DETECTION AND CLASSIFICATION OF TOMATO CROP DISEASE USING DEEP LEARNING MODELS WITH VARIED OPTIMIZATION TECHNIQUES

A Thesis submitted in partial fulfilment of the

requirement for the award of degree of

MASTER OF ENGINEERING

in

INFORMATION TECHNOLOGY

Submitted By:

Abhishek Rana

(Roll Number: 21-801)

under the guidance of

Dr. Neelam Goel

Assistant Professor

Department of Information Technology

Department of Information Technology

University Institute of Engineering and Technology

Panjab University, Chandigarh – 160014, INDIA

2024

CERTIFICATE

I hereby certify that the work which is being submitted in this thesis titled, **“DETECTION AND CLASSIFICATION OF TOMATO CROP DISEASE USING DEEP LEARNING MODELS WITH VARIED OPTIMIZATION TECHNIQUES**”, in partial fulfillment of the requirement for the award of the degree of ‘Master of Engineering in Information Technology submitted in UIET, Panjab University, Chandigarh, is an authentic record of my work carried out under the supervision of Dr. Neelam Goel, Assistant Professor, Department of Information Technology from University Institute of Engineering and Technology, Panjab University, Chandigarh.

The matter presented in this thesis has not been submitted for the award of any other degree at this or any other university.

Abhishek Rana

M.E(IT)

Roll No: 21-801

This is to certify that the statements made above by the candidate are correct and true to the best of our knowledge.

Supervisor

Dr. Neelam Goel

Assistant professor (IT), UIET

Panjab University, Chandigarh

Coordinator	**Director**
Dr. Amandeep Verma	**Prof. Sanjeev Puri**
Department of IT	U.I.E.T
UIET, Panjab University	Panjab University
Chandigarh-160014	Chandigarh-160014

ACKNOWLEDGEMENT

It gives me immense pleasure to express my heartfelt gratitude and sincere appreciation to my highly respected and esteemed mentor Dr. Neelam Goel (Assistant Professor, UIET) for her guidelines, encouraging words, constructive feedback, constant monitoring, and valuable suggestions, and ever available unconditional support during the entire course of this dissertation. Through her immense experience and invaluable guidance, she helped me in completing this challenge. I gratefully applaud her helpful suggestions and cooperative behavior during this project.

I also extend my sincere thanks to the rest of the Information faculty and staff membersof the Department of Information Technology, University Institute of Engineering and Technology, PU, Chandigarh, for their intellectual support.

I also wish to express my gratitude to my family and friends who provided me with constant moral support and encouragement throughout my study. In the end, I would like to express my sincere thanks to all my friends and others who helped me directly or indirectly during this project work.

Finally, I bow to Almighty, who gave me the strength and perseverance to carry out this work with dedication and sincerity.

ABHISHEK RANA

Roll No. 21-801

M.E (IT) UIET (PU), CHD.

ABSTRACT

Tomato leaf diseases can severely impact crop yield and quality, resulting in substantial economic losses for farmers. Effective management of these diseases relies on early detection and accurate classification, which are crucial for mitigating their adverse effects. This thesis focuses on classifying tomato leaf diseases using advanced deep learning models, including Convolutional Neural Networks (CNN), ResNet-50, InceptionV3, and EfficientNetB0. The study utilizes the Plant Village database to train and evaluate these models, with a particular emphasis on comparing their performance using two different optimizers, Adam and Nadam. EfficientNetB0 stands out among the models, achieving an impressive accuracy of 99%. Its superior performance in this classification task is noteworthy, especially given its low computational requirements. This highlights EfficientNetB0's efficiency not only in terms of accuracy but also in reducing the computational burden, making it a highly effective tool for practical applications in agriculture. The study also explores the role of transfer learning, a technique that uses pre-trained knowledge from large datasets to enhance model performance. By applying transfer learning, the study achieved significant improvements in both accuracy and training efficiency, demonstrating the practicality of this approach for real-world scenarios. Furthermore, the comparative analysis underscores the varying capabilities of the different models, with EfficientNetB0 consistently outperforming the others in terms of both accuracy and computational efficiency. The research findings suggest that deep learning, particularly when combined with transfer learning, offers a robust solution for the early detection and classification of tomato leaf diseases. These insights are not only valuable for improving crop management practices but also for minimizing the economic impact of plant diseases on the agricultural sector. Ultimately, the study provides practical, data-driven solutions that can significantly aid farmers in managing and mitigating the effects of tomato leaf diseases.

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

1D	One Dimensional
2D	Two Dimensional
3D	Three Dimensional
AI	Artificial Intelligence
ABCK	Ant Bee Colony Algorithm
Adam	Adaptive Moment Estimation
ANN	Artificial Neural Network
B-ARNet	Both-channel Residual Attention Network
BWTR	Binary Wavelet Transform combined with Retinex
CL	Convolutional Layer
CNN	Convolutional Neural Networks
DAE	Deep Auto-Encoder
DBN	Deep Belief Network

DBN	Deep Boltzmann Network
DCNN	Deep Convolutional Neural Network
DL	Deep Learning
FC	Fully Connected
FN	False Negative
FP	False Positive
FLOPs	Floating Point Operation per second
GDP	Gross Domestic Product
KSW	Kernels and Sparse Weights
MBConv	Mobile Inverted Bottleneck Convolution
ML	Machine Learning
Nadam	Nesterov-accelerated Adaptive Moment Estimation
NLP	Natural Language Processing

RF	Random Forest
RBF	Radial Basis Function
ReLU	Rectified Linear Unit
ResNet	Residual Neural Network
RMSProp	Root Mean Square Propagation
RNN	Recurrent Neural Networks
SE-ResNet	Squeeze-and-Excitation Residual Neural Network
SGD	Stochastic Gradient Descent
SMOTE	Synthetic Minority Oversampling Technique
TN	True Negative
TP	True Positive
ToMV	Tomato Mosaic Virus
TYCLV	Tomato Yellow Leaf Curl Virus
VGG	Visual Geometry Group

TABLE OF CONTENTS

CHAPTER 1
INTRODUCTION AND LITERATURE REVIEW

This chapter gives an introduction to the research work. The first section discusses the motivation of the research. In the next section related work is discussed followed by research gaps. After that, the research problem and objectives are presented. The thesis organization is given in the last section.

1.1 PROLOGUE

In the context of agricultural imperative, the automated detection of plant diseases via leaf analysis is of utmost importance. Additionally, prompt identification of these diseases enhances both agricultural productivity and quality [1]. In various agricultural economies, farmers endure significant monetary losses each year owing to crop diseases [2]. India, primarily reliant on agriculture, considers it a substantial component of its Gross Domestic Product (GDP). Agriculture contributes 15% to India's GDP and accounts for 13% of its total exports. Roughly 65% of India's population relies on agriculture for their sustenance, either directly or indirectly [3].

Tomatoes are pivotal due to their high market demand and nutritional value. The antioxidants present in tomatoes are essential for the overall health of a human being. Nevertheless, the cultivation of this esteemed commodity is vulnerable to interference by insects and pests, causing various diseases in tomato plants. Farmers need extensive knowledge to combat these diseases manually. Each year, farmers face multiple challenges in their endeavour to grow resilient crops. The infestation of insects and pests disrupts the production process, leading to decreased output. This presents a considerable economic challenge, especially affecting our farming sector.

Despite relying on pesticides for safeguarding tomato plants, farmers frequently lack disease expertise. Excessive pesticide usage carries health hazards and risks crop damage from misdiagnosis. Manual disease diagnosis, especially in remote regions, is laborious and demanding, leading to inaccurate prognoses and ineffective prevention strategies. Introducing a machine-driven system can promptly detect diseased tomato plants, identify the disease type, and enhance crop yield, minimizing losses effectively.

1.2 RELATED WORK

A great amount of contribution has been made in the field of Tomato plant disease classification using Machine and Deep learning in the past decade. Various techniques have been proposed in the literature to classify different types of diseases affecting tomato plant using Machine and Deep Learning models.

The following section briefly reviews the various machine and deep learning-based tomato plant disease classification models.

Chen et al. (2022)

Chen et al. utilized AlexNet to classify tomato leaf diseases and implemented the model. A dataset consisting of 18345 training samples and 4585 testing samples across ten classes, including one healthy class, was employed. By using the Adam optimizer, a top accuracy of 98.00% was achieved. The trained model was subsequently deployed in a mobile application for identifying tomato leaf diseases [4].

Sakkarvarthi et al. (2022)

Sakkarvarthi et al. implemented a CNN model for detecting and classifying tomato leaf diseases. Inside the model, two convolutional and two pooling layers were present. Proposed model outperformed the other techniques used namely InceptionV3, ResNet 152, and VGG19 with training accuracy of 98% [5].

Mia et al. (2021)

Mia et al. utilized CNN-based transfer learning and traditional machine learning techniques to compare their performance in classifying cucumber diseases. The study involved 525 image samples across six disease classes, which were preprocessed and augmented to a total of 4200 images. Various machine learning algorithms were then applied, along with transfer-learning models VGG-16, InceptionV3, and MobileNetV2. Among all methods, the CNN-based MobileNetV2 achieved the highest accuracy of 93.23% [6].

Thangaraj et al. (2021)

Thangaraj et al. utilized a transfer learning-based deep neural network model to identify nine classes of tomato leaf diseases and one class of healthy leaves. They evaluated the impact of different optimizers, including SGD, Adam, and RMSprop, on the model's performance. Among these, the Modified-Xception model achieved the highest accuracy of 99.55% when using the Adam optimizer [7].

Zhao et al. (2021)

Zhao et al. conducted tomato leaf disease classification using a deep convolutional neural network (DCNN) that incorporates residual blocks and attention extraction modules. The dataset comprises ten classes, including one healthy class, with an initial sample size of 4585, which was augmented to 22925 images. Among the models tested, SE-ResNet-50 achieved the highest accuracy of 96.81%, outperforming ResNet-50 [8].

Biswas et al. (2021)

Biswas et al. employed multiple machine learning classifiers for carrot disease classification, using segmentation-based feature extraction and ranking techniques. The study utilized a dataset of 599 carrot images, enhancing contrast with histogram equalization and performing further segmentation with K-means clustering. To handle imbalanced data, authors used Synthetic Minority Oversampling Technique (SMOTE). Random Forest (RF) achieved the highest accuracy of 94.17% by utilizing the top nine ranking features [9].

Ahmad et al. (2020)

Ahmad et al. utilized two datasets of tomato leaf disease images: one from a laboratory setting and another self-collected from the field. Employing feature extraction and parameter tuning on ResNet, VGG-19, VGG-16, and InceptionV3 models, authors observed performance variations ranging from 10.00% to 15.00% across various metrics. Among these variances, InceptionV3 with parameter tuning achieved the highest accuracy of 99.60% using the laboratory-based dataset [10].

Agarwal et al. (2020)

Agarwal et al. developed a simplified yet effective CNN model consisting of three convolutional layers, three max-pooling layers, and two fully connected layers for classification of tomato leaves. Data augmentation techniques were applied to generate new sample images, balancing the image quantity across all ten classes. When compared to VGG-16, MobileNet, and InceptionV3, the proposed model achieved an improved accuracy of 91.20% [11].

Basavaiah et al. (2020)

Basavaiah et al. utilized a dataset comprising five classes of tomato leaves and extracted features including Hu moments, color histograms, local binary patterns, and Haralick texture features for classification. They applied various machine learning algorithms, with Random Forest (RF) achieving the highest accuracy of 94.00%. However, the proposed method is not fully automatic and covers only a limited number of tomato leaf diseases [12].

Chen et al. (2020)

Chen et al. performed disease classification on tomato leaves using a combination of B-ARNet and ABCK-BWTR. Images from five classes were denoised and enhanced using the Binary Wavelet Transform combined with Retinex (BWTR). Background separation was conducted through KSW optimization using the Artificial Bee Colony Algorithm (ABCK). The classification was carried out with the Both-channel Residual Attention Network (B-ARNet), achieving an accuracy of 89.00% [13].

Kannan et al. (2020)

Kannan E et al. employed a pre-trained ResNet-50 model to classify tomato leaf diseases across six classes, utilizing a dataset of 12206 images. Data augmentation techniques were applied to further increase the number of images. By using ResNet-50, author achieved an accuracy of 97.00% [14].

Kartik et al. (2020)

Karthik et al. utilized two deep learning architectures on the PlantVillage dataset to detect three tomato plant diseases: early blight, late blight, and leaf mold. The first architecture employed a feed-forward CNN with residual learning, while the second combined CNN with both attention mechanisms and residual learning. The attention-based residual CNN architecture achieved the highest accuracy of 98.00% [15].

Elhassouny and Smarandache (2019)

Elhassouny and Smarandache developed a CNN-based model for identifying tomato leaf diseases, which was deployed on a mobile application. Authors used the MobileNet model to classify nine common diseases and one healthy class. The model was trained on a dataset of 7176 images, achieving an accuracy of 90.30% [16].

Widiyanto et al. (2019)

Widiyanto et al. developed a CNN model for the detection of four diseases in tomato plants: Late blight, Septoria leaf spot, Mosaic virus, and Yellow leaf curl virus, as well as healthy leaves. Authors trained the model using 1000 images per class sourced from the PlantVillage dataset. The model attained an impressive overall classification accuracy of 96.6% across the five classes [17].

TM et al. (2018)

TM et al. enhanced the LeNet architecture for classifying tomato leaf diseases by incorporating an additional block comprising of convolution layers, activation layers, and pooling layers. To speed up the training process, they resized the dataset images to 60 × 60 pixels, achieving an accuracy of 94.00–95.00% [18].

The results of earlier investigations, which are described in table 1.1 with the information like database, classification model, and classification with accuracy, indicate the challenges of multi-class classification of Tomato Plant Disease.

Table 1.1 Summary of the research work

Reference	Year	Database	Model	Classification	Accuracy (%)
[4]	2022	Plant Village	AlexNet	10-way	98.00
[5]	2022	Plant Village	CNN	10-way	98.00
[7]	2021	Plant Village	Modified Xception	10-way	99.55
[8]	2021	Plant Village	SE-Resnet-50	10-way	96.81
[10]	2020	Plant Village+ Self-Collected	InceptionV3	4-way	99.60
[11]	2020	Plant Village	CNN	10-way	91.20
[12]	2020	Plant Village	RF	5-way	94.00
[13]	2020	Self-Collected	B-ARNet + ABCK-BWTR	5-way	89.00
[14]	2020	Plant Village	ResNet-50	6-way	97.00
[15]	2020	Plant Village	Attention-based residual CNN	3-way	98.00
[16]	2019	Plant Village	MobileNet	10-way	90.30
[17]	2019	Plant Village	CNN	5-way	96.60
[18]	2018	Plant Village	LeNet	10-way	94.00-95.00

1.3 RESEARCH GAPS

The research gaps are identified from the literature review are mentioned below:

i) Existing datasets for tomato crop diseases are often small and lack diversity, limiting model generalization. Research is needed to expand these datasets or develop effective data augmentation methods.
ii) Deep learning models typically require significant computational resources, hindering real-time application in field conditions. Investigating optimization techniques for resource-constrained devices remains a key gap.
iii) Deep learning models for crop disease detection are often seen as "black boxes." Enhancing model explainability and interpretability is essential for gaining user trust and adoption.
iv) Current models frequently focus on a limited number of diseases. There is a need for research on models capable of accurately detecting and classifying multiple diseases simultaneously, even with overlapping symptoms.

1.4 PROBLEM STATEMENT

The detection and classification of tomato crop diseases are critical for ensuring agricultural productivity and food security. However, traditional methods of disease identification, which often rely on manual inspection by experts, are time-consuming, labour-intensive, and prone to human error. The advent of deep learning models offers a promising solution, yet challenges remain in optimizing these models for accuracy and efficiency. This thesis addresses the problem by leveraging deep learning models, specifically convolutional neural networks (CNNs), to detect and classify various tomato crop diseases. Additionally, it explores the impact of different optimization techniques on model performance, aiming to enhance accuracy, reduce computational costs, and ensure real-time applicability. By systematically analysing and comparing these optimization strategies, the research seeks to develop a robust and efficient framework for automated disease detection, thereby contributing to more sustainable and productive agricultural practices.

1.5 RESEARCH OBJECTIVES

The objectives of this work are:

i. To evaluate the effectiveness of various deep learning models (CNN, ResNet-50, InceptionV3, and EfficientNetB0) in classifying diseases in tomato plants.

ii. To analyse the impact of different optimization algorithms (Adam and Nadam) on the performance of deep learning models in the context of tomato plant disease classification, with a focus on key evaluation metrics such as accuracy and loss.

iii. To assess the efficacy of transfer learning techniques using pre-trained models in enhancing the classification accuracy of tomato plant diseases.

1.6 THESIS ORGANIZATION

This dissertation is divided into five chapters. A brief overview of each chapter is presented as follows:

Chapter 1: This chapter aims to present the motivation, research gaps, problem statement, and objectives along with the literature review. It tells about what work has been done for the present goal. In addition, chapter one helps in understanding thelimitations of the techniques previously implemented and what can be improved on. Furthermore, this chapter also presents the problem statement and objectives of this study.

Chapter 2: This chapter illustrates the background required to understand the problem of Tomato Plant Diseases. Concepts of machine learning, deep learning, transfer learning, and CNN is also presented in this chapter.

Chapter 3: This chapter describes about data collection and the proposed methodology. In addition, it gives a step-by-step detail of the process conducted to achieve the aim and it also discusses the implementation details of the proposed methodology.

Chapter 4: This chapter presents the results and their discussion. It gives the comparison of the developed models and confirms the results of each classification model through the provided tables and plots. Accuracy and Loss of all the models are presented in this chapter.

Chapter 5: This chapter states the conclusion based on the results obtained in the previous chapter. In addition to this, the future scope of this work is also discussed.

CHAPTER 2
BACKGROUND

This chapter gives the background details of the tomato plant leaf and a brief overview of tomato plant disease and various diseases affecting the plant is also presented. In this chapter, deep learning and transfer learning concepts are discussed. Brief introduction of CNN architecture is also presented in the last section.

2.1 TOMATO PLANT LEAF

The tomato plant is a versatile and widely cultivated plant known for producing one of the world's most popular fruits—the tomato. Native to South America, the tomato plant thrives in a variety of climates, making it a staple in gardens and farms globally [19]. The plant has a bushy or vine-like growth habit, depending on the variety, with green, feathery leaves and clusters of yellow flowers that eventually develop into the fruit. Tomato plants require a good amount of sunlight, well-drained soil, and consistent watering to produce abundant, flavourful fruits. They are also sensitive to temperature fluctuations, with both extreme heat and cold potentially hindering growth and fruit production. The tomato plant is not just valued for its fruits, which are rich in vitamins A and C, but also for its adaptability and relatively easy cultivation, making it a favourite among both commercial growers and home gardeners.

Figure 2.1 illustrates the top 10 tomato-producing countries in terms of production volume, measured in thousand tonnes. China, as the leading producer, significantly outpaces other countries with a production of 64,866 thousand tonnes. India, the second-largest producer, contributes 20,573 thousand tonnes. Turkey and the United States also have notable production figures, with 13,204 and 12,227 thousand tonnes, respectively. Egypt, Italy, and Iran, producing 6,731, 6,248, and 5,787 thousand tonnes respectively, from the mid-tier of producers. Spain, Mexico, and Brazil round out the top 10 with 4,313, 4,137, and 3,754 thousand tonnes respectively. This chart highlights the dominance of China and the significant contributions of other leading producers in global tomato production. Understanding this distribution is crucial for addressing market dynamics and agricultural strategies.

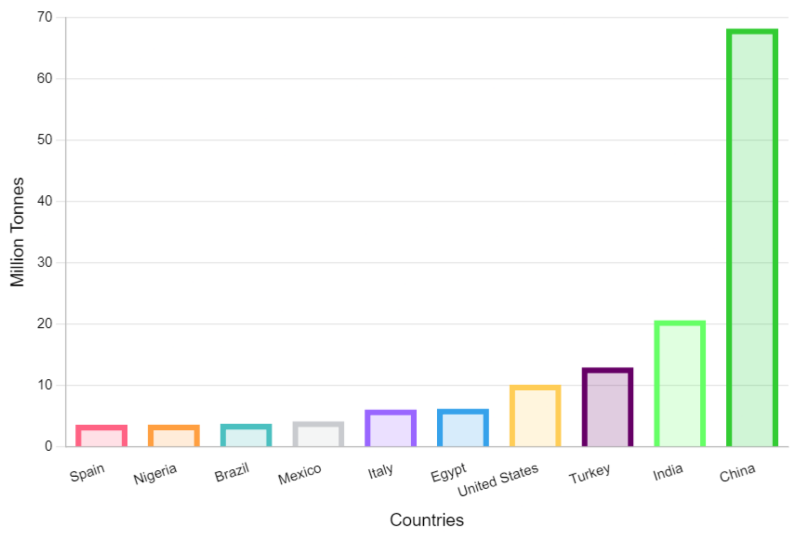


Figure 2.1: Top 10 tomato producing nations as of 2022 (in million tonnes)

2.2 TOMATO PLANT LEAF DISEASE

Tomato plants are widely cultivated across the globe, and their leaves play a critical role in the plant's growth and fruit production. However, various diseases can affect tomato leaves, leading to significant losses in yield and quality. Tomato leaf diseases are caused by factors such as fungal infections, bacterial contamination, and unfavourable environmental conditions [20]. Common diseases include early blight, late blight, and Septoria leaf spot, all of which can cause extensive damage to the plant by reducing photosynthesis and causing premature leaf drop. These diseases thrive in humid conditions and can quickly spread if not properly managed. Therefore, detecting and managing tomato leaf diseases at an early stage is crucial for maintaining healthy plants and ensuring a good harvest. In this study, the focus will be on identifying and addressing the common diseases that affect tomato plant leaves.

2.2.1 Bacterial Spot

Bacterial spot, caused by the bacterium *Xanthomonas spp.*, is a destructive disease affecting tomato plants [21]. It manifests as small, water-soaked lesions that become dark and greasy on leaves, stems, and fruits. Over time, these spots can coalesce, leading to extensive defoliation and fruit blemishes, significantly reducing the plant's photosynthetic ability and the marketability of the fruit. Warm, humid conditions favour the spread of bacterial spot, which can be transmitted through contaminated seeds, soil, tools, and splashing water. Effective management includes using disease-free seeds, practicing crop rotation, and applying appropriate bactericides. Figure 2.2 provides an example of Bacterial Spot disease.

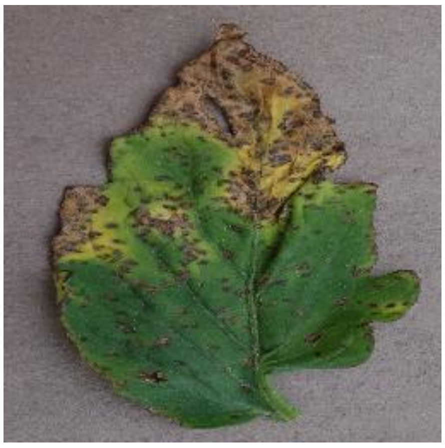


Figure 2.2: Bacterial Spot Disease

2.2.2 Early Blight

Early blight, caused by the fungus *Alternaria solani*, is a common and damaging disease of tomato plants. It typically starts with small, irregularly shaped brown spots on older leaves, which develop characteristic concentric rings as they expand. Infected leaves turn yellow and die, leading to significant defoliation and reduced fruit yield and quality. Also, the fungus can infect stems and fruits, causing dark, sunken lesions [22]. Managing early blight involves practicing crop rotation, removing infected plant debris, using resistant varieties, and applying fungicides as necessary. Figure 2.3 provides an example of Early Blight disease.

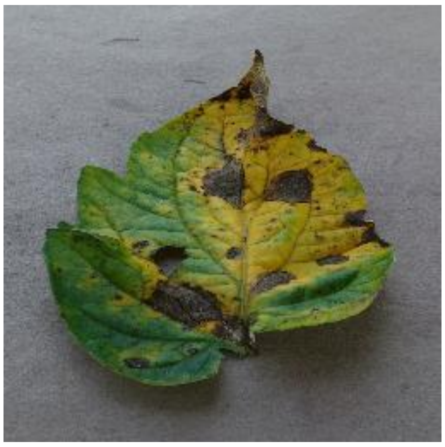


Figure 2.3: Early Blight Disease

2.2.3 Late Blight

Late blight, caused by the oomycete *Phytophthora infestans*, is a highly destructive disease of tomato plants. It appears as water-soaked lesions on leaves and stems that quickly turn brown and spread, often accompanied by white, fuzzy growth on the undersides of leaves under humid conditions. This disease can rapidly devastate an entire crop, particularly in cool, wet weather [22]. Effective management strategies include using resistant varieties, ensuring proper plant spacing for air circulation, removing infected plants, and applying fungicides preventatively. Figure 2.4 provides an example of Late Blight disease.

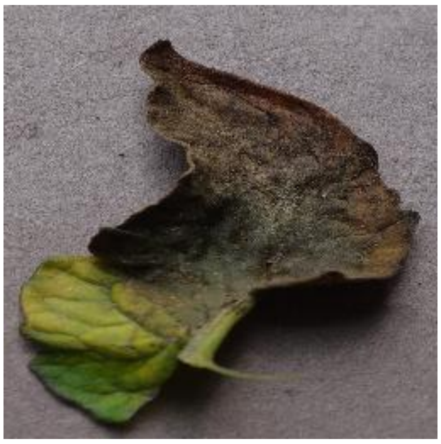


Figure 2.4: Late Blight Disease

2.2.4 Leaf Mold

Leaf mold, caused by the fungus *Passalora fulva*, primarily affects tomato plants grown in greenhouses or humid conditions. It presents as yellow spots on the upper leaf surfaces, which correspond to olive-green to grey mold growth on the undersides. Severe infections can lead to significant leaf drop, reducing the plant's ability to photosynthesize and impacting fruit production [23]. Managing leaf mold involves improving air circulation, reducing humidity, using resistant cultivars, and applying appropriate fungicides. Figure 2.5 provides an example of Leaf Mold disease.

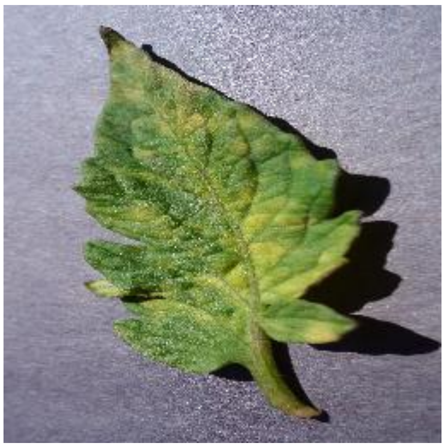


Figure 2.5: Leaf Mold Disease

2.2.5 Septoria Leaf Spot

Septoria leaf spot, caused by the fungus *Septoria lycopersici*, is a common disease of tomato plants. It is characterized by small, circular spots with dark borders and tan or grey centres on the lower leaves. As the disease progresses, spots can merge, causing extensive defoliation, weakening the plant, and exposing fruits to sunscald [24]. Effective management includes practicing crop rotation, removing plant debris, ensuring good air circulation, and using fungicides to protect plants. Figure 2.6 provides an example of Septoria Leaf Spot disease.

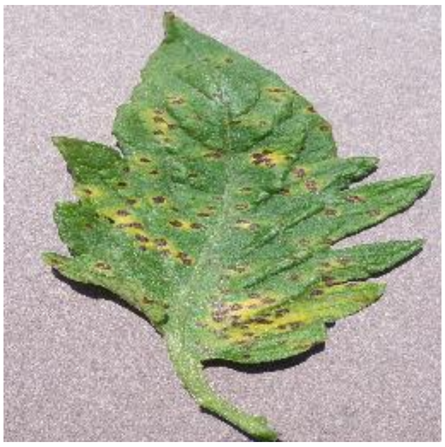


Figure 2.6: Septoria Leaf Spot Disease

2.2.6 Spider Mites

Spider mites, tiny arachnids of the Tetranychidae family, are common pests of tomato plants. They feed on the undersides of leaves, causing stippling, yellowing, and eventual leaf drop. Severe infestations can lead to bronzing of foliage and significant plant stress, reducing fruit yield and quality. Spider mites thrive in hot, dry conditions [23]. Management involves regular monitoring, encouraging natural predators, using insecticidal soaps or oils, and applying miticides if necessary. Figure 2.7 provides an example of Spider Mites disease.

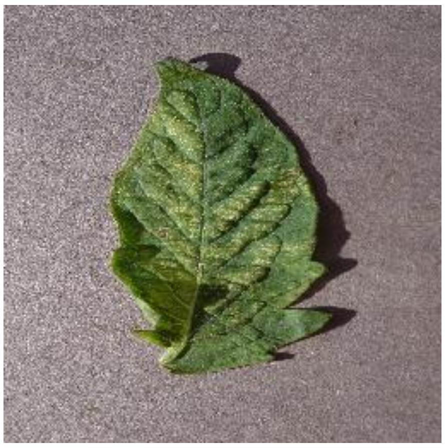


Figure 2.7: Spider Mites Disease

2.2.7 Target Spot

Target spot, caused by the fungus *Corynespora cassiicola*, affects tomato plants by producing small, dark, concentric spots on leaves, stems, and fruits. The disease can cause significant defoliation and fruit lesions, impacting yield and marketability. Warm, humid conditions favor its spread [21]. Managing target spot includes using disease-free seeds, practicing crop rotation, applying appropriate fungicides, and maintaining good field sanitation. Figure 2.8 provides an example of Target Spot disease.

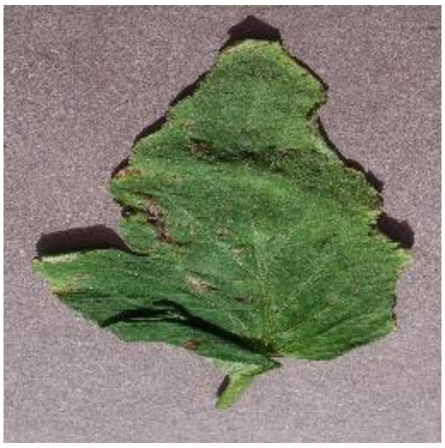


Figure 2.8: Target Spot Disease

2.2.8 Tomato Mosaic Virus (ToMV)

Tomato mosaic virus (ToMV) is a highly contagious virus that affects tomato plants, causing symptoms such as mottling, mosaic patterns, and distortion of leaves. Infected plants often exhibit stunted growth and reduced fruit yield and size. The virus spreads through contaminated soil, tools, hands, and infected plant debris [23]. Management strategies include using resistant varieties, practicing good sanitation, avoiding handling plants when wet, and removing infected plants to prevent the spread of the virus. Figure 2.9 provides an example of Tomato Mosaic Virus disease.

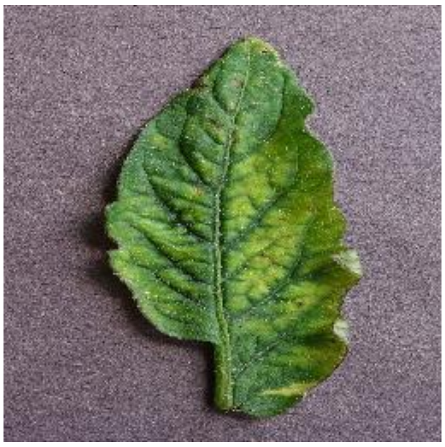


Figure 2.9: Tomato Mosaic Virus Disease

2.2.9 Tomato Yellow Leaf Curl Virus (TYLCV)

Tomato yellow leaf curl virus (TYLCV) is transmitted by whiteflies and causes severe yellowing and upward curling of leaves, along with stunted plant growth. Infected plants produce fewer and smaller fruits, significantly reducing yield. TYLCV is particularly devastating in warm climates where whiteflies thrive [23]. Effective management includes using resistant varieties, controlling whitefly populations with insecticides, implementing reflective mulches, and practicing good field hygiene to reduce virus spread. Figure 2.10 provides an example of Tomato Yellow Leaf Curl Virus disease.

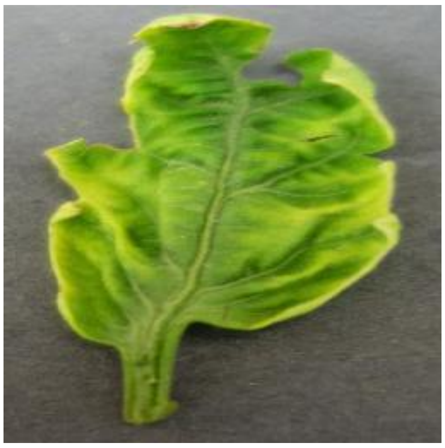


Figure 2.10: Tomato Yellow Leaf Curl Virus Disease

2.3 MACHINE LEARNING

The research on tomato leaf diseases has seen significant progress due to the application of machine learning (ML) algorithms. Studies have developed computer-aided systems using machine learning to analyse disease processes through image data. Techniques such as Support Vector Machines (SVM), Random Forest, and K-Nearest Neighbors are employed in this domain.

A machine learning framework typically includes components like feature extraction, feature training, and classification. Several ML algorithms, including Naïve Bayes, K-Nearest Neighbor, Random Forest, and Non-linear SVM with Radial Basis Function (RBF) kernel, have been utilized for training features [25].

Machine learning-based classification is highly automated, comparing test images to labelled images to detect subtle differences that might not be visible to the human eye. Recent studies indicate that machine learning algorithms can predict tomato diseases with greater accuracy than experienced plant entomologists. Techniques such as Support Vector Machines (SVM), Independent Component Analysis, and regression are commonly used for classification. Among these, SVM has been particularly effective, achieving high accuracy in classification tasks. However, SVM has been noted for its underperformance on raw data, and hand-crafted features are often essential for SVM-based models. Despite this, SVM performs well in distinguishing diseased leaves from healthy ones and excels in multi-class classification tasks. Figure 2.11 describes the hierarchy of Artificial Intelligence (AI), Machine Learning (ML) and Deep Learning (DL).

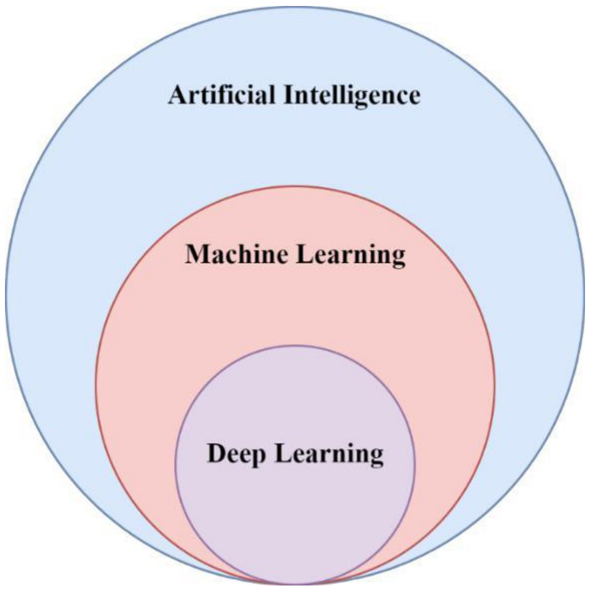


Figure 2.11: Hierarchy of AI, ML and DL

Machine Learning approaches are of three types namely; Supervised Learning, Unsupervised Learning and Reinforcement Learning.

2.3.1 Supervised Learning

Supervised learning makes uses of input data often represented as X and a target value Y such that Y = f(X). The goal is to iterate over the same X multiple times such that the model is able to correctly predict the corresponding Y for fresh data. It is termed as supervised learning as we are labelling our input data with the correct output data beforehand for the models or classifiers to learn from and be as close and accurate as possible to the true values [26]. This can also be understood by the example of a teacher supervising a student at each step and validating whether a student achieved the desired result. Classification and Regression are supervised ML problems. Classification is used when the output variable is a category and regression is used when the output variable is a real value. Figure 2.12 shows the detailed process of supervised learning.

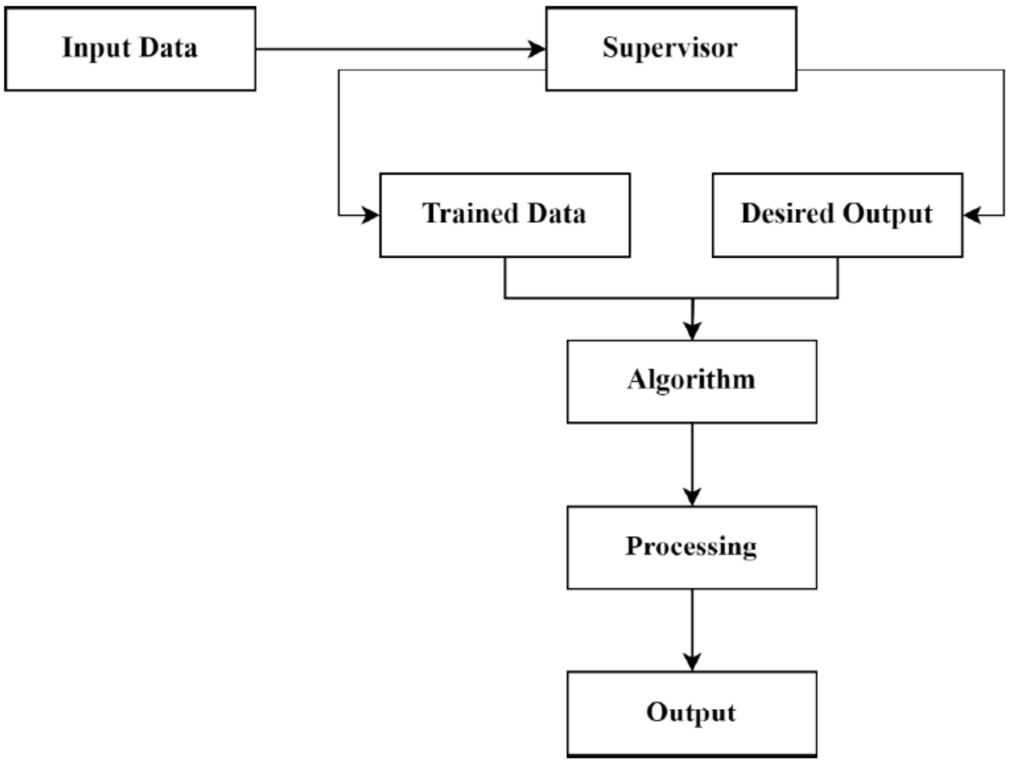


Figure 2.12: Supervised Machine Learning Process

2.3.2 Unsupervised Learning

In Unsupervised learning only the input data X is provided to the models and no output labels Y. This requires the algorithms to work on their own to extract and present some meaningful insights from the data. There is no form of supervision involved to direct the models toward a particular outcome [26]. Clustering and Association are types of Unsupervised learning. Complete process of Unsupervised learning is shown in Figure 2.13.

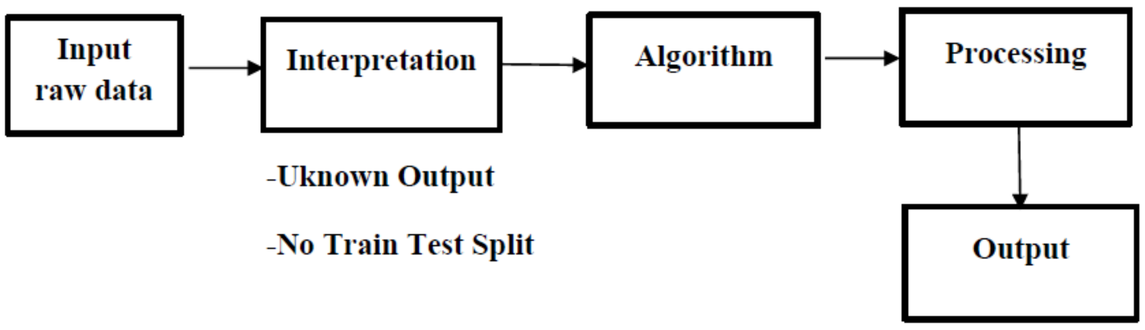


Figure 2.13: Unsupervised Machine Learning Process

2.4 DEEP LEARNING

In the last decade, deep learning (DL) has been used to predict and classify tomato leaf diseases [27]. Many deep learning technologies are now being used to enhance the detection and prediction of these diseases. The way neurons in the human brain process information serves as the basis for deep learning method's working principles. Artificial neurons, which are tiny nodes that are the most fundamental component of deep learning networks, are typically arranged in layers with connections between each neuron and every neuron in the layer below them via weighted connections.

Deep learning (DL) is a subset of machine learning, which is further of Artificial Intelligence (AI). Deep learning has been defined as "a new field of machine learning study that has been launched with the purpose of getting machine learning closer to oneof its initial goals: Artificial Intelligence." Deep learning can be classified into twocategories – generative architecture and discriminative architecture. Generative architecture can be subdivided into four types - Recurrent Neural Network (RNN), Deep Auto-Encoder (DAE), Deep Boltzmann Machine (DBM), Deep Belief Networks (DBN), and discriminative architecture can be divided into Convolutional Neural Networks (CNN) and Recurrent Neural Network.

ANN-based classification approach can produce the most convincing results (approximately 93.19%). Deep neural network algorithms have high speed and high performance in classification, recognition, interpretation, image analysis, etc.

With the aid of neuroimaging data, deep learning models have demonstrated impressive abilities in spotting obscured interpretations, figuring out connections between various image regions, and spotting patterns associated with particular diseases. Data preprocessing, feature extraction, and classification are the main steps that are included in classification tasks but with the development of deep learning techniques, all of these steps can be combined together into one.

Since, deep learning is based on deep neural network that can extract hundreds of characteristics from input data and simultaneously improves classification performance with high accuracy. Deep learning models are performing outstanding across a variety of applications, like audio and speech processing, visual data processing, natural language processing (NLP), image classification etc.

2.4.1 TRANSFER LEARNING

Transfer learning, used in deep learning, is the reuse of a pre-trained model on a new problem. As opposed to training a model from scratch, the transfer learning method allowed the use of weights trained previously on a specific task to be reused as thestarting point for a model on another task. Transfer learning is defined as a method of adjusting a pre-trained network for the user. In simple words, transfer learning is defined as the transfer of knowledge. Transfer learning is a method that not only uses samples from the target domain but also from a variety of related areas [28].

Transfer learning from ImageNet is employed, and the loss function values are weighted such that each class has the same weight, in order to get around the requirement for a sizable and well-balanced dataset.

There are two general approaches to use the transfer learning technique:

i) a pre-trained model using ImageNet weights that may be utilized as a feature extractor.

ii) fine-tuning a pre-trained model on a newly defined problem.

Pre-trained CNNs have proven to be good in the automated identification of diseases in tomato leaves images in recent research. Pre-trained deep neural networks that have been successfully employed include AlexNet, VGG16, VGG19, ResNet-34, ResNet-50, U-Net, SqueezeNet, InceptionV3, and DenseNet201.

Transfer learning can improve accuracy and cut down the training time when dealing with small datasets. With large datasets, transfer learning models take longer time for training thus it increases the computational cost of the model.

2.5 CONVOLUTION NEURAL NETWORK (CNN)

Convolutional neural networks (CNNs), often referred to as ConvNet, are among the most widely used deep learning architectures, particularly for image-related tasks. CNNs excel at identifying and extracting important patterns from images without requiring human intervention, making them increasingly popular for a variety of applications, including language modelling, sentiment analysis, and language translation. The strength of CNNs lies in their ability to process spatial information and extract features by stacking multiple convolutional layers, which allows the network to learn progressively more abstract representations of the input as it moves through the layers [29].

CNN architectures are typically classified as either 2D or 3D, depending on the type of data they process. While 3D CNNs offer better performance by capturing more detailed spatial information, they come with higher computational costs, increased complexity, and greater memory requirements. In contrast, 2D CNNs are easier to train, require less computational power, and are quicker to implement, making them a more practical choice for many applications. CNNs remain one of the most powerful and effective supervised learning models, particularly in the field of image processing. Models with modifications in CNN architectures: AlexNet, GoogLeNet, VGG, ZfNet, DenseNet, Inception-V3, ResNet, Xception, MobileNet, CapsuleNet, ResNext, HRNetV2, SqueezeNet.

2.5.1 ARCHITECTURE

The basic architecture which serves as the foundation for all CNN models [30], is described in figure 2.14.

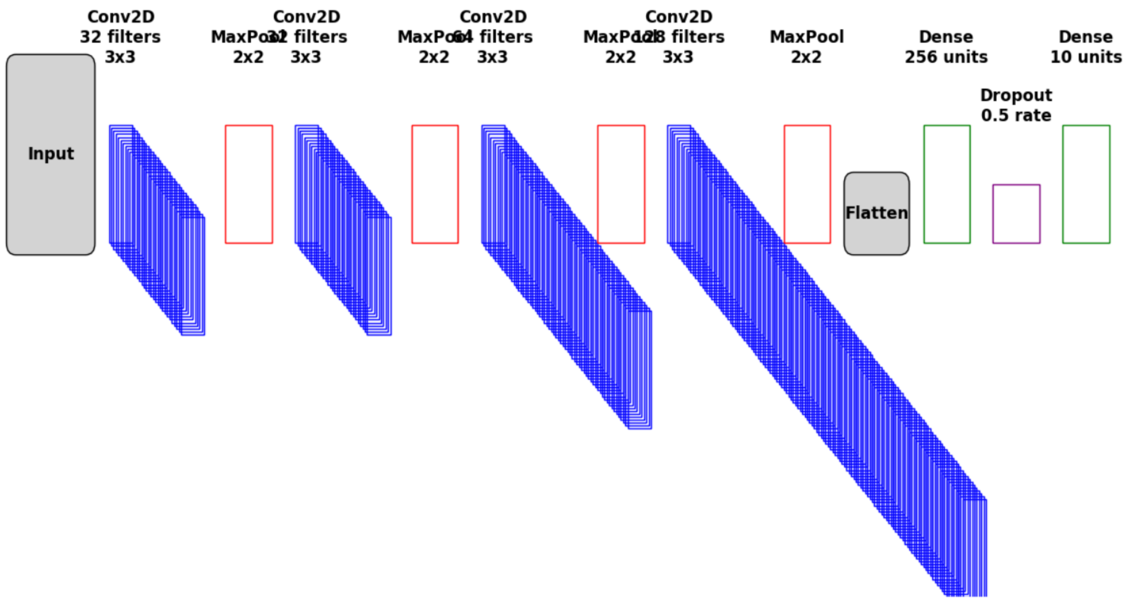


Fig. 2.14: A typical CNN architecture

The CNN architecture primarily comprises of three layers:

(1) Convolutional layer, (2) Pooling layer, and (3) Fully connected (FC) layer.

These layers are combined to form the CNN structure. The number and types of these layers may vary depending on the specific problem being addressed. The functions of each component of the CNN are explained below.

2.5.1.1 CONVOLUTIONAL LAYER

The convolutional layer (CL) is a fundamental component of CNNs, responsible for generating output through the convolution of two inputs. It is the first layer applied to the input data to produce a feature map, effectively extracting features from the input layer. The CL takes the input data and a kernel, also known as a filter or weight vector, with the kernel size defining its dimensions. For image inputs, the kernel is typically a two-dimensional array of binary values (0s and 1s), and the output is shaped according to the kernel [31].

With a specified stride value, the kernel moves across the width of the input image, sliding to the right until it has covered the entire width. It then moves down to the start of the next row and repeats the process until the entire image is processed. The resulting output serves as the input for the next layer in the network.

The hyperparameters for the convolutional layer include the kernel shape, the number of filters, the stride dimensions, and padding. The stride determines how many steps the filter takes as it slides over the image. Applying a filter to the previous layer produces a feature map. The convolution operation, denoted as conv(I·k), involves taking the dot product of the input image I and the convolution kernel k, resulting in a convolved feature map Fc.

$$
\begin{aligned}
Fc &= conv\ (\mathrm{i}, \mathrm{j}) \\
&= (I\,.\,K)\ (\mathrm{i}, \mathrm{j}) \\
&= \sum m \sum n\ I\ (m, n)\ k\ (\mathrm{i} - m, \mathrm{j} - n) \qquad (2.1)
\end{aligned}
$$

$conv$ (i, j) represents a 2D discrete convolution operator as shown in Equation 2.1. This operator illustrates how the convolution kernel k slides spatially over the input image I to perform elementwise multiplication and summation, resulting in the output of a convolved feature map Fc. The following are the advantages of Convolutional Layers: -

Sparse Connectivity: In fully connected neural networks, each neuron in one layer is connected to every neuron in the next layer. In contrast, CNNs feature sparse connectivity, where only a limited number of weights are shared between adjacent layers. This results in fewer required weights and a smaller amount of memory needed for storage.

Weight Sharing: In CNNs, weights are not pre-defined between neurons in adjacent layers. Instead, every pixel in the input matrix contributes to learning a single set of weights for the entire input. This approach significantly reduces training time by allowing the model to learn just one set of weights rather than multiple sets.

2.5.1.2 PADDING

Padding is often applied when the filter does not fit the input matrix. To ensure compatibility, the convolution filter size must be smaller than the input image size. If they do not match, zero padding is added to the image to align with the filter's dimensions. This method, known as "same padding," is commonly used in CNNs to maintain the size of the feature map and preserve spatial dimensions throughout the network. By adding padding, the input image size increases, which in turn enlarges the output feature map. This approach helps in retaining important edge information and ensures consistent output dimensions for subsequent layers [32].

2.5.1.3 POOLING LAYER

After the convolutional layer, the pooling layer is employed to reduce dimensions, which helps in minimizing the number of parameters and consequently lowers training time. This layer performs down-sampling on each feature map, decreasing its height and width to create smaller feature maps while retaining the most significant features throughout the pooling process. The two most common types of pooling are max pooling and average pooling. Max pooling selects the maximum value from the pooling window, whereas average pooling computes the average value within the window. Including a max-pooling layer between convolutional layers helps to progressively reduce the spatial size of the representation while preserving critical information [33].

2.5.1.4 ACTIVATION FUNCTION

When a neural network handles nonlinear inputs, it becomes more powerful and capable. The activation function plays a crucial role by determining whether a neuron should be activated based on a given input and producing the corresponding output. These functions are essential for learning the relationships between variables in the network [34].

Various activation functions are available, including tanh, sigmoid, ReLU, leaky ReLU, and Parametric Linear Units. However, the Rectified Linear Unit (ReLU) is the most commonly used activation function in the hidden layers.

$$ReLU = \max(0, x) \tag{2.2}$$

In the above equation, x is a real-valued number and is a small constant.

2.5.1.5 FULLY CONNECTED LAYER

The final layer in a CNN architecture is the fully connected (FC) layer, which operates similarly to a feed-forward artificial neural network (ANN). Every neural network includes at least one fully connected layer. The input to the FC layer comes from the last pooling or convolutional layer and is typically provided as a flattened vector of feature maps.

This fully connected layer processes the input from previous layers and transforms it into a set number of predefined classes. The FC layer contains its own biases and weights associated with the neurons [35]. During the forward pass, the output of the FC layer is used to calculate errors, which are then assessed using a loss function such as SVM or Softmax to determine the error gradient. This process completes one training cycle, including both the forward and backward passes.

2.5.1.6 LOSS FUNCTION

To assess the expected error across training samples in a CNN model, various loss functions are employed in the output layer. These loss functions help guide the optimizer by indicating whether it is moving in the correct or incorrect direction. The loss function evaluates how well the training process minimizes the discrepancy between the true labels and the predicted labels [36].

Deep convolutional neural networks (DCNNs) can use different loss functions tailored to specific tasks, such as Softmax and cross-entropy. The Softmax loss function is used to predict one class out of K mutually exclusive classes, with the Softmax layer providing a probability distribution where the output values sum to 1. For binary classification tasks, binary cross-entropy is used, while categorical cross-entropy is applied for multi-class classification tasks.

2.5.1.7 Model Hyper-Parameters

2.5.1.7.1 Batch Size

The batch size is an integer that determines how much input data is fed into the model during each iteration. Choosing the right batch size is essential because it affects the training speed. Larger batch sizes reduce the number of steps per epoch, but an inappropriate batch size can lead to significant accuracy variations and impact model stability. In our study, we used a batch size of 32 for training, which is appropriate given the number of images in the dataset.

2.5.1.7.2 Model Checkpoint Callback

This parameter functions as a callback, which includes predefined rules applied at different stages of training to assist and oversee the process. Callbacks are triggered at each epoch, providing considerable control over the training. Model checkpoint callbacks, in particular, are used to save the model or its weights at designated intervals, allowing training to be resumed from those saved points. Despite training all the mentioned models simultaneously, we implemented this parameter as a precautionary measure to handle potential hardware failures, ensuring that progress could be restored from the most recent checkpoint.

2.5.1.7.3 Reduce Learning Rate

The "Reduce Learning Rate on Plateau" or "reduce_lr" callback parameter is used to fine-tune a deep learning model to achieve optimal accuracy. In the later stages of training, the model's accuracy may fluctuate within a certain range if weight updates are too large. By lowering the learning rates, the model can make finer adjustments to its weights, leading to enhanced accuracy and efficiency. This technique helps ensure better convergence and improved performance throughout the training process.

Mathematical expression for calculating new learning rate is mentioned below in equation 2.3.

$$\textit{\textbf{New Learning Rate}} = \textit{Current Rate x Factor} \tag{2.3}$$

2.5.1.7.4 Early Stopping Callback

The Early Stopping Callback is used to terminate model training once convergence is achieved before completing the predetermined number of epochs. It monitors the model's accuracy or loss continuously, and if no improvement is detected over a specified number of epochs (patience), it halts the training and saves the model's configuration and weights [37]. In our setup, we configured a patience of 50 epochs to ensure that training stops early if progress stalls.

The advantages of using CNNs compared to traditional neural networks are:

1. Weight Sharing: CNNs utilize weight sharing, which reduces the number of trainable parameters and helps prevent overfitting.
2. Efficient Feature Extraction and Classification: CNNs perform feature extraction and classification simultaneously, leading to highly organized model outputs.
3. Scalability: CNNs facilitate easier implementation of large-scale networks compared to other neural networks.

2.5.1.8 REGULARIZATION

Overfitting is a common issue with neural networks, where a model performs well on existing data but fails to generalize to new data. Regularization techniques are employed to prevent overfitting, thereby enhancing the model's accuracy when encountering completely new data [38]. Some common techniques for regularization are-

2.5.1.8.1 DROPOUT

Dropout involves randomly deactivating both hidden and visible neurons in a neural network. Specifically, it sets the output of a certain proportion of neurons in a hidden layer to zero, as defined by the dropout ratio. The dropout ratio represents the probability of a neuron being dropped from a particular layer. For instance, a dropout ratio of 1 means that all neurons in that layer are deactivated, outputting zero. Conversely, a dropout ratio of 0.2 means that 20% of the neurons are randomly dropped from the network. Neurons that are dropped do not contribute to the forward or backward propagation steps. This process effectively samples a new network architecture during each forward and backward pass, while all architectures continue to share the same parameters [39].

2.5.1.8.2 BATCH NORMALIZATION

Batch Normalization is a technique used in training deep neural networks that normalizes and standardizes the inputs to each layer for every mini-batch processed by the network. This technique helps regulate the learning process, often leading to improved training efficiency and better model performance. It allows deep neural networks to train more quickly and steadily and can also help to reduce overfitting, which is particularly useful when working with small datasets [40].

However, Keras founder François Chollet has suggested that applying batch normalization after nonlinear activation functions like ReLU yields better results. The advantages of batch normalization are as follows:

- It can address issues related to poor weight initialization.
- It decreases the time needed for network convergence.
- It helps reduce the dependence on hyper-parameter tuning.
- It lowers the risk of overfitting.

CHAPTER 3

METHODOLOGY AND IMPLEMENTATION

In this chapter, the methodology used to develop classification models for Tomato Plant disease is presented. The first section gives the information about material and methods used in the work. The proposed methodology used for this work is discussed in the next section. Thereafter, the implementation detail of the proposed methodology is presented.

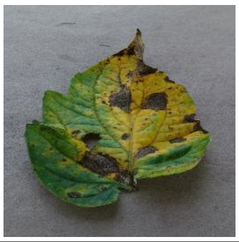
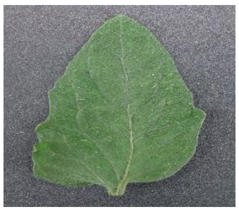
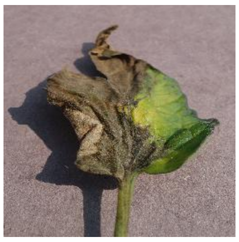
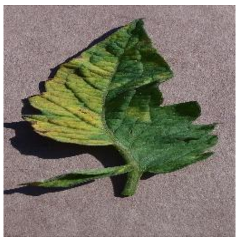
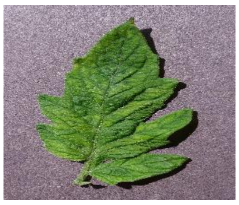
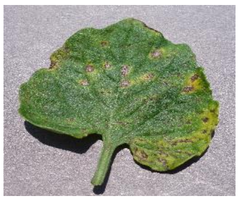
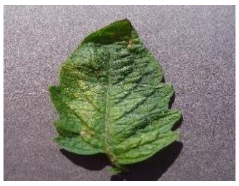
3.1 MATERIAL AND METHODS

3.1.1 DATA ACQUISITION

The dataset used is a subset of Plant Village Dataset by Mohanty et al. 2016 which has 38 different kinds of plants and approximately 54,306 images in the repository [41]. Total number of images in the tomato leaf subset was 18,800 and the number of images varies across these classes. Such imbalance in the dataset can potentially impact the performance of models. In this research, we have considered the tomato leaf dataset consisting of 1000 images of bacterial spot, 1000 images of early blight, 1000 images of late blight, 1000 images of leaf mold, 1000 images of mosaic virus, 1000 images of septoria leaf spot, 1000 images of spider mites, 1000 images of target spot, 1000 images of yellow leaf curl virus, and 1000 healthy tomato leaves. The images are in JPG format with 8-20KB size. The tomato dataset comprises of 10,000 tomato leaf images. The tomato disease classes used in experimentation are presented in Table 3.1.

Table 3.1 The number of Tomato Leaf images per diseased class

S.No	Class	No. of Images	Sample Image
1	Bacterial Spot	1000	

2	Early Blight	1000	
3	Healthy	1000	
4	Late Blight	1000	
5	Leaf Mold	1000	
6	Mosaic Virus	1000	
7	Septoria Leaf Spot	1000	
8	Spider Mites	1000	

9	Target Spot	1000	
10	Yellow Leaf Curl Virus	1000	

3.2 PROPOSED METHODOLOGY

The proposed method used for the classification of images for Tomato plant disease is shown in figure 3.1.

Firstly, the data is retrieved from the Plant Village dataset for the diagnosis of diseases. The second stage is to extract features for improving the performance of the classification task. Then, the images are split into training and validation subsets.

The last stage is designing different CNN based models and classifying the disease into a specific class.

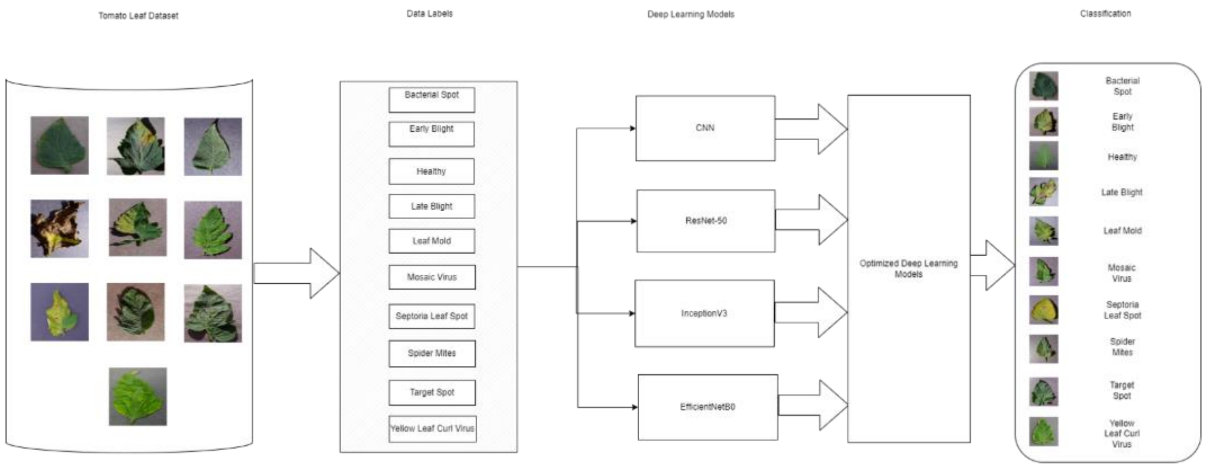


Fig. 3.1: Proposed Methodology

The details of each step is given in the below subsections:

3.2.1 Feature Extraction

Feature extraction is a critical step in machine learning and computer vision, especially with image data. It involves transforming raw data into measurable characteristics or features that can be used to build predictive models. In image processing, techniques may include identifying edges, textures, shapes, and colors [42]. For the PlantVillage dataset, features might include leaf shapes, color patterns, and visual indicators of disease. Advanced methods like CNNs automated feature extraction, enhance model accuracy in tasks like plant disease detection.

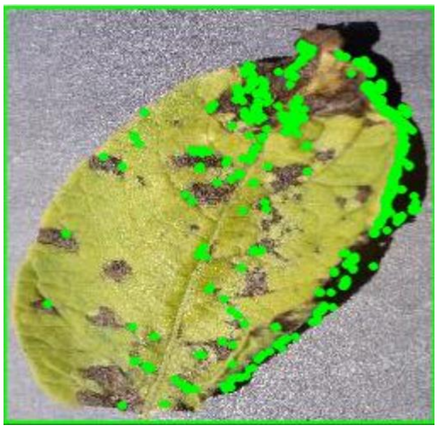


Fig 3.2: Feature Extraction

3.2.2 DATA SPLITTING

The dataset is divided into two different sets which are (1) training set (2) validation set with a split ratio of 80:20. The dataset consisted a total of 10000 images in which there are 8000 training images and 2000validation images. The number of images in training and validation set for multiclass classification are summarized in Table 3.2.

Table 3.2 Images in training and validation set

S.no	Class	Training	Validation	Total images
1	Bacterial Spot	800	200	1000
2	Early Blight	800	200	1000
3	Healthy	800	200	1000
4	Late Blight	800	200	1000
5	Leaf Mold	800	200	1000

6	Mosaic Virus	800	200	1000
7	Septoria Leaf Spot	800	200	1000
8	Spider Mites	800	200	1000
9	Target Spot	800	200	1000
10	Yellow Leaf Curl Virus	800	200	1000
	TOTAL	8000	2000	**10000**

After training the models on training dataset, the CNN models are tested for the classification performance with the validation dataset. Performance metrics, such as accuracy and loss that were independently computed using the validation data were used to assess the classifier's effectiveness.

3.3 TOMATO LEAF DISEASE CLASSIFICATION MODELS

In this study, four different classification models are implemented to perform 10-way classification task for early diagnosis of tomato plant disease. The first model is a CNN model, remaining are ResNet-50, InceptionV3 and EfficientNetB0 which are based on transfer learning technique and are used with pre-trained ImageNet weights. All the proposed models are discussed below-

3.3.1 CONVOLUTION NEURAL NETWORK (CNN)

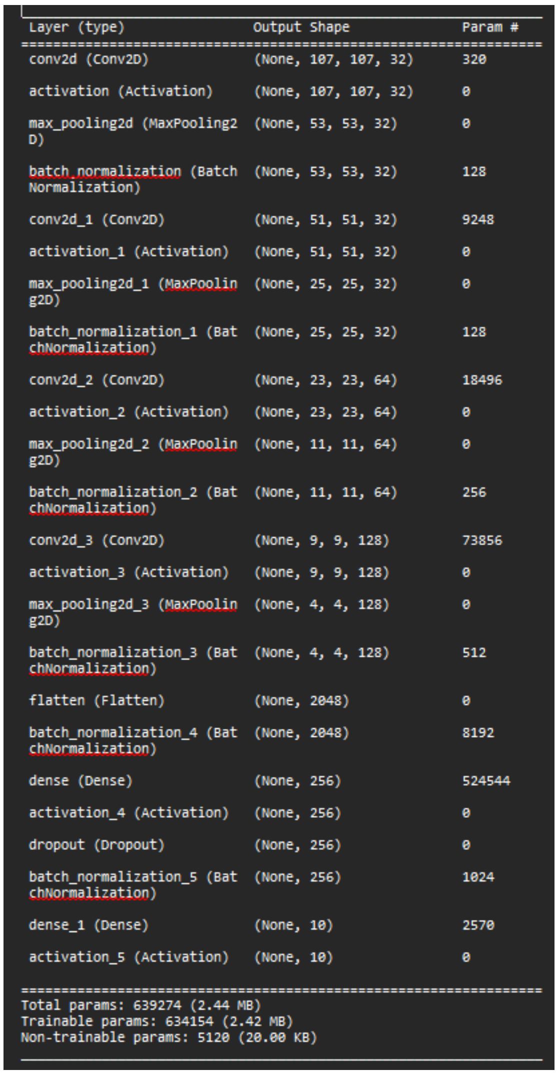
The CNN from scratch architecture be composed of the following different layers:

4 convolutional layers (CL), followed by ReLU activation and 4 max-pooling layers; 6 batch normalization layers; one dropout layer; a flatten layer; a fully connected layer with 256 neurons followed by a dropout layer; and finally, an output layer with Softmax activation. A brief description of the proposed architecture is given below:

- Tomato leaves data is fed into the CNN 's first layer i.e. a convolutional layer to perform convolution operation of input images and filter with resultant of multiple feature maps. The convolution layers have 32-32-64-128 output channels with kernel size of 3×3.
- Each convolutional layer is followed by a max-pooling layer applied with pool size of 2×2. The pooling layers are the down-sampling layers with generation of multiple pooled maps which is followed by batch normalization layer to standardize the

inputs to a layer for each mini-batch entering the network.

- Next, the pooled feature maps were flattened to a 1D vector as input to the subsequent fully connected layer with 256 neurons. The last dense layer is followed by a dropout layer with a dropout rate of 0.5, means 50% of the nodes would be dropped out in the layers for ensuring regularization and prevents from overfitting.
- The final layer is the output layer with 10 nodes with Softmax activation function to determine the probabilities of each possible class of the classification task. During model compilation the Loss function used is 'Categorical cross-entropy', and 'Adam' and ‘Nadam’ are used as optimizers.
- CNN model has a total of 639,274 parameters, in which 634,154 are trainable parameters and 5120 are non- trainable parameters. CNN model summary is shown in figure 3.5.



```
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 107, 107, 32)      320

 activation (Activation)     (None, 107, 107, 32)      0

 max_pooling2d (MaxPooling2  (None, 53, 53, 32)        0
 D)

 batch_normalization (Batch  (None, 53, 53, 32)        128
 Normalization)

 conv2d_1 (Conv2D)           (None, 51, 51, 32)        9248

 activation_1 (Activation)   (None, 51, 51, 32)        0

 max_pooling2d_1 (MaxPoolin  (None, 25, 25, 32)        0
 g2D)

 batch_normalization_1 (Bat  (None, 25, 25, 32)        128
 chNormalization)

 conv2d_2 (Conv2D)           (None, 23, 23, 64)        18496

 activation_2 (Activation)   (None, 23, 23, 64)        0

 max_pooling2d_2 (MaxPoolin  (None, 11, 11, 64)        0
 g2D)

 batch_normalization_2 (Bat  (None, 11, 11, 64)        256
 chNormalization)

 conv2d_3 (Conv2D)           (None, 9, 9, 128)         73856

 activation_3 (Activation)   (None, 9, 9, 128)         0

 max_pooling2d_3 (MaxPoolin  (None, 4, 4, 128)         0
 g2D)

 batch_normalization_3 (Bat  (None, 4, 4, 128)         512
 chNormalization)

 flatten (Flatten)           (None, 2048)              0

 batch_normalization_4 (Bat  (None, 2048)              8192
 chNormalization)

 dense (Dense)               (None, 256)               524544

 activation_4 (Activation)   (None, 256)               0

 dropout (Dropout)           (None, 256)               0

 batch_normalization_5 (Bat  (None, 256)               1024
 chNormalization)

 dense_1 (Dense)             (None, 10)                2570

 activation_5 (Activation)   (None, 10)                0

=================================================================
Total params: 639274 (2.44 MB)
Trainable params: 634154 (2.42 MB)
Non-trainable params: 5120 (20.00 KB)
_________________________________________________________________
```

Fig. 3.3: CNN model summary

3.3.2 RESNET-50

ResNet stands for Residual Network. In 2015, a variant ResNet-152, won the ILSVRC 2015 challenge, which is composed of 152 layers. ResNet-50 is a convolutional neural network that is 50 layers deep. Residual Networks are the first deeper neural networks that allowed for the training of thousands of layers while still delivering outstanding performance. ResNets are inspired by the VGG neural networks [43]. Variants of ResNets are-ResNet-34, ResNet-50, ResNet-101, ResNet-152.

3.3.2.1 Residual Block

ResNet is able to train such deep network because ResNet have the residual connections, which are not seen in any other deep neural network. Figure 3.4 shows the residual block of the ResNet.

Residual blocks enabled ResNet to link the outputs from previous layers to the outputs of stacked layers. Therefore, each layer not only see the previous layer‘s result but also of the layers behind the previous layer, making it possible to train much deeper networks [44].

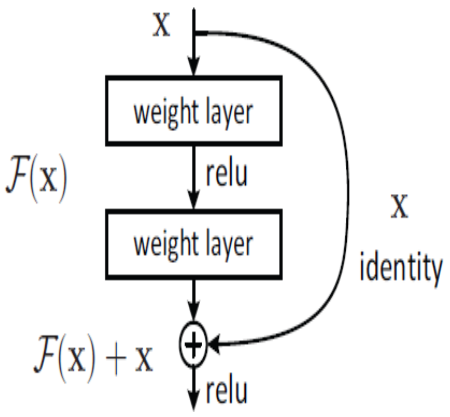


Fig. 3.4: Residual block [44]

A ResNet variant ResNet-50 model has 50 layers including 48 convolutional layers, 1 layer of max-pooling, and 1 layer of average pooling. ResNet-50 uses shortcut connections that skip three layers. Figure 3.5 shows building block for ResNet-50.

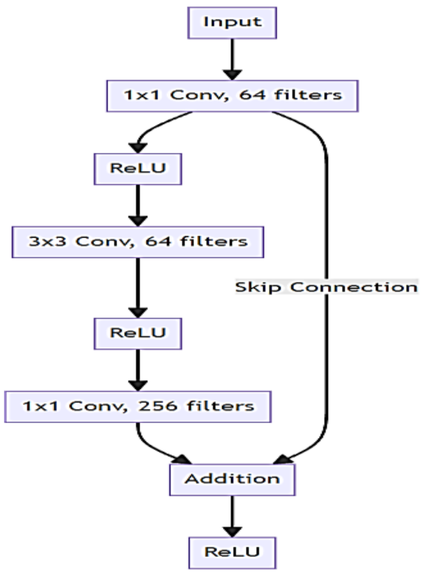


Fig. 3.5: A building block for ResNet-50

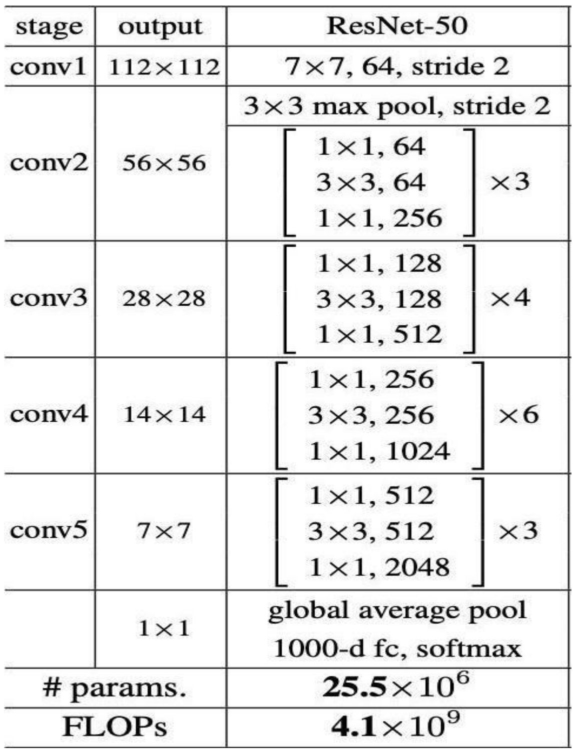
Architecture of ResNet-50 contains 5 blocks of convolution layers.

- First block has **1 layer** of a convolution with a kernel size of 7×7 and 64 different kernels all with a stride of size 2.
- Second block has a max-pooling with a stride size of 2. Then, three layers repeated for 3 times i.e. one layer with 64 filters of size 1×1; second with 64 filtersof size 3×3; third with 256 kernels of size 1×1, giving total **9 layers**.
- Third block have 3 layers with filter of 1×1, 128 after that a filter of 3×3, 128 and at last a filter of 1 × 1, 512. This step is repeated 4 times resulting a total of **12 layers**.
- In fourth block, there is a kernel of 1×1, 256 and second kernel with 3 × 3, 256 and third with 1 × 1, 1024 and this is repeated 6 times giving a total of **18 layers**.
- And then again in fifth block, a 1×1, 512 kernel with two of 3 × 3, 512 and 1 × 1, 2048 and this was repeated 3 times giving a total of **9 layers**.
- After that a average pooling and a fully connected layer containing 1000 nodes and **1 layer** of softmax function.

The batch normalization layer in ResNet is placed after every convolutional layer.

There are total= 1 + 9 + 12 + 18 + 9 + 1 = 50 layers in deep Convolutional neural network ResNet-50.

Figure 3.6 represents the ImageNet architecture with Building blocks.



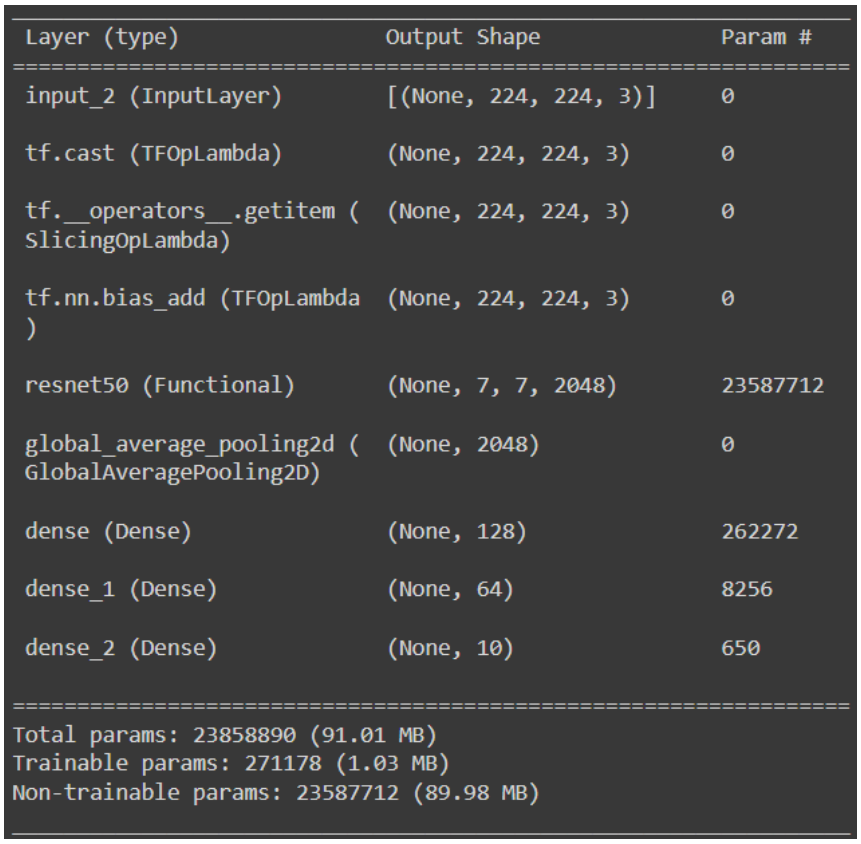
stage	output	ResNet-50
conv1	112×112	7×7, 64, stride 2
conv2	56×56	3×3 max pool, stride 2
		$\begin{bmatrix}1\times1, 64\\ 3\times3, 64\\ 1\times1, 256\end{bmatrix}\times3$
conv3	28×28	$\begin{bmatrix}1\times1, 128\\ 3\times3, 128\\ 1\times1, 512\end{bmatrix}\times4$
conv4	14×14	$\begin{bmatrix}1\times1, 256\\ 3\times3, 256\\ 1\times1, 1024\end{bmatrix}\times6$
conv5	7×7	$\begin{bmatrix}1\times1, 512\\ 3\times3, 512\\ 1\times1, 2048\end{bmatrix}\times3$
	1×1	global average pool 1000-d fc, softmax
# params.		$\mathbf{25.5}\times10^6$
FLOPs		$\mathbf{4.1}\times10^9$

Fig. 3.6: ImageNet architecture with Building blocks [44]

ResNet-50 with pre-trained ImageNet weights used to extract the features from the tomato leaves. The extracted features are then given to the new densely connected classifier,which is trained from scratch. The basic densely connected classifier is removed and changed with a new classifier. The features are flattened to a 1D vector and converged to the classifier.

The first layer is a fully connected layer with 2048 neurons with ‘glorot uniform’ as the kernel initializer followed by a batch normalization layer, activation function ‘ReLU ‘and a dropout layer with a 0.2 dropout rate. This is followed by another fully connected layer with 1024 neurons and a batch normalization, activation as ReLU and a dropout layer with a dropout ratio of 0.2. The last fully connected layer has 10 nodes with Softmax activation to output prediction of the 10-way classification task.

Figure 3.7 represents ResNet-50 model summary. Total parameters are: 23,858,890 with total trainable parameters: 271,178 and non-trainable parameters: 23,587,712.



```
Layer (type)                Output Shape              Param #
=================================================================
input_2 (InputLayer)        [(None, 224, 224, 3)]     0

tf.cast (TFOpLambda)        (None, 224, 224, 3)       0

tf.__operators__.getitem (  (None, 224, 224, 3)       0
SlicingOpLambda)

tf.nn.bias_add (TFOpLambda  (None, 224, 224, 3)       0
)

resnet50 (Functional)       (None, 7, 7, 2048)        23587712

global_average_pooling2d (  (None, 2048)              0
GlobalAveragePooling2D)

dense (Dense)               (None, 128)               262272

dense_1 (Dense)             (None, 64)                8256

dense_2 (Dense)             (None, 10)                650

=================================================================
Total params: 23858890 (91.01 MB)
Trainable params: 271178 (1.03 MB)
Non-trainable params: 23587712 (89.98 MB)
```

Fig. 3.7: Summary of ResNet-50 model

3.3.3 INCEPTIONV3

InceptionV3 is a convolutional neural network (CNN) architecture that builds upon the success of its predecessors in the Inception family. Introduced by Google in 2015, it achieves high performance on image recognition taks while maintaining computational efficiency. InceptionV3 incorporates several enhancements over previous versions, including factorized convolutions, batch normalization, and the use of auxiliary classifiers to improve gradient propagation. These modifications help reduce the number of parameters and computational cost, making the network both powerful and efficient [45].

3.3.3.1 Inception Module

The inception module, a core component of the InceptionV3 architecture, is designed to handle multiple spatial scales simultaneously. It achieves this by applying multiple convolutions with different filter sizes (1x1, 3x3, 5x5) and a max-pooling operation within the same module. These outputs are concatenated along the depth dimensions, allowing the network to capture diverse features from the input image. This multi-scale processing capability enhances the network's ability to recognize complex patterns and reduces the risk of overfitting. Figure 3.8 illustrates a typical Inception module.

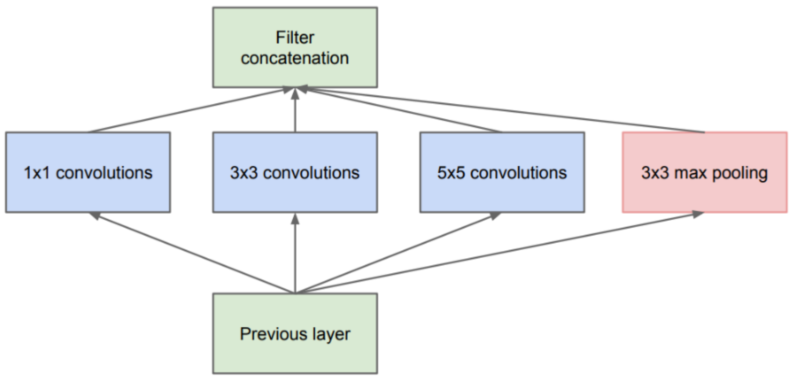


Fig 3.8: Inception Module [46]

3.3.3.2 Reduction Module

The reduction module in InceptionV3 is designed to downsample the spatial dimensions of the feature maps while increasing the network's depth, which is crucial for maintaining computational efficiency. This is achieved through a combination of stride convolutions and pooling layers, reducing the number of operations required in subsequent layers without losing important information. Figure 3.9 illustrates a typical Inception module with Reduction module, which uses multiple operations, including 1x1, 3x3, and 5x5 convolutions, as well as 3x3 max pooling, applied simultaneously to the input data. These operations are then concatenated along the depth dimension. The 1x1 convolutions, present both before and after the larger convolutions, serve to reduce computational costs and act as dimensionality reduction layers. Together, the reduction and Inception modules ensure that salient features are preserved, allowing the network to perform accurate and efficient image recognition even after the spatial resolution has been reduced.

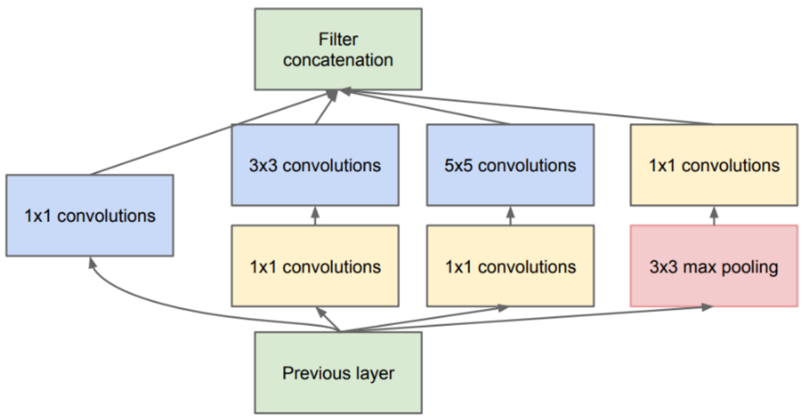
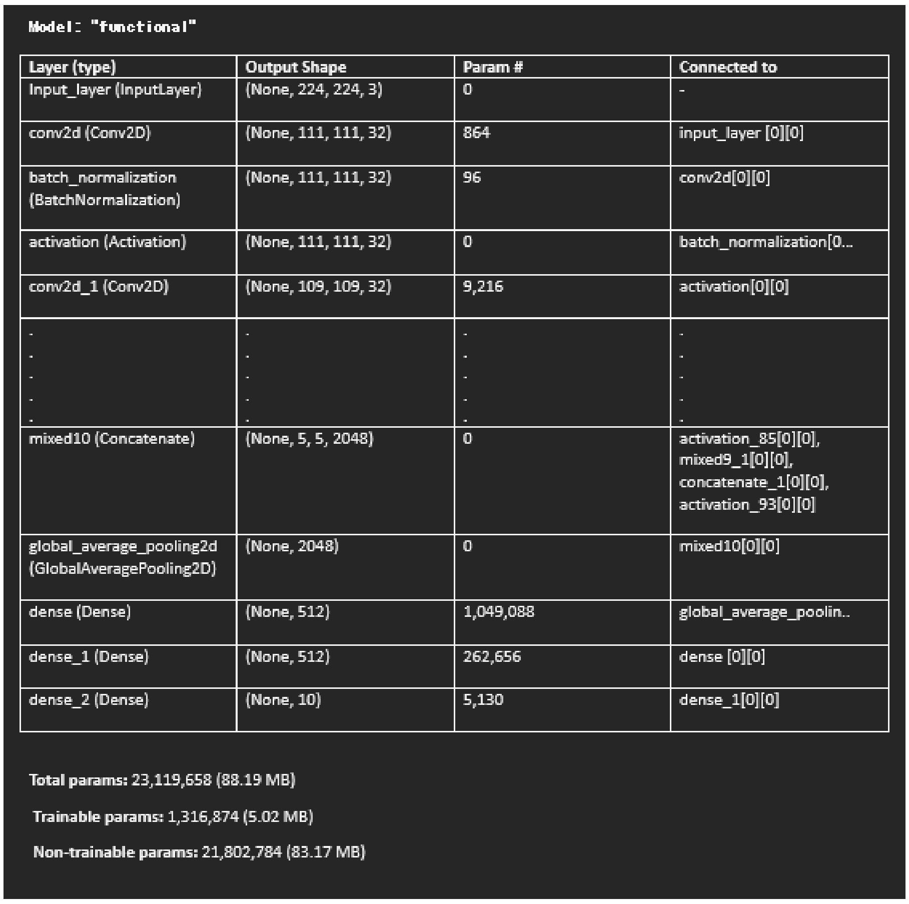


Fig 3.9: Inception module with Reduction Module [46]

A brief description of the proposed architecture is given below:

- Input to the network is a 299x299x3 tomato leaf image. The initial convolutional layer applies a set of 32 filters of size 3x3 with a stride of 2, followed by a batch normalization and a ReLU activation function.
- Followed by additional convolutional layers with 32 filters of size 3x3, then 64 filters of size 3x3, and a max-pooling layer with a pool size of 3x3 and stride of 2.
- The architecture utilizes multiple Inception modules (Inception-A, Inception-B, Inception-C) with different configurations of convolutional layers (1x1, 3x3, 5x5) and pooling layers to capture a wide range of features.
- Each module concatenates the outputs of these different convolutions, increasing the depth of the network while maintaining a manageable computational load.
- Reduction modules (Reduction-A and Reduction-B) are used at various points in the network to reduce the spatial dimensions of the feature maps while increasing the depth.
- These modules use a combination of max-pooling and convolutional layers to downsample the feature maps.
- An auxiliary classifier is added after some intermediate layers to provide additional gradient flow during training. This classifier includes a global average pooling layer, a fully connected layer with 1024 neurons, and a Softmax activation function.
- The final part of the network consists of a global average pooling layer that reduce the feature maps to a 1x1x2048 vector, followed by a dropout layer with a dropout rate of 0.5 to prevent overfitting.
- A fully connected layer with a Softmax activation function having 10 nodes is used to output the probabilities of each class of the classification task. During model compilation, the loss function used is 'Categorical cross-entropy' and 'Adam' and 'Nadam' are used as optimizers.
- InceptionV3 model has a total of 23,119,658 parameters, in which 1,316,874 are trainable parameters and 21,802,784 are non-trainable parameters. InceptionV3 model summary is shown in figure 3.10.
- Since, there were too many layers to present here in the study, only a handful of layers are shown in the archiecture figure.


Model: "functional"

Layer (type)	Output Shape	Param #	Connected to
Input_layer (InputLayer)	(None, 224, 224, 3)	0	-
conv2d (Conv2D)	(None, 111, 111, 32)	864	input_layer [0][0]
batch_normalization (BatchNormalization)	(None, 111, 111, 32)	96	conv2d[0][0]
activation (Activation)	(None, 111, 111, 32)	0	batch_normalization[0...
conv2d_1 (Conv2D)	(None, 109, 109, 32)	9,216	activation[0][0]
. . . . .	. . . . .	. . . . .	. . . . .
mixed10 (Concatenate)	(None, 5, 5, 2048)	0	activation_85[0][0], mixed9_1[0][0], concatenate_1[0][0], activation_93[0][0]
global_average_pooling2d (GlobalAveragePooling2D)	(None, 2048)	0	mixed10[0][0]
dense (Dense)	(None, 512)	1,049,088	global_average_poolin..
dense_1 (Dense)	(None, 512)	262,656	dense [0][0]
dense_2 (Dense)	(None, 10)	5,130	dense_1[0][0]

Total params: 23,119,658 (88.19 MB)

Trainable params: 1,316,874 (5.02 MB)

Non-trainable params: 21,802,784 (83.17 MB)

Fig 3.10: Summary of InceptionV3 model

3.3.4 EFFICIENTNETB0

EfficientNetB0 is a groundbreaking convolutional neural network (CNN) architecture developed by Google that sets a new standard for balancing accuracy and efficiency in image recognition tasks. Introduced in 2019, it employs a novel approach called compound scaling to systematically scale up the network's dimensions- depth, width, and resolution- in a balanced manner. EfficientNetB0 achieves state-of-the-art performance with significantly fewer parameters and FLOPs (floating point operations per second) compared to traditional CNN architectures, making it highly efficient for various applications, from mobile devices to large-scale cloud environments [47].

3.3.4.1 Compound Scaling

Compound scaling is the key innovation behind EfficientNetB0's superior performance. Instead of arbitrarily scaling the network's depth, width, or resolution, compound scaling uses a fixed set of scaling coefficients to simultaneously and proportionally increase all three dimensions. This method ensures a more balanced and efficient model expansion, leading to improved accuracy without excessive computational cost. By optimizing the trade-offs between these dimensions, compound scaling enables

EfficientNetB0 to achieve optimal performance with minimal resources, making it an attractive choice for both research and practical deployment in machine learning tasks. Figure 3.11 depicts various scaling techniques that are used to scale up the network's dimensions.

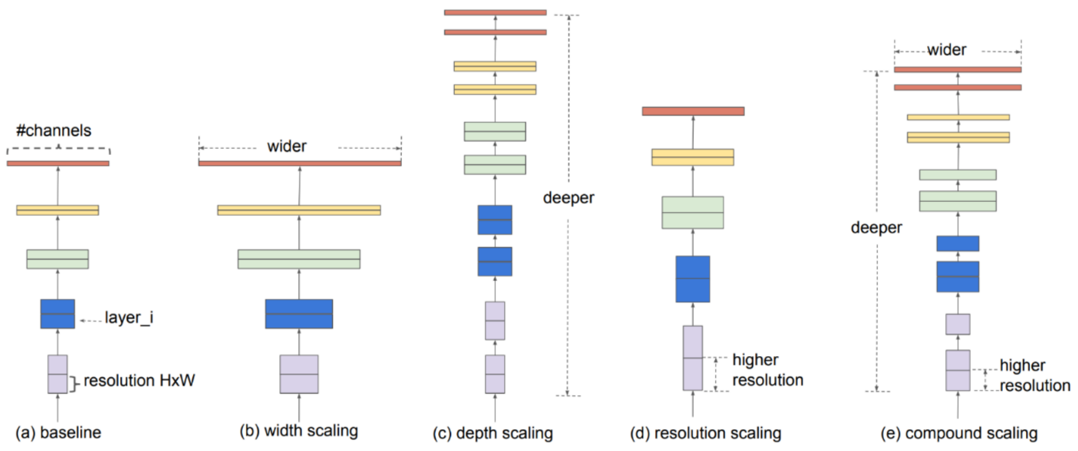


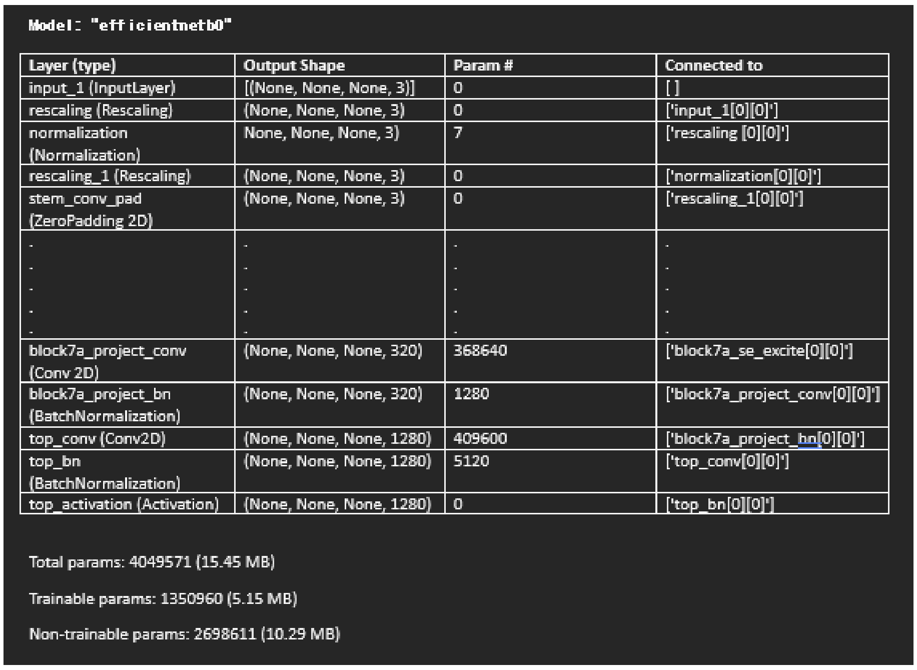
Fig 3.11: Depiction of various scaling techniques [47]

A brief description of the proposed architecture is given below:

- Input to the network is a 224x224x3 tomato leaf image. The initial convolutional layer applies a set of 32 filters of size 3x3 with a stride of 2, followed by a batch normalization layer and a Swish activation function.
- EffcientNetB0 model is based on a compound scaling method, which uniformly scales the depth, width, and resolution of the network to achieve better performance.
- Architecture is divided into several blocks, each consisting of mobile inverted bottleneck convolution (MBConv) layers with squeeze-and-excitation optimization.
    - ➢ Stage 1: One MBConv1 block with 16 filters and a kernel size of 3x3.
    - ➢ Stage 2: Two MBConv6 blocks with 24 filters and a kernel size of 3x3, with a stride of 2 in the first block for downsampling.
    - ➢ Stage 3: Two MBConv6 blocks with 40 filters and a kernel size of 5x5, with a stride of 2 in the first block for downsampling.
    - ➢ Stage 4: Three MBConv6 blocks with 80 filters and a kernel size of 3x3, with a stride of 2 in the first block for downsampling.
    - ➢ Stage 5: Three MBConv6 blocks with 112 filters and a kernel size of 5x5.
    - ➢ Stage 6: Four MBConv6 blocks with 192 filters and a kernel size of 5x5, with a stride of 2 in the first block for downsampling.
    - ➢ Stage 7: One MBConv6 block with 320 filters and a kernel size of 3x3.
- Network transitions to a 1280-filter convolutional layer followed by a global average pooling

layer that reduces the feature maps to a 1x1x1280 vector, followed by a dropout layer with a dropout rate of 0.2 is applied to prevent overfitting.

- A fully connected layer with a Softmax activation function having 10 nodes is used to output the probabilities of each class of the classification task. During model compilation, the loss function used is ‘Categorical cross-entropy’ and ‘Adam’ and ’Nadam’ are used as optimizers.
- EfficientNetB0 model has a total of 4,049,571 parameters, in which 1,350,960 are trainable parameters and 2,698,611 are non-trainable parameters. EfficientNetB0 model summary is shown in figure 3.12.
- Since, there were too many layers to present here in the study, only a handful of layers are shown in the archiecture figure.



Model: "efficientnetb0"

Layer (type)	Output Shape	Param #	Connected to
input_1 (InputLayer)	[(None, None, None, 3)]	0	[ ]
rescaling (Rescaling)	(None, None, None, 3)	0	['input_1[0][0]']
normalization (Normalization)	None, None, None, 3)	7	['rescaling [0][0]']
rescaling_1 (Rescaling)	(None, None, None, 3)	0	['normalization[0][0]']
stem_conv_pad (ZeroPadding 2D)	(None, None, None, 3)	0	['rescaling_1[0][0]']
. . . . .	. . . . .	. . . . .	. . . . .
block7a_project_conv (Conv 2D)	(None, None, None, 320)	368640	['block7a_se_excite[0][0]']
block7a_project_bn (BatchNormalization)	(None, None, None, 320)	1280	['block7a_project_conv[0][0]']
top_conv (Conv2D)	(None, None, None, 1280)	409600	['block7a_project_bn[0][0]']
top_bn (BatchNormalization)	(None, None, None, 1280)	5120	['top_conv[0][0]']
top_activation (Activation)	(None, None, None, 1280)	0	['top_bn[0][0]']

Total params: 4049571 (15.45 MB)

Trainable params: 1350960 (5.15 MB)

Non-trainable params: 2698611 (10.29 MB)

Fig 3.12: Summary of EfficientNetB0 model

3.4 Evaluation Metrics

Evaluation metrics are essential tools for quantifying the performance of DL models. They offer valuable insights into a model’s training effectiveness and its consistency during the process. These metrices enable researchers and practitioners to gauge the model’s accuracy, loss, and other performance indicators, giving a comprehensive understanding of how well the model performs and how reliable are its predictions. By interpreting these metrics, we can make informed decisions about model improvements and identify areas that require attention to optimize the model's overall performance [48].

Equations (3.1) and (3.2) show the formula of both performance metric wherein TP and TN are the number of positive and negative instances identified correctly, respectively. FP and FN are the numbers of misclassified positive and negative cases, respectively.

3.4.1 Accuracy: Accuracy serves as a metric to assess a model's performance in classification tasks. It quantifies the proportion of correctly classified predictions among all the predictions made by the model. A model is deemed effective if it achieves a high accuracy, indicating its ability to make accurate predictions on the given dataset.

Mathematical expression for accuracy of a model is shown in equation 3.1.

$$\boldsymbol{Accuracy} = TP + TN/TP + TN + FP + FN \qquad (3.1)$$

3.4.2 Loss: The loss value of a model reflects its performance at each iteration or epoch, indicating how well or poorly it has performed. It measures the disparity between the model's predicted outputs and the actual expected outputs. The objective of training is to minimize this loss. In this paper, the cross-entropy loss function is employed for the conducted studies. Equation 3.2 represents the mathematical expression of cross-entropy loss.

$$\mathbf{Loss} = \sum_{n=1}^{\infty} \mathrm{yi\ X\ log\neg yi} \qquad (3.2)$$

3.5 IMPLEMENTATION DETAILS

3.5.1 PYTHON

Python is a versatile, high-level programming language that has gained widespread popularity in recent years. Developed by the Python Software Foundation and designed by Guido van Rossum, Python succeeded the ABC programming language. The initial release of Python 0.9.0 was on February 20, 1991. Subsequent versions included Python 2.0 in 2000 and Python 3.0 in 2008, with Python 2.7.18 marking its discontinuation in 2020. The current stable version, Python 3.10.7, was released on

September 7, 2022. The official website is python.org, and Python's source code is available under the GNU General Public License (GPL).

Key Features of Python [49]:

- Simple and Easy to Learn: Despite being a high-level language, Python's syntax is straightforward, making it accessible for beginners. Many can learn to code in Python within hours or days.
- Freeware and Open Source: Python is developed under an OSI-approved open-source license, making it free to use for both personal and commercial purposes. It can be downloaded, modified, and redistributed at no cost.
- Procedure-Oriented and Object-Oriented: Python supports both procedural and object-oriented programming paradigms. It emphasizes design around data and objects, while also focusing on reusable code.
- High-Level Language: Python abstracts away system architecture details and memory management, allowing programmers to focus on writing code.
- Portability: Python code can run on various machines without modification, making it a platform-independent language.
- Dynamically Typed: Python determines variable types at runtime, offering flexibility in coding.
- Interpreted: Python executes code line by line, which facilitates debugging and interactive development.
- Extensible: Python can be extended with other languages like C++, enhancing its versatility.
- Extensive Library: Python comes with a comprehensive set of libraries for tasks ranging from image manipulation to database management, reducing the need for custom code.
- Beginner-Friendly: Python supports a wide range of applications, from text processing to web browsers and games, making it ideal for newcomers.
- Easy to Debug: Python's design makes it easier to identify and fix errors in code.
- GUI Programming Support: Python supports graphical user interface development through modules such as PyQt5, PyQt4, wxPython, and Tk, with PyQt5 being a popular choice for creating GUI applications.

Python is in high demand across various fields, including [49]:

1. Data Science: Involves data analytics, mining, and big data. Key libraries include:
    - NumPy: Provides mathematical functions for handling large arrays.
    - Matplotlib: A library for creating a wide range of visualizations, including line graphs

and scatter plots.

- SciPy: Used for scientific and mathematical computing, with modules for optimization, linear algebra, and signal processing.

2. Machine Learning: Libraries such as NumPy, Matplotlib, and SciPy are used, along with:
   - Scikit-Learn: Offers algorithms and functions for data mining and analysis, facilitating the implementation of machine learning models.
   - TensorFlow: Developed by Google for fast numerical computing and deep learning model creation.
   - Keras: Provides tools for building and training neural networks, designed for rapid experimentation in deep learning.

3. Internet of Things (IoT): Python is used for developing IoT devices, with popular microcontrollers like Micropython.

3.5.2 Experimental Setup

All experiments are conducted using Python, with the neural network constructed via the Keras deep learning library and TensorFlow as the backend. TensorFlow is a free and open-source software library designed for machine learning and artificial intelligence, particularly for training and inference with deep neural networks [50]. Keras, also an open-source library, provides a Python interface for artificial neural networks (ANN) and serves as a frontend for TensorFlow. It offers implementations of various neural network components, including layers, loss functions, activation functions, and optimizers [51].

In this study, training and validation routines are carried out on Google Colab, which supports Python 3 and offers GPU integration for developing CNN models. Google Colab facilitates collaborative work and notebook execution with GPU support.

The experiments are performed on a laptop equipped with an Intel Core i5 processor, 8GB of RAM, and an NVIDIA GTX 1650 GPU with 4GB of memory. Four distinct models are trained on a dataset of 10,000 images for a 10-class classification task of tomato plants. For all models, the 'Adam' and 'Nadam' optimizers are used, with a batch size of 32 and 30 epochs. The batch size specifies the number of samples processed in one forward/backward pass during training. If the batch size is higher, the more memory space will be required.

An epoch refers to the number of times the entire training dataset has passed through the deep learning model, which includes one forward pass and one backward pass of all training examples. In this study, each model is trained for 30 epochs.

Steps per epoch are calculated as the ratio of total training samples to batch size. For the CNN model, 250 steps are needed to complete one epoch; for ResNet-50, 200 steps per epoch; and for both InceptionV3 and EfficientNetB0, 250 steps per epoch.

The Model Checkpoint callback is utilized with model.fit() to periodically save the model or its weights, allowing for later resumption of training from the saved state.

A loss function evaluates how accurately the neural network models the training data by comparing the target values with the predicted outputs. Classification loss functions are used in classification tasks, such as Binary Cross-Entropy for binary classification and Categorical Cross-Entropy for multi-class classification. For all four models in this study, the loss function used is Categorical Cross-Entropy.

Table 3.3 represents training parameters for CNN model, Table 3.4 represents training parameters for ResNet-50 model, Table 3.5 represents training parameters for InceptionV3 model and Table 3.6 represents training parameters for EfficientNetB0 model.

Table 3.3: Training parameters for CNN

S.No.	Parameter	Value
1	Number of epochs	30
2	Batch size	32
3	Learning rate	0.001
4	Optimizers	Adam, Nadam
5	Loss function	Categorical cross-entropy
6	Metrics	Accuracy, Loss

Table 3.4: Training parameters for ResNet-50

S.No.	Parameter	Value
1	Number of epochs	30
2	Batch size	32
3	Kernel initializer	Glorot uniform
4	Optimizers	Adam, Nadam
5	Learning rate	0.001
6	Loss function	Categorical cross-entropy
7	Metrics	Accuracy, Loss
8	Weights	ImageNet

Table 3.5: Training parameters for InceptionV3

S.No.	Parameter	Value
1	Number of epochs	30
2	Batch size	32
3	Optimizers	Adam. Nadam
4	Learning rate	0.001
5	Loss function	Categorical cross-entropy
6	Metrics	Accuracy, Loss
7	Weights	ImageNet

Table 3.6: Training parameters for EfficientNetB0

S.No.	Parameter	Value
1	Number of epochs	20
2	Batch size	32
3	Optimizers	Adam, Nadam
4	Learning rate	0.001
5	Loss function	Categorical cross-entropy
6	Metrics	Accuracy, Loss
7	Weights	ImageNet

CHAPTER 4

RESULTS AND DISCUSSION

This chapter presents the experiment results and discussion. In the first section, the results of the four models are discussed with all the evaluation metrics. After that, in next section, a brief discussion of the results is presented.

4.1 RESULTS

Experiments are performed to classify the Tomato plant disease into nine diseased classes and one healthy class. For this, the whole dataset is divided into training and validation datasets with 80:20 split ratio. As discussed in previous chapter, there are total 10000 images in which training dataset has 8000 images and validation dataset has 2000 images. All the number of images with their diagnostic group are shown in previous chapter. CNN, ResNet-50, InceptionV3, and EfficientNetB0 are applied using Adam and Nadam to see the classification performance of the models and evaluation metrics like accuracy and loss are calculated.

4.1.1 Training Results

Training is performed on 8000 images with 30 epochs and 250 steps per epochs for CNN, InceptionV3, EfficientNetB0 and 200 steps per epoch for ResNet-50. Training accuracy and training loss of dataset with all four models training using Adam is given in table 4.1 and using Nadam is given in table 4.2.

The accuracy of training dataset with CNN is 93.71% using Adam and 97.37% using Nadam, with ResNet-50 it is 97.53% using Adam and 97.47% using Nadam, 91.68% with InceptionV3 using Adam and 92.12% using Nadam, and with EfficientNetB0 it is 99.90% using Adam and 99.92% using Nadam. Overall, the training performance for all four models is sufficiently good. All the four models reached an average training accuracy of 91%.

EfficientNetB0 attained the highest training accuracy which is 99.92% followed by ResNet-50 then CNN followed by InceptionV3 when using Nadam.

Table 4.1 Training performance for Adam Optimizer

S. No.	Model	Epoch	Accuracy	Loss
1	CNN	30	0.9371	1.0771
2	ResNet-50	30	0.9753	0.0731
3	InceptionV3	30	0.9168	0.2380
4	EfficientNetB0	30	0.9990	0.0037

Table 4.2 Training performance for Nadam Optimizer

S. No.	Model	Epoch	Accuracy	Loss
1	CNN	30	0.9737	1.0887
2	ResNet-50	30	0.9747	0.0772
3	InceptionV3	30	0.9212	0.2164
4	EfficientNetB0	30	0.9992	0.0023

4.1.2 Validation Results

Validation on 2000 image samples with 30 epochs and 250 steps per epoch for CNN, InceptionV3, EfficientNetB0 and 200 steps per epoch for ResNet-50 is performed. Validation accuracy and validation loss of dataset with all four models training using Adam is given in table 4.3 and using Nadam is given in table 4.4.

The accuracy of validation dataset with CNN is 96.30% using Adam and 95.87% using Nadam, with ResNet-50 it is 95.45% using Adam and 94.75% using Nadam, 83.97% with InceptionV3 using Adam and 84.32% using Nadam, and with EfficientNetB0 it is 98.40% using Adam and 98.75% using Nadam. EfficientNetB0 attained the highest validation accuracy which is 98.75% followed by CNN then ResNet-50 followed by InceptionV3 when using Nadam.

Table 4.3 Validation Performance for Adam Optimizer

S. No.	Model	Epoch	Accuracy	Loss
1	CNN	30	0.9630	1.2649
2	ResNet-50	30	0.9545	0.1408
3	InceptionV3	30	0.8397	0.5325
4	EfficientNetB0	30	0.9840	0.0343

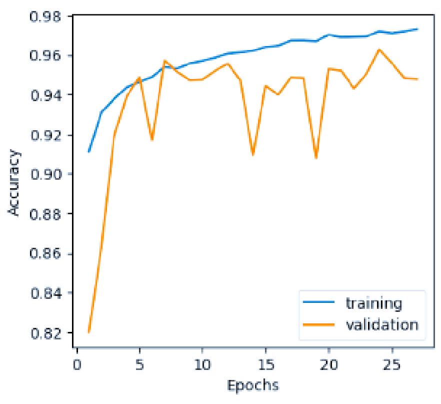
Table 4.4 Validation Performance for Nadam Optimizer

S. No.	Model	Epoch	Accuracy	Loss
1	CNN	30	0.9587	1.3283
2	ResNet-50	30	0.9475	0.1459
3	InceptionV3	30	0.8432	0.4999
4	EfficientNetB0	30	0.9875	0.0402

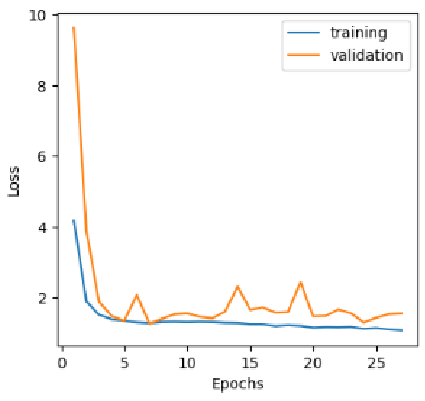
Plots showing learning curves of training accuracy and loss, validation accuracy and loss for CNN using Adam is shown in figure 4.1. As shown in figure 4.1(a), training accuracy is increasing with increase in epochs whereas the validation accuracy increases and decreases with alternative epochs. In figure 4.1(b), training loss is continuously decreasing with learning whereas the validation loss is continuously decreasing until 5th epoch then slightly increase and alternatively decreases and increases.

Plots of training and validation accuracy, training and validation loss for ResNet-50 using Adam is shown in figure 4.2. Training accuracy is increasing with increase in epochs until 6th epoch and then decreases and then gradually increases and after 15th epoch increases and decreases alternatively and the validation accuracy is increasing and decreasing with alternative epochs, as shown in figure 4.2(a)

Training loss curve is decreases with epochs, only slight increase at 6th and 16th epoch whereas the validation loss curve is decreasing and increasing with alternating epochs but after 17th epoch it is slightly increasing and then decreasing causing the straight line, as shown in figure 4.2(b).

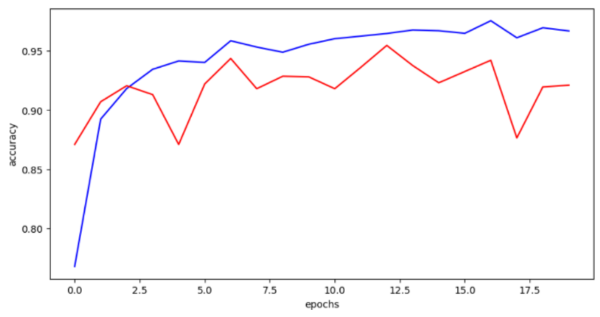


(a)

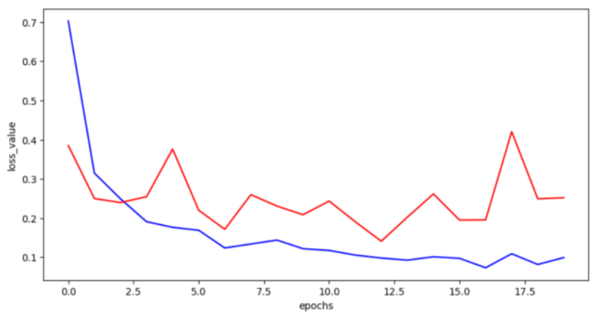


(b)

Fig.4.1 Plot for CNN using Adam (a) training and validation Accuracy (b) training and validation loss



(a)

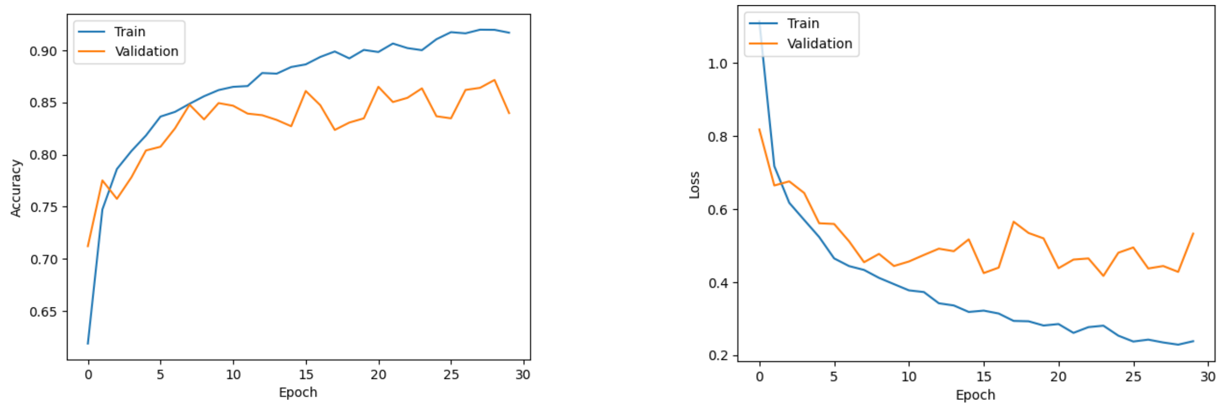


(b)

Fig.4.2 Plot for ResNet-50 using Adam (a) training and validation Accuracy (b)training and validation loss

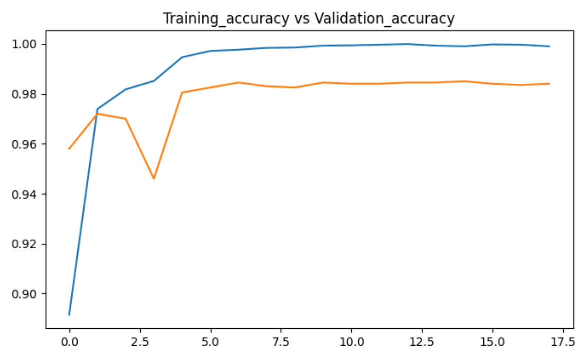
Plots showing training accuracy, validation accuracy and training loss and validation loss for InceptionV3 using Adam is shown in figure 4.3. In figure 4.3(a), training accuracy gradually increases until 24th epoch when it decreases and then increases whereas validation accuracy initially increases but sudden decrease at 3rd epoch, after that it alternatively increases and decreases. In figure 4.3(b), training loss gradually decreases but slight increase at 21st epoch and again it starts gradually decreases and with validation loss curve, it is decreasing and increasing with alternating epochs.

Plots showing training accuracy, validation accuracy and training loss and validation loss for EfficientNetB0 using Adam is shown in figure 4.4. Training accuracy is increases with the increasing epochs till 4th epoch then take almost plateau shape and validation accuracy is increasing with epochs then decreases at 3rd epoch and then increases till 4th epoch then take almost plateau shape, as shown infigure 4.4(a). In figure 4.4(b), training loss gradually decreases till 9th epoch and then curve takes plateau shape and validation loss decreases with epoch but increases at 1st epoch and then decreases gradually till 9th epoch and then it takes plateau shape.

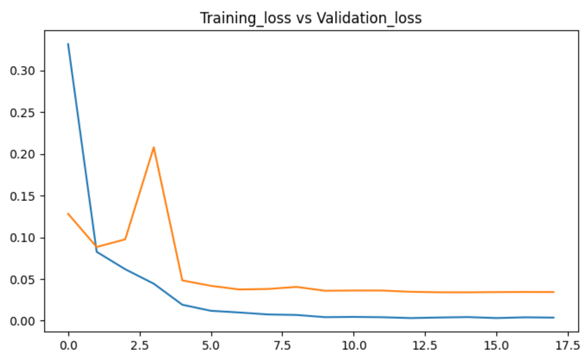


(a) (b)

Fig.4.3 Plot for InceptionV3 using Adam (a) training and validation Accuracy (b) training and validation loss



(a)



(b)

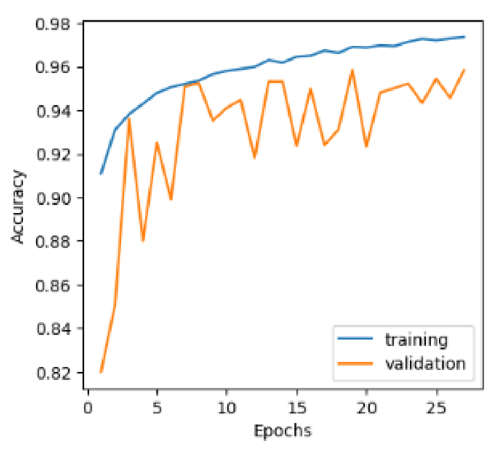
Fig.4.4 Plot for EfficientNetB0 using Adam (a) training and validation Accuracy (b) training and validation loss

Among the four models using Adam, EfficientNetB0 has the lowest training loss value of 0.0037 as well as validation loss value of 0.0343 and ResNet-50 has the second lowest training loss value as well as validation loss value, CNN has the highest training loss value as well as validation loss value.

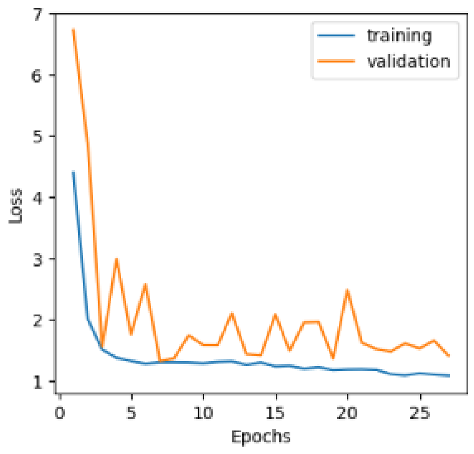
Plots showing learning curves of training accuracy and loss, validation accuracy and loss for CNN using Nadam is shown in figure 4.5. As shown in figure 4.5(a), training accuracy is increasing with increase in epochs whereas the validation accuracy increases and decreases with alternative epochs. In figure 4.5(b), training loss is continuously decreases with learning whereas the validation loss is first continuously decreases until 3rd epoch then alternatively increases and decreases.

Plots of training and validation accuracy, training and validation loss for ResNet-50 using Nadam are shown in figure 4.6. Training accuracy is increasing with increase in epochs and the validation accuracy is increasing and decreasing with alternative epochs, as shown in figure 4.6(a)

Training loss curve is decreasing with epochs, whereas the validation loss curve is decreasing and increasing with alternating epochs but after 9th epoch it is slightly increasing and then decreasing at 12th epoch as shown in figure 4.6(b).

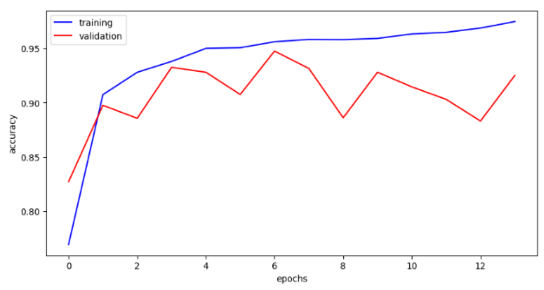


(a)

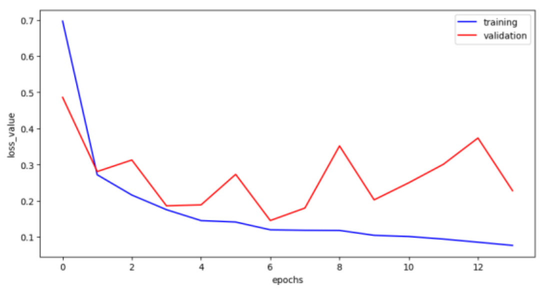


(b)

Fig.4.5 Plot for CNN using Nadam (a) training and validation Accuracy (b) training and validation loss



(a)

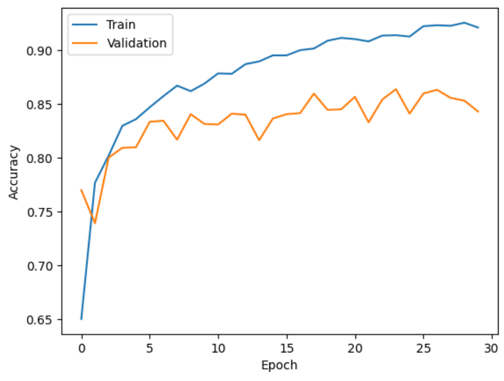


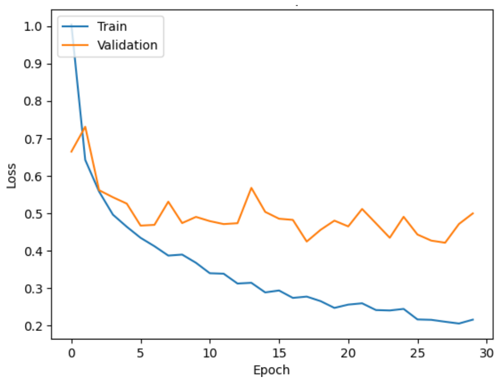
(b)

Fig.4.6 Plot for ResNet-50 using Nadam (a) training and validation Accuracy (b)training and validation loss

Plots showing training accuracy, validation accuracy and training loss and validation loss for InceptionV3 using Nadam is shown in figure 4.7. In figure 4.7(a), training accuracy gradually increases whereas validation accuracy initially decreases but sudden increase at 1st epoch, after that it alternatively increases and decreases. In figure 4.7(b), training loss gradually decreases and validation loss curve initially increases and then decreases till 3rd epoch and then increasing and decreasing with alternating epochs.

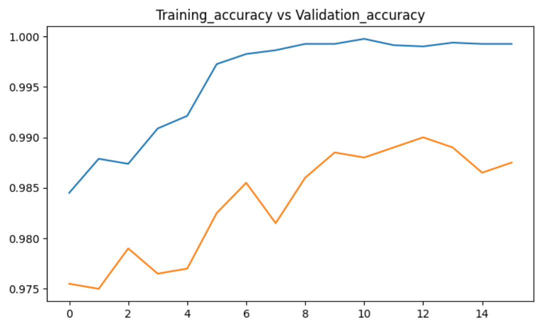
Plots showing training accuracy, validation accuracy and training loss and validation loss for EfficientNetB0 using Nadam is shown in figure 4.8. Training accuracy is increasing and decreasing till 5th epoch and then gradually increasing, Validation accuracy initially decreases and then increases and decreases with alternating epoch as shown in figure 4.8(a). In figure 4.8(b), training loss gradually decreases till 3rd epoch and after a slight increase it again decreases till 9th epoch taking a plateau shape, Validation loss decreases and increases with alternating epoch but increases at 6th epoch and then decreases gradually till 12th epoch and then increasing and decreasing at 14th epoch.

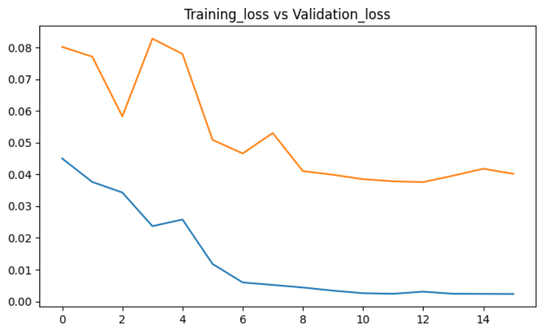




(a) (b)

Fig.4.7 Plot for InceptionV3 using Nadam (a) training and validation Accuracy (b) training and validation loss





(a) (b)

Fig.4.8 Plot for EfficientNetB0 using Nadam (a) training and validation Accuracy (b) training and validation loss

Among the four models using Nadam, EfficientNetB0 has the lowest training loss value of 0.0023 as well as validation loss value of 0.0402 and ResNet-50 has the second lowest training loss value as well as validation loss value, CNN has the highest training loss value as well as validation loss value.

4.1.3 Prediction Results

Prediction of some random tomato leaves demonstrated the efficacy of various deep learning models. By accurately identifying and classifying these leaves, the study validated the robustness of the models in handling diverse and unstructured data. This capability is crucial for practical applications, such as assisting farmers in diagnosing plant diseases with high precision and reliability, ultimately enhancing crop management and productivity. Figure 4.9 shows prediction of some random tomato leaf images which are affected by various tomato plant diseases. The actual and predicted labels, along with their probabilities, are also shown. For example, in the first image, the actual disease is MosaicVirus, and the model correctly predicts it with 100% probability.

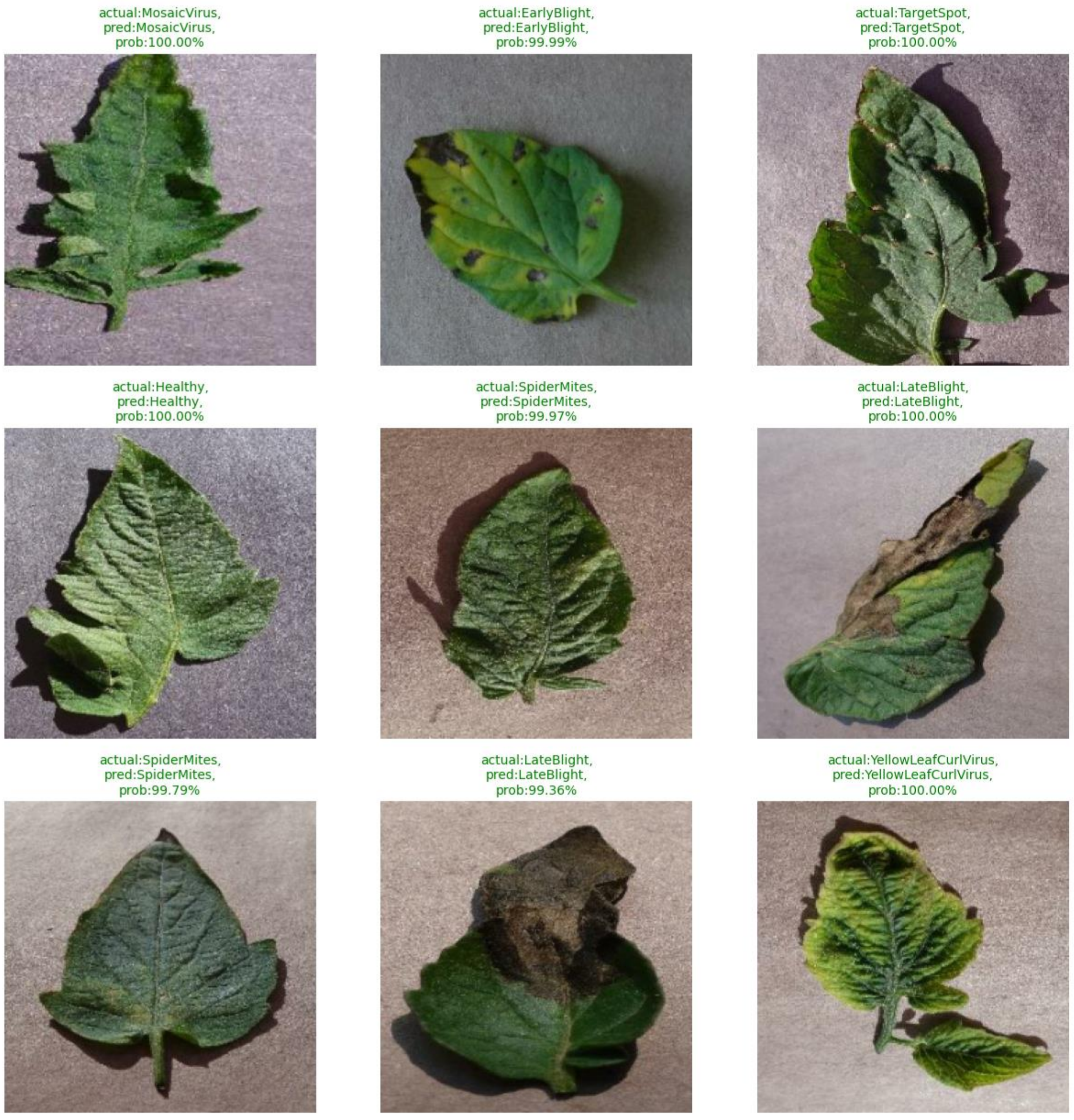


Fig.4.9 Prediction of some random tomato leaves.

4.2 Discussion

According to the results, EfficientNetB0 outperformed the CNN, ResNet-50, and InceptionV3 models. EfficientNetB0 achieved a training accuracy of 99.92% and a validation accuracy of 98.75% using Nadam optimizer, thanks to its convolutional base trained on the ImageNet database, which facilitates effective feature extraction for multiclass classification. It also recorded the lowest training and validation losses among the four models. The ResNet-50 model had the second lowest losses, while CNN showed the highest training loss and validation loss. Although transfer learning delivers good results on small datasets, the complexity of classifying tomato leaves suggests that a larger dataset is preferable for optimal model training. Limited datasets can lead to overfitting, reducing the model's overall effectiveness. Thus, choosing an efficient training approach is crucial to minimize training time and computational cost. With limited data, deep learning models may struggle to extract generalized high-level features, impacting their overall performance. The transfer learning approach used in this study offers higher accuracy and faster training speed than existing methods, despite its high computational cost. Most researches, in general, concentrated on using a single optimization technique. In this study, two optimization techniques namely, Adam and Nadam are used to showcase the effect of varied optimization techniques. Accurate classification of tomato leaf diseases can assist farmers in diagnosing various conditions, reducing the risk of misdiagnosis, and improving crop production.

CHAPTER 5

CONCLUSION AND FUTURE SCOPE

This chapter concludes the proposed work and also discusses its future scope such as what other techniques can be added to the current work to make the classification task more accurate.

5.1 CONCLUSION

In this study, we focused on the classification of tomato leaves using images sourced from the Plant Village database. The results highlight the effectiveness of various optimizers in training a convolutional neural network for this purpose. Four deep learning models namely CNN, ResNet-50, InceptionV3, and EfficientNetB0 were implemented to classify a total of 10000 images across ten different classes, with accuracy and loss calculated for each model. Among these, EfficientNetB0 outperformed CNN, ResNet-50, and InceptionV3, achieving the highest accuracy of 99.00%. This superior performance can be attributed to EfficientNetB0's optimized architecture, which balances depth, width, and resolution, thereby enhancing feature extraction and classification capabilities. The study underscores the importance of selecting appropriate models and optimizers to improve classification accuracy in agricultural applications. Moreover, the findings suggest that advanced neural network architectures, such as EfficientNetB0, are highly effective for plant disease detection, which could significantly benefit precision agriculture and crop management practices.

5.2 FUTURE SCOPE

Many aspects mentioned below can be explored and used in future to expand the work in classification of Tomato plant disease.

- Various data augmentation techniques like zooming, rotation etc. can be applied.
- Additionally, the pre-training of CNN, like AlexNet, Xception, MobileNet, auto-encoders or sparse auto-encoders, and other variants of ResNet can be used for more accurate results.
- Cross-validation can be performed for evaluating and comparing learning techniques by dividing dataset into training and validation set.
- Development of robust and scalable CNN architectures, the creation of large-scale annotated datasets, and the integration of domain knowledge from plant pathology experts into the model training process.

- Additionally, exploring alternative modalities such as hyperspectral imaging and multispectral imaging for disease detection could further improve the accuracy and reliability of automated diagnosis systems.

REFERENCES

1. Hassan, S.M., Maji, A.K., Jasiński, M., Leonowicz, Z., Jasińska, E.: Identification of plant-leaf diseases using cnn and transfer-learning approach. Electronics. 10, (2021). https://doi.org/10.3390/electronics10121388.

2. Habiba, S.U., Islam, M.K.: Tomato Plant Diseases Classification Using Deep Learning Based Classifier from Leaves Images. In: 2021 International Conference on Information and Communication Technology for Sustainable Development, ICICT4SD 2021 - Proceedings. pp. 82–86. IEEE, Dhaka (2021). https://doi.org/10.1109/ICICT4SD50815.2021.9396883.

3. Bedi, P., Gole, P.: Plant disease detection using hybrid model based on convolutional autoencoder and convolutional neural network. Artif. Intell. Agric. 5, 90–101 (2021). https://doi.org/10.1016/j.aiia.2021.05.002.

4. Chen, H.C., Widodo, A.M., Wisnujati, A., Rahaman, M., Lin, J.C.W., Chen, L., Weng, C.E.: AlexNet Convolutional Neural Network for Disease Detection and Classification of Tomato Leaf. Electronics. 11, (2022). https://doi.org/10.3390/electronics11060951.

5. Sakkarvarthi, G., Sathianesan, G.W., Murugan, V.S., Reddy, A.J., Jayagopal, P., Elsisi, M.: Detection and Classification of Tomato Crop Disease Using Convolutional Neural Network. Electronics. 11, (2022). https://doi.org/10.3390/electronics11213618.

6. Mia, M.J., Maria, S.K., Taki, S.S., Biswas, A.A.: Cucumber disease recognition using machine learning and transfer learning. Bull. Electr. Eng. Informatics. 10, 3432–3443 (2021). https://doi.org/10.11591/eei.v10i6.3096.

7. Thangaraj, R., Anandamurugan, S., Kaliappan, V.K.: Automated tomato leaf disease classification using transfer learning-based deep convolution neural network. J. Plant Dis. Prot. 128, 73–86 (2021). https://doi.org/10.1007/s41348-020-00403-0.

8. Zhao, S., Peng, Y., Liu, J., Wu, S., Pethybridge, J., Hay, F.: Tomato Leaf Disease Diagnosis Based on Improved Convolution Neural Network by Attention Module. Agriculture. 11, (2021). https://doi.org/10.3390/agriculture.

9. Biswas, A.A., Zulfiker, M.S., Rajbongshi, A., Mia, M.J., Majumder, A.: Feature Ranking Based Carrot Disease Recognition Using MIFS Method. In: Abraham, A. (ed.) Hybrid Intelligent Systems 2021. pp. 56–68. Springer, Cham (2022). https://doi.org/10.1007/978-3-030-96305-7_6.

10. Ahmad, I., Hamid, M., Yousaf, S., Shah, S.T., Ahmad, M.O.: Optimizing pretrained convolutional neural networks for tomato leaf disease detection. Complexity. 2020, (2020). https://doi.org/10.1155/2020/8812019.

11. Agarwal, M., Singh, A., Arjaria, S., Sinha, A., Gupta, S.: ToLeD: Tomato Leaf Disease Detection using Convolution Neural Network. In: International Conference on Computational Intelligence and Data Science (ICCIDS 2019). pp. 293–301. Elsevier B.V., Gurugram (2020). https://doi.org/10.1016/j.procs.2020.03.225.

12. Basavaiah, J., Arlene Anthony, A.: Tomato Leaf Disease Classification using Multiple Feature Extraction Techniques. Wirel. Pers. Commun. 115, 633–651 (2020). https://doi.org/10.1007/s11277-020-07590-x.

13. Chen, X., Zhou, G., Chen, A., Yi, J., Zhang, W., Hu, Y.: Identification of tomato leaf diseases

based on combination of ABCK-BWTR and B-ARNet. Comput. Electron. Agric. 178, (2020). https://doi.org/10.1016/j.compag.2020.105730.

14. E, N.K., M, K., P, P., R, A., S, V.: Tomato Leaf Disease Detection using Convolutional Neural Network with Data Augmentation. In: 2020 5th International Conference on Communication and Electronics Systems (ICCES). pp. 1125–1132. IEEE, Coimbatore (2020). https://doi.org/10.1109/ICCES48766.2020.9138030.

15. Karthik, R., Hariharan, M., Anand, S., Mathikshara, P., Johnson, A., Menaka, R.: Attention embedded residual CNN for disease detection in tomato leaves. Appl. Soft Comput. 86, (2020). https://doi.org/10.1016/j.asoc.2019.105933.

16. Elhassouny, A., Smarandache, F.: Smart mobile application to recognize tomato leaf diseases using Convolutional Neural Networks. In: 2019 International Conference of Computer Science and Renewable Energies (ICCSRE). pp. 1–4. IEEE, Agadir (2019). https://doi.org/10.1109/ICCSRE.2019.8807737.

17. Widiyanto, S., Fitrianto, R., Wardani, D.T.: Implementation of Convolutional Neural Network Method for Classification of Diseases in Tomato Leaves. In: 2019 Fourth International Conference on Informatics and Computing (ICIC). pp. 1–5. IEEE, Semarang (2019). https://doi.org/10.1109/ICIC47613.2019.8985909.

18. TM, P., Pranathi, A., Kandiraju Sai, A., Nagaratna B., C., Koolagudi G., S.: Tomato Leaf Disease Detection using Convolutional Neural Networks. In: 2018 Eleventh International Conference on Contemporary Computing (IC3), Noida, India. pp. 1–5. IEEE, Noida (2018). https://doi.org/10.1109/ic3.2018.8530454.

19. Samal, S., Verma, V.: An Optimized Deep Learning Model for the Detection and Classification of Tomato Plant Leaf Diseases Using Self-Developed CNN Model. J. Propuls. Technol. 44, 2116–2129 (2023). https://doi.org/https://doi.org/10.1007/s11042-024-18978-3.

20. Sharmila, M., Natarajan, M.: An Optimized Machine Learning Model for Tomato Leaf and Fruit Disease Detection using Kernel Extreme Learning Machine (KELM) Model with Firefly Optimization. Int. J. Intell. Syst. Appl. Eng. 11, 772–788 (2023).

21. Abdulridha, J., Ampatzidis, Y., Kakarla, S.C., Roberts, P.: Detection of target spot and bacterial spot diseases in tomato using UAV-based and benchtop-based hyperspectral imaging techniques. Precis. Agric. 21, 955–978 (2020). https://doi.org/10.1007/s11119-019-09703-4.

22. Foolad, M.R., Merk, H.L., Ashrafi, H.: Genetics, genomics and breeding of late blight and early blight resistance in tomato. CRC. Crit. Rev. Plant Sci. 27, 75–107 (2008). https://doi.org/10.1080/07352680802147353.

23. Paul, S.G., Biswas, A.A., Saha, A., Zulfiker, M.S., Ritu, N.A., Zahan, I., Rahman, M., Islam, M.A.: A real-time application-based convolutional neural network approach for tomato leaf disease classification. Array. 19, 100313 (2023). https://doi.org/10.1016/j.array.2023.100313.

24. Concepcion, R., Lauguico, S., Dadios, E., Bandala, A., Sybingco, E., Alejandrino, J.: Tomato septoria leaf spot necrotic and chlorotic regions computational assessment using artificial bee colony-optimized leaf disease index. In: IEEE Region 10 Annual International Conference, Proceedings/TENCON. pp. 1243–1248. IEEE, Osaka (2020). https://doi.org/10.1109/TENCON50793.2020.9293743.

25. Qasrawi, R., Amro, M., Zaghal, R., Sawafteh, M., Polo, S.V.: Machine Learning Techniques for Tomato Plant Diseases Clustering, Prediction and Classification. In: Proceedings - 2021 International Conference on Promising Electronic Technologies, ICPET 2021. pp. 40–45.

IEEE, Deir El-Balah (2021). https://doi.org/10.1109/ICPET53277.2021.00014.

26. Gowri, S.G., Devi, R., Sethuraman, K.: Machine Learning. Int. J. Res. Anal. Rev. 6, 197–208 (2019).

27. Mohana Saranya, S., Rajalaxmi, R.R., Prabavathi, R., Suganya, T., Mohanapriya, S., Tamilselvi, T.: Deep Learning Techniques in Tomato Plant - A Review. J. Phys. Conf. Ser. 1767, (2021). https://doi.org/10.1088/1742-6596/1767/1/012010.

28. Chamli Deshan, L.A., Hans Thisanke, M.K., Herath, D.: Transfer Learning for Accurate and Efficient Tomato Plant Disease Classification Using Leaf Images. In: 2021 IEEE 16th International Conference on Industrial and Information Systems, ICIIS 2021. pp. 168–173. IEEE, Kandy (2021). https://doi.org/10.1109/ICIIS53135.2021.9660681.

29. Sahu, M., Dash, R.: A Survey on Deep Learning: Convolution Neural Network (CNN). In: Mishra, D., Buyya, R., Mohapatra, P., and Patnaik, S. (eds.) Intelligent and Cloud Computing 2019. pp. 317–325. Springer, Singapore (2021). https://doi.org/10.1007/978-981-15-6202-0_32.

30. Cong, S., Zhou, Y.: A review of convolutional neural network architectures and their optimizations. Artif. Intell. Rev. 56, 1905–1969 (2023). https://doi.org/10.1007/s10462-022-10213-5.

31. Albawi, S., Mohammed, T.A., Al-Zawi, S.: Understanding of a convolutional neural network. In: Proceedings of 2017 International Conference on Engineering and Technology, ICET 2017. pp. 1–6. IEEE, Antalya (2017). https://doi.org/10.1109/ICEngTechnol.2017.8308186.

32. Wiranata, A., Wibowo, S.A., Patmasari, R., Rahmania, R., Mayasari, R.: Investigation of Padding Schemes for Faster R-CNN on Vehicle Detection. In: Proceedings - 2018 International Conference on Control, Electronics, Renewable Energy and Communications, ICCEREC 2018. pp. 208–212. IEEE, Bandung (2018). https://doi.org/10.1109/ICCEREC.2018.8712086.

33. Galanis, N.I., Vafiadis, P., Mirzaev, K.G., Papakostas, G.A.: Convolutional Neural Networks: A Roundup and Benchmark of Their Pooling Layer Variants. Algorithms. 15, (2022). https://doi.org/10.3390/a15110391.

34. Hao, W., Yizhou, W., Yaqin, L., Zhili, S.: The Role of Activation Function in CNN. In: Proceedings - 2020 2nd International Conference on Information Technology and Computer Application, ITCA 2020. pp. 429–432. IEE, Guangzhou (2020). https://doi.org/10.1109/ITCA52113.2020.00096.

35. Yang, C., Yang, Z., Hou, J., Su, Y.: A Lightweight Full Homomorphic Encryption Scheme on Fully-connected Layer for CNN Hardware Accelerator achieving Security Inference. In: 2021 28th IEEE International Conference on Electronics, Circuits, and Systems, ICECS 2021. pp. 1–4. IEEE, Dubai (2021). https://doi.org/10.1109/ICECS53924.2021.9665520.

36. Xie, N., Li, X., Li, K., Yang, Y., Shen, H.T.: Statistical Karyotype Analysis Using CNN and Geometric Optimization. IEEE Access. 7, 179445–179453 (2019). https://doi.org/10.1109/ACCESS.2019.2951723.

37. Sato, N., Fukuyama, Y., Iizaka, T., Matsui, T.: A Correntropy Based Artificial Neural Network using Early Stopping for Daily Peak Load Forecasting. In: 2020 59th Annual Conference of the Society of Instrument and Control Engineers of Japan, SICE 2020. pp. 581–586. IEEE, Chiang Mai (2020). https://doi.org/10.23919/sice48898.2020.9240336.

38. Hui, T.W., Tang, X., Loy, C.C.: A Lightweight Optical Flow CNN - Revisiting Data Fidelity

and Regularization. IEEE Trans. Pattern Anal. Mach. Intell. 43, 2555–2569 (2021). https://doi.org/10.1109/TPAMI.2020.2976928.

39. Dileep, P., Das, D., Bora, P.K.: Dense layer dropout based CNN architecture for automatic modulation classification. In: 26th National Conference on Communications, NCC 2020. pp. 1–5. IEEE, Kharagpur (2020). https://doi.org/10.1109/NCC48643.2020.9055989.

40. Wang, S.H., Hong, J., Yang, M.: Sensorineural hearing loss identification via nine-layer convolutional neural network with batch normalization and dropout. Multimed. Tools Appl. 79, 15135–15150 (2020). https://doi.org/10.1007/s11042-018-6798-3.

41. Mohanty, S.P., Hughes, D.P., Salathé, M.: Using deep learning for image-based plant disease detection. Front. Plant Sci. 7, 1–10 (2016). https://doi.org/10.3389/fpls.2016.01419.

42. Reda, M., Suwwan, R., Alkafri, S., Rashed, Y., Shanableh, T.: AgroAId: A Mobile App System for Visual Classification of Plant Species and Diseases Using Deep Learning and TensorFlow Lite. Informatics. 9, (2022). https://doi.org/10.3390/informatics9030055.

43. Baser, P., Saini, J.R., Kotecha, K.: TomConv: An Improved CNN Model for Diagnosis of Diseases in Tomato Plant Leaves. In: International Conference on Machine Learning and Data Engineering. pp. 1825–1833. Elsevier B.V., Dehradun (2023). https://doi.org/10.1016/j.procs.2023.01.160.

44. He, K., Zhang, X., Ren, S., Sun, J.: Deep Residual Learning for Image Recognition. In: Computer Vision and Pattern Recognition (CVPR), 2016. pp. 770–778. IEEE, Las Vegas (2016). https://doi.org/10.1109/CVPR33180.2016.

45. Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J.: Rethinking the Inception Architecture for Computer Vision. In: Computer Vision and Pattern Recognition (CVPR), 2016. pp. 2818–2826. IEEE, Las Vegas (2016). https://doi.org/10.1109/CVPR33180.2016.

46. Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., Rabinovich, A.: Going deeper with convolutions. In: Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition. pp. 1–9. IEEE, Boston (2015). https://doi.org/10.1109/CVPR.2015.7298594.

47. Tan, M., Le, Q. V: EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks. In: Chaudhuri, K. and Salakhutdinov, R. (eds.) 36th International Conference on Machine Learning. pp. 6105–6114. PMLR, California (2019).

48. Agnihotri, S., Gupta, J., Garg, N., Khatri, P.: Comparative Analysis of Tomato Leaf Disease Detection Using Machine Learning. In: 2023 6th International Conference on Information Systems and Computer Networks, ISCON 2023. pp. 1–5. IEEE, Mathura (2023). https://doi.org/10.1109/ISCON57294.2023.10112092.

49. Siva, P.N., Yamaganti, R.: A Review on Python for Data Science, Machine Learning and IOT. Int. J. Sci. Eng. Res. 10, 851–858 (2019).

50. Pang, B., Nijkamp, E., Wu, Y.N.: Deep Learning With TensorFlow: A Review. J. Educ. Behav. Stat. 45, 227–248 (2020). https://doi.org/10.3102/1076998619872761.

51. Begum, N., Hazarika, M.K.: Maturity detection of tomatoes using transfer learning. Meas. Food. 7, (2022). https://doi.org/10.1016/j.meafoo.2022.100038.

List of Publications

- Rana, A., Goel, N.: Detection and Classification of Tomato Crop Disease using Deep Learning Models with varied optimization techniques. In: Congress of Intelligent Systems (CIS)., Bengaluru, Springer (2024). (**Accepted & Presented**)