

# Corn Leaf Disease Detection Using Deep Convolution Neural Network

by

Rawhatur Rabbi

19101422

Mohammad Yasin Arefin

19101317

Iffat Fahmida Turna

19101542

Zahra Zannat

19101559

A thesis submitted to the Department of Computer Science and Engineering  
in partial fulfillment of the requirements for the degree of  
B.Sc. in Computer Science

Department of Computer Science and Engineering  
School of Data and Sciences  
Brac University  
January 2023

© 2023. Brac University  
All rights reserved.

# Declaration

It is hereby declared that

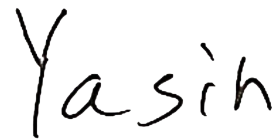
1. The thesis submitted is my/our own original work while completing degree at Brac University.
2. The thesis does not contain material previously published or written by a third party, except where this is appropriately cited through full and accurate referencing.
3. The thesis does not contain material which has been accepted, or submitted, for any other degree or diploma at a university or other institution.
4. We have acknowledged all main sources of help.

**Student's Full Name & Signature:**



---

Rawahatur Rabbi  
19101422



---

Mohammad Yasin Arefin  
19101317



---

Iffat Fahmida Turna  
19101542



---

Zahra Zannat  
19101559

# Approval

The thesis titled “Corn Leaf Disease Detection using Deep Convolution Neural Network” submitted by

1. Rawahatur Rabbi ( 19101422 )
2. Mohammad Yasin Arefin ( 19101317 )
3. Iffat Fahmida Turna ( 19101542 )
4. Zahra Zannat ( 19101559 )

Of Fall, 2022 has been accepted as satisfactory in partial fulfillment of the requirement for the degree of B.Sc. in Computer Science on January, 2023.

## Examining Committee:

Supervisor:  
(Member)



---

Dewan Ziaul Karim  
Lecturer  
Department of Computer Science and Engineering  
Brac University

Thesis Coordinator:  
(Member)

---

Md. Golam Rabiul Alam, PhD  
Professor  
Department of Computer Science and Engineering  
Brac University

Head of Department:  
(Chair)

---

Sadia Hamid Kazi, PhD  
Chairperson and Associate Professor  
Department of Computer Science and Engineering  
Brac University

# Abstract

Detecting corn leaf diseases helps farmers identify and treat impacted crops. Early disease identification reduces crop loss. Manual leaf diagnostic imaging takes time and is prone to mistakes. This thesis proposes a deep convolutional neural network (CNN) model for autonomous corn leaf disease identification. PlantVillage and PlantDoc were utilized. The dataset contains 4,188 photos of healthy maize leaves and three corn leaf illnesses. The photos have disease labels. We rotated, flipped, and scaled images for augmentation. After augmentation, the total number of photos in the dataset is about 12,000. We trained our CNN model using pre-trained architectures like InceptionResNetV2, MobileNetV2, ResNet50, VGG19, InceptionV3, VGG16, and DenseNet201. These architectures were chosen for their image feature extraction and large dataset learning capabilities. We used transfer learning to fine-tune a model using a pre-trained model. The model accurately detects corn leaf diseases in new photos. The model is computationally light, making it suited for smartphones and drones. A maize leaf disease detection mobile app was created using the proposed CNN model. The application can detect corn leaves uploaded by anyone. An API analyzes an image using our proposed model from the device's camera or gallery when a user selects it.

**Keywords:** CNN, Deep Learning, Image processing, Machine learning, Proposed Model, Transfer Learning

## **Acknowledgement**

First and foremost, all praise to the Almighty for whom our thesis has been completed fluidly without any interruptions.

We would like to express our profound gratitude to our Supervisor, Mr. Dewan Ziaul Karim, for his kind supervision and endless support throughout our work.

And finally, we are grateful to our parents, whose encouragement and prayers has been crucial towards our work.

# Table of Contents

Declaration	i
Approval	ii
Abstract	iii
Acknowledgment	iv
Table of Contents	v
List of Figures	vii
List of Tables	viii
<b>1 Introduction</b>	<b>1</b>
1.1 Introduction . . . . .	1
1.2 Problem Statement . . . . .	2
1.3 Research Objectives . . . . .	2
1.4 Literature Review . . . . .	3
<b>2 Dataset &amp; Data Pre-processing</b>	<b>7</b>
2.1 Data Source . . . . .	7
2.2 Data sample . . . . .	7
2.3 Resizing image: . . . . .	8
2.4 Normalization and scaling images: . . . . .	8
2.5 Data augmentation: . . . . .	8
2.6 Data labels: . . . . .	9
2.7 Data classification: . . . . .	9
2.8 Training Set: . . . . .	10
2.9 Testing Set: . . . . .	10
2.10 Validation Set: . . . . .	10
<b>3 Methodology</b>	<b>11</b>
3.1 Research Methodology . . . . .	11
3.2 CNN Model . . . . .	12
<b>4 Model Implementation</b>	<b>13</b>
4.1 Pre-trained CNN Models . . . . .	13
4.1.1 VGG 16 . . . . .	13
4.1.2 VGG 19 . . . . .	14

4.1.3	ResNet50 . . . . .	14
4.1.4	MobileNetV2 . . . . .	15
4.1.5	InceptionV3 . . . . .	16
4.1.6	InceptionResNetV2 . . . . .	16
4.1.7	DenseNet201 . . . . .	17
4.2	Proposed CNN Model . . . . .	18
4.2.1	Custom Model . . . . .	18
4.2.2	Keras Visualize of our model . . . . .	20
4.3	Tuning and Hyper Parameters . . . . .	21
4.3.1	Dropout Layer . . . . .	21
4.3.2	ReLU . . . . .	22
4.3.3	Softmax . . . . .	22
4.3.4	Adam Optimizer . . . . .	23
4.3.5	Max pooling . . . . .	23
4.3.6	Average Pooling . . . . .	23
4.3.7	Learning Rate scheduler . . . . .	24
<b>5</b>	<b>Performance Analysis</b>	<b>25</b>
5.1	Performance Parameter . . . . .	25
5.2	Performance of Pre-trained Model . . . . .	26
5.2.1	VGG16: . . . . .	26
5.2.2	VGG19: . . . . .	27
5.2.3	ResNet50: . . . . .	27
5.2.4	MobileNetV2: . . . . .	28
5.2.5	InceptionV3: . . . . .	30
5.2.6	InceptionResNetV2: . . . . .	30
5.2.7	DenseNet201: . . . . .	32
5.3	Performance of Proposed Model . . . . .	33
5.3.1	Custom Model . . . . .	33
5.4	Result comparison: . . . . .	34
5.5	Accuracy Comparison on Related Work . . . . .	37
5.6	Application of our Proposed Model . . . . .	38
<b>6</b>	<b>Conclusion</b>	<b>40</b>
6.1	Conclusion . . . . .	40
6.2	Future Work . . . . .	40
6.3	Challenges . . . . .	41
	<b>Bibliography</b>	<b>43</b>

# List of Figures

2.1	Sample Data . . . . .	7
2.2	Initial data count . . . . .	8
2.3	Comparison of Data before and after augmentation . . . . .	9
3.1	Flow chart of our Work . . . . .	11
3.2	CNN Model architecture . . . . .	12
4.1	Keras Visualizer of Custom Model . . . . .	21
4.2	CNN Model architecture . . . . .	21
4.3	ReLU Function . . . . .	22
4.4	Example of Max pooling Layer . . . . .	23
4.5	Example of Average pooling Layer . . . . .	24
5.1	Training and validation graph of VGG16 . . . . .	26
5.2	Confusion matrix of VGG16 . . . . .	26
5.3	Training and validation graph of VGG19 . . . . .	27
5.4	Confusion matrix of VGG19 . . . . .	27
5.5	Training and validation graph of ResNet50 . . . . .	28
5.6	Confusion matrix of ResNet50 . . . . .	28
5.7	Training and validation graph of MobileNetV2 . . . . .	29
5.8	MobileNetV2 confusion matrix . . . . .	29
5.9	Training and validation graph of InceptionV3 . . . . .	30
5.10	Confusion matrix of InceptionV3 . . . . .	30
5.11	Training and validation graph of InceptionResNetV2 . . . . .	31
5.12	InceptionResNetV2 confusion matrix . . . . .	31
5.13	Training and validation graph of DenseNet201 . . . . .	32
5.14	Confusion matrix of DenseNet201 . . . . .	32
5.15	Training and validation graph of Custom Model . . . . .	33
5.16	Confusion matrix of Custom Model . . . . .	34
5.17	Classification Report of Custom Model . . . . .	34
5.18	Accuracy Comparison . . . . .	35
5.19	Mobile App prediction . . . . .	39



# List of Tables

2.1	Data classification . . . . .	9
4.1	Model summary for VGG 16. . . . .	13
4.2	Model summary for VGG 19. . . . .	14
4.3	Model summary for ResNet50. . . . .	15
4.4	Model summary for MobileNetV2. . . . .	15
4.5	Model summary for InceptionV3. . . . .	16
4.6	Model summary for InceptionResNetV2. . . . .	17
4.7	Model summary for DenseNet201. . . . .	17
4.8	Model summary for Custom_Model. . . . .	19
5.1	Number of Parameters . . . . .	35
5.2	Precision . . . . .	36
5.3	Recall . . . . .	36
5.4	f1-score . . . . .	37
5.5	Comparison on related work . . . . .	38

# Chapter 1

## Introduction

### 1.1 Introduction

Corn is a vital food and feed crop. Except for rice and wheat, it has the biggest plant area and total yield in the world. However, the number of corn disease species and the extent of harm they inflict have increased in recent years, owing mostly to changes in farming practices, pathogen variety variation, and poor plant protection measures. Some of the most common leaf diseases are curvularia leaf spot, dwarf mosaic, gray leaf spot, blight, brown spot, round spot, and Common rust. Corn leaf disease, in particular, is dangerous and will have an impact on corn production as well as people's lives[1].

CNN is composed of a multitude of layers. CNN is a deep learning method that can classify, segment, recognize and detect images and objects. CNN technology has expanded into medical imaging, autonomous driving, robotics, and agricultural imaging. There have been numerous picture studies, such as the classification of illness in 15 food crops using 5 convolutional layers with googleNet. Mohanty et al. categorized 14 crops, including corn. 26 diseases were categorized. 54,306 images were tested. For classification, AlexNet and GoogleNet were used as deep learning neural networks. The accuracy was 31.4%. CNN was used to classify corn leaf images for disease detection. Sibiya & Sumbwanyambe classified corn leaf diseases using CNN. Northern leaf blight, common rust, and Cercospora are disease classifications. CNN architecture was not explained in detail, but it used 50 hidden layers of convolution, ReLU, and pooling layers. Each class used 100 images, 70% for training and 30% for testing. In this paper we suggest a lightweight CNN model with comparable testing accuracy.

## 1.2 Problem Statement

Maize is grown all over the world. The United States, China, Brazil, and India are some of the countries that produce the most corn. Corn is becoming a popular crop in Northern Bangladesh because it doesn't need much water. India's food crop is the third most important in the world. Blight, Gray Leaf Spot, and Common rust are the most common diseases that affect corn leaves. Corn diseases can cause a country to lose a lot of money. The first signs of corn disease show up very quickly. If diseases are found early and the right steps are taken, it may be possible to keep the quality of the crop [6]. Corn leaf disease needs to be easy to spot so that risks can be quickly found and dealt with. We can find corn leaf disease with the help of computer vision and machine learning.

[12] says that among the 5 algorithms Naive Bayes, SVM, Decision tree, KNN, and Random Forest (RF) had the best accuracy at 80.68%, which is not good enough.

In [7], Convolutional Neural Network (CNN) is used and the achieved accuracy is 88.46%. But, the model can only detect two diseases which are Blight and Common Rust.

The purpose of this research is to develop a system for automatically detecting corn leaf diseases using computer vision and machine learning techniques such as Convolutional Neural Network.

## 1.3 Research Objectives

The purpose of this study is to develop a CNN-based model that is capable of accurately diagnosing and categorizing diseases that affect maize leaves. This system will take images as input. We will use Deep Learning to figure out whether or not the images show disease. After any necessary preprocessing, the images will be run through the proposed CNN model, which will sort them into four groups. These are the goals of this research:

- Know how image processing works and how it is done.
- To know how to use techniques like Denoising and Reshaping to prepare data.
- To use images to build a model that can find Corn diseases.
- To figure out what deep learning means for our model.
- To understand the importance of hyper parameter optimization.

## 1.4 Literature Review

Crops are a significant food for human beings. We directly or indirectly depend on these crops. Corn is a famous food all over the world, especially in Asia and Europe. Corn kernels come in six varieties: flint, flour, dent, pop, sweet, and waxy. Corn flour is made from flour corn, which is predominantly farmed in the Andean region of South America. Waxy corn is a Chinese crop with a texture similar to sticky rice. Flint corn has a hard outer shell with kernels in a variety of shades ranging from white to crimson which is similar to dent corn. Sweet corn has a higher natural sugar content than other forms of maize, making it extremely sweet.

Maize is susceptible to diseases such as gray leaf spot, blight, tar spot, and common rust and southern rust during the growing season. Early disease treatment is essential for maintaining a healthy corn crop and protecting yields. Maize seed rots and seedling blights are caused by many fungal species belonging to numerous genera (*Fusarium* spp., *Rhizoctonia* spp., *Pythium* spp., *Diplodia* spp., *Penicillium* spp., *Trichoderma* spp). These fungus are all common microbes found in cornfields. They can be discovered in crop leftovers and soil. It is preferable to avoid or manage a disease outbreak when the disease is at low levels, rather than attempting to cope with a disease that has already caused major harm, in order to efficiently manage corn for disease. Weekly field scouting can reveal whether illnesses are present, their severity, and the risk of crop loss if not treated.

In paper [11], An optimized custom DenseNet model for diagnosing and classifying leaf diseases was shown. This custom DenseNet model has less parameters than most CNN designs, so it takes less time to run. Here has also trained VGGNet, XceptionNet, EfficientNet, and NASNet, which are all well-known CNN models, to recognize and classify corn leaf diseases. Twelve thousand three hundred and thirty-two images with a size of 250 x 250 pixels were collected by hand from different sources and split into four crop classes. In addition, a Dataset called "Total Leaf Dataset" is used here. The model has been right 98.06 percent of the time. In the future, a mobile app is expected to be launched which will be able to detect diseases on corn leaf.

In paper [1], GoogLeNet and Cifar10 models for recognizing leaf diseases based on deep learning are made better. The main goals of this paper were to cut down on the number of network parameters and improve how well corn leaf diseases could be found. The Cifar10 model is only capable of identifying maize leaf diseases with an average accuracy of 98.8 percent, but the GoogLeNet model is able to recognize eight different types of corn leaf diseases with an average accuracy of 98.9 percent. There were a total of 500 photographs sourced from various locations, such as the websites of Plant Village and Google, that demonstrated various stages of maize leaf disease. These photographs have been organized into nine distinct categories. Finding new varieties of corn diseases and pests will be the primary focus of future research; this will be followed by the development of new algorithms and other forms of deep learning structures for use in the testing and training of models.

In paper [6] real-life-based deep convolutional neural network method is proposed. The model identifies the two most common corn diseases in India. They are Leaf

Blight and Common Rust. The images that were taken from the corn plants themselves are used as the dataset. The network is given an image with a resolution of 150x150 pixels. For each of the weights, a batch size of 32 was selected, and Glorot uniform initialization was used. Maximum pooling with a pool size of two by two is utilized here. Additionally, the relu activation function is utilized throughout the network with the exception of the last layer. The Softmax activation function is used for the final layer's processing. In order to obtain an accuracy of more than 96%, other network-hyper parameters, such as the learning rate and the maximum epoch, were altered during training. The accuracy of this model was determined to be 88.46 percent on the whole.

In paper [13], the ResNet50 model was proposed. Adam approach optimized the model by adding L2 regularization. Reduces overfitting. ResNet50 recognizes better than other models. The data set has  $98.52\pm\%$  image recognition accuracy, and agriculture has  $97.826\pm\%$ . SVM, genetic algorithm, local discriminant mapping, and local linear embedding algorithms operate poorly with weak resilience. 2309 photos of mosaic, gray spot, rust, and leaf blight corn diseases were collected. Brightening, translation, and flipping improved identification accuracy. The input module, four blocks (3, 4, 6, and 3 in each module), and output module comprise the network. Levels used ReLU activation functions. Batch canonicalization units improved model flexibility. ADAM optimizes network recognition accuracy. After loading the data file, the pretraining model was loaded followed by a data reader which helps for image classification. Optimization strategy and parameters were set before fine-tuning. Then, the model parameters were trained and evaluated. When the ReLU function is applied, recognition accuracy is maximized. If  $7*7$  stacking layers are replaced with  $3*3$  stacking layers, the layer of ResNet50 network is improved as it effectively reduces the amount of calculation, without changing the initial receptive field.

In paper [5], the CNN recognition model has been proposed based on a combination of data augmentation and transfer learning. The authors compared and analyzed the original GoogleNet network and then used other CNN to transfer the learning effect. The training data set has used four ways to add more data: random cropping, horizontal flipping, vertical flipping, and center cropping. This model can be used to analyze remote sensing images, especially to figure out what they are. It can be used in many different ways in agriculture. Two changes are made: (1) the data are improved to make the model more robust and general, and (2) migration learning is used to speed up the training process and reduce the amount of overfitting. The network model has been made as good as it can be by using Adam's (Adaptive Moment Estimation) optimization algorithm. Plant Village was used to get the data set. In this study, 4,354 pictures of corn leaves were used. These pictures were put into four groups: Corn Gray Leaf Spot, Corn Common Rust, Corn Healthy, and Corn Northern Leaf Blight. The image resolution was set to 224 x 224 for both training and testing data by the authors. Lastly, they changed the PIL Image or Nddarray to a tensor, normalized the data to [0,1], and made sure it was the same for each channel. In the first part of the experiment, they kept changing the training hyper parameters, such as the learning rate, optimizer, and batch size, so that the model could be more stable. This improved CNN model records the highest precision

of 96.05%, recall rate of 97.01%, and F1 score of 96.53% when compared to VGG19, VGG16, and ResNet18.

In Paper[15] an expert system is introduced using the forward chaining method, an artificial intelligence-based system for disease detection in corn plants at an early stage. Here, PHP programming language and MySQL database are being used for system development. It's a step-by-step process of making a Data Flow Diagram, a context diagram for creating database design followed by coding the symptoms(31 symptoms for 6 diseases) of corn diseases. Based on test methods, its user acceptance is 84% and for the black box method, the system's functionality is 100%. The designed system application can not only detect disease but also provides a treatment solution in the form of a consultation tool for the farmers.

In paper [16], deep transfer modeling or in other words deep neural network has been proposed for identifying and classifying corn plant leaf disease. For fast and accurate detection AlexNet Model and Plant Village Data set containing two categories of diseases have been used. Cercospora comprises 1363 photos, whereas Gray contains 929 images. The AlexNet model introduced has 5 convolution layers, 3 max-pooling layers, and multiple epochs such as 25, 50, 75, and 100 times iteration. Accuracy performance improves as the number of iterations increases. When the various approaches are compared, the proposed method achieves an accuracy rate of 99.16%. Though a huge number of data sets and iterated AlexNet CNN associated with epoch iteration can turn into main challenges while following this study. The benefits are getting instant information, reduced outbreaks, upsurges and huge losses won't be threatening to the farmers anymore.

In paper [14], Deep neural networks are used to detect corn diseases. At first, the classification of AI ( ML and DL) and also types of corn are described. Then a camera takes the pictures of each corn type and the image is then processed, fragmented, and fed to the AI method. As the data set gets bigger, it becomes easier to predict diseases. The accuracy rate of ANN is thought to be about 98% which is the highest of the other monitoring predictions.

In paper[8], CNN Approach is used to predict corn leaf diseases. Over 50 k images are taken from various villages and then over 2000 are analyzed. After processing and normalizing the data, it is separated into training and testing sets. The model is then trained by this and then used to detect which leaf is healthy and which is not. An accuracy of 98.78% can be achieved by this method.

In [2] used deep CNN. Late blight, early blight, and healthy potatoes were expected. They had 98.33% accuracy on the same data set. Each input image has pooling, fully connected convolutional layers. Plant Village-based CNN model. 1000 Early Blight and Late Blight shots, 250 healthy plant photos. Classifying images utilizes 0-1 probabilities. Image collection, training and validation preprocessing, image augmentation, deep CNN training, and model fine-tuning are involved. The two-step procedure is Preprocessed CNN. Grayscale converts RGB pictures. 3-becomes-1 in the color channel. Gaussian blur and median filter eliminate noise. All photos were categorized after preprocessing. This design has 2 thick and 5 convolution layers. The first layer has 1024 filters, the last layer has 3 outputs. All layers are ReLu

and 33. Tri-MaxPooling was used. Normalizing batches accelerates learning. Adam adjusted weights. Each batch of 32 images has 32 photos. They scored 98.33%. Recall and F1 both equal 0.9809. The model performance confusion matrix. Deep CNN for Potato plants reaches 98.33% accuracy, compared to RGB Imaging's 95%. TLNet (Tomato Leaf Net) is a convolutional neural network model for predicting tomato leaf diseases. Model predicts 7 leaf diseases and 1 healthy class. Image-based training, validation, and testing. Methodology includes dataset prep and Deep CNN implementation. Normalized, enhanced, label-encoded dataset. After batch normalization and max pooling[2x2], the model has 5 convolutional and 3 dense layers[3x3]. Pesticides and pharmaceutical drugs use segmentation. The 12000-photo model is 98.77% accurate. This research can find other leaf diseases. CNN is used to identify rice diseases from leaves. Image analysis uses deep learning. Machine learning predicted 4 major rice illnesses from their leaves, improving agricultural productivity. This CNN-based algorithm accurately forecasts rice leaf diseases. They utilized 900 images to identify diseases. Four rice diseases were identified using 10-fold cross-validation. Image preprocessing and DCN were used. This project intended to build a cost-effective, error-free machine learning system for recognizing rice plant diseases.

# Chapter 2

## Dataset & Data Pre-processing

### 2.1 Data Source

This dataset uses the well-known PlantVillage [9] and PlantDoc [3] datasets. 4,188 images of three different forms of maize leaf diseases as well as healthy corn leaves are included in the dataset that was gathered. After augmentation, the number reached 12,000 units. 8,400 files overall from those were selected for training. 1,200 for the test and 2,400 for validation. Unneeded photos were eliminated from the dataset. The dataset was obtained from Kaggle.

### 2.2 Data sample

Most pictures were in jpg format and a few in JPEG format. The dimension of those pictures was not uniform. The pictures were rescaled to achieve uniform dimensions.

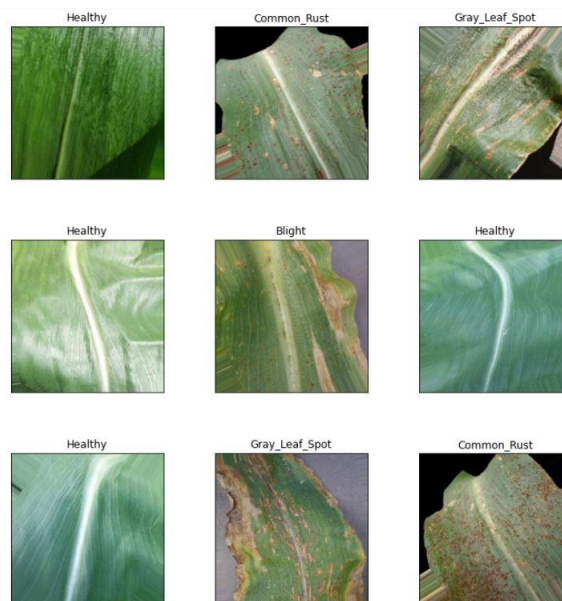


Figure 2.1: Sample Data



## 2.3 Resizing image:

Every image must be resized to the same size to be fed to a CNN network. In our research, we are using InceptionResNetV2, MobileNetV2, ResNet50, VGG19, InceptionV3, VGG16, DenseNet201 and our custom CNN model. Images are resized according to the model. For all the pre trained model and our custom model, images are scaled to 224 x 224 pixels. Image resizing was achieved via tensorflow's ImageDataGenerator library.

## 2.4 Normalization and scaling images:

In image processing, normalization means to bring down pixel values within 0 and 1. Pixel values range between 0 to 255. So, dividing each pixel by 255 will bring down pixel values between 0 and 1. With small numbers for each pixel (0 to 1) computation becomes efficient and faster. This rescaling of pixel values was achieved by ImageDataGenerator with parameter  $rescale = 1 / 255$ .

## 2.5 Data augmentation:

In order to get better accuracy, enough data needs to be fed to the network. Another reason to augment data is data imbalance. In classification problem, imbalanced data refers to a situation where data count per class is not even. For example, in our dataset data count for Gray leaf spot was low. Data imbalance can hamper model accuracy. With imbalanced data, the accuracy shown might be fake. Below is the data count before augmentation.

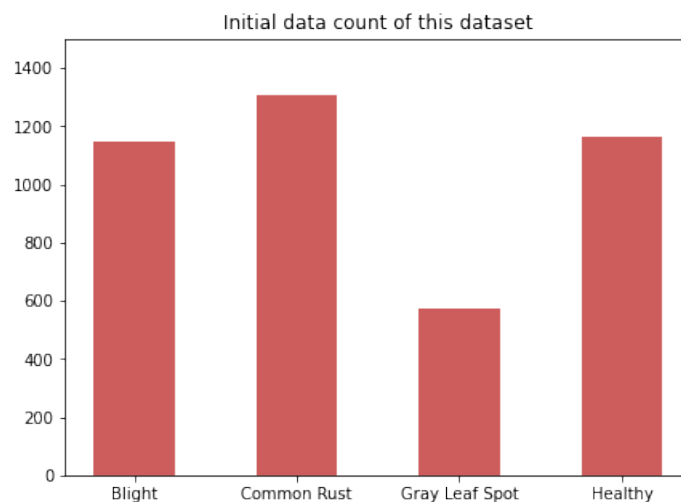


Figure 2.2: Initial data count

As we can see, data count of the Gray Leaf Spot class is lower compared to other classes. This imbalance might produce wrong results. So, data for this class must be augmented. We used python Augmenters library to augment image within the Gray Leaf Spot folder. Augmenters uses some techniques like flipping, rotating, skewing existing images to produce new images.

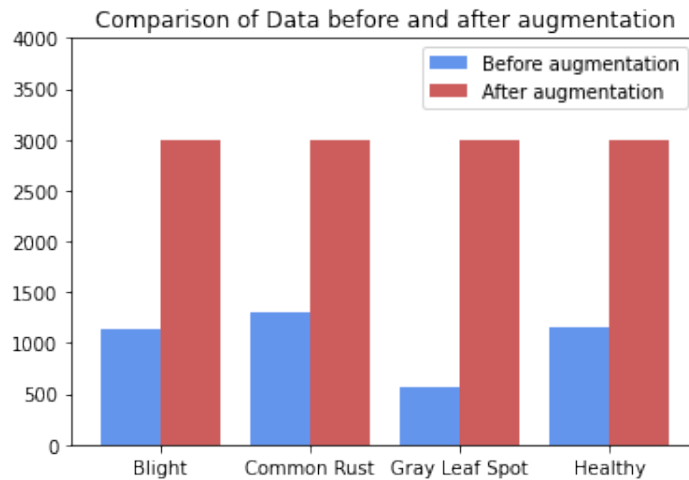


Figure 2.3: Comparison of Data before and after augmentation

## 2.6 Data labels:

The dataset was divided into 4 distinct labels. Those are Blight, Common Rust, Gray Leaf Spot, Healthy. So, there are more than 2 classes. That's why the categorical classifier is used. Also, labels were balanced using data augmentation.

## 2.7 Data classification:

We trained the model using 70% of the data, validated it with 20% of data and tested it with the remaining 10% of data.

Class	Training set	Validation set	Testing set	Total
Blight	2,100	600	300	<b>3,000</b>
Common Rust	2,100	600	300	<b>3,000</b>
Gray Leaf Spot	2,100	600	300	<b>3,000</b>
Healthy	2,100	600	300	<b>3,000</b>
Total	<b>8,400</b>	<b>2,400</b>	<b>1,200</b>	<b>12,000</b>

Table 2.1: Data classification

## **2.8 Training Set:**

Training data is used to update model weights for input-to-output mapping. Neural networks learn from real-world data and solutions to generalize input-output relationships. System inputs may include algorithms or output labels. This training phase is handled by an optimization approach that seeks weights that perform well on the training dataset.

## **2.9 Testing Set:**

Our system learns from the training set to improve real-world data. Positive test results improve the algorithm's confidence in the real world.

## **2.10 Validation Set:**

Validation is sometimes considered part of the training phase. A validation set is used to tweak models. The validation set eliminates overfitting and optimizes it.

# Chapter 3

## Methodology

### 3.1 Research Methodology

We collected the data and pre-processed it. We augmented the data for label balancing. Pre-processing also includes splitting the dataset into train, test, and validation folders. We trained with 70% of the data, validated with 20%, and tested with 10%. After that, the dataset will go through several CNN architectures like InceptionResNetV2, MobileNetV2, ResNet50, VGG19, InceptionV3, VGG16, DenseNet201 and a custom model. Finally, accuracy and loss graphs for each epoch and confusion matrix will be generated to evaluate model performance.

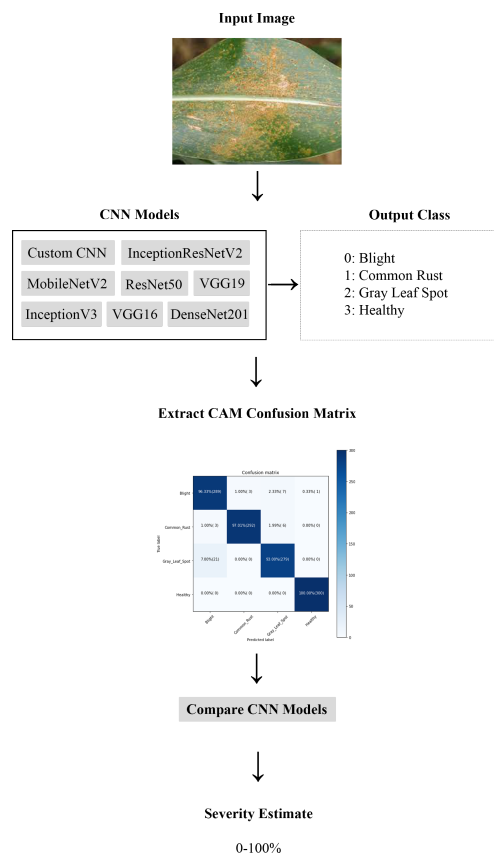


Figure 3.1: Flow chart of our Work

## 3.2 CNN Model

We will use Convolutional neural networks (CNN) for our research purpose. It's basically neural networks with multiple convolutional layers for segmentation, image processing, classification, and other auto-correlated data. Both hardware and software systems for neural networks are modeled after the process of neurons inside the human brain. CNN's "neurons" work approximately such as the frontal lobe of humans and other animals that process visual signals. Standard neural networks' fragmented image processing difficulty is avoided by organizing the layers of neurons in such a way that they create the full central vision. We will use pre-trained models such as EfficientNetB2, Alexnet, InceptionNetV3, and a custom 22-layer CNN model.

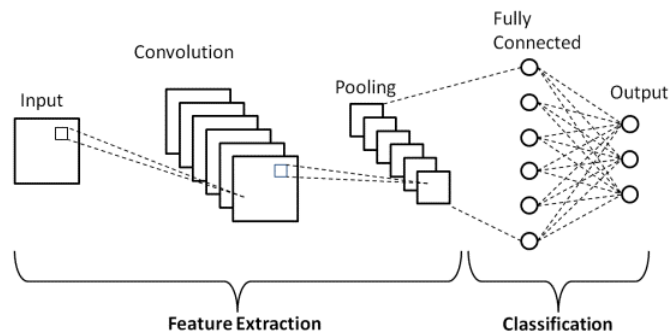


Figure 3.2: CNN Model architecture

# Chapter 4

## Model Implementation

### 4.1 Pre-trained CNN Models

#### 4.1.1 VGG 16

One of the most efficient computer vision models available right now is the CNN (Convolutional Neural Network) variant known as VGG16. By analyzing the networks and increasing the depth with an architecture that uses incredibly small (3 By 3) convolution filters, the model demonstrated a considerable improvement over the state-of-the-art configurations. Karen Simonyan and Andrew Zisserman created the first version of this CNN model at Oxford. Although the technique was published in 2013, the ILSVRC ImageNet Challenge in 2014 was when it was initially made public. They named it VGG in honor of their group, Oxford's Visual Geometry Group. In the VGG16 model, 16 weighted layers are indicated by the number 16. VGG16 consists of 5 max-pooling layers, 16 weight layers, 13 convolutional layers, and 3 dense layers, for a total of 21 layers. The total number of parameters in VGG16 is 138 million. It's important to remember that each of the conv kernels is 3x3, whereas the maxpool kernels are 2x2 with a 2-stride in this situation. Although the method was released in 2013, it was first made public during the ILSVRC ImageNet Challenge in 2014. The sixteen in VGG16 stands for sixteen weighted layers. VGG16 includes a total of 21 layers, nonetheless, there are just 16 weight layers: 3 dense layers, 5 maximum pooling layers, and 13 convolutional layers.

Layer (type)	Output Shape	Param #
vgg16 (Functional)	(None, 7, 7, 512)	14714688
flatten (Flatten)	(None, 25088)	0
dense (Dense)	(None, 256)	6422784
dropout (Dropout)	(None, 256)	0
dense_1 (Dense)	(None, 4)	1028
Total params: 21,138,500		
Trainable params: 21,138,500		
Non-trainable params: 0		

Table 4.1: Model summary for VGG 16.

### 4.1.2 VGG 19

VGG-19 is the name of a convolutional neural network with 19 layers. It has 138 million Parameters. A pre-trained version of the network that has been trained on more than a million images is present in the ImageNet database. A number of animals, a mouse, a keyboard, and a pencil are among the 1000 different item categories that the pre-trained network can classify images into. As a result, the network now includes rich feature representations for a range of images. Images having a resolution of 224 by 224 are accepted by the network. In order to capture up-and-down and left-and-right motion, the convolutional layers of VGG make use of the smallest feasible receptive field. The next step is a ReLU activation method. Since the ReLU activation function outputs the input when it is negative and returns zero when it is positive, it is regarded as a piecewise linear activation function. After convolution, the stride value to preserve The spatial resolution is 1 pixel.

Layer (type)	Output Shape	Param #
vgg19 (Functional)	(None, 7, 7, 512)	20024384
flatten (Flatten)	(None, 25088)	0
dense (Dense)	(None, 256)	6422784
dropout (Dropout)	(None, 256)	0
dense_1 (Dense)	(None, 4)	1028
Total params: 26,448,196		
Trainable params: 26,448,196		
Non-trainable params: 0		

Table 4.2: Model summary for VGG 19.

### 4.1.3 ResNet50

An example of a convolutional neural network is ResNet-50, which has 50 layers. Over 23 million different parameters can be trained on it. It is possible to load a network that has been pre-trained using more than a million images in the ImageNet database. It functions as the ResNet Convolutional Neural Network (CNN) architecture, which is a reliable conceptual framework for many computer vision applications. A keyboard, pencil, mouse, and other animal images are among the 1000 different item categories that the pre-trained network can identify photos into. As a result, the network has learned detailed feature representations for a range of images. An image with 224 by 224 pixels can be uploaded to the network. ResNet is accessible in a variety of forms, each with a distinct quantity of layers. Up to tens of thousands of layers in a network may be trained without doubling the number of activation mistakes. ResNets and identity mapping can be used to solve the vanishing gradient problem. Using the ImageNet database, we could import a network that has already been trained on more than a million images. The previously trained network can recognize more than a thousand other objects in addition to keyboards, pencils, and mice. When utilizing this method, deep neural networks with more neural layers are more effective and make fewer errors overall. Due to the introduction of skip connections, it is now possible to train far deeper networks than was previously allowed.

Layer (type)	Output Shape	Param #
resnet50 (Functional)	(None, 7, 7, 2048)	23587712
flatten (Flatten)	(None, 100352)	0
dense (Dense)	(None, 256)	25690368
dropout (Dropout)	(None, 256)	0
dense_1 (Dense)	(None, 4)	1028
Total params: 49,279,108		
Trainable params: 49,225,988		
Non-trainable params: 53,120		

Table 4.3: Model summary for RestNet50.

#### 4.1.4 MobileNetV2

The name "MobileNet-v2" refers to a convolutional neural network with 53 layers. There are 3.4 million of them. In the ImageNet database, there is a version of the network that has already been trained on more than a million pictures. The network that has already been trained can put photos into 1000 different types of items. Because of this, the network has detailed representations of many different pictures' features. Images with up to 224 by 224 pixels can be sent through the connection. Mobile Nets are a depth-wise separable convolution design that uses as few connections as possible to reduce the size of the model and the complexity of the channel. Both embedded and mobile applications can benefit from the method. The expert has added the following two universally specific networks to this type of system: This strategy finds a good balance between how well it predicts and how long it takes. If needed, the hyperparameters can also be used to choose a good scaled model that fits the problem's constraints.

Layer (type)	Output Shape	Param #
mobilenetv2.1.00.224 (Functional)	(None, 7, 7, 1280)	2257984
flatten_2(Flatten)	(None, 62720)	0
dense_4 (Dense)	(None, 256)	16056576
dropout_2 (Dropout)	(None, 256)	0
dense_5 (Dense)	(None, 4)	1028
Total params: 18,315,588		
Trainable params: 18,281,476		
Non-trainable params: 34,112		

Table 4.4: Model summary for MobileNetV2.



### 4.1.5 InceptionV3

On the ImageNet dataset, an image recognition model by the name of Inception v3 has shown greater than 78.1% accuracy. The model depicts the confluence of several concepts created by numerous scholars over time. Some of the symmetric and asymmetric building blocks that make up the model itself include convolutions, concatenations, average pooling, dropouts, max pooling, and completely linked layers. The batch normalization method, which is used for the stimulation parameters as well, is heavily utilized by the model. Using Softmax, the loss is calculated. One of the most effective approaches for transfer learning is the Inception-v3 model. This allows us to retrain the last layers of the present goods, which reduces a significant amount of time. The ImageNet database was used to train Inception-v3, which shows that the model can be applied to a lower dimension and provide high-precision assessments without retraining. It has under 25 million parameters which are almost 24 million. The Inception Layers are a collection of layers (11 convolutional layers, 33 convolutional layers, and 55 convolutional layers) that bring together the result filters into a unified output value to provide the inputs for the subsequent steps. To preserve any tangible benefits, changes to an Inception Network must be managed carefully. Only a few of the mechanisms used in parametric convolution comprise batch normalization, downsampling, and parallel processing. Two sets of parameters: 5 million (V1) and 23 million (V2) (V3).

Layer (type)	Output Shape	Param #
inception_v3 (Functional)	(None, 5, 5, 2048)	21802784
flatten (Flatten)	(None, 51200)	0
dense (Dense)	(None, 256)	13107456
dropout (Dropout)	(None, 256)	0
dense_1 (Dense)	(None, 4)	1028
Total params: 34,911,268		
Trainable params: 34,876,836		
Non-trainable params: 34,432		

Table 4.5: Model summary for InceptionV3.

### 4.1.6 InceptionResNetV2

Inception-ResNet-v2 is an expansion of the Inception family of architectures and includes convolution layers. It is trained using images from the ImageNet database. It has 56 million parameters. The network, which has 164 layers and can classify pictures into 1000 distinct sorts of items, can identify 1,000 various product classifications. As a result, the system has acquired knowledge of many complex graphical multilayer perceptrons. With a set of 299-by-299-pixel input images, class probabilities are computed. The Inception and Residual Link frameworks serve as its foundation. Multiple convolutional filters and residual connections are employed in the Inception-Resnet block. Utilizing residual blocks decreases the learning curve while also avoiding the performance degradation brought on by deep structures. The model reflects the routing protocols of Resnet-fundamental v2.

Layer (type)	Output Shape	Param #
inception_resnet_v2 (Functional)	(None, 5, 5, 1536)	54336736
flatten (Flatten)	(None, 38400)	0
dense (Dense)	(None, 256)	9830656
dropout (Dropout)	(None, 256)	0
dense_1 (Dense)	(None, 4)	1028
Total params: 64,168,420		
Trainable params: 64,107,876		
Non-trainable params: 60,544		

Table 4.6: Model summary for InceptionResNetV2.

#### 4.1.7 DenseNet201

DenseNet-201 is the name of a convolutional neural network with 201 levels of depth. A network that has been pre-trained with more than a million images is present in the ImageNet database. 1000 different item categories can be used by the training algorithm to classify images. As a result, the network has gathered thorough feature representations for a range of images. The network can support up to  $224 \times 224$  pixels in pictures. The principle behind DenseNet201 is that convolutional networks may be trained to be significantly deeper, more precise, and more effective if they have shorter connections between layers that are near the source and those that are close to the outcome.

Layer (type)	Output Shape	Param #
densenet201 (Functional)	(None, 7, 7, 1920)	18321984
flatten (Flatten)	(None, 94080)	0
dense (Dense)	(None, 256)	24084736
dropout (Dropout)	(None, 256)	0
dense_1 (Dense)	(None, 4)	1028
Total params: 42,407,748		
Trainable params: 42,178,692		
Non-trainable params: 229,056		

Table 4.7: Model summary for DenseNet201.

## 4.2 Proposed CNN Model

### 4.2.1 Custom Model

Most of the pre-trained CNN models are very deep and have way too much parameters. Huge number of parameters causes greater computational cost and training time. For instance, InceptionResNetV2 has around 60 million parameters, DenseNet201 has 40 million and so on. The lowest parameter count is in MobileNetV2. It has about 10 million trainable parameters. So, the goal is to propose a lightweight model with reasonable testing accuracy.

The model contains overall 6 2D convolution layers and 2 fully connected dense layers. A kernel size of 3x3 is used for all convolution layers. For pooling layers, kernel size of 2x2 is used. The number of parameters in our model is about 4 million.

The model starts with a convolution layer with input shape 224 x 224 x 3. The resolution is multiplied by 3 because the images are converted to RGB where each pixel contains a combination of 3 different colors. It includes 32 feature maps with ReLu activation. This layer is followed by the AveragePooling layer and BatchNormalization layer. Similar convolution layers with 64 feature maps followed by the AveragePooling and BatchNormalization layers are used in the next layers. Filter sizes are gradually increased and it often results in better performance. At first MaxPooling was used after the first two convolution layers. Replacing them with average pooling increased the validation and testing accuracy slightly.

After that convolution, BatchNormalization, MaxPooling sequence is followed 3 more times. The kernel sizes of those 3 convolution layers are 128, 256, 512 respectively.

There is another convolution layer after that which is followed by the GlobalAveragePooling layer. Use of GlobalAveragePooling layer significantly increased the performance. GlobalAveragePooling layer has no parameters to optimize. Thus, overfitting is avoided in this layer. Then it contains a dense layer which is followed by a dropout layer with a rate of 25%. Dropout helps to reduce overfitting by discarding random neurons from the network. A dense layer with softmax activation function concludes this model.

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 222, 222, 32)	896
average_pooling2d (AveragePooling2D)	(None, 111, 111, 32)	0
batch_normalization (BatchNormalization)	(None, 111, 111, 32)	128
conv2d_1 (Conv2D)	(None, 109, 109, 64)	18496
average_pooling2d_1 (AveragePooling2D)	(None, 54, 54, 64)	0
batch_normalization_1 (BatchNormalization)	(None, 54, 54, 64)	256
conv2d_2 (Conv2D)	(None, 52, 52, 128)	73856
max_pooling2d (MaxPooling2D)	(None, 26, 26, 128)	0
batch_normalization_2 (BatchNormalization)	(None, 26, 26, 128)	512
conv2d_3 (Conv2D)	(None, 24, 24, 256)	295168
max_pooling2d_1 (MaxPooling2D)	(None, 12, 12, 256)	0
batch_normalization_3 (BatchNormalization)	(None, 12, 12, 256)	1024
conv2d_4 (Conv2D)	(None, 10, 10, 512)	1180160
max_pooling2d_2 (MaxPooling2D)	(None, 5, 5, 512)	0
batch_normalization_4 (BatchNormalization)	(None, 5, 5, 512)	2048
conv2d_5 (Conv2D)	(None, 3, 3, 512)	2359808
global_average_pooling2d (GlobalAveragePooling2D)	(None, 512)	0
batch_normalization_5 (BatchNormalization)	(None, 512)	2048
flatten (Flatten)	(None, 512)	0
dense (Dense)	(None, 512)	262656
dropout (Dropout)	(None, 512)	0
dense_1 (Dense)	(None, 4)	2052
Total params: 4,199,108		
Trainable params: 4,196,100		
Non-trainable params: 3,008		

Table 4.8: Model summary for Custom\_Model.

## 4.2.2 Keras Visualize of our model

The architecture of a custom deep convolutional neural network model made for this thesis to find corn leaf diseases is shown in the diagram made with Keras Visualizer. The diagram shows how the model's different layers connect to each other. The diagram is made up of different blocks, and each block shows a group of layers. The first block is the input layer, which gets an image as input. The image is then sent through a series of convolutional layers, which are made to pull out features from the image. Filters scan the image and pull out certain parts, like edges and textures, from these layers. The number in the top right corner of each block shows how many filters are in each layer.

The convolutional layers are followed by a few average pooling layers. Using these layers, we can compress the feature maps without losing any useful information. This improves the model's efficiency and decreases the workload. The layers that follow the average pooling ones are the batch normalization layers, which standardize the activations from the preceding layers. Multiple max-pooling layers follow the batch normalization layers. These layers help compress feature maps without losing any useful information. It improves the model's performance and lessens the load it must carry as a result.

The model also features a global average pooling layer. By applying this layer, the feature maps' dimensionality reduction to a single number is accomplished. A flatten layer follows this one and its purpose is to prepare the data for the fully connected dense layers. Finally, the thick layers are utilized to produce predictions using the information learned in earlier levels. Finally, the output layer reveals the likelihood that the sickness will have an impact on the image we fed it. It's not a skintight fit because of the model's dropout layers. In order to achieve a desired result, dropout layers eliminate neurons at random. Because of this, the model is less reliant on a single neuron. This schematic depicts the basic layout of a model for a deep convolutional neural network. This model is intended to analyze a photograph of a corn leaf and determine the presence or absence of a certain disease. To prevent overfitting, the model employs several different types of layers, including convolutional ones, pooling ones, batch normalization ones, max pooling ones, global average pooling ones, flatten ones, dense ones, and dropout ones.

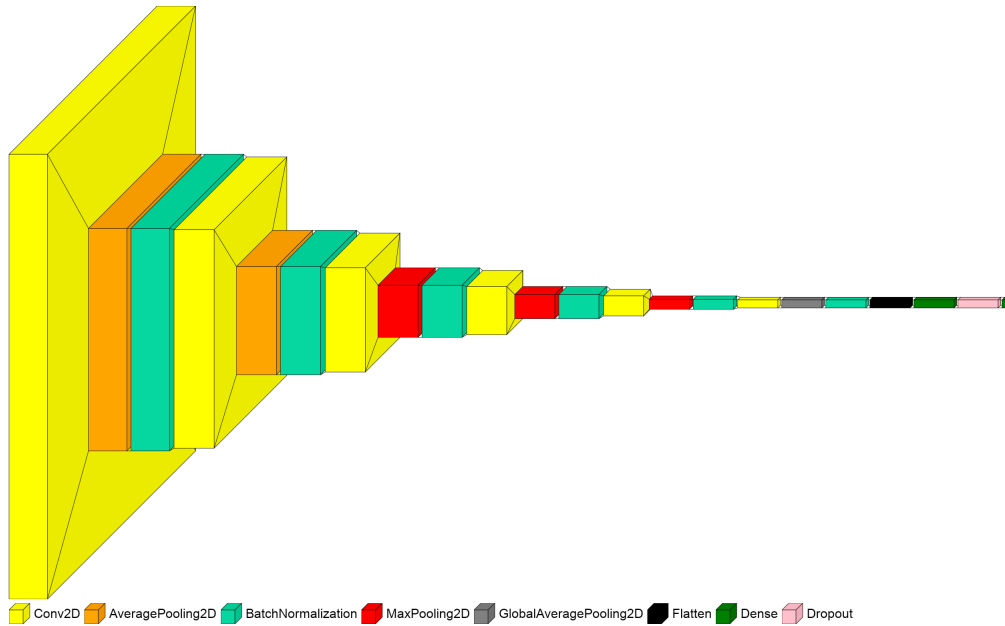


Figure 4.1: Keras Visualizer of Custom Model

## 4.3 Tuning and Hyper Parameters

### 4.3.1 Dropout Layer

As a mask, the Dropout layer keeps all other neurons functioning while removing certain neurons' contributions to the next layer. While some of the characteristics of the input vector are deleted when a Dropout layer is applied to it, certain latent neurons are destroyed when the Dropout layer is applied to a hidden layer. Dropout layers play a key role in the development of CNNs as they prevent the training data from becoming overfit. The initial batch of training dataset has a disproportionately big influence on learning if they are not there. As a consequence, features that appear in samples or batches after the current ones would not be started learning:

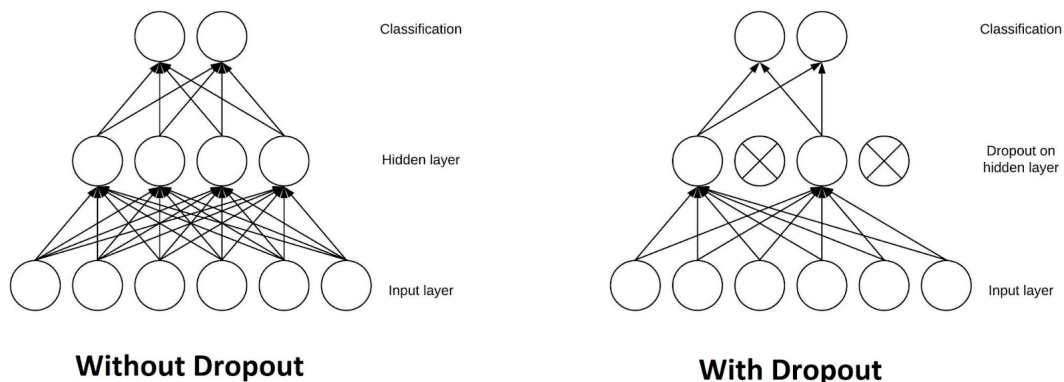


Figure 4.2: CNN Model architecture

Consider presenting a CNN 10 pictures of circles, one by one as part of the training process. CNN will be confused if we subsequently show it a picture of a rectangle since it doesn't know that straight lines may exist. We may prevent these occurrences by introducing Dropout layers into the network's design to reduce overfitting.

### 4.3.2 ReLU

Rectified linear activation function (ReLU) is a non-linear or piecewise linear function that returns the input value unmodified if the input is positive and returns zero if the input is negative. The activation function is the one that is most often used in neural networks, especially Convolutional Neural Networks (CNNs) as well as Multilayered Perceptrons. Even though it is simple to use, it exceeds the tanh and sigmoid, which were its precursors, in case of efficiency. It may be represented mathematically as follows:

$$f(x) = \max(0, x) \quad (4.1)$$

The way that it is depicted graphically is as,

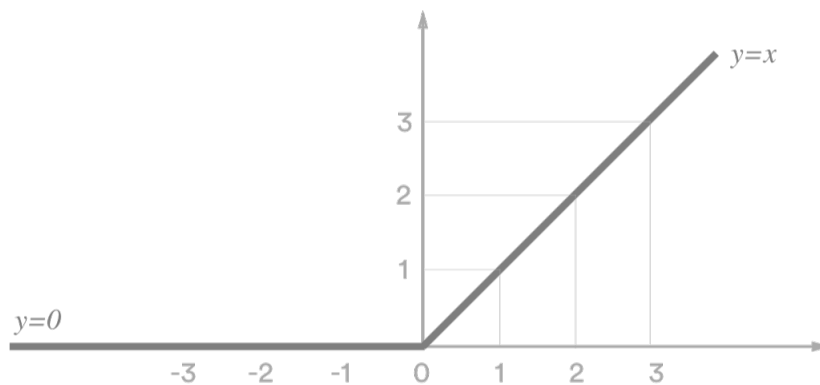


Figure 4.3: ReLU Function

### 4.3.3 Softmax

The softmax function is a type of mathematical function that has widespread application in the field of machine learning, particularly when dealing with situations involving several classes of categorized data. It is employed in the process of transforming a vector of real numbers into a probability distribution in which the total value of the vector's components is equal to 1. After applying the exponential function to each member of the input vector of real numbers, the softmax function normalizes the resulting vector so that the elements of the output vector sum up to one. The input vector of real numbers is required for the softmax function. Because of this, the output of the softmax function can be understood as a probability distribution over the various classes. When solving issues involving the categorization of many classes, it is frequently applied as the activation function to the output layer of a neural network. Since the softmax function can be utilized to regularize the model, it is also capable of assisting in the prevention of overfitting.

The softmax formula is as follows:

$$\text{Softmax}(x_i) = \frac{\exp(x_i)}{\sum_j \exp(x_j)} \quad (4.2)$$

### 4.3.4 Adam Optimizer

Adam, a distinct optimization approach, which continuously iterates "adaptive moment estimation" to enhance neural network weights more effectively. Adam solves non-convex problems faster and more efficiently with fewer resources than other optimization tools using stochastic gradient descent. It works best in extremely large data sets by maintaining "tighter" gradients during numerous learning iterations. To improve a range of neural networks, Adam combines the benefits of two different stochastic gradient approaches, adaptive gradients and root mean square propagation.

### 4.3.5 Max pooling

An example of a pooling procedure is "max pooling," which selects the highest number of elements from the omitted region of the feature map. As a consequence of this, the feature map that would be produced as an outcome for the max-pooling layer might be one that had the most significant aspects of the previous feature map.

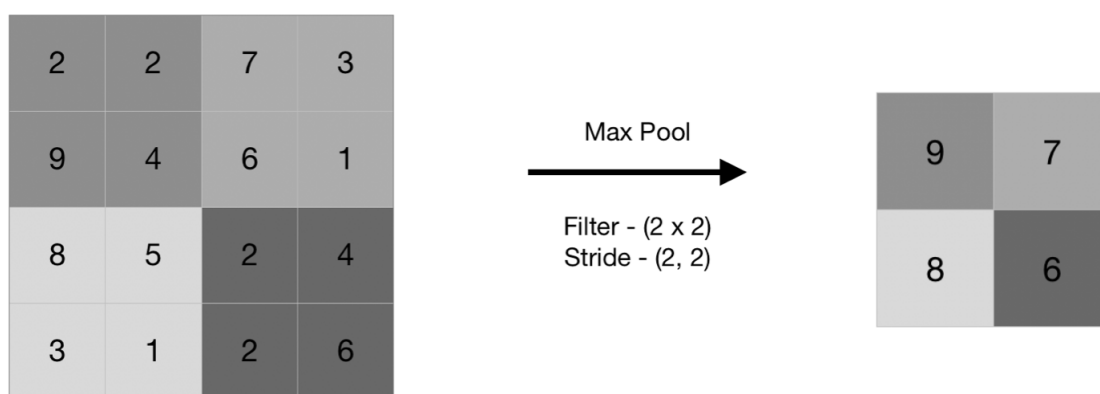


Figure 4.4: Example of Max pooling Layer

### 4.3.6 Average Pooling

The average pooling method figures out what the mean result represents for each component in the part of the feature map for which the filter is functioning. Therefore, maximum pooling will offer you the feature that is the most noticeable in a certain patch of the feature map, while average pooling will give you an average of all the features that are present in a patch.



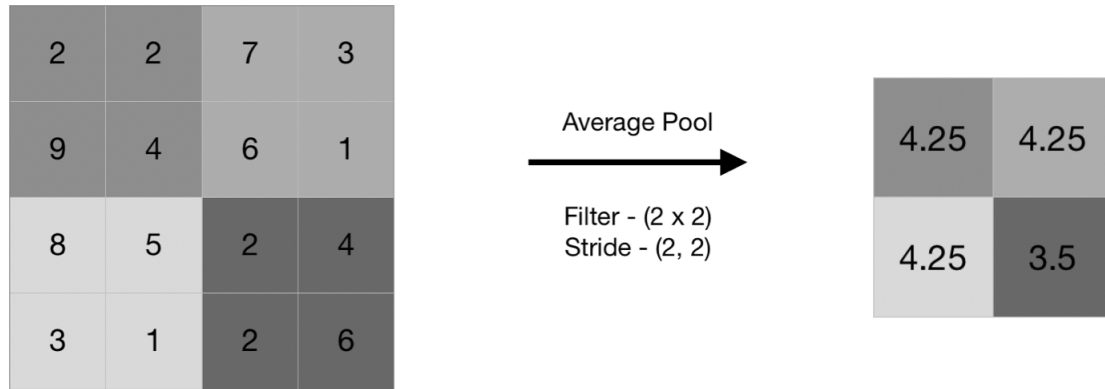


Figure 4.5: Example of Average pooling Layer

### 4.3.7 Learning Rate scheduler

Learning rate is a hyperparameter in a neural network that needs to be tested and tuned. Learning rate regulates how fast the weights of a neural network should be updated. It is good to decrease the learning rate gradually to achieve better performance. We used an initial learning rate .0002 and decreased it to .0001 from epoch 28 and finally decreased it to .00005 from epoch 46 to 60. Learning rate had a huge impact on the model performance.

# Chapter 5

## Performance Analysis

The purpose of performance analysis is to validate the novel technological applications that have been utilized to improve performance. This gives us an overview of the entire work and shows the accomplishments and shortcomings of a specific task, allowing us to learn new techniques and improve old ones. It can also be used to assess the merits and shortcomings of others. The most significant parts of performance evaluation are tactical and technical evaluation, movement analysis, and data collection.

### 5.1 Performance Parameter

To evaluate and compare these models, the F1 score, precision, accuracy, and recall were calculated. This was done to analyze, examine, and highlight a comparison research of conventional and pre-trained CNN model conclusions. This part begins with the performance indicator formulae utilized throughout the inquiry. The confusion matrix gives precision, recall, f1-score, and accuracy. Total correct predictions divided by total predictions give testing accuracy. However, each category should calculate precision, recall, and f1-score.  $M$  is the confusion matrix and  $M_{ij}$  is the number (predictions) in the cell on the  $i$ th row and  $j$ th column.

$$Accuracy = \frac{\sum_{i=1}^4 M_{ii}}{total\ predictions} \quad (5.1)$$

$$Precision_i = \frac{\sum_{i=1}^4 M_{ii}}{\sum_{j=1}^4 M_{ji}} \quad (5.2)$$

$$Recall_i = \frac{\sum_{i=1}^4 M_{ii}}{\sum_{j=1}^4 M_{ij}} \quad (5.3)$$

$$F1 = \frac{2 * Precision * Recall}{Precision + Recall} \quad (5.4)$$

## 5.2 Performance of Pre-trained Model

In our study, we showed a loss and accuracy comparison for each epoch for the training and testing phases. It allows us to see how well our model performs in each epoch of training and testing. It shows if the model is overfitting or underfitting.

### 5.2.1 VGG16:

We were able to get a testing accuracy of 98 percent using the VGG16 model. In Figure, the accuracy and loss graphs for VGG16 are shown. We saw that the loss decreased to a sufficient extent over time in both the training and validation loss graphs. It encourages us to believe that regression is not taking place.

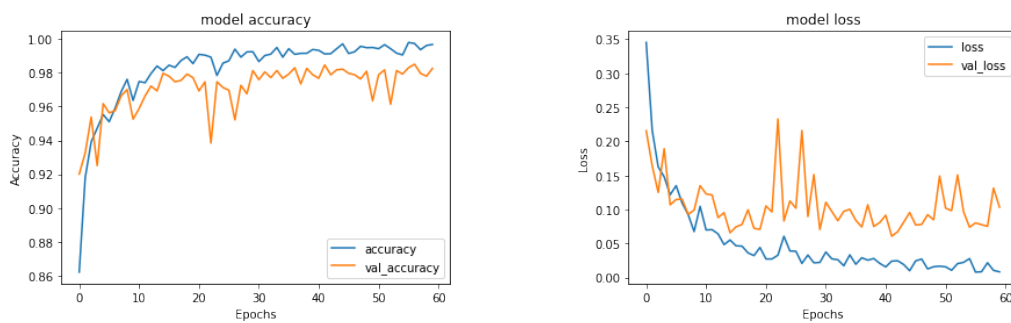


Figure 5.1: Training and validation graph of VGG16

From the confusion matrix we can see, this VGG16 model is also confused between Blight and Gray Leaf Spot. 4.00% of Gray Leaf Spot is being identified as Blight and 1.67% of Blight is being identified as Gray Leaf Spot.

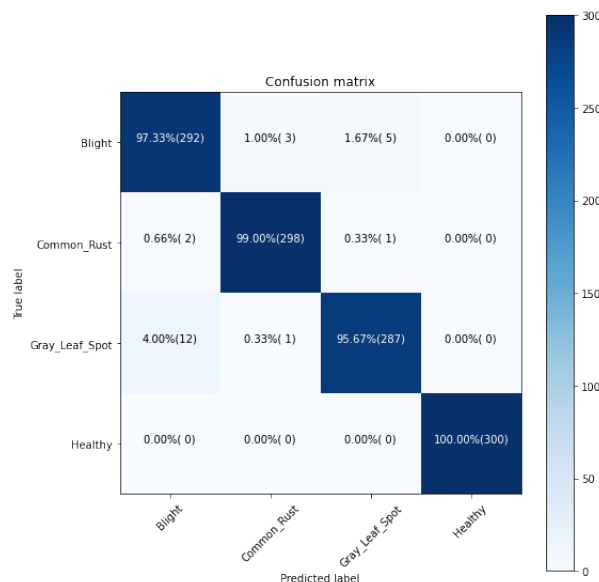


Figure 5.2: Confusion matrix of VGG16

## 5.2.2 VGG19:

We were able to get a testing accuracy of 95.25 percent using the VGG-19 model. In Figure, the accuracy and loss graphs for VGG19 can be seen. In the training loss graphs and validation loss graphs, we saw that the loss decreased over time with low bias and low variance. Due to the convergence of the training and testing curves, we can therefore estimate the acceleration and declination of VGG19. There are irregularities in the validation curve from time to time due to the volume of data presented.

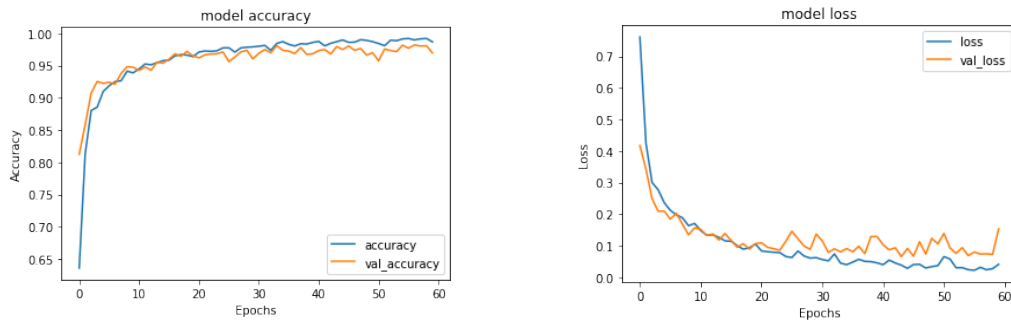


Figure 5.3: Training and validation graph of VGG19

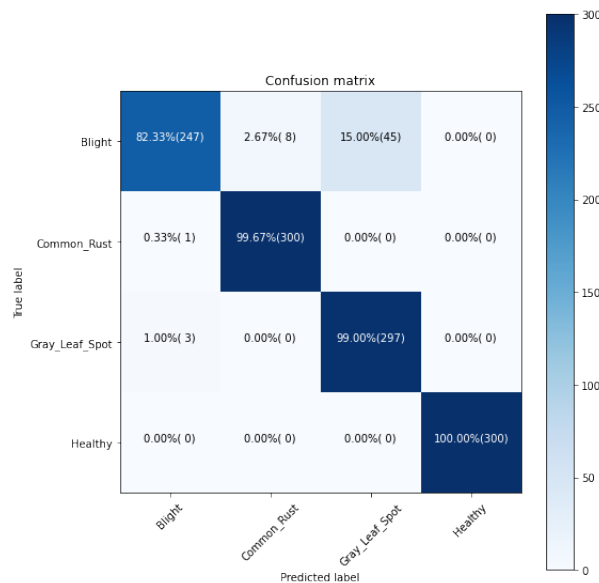


Figure 5.4: Confusion matrix of VGG19

From the confusion matrix, we can see that the model VGG19 is also confused between Blight and Gray Leaf Spot, with a percentage of 15.00% Blight identified as Gray Leaf Spot. Moreover, 2.67% of Blight is identified as Common Rust. Then again, 1.00% of Gray leaf Spot is identified as Blight.

## 5.2.3 ResNet50:

The ResNet50 model achieved a testing accuracy of 96.58 percent. As for ResNet50, Figure illustrates the accuracy and loss graphs. This graph shows subtle variations

that represent differences between the loss value training and validation curves. The curves also display the Resnet50 model's accuracy throughout training and testing as well as its loss.

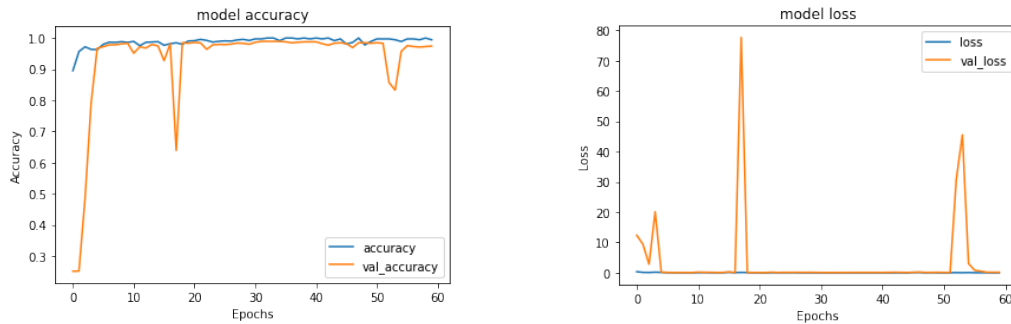


Figure 5.5: Training and validation graph of ResNet50

the confusion matrix, we can see that the model ResNet50 is also confused between Blight and Gray Leaf Spot Mostly, with 7.00% of Gray Leaf Spot identified as Blight. The confusion matrix for the true table Gray Leaf Spot shows that the model correctly identified 279 out of 300 test images, and the rest (21) were mis-predicted as Blight. On the other hand, 2.33% of Blight is identified as Gray Leaf Spot. Moreover, 1.99% of Common Rust is identified as Gray Leaf Spot.

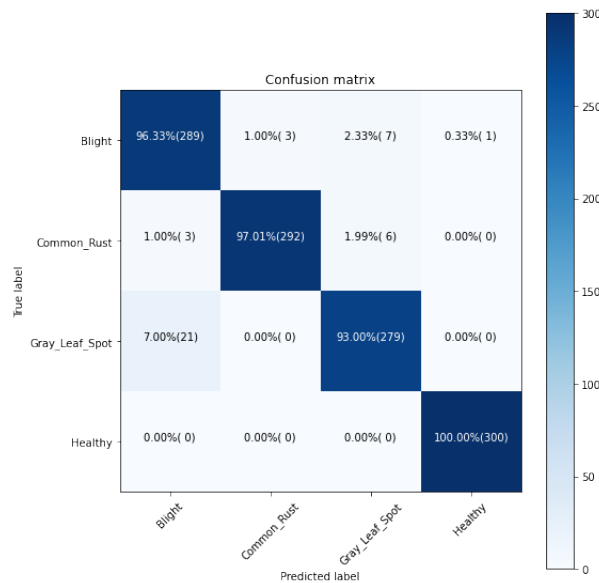


Figure 5.6: Confusion matrix of ResNet50

## 5.2.4 MobileNetV2:

The testing accuracy that was able to be achieved with the MobileNetV2 model in this case was 96.58 percent. Figure illustrates the accuracy and loss graphs for MobileNetV2. The training and testing accuracy and loss of the MobileNetV2 model are also shown in the Figure. The validation curve occasionally varies as a result of the volume of data offered.

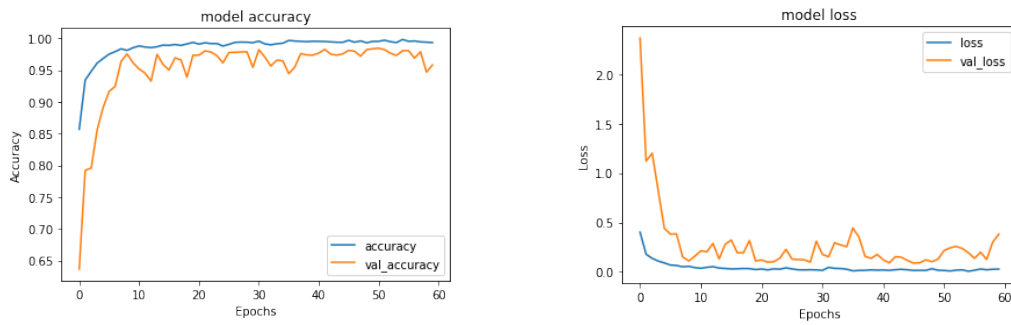


Figure 5.7: Training and validation graph of MobileNetV2

From the confusion matrix, we can see that the model MobileNetV2 is mostly confused between Blight and Gray Leaf Spot, with 6.00% of Gray Leaf Spot identified as Blight. Then again, 6.00% of Blight is identified as Gray Leaf Spot.

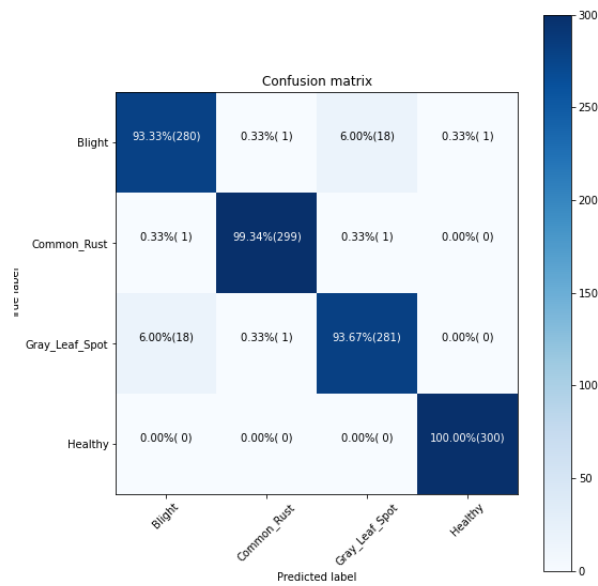


Figure 5.8: MobileNetV2 confusion matrix

### 5.2.5 InceptionV3:

The testing accuracy using the InceptionV3 model was found to be 85.17 percent. Accuracy and loss graphs for the InceptionV3 model are shown in Figure. Despite some small variances, the calibration and testing curves converge. As the InceptionV3 model is trained and validated, the loss values decrease in a linear fashion. The Inception V3 model’s accuracy during training and testing has dropped because of the curves.

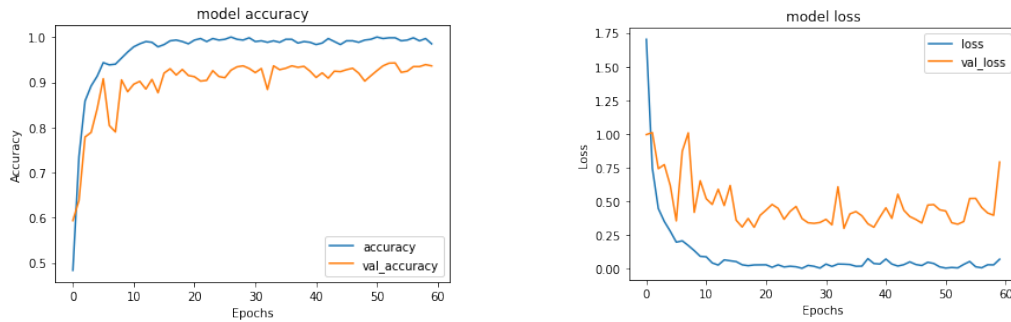


Figure 5.9: Training and validation graph of InceptionV3

This InceptionV3 model, like other models, erroneously distinguishes between Gray Leaf Spot and Blight. Gray leaf spot is being classified as blight in 11.00% of cases.

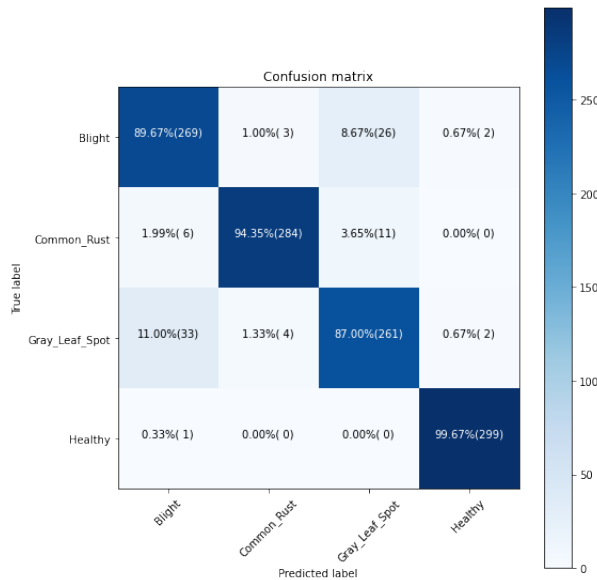


Figure 5.10: Confusion matrix of InceptionV3

### 5.2.6 InceptionResNetV2:

In testing, the Inception ResNetV2 model’s accuracy was 98.58 percent. The figure shows the accuracy and loss graphs for InceptionResNetV2. The differences between the loss value training and validation curves are represented by the graph’s

slight variations. The curves also display the InceptionResnetV2 model's accuracy throughout training and testing as well as its loss.

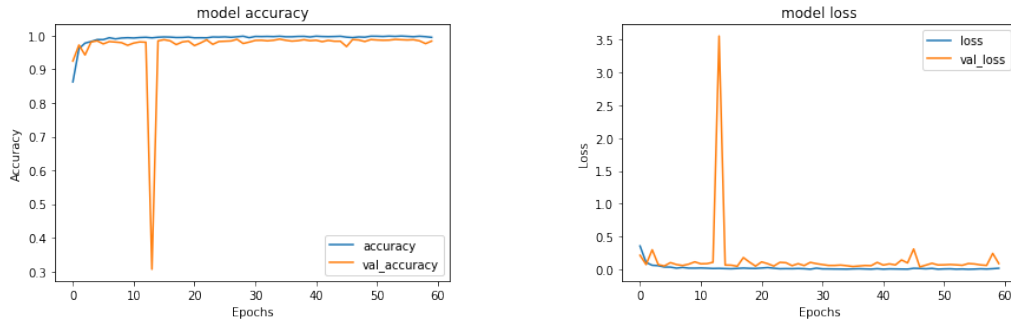


Figure 5.11: Training and validation graph of InceptionResNetV2

From the confusion matrix we can see, the InceptionResNetV2 model is mostly confused between Blight and Gray Leaf Spot, with 3.33% of Gray Leaf Spot identified as Blight. Moreover, 0.67% of Gray Leaf Spot is identified as Common Rust. Then again, 0.67% of Blight is identified as Gray Leaf Spot.

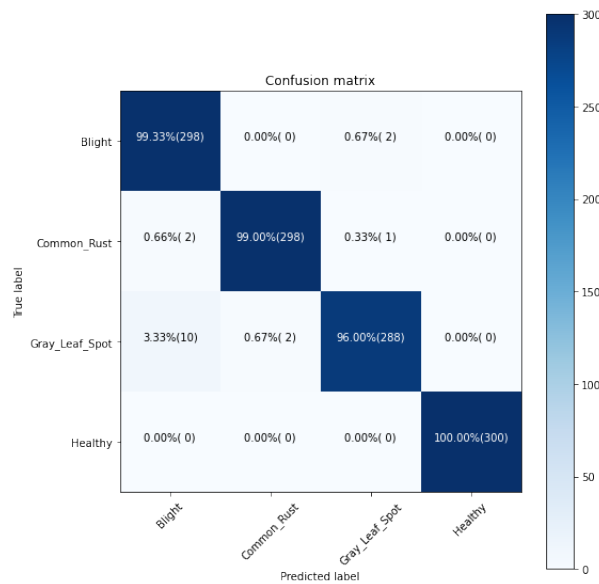


Figure 5.12: InceptionResNetV2 confusion matrix



### 5.2.7 DenseNet201:

Using the DenseNet201 model, a trial efficiency score of 99.17 percent was attained. Figure depicts the DenseNet201 model’s testing and training effectiveness as well as its cost. The differences between the loss value training and validation curves are represented by the graph’s fluctuations. The graphs also depict the training and testing precision as well as the loss of the DenseNet201 model.

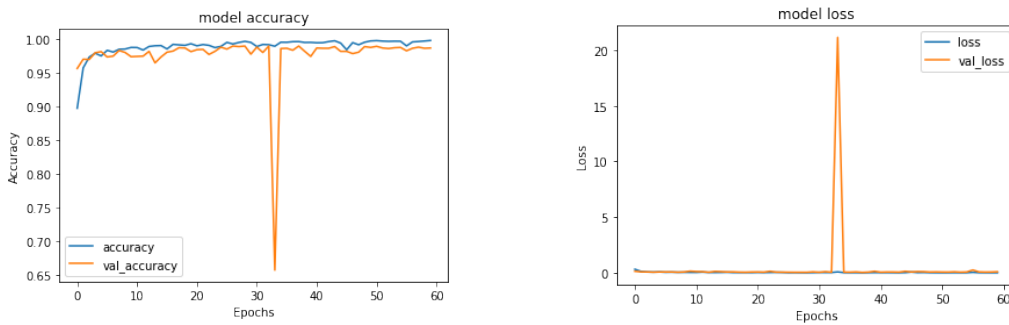


Figure 5.13: Training and validation graph of DenseNet201

From the confusion matrix, we can see that the model DenseNet201 is confused between Blight and Gray Leaf Spot, with 1.00% of Gray Leaf Spot identified as Blight. Then again, 0.67% of Blight is identified as Gray Leaf Spot.

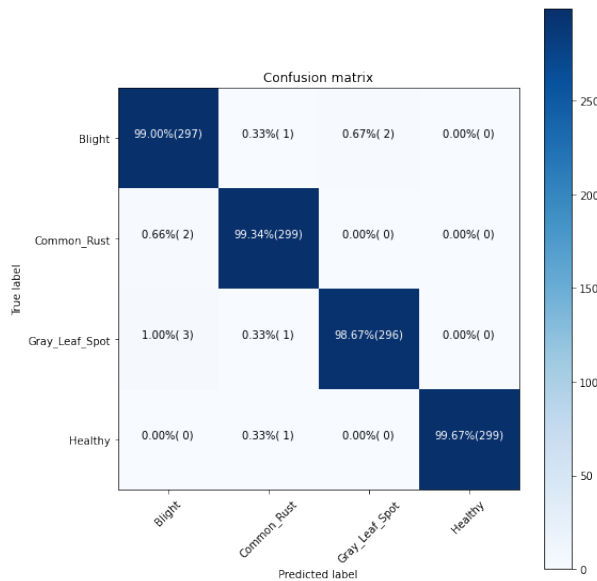


Figure 5.14: Confusion matrix of DenseNet201

## 5.3 Performance of Proposed Model

### 5.3.1 Custom Model

The proposed model is a lightweight, high-performance model. Like pre-trained models it also jumps to a high training accuracy very quickly. The training accuracy keeps increasing while the training loss keeps decreasing. On the other hand, the validation accuracy starts with a low value, 0.2506. This is because the neural network starts with random weights. Weights get optimized with epochs. For example, DenseNet201 starts with a high validation accuracy because its weights are already optimized. The validation accuracy also gets better as we run more epochs and validation loss gradually decreases. Till epoch 27, learning rate .0002 was used and from epoch 28 learning rate was reduced to .0001 with a lr-scheduler. That is when both the training accuracy and validation accuracy jumps to a better value. It can be seen from the graphs also. Moreover, that is when both the training and validation loss reached their lowest value so far (till epoch 28). The network keeps getting better after that with few ups and downs. From epoch 46, the learning rate was reduced to .00005. It did not immediately increase the validation accuracy but later in epoch 54 it reached the highest validation accuracy of 98.46%. Besides training accuracy, validation accuracy is also important. Validation accuracy determines how the network will perform with new data. It is used to fine tune the hyperparameters. To reduce overfitting, it is necessary to have a good validation accuracy also. From the graph we can see the proposed model converges to a good training and validation accuracy. It took an average of 38.5 seconds per epoch and overall training time was 38 minutes. The testing accuracy of this model was 99.17

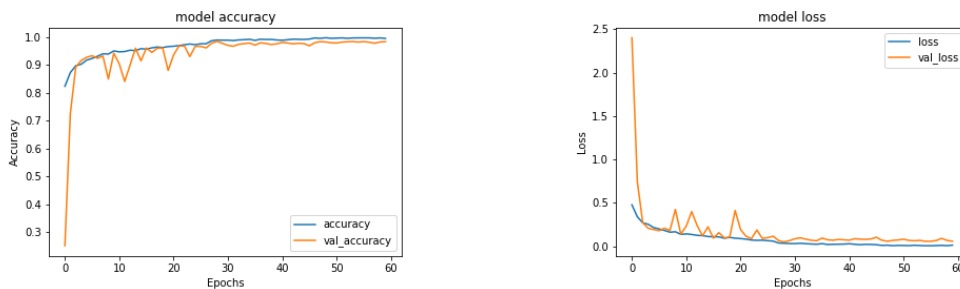


Figure 5.15: Training and validation graph of Custom Model

A confusion matrix is generated for each model where the row defines the actual label and the column defines the predicted label. Each cell of the matrix has number of samples along with the percentage. For example, the cell in row Blight and Column Blight has a number 296 and a percentage 98.67. It means 296 samples identified as Blight are actually Blight. The percentage can be found by dividing 296 by the total number of samples which is 300 for Blight. From the confusion matrix, it can be said that the model is finding Blight & Gray Leaf Spot the most difficult to identify. On the other hand, identification for healthy leaves is 100% like most other models. The model is mostly confused in identifying Blight & Gray Leaf Spot. 1.33% of Gray Leaf Spot is identified as Blight.

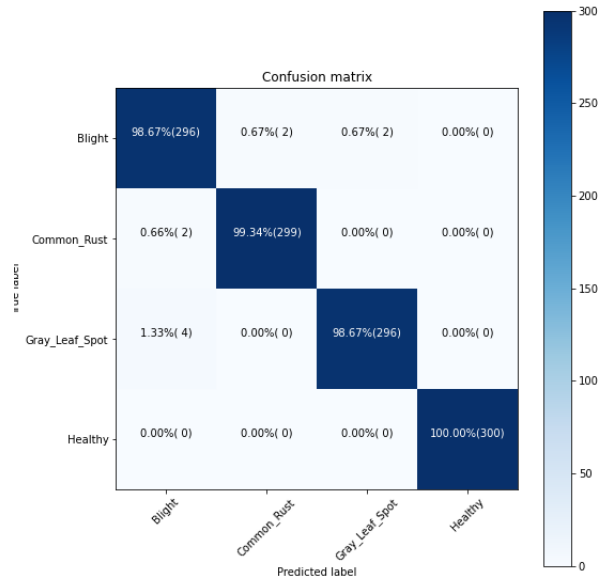


Figure 5.16: Confusion matrix of Custom Model

In our thesis, we used the sklearn library to generate a classification report for our custom deep convolutional neural network model for the detection of corn leaf diseases. The report includes several metrics, including precision, recall, and F1 score.

	precision	recall	f1-score	support
Blight	0.98	0.99	0.98	300
Common_Rust	0.99	0.99	0.99	300
Gray_Leaf_Spot	0.99	0.99	0.99	300
Healthy	1.00	1.00	1.00	300
accuracy			0.99	1200
macro avg	0.99	0.99	0.99	1200
weighted avg	0.99	0.99	0.99	1200

Figure 5.17: Classification Report of Custom Model

## 5.4 Result comparison:

We tried 8 models: InceptionResNetV2, MobileNetV2, ResNet50, VGG19, InceptionV3, VGG16, DenseNet201 and Custom model. Among these, DenseNet201 and our proposed Custom Model both have the highest accuracy of 99.17%. Though the DenseNet201 also has the same accuracy as the Custom Model, the Custom model will perform better as it is lightweight and has a lower number of parameters. The Custom Model has been fed only the precise and relevant datasets, so it has become a lightweight model. Furthermore, the parameters of DenseNet201 and Custom Model are 42, 178,692, and 4,196,100, respectively. Because there are fewer

parameters in the Custom Model, the accuracy will be higher. According to Table, even though DenseNet201 has the same accuracy, the proposed custom model is the most performant architecture among the designs tested so far in this study.

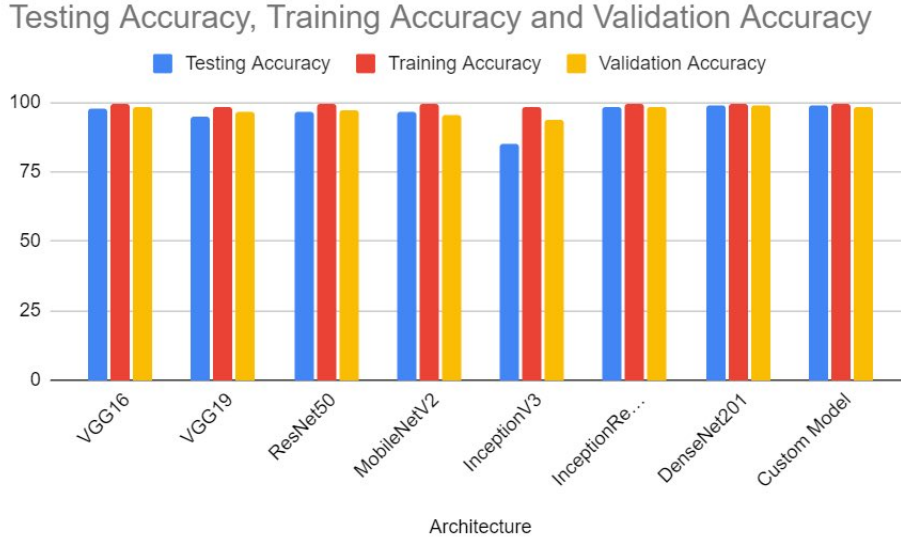


Figure 5.18: Accuracy Comparison

From the table, we can observe that our proposed Custom Model has a lower number of trainable parameters in comparison to other pre-trained models. As a result, our custom model can provide higher accuracy and better performance among all of them.

Architecture	Total Parameters	Trainable Parameters	Non Trainable Parameters
VGG16	26,448,196	26,448,196	0
VGG19	21,138,500	21,138,500	0
ResNet50	49,279,108	49,225,988	53,120
MobileNetV2	21,138,500	21,138,500	0
InceptionV3	34,911,268	34,876,836	34,432
InceptionResNetV2	64,168,420	64,107,876	60,544
DenseNet201	42,407,748	42,178,692	229,065
Custom Model	4,199,108	4,196,100	3,008

Table 5.1: Number of Parameters

Precision is a measure of how many correctly predicted outcomes there are (true positives). We can notice that there are just slight changes when we compare the precision rates of Custom Model with DenseNet201. The rate can occasionally be slightly greater than the DenseNet201. For instance, the Custom Model has a rate of 99.03%, which is extremely close to the precision rate of Common Rust for DenseNet201 at 99.01%. However, when it comes to Healthy, both models have the same rate.

<b>Architecture</b>	<b>Blight</b>	<b>Common Rust</b>	<b>Gray Leaf Spot</b>	<b>Healthy</b>
VGG16	95.42	98.68	97.95	100.0
VGG19	98.41	97.4	86.84	100.0
ResNet50	92.33	98.98	95.55	99.67
MobileNetV2	93.65	99.34	93.67	99.67
InceptionV3	87.06	97.59	87.58	98.68
InceptionResNetV2	96.13	99.33	98.97	100.0
DenseNet201	98.34	99.01	99.33	100.0
Custom Model	97.05	99.33	99.32	100.0

Table 5.2: Precision

Recall is a measure that expresses the proportion of positive cases correctly predicted by the classifier out of all positive examples in the data. When we compare the recall rates of the Custom Model with DenseNet201, we can see that there are only very few differences similar to precision rates. For example, the recall rate of Gray Leaf Spot for DenseNet201 is 98.67%, whereas the rate of the Custom Model is 98.0%, which is very similar to it. Custom Model got 100% of recall rate in the category of Healthy, compared to DenseNet201's 99.67%.

<b>Architecture</b>	<b>Blight</b>	<b>Common Rust</b>	<b>Gray Leaf Spot</b>	<b>Healthy</b>
VGG16	97.33	99.0	95.67	100.0
VGG19	82.33	99.67	99.0	100.0
ResNet50	96.33	97.01	93.0	100.0
MobileNetV2	93.33	99.34	93.67	100.0
InceptionV3	89.67	94.35	87.0	99.67
InceptionResNetV2	99.33	99.0	96.0	100.0
DenseNet201	99.0	99.34	98.67	99.67
Custom Model	98.67	99.0	98.0	100.0

Table 5.3: Recall

F1 score is balancing precision and recall on the positive class. It is a measurement that combines recall and precision. Again, we can see that there is only very little difference, when we compare the F1 Score rates of Custom Model with DenseNet201. Both the models, Custom Model and DenseNet201 have the same rate for Common Rust which is 99.17%.

<b>Architecture</b>	<b>Blight</b>	<b>Common Rust</b>	<b>Gray Leaf Spot</b>	<b>Healthy</b>
VGG16	96.37	98.84	96.8	100.0
VGG19	89.66	98.52	92.52	100.0
ResNet50	94.29	97.99	94.26	99.83
MobileNetV2	93.49	99.34	93.67	99.83
InceptionV3	88.34	95.95	87.29	99.17
InceptionResNetV2	97.7	99.17	97.46	100.0
DenseNet201	98.67	99.17	99.0	99.83
Custom Model	97.85	99.17	98.66	100.0

Table 5.4: f1-score

## 5.5 Accuracy Comparison on Related Work

In the table that follows, a comparison has been made that is supported by the level of correctness that these thesis papers achieved. They were able to detect maize leaf disease in the first article with an accuracy of 98.06% using an improved dense convolutional neural network model. In the second paper, we can find that upgraded deep convolutional neural networks can identify maize leaf illnesses with an accuracy of about 98.8%. For real-time disease detection in maize plants, the third paper employed Deep Convolutional Neural Network approach and attained an accuracy of 88.46%. The following article discusses a maize disease identification technique based on upgraded ResNet50 that has a 98.52% accuracy rate. In the following article, CNN demonstrated the accuracy of 97.01% using a mix of data augmentation and transfer learning. An 84% accurate forward chaining approach was used in the sixth article. In the seventh publication, they assessed deep transfer modeling for maize leaf disease detection, which had the best accuracy rate of about 99.16%. The following article uses deep neural networks to identify maize diseases with an accuracy of 98% in order to support agricultural productivity. The final entry in the table discusses CNN's approach to digital agricultural systems support for disease detection in maize leaves.

<b>Approaches</b>	<b>Methods</b>	<b>Accuracy</b>
Our Paper	Custom Model	99.17%
“An optimized dense convolutional. . . in corn leaf” [11]	Custom densenet	98.06%
“Identification of maize leaf. . . convolutional neural networks” [1]	GoogLeNet	98.8%
“Deep convolutional neural network. . . corn plant disease recognition” [6]	Deep Convolutional Neural Network	88.46%
“Research on maize disease. . . improved resnet50” [13]	ResNet50	98.52%
“The identification of corn leaf. . . data augmentation” [5]	CNN based on a combination of data augmentation and transfer learning	97.01%
“Implementation of forward chaining. . . muara tami district” [15]	Forward chaining method	84%
“Corn disease detection. . . the crop yield” [14]	ANN	98%
“A cnn approach for. . . agricultural system” [8]	CNN	98.78%

Table 5.5: Comparison on related work

In contrast, our custom model has been used to build deep convolutional neural network methods for identifying maize leaf illnesses, and it has an accuracy of 99.17%, which is greater than any other similar papers above. The better accuracy rate of this model shows that, due to its portability and lightweight, it can quickly and reliably identify any corn disease. This model can assure to do better in case of results than the others at disease detection.

## 5.6 Application of our Proposed Model

We made an app with flutter framework. We have chosen flutter because it is cross-platform. We have a client-server architecture to let the server do the heavy lifting while the client UI can go smoothly. We have an api built with python FastApi library. FastApi is a lightweight, no batteries included python library. We used keras save model to save the model into .h5 format. And load it into the FastApi server instance. The model is loaded only once and will be used in subsequent api calls. There are two endpoints one endpoint is for welcome message and another is for uploading corn leaf pictures. HTTP POST method should be used to call the prediction endpoint. The server will get the image from POST and preprocess it just like it was done with the training images. Converting it to RGB and rescale it to 224x224. Then the server will call the predict function and get probabilities

for each category. The server will reply with a JSON containing class names and probabilities.

The frontend is made with flutter where images can be uploaded from files or from camera. It interacts with the server and shows the result in a good UI.

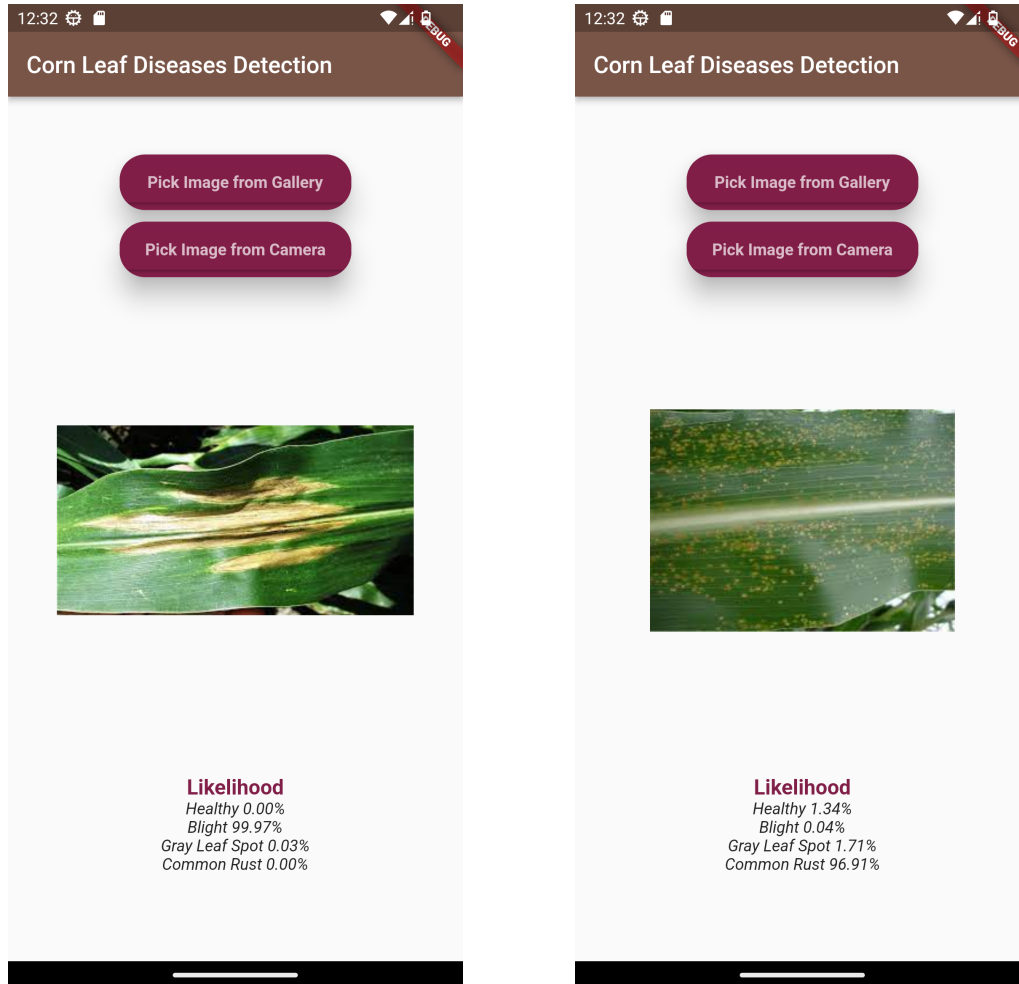


Figure 5.19: Mobile App prediction



# Chapter 6

## Conclusion

### 6.1 Conclusion

Corn is one of the most important and demanded crops in today's world. However, the production of this crop is decreasing day by day because of excessive diseases of corn and corn leaves. In some cases, identifying the primary disease is becoming a challenge for the researchers. In this paper, we aim to identify the diseases of corn leaves using computer vision, machine learning, deep learning and image processing. We have used a deep CNN model that can predict 3 classes of corn leaf diseases along with 1 healthy class in this research. The 3 classes include Blight, Common Rust, Gray Leaf Spot. The model included almost 12000 augmented images and performed well by achieving the highest accuracy of 99.17% using our proposed model. Our proposed methodology can provide a systematic, practical, and efficient way of corn leaf disease identification which will lead the way towards creating an efficient digital agricultural system. This paper also illustrates the importance of tuning hyperparameters like learning rate. Moreover, an lightweight model with low number of parameters is presented in this paper. With some changes, this model might be successful in identifying other leaf diseases.

### 6.2 Future Work

Our thesis research and development on corn leaf disease detection using deep convolutional neural networks yielded encouraging results. However, there are some areas where further work may be done to increase the model's accuracy and resilience.

Expanding the dataset used to train the model will be one of the most important areas of future endeavor. The existing dataset of 12,000 photos is restricted in terms of illness representation and image diversity. The algorithm can be trained to detect a wider range of diseases by collecting additional photos that include a wider range of diseases and differences in leaf appearance.

The program is currently trained to detect only three distinct illnesses of maize leaves. However, many additional diseases can harm corn leaves, and it would be good to broaden the model's detection capabilities to detect a broader spectrum of diseases. This can be accomplished by gathering more photos of various diseases and fine-tuning the model to detect them.

Another topic of future research will be to investigate various neural network designs. This thesis' model made advantage of pre-trained architectures that were

fine-tuned for our specific goal. Other topologies, such as recurrent neural networks or graph neural networks, may be more suitable for this task.

In addition, we will attempt to boost the variety of the training dataset via GAN-based augmentation. Generative Adversarial Networks (GANs) can be used to create new images that are similar to the original dataset but have different leaf attributes. This can aid in improving the model's capacity to generalize to new images and preventing overfitting.

## 6.3 Challenges

As we did research and worked on our thesis, we ran into a number of problems. One of the biggest problems was that there wasn't enough computing power.

Deep convolutional neural networks, like the ones our model uses, take a lot of computing power to train and run. The fact that there were 12,000 images added to the amount of work that had to be done. We had to use powerful hardware, like GPUs, to keep up with the amount of computing work. But this was still hard to do because the training process took a long time and needed a lot of computer resources.

Finding the best hyperparameters for the model was also hard and took a lot of trial and error.

Lastly, making a mobile app that is easy to use, accessible, and user-friendly was a challenge in and of itself. We had to make sure that the app's user interface was easy to understand and that it worked well on a vast variety of devices.

# Bibliography

- [1] X. Zhang, Y. Qiao, F. Meng, C. Fan, and M. Zhang, “Identification of maize leaf diseases using improved deep convolutional neural networks,” *Ieee Access*, vol. 6, pp. 30 370–30 377, 2018.
- [2] M. Al-Amin, D. Z. Karim, and T. A. Bushra, “Prediction of rice disease from leaves using deep convolution neural network towards a digital agricultural system,” in *2019 22nd International Conference on Computer and Information Technology (ICCIT)*, IEEE, 2019, pp. 1–5.
- [3] J. A. Pandian and G. Geetharamani, “Data for: Identification of plant leaf diseases using a 9-layer deep convolutional neural network,” *Mendeley Data*, vol. 1, p. 2019, 2019.
- [4] M. A. Al Mamun, D. Z. Karim, S. N. Pinku, and T. A. Bushra, “Tlnet: A deep cnn model for prediction of tomato leaf diseases,” in *2020 23rd International Conference on Computer and Information Technology (ICCIT)*, IEEE, 2020, pp. 1–6.
- [5] R. Hu, S. Zhang, P. Wang, G. Xu, D. Wang, and Y. Qian, “The identification of corn leaf diseases based on transfer learning and data augmentation,” in *proceedings of the 2020 3rd international conference on computer science and software engineering*, 2020, pp. 58–65.
- [6] S. Mishra, R. Sachan, and D. Rajpal, “Deep convolutional neural network based detection system for real-time corn plant disease recognition,” *Procedia Computer Science*, vol. 167, pp. 2003–2010, 2020.
- [7] S. Mishra, R. Sachan, and D. Rajpal, “Deep convolutional neural network based detection system for real-time corn plant disease recognition,” *Procedia Computer Science*, vol. 167, pp. 2003–2010, 2020.
- [8] K. P. Panigrahi, A. K. Sahoo, and H. Das, “A cnn approach for corn leaves disease detection to support digital agricultural system,” in *2020 4th International Conference on Trends in Electronics and Informatics (ICOEI)(48184)*, IEEE, 2020, pp. 678–683.
- [9] D. Singh, N. Jain, P. Jain, P. Kayal, S. Kumawat, and N. Batra, “Plantdoc: A dataset for visual plant disease detection,” in *Proceedings of the 7th ACM IKDD CoDS and 25th COMAD*, 2020, pp. 249–253.
- [10] M. Syarief and W. Setiawan, “Convolutional neural network for maize leaf disease image classification,” *Telkomnika (Telecommunication Computing Electronics and Control)*, vol. 18, no. 3, pp. 1376–1381, 2020.

- [11] A. Waheed, M. Goyal, D. Gupta, A. Khanna, A. E. Hassanien, and H. M. Pandey, “An optimized dense convolutional neural network model for disease recognition and classification in corn leaf,” *Computers and Electronics in Agriculture*, vol. 175, p. 105456, 2020.
- [12] M. D. Chauhan *et al.*, “Detection of maize disease using random forest classification algorithm,” *Turkish Journal of Computer and Mathematics Education (TURCOMAT)*, vol. 12, no. 9, pp. 715–720, 2021.
- [13] G. Wang, H. Yu, and Y. Sui, “Research on maize disease recognition method based on improved resnet50,” *Mobile Information Systems*, vol. 2021, 2021.
- [14] C. Ashwini and V. Sellam, “Corn disease detection based on deep neural network for substantiating the crop yield,” *Appl. Math*, vol. 16, no. 3, pp. 423–433, 2022.
- [15] E. Pawan, R. M. Thamrin, W. Widodo, S. H. B. S. H. Bei, and J. J. Luanmasa, “Implementation of forward chaining method in expert system to detect diseases in corn plants in muara tami district,” *International Journal of Computer and Information System (IJCIS)*, vol. 3, no. 1, pp. 27–33, 2022.
- [16] R. K. Singh, A. Tiwari, and R. K. Gupta, “Deep transfer modeling for classification of maize plant leaf disease,” *Multimedia Tools and Applications*, vol. 81, no. 5, pp. 6051–6067, 2022.