

A  
Major Project Report  
on  
**“Tomato Leaf Disease Detection Using  
CNN”**

Presented by

Shruti Gorgile  
Komal Dake  
Ambika Gangawar  
Shrutika Deshmukh

Department of Computer Science & Engineering

Guided By

**Mr. M.N.Bhandare**

(Department of Computer Science & Engineering)

Submitted to



**MAHATMA GANDHI MISSION'S COLLEGE OF ENGINEERING  
UNDER**

**Dr. Babasaheb Ambedkar Technological University, Lonere.**

**Academic Year 2024-2025**

## **CERTIFICATE**

This is to certify that the project entitled  
**“Tomato Leaf Disease Detection Using  
CNN”**

Presented by

Shruti Gorgile

Komal Dake

Ambika Gangawar

Shrutika Deshmukh

in satisfactory manner as a partial  
fulfilment of Final Year Engineering

To

**MGM’s College of Engineering, Nanded**

Under

**Dr. Babasaheb Ambedkar Technological University, Lonere**

Has been carried out under my guidance,

**Mr. M. N.Bhandare**

Major Project Guide

**Dr.A. M. Rajurkar**

HOD

MGM’s College of Engineering, Nanded

**Dr. Lathkar G. S.**

Director

MGM’s College of Engineering, Nanded



## ACKNOWLEDGEMENT

We are deeply grateful to our major project guide **Mr.M.N.Bhandare** for his able guidance throughout this work. It has been an altogether different experience to work with him and we would like to thanks him for his help, suggestions and numerous discussions.

We gladly take this opportunity to thanks **Dr. Rajurkar A.M.** (Head of Computer Science & Engineering, MGM's College of Engineering, Nanded).

We are heartily thankful to **Dr. Lathkar G.S.** (Director, MGM's College of Engineering, Nanded) for providing facility during progress of major project, also for her kindly help, guidance and inspiration.

Last but not least we are also thankful to all those who help directly or indirectly to develop this major project and complete it successfully.

With Deep Reverence,

32\_ Shruti Gorgile  
54\_Komal Dake  
58\_Ambika Gangawar  
60\_Shrutika Deshmukh

# ABSTRACT

## Tomato Leaf Disease Prediction Using Machine Learning Techniques

Tomato leaf diseases significantly impact the yield and quality of tomato crops, posing a major challenge for farmers and agricultural stakeholders. Early and accurate detection of these diseases is crucial for implementing timely intervention strategies to minimize crop loss and ensure food security. This study focuses on the development of a machine learning-based model to predict and classify tomato leaf diseases from image data. Leveraging a dataset of labeled tomato leaf images, we preprocess the data through techniques such as image augmentation and normalization to enhance model performance. Various machine learning algorithms, including Convolutional Neural Networks (CNNs), are employed to extract features and identify patterns associated with different tomato leaf diseases. The model's performance is evaluated using metrics such as accuracy, precision, recall, and F1-score. The results demonstrate that the proposed model achieves high accuracy in detecting and classifying common tomato leaf diseases, such as early blight, late blight, and leaf mold. The study underscores the potential of machine learning in automating disease diagnosis in agriculture, providing a cost-effective and scalable solution for farmers. The implementation of this model could lead to more informed decision-making, optimized pesticide usage, and improved crop management practices, ultimately contributing to enhanced agricultural productivity and sustainability.

**Technology Used** : Python, Machine Learning Algorithm, Image Processing, Flask, HTML,CSS, JAVASCRIPT

32\_Shruti Gorgile

54\_Komal Dake

58\_Ambika Gangawar

60\_Shrutika Deshmukh

## INTRODUCTION

### 1.1 Background

Even though tomatoes are one of the most significant crops in the world, diseases like grey leaf spot can harm the leaves and ultimately reduce fruit production. This disease develops into four stages: contact, invasion, latency, and onset. Early detection ensures proper crop management and prevents outbreaks. Traditional methods of detecting this disease are expensive and time-consuming, especially for larger farms. Image processing provides a more economical and effective method by automating the identification of diseases. In the study, 11,000  $256 \times 256$  pixel images of tomato leaves with ten distinct disease kinds are employed. The dataset consists of 1000 testing photos, 3,000 validation images, and 7,000 training images. The validation set is also used to test and evaluate model performance. The 7,000-image training dataset is used to train deep learning algorithms, which allow them to recognize traits associated with various diseases. Deep learning particularly convolutional neural networks (CNNs), which excel at picture classification may enable the prompt and precise diagnosis of plant diseases. These scalable techniques increase detection accuracy, making them crucial for modern farming.

### 1.2 Problem Description

Manual methods for detecting plant diseases are ineffective and time-consuming, particularly for large farms. While CNN-based tomato leaf disease diagnosis often requires a team of experts, traditional disease identification techniques, such eye inspection, are prone to errors. Furthermore, early disease diagnosis which is essential for managing and avoiding plant diseases cannot be provided by current approaches. Thus, a technique for reliably, effectively, and automatically identifying tomato plant diseases that can provide successful prevention and early identification is needed.

### 1.3 Objectives

Even though tomatoes are one of the most important crops in the world, diseases like grey leaf spot can harm the leaves and reduce yield. Early detection is essential for both epidemic prevention and efficient crop management. Traditional methods of detection

are expensive and time-consuming. Image processing provides a faster and more cost-effective solution since it automates disease identification. The 11,000 tomato leaf photos used in this study were scaled to 256 by 256 pixels and divided into 7,000 training, 3,000 validation, and 1000 testing images. The training dataset enables the deep learning models to efficiently learn the features of illnesses.

One of the study's goals is to create a CNN model for classifying common tomato leaf diseases, such as leaf mold, bacterial spots, early and late blight, and others.

1. Analyze the CNN model's long-term performance.
2. To promote sustainable agriculture, provide an automated, cost-effective early disease diagnosis approach.

Because deep learning and CNNs in particular are so good at picture classification, they are crucial to modern agriculture and enable rapid and accurate disease diagnosis.

## **1.4 Motivation**

Recognizing and identifying leaf disease is the way to prevent huge farms from losing money on crop disease detection and productivity. It also helps agricultural institutes and biological research. CNN-Based Tomato Leaf Disease Detection

## **1.5 Project Organization**

This research project's remaining sections are arranged as follows:

The literature survey, which outlines the current methods for categorizing plant leaf diseases in photos, is explained in Chapter 2: Literature Review. In Chapter 3: Methodology, the dataset used in this CNN study on tomato leaf disease detection is fully explained. Data collection, preprocessing, statistics, and dividing the dataset into train, valid, and test datasets are all covered in this. Chapter 4 describes the Convolutional Neural Network (CNN) model design and training process used for our image classification task. Chapter 5: The algorithm process and related Python code and visualization of the training outcomes and prediction analysis obtained using the proposed CNN-based approach.

## **LITERATURE REVIEW**

The subject of plant disease identification has been significantly impacted by the exponential expansion of artificial intelligence in recent years. Computer vision-based techniques have historically concentrated on differentiating characteristics like form, color, and texture from disease spots on plant leaves. Since these methods mostly rely on manual analysis and specialized knowledge, which can be time-consuming and prone to errors, they are less successful. A trend toward autonomous, more precise detection systems has been sparked by the advancement of deep learning, and CNNs in particular. Many studies have been conducted on CNNs with the goal of increasing the classification accuracy of plant diseases, especially in tomatoes.

### **2.1 Convolutional Neural Networks: An Overview and Uses**

CNNs are thoroughly examined by Mane and Kulkarni 2023[1], the authors of a Survey of Convolutional Neural Networks and Its Major Applications, who also point out how CNNs' hierarchical structure can be utilized to automatically extract attributes from data. To put it another way, CNNs excel at machine vision, where traits like pattern complexity are crucial for differentiating between a healthy and diseased leaf.

### **2.2 Deep Learning-Based Tomato Disease Classification**

The study by Aravind KR, Raja P, and Anirudh R (2023)[2] focused on diseases of tomato crops, including Early Blight, Late Blight, and Leaf Mold. Researchers have applied pre-trained deep learning models to images of tomato leaves. Despite being trained using VGG16, CNNs' accuracy in disease classification improves, as seen by their 98.67% success rate. Their findings also demonstrated the importance of diverse datasets, and they even suggested developing a mobile application to assist farmers in identifying illnesses.



## **2.3 Using Attention-Embedded Residual CNN to Identify Tomato Disease**

Retal.'s paper, Attention Embedded Residual CNN for Disease Detection in Tomato Leaves, 2020[3], suggested a CNN architecture that is improved with residual blocks and attention modules to enable the model to effectively learn input-output mappings. When it came to tomato leaf disease classification, this model outperformed more conventional CNN models like VGG16 and Inception-v3, with an accuracy of 98.3%. The model's attention modules improved disease detection performance by concentrating on the most pertinent aspects in the pictures.

## **2.4 Deep Learning for Apple Leaf Disease Recognition**

Zhong Yong and Zhao Ming (2020)[4] used CNN and deep learning to identify apple leaf diseases. They obtained astonishingly effective sickness detection by training their algorithm on a large collection of apple leaf photos. Their study offers compelling proof that real-time deep learning algorithms can help farmers identify illnesses in fields, nurseries, or orchards early.

## **2.5 AI Detection of Pests and Diseases in Bananas**

Using a set of photos of banana plants, Selvaraj et al. (2019)[5] investigated AI-powered techniques for identifying pests and illnesses in bananas. They showed how real-time pest and disease detection using a smartphone app may transform farming. This strategy demonstrates how machine learning and computer vision techniques can be used in agriculture for purposes other than tomatoes.

## **2.6 Rich feature hierarchies for segmenting and detecting objects**

Girshick et al. (2014)[6] developed a deep learning-based approach called R-CNN, which blends CNNs with region-based processing for object detection. The authors' work improved computer vision by increasing object recognition accuracy, which has been used to plant disease identification tasks, even though it did not specifically address plant disease detection.

## **2.7 Deep Learning-Based Tomato Disease Detection**

Based on CNN-based object detectors, Fuentes et al. (2023)[7] suggested using deep learning to detect pests and diseases in tomato plants. With class annotations and data extension, they decreased false positives and increased model accuracy. In a variety of conditions, their computer accurately detected nine distinct pests and illnesses on tomato leaves.

## **2.8 CNNs for Crop Disease Detection via Mobile**

"DeepPlantPathologist," a CNN model created by the authors of A Deep Convolutional Neural Network for Mobile Capture Device-Based Crop Disease Classification (Picon et al., 2023)[8], is intended to categorize crop diseases using photos taken with mobile devices while in the field. Under real-world circumstances, our model exhibits good accuracy and has a lot of promise for farmers who need a trustworthy tool for diagnosing illnesses in real time.

## METHODOLOGY

The dataset utilized in this study on tomato leaf disease detection using CNN is thoroughly described in this chapter, along with its collection, preprocessing, statistics, and classification into train, valid, and test datasets.

### 3.1 Flow chart for methodology

The methodology flow chart, which explains how to identify tomato leaf diseases, is displayed in the accompanying graphic.

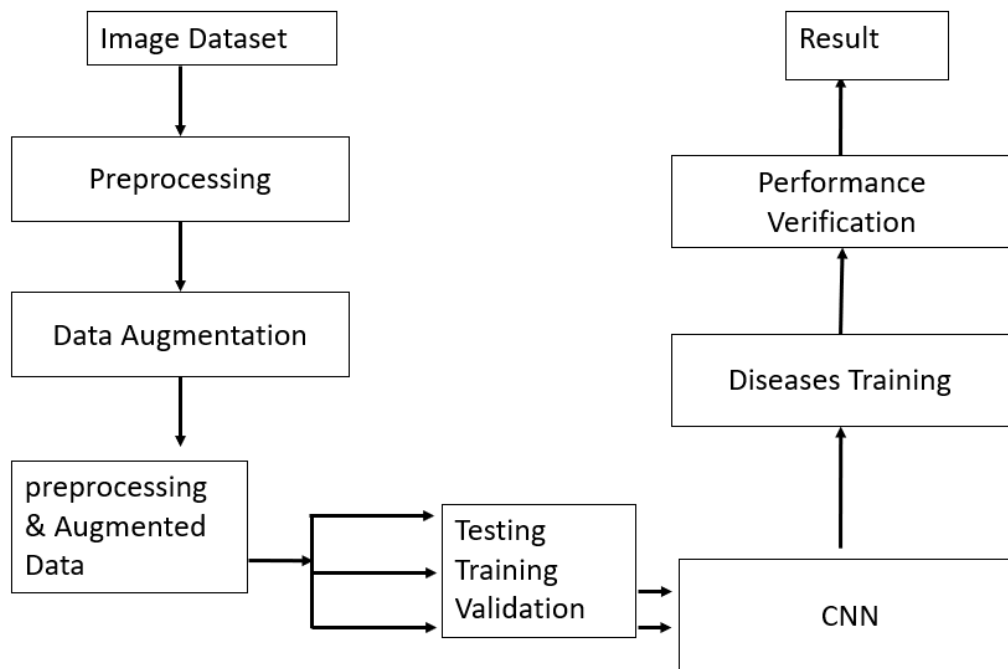
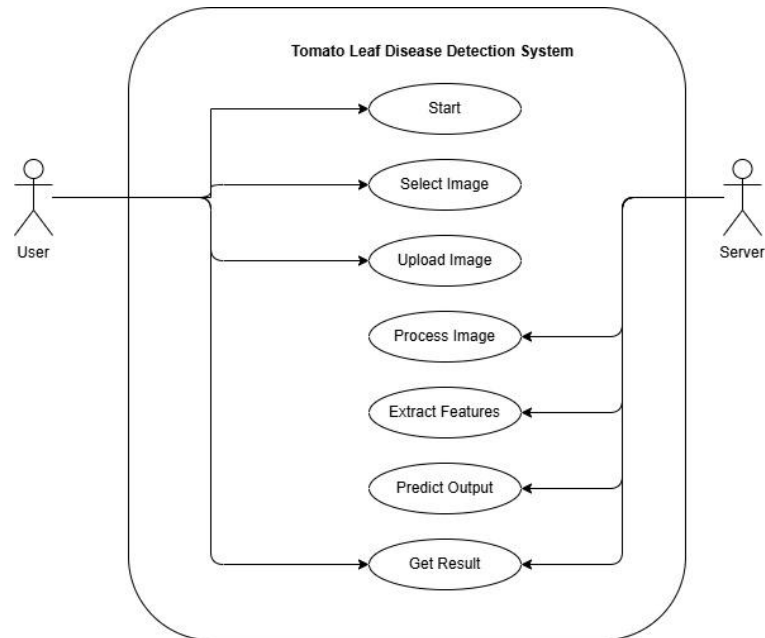


Figure 3.1 Methodology Flow Chart

### 3.2 Use Case Diagram

The Tomato Leaf Disease Detection System's Use Case Diagram shows interactions between the User and the Server. The user initially selects and uploads a photo of a tomato leaf via the system interface. The server uses a CNN model to evaluate whether a disease is present after processing the submitted image and extracting relevant information. When the prediction is finished, the user gets the answer. The diagram clearly shows the sequential steps of the process, such as image selection, processing,

feature extraction, output prediction, result retrieval, and process starting, with a clear division between user input and server-based processing.

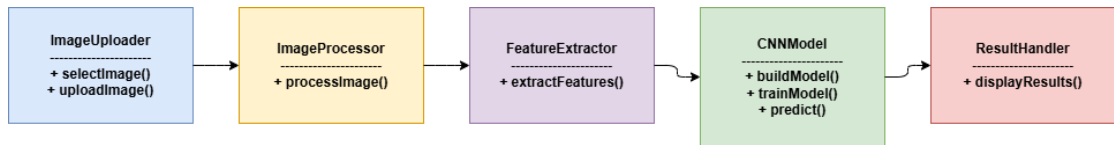


**Figure 3.2 Use Case Diagram**

### 3.3 Class Diagram

This Class Diagram shows the key components and their functions of the Tomato Leaf Disease Detection System. These classes are divided into five categories:

1. It provides two methods for selecting and uploading images: `selectImage()` and `uploadImage()`.
2. Its `processImage()` function processes the given images.
3. This class uses the function `extractFeatures()` to extract relevant features from the processed images.
4. **CNNModel**: Creates, trains, and forecasts the outputs of the diseases using `buildModel()`, `trainModel()`, and `predict()`.
5. **ResultHandler**: Displays the result of the prediction using `displayResults()`.



**Figure 3.3 Class Diagram**

### 3.4 Gathering Datasets

This project's dataset was obtained via Kaggle. 11,000 tomato leaf photos in eleven classes make up the dataset:

1. Bacterial Spot on Tomato
2. Early blight of tomatoes
3. Late blight in tomatoes
4. Leaf mold for tomatoes
5. Leaf Spot for Tomato Septoria
6. Two-spotted spider mites in tomatoes
7. Target\_Spot\_Tomato
8. Yellow Leaf Curl Virus in Tomato
9. The virus known as tomato mosaic
10. Healthy tomato

The photographs were captured in various lighting circumstances, seasons, and locales.

### 3.5 Dataset Statistics

Ten classes each representing a tomato leaf disease and a healthy class are created from the 11,000 images in the collection. Table 3.1 displays the class distribution of the dataset.

Sr.No	Type of Tomato Leaf Disease	No. Of Images
1	Tomato_Bacterial_spot	1100
2	Tomato_Early_blight	1100
3	Tomato_Late_blight	1100
4	Tomato_Leaf_Mold	1100
5	Tomato_Septoria_leaf_spot	1100
6	Tomato_Spider_mites_Two_spotted_spider_mite	1100

7	Tomato__Target_Spot	1100
8	Tomato_YellowLeaf__Curl_Virus	1100
9	Tomato__Tomato_mosaic_virus	1100
10	Tomato_healthy.	1100

**Table 3.1 Class distribution of the dataset.**

### 3.6 Preprocessing

The dataset was preprocessed using a range of methods, including data augmentation, standardization, and scaling, prior to CNN model training. A common method in image processing, particularly computer vision, is data augmentation, which increases the amount and variety of training data by applying realistic yet random modifications.

To expand the dataset's size and diversity, data augmentation techniques like rotation, flipping (horizontal and vertical), and random cropping were used.

During training, pixel values between 0 and 255 were normalized to enhance convergence. Additionally, scaling was employed to standardize the image size to 256x256 pixels in order to reduce the computational cost of training the mode. Furthermore, because the batch size is 32, the photos were provided in batches.

### 3.7 Dataset Split

The training dataset, validation dataset, and test dataset were the three subsets into which the dataset was separated. The model was trained using the training dataset, the hyperparameters were adjusted using the validation dataset, and the model's performance on unidentified data was assessed using the test dataset. Random subsets were created from 80% of the training dataset's photographs, 10% of the validation dataset's images, and 10% of the test dataset's images. Table 3.2 displays the lengths of the test, validation, and train datasets.

Train dataset length	7000
Valid dataset length	3000
Test dataset length	1000
Total dataset length	11000

**Table 3.2 shows the train, valid, and test dataset lengths.**

The dataset split ensures that the model is trained on a sufficiently large dataset, allowing for a fair evaluation of the model's performance on unseen data. The system's methodology was described in this chapter. The next part, part 4, presents the CNN model design and training process. The dataset split presented in this chapter is used to train the model and evaluate its performance.

## CNN MODEL ARCHITECTURE AND TRAINING PROCESS

The CNN model architecture for our image classification challenge and the model training process are covered in this chapter. Several convolutional and pooling layers are used in the design, followed by fully linked layers and softmax output layers. During the training phase, all of the model's parameters are initialized, the loss is determined, the optimization strategy is selected, and the model's parameters are iteratively updated using gradient descent and backpropagation.

### 4.1 CNN Model Architecture

Convolutional neural networks, or CNNs, are artificial neural networks that are primarily used for processing, recognizing, and analyzing images and videos. It is made to automatically extract important elements from the raw pixel data of an image to recognize faces, objects, formations, and patterns. The structure and function of the visual cortex of the brain serve as an inspiration for CNNs. Each of the network's many interconnected layers has several neurons that use the input data to carry out simple computations. The layers are often arranged in the following order: fully connected, pooling, and convolutional. Figure 4.1 below depicts the architecture of the CNN model with appropriately connected layers.

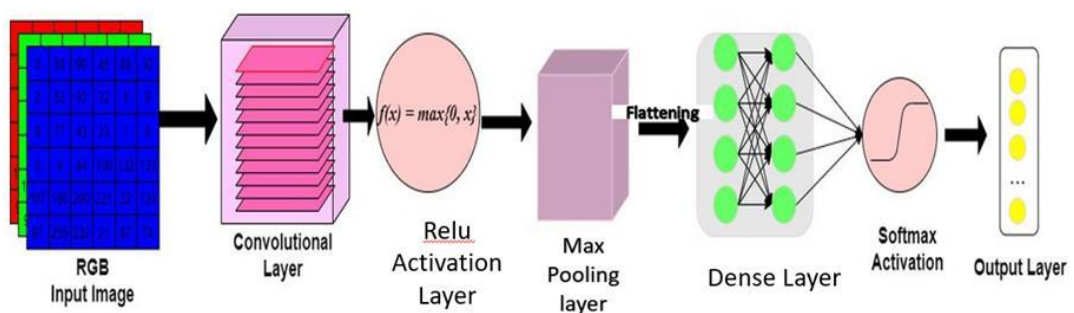
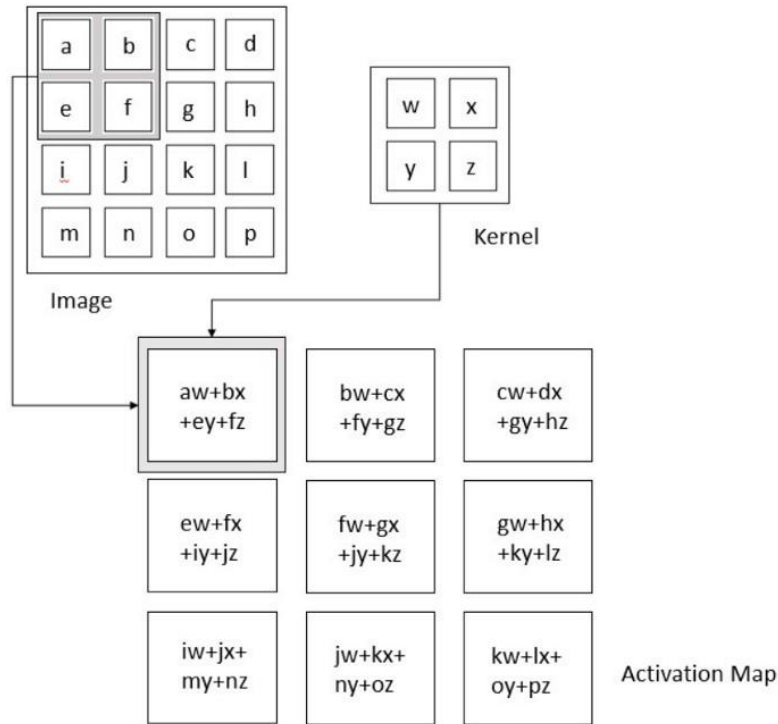


Figure 4.1 CNN Model Architecture



### 4.1.1 Convolution Layer

The primary parts of a CNN are convolutional layers. By moving across the full image and doing a dot product between the filter and the input pixels, they apply filters or kernels to the input image. By emphasizing the areas of the input image that are most crucial for identifying a particular pattern or object, this procedure creates a feature map. The kernel filter for the mathematical operation using the input image is displayed in Fig. 4.2.



**Figure 4.2 Convolution Layer**

The mathematical operation of the convolution layer is shown in Figure 4.2. Here, a 2x2 kernel filter is used to convolve an input image. The resulting feature map will highlight the edges and corners of the item.

### 4.1.2 Relu Activation Function

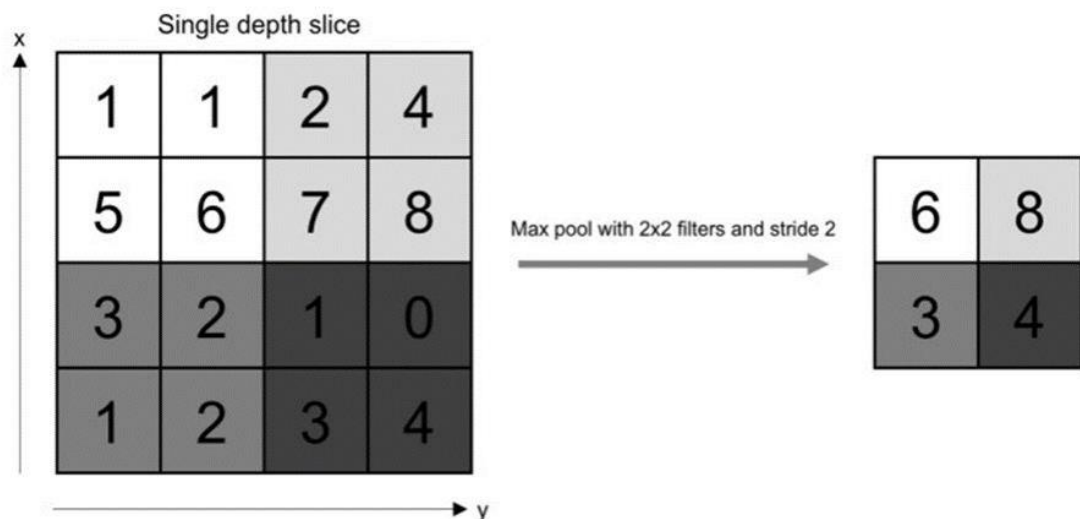
The Rectified Linear Unit (ReLU) activation function is a common CNN activation function. By adding nonlinearity, it improves the network's ability to depict complex relationships between the input and output data. The ReLU function allows only positive values to pass through the neuron. A positive input causes the neuron's output

to equal the input value; a negative input causes the output to equal zero.  $f(x) = \max\{0, x\}$  is the formula for the ReLU function.

### 4.1.3 Pooling Layers

Convolutional neural networks, which are employed in computer vision applications, have pooling layers. They preserve key characteristics while reducing the spatial dimensionality, or width and height. Another common type of pooling layer is max pooling, which moves the greatest value in a certain area of the input feature map to the layer that comes after it. As is customary, the user will also select the size of this type's specified zone. This size is referred to as the kernel size or pooling window. The max pooling operations are shown in Fig. 4.3. Let's examine an example using a 6x6 input feature map and a 2x2 pooling window. The following is how the max pooling process would proceed:

1. Non-overlapping 2x2 zones are constructed using the input feature map.
2. Each region's maximum value is determined.
3. The highest values from each region are combined to build a new 3x3 feature map.



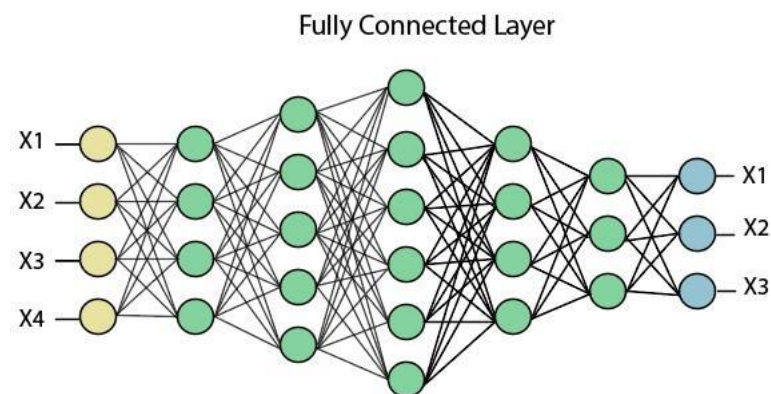
**Figure 4.3 Max Pooling Layer**

#### 4.1.4 Layer Completely Connected

Fully connected input layer: Before moving on to the following step, the output from the earlier layers is "flattened" and converted into a single vector.

While the first completely connected layer adds weights to the inputs obtained from feature analysis in order to anticipate the correct label, the fully connected output layer delivers the final probability for each label.

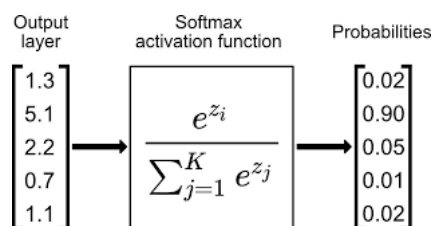
Fig. 4.4 shows how the completely connected layer functions internally.



**Figure 4.4 Fully Connected Layer**

#### 4.1.5 Softmax Activation Function

Most artificial neural networks use a mathematical function called the softmax activation function, particularly when dealing with multi-class classification tasks. In the output layer of the neural network, it is used to generate a probability distribution among the potential classes.



**Figure 4.5 Softmax Activation Function**

where the total (i.e.  $\sum$ ) is calculated over all  $j$  items in the input vector, and  $Z_i$  is the  $i$ -th element.

#### **4.1.6 Entropy of Sparse Categorical Cross**

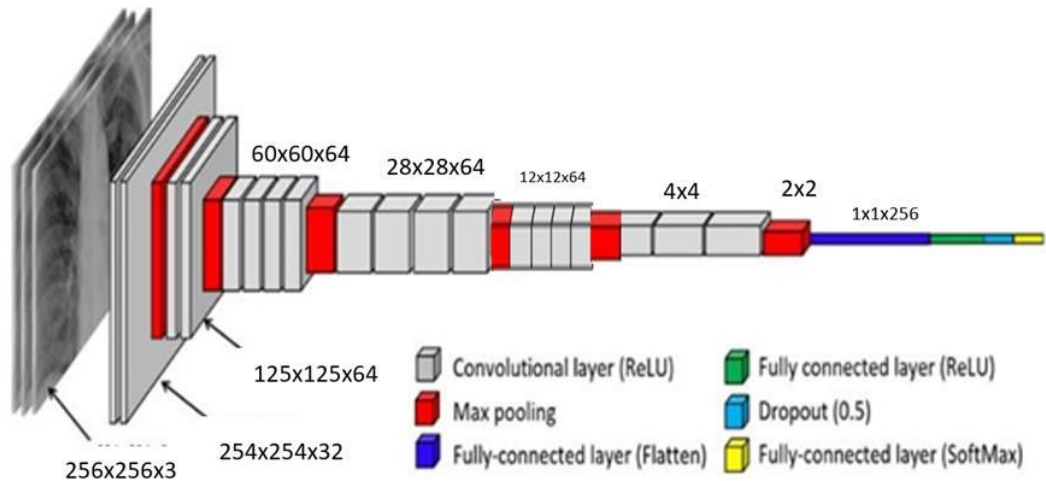
The sparse categorical cross-entropy loss function is used to account for the sparsity of the target label tensor and calculate the crossentropy loss between the target label tensor and the projected probability distribution. The formula for sparse categorical cross-entropy is as follows: Log and sum operations are applied to the anticipated probability distribution of each sample in the batch, but only over the element that corresponds to the class index of the target label.  $y\_true$  is the tensor of the target label, and  $y\_pred$  is the output that the model predicts.

#### **4.1.7 The Adam Optimizer**

Adam is an adaptive moment estimation optimizer used to optimize a neural network's training parameters. To put it another way, Adam is an optimizer that modifies the weights and biases of a neural network while it is being trained.

#### **4.1.8 The CNN Model Synopsis**

A multi-level Convolutional Neural Network (CNN) model with six levels and a diminishing output form as the levels drop is created using convolution and max pooling layers. A batch size of 32 is used while sending the input data. The convolution layers have a 3x3 kernel size, ReLU activation, and filter widths of 32, 64, 64, 64, 64, and 64, respectively. Each convolution layer in the model is followed by max pooling layers. 2x2 is the largest pooling size. The output form depends on the input size, kernel size, padding, and stride of each layer. The number of filters, kernel size, and number of preceding filters will all affect the output parameters of each convolution layer. However, the number of input and output channels affects a dense layer's output characteristics. Figure 4.6 provides a summary of all the CNN model data.



**Figure 4.6 Construction layers of CNN model at each level**

The information is formatted as follows:

**a. Layer 1 Convolution**

Convolutional layer: ReLU activation, 32 filters, 3x3 kernel size

Maximum pooling layer: 2x2 pool

Shape of Input: (224, 224, 3)

Shape of Output: (224, 224, 32)

896 parameters

**b. Layer of Convolution 2 Convolutional layer: 3x3 kernel size, 64 filters, ReLU activation**

Maximum pooling layer: 2x2 pool size

Shape of Input: (224, 224, 32)

Shape of Output: (112, 112, 64)

18,496 parameters

**c. Layer 3 Convolution**

Convolutional layer: 3x3 kernel size, 64 filters, and ReLU activation

Maximum pooling layer: 2x2 pool size

Shape of Input: (112, 112, 64)

Shape of Output: (56, 56, 64)

36,928 parameters

d. Layer 4 Convolution

Convolutional layer: 3x3 kernel size, 64 filters, and ReLU activation

Maximum pooling layer: 2x2 pool size

Shape of Input: (56, 56, 64)

Shape of Output: (28, 28, 64)

36,928 parameters

e. Layer 5 Convolutional

Convolutional layer: 3x3 kernel size, 64 filters, and ReLU activation

Maximum pooling layer: 2x2 pool size

Shape of Input: (28, 28, 64)

Shape of Output: (14, 14, 64)

36,928 parameters

f. Layer 6 Convolutional

Convolutional layer: 3x3 kernel size, 64 filters, and ReLU activation

Maximum pooling layer: 2x2 pool size

Shape of Input: (14, 14, 64)

Shape of Output: (7, 7, 64)

36,928 parameters

g. Layer Layer Flattening:

Shape of Flattened Input: (7, 7, 64)

Shape of Output: (576)

0 parameters

h. Layer Completely Connected One Layer: Dense Units: 64

ReLU is activated.

Shape of Input: (576)

Shape of Output: (64)

36,928 parameters

i. Layer of Output: Ten dense units (for ten classes)

Activation: Softmax

Shape of Input: (64)

Shape of Output: (10)

650 parameters

The following graphic shows the CNN model data overview of the output shapes and respected parameters for each max pooling layer and convolution layer.

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 222, 222, 32)	896
max_pooling2d (MaxPooling2D)	(None, 111, 111, 32)	0
conv2d_1 (Conv2D)	(None, 109, 109, 64)	18,496
max_pooling2d_1 (MaxPooling2D)	(None, 54, 54, 64)	0
conv2d_2 (Conv2D)	(None, 52, 52, 64)	36,928
max_pooling2d_2 (MaxPooling2D)	(None, 26, 26, 64)	0
conv2d_3 (Conv2D)	(None, 24, 24, 64)	36,928
max_pooling2d_3 (MaxPooling2D)	(None, 12, 12, 64)	0
conv2d_4 (Conv2D)	(None, 10, 10, 64)	36,928
max_pooling2d_4 (MaxPooling2D)	(None, 5, 5, 64)	0
conv2d_5 (Conv2D)	(None, 3, 3, 64)	36,928
max_pooling2d_5 (MaxPooling2D)	(None, 1, 1, 64)	0
flatten (Flatten)	(None, 64)	0
dense (Dense)	(None, 64)	4,160
dense_1 (Dense)	(None, 10)	650

**Total params:** 171,914 (671.54 KB)

**Trainable params:** 171,914 (671.54 KB)

**Non-trainable params:** 0 (0.00 B)

**Figure 4.7 CNN model output shapes and parameters.**

Before the photographs are fed into the algorithm, they are supplied in batches of 32 sizes. Just take note of the first column in the output shape, which shows that the batch size is 32 photos.

## Model Specifications

A. The output shape is reduced layer by layer.

The output shape of each layer can be ascertained in two ways:

1. Convolution is simple from the kernel filter to the convolution layer.

Input (height or width) - kernel(height or width + 1) = output shape (i, i) is the input size.

The size of the kernel is (k,k). Shape of output =  $i - k + 1$ .

This one applies when stride is 1 and padding is false. Simply divide the convolution layer in half using maximum pooling. The size of the convolution layer is (x,x).

Shape of output = floor (x/2)

B. Parameter Calculation

The kernel filter size for the first convolution layer is (m, n). (m x n x channels + 1)x the number of filters is the output parameter. The output parameters for the next layers are (m x n x the number of previous filters+1)x the number of filters. The thick layer's input channel number is i. J is the number of the output channel.

Output parameters =  $j \times (i+1)$

## 4.2 Training Process

During the training phase, the model parameters are iteratively updated using gradient descent and backpropagation, the loss function is determined, an optimization algorithm is selected, and the model parameters are initialized. A thorough process for teaching the neural network to identify tomato leaf disease is as follows:

### 4.2.1. Data Preprocessing

The following procedures are involved in loading and preparing the training data before the training process begins:

Scaling the input photographs to fall between 0 and 1 is known as data normalization.

The pixel values are divided by 255 to guarantee that all of the input values received by the neural network fall within a uniform range.

Data splitting is the process of separating the original data collection into training and validation data sets. In this case, the training set is used for training, while the validation set is used to evaluate and improve the model.

Batching: The model can learn from different subsets of the data and training efficiency



is increased by breaking the data up into digestible chunks.

Resizing: The input photos are reduced in size because the model requires them to be 224x224.

#### **4.2.2. Defining the Model Architecture**

Determining the CNN's architecture, including the type and number of layers, is the second step in the training process. Layer Number and Type: The fully linked, dense layer of the model makes the diagnosis after several layers of convolution and max pooling have extracted features.

Activation's Functions ReLU (Rectified Linear Unit) activation is used by the convolutional and dense layers to give the model non-linearity and aid in the comprehension of intricate patterns.

Optimizer: To avoid oscillations during the training phase, the Adam optimizer algorithm modifies the learning rate to modify the values of the model parameters.

Loss Function: Minimal The model uses categorical cross-entropy to determine the difference between the class labels for classification and the model's class label output. Section 4.1 discusses these elements of the model architecture.

#### **4.2.3. Compiling the Model**

The model architecture needs to be compiled after it has been defined. This calls for:

Optimizer: Because of its ability to manage big datasets and adjust the learning rate during training, the Adam optimizer was selected.

Loss Function: When the objective variable is a single integer that represents the class, the Sparse Categorical Cross-Entropy loss function is useful for managing multi-class classification problems.

Metrics: Since accuracy demonstrates how well the model detects tomato leaf diseases, it is the most popular metric for monitoring the model's performance during training. Sections 4.1.4 (Adam optimizer) and 4.1.5 (Sparse Categorical Cross-Entropy loss function) provide an explanation of this stage.

#### **4.2.4. Training the Model**

Model training is the last phase of the training procedure. This comprises:

Data Feeding: The model receives batches of the training data. The model calculates the output (predicted class) and the loss for each batch.

Parameter Adjustment: To minimize the loss function, the backpropagation method and the Adam optimizer are used to modify the model's parameters (weights).

The number of epochs indicates how many times the model will run through all of the training data. It is typical practice to use multiple training epochs to enable the model to converge toward the global optimum.

Validation: To demonstrate that the model is in fact preventing overfitting and to further monitor model performance, the validation set is run against for every training period. The test set will be tested using the trained model.

## IMPLEMENTATION AND RESULTS

The algorithmic procedure and related Python program code are explained in this chapter.

### 5.1 Algorithm Process

**Step 1:** Import all necessary libraries.

**Step 2:** Accept input parameters such the number of classes, batch size, and image size.

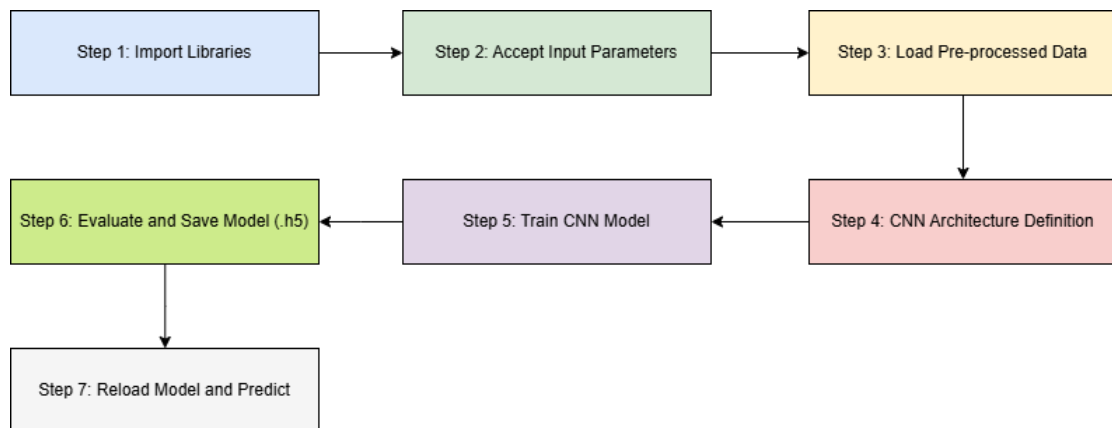
**Step 3:** Load the pre-processed photos and dataset.

**Step 4:** Indicate the CNN architecture's structure.

**Step 5:** Run the provided CNN model across a variety of epochs for training.

**Step 6:** Assessing the model's performance and saving it in.h5 format.

**Step 7:** Reload the model and make predictions about the tomato leaf photos.



**Figure 5.1 Algorithm Process**

### 5.2 Main Code

```
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
import tensorflow as tf
import pandas as pd
import os, requests, cv2, random
from tensorflow.keras.models import Sequential
```

```

from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.optimizers import Adam
from tensorflow.keras import models
from tensorflow.keras import Sequential, layers
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow import keras
from sklearn.metrics import confusion_matrix, classification_report

train_datagen = ImageDataGenerator(rescale=1/255.0,
                                   shear_range=0.2,
                                   zoom_range=0.2,
                                   horizontal_flip=True,
                                   validation_split=0.3)# specifying the validation split inside the
function
test_datagen = ImageDataGenerator(rescale=1/255.0,
                                   shear_range=0.2,
                                   zoom_range=0.2,
                                   horizontal_flip=True)
train_gen = train_datagen.flow_from_directory(
    train_data_dir,
    target_size=(224, 224),
    batch_size=32,
    shuffle=True,
    class_mode='categorical',
    subset='training')
val_gen = train_datagen.flow_from_directory(
    train_data_dir,
    target_size=(224,224),
    batch_size=32,
    shuffle=True,
    class_mode='categorical',
    subset='validation')
test_gen = test_datagen.flow_from_directory(

```

```

test_data_dir,
target_size=(224, 224),
batch_size=32,
class_mode='categorical',
shuffle = False)

cnn = models.Sequential()
cnn.add(layers.Conv2D(32, kernel_size=(3,3), activation='relu', input_shape=[224,
224,3])),
cnn.add(layers.MaxPooling2D(pool_size=(2, 2))),
cnn.add(layers.Conv2D(64, (3,3), activation='relu')),
cnn.add(layers.MaxPooling2D((2, 2))),
cnn.add(layers.Conv2D(64, (3,3), activation='relu')),
cnn.add(layers.MaxPooling2D((2, 2))),
cnn.add(layers.Conv2D(64, (3, 3), activation='relu')),
cnn.add(layers.MaxPooling2D((2, 2))),
cnn.add(layers.Conv2D(64, (3, 3), activation='relu')),
cnn.add(layers.MaxPooling2D((2, 2))),
cnn.add(layers.Conv2D(64, (3, 3), activation='relu')),
cnn.add(layers.MaxPooling2D((2, 2))),
cnn.add(layers.Flatten()),
cnn.add(layers.Dense(64,activation='relu'))
#output layer
cnn.add(layers.Dense(10,activation='softmax'))
cnn.summary()
opt = keras.optimizers.Adam(learning_rate=0.0001)

cnn.compile(optimizer=opt,loss='categorical_crossentropy',metrics=['accuracy'])
es = EarlyStopping(monitor = 'val_accuracy',
                    mode = 'max',
                    patience = 20,
                    verbose = 1,
                    restore_best_weights = True)
history = cnn.fit(x = train_gen,

```

```

callbacks = [es],
# steps_per_epoch = 7000//32,
epochs = 150,
# validation_steps = 3000//32,
validation_data = val_gen)
plt.plot(history.history['loss'], label='Training loss')
plt.plot(history.history['val_loss'], label='Validation loss')
plt.legend()
plt.show()
plt.plot(history.history['accuracy'], label='Training accuracy')
plt.plot(history.history['val_accuracy'], label='Validation accuracy')
plt.legend()
plt.show()
cnn.save('tomato.h5')

```

### 5.3 Prediction of an Image Code

```

import numpy as np
import tensorflow as tf
from tensorflow.keras.preprocessing.image import load_img, img_to_array
import os

# Load the trained model
model = tf.keras.models.load_model('tomato.h5')

# Define the class names (update this list based on your dataset's class labels)
class_names = list(train_gen.class_indices.keys())

# Function to preprocess and predict a single image
def predict_image(image_path):
    # Load the image
    img = load_img(image_path, target_size=(224, 224))
    plt.imshow(img) # Optional: display the image
    plt.axis('off')
    plt.show()

    # Preprocess the image
    img_array = img_to_array(img) / 255.0 # Normalize

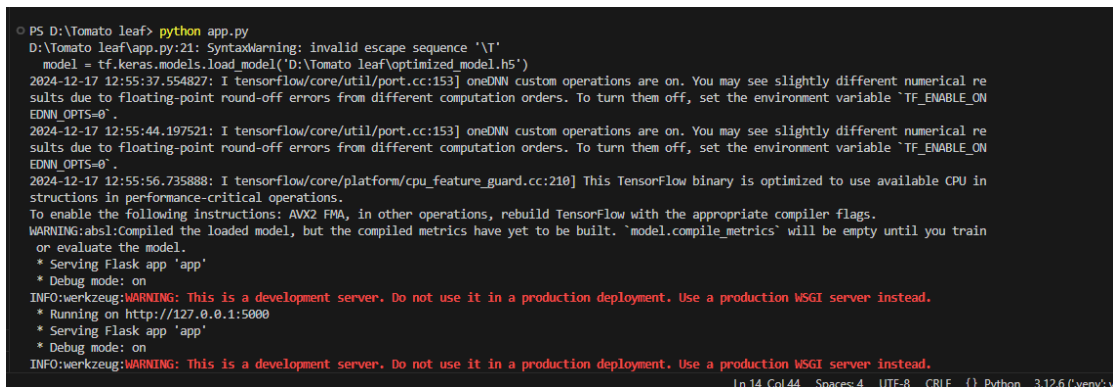
```

```

img_array = np.expand_dims(img_array, axis=0) # Add batch dimension
# Predict the class
predictions = model.predict(img_array)
predicted_class = np.argmax(predictions[0]) # Index of the highest score
confidence = predictions[0][predicted_class] # Confidence score
# Display results
print(f"Predicted Class: {class_names[predicted_class]}")
print(f"Confidence: {confidence:.2f}")

# Directory containing test images (replace 'test_images' with your directory path)
test_images_dir = 'test_images'
# Predict for all images in the directory
for filename in os.listdir(test_images_dir):
    if filename.lower().endswith(('.png', '.jpg', '.jpeg')): # Filter for image files
        file_path = os.path.join(test_images_dir, filename)
        print(f"Processing {filename}...")
        predict_image(file_path)

```



```

PS D:\Tomato leaf> python app.py
D:\Tomato leaf\app.py:21: SyntaxWarning: invalid escape sequence '\t'
  model = tf.keras.models.load_model('D:\Tomato leaf\optimized_model.h5')
2024-12-17 12:55:37.554827: I tensorflow/core/util/port.cc:153] oneDNN custom operations are on. You may see slightly different numerical re
sults due to floating-point round-off errors from different computation orders. To turn them off, set the environment variable `TF_ENABLE_ON
EDNN_OPTS=0`.
2024-12-17 12:55:44.197521: I tensorflow/core/util/port.cc:153] oneDNN custom operations are on. You may see slightly different numerical re
sults due to floating-point round-off errors from different computation orders. To turn them off, set the environment variable `TF_ENABLE_ON
EDNN_OPTS=0`.
2024-12-17 12:55:56.735888: I tensorflow/core/platform/cpu_feature_guard.cc:210] This TensorFlow binary is optimized to use available CPU in
structions in performance-critical operations.
To enable the following instructions: AVX2 FMA, in other operations, rebuild TensorFlow with the appropriate compiler flags.
WARNING:absl:Compiled the loaded model, but the compiled metrics have yet to be built. `model.compile_metrics` will be empty until you train
or evaluate the model.
* Serving Flask app 'app'
* Debug mode: on
INFO:werkzeug:WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
* Running on http://127.0.0.1:5000
* Serving Flask app 'app'
* Debug mode: on
INFO:werkzeug:WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.

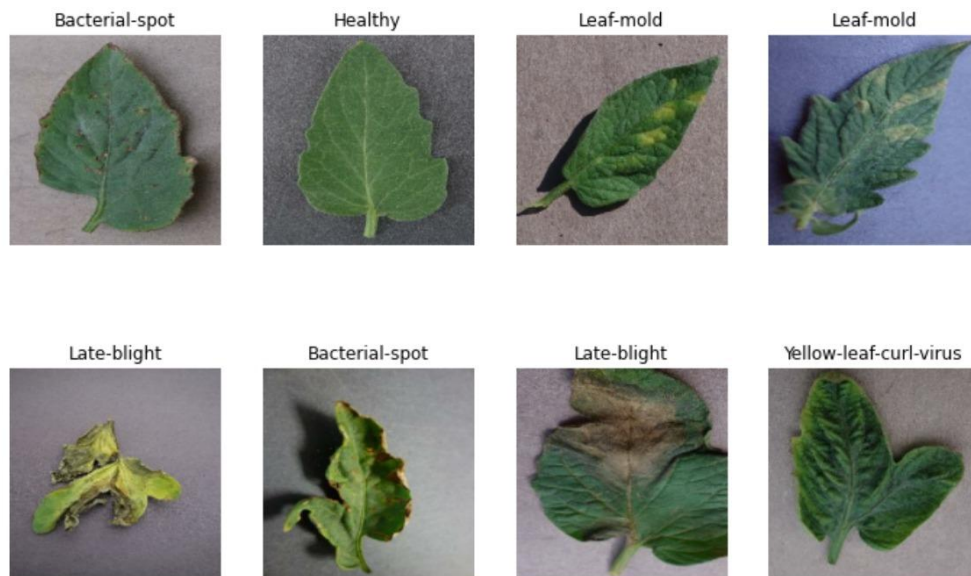
```

**Figure 5.2 Terminal Used for Model Execution**

This code offers the fundamental structure for putting the suggested algorithmic method for CNN-based tomato leaf disease detection into practice. However, this will be tailored according to the parameters and the dataset's specificity. The visualization and interpretation of training outcomes using the suggested CNN-based technique through predictive analysis will be covered .

## 5.4 Visualization

The purpose of this work is to identify the various tomato leaf diseases using CNNs. This experiment used the Plant Village dataset, which was downloaded from the Kaggle website and consists of ten different types of tomato leaf pictures with labels identifying healthy and damaged leaves. Figure 5.3 shows the many types of tomato leaf images together with their corresponding labels.



**Figure 5.3 Visualization of tomato leaf images**

## 5.5 Training results

The training results for a CNN typically include measures for accuracy and loss during the training process. Here, the term "loss metric" refers to the model's overall performance on training data, which is often quantified by comparing the actual output to the model's anticipated output. The loss function should be minimized during the training procedure. As described in the CNN model architecture concept in section 4.1 and the training procedure in section 4.2 of chapter 4. 50, 100, and 150 epochs were used to train the CNN model. Table 2 displays the train, valid, and test dataset lengths as covered in Chapter 3. A change in the number of epochs has been shown to improve accuracy and decrease loss, according to the data displayed in Table 5.1.



<b>Sr. No.</b>	<b>No. of Epochs</b>	<b>Train Accuracy</b>	<b>Train Loss</b>	<b>Valid Accuracy</b>	<b>Valid Loss</b>
1	50	0.92	0.19	0.91	0.24
2	100	0.97	0.06	0.94	0.17
3	150	0.96	0.1	0.93	0.2

**Table 5.1 Comparison of Loss and Accuracy for train and valid at different epochs.**

This covers the initial model parameter setting, loss function specification, optimization technique selection, and iterative model parameter update through gradient descent and backpropagation. It outlines the sequential procedure for training a neural network:

1. The preprocessed training data is loaded.

To accomplish this, the data needs to be prepared in accordance with the model's requirements, separated into batches, and normalized. This phase is covered in Chapter 3 Sections 3.3 and 3.5.

2. Definition of the Model Architecture This is the second training step where the neural network's architecture is specified. It entails specifying the optimizer, loss function, activation functions, and number and kind of layers. The CNN model's design is explained in Section 4.1.

3. Putting the model together. This involves setting up an optimizer, a loss function, and any other metrics that need to be monitored throughout the training process for the model. Subsections 4.1.4 and 4.1.5 of the current chapter contain the Adam Optimizer and the Sparse Categorical Cross entropy Loss function, respectively.

4. Model training: The last phase of the training procedure is this. In order to minimize the loss function, it feeds the model with training data, calculates the output, and modifies the model parameters using the Adam optimizer algorithm. How many times the complete training dataset is utilized to train the model depends on the number of training epochs. In the Next Chapter – 5 Results, under Training Results, the trained model is tested on the test set. The CNN model architecture was described in this chapter, along with the sequential steps required to train a neural network model, such as loading and preparing the training data, establishing the model architecture,

assembling the model, and training the model. The following chapter, Chapter 5, presents the algorithm process and implementation code for the automatic detection of tomato leaf disease.

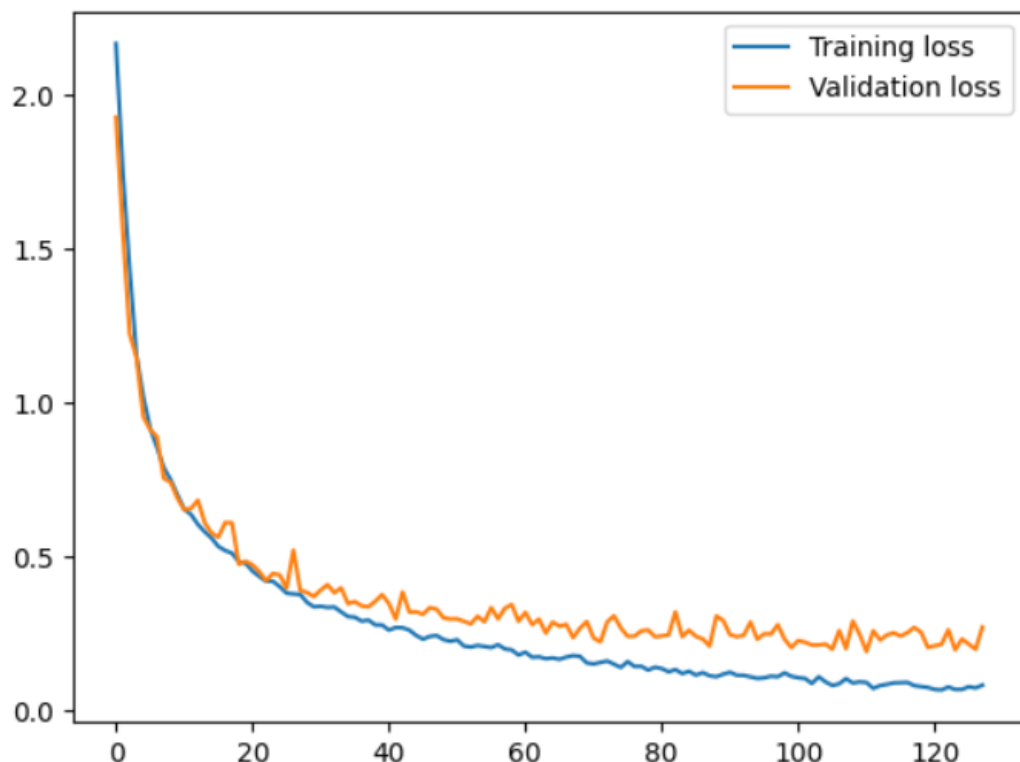
```
Epoch 121/150
219/219 ————— 208s 942ms/step - accuracy: 0.9786 - loss: 0.0606 - val_accuracy: 0.9317 - val_loss: 0.2099
Epoch 122/150
219/219 ————— 446s 2s/step - accuracy: 0.9760 - loss: 0.0629 - val_accuracy: 0.9337 - val_loss: 0.2150
Epoch 123/150
219/219 ————— 288s 1s/step - accuracy: 0.9764 - loss: 0.0653 - val_accuracy: 0.9167 - val_loss: 0.2620
Epoch 124/150
219/219 ————— 170s 769ms/step - accuracy: 0.9738 - loss: 0.0736 - val_accuracy: 0.9350 - val_loss: 0.1969
Epoch 125/150
219/219 ————— 177s 799ms/step - accuracy: 0.9780 - loss: 0.0630 - val_accuracy: 0.9290 - val_loss: 0.2320
Epoch 126/150
219/219 ————— 174s 787ms/step - accuracy: 0.9692 - loss: 0.0822 - val_accuracy: 0.9347 - val_loss: 0.2159
Epoch 127/150
219/219 ————— 174s 789ms/step - accuracy: 0.9713 - loss: 0.0739 - val_accuracy: 0.9327 - val_loss: 0.1991
Epoch 128/150
219/219 ————— 178s 806ms/step - accuracy: 0.9714 - loss: 0.0812 - val_accuracy: 0.9163 - val_loss: 0.2710
Epoch 128: early stopping
Restoring model weights from the end of the best epoch: 108.
```

---

```
Epoch 128/150
219/219 ————— 178s 806ms/step - accuracy: 0.9714 - loss: 0.0812 - val_accuracy: 0.9163 - val_loss: 0.2710
Epoch 128: early stopping
Restoring model weights from the end of the best epoch: 108.
```

**Figure 5.4 Loss and Accuracy at 150 epochs**

The below figure shows the variations of accuracy and loss for both train and validation.





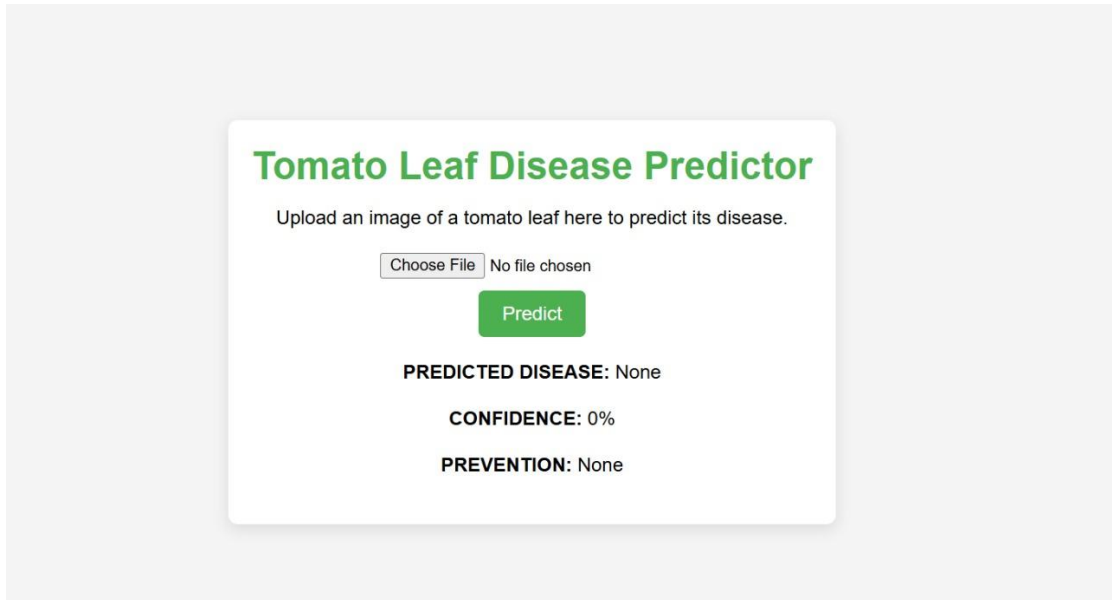
**Figure 5.5 Plotted the accuracy and loss for both train and validation.**

The accompanying training and validation accuracy graph shows that validation accuracy increases in tandem with training accuracy. As can be seen from the training and validation loss graphs, the validation loss likewise drops as the training loss does.

## 5.6 Predicted Images

In this project, the trained CNN model saved as `saved_model.h5` is applied during picture prediction to identify tomato leaf diseases. In order to create an interactive and user-friendly experience, the web interface application is designed with HTML, JavaScript, and CSS. This enables end users to choose photographs from their local systems and submit them for disease prediction. The backend features, such as image processing and prediction, are controlled by the Flask. The CNN model is used to process the uploaded photos, and the prediction results are returned by the Flask server. The uploaded photos are classified using the dynamically loaded model before being shown to the user via the web interface. The file `tomatoDL.py` contains the entire web interface. This file contains all of the code required to accept image uploads, deliver the

webpage, and use the trained model to generate predictions. Users may upload photographs, receive forecasts, and view the outcomes in real time thanks to the way the interface is set up.



**Figure 5.6 Home Page**

### **Different types of tomato leaf images predicted with predicted label and confidence**

Different tomato leaf picture types can be anticipated depending on the disease kind in the leaves in a CNN-based tomato leaf disease detection system. As covered in chapter 4, the CNN model and training procedure are two examples of typical tomato leaf picture types that can be predicted using a CNN.

## Tomato Leaf Disease Predictor

Upload an image of a tomato leaf here to predict its disease.

Choose File fc963fa5-71....S 2706.JPG

Predict

**PREDICTED DISEASE:** Septoria\_leaf\_spot

**CONFIDENCE:** 100%

**PREVENTION:** Practice crop rotation, remove plant debris, and apply appropriate fungicides.



**Figure 5.7 Predicted Image**

## CONCLUSION

This study presents a CNN-based approach for detecting ten common tomato leaf diseases, such as mosaic virus, bacterial spot, early blight, late blight, and yellow leaf curl virus. After being trained on many datasets of tomato leaf pictures, the model accurately distinguishes between diseased and healthy leaves, making it a reliable tool for monitoring the health of tomato crops. Its user-friendly web interface allows users to upload photos of leaves and receive real-time predictions. Its quick picture analysis makes it ideal for large-scale agricultural applications, allowing farmers to move quickly to minimize losses and boost output.

The technology has a lot of potential for future development and expanded application. Real-time monitoring can be provided by integrating cameras on robotic platforms or drones to facilitate continuous field observation and early sickness detection. Using transfer learning with pre-trained CNN models like ResNet or Inception can further improve accuracy and generalization, especially when dealing with sparse input. Through cloud-based implementation, the system may be available worldwide, allowing for remote forecasting and scalability. Furthermore, the addition of IoT devices, such as soil moisture and humidity sensors, will enable comprehensive crop health monitoring by combining environmental data with disease projections. The method can also be extended to detect diseases in other crops, such as potatoes, cucumbers, and peppers, to boost its versatility and impact across a variety of agricultural industries. As a result of these advancements, the system will become a vital tool for promoting effective, environmentally responsible, and sustainable farming practices.

## REFERENCES

- [1]. D. T. Mane and U. V. Kulkarni, "A survey on supervised Convolutional Neural Network and its major applications," *International Journal of Rough Sets and Data Analysis*, vol. 4, no. 3, pp. 71–82, 2017.
- [2]. *Procedia Computer Sci.* 2018; 133:1040–7; Aravind KR, Raja P, Anirudh R. Classification of Tomato Crop Diseases Using A Pre-Trained Deep Learning Algorithm.
- [3] Karthik R, Hariharan M, Mathikshara Priyanka, Anand Sundar, Johnson Annie, and Menaka R. Remaining CNN with attention integrated for tomato leaf disease detection. *Applied Soft Computing*, 2020.
- [4]. Zhao Ming, Zhong Yong. studies on deep learning for identifying apple leaf disease. 168:105146 in *Compute Electron Agric.* 2020.
- [5]. Ruiz H, Vergara A, Selvaraj MG, et al. AI-powered pest and disease detection for bananas. *Plant Methods*, 15:92 (2019).
- [6] T. Darrell, J. Malik, J. Donahue, and R. Girshick. "Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation." 2014, pp. 580-587; *Proceedings of the IEEE Conference on Pattern Recognition and Computer Vision, CVPR '14*.
- [7]. Park DS, Kim SC, Yoon S, and Fuentes A. A powerful deep learning-based detector for identifying pests and tomato plant illnesses in real time. *Sensors*. 2017:17, 2022.
- [8] Picon A, Echazarra J, Johannes A, Ortiz-Barredo A, Seitz M, and Alvarez-Gila A. Deep convolutional neural networks for the classification of agricultural diseases in the wild using mobile capture devices. 280–90 in *Computer Electron Agric.* 2019;1(161).