POTATO LEAF DISEASE DETECTION AND CLASSIFICATION USING CONVOLUTIONAL NEURAL NETWORK

By

1. Md. Firozzaman 180201130

B. Sc. Engineering in Computer Science and Engineering

Department of Computer Science and Engineering

Faculty of Electrical and Computer Engineering

Bangladesh Army University of Science and Technology (BAUST)

JANUARY 2023

Potato Leaf Disease Detection and Classification Using Convolutional Neural Network

This thesis is submitted in partial fulfillment of the requirement for the degree of B.Sc. Engineering in Computer Science & Engineering.

By

1. Md. Firozzaman 180201130

Supervised by

Md. Mamun Hossain

Assistant Professor, Dept. of CSE

Department of Computer Science and Engineering

Bangladesh Army University of Science and Technology (BAUST)

Saidpur Cantonment, Nilphamari, Bangladesh

The thesis titled **"Potato Leaf Disease Detection and Classification Using Convolutional Neural Network"** submitted by ID. 180201130, Session 2018-2019 has been presented on "17/01/2023" and accepted as satisfactory in fulfillment of the requirement for the degree of Bachelor of Science in Computer Science and Engineering (CSE) as B.Sc. Engineering to be awarded by the Bangladesh Army University of Science and Technology (BAUST).

Board of Examiners

1. _____ Supervisor

Mr. Md. Mamun Hossain

Assistant Professor

Department of Computer Science and Engineering (CSE)

Bangladesh Army University of Science and Technology (BAUST)

2. _____ Member

Mr. Md. Al-Hasan

Assistant Professor

Department of Computer Science and Engineering (CSE)

Bangladesh Army University of Science and Technology (BAUST)

3. _____ Member

Mrs. Ashrafun Zannat

Lecturer

Department of Computer Science and Engineering (CSE)

Bangladesh Army University of Science and Technology (BAUST)

4. _____ Member

Mr. Md. Zahid Hassan

Lecturer

Department of Computer Science and Engineering (CSE)

Bangladesh Army University of Science and Technology (BAUST)

_____

Dr. S.M. Jahangir Alam

Professor and Head

Department of Computer Science and Engineering (CSE)

Bangladesh Army University of Science and Technology (BAUST)

CANDIDATE'S DECLARATION

It is hereby declared that the contents of this project are original and any part of it has not been submitted elsewhere for the award if any degree or diploma.

1.	Md. Firozzaman	180201130	

ACKNOWLEDGMENTS

First of all, I am grateful to almighty Allah whose infinite grace enabled me to complete this thesis successfully. Thereafter, my sincerest thanks and gratitude to my honorable thesis supervisor Md. Mamun Hossain, Assistant Professor in the Department of Computer Science and Engineering at Bangladesh Army University of Science and Technology, Saidpur. His valuable suggestions, positive advice, inspiration, patience, and enthusiasm has stimulated me to complete this thesis and his sincere guidance helped me in all the research and writing of the thesis.

I would like to express my special thanks to the Head of the Department of Computer Science and Engineering for his valuable suggestions, positive advice, encouragement, and sincere guidance throughout my thesis work. I also convey my special thanks and gratitude to all of the respected teachers of the department of CSE including other departments.

I sincerely thank to respected hon'ble Vice Chancellor sir and respective staffs, Chief of Army and staffs, board of trustees and all members including academic members, and so on, those who led this institution and made me a bright future.

I like to thank all of my friends and the staffs of the department for their valuable suggestions and assistance. Finally, I would like to thank my parents and my family for their endless love, prayer, care, sacrifices and great support for my education and life.

ABSTRACT

Potato is one of the remarkable crops cultivated in Bangladesh. It is a well-known vegetable to all of us. Every year 10-12 million metric tons of potatoes is produced in Bangladesh. But this production is disrupted due to some serious disease of potato leaf like early blight and late blight. These diseases spread out over the crops field within a few days. As a result, the crops field is fatally damaged and increases the cost of production because of late detection of the disease. The aim here is to detect the leaf disease as early as possible. The diagnosis of leaf disease will be performed using leaf pictures of potato that we are going to do through CNN algorithm. This paper offers a deep learning base system where image processing techniques are used to detect, analyze, and classify the disorder leaf from the healthy. In this analysis, more than 1300 pictures of healthy and unhealthy collected from the crops field of potato that are divided into three parts, among them 80% images are used to train, 10% for validation and rest of 10% for testing. The proposed model is utilized to recognition and characterize healthy and diseased leaves of potato. The model has gained a total of 94% accuracy. Our result shows that CNN algorithm is the best way to detect potato leaf disease in advance and remedy fatal crops damage.

Keywords: Potato Leaf, Disease, Detection, Classification, CNN, Machine Learning, Deep Learning, Image Processing.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

Introduction

1.1 Overview

Potato is one of the remarkable crops cultivated in Bangladesh. It is well known and an important basic food in many countries around the world. It is also called the root of all vegetables. Bangladesh is an agricultural economy and grows different kinds of potatoes that occupy an important part in our country. The demand for potatoes is growing across the world day by day, and it is also required to export as much as our region can, so the main aspect is increasingly producing potatoes. But the fact is that in the last few years the exports and produce have been decreasing because of some serious disease of potato leaf-like early blight, late blight etc. The farmers also have to suffer due to this reason. Deep learning algorithms, especially the advent of various flavors of deep NNs [1-6] play an important role in classification and detection of these types of diseases. It has been well-accepted that deep learning algorithms perform quite well in image classification and detection compared to traditional image-processing algorithms [7]. To distinguish this blight disease, we have used a deep Convolutional Neural Network Model. That will be helpful for farmers to get rid of these blights and enhance potato production increasingly.

1.2 Background and Present State

The detection of potato leaf disease accurately in advance is very difficult. In most developing countries, detection and identification process of blight diseases are performed manually by trained personnel inspecting the crops field. This process is tedious and, in some cases, impractical due to the unavailability of a disease expert in remote regions [8]. On the other hand, the recent advances in image processing for rapid and automated disease identification using images of plant leaves can make the process far more efficient and timelier [9-11]. In the recent past, many researchers worked on the classification and detection of the plant diseases using deep NNs. In 2018, A classification model using CNN for distinguishing 58 classes of healthy and diseased plant dataset was developed by Ferentinos and Konstantinos [12] that improved in 2019 by Arsenovic et al. [13] using by leaf

images in field conditions. Another work, using fuzzy C-means clustering [14] identifying the severity of potato late blight disease from field but with few images. Here used only 300 images for training set also have a disadvantage is that images captured by untrained farmers and contain cluster of leaves with background visible in several segments. Another research in 2018, [15] using CNN model methodology to implement their project. The advantage of this model is that only leaf area was calculated, and fungi caused diseases in sugarcane can be recognized. The disadvantage of is that high computational complexity was required to implement it. Most of the works in earlier on the detection and classification of leaf disease of potato were various drawbacks such as limitation of dataset, heigh computational complexity, more processing time etc. Therefore, we attempted to work on these issues here.

1.3 Problem Statement

Potato production is disrupted due to some serious disease of potato leaf - like Early blight and Late blight that are fungal disease and common occurrence where potato is cultivated. These are foliage diseases of potato that start as uneven light green lesions near the tip and the margins of the leaf and then spreads into large brown to purplish black necrotic patches [16]. Blight incites the tuber rot that causes premature defoliation of potato. Since most of the farmers of our country are not literate, they don't know about the disease, and they cannot monitor the crops area properly. The bight spread out increasingly around the crops field within a few weeks under a conducive condition [17]. As a result, crops fields are fatally damage and farmers fall into an economic crisis.

1.4 Objectives

- To detect crops disease in advance.
- To classify the disorder and healthy leaves.
- To use appropriate fertilizer in advance.
- To reduce the extensive crops damage.
- To grow the potato production.
- To build an automatic crops disease detection and monitoring system.
- To develop the economic progress.

1.5 Scopes and Limitations

The main task of this thesis is to detect the effected blight leaves from the healthy in advance. This is done by creating a multiclass image classification model. The model can classify the late blight, early blight, and healthy leaves properly. Since our model resolves multiclass image classification problems, we can apply it to other crops diseases where object detection is involved, and we can monitor crop growth using it. Also, the crops field can be monitored all the time by built up an automatic system. As a result, diseases can be detected in advance and reduced the crops damage to a great extent.

Our model may not predict perfectly due to obscure, shaded and cluster images of crops leaves. Collecting a diverse and representative dataset with sufficient samples for each class can be challenging. The model is computationally intensive, especially when dealing with large-scale datasets or complex architectures also training model with multiple classes will require substantial computational resources, including powerful hardware GPUs and longer training times.

1.6 Summary

In this chapter, we have discussed potato foliar disease then we covered the background of disease detection in previous and current studies. Also, we have described the problem statement and drawbacks of the manual disease detection. After that we show the objectives of our research. Finally, we have shown the various scopes of our project and some limitations.

CHAPTER 2

Literature Review

2.1 Overview

A literature review is an overview of the previously published works on a topic. The term can refer to a full scholarly paper or a section of a scholarly work such as a book, or an article. A good literature review can ensure that a proper research question has been asked and a proper theoretical framework or research methodology have been chosen [18]. We have reviewed some research papers, journals, and books for the purpose of our thesis.

2.2 Neural Networks

Artificial neural networks are usually simply called neural networks or neural nets (NNs), are computing systems inspired by the biological neural networks that constitute animal brains [19]. NNs models are composed of interconnected nodes or neurons. Each connection of nodes, like the synapses in a biological brain, can transmit a signal to other neurons. Artificial neural network is made of three layers namely input layer, output layer, and hidden layers. There must be a connection from the neurons in the input layer with the neurons in the hidden layer and the neurons from hidden layer with the neurons of the output layer.

In Figure 2.1 is shown a typical ANN architecture where the input layer takes inputs x_i and multiplied by the corresponding weight ω_1. Weight refers to a parameter that is associated with each connection between neurons. It represents the strength or importance of the connection between two neurons, and it determines how much influence the input from one neuron has on the output of another neuron. Then the hidden layer receives the raw information from the input layer for processes. In the hidden layer, the weighted inputs are summed up, and added a bias b that is an additional parameter associated with each neuron that allows the network to adjust the output based on a constant value also it represents the tendency of a neuron to be activated or not, regardless of the inputs it receives. After that, an activation function f is applied to produce the output of the neuron. Finally, the obtained value is sent to the output layer which will also process the information from the hidden layer and give the output.

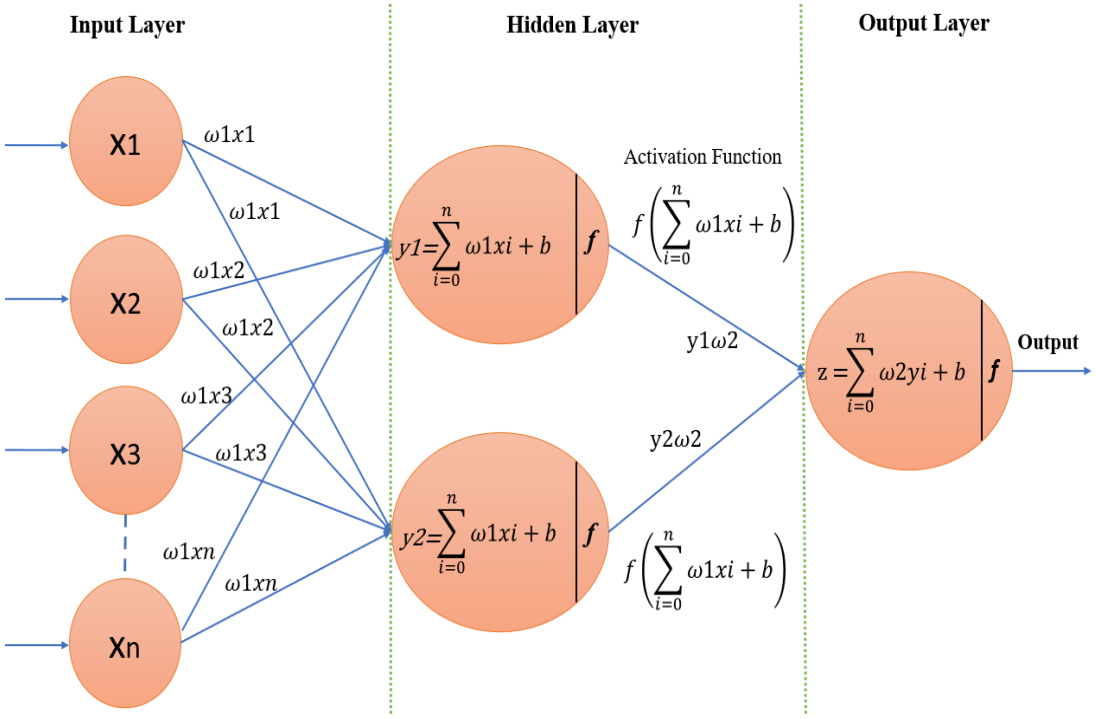

Figure 2.1 : A Typical Artificial Neural Network (ANN).

Neural networks are capable of learning and improving their performance by adjusting the strength of the connections between neurons based on feedback from the input data. These are used for a variety of tasks, including image and speech recognition, natural language processing, and predictive analytics. They have proven to be particularly effective for solving complex problems that are difficult to solve using traditional rule-based programming.

Artificial Neural Networks (ANNs) can be used for multiclass image classification tasks, but it may not perform as well because 2-dimensional images need to be converted to 1-dimensional vectors. This increases the number of trainable parameters exponentially as a result the increasing trainable parameters takes storage and processing capability which is more expensive. In this case, Convolutional Neural Networks (CNNs) are used to solve those tasks. CNNs are a specialized type of Artificial Neural Network that is particularly well-suited for tasks involving images, videos, and other grid-like data. They have gained widespread popularity and are commonly used in various computer vision applications.

Our task is on a multiclass image processing problem where classifying the different potato leaves is performed by a CNN model. Therefore, we will discuss CNN in the next section.

2.3 Convolutional Neural Networks

A Convolutional Neural Network (CNN) is a type of deep learning neural network that is commonly used for image and video processing tasks, such as image classification, object detection, and segmentation. It is designed to take advantage of the spatial structure of the input data, such as the grid-like structure of pixels in an image. It is more powerful and accurate for solving a multiclass image classification problem rather than ANN.

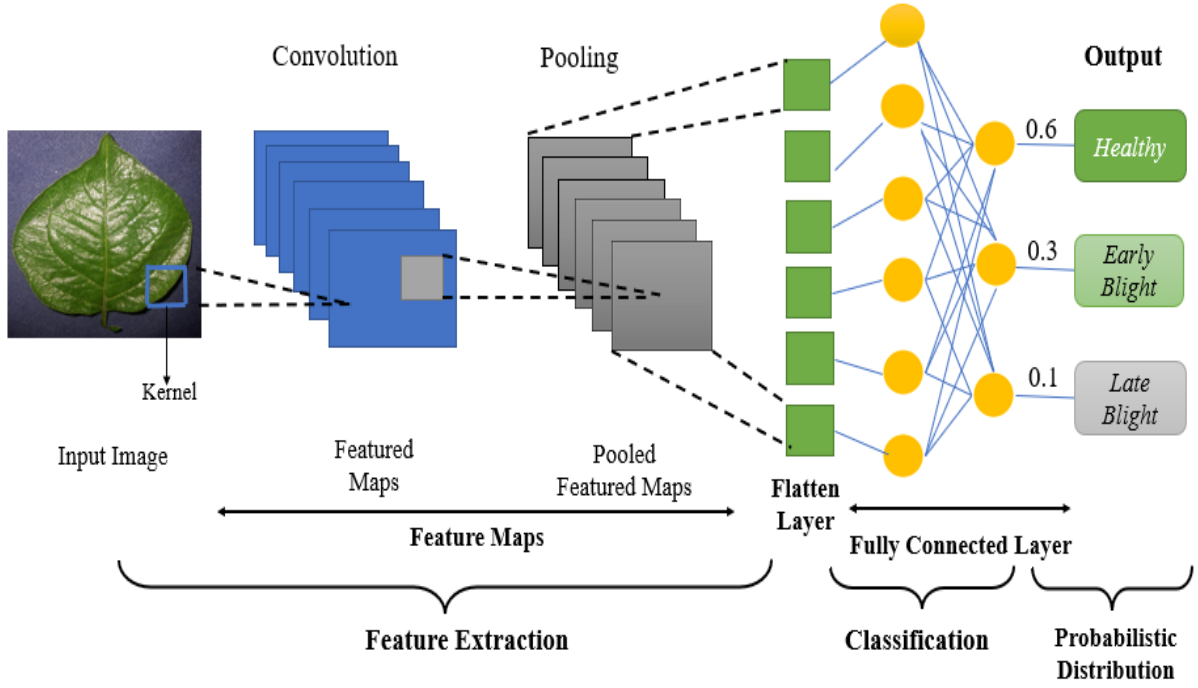

Figure 2.2 : A Typical Convolutional Neural Network (CNN).

Convolutional Neural Network consists of multiple layers that perform different types of operations on the input image. We chose to simplify its presentation by considering that it can be decomposed into two parts: a feature extraction part that involves convolution and pooling operations another is classification part consist of flattening layer and dense layers where classified the input images into different class based on their probabilistic values.

Now, we will discuss step by step on the different layers of the CNN architecture showed in Figure 2.2:

2.3.1 Feature extraction

In Convolutional Neural Networks (CNNs), feature extraction is the process of automatically identifying and extracting relevant features from raw input data, such as images, to create a representation that is more suitable for the task at hand, such as image classification or object detection. In the feature extraction part, convolution and pooling layers are involved.

2.3.1.1 Convolution layer

In the convolutional layer, the network applies a set of learnable filters, also called kernels or feature detectors, to the input image to extract local features such as edges, shapes, and textures. The filters slide or convolve over the input image, producing a set of feature maps that represent the response of each filter at each location of the image. The convolution operation is designed to capture spatial relationships between neighboring pixels and to preserve the spatial arrangement of the input data.

2.3.1.2 Pooling layer

In the pooling layer, the network reduces the spatial dimensions of the output feature maps by down sampling them. The most common pooling operation is max-pooling, which selects the maximum value within a local region of the feature map. This operation helps to reduce the sensitivity of the network to small variations in the input image and makes the feature maps more robust to noise and distortion.

2.3.2 Classification

Classification in Convolutional Neural Networks (CNNs) refers to the process of assigning a label or a category to an input image based on its content or features. In CNNs, the classification task is typically performed using the fully connected layer at the end of the network.

2.3.2.1 Flattening

The flattened layer is typically placed between the convolutional and fully connected layers. It is used to reshape the output of the convolutional layers into a 1D vector, which can be fed into the fully connected layers for classification or regression. After the convolutional and pooling layers, the output is usually a 3D tensor (height, width, depth). The flattened layer

takes this 3D tensor and reshapes it into a 1D vector by "flattening" the tensor, which means that all of the elements are arranged into a single row. For example, a 3D tensor of size (32, 32, 64) would be flattened into a 1D vector of size 64 x 1024 = 2048, where 64 is the depth of the tensor and 1024 is the total number of pixels in the height and width dimensions and 2048 means that it is a single row of numbers that contains 2048 elements, where each element is a scalar value. The flattened layer serves as a bridge between the convolutional and fully connected layers, allowing the output of the convolutional layers to be fed into a traditional neural network architecture. Without the flattened layer, the output of the convolutional layers would not have a fixed size, which would make it difficult to connect them to the fully connected layers.

2.3.2.2 Fully connected layer

The fully connected layer is a type of layer in a neural network, typically used in the final stage of the network for tasks such as classification or regression. It is also known as a dense layer or a fully connected neural network layer. This layer is useful for tasks such as image classification, where the input data can be flattened into a one-dimensional vector and fed into here for classification. In fully connected layer, each neuron is connected to every neuron in the previous layer. The output of the previous layer is flattened into a one-dimensional vector, and each neuron in the fully connected layer takes in this vector as input and computes a weighted sum, followed by an activation function. The weights and biases of the fully connected layer are learned during training through backpropagation, allowing the network to learn complex relationships between the input data and the output.

2.4 Summary

In this chapter, we have discussed the basic concept of neural network and its applications. And a typical structure of a convolutional neural network with different layers and those application with a view to multiclass image classification.

CHAPTER 3

Methodology

3.1 Overview

Methodology is a systematic explanation used to identify, select, process and analysis information about the research topic. It explains the research design and methods to understand readers about the working procedure. In this chapter, we will discuss the working flow of our proposed methodology to detect and classify the leaf disease of potato.

3.2 Proposed Methodology

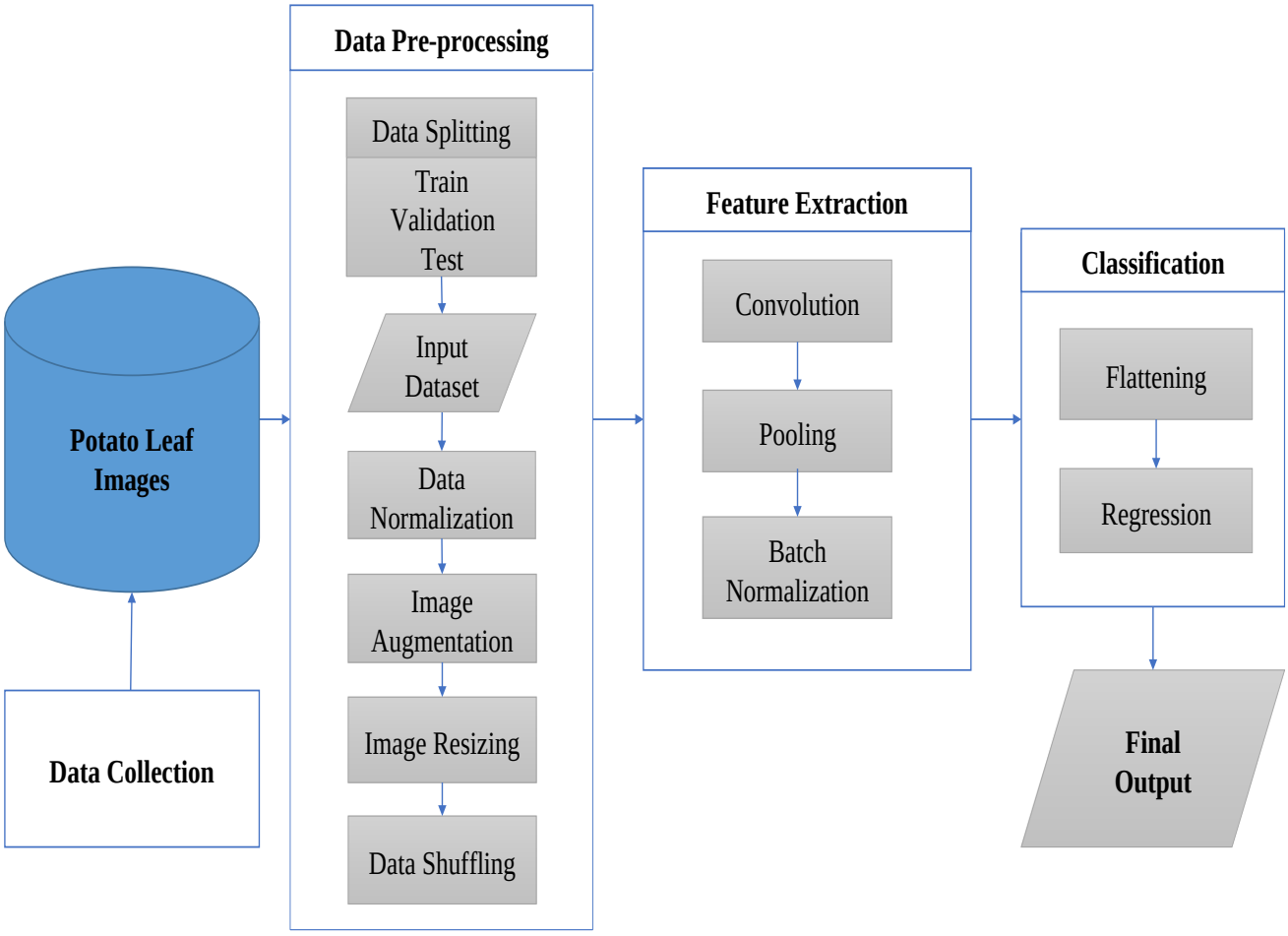

Figure 3.1 : Proposed methodology for potato leaf disease detection and classification.

The proposed methodology is built using a CNN model. CNN is an efficient recognition algorithm which is widely used in pattern recognition and multiclass image processing. It is a supervised type of classification model that can classify and detect disease properly.

Now, we will discuss the overall procedure of our proposed methodology is shown in Figure 3.1:

3.3 Data Collection

Data Collection is a process of gathering data from various sources for analysing and finding answer of a research problems. In this thesis, our input data is a single image of potato leaf. Three kinds of picture we have collected from the crops field in the presence of sunlight in different location of the crops area. These are disorder leaf and healthy leaf. We have collected two different types of disorder leaf – late blight and early blight according to their individual characteristics. The pictures are taken on a plane area by a smartphone camera.

3.4 Data Pre-processing

Data pre-processing is the process used to convert raw data into clean data set. It is an important step for data analysis. This stage improves accuracy and reliability by removes missing or inconsistent data values resulting from human or computer error. Data pre-processing also increase the data's quality and makes it easier for machine learning algorithms to read, use, and interpret data. Our data is the leaf image of potato. In the pre-processing stage, the input image converted into RGB grids of pixels with channels after that image is converted into floating-point tensors for input to neural networks and resizing, rescale and normalization has done.

3.4.1 Data splitting

Data splitting is a technique by which divide the dataset into two or more subsets. This process helps in assessing the model's performance, optimizing hyperparameters, and estimating the ability of model to generalize to unseen data. Typically, with a two-part split, one part is used to train the model and the other to evaluate or test. On the other hand, with a three-part split, one part is used to train the model, one part is to pretest or validate and the other for final testing. However, the exact split depends on the size of the dataset, the

complexity of the problem, and the available computational resources. We have categorized 80% of data for training, 10% for validation and the remaining 10% for testing purposes.

3.4.2 Input dataset

Our dataset contains potato leaf images. To input it, the dataset will set up on the machine in a specific directory structure after that we will point to these directories in the code from where fetch the images. The input is JPEG content that will decode to RGB grids of pixels with channels.

3.4.3 Data normalization

Normalization is a technique often applied as part of data preparation for machine learning algorithms. The goal of normalization is to change the values of numeric columns in the dataset to use a common scale, without distorting differences in the ranges of values or losing information. Data normalization reduces redundant data, maintains data consistency, and gives a better and quicker execution.

3.4.4 Data augmentation

Data augmentation is a technique used to reduce overfitting when training a machine learning model and increase the amount of data by adding slightly modified copies of already existing data. Machine learning models need to identify an object in any condition, even if it is rotated, flipped, zoomed in noisy image or a different brightness. This different condition of the image is created using data augmentation technique. In this case, transformed the training dataset by rotating image at different angles, flipping vertically or horizontally over one axis, zoom in/out, adding noise on original image and changing the brightness of the image. Finally, the different diversity of the training dataset makes more robust the model.

3.4.5 Image resizing

Image resizing is one of the essential operations in Convolutional Neural Networks, which are widely used for various computer vision tasks such as image classification, object detection, and segmentation. In CNNs, image resizing refers to the process of changing the spatial dimensions (width and height) of an image while preserving its content. Resizing images to a fixed size ensures that all input images have the same dimensions, which is

necessary to feed them into the network. Most CNN architectures require fixed-size inputs to operate efficiently. Resizing allows the network to process images in batches, which is crucial for parallel computing on GPUs. It also increased the computational efficiency of the model by reducing the number of pixels, resulting in faster computation and lower memory requirements. It allows CNN to process more images in a given time frame, improving training and inference speed.

3.4.6 Data shuffling

Data shuffling in a CNN refers to the randomization of the order of training data samples during each epoch of the training process. During CNN training, the model is optimized using a stochastic optimization algorithm such as stochastic gradient descent (SGD) or Adam. These algorithms work by computing the gradients of the loss function with respect to the model parameters using a small subset of the training data at a time, known as a mini batch. By repeatedly updating the model parameters based on these mini-batch gradients, the model gradually learns to make better predictions. When shuffling the training data, the order of samples within each mini-batch is randomized, ensuring that each mini-batch contains a random assortment of samples from throughout the dataset. Shuffling the training data can help to improve the performance of CNN, as it reduces the risk of the model becoming biased towards specific patterns or correlations in the data that might exist if the data is presented in a specific order. It also ensures that the model is exposed to a diverse range of samples during training, which can help it learn more robust features that generalize better to new, unseen data.

3.5 Model Building

A deep learning model typically consists of multiple layers of interconnected artificial neurons, also known as nodes, that process and transform the input data to produce the desired output. These models use complex optimization algorithms to adjust the weights and biases of the nodes, based on the error between the predicted output and the actual output, to minimize the prediction error and improve the model's performance. Our proposed model is built using convolutional neural network (CNN) that is a type of deep neural network and widely used in computer vision tasks, such as image classification, object detection, and segmentation. CNNs are designed to automatically learn and extract features from input data,

such as images, and use them to make predictions. Our aim is detection and classification of potato leaf disease that could very well performed by a CNN model.

The model is mainly divided into two main parts; one is for extracting features of the images and another for classification:

3.5.1 Feature extraction

Feature extraction in CNN is the process of identifying and extracting meaningful patterns or features from input images or data. The features are typically low-level or mid-level representations of the input data that can be used for further processing or classification. It is a crucial step in CNNs, as it enables the network to identify and learn from meaningful patterns in the input data, which can significantly improve the accuracy and robustness of the model. The overall process of feature extraction is accomplished after performing convolution and pooling operations after that performing batch normalization.

3.5.1.1 Convolution

The convolutional operation performed into the convolutional layers are responsible for performing feature extraction from the input data. The operation of a convolutional layer involves convolving a set of learnable filters with the input data. Each filter is a small matrix that slides over the input data, and at each position, performs a dot product between its weights and the corresponding region of the input. The output of this operation is a single number, which corresponds to a specific feature in the input data.

By applying multiple filters to the input data simultaneously, the convolutional layer produces a set of feature maps, each of which highlights a different feature in the input. These feature maps are then passed through a non-linear activation function, such as the rectified linear unit (ReLU), to introduce non-linearity into the model. The size of filters and the stride (the amount by which the filter moves between each computation) determine the granularity of the features that are extracted from the input data. The filters themselves are learnable parameters that are optimized during the training process using backpropagation. The convolution operation is a powerful tool for feature extraction in CNNs, allowing the network to automatically learn and identify relevant features in the input data.

3.5.1.2 Pooling

The dimension of convoluted output features is reduced by the pooling layer. This is the next layer of convolution layer. The main purpose of a pooling layer in a convolutional neural network is to reduce the spatial dimensions of the output volume from the convolutional layer, while also retaining the most important features. Pooling works by partitioning the input image into non-overlapping rectangular regions and then computing a summary statistic, such as the maximum or average value, for each region. The resulting output volume has fewer dimensions than the input volume, which reduces the computational complexity of the network and helps prevent overfitting. Pooling can also help with translation invariance, which means that CNN can still recognize an object even if its position in the input image is slightly shifted. This is because the pooling operation takes the maximum or average value within a region, so it is less affected by small translations of the object within the region. Overall, the pooling layer is an important component of CNNs, helping to reduce the spatial dimensions of the input volume while preserving the most important features and aiding in translation invariance.

3.5.1.3 Batch normalization

Batch normalization is a technique used to improve the performance of deep neural networks, including convolutional neural networks (CNNs). The purpose of batch normalization in convolutional neural networks (CNNs) is to improve the training speed, stability, and performance of the network. During the training process of a deep neural network, the distribution of the input to each layer changes as the parameters of the previous layers are updated. This phenomenon is called internal covariate shift. Internal covariate shift makes it difficult to train deep neural networks, as the optimization process becomes unstable, and the network may converge to a suboptimal solution. Batch normalization addresses this problem by normalizing the input to each layer, making the distribution more stable and reducing the internal covariate shift. By subtracting the mean and dividing by the standard deviation of the activations of each layer over a mini-batch of samples, batch normalization ensures that the activations have zero mean and unit variance. This normalization helps to stabilize the learning process and makes the optimization of deep neural networks faster and more effective. It also helps to reduce overfitting, as it adds a small amount of noise to the activations of each layer, which acts as a form of regularization. This regularization can

improve the generalization performance of the network on unseen data. Batch normalization is an important technique for improving the performance of convolutional neural networks, especially when dealing with deep architectures and large datasets.

3.5.2 Classification

In Convolutional Neural Networks (CNNs), classification refers to the task of assigning a label or category to an input image. CNNs are particularly well-suited for image classification tasks because they are designed to learn and extract relevant features from images. It has numerous applications in computer vision, such as object detection, image recognition, and facial recognition. A typical CNN for image classification consists of several convolutional layers, pooling layers, and one or more fully connected layers. The convolutional layers apply a set of filters to the input image to extract features, while the pooling layers downsample the feature maps to reduce their spatial dimensionality. The fully connected layers then take the flattened feature maps and produce a probability distribution over the possible categories or labels.

3.5.2.1 Flattening

The flattening layer in a Convolutional Neural Network (CNN) is to convert the output of the convolutional layers, which is a 3D tensor, into a 1D feature vector that can be fed into a fully connected layer for classification or regression tasks. The output of the convolutional layers is typically a 3D tensor with dimensions (height, width, depth). The height and width dimensions correspond to the spatial dimensions of the input image, while the depth dimension corresponds to the number of filters in the convolutional layer. The Flatten layer is used to flatten this 3D tensor into a 1D vector with length equal to the product of the height, width, and depth dimensions. By flattening the output of the convolutional layers, we can feed it into a fully connected layer, which can then perform the final classification or regression task. The output of the fully connected layer is typically a vector of probabilities, which represents the likelihood of the input belonging to each class in the classification task or the predicted value in the regression task.

3.5.2.2 Final regression and classification

The final regression or classification process is accomplished after flattening the output features of convolutional layers into 1D feature vector that can be fed into a fully connected layer for classification or regression tasks. A fully connected layer is a type of layer in a convolutional neural network (CNN) where each neuron in the layer is connected to every neuron in the previous layer. This means that the output of each neuron in the dense layer is a weighted sum of the outputs of all the neurons in the previous layer, followed by an activation function. Dense layers are typically used at the end of a CNN architecture to perform the final classification or regression task based on the features learned by the preceding convolutional and pooling layers. The dense layer takes the output of the last convolutional or pooling layer and flattens it into a one-dimensional vector before passing it through the fully connected layer. The purpose of dense layer is to learn a nonlinear combination of the features extracted by the preceding layers that can discriminate between the different classes or predict the target variable. The output of the dense layer is then passed through a final activation function, such as softmax for classification or linear for regression. After that the activation function produces the final output of the network.

3.6 Summary

In this chapter, we have discussed the proposed methodology to detect and classify the disorder leaf from the healthy. In the proposed methodology, several processes have been discussed like data collection, data pre-processing and building our proposed model where are used CNN algorithm.

CHAPTER 4

Implementation

4.1 Overview

In this chapter, we will discuss how we implement the methodology of the potato leaf disease detection system using CNN algorithm. We will discuss about various tools and technology used for the implementation of our methodology. The implementation is performed using by python programming language and the jupyter notebook IDE is used to write and test our programs. There are various tools used such as TensorFlow, Keras, NumPy, Pandas to performed deep learning operation and data analysis task also matplotlib for visualizing our data and sklearn library to measure the accuracy of model.

4.2 Environment Set-up

This process involves identifying the required software components, installing them on the computer or server, and setting up the necessary configurations and parameters. To implement our task, we have installed Python 3.9.7 version and Tensorflow 2.11.0 library also Jupyter notebook IDE platform that become increasingly popular in data science and scientific computing communities because this allows for exploratory data analysis, data visualization, and the creation of reproducible research.

4.3 Tools and Technologies

Our main goal is to resolve a multiclass image classification problem that will be performed by deep CNN model. We have implemented that using by python programming language with following tools and technologies:

4.3.1 Python

Python is a high-level, interpreted programming language that was first released in 1991.Python's syntax is simple and clean, and its code is often easier to read and maintain than code written in other languages. One of the main strengths of Python is its versatility. This language can be used for a wide range of applications, including web development,

scientific computing, data analysis, artificial intelligence, machine learning, and more. It has a large and active community of developers who contribute to the development of new libraries and frameworks that make it even more powerful and flexible. Python is an interpreted language, which means that it does not need to be compiled before it can be run. This makes it easy to write and test code quickly, and it also means that Python code can be run on any platform without the need for any special software or hardware. It is an object-oriented language, which means that it allows you to define classes and objects, and to create complex data structures that can be used to build sophisticated applications. It also has several built-in data types and functions that make it easy to manipulate data and perform calculations. Python's popularity has led to the development of many third-party libraries and frameworks that make it even more powerful and versatile. Some of the most popular libraries and frameworks for Python include NumPy, Pandas, Matplotlib, Scikit-learn, Django, Flask, and TensorFlow. We will discuss below some libraries that are useful in our implementation.

4.3.2 TensorFlow

TensorFlow is an open-source software library for numerical computation and machine learning developed by Google Brain team. It was first released in 2015 and has become one of the most popular machine learning frameworks used by developers and researchers. It allows users to build and train various types of machine learning models, including deep learning models, on different hardware platforms such as CPUs, GPUs, and TPUs. It is widely used for building and training neural networks, including deep neural networks. In fact, TensorFlow was originally designed for developing and training deep learning models. TensorFlow provides a wide range of tools and libraries that make it easy to build and train different types of neural networks, including fully connected neural networks, convolutional neural networks (CNNs), recurrent neural networks (RNNs), and more. For example, to build a fully connected neural network using TensorFlow, you can define the network architecture as a set of layers, where each layer applies a set of linear and non-linear transformations to the input data. You can then use the backpropagation algorithm to update the weights of the network during training, minimizing a loss function that measures the difference between the predicted output and the true output. Similarly, to build a CNN using TensorFlow, you can define a set of convolutional layers that apply a set of filters to the input data, extracting local features that are useful for the task at hand. You can then use pooling layers to downsample

the feature maps and reduce the computational cost of the network, and fully connected layers to classify the input data. TensorFlow also provides a variety of optimization algorithms and regularization techniques that can be used to improve the performance of the neural network and prevent overfitting.

4.3.3 Keras

Keras is a high-level neural network API that runs on top of TensorFlow (as well as other deep learning frameworks such as Theano and CNTK). It was developed with the goal of making it easy to build and experiment with different types of deep learning models, while still providing the flexibility and power of TensorFlow. It provides a simple and intuitive interface for building and training deep learning models. It includes a wide range of pre-built layers, such as convolutional layers, recurrent layers, and fully connected layers, that can be easily combined to form complex neural network architectures. It also provides several utilities for data preprocessing, such as image augmentation and sequence padding, as well as tools for monitoring and visualizing the training process.

One of the main advantages of using Keras with TensorFlow is that it allows for rapid prototyping and experimentation with different neural network architectures. Since Keras abstracts away many of the low-level details of TensorFlow, it can be used to quickly test out different architectures and hyperparameters, without needing to worry about the details of how the model is implemented. Furthermore, Keras is designed to be user-friendly and accessible to developers and researchers of all skill levels. It's simple and intuitive interface makes it easy to get started with deep learning, while still providing the flexibility and power needed for more advanced tasks. As a result, it has become one of the most popular deep learning APIs used by developers and researchers today.

4.3.4 NumPy

NumPy is a fundamental library for scientific computing in Python that provides support for powerful N-dimensional array objects and tools for working with these arrays. It is widely used in building neural networks and other machine learning algorithms. NumPy arrays are used to store and manipulate the data used in neural networks. The input data and the weights of the neural network are typically represented as NumPy arrays. The output of the neural network is also often a NumPy array. NumPy is a critical tool for building neural networks

and other machine learning algorithms in Python. Its powerful N-dimensional array objects and tools for working with these arrays make it easy to manipulate and transform data, initialize weights, and perform operations needed for building and training neural networks.

4.3.5 Matplotlib

Matplotlib is a data visualization library for Python that allows you to create a wide range of charts, plots, and graphs. It provides a simple yet powerful interface for creating high-quality visualizations in Python. Matplotlib provides a wide range of customizable plot types, including line plots, scatter plots, bar plots, histograms, pie charts, and more. It also supports various types of data visualization, including 2D and 3D visualizations, interactive plots, and animations. One of the advantages of Matplotlib is its flexibility and customization. You can customize almost every aspect of the plot, including colors, fonts, labels, and annotations. Additionally, Matplotlib integrates well with other Python libraries such as NumPy, Pandas, and SciPy, making it a powerful tool for data visualization and analysis. It is widely used in scientific computing, data analysis, and machine learning, among other fields.

4.3.6 Scikit-learn

Scikit-learn is a Python library for machine learning that provides a comprehensive set of tools and algorithms for data preprocessing, model selection, and model training and evaluation. It is built on top of other scientific computing libraries such as NumPy, SciPy, and Matplotlib. It is designed to be simple, efficient, and accessible to everyone, making it a popular choice for both beginners and experts in the field of machine learning. Scikit-learn is widely used in industry and academia for a variety of applications, such as natural language processing, image recognition, and predictive modeling. Its simplicity, flexibility, and extensive documentation make it an excellent choice for machine learning tasks in Python.

4.3.7 Seaborn

Seaborn is a Python data visualization library based on Matplotlib that provides a high-level interface for creating informative and attractive statistical graphics. While Matplotlib provides basic functionality for creating visualizations, Seaborn provides additional features and customization options to make it easier to create complex and aesthetically pleasing

visualizations. Seaborn provides a wide range of visualizations for different types of data, including:

- Scatter plots, line plots, and bar plots
- Heatmaps and cluster maps
- Pair plots and joint plots
- Violin plots, box plots, and swarm plots
- Factor plots and categorical plots

Seaborn also includes specialized visualizations for statistical analysis, such as regression plots, distribution plots, and time series plots. One of the key advantages of Seaborn is its ability to create visually appealing graphics with minimal code. It provides a range of customization options, such as color palettes, themes, and style sheets, that make it easy to create consistent and professional-looking visualizations. In addition to its visualizations, Seaborn also includes tools for data exploration and analysis, such as tools for aggregating and summarizing data, as well as tools for computing correlations and statistical tests.

4.4 Data Pre-processing

Data pre-processing is the process of transforming raw data into a format that can be easily understood by machine learning algorithms. The objective of data pre-processing is to prepare the data for analysis, improve the accuracy of the machine learning model, and reduce the risk of errors. Data Pre-processing typically involves several steps, the following techniques we will apply to pre-process our data:

4.4.1 Input dataset

Our dataset is divided into three parts: One part is used to train the model; one part is for validation and the other for final testing. We can input dataset by referring to the file path stored on the local file system or a remote server.

For example:

```
train_dataset_path='C:/Users/HP/Desktop/PYTHON/THESIS/Book/Training'

validation_dataset_path='C:/Users/HP/Desktop/PYTHON/THESIS/Book/Validation'

test_dataset='C:/Users/HP/Desktop/PYTHON/THESIS/Book/Testing'
```

4.4.2 Data normalization

Normalization is a common preprocessing technique used in machine learning to scale and transform input data so that it has a common scale or distribution. The goal of normalization is to improve the performance and stability of machine learning algorithms by ensuring that the features are on a similar scale and that they have a similar range of values. We will normalize our data by a simple technique called Min-Max Scaling that is performed using by ImageDataGenerator class in Keras. that performs various data augmentation and preprocessing steps to the training data, including rescaling, rotation, shifting, shearing, zooming, and flipping. The following statement is showed the implementation of Min-Max Scaling:

```
Train_Data = ImageDataGenerator(rescale=1.0/255)
```

4.4.3 Image augmentation

Image augmentation is a technique commonly used in Convolutional Neural Networks (CNNs) to artificially increase the size of the training dataset by applying various image transformations to the original images. Image augmentation makes CNN more robust to variations in the input images by increasing training data, improving performance on unseen data, reducing overfitting, and increasing the balance of the dataset. The ImageDataGenerator class from Keras is used to define the image augmentation parameters, such as rotation range, shift range, zoom range, and flipping:

```
data_generator = ImageDataGenerator(
    rotation_range = 30,         # Rotate images by up to 30 degrees
    width_shift_range = 0.1,  # Shift images horizontally by up to 10% of the image width
    height_shift_range = 0.1, # Shift images vertically by up to 10% of the image height
    zoom_range = 0.2,           # Zoom in/out of images by up to 20%
    horizontal_flip = True,     # Flip images horizontally
    vertical_flip = False,        # Do not flip images vertically
)
```

4.4.4 Image resizing

In deep learning and convolutional neural networks, it is common to resize images to a certain size before feeding them into the network. This can be define by the following code statement:

```
Image_Resizing = tf.image.resize(image, size=(IMG_WIDTH, IMG_HEIGHT))
```

Also, passing the image size as argument into the 'train_datagen.flow_from_directory()'. It is a method in the Keras that is used for loading image data from a directory and generating batches of augmented data for training a deep learning model. It is typically used when working with large image datasets that cannot fit into memory all at once.

For example:

```
train_generator=train_datagen.flow_from_directory(train_datase
                t_path,target_size=(IMG_WIDTH,IMG_HEIGHT),
                batch_size=BATCH_SIZE)
```

4.4.5 Data shuffling

Data shuffling is a process of mixing up the images randomly from a dataset. It plays an important role in image preprocessing where data is sorted by their class or target. Shuffling Data reduces variance and makes sure that models remain general and overfit less. In Python, Data shuffling is implemented by the following code statement for multiclass image classification:

```
Shuffle_Data = train_datagen.flow_from_directory(shuffle=True)
```

In the context of a convolutional neural network, 'shuffle=True' typically refers to shuffling the training data during each epoch of the training process. When training a CNN, it's common to use mini-batches of training data, where each mini-batch consists of a small subset of the training examples. By default, these mini-batches are sampled in order from the training data set, so each mini-batch contains consecutive examples. However, setting 'shuffle = True' means that the order of the training examples is randomized during each epoch. This helps to break any patterns or correlations in the ordering of the data, which can be especially important if the data is naturally ordered (such as time-series data). By shuffling the data, CNN is forced to learn more robust features that are not dependent on the order of the data.

4.5 Model Building

Our model is built by CNN algorithm. The purpose of creating the model in CNN is to learn a set of weights and biases that can accurately classify images or other types of data. The model is trained on a large dataset of labeled images, using a process called backpropagation, where the error between the predicted output and the true output is minimized using optimization techniques like stochastic gradient descent. Once the model is trained, it can be used to classify new images with high accuracy. Now we will see what operations are performed our model on the input dataset.

4.5.1 Convolution

The convolution operation is a mathematical operation that involves applying a set of learnable filters, also known as kernels, to the input image. These filters have small spatial dimensions (typically 3x3 or 5x5) and are applied to every possible position of the input image. By applying multiple filters to the input image, the convolutional layer can learn to extract a set of high-level features called feature maps that are important for the given task, such as edges, textures, and patterns. The output feature maps produced by the convolutional layer are then fed into subsequent layers of the CNN for further processing, such as pooling and fully connected layers, to ultimately make predictions on the input image. Implementing a convolutional layer in a Convolutional Neural Network (CNN) involves several steps, including defining the input and output shapes, specifying the number and size of the filters, performing the convolution operation, and applying activation functions. Here is a simple example of how to code a convolutional layer in Python using the Keras API:

Importing Required Library:

```
from tensorflow.keras.layers import Conv2D
```

Create a new instance of a Sequential model:

```
model = Sequential()
```

Defining input and output shapes:

```
input_shape = (32, 32, 3)
output_channels = 64
```

kernel_size = (3, 3)

Creating a convolutional layer:

conv_layer = Conv2D(output_channels, kernel_size, activation='relu', padding='same', input_shape=input_shape)

Performing convolution on input data:

input_data = np.random.rand(1, 32, 32, 3)

output_data = conv_layer(input_data)

Printing output shape:

print(output_data.shape)

In this example, we first import the Conv2D layer from the Keras API. We then define the input and output shapes of the layer, which includes the shape of the input data (32x32 pixels with 3 color channels) and the number of output channels (64). We also specify the size of the filter (3x3) and the activation function to use (ReLU). We then create a convolutional layer using the Conv2D function and set the padding type to 'same' to ensure that the output feature map has the same spatial dimensions as the input. Finally, we perform convolution on some input data (randomly generated in this case) using the conv_layer object and print the output shape.

4.5.2 Downsampling or pooling

Downsampling refers to the process of reducing the spatial resolution of the feature map to extract the most important information and reduce the computational complexity of the network. This is typically accomplished using a pooling layer, which takes the input feature map and applies a downsampling operation to it. There are several types of pooling layers that can be used for downsampling in a CNN, including max pooling, average pooling, and global pooling. Max pooling is the most used type of pooling layer, and it works by taking the maximum value of each non-overlapping sub-region of the input feature map. We will use this operation in our model.

Implementing Max pooling involves several steps as follows:

Importing Required Library:

from keras.layers import MaxPooling2D

Adding a max pooling layer:

```
model.add(MaxPooling2D(pool_size=(2,2)))
```

In this example, the MaxPooling2D layer is added after the convolutional layer with a pooling window size of (2,2). This means that the input feature map will be downsampled by a factor of 2 in both the horizontal and vertical dimensions, resulting in a feature map with half the spatial resolution.

4.5.3 Batch normalization

Batch normalization is typically used in convolutional neural networks to improve the speed, stability, and generalization of the network. The normalization is performed on a mini-batch of the training data. By normalizing the input data to each layer, batch normalization can help to reduce the internal covariate shift and stabilize the training process. It is inserted after a convolutional or fully connected layer in a CNN architecture, and it can be followed by activation functions and other layers as needed.

Importing Required Library:

```
from keras.layers import BatchNormalization
```

Adding batch normalization layer:

```
model.add(BatchNormalization())
```

In this example, a BatchNormalization layer is added after the convolutional layer. The normalization is performed on the activations within each batch, using the mean and variance of the batch to normalize the data.

4.5.4 Flattening

Flattening is a process that transform the 2-dimensional feature maps produced by the convolutional and pooling layers into a 1-dimensional vector that can be fed into a fully connected layer for classification. This is necessary because fully connected layers only accept 1-dimensional inputs. Here is an example code for adding a flattening layer to a CNN using Keras:

Importing Required Library:

```
from keras.layers import Dense, Flatten
```

Flattening the output:

```
model.add(Flatten())
```

In this example, a Flatten layer is added after the max pooling layer. The output of the flattened layer is a 1-dimensional vector that is passed through a fully connected layer and an output layer.

4.5.5 Classification

Dense layer performs the final classification of the CNN model. It is typically added at the end of the network, after the convolutional and pooling layers. It takes the flattened output features from the pooling layers as input and combines them in a way that allows the network to make a prediction about the class of the input image. The dense layer consists of a set of neurons, each of which is connected to every neuron in the previous layer. The output of the dense layer is computed by taking a weighted sum of the inputs and passing the result through an activation function, such as a sigmoid or ReLU. Let's see the implementation procedure of the dense layer by a simple example:

Importing Required Library:

```
from keras.layers Dense
```

Adding a dense layer with 64 neurons and ReLU activation:

```
model.add(Dense(64, activation='relu'))
```

Adding the output layer with 10 neurons and softmax activation for multi-class classification:

```
model.add(Dense(10, activation='softmax'))
```

In this example, we first add several convolutional and pooling layers to the model to extract features from the input images. Then, we flatten the output from the convolutional layers into a one-dimensional vector before adding a dense layer with 64 neurons and ReLU activation. Finally, we add an output layer with 10 neurons and softmax activation for multi-class classification.

4.5.6 Proposed model

Our proposed model has been built by CNN algorithm that performs more effectively for classification of disorder and healthy potato leaves. The model is shown below:

```
def create_model():
    model = Sequential([

    Conv2D(filters=64, kernel_size=(5, 5), padding='same', input_shape=(IMG_WIDTH,
    IMG_HEIGHT, 3)),
    Activation('relu'),
    MaxPooling2D(pool_size=(2, 2)),
    BatchNormalization(),

    Conv2D(filters=32, kernel_size=(3, 3), padding='same', kernel_regularizer=l2(0.01)),
    Activation('relu'),
    MaxPooling2D(pool_size=(2, 2)),
    BatchNormalization(),

    Conv2D(filters=32, kernel_size=(3, 3), padding='same', kernel_regularizer=l2(0.01)),
    Activation('relu'),
    MaxPooling2D(pool_size=(2, 2)),
    BatchNormalization(),

    Flatten(),
    Dense(units=70, activation='relu'),
    Dropout(0.5),
    Dense(units=3, activation='softmax')
])
 return model
```

This model has three convolutional layers, each followed by a max pooling layer. After each pooling layer, a batch normalization is performed to reduce the computational complexity after that added two fully connected layers. The input shape is (256,256,3) because we are using the RGB images of size 256x256. The last layer has 3 neurons with a softmax activation function because we will classify the images into 3 classes (digits 0-2) for late blight, early blight, and healthy potato leaves.

4.6 Model Compilation

In a CNN algorithm, model compilation refers to the process of configuring the model for training by specifying the optimizer, loss function, and metrics to be used during the training process. The optimizer is responsible for updating the weights of the neural network during training, to minimize the loss function. The loss function is a measure of how well the model is performing on the training data, and it is used to guide the optimization process. In CNNs, the most used loss function is cross-entropy, which is a measure of the difference between the predicted class probabilities and the true class labels. Finally, metrics are used to evaluate the performance of the model during training and testing. Common metrics used in CNNs include accuracy, precision, recall, and F1-score. We have used the following parameter to compile our model:

```
optimizer = Adam(learning_rate=0.01)
cnn_model.compile(optimizer=optimizer, loss=CategoricalCrossentropy(),
metrics=['accuracy'])
```

Here, we have used the Adam optimizer, cross-entropy loss function, and accuracy as the evaluation metric. By compiling the model with these settings, we can then begin training the model on our dataset.

4.7 Model Training

Model training refers to the process of teaching a neural network to perform a specific task by showing it a large amount of data. The goal of training a neural network is to find a set of weights that produces good performance on the evaluation metric. During training, the neural network adjusts its internal parameters (also known as weights) in order to minimize the difference between its output and the expected output for each input in the training data. This process is usually done by using an optimization algorithm such as stochastic gradient descent (SGD) to update the weights of the network based on the error between the predicted output and the actual output. The code on how to train model is shown below:

```
history = cnn_model.fit(train_generator, epochs=50,validation_data=validation_generator,
        verbose=2,callbacks=[reduce_lr]
```

4.8 Model Evaluation

Evaluation is the process of measuring how well the trained neural network performs on new, unseen data. This is typically done by using a separate dataset, known as the validation or test set, which is not used during training. The neural network is given inputs from the test set, and its output is compared to the expected output. The performance of the neural network is then measured using a predefined evaluation metric, such as accuracy, precision, recall, F1-score, or mean squared error (MSE).

We can evaluate our model by the following code:

```
test_loss, test_accuracy = cnn_model.evaluate(test_generator,batch_size=BATCH_SIZE)

print(f"Test Loss:     {test_loss}")

print(f"Test Accuracy: {test_accuracy}")
```

Here, we evaluate the model on the test data using the evaluate() function, and print out the test loss and accuracy.

4.9 Predictions

Predictions refer to the output of a trained model when it is given input data. The goal of machine learning is to train a model that can make accurate predictions on new, unseen data. The accuracy of a model's predictions is evaluated using performance metrics such as accuracy, precision, recall, or F1 score. These metrics compare the model's predictions to the true labels or values of the test data to determine how well the model is performing. Model predictions are an essential part of machine learning because they allow us to make predictions about new, unseen data. These predictions can be used in a variety of applications, such as image classification, natural language processing, and speech recognition. We will predict our model accuracy by a predictions function is shown below:

```
Predictions = cnn_model.predict(test_ds)

y_pred = np.argmax(Predictions, axis=1)

y_true = test_generator
```

Prediction Function:

```
def  predict(model,img):
```

```
    img_array=tf.keras.preprocessing.image.img_to_array(image        s[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
```

This function returns a predicted class of potato leaf with a confidence value.

Plotting Images:

```
plt.figure(figsize=(15,15))
for images,labels in test_ds.take(1):
  for i in range(9):
    plt.imshow(images[i].numpy().astype("uint8"))
    All=plt.subplot(3,3,i+1)
    predicted_class,confidence=predict(create_model(),images[i].numpy())
    actual_class=class_names[labels[i]]
    plt.title(f"Actual:{actual_class},\nPredicted:{predicted_class}.
   \nConfidence:{confidence}%")
    plt.axis("off")
```

4.10 Accuracy and Loss Curve

The accuracy and loss curve are plotted together to evaluate the performance of a trained model. The accuracy curve shows how well the model can classify the input data, while the loss curve shows how well the model is converging during training. If the accuracy is high and the loss is low, it indicates that the model is performing well and can make accurate predictions on new data. If the accuracy is low and the loss is high, it indicates that the model is not performing well and may need to be retrained or adjusted. The accuracy and loss of our model is calculated by the following code statements:

```
history = cnn_model.fit(train_generator, epochs=EPOCHS,
                    validation_data = validation_generator,
          verbose=2,callbacks=[reduce_lr])
train_accuracy = history.history['accuracy']
val_accuracy = history.history['val_accuracy']
train_loss = history.history['loss']
val_loss = history.history['val_loss']
learning_rate = history.history['lr']
```

4.11 Summary

In this chapter, we have discussed the implementation procedure of our system. We talked about the necessary libraries involving the implementation of our project and data preprocessing, model building, compiling, and training techniques. Finally, we discussed evaluation, and prediction of our model.

CHAPTER 5

Results and Analysis

5.1 Overview

This chapter will describe the outcomes of our methodology. We will see various results and numerical analysis of the implementation and a comparative discussion of different accuracy matrices over the prediction of our model.

5.2 Evaluation Metrics

Evaluation metrics refer to the quantitative measures used to assess the performance and effectiveness of a machine learning model or algorithm. These metrics provide objective assessments of how well the model is performing on a given task or dataset. They help in comparing different models, tuning hyperparameters, and making informed decisions about the model's suitability for a particular problem. The choice of evaluation metrics depends on the nature of the problem. There are various evaluation metrics used in different machine learning problems such as Accuracy, Precision, Recall, F1 score, ROC curve, Confusion Matrix, R-squared, Mean Squared Error etc. Now, we will evaluate our model using some evaluation metrics that are suitable for the model:

5.3 Accuracy

Accuracy is a commonly used evaluation metric in machine learning that measures the proportion of correctly classified instances out of the total number of instances in a dataset. It provides a simple and intuitive measure of the overall performance of a classification model. We will classify in different images of potato leaf that is a multiclass image classification problem. The accuracy for multiclass image classification is calculated by the total number of correctly classified images for each class and divided by the total number of images.

$$\boldsymbol{Accuracy} = \frac{\boldsymbol{Number\ of\ Correctly\ Classified\ Images}}{\boldsymbol{Total\ Number\ of\ Images}}$$

Our model has gained a total of 94% accuracy over 1350 potato leaf images.

5.4 Confusion Matrix

A confusion matrix for multiclass image classification is a 2D matrix that summarizes the performance of a machine learning algorithm in classifying images into multiple classes. The rows of the matrix represent the true labels of the images, and the columns represent the predicted labels of the images. Each element in the matrix shows the number of images that belong to a particular true label or a particular predicted label. The diagonal elements of the matrix represent the number of images that were correctly classified, while the off-diagonal elements represent the misclassified images. We have three classes of leaf images such as early blight, healthy and late blight. Now, we will see the number of actual class and predicted class of a particular type of image by a confusion matrix:

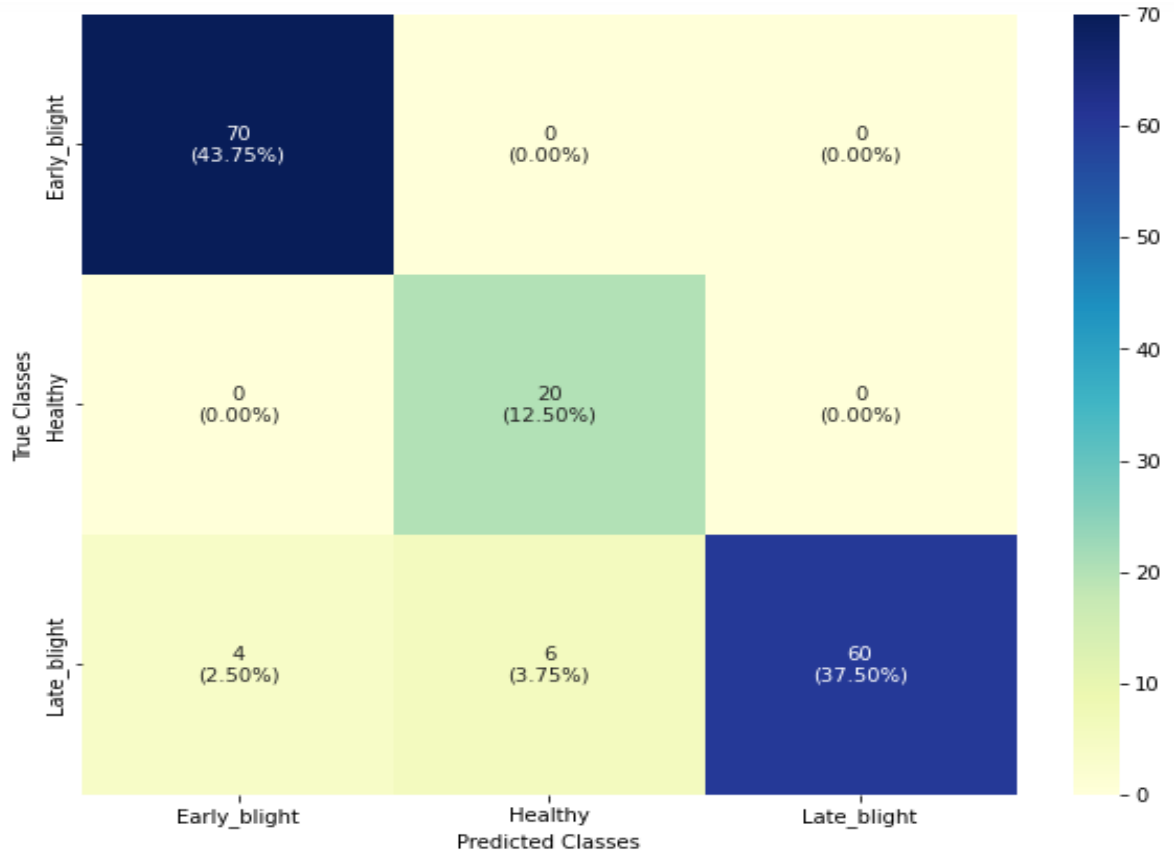

Figure 5.1 : Confusion matrix for potato leaf disease detection and classification.

The confusion matrix in Figure 5.1 is shown that there are 70 images belong to early blight class are correctly classified, while there is no image of class early blight are misclassified as class healthy and late blight, respectively. Similarly, there are 20 images belonging to class

healthy that are correctly classified, while any image not belonging to class healthy are misclassified as class early and late blight respectively. Similarly, there are 60 images belonging to class late blight that are correctly classified, while 4 and 6 images belonging to class late blight are misclassified as class early blight and healthy, respectively.

5.5 Classification Report

Classification report is used to evaluate the performance of a model. It provides a summary of the model's accuracy and other metrics that are useful for understanding its behavior. It typically includes a table that summarizes the precision, recall, and F1 score for each class, as well as the overall performance metrics averaged across all classes. It is a useful tool for evaluating the effectiveness of a CNN model for multiclass image classification tasks and identifying areas for improvement. Now, we will discuss the various performance metrices of the classification report of our model which results are shown in Table 5.1

5.5.1 Precision

Precision is the ratio of true positives to the total number of predicted positives, which measures the accuracy of the positive predictions. It means that how many of the correctly predicted cases actually turned out to be positive. Precision is useful in the cases where False Positive is a higher concern than False Negatives. It measures the model's ability to correctly identify positive samples.

Precision for multiclass classification,

$$\boldsymbol{Precision} = \frac{\sum_{i=1}^{n} \boldsymbol{TruePositives_i}}{\sum_{i=1}^{n} \boldsymbol{TruePositives_i} + \sum_{i=1}^{n} \boldsymbol{FalsePositives_i}}$$

5.5.2 Recall (Sensitivity)

Recall is the ratio of true positives to the total number of actual positives. It explains how many of the actual positive cases we were able to predict correctly with our model. Thus, it measures the model's ability to identify all positive samples. It is important when False Negative is of higher concern than False Positive. In medical diagnosis where it doesn’t matter to predict positive of an actual negative result, but it will be dangerous when actual disease is not predicted.

Recall for multiclass classification,

$$\boldsymbol{Recall} = \frac{\sum_{i=1}^{n} \boldsymbol{TruePositives_i}}{\sum_{i=1}^{n} \boldsymbol{TruePositives_i} + \sum_{i=1}^{n} \boldsymbol{FalseNegatives_i}}$$

5.5.3 F1 Score

F1 Score is the harmonic mean of precision and recall, which provides a balanced measure of the model's overall performance. It is maximum when Precision is equal to Recall. F1 Score could be an effective evaluation metric when False Positive and False Negative are equally costly, adding more data the outcome is not changed remarkably and True Negative is high.

F1 Score for multiclass classification,

$$\boldsymbol{F1\ Score} = \frac{\boldsymbol{2 \times (Recall \times Precision)}}{\boldsymbol{Recall + Precision}}$$

5.5.4 Support

Support is the number of samples or instances in each class of a given dataset. It is an important metric to consider while evaluating the performance of a classification model because it helps us understand the distribution of classes in the dataset and allows us to understand whether the model is performing well on all classes or only on some of them. It can also highlight class imbalances or bias in the data, which may affect the accuracy of the model's predictions.

Table 5.1 : Classification report for the proposed CNN model.

	precision	**recall**	**f1-score**	**support**
Early Blight	0.95	1.00	0.97	70
Healthy	0.77	1.00	0.87	20
Late Blight	1.00	0.86	0.92	70
accuracy			0.94	160
macro avg	0.91	0.95	0.92	160
weighted avg	0.95	0.94	0.94	160

5.6 Accuracy and Loss Curve

Accuracy and loss curve are two common metrics used to evaluate the performance of a model during training and testing. They are plotted based on training and validation dataset. We can see from the accuracy and loss curve if a model has learned well or not.

5.6.1 Training and validation accuracy

The accuracy curve is a plot of the accuracy of the model on the training and validation datasets against the number of epochs during training. The main purpose of the accuracy curve is to help identify whether the model is learning to classify the images correctly. If the accuracy curve shows a steady increase over time for both the training and validation datasets, this suggests that the model is learning to generalize well to new data. However, if the accuracy curve shows a high accuracy on the training dataset but a low accuracy on the validation dataset, this suggests that the model is overfitting the training data and may not perform well on new, unseen data.

Accuracy curve of our model,

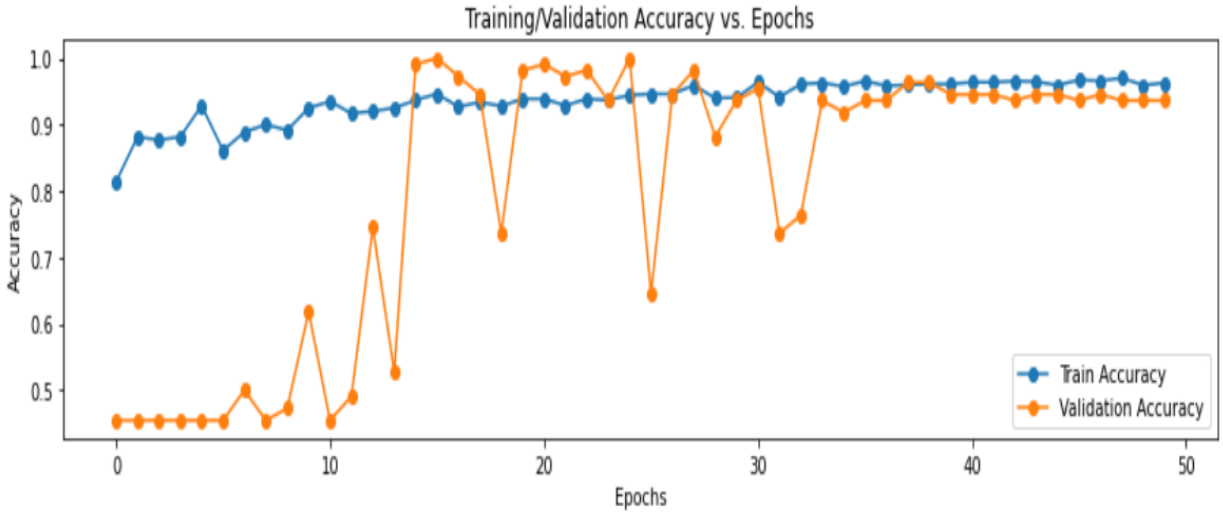

Figure 5.2 : Training and validation accuracy of the proposed CNN model.

In Figure 5.2 is shown the accuracy curve of our model. The curve shows a steady increase over time for both the training and validation datasets. Therefore, our model performs well on new and unseen data.

5.6.2 Training and validation loss

The loss curve is a plot of the loss function of the model on the training and validation datasets against the number of epochs during training. The loss function is a measure of how well the model can predict the correct class labels for the images in the dataset. During training, the goal is to minimize the loss function. The purpose of the loss curve is to help identify whether the model is converging to a good solution. If the loss curve shows a steady decrease over time for both the training and validation datasets, this suggests that the model is learning to make more accurate predictions. However, if the loss curve shows a lot of fluctuation or does not show a clear downward trend, this suggests that the model may not be learning effectively and may require adjustments to the model architecture or training process.

Loss curve of our model,

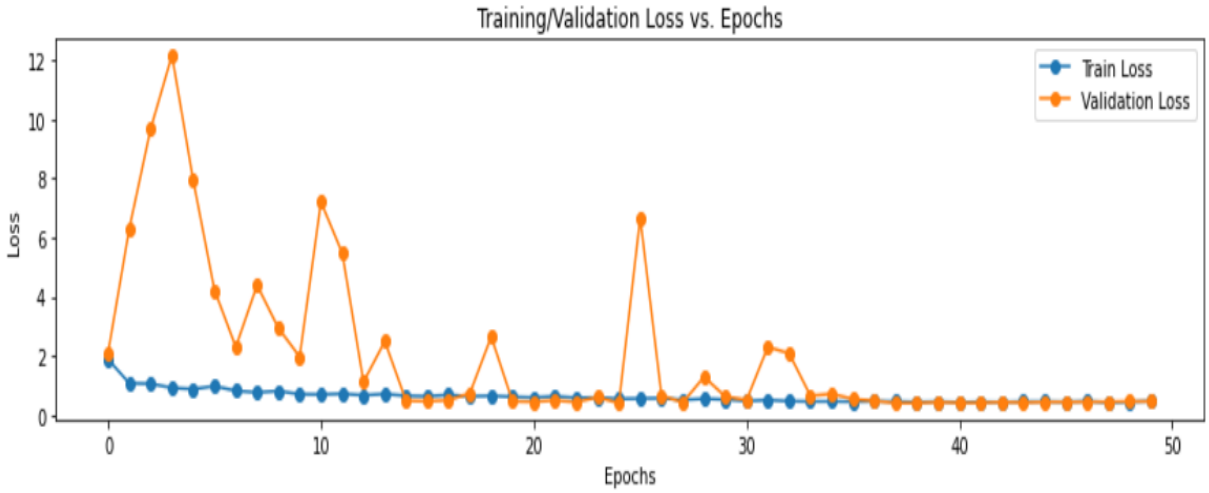

Figure 5.3 : Training and validation loss of the proposed CNN model.

In Figure 5.3 is shown the loss function of our model. The curve shows a steady decrease over time and a clear downward trend for both the training and validation datasets which suggests that our model can predict more accurately.

5.6.3 Learning rate

We will also see the learning rate of our model. The learning rate is typically set as a hyperparameter before training begins, and it can be adjusted during training based on the performance of the model on a validation set. It determines how quickly or slowly the

network adapts to the training data. A high learning rate means that the weights will be updated by a large amount during each training iteration, which can lead to overshooting and unstable behavior. On the other hand, a low learning rate means that the weights will be updated slowly, and it may take longer for the network to converge to an optimal solution.

Learning rate of our model,

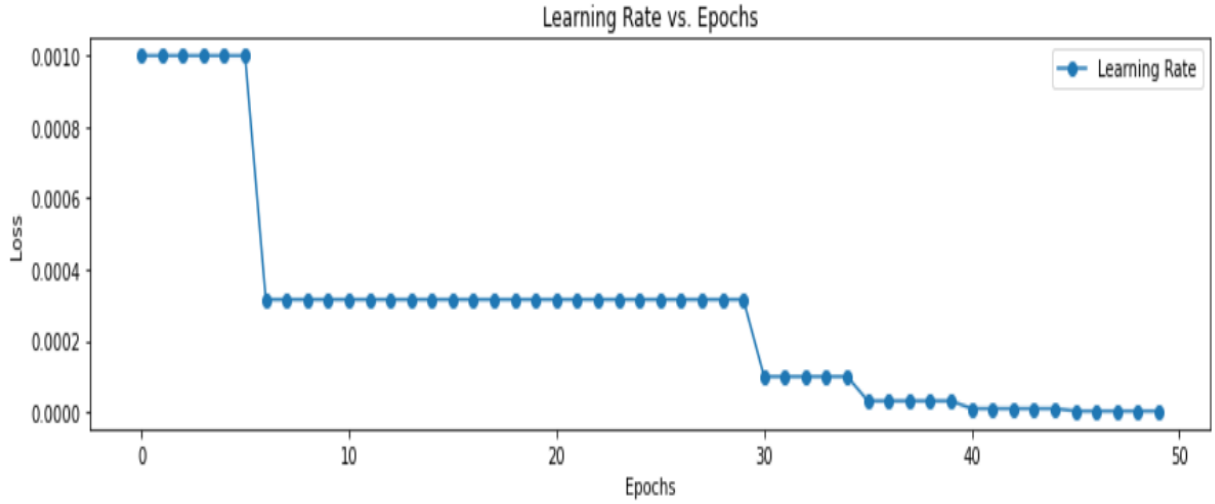

Figure 5.4 : Learning rate of the proposed CNN model.

In Figure 5.4 is shown the learning rate of our model where we have used Adam optimizer. The optimizer updated the model's parameters (weights and biases) efficiently during the training process and minimizes the loss function. The figure shows a steady decrease of loss over time which finally have become close to zero.

5.7 Summary

There are various results have shown in this chapter. We have seen the performance of our model based on the several metrices such as confusion matrix, classification report also validation and loss curve. Our CNN model achieved 94% accuracy.

CHAPTER 6

Conclusions and Future Work

6.1 Conclusions

In this paper, we discussed a multiclass image classification problem. The discussion is covered over the classification of two potato leaf diseases. To detect the leaf disease from healthy we built up a methodology with CNN algorithm and we found that CNN is the best way to perform this type of classification object. Our model has gained a total of 94% accuracy over 1350 images. We think this type of project will play a vital role in our agriculture sector. Most of the farmers of rural area in Bangladesh are not literate and they don't know about the disease properly. We think that this work can change the situation of the potato grower in Bangladesh. It is concluded that the proposed method recognizes the leaf diseases of potato effectively in advance.

6.2 Future Recommendation

We will enrich our dataset by experimenting with different images to achieve heigh-performance of the model. And we will apply different models on the dataset and compare the performance of the model with the previous models to check the efficiency and achieve high accuracy. Also, an android and a web application will build up where include different types of crops and vegetables to detect their various diseases. And we will create a system where the farmers can easily get instant service and advice on their problem by detecting the disease. The whole project can be put up on the internet and users can simply sit at home and use the system to detect the disease and spray the required disinfectant. The interface will be connected to the internet and then to the database. Also, we will try that our proposed methodology will be applied in various applications where multiclass image classification especially leaf plant recognition is involved.

6.3 Limitations

Our model will perform on multiclass image classification which developed by CNN algorithm. The performance of CNN algorithms is highly dependent on the quantity and quality of the dataset used for training. If the dataset is limited, biased, or not representative

of the overall population of the target disease, the CNN model may not be accurate enough to detect the disease. Although CNN models can learn to classify leaf diseases based on images, they may not be able to generalize to unseen conditions, such as varying lighting conditions, camera angles, and environmental factors. This can lead to false positives or negatives and inaccurate disease detection. It is computationally expensive and requires significant computational resources, such as GPUs, to achieve good performance which can be a limitation for some applications. Sometimes it is difficult to understand for CNN algorithm how the model is making decisions and what features are being used to detect the disease due to the complex architecture and large number of parameters.

REFERENCES

[1] A. Krizhevsky, I. Sutskever and G. E. Hinton, "Imagenet classification with deep convolutional neural networks", *Communications of the ACM*, vol. 60, no. 6, pp. 84-90, 2017.

[2] Y. Toda and F. Okura, "How convolutional neural networks diagnose plant disease", *Plant Phenomics*, vol. 2019, article 9237136, pp. 1–14, 2019.

[3] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient based learning applied to document recognition", *Proceedings of the IEEE*, vol. 86, no. 11, p. 2278–2324, 1998.

[4] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition", 2014. [Online]. Available: http://arxiv.org/abs/1409.1556.

[5] C. Szegedy, W. Liu, Y. Jia et al, "Going deeper with convolutions", *in 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Boston, 2015.

[6] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition", *in 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778, Las Vegas, NV, USA, June 2016.

[7] N. O'Mahony, S. Campbell, A. Carvalho et al., "Deep learning vs. traditional computer vision", in Advances in Computer Vision. CVC 2019. *Advances in Intelligent Systems and Computing*, vol 943, pp. 128–144, Springer, Cham, 2019.

[8] A. Vibhute and S. K. Bodhe, "Applications of image processing in agriculture: a survey", *International Journal of Computer Applications*, vol. 52, no. 2, pp. 34–40, 2012.

[9] J. G. A. Barbedo, "Plant disease identification from individual lesions and spots using deep learning", *Biosystems Engineering*, vol. 180, pp. 96–107, 2019.

[10] S. Parkes and S. Teltscher, "I.C.T Facts and Figures-the World", in 2015, *The International Telecommunication Union (ITU)*, Geneva, 2015.

[11] S. P. Mohanty, D. P. Hughes, and M. Salathé, "Using deep learning for image-based plant disease detection", *Frontiers in Plant Science*, vol. 7, article 1419, 2016.

[12] K. P. Ferentinos, "Deep learning models for plant disease detection and diagnosis", *Computers and Electronics in Agriculture*, vol. 145, p. 311–318, 2018.

[13] M. Arsenovic, M. Karanovic, S. Sladojevic, A. Anderla, and D. Stefanovic, "Solving current limitations of deep learning-based approaches for plant disease detection", *Symmetry*, vol. 11, no. 7, p. 939, 2019.

[14] S. Biswas, B. Jagyasi, B. P. Singh, and M. Lal, "Severity identification of potato late blight disease from crop images captured under uncontrolled environment", *in 2014 IEEE Canada International Humanitarian Technology Conference - (IHTC)*, pp. 1–5, Montreal, QC, Canada, June 2014

[15] Parul Sharma, Yash Paul Singh Berwal, Wiqas Ghai "Krishi Mitr (Farmer's Friend): Using Machine Learning to Identify Diseases in Plants", *In 2018 IEEE International Conference on Internet of Things*

and Intelligence System (IoTaIS), pp 29-34. IEEE 2018.

[16] R. K. Arora, S. Sharma, and B. P. Singh, "Late blight disease of potato and its management", *Potato Journal*, vol. 41, no. 1, pp. 16–40, 2014.

[17] A. J. Haverkort, P. C. Struik, R. G. F. Visser, and E. J. P. R. Jacobsen, "Applied biotechnology to combat late blight in potato caused by Phytophthora infestans", *Potato Research*, vol. 52, no. 3, pp. 249–264, 2009.

[18] Wikipedia, "Literature review", 21 April 2023. [Online].
Available: https://en.wikipedia.org/wiki/Literature_review. [Accessed 15 May 2023].

[19] "Artificial neural network", 12 May 2023. [Online].
Available: https://en.wikipedia.org/wiki/Artificial_neural_network. [Accessed 15 May 2023].

[20] G. K. Majji, V Applalanaidu, "A Review of Machine Learning Approaches in Plant Leaf Disease Detection and Classification", *In 2021 IEEE Proceedings of the Third International Conference on Intelligent Communication Technologies and Virtual Mobile Networks (ICICV 2021)*, pp. 716-724, 2021.

[21] Islam, Monzurul, Anh Dinh, Khan Wahid, and Pankaj Bhowmik, "Detection of potato diseases using image segmentation and multiclass support vector machines", *2017 IEEE 30th Canadian conference on electrical and computer engineering (CCECE)*, pp. 1-4, 2017.

[22] Md. Khalid Rayhan Asif, Md. Asfaqur Rahman, Most. Hasna Hena "CNN based Disease Detection Approach on Potato Leaves", *in 2020 Proceedings of the Third International Conference on Intelligent Sustainable Systems [ICISS 2020]*, pp 428-432 IEEE,2020.

[23] Mustafa Merchant, Vishwajeet Paradkar, Meghna Khanna, Soham Gokhale. "Mango Leaf Deficiency Detection Using Digital Image Processing and Machine Learning", *2018 3rd International Conference for Convergence in Technology (I2CT)*, pp 1-3. IEEE 2018.

[24] Melike Sardogan, Adem Tuncer, Yunus Ozen. "Plant Leaf Disease Detection and Classification based on CNN with LVQ Algorithm", *2018 3rd International Conference in Computer Science and Technology*. pp 382-385. IEEE 2018.

[25] Ungsumalee Suttapakti, Aekapop Bunpeng. "Potato Leaf Disease Classification Based on Distinct Color and Texture Feature Extraction", *2019 19th International Symposium on Communications and Information Technologies (ISCIT)*. pp 82-85. IEEE 2019.

[26] Anushka Bangal, Dhiraj Pagar, Hemant Patil, Neha Pande, "Potato Leaf Disease Detection and Classification Using CNN", *International Journal of Research Publication and Reviews*, vol. 3, no. 5, pp. 1510-1515, 2022.

APPENDIX A

Importing the Required Libraries:

```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
import os
os.environ["TF_CPP_MIN_LOG_LEVEL"] = "2"
import warnings
warnings.filterwarnings('ignore')
from sklearn.metrics import confusion_matrix, classification_report, accuracy_score
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Activation, BatchNormalization, Conv2D, Dense, Dropout, Flatten, MaxPooling2D
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.optimizers.legacy import Adam
from tensorflow.keras.losses import CategoricalCrossentropy
from tensorflow.keras.regularizers import l2
from tensorflow.keras.callbacks import ReduceLROnPlateau, EarlyStopping
from tensorflow.keras import layers,models,datasets
```

Data Pre-processing:

```
IMG_WIDTH = 256
IMG_HEIGHT = 256
BATCH_SIZE =32
EPOCHS = 50
train_dataset_path = 'C:/Users/HP/Desktop/PYTHON/THESIS/Book/Training'
train_datagen = ImageDataGenerator(rescale=1.0/255,
                                   rotation_range=30,
                                   horizontal_flip=True,  # random_flip can also be used
                                   zoom_range=0.2,
                                   width_shift_range=0.2,
                                   height_shift_range=0.2,
                                   fill_mode='nearest')
train_generator = train_datagen.flow_from_directory(train_dataset_path,
                                                    target_size=(IMG_WIDTH, IMG_HEIGHT),
                                                    batch_size=BATCH_SIZE,
                                                    class_mode='categorical',
                                                    shuffle=True)

validation_dataset_path = 'C:/Users/HP/Desktop/PYTHON/THESIS/Book/Validation'
validation_datagen = ImageDataGenerator(rescale=1.0/255)
validation_generator = validation_datagen.flow_from_directory(validation_dataset_path,
                                                              target_size=(IMG_WIDTH, IMG_HEIGHT),
                                                              batch_size=BATCH_SIZE,
                                                              class_mode='categorical',
                                                              shuffle=True)
```

```
test_dataset = 'C:/Users/HP/Desktop/PYTHON/THESIS/Book/Testing'
test_datagen = ImageDataGenerator(rescale=1.0/255)
test_generator = test_datagen.flow_from_directory(test_dataset,
                                                  shuffle=False,
                                                  batch_size=BATCH_SIZE,
                                                  target_size = (IMG_WIDTH, IMG_HEIGHT),
                                                  class_mode='categorical')
```

Class Names:

```
class_names = list(train_generator.class_indices.keys())
print(class_names)
```

Class Labels:

```
labels = {value: key for key, value in train_generator.class_indices.items()}
print("Label Mappings for classes present in the training and validation datasets\n")
for key, value in labels.items():
    print(f"{key} : {value}")
```

Sub Plotting of Training Images:

```
fig, ax = plt.subplots(nrows=2, ncols=5, figsize=(15, 12))
idx = 0
for i in range(2):
    for j in range(5):
        label = labels[np.argmax(train_generator[0][1][idx])]
        ax[i, j].set_title(f"{label}")
        ax[i, j].imshow(train_generator[0][0][idx][:, :, :])
        ax[i, j].axis("off")
        idx += 1
plt.tight_layout()
```

```
plt.suptitle("Sample Training Images", fontsize=21)
plt.show()
```

Proposed CNN Model:

```
def create_model():
    model = Sequential([Conv2D(filters=64, kernel_size=(5, 5),
    padding='same',      input_shape=(IMG_WIDTH,     IMG_HEIGHT, 3)),
    Activation('relu'),
    MaxPooling2D(pool_size=(2, 2)),
    BatchNormalization(),

    Conv2D(filters=32, kernel_size=(3, 3), padding='same', kernel_regularizer=l2(0.01)),
    Activation('relu'),
    MaxPooling2D(pool_size=(2, 2)),
    BatchNormalization(),

    Conv2D(filters=32, kernel_size=(3, 3), padding='same', kernel_regularizer=l2(0.01)),
    Activation('relu'),
    MaxPooling2D(pool_size=(2, 2)),
    BatchNormalization(),

    Flatten(),
    Dense(units=70, activation='relu'),
    Dropout(0.5),
    Dense(units=3, activation='softmax')
  ])
return model
cnn_model = create_model()
reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=np.sqrt(0.1), patience=5)
```

Model Compilation:

```
cnn_model.compile(optimizer='adam',
                  loss=CategoricalCrossentropy(),
                  metrics=['accuracy'])
```

Training and Validation History:

```
history = cnn_model.fit(train_generator, epochs=EPOCHS,
validation_data=validation_generator,
                    verbose=2,
                    callbacks=[reduce_lr])
```

Accuracy and Loss Curve:

```
train_accuracy = history.history['accuracy']
val_accuracy = history.history['val_accuracy']
train_loss = history.history['loss']
val_loss = history.history['val_loss']
learning_rate = history.history['lr']
fig, ax = plt.subplots(nrows=3, ncols=1, figsize=(12, 10))
ax[0].set_title('Training/Validation Accuracy vs. Epochs')
ax[0].plot(train_accuracy, 'o-', label='Train Accuracy')
ax[0].plot(val_accuracy, 'o-', label='Validation Accuracy')
ax[0].set_xlabel('Epochs')
ax[0].set_ylabel('Accuracy')
ax[0].legend(loc='best')
ax[1].set_title('Training/Validation Loss vs. Epochs')
ax[1].plot(train_loss, 'o-', label='Train Loss')
ax[1].plot(val_loss, 'o-', label='Validation Loss')
ax[1].set_xlabel('Epochs')
```

```
ax[1].set_ylabel('Loss')
ax[1].legend(loc='best')
ax[2].set_title('Learning Rate vs. Epochs')
ax[2].plot(learning_rate, 'o-', label='Learning Rate')
ax[2].set_xlabel('Epochs')
ax[2].set_ylabel('Loss')
ax[2].legend(loc='best')
plt.tight_layout()
plt.show()
```

Predictions:

```
predictions = cnn_model.predict(test_generator)
Y_pred = np.argmax(predictions, axis=1)
y_true = test_generator.classes
fig, ax = plt.subplots(nrows=2, ncols=5, figsize=(12, 10))
idx = 0
for i in range(2):
    for j in range(5):
        predicted_label = labels[np.argmax(predictions[idx])]
        ax[i, j].set_title(f"{predicted_label}")
        ax[i, j].imshow(test_generator[0][0][idx])
        ax[i, j].axis("off")
        idx += 1
plt.tight_layout()
plt.suptitle("Test Dataset Predictions", fontsize=20)
plt.show()
```

Model Evaluation:

```
test_loss, test_accuracy = cnn_model.evaluate(test_generator, batch_size=BATCH_SIZE)
print(f"Test Loss:     {test_loss}")
print(f"Test Accuracy: {test_accuracy}")
```

Confusion Matrix:

```
c_matrix = confusion_matrix(test_generator.classes, predictions.argmax(axis=1))
print(c_matrix)
sns.heatmap(c_matrix, annot=True,cmap="OrRd",fmt=".1f")
plt.show()
cf_mtx = confusion_matrix(y_true, Y_pred)
group_counts = ["{0:0.0f}".format(value) for value in cf_mtx.flatten()]
group_percentages = ["{0:.2%}".format(value) for value in cf_mtx.flatten()/np.sum(cf_mtx)]
box_labels = [f"{v1}\n({v2})" for v1, v2 in zip(group_counts, group_percentages)]
box_labels = np.asarray(box_labels).reshape(3,3)
plt.figure(figsize = (10, 8))
sns.heatmap(cf_mtx, xticklabels=labels.values(), yticklabels=labels.values(),
        cmap="YlGnBu", fmt="", annot=box_labels)
plt.xlabel('Predicted Classes')
plt.ylabel('True Classes')
plt.show()
```

Classification Report:

```
print(classification_report(y_true, Y_pred, target_names=labels.values()))
```

Prediction Function:

```
test_ds = tf.keras.preprocessing.image_dataset_from_directory(
    "Testing",
    shuffle=True,
```

```
    image_size=(IMG_WIDTH, IMG_HEIGHT),
    batch_size=BATCH_SIZE
)
Predictions = cnn_model.predict(test_ds)
y_pred = np.argmax(Predictions, axis=1)
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
```

Sub plotting:

```
plt.figure(figsize=(15,15))
for images,labels in test_ds.take(1):
    for i in range(9):
        plt.imshow(images[i].numpy().astype("uint8"))
        All=plt.subplot(3,3,i+1)
        predicted_class,confidence=predict(create_model(),images[i].numpy())
        actual_class=class_names[labels[i]]
        plt.title(f"Actual : {actual_class},\nPredicted:{predicted_class}.\nConfidence :
          {confidence}%")
        plt.axis("off")
```

BIOGRAPHY

of

Md. Firozzaman

Md. Firozzaman was born in 1996 in Thakurgaon, Bangladesh. He is the only son and the third of four children of his parents. His father Md. Abul Kasem, is a homoeopathic doctor, and mother Mst. Firoza Yeasmin is a housewife. In 2014, he passed S.S.C in Science Group from Pirganj Pilot Heigh School and in 2016, he completed H.S.C in Science Group from Hajipur College, Pirganj, Thakurgaon. In 2023, he graduated from the Bangladesh Army University of Science and Technology (BAUST), Saidpur, Nilphamari, Bangladesh, with a bachelor's degree in Computer Science and Engineering.

He was involved in various extracurricular activities in BAUST such as BAUST Programming Club and BAUST Sporting Club. He is passionate about sports; Cricket, Football, Badminton and Swimming are among them, and he also likes to cook. He is highly interested in competitive programming and problem solving. He was participating in various coding contest such as International Collegiate Programming Contest, Samsung SRBD Code Contest and Code Samurai 2022. As well as, he has been participating in various programming competitions regularly in Codeforces online judge.

He has research interests in Computer Vision, Data Science, NLP, Graph theory and Bioinformatics. Also being his subject of interest are Religious Studies, Science, Literature, History and Philosophy.

Md. Firozzaman
01722646579
smdfirozzaman@gmail.com
Pirganj, Thakurgaon, Bangladesh