3420
Epoch 8/100
22/22 [=====] - 21s 963ms/step - loss: 6.1637 - accuracy: 0.3516 - val_loss: 3.1156 - val_accuracy: 0.5803
Epoch 9/100

Epoch 91/100
22/22 [=====] - 21s 965ms/step - loss: 2.0743 - accuracy: 0.4845 - val_loss: 2.2444 - val_accuracy: 0.3990
Epoch 92/100
22/22 [=====] - 21s 965ms/step - loss: 2.7109 - accuracy: 0.4801 - val_loss: 3.1781 - val_accuracy: 0.5285
Epoch 93/100
22/22 [=====] - 21s 957ms/step - loss: 2.8333 - accuracy: 0.4476 - val_loss: 1.6519 - val_accuracy: 0.5233
Epoch 94/100
22/22 [=====] - 21s 962ms/step - loss: 3.6570 - accuracy: 0.4225 - val_loss: 4.0801 - val_accuracy: 0.4663
Epoch 95/100
22/22 [=====] - 21s 961ms/step - loss: 3.7363 - accuracy: 0.4446 - val_loss: 1.9273 - val_accuracy: 0.6010
Epoch 96/100
22/22 [=====] - 21s 960ms/step - loss: 5.2088 - accuracy: 0.4239 - val_loss: 6.7811 - val_accuracy: 0.5026
Epoch 97/100
22/22 [=====] - 21s 963ms/step - loss: 7.7484 - accuracy: 0.3722 - val_loss: 4.8337 - val_accuracy: 0.3472
Epoch 98/100
22/22 [=====] - 21s 961ms/step - loss: 7.4681 - accuracy: 0.4284 - val_loss: 5.4645 - val_accuracy: 0.4404
Epoch 99/100
22/22 [=====] - 21s 958ms/step - loss: 2.6660 - accuracy: 0.4446 - val_loss: 3.5240 - val_accuracy: 0.4404
Epoch 100/100
22/22 [=====] - 21s 958ms/step - loss: 3.5595 - accuracy: 0.4165 - val_loss: 1.9300 - val_accuracy: 0.4404

```

Figure 4. 45 EfficientB3 training and validation accuracy and loss

The graphs below indicate the training and validation loss and accuracy curves of EfficientNetB3. Training and validation accuracy very slowly rises but never gets better than 50% even in the final epochs. Which is the same to training and validation loss, as illustrated in the training loss and accuracy curve in figure 4.46 and 4.47. The graph reveals that EfficientNetB3 performed very badly on our dataset. Totally, EfficientNet model is the worst model experimented trained on our dataset when it is compared with the other state-of-the-art models.

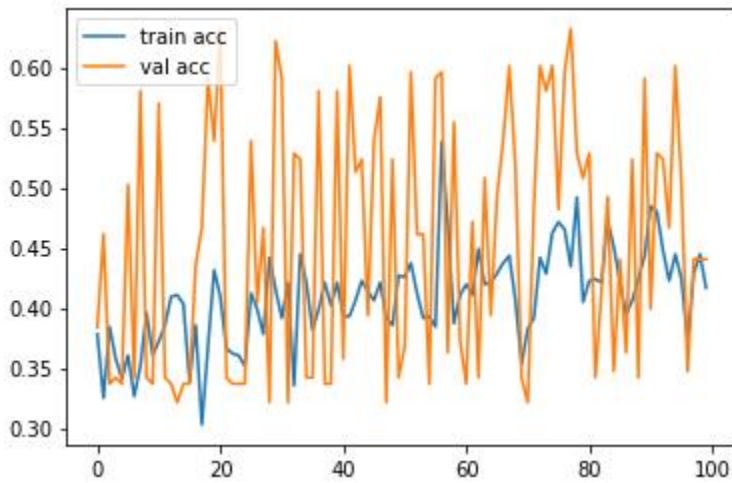


Figure 4. 46 EfficientNetB3 training and validation accuracy graph

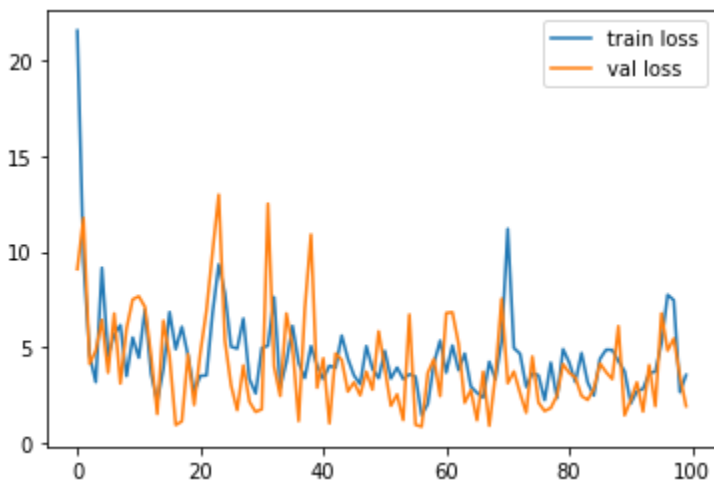


Figure 4. 47 EfficientNetB3 training and accuracy loss graph

As clearly depicted in figure 4.48 the confusion matrix shows the following results. There are 34 mealybug class images, 34 bacterial wilt class images, and 32 healthy class images in our test dataset. From our 34 mealybug images it predicted all them correctly to the mealybug class, from 34 images of bacterial wilt, EfficientNetB3 predicted all of the images to the mealybug class, from 32 images of healthy class, EfficientNetB3 predicted all of the images to the mealybug class. Totally, EfficientNetB3 performed less when compared with our model and the other state-of-the-art models on our test dataset with an accuracy of 34% and a loss of 64%

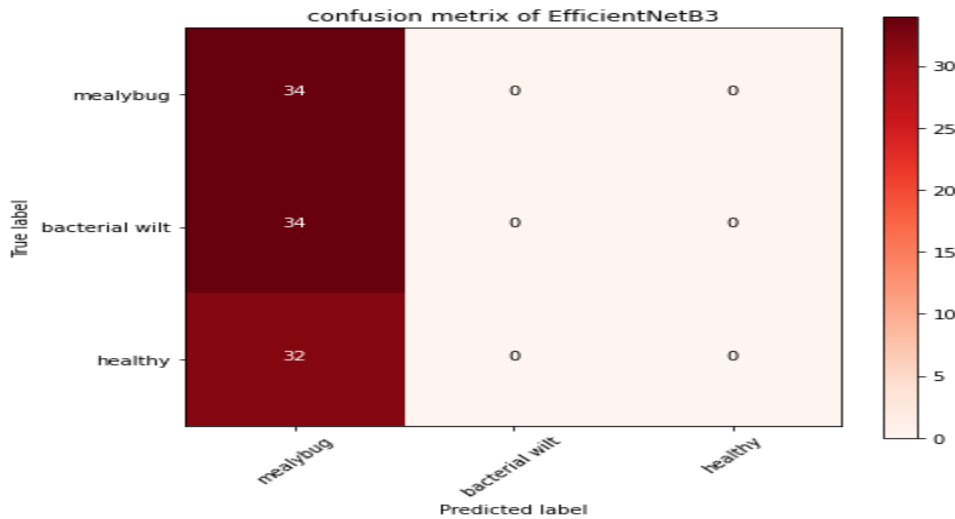


Figure 4. 48 EfficientNetB3 confusion matrix

4.2.7 Comparing our model with machine learning classifiers

CNN is designed to process multidimensional data like image. The input data is initially forwarded to A CNN feature extraction network, and then the resultant features are forwarded to a classifier network. Traditional machine learning algorithms like SVM (support vector machine), Random Forest Classifier, and XBoost are used for classification this phase.

4.2.7.1 Comparison with Random Forest Classifier

Random forest classifier registered classification accuracy of 97.92%. as depicted in figure 4.49, Random Forest missed 4 images from a total of 193 images.

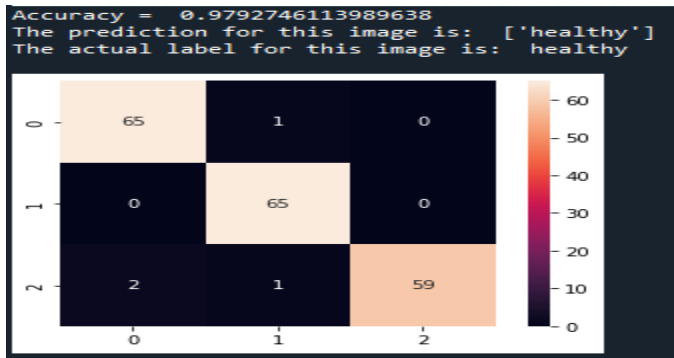


Figure 4. 49 Random Forest Classifier

4.2.4.2 Comparison with XBoost Classifier

Random forest classifier registered classification accuracy of 97.40%. As depicted in figure 4.50, XBoost miss-classified 5 images from a total of 193 images.

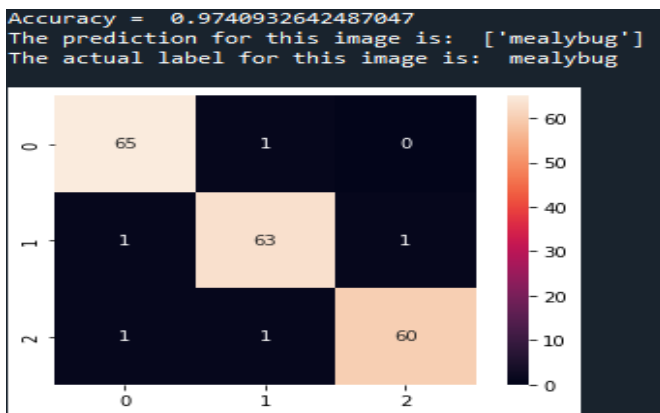


Figure 4. 50 XBoost Classifier

4.2.4.3 Comparison with Support Vector Machine

Support Vector Machine registered classification accuracy of 97.92%. As depicted in figure 4.51, Support Vector Machine miss-classified 4 images from a total of 193 images.

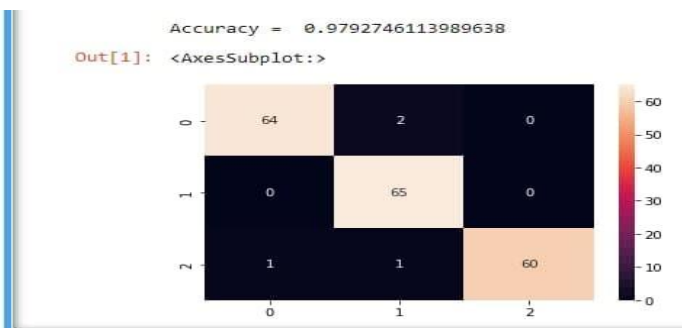


Figure 4. 51 SVM classification

4.3 Summary of comparisons between proposed model and state-of-the-art models

Table 4. 1 Summary of comparisons

Models	Training accuracy, loss (in %)	Validation accuracy, loss	Number of misses during prediction	Model size (in Mb)	Training time (in minutes)
Our model	99.68, 0.0092	98.87, 0.0303	0	21	8 minutes
Our model (without segmentation)	99.26, 0.0238	97.93, 0.0738	0	21	10.8
AlexNet50	97.34, 0.0854	94.30, 0.2117	8	321	24
VGG16	99.89, 0.001	98.96, 0.0254	0	57.1	20
ResNet50	95.72, 0.1416	93.26, 0.2026	3	93.9	13
Inceptionv3	99.78, 0.007	99.48, 0.1030	3	85.8	23
DenseNet201	99.88, 0.008	99.48, 0.1155	0	75	27
EfficientNetB3	48.01, 2.710	52.01, 3.178	66	250	41

4.4 Discussions

With the exception of EfficientNet, all of the models were evaluated on a second dataset that was not viewed during the model's training and yielded positive results. Model performance while training is measured using classification accuracy metrics, while model performance during testing is measured using the Confusion matrix. Overall, our model is as accurate as the finest pre-trained models, but it is considerably smaller and requires less training time.

The dataset that we utilized to train the model, i.e., the images in the dataset are easily classified with human eyes, is the fundamental reason that the proposed model and most state-of-the-art models offer good results. Another reason our proposed model performs well is that we used smaller filters in the network's convolution layer. The use of lower convolution sizes aids in the identification of extremely small features that are utilized to distinguish between the input image and the output image, and the risk of losing a crucial feature is greatly reduced.

This chapter describes in detail the experimental evaluation of the suggested methodology for automatic detection of Enset disease and pests. The dataset used and the proposed model's

implementation are detailed in depth. In addition, the outcomes of the tests are reported and compared to current models. Our model had a high level of classification accuracy and properly predicted all of the test image datasets. When compared with previous researches on onset disease identification, our model registered better accuracy and less training time. In comparison to state-of-the-art models, it was also trained faster and weighs less.

Table 4. 2 summary of comparison between our model and other models from related works

Researcher Name	Test accuracy
Y. kibru	97.86%
Kibru A. and Getahun T.	94.04%
G. Selvaraj	97.36%
Our model	99.97%

CHAPTER FIVE

CONCLUSION AND FUTURE WORKS

5.1 Conclusions

Bacterial Wilt and root mealybugs are wreaking havoc on Enset production, reducing the quantity and quality of the crop. Robots, temperature and moisture sensors, aerial photos, GPS technology, and computer visions will all be used in future agriculture. Planting, harvesting, improved weather analysis, weeding, and plant health detection and monitoring are all examples of computer vision-artificial intelligence (AI) models. To begin the future farming approach of Enset, we built this Enset disease detection using computer vision utilizing CNN.

CNN was highlighted as a significant contributor and a recent research topic for the automatic detection of Enset diseases and pests in this study. CNNs are local invariant, which means they may be used to detect a specific symptom of sickness in any image, independent of its spatial location. A CNN computes the same picture characteristics in all spatial areas.

Following a thorough study of the literature and relevant research, we discovered that detection of Enset diseases and pests receives little attention and remains an unsolved problem. Using images of normal and diseased Enset, a deep CNN system for detecting Enset diseases and pests was constructed. As a result, reliance on human specialists' talent and experience is lessened.

The importance of the proposed system is that it will result in a shift in detection progress in terms of accuracy and efficiency. In the detection of Enset diseases and pests, the system has achieved a 99.68% training accuracy, 98.87% validation accuracy, and 99.67% testing accuracy. As a result, the established model can assist specialists, investors, and farmers in accurately detecting diseases and pests before they are spread. The results of our experiment show that the suggested CNN model may considerably aid in the accurate detection of Bacterial Wilt, root mealybug, and healthy Enset images with minimal computing effort and limited number of images.

5.2 Contributions

Segmentation: We present the best approach for segmenting only the ROI portion of the Enset leaves and root mealybugs, so that the model to be created can learn solely on the ROI portion of the dataset.

CNN development techniques: we present the best CNN development techniques by experimenting on different optimization techniques, batch sizes, epochs and split sizes.

Improvements in performance and efficiency: the suggested model performs better in terms of accuracy, loss, training time, and model size. In terms of precision, we have a training accuracy of 99.68% and a testing accuracy of 98.87%. In comparison to earlier versions, the model is much lighter and trains much faster. The ROI segmentation is also utilized to improve the model's performance.

5.3 future works

The deep CNN model suggested in this study can be utilized to detect problems such as bacterial wilt and mealybugs in bananas. Other diseases seen in Enset include sigatoka, leaf speckle, and Cordana. We were, however, detecting healthy, bacterial wilt, and root mealybug due to a lack of data (picture) in the remaining situations. Because the data we used for the aforementioned situations was also modest, more data may be obtained to further train the model. Furthermore, the possibility of automatically determining the severity of the disease to assist farmers can be examined.

Future agriculture will use sophisticated technologies such as autonomous drones and mobile applications. We recommend developing mobile application that automatically detects the diseases in real time. We also recommend to further study on developing autonomous drones that can detect the diseases and take measures accordingly.

References

- [1] A. M. and H. B. , "Distribution and Management of Bacterial Wilt (*Xanthomonas Campestris* pv. *Musacearum*) of enset in Ethiopia," vol. 7, no. 2, 2020, 26 May 2020.
- [2] K. a. S. Alemu, "Enset in north Omo region," *Farmer's Res. Project Technical Pamphlet*, vol. 1, p. 49, 2019.
- [3] T. M. F. C. C. R. P. Project, "Integrated Management of Bacterial Wilt of Enset (*Ensete ventricosum* (Welw.) Cheesman) caused by *Xanthomonas campestris* pv. *musacearum* in Ethiopia," McKnight foundation, Hawassa, October 2013.
- [4] G. Intelligence, "The mobile economy Africa 2016," GSM association, London, 2016.
- [5] R. C. G. • R. E. Woods, *Digital Image Processing*, Digital Image Processing: Pearson, 2018.
- [6] A. F. B. G. a. K. K. Addis T, "Biology of the Enset Root Mealybug, *Cataenococcus ensete* and its Geographical Distribution in Southern Ethiopia," *Journal of Applied*, vol. 8, no. 1, p. 251–260, 2008.
- [7] T. A. T. A. A. T. A. T. A. a. G. B. Ferdu Azerefegne, *An IPM guide for Enset root mealybug (Cataenococcus ensete) in Enset production*, Kampala (UGA): Bioversity Internationa, 2009.
- [8] H. Tian, T. Wang, Y. Liu, X. Qiao and Y. Li, "Computer Vision Technology in Agricultural Automation," *Information Processing in Agriculture*, vol. 7, 2019.
- [9] F. A. a. G. B. TEMESGEN ADDIS, "DENSITY AND DISTRIBUTION OF ENSET ROOT MEALYBUGS ON ENSET," *African Crop Science Journal*, vol. 14, no. 1, pp. 67-74, 2014.
- [10] G. D. M. J. K. S. P. V. L. M. A. O. W. Blomme, "Bacterial diseases of bananas and enset: current state of knowledge and integrated approaches toward sustainable management," *Frontiers in Plant Science*, vol. 8, no. 1664-462X, p. 25, 2017.
- [11] Y. K. AFEWORK, "DEVELOPING BACTERIAL WILT DETECTION MODEL ON ENSET CROP USING A DEEP LEARNING APPROACH," *DEVELOPING BACTERIAL WILT DETECTION MODEL ON ENSET CROP USING A DEEP LEARNING APPROACH*, no. AAU, 2019.
- [12] G. T. Kibru Abera Ganore, "Ethiopian Enset Diseases Diagnosis Model Using Image Processing and Machine Learning Techniques," *International Journal of Intelligent Information Systems*, vol. 9, no. 1, pp. 1-5, 2020.
- [13] F. Sultana, A. Sufian and P. Dutta, "Advancements in Image Classification using Convolutional Neural Network," in *International Conference on Research in Computational Intelligence and Communication Networks*, 2018.

- [14] A. & S. P. Tsegaye, "ANALYSIS OF ENSET (ENSETE VENTRICOSUM) INDIGENOUS PRODUCTION METHODS AND FARM-BASED BIODIVERSITY IN MAJOR ENSET-GROWING REGIONS OF SOUTHERN ETHIOPIA," *Cambridge University Press*, vol. 38, no. 3, pp. 291-315, 2002.
- [15] M. Aytenfsu and B. Haile, "Distribution and Management of Bacterial Wilt (*Xanthomonas Campestris* pv. *Musacearum*) of Enset (*Ensete Ventricosum*) in Ethiopia: a Review," *International Journal of Research in Agriculture and Forestry*, vol. 7, no. 2, pp. 40-53, 2020.
- [16] G. Z. A. Institute, "Prevention of Bacterialwilt of enset," The Mcknight Foundation, Butajira, 2017.
- [17] J. S. B. Zerihun Yemataw, "The Distribution of Enset Pests and Pathogens and a 1 Genomic Survey of Enset *Xanthomonas* Wilt," *Bioversity International*, pp. 12-16, 2020.
- [18] T. A., "Insect and mite pests of horticultural and miscellaneous plants in Ethiopia," p. 115, 2013.
- [19] A. T., "Biology of Enset root mealybug (*Cataenococcus ensete* Williams and Matile-Ferrero) (Homoptera: Pseudococcidae) and its geographical distribution in southern Ethiopia," *MSc thesis*, p. 81, 2005.
- [20] B. D. a. R. V. Driesche, "Encapsulation rates of three encyrtid parasitoids by three mealybug species (Homoptera: Pseudococcidae) found commonly as pests in commercial," vol. 22, pp. 147-150, 2001.
- [21] V. H. a. R. B. M. Sonka, "Image pre-processing in image processing, analysis and machine vision," pp. 56-111, 2016.
- [22] E. W. Richard and C. G. Rafeal, *Digital image processing*, london: Pearson, 2011.
- [23] R. C. G. • R. E. Woods, *Digital Image Processing*, New York: Pearson, 2018.
- [24] N. Hussain Dar, *Image segmentation Techniques and its application*, ResearchGate, 2020.
- [25] A. Janwale, "Plant Leaves Image Segmentation Techniques: A Review," *INTERNATIONAL JOURNAL OF COMPUTER SCIENCES AND ENGINEERING*, vol. 5, pp. 147-150, 2017.
- [26] N. M. Zaitoun and M. J. Aqel, "Survey on Image Segmentation Techniques," in *International Conference on Communication, Management and Information Technology (ICCMIT)*, 2015.
- [27] S. Chris and T. Breckon, *Fundamentals of Digital Image Processing*, London: John Wiley & Sons Ltd, 2011.
- [28] T. N. ME, "Efficient Deep Learning Approaches for Health Informatics," *Deep Learning and Parallel Computing Environment for Bioengineering Systems*, pp. 123-137, 2019.
- [29] S. Avalos and J. Ortiz, *Geological modeling using a recursive convolutional neural networks approach*, 2019.

- [30] T. M. Mitchell, machine learning, Newyork, 2014.
- [31] Iftikharul Islam, Sujit Kumar Mondal and Bappa Sarkar, "Deep Learning based object detection from image using Convolutional Neural Network," vol. 8, no. 1356, May 2020.
- [32] A. M. Ajay Shrestha, "Review of Deep Learning Algorithms and Review of Deep Learning Algorithms and," *open access journal*, p. 99, 2019.
- [33] X. L. Y. Z. W. J. Weilong MO, "Image recognition using convolutional neural network combined with ensemble learning algorithm," *Journal of Physics: Conference Series*, vol. 1237, no. 2, 2019.
- [34] Y. B. A. C. Ian Goodfellow, Deep Learning, MIT Press, 2016.
- [35] T. Gupta and K. Raza, "Optimizing Deep Feedforward Neural Network Architecture: A Tabu Search Based Approach," *Neural Processing Letters*, vol. 51, 2020.
- [36] J. Schmidhuber, "Deep learning in neural networks: An overview," *Neural Networks*, vol. 61, pp. 85-117, 2015.
- [37] G. E. H. Vinod Nair, "Rectified Linear Units Improve Restricted Boltzmann Machines," in *ICML*, Toronto, Canada, 2010.
- [38] A. F. Agarap, "Deep Learning using Rectified Linear Units (ReLU)," 2018.
- [39] A. Oken, "An Introduction To and Applications of Neural Networks," 2017.
- [40] M. K. Gergely Friedl, "Population and Gradient Based Optimization Techniques, a Theoretical Overview," *Acta Technica Jaurinensis*, vol. 7, no. 4, p. 378–387, 2014.
- [41] E. F. M. A. H. C. J. P. Ian H. Witten, Practical Machine Learning Tools and Techniques, San Francisco, USA: Morgan Kaufmann Publishers Inc., 2016.
- [42] S. S. Siddharth Sharma and A. Athaiya, "ACTIVATION FUNCTIONS IN NEURAL," *International Journal of Engineering Applied Sciences and Technology*, vol. 4, no. 12, pp. 310-316, 2020.
- [43] X. Ying, "Ensemble learning," 2014.
- [44] A. Mikołajczyk and M. Grochowski, "Data augmentation for improving deep learning in image classification problem," in *2018 International Interdisciplinary PhD Workshop (IIPhDW)*, 2018.
- [45] S. C. a. G. A. a. S. V. a. M. M. D. Wong, "Understanding Data Augmentation for Classification: When to Warp?," in *2016 International Conference on Digital Image Computing: Techniques and Applications (DICTA)*, 2016.
- [46] Y. L. a. X. W. a. Z. M. a. W. C. Tan, "Data Augmentation for ML-driven Data Preparation and Integration," *Proc. VLDB Endow.*, vol. 14, pp. 3182-3185, 2021.

- [47] L. Prechelt, "Early Stopping - But When?," 2000.
- [48] N. H. G. K. A. S. I. a. S. R. Srivastava, "Dropout: A simple way to prevent neural networks from overfitting," *Journal of Machine Learning Research*, vol. 15, p. 1929–1958, 2014.
- [49] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever and R. Salakhutdinov, "Dropout: A Simple Way to Prevent Neural Networks from," *Journal of Machine Learning Research*, vol. 15, pp. 1929-1958, 2014.
- [50] X. G. Haibing Wu, "Towards Dropout Training for Convolutional Neural Networks," *Neural Networks: the official journal of the international neural network society*, vol. 71, pp. 1-10, 2015.
- [51] R. Sun, "Optimization for Deep Learning: An Overview," *Journal of the Operations Research Society of China*, vol. 8, no. 1, p. 249–294, 2020.
- [52] H. Shaziya, "A Study of the Optimization Algorithms in Deep Learning," *International Conference on Inventive Systems and Contro*, 2020.
- [53] M. C. M. a. M. Hein, "Variants of RMSProp and Adagrad with logarithmic regret bounds," *Proceedings of the 34th International Conference on Machine Learning*, vol. 70, p. 2545–2553, 2017.
- [54] B. J. Kingma Diederik, "Adam: A Method for Stochastic Optimization," *International Conference on Learning Representations*, 2014.
- [55] S. I. a. C. Szegedy, "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift," *ArXiv*, 2015.
- [56] M. Hamouda, K. Etabaa and M. Bouhlel, "Framework for Automatic Selection of Kernels based on Convolutional Neural Networks and CkMeans Clustering Algorithm," *International Journal of Image and Graphics*, vol. 19, 2019.
- [57] T. Bezdan and N. Bacanin, "Convolutional Neural Network Layers and Architectures," in *Sinteza 2019*, 2019.
- [58] S. A. S. F. C. A. D. D. Ghosh Anirudha, "Fundamental Concepts of Convolutional Neural Network," pp. 519-567, 2020.
- [59] S. Indolia, A. K. Goswami and S. Mishra, "Conceptual Understanding of Convolutional Neural Network - A Deep Learning Approach," *Procedia Computer Science*, vol. 132, pp. 679-688, 2018.
- [60] T. A. M. S. A.-Z. S. ALBAWI, "Understanding of a Convolutional Neural Network," *Understanding of a Convolutional Neural Network*, 2017.
- [61] G. Anirudha, S. Farhana, A. Sufian and C. Amlan, "Fundamental Concepts of Convolutional Neural Network," in *Recent Trends and Advances in AI and IOT*, 2020, pp. 519-567.

- [62] S. S. Basha, S. R. Dubey, V. Pulabaigari and S. Mukherjee, "Impact of Fully Connected Layers on Performance of Convolutional Neural Networks for Image Classification," *Neurocomputing*, vol. 378, 2019.
- [63] T. Bezlan and N. Becanin, "Convolutional Neural Network Layers and Architectures," in *Sinteza 2019 Project*, 2019.
- [64] Y. Lecun, L. Bottou, Y. Bengio and P. Haffner, "Gradient-Based Learning Applied to Document Recognition," *Proceedings of the IEEE*, vol. 86, pp. 2278-2324, 2001.
- [65] I. S. a. G. E. H. A. Krizhevsky, "Imagenet classification with deep convolutional neural networks," *Advances in Neural Information Processing Systems*, vol. 25, p. 1097–1105, 2012.
- [66] K. S. a. A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *CoRR*, pp. 1409-1556, 2014.
- [67] T. Srikanth, "Transfer Learning using VGG16 with Deep CNN for Classifying Images," *International Journal of Scientific and Research Publications*, vol. 9, 2019.
- [68] W. L. Y. J. P. S. S. R. D. A. D. E. V. V. C. Szegedy, "Going deeper with convolutions," in *In The IEEE Conference on Computer*, 2015.
- [69] X. Z. S. R. a. J. S. K. He, "Deep residual learning for image recognition," in *In The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*,, 2016.
- [70] Z. L. a. K. Q. W. G. Huang, "Densely connected convolutional networks," in *CoRR, abs*, 2016.
- [71] B. S. Basavaraj Tigadi, "Banana Plant Disease Detection and Grading Using Image Processing," vol. 6, no. 6, 2016.
- [72] S. P. Deenan and J. SatheeshKumar, "Study on Banana Leaf Disease Identification Using Image Processing Methods," vol. 2, no. 2, 2014.
- [73] M. Selvaraj, A. Vergara, H. Ruiz, N. Safari, S. Elayabalan, W. Ocimati and G. Blomme, "AI-powered banana diseases and pest detection," *Plant Methods*, vol. 15, no. 1, 2019.
- [74] Y. Kibru, "DEVELOPING BACTERIAL WILT DETECTION MODEL ON ENSET CROP USING A DEEP LEARNING APPROACH," Oct, 2019.
- [75] D. J. Harlan, "STEM research handbook," no. USA:NSTA Press, 2011.
- [76] V. Nasteski, "An overview of the supervised machine learning methods," *HORIZONS.B*, vol. 4, pp. 51-62, 2017.
- [77] A. Ghosh, A. Sufian, F. Sultana, A. Chakrabarti and D. De, in *Fundamental Concepts of Convolutional Neural Network*, 2020, pp. 519-567.

- [78] R. Patel and S. Patel, "A Comprehensive Study of Applying Convolutional Neural Network for Computer Vision," *International Journal of Advanced Science and Technology*, vol. 6, pp. 2161-2174, 2020.
- [79] M. Browne and S. Ghidary, *Convolutional Neural Networks for Image Processing: An Application in Robot Vision*, Researchgate, 2003.
- [80] R. N. M. D. R. Yamashita, "Convolutional neural networks: an overview and application in radiology," *Insights into Imaging*, vol. 9, no. 4, p. 611–629, 2018.
- [81] K. D. K. a. M. S. Richard, "Optimally Splitting Cases for Training and Testing High Dimensional," *BMC medical genomics*, vol. 4, no. 31, pp. 1-8, 2011.
- [82] M. G. Forero, J. Ávila-Navarro and S. Herrera-Rivera, "New Method for Extreme Color Detection in Images," *Pattern Recognition*, pp. 89-97, 2020.
- [83] S. C. Wong, A. Gatt, V. Stamatescu and M. D. McDonnell, "Understanding data augmentation for classification," in *IEEE*, Gold Coast, QLD, Australia, 2016.
- [84] C. S. a. T. M. Khoshgoftaar, "A survey on Image Data Augmentation," *Journal of Big Data*, vol. 6, 2019.
- [85] V. J. A. M. Neha Sharma, "An Analysis Of Convolutional Neural Networks For Image Classification," *Procedia Computer Science*, vol. 132, no. 1, pp. 377-384, 2018.
- [86] J. Stastny, "A Brief Introduction to Image Pre- Processing for Object Recognition," in *Conference: ICSC - International Conference on Soft Computing Applied in Computer and Economic Enviroment. Kunovice 2007*, Mendel University in Brno, January 2007.
- [87] S. S. a. N. S. S. S. H. H., "A survey on image analysis techniques in agricultural product," *indian journal of science and technology*, vol. 9, 2016.
- [88] Benyamin Ghojogh, Maria Samad, Sayema Mashhadi and Tania Kapoor, *Feature Selection and Feature Extraction in Pattern Analysis: A Literature Review*, india: researchgate, 2019.
- [89] K. S. Krishna and S. Akansha, *A Study Of Image Segmentation Algorithms For Different Types Of Images*, India: IJCSI International Journal of Computer Science Issues, 2010.
- [90] A. Anand, *Image classification*, 2017.
- [91] D. Rajesh G, *Role of Artificial Neural Networks (ANN) in Image Processing*, India: International Journal of Innovative Research in Computer , 2016.
- [92] M. Rouse, "whatis.techtarget.com," 18 Dec 2016. [Online]. Available: <https://whatis.techtarget.com/definition/unsupervised-learning>. [Accessed 1 Feb 2020].

[93] R. S. S. a. A. G. Barto, Reinforcement Learning: an introduction, London, England: The MIT Press, 2015.

[94] J. J. a. Y. S. F. Li, "CS231n: Convolutional Neural Networks for Visual," Stanford University, 2018. [Online]. Available: <http://cs231n.stanford.edu/index.html>. [Accessed 12 07 2021].