PROJECT REPORT ON

TOMATO LEAF DISEASE DETECTION USING CNN

Submitted in partial fulfilment of the Requirement
for the award of the degree of

BACHELOR OF TECHNOLOGY
IN
ELECTRONICS AND COMMUNICATION ENGINEERING

Submitted by

Project Associates	Regd. No
T. Bindu Madhavi	19091A0424
K. Yerikalaiah	19091A04S5
T. Vamsi	19091A04P7
S, Naveen Kumar	19091A04C3

Under the Esteemed Guidance of
Dr. P. V. Gopi Krishna Rao
Ph. D, MISOI, MIAENG
Professor, RGMCET, Nandyal

(ESTD-1995)

DEPARTMENT OF
ELECTRONICS AND COMMUNICATION ENGINEERING

RAJEEV GANDHI MEMORIAL COLLEGE
OF ENGINEERING AND TECHNOLOGY
(AUTONOMOUS)
Affiliated to J.N.T.U.A - Anantapuramu, Approved by A.I.C.T.E - New Delhi,
Accredited by N.B.A - New Delhi, Accredited by NAAC with A+ Grade – New Delhi
NANDYAL -518501, Nandyal Dist. A.P.

YEAR: 2019 - 2023

RAJEEV GANDHI MEMORIAL COLLEGE OF ENGINEERING & TECHNOLOGY

AUTONOMOUS

(Approved by A.I.C.T.E - New Delhi, Affiliated to JNTUA - Anantapuramu, Accredited by NBA - New Delhi, Accredited by NAAC with 'A+' Grade - New Delhi)

NANDYAL - 518 501, A.P, India

CERTIFICATE

This is to certify that the dissertation entitled "TOMATO LEAF DISEASE DETECTION USING CNN " is being submitted by T. Bindu Madhavi (19091A0424), K. Yerikalaiah (19091A04S5), T. Vamsi ((19091A04P7), S. Naveen Kumar (19091A04C3) under the guidance of Dr. P. V Gopi Krishna Rao, Professor for Project of the award of B.Tech Degree in Electronics and Communication Engineering, Rajeev Gandhi Memorial College of Engineering & Technology, Nandyal (Autonomous) (Affiliated to J.N.T.U.A Anantapuramu) is a record of bonafide work carried out by them under our guidance and supervision.

[illegible]

Dr. Kethepalli Mallikarjuna
Head of the Department

Dr. V.N.V. SATYA PRAKASH
[illegible] ISTE,MIETE,MIE.
Professor
Department of E.C.E.
R.G.M College of Engg & Tech., (Autonomous)
NANDYAL - 518 501, Kurnool (Dist), A.P.

[illegible] 3/5/23

Dr. P. V. Gopi Krishna Rao
Project Guide

Professor
Department of ECE
RGM College of Engg. & Tech. (Autonomous)
NANDYAL-518 501, Kurnool (Dist), A.P.

[illegible]

Signature of the External Examiner

Date of Viva-Voce: 05/05/2023

ACKNOWLEDGEMENT

We earnestly take the responsibility to acknowledge the following distinguished personalities who graciously allowed us to carry our project work successfully.

We express deep gratitude to our guide **Dr. P. V. Gopi Krishna Rao**, M.E. Ph.D, Professor in ECE Department, RGMCET, Nandyal for his guidance, incessant help and encouragement throughout the course of the project work, which contributed, to the successful completion of this project.

We would like to thank **Dr. Kethepalli Mallikarjuna**, M.Tech., Ph.D, Professor & Head, Department of ECE, RGMCET, Nandyal, for this valuable suggestions and encouragement whenever we encountered difficulties during the project work.

We would like to express my deep sense of gratitude and indebtednessto **Dr. T. Jayachandra Prasad**, Principal, RGMCET, Nandyal for his sustained encouragement and moral support throughout the course.

We are thankful to our honorable Chairman **Dr. M. Santhiramudu**, RGM Group of Institutions and **Sri. M. Sivaram**, Managing Director, RGMCET, Nandyal for providing us with required facilities in the college.

We express our thanks to all the faculty and staff members of ECE Department, RGMCET, Nandyal, for their support in various aspects in completing the project.

Project Associates

T. Bindu Madhavi	19091A0424
K. Yerikalaiah	19091A04S5
T. Vamsi	19091A04P7
S. Naveen Kumar	19091A04C3

INDEX PAGE NO

ABSTRACT

Plant diseases cause low agricultural productivity. Plant diseases are challenging to control and identify by the majority of farmers. In order to reduce future losses, early disease diagnosis is necessary. This study presents a deep learning approach for detecting tomato leaf diseases using Convolutional Neural Networks (CNNs). The proposed method involves preprocessing the tomato leaf images, followed by training the CNN model to classify them into one of ten categories: healthy, yellow leaf curl virus (YLCV), bacterial spot (BS), early blight (EB), leaf mold (LM), spectoria leaf spot (SLS) target spot (TS), two spotted spider mite spot(TSSMS), mosaic virus(MV) and late blight (LB). The model was trained using a dataset of 16021 tomato leaf images. The training was conducted for 10 epochs, 20 epochs, and 50 epochs, and the accuracy achieved was 64%, 94%, and 97%, respectively. These results demonstrate the effectiveness of the proposed approach in detecting tomato leaf diseases, and the performance improves with increasing epochs. The automated approach can aid in the early detection and prevention of tomato diseases, which can ultimately help in improving the yield and quality of tomato crops.

List of Abbreviations

CNN	Convolutional Neural Networks
MPL	Max Pooling layer
YLCV	Yellow Leaf Curl Virus
BS	Bacterial Spot
EB	Early Blight
LM	Leaf Mold
SLS	Spectorial Leaf Spot
TS	Target Spot
TSSMS	Two spotted Spider Mite Spot
MV	Mosaic Virus
LB	Late Blight

List of Figures

List of Tables

CHAPTER – 1

INTRODUCTION

1.1 Background

Tomatoes are one of the most widely cultivated and consumed crops globally, making it a significant part of the agriculture industry. Unfortunately, tomato plants are susceptible to various diseases, which can lead to significant economic losses due to reduced yield and quality. One of the most common diseases that affect tomato plants is grey leaf spot, which damages and kills the leaves, ultimately hindering the plant's ability to produce fruit. The infection caused by the pathogen responsible for grey leaf spot in tomato plants progresses through four phases: contact, invasion, latency, and onset. Detecting diseases early can help prevent large-scale pandemics and enable appropriate management practices.

Traditional methods of detecting diseases are time-consuming and expensive, especially when the farm is extensive, making it challenging to monitor each plant. Thus, there is a need for a more efficient and cost-effective solution. Image processing techniques can automate the detection of diseases in leaves, thereby saving time, money, and effort. The dataset used in this study consists of 16021 images of ten types of diseased tomato leaf images. All images are resized to 256 x 256 and divided into three parts, namely, training, testing, and validation datasets. By analyzing the features of diseased leaves, image processing technology can accurately diagnose illnesses quickly. Deep learning techniques, specifically convolutional neural networks (CNNs), have proven to be effective in image classification tasks, including plant disease detection.

1.2 Problem Statement

Manual plant disease detection methods are time-consuming and inefficient, particularly for large-scale farms. Traditional disease detection techniques, such as visual inspection, are susceptible to errors and often require a team of experts. Moreover, early disease detection is essential to control and prevent plant diseases, which traditional techniques cannot guarantee. Therefore, there is a need for an accurate, efficient, and automated disease detection approach for tomato plants that can provide early detection and effective prevention.

1.3 Objectives

The main objective of this study is to develop an automated system for detecting and classifying tomato leaf diseases using CNN. Specifically, this study aims to:

1 Develop a CNN model that can accurately detect and classify common tomato leaf diseases, such as early blight, late blight mold, bacteria spot, leaf mold, target spot, yellow leaf curl virus, two spotted spider mite, mosaic virus and septoria leaf spot.
2 Compare the performance of the developed CNN model with at different epochs.
3 Contribute to sustainable agriculture by providing a cost-effective, automated solution to identify tomato leaf diseases at an early stage, thereby enabling farmers to take preventive measures and reduce crop losses.

1.4 Motivation

Identifying and recognition of leaves disease is the solution for saving the reduction of large farm in crop disease detection and profit in productivity, it is beneficial in agricultural institute, Biological research.

1.5 Organization of the Project

The remaining sections of this study are organized as follows:

- Chapter 2: Literature Review – provides about Literature Survey in which the existing methods for image classification of Plant leaf diseases were described.
- Chapter 3: Methodology - presents a detailed description of the dataset used in this study on tomato leaf disease detection using CNN, including dataset collection, preprocessing, datset statistics and dataset split for train, valid and test datasets.
- Chapter 4: This chapter describes the Convolutional Neural Network (CNN) model architecture used for our image classification task and the process of training the model.
- Chapter 5: This chapter, describe the algorithm process and related python program code.
- Chapter 6: This chapter, that presents the visualization of the training results and predictive analysis obtained from the proposed CNN-based approach.
- Chapter 7: This chapter, describes the conclusion and future scope.

CHAPTER – 2

LITERATURE REVIEW

In this chapter, presents literature survey of traditional plant disease detection approaches based on computer vision technologies are commonly utilized to extract the texture, shape, colour, and other features of disease spots. This method has a low identification efficiency because it relies on an extensive expert understanding of agricultural illnesses. Many academics have conducted significant research based on deep learning technology to increase the accuracy of plant disease detection in recent years, thanks to the fast growth of artificial intelligence technology. The majority of existing techniques to plant disease analysis are based on disease classification.

A. A Survey on Supervised Convolutional Neural Network and Its Major Applications; D. T. Mane and U. V. Kulkarni

With the advent of deep learning, the world has proceeded into the new era of machine learning. With the main intention of getting closer to the original goal of machine learning, that is, Artificial Intelligence, deep learning has opened up new avenues to explore. Artificial Neural Networks (ANNs) are biologically motivated machine learning algorithms applied to solve problems, where conventional approach fails, such as computer vision. It takes in the input, let it be an image or an audio signal, extracts features which describe the input and generalizes these features so that the results obtained can be replicated for other examples of the input. This paper gives an overview of a particular type of ANN, known as supervised Convolutional Neural Network (CNN) and gives information of its development and results in various fields. Initially, we see the history of CNN followed by its architecture and results of its applications. The references of the few used papers have been mentioned here.

B. Tomato crop disease classification Using A Pre-Trained Deep Learning Algorithm; Aravind KR, Raja P, Anirudh R.

A study on the classification of three major tomato crop diseases - Early Blight, Late Blight, and Leaf Mold - using a pre-trained deep learning algorithm called VGG16. The authors describe the dataset used for the study, which consisted of images of tomato leaves infected with the three diseases and healthy leaves. The VGG16 algorithm was fine-tuned using transfer learning to classify the images into the four categories. The authors report that the VGG16 algorithm achieved an accuracy of 98.67% in classifying the images, outperforming other algorithms such as Random Forest and K-Nearest Neighbours. The paper also discusses the limitations of the study and potential areas for future research, such as the use of more diverse datasets and the development of a mobile application for farmers to identify crop diseases.

C. Attention Embedded Residual CNN for Disease Detection in Tomato Leaves; Karthik R, Hariharan M, Anand Sundar, Mathikshara Priyanka, Johnson Annie, Menaka R.

A dataset consisting of images of tomato leaves affected by five different diseases - Early Blight, Late Blight, Leaf Mold, Septoria Leaf Spot, and Spider Mites - and healthy leaves. The proposed CNN architecture consists of residual blocks, which enable the network to learn the mapping between the input and output more efficiently, and attention modules, which help the network to focus on the most important features in the images. The authors report that the proposed approach achieved an accuracy of 98.3% in detecting tomato crop diseases, outperforming other state-of-the-art approaches such as VGG16 and Inception-v3. The paper also provides a detailed analysis of the performance of the proposed approach on different disease classes and provides visualizations of the attention maps generated by the attention modules.

D. Research on deep learning in apple leaf disease recognition. Comput Electron Agric; Zhong Yong, Zhao Ming.

The article presents a study on the use of deep learning algorithms for the recognition of apple leaf diseases. The authors developed a deep learning framework that uses a convolutional neural network (CNN) to automatically identify and classify different apple leaf diseases based on images. The authors trained their model on a large dataset of apple leaf images and achieved high accuracy in disease recognition across multiple apple cultivars. They also demonstrated the potential for their model to be used in real-world scenarios, such as in orchards and nurseries. The findings of this study may have practical applications in the agricultural industry by providing a tool for early detection and diagnosis of apple leaf diseases. This could ultimately lead to improved crop yields and reduced economic losses for apple farmers.

Overall, this article demonstrates the potential for deep learning algorithms to revolutionize the field of crop disease detection and management, with practical applications in a range of crops and settings.

E. AI-powered banana diseases and pest detection. Plant Methods. 2019; 15:92; Selvaraj MG, Vergara A, Ruiz H, et al.

A dataset of banana plant images and show that it can accurately detect the presence of diseases and pests with high accuracy. They also demonstrate that the method can be applied in real-world settings using a smartphone app that allows farmers to easily capture and upload images of their plants for analysis. Overall, the study shows the potential of machine learning techniques for plant disease and pest detection and highlights the importance of developing practical and accessible tools to support farmers in monitoring and managing their crops. The authors suggest that their approach could be extended to other crops and regions, contributing to the development of more sustainable and efficient agricultural practices.

F. Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation; Girshick, R., J. Donahue, T. Darrell, and J. Malik.

The paper highlights the potential of deep learning techniques for object detection and segmentation tasks, and introduces a novel approach that combines convolutional neural networks with region-based processing for improved accuracy and efficiency. The R-CNN approach has since been extended and improved in subsequent research, and has become a widely used method in computer vision and object recognition.

G. Characteristics of tomato plant diseases—a study for tomato plant disease identification; Fuentes A, Yoon S, Youngki H, Lee Y, Park DS.

Deep learning was proposed by Fuentes et al. for identifying diseases and pests in tomato plant photos acquired at varying camera resolutions. Deep learning metaarchitectures, as well as multiple CNN object detectors, were utilized. Data expansion and local and global class annotation were utilized to boost training accuracy and decrease false positives. A large-scale tomato disease dataset was used for end-to-end training and testing. The system correctly detected nine different pests and diseases from the complicated scenarios

G. A Deep convolutional neural networks for mobile capture device-based crop disease classification in the wild. Comput Electron Agric; Picon A, Alvarez-Gila A, Seitz M, Ortiz-Barredo A, Echazarra J, Johannes A.

The article presents a study on the use of deep convolutional neural networks (CNNs) for crop disease classification using images captured by mobile devices in the field. The authors developed a CNN-based model called "DeepPlantPathologist" that can automatically classify crop diseases based on images of leaves captured in the field.

The authors trained their model on a large dataset of crop images and achieved high accuracy in disease classification across multiple crop types. They also demonstrated the potential for their model to be used in the field with mobile devices, allowing for real-time disease detection and diagnosis.

Overall, this article demonstrates the potential for deep CNNs to revolutionize crop disease management by providing an efficient and accurate tool for disease detection and diagnosis in the field. This technology could ultimately lead to improved crop yields and reduced economic losses for farmers.

This chapter represented the literature survey of traditional plant disease detection approaches based on computer vision technologies are commonly utilized to extract the texture, shape, colour, and other features of disease spots. In the chapter 3, will presents a detailed description of the dataset used in this study on tomato leaf disease detection using CNN, including dataset collection, preprocessing, datset statistics and dataset split for train, valid and test datasets.

CHAPTER – 3

METHODOLOGY

This chapter presents a detailed description of the dataset used in this study on tomato leaf disease detection using CNN, including dataset collection, preprocessing, datset statistics and dataset split for train, valid and test datasets.

3.1 Methodology Flow chart

The following figure shows the methodology flow chart, it describes the way of approached to detect the tomato leaf diseases.

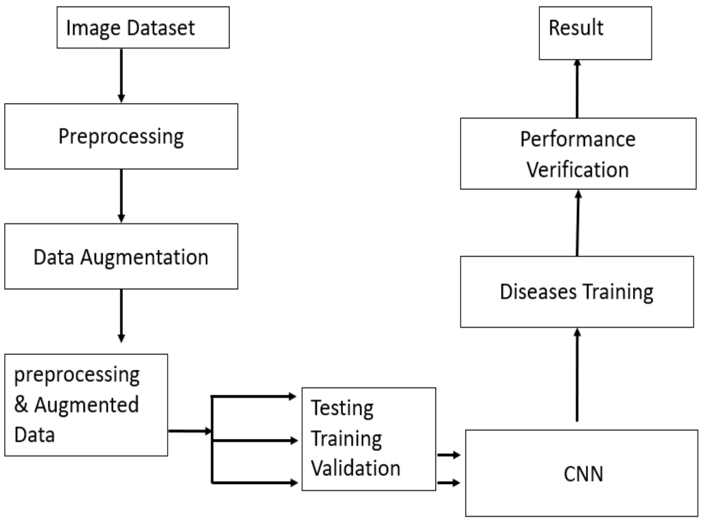


Figure 3.1 Methodology flow chart

3.2 Dataset Collection

The dataset used for this project was obtained from Kaggle. The dataset consists of 16,021 tomato leaf images with ten classes: Tomato_Bacterial_spot,Tomato_Early_blight,Tomato_Late_blight,Tomato_Leaf_Mold,Tomato_Septoria_leaf_spot,Tomato_Spider_mites_Two_spotted_spider _mite,Tomato__Target_Spot,Tomato__Tomato_YellowLeaf__Curl_Virus,Tomato__Tomato_mosaic_virus and Tomato_healthy. The images were captured from different locations, seasons, and under different lighting conditions.

3.3 Dataset Statistics

The dataset consists of 16,021 images with ten different classes representing different types of tomato leaf diseases and a healthy class. Table 1 presents the class distribution of the dataset.

Table 3.1 Class distribution of the dataset.

S.No	Type of tomato leaf disease	No. of images
1	Tomato_Bacterial_spot	2,127
2	Tomato_Early_blight	1,000
3	Tomato_Late_blight	1,909
4	Tomato_Leaf_Mold	952
5	Tomato_Septoria_leaf_spot	1,771
6	Tomato_Spider_mites_Two_spotted_spider_mite	1,676
7	Tomato__Target_Spot	1,404
8	Tomato__Tomato_YellowLeaf__Curl_Virus	3,209
9	Tomato__Tomato_mosaic_virus	373
10	Tomato_healthy	1,591

3.4 Preprocessing

Before training the CNN model, the dataset was preprocessed using several techniques such as data augmentation, normalization, and resizing. **Data Augmentation** is a very popular technique in image processing, especially computer vision to increase the diversity and amount of training data by applying random (but realistic) transformations. Data augmentation techniques such as rotation, flipping (horizontal and vertical), and random cropping were applied to increase the size of the dataset and introduce more variability in the data.

Normalization was applied to scale the pixel values between 0 and 255 to improve the convergence of the model during training.

Resizing was also applied to standardize the image size to 256x256 pixels to reduce the computational cost of training the mode and also send the images as batches as the batch size is 32.

3.5 Dataset Split

To evaluate the performance of the CNN model, the dataset was split into three subsets, namely the training dataset, validation dataset, and test dataset. The training dataset was used to train the model, the validation dataset was used to tune the hyperparameters, and the test dataset was used to evaluate the performance of the model on unseen data.

The dataset was split randomly into the three subsets, with the training dataset containing 80% of the images, the validation dataset containing 10% of the images, and the test dataset containing 10% of the images. Table 2 shows the dataset lengths for the train, validation, and test datasets.

Table 3.2 Dataset lengths for train, valid and test.

Train dataset length	400
Valid dataset length	50
Test dataset length	51
Total dataset length	(400+50+51)=501

The dataset split ensures that the model is trained on a sufficiently large dataset while also allowing for a fair evaluation of its performance on unseen data.

This chapter described about the methodology of the system. In the next chapter, Chapter 4, presents the CNN model architecture and the training process. The dataset split presented in this chapter is used to train and evaluate the performance of the model.

CHAPTER – 4

CNN MODEL ARCHITECTURE AND TRAINING PROCESS

In this chapter, describes the Convolutional Neural Network (CNN) model architecture used for our image classification task and the process of training the model. The architecture includes multiple convolutional and pooling layers, followed by fully connected layers, and ends with a softmax output layer. The training process involves initializing the model parameters, defining the loss function, selecting an optimization algorithm, and iteratively updating the model parameters using backpropagation and gradient descent.

4.1 CNN Model Architecture

A Convolutional Neural Network (CNN) is a type of artificial neural network commonly used for image and video analysis, recognition, and processing. It is designed to automatically extract meaningful features from raw pixel data of an image, enabling it to recognize objects, faces, shapes, and patterns.

CNNs are inspired by the structure and function of the visual cortex in the brain. The network is made up of a series of interconnected layers, each consisting of several neurons that perform simple computations on the input data. The layers are typically arranged in a specific order, including convolutional layers, pooling layers, and fully connected layers. The following fig 2 shows the CNN model architecture with properly connected layers.

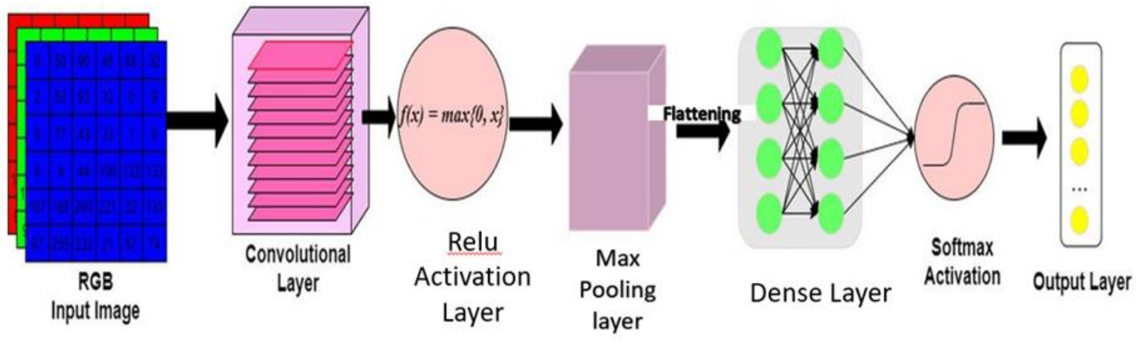


Figure 4.1 CNN model architecture

4.1.1 Convolution Layer

Convolutional layers are the core building blocks of a CNN. They apply filters or kernels to the input image, sliding over the entire image and performing a dot product between the filter and the input pixels. This process generates a feature map, highlighting the regions of the input image that are most important for recognizing a particular pattern or object. Fig 3 shows the mathematical operation kernel filter with input image.

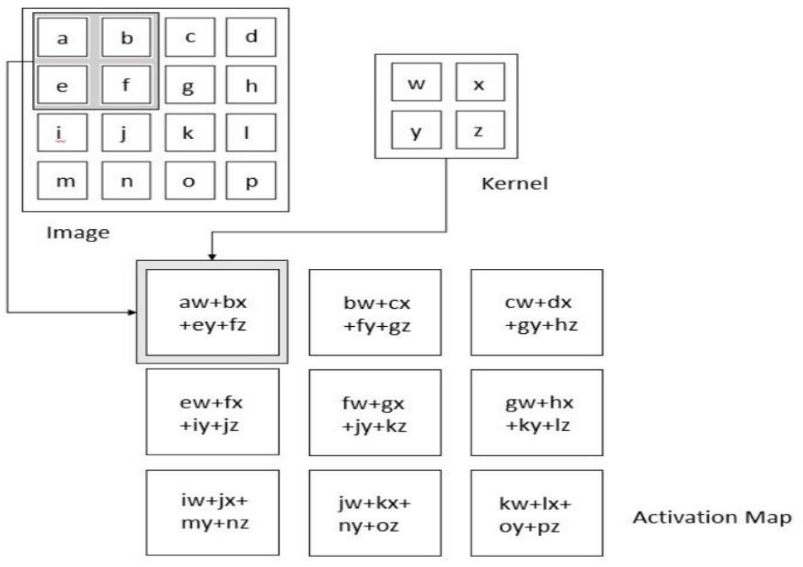


Figure 4.2 Convolution layer

Fig 3 illustrates the mathematical operation of the convolution layer, where a 2x2 kernel filter is convolved with an input image. The resulting feature map highlights the edges and corners of the object in the image.

4.1.2 Relu Activation Function

The Rectified Linear Unit (ReLU) activation function is a widely used activation function in CNNs. It introduces nonlinearity into the network and improves its ability to model complex relationships between the input and output data. The ReLU function only allows positive values to pass through the neuron. When the input to the neuron is positive, the output is equal to the input value, and when the input is negative, the output is equal to zero.

The ReLU function is defined as $f(x) = \max\{0,x\}$.

4.1.3 Pooling Layers

Pooling layers are an essential component of Convolutional Neural Networks (CNNs) used in computer vision applications. They are used to reduce the spatial dimensionality (width and height) of the input data while retaining its essential features.

Max pooling is a commonly used type of pooling layer in which the maximum value within a defined region of the input feature map is selected and then passed on to the next layer. The size of this defined region (often referred to as the pooling window or kernel size) is typically set by the user. Fig 4 shows the operation of max pooling.

For example, let's say we have an input feature map with a size of 6x6 and a pooling window size of 2x2. The max pooling operation would take place as follows:

1. The input feature map is divided into non-overlapping regions of size 2x2.
2. The maximum value within each region is identified.
3. A new feature map is created with a size of 3x3, consisting of the maximum values from each region.

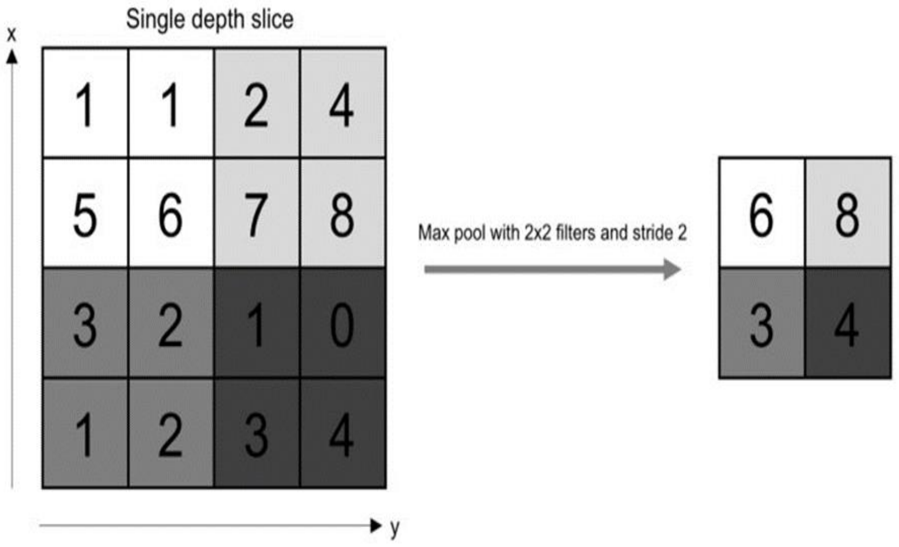


Figure 2.3 Max pooling layer

4.1.4 Fully Connected Layer

- **Fully connected input layer** – The preceding layers' output is "flattened" and turned into a single vector which is used as an input for the next stage.
- **The first fully connected layer** – adds weights to the inputs from the feature analysis to anticipate the proper label.
- **Fully connected output layer** – offers the probability for each label in the end.

Fig 5 shows the internal working of fully connected layer

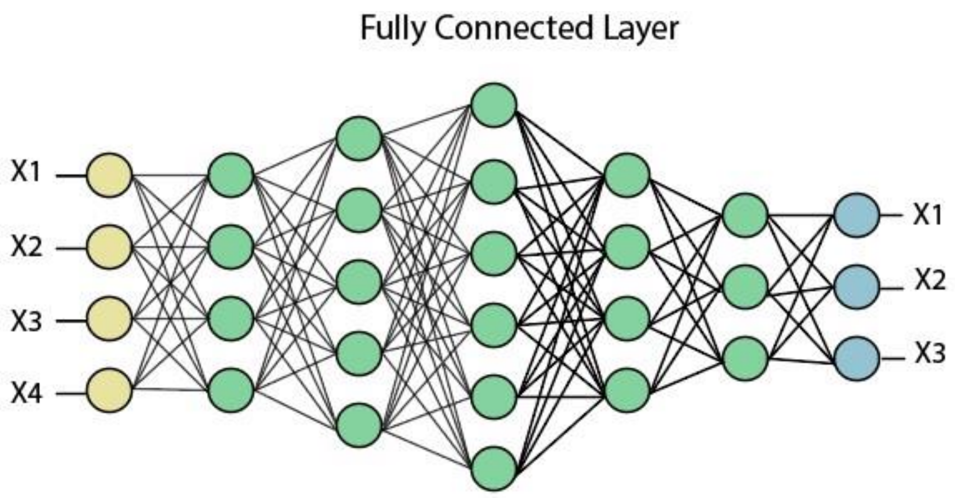
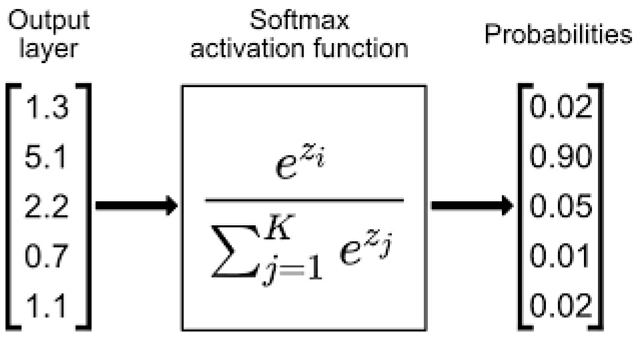


Figure 4.4 Fully connected layer

4.1.5 Softmax Activation Function

The softmax activation function is a mathematical function that is commonly used in artificial neural networks, particularly in multi-class classification problems. It is used to produce a probability distribution over the possible classes in the output layer of a neural network.


Output layer
Softmax activation function
Probabilities
1.3
5.1
2.2
0.7
1.1
$\frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$
0.02
0.90
0.05
0.01
0.02

Where Zi is the i-th element of the input vector and the sum (i.e. $\sum_{j=1}^{K} e^{z_j}$)is taken over all elements j in the input vector.

4.1.6 Sparse Categorical Cross entropy

The sparse categorical cross-entropy loss function calculates the cross-entropy loss between the predicted probability distribution and the target label tensor, taking into account the sparsity of the target label tensor. The formula for sparse categorical cross-entropy is:

$$CE = -\sum_{neuron=1}^{classes} y_{true_{neuron}} * \ln(y_{pred_{neuron}})$$

where y_true is the target label tensor, y_pred is the predicted output of the model, and the log and sum operations are performed over the predicted probability distribution for each sample in the batch, but only considering the element corresponding to the class index of the target label.

4.1.7 Adam Optimizer

Adam (Adaptive Moment Estimation) optimizer is a stochastic gradient descent optimization algorithm used to optimize the parameters of a neural network during training. In other terms Adam is an optimizer used to update the wights and biases of a neural network during training.

4.1.8 Summary of The CNN Model

The construction of a Convolutional Neural Network (CNN) model with multiple levels, each consisting of convolution layers and max pooling layers. The model has six levels with decreasing output shape reduction, and the input data is sent as batches with a batch size of 32. The convolution layers have 32, 64, 64, 64, 64, and 64 filters, respectively, with a kernel size of 3x3 and ReLU activation. Each convolution layer is followed by a max pooling layer with a pool size of 2x2. The output shape of each layer is calculated based on the input size, kernel size, padding, and stride. The output parameters of each convolution layer are calculated based on the kernel filter size, number of filters, and number of previous filters, while the output parameters of the dense layer are based on the input and output channel numbers. The overall CNN model data is summarized in figure 6

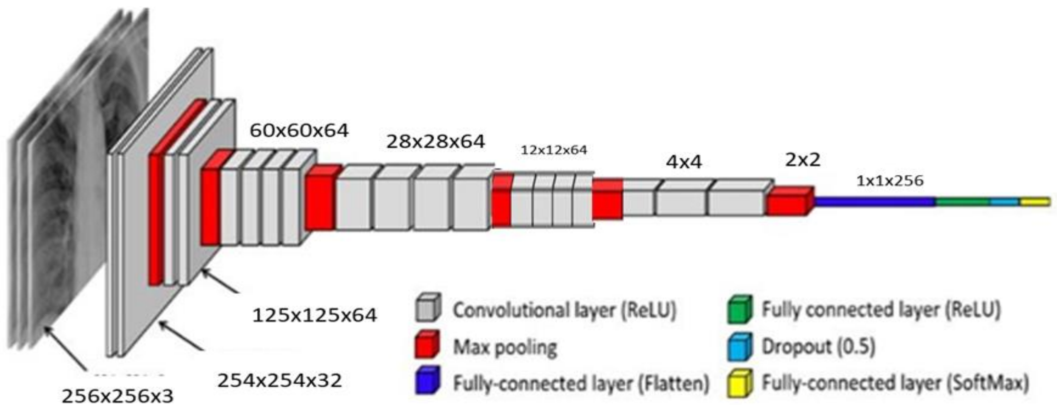
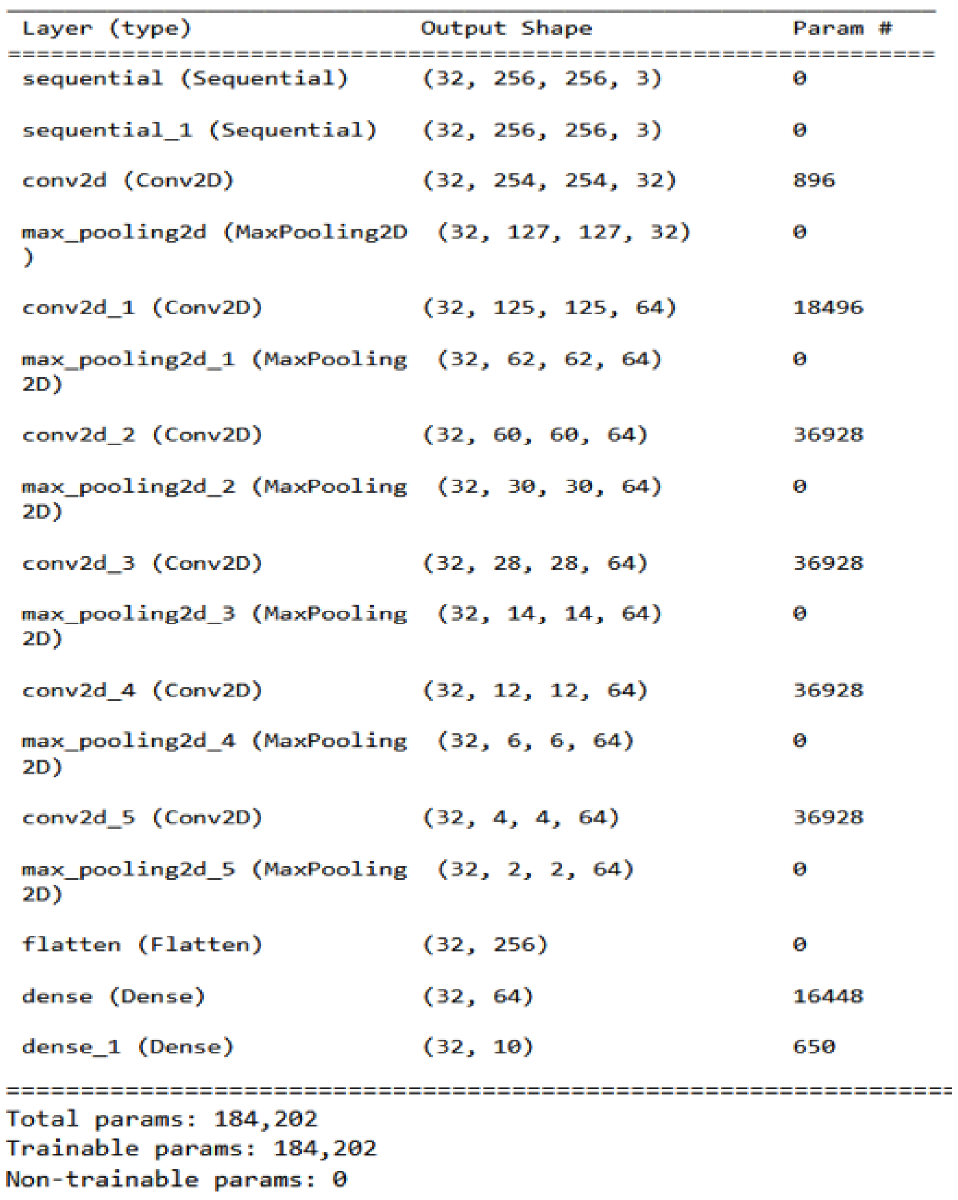

60x60x64
28x28x64
12x12x64
4x4
2x2
1x1x256
125x125x64
254x254x32
256x256x3
Convolutional layer (ReLU)
Max pooling
Fully-connected layer (Flatten)
Fully connected layer (ReLU)
Dropout (0.5)
Fully-connected layer (SoftMax)

Figure 4.5 Construction layers of CNN model at each level

The data is given format as following way are :

a. Convolutional layer: 32 filters, kernel size 3x3, ReLU activation
    i. Max pooling layer: pool size 2x2
b. Convolutional layer: 64 filters, kernel size 3x3, ReLU activation
    i. Max pooling layer: pool size 2x2
c. Convolutional layer: 64 filters, kernel size 3x3, ReLU activation
    i. Max pooling layer: pool size 2x2
d. Convolutional layer: 64 filters, kernel size 3x3, ReLU activation
    i. Max pooling layer: pool size 2x2
e. Convolutional layer: 64 filters, kernel size 3x3, ReLU activation
    i. Max pooling layer: pool size 2x2
f. Convolutional layer: 64 filters, kernel size 3x3, ReLU activation
    i. Max pooling layer: pool size 2x2

The following figure shows the summary of CNN model data of the otput shapes and respected parameters at each convolution layer and max pooling layers.



```
 Layer (type)                Output Shape              Param #
=================================================================
 sequential (Sequential)     (32, 256, 256, 3)         0

 sequential_1 (Sequential)   (32, 256, 256, 3)         0

 conv2d (Conv2D)             (32, 254, 254, 32)        896

 max_pooling2d (MaxPooling2D  (32, 127, 127, 32)       0
 )

 conv2d_1 (Conv2D)           (32, 125, 125, 64)        18496

 max_pooling2d_1 (MaxPooling  (32, 62, 62, 64)         0
 2D)

 conv2d_2 (Conv2D)           (32, 60, 60, 64)          36928

 max_pooling2d_2 (MaxPooling  (32, 30, 30, 64)         0
 2D)

 conv2d_3 (Conv2D)           (32, 28, 28, 64)          36928

 max_pooling2d_3 (MaxPooling  (32, 14, 14, 64)         0
 2D)

 conv2d_4 (Conv2D)           (32, 12, 12, 64)          36928

 max_pooling2d_4 (MaxPooling  (32, 6, 6, 64)           0
 2D)

 conv2d_5 (Conv2D)           (32, 4, 4, 64)            36928

 max_pooling2d_5 (MaxPooling  (32, 2, 2, 64)           0
 2D)

 flatten (Flatten)           (32, 256)                 0

 dense (Dense)               (32, 64)                  16448

 dense_1 (Dense)             (32, 10)                  650

=================================================================
Total params: 184,202
Trainable params: 184,202
Non-trainable params: 0
```

Figure 4.6 CNN model output shapes and parameters.

The images are sent as batches, the batch size is 32 and before fed to the algorithm. In the output shape just observe that first column represents batch size that means it contains 32 images.

Parameters

A. Output Shape reduction from one layer to other.

   There are two cases to determine the output shape of each layer :

   1. At the kernel filter to the convolution layer to convolution then just do,

      Output shape = Input(height or width) -kernel(height or width +1)

Input size is (i, i)

Kernel size is (k,k)

Output shape = i-k + 1

This one is applicable for padding is false and stride is 1

2. At max pooling then just take half of the size of the convolution layer.

   Convolution Layer size is (x,x).

   Output shape= floor (x/2)

B. Parameters Calculation

a. For first convolution layer

   Kernel filter size is (m, n)

   Output parameter = (m x n x channels +1)x no. of filters

b. For remaining layers

   Output parameters = (m x n x no. of previous filters+1)x no. of filters

c. For Dense layer

   Input Channel no is i.

   Output Channel no is j.

   Output parameters = j x (i+1)

4.2 Training Process

The training process involves initializing the model parameters, defining the loss function, selecting an optimization algorithm, and iteratively updating the model parameters using backpropagation and gradient descent. It discuss step-by-step process involved in training a neural network:

1. The first step in the training process is loading and preprocessing the training data. This involves normalizing the data, splitting it into batches, and converting it into the appropriate format for the model. This step is discussed in Chapter 3, section 3.3 and 3.5.
2. The second step in the training process is defining the model architecture. This step involves specifying the neural network's architecture, including the number and type of layers, activation functions, optimizer, and loss function. The architecture of the CNN model is discussed in section 4.1.

3. The third step in the training process is compiling the model. This step involves configuring the model for training by specifying the optimizer, loss function, and any additional metrics to track during training. The Adam optimizer and Sparse Categorical Cross entropy loss function are discussed in current chapter sub section 4.1.4 and sub section 4.1.5, respectively.
4. The final step in the training process is training the model. This step involves feeding the training data into the model, computing the output, and adjusting the model parameters using the Adam optimizer algorithm to minimize the loss function. The number of training epochs determines how many times the entire training dataset is used to train the model. The trained model is used for testing on the test set in the next chapter – 6 Results under the section 6.2 Training Results.

This chapter discussed CNN model architecture and the step-by-step process involved in training a neural network model, including loading and preprocessing the training data, defining the model architecture, compiling the model, and training the model. In the next chapter, Chapter 5, presents the algorithm process, implementation code for the automatic detection for tomato leaf disease detection.

CHAPTER – 5

SOFTWARE DESCRIPTION

In this chapter, describes the algorithm process and related python program code.

5.1 Algorithm Process

Step. 1 - Import the necessary libraries.

Step. 2 - Set the input parameters, such as image size, batch size, and number of classes**.**

Step. 3 – Load the dataset and preprocess the images

Step. 4 -Define the CNN model architecture.

Step. 5 - To train the CNN model at different epochs.

Step. 6 – Evaluate the performance of the model and save the model with .h5 format

Step. 7 - Reload the model and predict the tomato leaf images.

5.2 PYTHON Program Code

5.2.1 Main Code

```
import tensorflow as tf
from tensorflow.keras import models, layers
import matplotlib.pyplot as plt
import numpy as np
IMAGE_SIZE = 256
BATCH_SIZE = 32
CHANNELS = 3
EPOCHS = 50
dataset = tf.keras.preprocessing.image_dataset_from_directory(

r"C:\Users\KP\OneDrive\Desktop\pythonprojectdeep\Datasets\PlantVillag
e",
        shuffle = True,
        image_size = (IMAGE_SIZE,IMAGE_SIZE),
        batch_size = BATCH_SIZE
)
class_names = dataset.class_names
```

```
class_dict = {}
count  = 0
for names in class_names:
   class_dict[names] = count
    count = count + 1
class_dict
for image_batch, label_batch in dataset.take(1):
    print(image_batch.shape)
    print(label_batch.numpy())
for image_batch, label_batch in dataset.take(1):
    print(image_batch[0].numpy())
for image_batch, label_batch in dataset.take(1):
    print(image_batch[0].shape)
plt.figure(figsize=(20,20))
for image_batch, label_image in dataset.take(1):
    for i in range(12):
        ax = plt.subplot(3,4,i+1)
        plt.title(class_names[label_batch[i]])
        plt.imshow(image_batch[i].numpy().astype("uint8"))
        plt.axis("off")
def get_dataset_partition(ds, train_split = 0.8, val_split = 0.1, test_split = 0.1,
shuffle = True, shuffle_size =1000):
    ds_size = len(ds)
   if shuffle:
         ds = ds.shuffle(shuffle_size, seed=12)
    train_size = int(train_split*ds_size)
    val_size = int(val_split*ds_size)
    train_ds1 = ds.take(train_size)
    val_ds1 = ds.skip(train_size).take(val_size)
    test_ds1 = ds.skip(train_size).skip(val_size)
    return train_ds1, val_ds1, test_ds1
train_ds, val_ds, test_ds = get_dataset_partition(dataset)
train_ds=train_ds.cache().shuffle(1000).prefetch(buffer_size=tf.data.AUTOTU
NE)
val_ds=val_ds.cache().shuffle(1000).prefetch(buffer_size = tf.data.AUTOTUNE)
test_ds=test_ds.cache().shuffle(1000).prefetch(buffer_size                =
tf.data.AUTOTUNE)
```

```
resize_rescale = tf.keras.Sequential([
    layers.experimental.preprocessing.Resizing(IMAGE_SIZE, IMAGE_SIZE),
    layers.experimental.preprocessing.Rescaling(1.0/255)
])
data_augmentation = tf.keras.Sequential([
    layers.experimental.preprocessing.RandomFlip("horizontal_and_vertical"),
    layers.experimental.preprocessing.RandomRotation(0.2)
])
train_ds = train_ds.map(
    lambda x, y: (data_augmentation(x, training=True), y)
).prefetch(buffer_size=tf.data.AUTOTUNE)
input_shape = (BATCH_SIZE, IMAGE_SIZE, IMAGE_SIZE, CHANNELS)
n_classes = 10
model = models.Sequential([
    resize_rescale,
    data_augmentation,
    layers.Conv2D(32, (3,3), activation = 'relu', input_shape = input_shape),
    layers.MaxPooling2D((2,2)),
    layers.Conv2D(64, kernel_size = (3,3), activation = 'relu'),
    layers.MaxPooling2D((2,2)),
    layers.Conv2D(64, kernel_size = (3,3), activation = 'relu'),
    layers.MaxPooling2D((2,2)),
    layers.Conv2D(64, (3,3), activation = 'relu'),
    layers.MaxPooling2D((2,2)),
    layers.Conv2D(64, (3,3), activation = 'relu'),
    layers.MaxPooling2D((2,2)),
    layers.Conv2D(64, (3,3), activation = 'relu'),
    layers.MaxPooling2D((2,2)),
    layers.Flatten(),
    layers.Dense(64, activation = 'relu'),
    layers.Dense(n_classes, activation = 'softmax'),
])
model.build(input_shape = input_shape)
model.summary()
model.compile(
        optimizer = 'adam',
        loss = tf.keras.losses.SparseCategoricalCrossentropy(from_logits =
False),
        metrics = ['accuracy'] )
history = model.fit(
        train_ds,
        epochs = EPOCHS,
        batch_size = BATCH_SIZE,
        verbose = 1,
        validation_data = val_ds )
scores = model.evaluate(test_ds)
history.history.keys()
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']
plt.figure(figsize=(15,15))
plt.subplot(1,2,1)
plt.plot(range(EPOCHS), acc, label= 'Training Accuracy')
plt.plot(range(EPOCHS), val_acc, label = 'Validation Accuracy')
plt.legend(loc='lower right')
plt.title('Training and Validation Accuracy')
plt.subplot(1, 2, 2)
plt.plot(range(EPOCHS), loss, label = 'Training loss')
plt.plot(range(EPOCHS), val_loss, label = 'Validation loss')
plt.legend(loc = 'upper right')
plt.title('Training and Validation loss')
plt.show()
```

5.2.2 Prediction of an Image Code

```
%%writefile tomatoDL.py
import streamlit as st
import tensorflow as tf
from tensorflow.keras.preprocessing import image
from tensorflow.keras.models import load_model
import numpy as np
import matplotlib.pyplot as plt
from PIL import Image
```

```
st.set_option('deprecation.showfileUploaderEncoding', False)
st.title('Image classifier Deep learning')
st.text('Upload the Image')
class_names=['Tomato_Bacterial_spot',
 'Tomato_Early_blight',
 'Tomato_Late_blight',
 'Tomato_Leaf_Mold',
 'Tomato_Septoria_leaf_spot',
 'Tomato_Spider_mites_Two_spotted_spider_mite',
 'Tomato__Target_Spot',
 'Tomato__Tomato_YellowLeaf__Curl_Virus',
 'Tomato__Tomato_mosaic_virus',
 'Tomato_healthy']
reloaded_model = tf.keras.models.load_model('saved_model.h5')
uploaded_file = st.file_uploader("Choose an Image...",type='JPG')
if uploaded_file is not None:
    img=Image.open(uploaded_file)
    st.image(img,caption='Uploaded Image')
    if st.button('PREDICT'):
        st.write('Result..')
        test_image = image.img_to_array(img)
        test_image = np.expand_dims(test_image, axis = 0)
        prediction = reloaded_model.predict(test_image,batch_size=32)
        st.write("Predicted label : ", class_names[np.argmax(prediction[0])])
        confidence = round(100 * (np.max(prediction[0])), 2)
        st.write('Confidence : ',confidence)
! streamlit run tomatoDL.py
```

This code provides a basic framework for implementing the proposed algorithmic approach for tomato leaf disease detection using CNN. However, it may require modifications based on the specific dataset and requirements. In the next chapter 6 ,that will presents the visualization of the training results and predictive analysis obtained from the proposed CNN-based approach

CHAPTER – 6

RESULTS

In this chapter, that presents the visualization of the training results and predictive analysis obtained from the proposed CNN-based approach.

6.1 Visualization

The aim of this study is to detect the different types of tomato leaf diseases using convolutional neural networks (CNNs). The dataset used for this study is Plant Village, it taken from the Kaggle website, which contains ten types of tomato leaf images with labels indicating healthy and unhealthy leaves. Figure 8 shows the different types of tomato leaf images with their corresponding labels.

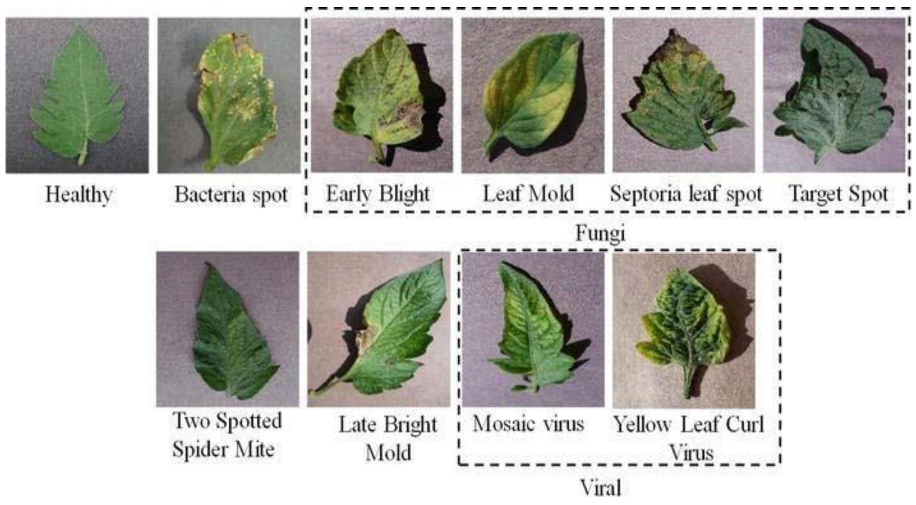


Figure 6.1 visualization of tomato leaf images

6.2 Training results

The training results for a CNN typically include the loss and accuracy metrics during the training process. The loss metric is a measure of how well the model is performing on the training data and is typically calculated as the difference between the predicted output and the actual output. The goal of the training process is to minimize the loss function. As discussed chapter 4, section 4.1 the concept of CNN model architecture and section 4.2 the training process.

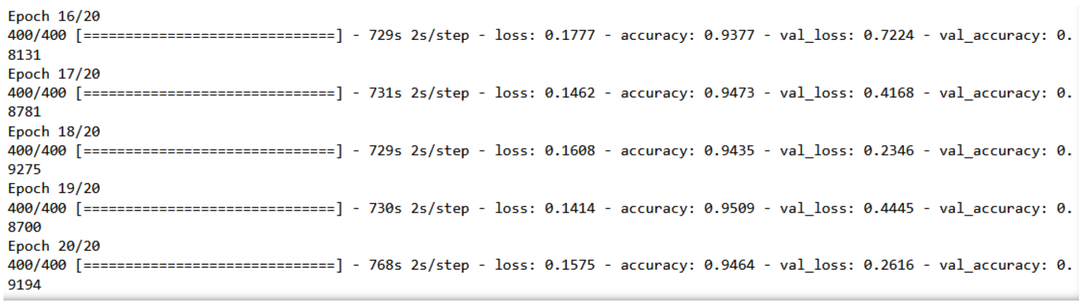
CNN model was trained at 10epochs, 20 epochs and 50epochs. The total dataset length is 501. As discussed chapter 3 Table 2 shows the dataset lengths for train, valid and test.

From the results shown in Table 3, it is observed that the accuracy increases and the loss decreases with a modification in the number of epochs.

Table 6.1 Comparison of Loss and Accuracy for train and valid at different epochs.

S. No	**No. of epochs**	**Train Accuracy**	**Train Loss**	**Valid Accuracy**	**Valid Loss**
1	10	0.64	0.34	0.61	0.45
2	20	0.94	0.15	0.91	0.26
3	50	0.97	0.08	0.95	0.14

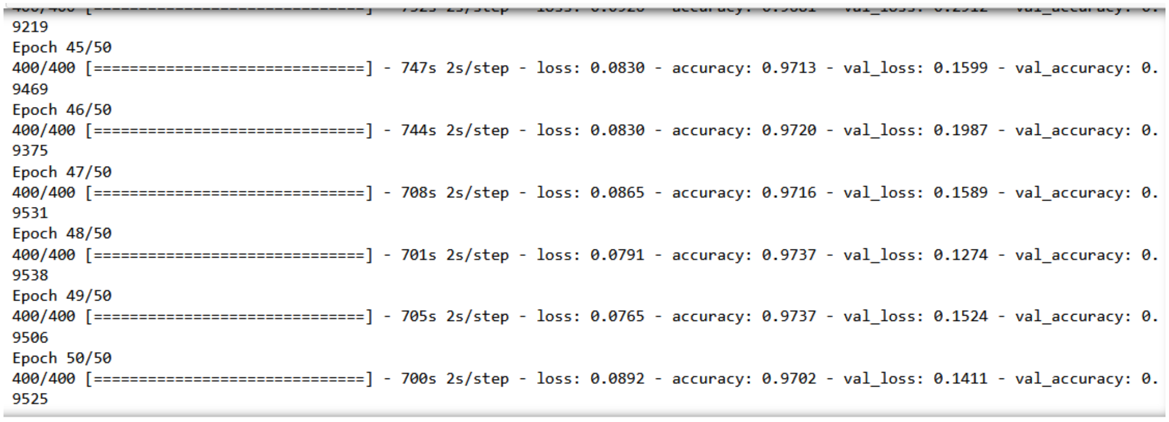
The following figure shows the last five epochs at the given 20 epochs running time, accuracy and loss, it is taken from the our implemented part of CNN model output screenshot.



```
Epoch 16/20
400/400 [==============================] - 729s 2s/step - loss: 0.1777 - accuracy: 0.9377 - val_loss: 0.7224 - val_accuracy: 0.
8131
Epoch 17/20
400/400 [==============================] - 731s 2s/step - loss: 0.1462 - accuracy: 0.9473 - val_loss: 0.4168 - val_accuracy: 0.
8781
Epoch 18/20
400/400 [==============================] - 729s 2s/step - loss: 0.1608 - accuracy: 0.9435 - val_loss: 0.2346 - val_accuracy: 0.
9275
Epoch 19/20
400/400 [==============================] - 730s 2s/step - loss: 0.1414 - accuracy: 0.9509 - val_loss: 0.4445 - val_accuracy: 0.
8700
Epoch 20/20
400/400 [==============================] - 768s 2s/step - loss: 0.1575 - accuracy: 0.9464 - val_loss: 0.2616 - val_accuracy: 0.
9194
```

Figure 6.2 Loss and Accuracy at 20 epochs

The following figure shows the last six epochs at the given 20 epochs running time, accuracy and loss, it is taken from the our implemented part of CNN model output screenshot



```
9219
Epoch 45/50
400/400 [==============================] - 747s 2s/step - loss: 0.0830 - accuracy: 0.9713 - val_loss: 0.1599 - val_accuracy: 0.
9469
Epoch 46/50
400/400 [==============================] - 744s 2s/step - loss: 0.0830 - accuracy: 0.9720 - val_loss: 0.1987 - val_accuracy: 0.
9375
Epoch 47/50
400/400 [==============================] - 708s 2s/step - loss: 0.0865 - accuracy: 0.9716 - val_loss: 0.1589 - val_accuracy: 0.
9531
Epoch 48/50
400/400 [==============================] - 701s 2s/step - loss: 0.0791 - accuracy: 0.9737 - val_loss: 0.1274 - val_accuracy: 0.
9538
Epoch 49/50
400/400 [==============================] - 705s 2s/step - loss: 0.0765 - accuracy: 0.9737 - val_loss: 0.1524 - val_accuracy: 0.
9506
Epoch 50/50
400/400 [==============================] - 700s 2s/step - loss: 0.0892 - accuracy: 0.9702 - val_loss: 0.1411 - val_accuracy: 0.
9525
```

Figure 6.3 Loss and Accuracy at 50 epochs

Evaluate the performance of the model on test dataset. The below table shows the obtained accuaracy and loss at different epochs

Table 6.2 Comparison of Loss and Accuracy for test at different epochs.

S.No	**No. of Epochs**	**Train Accuracy**	**Loss Accuracy**
1	10	0.60	0.38
2	20	0.90	0.28
3	50	o.96	0.12

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

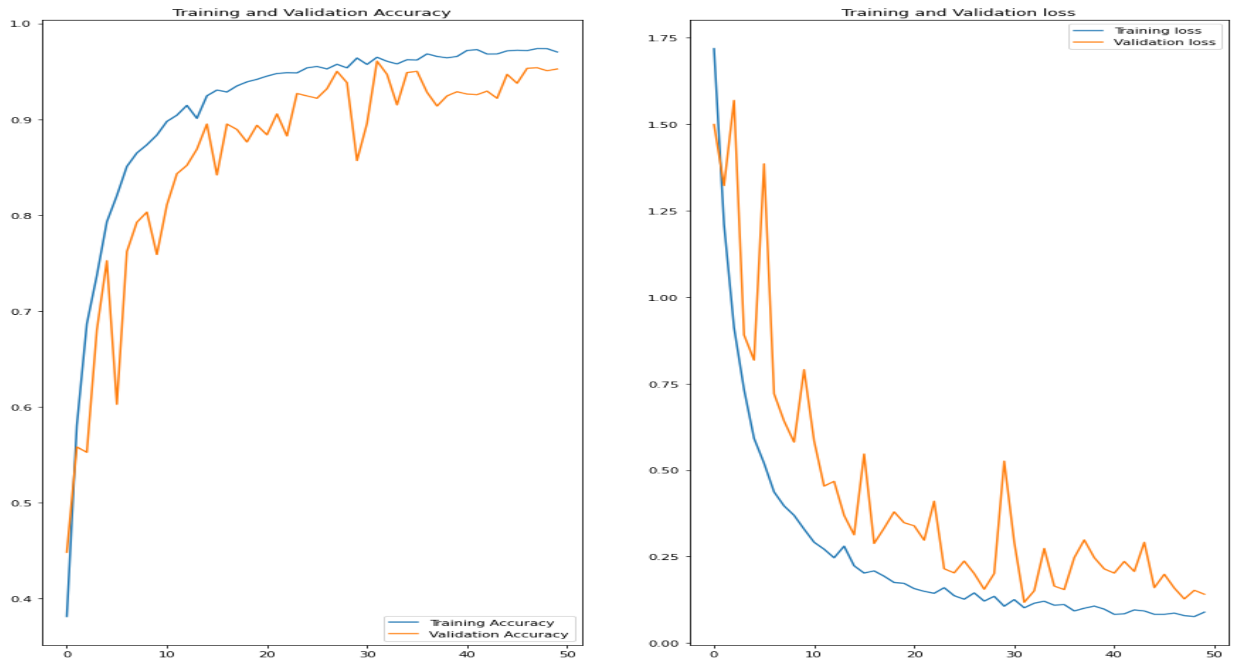


Figure 6.4 Plotted the accuracy and loss for both train and validation.

The above Training and Validation Accuracy graph shows that with the increase in the Training accuracy there is a increase in the Validation accuracy. From the Training and the Validation loss graph, it shows that with the decrease in the training loss there is a decrease in the validation loss.

6.3 Predicted Images

Actually, the trained CNN model saved with saved_model.h5, at the time of predicting an image reload it and do predictions. In this project, the interface designed a dynamic path for image predictions using a web interface page, a web page with an interface that allows end-users to select images of their choice and perform predictions on them. The implementation of the interface page is kept in one file named 'tomatoDL.py'. This file contains the necessary code to run the web interface page and process the selected images for predictions. The following command run on jupyter notebook then open the web page on by default browser and end user select images themself an-d do predictions. The implementation details of this process can be found in Chapter 5, Section 5.2.2. prediction of an image.

```
!streamlit run tomatoDL.py
```

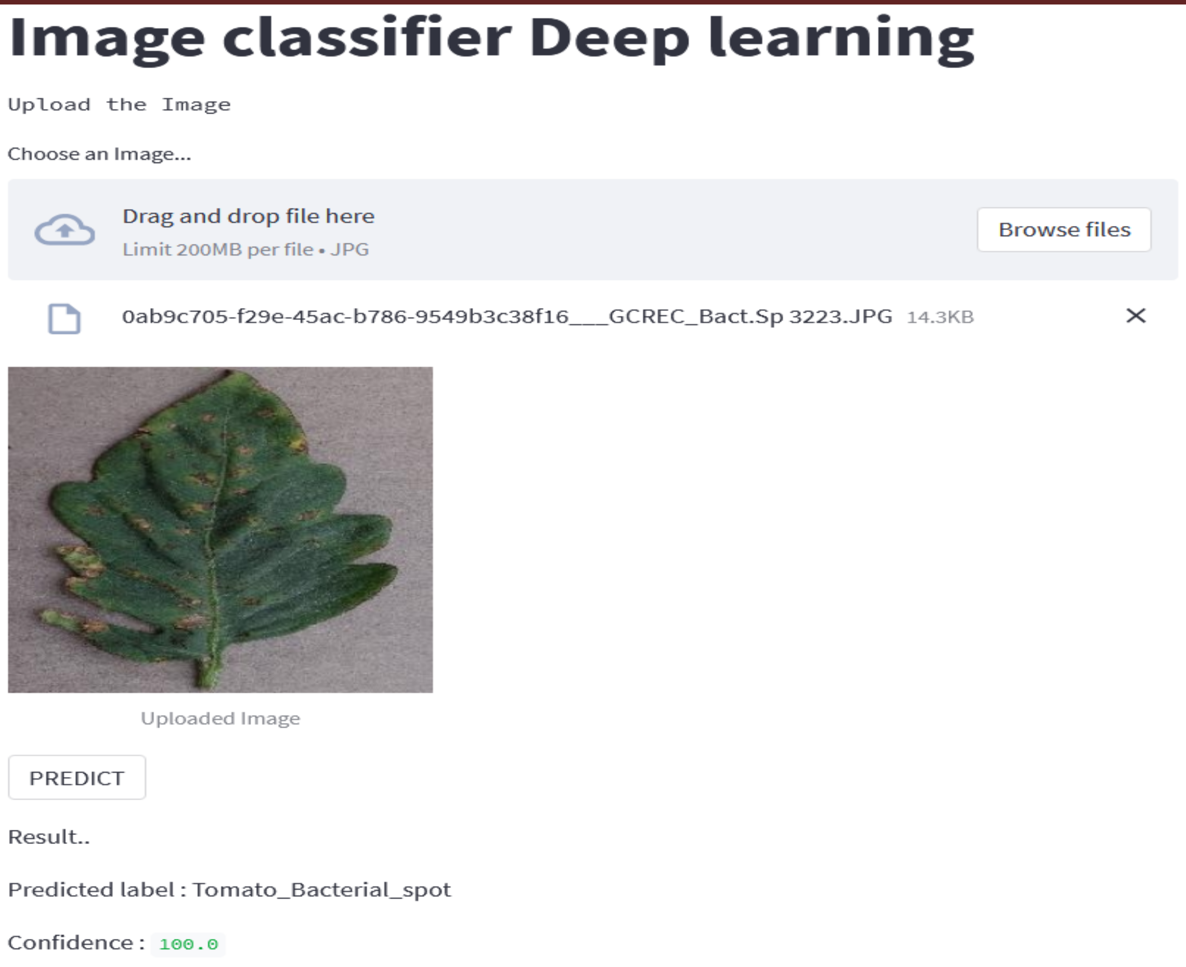


Image classifier Deep learning

Upload the Image

Choose an Image...

Drag and drop file here
Limit 200MB per file • JPG

Browse files

0ab9c705-f29e-45ac-b786-9549b3c38f16___GCREC_Bact.Sp 3223.JPG 14.3KB

Uploaded Image

PREDICT

Result..

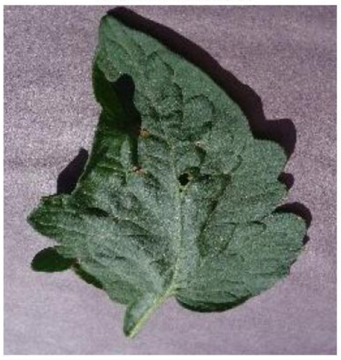
Predicted label : Tomato_Bacterial_spot

Confidence : 100.0

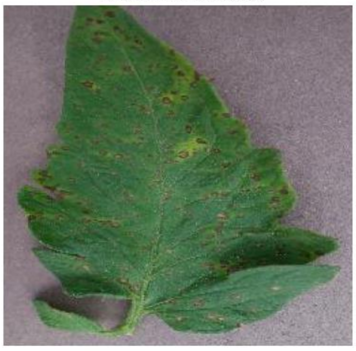
Different types of tomato leaf images predicted with predicted label and confidence

In a tomato leaf disease detection system using CNN, there are different types of tomato leaf images that can be predicted based on the type of disease present in the leaves. Some common types of tomato leaf images that can be predicted using a CNN include, as discussed in chapter 4 the concept of CNN model and training process.

Predicted : Tomato__Target_Spot,
Confidence : 99.82%

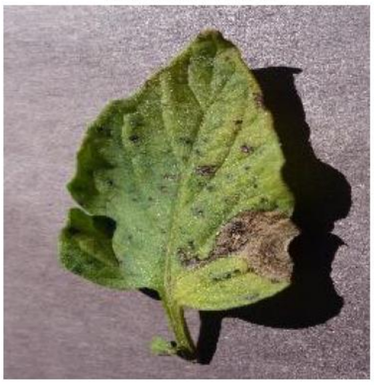


Predicted : Tomato_Septoria_leaf_spot,
Confidence : 100.0%

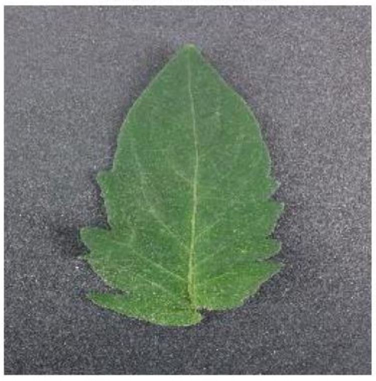


Predicted : Tomato__Tomato_YellowLeaf__Curl_Virus,
Confidence : 99.87%

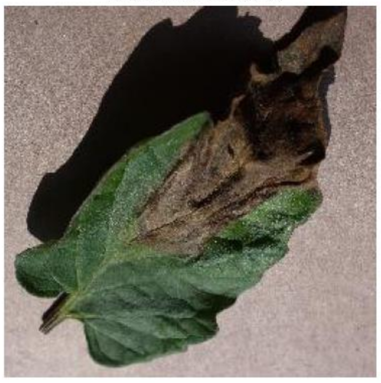
Predicted : Tomato_Early_blight,
Confidence : 99.97%



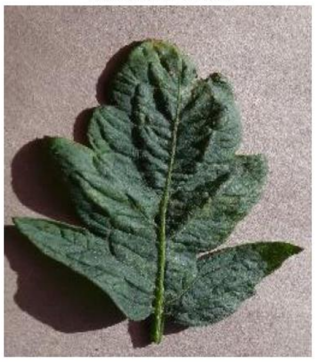
Predicted : Tomato_healthy,
Confidence : 99.99%



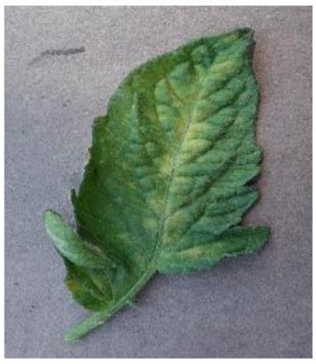
Predicted : Tomato_Late_blight,
Confidence : 99.52%



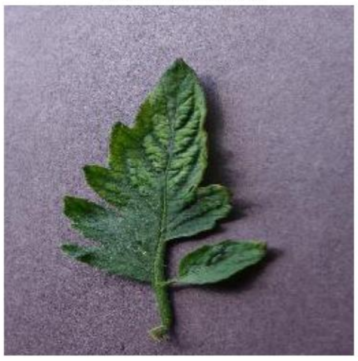
Predicted : Tomato_Spider_mites_Two_spotted_spider_mite,
Confidence : 100.0%



Predicted : Tomato_Leaf_Mold,
Confidence : 99.89%



Predicted : Tomato__Tomato_mosaic_virus,
Confidence : 99.77%



CHAPTER - 7

CONCLUSION AND FUTURE SCOPE

7.1 Conclusion

In this project, we have presented a approach for tomato leaf disease detection using convolutional neural networks (CNN). We trained a deep learning model using a dataset of tomato leaf images, which was collected from various sources. The trained model was able to accurately detect the presence of ten common tomato leaf diseases, namely, bacterial spot, early blight, late blight, leaf mold, spectorial leaf spot, spider mites two spotted spider mite, target spot, yello leaf curl virus, mosaic virus and healthy. The proposed system is designed to provide an easy-to-use and efficient solution for detecting tomato leaf diseases. It uses a web interface page that allows end-users to upload images of tomato leaves and get real-time predictions on the presence of diseases. The system is capable of processing a large number of images quickly, making it ideal for use in agricultural applications.

7.2 Future Scope

Tomato leaf disease detection using CNN has great potential for future applications. Here are some possible future scopes for this technology:

- Real-time disease detection: The current project used pre-captured images of tomato leaves for disease detection. In the future, the system can be designed to detect diseases in real-time using a camera attached to a robotic arm that moves around the tomato plants. This would enable early detection and treatment of diseases, thus improving crop yields and reducing losses.
- Transfer learning: The current project used a CNN model. In the future, transfer learning can be used to improve the accuracy of the model. This would involve using pre-trained CNN models that have been

trained on a large dataset and fine-tuning them on the tomato leaf disease dataset.

- Deployment on mobile devices: The current project was implemented on a desktop computer. In the future, the system can be optimized for deployment on mobile devices such as smartphones and tablets. This would enable farmers to use the system in the field for real-time disease detection and treatment.

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] Aravind KR, Raja P, Anirudh R. Tomato crop disease classification Using A Pre-Trained Deep Learning Algorithm, Procedia Comput Sci. 2018; 133:1040–7.

[3] Karthik R, Hariharan M, Anand Sundar, Mathikshara Priyanka, Johnson Annie, Menaka R. Attention embedded residual CNN for disease detection in tomato leaves. Applied Soft Comput. 2020.

[4] Zhong Yong, Zhao Ming. Research on deep learning in apple leaf disease recognition. Comput Electron Agric. 2020; 168:105146.

[5] Selvaraj MG, Vergara A, Ruiz H, et al. AI-powered banana diseases and pest detection. Plant Methods. 2019; 15:92.

[6] Girshick, R., J. Donahue, T. Darrell, and J. Malik. "Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation." CVPR '14 Proceedings of the 2014 IEEE Conference on Computer Vision and Pattern Recognition. Pages 580-587. 2014.

[7] Fuentes A, Yoon S, Kim SC, Park DS. A robust deep-learning-based detector for real-time tomato plant diseases and pest’s recognition. Sensors. 2022; 2017:17.

[8] Picon A, Alvarez-Gila A, Seitz M, Ortiz-Barredo A, Echazarra J, Johannes A. Deep convolutional neural networks for mobile capture device-based crop disease classification in the wild. Comput Electron Agric. 2019;1(161):280–90.