DISEASE IDENTIFICATION IN TOMATO LEAVES USING INCEPTION V3 CONVOLUTIONAL NEURAL NETWORKS

A Capstone project report submitted

in partial fulfillment of requirement for the award of degree

BACHELOR OF TECHNOLOGY

in

ELECTRONICS & COMMUNICATION ENGINEERING

by

N.BHAVITH	**19K41A0450**
D.KONDAL RAO	**20K45A0403**
RAGHAV BANG	**19K41A0454**

Under the guidance of

Mr. S Srinivas

Asst. Prof, Department of ECE.

SR Engineering College

Innovation . Creativity . Entrepreneurship

SR ENGINEERING COLLEGE

Ananthasagar, Warangal.

CERTIFICATE

This is to certify that this project entitled **"DISEASE IDENTIFICATION IN TOMATO LEAVES USING INCEPTION V3 CONVOLUTIONAL NEURAL NETWORKS**" is the bonafide work carried out by **N.BHAVITH**, **D.KONDAL RAO and RAGHAV BANG** as a Capstone project for the partial fulfillment to award the degree **BACHELOR OF TECHNOLOGY** in **ELECTRONICS & COMMUNICATION ENGINEERING** during the academic year 2022-2023 under our guidance and Supervision.

Mr. S Srinivas
Asst. Professor (ECE),
S R Engineering College,
Ananthasagar, Warangal.

Dr. Sandip Bhattacharya
Assoc. Prof. & HOD (ECE),
S R Engineering College,
Ananthasagar, Warangal.

ACKNOWLEDGEMENT

We wish to take this opportunity to express our sincere gratitude and deep sense of respect to our beloved Principal, **Dr. V. MAHESH,** for his continuous support and guidance to complete this project in the institute.

We express our thanks to Head of the ECE Department **Dr. Sandip Bhattacharya, Associate Professor,** for his encouragement and support.

We owe an enormous debt of gratitude to our project guide **Mr. S Srinivas, Asst. Professor** as well as project co-ordinators **Mr. S. Srinivas, Asst. Prof., Dr.R. Shashank, Asst. Prof.**, for guiding us from the beginning through the end of the Capstone project with their intellectual advices and insightful suggestions. We truly value their consistent feedback on our progress, which was always constructive, encouraging and ultimately drove us to the right direction.

Finally, we express our thanks to all the teaching and non-teaching staff of the department for their suggestions and timely support.

ABSTRACT

Tomatoes are the most widely grown vegetable, used in a wide variety of dishes around the world. After potatoes and sweet potatoes, it is the third most extensively cultivated crop in the world. However, due to several diseases, both the quality and quantity of tomato harvests decline. To maximize tomato yields, it is important to identify and eradicate the many diseases that harm the crop as early as possible. In this paper, we investigate the potential of deep learning techniques for diagnosing diseases on tomato leaves. The use of automatic methods for tomato leaf disease detection is helpful because it reduces the amount of monitoring needed in large-scale crop farms and does so at a very early stage when the signs of the disease identified on plant leaves are still easy to cure. The Kaggle dataset for tomato leaf disease was used for the study. A technique based on convolutional neural networks is used for disease identification and classification. Deep learning models, such as Inception V3 are used in this work. This proposed system obtained an accuracy of 99.60%, suggesting that the neural network approach is effective even under difficult situations.

CONTENTS

LIST OF TABLES

LIST OF ACRONYMS

ACRONYM	ABBREVIATION
AI	Artificial Intelligence
ANN	Artificial Neural Network
CNN	Convolutional Neural Network
FC	Fully Connected
ML	Machine learning
SVM	Support Vector Machine

CHAPTER 1

INTRODUCTION

1.1 OVERVIEW OF THE PROJECT

India is a country with a majority of the population relying heavily on the agricultural sector. Tomato is the most common vegetable used across India. The three most important antioxidants namely vitamin E, vitamin C and beta-carotene are present in tomatoes. They are also rich in potassium, a very important mineral for good health. Tomato crop cultivation area in India spans around 3,50,000 hectares approximately and the production quantities roughly sum up to 53,00,000 tons, making India the third largest tomato producer in the world. The sensitivity of crops coupled with climatic conditions have made diseases common in the tomato crop during all the stages of its growth. Disease affected plants constitute 10-30% of the total crop loss. Identification of such diseases in the plant is very important in preventing any heavy losses in yield as well as the quantity of the agricultural product. Monitoring the plant diseases manually is a difficult task due to its complex nature and is a time consuming process. Therefore, there is a need to reduce the manual effort put into this task, while making accurate predictions and ensuring that the farmers' lives are hassle free.

Visually observable patterns are difficult to decipher at a single glance, leading to many farmers making inaccurate assumptions regarding the disease. As a result, prevention mechanisms taken by the farmers may be ineffective and sometimes harmful. Farmers usually come together and implement common disease prevention mechanisms, as they lack expert advice on how to deal with their crop infestation. There has been circumstances where due to inadequate knowledge or misinterpretation regarding the intensity of the disease, over-dosage or under-dosage of the pesticide has resulted in crop damage. This is the underlying motivation for the proposed methodology that aims to accurately detect and classify diseases in the tomato crop.

Monitoring plant health condition and detecting symptoms early are necessary to reduce disease spread which helps the farmers in effective management practices and helps to increase productivity. Thus, disease diagnosis in crops plays a major role in preserving agriculture productivity. In general, the plant disease analysis is performed manually or by visual observation or through a microscope, which are in fact slow and there might be a

threat of error. Hence, diagnosis of crop disease via some automatic techniques is valuable as it decreases a huge work of monitoring of crops in large farms, and at the very initial stage itself it senses the symptoms of diseases, i.e. when they appear on crop leaves. So, we need such techniques where we can easily detect the disease in plants using their leaf images which can accurately classify them according to the disease the plant is suffering and there should be a minimum error.

The machine learning image processing technology is developing rapidly and is widely used in all aspects, including the agricultural field. Applying machine learning and image processing technology to crop disease recognition has incomparable advantages over traditional manual diagnosis and recognition methods. People only need to collect a small number of disease image samples. The process involves the following steps: firstly, the dataset is pre-processed; secondly, the feature extraction algorithm is used to extract the features of the disease area in the image; lastly, the obtained feature information is sent to the classifier for training and the model parameters are obtained. The generated model can be used to detect the disease category. Training with a large number of datasets is time consuming due to the lack of datasets and the poor generalization ability of the model. Moreover, the development of agricultural modernization towards the direction of intelligence has highlighted the shortcomings of these traditional image detection methods.

The development of new technology has enabled not only discovering the characteristics of things artificially but also collecting a large number of data, designing algorithms and programming, mining laws from data and building models by using advanced computer hardware facilities. Deep learning is a representative branch of artificial intelligence. Although this concept was only introduced in 2012, various network models have been produced after its development. At present, deep learning is widely used in various fields, especially in computer vision. It efficiently solves the tasks of image classification, object detection and semantic segmentation. Compared with the traditional pattern detection method, the disease detection method based on the deep convolutional network (CNN) abandons the sophisticated image pre-processing and feature extraction operations and uses the end-to-end structure to simplify the detection process. CNN can be used to train the model prediction results, which not only can save time and workforce but also can make a real-time judgment, as long as a large number of crop disease image datasets can be obtained. This way CNN models can greatly reduce the great losses caused by the disease.

The methodology suggested in the project pertains to the most common diseases found in the tomato plant like, Bacterial leaf spot and Septorial leaf spot, Yellow Leaf Curl among many others. Any leaf image given as input can be classified into one of the disease classes or can be deemed healthy. The database used for evaluation is a subset of Plant Village, a repository that contains 54,306 images of 14 crops infested with 26 diseases. The subset includes around 18160 images of tomato leaf diseases.

Broadly, the proposed methodology consists of three major steps: data acquisition, preprocessing and classification. The images used for the implementation of the proposed methodology were acquired from a publicly available dataset called Plant Village, as mentioned earlier. In the next step, the images were re-sized to a standard size before feeding it into the classification model. The final step is the classification of the input images with the use of a slight variation of the deep learning convolutional neural network (CNN) standard model called the Inception V3 which consists of the convolutional, activation, pooling and fully connected layers.

CHAPTER 2
LITERATURE SURVEY

2.1 EXISTING METHODS

Machine learning algorithms are applied in various fields, but feature engineering remains the main problem. With the emergence of deep neural network, the promising results are available for plant pathology without laborious feature engineering. Deep neural networks significantly increase the image classification accuracy. This section provides a various deep learning technique used by researchers in plant disease identification.

Mohit Agarwal [1] in this work, discussed about tomato problems may be divided into two sections: bacteria or fungi or poor cultivation habits causing 16 diseases while insects causing 5 other type of diseases. Ralstonia solanacearum bacteria causes serious form of Bacterial wilt. This bacterium can survive in soil for long time period and enter roots through natural wounds made during secondary roots emergence or man made during cultivating or transplanting or even insects. High moisture and high temperature favors disease development. The bacteria fills, water conducting tissue of plant, with slime by multiplying rapidly inside it. This results in affecting the vascular system of plant, while the leaves may stay green. On a cross section view of an infected plant stem, it appears brown with yellowish material coming out of it. They developed a CNN based model to detect the disease in tomato crop. In the proposed CNN based architecture there are 3 convolution and max pooling layers with varying number of filters in each layer. For the experiment purpose, they have taken the tomato leaf data from Plant Village dataset. In the dataset there are 9 disease classes and class which is having the healthy images. As the images inside class is not balanced, so that data augmentation techniques have been applied to balance the images inside the class. Experimentally, it is observed the testing accuracy of the model is ranging from 76% to 100% for the classes. Moreover, the average testing accuracy of the model is 91.2%.

Qimei Wang [2] in this study, developed tomato disease detection architectures based on deep convolutional neural networks and object detection models are proposed. Four different neural networks were selected and combined with two object detection models. Experiments with data collected from the Internet show that the proposed methods are very accurate and efficient in detecting tomato disease types and in segmenting shapes

of infected areas. Experiment results indicate that ResNet-101 has the highest detection rate, but takes the longest time for training and detection. MobileNet has the shortest detection time, but is less accurate than ResNet-101. In general, different models can be chosen according to the actual needs. Dataset in this study also includes a variety of complex backgrounds, allowing the architectures to improve their abilities to recognize complex images. Two object detection architectures, i.e., Faster R-CNN and Mask R-CNN, are combined with four different deep convolutional neural networks. Deep convolutional neural networks are used to automatically extract original image features, and object detection architectures are used to identify, classify, and locate diseased sites in feature maps two object detection architectures are used for different purposes of Faster R-CNN is to identify and locate diseased tomatoes, while that of Mask R-CNN is to segment specific lesion areas on diseased tomatoes.

Jun Liu [3] in this paper, improved Yolo V3 algorithm and proposed to detect tomato diseases and insect pests. Yolo V3 network was improved by using multi-scale feature detection based on image pyramid, object bounding box dimension clustering and multi-scale training. The experimental results show that the detection accuracy of the algorithm is 92.39% and the detection time is only 20.39 ms. Therefore, for the task of tomato diseases and pests detection, the improved Yolo V3 algorithm proposed in this paper can not only maintain a high detection rate, but also meet the real-time detection requirements, and can detect the location and category of tomato diseases and insect pests accurately and quickly. The image data of tomato diseases and insect pests under real natural environment were collected, and corresponding data processing was carried out to build tomato diseases and pests detection dataset. Based on the YOLO v3 model as the main body, the image pyramid structure is adopted to fuse features of different levels to obtain feature maps of different scales for location and category prediction. Then, the dimension of the object box is clustered, and the number of anchor box is increased, so that the model can obtain more edge information of the object. Finally, in the training process, multi-size images are used for training, so that the model can adapt to images of different resolutions. Experiments show that the improved YOLO v3 algorithm can improve the detection speed while ensuring the detection accuracy.

Prajwala TM [4] in this paper, proposed a methodology that uses a convolutional neural network model to classify tomato leaf diseases obtained from the Plant Village dataset. The architecture used is a simple convolutional neural network with minimum

number of layers to classify the tomato leaf diseases into 10 different classes. Thus, the above mentioned model can be made use of as a decision tool to help and support farmers in identifying the diseases that can be found in the tomato plant. With an accuracy of 94-95% the methodology proposed can make an accurate detection of the leaf diseases with little computational effort. Broadly, the proposed methodology consists of three major steps: data acquisition, pre-processing and classification. The images used for the implementation of the proposed methodology were acquired from a publicly available dataset called Plant Village, as mentioned earlier. In the next step, the images were re-sized to a standard size before feeding it into the classification model. The final step is the classification of the input images with the use of a slight variation of the deep learning convolutional neural network (CNN) standard model called the LeNet which consists of the convolutional, activation, pooling and fully connected layers.

Xuewei Wang [5] in this paper, proposed an early recognition method of tomato leaf spot based on MobileNetv2-YOLOv3 model to achieve a good balance between the accuracy and real-time detection of tomato gray leaf spot. This method improves the accuracy of the regression box of tomato gray leaf spot recognition by introducing the GIoU bounding box regression loss function. A MobileNetv2-YOLOv3 lightweight network model, which uses MobileNetv2 as the backbone network of the model, is proposed to facilitate the migration to the mobile terminal. The pre-training method combining mix-up training and transfer learning is used to improve the generalization ability of the model. The images captured under four different conditions are statistically analyzed. The recognition effect of the models is evaluated by the F1 score and the AP value, and the experiment is compared with Faster-RCNN and SSD models. Experimental results show that the recognition effect of the proposed model is significantly improved. In the test dataset of images captured under the background of sufficient light without leaf shelter, the F1 score and AP value are 94.13% and 92.53%, and the average IOU value is 89.92%. In all the test sets, the F1 score and AP value are 93.24% and 91.32%, and the average IOU value is 86.98%. The object detection speed can reach 246 frames/s on GPU, the extrapolation speed for a single 416×416 picture is 16.9 ms, the detection speed on CPU can reach 22 frames/s, the extrapolation speed is 80.9 ms and the memory occupied by the model is 28 MB.

Naresh K. Trivedi [6] in this article, discussed about tomato is one of the most essential and consumable crops in the world. Tomatoes differ in quantity depending on how

they are fertilized. Leaf disease is the primary factor impacting the amount and quality of crop yield. As a result, it is critical to diagnose and classify these disorders appropriately. Different kinds of diseases influence the production of tomatoes. Earlier identification of these diseases would reduce the disease's effect on tomato plants and enhance good crop yield. Different innovative ways of identifying and classifying certain diseases have been used extensively. The motive of work is to support farmers in identifying early-stage diseases accurately and informing them about these diseases. The Convolutional Neural Network (CNN) is used to effectively define and classify tomato diseases. Google Colab is used to conduct the complete experiment with a dataset containing 3000 images of tomato leaves affected by nine different diseases and a healthy leaf. The complete process is described: Firstly, the input images are preprocessed, and the targeted area of images are segmented from the original images. Secondly, the images are further processed with varying hyper-parameters of the CNN model. Finally, CNN extracts other characteristics from pictures like colors, texture, and edges, etc. The findings demonstrate that the proposed model predictions are 98.49% accurate.

Vijai Singh [7] in this paper presents the survey on different diseases classification techniques used for plant leaf disease detection and an algorithm for image segmentation technique that can be used for automatic detection as well as classification of plant leaf diseases later. Banana, beans, jackfruit, lemon, mango, potato, tomato, and sapota are some of those ten species on which proposed algorithm is tested. Therefore, related diseases for these plants were taken for identification. With very less computational efforts the optimum results were obtained, which also shows the efficiency of proposed algorithm in recognition and classification of the leaf diseases. Another advantage of using this method is that the plant diseases can be identified at early stage or the initial stage. Agricultural productivity is something on which economy highly depends. This is the one of the reasons that disease detection in plants plays an important role in agriculture field, as having disease in plants are quite natural. If proper care is not taken in this area then it causes serious effects on plants and due to which respective product quality, quantity or productivity is affected. For instance a disease named little leaf disease is a hazardous disease found in pine trees in United States. Detection of plant disease through some automatic technique is beneficial as it reduces a large work of monitoring in big farms of crops, and at very early stage itself it detects the symptoms of diseases i.e. when they appear on plant leaves. This paper presents an algorithm for image segmentation technique which is used for automatic detection and

classification of plant leaf diseases. It also covers survey on different diseases classification techniques that can be used for plant leaf disease detection. Image segmentation, which is an important aspect for disease detection in plant leaf disease, is done by using genetic algorithm.

Xuewei Sun [8] in this paper, identified traditional digital image processing methods extract disease features manually, which have low efficiency and low recognition accuracy. To solve this problem. They proposed a convolutional neural network architecture FL-EfficientNet (Focal loss EfficientNet), which is used for multi-category identification of plant disease images. Firstly, through the Neural Architecture Search technology, the network width, network depth, and image resolution are adaptively adjusted according to a group of composite coefficients, to improve the balance of network dimension and model stability; Secondly, the valuable features in the disease image are extracted by introducing the moving flip bottleneck convolution and attention mechanism; Finally, the Focal loss function is used to replace the traditional Cross-Entropy loss function, to improve the ability of the network model to focus on the samples that are not easy to identify. The experiment uses the public data set new plant diseases dataset (NPDD) and compares it with ResNet50, DenseNet169, and EfficientNet. The experimental results show that the accuracy of FL-EfficientNet in identifying 10 diseases of 5 kinds of crops is 99.72%, which is better than the above comparison network. At the same time, FL-EfficientNet has the fastest convergence speed, and the training time of 15 epochs is 4.7 h

Aditya Khamparia [9] in this paper discussed that, agriculture plays a significant role in the growth and development of any nation's economy. But, the emergence of several crop-related diseases affects the productivity in the agriculture sector. To cope up this issue and to make aware the farmers to prevent the expansion of diseases in crops and to implement effective management, crop disease diagnosis plays its significant role. Researchers had already used many techniques for this purpose, but some vision-related techniques are yet to be explored. Commonly used techniques are support vector machine, k-means clustering, radial basis functions, genetic algorithm, image processing techniques like filtering and segmentation, deep structured learning techniques like convolutional neural network. We have designed a hybrid approach for detection of crop leaf diseases using the combination of convolutional neural networks and autoencoders. This research paper provides a novel technique to detect crop diseases with the help of convolutional encoder networks using crop leaf images. We have obtained our result over a 900-image

dataset, out of which 600 constitute the training set and 300 test set. We have considered 3 crops and 5 kinds of crop diseases. The proposed network was trained in such a way that it can distinguish the crop disease using the leaf images. Different convolution filters like 2×2 and 3×3 are used in proposed work. It was observed that the proposed architecture achieved variation in accuracy for the different number of epochs and for different convolution filter size. We reached 97.50% accuracy for 2×2 convolution filter size in 100 epochs, while 100% accuracy for 3×3 filter size which is better than other conventional methods.

Gianni Fenu [10] in this paper, performed an analysis and classification of forecasting models for plant and crop disease over the past 10 years (2010–2020). Forty-six research works were identified and reviewed, with an examination of the approaches adopted as well as the pre-processing techniques and data used. Issues and concerns were discussed. In this study, the prediction of plant and crop disease is a complex problem to be solved due to the interaction of several environmental and climatic factors. Over the last 10 years, the literature has presented considerable advancements in understanding these dynamic processes by adopting different scientific approaches. They observed, the problem under study requires high-quality, labeled data. However, the lack of open data is slowing the advance of knowledge in this agricultural sub-domain. Indeed, regarding the state of the art, only a limited number of contributions has been presented in the literature from 2010 to today. The majority of these have focused on few pathogens and crops; furthermore, only a few of these have considered data from various heterogeneous sources to predict disease occurrence. These gaps are hindering progress in achieving development goals and creating products that are able to face real world scenarios, and so more effort is required in data collection and in developing novel solutions to prevent and mitigate the impact of crop and plant disease to food production, especially for those crops which represent staple foods for millions of people who live in the least developed countries. They present an analysis and classification of research studies conducted over the past decade that forecast the onset of disease at a pre-symptomatic stage (i.e., symptoms not visible to the naked eye) or at an early stage and examine the specific approaches and methods adopted, pre-processing techniques and data used, performance metrics, and expected results, highlighting the issues encountered. The results of the study reveal that this practice is still in its infancy and that many barriers need to be overcome.

Nagamani H S [11] in this paper, discussed about plant diseases cause low agricultural productivity. Plant diseases are challenging to control and identify by the

majority of farmers. In order to reduce future losses, early disease diagnosis is necessary. This study looks at how to identify tomato plant leaf disease using machine learning techniques, including the Fuzzy Support Vector Machine (Fuzzy-SVM), Convolution Neural Network (CNN), and Region-based Convolution Neural Network (R-CNN). The findings were confirmed using images of tomato leaves with six diseases and healthy samples. Image scaling, color thresholding, flood filling approaches for segmentation, gradient local ternary pattern, and Zernike moments' features are used to train the pictures. R-CNN classifiers are used to classify the illness kind. The classification methods of Fuzzy SVM and CNN are analyzed and compared with R-CNN to determine the most accurate model for plant disease prediction. The R-CNN-based Classifier has the most remarkable accuracy of 96.735 percent compared to the other classification approaches.

Altalak [12] in this paper, discussed traditional disease diagnosis systems for tomato leaves involve high costs and risk of misjudgment and are now supplanted with computer technology, including computer vision, deep learning, and machine learning methods. The proposed a hybrid deep learning approach for detecting and classifying tomato plant leaf diseases early. This hybrid system is a combination of a convolutional neural network (CNN), convolutional attention module (CBAM), and support vector machines (SVM). Initially, the proposed model can detect nine different tomato diseases but is not limited to this. The proposed system is tested using a database containing images of tomato leaves. The obtained results were very encouraging, giving us accuracy up to 97.2%, which can be improved with the improvement of learning processes. The proposed system is very efficient and lightweight, so the farmer can install it on any smart device having a digital camera and processing capabilities. With a bit of training, a farmer can detect any disease immediately, which will help him take timely pre-emptive action.

Karthik [13] in this paper, discussed about automation in plant disease detection and diagnosis is one of the challenging research areas that has gained significant attention in the agricultural sector. Traditional disease detection methods rely on extracting handcrafted features from the acquired images to identify the type of infection. Also, the performance of these works solely depends on the nature of the handcrafted features selected. This can be addressed by learning the features automatically with the help of Convolutional Neural_Networks (CNN). This research presents two different deep architectures for detecting the type of infection in tomato leaves. The first architecture applies residual learning to learn significant features for classification. The second

architecture applies attention mechanism on top of the residual deep network. Experiments were conducted using Plant Village Dataset comprising of three diseases namely early blight, late blight, and leaf mold. The proposed work exploited the features learned by the CNN at various processing hierarchy using the attention mechanism and achieved an overall accuracy of 98% on the validation sets in the 5-fold cross-validation.

Melike Sardogan [14] in this paper, discussed about the early detection of diseases is important in agriculture for an efficient crop yield. The bacterial spot, late blight, septoria leaf spot and yellow curved leaf diseases affect the crop quality of tomatoes. Automatic methods for classification of plant diseases also help taking action after detecting the symptoms of leaf diseases. This paper presents a Convolutional Neural Network (CNN) model and Learning Vector Quantization (LVQ) algorithm based method for tomato leaf disease detection and classification. The dataset contains 500 images of tomato leaves with four symptoms of diseases. We have modeled a CNN for automatic feature extraction and classification. Color information is actively used for plant leaf disease researches. In our model, the filters are applied to three channels based on RGB components. The LVQ has been fed with the output feature vector of convolution part for training the network. The experimental results validate that the proposed method effectively recognizes four different types of tomato leaf diseases.

Iftikhar Ahmad [15] in this paper, discussed about vegetable and fruit plants facilitate around 7.5 billion people around the globe, playing a crucial role in sustaining life on the planet. The rapid increase in the use of chemicals such as fungicides and bactericides to curtail plant diseases is causing negative effects on the agro-ecosystem. The high scale prevalence of diseases in crops affects the production quantity and quality. Solving the problem of early identification/diagnosis of diseases by exploiting a quick and consistent reliable method will benefit the farmers. In this context, our research work focuses on classification and identification of tomato leaf diseases using convolutional neural network (CNN) techniques. We consider four CNN architectures, namely, VGG-16, VGG-19, ResNet, and Inception V3, and use feature extraction and parameter-tuning to identify and classify tomato leaf diseases. We test the underlying models on two datasets, a laboratory-based dataset and self-collected data from the field. We observe that all architectures perform better on the laboratory-based dataset than on field-based data, with performance on various metrics showing variance in the range 10%–15%. Inception V3 is identified as the best performing algorithm on both datasets.

Nikita Sareen [16] in this paper, discussed about the global economic system directly depends on the agricultural and farming industry. However, plant diseases are a peril to crops that deteriorates the quality as well as quantity of agricultural produce which pose huge loss to the economy. Therefore, early detection of diseases is crucial to prevent the wrecking of crops. The main aim of this study is to detect the early blight disease in tomato plants beforehand using a prominent deep learning approach, i.e., Convolutional Neural Network (CNN). The current study uses an image-based Tomato Early Blight Disease (TEBD) dataset to develop a disease prediction model. Various image processing techniques, i.e., Background Removal, Augmentation, Resizing, Noise Removal, and Segmentation were applied to obtain a refined dataset. Further, the modified TEBD dataset was trained using the CNN approach to develop an image-based TEBD prediction model. The impact of hyper parameters such as epochs and mini-batch size were also analyzed for different train-test ratios. Afterwards, the performance of the proposed model was precisely evaluated using various performance metrics, i.e., Accuracy, F1-Score, Test loss, Precision, and Recall. The mean value was calculated of different performance metrics for all nine splitting ratios. The best mean accuracy achieved was 98.10% having batch-size 64 and 15 epochs. This model can be used by farmers to reduce the workload and thus will become feasible for them in early treatment and curing of disease. In this way, the proposed approach will prevent the plants from getting severely affected in the early stage.

Garima Shrestha [17] in this paper, discussed about the study of agricultural productivity is a valuable component for the growth of Indian economy. Therefore the contribution of food crops and cash crops is highly important for both the environment and human beings of this country. Every year crops succumb to several diseases. Due to inadequate diagnosis of such diseases and not knowing symptoms of the disease and its treatment many plants die. This study provides insights into an overview of the plant disease detection using different algorithms. A CNN based method for plant disease detection has been proposed here. Simulation study and analysis is done on sample images in terms of time complexity and the area of the infected region. It is done by image processing technique. A total of 15 cases have been fed to the model, out of which 12 cases are of diseased plant leaves namely, Bell Paper Bacterial Spot, Potato Early Blight, Potato Late Blight, Tomato Target Spot, Tomato Mosaic Virus, Tomato Yellow Leaf Curl Virus, Tomato Bacterial Spot, Tomato Early Blight, Tomato Late Blight, Tomato Leaf Mold, Tomato Septoria Leaf Spot and Tomato Spider Mites and 3 cases of healthy leaves namely,

Bell Paper Healthy, Potato Healthy and Tomato Healthy. Test accuracy is obtained as 88.80%. Different performance matrices are derived for the same.

Hsing-Chung Chen [18] in this paper, about limited retrieval of reserves and restricted capability in plant pathology, automation of processes becomes essential. All over the world, farmers are struggling to prevent various harm from bacteria or pathogens such as viruses, fungi, worms, protozoa, and insects. Deep learning is currently widely used across a wide range of applications, including desktop, web, and mobile. In this study, the authors attempt to implement the function of AlexNet modification architecture-based CNN on the Android platform to predict tomato diseases based on leaf image. A dataset with of 18,345 training data and 4,585 testing data was used to create the predictive model. The information is separated into ten labels for tomato leaf diseases, each with 64 × 64 RGB pixels. The best model using the Adam optimizer with a realizing rate of 0.0005, the number of epochs 75, batch size 128, and an uncompromising cross-entropy loss function, has a high model accuracy with an average of 98%, a strictness rate of 0.98, a recall value of 0.99, and an F1-count of 0.98 with a loss of 0.1331, so that the classification results are good and very precise.

Shengyi Zhao [19] in this paper, discussed about how the world economic system directly or indirectly depends on the agricultural sector and farming industry about crop disease diagnosis is of great significance to crop yield and agricultural production. Deep learning methods have become the main research direction to solve the diagnosis of crop diseases. This paper proposed a deep convolutional neural network that integrates an attention mechanism, which can better adapt to the diagnosis of a variety of tomato leaf diseases. The network structure mainly includes residual blocks and attention extraction modules. The model can accurately extract complex features of various diseases. Extensive comparative experiment results show that the proposed model achieves the average identification accuracy of 96.81% on the tomato leaf diseases dataset. It proves that the model has significant advantages in terms of network complexity and real-time performance compared with other models. Moreover, through the model comparison experiment on the grape leaf diseases public dataset, the proposed model also achieves better results, and the average identification accuracy of 99.24%. It is certified that add the attention module can more accurately extract the complex features of a variety of diseases and has fewer parameters. The proposed model provides a high-performance solution for crop diagnosis under the real agricultural environment.

Amreen Abbas [20] in this paper, discussed about plant diseases and pernicious insects are a considerable threat in the agriculture sector. Therefore, early detection and diagnosis of these diseases are essential. The ongoing development of profound deep learning methods has greatly helped in the detection of plant diseases, granting a vigorous tool with exceptionally precise outcomes but the accuracy of deep learning models depends on the volume and the quality of labeled data for training. They have developed a deep learning based framework to recognize the tomato plant diseases by investigating tomato leaf images. This method would help farmers in classification of diseases affecting tomato cultivation by simply taking an image of diseased leaves, instead of going after costly expert analysis. In the proposed method, we generated synthetic images using Conditional Generative Adversarial Network (C-GAN) (Mirza and Osindero, 2014) and augmented these images to the dataset for training purpose. Thereafter a DenseNet model (Huang et al., 2017) was trained on tomato plant images and tomato synthetic images (generated by the C-GAN model) for the detection of tomato disease from the leaf images. They have proposed a deep learning-based method for tomato disease detection that utilizes the Conditional Generative Adversarial Network (C-GAN) to generate synthetic images of tomato plant leaves. Thereafter, a DenseNet121 model is trained on synthetic and real images using transfer learning to classify the tomato leaves images into ten categories of diseases. The proposed model has been trained and tested extensively on publicly available PlantVillage dataset. The proposed method achieved an accuracy of 99.51%, 98.65%, and 97.11% for tomato leaf image classification into 5 classes, 7 classes, and 10 classes, respectively. The proposed approach shows its superiority over the existing methodologies.

Sunil S. Harakannanavar [21] in this paper, discussed about agriculture provides food to all the human beings even in case of rapid increase in the population. It is recommended to predict the plant diseases at their early stage in the field of agriculture is essential to cater the food to the overall population. But it unfortunate to predict the diseases at the early stage of the crops. The idea behind the paper is to bring awareness amongst the farmers about the cutting-edge technologies to reduce diseases in plant leaf. Since tomato is merely available vegetable, the approaches of machine learning and image processing with an accurate algorithm is identified to detect the leaf diseases in the tomato plant. In this investigation, the samples of tomato leaves having disorders are considered. With these disorder samples of tomato leaves, the farmers will easily find the diseases based on the early symptoms. Firstly, the samples of tomato leaves are resized to 256 × 256 pixels and

then Histogram Equalization is used to improve the quality of tomato samples. The K-means clustering is introduced for partitioning of dataspace into Voronoi cells. The boundary of leaf samples is extracted using contour tracing. The multiple descriptors viz., Discrete Wavelet Transform, Principal Component Analysis and Grey Level Co-occurrence Matrix are used to extract the informative features of the leaf samples. Based on the challenges discussed above and combined techniques using image processing (IP) and ML, the proposed model provide better accuracy. Keeping all these things in mind, in this paper an algorithm based on ML and IP tools to automatically detect leaf diseases is proposed. The contribution for the above proposed framework is done in three stages. Firstly, the HE and K-means clustering are employed to maximize the quality and segment the leaf samples. Based on the K-means clustering response, the leaf is diseased or not can be predicted at the early stage of operation. Secondly, The DWT, PCA and GLCM are used to extract the informative regions/features of the samples. Lastly, as a part of machine learning approaches the SVM, KNN and CNN are used to classify the features. Finally, the extracted features are classified using machine learning approaches such as Support Vector Machine (SVM), Convolutional Neural Network (CNN) and K-Nearest Neighbor (K-NN). The accuracy of the proposed model is tested using SVM (88%), K-NN (97%) and CNN (99.6%) on tomato disordered samples.

T. Anandhakrishnan [22] in this paper, discussed about plant leaf diseases are a major significant risk to food security. In many situation the agriculture production may be reduced, which consequently reduces the nation's economy, if the crops get affected due to diseases. Generally, diseases affect the leaves of the crops which should be identified in the early stage so that the quality and quantity of the produce may be increased. To detect the leaf diseases at an early stage and taking proper remedial actions will be more helpful for the farmers. So there is a need for an automatic system for leaf disease recognition that identifies and classifies the leaf diseases at an early stage. The highlights of the objective work focus on the DCNN models of leaf images used and overall performance according to the performance metrics that are been applied for plant disease identification. During the past decade many researchers are focusing on leaf disease recognition by proposing various methods and techniques using traditional image processing and machine learning techniques. This motivation of the proposed DCNN work is suited for increasing performance accuracy and minimizing response time in the identification of tomato leaf diseases. In this paper we have proposed an automatic system for leaf disease identification

in tomato leaves using Deep Convolutional Neural Network (CNN) since, DCNN focuses on agriculture areas during recent years. In the proposed work we have used 18160 images of tomato leaf diseases which are collected from plant village data set. We have split the dataset that contains 60% of images from the dataset for training and 40% of images for testing. With our proposed DCNN model we have obtained 98.40% of accuracy for the testing set.

Robert G. de Luna [23] in this paper, discussed about smart farming system using necessary infrastructure is an innovative technology that helps improve the quality and quantity of agricultural production in the country including tomato. Since tomato plant farming take considerations from various variables such as environment, soil, and amount of sunlight, existence of diseases cannot be avoided. The recent advances in computer vision made possible by deep learning has paved the way for camera-assisted disease diagnosis for tomato. This study developed the innovative solution that provides efficient disease detection in tomato plants. A motor-controlled image capturing box was made to capture four sides of every tomato plant to detect and recognize leaf diseases. A specific breed of tomato which is Diamante Max was used as the test subject. The system was designed to identify the diseases namely Phoma Rot, Leaf Miner, and Target Spot. Using dataset of 4,923 images of diseased and healthy tomato plant leaves collected under controlled conditions, we train a deep convolutional neural network to identify three diseases or absence thereof. The system used Convolutional Neural Network to identify which of the tomato diseases is present on the monitored tomato plants. The F-RCNN trained anomaly detection model produced a confidence score of 80 % while the Transfer Learning disease recognition model achieves an accuracy of 95.75 %. The automated image capturing system was implemented in actual and registered a 91.67 % accuracy in the recognition of the tomato plant leaf diseases.

Parul Sharma [24] in this paper, discussed about food security for the 7 billion people on earth requires minimizing crop damage by timely detection of diseases. Most deep learning models for automated detection of diseases in plants suffer from the fatal flaw that once tested on independent data, their performance drops significantly. This work investigates a potential solution to this problem by using segmented image data to train the convolutional neural network (CNN) models. As compared to the F-CNN model trained using full images, S-CNN model trained using segmented images more than doubles in performance to 98.6% accuracy when tested on independent data previously unseen by the

models even with 10 disease classes. Not only this, by using tomato plant and target spot disease type as an example, we show that the confidence of self-classification for S-CNN model improves significantly over F-CNN model. This research work brings applicability of automated methods closer to non-experts for timely detection of diseases. They concluded that the performance of the models improved (12% higher on average) in every plant type under investigation with accuracies consistently above 75% in the most complex case with 10 diseases. This research work follows a similar approach: the image datasets are segmented to use maximum information about the disease symptoms by extracting affected parts of the leaves instead of whole leaf images. By quantifying the results of the two models and comparing the self-classification confidence for each disease type, it is shown that we can train using more meaningful data and very good results can be achieved even in real world conditions.

Hsueh-Hung Cheng [25] in this paper, discussed about Tomatoes are an essential crop in Taiwan and in many other countries worldwide. Conventionally, plant disease identification has relied on naked-eye examination in fields by experienced farmers and culture and microscopic examination in laboratories by plant pathologists. This study developed a method for the automatic identification of eight tomato disease and pest categories using images of tomato leaves, convolutional neural networks (CNNs), and a chatbot controller. Approximately 9,000 images of tomato leaves affected by 11 diseases and pests were collected in fields or greenhouses. The images and the respective lesions on the leaves were sorted into eight categories. Three CNN an anomaly detection model (ADM), a disease identification model (DIM), and a leaf mold/powdery mildew II distinguishing model (LPDM) were respectively trained to detect anomalies (i.e., non-leaf images), to sort the images into the eight categories, and to distinguish between leaf mold and powdery mildew II. The three CNNs were hosted on a cloud service. The chatbot controller was programmed to manage the communication between the CNNs and the users through LINE, a mobile instant messaging application. The trained ADM achieved an accuracy of 97.40% in the detection of anomalous images. The trained DIM achieved an accuracy of 93.63% in the categorization of images into the eight tomato disease and pest categories. The trained LPDM achieved an accuracy of 98.70% in the distinction between leaf mold and powdery mildew II. The proposed system can assist farmers in timely identification of tomato leaf diseases and provide simple suggestions for the treatment and prevention of diseases in the identified category.

Geetharamani G [26] in this paper, discussed about a novel plant leaf disease identification model based on a deep convolutional neural network (Deep CNN). This accuracy of the proposed work is greater than the accuracy of traditional machine learning approaches. The proposed model is also tested with respect to its consistency and reliability. This work intends to use a Deep CNN based model to solve the plant leaf disease identification problems. The Deep CNN is a class of deep learning algorithm. The deep learning extends traditional machine learning by adding more complexity and hierarchical data representations into the model. Transfer learning has been used in various applications, such as plant classification, software defect prediction, activity recognition and sentiment classification. In this research, the performance of the proposed Deep CNN model has been compared with popular transfer learning approaches, such as AlexNet, VGG16, Inception-v3 and ResNet. The Deep CNN model is trained using an open dataset with 39 different classes of plant leaves and background images. Six types of data augmentation methods were used: image flipping, gamma correction, noise injection, principal component analysis (PCA) color augmentation, rotation, and scaling. We observed that using data augmentation can increase the performance of the model. The proposed model was trained using different training epochs, batch sizes and dropouts. Compared with popular transfer learning approaches, the proposed model achieves better performance when using the validation data. After an extensive simulation, the proposed model achieves 96.46% classification accuracy.

Lilian Mkonyi [27] in this paper, discussed about the agricultural sector is highly challenged by plant pests and diseases. A high–yielding crop, such as tomato with high economic returns, can greatly increase the income of small holder farmer's income when its health is maintained. This work introduces an approach to strengthen phytosanitary capacity and systems to help solve tomato plant pest Tuta absoluta devastation at early tomato growth stages. We present a deep learning approach to identify tomato leaf miner pest (Tuta absoluta) invasion. This paper introduces transfer learning, a deep learning approach, for identifying the invasion of *T. absoluta* at early stages. The approach reinforces classification of leaf images collected from a field setup in a controlled environment (controlled environment refers to preventing the spread of *T. absoluta* to other neighboring tomato fields using net house). Convolution Neural Network was selected as it performs automatic feature extraction thereby saving experts from the labor-intensive task of feature extraction that usually generates erroneous results hence making it more

accurate and computational efficient. In essence, training a CNN through transfer learning has the ability of improving computational performance by speeding up the training time through reuse of models that were trained on similar tasks. Transfer learning allows the use of fewer data in training a neural network compared to training from scratch that requires large amounts of data. The Convolutional Neural Network architectures (VGG16, VGG19, and ResNet50) were used in training classifiers on tomato image dataset captured from the field containing healthy and infested tomato leaves. We evaluated performance of each classifier by considering accuracy of classifying the tomato canopy into correct category. Experimental results show that VGG16 attained the highest accuracy of 91.9% in classifying tomato plant leaves into correct categories. Our model may be used to establish methods for early detection of Tuta absoluta pest invasion at early tomato growth stages, hence assisting farmers overcome yield losses.

Thair A. Salih [28] in this paper, discussed about the tomato crop is an important staple in the market and it is one of the most common crops daily consumed. Plant or crop diseases cause reduction of quality and quantity of the production; therefore detection and classification of these diseases are very necessary. There are many types of diseases that infect tomato plant like (bacterial spot, late blight, sartorial leaf spot, tomato mosaic and yellow curved). Early detection of plant diseases increases production and improves its quality. It is important to find a smart technology that aims to detect and classify diseases that affect tomato plants with high accuracy. This approach helps the farmers to identify the types of diseases that infect crop. The main object of the current work is to apply a modern technique to identify and classify the disease. Intelligent technique is based on using convolution neural network (CNN) which is a part of machine learning to obtain an early detection about the situation of plants. CNN method depends on feature extraction (such as color, leaves edge, etc.) from input image and on this basis the decision of classification is done. A Matlab m-file has been used to build the CNN structure. A dataset obtained from plant village has been used for training the network (CNN). The suggested neural network has been applied to classify six types of tomato leaves situation (one healthy and five types of leave plant diseases). The results show that the convolution neural network (CNN) has achieved a classification accuracy of 96.43%.

Alvaro F. Fuentes [29] in this paper, discussed about a fundamental problem that confronts deep neural networks is the requirement of a large amount of data for a system to be efficient in complex applications. Promising results of this problem are made possible

through the use of techniques such as data augmentation or transfer learning of pre-trained models in large datasets. But the problem still persists when the application provides limited or unbalanced data. In addition, the number of false positives resulting from training a deep model significantly cause a negative impact on the performance of the system. The system consists of three main units: First, a Primary Diagnosis Unit (Bounding Box Generator) generates the bounding boxes that contain the location of the infected area and class. The promising boxes belonging to each class are then used as input to a Secondary Diagnosis Unit (CNN Filter Bank) for verification. In this second unit, misclassified samples are filtered through the training of independent CNN classifiers for each class. The result of the CNN Filter Bank is a decision of whether a target belongs to the category as it was detected (True) or not (False) otherwise. Finally, an integration unit combines the information from the primary and secondary units while keeping the True Positive samples and eliminating the False Positives that were misclassified in the first unit. By this implementation, the proposed approach is able to obtain a recognition rate of approximately 96%, which represents an improvement of 13% compared to our previous work in the complex task of tomato diseases and pest recognition. Furthermore, our system is able to deal with the false positives generated by the bounding box generator, and class unbalances that appear especially on datasets with limited data. Qualitative and quantitative results show that despite the complexity of real-field scenarios, plant diseases are successfully recognized. The insights drawn from our research helps to better understand the strengths and limitations of plant diseases recognition.

Reesali Mohanty [30] in this paper, discussed about the plants are the primary source of food for the whole population, the disease in plants leads to low crop production and reduces food quality. Early detection of the plant disease can save both time and labor costs. In this paper, our main objective is to create an end-to-end system for detecting tomato disease using the machine learning algorithms like logistic regression, support vector machine, and random forest algorithms. To extract the features from the image Histogram of Oriented gradients is used, and for evaluating the model, classification metrics are used like precision, recall, and f1 score. Out of all these algorithms, the support vector machine performed best by using HOG as a feature descriptor and at the last step, image classification is deployed with the help of Streamlight application.

The comparison of various existing models done by researchers with their respective accuracy values are presented in the Table 1.

Table 1. Comparison of different related works

S.No.	Paper	Model	Accuracy
1	Mohit Agarwal [1]	Custom CNN model Conv. Layer – 5 Pooling Layer – 3	91.2%
2	Jun Liu [3]	Improved YOLO V3	92.39%
3	Prajwala TM [4]	LeNet	94.85%
4	Xuewei Wang [5]	GloU + MobileNet v2 – YOLO v3	94.13%
5	Naresh K. Trivedi [6]	Custom CNN Model Conv. Layer - 8 Pooling layer - 8	98.48%
6	Vijai Singh [7]	SVM	97.6%
7	Aditya khamparia [9]	Custom CNN model Conv. Layers – 8 Pooling layers – 2 Dense layers – 2	86.78%
8	Gianni Fenu [10]	Random Forest	93.33%
9	Nagamani H S [11]	R-CNN	96.73%
10	Altalak [12]	ResNet50-CBAM + SVM	97.2%

2.2 MOTIVATION AND SCOPE OF THE WORK

The farmers generally lack prevention and control experience and measures in production; the disease is often misdiagnosed or not prevented and controlled timely; this condition results in tomato production reduction or crop failure, which causes severe economic losses to farmers. Therefore, tomato leaf diseases should be identified in the early stage, which will be important in avoiding or reducing the economic loss caused by the disease. Therefore, deep learning method is proposed to realize the early recognition of tomato leaf diseases, and there is a need to develop the web app of image detection mobile terminal of tomato leaf diseases to realize real-time detection of this plant.

2.3 PROBLEM STATEMENT

"The quality and quantity of tomato crops go down due to various types of diseases. So, to detect and classify the disease a deep learning-based approach is required."

CHAPTER 3

PROPOSED SOLUTION AND METHODOLOGY

3.1 PROPOSED SOLUTIONS

To address this problem three solutions are proposed using Machine learning techniques like Decision Tree, Random Forest and SVM, Artificial Neural Networks and Deep Learning CNN techniques VGG 16, Custom CNN Architectures, Efficient Net B3 and Inception V3 models.

3.1.1 Tomato leaf disease detection based on machine learning techniques

Machine learning approaches can be used for image classification by training a model to recognize patterns and features in images that correspond to different classes or categories. Here are some steps involved in using machine learning for image classification:

1. Collect and pre-process data: Collect a set of images that represent the different classes you want to classify. Pre-process the data by resizing the images, normalizing the pixel values, and separating the images into training and testing sets.

2. Select a machine learning algorithm: There are several machine learning algorithms that can be used for image classification, such as Convolutional Neural Networks (CNNs), Support Vector Machines (SVMs), and Random Forests. CNNs are the most popular and effective algorithms for image classification.

3. Train the model: Train the selected algorithm using the training set. During training, the algorithm learns to recognize patterns and features in the images that correspond to the different classes. The training process involves optimizing the algorithm's parameters to minimize the error between the predicted and actual classes.

4. Evaluate the model: Evaluate the trained model using the testing set to measure its accuracy and generalization performance. The accuracy of the model can be improved by adjusting the hyper parameters and adding more training data.

5. Use the model for classification: Once the model is trained and evaluated, it can be used for image classification by inputting new images and predicting their class based on the learned patterns and features. All these steps are represented in a form of block diagram in the Fig 3.8.

Decision Tree: A Decision Tree is a supervised learning technique as shown in Fig. 3.1 that can be used for both classification and Regression problems, but mostly it is preferred for solving Classification problems. It is a tree-structured classifier, where internal nodes represent the features of a dataset, branches represent the decision rules and each leaf node represents the outcome. There are two nodes, which are the Decision Node and Leaf Node**.** Decision nodes are used to make any decision and have multiple branches, whereas Leaf nodes are the output of those decisions and do not contain any further branches.

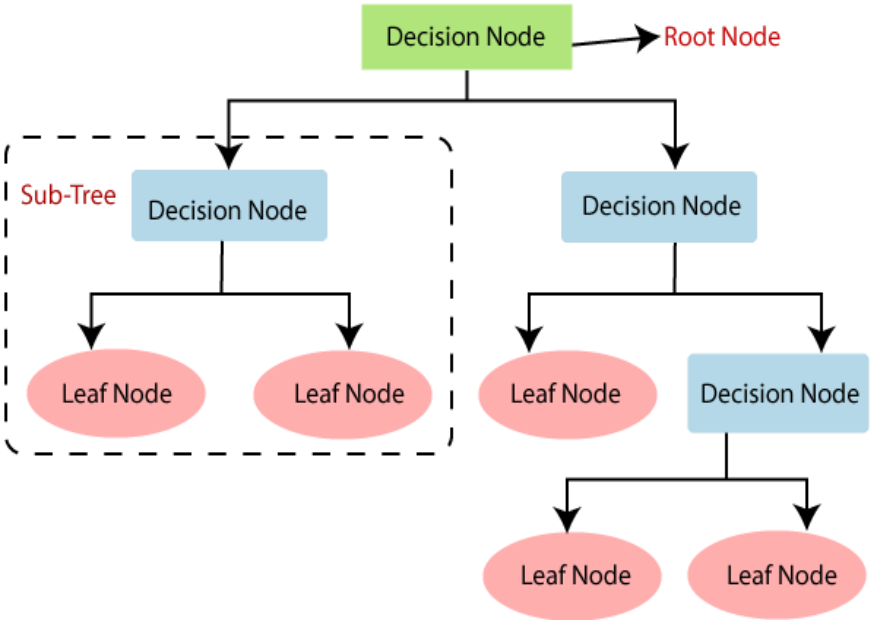


Fig. 3.1 Decision Tree

This method consists of 3 parts:

- Partitioning the nodes
- Finding the terminal nodes
- Allocation of class label to terminal node

Step 1: Begin the tree with the root node, which contains the complete dataset.

Step 2: Find the best attribute in the dataset using the Attribute Selection Measure (ASM).

Step 3: Divide the S into subsets that contains possible values for the best attributes.

Step 4: Generate the decision tree node, which contains the best attribute in the over all model.

Step 5: Recursively make new decision trees using the subsets of the dataset created in step - 3. Continue this process until a stage is reached where you cannot further classify the nodes and call the final node as a leaf node.

Random forest: Random forest is used for both regression and classification-based applications as shown in Fig. 3.2. This algorithm is flexible and easy to use. Most of the time this algorithm gives accurate results even without hyper tuning the parameters. It builds many decision trees which on merging forms as a forest. While building the decision trees, adds more randomness to the model. This algorithm searches for the best feature in the random subset of features, which results in the formation of a better model.

As the name suggests, Random Forest is a classifier that contains a number of decision trees on various subsets of the given dataset and takes the average to improve the predictive accuracy of that dataset. Instead of relying on one decision tree, the random forest takes the prediction from each tree and based on the majority votes of predictions, and it predicts the final output. The greater number of trees in the forest leads to higher accuracy and prevents the problem of overfitting.

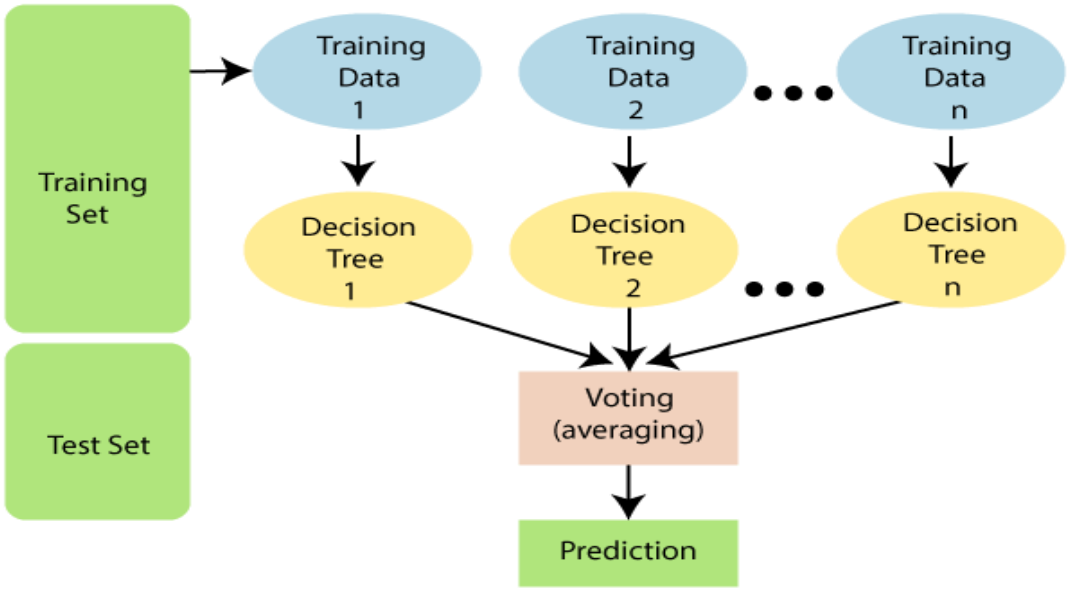


Fig. 3.2 Random forest

Random Forest works in two-phase first is to create the random forest by combining the N decision tree, and the second is to make predictions for each tree created in the first phase. The Working process can be explained in the below steps and diagram:

Step 1: Select random K data points from the training set.

Step 2: Build the decision trees associated with the selected data points (Subsets).

Step 3: Choose the number N for decision trees that you want to build.

Step 4: Repeat Step 1 & 2.

Step 5: For new data points, find the predictions of each decision tree, and assign the new data points to the category that wins the majority votes.

3.1.2 Tomato leaf disease detection based on ANN techniques

Inspired by the properties of biological neural networks, Artificial Neural Networks are statistical learning algorithms and are used for a variety of tasks, from relatively simple classification tasks to computer vision and speech recognition. ANNs are implemented as a system of interconnected processing elements, called nodes as shown in Fig. 3.3, which are functionally analogous to biological neurons. The connections between different nodes have numerical values, called weights, and by altering these values in a systematic way, the network is eventually able to approximate the desired function.

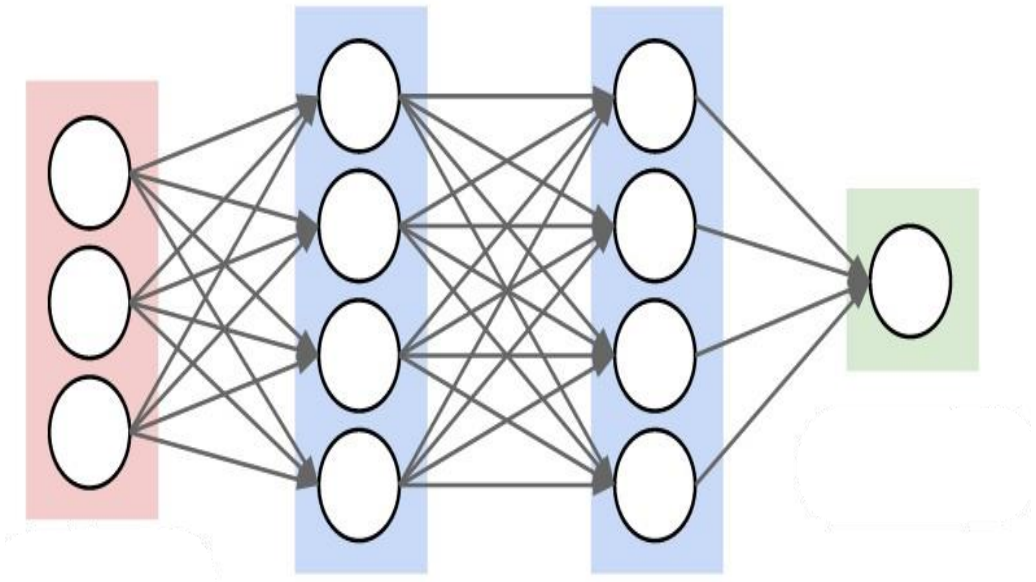


Fig. 3.3 Artificial Neural Network

The hidden layers can be thought of as individual feature detectors, recognizing more and more complex patterns in the data as it is propagated throughout the network. For example, if the network is given a task to recognize a face, the first hidden layer might act as a line detector, the second hidden takes these lines as input and puts them together to form a nose, the third hidden layer takes the nose and matches it with an eye and so on,

until finally the whole face is constructed. This hierarchy enables the network to eventually recognize very complex objects.

3.1.3 Tomato leaf disease detection based on CNN techniques

A convolutional neural network is a subset of machine learning. It is one of the various types of artificial neural networks which are used for different applications and data types. A CNN is a kind of network architecture for deep learning algorithms and is specifically used for image recognition and tasks that involve the processing of pixel data.

A deep learning CNN consists of three layers: a convolutional layer, a pooling layer and a fully connected (FC) layer. The convolutional layer is the first layer while the FC layer is the last.

From the convolutional layer to the FC layer, the complexity of the CNN increases. It is this increasing complexity that allows the CNN to successively identify larger portions and more complex features of an image until it finally identifies the object in its entirety.

Convolutional layer: It is the main building block of a CNN. It contains a set of filters or kernels, parameters of which are to be learned throughout the training as shown in Fig. 3.4. The size of filter is usually smaller than the actual image. Each filter convolves with the image and creates an activation map.

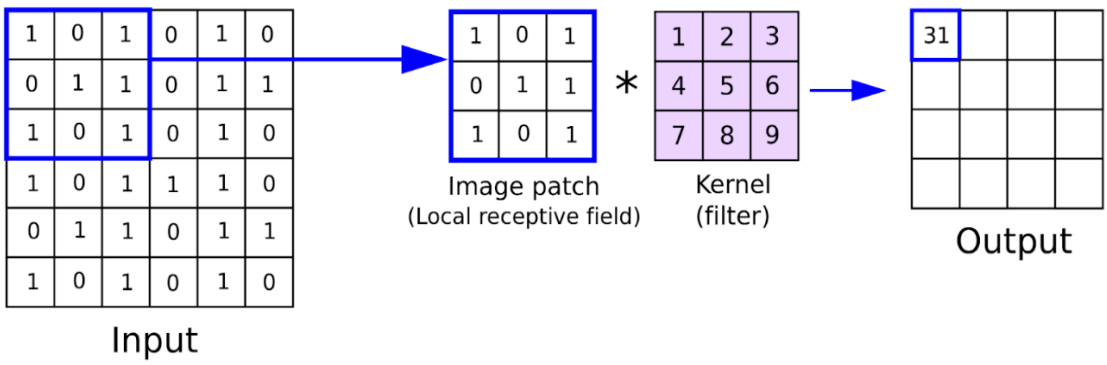


Fig. 3.4 Convolutional layer

Pooling layer: Its function is to progressively reduce the spatial size of the representation to reduce the amount of parameters and computation in the network. Pooling layers operates on each feature map independently as shown in Fig. 3.5.

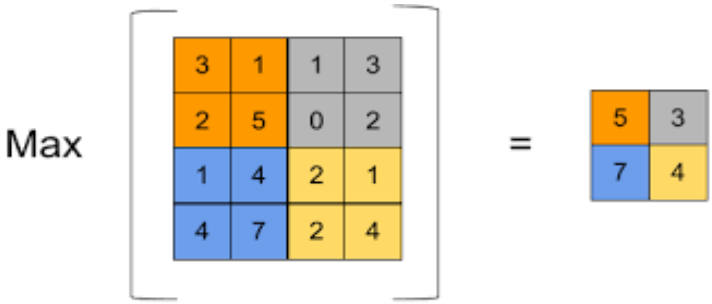


Fig. 3.5 Pooling layer

Flattening layer: It is used to convert all the resultant 2-Dimensional arrays from pooled feature maps into a single long continuous linear vector. The flattened matrix is fed as input to the fully connected layer to classify the image as shown in Fig. 3.6.

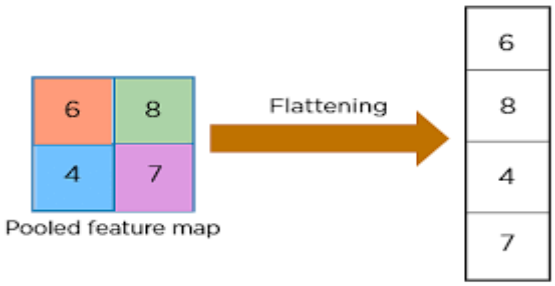


Fig. 3.6 Flattening layer

Dense layer: It is simple layer of neurons in which each neuron receives input from all the neurons of previous layer, thus called as dense. Dense layer is used to classify image based on output from convolutional layers as shown in Fig 3.7.

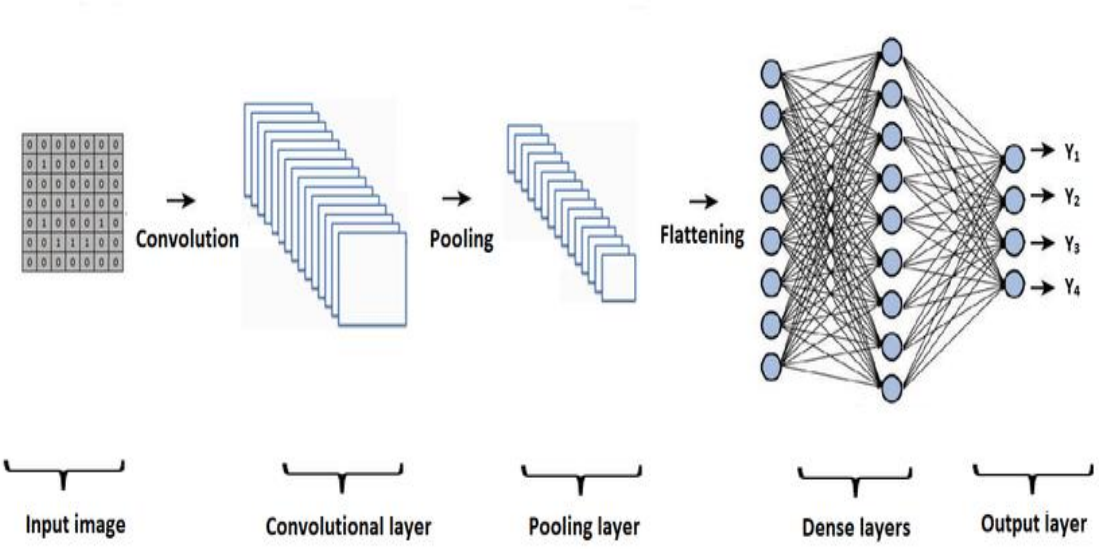


Fig. 3.7 Architecture of CNN

3.2 PROPOSED METHODOLOGY

Using convolutional neural networks (CNN), it is possible to create an algorithm for classifying images based on their raw data inputs. In CNN, features of input images are extracted by the process called feature extraction and other neural network helps to classify the image features. CNN applies various kernels over each picture to extract the image's characteristics, and it modifies the kernel by network propagation. After that, feature maps are created by convolving a kernel over the entire picture. The major feature extraction process occurs in the initial section of the neural network, which uses Convolutional layers and Pooling layers. The neural network's final stages include dense layers, which operate as the classifier layer and execute a variety of non-linear modifications on the acquired information. We tested many popular deep learning architectures for tomato leaf disease identification, including VGG 16, Custom CNN, EfficientNet B3, and Inception V3, and found that a variant of the Inception V3 architecture produced the best results.

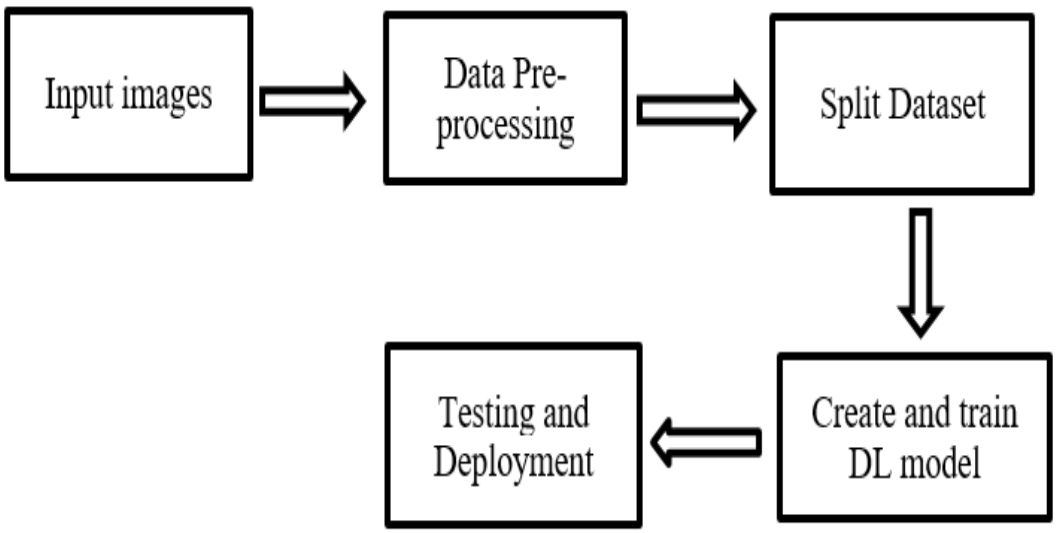


Fig. 3.8 Block diagram of proposed methodology

3.2.1 DATASET

Images of Tomato disease have been taken from Plant Village dataset. The dataset includes over 50,000 images of 14 crops, such as tomatoes, potatoes, grapes, apples, corn, blueberry, raspberry, soybeans, squash and strawberry.

We selected tomato as our target crop. The images of various classes of tomato are as follows:

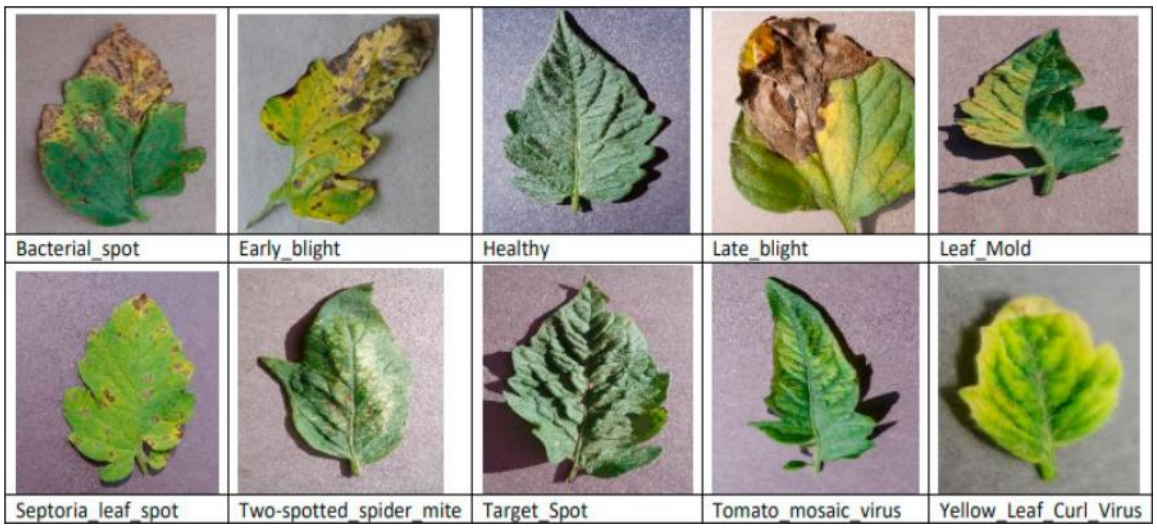


Fig. 3.9 Class wise sample image of the dataset

There are mainly nine types of diseases in tomato as shown in Fig. 3.9: 1) Target Spot, 2) Mosaic virus, 3) Bacterial spot, 4) Late blight, 5) Leaf Mold, 6) Yellow Leaf Curl Virus, 7) Spider mites: Two-spotted spider mite, 8) Early blight and 9) Septoria leaf spot. In proposed work, there are 10000 images in training dataset, 7000 images in validation dataset and 500 images in testing dataset. Out of 10000 training images, 1000 images belong to healthy category and 1000 images belong to each tomato disease category described above. In validation set each class has 700 images and test set has 50 images in each class. For testing, we randomly picked 50 images from each class from training set and removed them from those folders. From remaining training dataset, we built our project training dataset by putting same number of images (1000) in each class. When the images in any class were less than 1000, we used data augmentation technique to generate some new images. Augmentation was done using Augmentor package of python and it helps to build similar new images by rotating, flipping, cropping and resizing the existing images. When images in any class in training dataset were more than 1000, we picked first 1000 images. We followed same process for validation dataset and made all classes have 700 images each. This process is necessary to prevent bias for any particular class during training of CNN. Size of all the images is 256 × 256 and format is jpg.

3.2.2 DATA PREPROCESSING

Normalization:

It is referred to as data re-scaling, it is the process of projecting image data pixels (intensity) to a predefined range (usually (0, 1) or (-1, 1)). This is commonly used on

different data formats, and you want to normalize all of them to apply the same algorithms over them.

Its benefits include:

- Fairness across all images - For example, scaling all images to an equal range of [0,1] or [-1,1] allows all images to contribute equally to the total loss rather than when other images have high and low pixels ranges give strong and weak loss, respectively.
- Provides a standard learning rate - Since high pixel images require a low learning rate and low pixel images high learning rate, re-scaling helps provide a standard learning rate for all images.

Data augmentation:

It is the process of making minor alterations to existing data to increase its diversity without collecting new data as shown in Fig 3.10.

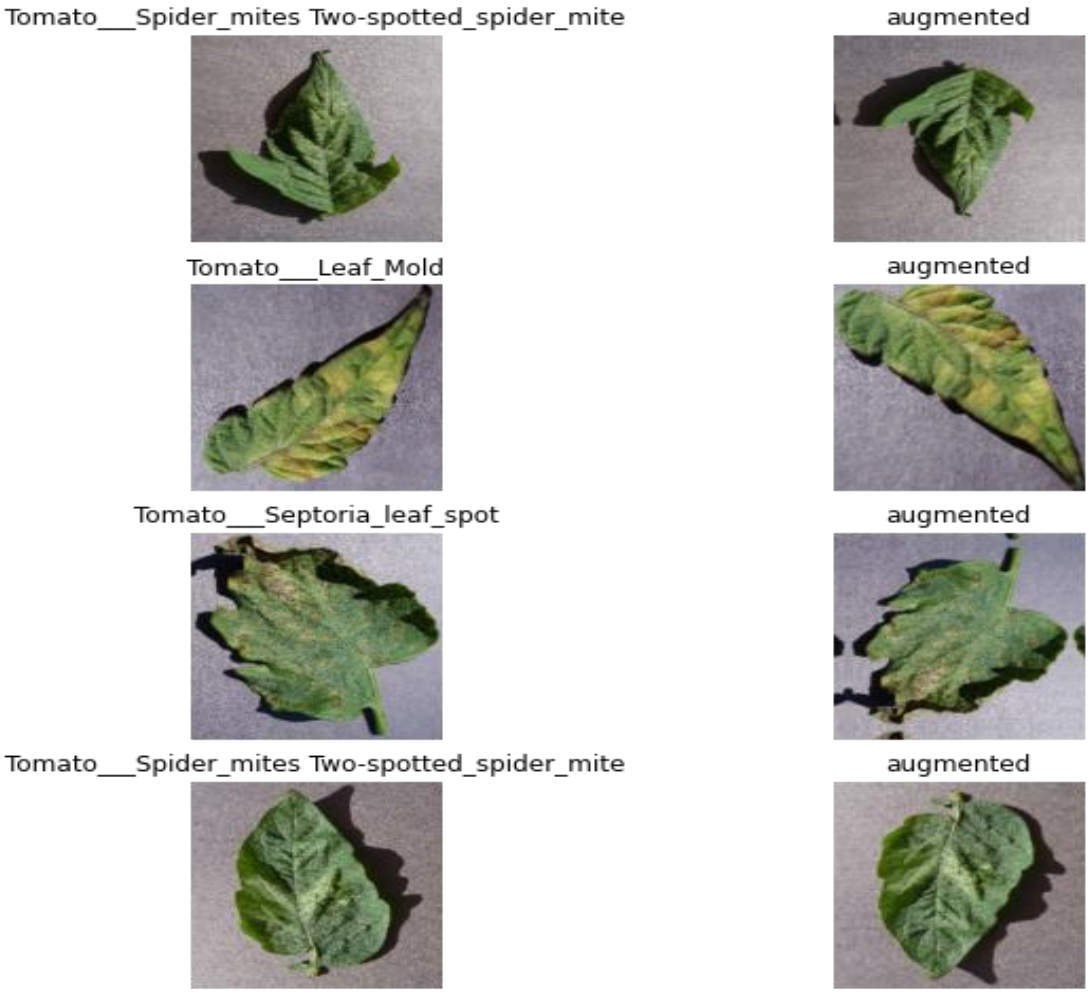


Fig. 3.10 Original vs Augmented images

- It is a technique used for enlarging a dataset. Standard data augmentation techniques include horizontal & vertical flipping, rotation, cropping, shearing, etc.
- Performing data augmentation helps in preventing a neural network from learning irrelevant features. This results in better model performance.

3.2.3 INCEPTION V3 ARCHITECTURE

With its latest version Inception V3, CNN provides the basis for Inception's sophisticated artificial intelligence. Convolutional, pooling, and fully-linked layers are only some of the components that make up this design. The convolutional layers are responsible for contracting features from the input image. The pooling layers reduce the amount of data by grouping spatially close data points into a single unit. To create predictions, Layers that are fully connected include both convolutional and pooling components. To avoid overfitting the Inception V3 architecture contains batch normalization layers and dropout layers.

The modified InceptionV3 model illustrated in Fig. 3.11 has the same input layers as a regular one, but there is a change in a fully connected layer. It consists of three dense layers, with 2048 and 4096 neurons in the starting two layers both with activation function as relu, and 10 neurons in the last layer with activation function as softmax which are used for classifying 10 classes of tomato leaves. These additional layers boost the model's recognition accuracy and trainable parameters. The difference between Inception V3 and its modified version is presented in Table 2.

Table 2. Difference between Inception V3 and Modified Inception V3

	Input Shape	**Output Shape**	**Trainable Parameters**	**Non Trainable Parameters**
Inception V3	299×299×3	8 × 8 × 2048 (1000 channels)	23,817,352	34,432
Modified Inception V3	224×224×3	5 × 5 × 2048 (10 channels)	34,398,378	34,432

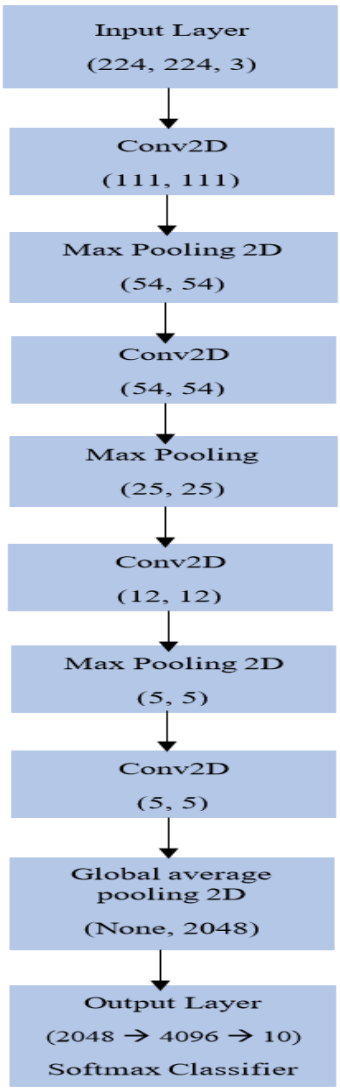


Fig. 3.11 Architecture of Inception V3

CHAPTER 4
RESULTS AND ANALYSIS

In the suggested work, we also compared the efficacy of several pre-trained and custom models. Following is a brief explanation of several pre-trained and custom models:

4.1 VGG 16 MODEL

ImageNet Large-Scale Visual Recognition Challenge (ILSVRC) is an annual event to showcase and challenge computer vision models. In the 2014 ImageNet challenge, Karen Simonyan & Andrew Zisserman from Visual Geometry Group, Department of Engineering Science, University of Oxford showcased their model in the paper titled "VERY DEEP CONVOLUTIONAL NETWORKS FOR LARGE-SCALE IMAGE RECOGNITION," which won the 1st and 2nd place in object detection and classification. A convolutional neural network is also known as a ConvNet, which is a kind of artificial neural network.

A convolutional neural network has an input layer, an output layer, and various hidden layers. VGG16 is a type of CNN (Convolutional Neural Network) that is considered to be one of the best computer vision models to date. The creators of this model evaluated the networks and increased the depth using an architecture with very small (3 × 3) convolution filters, which showed a significant improvement on the prior-art configurations. They pushed the depth to 16 to 19 weight layers making it approx. 13.8 million trainable parameters. VGG16 is object detection and classification algorithm which is able to classify 1000 images of 1000 different categories with 92.7% accuracy. It is one of the popular algorithms for image classification and is easy to use with transfer learning.

The 16 in VGG16 refers to 16 layers that have weights. In VGG16 there are thirteen convolutional layers, five Max Pooling layers, and three Dense layers which sum up to 21 layers but it has only sixteen weight layers i.e., learnable parameters layer. VGG16 takes input tensor size as 224, 244 with 3 RGB channel. Most unique thing about VGG16 is that instead of having a large number of hyper-parameters they focused on having convolution layers of 3x3 filter with stride 1 and always used the same padding and maxpool layer of 2x2 filter of stride 2. The convolution and max pool layers are consistently arranged throughout the whole architecture. Conv-1 Layer has 64 number of filters, Conv-2 has 128 filters, Conv-3 has 256 filters, Conv 4 and Conv 5 has 512 filters. Three Fully-Connected

(FC) layers follow a stack of convolutional layers: the first two have 4096 channels each, the third performs 1000-way ILSVRC classification and thus contains 1000 channels (one for each class). The final layer is the soft-max layer as shown in Fig. 4.1.

We slightly tuned the model in the final layer includes 10 output nodes and two preliminary layers with 4096 hidden nodes each. The 10 tomato class subsets were represented by an output layer with 10 dimensions, and we tried to employ the VGG 16 weights once they had been pre-trained. VGG16 transfer learning fared poorly on our mock tomato leaf dataset from the Plant Village dataset files. After training our code for 20 iterations, we got an accuracy of 91.2% on tests and 90.2% on validations.

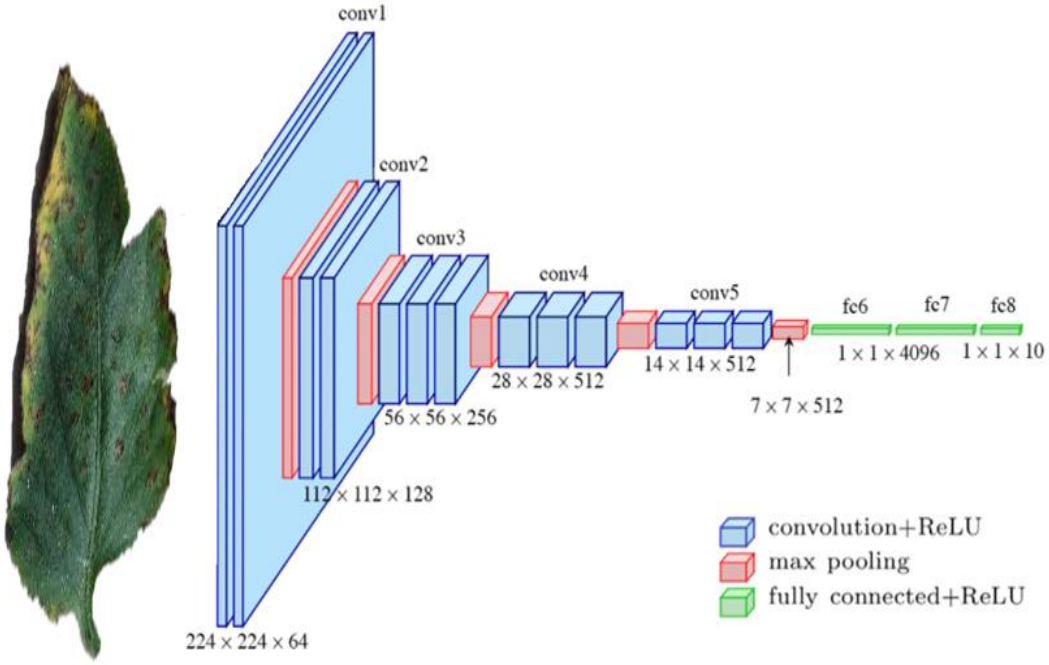


Fig 4.1 Architecture of VGG 16

4.2 CUSTOM CNN MODEL

The Custom CNN model has an activation function of ReLu, 64 filters of size 3 x 3, a max pooling layer with a pool size of 2 x 2, and batch normalization following each of the three convolution layers. The input shape is 224 x 224 x 3. There's also a flattened layer for merging the several discrete 2-D arrays produced by the pooled feature maps into a single 3-D one. The image classification fully connected layer takes in the distorted matrix as input. A total of three dense layers (with 1024, 2048, and 10 neurons) make up the fully connected layer. The first two dense levels use the ReLu activation function, while the third

dense layer uses softmax as shown in Fig. 4.2. After 20 rounds of training, our model was able to attain a 90.4% accuracy during testing and a 92.4% accuracy during validation.

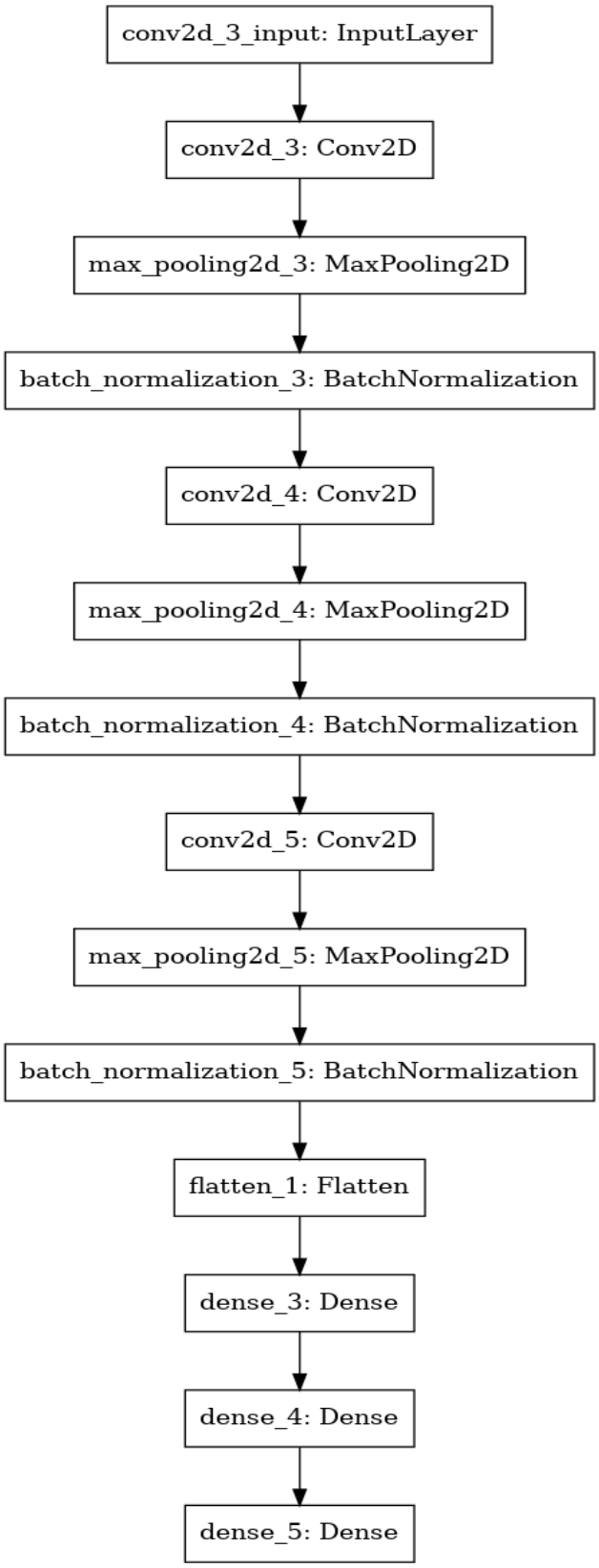


Fig 4.2 Custom CNN Architecture

4.3 EFFICIENT NET B3 MODEL

EfficientNet B3 is a convolutional neural network architecture that was proposed in 2019 by Mingxing Tan and Quoc V. Le. It is part of a family of models called EfficientNets, which are designed to be highly efficient in terms of computational resources while achieving state-of-the-art performance on image classification tasks.

EfficientNet B3 has 28 convolutional layers and a total of 12 million parameters. The architecture is based on a compound scaling method that scales the width, depth, and resolution of the network in a systematic way. This means that the network is designed to be both deeper and wider than previous networks, while also having a higher input resolution. The architecture of EfficientNet B3 is made up of three main components: the stem, the blocks, and the head. The stem is a series of convolutional and pooling layers that process the input image and extract low-level features. The blocks are a series of repeated building blocks, each consisting of multiple convolutional layers and a squeeze-and-excitation block that learns to weight the importance of each channel. The head is a final set of fully connected layers that map the features learned by the earlier layers to the output classes.

Overall, EfficientNet B3 represents a significant advance in the design of convolutional neural network architectures, demonstrating that it is possible to achieve both high performance and high efficiency by carefully balancing the trade-off between depth, width, and resolution as shown in Fig. 4.3.

The EfficientNet-B3 model is a five-layer variant of the EfficientNet family of models. The first layer is the Input Layer, which receives a 224×224 input picture and runs it through a 3×3 convolutional layer with a stride of 1. The second layer is the EfficientNet Block, which is made up of a succession of inverted bottleneck blocks with a 3×3-size convolutional kernel. This layer is in charge of lowering the number of parameters while increasing the receptive field. The third layer is the Squeeze-and-Excitation Layer, which is in charge of determining channel-wise attention weights, which are subsequently used to scale the previous layer's output. The Output Layer is the fourth layer, comprising a global average pooling layer, a fully connected layer for classification, and a 3x3 convolutional layer with a stride of 1. The fifth layer is the Softmax Layer, which is in charge of generating class probabilities for the input image. We slightly changed the fully connected

layer by adding a dense layer of 10 neurons with softmax activation functions. We trained the model for 20 epochs and got 98.88% testing accuracy and 99.77% validation accuracy.

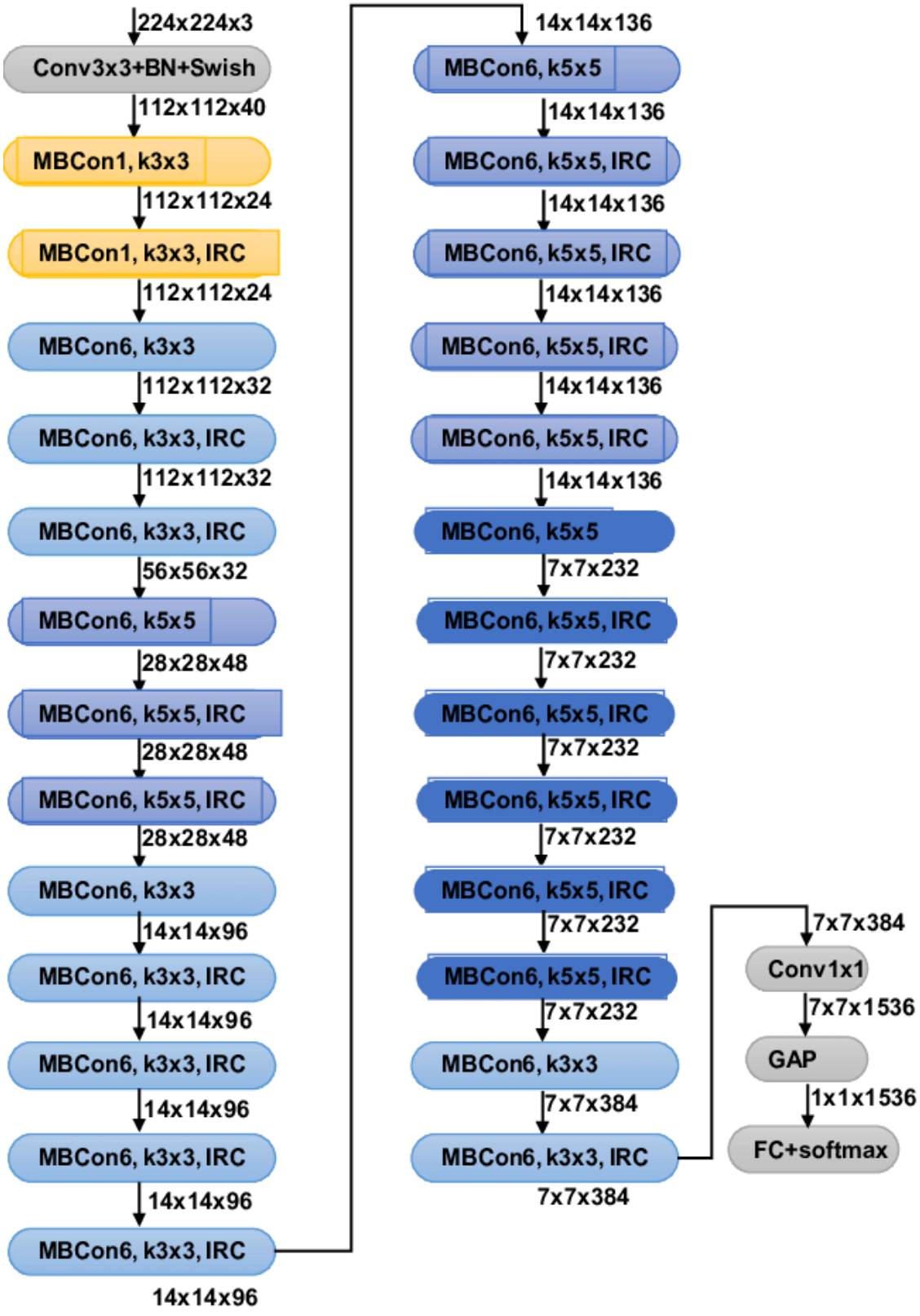


Fig 4.3 EfficientNet B3 Architecture

4.4 INCEPTION V3 MODEL

Inception V3 is a convolutional neural network (CNN) architecture that was developed by Google researchers in 2015 as an improvement over its predecessor, Inception V1 and V2. The primary goal of the Inception V3 architecture was to achieve higher accuracy on the ImageNet classification task while reducing the number of parameters and computation required compared to previous Inception models.

The Inception V3 architecture consists of multiple layers, including convolutional, pooling, and fully connected layers. The most notable feature of Inception V3 is the use of "inception modules" that allow the network to simultaneously learn features at multiple spatial scales. An inception module is composed of several branches of convolutional layers, each performing a different operation, such as 1x1 convolution, 3x3 convolution, or 5x5 convolution. These branches are then concatenated together along the channel dimension, allowing the network to learn a wide range of features at different scales and resolutions. The resulting feature maps are then fed into the next layer of the network.

Another key aspect of Inception V3 is the use of batch normalization, which helps to speed up training and improve generalization performance. In addition, Inception V3 uses "factorized 7x7 convolutions," which decompose a large 7x7 convolution into a series of smaller convolutions, further reducing the number of parameters and computation required. Overall, the Inception V3 architecture has shown impressive performance on a variety of computer vision tasks, including object detection, image segmentation, and visual question answering. It has also been used as a backbone for transfer learning in various applications due to its ability to extract high-level features from images with great accuracy as shown in Fig. 4.4.

A CNN model called Inception V3 is used to categorize images. Its architecture is comprised of 48 layers in total, some of which are pooling layers, fully connected layers, convolutional layers, and inception modules. There is a total of five distinct levels in the current V3 Inception model. The Input Layer is the 1st layer, it is in charge of taking an image as input. Convolutional Layers are used in the second layer to create feature mappings from the input image. Filters are used to find patterns in images. To reduce the overall footprint of the feature maps, the third layer, titled Pooling Layers, is used to lower the number of parameters by steadily decreasing the spatial dimensions of the

representation or image. Fourthly, the feature maps are classified by Fully Connected Layers. The fifth layer is Inception Modules, a convolutional layer that uses a combination of 1×1, 3×3, and 5×5 convolution filters to reduce the number of parameters and increase model performance. We changed the fully connected layer slightly by adding extra dense layers and raising its count to three. The first two dense layers are made up of 2048 and 4096 nodes with an activation function as relu, while the last dense layer is made up of 10 nodes with an activation function as softmax. We trained the model for 20 epochs and got 99.60% testing accuracy and 99.25% validation accuracy as illustrated in Fig.4.5. The validation loss curve of the proposed model is shown in Fig.4.6. After training the model over 20 epochs we got the following results are presented in Table 3 and 4.

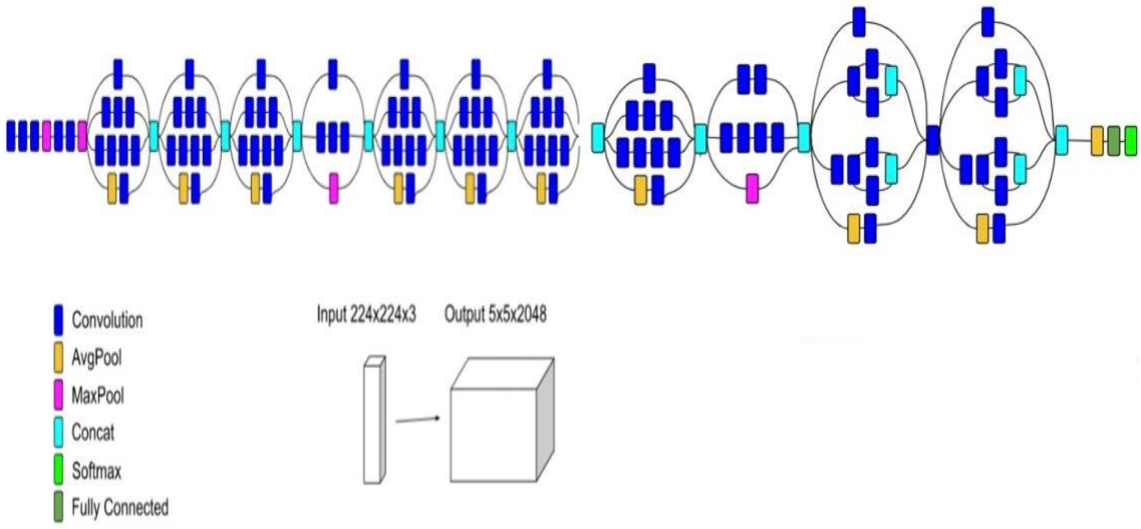


Fig 4.4 Inception V3 Architecture

Table 3. Performance of model in each class

Class	F1 score	Recall	Precision
0	0.99	0.98	1.00
1	1.00	1.00	1.00
2	1.00	1.00	1.00
3	1.00	1.00	1.00
4	1.00	1.00	1.00
5	0.99	0.98	1.00
6	0.98	1.00	0.96
7	1.00	1.00	1.00
8	1.00	1.00	1.00
9	1.00	1.00	1.00

Accuracy, precision, recall, and F1 score are metrics used to evaluate the performance of classification models.

Accuracy: It measures the proportion of correct predictions (True Positive and True Negative) made by the model. It is calculated by dividing the number of correct predictions by the total number of predictions made by the model.

Accuracy = (TP + TN) / (TP + TN + FP + FN)

Precision: It measures the proportion of true positive predictions among all the positive predictions made by the model. It is calculated by dividing the number of true positive predictions by the sum of true positive and false positive predictions.

Precision = TP / (TP + FP)

Recall: It measures the proportion of true positive predictions among all the actual positive instances in the data. It is calculated by dividing the number of true positive predictions by the sum of true positive and false negative predictions.

Recall = TP / (TP + FN)

F1 score: It is the harmonic mean of precision and recall. It is a combined metric that takes into account both precision and recall. It is calculated as follows:

F1 Score = 2 * (Precision * Recall) / (Precision + Recall)

Where, TP = True Positive,

TN = True Negative,

FP = False Positive,

FN = False Negative

Out of all the tomato leaves that had the disease, Table 3 displays the precision, which indicates the number of tomato leaves that were correctly recognized as having it. For all the tomato leaves that had the disease, recall indicates how many tomato leaves the model accurately identified as having it. The precision and recall values are combined into a single score by the F1 score. All the above parameters are generated after running the model on test data of 500 samples.

Table 4. Accuracy and Loss of Inception V3 after each epoch

Epoch	Train loss	Train Accuracy	Validation loss	Validation Accuracy
1	1.3689	0.5129	5.0945	0.2851
2	0.5769	0.8178	1.8762	0.5096
3	0.3860	0.8820	0.8850	0.7871
4	0.2912	0.9083	0.7179	0.7770
5	0.2652	0.9153	0.5124	0.8506
6	0.2428	0.9253	0.3028	0.8954
7	0.1825	0.9424	0.3953	0.8880
8	0.1724	0.9479	0.2858	0.9106
9	0.1531	0.9506	0.8844	0.7897
10	0.1552	0.9519	0.1631	0.9497
11	0.1506	0.9521	0.3313	0.9039
12	0.1357	0.9574	0.5031	0.8633
13	0.0518	0.9822	0.0499	0.9844
14	0.0428	0.9857	0.0825	0.9759
15	0.0453	0.9837	0.0595	0.9820
16	0.0283	0.9909	0.0333	0.9913
17	0.0203	0.9936	0.0308	0.9914
18	0.0176	0.9936	0.0389	0.9893
19	0.0212	0.9922	0.0300	0.9910
20	0.0154	0.9952	0.0291	0.9926

In machine learning, an epoch refers to one full iteration of the training data through a neural network. During each epoch, the neural network makes predictions on the training data, computes the error or loss between its predictions and the actual values, and updates its internal parameters to improve its predictions.

As the number of epochs increases, the model has more opportunities to learn and adjust its internal parameters to better fit the training data. This can lead to an increase in accuracy, as the model becomes more familiar with the patterns and relationships within the training data as shown in Table 4.

However, it's important to note that increasing the number of epochs can also lead to overfitting, where the model becomes too specialized to the training data and performs poorly on new, unseen data. Therefore, it's important to monitor the model's performance on both the training and validation data to determine the optimal number of epochs for a given task.

Out of all the leaves that had the disease, Table 3 displays the precision, which indicates the number of leaves that were correctly recognized as having it. For all the leaves that had the disease, recall indicates how many leaves the model accurately identified as having it. The precision and recall values are combined into a single score by the F1 score. All the above parameters are generated after running the model on test data of 500 samples.

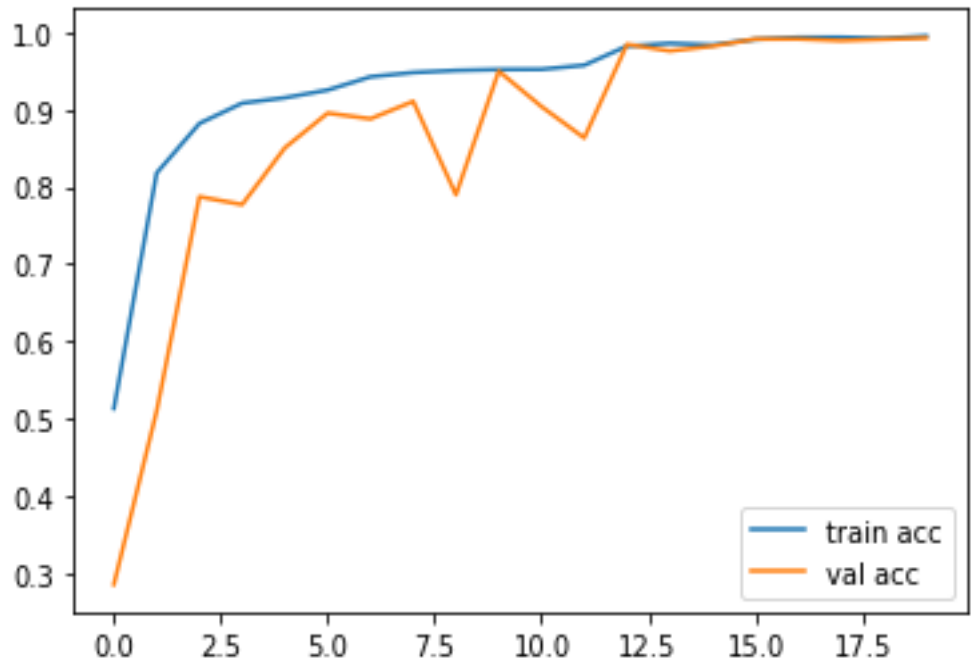


Fig 4.5 Plots of training and validation accuracy

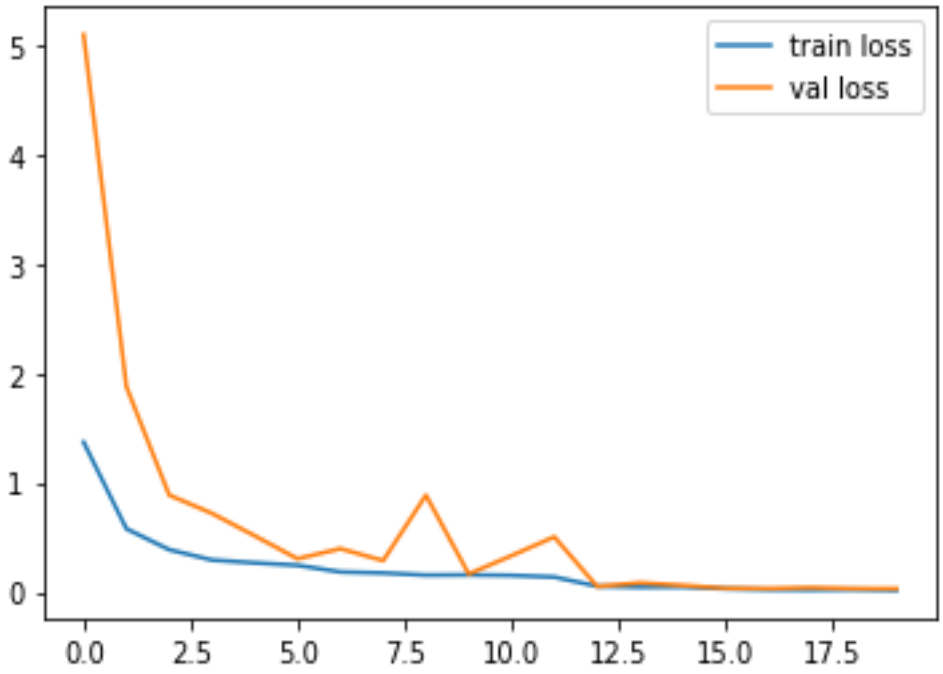


Fig 4.6 Plots of training and validation loss

Accuracy and loss curves are graphical representations of a machine learning model's performance during training and validation. These curves are important because they allow us to analyse and optimize a model's performance.

The accuracy curve shows the model's accuracy over time as it learns from the training data. Typically, the accuracy increases as the model learns to make better predictions. However, if the model is overfitting, the accuracy may plateau or even decrease over time.

The loss curve shows the model's loss over time as it learns from the training data. Loss is a measure of how well the model's predictions match the true labels. Ideally, the loss should decrease over time as the model learns to make better predictions. However, if the model is overfitting, the loss may decrease initially but then start to increase as the model memorizes the training data.

By analysing the accuracy and loss curves, we can determine if a model is overfitting or under fitting. If the accuracy is high on the training data but low on the validation data, it's a sign of overfitting. If the accuracy is low on both the training and validation data, it's a sign of under fitting. They provide valuable insights into whether a model is overfitting or underfitting, and they allow us to optimize the model's performance.

Both Fig 4.5 and 4.6 are generated after running the model over 20 epochs on the training and validation dataset. They represent the loss and accuracies of the model after each epoch.

Table 5. Performance of several models

S.No.	Model	Trainable parameters	Non-Trainable parameters	Accuracy
1	VGG 16	134,301,514	0	91.20%
2	Efficient Net B3	10,711,602	87,303	98.88%
3	Custom model	46,499,082	384	90.40%
4	Proposed model (Inception V3)	34,398,378	34,432	99.60%

Table 5 shows how various models performed after being put through the same number of epochs and dataset. Inception v3 did better than the other models.

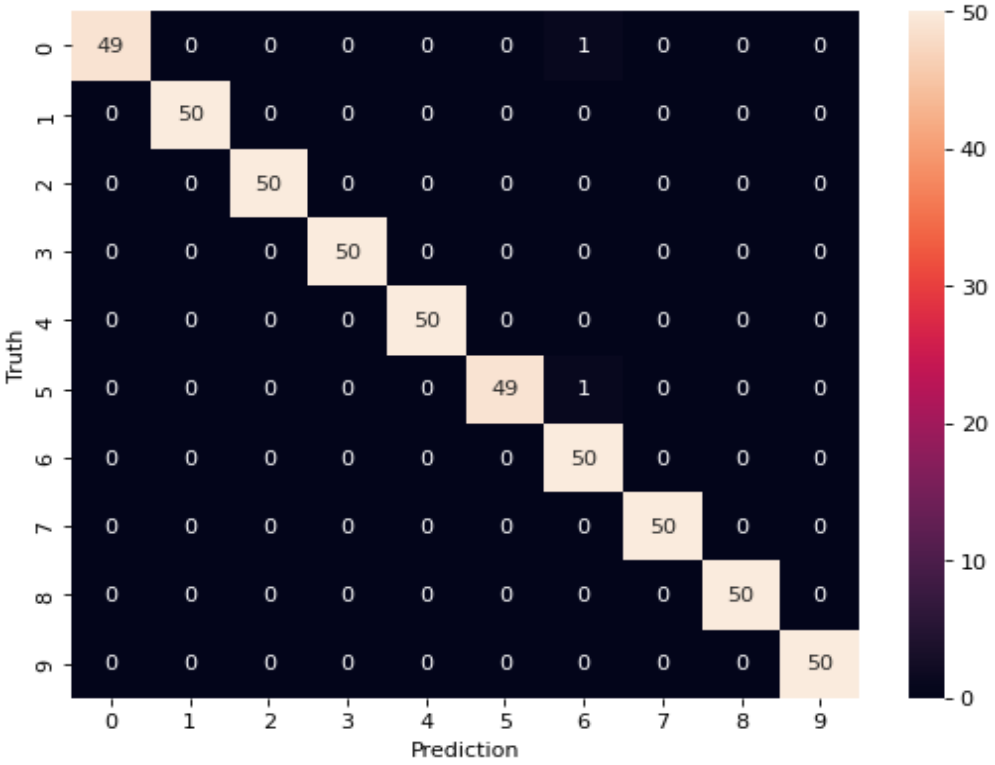


Fig 4.7 Confusion matrix of proposed model

A confusion matrix is a table that is used to evaluate the performance of a classification model by comparing the predicted labels of the model to the true labels of the data. It is a matrix with four cells that represent the number of true positive (TP), false positive (FP), true negative (TN), and false negative (FN) predictions.

The relevance of a confusion matrix is that it provides a comprehensive evaluation of the performance of a classification model, allowing you to see how well it is doing in terms of correctly predicting positive and negative instances, and identifying false positives and false negatives. This information can be used to adjust and improve the model to achieve better accuracy, precision, recall, and F1 score. The confusion matrix is also used in other evaluation metrics such as ROC curve, AUC, and precision-recall curve.

Fig. 4.7 shows the number of accurate and inaccurate predictions, with count values for 10 classes and 50 test images for each class.

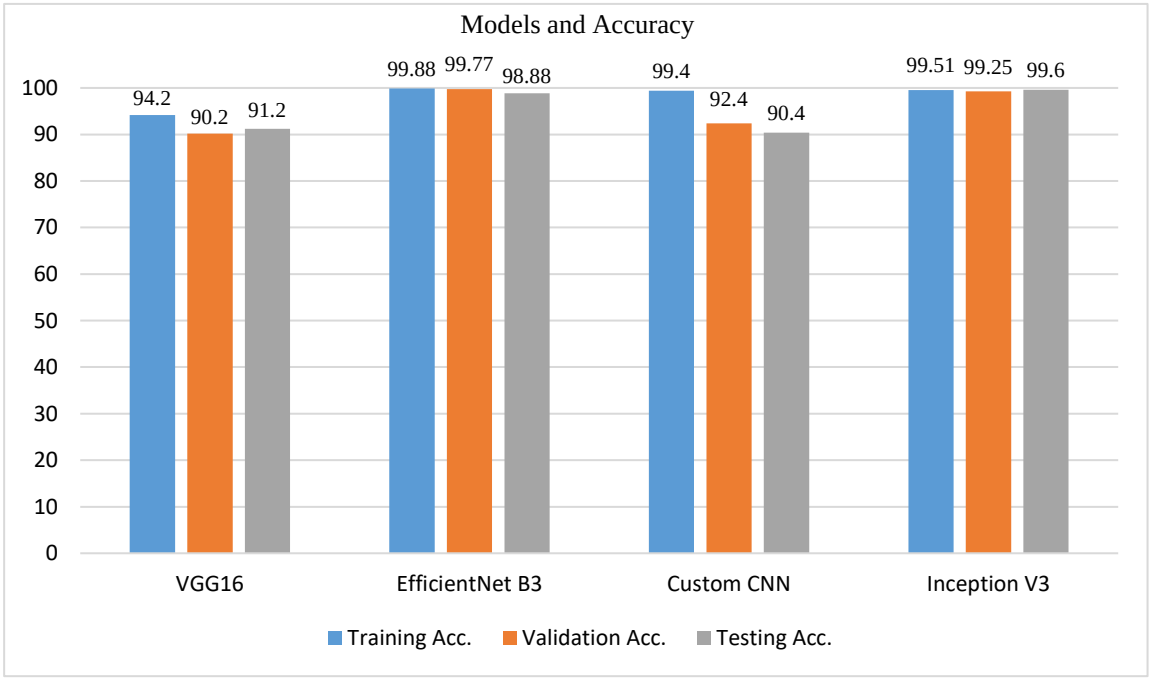


Fig 4.8 Accuracies of various models

Training accuracies, validation accuracies and testing accuracies of various CNN models like VGG 16, Custom CNN, EfficientNet B3 and Inception V3 are shown in Fig. 4.8 and we can clearly observe that Inception V3 model performed better when compared to others.

CHAPTER 5
CONCLUSION

5.1 CONCLUSION

Tomato leaf disease control is difficult and consistently impacts the overall cost of output throughout the growing season. Bacterial spots, early blight, leaf mold, Septoria leaf spots, mosaic virus, and many more diseases have devastating effects on plant development. When dealing with tomatoes, early detection of leaf diseases is essential for preventing crop failure. Manually identifying a pest or a disease is a costly and time-taking process. Because of this, providing automated AI image-based solutions to farmers is vital. In this study, we present a CNN-based approach for early disease detection. In the work proposed, a CNN model is trained using VGG16, Custom CNN, Efficient Net B3, and Inception V3, with Inception V3 achieving the highest accuracy (99.60%) of the four.

5.2 FUTURE SCOPE

The proposed approach for identifying tomato disease is a ground-breaking notion. In the future, we will expand the model to include certain abiotic diseases due to the deficiency of nutrient values in the crop leaf. Our long-term objective is to increase unique data collection and accumulate a vast amount of data on several diseases of plants. To improve accuracy, we will apply subsequent technology in the future.

BIBLIOGRAPHY

[1] Agarwal, M., Singh, A., Arjaria, S., Sinha, A., & Gupta, S. (2020). ToLeD: Tomato leaf disease detection using convolution neural network. *Procedia Computer Science*, *167*, 293-301.

[2] Wang, Q., Qi, F., Sun, M., Qu, J., & Xue, J. (2019). Identification of tomato disease types and detection of infected areas based on deep convolutional neural networks and object detection techniques. *Computational intelligence and neuroscience*, *2019*.

[3] Liu, J., & Wang, X. (2020). Tomato diseases and pests detection based on improved Yolo V3 convolutional neural network. *Frontiers in plant science*, *11*, 898.

[4] Tm, P., Pranathi, A., SaiAshritha, K., Chittaragi, N. B., & Koolagudi, S. G. (2018, August). Tomato leaf disease detection using convolutional neural networks. In *2018 eleventh international conference on contemporary computing (IC3)* (pp. 1-5). IEEE.

[5] Liu, J., & Wang, X. (2020). Early recognition of tomato gray leaf spot disease based on MobileNetv2-YOLOv3 model. *Plant Methods*, *16*(1), 1-16.

[6] Trivedi, N. K., Gautam, V., Anand, A., Aljahdali, H. M., Villar, S. G., Anand, D., ... & Kadry, S. (2021). Early detection and classification of tomato leaf disease using high-performance deep neural network. *Sensors*, *21*(23), 7987.

[7] Singh, V., & Misra, A. K. (2017). Detection of plant leaf diseases using image segmentation and soft computing techniques. *Information processing in Agriculture*, *4*(1), 41-49.

[8] Sun, X., Li, G., Qu, P., Xie, X., Pan, X., & Zhang, W. (2022). Research on plant disease identification based on CNN. *Cognitive Robotics*, *2*, 155-163.

[9] Khamparia, A., Saini, G., Gupta, D., Khanna, A., Tiwari, S., & de Albuquerque, V. H. C. (2020). Seasonal crops disease prediction and classification using deep convolutional encoder network. *Circuits, Systems, and Signal Processing*, *39*(2), 818-836.

[10] Fenu, G., & Malloci, F. M. (2021). Forecasting plant and crop disease: an explorative study on current algorithms. *Big Data and Cognitive Computing*, *5*(1), 2.

[11] Nagamani, H. S., & Sarojadevi, H. (2022). Tomato Leaf Disease Detection using Deep Learning Techniques. International Journal of Advanced Computer Science and Applications, 13(1).

[12] Altalak, M., Uddin, M. A., Alajmi, A., & Rizg, A. (2022). A Hybrid Approach for the Detection and Classification of Tomato Leaf Diseases. Applied Sciences, 12(16), 8182.

[13] Karthik, R., Hariharan, M., Anand, S., Mathikshara, P., Johnson, A., & Menaka, R. (2020). Attention embedded residual CNN for disease detection in tomato leaves. *Applied Soft Computing*, *86*, 105933.

[14] Sardogan, M., Tuncer, A., & Ozen, Y. (2018, September). Plant leaf disease detection and classification based on CNN with LVQ algorithm. In *2018 3rd international conference on computer science and engineering (UBMK)* (pp. 382-385). IEEE.

[15] Ahmad, I., Hamid, M., Yousaf, S., Shah, S. T., & Ahmad, M. O. (2020). Optimizing pretrained convolutional neural networks for tomato leaf disease detection. *Complexity*, *2020*, 1-6.

[16] Sareen, N., Chug, A., & Singh, A. P. (2022). An image based prediction system for early blight disease in tomato plants using deep learning algorithm. *Journal of Information and Optimization Sciences*, *43*(4), 761-779. [17] Shrestha, G., Das, M., & Dey, N. (2020, October). Plant disease detection using CNN. In *2020 IEEE Applied Signal Processing Conference (ASPCON)* (pp. 109-113). IEEE.

[18] Chen, H. C., Widodo, A. M., Wisnujati, A., Rahaman, M., Lin, J. C. W., Chen, L., & Weng, C. E. (2022). AlexNet convolutional neural network for disease detection and classification of tomato leaf. *Electronics*, *11*(6), 951.

[19] Zhao, S., Peng, Y., Liu, J., & Wu, S. (2021). Tomato leaf disease diagnosis based on improved convolution neural network by attention module. *Agriculture*, *11*(7), 651.

[20] Abbas, A., Jain, S., Gour, M., & Vankudothu, S. (2021). Tomato plant disease detection using transfer learning with C-GAN synthetic images. *Computers and Electronics in Agriculture*, *187*, 106279.

[21] Harakannanavar, S. S., Rudagi, J. M., Puranikmath, V. I., Siddiqua, A., & Pramodhini, R. (2022). Plant leaf disease detection using computer vision and machine learning algorithms. *Global Transitions Proceedings*, *3*(1), 305-310.

[22] Anandhakrishnan, T., & Jaisakthi, S. M. (2022). Deep Convolutional Neural Networks for image based tomato leaf disease detection. *Sustainable Chemistry and Pharmacy*, *30*, 100793.

[23] De Luna, R. G., Dadios, E. P., & Bandala, A. A. (2018, October). Automated image capturing system for deep learning-based tomato plant leaf disease detection and recognition. In *TENCON 2018-2018 IEEE Region 10 Conference* (pp. 1414-1419). IEEE.

[24] Sharma, P., Berwal, Y. P. S., & Ghai, W. (2020). Performance analysis of deep learning CNN models for disease detection in plants using image segmentation. *Information Processing in Agriculture*, *7*(4), 566-574.

[25] Cheng, H. H., Dai, Y. L., Lin, Y., Hsu, H. C., Lin, C. P., Huang, J. H., ... & Kuo, Y. F. (2022). Identifying tomato leaf diseases under real field conditions using convolutional neural networks and a chatbot. *Computers and Electronics in Agriculture*, *202*, 107365.

[26] Geetharamani, G., & Pandian, A. (2019). Identification of plant leaf diseases using a nine-layer deep convolutional neural network. *Computers & Electrical Engineering*, *76*, 323-338.

[27] Mkonyi, L., Rubanga, D., Richard, M., Zekeya, N., Sawahiko, S., Maiseli, B., & Machuve, D. (2020). Early identification of Tuta absoluta in tomato plants using deep learning. *Scientific African*, *10*, e00590.

[28] Salih, T. A. (2020). Deep learning convolution neural network to detect and classify tomato plant leaf diseases. *Open Access Library Journal*, *7*(05), 1.

[29] Fuentes, A. F., Yoon, S., Lee, J., & Park, D. S. (2018). High-performance deep neural network-based tomato plant diseases and pests diagnosis system with refinement filter bank. *Frontiers in plant science*, *9*, 1162.

[30] Mohanty, R., Wankhede, P., Singh, D., & Vakhare, P. (2022, May). Tomato Plant Leaves Disease Detection using Machine Learning. In *2022 International Conference on Applied Artificial Intelligence and Computing (ICAAIC)* (pp. 544-549). IEEE.

In [ ]:
```
# Importing the required Libraries
import numpy as np
import seaborn as sns
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.models import Sequential
from tensorflow.keras import layers
from tensorflow.keras.utils import plot_model
from tensorflow.keras.layers import Activation, Dense, Flatten, BatchNormalization, Conv2D, MaxPool2D
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.metrics import categorical_crossentropy
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.callbacks import ModelCheckpoint, EarlyStopping, ReduceLROnPlateau
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
from sklearn.model_selection import train_test_split
import itertools
import os
import glob
import warnings
warnings.simplefilter(action="ignore", category=FutureWarning)
import matplotlib.pyplot as plt
%matplotlib inline
```

In [ ]:
```
train_path = "/kaggle/input/tomatoleafdisease/Tomato Leaf Diseases - Copy/Training Set"
valid_path = "/kaggle/input/tomatoleafdisease/Tomato Leaf Diseases - Copy/Validation Set"
test_path = "/kaggle/input/tomatoleafdisease/Tomato Leaf Diseases - Copy/Testing set"
```

In [ ]:
```
train_datagen = ImageDataGenerator(rescale = 1./255,
                                   shear_range = 0.2,
                                   zoom_range = 0.2,
                                   horizontal_flip = True,
                                   vertical_flip=True)

valid_datagen = ImageDataGenerator(rescale = 1./255)

test_datagen = ImageDataGenerator(rescale = 1./255)
```

In [ ]:
```
# Augmenting training data for more training examples
training_set = train_datagen.flow_from_directory("/kaggle/input/tomatoleafdisease/Tomato Leaf Diseases - Co
                                                 target_size = (224, 224),
                                                 batch_size = 32,
                                                 class_mode = 'categorical')
```

```
Found 9500 images belonging to 10 classes.
```

In [ ]:
```
validation_set = valid_datagen.flow_from_directory("/kaggle/input/tomatoleafdisease/Tomato Leaf Diseases -
                                            target_size = (224, 224),
                                            batch_size = 32,
                                            class_mode = 'categorical')
```

```
Found 7000 images belonging to 10 classes.
```

In [ ]:
```
iv3_model=tf.keras.applications.inception_v3.InceptionV3(weights='imagenet',
                                                          include_top=False,
                                                          input_shape = (224, 224, 3),
                                                          pooling='avg')
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/inception_v3/inception_v
3_weights_tf_dim_ordering_tf_kernels_notop.h5
87916544/87910968 [==============================] - 1s 0us/step
87924736/87910968 [==============================] - 1s 0us/step
```

In [ ]:
```
iv3_model.summary()
```

Model: "inception_v3"

Layer (type)	Output Shape	Param #	Connected to
input_1 (InputLayer)	[(None, 224, 224, 3)	0	
conv2d (Conv2D)	(None, 111, 111, 32)	864	input_1[0][0]
batch_normalization (BatchNorma	(None, 111, 111, 32)	96	conv2d[0][0]
activation (Activation)	(None, 111, 111, 32)	0	batch_normalization[0][0]
conv2d_1 (Conv2D)	(None, 109, 109, 32)	9216	activation[0][0]
batch_normalization_1 (BatchNor	(None, 109, 109, 32)	96	conv2d_1[0][0]
activation_1 (Activation)	(None, 109, 109, 32)	0	batch_normalization_1[0][0]
conv2d_2 (Conv2D)	(None, 109, 109, 64)	18432	activation_1[0][0]
batch_normalization_2 (BatchNor	(None, 109, 109, 64)	192	conv2d_2[0][0]
activation_2 (Activation)	(None, 109, 109, 64)	0	batch_normalization_2[0][0]
max_pooling2d (MaxPooling2D)	(None, 54, 54, 64)	0	activation_2[0][0]
conv2d_3 (Conv2D)	(None, 54, 54, 80)	5120	max_pooling2d[0][0]
batch_normalization_3 (BatchNor	(None, 54, 54, 80)	240	conv2d_3[0][0]
activation_3 (Activation)	(None, 54, 54, 80)	0	batch_normalization_3[0][0]
conv2d_4 (Conv2D)	(None, 52, 52, 192)	138240	activation_3[0][0]
batch_normalization_4 (BatchNor	(None, 52, 52, 192)	576	conv2d_4[0][0]
activation_4 (Activation)	(None, 52, 52, 192)	0	batch_normalization_4[0][0]
max_pooling2d_1 (MaxPooling2D)	(None, 25, 25, 192)	0	activation_4[0][0]
conv2d_8 (Conv2D)	(None, 25, 25, 64)	12288	max_pooling2d_1[0][0]
batch_normalization_8 (BatchNor	(None, 25, 25, 64)	192	conv2d_8[0][0]
activation_8 (Activation)	(None, 25, 25, 64)	0	batch_normalization_8[0][0]
conv2d_6 (Conv2D)	(None, 25, 25, 48)	9216	max_pooling2d_1[0][0]
conv2d_9 (Conv2D)	(None, 25, 25, 96)	55296	activation_8[0][0]
batch_normalization_6 (BatchNor	(None, 25, 25, 48)	144	conv2d_6[0][0]
batch_normalization_9 (BatchNor	(None, 25, 25, 96)	288	conv2d_9[0][0]
activation_6 (Activation)	(None, 25, 25, 48)	0	batch_normalization_6[0][0]
activation_9 (Activation)	(None, 25, 25, 96)	0	batch_normalization_9[0][0]
average_pooling2d (AveragePooli	(None, 25, 25, 192)	0	max_pooling2d_1[0][0]
conv2d_5 (Conv2D)	(None, 25, 25, 64)	12288	max_pooling2d_1[0][0]
conv2d_7 (Conv2D)	(None, 25, 25, 64)	76800	activation_6[0][0]
conv2d_10 (Conv2D)	(None, 25, 25, 96)	82944	activation_9[0][0]
conv2d_11 (Conv2D)	(None, 25, 25, 32)	6144	average_pooling2d[0][0]
batch_normalization_5 (BatchNor	(None, 25, 25, 64)	192	conv2d_5[0][0]
batch_normalization_7 (BatchNor	(None, 25, 25, 64)	192	conv2d_7[0][0]
batch_normalization_10 (BatchNo	(None, 25, 25, 96)	288	conv2d_10[0][0]
batch_normalization_11 (BatchNo	(None, 25, 25, 32)	96	conv2d_11[0][0]
activation_5 (Activation)	(None, 25, 25, 64)	0	batch_normalization_5[0][0]
activation_7 (Activation)	(None, 25, 25, 64)	0	batch_normalization_7[0][0]

activation_10 (Activation)	(None, 25, 25, 96)	0	batch_normalization_10[0][0]
activation_11 (Activation)	(None, 25, 25, 32)	0	batch_normalization_11[0][0]
mixed0 (Concatenate)	(None, 25, 25, 256)	0	activation_5[0][0] activation_7[0][0] activation_10[0][0] activation_11[0][0]
conv2d_15 (Conv2D)	(None, 25, 25, 64)	16384	mixed0[0][0]
batch_normalization_15 (BatchNo	(None, 25, 25, 64)	192	conv2d_15[0][0]
activation_15 (Activation)	(None, 25, 25, 64)	0	batch_normalization_15[0][0]
conv2d_13 (Conv2D)	(None, 25, 25, 48)	12288	mixed0[0][0]
conv2d_16 (Conv2D)	(None, 25, 25, 96)	55296	activation_15[0][0]
batch_normalization_13 (BatchNo	(None, 25, 25, 48)	144	conv2d_13[0][0]
batch_normalization_16 (BatchNo	(None, 25, 25, 96)	288	conv2d_16[0][0]
activation_13 (Activation)	(None, 25, 25, 48)	0	batch_normalization_13[0][0]
activation_16 (Activation)	(None, 25, 25, 96)	0	batch_normalization_16[0][0]
average_pooling2d_1 (AveragePoo	(None, 25, 25, 256)	0	mixed0[0][0]
conv2d_12 (Conv2D)	(None, 25, 25, 64)	16384	mixed0[0][0]
conv2d_14 (Conv2D)	(None, 25, 25, 64)	76800	activation_13[0][0]
conv2d_17 (Conv2D)	(None, 25, 25, 96)	82944	activation_16[0][0]
conv2d_18 (Conv2D)	(None, 25, 25, 64)	16384	average_pooling2d_1[0][0]
batch_normalization_12 (BatchNo	(None, 25, 25, 64)	192	conv2d_12[0][0]
batch_normalization_14 (BatchNo	(None, 25, 25, 64)	192	conv2d_14[0][0]
batch_normalization_17 (BatchNo	(None, 25, 25, 96)	288	conv2d_17[0][0]
batch_normalization_18 (BatchNo	(None, 25, 25, 64)	192	conv2d_18[0][0]
activation_12 (Activation)	(None, 25, 25, 64)	0	batch_normalization_12[0][0]
activation_14 (Activation)	(None, 25, 25, 64)	0	batch_normalization_14[0][0]
activation_17 (Activation)	(None, 25, 25, 96)	0	batch_normalization_17[0][0]
activation_18 (Activation)	(None, 25, 25, 64)	0	batch_normalization_18[0][0]
mixed1 (Concatenate)	(None, 25, 25, 288)	0	activation_12[0][0] activation_14[0][0] activation_17[0][0] activation_18[0][0]
conv2d_22 (Conv2D)	(None, 25, 25, 64)	18432	mixed1[0][0]
batch_normalization_22 (BatchNo	(None, 25, 25, 64)	192	conv2d_22[0][0]
activation_22 (Activation)	(None, 25, 25, 64)	0	batch_normalization_22[0][0]
conv2d_20 (Conv2D)	(None, 25, 25, 48)	13824	mixed1[0][0]
conv2d_23 (Conv2D)	(None, 25, 25, 96)	55296	activation_22[0][0]
batch_normalization_20 (BatchNo	(None, 25, 25, 48)	144	conv2d_20[0][0]
batch_normalization_23 (BatchNo	(None, 25, 25, 96)	288	conv2d_23[0][0]
activation_20 (Activation)	(None, 25, 25, 48)	0	batch_normalization_20[0][0]
activation_23 (Activation)	(None, 25, 25, 96)	0	batch_normalization_23[0][0]
average_pooling2d_2 (AveragePoo	(None, 25, 25, 288)	0	mixed1[0][0]
conv2d_19 (Conv2D)	(None, 25, 25, 64)	18432	mixed1[0][0]

conv2d_21 (Conv2D)	(None, 25, 25, 64)	76800	activation_20[0][0]
conv2d_24 (Conv2D)	(None, 25, 25, 96)	82944	activation_23[0][0]
conv2d_25 (Conv2D)	(None, 25, 25, 64)	18432	average_pooling2d_2[0][0]
batch_normalization_19 (BatchNo	(None, 25, 25, 64)	192	conv2d_19[0][0]
batch_normalization_21 (BatchNo	(None, 25, 25, 64)	192	conv2d_21[0][0]
batch_normalization_24 (BatchNo	(None, 25, 25, 96)	288	conv2d_24[0][0]
batch_normalization_25 (BatchNo	(None, 25, 25, 64)	192	conv2d_25[0][0]
activation_19 (Activation)	(None, 25, 25, 64)	0	batch_normalization_19[0][0]
activation_21 (Activation)	(None, 25, 25, 64)	0	batch_normalization_21[0][0]
activation_24 (Activation)	(None, 25, 25, 96)	0	batch_normalization_24[0][0]
activation_25 (Activation)	(None, 25, 25, 64)	0	batch_normalization_25[0][0]
mixed2 (Concatenate)	(None, 25, 25, 288)	0	activation_19[0][0] activation_21[0][0] activation_24[0][0] activation_25[0][0]
conv2d_27 (Conv2D)	(None, 25, 25, 64)	18432	mixed2[0][0]
batch_normalization_27 (BatchNo	(None, 25, 25, 64)	192	conv2d_27[0][0]
activation_27 (Activation)	(None, 25, 25, 64)	0	batch_normalization_27[0][0]
conv2d_28 (Conv2D)	(None, 25, 25, 96)	55296	activation_27[0][0]
batch_normalization_28 (BatchNo	(None, 25, 25, 96)	288	conv2d_28[0][0]
activation_28 (Activation)	(None, 25, 25, 96)	0	batch_normalization_28[0][0]
conv2d_26 (Conv2D)	(None, 12, 12, 384)	995328	mixed2[0][0]
conv2d_29 (Conv2D)	(None, 12, 12, 96)	82944	activation_28[0][0]
batch_normalization_26 (BatchNo	(None, 12, 12, 384)	1152	conv2d_26[0][0]
batch_normalization_29 (BatchNo	(None, 12, 12, 96)	288	conv2d_29[0][0]
activation_26 (Activation)	(None, 12, 12, 384)	0	batch_normalization_26[0][0]
activation_29 (Activation)	(None, 12, 12, 96)	0	batch_normalization_29[0][0]
max_pooling2d_2 (MaxPooling2D)	(None, 12, 12, 288)	0	mixed2[0][0]
mixed3 (Concatenate)	(None, 12, 12, 768)	0	activation_26[0][0] activation_29[0][0] max_pooling2d_2[0][0]
conv2d_34 (Conv2D)	(None, 12, 12, 128)	98304	mixed3[0][0]
batch_normalization_34 (BatchNo	(None, 12, 12, 128)	384	conv2d_34[0][0]
activation_34 (Activation)	(None, 12, 12, 128)	0	batch_normalization_34[0][0]
conv2d_35 (Conv2D)	(None, 12, 12, 128)	114688	activation_34[0][0]
batch_normalization_35 (BatchNo	(None, 12, 12, 128)	384	conv2d_35[0][0]
activation_35 (Activation)	(None, 12, 12, 128)	0	batch_normalization_35[0][0]
conv2d_31 (Conv2D)	(None, 12, 12, 128)	98304	mixed3[0][0]
conv2d_36 (Conv2D)	(None, 12, 12, 128)	114688	activation_35[0][0]
batch_normalization_31 (BatchNo	(None, 12, 12, 128)	384	conv2d_31[0][0]
batch_normalization_36 (BatchNo	(None, 12, 12, 128)	384	conv2d_36[0][0]
activation_31 (Activation)	(None, 12, 12, 128)	0	batch_normalization_31[0][0]
activation_36 (Activation)	(None, 12, 12, 128)	0	batch_normalization_36[0][0]

conv2d_32 (Conv2D)	(None, 12, 12, 128)	114688	activation_31[0][0]
conv2d_37 (Conv2D)	(None, 12, 12, 128)	114688	activation_36[0][0]
batch_normalization_32 (BatchNo	(None, 12, 12, 128)	384	conv2d_32[0][0]
batch_normalization_37 (BatchNo	(None, 12, 12, 128)	384	conv2d_37[0][0]
activation_32 (Activation)	(None, 12, 12, 128)	0	batch_normalization_32[0][0]
activation_37 (Activation)	(None, 12, 12, 128)	0	batch_normalization_37[0][0]
average_pooling2d_3 (AveragePoo	(None, 12, 12, 768)	0	mixed3[0][0]
conv2d_30 (Conv2D)	(None, 12, 12, 192)	147456	mixed3[0][0]
conv2d_33 (Conv2D)	(None, 12, 12, 192)	172032	activation_32[0][0]
conv2d_38 (Conv2D)	(None, 12, 12, 192)	172032	activation_37[0][0]
conv2d_39 (Conv2D)	(None, 12, 12, 192)	147456	average_pooling2d_3[0][0]
batch_normalization_30 (BatchNo	(None, 12, 12, 192)	576	conv2d_30[0][0]
batch_normalization_33 (BatchNo	(None, 12, 12, 192)	576	conv2d_33[0][0]
batch_normalization_38 (BatchNo	(None, 12, 12, 192)	576	conv2d_38[0][0]
batch_normalization_39 (BatchNo	(None, 12, 12, 192)	576	conv2d_39[0][0]
activation_30 (Activation)	(None, 12, 12, 192)	0	batch_normalization_30[0][0]
activation_33 (Activation)	(None, 12, 12, 192)	0	batch_normalization_33[0][0]
activation_38 (Activation)	(None, 12, 12, 192)	0	batch_normalization_38[0][0]
activation_39 (Activation)	(None, 12, 12, 192)	0	batch_normalization_39[0][0]
mixed4 (Concatenate)	(None, 12, 12, 768)	0	activation_30[0][0] activation_33[0][0] activation_38[0][0] activation_39[0][0]
conv2d_44 (Conv2D)	(None, 12, 12, 160)	122880	mixed4[0][0]
batch_normalization_44 (BatchNo	(None, 12, 12, 160)	480	conv2d_44[0][0]
activation_44 (Activation)	(None, 12, 12, 160)	0	batch_normalization_44[0][0]
conv2d_45 (Conv2D)	(None, 12, 12, 160)	179200	activation_44[0][0]
batch_normalization_45 (BatchNo	(None, 12, 12, 160)	480	conv2d_45[0][0]
activation_45 (Activation)	(None, 12, 12, 160)	0	batch_normalization_45[0][0]
conv2d_41 (Conv2D)	(None, 12, 12, 160)	122880	mixed4[0][0]
conv2d_46 (Conv2D)	(None, 12, 12, 160)	179200	activation_45[0][0]
batch_normalization_41 (BatchNo	(None, 12, 12, 160)	480	conv2d_41[0][0]
batch_normalization_46 (BatchNo	(None, 12, 12, 160)	480	conv2d_46[0][0]
activation_41 (Activation)	(None, 12, 12, 160)	0	batch_normalization_41[0][0]
activation_46 (Activation)	(None, 12, 12, 160)	0	batch_normalization_46[0][0]
conv2d_42 (Conv2D)	(None, 12, 12, 160)	179200	activation_41[0][0]
conv2d_47 (Conv2D)	(None, 12, 12, 160)	179200	activation_46[0][0]
batch_normalization_42 (BatchNo	(None, 12, 12, 160)	480	conv2d_42[0][0]
batch_normalization_47 (BatchNo	(None, 12, 12, 160)	480	conv2d_47[0][0]
activation_42 (Activation)	(None, 12, 12, 160)	0	batch_normalization_42[0][0]
activation_47 (Activation)	(None, 12, 12, 160)	0	batch_normalization_47[0][0]

average_pooling2d_4 (AveragePoo	(None, 12, 12, 768)	0	mixed4[0][0]
conv2d_40 (Conv2D)	(None, 12, 12, 192)	147456	mixed4[0][0]
conv2d_43 (Conv2D)	(None, 12, 12, 192)	215040	activation_42[0][0]
conv2d_48 (Conv2D)	(None, 12, 12, 192)	215040	activation_47[0][0]
conv2d_49 (Conv2D)	(None, 12, 12, 192)	147456	average_pooling2d_4[0][0]
batch_normalization_40 (BatchNo	(None, 12, 12, 192)	576	conv2d_40[0][0]
batch_normalization_43 (BatchNo	(None, 12, 12, 192)	576	conv2d_43[0][0]
batch_normalization_48 (BatchNo	(None, 12, 12, 192)	576	conv2d_48[0][0]
batch_normalization_49 (BatchNo	(None, 12, 12, 192)	576	conv2d_49[0][0]
activation_40 (Activation)	(None, 12, 12, 192)	0	batch_normalization_40[0][0]
activation_43 (Activation)	(None, 12, 12, 192)	0	batch_normalization_43[0][0]
activation_48 (Activation)	(None, 12, 12, 192)	0	batch_normalization_48[0][0]
activation_49 (Activation)	(None, 12, 12, 192)	0	batch_normalization_49[0][0]
mixed5 (Concatenate)	(None, 12, 12, 768)	0	activation_40[0][0] activation_43[0][0] activation_48[0][0] activation_49[0][0]
conv2d_54 (Conv2D)	(None, 12, 12, 160)	122880	mixed5[0][0]
batch_normalization_54 (BatchNo	(None, 12, 12, 160)	480	conv2d_54[0][0]
activation_54 (Activation)	(None, 12, 12, 160)	0	batch_normalization_54[0][0]
conv2d_55 (Conv2D)	(None, 12, 12, 160)	179200	activation_54[0][0]
batch_normalization_55 (BatchNo	(None, 12, 12, 160)	480	conv2d_55[0][0]
activation_55 (Activation)	(None, 12, 12, 160)	0	batch_normalization_55[0][0]
conv2d_51 (Conv2D)	(None, 12, 12, 160)	122880	mixed5[0][0]
conv2d_56 (Conv2D)	(None, 12, 12, 160)	179200	activation_55[0][0]
batch_normalization_51 (BatchNo	(None, 12, 12, 160)	480	conv2d_51[0][0]
batch_normalization_56 (BatchNo	(None, 12, 12, 160)	480	conv2d_56[0][0]
activation_51 (Activation)	(None, 12, 12, 160)	0	batch_normalization_51[0][0]
activation_56 (Activation)	(None, 12, 12, 160)	0	batch_normalization_56[0][0]
conv2d_52 (Conv2D)	(None, 12, 12, 160)	179200	activation_51[0][0]
conv2d_57 (Conv2D)	(None, 12, 12, 160)	179200	activation_56[0][0]
batch_normalization_52 (BatchNo	(None, 12, 12, 160)	480	conv2d_52[0][0]
batch_normalization_57 (BatchNo	(None, 12, 12, 160)	480	conv2d_57[0][0]
activation_52 (Activation)	(None, 12, 12, 160)	0	batch_normalization_52[0][0]
activation_57 (Activation)	(None, 12, 12, 160)	0	batch_normalization_57[0][0]
average_pooling2d_5 (AveragePoo	(None, 12, 12, 768)	0	mixed5[0][0]
conv2d_50 (Conv2D)	(None, 12, 12, 192)	147456	mixed5[0][0]
conv2d_53 (Conv2D)	(None, 12, 12, 192)	215040	activation_52[0][0]
conv2d_58 (Conv2D)	(None, 12, 12, 192)	215040	activation_57[0][0]
conv2d_59 (Conv2D)	(None, 12, 12, 192)	147456	average_pooling2d_5[0][0]
batch_normalization_50 (BatchNo	(None, 12, 12, 192)	576	conv2d_50[0][0]
batch_normalization_53 (BatchNo	(None, 12, 12, 192)	576	conv2d_53[0][0]

batch_normalization_58 (BatchNo	(None, 12, 12, 192)	576	conv2d_58[0][0]
batch_normalization_59 (BatchNo	(None, 12, 12, 192)	576	conv2d_59[0][0]
activation_50 (Activation)	(None, 12, 12, 192)	0	batch_normalization_50[0][0]
activation_53 (Activation)	(None, 12, 12, 192)	0	batch_normalization_53[0][0]
activation_58 (Activation)	(None, 12, 12, 192)	0	batch_normalization_58[0][0]
activation_59 (Activation)	(None, 12, 12, 192)	0	batch_normalization_59[0][0]
mixed6 (Concatenate)	(None, 12, 12, 768)	0	activation_50[0][0] activation_53[0][0] activation_58[0][0] activation_59[0][0]
conv2d_64 (Conv2D)	(None, 12, 12, 192)	147456	mixed6[0][0]
batch_normalization_64 (BatchNo	(None, 12, 12, 192)	576	conv2d_64[0][0]
activation_64 (Activation)	(None, 12, 12, 192)	0	batch_normalization_64[0][0]
conv2d_65 (Conv2D)	(None, 12, 12, 192)	258048	activation_64[0][0]
batch_normalization_65 (BatchNo	(None, 12, 12, 192)	576	conv2d_65[0][0]
activation_65 (Activation)	(None, 12, 12, 192)	0	batch_normalization_65[0][0]
conv2d_61 (Conv2D)	(None, 12, 12, 192)	147456	mixed6[0][0]
conv2d_66 (Conv2D)	(None, 12, 12, 192)	258048	activation_65[0][0]
batch_normalization_61 (BatchNo	(None, 12, 12, 192)	576	conv2d_61[0][0]
batch_normalization_66 (BatchNo	(None, 12, 12, 192)	576	conv2d_66[0][0]
activation_61 (Activation)	(None, 12, 12, 192)	0	batch_normalization_61[0][0]
activation_66 (Activation)	(None, 12, 12, 192)	0	batch_normalization_66[0][0]
conv2d_62 (Conv2D)	(None, 12, 12, 192)	258048	activation_61[0][0]
conv2d_67 (Conv2D)	(None, 12, 12, 192)	258048	activation_66[0][0]
batch_normalization_62 (BatchNo	(None, 12, 12, 192)	576	conv2d_62[0][0]
batch_normalization_67 (BatchNo	(None, 12, 12, 192)	576	conv2d_67[0][0]
activation_62 (Activation)	(None, 12, 12, 192)	0	batch_normalization_62[0][0]
activation_67 (Activation)	(None, 12, 12, 192)	0	batch_normalization_67[0][0]
average_pooling2d_6 (AveragePoo	(None, 12, 12, 768)	0	mixed6[0][0]
conv2d_60 (Conv2D)	(None, 12, 12, 192)	147456	mixed6[0][0]
conv2d_63 (Conv2D)	(None, 12, 12, 192)	258048	activation_62[0][0]
conv2d_68 (Conv2D)	(None, 12, 12, 192)	258048	activation_67[0][0]
conv2d_69 (Conv2D)	(None, 12, 12, 192)	147456	average_pooling2d_6[0][0]
batch_normalization_60 (BatchNo	(None, 12, 12, 192)	576	conv2d_60[0][0]
batch_normalization_63 (BatchNo	(None, 12, 12, 192)	576	conv2d_63[0][0]
batch_normalization_68 (BatchNo	(None, 12, 12, 192)	576	conv2d_68[0][0]
batch_normalization_69 (BatchNo	(None, 12, 12, 192)	576	conv2d_69[0][0]
activation_60 (Activation)	(None, 12, 12, 192)	0	batch_normalization_60[0][0]
activation_63 (Activation)	(None, 12, 12, 192)	0	batch_normalization_63[0][0]
activation_68 (Activation)	(None, 12, 12, 192)	0	batch_normalization_68[0][0]
activation_69 (Activation)	(None, 12, 12, 192)	0	batch_normalization_69[0][0]

mixed7 (Concatenate)	(None, 12, 12, 768)	0	activation_60[0][0] activation_63[0][0] activation_68[0][0] activation_69[0][0]
conv2d_72 (Conv2D)	(None, 12, 12, 192)	147456	mixed7[0][0]
batch_normalization_72 (BatchNo	(None, 12, 12, 192)	576	conv2d_72[0][0]
activation_72 (Activation)	(None, 12, 12, 192)	0	batch_normalization_72[0][0]
conv2d_73 (Conv2D)	(None, 12, 12, 192)	258048	activation_72[0][0]
batch_normalization_73 (BatchNo	(None, 12, 12, 192)	576	conv2d_73[0][0]
activation_73 (Activation)	(None, 12, 12, 192)	0	batch_normalization_73[0][0]
conv2d_70 (Conv2D)	(None, 12, 12, 192)	147456	mixed7[0][0]
conv2d_74 (Conv2D)	(None, 12, 12, 192)	258048	activation_73[0][0]
batch_normalization_70 (BatchNo	(None, 12, 12, 192)	576	conv2d_70[0][0]
batch_normalization_74 (BatchNo	(None, 12, 12, 192)	576	conv2d_74[0][0]
activation_70 (Activation)	(None, 12, 12, 192)	0	batch_normalization_70[0][0]
activation_74 (Activation)	(None, 12, 12, 192)	0	batch_normalization_74[0][0]
conv2d_71 (Conv2D)	(None, 5, 5, 320)	552960	activation_70[0][0]
conv2d_75 (Conv2D)	(None, 5, 5, 192)	331776	activation_74[0][0]
batch_normalization_71 (BatchNo	(None, 5, 5, 320)	960	conv2d_71[0][0]
batch_normalization_75 (BatchNo	(None, 5, 5, 192)	576	conv2d_75[0][0]
activation_71 (Activation)	(None, 5, 5, 320)	0	batch_normalization_71[0][0]
activation_75 (Activation)	(None, 5, 5, 192)	0	batch_normalization_75[0][0]
max_pooling2d_3 (MaxPooling2D)	(None, 5, 5, 768)	0	mixed7[0][0]
mixed8 (Concatenate)	(None, 5, 5, 1280)	0	activation_71[0][0] activation_75[0][0] max_pooling2d_3[0][0]
conv2d_80 (Conv2D)	(None, 5, 5, 448)	573440	mixed8[0][0]
batch_normalization_80 (BatchNo	(None, 5, 5, 448)	1344	conv2d_80[0][0]
activation_80 (Activation)	(None, 5, 5, 448)	0	batch_normalization_80[0][0]
conv2d_77 (Conv2D)	(None, 5, 5, 384)	491520	mixed8[0][0]
conv2d_81 (Conv2D)	(None, 5, 5, 384)	1548288	activation_80[0][0]
batch_normalization_77 (BatchNo	(None, 5, 5, 384)	1152	conv2d_77[0][0]
batch_normalization_81 (BatchNo	(None, 5, 5, 384)	1152	conv2d_81[0][0]
activation_77 (Activation)	(None, 5, 5, 384)	0	batch_normalization_77[0][0]
activation_81 (Activation)	(None, 5, 5, 384)	0	batch_normalization_81[0][0]
conv2d_78 (Conv2D)	(None, 5, 5, 384)	442368	activation_77[0][0]
conv2d_79 (Conv2D)	(None, 5, 5, 384)	442368	activation_77[0][0]
conv2d_82 (Conv2D)	(None, 5, 5, 384)	442368	activation_81[0][0]
conv2d_83 (Conv2D)	(None, 5, 5, 384)	442368	activation_81[0][0]
average_pooling2d_7 (AveragePoo	(None, 5, 5, 1280)	0	mixed8[0][0]
conv2d_76 (Conv2D)	(None, 5, 5, 320)	409600	mixed8[0][0]
batch_normalization_78 (BatchNo	(None, 5, 5, 384)	1152	conv2d_78[0][0]
batch_normalization_79 (BatchNo	(None, 5, 5, 384)	1152	conv2d_79[0][0]

batch_normalization_82 (BatchNo	(None, 5, 5, 384)	1152	conv2d_82[0][0]
batch_normalization_83 (BatchNo	(None, 5, 5, 384)	1152	conv2d_83[0][0]
conv2d_84 (Conv2D)	(None, 5, 5, 192)	245760	average_pooling2d_7[0][0]
batch_normalization_76 (BatchNo	(None, 5, 5, 320)	960	conv2d_76[0][0]
activation_78 (Activation)	(None, 5, 5, 384)	0	batch_normalization_78[0][0]
activation_79 (Activation)	(None, 5, 5, 384)	0	batch_normalization_79[0][0]
activation_82 (Activation)	(None, 5, 5, 384)	0	batch_normalization_82[0][0]
activation_83 (Activation)	(None, 5, 5, 384)	0	batch_normalization_83[0][0]
batch_normalization_84 (BatchNo	(None, 5, 5, 192)	576	conv2d_84[0][0]
activation_76 (Activation)	(None, 5, 5, 320)	0	batch_normalization_76[0][0]
mixed9_0 (Concatenate)	(None, 5, 5, 768)	0	activation_78[0][0] activation_79[0][0]
concatenate (Concatenate)	(None, 5, 5, 768)	0	activation_82[0][0] activation_83[0][0]
activation_84 (Activation)	(None, 5, 5, 192)	0	batch_normalization_84[0][0]
mixed9 (Concatenate)	(None, 5, 5, 2048)	0	activation_76[0][0] mixed9_0[0][0] concatenate[0][0] activation_84[0][0]
conv2d_89 (Conv2D)	(None, 5, 5, 448)	917504	mixed9[0][0]
batch_normalization_89 (BatchNo	(None, 5, 5, 448)	1344	conv2d_89[0][0]
activation_89 (Activation)	(None, 5, 5, 448)	0	batch_normalization_89[0][0]
conv2d_86 (Conv2D)	(None, 5, 5, 384)	786432	mixed9[0][0]
conv2d_90 (Conv2D)	(None, 5, 5, 384)	1548288	activation_89[0][0]
batch_normalization_86 (BatchNo	(None, 5, 5, 384)	1152	conv2d_86[0][0]
batch_normalization_90 (BatchNo	(None, 5, 5, 384)	1152	conv2d_90[0][0]
activation_86 (Activation)	(None, 5, 5, 384)	0	batch_normalization_86[0][0]
activation_90 (Activation)	(None, 5, 5, 384)	0	batch_normalization_90[0][0]
conv2d_87 (Conv2D)	(None, 5, 5, 384)	442368	activation_86[0][0]
conv2d_88 (Conv2D)	(None, 5, 5, 384)	442368	activation_86[0][0]
conv2d_91 (Conv2D)	(None, 5, 5, 384)	442368	activation_90[0][0]
conv2d_92 (Conv2D)	(None, 5, 5, 384)	442368	activation_90[0][0]
average_pooling2d_8 (AveragePoo	(None, 5, 5, 2048)	0	mixed9[0][0]
conv2d_85 (Conv2D)	(None, 5, 5, 320)	655360	mixed9[0][0]
batch_normalization_87 (BatchNo	(None, 5, 5, 384)	1152	conv2d_87[0][0]
batch_normalization_88 (BatchNo	(None, 5, 5, 384)	1152	conv2d_88[0][0]
batch_normalization_91 (BatchNo	(None, 5, 5, 384)	1152	conv2d_91[0][0]
batch_normalization_92 (BatchNo	(None, 5, 5, 384)	1152	conv2d_92[0][0]
conv2d_93 (Conv2D)	(None, 5, 5, 192)	393216	average_pooling2d_8[0][0]
batch_normalization_85 (BatchNo	(None, 5, 5, 320)	960	conv2d_85[0][0]
activation_87 (Activation)	(None, 5, 5, 384)	0	batch_normalization_87[0][0]
activation_88 (Activation)	(None, 5, 5, 384)	0	batch_normalization_88[0][0]

```
activation_91 (Activation)      (None, 5, 5, 384)    0           batch_normalization_91[0][0]
__________________________________________________________________________________________________
activation_92 (Activation)      (None, 5, 5, 384)    0           batch_normalization_92[0][0]
__________________________________________________________________________________________________
batch_normalization_93 (BatchNo (None, 5, 5, 192)    576         conv2d_93[0][0]
__________________________________________________________________________________________________
activation_85 (Activation)      (None, 5, 5, 320)    0           batch_normalization_85[0][0]
__________________________________________________________________________________________________
mixed9_1 (Concatenate)          (None, 5, 5, 768)    0           activation_87[0][0]
                                                                 activation_88[0][0]
__________________________________________________________________________________________________
concatenate_1 (Concatenate)     (None, 5, 5, 768)    0           activation_91[0][0]
                                                                 activation_92[0][0]
__________________________________________________________________________________________________
activation_93 (Activation)      (None, 5, 5, 192)    0           batch_normalization_93[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 (Globa (None, 2048)         0           mixed10[0][0]
==================================================================================================
Total params: 21,802,784
Trainable params: 21,768,352
Non-trainable params: 34,432
__________________________________________________________________________________________________
```

In [ ]:

```
iv3model = Sequential()
iv3model.add(iv3_model)
#iv3model.add(Flatten())
iv3model.add(Dense(2048,activation="relu"))
iv3model.add(Dense(4096,activation="relu"))
iv3model.add(Dense(10, activation="softmax"))
iv3model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
inception_v3 (Functional)    (None, 2048)              21802784
_________________________________________________________________
dense (Dense)                (None, 2048)              4196352
_________________________________________________________________
dense_1 (Dense)              (None, 4096)              8392704
_________________________________________________________________
dense_2 (Dense)              (None, 10)                40970
=================================================================
Total params: 34,432,810
Trainable params: 34,398,378
Non-trainable params: 34,432
_________________________________________________________________
```

In [ ]:

```
iv3model.compile(loss='categorical_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])
```

In [ ]:

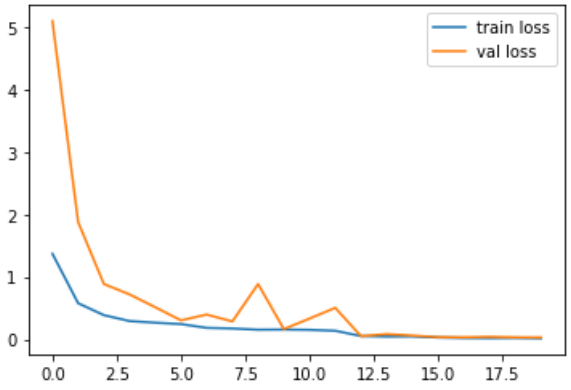
```
model_save = ModelCheckpoint('./iv3.h5',
                             save_best_only = True,
                             save_weights_only = False,
                             monitor = 'val_loss',
                             mode = 'min', verbose = 1)
early_stop = EarlyStopping(monitor = 'val_loss', min_delta = 0.001,
                           patience = 10, mode = 'min', verbose = 1,
                           restore_best_weights = True)
reduce_lr = ReduceLROnPlateau(monitor = 'val_loss', factor = 0.3,
                              patience = 2, min_delta = 0.001,
                              mode = 'min', verbose = 1)
model = iv3model.fit(x = training_set, validation_data=validation_set, epochs=20, callbacks = [model_save,
```

```
Epoch 1/20
297/297 [==============================] - 273s 873ms/step - loss: 1.3689 - accuracy: 0.5129 - val_loss: 5.
0945 - val_accuracy: 0.2851

Epoch 00001: val_loss improved from inf to 5.09448, saving model to ./iv3.h5
Epoch 2/20
297/297 [==============================] - 150s 506ms/step - loss: 0.5769 - accuracy: 0.8178 - val_loss: 1.
8762 - val_accuracy: 0.5096

Epoch 00002: val_loss improved from 5.09448 to 1.87621, saving model to ./iv3.h5
Epoch 3/20
297/297 [==============================] - 150s 504ms/step - loss: 0.3860 - accuracy: 0.8820 - val_loss: 0.
8850 - val_accuracy: 0.7871

Epoch 00003: val_loss improved from 1.87621 to 0.88495, saving model to ./iv3.h5
Epoch 4/20
297/297 [==============================] - 150s 504ms/step - loss: 0.2912 - accuracy: 0.9083 - val_loss: 0.
7179 - val_accuracy: 0.7770

Epoch 00004: val_loss improved from 0.88495 to 0.71791, saving model to ./iv3.h5
Epoch 5/20
297/297 [==============================] - 149s 501ms/step - loss: 0.2652 - accuracy: 0.9153 - val_loss: 0.
5124 - val_accuracy: 0.8506

Epoch 00005: val_loss improved from 0.71791 to 0.51238, saving model to ./iv3.h5
Epoch 6/20
297/297 [==============================] - 150s 505ms/step - loss: 0.2428 - accuracy: 0.9253 - val_loss: 0.
3028 - val_accuracy: 0.8954

Epoch 00006: val_loss improved from 0.51238 to 0.30280, saving model to ./iv3.h5
Epoch 7/20
297/297 [==============================] - 148s 499ms/step - loss: 0.1825 - accuracy: 0.9424 - val_loss: 0.
3953 - val_accuracy: 0.8880

Epoch 00007: val_loss did not improve from 0.30280
Epoch 8/20
297/297 [==============================] - 149s 503ms/step - loss: 0.1724 - accuracy: 0.9479 - val_loss: 0.
2858 - val_accuracy: 0.9106

Epoch 00008: val_loss improved from 0.30280 to 0.28583, saving model to ./iv3.h5
Epoch 9/20
297/297 [==============================] - 153s 515ms/step - loss: 0.1531 - accuracy: 0.9506 - val_loss: 0.
8844 - val_accuracy: 0.7897

Epoch 00009: val_loss did not improve from 0.28583
Epoch 10/20
297/297 [==============================] - 150s 503ms/step - loss: 0.1552 - accuracy: 0.9519 - val_loss: 0.
1631 - val_accuracy: 0.9497

Epoch 00010: val_loss improved from 0.28583 to 0.16306, saving model to ./iv3.h5
Epoch 11/20
297/297 [==============================] - 147s 496ms/step - loss: 0.1506 - accuracy: 0.9521 - val_loss: 0.
3313 - val_accuracy: 0.9039

Epoch 00011: val_loss did not improve from 0.16306
Epoch 12/20
297/297 [==============================] - 148s 497ms/step - loss: 0.1357 - accuracy: 0.9574 - val_loss: 0.
5031 - val_accuracy: 0.8633

Epoch 00012: val_loss did not improve from 0.16306

Epoch 00012: ReduceLROnPlateau reducing learning rate to 0.0003000000142492354.
Epoch 13/20
297/297 [==============================] - 147s 493ms/step - loss: 0.0518 - accuracy: 0.9822 - val_loss: 0.
0499 - val_accuracy: 0.9844

Epoch 00013: val_loss improved from 0.16306 to 0.04991, saving model to ./iv3.h5
Epoch 14/20
297/297 [==============================] - 150s 504ms/step - loss: 0.0428 - accuracy: 0.9857 - val_loss: 0.
0825 - val_accuracy: 0.9759

Epoch 00014: val_loss did not improve from 0.04991
Epoch 15/20
297/297 [==============================] - 148s 498ms/step - loss: 0.0453 - accuracy: 0.9837 - val_loss: 0.
0595 - val_accuracy: 0.9820

Epoch 00015: val_loss did not improve from 0.04991

Epoch 00015: ReduceLROnPlateau reducing learning rate to 9.000000427477062e-05.
Epoch 16/20
```

```
297/297 [==============================] - 148s 498ms/step - loss: 0.0283 - accuracy: 0.9909 - val_loss: 0.
0333 - val_accuracy: 0.9913

Epoch 00016: val_loss improved from 0.04991 to 0.03334, saving model to ./iv3.h5
Epoch 17/20
297/297 [==============================] - 147s 496ms/step - loss: 0.0203 - accuracy: 0.9936 - val_loss: 0.
0308 - val_accuracy: 0.9914

Epoch 00017: val_loss improved from 0.03334 to 0.03075, saving model to ./iv3.h5
Epoch 18/20
297/297 [==============================] - 147s 495ms/step - loss: 0.0176 - accuracy: 0.9936 - val_loss: 0.
0389 - val_accuracy: 0.9893

Epoch 00018: val_loss did not improve from 0.03075
Epoch 19/20
297/297 [==============================] - 150s 503ms/step - loss: 0.0212 - accuracy: 0.9922 - val_loss: 0.
0300 - val_accuracy: 0.9910

Epoch 00019: val_loss improved from 0.03075 to 0.03000, saving model to ./iv3.h5

Epoch 00019: ReduceLROnPlateau reducing learning rate to 2.700000040931627e-05.
Epoch 20/20
297/297 [==============================] - 148s 499ms/step - loss: 0.0154 - accuracy: 0.9952 - val_loss: 0.
0291 - val_accuracy: 0.9926

Epoch 00020: val_loss improved from 0.03000 to 0.02911, saving model to ./iv3.h5
```

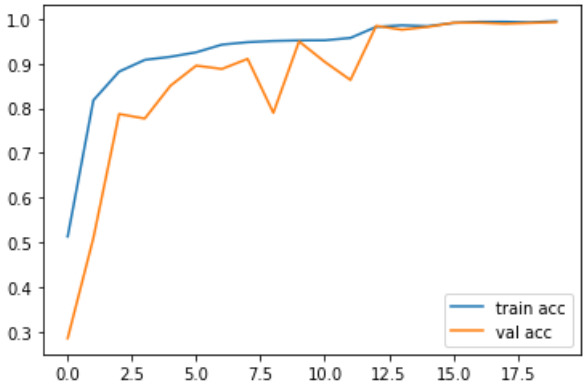
In [ ]:

```
# Loss Plot
plt.plot(model.history['loss'], label='train loss')
plt.plot(model.history['val_loss'], label='val loss')
plt.legend()
plt.show()
```



In [ ]:

```
# accuracy plot
plt.plot(model.history['accuracy'], label='train acc')
plt.plot(model.history['val_accuracy'], label='val acc')
plt.legend()
plt.show()
```



In [ ]:

```
print("training_accuracy", model.history['accuracy'][-1])
print("validation_accuracy", model.history['val_accuracy'][-1])
```

```
training_accuracy 0.9951578974723816
validation_accuracy 0.9925714135169983
```

In [ ]:
```
test_set = test_datagen.flow_from_directory("/kaggle/input/tomatoleafdisease/Tomato Leaf Diseases - Copy/Te
                                            target_size = (224, 224),
                                            batch_size = 500,
                                            class_mode = 'categorical')
```

```
Found 500 images belonging to 10 classes.
```

In [ ]:
```
test_data, test_labels = test_set.next()
```

In [ ]:
```
test_scores = iv3model.evaluate(test_data, test_labels)
print("Test Accuracy: ",(test_scores[1]))
```

```
16/16 [==============================] - 3s 73ms/step - loss: 0.0107 - accuracy: 0.9960
Test Accuracy:  0.9959999918937683
```

In [ ]:
```
y_pred_test=iv3model.predict(test_data).argmax(axis=1)
print(y_pred_test)
```

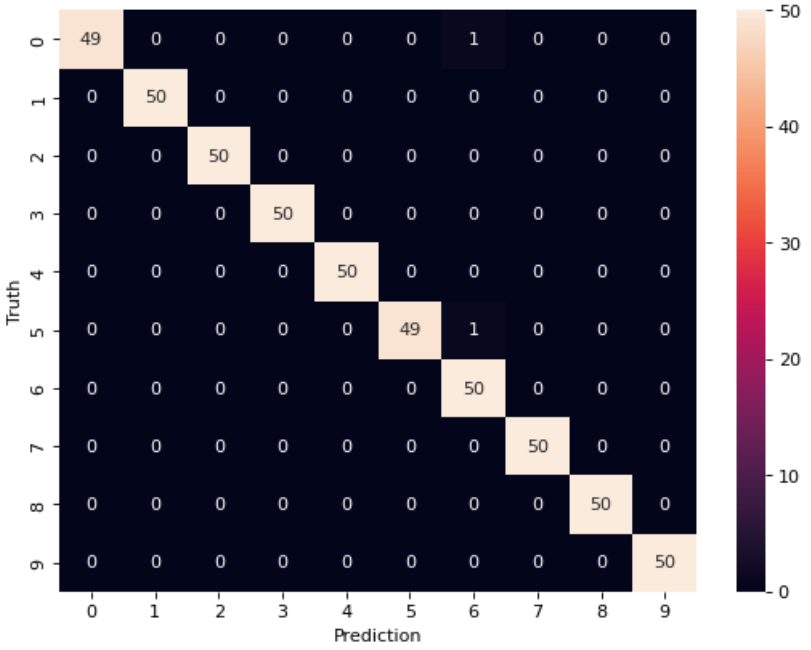
```
[5 9 6 6 6 2 8 4 3 7 5 1 3 9 7 4 3 6 8 6 6 9 9 3 9 4 6 8 7 0 0 5 8 7 2 6 8
 6 3 9 1 7 5 6 2 9 4 5 2 6 8 7 5 1 5 0 6 0 5 5 3 2 5 6 4 1 0 1 1 9 6 7 6 7
 3 4 5 0 3 2 9 8 0 2 7 3 8 8 6 3 3 1 0 6 4 1 6 8 3 6 9 6 7 0 2 8 1 0 5 7 7
 0 1 9 0 3 4 1 7 2 9 4 8 4 3 8 4 7 6 0 2 0 8 0 3 0 1 1 9 9 1 2 2 3 3 7 5 3
 1 1 5 6 9 3 7 0 0 5 6 9 6 2 3 7 9 2 1 5 4 3 4 1 0 4 4 8 7 3 4 7 2 4 9 4 9
 7 9 9 8 5 5 4 9 2 1 7 6 8 0 1 8 9 6 8 1 0 2 9 0 4 1 9 1 7 9 7 6 7 1 1 5 9
 7 5 8 9 1 1 7 8 2 5 5 0 4 2 8 6 5 6 2 1 9 4 7 8 1 9 9 3 3 0 7 8 6 5 2 4 2
 0 3 0 0 0 3 0 7 8 2 9 9 0 2 4 0 5 7 7 3 7 1 3 2 0 7 7 5 1 5 4 8 4 5 2 2 5
 7 8 8 8 9 3 3 6 0 4 4 5 2 3 6 8 0 5 3 1 1 9 4 2 6 8 4 3 8 4 6 9 0 7 5 1 6
 1 0 0 3 1 5 5 2 4 1 2 7 4 8 7 0 0 0 0 4 3 1 8 8 9 6 3 1 8 8 2 4 5 1 1 6 3
 9 0 7 7 6 9 6 5 8 2 2 4 3 7 2 5 2 0 3 2 9 7 6 6 4 2 8 3 8 6 5 4 9 5 6 3 2
 6 1 1 7 7 1 2 9 3 4 5 3 7 5 4 0 3 1 4 2 1 8 2 2 3 7 6 8 0 9 5 3 5 8 6 7 3
 9 0 8 5 4 9 0 8 8 7 5 8 0 6 9 8 5 2 4 6 4 2 0 1 5 9 2 6 5 1 2 5 7 4 2 1 4
 6 7 1 3 4 9 3 8 9 9 4 6 2 8 3 4 6 2 4]
```

In [ ]:
```
#Predicting the test data

pred_labels = iv3model.predict(test_data)
def roundoff(arr):
    arr[np.argwhere(arr != arr.max())] = 0
    arr[np.argwhere(arr == arr.max())] = 1
    return arr

for labels in pred_labels:
    labels = roundoff(labels)

print(classification_report(test_labels, pred_labels))
```

```
              precision    recall  f1-score   support

           0       1.00      0.98      0.99        50
           1       1.00      1.00      1.00        50
           2       1.00      1.00      1.00        50
           3       1.00      1.00      1.00        50
           4       1.00      1.00      1.00        50
           5       1.00      0.98      0.99        50
           6       0.96      1.00      0.98        50
           7       1.00      1.00      1.00        50
           8       1.00      1.00      1.00        50
           9       1.00      1.00      1.00        50

   micro avg       1.00      1.00      1.00       500
   macro avg       1.00      1.00      1.00       500
weighted avg       1.00      1.00      1.00       500
 samples avg       1.00      1.00      1.00       500
```

In [ ]:
```
pred_ls = np.argmax(pred_labels, axis=1)
test_ls = np.argmax(test_labels, axis=1)

conf_arr = confusion_matrix(test_ls, pred_ls)

plt.figure(figsize=(8, 6), dpi=80, facecolor='w', edgecolor='k')

ax = sns.heatmap(conf_arr, annot=True, fmt='d')
plt.xlabel('Prediction')
```

```
plt.ylabel('Truth')
plt.show(ax)
```



In [ ]:

```
from tensorflow.keras.preprocessing import image
from tensorflow.keras.models import load_model
# model = load_model("my_best_model.hdf5")
img = image.load_img('/kaggle/input/tomatoleafdisease/Tomato Leaf Diseases - Copy/Testing set/Tomato___Late
x = image.img_to_array(img)/255
x = np.expand_dims(x, axis=0)
classes = iv3model.predict(x)
print (classes)
# predicting images
#image = image.convert("RGB")
result = np.argmax(classes)
dic=dict(training_set.class_indices)
print([key for key in dic.keys()][result])
```

```
[[1.4443330e-06 2.0371537e-05 9.9997389e-01 7.8946465e-07 2.5416443e-06
  3.7105227e-08 2.2189280e-08 3.7500463e-08 1.5463313e-11 8.8439037e-07]]
Tomato___Late_blight
```