

## **ABSTRACT**

Maize is the world's top agricultural crop and a cereal grain that was cultivated in Central America. Maize is called the second biggest cereal crop after rice in Bangladesh. As Bangladesh is a flood prone country and maize roots can go as deep as 6 feet to absorb water and can grow in sandy soil, nowadays farmers are growing more maize crop. Farmers are getting more profit than any other crops. But maize has some common disease that always occur like northern leaf blight, gray leaf spot, common rust, brown spot, mosaic etc. If these diseases can't be detected in time and taken prevention, the loss is irreparable. With the purpose to overcome this situation, we proposed a Convolutional Neural Network (CNN) based approach to detect maize diseases in a short time with high efficiency. CNNs are especially effective at extracting meaningful features from images, learning to associate the extracted features with the corresponding disease labels, applying its learned knowledge to predict the most likely disease present in the leaf and adjusting the learning rate and regularization techniques to improve the model's accuracy, generalization, and robustness. Using 4188 images of three infected types and healthy types this study has achieved an outstanding accuracy rate. The results of the study show that the suggested method detects the presence of maize leaf diseases with an impressive detection rate of 98.36%. On the basis of these detections, timely implementation of appropriate management methods can successfully limit the effects of these diseases on crop quality and quantity.

# TABLE OF CONTENTS

<b>CONTENTS</b>	<b>PAGE</b>
Approval	i
Declaration	ii
Acknowledgements	iii
Abstract	iv
 <b>CHAPTERS</b>	
<b>CHAPTER 1: INTRODUCTION</b>	<b>1-4</b>
1.1 Introduction	1
1.2 Motivation	2
1.3 Rationale of the Study	2
1.4 Research Questions	3
1.5 Expected Output	4
1.6 Project Management and Finance	4
1.7 Report Layout	4
 <b>CHAPTER 2: LITERATURE REVIEW</b>	<b>5-9</b>
2.1 Related Works	5
2.2 Comparative Analysis and Summary	6
2.3 Scope of the Problem	7
2.4 Challenges	8

<b>CHAPTER 3: RESEARCH METHODOLOGY</b>	<b>10-26</b>
3.1 Research Subject and Instrumentation	10
3.2 Proposed Research Procedure	10
3.3 Data Collection Procedure/Dataset Utilized	11
3.4 Project Implementation	12
<b>CHAPTER 4: EXPERIMENTAL RESULTS AND DISCUSSION</b>	<b>27-36</b>
4.1 Experimental Setup	27
4.2 Experimental Results & Analysis	27
4.3 Discussion	36
<b>CHAPTER 5: IMPACT ON SOCIETY &amp; ENVIRONMENT</b>	<b>37-39</b>
5.1 Impact on Society	37
5.2 Impact on Environment	38
5.3 Ethical Aspects	39
<b>CHAPTER 6: SUMMARY, CONCLUSION AND IMPLICATION FOR FUTURE RESEARCH</b>	<b>40-41</b>
6.1 Summary of the Study	40
6.2 Conclusions	41
6.3 Implication for Further Study	41
<b>REFERENCES</b>	<b>42-43</b>
<b>APPENDICES</b>	<b>44</b>
<b>PLAGARISM REPORT</b>	<b>45</b>

## LIST OF FIGURES

FIGURES	PAGE NO
Fig-3.1: Block diagram of proposed research procedure.	10
Fig-3.2: Northern Blight	11
Fig-3.3: Common Rust	11
Fig-3.4: Gray Leaf Spot	11
Fig-3.5: Healthy	11
Fig-3.6: Import Dependencies	12
Fig-3.7: Loading Dataset	13
Fig-3.8: Class Names	14
Fig-3.9: Shuffled Dataset	15
Fig-3.10: Dataset partition function	16
Fig-3.11: Dataset partition	17
Fig-3.12: Preprocessing the train, validation & test dataset	18
Fig-3.13: CNN Model.	20
Fig-3.14: CNN Architecture	21
Fig-3.15: Model Compilation	24
Fig-3.16: Model Training.	25
Fig-3.17: Convolutional Layer.	26
Fig-3.18: Maxpooling Layer.	26
Fig-4.1: Model evaluate	27
Fig-4.2: Training & Validation Accuracy & Loss	28
Fig-4.3: Maize leaves prediction	29

Fig-4.4: Maize leave disease detection using deep learning model.	31
Fig-4.5: Evaluation Metrics	33
Fig-4.6: Performance Metrics	34
Fig-4.7: Confusion Matrix	35
Fig-4.8: Save Model.	35
Fig-4.9: Accuracy comparison graph among research papers and our proposed work.	36

## **LIST OF TABLES**

<b>TABLES</b>	<b>PAGE NO</b>
3.1 Image dataset	11
3.2 Train, test & validation set with a split ratio.	18
3.3 CNN Model Summary	23
4.1 Precision, Recall & F1 Score	33

# CHAPTER 1

## INTRODUCTION

### 1.1 Introduction

Bangladesh is one of the most populated and developing countries in the world. It is developing day by day and its development depends on mostly agriculture. As we know our nation has about 700 small to large rivers. During the monsoon, our country's northern region and numerous other locations flood. As a result, a lot of crops are lost, which causes a great deal of economic loss for both farmers and the nation. Our primary crop for supplying the needs of the nation is rice. Wheat, fruits, vegetables, and other crops are examples. These crops cannot flourish in sandy soil or submerged in water. As a result, the farmers in Bangladesh's northern region previously encountered many challenges and losses due to flooding. However, recently they have started growing maize on their land since it can grow in sandy soil and has roots that can penetrate as deep as six feet to receive water. Additionally, the demand for maize is rising daily. Therefore, because maize is more profitable than rice or other crops, farmers are more interested in producing it.

In an article published in "The Daily Star" on December 9, 2022, it was said that the entire demand for maize that year was 70 to 75 lakh tons, of which, according to the FAO, 48 lakh tons will be produced locally. This amount is 30% more than the average for the previous five years. Maize costs had risen by 25% over the previous year. Thus, maize is now referred to as the second-largest grain crop after rice. As a result, both the demand for and growth of maize are rising. More and more maize leaves are being planted by farmers.

However, the quick growth of maize is being hampered by leaf disease. Experts can take preventative action by observing the symptoms, yet occasionally they become confused because other diseases exhibit the same symptoms. New farmers may run into trouble. Some diseases are bacterial, some are fungal, some are caused by viruses, some are caused by climate change, some are caused by temperature, some are caused by humidity, etc. The growth will be increased if these diseases and their types are identified in time.

As technology developed, machines are now executing many human jobs more effectively & nicely utilizing Artificial Intelligence (AI), Machine Learning (ML), & other techniques. By analyzing photos of maize leaves, we can apply machine learning (ML) to identify the different types of illnesses. Compared to manual detection, it is less expensive. This method will be very helpful to the poor farmers.

The mostly common maize diseases are gray leaf spot, southern leaf blight, northern leaf blight, round spot, common rust, brown spot and mosaic. The primary goal of our

research is to develop a useful disease detection system for maize leaves based on Convolutional Neural Networks (CNN). We constructed a CNN-based model for four types of illnesses (Northern leaf blight, Gray spot, Common rust, and Healthy) and tested it for detection of maize diseases using maize leaf image dataset.

## **1.2 Motivation**

As “Maize” is one of the most important crops and many countries take it as main food, farmers grow it in wide range. But if any disease occurs and farmers cannot detect in time, they will be in loss and also the demand will be higher than production resulting price increase.

Agriculture is one of the primary pillars of human nutrition and progress as we stand at the crossroads of an ever-changing technological landscape. Maize, being a critical staple crop around the world, plays a critical role in ensuring food security and supporting livelihoods. The uncontrolled emergence of diseases in maize fields, on the other hand, has become a huge concern, affecting crop productivity and economic stability for farmers globally.

Deep learning's potential, particularly Convolutional Neural Networks (CNNs), has transformed several sectors, and it is past time to apply this transformative technology to combat the threat of maize leaf diseases. Ensure the future of agriculture. Our motivation stems from our desire to use the power of CNNs to improve the way we diagnose and treat these diseases, ultimately contributing to a more sustainable and safe agricultural future.

If we can detect disease just simply by a picture of leaves, farmers will be able to take measures.

## **1.3 Rationale of the Study**

Maize is one of the most important crops globally, and its productivity can be severely affected by various diseases that affect the leaves.

Traditionally, the identification of plant diseases has relied on manual observation by experts, which can be time-consuming, labor-intensive, and subjective. Moreover, the accuracy of human diagnosis can vary depending on the expertise of the individual. Therefore, there is a need for automated systems that can accurately and efficiently detect and classify diseases in maize leaves.

Deep learning, a subfield of artificial intelligence, has shown great promise in image recognition and classification tasks. By leveraging large amounts of labeled data, deep learning models can learn complex patterns and features that are difficult for traditional

algorithms to capture. This study aims to utilize deep learning techniques to develop an automated system that can accurately identify and classify different types of diseases in maize leaves.

The benefits of using deep learning for maize leaf disease detection are numerous. Firstly, it can significantly reduce the time and effort required for disease diagnosis. By automating the process, farmers and researchers can quickly detect diseases and take timely actions to prevent further spread and minimize crop losses. Secondly, deep learning models have the potential to achieve high accuracy rates in disease identification, thereby reducing the chances of misdiagnosis and providing reliable results. Lastly, an automated system can be easily deployed and used by non-experts, making it accessible to farmers in remote areas who may not have access to specialized plant pathologists.

Overall, the rationale of this study is to leverage the power of deep learning to develop an accurate and efficient system for maize leaf disease detection. By doing so, it aims to contribute to the improvement of maize crop management, increase agricultural productivity, and help mitigate the economic and food security challenges associated with plant diseases.

#### **1.4 Research Questions**

- In comparison to conventional techniques, how effective is deep learning at identifying illnesses of maize leaves?
- What deep learning architecture is most effective in spotting illnesses in maize leaves?
- What deep learning algorithms are the most effective and reliable for detecting maize leaf disease?
- What are the main obstacles and restrictions to applying deep learning to the diagnosis of maize leaf disease, and how may they be overcome?
- How effective are deep learning models at detecting maize leaf disease compared to other cutting-edge methods like image processing algorithms or professional human diagnosis?
- Can deep learning models be used to monitor and diagnose maize leaf diseases in agricultural fields in real-time?



## 1.5 Expected Output

The successful development and implementation of a deep learning-based maize leaf disease detection system would have several significant impacts:

- **Enhanced Disease Management:** The automated detection system will give farmers and other agricultural experts a dependable tool to efficiently check the health of their maize harvests. This would make it easier to implement early intervention strategies, such as focused pesticide applications and effective disease management techniques, which would increase crop health and yields.
- **Cost and Time Savings:** The deep learning system would drastically cut the time and effort necessary for disease diagnosis, enabling farmers to utilize resources more effectively. It would replace or supplement manual disease inspection methods.
- **Increased Food Security:** The deep learning system would drastically cut the time and effort necessary for disease diagnosis, enabling farmers to utilize resources more effectively. It would replace or supplement manual disease inspection methods.

## 1.6 Project Management and Finance

The research project for maize leaf disease detection has been conducted without any external funding or resources.

**Finance:** It is a fully self-funded research project.

## 1.7 Report Layout

- Related Work Study
- Proposed Research Methodology
- Experimental Result and Discussion
- Summary, Conclusion and Future Analysis
- Reference

## **CHAPTER 2**

### **LITERATURE REVIEW**

#### **2.1 Related Works**

Several techniques have been developed to aid in the simple and reliable identification of leaf diseases. Such disorders have been detected and classified using a variety of methods including digital image processing, support vector machines (SVM), artificial neural networks, and other related techniques.

Song et al. proposed an SVM-based technique that identified numerous maize leaf diseases with an accuracy of 89.6% [1]. Although it is not the most efficient approach, this SVM-based classification strategy is suitable for both large and small datasets.

Chen and Wang devised a method for identifying maize leaf disease that used a probabilistic neural network (PNN) and image processing methodology resulting in a 90.4% accuracy [2]. However, the PNN classifier in this scenario suffers from a problem in which increasing the training data reduces precision and analysis performance.

Xu et al. developed a system for identifying maize leaf disease based on dynamic weighted multi-classifier fusion, getting an overall detection rate of 94.71% across seven distinct kinds of corn leaves [3].

Md. Al-Amin et al. proposed a CNN model for identifying potato diseases from potato leaves, which achieved 98.33% accuracy [4].

Dechant et al. used a CNN model to identify northern leaf spot disease in maize, with a 96.7% accuracy [5].

Lu et al. suggested a new method for identifying rice illnesses based on deep convolutional neural networks (CNNs) [20]. The suggested CNNs-based model achieves 95.48% accuracy using the 10-fold cross-validation technique.

G. Wang et al. evaluates deep convolutional neural networks trained on apple black rot images from the PlantVillage dataset. The deep VGG16 model outperforms shallow networks, achieving an accuracy of 90.4% on a hold-out test set.[21]

F. Qin et al. analyzed four alfalfa leaf diseases using pattern recognition algorithms. Digital images were segmented using twelve methods, extracting texture, color, and shape features. The optimal SVM model with 45 key features achieved 97.64% and 94.74% recognition accuracies, while semi-supervised models achieved 80% recognition accuracies.[26]

Dionis A. Padilla et al. explores the use of Convolutional Neural Network and OpenMP for disease identification in corn leaf. The study achieved 93% accuracy in detecting Leaf Blight, 89% in rust, and 89% in spot, and a high percentage of classification using OpenMP. The system's advantages include improved execution time rate. [27]

Qi Zhao et al. in their study used the principal component analysis method and support vector machine were used to identify common diseases on maize leaves, achieving 90.74% accuracy in natural light conditions, indicating practical value. [28]

### **2.3 Comparative Analysis & summary**

A comparative analysis has been given on the related works below:

- The accuracy of the SVM-based approach was 89.6%. SVM-based classification strategies work well with both large and small datasets.
- Image processing with probabilistic neural network (PNN) 90.4% accuracy was achieved. When the training data is increased, the PNN classifier suffers from decreased precision and analysis performance.
- For seven different types of maize leaves, dynamic weighted multi-classifier fusion achieved an overall detection rate of 94.71%. This method employs the merging of many classifiers.
- The CNN model identified potato illnesses from potato leaves with 98.33% accuracy. This indicates CNNs' efficacy in illness detection.
- The CNN model for maize northern leaf spot disease identified the disease with 96.7% accuracy. This demonstrates CNNs' ability to detect specific illnesses in maize.
- Using the 10-fold cross-validation technique, a CNNs-based model for rice diseases obtained 95.48% accuracy. This approach is specifically designed for identifying rice diseases.

In summary, SVM-based techniques, PNN, multi-classifier fusion, CNN models, PCA, and pattern recognition algorithms have all been investigated for maize or different leaf disease diagnosis. The CNN-based models have yielded excellent results, with high accuracies in diagnosing illnesses in maize and potato plants. Each approach has advantages and disadvantages in terms of scalability, performance, and generality. The approach chosen is determined by the problem's specific requirements and restrictions.

### 2.3 Scope of the Problem

The problem of "maize leaf disease detection using CNN" involves developing a system to automatically detect and classify diseases affecting maize plants by analyzing photographs of their leaves. This problem is related to computer vision and machine learning.

Here are some aspects that could be considered within the scope of this problem:

- **Dataset collection:** Creating a broad and representative dataset of images of maize leaves, including healthy and disease-affected leaves.
- **Image pre-processing:** Using image pre-processing techniques such as scaling, normalization, and noise reduction to improve the quality and standardization of the input data for the CNN model.
- **CNN model architecture:** Creating and deploying a convolutional neural network (CNN) model specifically tuned for the detection of maize leaf disease. The model should be able to learn relevant features from the images provided and forecast disease accurately.
- **Training and validation:** The dataset is divided into training, testing and validation sets, and the CNN model is trained using the training set. The model should learn to distinguish between healthy leaves and various illness classes, and its performance should be optimized through an iterative process.
- **Model evaluation:** Evaluating the trained model's performance using relevant assessment metrics such as accuracy, precision, recall, and F1-score. This helps to assess the model's accuracy in identifying and classifying various maize leaf diseases.
- **Deployment and usability:** Developing an interface or program that allows users to simply upload images of maize leaves and receive predictions on disease prevalence. The system should be simple to use and deliver accurate and fast results.
- **Performance optimization:** Investigating approaches for improving model performance, such as data augmentation, hyperparameter optimization, and transfer learning. When it comes to detecting illnesses in maize leaves, optimizing the model can lead to improved accuracy and generalization.
- **Scalability and adaptability:** Designing the system to be applicable to large-scale datasets and adaptive to detect illnesses in maize leaves from various geographies and settings. This may entail taking into account differences in lighting conditions, leaf sizes, and disease presentations.

## 2.4 Challenges

Building a convolutional neural network (CNN) model for maize leaf disease detection can come with several challenges. Here are some key challenges you may encounter:

- ❖ **Limited and imbalanced data:** It can be difficult to obtain a broad and well-labeled dataset of maize leaf photos with various disease kinds. Furthermore, the dataset may have a class imbalance, with some illness classes being underrepresented. This may have an impact on the model's ability to recognize all illness kinds accurately.
- ❖ **Annotation and labeling:** Annotating and labeling maize leaf photos manually for various disease kinds can be a time-consuming and labor-intensive operation. To correctly identify and classify the diseases depicted in the photos, domain expertise is required. It is critical to have accurate and consistent annotations across the dataset when training a dependable model.
- ❖ **Overfitting and generalization:** CNN models are prone to overfitting, particularly when the dataset is small or unevenly distributed. When a model overfits, it learns to memorize the training data rather than capturing general patterns. When the model sees new, previously unseen maize leaf images, it may perform poorly. It is critical to have appropriate generalization while developing a robust model.
- ❖ **Preprocessing and data augmentation:** Preprocessing maize leaf images to improve disease detection features can be difficult. To prepare the data for training, image changes such as cropping, scaling, and normalization must be used. Creating enhanced data, such as rotations, flips, and brightness modifications, can also assist prevent overfitting and improve model performance.
- ❖ **Model architecture and hyper parameter tuning:** Choosing the right CNN architecture and adjusting its hyper parameters can have a big impact on the model's performance. Experimentation and domain knowledge are required to select the appropriate number and kind of convolutional layers, pooling layers, and fully connected layers. Furthermore, optimizing hyper parameters such as learning rate, batch size, and regularization approaches is critical for attaining optimal outcomes.
- ❖ **Computational resources:** Training CNN models for maize leaf disease detection sometimes necessitates substantial computational resources, particularly when dealing with huge datasets or complicated structures. Deep

model training can be time-consuming and may necessitate the use of powerful GPUs or distributed computing resources. Access to appropriate hardware and computational resources might be difficult for individuals or groups with minimal funds.

- ❖ **Deployment and scalability:** After creating a good CNN model, deploying it in real-world applications and ensuring scalability might be difficult. Integrating the model into a real-time image processing application or system that can manage varied workloads necessitates careful engineering and consideration of performance, latency, and dependability.

Addressing these difficulties requires a combination of subject experience, data preparation, experimentation, and rigorous model design. Collaboration with domain experts, transfer learning techniques, and the use of existing resources can all help overcome these challenges and build an effective CNN model for maize leaf disease detection.

## CHAPTER 3

### RESEARCH METHODOLOGY

#### 3.1 Research Subject and Instrumentation

**Research Subject:** The proposed study is about maize leaf disease detection. Many researchers have used several techniques like SVM, Digital Image Processing, PNN, OpenMP, Multi Classifier, Deep learning etc. Though they got a higher accuracy using this techniques, they often require longer convergence time, and the increased number of factors involved tends to reduce the identification rate. In this study, I have used Deep learning Convolutional Neural Network (CNN) algorithm to get a higher accuracy with efficient time limit and good identification rate.

**Instrumentation:** To conduct this research I have used a personal computer having Windows OS, 8GB RAM, Core I5 3<sup>rd</sup> Gen processor. I have used Broadband internet. To perform the research the platform I chose is Jupyter Notebook, Python language, different libraries like tensorflow, keras, OpenCV, Scikit-learn, Numpy, Pandas etc. For writing the research paper MS Word Pro plus 2016 is used.

#### 3.2 Proposed Research Procedure

To conduct this research some basic steps were followed. These steps are data collection, data pre-processing, image labelling, model training, model evaluate, model testing, result analysis etc as shown in Fig-3.1. These are some general steps that were maintained well during the research.

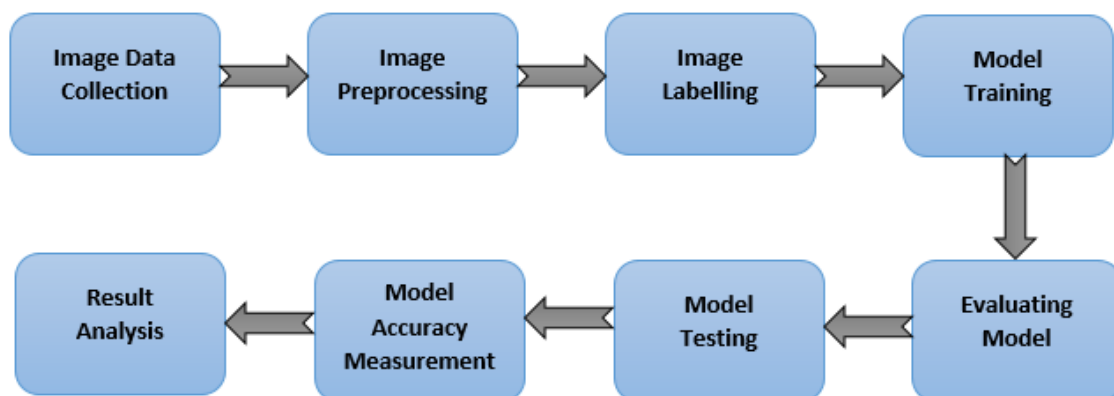


Fig-3.1: Block diagram of proposed research procedure.

### 3.3 Dataset Collection

The dataset we used for this research has been downloaded from Kaggle dataset. This dataset is made using PlantVillage and PlantDoc dataset. Dataset contains total 4188 images including three types of infected maize leaves and healthy leaves as shown in Table -3.1.

**Table-3.1: Image Dataset**

Category	Numbers
Northern Leaf Blight	1146
Common Rust	1306
Gray Leaf Spot	574
Healthy	1162
Total	4188



Fig-3.2: Northern Blight

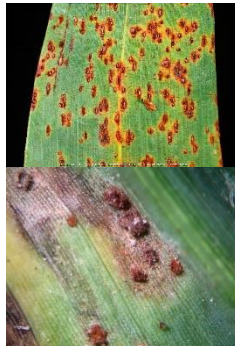


Fig-3.3: Common Rust



Fig-3.4: Gray Leaf Spot



Fig-3.5: Healthy

Fig -3.2, 3.3, 3.4 & 3.5 shows the northern blight, Common Rust, Gray leaf spot & Healthy maize leaves respectively.



### 3.4 Project Implementation

As this is a research based project and the project is implemented on Jupyter Notebook, all the procedure will be described in the following sections.

**3.4.1 Import Dependencies:** At first all the libraries we need are imported as Fig-3.6.

```
In [1]: import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from tensorflow.keras import models, layers
from sklearn.metrics import classification_report
from sklearn.metrics import precision_score, recall_score, f1_score, accuracy_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import ConfusionMatrixDisplay
import pickle
```

Fig-3.6: Import Dependencies

**Tensorflow:** TensorFlow is a Google open-source toolkit and framework for numerical computing and machine learning. It enables the development and deployment of deep learning models that use tensors to represent data. TensorFlow supports deep neural network training and serves as an efficient backend for executing operations on hardware devices such as CPUs, GPUs, and TPUs.

**Numpy:** Numpy is a strong Python numerical computation package. The word "Numpy" is an abbreviation for "Numerical Python." It supports massive, multi-dimensional arrays and matrices, as well as a set of mathematical algorithms for effectively operating on big arrays. Numpy is a programming language that is widely used in scientific computing, data analysis, and machine learning applications.

**Matplotlib:** Matplotlib is a popular Python data visualization package. It offers a comprehensive set of tools and functions for building high-quality static, animated, and interactive visualizations in Python. Matplotlib is intended to be versatile and adaptable, allowing users to construct a wide range of plots, charts, histograms, scatter plots, bar graphs, and other graphics. Matplotlib is widely used in the scientific and data analytic fields for tasks such as data exploration, study presentation, and the creation of publication-quality visuals.

**Pandas:** Pandas is an open-source Python library that provides high-performance data manipulation and analysis features. It is popular for working with structured data, such as tabular or CSV files, and includes a DataFrame object for flexible organization. Pandas can load, save, clean, preprocess, filter, merge, transform, and conduct statistical computations on data. It works nicely with other libraries such as Matplotlib for data visualization and scikit-learn for machine learning tasks.

### 3.4.2 Dataset load

The dataset is loaded to Jupyter Notebook for further research. The dataset is loaded as a tensorflow dataset using keras preprocessing.

```
In [2]: BATCH_SIZE = 32
        IMAGE_SIZE = 150
        CHANNELS=3
        EPOCHS=40

In [3]: dataset = tf.keras.preprocessing.image_dataset_from_directory(
        "G:\Archive\data",
        seed=123,
        shuffle=True,
        image_size=(IMAGE_SIZE, IMAGE_SIZE),
        batch_size=BATCH_SIZE
    )

Found 4188 files belonging to 4 classes.
```

Fig-3.7: Loading Dataset

In the code snippet (Fig-3.7) provided, a dataset is being created using the `tf.keras.preprocessing.image_dataset_from_directory` function. Let's break down the different components and parameters involved:

#### Constants:

**BATCH\_SIZE = 32:** It specifies the number of samples in each dataset batch. Each batch will have 32 images in this example.

**IMAGE\_SIZE = 150:** This setting indicates the appropriate size for scaling images in the dataset. The photos will be downsized to a square form with 150x150 pixel dimensions.

**CHANNELS = 3:** The number of color channels in the photos is represented by the usual RGB (Red, Green, Blue) color channels.

**"G:Archive/data":** This is the location of the image data's directory. The function will look in this directory and its subdirectories for images.

**seed=123:** This is the seed for rearranging the data. It ensures that the data is shuffled consistently each time the code is executed, which aids in repeatability.

**shuffle=True:** This suggests that the dataset be randomized. Changing the order in which the photos are displayed during training by shuffling the data can help improve the model's learning.

**image\_size=(IMAGE\_SIZE, IMAGE\_SIZE):** This specifies the preferred picture size for the dataset's photos. The photos will be scaled to a square form with dimensions of 150x150 pixels in this scenario.

**batch\_size=BATCH\_SIZE:** This determines how many images are in each batch of the dataset. Data batching allows the model to process several photos at the same time, which can enhance training efficiency.

The dataset object that results, named dataset, can be used to train a machine learning model. It will include batches of photos and labels, ready to be put into a TensorFlow model for training.

In Fig-3.7, we can see that 4188 images were found of 4 classes.

```
In [4]: class_names = dataset.class_names
        class_names

Out[4]: ['Blight', 'Common_Rust', 'Gray_Leaf_Spot', 'Healthy']
```

Fig-3.8: Class Names

The dataset has 4 classes as shown in Fig-3.8, the classes are:

1. Blight
2. Common\_Rust
3. Gray\_Leaf\_Spot
4. Healthy

These class names represent different categories or labels for the data in our dataset.

### 3.4.3 Display Shuffled Dataset

The following code is written to see some images of a batch from our dataset if they are shuffled or not.

```
In [6]: plt.figure(figsize=(10, 10))
        for image_batch, labels_batch in dataset.take(1):
            for i in range(9):
                ax = plt.subplot(3, 3, i + 1)
                plt.imshow(image_batch[i].numpy().astype("uint8"))
                plt.title(class_names[labels_batch[i]])
                plt.savefig("Train dataset.png", format="png", facecolor="white")
                plt.axis("off")
```

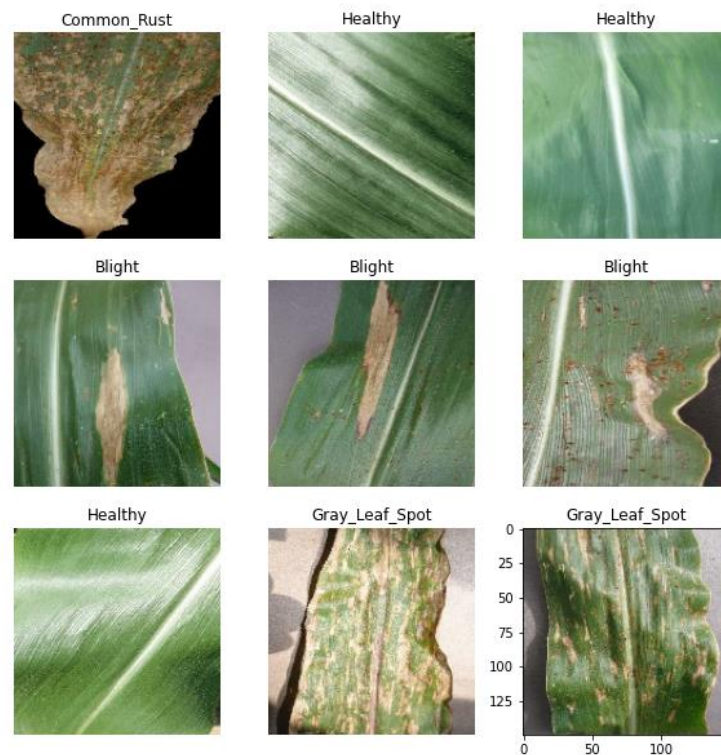


Fig-3.9: Shuffled Dataset

From the Fig-3.9, we can see that the dataset is shuffled well.

### 3.4.4 Dataset Partition

To perform CNN model for training & result analysis we need to split the dataset into train, validation & test dataset. The following function is for dataset partition.

```
In [7]: def get_dataset_partitions_tf(ds, train_split=0.8, val_split=.05, test_split=0.15, shuffle=True, shuffle_size=1000):
        assert (train_split + val_split + test_split) == 1

        ds_size = len(ds)

        if shuffle:
            ds = ds.shuffle(shuffle_size, seed=12)

        train_size = int(train_split * ds_size)
        val_size = int(val_split * ds_size)

        train_ds = ds.take(train_size)
        val_ds = ds.skip(train_size).take(val_size)
        test_ds = ds.skip(train_size).skip(val_size)

        return train_ds, val_ds, test_ds
```

Fig-3.10; Dataset partition function

The code in Fig-3.10 defines a function called `get_dataset_partitions_tf()` that is used to split a TensorFlow dataset into train, validation, and test sets.

The function takes several parameters:

**ds:** The input TensorFlow dataset that needs to be split.

**train\_split:** The proportion of the dataset to allocate for the training set. Defaults to 0.8, meaning 80% of the dataset will be used for training.

**val\_split:** The proportion of the dataset to allocate for the validation set. Defaults to 0.05, meaning 5% of the dataset will be used for validation.

**test\_split:** The proportion of the dataset to allocate for the test set. Defaults to 0.15, meaning 15% of the dataset will be used for testing.

**shuffle:** A boolean value indicating whether to shuffle the dataset before splitting. Defaults to True.

**shuffle\_size:** The number of elements from the dataset to use as a buffer for shuffling. Defaults to 1000.

The function first checks if the sum of **train\_split**, **val\_split**, and **test\_split** is equal to 1, ensuring that the splits add up to the entire dataset.

The size of the input dataset (**ds\_size**) is determined by calculating the length of the dataset (**ds**).

If **shuffle** is True, the dataset is shuffled using the specified **shuffle\_size** and **seed** (set to 12 in this case).

The sizes of the training and validation sets are calculated based on the proportions specified and the size of the dataset.

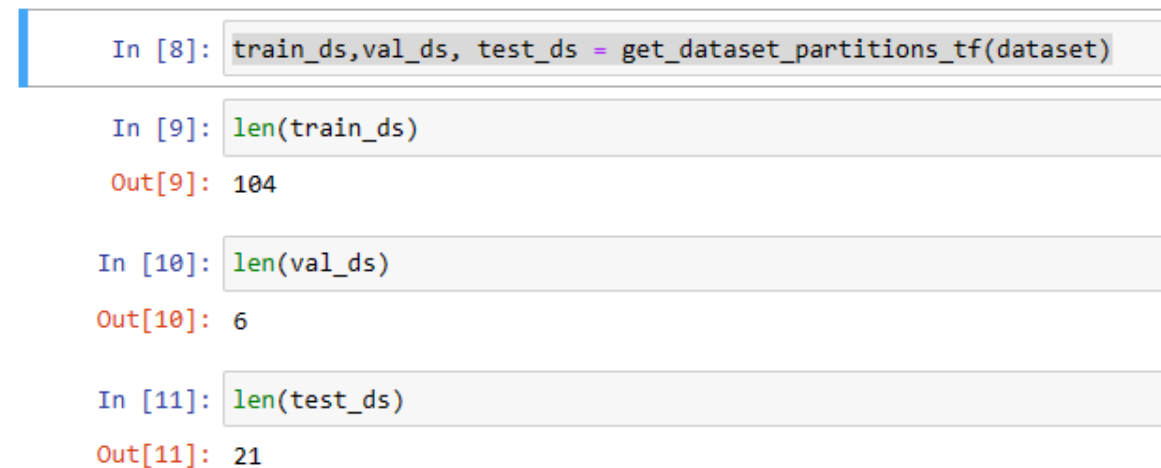
The `take()` and `skip()` methods are used to split the dataset into three parts. `take()` retrieves a specified number of elements from the beginning of the dataset, while `skip()` skips a specified number of elements.

**train\_ds** is assigned the first `train_size` elements of the shuffled dataset, representing the training set.

**val\_ds** is assigned the next `val_size` elements of the shuffled dataset after skipping the training set, representing the validation set.

**test\_ds** is assigned the remaining elements of the shuffled dataset after skipping both the training and validation sets, representing the test set.

Finally, the function returns the three dataset partitions: **train\_ds**, **val\_ds**, and **test\_ds**.



```
In [8]: train_ds, val_ds, test_ds = get_dataset_partitions_tf(dataset)

In [9]: len(train_ds)
Out[9]: 104

In [10]: len(val_ds)
Out[10]: 6

In [11]: len(test_ds)
Out[11]: 21
```

Fig-3.11: Dataset partition

From the above Fig-3.11 we see finally the dataset is partitioned into train, test & validation set using ``get_dataset_partitions_tf()`` function.

The length of train, validation & test dataset are 104,6,21 batches respectively where each batch contains 32 image.

**Table-3.2: Train, test & validation set with a split ratio.**

Category	Training Set	Validation Set	Test Set
Maize leaves	3328	192	668

Table-3.2 shows the amount of dataset in train, test & validation set

### 3.4.5 Dataset Pre-processing

```
In [12]: train_ds = train_ds.cache().shuffle(1000).prefetch(buffer_size=tf.data.AUTOTUNE)
         val_ds = val_ds.cache().shuffle(1000).prefetch(buffer_size=tf.data.AUTOTUNE)
         test_ds = test_ds.cache().shuffle(1000).prefetch(buffer_size=tf.data.AUTOTUNE)

In [13]: resize_and_rescale = tf.keras.Sequential([
         layers.experimental.preprocessing.Resizing(IMAGE_SIZE, IMAGE_SIZE),
         layers.experimental.preprocessing.Rescaling(1.0/255),
         ])
```

Fig-3.12: Preprocessing the train, validation & test dataset

The code provided in Fig-3.12 is related to preprocessing and preparing datasets for training, validation, and testing in a machine learning model.

`train_ds = train_ds.cache().shuffle(1000).prefetch(buffer_size=tf.data.AUTOTUNE):`

**train\_ds.cache()** caches the elements of the dataset in memory or on disk, which can improve training performance by reducing the time it takes to read data.

**train\_ds.shuffle(1000)** shuffles the elements of the dataset with a buffer size of 1000. This randomizes the order of the examples, which is useful during training to prevent the model from learning patterns specific to the order of the data.

**train\_ds.prefetch(buffer\_size=tf.data.AUTOTUNE)** prefetches elements from the dataset, allowing data to be loaded asynchronously while the model is busy with training or inference. The **tf.data.AUTOTUNE** argument dynamically determines the optimal buffer size to use based on available system resources.

`val_ds = val_ds.cache().shuffle(1000).prefetch(buffer_size=tf.data.AUTOTUNE):`

Performs similar operations as described above but on the validation dataset (val\_ds). Caching, shuffling, and prefetching are applied to the validation dataset to prepare it for evaluation during training.

**test\_ds= test\_ds.cache().shuffle(1000).prefetch(buffer\_size=tf.data.AUTOTUNE):**

This prepares the test dataset (test\_ds) similarly to the previous lines by caching, shuffling, and prefetching the data. However, because the test set is often used for evaluation and should ideally represent unobserved data, shuffling the test dataset may not be necessary.

By caching, shuffling, and prefetching the data, these activities assist optimize the data pipeline and improve the training process, resulting in greater utilization of system resources and faster training.

The Figure also defines a TensorFlow Keras sequential model called `resize_and_rescale`. This model applies two preprocessing layers to input images:

**Keras:** Keras is a Python-based open-source deep learning framework that provides a user-friendly and modular interface for creating and training deep neural networks. Keras is popular due to its simplicity and ease of use. It was originally designed as an API on top of existing frameworks such as TensorFlow, Theano, and CNTK. A user-friendly API, modular and extendable features, support for various backends, simple model training and evaluation, and versatility in network topologies such as feedforward networks, convolutional neural networks, and recurrent neural networks are among the key characteristics.

**layers. experimental. preprocessing.Resizing:** This layer resizes the input images to a specified target size. The `IMAGE_SIZE` variable represents the desired size of the images that is 150.

**layers. experimental. preprocessing.Rescaling:** This layer performs rescaling of the pixel values in the range [0, 255] to a normalized range [0, 1]. It divides each pixel value by 255 to achieve this scaling.

Overall, the `resize_and_rescale` model takes an input image and first resizes it to the `IMAGE_SIZE` dimensions, and then scales the pixel values to the [0, 1] range.



### 3.4.6 CNN Model Building for Disease Detection

After preprocessing (resize & rescale), now our dataset is ready for training. We used deep learning technique- Convolutional Neural Network (CNN) for maize leaf disease detection. The CNN model architecture is described below.

```
In [17]: input_shape = (BATCH_SIZE, IMAGE_SIZE, IMAGE_SIZE, CHANNELS)
         n_classes = 4

         model = models.Sequential([
             resize_and_rescale,
             layers.Conv2D(32, kernel_size = (3,3), activation='relu', input_shape=input_shape),
             layers.BatchNormalization(input_shape=input_shape),
             layers.MaxPooling2D((2, 2)),
             layers.Dropout(0.2),
             layers.Conv2D(64, kernel_size = (3,3), activation='relu'),
             layers.BatchNormalization(input_shape=input_shape),
             layers.MaxPooling2D((2, 2)),
             layers.Conv2D(64, kernel_size = (3,3), activation='relu'),
             layers.MaxPooling2D((2, 2)),
             layers.Conv2D(128, kernel_size = (3,3), activation='relu'),
             layers.MaxPooling2D((2, 2)),

             layers.Flatten(),
             layers.Dense(128, activation='relu'),
             layers.Dense(n_classes, activation='softmax'),
         ])

         model.build(input_shape=input_shape)
```

Fig-3.13: CNN Model.

In our proposed model (Fig-3.13) 4 convolutional layers has been used of kernel size (3x3) which has 32,64,64,128 filter respectively. ReLu activation function has been used in all layers except the last one that used softmax activation function. The input\_shape(32,150,150,3) parameter specifies the shape of the input data. 4 MaxPooling layer has been used with a pool size of (2x2). 2 Batch-Normalization layer has been used with input\_shape (32,150,150,3) after 1<sup>st</sup> and 2<sup>nd</sup> Convolutional layer. 1 Dropout Layer has been used of 0.2 rate. 1 flatten layer has been used & 2 fully connected layer has been used with 128 & 4 filter respectively. Model summary is given in Table- 3.3.

**layers.Conv2D(32,kernel\_size=(3,3),activation='relu',input\_shape=input\_shape):**

This line creates a convolutional layer with 32 filters, a kernel size of 3x3, and the ReLU activation function. The input\_shape parameter specifies the shape of the input data.

**layers. BatchNormalization(input\_shape=input\_shape):** This line adds a batch normalization layer to normalize the activations of the previous layer. It helps stabilize the learning process and improve the overall performance of the network.

**layers. MaxPooling2D((2, 2)):** This line adds a max-pooling layer with a pool size of 2x2. Max-pooling reduces the spatial dimensions of the input, effectively downsampling the feature maps.

**layers. Dropout(0.2):** This line adds a dropout layer that randomly sets 20% of the input units to 0 at each training step. Dropout is a regularization technique that helps prevent overfitting by introducing noise during training.

**layers. Flatten():** This line adds a flatten layer to the model. The purpose of this layer is to convert the multidimensional output from the previous layer into a flat (1D) vector. It essentially reshapes the data to prepare it for the subsequent dense layers.

**layers. Dense(128, activation='relu'):** This line adds another fully connected (dense) layer with 128 neurons and the ReLU activation function. It takes the flattened output from the previous layer as its input and applies the activation function to generate a new set of features.

**layers. Dense(n\_classes, activation='softmax'):** This line adds the final dense layer, which is often used as the output layer in classification tasks. The `n_classes` variable represents the number of classes or categories in the classification problem. The activation function used here is softmax, which produces a probability distribution over the classes, indicating the model's confidence for each class prediction.

**model. build(input\_shape=input\_shape),** set the input shape of the model to `input_shape`, which should match the shape of our input data. This step is necessary to define the model's architecture correctly and ensure compatibility between the input shape and the subsequent layers.

### 3.4.7 CNN Model Architecture

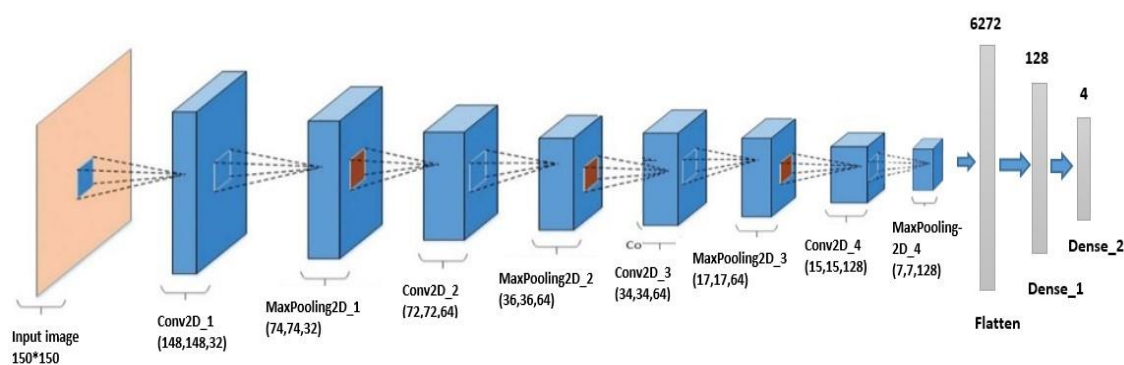


Fig-3.14: CNN Architecture

The functions and architecture used in the model as in Fig-3.14 are further explained below:

**Convolutional layer:** A convolutional layer is an important component of convolutional neural networks (CNNs) used for image and video recognition. It learns local patterns and features by performing convolution operations on input data. The convolution is defined by an image kernel. The image kernel is a small matrix. In our model 3x3 kernel matrix is used. The kernel first moves horizontally, then shift down and again moves horizontally. The output matrix is obtained by adding the kernel and image pixel values in a dot product as shown in Fig-3.17. Convolutional layers have width, height, and depth dimensions and are trained to reduce loss functions using backpropagation. If the convolution layer  $i$  has an input  $X$ , it calculates, as given in Eq. (1).

$$CO = f(W_i * X) \dots \dots \dots (1)$$

The activation function is represented by  $f$ , while the convolution kernels of the layer are represented by  $W_i$ . The convolution operation is represented by the  $*$  symbol.

**Activation Function:** ReLU and softmax activation functions are commonly utilized in neural networks for a variety of purposes. ReLU is a non-linear function used in hidden layers that allows the network to learn complicated correlations between inputs and outputs. It is computationally efficient and aids in the mitigation of the vanishing gradient problem during backpropagation. Softmax, on the other hand, is an activation function used in the output layer for multi-class classification issues, calculating probabilities for each class in a way that adds up to 1. It magnifies the relative differences between input values, allowing the network to make confident predictions by selecting the class with the highest probability.

**BatchNormalization:** Batch normalization is a deep learning technique for improving training performance and speeding up convergence. It is used as a layer in neural network architecture to normalize inputs in order to achieve a zero mean and unit variance. The layer transforms the network by performing batch statistics, normalizing, scaling and shifting, and activation function, minimizing internal covariate shift and increasing non-linearity. This procedure aids in the stabilization of the learning process and lowers internal covariate shift.

**MaxPooling Layer:** The MaxPooling layer is a convolutional neural network (CNN) that shrinks the spatial dimensions of input feature maps while keeping critical information. It divides the input into non-overlapping pooling windows and returns the maximum value from each. Dimension reduction, translation invariance, and resistance to tiny translations are all advantages. In the model, the MaxPooling2D layer divides the input feature maps into non-overlapping rectangular regions. With a stride of 1 move into the feature maps and maximize the regions value as shown in Fig-3.18. The result is a downsampled feature map with reduced spatial dimensions and the same depth (number of channels) as the input.

**Dropout:** In deep learning models, the dropout layer is a regularization technique used to prevent overfitting. At each training step, it randomly sets a fraction of the input units to 0, causing some neurons to be dropped out while the remaining neurons contribute

to computation. Co-adaptation is reduced, and the network is forced to learn more robust characteristics. It does ensemble learning by training diverse subnetworks at each phase and thereby boosting generalization performance. The dropout layer is turned off during testing or inference, and all neurons are utilised, but outputs are scaled by the dropout rate to maintain the predicted sum of activations.

**Fully Connected Layer:** The output is passed into one or more fully connected layers after many convolutional and pooling layers. These layers, like a regular neural network, have connections between all neurons in the current layer and the previous layer. Fully connected layers are in charge of learning high-level representations and forecasting based on extracted information.

**Adam Optimizer:** The Adam optimizer is a deep learning optimization technique that is used to train neural networks. It is a stochastic gradient descent (SGD) extension noted for its effectiveness in optimizing complex models. The algorithm has a distinct learning rate for each parameter in the network and adjusts it based on gradients and prior history. It combines the advantages of AdaGrad and RMSprop, which modify the learning rate for each parameter based on the magnitude of the gradient. Adam improves on RMSprop by introducing momentum terms, which accelerates convergence. Adam's key hyper-parameters include the learning rate (alpha), decay rates (beta1 and beta2), and epsilon, which can be tweaked to improve performance on specific tasks and datasets

**Table-3.3: CNN Model Summary**

Layer	Output Shape	Parameter
Conv2D_1	(32,148,148,32)	896
Activation_1	(32,148,148,32)	
BatchNormalization_1	(32, 148, 148, 32)	128
MaxPooling2D _1	(32, 74, 74, 32)	0
Dropout_1	(32, 74, 74, 32)	0
Conv2D_2	(32,72,72,64)	18496
Activation_2	(32,72,72,64)	
BatchNormalization_2	(32,72,72,64)	256
MaxPooling2D _2	(32,36,36,64)	0
Conv2D_3	(32,34,34,64)	36928
Activation_3	(32,34,34,64)	
MaxPooling2D _3	(32,17,17,64)	0
Conv2D_4	(32,15,15,64)	73856
Activation_4	(32,15,15,128)	
MaxPooling2D _4	(32,7,7,128)	0
Flatten_1	(32, 6272)	0
Dense_1	(32,128)	802944
Dense_2	(32,4)	516

### 3.4.8 CNN Model Compilation

After the model is built now it's time for compile the model. As shown in the Fig-3.14 model is compiled with adam optimizer.

```
In [19]: model.compile(  
        optimizer='adam',  
        loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=False),  
        metrics=['accuracy']  
    )
```

Fig-3.15: Model Compilation

The code provided in Fig-3.14 is written in Python and uses the TensorFlow framework to compile a neural network model. Here's a breakdown of what each line does:

- **optimizer='adam':** This parameter specifies the optimizer to be utilized during training. Adam is a famous optimization technique that modifies the learning rate for each parameter in this instance.
- **loss=tf. keras. losses.SparseCategoricalCrossentropy(from\_logits=False):** Sets the loss function that will be used to train the model. Sparse Categorical Crossentropy was chosen as the loss function in this case. When the targets (labels) are integers and not one-hot encoded, this method is employed. The from\_logits option is set to False, indicating that before calculating the loss, the model's output probabilities are passed via a softmax activation function.
- **metrics=['accuracy']:** The metrics that will be computed and shown during training and evaluation are specified. It's accuracy in this example, which is a popular metric for classification assignments. It calculates the percentage of accurately predicted labels among all samples.

Once the model is compiled with these settings, it is ready for training and evaluation.

### 3.4.9 CNN Model Training

After compiling the model, it is ready for training. In this case used fit() function with some parameters. All are described in the following sections.

```
In [20]: history = model.fit(
        train_ds,
        batch_size=BATCH_SIZE,
        validation_data=val_ds,
        verbose=1,
        epochs=EPOCHS,
    )
```

```
Epoch 1/40
104/104 [=====] - 142s 1s/step - loss: 0.7309 - accuracy: 0.7563 - val_loss: 1.2750 - val_accuracy: 0.4062
Epoch 2/40
104/104 [=====] - 100s 959ms/step - loss: 0.3363 - accuracy: 0.8532 - val_loss: 1.8314 - val_accuracy: 0.3854
Epoch 3/40
104/104 [=====] - 101s 967ms/step - loss: 0.2662 - accuracy: 0.8956 - val_loss: 2.2440 - val_accuracy: 0.3073
Epoch 4/40
104/104 [=====] - 101s 974ms/step - loss: 0.2261 - accuracy: 0.9076 - val_loss: 0.6975 - val_accuracy: 0.7240
Epoch 5/40
104/104 [=====] - 103s 989ms/step - loss: 0.1982 - accuracy: 0.9224 - val_loss: 0.4738 - val_accuracy: 0.8125
Epoch 6/40
104/104 [=====] - 101s 971ms/step - loss: 0.1471 - accuracy: 0.9461 - val_loss: 0.2321 - val_accuracy: 0.9271
Epoch 7/40
104/104 [=====] - 100s 965ms/step - loss: 0.1060 - accuracy: 0.9688 - val_loss: 0.1668 - val_accuracy: 0.9844
Epoch 8/40
104/104 [=====] - 100s 965ms/step - loss: 0.0860 - accuracy: 0.9800 - val_loss: 0.1242 - val_accuracy: 0.9896
Epoch 9/40
104/104 [=====] - 100s 965ms/step - loss: 0.0700 - accuracy: 0.9888 - val_loss: 0.1226 - val_accuracy: 0.9896
Epoch 10/40
104/104 [=====] - 100s 965ms/step - loss: 0.0580 - accuracy: 0.9920 - val_loss: 0.1230 - val_accuracy: 0.9896
Epoch 11/40
104/104 [=====] - 100s 965ms/step - loss: 0.0480 - accuracy: 0.9940 - val_loss: 0.1187 - val_accuracy: 0.9896
Epoch 12/40
104/104 [=====] - 100s 965ms/step - loss: 0.0400 - accuracy: 0.9960 - val_loss: 0.1193 - val_accuracy: 0.9844
Epoch 13/40
104/104 [=====] - 100s 965ms/step - loss: 0.0340 - accuracy: 0.9970 - val_loss: 0.1210 - val_accuracy: 0.9948
```

Fig-3.16: Model Training.

The code provided in Fig-3.15 is using the fit() method of a model to train it for 40 epochs.

**train\_ds:** This is the training dataset that will be used to train the model. It's typically a tf.data.Dataset object containing the training samples and labels.

**batch\_size:** It represents the number of samples per gradient update. The dataset will be divided into batches of size 32 during training.

**validation\_data:** This parameter specifies the validation dataset used to evaluate the model's performance after each epoch. It's also typically a tf.data.Dataset object containing validation samples and labels.

**verbose:** This parameter controls the verbosity during training. Setting it to 1 will show a progress bar and training information for each epoch.

**epochs:** It determines the number of times the model will iterate over the entire training dataset. In this case, the model will be trained for 40 epochs.

After executing this code, the model is trained on the training dataset for the specified number of epochs. The training progress will be displayed, and after each epoch, the model's performance on the validation dataset will be evaluated. The training history, including metrics such as loss and accuracy, will be stored in the history object.

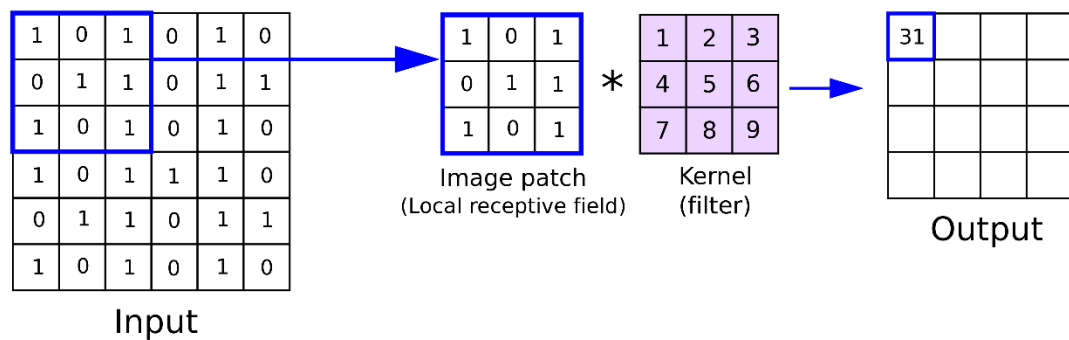


Fig-3.17: Convolutional Layer.

2 x 2 filter size

Default –stride 1

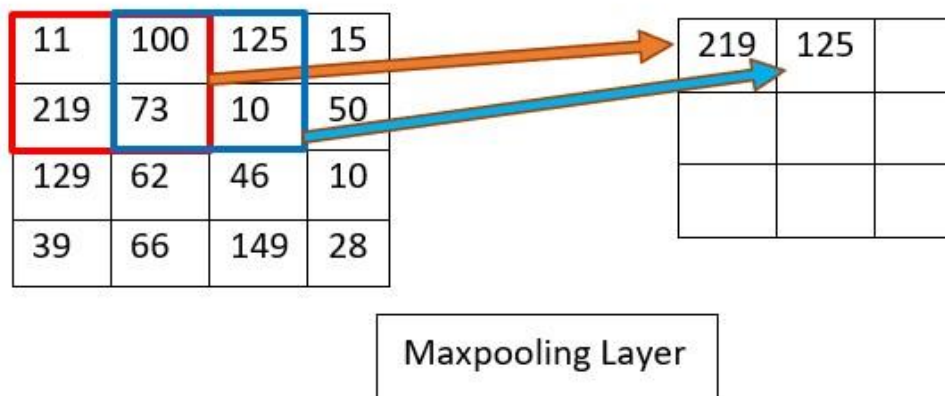


Fig-3.18: Maxpooling Layer.

## CHAPTER 4

### EXPERIMENTAL RESULTS AND DISCUSSION

#### 4.1 Experimental Setup

There was no setup created for the "Mazie Leaf Disease Detection" research. A PC & internet was used. As a platform Jupyter Notebook was used and Python language is the key language to do the research.

#### 4.2 Experimental Result & Analysis

In this chapter, the model we trained is evaluated using different Measurement function on test dataset.

##### 4.2.1 Model Evaluation

```
In [21]: scores = model.evaluate(test_ds)
          scores

21/21 [=====] - 27s 251ms/step - loss: 0.1261 - accuracy: 0.9836

Out[21]: [0.12609583139419556, 0.9836309552192688]
```

Fig-4.1: Model evaluate

The output provided in Fig:4.1, indicates the results of evaluating a model on a test dataset.

- The evaluation was carried out on 21 batches of data with a batch size of 32.
- Each batch took about 27 seconds to process, with an average step time of 251 milliseconds.
- On the test dataset, the reported loss is 0.1261.
- The model's accuracy on the test dataset is 0.9836, which means it achieved 98.36% accuracy.

These metrics show how well the model performed on the test dataset. The smaller the loss number, the more closely the model's predictions match the true data. A higher accuracy number, on the other hand, implies that the model's predictions were more accurate when compared to the ground truth labels.



## 4.2.2 Training & Validation Accuracy

```
In [25]: acc = history.history['accuracy']
val_acc = history.history['val_accuracy']

loss = history.history['loss']
val_loss = history.history['val_loss']
```

```
In [26]: plt.figure(figsize=(8, 8))
plt.subplot(1, 2, 1)
plt.plot(range(EPOCHS), acc, label='Training Accuracy')
plt.plot(range(EPOCHS), val_acc, label='Validation Accuracy')
plt.legend(loc='lower right')
plt.title('Training and Validation Accuracy')

plt.subplot(1, 2, 2)
plt.plot(range(EPOCHS), loss, label='Training Loss')
plt.plot(range(EPOCHS), val_loss, label='Validation Loss')
plt.legend(loc='upper right')
plt.title('Training and Validation Loss')
plt.show()
```

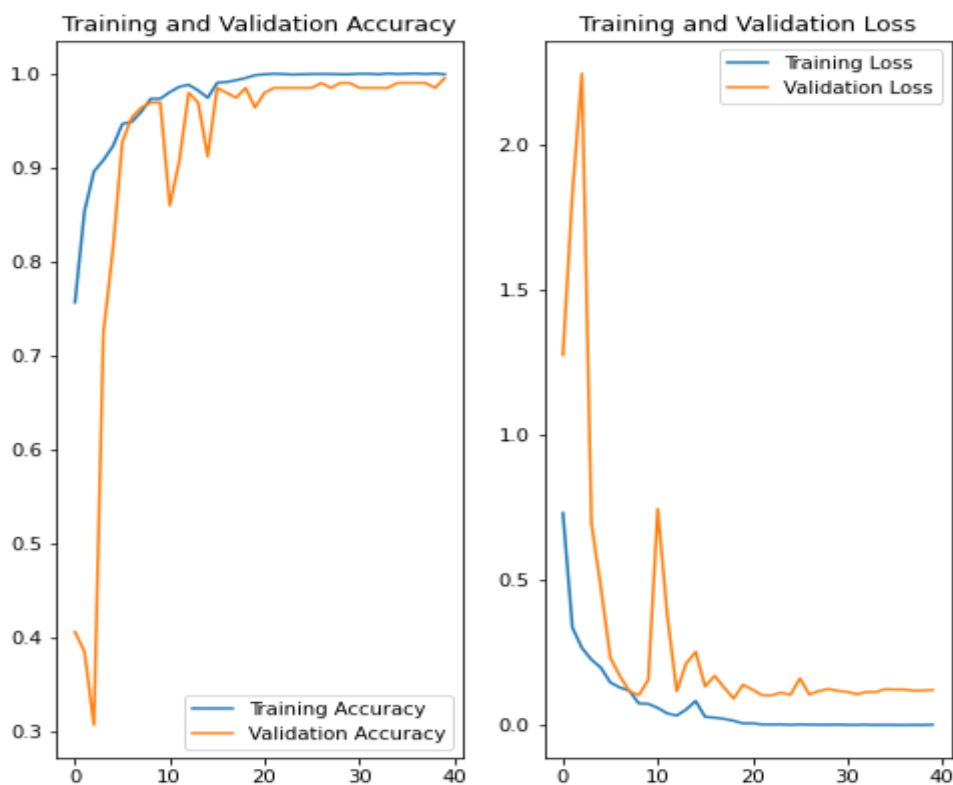


Fig-4.2: Training & Validation Accuracy & Loss

From the Fig-4.2 we can see that the accuracy, loss, and validation accuracy, validation loss we got from the forty epochs are stored and displayed through the graph.

From the graph we see that at first the accuracy was low & loss were very high. But rapidly this accuracy increased as the number of data for training was increased. After 20 epochs the accuracy was 99% and at 40<sup>th</sup> epochs the loss rate was 0.0020.

### 4.2.3 Maize Leaf disease Prediction

As the model is trained well & we got 98.36% accuracy on our test dataset, now it's time to predict the test dataset how accurate it predicts. In this case we will see the actual label and the predicted label their confidence to the prediction.

```
In [28]: def predict(model, img):
img_array = tf.keras.preprocessing.image.img_to_array(images[i].numpy())
img_array = tf.expand_dims(img_array, 0)

predictions = model.predict(img_array)

predicted_class = class_names[np.argmax(predictions[0])]
confidence = round(100 * (np.max(predictions[0])), 2)
return predicted_class, confidence

In [44]: plt.figure(figsize=(15, 15))
for images, labels in test_ds.take(1):
    for i in range(9):
        ax = plt.subplot(3, 3, i + 1)
        plt.imshow(images[i].numpy().astype("uint8"))

        predicted_class, confidence = predict(model, images[i].numpy())
        actual_class = class_names[labels[i]]

        plt.title(f"Actual: {actual_class},\n Predicted: {predicted_class}.\n Confidence: {confidence}%")

        plt.savefig('Test Dataset Prediction Using The Trained Model.png',format='png',facecolor='white')
        plt.axis("off")
```

Fig-4.3: Maize leaves prediction

From the code provided in Fig-4.3, `predict()` function is used to predict the class and map the class to its original name. Also some images from test dataset has been taken to predict and displayed in the Figure.

The provided code defines a `predict()` function that takes a trained model and an image as input and returns the predicted class and confidence level for that image.

- **`img_array=tf.keras.preprocessing.image.img_to_array(images[i].numpy())`**: This line converts the input image `img` to a NumPy array using `img_to_array()` function from the `tf.keras.preprocessing.image` module.
- **`img_array = tf.expand_dims(img_array, 0)`**: This line adds an extra dimension to the image array using `tf.expand_dims()`. This is necessary because the model expects input data in the shape of (batch\_size, image\_height, image\_width, num\_channels), and by adding an extra dimension, we effectively create a batch of size 1.

- **predictions = model.predict(img\_array):** This line feeds the image array to the model using the predict() method and obtains the predictions.
- **predicted\_class = class\_names[np.argmax(predictions[0]):** This line determines the predicted class by finding the index of the maximum value in the predictions array (np.argmax(predictions[0])) and using that index to access the corresponding class name from the class\_names array.
- **confidence = round(100 \* (np.max(predictions[0])), 2):** This line calculates the confidence level of the prediction by finding the maximum value in the predictions array (np.max(predictions[0])) and multiplying it by 100. The round() function is used to round the confidence value to two decimal places.

Finally, the function returns the predicted\_class and confidence as a tuple.

The next block of code provided is plotting a 3x3 grid of images from the test dataset and annotating each image with its actual class, predicted class, and confidence score. Additionally, it saves the plotted Figure as a PNG file named "Test Dataset Prediction Using The Trained Model.png" as Fig-4.4.

- Here's a breakdown of the code:
- The **plt.figure(figsize=(15, 15))** line sets the Figure size to 15x15 inches.
- The loop for images, labels in test\_ds.take(1): iterates over the first batch of images and labels from the test dataset.
- Within the loop, **plt.subplot(3, 3, i + 1)** creates a subplot in a 3x3 grid, with index i ranging from 0 to 8.
- **plt.imshow(images[i].numpy().astype("uint8"))** displays the image in the subplot. The numpy() method converts the image tensor to a NumPy array, and astype("uint8") ensures that the pixel values are treated as unsigned 8-bit integers.
- The predict() function is used to obtain the predicted class and confidence score for the current image.
- **class\_names[labels[i]]** retrieves the actual class label for the current image from the class\_names list.
- **plt.title(...)** sets the title of the subplot, displaying the actual class, predicted class, and confidence score.
- **plt.savefig(...)** saves the plotted Figure as a PNG file with the specified filename, format, and background color.
- **plt.axis("off")** turns off the axis ticks and labels for each subplot.

Overall, this code generates a visualization of the test dataset with predicted and actual labels, providing insights into how the trained model performs on the test images.

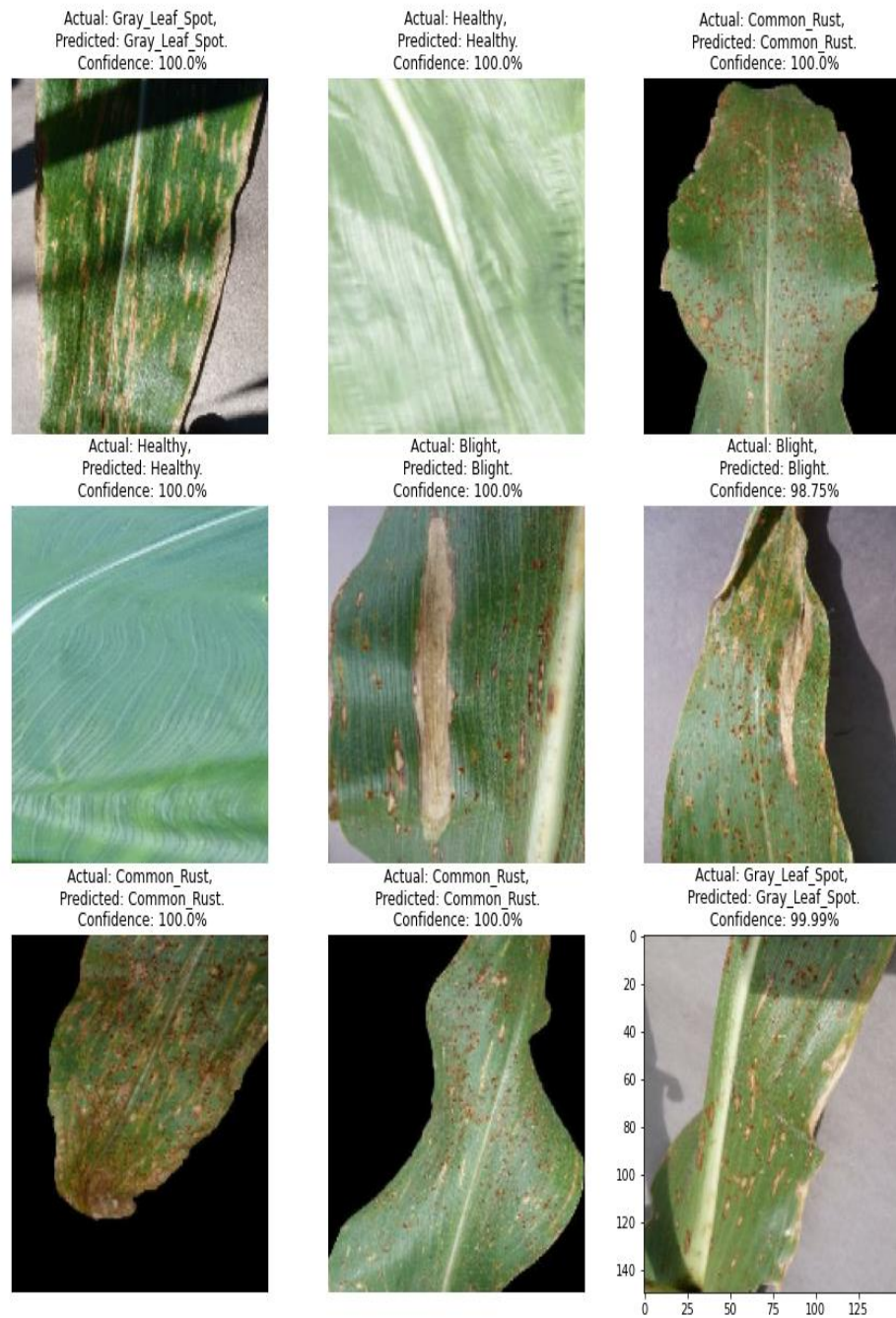


Fig-4.4: Maize leave disease detection using deep learning model.

#### **4.2.4 Precision, Recall, F1 score & Accuracy**

Precision, recall, F1 score, and accuracy score are commonly used metrics to evaluate the performance of classification models. Here's a brief explanation of each metric:

**1. Precision:** Precision is a measure of how many expected positive cases really occur. The model's ability to avoid false positives is highlighted. Precision is calculated as

$$\text{Precision} = \text{True Positives} / (\text{True Positives} + \text{False Positives}).$$

Precision gives information about the model's ability to make accurate positive predictions.

**2. Recall:** The model's ability to properly detect positive instances is measured by recall, also known as sensitivity or true positive rate. Its primary goal is to reduce false negatives. Recall is calculated as follows:

$$\text{Recall} = \text{True Positives} / (\text{True Positives} + \text{False Negatives})$$

Recall aids in determining the model's completeness in catching positive cases.

**3. F1 Score:** The F1 score is the harmonic mean of precision and recall, balancing the two measurements. It is a single value that combines precision and recall. The F1 score is calculated as follows:

$$\text{F1 Score} = 2 * (\text{Precision} * \text{Recall}) / (\text{Precision} + \text{Recall})$$

When you wish to examine both precision and recall at the same time, the F1 score comes in handy.

**4. Accuracy Score:** Accuracy is a widely used indicator that assesses the overall accuracy of the model's predictions. It computes the ratio of successfully predicted to total instances.

$$\text{Accuracy} = (\text{True Positives} + \text{True Negatives}) / (\text{True Positives} + \text{False Positives} + \text{False Negatives})$$

Accuracy indicates how well the model performs across all classes.

These metrics help assess different aspects of a classification model's performance. Depending on the specific problem and priorities, you may focus on optimizing one or more of these metrics.

```

In [33]: precision = precision_score(true_labels, predicted_labels, average='weighted')
         recall = recall_score(true_labels, predicted_labels, average='weighted')
         f1 = f1_score(true_labels, predicted_labels, average='weighted')
         accuracy = accuracy_score(true_labels, predicted_labels)

In [34]: precision, recall, f1, accuracy

Out[34]: (0.9837471911664192,
          0.9836309523809523,
          0.9836374497892231,
          0.9836309523809523)

```

Fig-4.5: Evaluation Metrics

The code provided in Fig-4.5 calculates several evaluation metrics for a classification task using the predicted labels and true labels. Here's an explanation of each metric:

- The *precision\_score()* function is used to calculate precision. The *average='weighted'* argument indicates that weighted averaging is used, which considers the support (the number of true instances for each label) when calculating the average precision. For this model precision is 98.37%
- The *recall\_score()* function is used to calculate recall. Similar to precision, *average='weighted'* is used for weighted averaging. For this model recall score is 98.36%
- The *f1\_score()* function is used to calculate the F1 score. As with precision and recall, *average='weighted'* is used for weighted averaging. For this model f1 score is 98.37%.
- The *accuracy\_score()* function is used to calculate accuracy. Unlike precision, recall, and F1 score, the *accuracy\_score* function does not require the average argument since it returns a single accuracy value. For this model accuracy score is 98.37%.

**Table-4.1: Precision, Recall & F1 Score**

Precision	Recall	F1 Score
98.37%	98.36%	98.36%

Table-4.1 shows the precision, recall & f1 score for the model we trained against our test dataset.

We can see these performance metrics in better way by visualizing in graph in Fig-4.6.

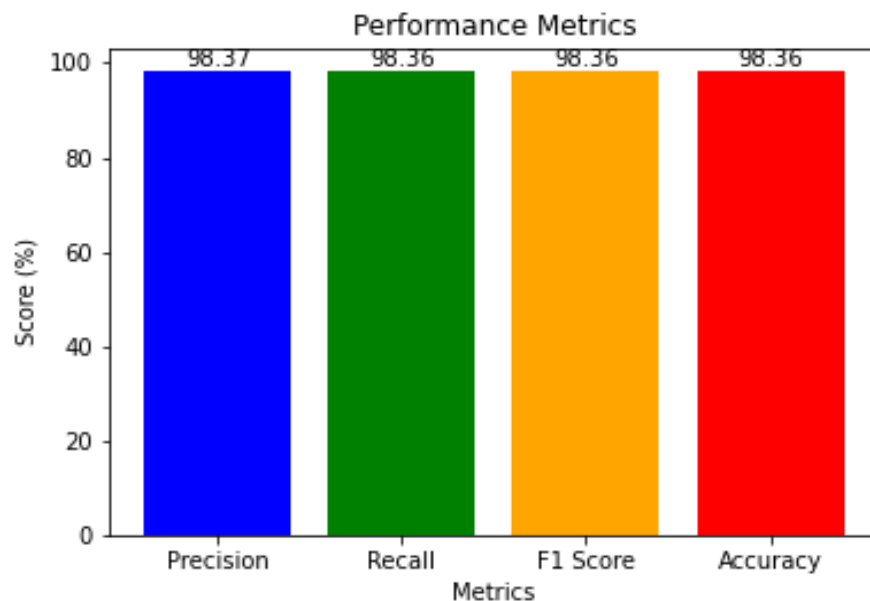


Fig-4.6: Performance Metrics

**Confusion Matrix:** A confusion matrix is a Table that is often used to evaluate the performance of a classification model. It provides a more detailed breakdown of the model's predictions and the actual class labels.

The confusion matrix consists of four components:

*True Positive (TP):* The number of cases that are accurately anticipated as positive (classified as positive).

*True Negative (TN):* The number of cases accurately predicted as negative (identified as being in the negative category).

*False Positive (FP):* The number of events that are anticipated as positive but are wrongly labeled as positive.

*False Negative (FN):* The number of cases that are anticipated as negative but are wrongly classified as negative.

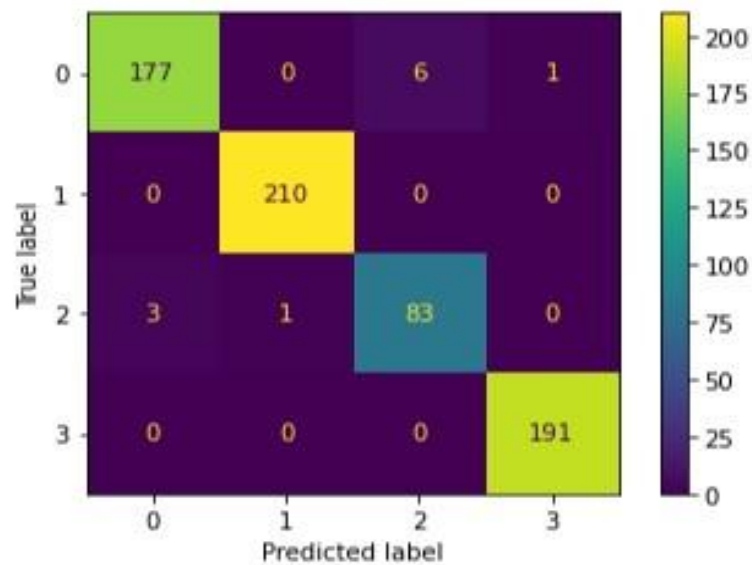


Fig-4.7: Confusion Matrix

Figure-4.7 counts true positive (TP), true negative (TN), false positive (FP), and false negative (FN) predictions. It provides a comprehensive view of how well the model's predictions align with the actual labels across different classes.

---

```
In [41]: import pickle
         pickle.dump(model, open("maize_leave_detection_model_98.36.pkl", "wb"))
```

---

Fig-4.8: Save Model.

Finally the model is saved using pickle as in Fig-4.8. Pickle is a Python package that allows you to serialize and deserialize Python objects. It enables you to serialize items and then reconstruct them from the serialized form.



### 4.3 Discussion

From the above result analysis we can say this model is better than many other model for better disease detection with higher accuracy result of 98.36% according to Table-4.3 By doing predictions over the test dataset we saw that the confidence rate for all image is almost 100%. So the model is predicting accurately in this case. The prediction might be wrong if the leaf is infected with more than one disease. However, with such a higher accuracy and almost 100% confidence we can say this model is better and efficient

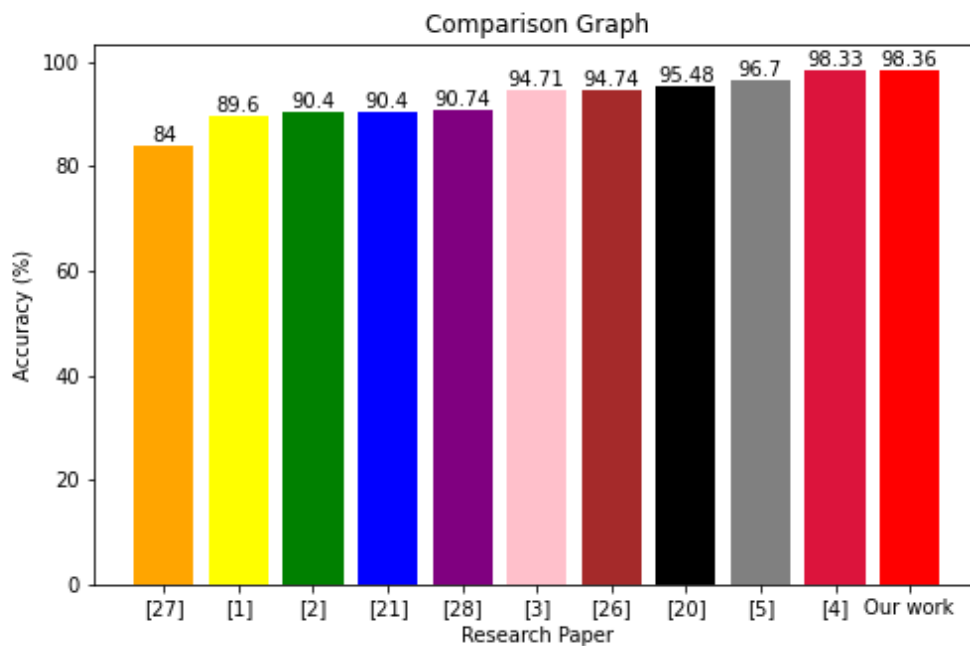


Fig-4.9: Accuracy comparison graph among research papers and our proposed work.

Comparing with others in the Fig-4.9, our model achieved 98.36% accuracy which is better than any other algorithms or model.

## CHAPTER 5

### IMPACT ON SOCIETY AND ENVIRONMENT

#### 5.1 Impact on Society

The research on maize leaf disease detection using deep learning can have several impacts on society. Here are some potential effects:

- i. **Improved crop yield:** Maize is a staple crop in many countries, and diseases can drastically impair productivity. Farmers can take prompt action to prevent disease transmission and limit its impact by employing deep learning algorithms to detect and diagnose diseases. This can lead to better crop yields, improved food security, and lower economic losses.
- ii. **Sustainable agriculture:** Early disease detection in maize plants allows farmers to deploy focused and accurate treatments, reducing the need for pesticide or fungicide use. Farmers may maximize their treatments while reducing the environmental impact of chemical applications by using deep learning models to identify specific diseases.
- iii. **Cost savings:** Crop diseases frequently cause significant losses for maize farmers. Deep learning for disease diagnosis can help farmers identify sick plants or places in their farms more effectively. This focused method allows them to focus their resources, such as treatments and interventions, on the regions that are affected, reducing unnecessary costs and optimizing their efforts.
- iv. **Knowledge transfer:** Deep learning-based disease detection in maize research can contribute to the development of open-source tools, datasets, and models. These tools can be given to farmers, extension workers, and academics all across the world, allowing knowledge transfer and capacity building. Such collaboration can give farmers with access to innovative technology and procedures regardless of their geographical location, helping the global farming community in the long term.
- v. **Data-driven decision-making:** Deep learning algorithms use large amounts of data to learn and predict. As research progresses, large datasets linked to maize illnesses can be created, contributing to a greater understanding of disease patterns, environmental conditions, and other elements that affect crop health.

This data-driven strategy can assist farmers, politicians, and academics in making informed decisions on sustainable farming methods.

- vi. **Research and innovation:** Agriculture's deep learning is continually evolving, and advancements in maize disease identification can stimulate new study and innovation. New chances to address additional agricultural challenges may emerge as scientists refine deep learning models and algorithms. The maize disease detection study could pave the way for future research in plant pathology, precision farming, and automated monitoring systems.

Overall, the impact of maize leaf disease detection using deep learning research has the potential to be far-reaching, improving farmers, agricultural practices and food security while supporting sustainable and data-driven agricultural techniques.

## 5.2 Impact on Environment

The research on "Maize leaf disease detection using Deep learning" can have both positive and negative impacts on the environment, depending on how it is implemented and utilized. Here are some potential impacts:

### Positive impacts:

- i. **Early detection and intervention:** Deep learning systems can detect diseases in maize plants early on. Farmers can then take immediate action, such as targeted pesticide application or disease management methods, minimizing disease spread. Early intervention can help to reduce crop losses and boost overall production.
- ii. **Reduced chemical usage:** Accurate disease detection can help to reduce the need for pesticides to be used indiscriminately. Rather than spraying pesticides on the entire crop, farmers can target specific parts or plants that are infected. This targeted method has the ability to reduce the amount of pesticides required, hence lowering their environmental impact and promoting sustainable farming practices.
- iii. **Efficient resource allocation:** Farmers may make better use of resources like water, fertilizer, and other inputs by precisely detecting sick plants. This concentrated technique guarantees that resources are correctly provided to healthy plants, reducing waste and increasing resource efficiency.

### Negative impacts:

- **Energy consumption:** Deep learning algorithms require a significant amount of computer power, which consumes energy. If the energy used to train and run these algorithms on massive datasets comes from nonrenewable sources, it can have an impact on the environment. It is vital to consider energy-efficient technologies or to employ renewable energy sources to mitigate this impact.
- **E-waste:** Implementing a deep learning system typically necessitates the use of computer hardware, sensors, and other electronic components. The disposal of these components, if not properly managed, can contribute to the accumulation of electronic rubbish (e-waste). To limit the environmental impact of e-waste, efficient recycling and disposal methods should be used.

Though these research has positive & negative impacts, in this era of technology, all sectors almost use AI, Machine Learning to have better efficiency & reliability. So, it is high time to use these advanced techniques in our agricultural sector, specially in disease detection of various kind of leaves. Though these research is for maize leaf disease detection, we can use the same process for different types of plants.

### 5.3 Ethical Aspects

Several significant factors rise to mind when evaluating the ethical aspects of a project like "maize leaf disease detection using CNN". Here are some ethical considerations:

- **Data privacy and security:** All the image data has been taken from kaggle dataset which has been made from "PlantVillage" and "PlantDoc" dataset.
- **Bias and fairness:** The model is shuffled well so there is no biasness of being dominant by one or two classes. Identification is also well.
- **Social and economic impact:** To detect maize leaf diseases our techniques have achieved a great accuracy. So this technique will be very useful for farmers to protect their crops and also the production will be better as the demand is rising. Instead of hiring an expert, these technique will be less expensive that will reduce the price also.

## **CHAPTER 6**

### **SUMMARY, CONCLUSION AND IMPLICATION FOR FUTURE RESEARCH**

#### **6.1 Summary of The Study**

The proposed model uses deep learning techniques to detect maize leaf diseases using a dataset of 4188 images. The architecture includes four convolutional layers with kernel sizes of 3x3, 32, 64, 64, and 128 filters, and the Rectified Linear Unit (ReLU) activation function. The input data has a shape of (32, 150, 150, 3). To enhance performance, four MaxPooling layers, two Batch-Normalization layers, a Dropout layer, and flatten layers are employed. The model achieves an impressive accuracy of 98.36% on the test dataset, demonstrating its robustness and consistency. The model's performance is also analyzed using a confusion matrix. The model's impact on society is significant, as it improves crop yield by identifying and addressing disease outbreaks, promotes sustainable agriculture by minimizing chemical treatments, and reduces crop loss due to diseases. It also facilitates knowledge transfer among farmers, researchers, and agricultural experts, enables data-driven decision-making, and encourages research and innovation in plant pathology. Overall, the model has the potential to significantly improve the detection of maize leaf diseases and contribute to sustainable agriculture.

## **6.2 Conclusion**

An effective CNN-based model has been successfully built to identify the three most frequent diseases affecting maize plants: gray leaf spot, common rust, and northern leaf blight, as well as healthy leaves. This model achieved an astounding 98.36% accuracy while requiring less training time to converge. It has outperformed prior research efforts at detecting illnesses in corn leaves. Several strategies, including modifying various model parameters, including the ReLU function, leveraging the Adam optimizer, and implementing pooling procedures, have all contributed to this exceptional performance. These modifications have proven to be an extremely successful method of increasing maize productivity. Farmers may easily implement this technology, which offers a cost-effective and time-saving alternative for detecting maize illnesses. As a result, the development of digital agricultural systems is aided. In the future, there is the potential to improve the existing model by including more deep learning approaches and algorithms. This increase would allow for the detection of a broader range of maize leaf diseases, boosting the model's capabilities for training and testing.

## **6.3 Implication For Future Research**

This study is free for students & teachers and those who are interested to learn, research and for further upgradation. This study will help to learn about CNN algorithm, why it's better & how efficient is. Researchers can use this paper to find the limitations and get solutions to build a better model to get almost 100% accuracy for disease detection. This study involve only 3 common types of disease. Researchers can use this as a tool to develop a better model for more types of diseases of different types of plant as well as crops for better productions, cost-savings to help our digital agriculture system.

## REFERENCES

- [1] K. Song, X. Y. Sun, and J. W. Ji, “Corn leaf disease recognition based on support vector machine method,” *Trans. Chin. Soc. Agricult. Eng.*, vol. 23, no. 1, pp. 155–157, Jan. 2007.
- [2] L. Chen and L. Y. Wang, “Research on application of probability neural network in maize leaf disease identification,” *J. Agricult. Mech. Res.*, vol. 33, no. 6, pp. 145–148, Jun. 2011.
- [3] L. F. Xu, X. B. Xu, and H. Min, “Corn leaf disease identification based on multiple classifiers fusion,” *Trans. Chin. Soc. Agricult. Eng.*, vol. 31, no. 14, pp. 194–201, 2015.
- [4] Al-Amin, M., Bushra, T. A., & Hoq, M. N. (2019, May). Prediction of Potato Disease from Leaves using Deep Convolution Neural Network towards a Digital Agricultural System. In 2019 1st International Conference on Advances in Science, Engineering and Robotics Technology (ICASERT) (pp. 1-5). IEEE.
- [5] C. Dechant et al., “Automated identification of northern leaf blight infected maize plants from field imagery using deep learning,” *Phytopathology*, vol. 107, no. 11, pp. 1426–1432, 2017.
- [6] N. Wang, K. Wang, R. Xie, J. Lai, B. Ming, and S. Li, “Maize leaf disease identification based on fisher discrimination analysis,” *Scientia Agricultura Sinica*, vol. 42, no. 11, pp. 3836–3842, 2009.
- [7] Z. Qi et al., “Identification of maize leaf diseases based on image technology,” *J. Anhui Agricult. Univ.*, vol. 43, no. 2, pp. 325–330, Feb. 2016.
- [8] F. Zhang, “Recognition of corn leaf disease based on quantum neural network and combination characteristic parameter,” *J. Southern Agriculture*, vol. 44, no. 8, pp. 1286–1290, 2013.
- [09] Y. LeCun, Y. Bengio, and G. Hinton, “Deep learning,” *Nature*, vol. 521, pp. 436–444, May 2015.
- [10] Y. Guo, Y. Liu, A. Oerlemans, S. Lao, S. Wu, and M. S. Lew, “Deep learning for visual understanding: A review,” *Neurocomputing*, vol. 187, pp. 27–48, Apr. 2016.
- [11] I. Lenz, H. Lee, and A. Saxena, “Deep learning for detecting robotic grasps,” *Int. J. Robot. Res.*, vol. 34, nos. 4–5, pp. 705–724, 2013.
- [12] B. Alipanahi, A. DeLong, M. T. Weirauch, and B. J. Frey, “Predicting the sequence specificities of DNA- and RNA-binding proteins by deep learning,” *Nature Biotechnol.*, vol. 33, pp. 831–838, Jul. 2015.
- [13] L. P. Zhang, G. S. Xia, T. Wu, L. Lin, and X. C. Tai, “Deep learning for remote sensing image understanding,” *J. Sensors*, vol. 2016, Jun. 2015, Art. no. 7954154.
- [14] Y. Bengio, “Learning deep architectures for AI,” *Found. Trends Mach. Learn.*, vol. 2, no. 1, pp. 1–127, 2009.
- [15] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman, “The PASCAL visual object classes (VOC) challenge,” *Int. J. Comput. Vis.*, vol. 88, no. 2, pp. 303–338, Jun. 2010.
- [16] O. Russakovsky et al., “ImageNet large scale visual recognition challenge,” *Int. J. Comput. Vis.*, vol. 115, no. 3, pp. 211–252, Dec. 2015.
- [17] A. Krizhevsky, I. Sutskever, and G. Hinton, “ImageNet classification with deep convolutional neural networks,” in *Proc. Adv. Neural Inf. Process. Syst.*, Lake Tahoe, NV, USA, 2012, pp. 1097–1105.

- [18] K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” in *Proc. Int. Conf. Learn. Represent.*, 2015, pp. 1–14.
- [19] M. D. Zeiler and R. Fergus, “Visualizing and understanding convolutional networks,” in *Proc. Eur. Conf. Comput. Vis.*, 2014, pp. 818–833.
- [20] Y. Lu, S. Yi, N. Zeng, Y. Liu, and Y. Zhang, “Identification of rice diseases using deep convolutional neural networks,” *Neuro-computing*, vol. 267, pp. 378–384, Dec. 2017.
- [21] G. Wang, Y. Sun, and J. X. Wang, “Automatic image-based plant disease severity estimation using deep learning,” in *Computational Intelligence and Neuroscience*, 2017, pp. 1–8.
- [22] F. C. Dai et al., “Curvularia leaf spot of maize: Pathogens and varietal resistance,” *Acta Phytopathologica Sinica*, vol. 28, no. 2, pp. 123–129, Feb. 1998.
- [23] X. X. Li et al., “The corn disease remote diagnostic system in China,” *J. Food Agricult. Environ.*, vol. 10, no. 1, pp. 617–620, Jan. 2012.
- [24] J. M. J. Ward, E. L. Stromberg, D. C. Nowell, and F. W. Nutter, Jr., “Gray leaf spot: A disease of global importance in maize production,” *Plant Disease*, vol. 83, no. 10, pp. 884–895, 1999.
- [25] S. A. Miller, F. D. Beed, and C. L. Harmon, “Plant disease diagnostic capabilities and networks,” *Annu. Rev. Phytopathol.*, vol. 47, no. 1, pp. 15–38, 2009.
- [26] F. Qin, D. Liu, B. Sun, L. Ruan, Z. Ma, and H. Wang, “Identification of alfalfa leaf diseases using image recognition technology,” *PLoS ONE*, vol. 11, no. 12, Article ID e0168274, 2016
- [27] Dionis A. Padilla; Ramon Alfredo I. Pajes; Jerome T. De Guzman, “ Detection of Corn Leaf Diseases Using Convolutional Neural Network With OpenMP Implementation ” , 2020 IEEE 12th International Conference on Humanoid, Nanotechnology, Information Technology, Communication and Control, Environment, and Management (HNICEM), 16 April 2021.
- [28] Qi Zhao , Jiang ZhaoHui , Yang ChunHe , Liu LianZhong , Rao Yuan, “Identification of maize leaf diseases based on image technology”, *Journal of Anhui Agricultural University* 2016 Vol.43 No.2 pp.325-330 ref.18



## **APPENDICES**

### **RESEARCH REFLECTION**

During this research I have faced some problems. The major two problems I have faced are “Algorithm choosing” and “CNN Model building”. At first I have tried some other algorithm like SVM, Random Forest Classification, Ensemble Learning etc, but I got very poor accuracy in evaluation. Then I have tried CNN algorithm and saw hope to get better accuracy. The next challenge was to build a proper model with different layers like Conv2D, Maxpooling2D, Dense layer, Dropout layer etc. After trying many times with different combination of layers finally I got the correct one to get a better model and resulting higher accuracy.

## PLAGARISM REPORT

### AN EFFICIENT MAIZE LEAF DISEASE DETECTION APPROACH USING CNN

#### ORIGINALITY REPORT

27%

SIMILARITY INDEX

22%

INTERNET SOURCES

16%

PUBLICATIONS

17%

STUDENT PAPERS

#### PRIMARY SOURCES

1

[dspace.daffodilvarsity.edu.bd:8080](https://dspace.daffodilvarsity.edu.bd/8080)

Internet Source

6%

2

Xihai Zhang, Yue Qiao, Fanfeng Meng, Chengguo Fan, Mingming Zhang.

"Identification of Maize Leaf Diseases Using Improved Deep Convolutional Neural Networks", IEEE Access, 2018

Publication

3%

3

Kshyanaprava Panda Panigrahi, Abhaya Kumar Sahoo, Himansu Das. "A CNN Approach for Corn Leaves Disease Detection to support Digital Agricultural System", 2020 4th International Conference on Trends in Electronics and Informatics (ICOEI)(48184), 2020

Publication

2%

4

[journals.dbuniversity.ac.in](https://journals.dbuniversity.ac.in)

Internet Source

1%

5

[www.researchgate.net](https://www.researchgate.net)

Internet Source

1%