CROP DISEASE PREDICTION USING MACHINE LEARNING

A Minor Project Report

in partial fulfillment of the degree

Bachelor of Technology

in

Computer Science & Artificial Intelligence

By

2203A51L15	G.GOVANSH
2203A51L42	N.NAMITHA
2203A51L56	V.SRILEKHA
2203A51L59	T.THRYLOKYA
2203A51L82	B.SUSHMITHA

Under the Guidance of

Dr. RATNESH RANJAN

Submitted to

SCHOOL OF COMPUTER SCIENCE & ARTIFICIAL INTELLIGENCE

SR UNIVERSITY, ANANTHASAGAR, WARANGAL

April, 2024.

SCHOOL OF COMPUTER SCIENCE & ARTIFICIAL INTELLIGENCE

CERTIFICATE

This is to certify that this project entitled **“CROP DISEASE PREDICTION USING MACHINE LEARNING**" is the bonafied work carried out by **GOVANSH, NAMITHA, THRYLOKYA, SRILEKHA, SUSHMITHA** as a Minor Project for the partial fulfillment to award the degree **BACHELOR OF TECHNOLOGY** in **COMPUTER SCIENCE & ARTIFICIAL INTELLIGENCE** during the academic year 2022-2023 under our guidance and Supervision.

Dr. Ratnesh ranjan
Designation,
SR University,
Ananthasagar, Warangal.

Dr. M.Sheshikala
Assoc. Prof. & HOD (CSE),
SRUniversity,
Ananthasagar, Warangal.

External Examiner

ACKNOWLEDGEMENT

We owe an enormous debt of gratitude to our project guide Dr. ratnesh ranjan, Assistant Professor as well as Head of the CSE Department Dr. M.Sheshikala, Associate Professor for guiding us from the beginning through the end of the Minor Project with their intellectual advices and insightful suggestions. We truly value their consistent feedback on our progress, which was always constructive and encouraging and ultimately drove us to the right direction. We express our thanks to the project co-ordinators Dr. P Praveen, Assoc. Prof for their encouragement and support. We wish to take this opportunity to express our sincere gratitude and deep sense of respect to our beloved Dean, Dr.Indrajeet Gupta,Professor for his continuous support and guidance to complete this project in the institute.Finally , we express our thanks to all the teaching and non-teaching staff of the department for their suggestions and timely support.

GOVANSH

NAMITHA

SRILEKHA

THRYLOKYA

SUSHMITHA

ABSTRACT

Plant diseases affect the growth of their respective species, therefore their early identification is very important. Many Machine Learning (ML) models have been employed for the detection and classification of plant diseases but, after the advancements in a subset of ML, that is, Deep Learning (DL), this area of research appears to have great potential in terms of increased accuracy. Many developed/modified DL architectures are implemented along with several visualization techniques to detect and classify the symptoms of plant diseases. Moreover,several performance metrics are used for the evaluation of these architectures/techniques. This review provides a comprehensive explanation of DL models used to visualize various plant diseases. In addition, some research gaps are identified from which to obtain greater transparency for detecting diseases in plants, even before their symptoms appear clearly.

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 Colabis 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.

CONTENTS

INTRODUCTION

Plants are an integral part of our lives because they produce food and shield us from dangerous radiation. Without plants, no life is imaginable; they sustain all terrestrial life and defend the ozone layer, which filters ultraviolet radiations. Tomato is a food-rich plant, a consumable vegetable widely cultivated. Worldwide, there are approximately 160 million tons of tomatoes consumed annually. The tomato, a significant contributor to reducing poverty, is seen as an income source for farm households. Tomatoes are one of the most nutrient-dense crops on the planet, and their cultivation and production have a significant impact on the agricultural economy. Not only is the tomato nutrient-dense, but it also possesses pharmacological properties that protect against diseases such as hypertension, hepatitis, and gingival bleeding. Tomato demand is also increasing as a result of its widespread use. According to statistics, small farmers produce more than 80% of agricultural output; due to diseases and pests, about 50% of their crops are lost. The diseases and parasiticinsects are the key factors impacting tomato growth, making it necessary to research the field crop disease diagnosis.

The manual identification of pests and pathogens is inefficient and expensive. Therefore, it is necessary to provide automated AI image-based solutions to farmers. Images are being used and accepted as a reliable means of identifying disease in image-based computer vision applications due to the availability of appropriate software packages or tools. They process images using image processing, an intelligent image identification technology which increases image recognition efficiency, lowers costs, and improves recognition accuracy.

Although plants are necessary for existence, they experience numerous obstacles. An early and accurate diagnosis helps decrease the risk of ecological damage. Without systematic disease identification, product quality and quantity suffer. This has a further detrimental effect on a country’s economy. Agricultural production must expand by 70% by 2050 to meetglobal food demands, according to the United Nations Food and Agriculture Organization (FAO). In opposition, chemicals used to prevent diseases, such as fungicides and bactericides, negatively impact the agricultural ecosystem. We therefore need quick and effective disease classification and detection techniques that can help the agro-ecosystem.
Advance disease detection technology, such as image processing and neural networks, will allow the design of systems capable of early disease detection for tomato plants. The plant

production can be reduced by 50% due to stress as a result. Inspecting the plant is the first step in finding disease, then figuring out what to work with based on prior experience is the next step. This method lacks scientific consistency because farmers' backgrounds differ, resulting in the process being less reliable. There is a possibility that farmers will misclassify a disease, and an incorrect treatment will damage the plant. Similarly, field visits by domain specialists are pricey. There is a need for the development of automated disease detection andclassification methods based on images that can take the role of the domain expert.

It is necessary to tackle the leaf disease issue with an appropriate solution. Tomato disease control is a complex process that takes constant account of a substantial fraction of production cost during the season. Vegetable diseases (bacteria, late mildew, leaf spot, tomato mosaic, and yellow curved) are prevalent. They seriously affect plant growth, which leads to reduced product quality and quantity. As per past research, 80–90% of diseases of plants appear on leaves. Tracking the farm and recognizing different forms of the disease with infected plants takes a long time. Farmers' evaluation of the type of plant disease might be wrong. This decision could lead to insufficient and counterproductive defence measures implemented in the plant. Early detection can reduce processing costs, reduce the environmental impact of chemical inputs, and minimize loss risk.

Many solutions have been proposed with the advent of technology. Here in this paper, the same solutions are used to recognize leaf diseases. Compared with other image regions, the main objective is to make the lesion more apparent. Problems such as (1) shifts in illumination and spectral reflectance, (2) poor input image contrast, and (3) image size and form range have been encountered. Pre-processing operations include image contrast, greyscale conversion, image resizing, and image cropping and filtering. The next step is the division of an image into objects. These objects are used to determine regions of interest as infected regions in the image. Unfortunately, the segmentation method has many problems:

- When the conditions of light differ from eligible photographs, colour segmentation fails.
- Regional segmentation occurs because of initial seed selection.
- Texture varieties take too long to handle.

The next step for classification is to determine which class belongs to the sample. Then, one or more different input variables of the procedure are surveyed. Occasionally, the methodis employed to identify a particular type of input. Improving the accuracy of the classificationis by far the most extreme classification challenge. Finally, the actual data are used to create and validate datasets dissimilar to the training set.

Tomatoes (biological name: Solanum lycopersicum) grows on mostly any well drained soil and Nine out ofn10 farmers grow tomatoes in their field. Many gardeners also grow tomatoes in their garden to use fresh grown tomato in their kitchens and get a good taste offood. However, farmers and gardeners are sometimes unable to get proper progress of the plant growth. The tomatoes may not sometime appear on plant or sometimes the tomatoes may get bad looking and disease-full black spots at bottom part.

The identification of tomato plant disease may start from, to diagnose the portion having infection in plant then to note the differences such as brown or black patches and holes on the plant and then to look for the insects also.Tomatoes and similar vegetables like potatoes or brinjal must not be planted on same farm for more than one time inperiod of three years. To maintain the fertility of soil we should ideally precede tomato planting by any member of grass family e.g. wheat, corn, rice, sugarcane etc.The 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 Bacterialwilt. This bacterium can survive in soil for long time period and enter roots through natural wounds made duringsecondary roots emergence or man made during cultivating or transplanting or even insects.High moisture and high temperature favours 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 yellowishmaterial coming out of it.

In the research article, we have proposed a novel method to identify the disease in tomato crop after analysing the images of leaves. The work will solve farmers problems of plant’s disease identification without running after plant scientists. It will thus help them cure the plant’s disease in timely fashion and will thus increase both quality and quantity of food crops produce and therefore help in increasing farmer’s profit. For the experiment purpose we have downloaded tomato leaves dataset from plant village. After downloading the dataset, we have developed a Convolution Neural Network model to classify the images. The performance of model has been analysed based on various parameter such as training accuracy, validation accuracy and testing accuracy and number of trainable and number trainable parameters with respect to pre-trained model

i. Existing System:

Plant Village:

Description: Plant Village is an online platform that utilizes machine learning for plant disease identification. It allows users to upload images of plant leaves and provides automated diagnosis of potential diseases. The system covers a wide range of crops and diseases.

Website: Plant Village

ii. Proposed Systems:

Key Features:

Image-Based Prediction: Implement machine learning models, possibly convolutional neural networks (CNNs), for image-based crop disease prediction. Enable the system to analyseimages of crops and identify symptoms associated with diseases.

Early Detection: Incorporate algorithms for early detection of diseases before visible symptoms appear. Leverage historical data and real-time monitoring to enhance early detection capabilities

Machine Learning Models: Identify the machine learning models chosen for crop disease prediction.

Training and Evaluation: Explain the process of model training and the evaluation metrics used to assess performance. The steps taken to validate and fine-tune the models for optimal results.

iii. Software Requirements:

1. Programming Language: Python: Widely used in the machine learning community with extensive libraries (e.g., TensorFlow, PyTorch, scikit-learn).

2. Machine Learning Algorithms:

- **TensorFlow**: Developed by Google, suitable for deep learning tasks.
- **PyTorch**: Known for its flexibility and dynamic computation graph.
- **scikit-learn**: A versatile machine learning library for various algorithms.

3. Data Processing and Analysis:

- **Pandas**: Data manipulation and analysis library in Python.
- **NumPy**: Efficient numerical operations for handling data arrays.

4. Image Processing:

- **OpenCV:** A computer vision library for image processing tasks.

5. Development Environment:

- **Jupyter Notebooks:** A popular choice for interactive data analysis**.**
- **Visual Studio Code**
- **PyCharm,** or any **IDE** of your preference.

6. Visualization Tools:

- **Matplotlib, Seaborn:** Libraries for creating static visualizations.
- **Plotly:** Interactive plotting library.

7. Testing Framework:

- **Pytest:** A testing framework for Python.

iv. Hardware Requirements:

Development and Training:

1. **Central Processing Unit (CPU):** A multi-core CPU is suitable for small to medium-sized datasets. For larger datasets and complex models, consider CPUs with higher core counts.

2. **Graphics Processing Unit (GPU**): GPUs significantly accelerate training times for deep learning models. High-performance GPUs, such as NVIDIA GeForce, Quadro, or Tesla series, are commonly used for ML tasks.

3. **Random Access Memory (RAM):** A sufficient amount of RAM is crucial for loading and processing large datasets.

Minimum: 16 GB; **Recommended**: 32 GB or higher for larger datasets.

Storage:

Solid State Drive (SSD): SSDs provide faster data access and are recommended, especially for datasets used during training. SSDs are beneficial for quick model loading and inference during deployment.

Network:

Network Interface Card (NIC): A high-speed NIC is essential for efficient data transfer, especially in distributed training or cloud deployments.

LITERATURE SURVEY

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 networkssignificantly increase the image classification accuracy. This section provides a various deep learning technique used by researchers in plant disease identification. We used AlexNet to trainclassify plant diseases that were not seen before. Model accuracy was substantially reduced while testing image conditions are different than training image. Sometimes, disease appearson upper sides of the leaves sometime, lower sides of the leaves. We trained both AlexNet and VGG16net with minimum batch size, weight and bias learning rate as hyper-parameters. Accuracy is negatively correlated with minimum batch size in case of VGG16net. Convolution and pooling layers together stacked in a module and applied to GoogLeNet architecture as Inception V4 for dimension reduction. We applied weights that are pre-trained on ImageNet to this architecture average pooling layer of 8 × 8 for fine tuning. In addition to that DenseNets with 122 layers is also fined tuned for plant disease recognition.

Caffe framework is used to develop a CNN with local response normalization for eight class classification. A CNN with local contrast normalization layer is designed for binary classification with ReLu as activation function. AlexNet and GoogleNet are trained and fine- tuned for classification disease regions and symptoms.We proposed 3 stage training CNN. In first stage, it learns the presence of lesions while in second stage it produces heat map to identify infection. Finally, features learned from previous stages are classified based on heat maps. We introduced saliency map method for localization of infected regions. This type of visualization improves classification accuracy. We identified the impact of depth of networkon classification accuracy. Even with transfer learning, high classification accuracy is achieved with low number of convolutional layers. We employed variable momentum rule to CNN for parameter learning from lesions images; that results in quick convergence with comparative good accuracy. Yamamoto et al retrieved detailed images by applying super resolution method over low-resolution method and thus achieved better classification accuracy. Performance of various CNN for plant disease identification depends on various factors: availability of limited number of annotated; poor representation of disease symptoms, image background and capturing conditions; limited variations in disease symptoms

2.1 A Pre-Trained Network Model for Detecting and Classifying Disease.

A pre-trained network model for detecting and classifying tomato disease has been proposed with 94–95% accuracy. The Tree Classification Model and Segmentation is used to detect and classify six different types of tomato leaf disease with a dataset of 300 images. A technique has been proposed to detect and classify plant leaf disease with an accuracy of 93.75%. The image processing technology and classification algorithm detect and classify plant leaf disease with better quality. Here, an 8-mega-pixel smartphone camera is used to collect sample data and divides it into 50% healthy and 50% unhealthy categories. The image processing procedure includes three elements: improving contrast, segmenting, and extracting features.Classification processes are performed via an artificial neural network using a multi-layer feed- forward neural network, and two types of network structures are compared. The result was better than the Multilayer Perceptron (MLP) network and Radial Basis Function (RBF) network results. The quest divides the plant blade's picture into healthy and unhealthy; it cannotdetect the form of the disease. Authors used leaf diseases to identify and achieve 87.2% classification accuracy through color space analysis, color time, histogram, and color coherence.

AlexNet and VGG 19 models have been used to diagnose diseases affecting tomato crops using a frame size of 13,262. The model is used to achieve 97.49% precision. A transfer learning and CNN Model is used to accurately detect diseases infecting dairy crops, reaching 95%. A neural network to determine and classify tomato plant leaf conditions using transfer learning as an AlexNet based deep learning mechanism achieved an accuracy of 95.75%. Resnet-50 was designed to identify 1000 diseases of tomato leaf, i.e., a total of 3000 pictures with the name of lesion blight, late blight, and the yellow curl leaf. The Network Activation function for comparison has been amended to Leaky-ReLU, and the kernel size has been updated to 11 × 11 for the first convolution layer. The model predicts the class of diseases with an accuracy of 98.30% and a precision of 98.0% after several repetitions. A simplified eight- layered CNN model has been proposed to detect and classify tomato leaf disease. In this paper, the author utilized the Plant Village dataset that contains different crop datasets. The tomato leaf dataset was selected and used to perform deep learning; the author used the disease classes and achieved a better accuracy rate.

A simple CNN model with eight hidden layers has been used to identify the conditions of a tomato plant. The proposed techniques show optimal results compared to other classical models. The image processing technique uses deep learning methods to identify and classify tomato plant diseases. Here, the author used the segmentation technique and CNN to implement a complete system. A variation in the CNN model has been adopted and applied toachieve better accuracy.

LeNet has been used to identify and classify tomato diseases with minimal resource utilization in CPU processing capability. Furthermore, the automatic feature extraction technique has been employed to improve classification accuracy. ResNet 50 model has been used to classify and identify tomato disease. The authors detected the diseases in multiple steps: Firstly, by segregating the disease dataset. Secondly, by adapting and adjusting the model based on the transfer learning approach, and lastly, by enhancing the quality of the model by using data augmentation. Finally, the model is authenticated by using the dataset. The model outperformed various legacy methods and achieved 97% accuracy. Hyperspectral images identify rice leaf diseases by evaluating different spectral responses of leaf blade fractions and identifying Sheath blight (ShB) leaf diseases. A spectral library has been created using different disease samples. An improved VGG16 has been used to identify appleleaf disease with an accuracy rate of 99.01%.

The author employed image processing, segmentation, and a CNN to classify leaf disease. This research attempts to identify and classify tomato diseases in fields and greenhouse plants. The author used deep learning and a robot in real-time to identify plant diseases utilizing the sensor’s image. AlexNet and SqueezeNet are deep learning architectures used to diagnose and categorize plant disease. The authors built convolutional neural network modelsusing leaf pictures of healthy and sick plants. An open-source Plant Village dataset with 87,848 images of 25 plants classified into 58 categories and a model was used to identify plant/disease pairs with a 99.53% success rate (or healthy plant). The authors suggest constructing a real-time plant disease diagnosis system based on the proposed model.

In this paper, the authors reviewed all CNN variants for plant disease classification. The authors also briefed all deep learning principles used for leaf disease identification and classification. The authors mainly focused on the latest CNN models and evaluated their performance. Here, the authors summarized CNN variants such as VGG16, VGG19, and ResNet. In this paper, the authors discuss pros, cons, and future aspects of different CNN variants.

This work is mainly focused on investigating an optimal solution for plant leaf disease detection. This paper proposes a segmentation-based CNN to provide the best solution to the defined problem. This paper uses segmented images to train the model compared to other models trained on the complete image. The model outperformed and achieved 98.6% classification accuracy. The model was trained and tested on independent data with ten disease classes.

2.2 A Detailed Learning Technique for the Identification of Disease Using Enhanced CNNs

The dataset for tomato leaves is built using data augmentation and image annotation tools. It consists of laboratory photos and detailed images captured in actual field situations.

1. The recognition of tomato leaves is proposed using a Deep Convolutional Neural Network (DCNN). Rainbow concatenation and GoogLeNet Inception V3 structure are all included.
2. In the proposed INAR-SSD model, the Inception V3 module and Rainbow concatenation detect these five frequent tomato leaf diseases.

The testing results show that the INAR-SSD model achieves a detection rate of 23.13 frames per second and detection performance of 78.80% mAP on the Apple Leaf Disease Dataset (ALDD). Furthermore, the results indicate that the innovative INAR-SSD (SSD withInception module and Rainbow concatenation) model produces more accurate and faster results for the early identification of tomato leaf diseases than other methods.

An EfficientNet, a convolutional neural network with 18,161 plain segmented tomato leaf images, is used to classify tomato diseases. Two leaf segmentation models, U-net and Modified U-net, are evaluated. The models' ability was examined categorically (healthy vs disease leaves and 6- and 10-class healthy vs sick leaves). The improved U-net segmentationmodel correctly classified 98.66% of leaf pictures for segmentation. EfficientNet-B7 surpassed 99.95% and 99.12% accuracy for binary and six-class classification, and EfficientNet-B4 classified images for ten classes with 99.89 percent accuracy.

Disease detection is crucial for crop output. Therefore, disease detection has led academics to focus on agricultural ailments. This research presents a deep convolutional neural network and an attention mechanism for analysing tomato leaf diseases. The networkstructure has attention extraction blocks and modules. As a result, it can detect a broad spectrum of diseases. The model also forecasts 99.24% accuracy in tests, network complexities, and real-time adaptability.

Convolutional Neural Networks (CNNs) have revolutionized image processing, especially deep learning methods. Over the last two years, numerous potential autonomous crop disease detection applications have emerged. These models can be used to develop an expert consultation app or a screening app. These tools may help enhance sustainable farming practices and food security. The authors looked at 19 studies that employed CNNs to identifyplant diseases and assess their overall utility.

To depict the illustrations, the authors depended on the Plant Village dataset. The authors did not evaluate the performance of the neural network topologies using typical performance metrics such as F1-score, recall, precision, etc. Instead, they assessed the model's accuracy and inference time. This article proposes a new deep neural network model and evaluates it using a variety of evaluation metrics.

DESIGN OF THE PROJECT

3.1 Convolutional Neural Network

The CNN is a neural network technology widely employed today to process or train the data in images. The matrix format of the Convolution is designed to filter the pictures. For data training, each layer is utilized in the Convolution Neural Network, including the following layers: input layer, convo layer, fully connected layer pooling layer, drop-out layerto build CNN, and ultimately linked dataset classification layer. It can map a series of calculations to the input test set in each layer. The complete architecture is shown below, anda description of the model is in Table 1.

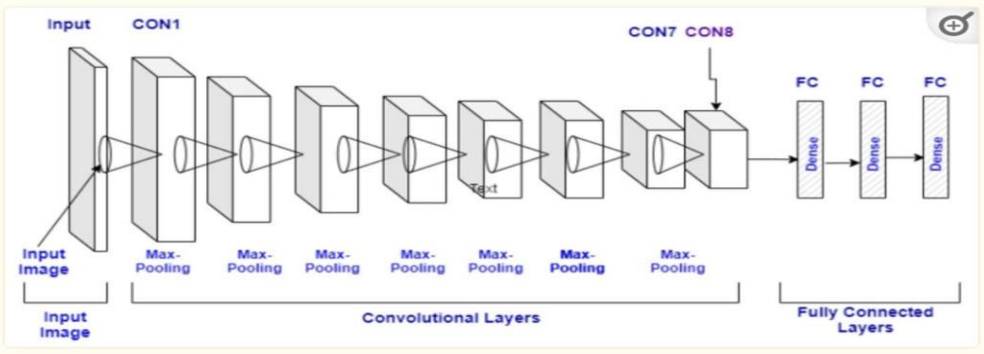


Figure 3

CNN model architecture.

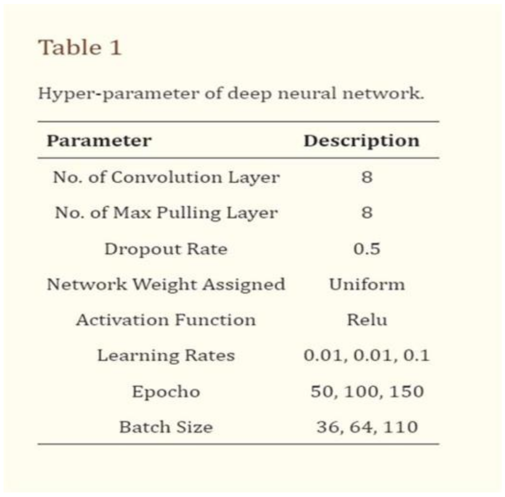


Table 1

Hyper-parameter of deep neural network.

Parameter	Description
No. of Convolution Layer	8
No. of Max Pulling Layer	8
Dropout Rate	0.5
Network Weight Assigned	Uniform
Activation Function	Relu
Learning Rates	0.01, 0.01, 0.1
Epocho	50, 100, 150
Batch Size	36, 64, 110

3.1.1. Convolutional Layer

A convolution layer is used to map characteristics using the convolution procedure with the presentation layer. Each function of the map is combined with several input characteristics. Convolution can be defined as a two-function operation and constitutes the basis of CNNs. Each filter is converted to each part of the input information, and a map or 2Dfunction map is generated. The complexity of the model encounters significant layer convolutional performance optimization. Calculated in the following equation for input z of the *i*th coalescent layer (1):

$$mi= f(Q\ i \times z) \times z \tag{1}$$

Where

$$\times$$

is a convolution operation and *f* is used for an activation function, and ***Q*** is a layer kernel convolution.

$$wi=[Qi1,\ Qi2,\ \ldots,\ QiJ]$$

,

$$J$$

is the kernel layer convolution amount. Each kernel of $\boldsymbol{Q_i}$ is a weight matrix $\boldsymbol{K} \times \boldsymbol{K} \times \boldsymbol{L}$. The number of input channels is *K* as the window size.

3.1.2. Pooling Layer

The pooling layer increases the number of parameters exponentially to maximize and improve precision. Furthermore, with growing parameters, the size of the maps is reduced. The pooling layer reduces the overall output of the convolution layer. It reduces the numberof training parameters by considering the spatial properties of an area representing a whole country. It also distributes the total value of all R activations to the subsequent activation in the chain. In the m-th max-pooled band, there are J-related filters that are combined.

$$pm=[p1,m,p2,m,\ \ldots,\ pj,m] \in Rj \tag{2}$$

pj,m=max(hj(m−1)N+r)

(3)

where N ∈ {1, …., R} is pooling shift allowing for overlap between pooling zones where N < R. It reduces the output dimensionality from K convolution bands to ***M*** = ((***K*** - ***R***))/(***N*** + 1) pooled bands and the resulting layer is ***p*** = [***p***_1, …, ***p_m***] ∈ ***R***^(***M***.***J***.)

Finally, a maximum of four quadrants indicates the value maximum with average pooling results.

3.1.3. Fully Connected Layer

Each layer in the completely connected network is connected with its previous and subsequent layers. The first layer of the utterly corresponding layer is connected to each nodein the pooling layer's last frame. The parameters used in the CNN model take more time because of the complex computation; it is the critical drawback of the fully linked sheet.
Thus, the elimination of the number of nodes and links will overcome these limitations. The dropout technique will satisfy deleted nodes and connections.

3.1.4. Dropout

An absence is an approach in which a randomly selected neuron is ignored during training, and they are "dropped out" spontaneously. This means that they are briefly omitted from their contribution to the activation of the downstream neurons on the forward transfer, and no weight changes at the back are applied to the neuron. Thus, it avoids overfitting and speeds up the process of learning. Overfitting is when most data has achieved an excellent percentage through the training process, but a difference in the prediction process occurs.
Dropout occurs when a neuron is located in the network in the hidden and visible layers.

Performance Evaluation Metrics. The accuracy, precision, recall, and F1-score measures are used to evaluate the model's performance. To avoid being misled by the confusion matrix, we applied the abovementioned evaluation criteria.

- ***Accuracy.*** Accuracy (A_{cc}) is a measure of the proportion of accurately classified predictions that have been made so far. It is calculated as follows:

Acc=TP+TNTP+TN+FP+FN.

Note that abbreviations such as "*TP*", "*TN*", "*FP*", and "*FN*" stand for "true positive", "true negative", "false positive", and "false negative", respectively.

- ***Precision***. Precision (*Pre*) is a metric that indicates the proportion of true positive outcomes. It is calculated as follows:

$$Pre=TPTP+FN$$

- ***Recall***. Recall (*Re*) is a metric that indicates the proportion of true positives that were successfully detected. It is calculated as follows:

$$Re=TPTP+FN.$$

- ***F1-Score.*** The *F1-Score* is calculated as the harmonic mean of precision and recall and is defined as follows:

$$F1\text{–}Score=Pre.RePre+Re \qquad (4)$$

Proposed Algorithm: Steps involved for Disease Detection

- **Step 1**: Input color of the image I_{RGB} of the leaf procure from the PlantVillage dataset.
- **Step 2**: Given I_{RGB}, generate the mask M_{veq} using CNN-based segmentation.
- **Step 3**: Encrust I_{RGB} with M_{veq} to get M_{mask}.
- **Step 4**: Divide the image M_{mask} into smaller regions K_{tiles} (square tiles).
- **Step 5:** Classify K_{tiles} from M_{mask} into Tomato.
- **Step 6**: Finally, K_{tiles} is the leaf part to detect disease.
- **Step 7**: Stop.

The disease detection starts with inputted image I_{RGB} from the multiclass dataset. After input image I_{RGB} is the mask segmented M_{veq} using CNN. The mask image is divided into a different region K_{tiles}. Afterward, it selects the Region of Interest (RoI), and the same is used to detect leaf disease.

The proposed algorithm for disease detection is given below:

Algorithm: Disease Detection

Input:

Take different classes of color images with disease I_{RGB} acquired from a dataset

Output: Disease detection;

```
• (a) For given input image ($I_{RGB}$), generate the masking ($M_{veg}$) using CNN-based seg mentation;
• (b) Overlay $I_{RGB}$ with $M_{veg}$ to get $M_{mask}$;
• (c) Divide the image $M_{mask}$ into smaller regions $K_{tiles}$ (square tiles);
• (d) for ($K_{tiles}$ in $M_{mask}$)
        Classify $K_{tiles}$ into $M_{mask}$ tomato diseases;
• (e) if $K_{tiles}$ is a disease, identify disease;
• (f)     end.
```

3.2. Decision Tree Model

A decision tree is a popular technique used to classify a big number of data and to extract the data with similar characteristics. The data are split into smaller subsets addition to that the tree is developed given in a final result consisting of the tree and leaf nodes.

3.3. Activation Functions

Activation functions (AFs) are mathematical functions commonly used to calculate weights and biases of a neural network, which in this study is ELM. An activation function generates outputs of ELM and patterns of the dataset. In the research, the activation functions introduced are such as and confusion matrix (CM) [25] are used to analyse the performance of the model. Classification accuracy is a method to measure the overall closeness of original labels to detected labels. To obtain a better understanding on the distribution of the accuracy across the labels, and the CM is used. It compares and shows theaccuracy of the classification of each label when a prediction is performed

1. Linear Function.
   It is straight-line function where the activation generates an exact copy of the input as the output of the weight[24], which is the weighted sum from neuron. It is defined in Eq. (5)
   $$f(x) = x \tag{5}$$
2. Sigmoid Function
   It is a non-linear function that exists in between the range of [0,1] thus is suitable in binary classification problems, which is defined as Eq. (6)
   $$f(x) = \frac{1}{1 - exp^{-x}} \tag{6}$$
3. Hyperbolic Tangent Function (TanH).
   It exists in between the range of [-1,1] which is calculated as Eq. (7)
   $$f(x) = \frac{1}{1 - exp^{-x}} \tag{7}$$
4. Rectified Linear Unit (ReLU) Function
   It is the most common used activation function in convolutional neural networks or deep learning. It exists in between the range of $[0, \infty]$, which is defined as
   $$f(x) = \begin{cases} -1, & if\ x < -1 \\ x, & if -1 \le x \le 1 \\ 1, & if\ x > 0 \end{cases} \tag{8}$$

3.4. Performance Metrics

To evaluate the performance of ELM in a plant disease classification, the classification accuracy and confusion matrix are used to analyse the performance of the model.

Classification accuracy is a method to measure the overall closeness of original labels to detected labels. To obtain a better understanding on the distribution of the accuracy acrossthe labels, and the CM is used. It compares and shows the accuracy of the classification ofeach label when a prediction is performed

IMPLEMENTATION

.4.1 Convolutional Neural Network Architectures

CNNs are a type of a deep learning network that commonly are applied on image classification tasks. In this type of network, the use of the so-called convolutional layers enables an hierarchical extraction of features, where simpler features such as edges are extracted in the first layers and more specific and complex features are extracted in deeper layers. The dimensionality of the input is decreased by the use of pooling layers. Fully connected neural networks are usually placed on top after the convolutional and pooling layers and act as classifiers using these high-level features.

Image classification has achieved great results, with various model architectures being developed over the last 10 years. Most of these deep learning models were proposed in the context of the "Large Scale Visual Recognition Challenge" (ILSVRC). These models include well-known architectures such as AlexNet, GoogLeNet, VGG, and ResNet, which have been widely used for image classification in different application domains.

4.2 Feature Extraction

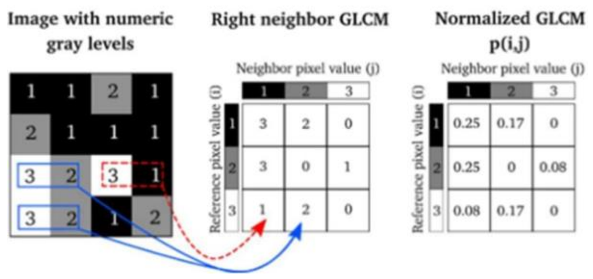
The next section involves feature extraction to reduce the features of the existing ones and then discarding the original features. This new reduced set of features are the features that will summarise most of the information contained in the original image. This process has a significant impact given when dealing with image with as high-dimensional data spaces such as RGB and HSV. For instance, if each pixel of a raw image shown in Figure 3 is to be applied as the features for the machine learning model, it can reach up to 196608 pixels per image. Haralick Texture: Texture is defined as the frequency of a pattern and colour that are visible in an image or object such diseased spots on the tomato leaves. Thus, Haralick texture is used to represent the texture of an image by using normalized co-occurrence matrices or we callgrey level co-occurrence matrices (GLCM) to obtain the distribution of grey values between pixels in a greyscale image to perform texture extraction. It is based on the matrices in the adjacent, which stores the position of the reference pixel and the neighbour pixel values (i, j). The texture is determined based on the number count of the pixel, i presence to the position next to the pixel j such as shown in Figure 4. Among the textures calculated, four of textures are addressed in this work, namely the contrast, correlation, inverse difference moments and entropy.

$$\text{Contrast} = \Sigma_i \Sigma_j (i-j)^2 P(i,j)$$

$$\text{Correlation} = \frac{\Sigma_i \Sigma_j (i-j)^2 P(i,j) - \mu_x \mu_y}{\sigma_x \sigma_y}$$

$$\text{Inverse Difference Moments} = \Sigma_i \Sigma_j \frac{P(i,j)}{1+(i-j)^2}$$

$$\text{Entrophy} = \Sigma_i \Sigma_j (i-j)^2 P(i,j) \log(P(i,j))$$



HSV Histogram: In the context of image processing, a histogram of an image refers to a histogram of the colour pixel intensity and it is applicable throughout different colour spaces such as greyscale, RGB, HSV, CMYK and etc. Each channel presented in an image is 8-bit in size, there are 256 possible intensities available to be displayed in the histogram. By simulating the rest of the channels, a three dimensional histogram can be constructed by combining the histogram of each channel such as the hue, saturation and values of the colour space. The main reason that the histogram of HSV colour space is used in this research because it abstracts the colour (hue) by separating it from saturation and pseudo-illumination (value). The space of the histogram is divided into many ranges, such as arranged as a regulargrid, each containing the same colour values.

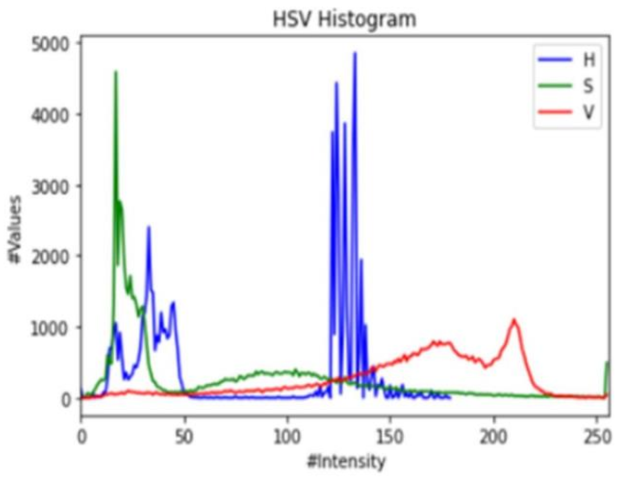


Figure 5. HSV Histogram

4.3 Code

```
import warnings
warnings.filterwarnings("ignore")
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout
from tensorflow.keras.optimizers import RMSprop
import matplotlib.pyplot as plt

# Set paths to train and validation folders
train_path = "C:/Users/sumee/Downloads/crop/tomato/train"
valid_path = "C:/Users/sumee/Downloads/crop/tomato/val"

# Define image data generators with data augmentation
image_size = (224, 224)
batch_size = 32
train_datagen = ImageDataGenerator(
    rescale=1.0/255,
    rotation_range=20,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True
)

valid_datagen = ImageDataGenerator(rescale=1.0/255)
train_generator = train_datagen.flow_from_directory(
    directory=train_path,
    target_size=image_size,
    batch_size=batch_size,
    class_mode="categorical"
)

valid_generator = valid_datagen.flow_from_directory(
    directory=valid_path,
    target_size=image_size,
    batch_size=batch_size,
    class_mode="categorical"
)

import os
import random
import matplotlib.pyplot as plt
import matplotlib.image as mpimg

def display_random_images(directory, num_images=9, num_columns=3):
    subfolders = [subfolder for subfolder in os.listdir(directory) if
os.path.isdir(os.path.join(directory, subfolder))]
    if len(subfolders) == 0:
        print(f"No subfolders found in {directory}")
        return

    random_subfolder = random.choice(subfolders)
    subfolder_path = os.path.join(directory, random_subfolder)
    image_files = [filename for filename in os.listdir(subfolder_path) if
filename.lower().endswith(('.png', '.jpg', '.jpeg'))]
    if len(image_files) < num_images:
        print(f"Number of available images in {subfolder_path} is less than {num_images}")
        return

    random_files = random.sample(image_files, num_images)
    num_rows = (num_images + num_columns - 1) // num_columns
    plt.figure(figsize=(15, 10))

    for i, filename in enumerate(random_files):
        img_path = os.path.join(subfolder_path, filename)
        img = mpimg.imread(img_path)
        plt.subplot(num_rows, num_columns, i + 1)
        plt.imshow(img)
        plt.axis('off')
        plt.title(filename)

    plt.tight_layout()
    plt.show()
```

```
# Call the function with the desired directory and number of images
display_random_images(train_path, num_images=9, num_columns=3)

# Call the function with the desired directory and number of images
display_random_images(valid_path, num_images=9, num_columns=3)

# Create a CNN model
num_classes = len(train_generator.class_indices)
model = Sequential([
    Conv2D(32, (3, 3), activation="relu", input_shape=(224, 224, 3)),
    MaxPooling2D(2, 2),
    Conv2D(64, (3, 3), activation="relu"),
    MaxPooling2D(2, 2),
    Flatten(),
    Dense(128, activation="relu"),
    Dropout(0.5),
    Dense(num_classes, activation="softmax")
])

from tensorflow.keras.optimizers import Adam

# Compile the model with Adam optimizer
model.compile(
    optimizer=Adam(learning_rate=0.0001), # You can adjust the learning rate as needed
    loss="categorical_crossentropy",
    metrics=["accuracy"]
)

from tensorflow.keras.callbacks import EarlyStopping

# Define the EarlyStopping callback
early_stopping = EarlyStopping(monitor='val_accuracy', patience=5,
restore_best_weights=True)

# Train the model with EarlyStopping
history = model.fit(
    train_generator,
    steps_per_epoch=len(train_generator),
    epochs=50,
    validation_data=valid_generator,
    validation_steps=len(valid_generator),
    callbacks=[early_stopping]  # Add the EarlyStopping callback
)

# Plot training and validation accuracy
plt.plot(history.history["accuracy"], label="Training Accuracy")
plt.plot(history.history["val_accuracy"], label="Validation Accuracy")
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.title("Training and Validation Accuracy")
plt.legend()
plt.show()

# Plot training and validation loss
plt.plot(history.history["loss"], label="Training Loss")
plt.plot(history.history["val_loss"], label="Validation Loss")
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.title("Training and Validation Loss")
plt.legend()
plt.show()

# Make predictions on test data
test_loss, test_accuracy = model.evaluate(valid_generator, steps=len(valid_generator))
print(f"Test Loss: {test_loss:.4f}")
print(f"Test Accuracy: {test_accuracy:.4f}")

# Save the trained model
model.save("trained_model.h5")
print("Trained model saved as 'trained_model.h5'")

from sklearn.metrics import confusion_matrix
import numpy as np
import seaborn as sns
# Load the saved model
loaded_model = tf.keras.models.load_model("trained_model.h5")

# Initialize variables
num_samples = len(valid_generator.filenames)
batch_size = 32
num_batches = int(np.ceil(num_samples / batch_size))
all_test_labels = []
all_predicted_labels = []

# Generate predictions in batches
for _ in range(num_batches):
    batch_images, batch_labels = next(valid_generator)
    batch_predictions = loaded_model.predict(batch_images)
    batch_predicted_labels = np.argmax(batch_predictions, axis=1)

    all_test_labels.extend(np.argmax(batch_labels, axis=1))
    all_predicted_labels.extend(batch_predicted_labels)

# Generate confusion matrix
cm = confusion_matrix(all_test_labels, all_predicted_labels)
```

```
# Plot the confusion matrix
class_names = [str(i) for i in range(num_classes)] # Replace with actual class names if
available
plt.figure(figsize=(10,  8))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", xticklabels=class_names,
yticklabels=class_names)
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()
```

APPLICATIONS

5.1 Methods and Materials

This section presents the material and main classifier models used in this research. Google Colab has been used for the implementation of the current research. 3.1.Dataset The dataset of tomato leaves is obtained from the Kaggle and it is a subset of the Plant Village dataset [16]. The focused dataset contains 10 classes representing the names of the diseases where each class contains the 1000 images such as shown in Figure 1:

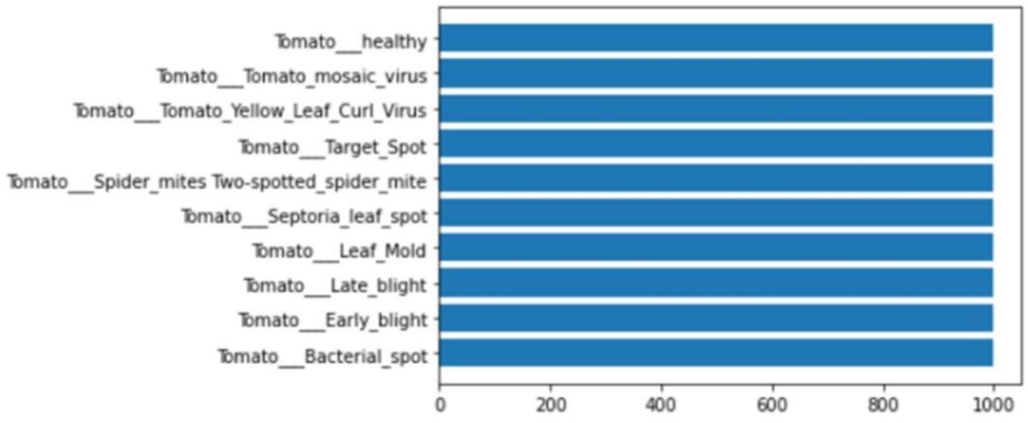


Figure 1. The distribution of classes of Tomato Leaf Diseases Dataset

5.2. Classifiers

5.2.1. Extreme Learning Machine Model

Extreme Learning Machine (ELM) model was introduced by Huang in 2006 [17]. It is a type of feedforward neural network with the simplest form [18]. The model is consists of multiple hidden nodes to form single hidden layer, where the weights between inputs and hidden nodes are randomly initialised and remain unchanged throughout training. The weights that connect the hidden nodes to the output are trained, but due to the simple structure of an ELM, these weights can be obtained by calculating a system of linear matrixequations [19]. Thus, such neural network requires no iterations which makes it faster with better generalisation performance than a back-propagation neural network. It is considered a supervised learning model because of the requirement of input nodes (features) and output nodes (labels/classes). ELM can be modelled as shown in Eq. (1):

$$\sum_{i=1}^{L} \beta_i \, g(w_i * x_j + b_i) = y_i \, , j = 1, \ldots, N \tag{1}$$

Where xj is the input, yi is the output, N is the number of samples, wi is the input weight, bi is the bias of the hidden layer and βi is the weight vector connecting the ith hidden node and output nodes. The above Eq. (1) can be written compactly as

$$T = H\beta$$

$$H = \begin{bmatrix} g(w_1 * x_1 + b_1) & \ldots & g(w_L * x_1 + b_N) \\ \vdots & & \vdots \\ g(w_1 * x_N + b_1) & \ldots & g(wL * x_N + b_N) \end{bmatrix}_{N \times L} \tag{3}$$

$$\beta = \begin{bmatrix} \beta_1^T \\ \vdots \\ \beta_L^T \end{bmatrix}_{L \times m} \quad T = \begin{bmatrix} t_1^T \\ \vdots \\ t_L^T \end{bmatrix}_{N \times m} \tag{4}$$

Where H is the hidden layer output matrix of the neural network; the ith column of H is the ith hidden node output with respect to inputs x1, x2, x3, …. xN. The β parameter is obtained by $\beta = H + T$. The Moore–Penrose generalized inverse of H is shown with H+

5.2.2. Support Vector Machine Model

A SVM model obtains a boundary or a hyperplane from features extracted to from the data to perform the classification operation. SVM is used to transform features into much higher dimensions, which makes it a good comparison to ELM that can handle large data in the same way. The SVM model is created to determine the linear discriminant function with the largest margin that separates each class of the data. The learning data that is the closest to the boundary is the support vector. This classifier provides a reliable solution in many areas such as image and object recognition, voice recognition, fingerprint recognition, and handwriting recognition .

EVALUATION

6.1 Proposed methods

The proposed algorithm for this paper was Convolutional Neural Networks to build a low- cost method to detect cassava infections through deep-learning with the implementation stepsin the sequence of dataset acquisition, labelling, training the model, testing/model evaluation using k-fold cross-validation, where k = 3 to achieve the desired accuracy.
The model building steps taken were:

- a) Environment preparation, loading and pre-processing Data – 35% time
- b) Defining Model architecture – 10% time
- c) Training the model – 50% time
- d) Estimation of performance – 5% time

The problem of this dataset was the size, very small which could lead the model to suffer from overfitting problems. Additionally, the classes were highly imbalanced being heavily biased towards CMD, CBSD classes and images were having poor resolution and low contrast. Deep learning techniques proposed in this study for addressing class imbalance to counter bias were algorithm method using class weight, focal loss in Lin et al, as a loss function for addressing extreme class imbalance and hybrid-method through resampling minority class using SMOTE (Synthetic Minority Oversampling Technique) discussed below

6.2 Experimental setup

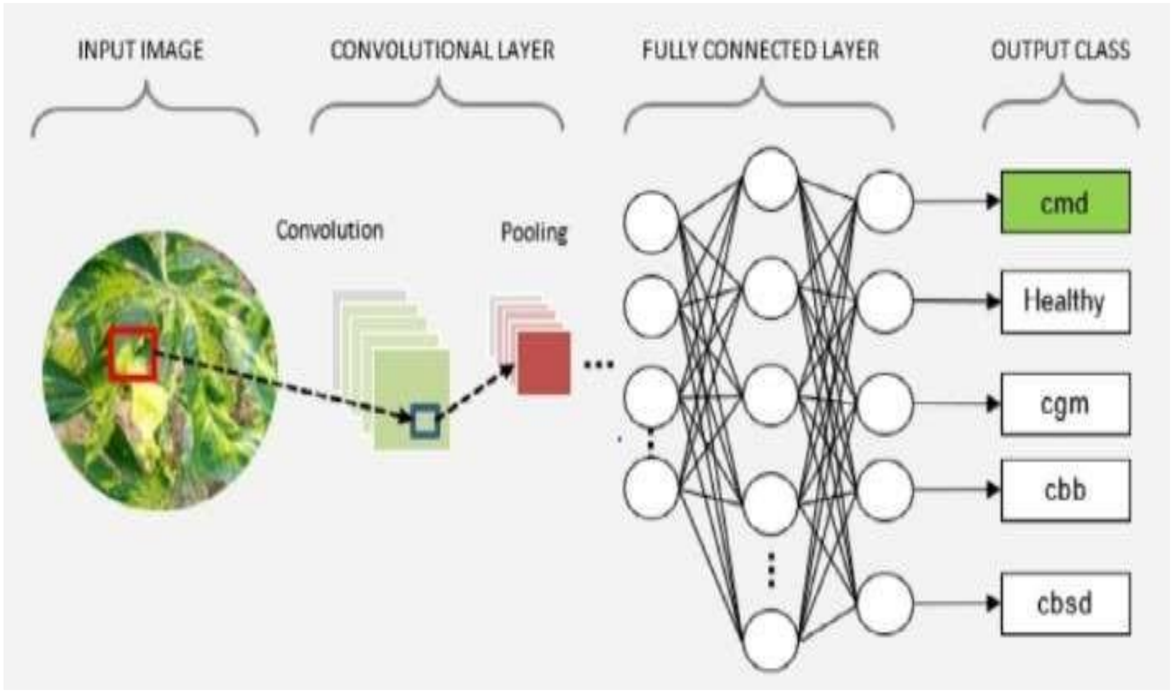
6.2.1 Dataset

Images used in the experiment were adopted from Kaggle. This dataset consisted of 5 fine-grained cassava leaf disease categories with 10,000 labelled images collected during a regular survey in Uganda, mostly crowdsourced from farmers taking images of their gardens, and annotated by experts at the National Crops Resources Research Institute in collaboration with Artificial Intelligence lab in Makerere University, Kampala. In order to be able to train the model successfully, we needed to find ways to preprocess the input images to improve contrast and secondly, we needed to counter class label skew (imbalanced in the dataset) as illustrated below

6.2.2 Model architecture

The architecture of the model used was composed of 3 convolutional layers and head of 4 fully connected layers. The first layer having 32 5 × 5 kernels to learn larger features, batch normalization and max-pooling of 3 × 3 pooled size. Second and third layers each consisted of two sets of convolutional layers each having 64 3 × 3 and 128 3 × 3 feature detectors, batch normalization, and max-pooling layers respectively. The layers were organized this way in order to allow the network to learn richer features by stacking together two sets of convolutions and batch normalization layer before max-pooling. The model architecture is

shown below. The purpose of the max-pooling layer is to apply volume spatial dimensions reduction to the input images. The head of the network consisted of 4 fully connected layers with 512 neurons in the first layer, 1024 neurons in the second and third layers and lastly 256 neurons in the fourth layer and a neuron per every category in the output layer corresponding to five different classes after parameters tuning and optimization with grid search. Dropout was used in the fully connected layers as regularizers to reduce generalization error and
over-fitting problems by encouraging the neural network to learn sparse features of raw observations which always yields good performance by empowering model's ability to generalize to new data. In general, the convolutional layers extract key features from the images and the fully connected layers focus on using the extracted features to classify imagesof cassava leaves into five different categories. Input image attributes takes an order 3 tensor,e.g., an image with H rows, W columns, and 3 channels (R, G, B color channels) on the inputlayer and a neuron per every category in the output layer corresponding to five different classes [cmd, cbsd (Bacteria Blight of Cassava), cgm (Green Mite infection), healthy and cbb]. The activation functions used in convolutional and hidden layers were ReLU (Rectifier Linear Unit) and the output layer was softmax (for a multi-class case) not sigmoid (for a binary case) function.



6.3. Image Pre-processing

This section involves image pre-processing methods that will be used to obtain optimum data from images of tomato leave. The methods involve image resizing, colour space conversion and image segmentation. After each image in the dataset are resized to 256*256- resolution in Figure 3, their colour spaces are then converted to HSV for further processing. The HSV colour space represents three elements that describe the colour (Hue), the level of colour dominance (Saturation) and the brightness level (Value). This particular colour space separates the image intensity from the colour information, thus the colour texture, the level ofdominance and the brightness of an image can be identified. Figure 3 shows the HSV scatterplot of tomato late blight disease for image segmentation and we observe that grey and black pixels were present in the saturation level of 0 to 60. Thus, by thresholding the saturation ranged from 60 and higher were performed to create a mask that remove the background of the image and retained as much of the information in the image. Next, the respective images were smoothened using the Gaussian filter to distinguish the edges. A maskis created to represent the shape of the leaf from the threshold, we created. After the closing morphological method was used to remove all the noise around the mask, a new segmented image is created by overlap the mask on the original image, as shown

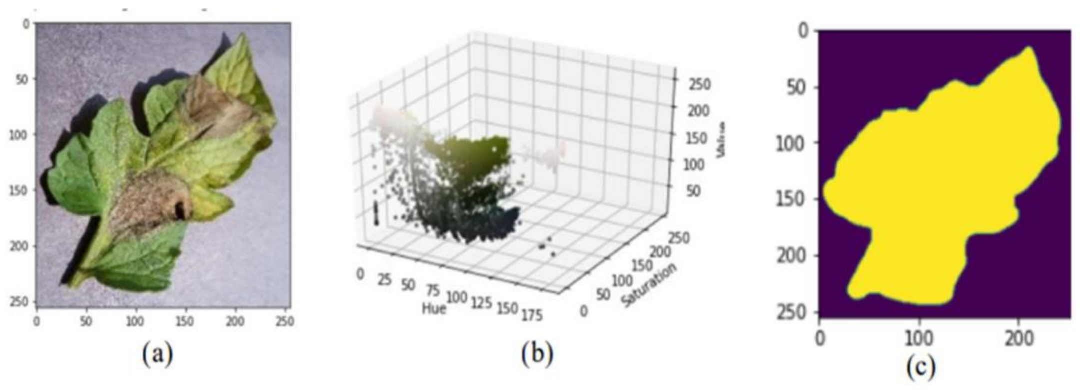


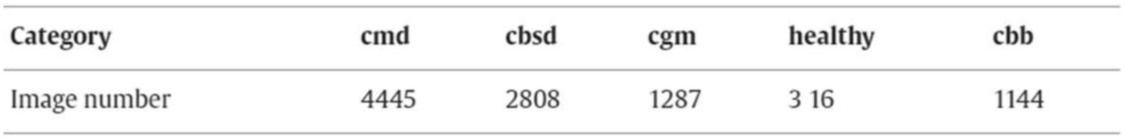
Figure 3. (a) Tomato Late Blight Image (Resized); (b) Scatter Plot of HSV from Image (a); Mask of Segmented Image (a)

TESTING

7.1 Environment Preparation, Data-Preprocessing, and Model Training

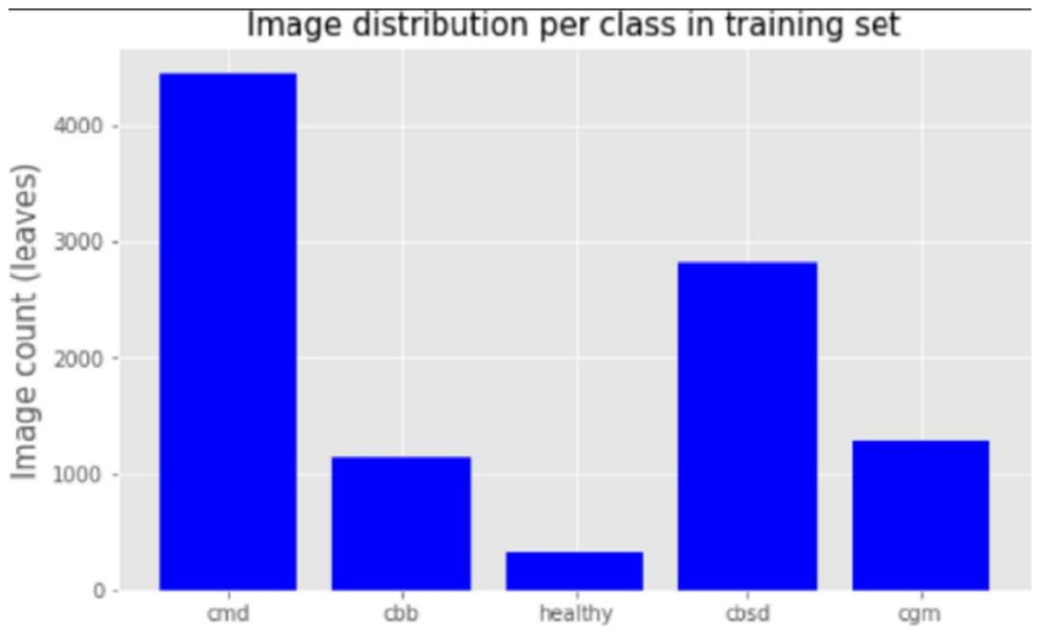
Turning the tide of this experiment started by installing 64-bits VS Code on a laptop having specifications of 16 GB RAM (Random Access Memory), 512 GB (Gigabyte) SSD (Solid State Drive) hard disk and Intel® Core™ i5-1235U CPU (Central Processing Unit) processor followed by Anaconda IDE (Integrated Development Environment) installation and all the necessary libraries including Python 3.7, Scikit-learn, Numpy, Tensorflow-2.0.0 none GPU (Graphics Processing Unit) version, Matplotlib and OpenCV (Open Computer Vision).

Raw color images of both healthy and unhealthy tomato leaves in the Joint Photographic Exper Group (JPEG) file was split into 5 directories representing each class (multi-class) other than splitting the images into healthy and unhealthy directories only (binary-class). This way, it would enable the classification of four different cassava disease categories of the high-impact yet challenging problems affecting agriculture. Unfortunately, there were a number of challenges with the dataset: The first one was, dataset being small in size, the second challenge being images having low resolution and poor contrast and the last most challenging was class label skew in that, the top class has 44.45% while the least represented class has 3.16%, revealing an order of magnitude difference as illustrated as shown below

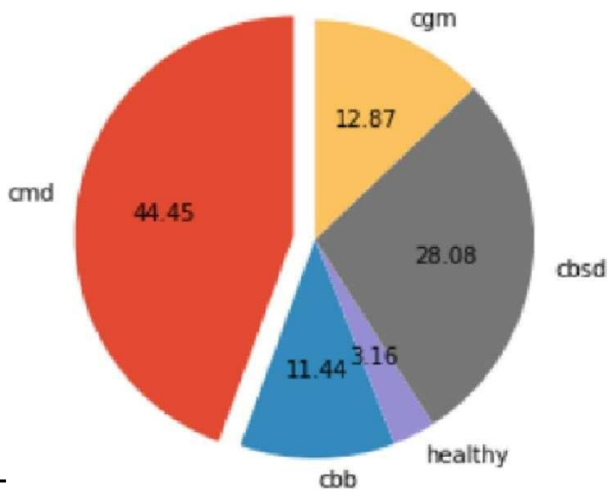


Category	cmd	cbsd	cgm	healthy	cbb
Image number	4445	2808	1287	3 16	1144

Table 1: Image number for each category of cassava leaf in training set.



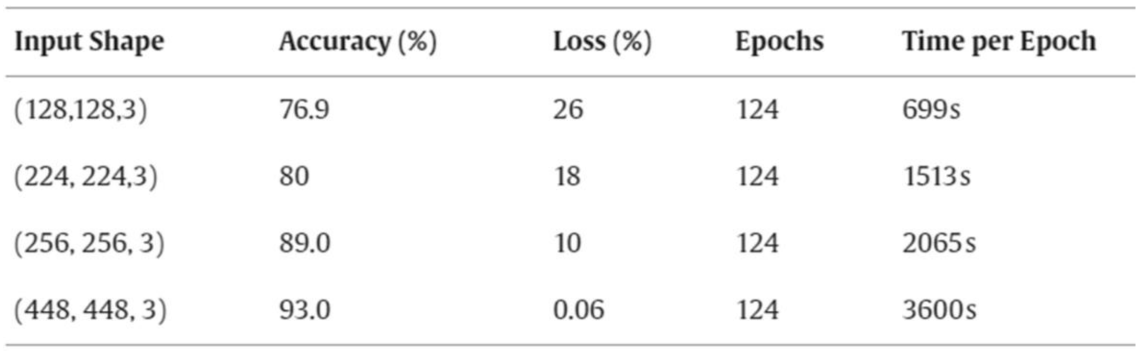
These images were highly imbalanced being heavily biased towards cmd and cbsd classes as shown above.



Pie chart showing the number of samples per class (unbalanced dataset). The top class has 44.45% while the least represented class has 3.16%, revealing an order of magnitude difference.

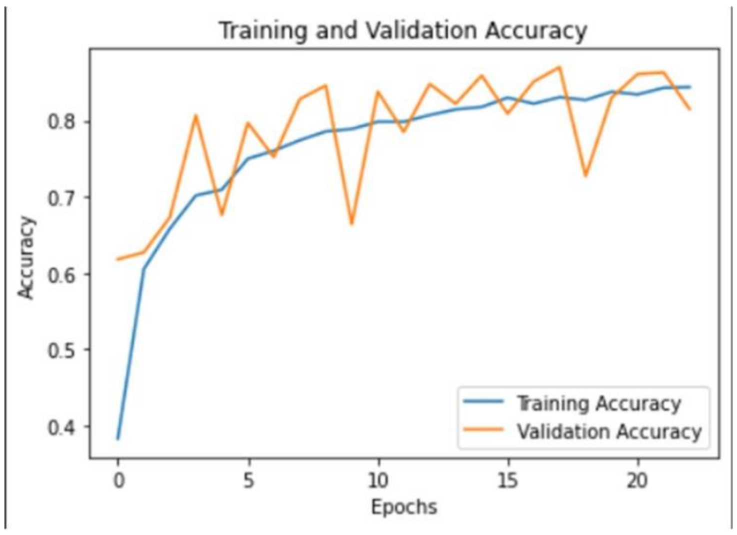
During training, there were two types of parameters: the parameters that were learned from the model and these were the weights and some other parameters that were beingtuned and these were hyper-parameters, for example, learning rates, the number of epoch, batch size, input shape, optimizer and the number of neurons in the hidden layers.

Table 2. Effect of different input image dimensions on CNN model accuracy.

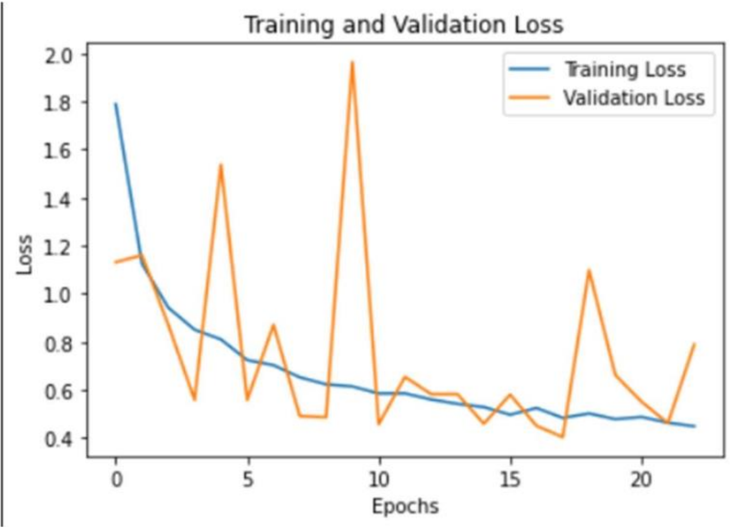


Input Shape	Accuracy (%)	Loss (%)	Epochs	Time per Epoch
(128,128,3)	76.9	26	124	699s
(224, 224,3)	80	18	124	1513s
(256, 256, 3)	89.0	10	124	2065s
(448, 448, 3)	93.0	0.06	124	3600s

In Table 2, Effect of different input image dimensions on CNN model accuracy were discussed. So, when CNN was training, it was trained with some of these hyper-parameters, however, the model was improved by finding some of the best values of these hyper-parameters through tuning techniques. One of the main hyperparameters tuned was the learning rate where we used Cyclic Learning Rate (CLR), the cycle consists of two kindsof steps; one step that increases linearly from minimum to maximum and the other that decreases linearly. The tuning was done through Learning Rate Finder (LRF) that basically tested several combinations of these values and eventually returned the best learning rate [minimum learning rate, maximum learning rate] for the model as shown below in Fig 6.We used a Cyclic Learning Rate because of the concept of super-convergence, the use of large learning rates regularizes the network which results in the reduction of all other kinds of regularization to keep a balance between overfitting and underfitting. The goal was alsoto maximum performance by minimizing the computational time required since the larger dimension image was being used. Other hyper-parameters selections and best choices were achieved through grid search with k-fold cross validation where k = 3. The reason being neural networks are difficult to configure and there are a lot of parameters that need to beset up beside, individual models can be very slow to train.



Tomato Disease Detection Accuracy

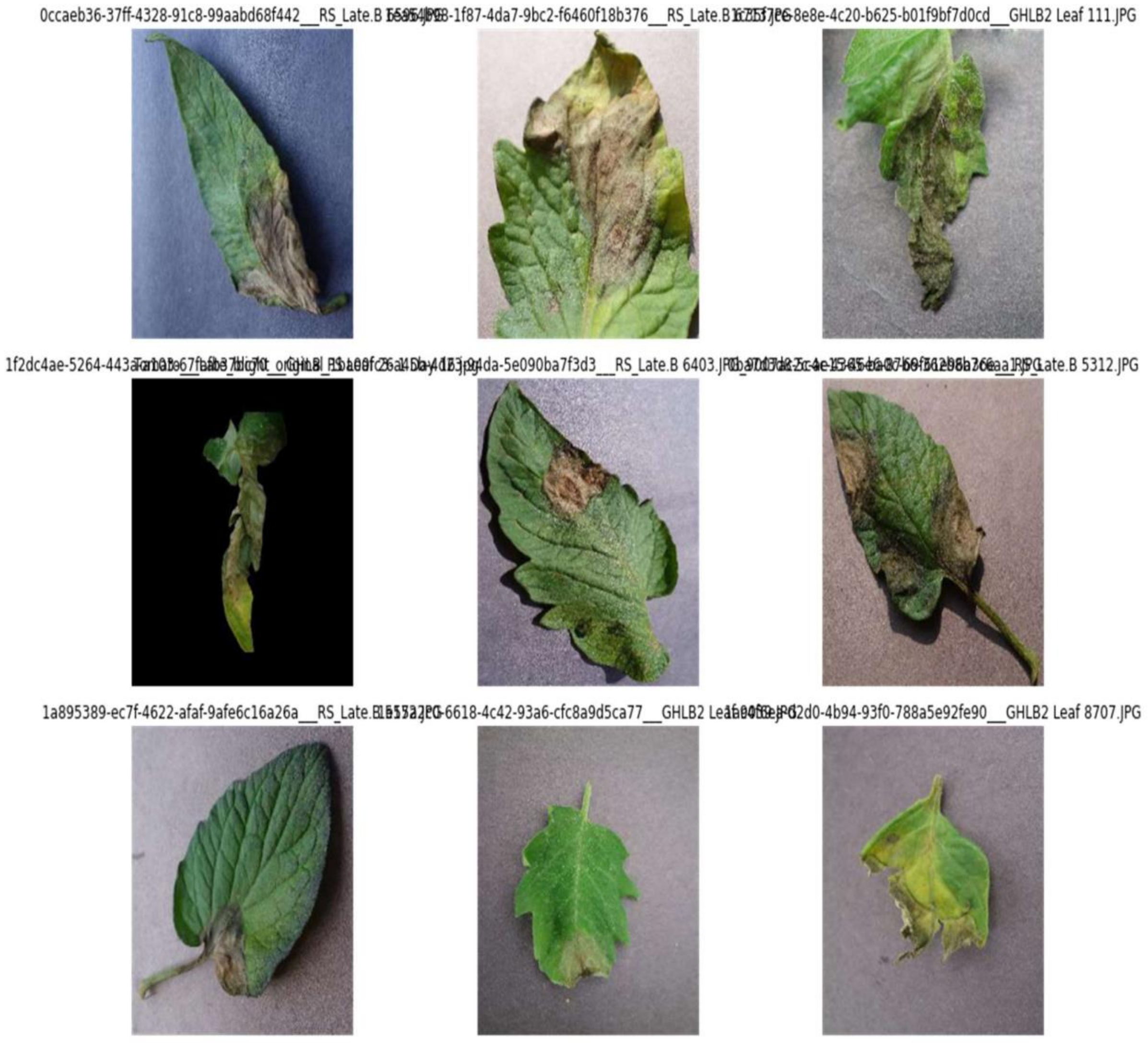


Tomato Disease Detection Loss

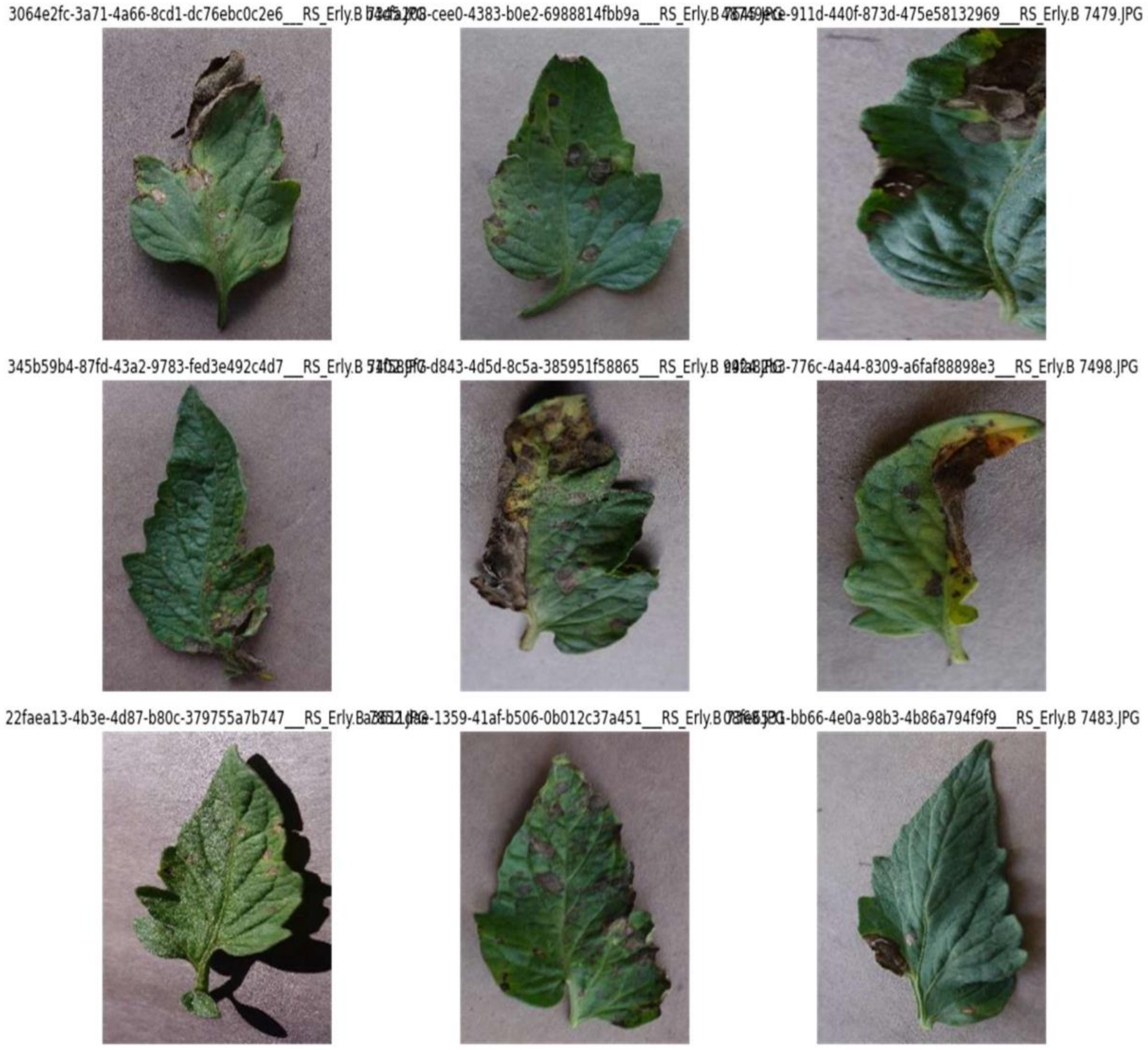
RESULTS

8.1 Experimental Images

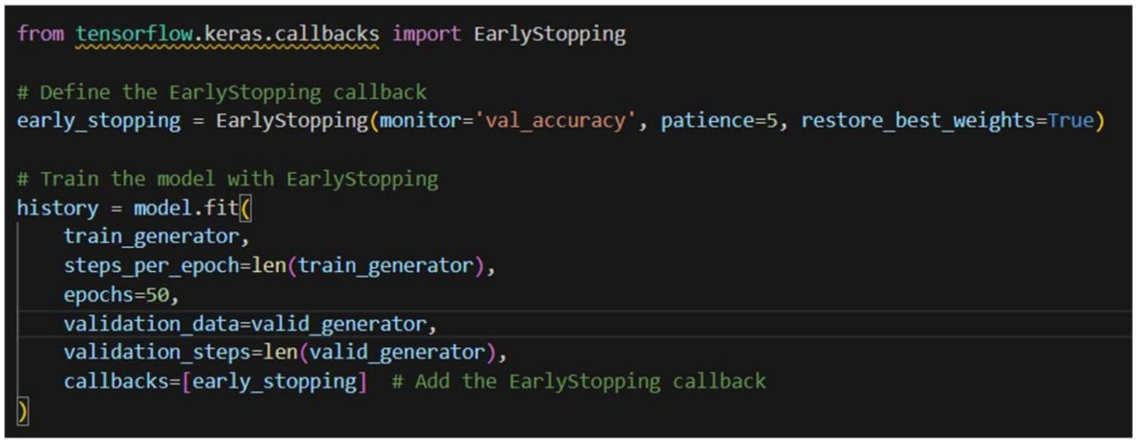
The dataset is this study includes 10 types of diseases found on tomato plants namely the bacterial spot, early blight, late blight, leaf mould, etc. These diseases are commonly observable on the surface of the leaf. The images of the bacterial spot, early blight, late blight, leaf mould are segmented and shown below. The background of the images is removed by performing a bitwise operation with the mask created from thresholding the saturation of the image. The characteristics of each leaf are retained with minimal background colour.



0ccaeb36-37ff-4328-91c8-99aabd68f442__RS_Late.B
8e8e-4c20-b625-b01f9bf7d0cd__GHLB2 Leaf 111.JPG
1f87-4da7-9bc2-f6460f18b376__RS_Late.B
1f2dc4ae-5264-443
94da-5e090ba7f3d3__RS_Late.B 6403.JPG
Late.B 5312.JPG
1a895389-ec7f-4622-afaf-9afe6c16a26a__RS_Late.B
6618-4c42-93a6-cfc8a9d5ca77__GHLB2 Leaf
2d0-4b94-93f0-788a5e92fe90__GHLB2 Leaf 8707.JPG


3064e2fc-3a71-4a66-8cd1-dc76ebc0c2e6__RS_Erly.B
-cee0-4383-b0e2-6988814fbb9a__RS_Erly.B
-911d-440f-873d-475e58132969__RS_Erly.B 7479.JPG
345b59b4-87fd-43a2-9783-fed3e492c4d7__RS_Erly.B
-d843-4d5d-8c5a-385951f58865__RS_Erly.B
-776c-4a44-8309-a6faf88898e3__RS_Erly.B 7498.JPG
22faea13-4b3e-4d87-b80c-379755a7b747__RS_Erly.B
-1359-41af-b506-0b012c37a451__RS_Erly.B
-bb66-4e0a-98b3-4b86a794f9f9__RS_Erly.B 7483.JPG

8.2 Results



```python
from tensorflow.keras.callbacks import EarlyStopping

# Define the EarlyStopping callback
early_stopping = EarlyStopping(monitor='val_accuracy', patience=5, restore_best_weights=True)

# Train the model with EarlyStopping
history = model.fit(
    train_generator,
    steps_per_epoch=len(train_generator),
    epochs=50,
    validation_data=valid_generator,
    validation_steps=len(valid_generator),
    callbacks=[early_stopping]  # Add the EarlyStopping callback
)
```

CODE FOR TRAINING

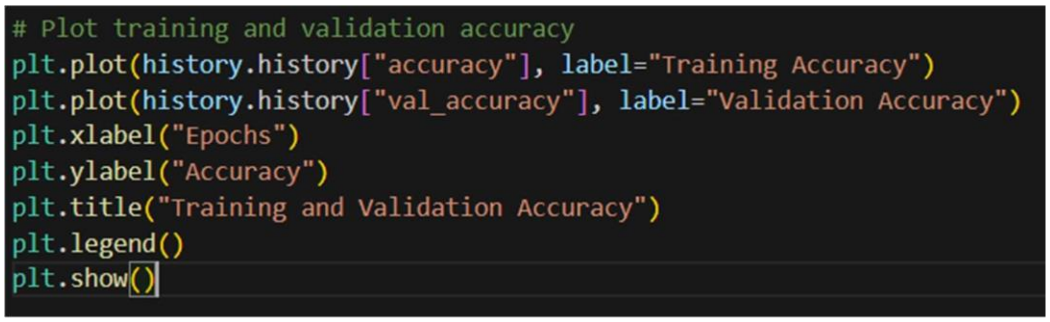
OUTPUT :

```
Epoch 1/50

313/313 [==============================] - 410s 1s/step - loss: 1.7887 -
accuracy: 0.3829 - val_loss: 1.1318 - val_accuracy: 0.6180

Epoch 2/50

313/313 [==============================] - 357s 1s/step - loss: 1.1250 -
accuracy: 0.6052 - val_loss: 1.1605 - val_accuracy: 0.6270

Epoch 3/50

313/313 [==============================] - 373s 1s/step - loss: 0.9427 -
accuracy: 0.6581 - val_loss: 0.8714 - val_accuracy: 0.6730

Epoch 4/50

313/313 [==============================] - 360s 1s/step - loss: 0.8511 -
accuracy: 0.7017 - val_loss: 0.5596 - val_accuracy: 0.8070

Epoch 5/50

313/313 [==============================] - 354s 1s/step - loss: 0.8109 -
accuracy: 0.7093 - val_loss: 1.5370 - val_accuracy: 0.6760

Epoch 6/50

313/313 [==============================] - 355s 1s/step - loss: 0.7247 -
accuracy: 0.7499 - val_loss: 0.5590 - val_accuracy: 0.7970

Epoch 7/50
```

```
313/313 [==============================] - 377s 1s/step - loss: 0.7037 -
accuracy: 0.7604 - val_loss: 0.8714 - val_accuracy: 0.7520

Epoch 8/50

313/313 [==============================] - 368s 1s/step - loss: 0.6532 -
accuracy: 0.7742 - val_loss: 0.4918 - val_accuracy: 0.8280

Epoch 9/50

313/313 [==============================] - 434s 1s/step - loss: 0.6240 -
accuracy: 0.7857 - val_loss: 0.4870 - val_accuracy: 0.8460

Epoch 10/50

313/313 [==============================] - 389s 1s/step - loss: 0.6151 -
accuracy: 0.7889 - val_loss: 1.9647 - val_accuracy: 0.6640

Epoch 11/50

313/313 [==============================] - 356s 1s/step - loss: 0.5860 -
accuracy: 0.7984 - val_loss: 0.4582 - val_accuracy: 0.8380

Epoch 12/50

313/313 [==============================] - 355s 1s/step - loss: 0.5863 -
accuracy: 0.7984 - val_loss: 0.6551 - val_accuracy: 0.7850

Epoch 13/50

313/313 [==============================] - 359s 1s/step - loss: 0.5609 -
accuracy: 0.8072 - val_loss: 0.5828 - val_accuracy: 0.8480

Epoch 14/50

313/313 [==============================] - 359s 1s/step - loss: 0.5427 -
accuracy: 0.8146 - val_loss: 0.5830 - val_accuracy: 0.8220

Epoch 15/50

313/313 [==============================] - 359s 1s/step - loss: 0.5295 -
accuracy: 0.8178 - val_loss: 0.4601 - val_accuracy: 0.8590

Epoch 16/50

313/313 [==============================] - 368s 1s/step - loss: 0.4978 -
accuracy: 0.8300 - val_loss: 0.5812 - val_accuracy: 0.8090

Epoch 17/50

313/313 [==============================] - 362s 1s/step - loss: 0.5255 -
accuracy: 0.8221 - val_loss: 0.4517 - val_accuracy: 0.8510
```

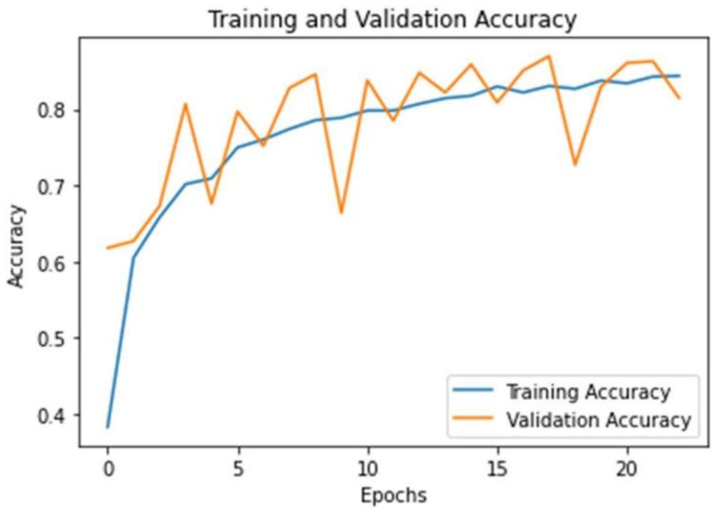
```
Epoch 18/50

313/313 [==============================] - 361s 1s/step - loss: 0.4844 -
accuracy: 0.8306 - val_loss: 0.4046 - val_accuracy: 0.8700

Epoch 19/50

313/313 [==============================] - 360s 1s/step - loss: 0.5029 -
accuracy: 0.8269 - val_loss: 1.0990 - val_accuracy: 0.7270

Epoch 20/50

313/313 [==============================] - 358s 1s/step - loss: 0.4792 -
accuracy: 0.8377 - val_loss: 0.6628 - val_accuracy: 0.8300

Epoch 21/50

313/313 [==============================] - 357s 1s/step - loss: 0.4884 -
accuracy: 0.8342 - val_loss: 0.5512 - val_accuracy: 0.8610

Epoch 22/50

313/313 [==============================] - 1583s 5s/step - loss: 0.4643 -
accuracy: 0.8429 - val_loss: 0.4624 - val_accuracy: 0.8630

Epoch 23/50

313/313 [==============================] - 377s 1s/step - loss: 0.4498 -
accuracy: 0.8439 - val_loss: 0.7899 - val_accuracy: 0.8150
```

RESULT OF CODE FOR TRAINING

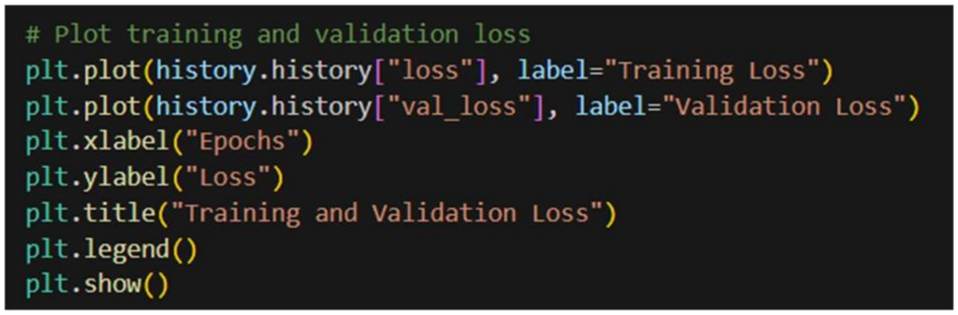


```python
# Plot training and validation accuracy
plt.plot(history.history["accuracy"], label="Training Accuracy")
plt.plot(history.history["val_accuracy"], label="Validation Accuracy")
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.title("Training and Validation Accuracy")
plt.legend()
plt.show()
```

CODE FOR VALIDATION ACCURACY

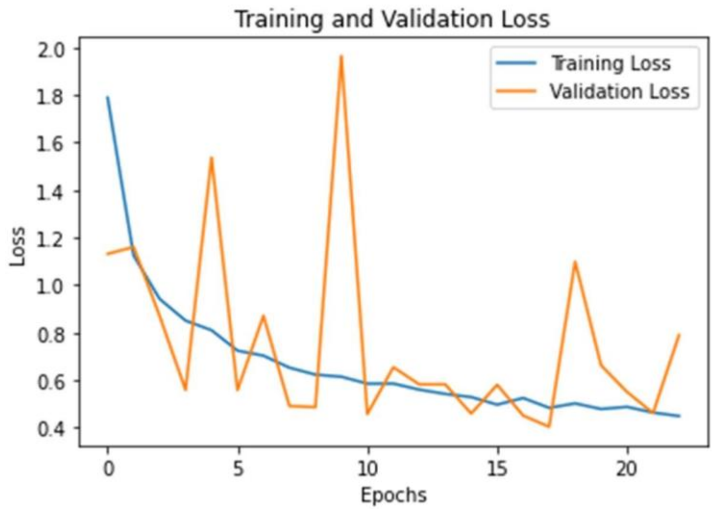


RESULT FOR VALIDATION ACCURACY

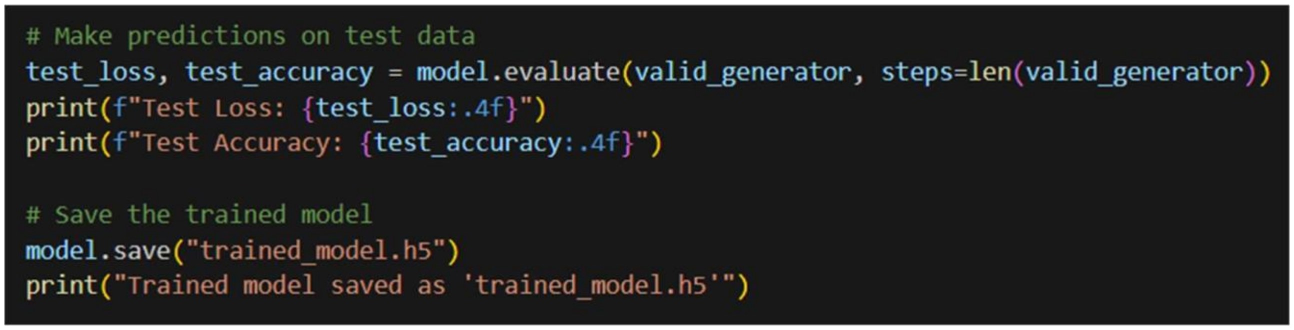


```
# Plot training and validation loss
plt.plot(history.history["loss"], label="Training Loss")
plt.plot(history.history["val_loss"], label="Validation Loss")
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.title("Training and Validation Loss")
plt.legend()
plt.show()
```

CODE FOR VALIDATION LOSS

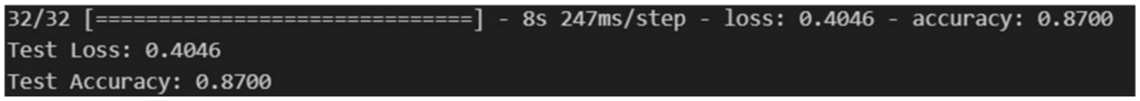


RESULT FOR VALIDATION LOSS



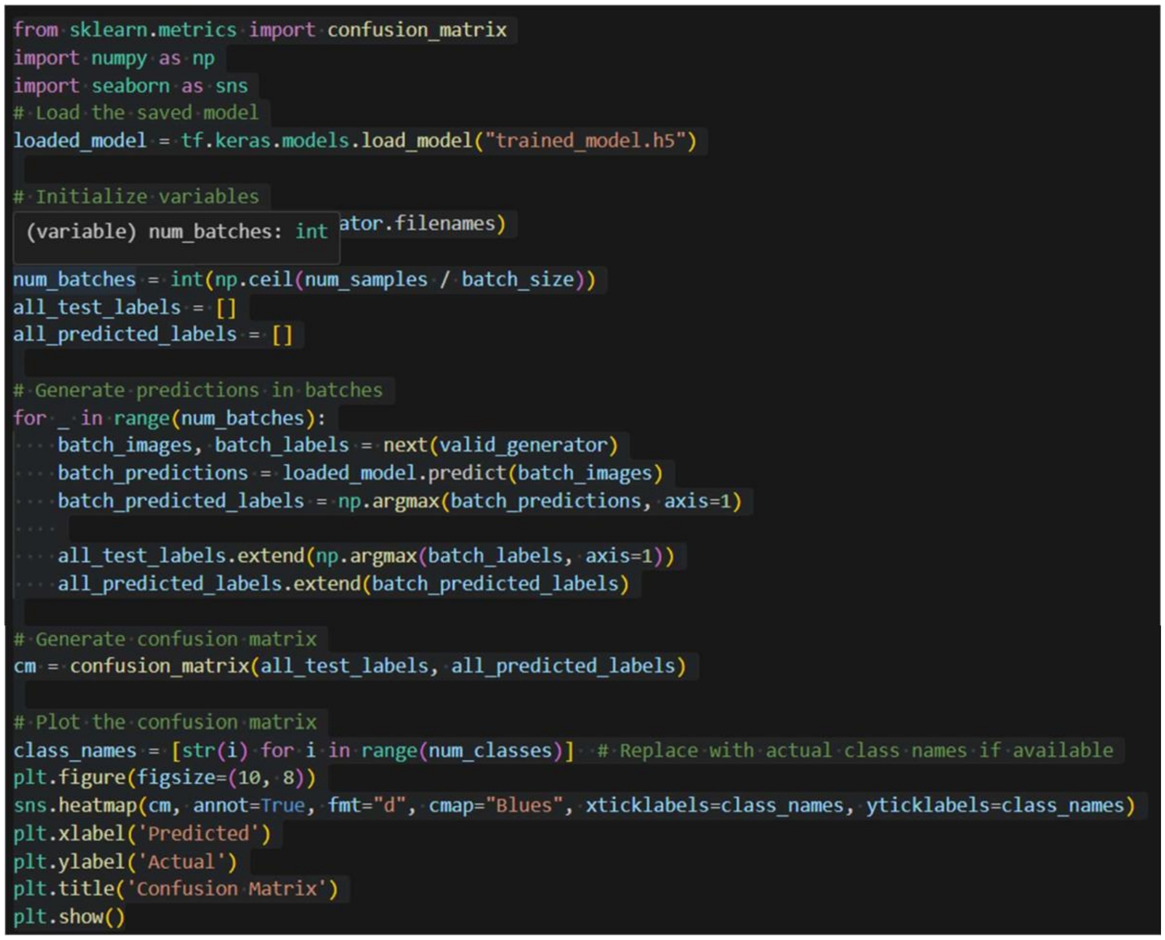
```
# Make predictions on test data
test_loss, test_accuracy = model.evaluate(valid_generator, steps=len(valid_generator))
print(f"Test Loss: {test_loss:.4f}")
print(f"Test Accuracy: {test_accuracy:.4f}")

# Save the trained model
model.save("trained_model.h5")
print("Trained model saved as 'trained_model.h5'")
```

PREDICTION ON DATA SETS



```
32/32 [==============================] - 8s 247ms/step - loss: 0.4046 - accuracy: 0.8700
Test Loss: 0.4046
Test Accuracy: 0.8700
```

RESULT FOR DATA SETS



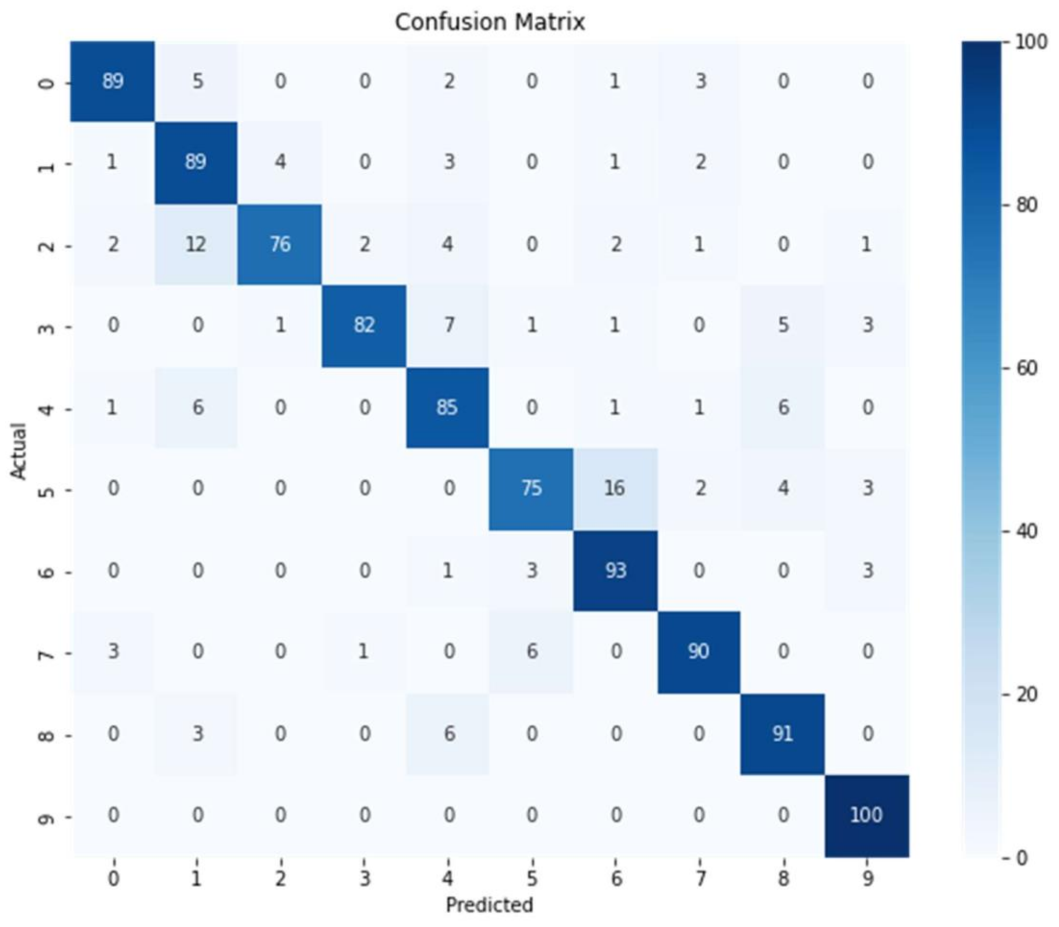
```
from sklearn.metrics import confusion_matrix
import numpy as np
import seaborn as sns
# Load the saved model
loaded_model = tf.keras.models.load_model("trained_model.h5")

# Initialize variables
 (variable) num_batches: int ator.filenames)

num_batches = int(np.ceil(num_samples / batch_size))
all_test_labels = []
all_predicted_labels = []

# Generate predictions in batches
for _ in range(num_batches):
    batch_images, batch_labels = next(valid_generator)
    batch_predictions = loaded_model.predict(batch_images)
    batch_predicted_labels = np.argmax(batch_predictions, axis=1)
    
    all_test_labels.extend(np.argmax(batch_labels, axis=1))
    all_predicted_labels.extend(batch_predicted_labels)

# Generate confusion matrix
cm = confusion_matrix(all_test_labels, all_predicted_labels)

# Plot the confusion matrix
class_names = [str(i) for i in range(num_classes)]  # Replace with actual class names if available
plt.figure(figsize=(10, 8))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", xticklabels=class_names, yticklabels=class_names)
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()
```

CODE FOR CONFUSION MATRIX

OUTPUT :

```
1/1 [==============================] - 0s 437ms/step
1/1 [==============================] - 0s 234ms/step
1/1 [==============================] - 0s 234ms/step
1/1 [==============================] - 0s 250ms/step
1/1 [==============================] - 0s 243ms/step
1/1 [==============================] - 0s 231ms/step
1/1 [==============================] - 0s 267ms/step
1/1 [==============================] - 0s 260ms/step
1/1 [==============================] - 0s 252ms/step
1/1 [==============================] - 0s 236ms/step
1/1 [==============================] - 0s 266ms/step
1/1 [==============================] - 0s 297ms/step
1/1 [==============================] - 0s 292ms/step
1/1 [==============================] - 0s 284ms/step
1/1 [==============================] - 0s 300ms/step
1/1 [==============================] - 0s 287ms/step
1/1 [==============================] - 0s 426ms/step
1/1 [==============================] - 0s 349ms/step
1/1 [==============================] - 0s 315ms/step
1/1 [==============================] - 0s 265ms/step
1/1 [==============================] - 0s 262ms/step
1/1 [==============================] - 0s 242ms/step
1/1 [==============================] - 0s 244ms/step
1/1 [==============================] - 0s 250ms/step
1/1 [==============================] - 0s 242ms/step
1/1 [==============================] - 0s 248ms/step
```

```
1/1 [==============================] - 0s 252ms/step
1/1 [==============================] - 0s 241ms/step
1/1 [==============================] - 0s 266ms/step
1/1 [==============================] - 0s 277ms/step
1/1 [==============================] - 0s 262ms/step
1/1 [==============================] - 0s 143ms/step
```



Confusion Matrix

CONCLUSION AND FUTURE SCOPE

In this survey presented an insight into existing research addressing the application of ML-based techniques for forecasting, detection, and classification of diseases and pests.

Data-sets containing weather, diseases, and pests data should keep records for long periods of time. Time-series ML models, such as CNN, can be employed to accurately forecast the occurrence of diseases and pests based on meteorological measurements series. NDVI measurements can also be helpful, since they provide additional information regarding the crop's development.

Detection and classification of pests and diseases can be performed using computer vision and deep-learning algorithms based on CNN models, which show better performance when compared with older image classification approaches based on "manual" features extraction. However, deep learning models require large amounts of data, which can be difficult to obtain. To tackle this issue, the use of transfer learning or few-shot learning methods can prove useful. Nonetheless, although the performance of deep learning-based methods is high for images acquired under controlled conditions, additional research is required regarding the analysis of images taken in the field, under real life conditions.

In the proposed work, we have 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, we 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 thatdata 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%. The storagespace needed by proposed model is of order of 1.5 MB whereas pretrained models have storage space needs of around 100 MB thus showing the benefit of the proposed model over pretrained models. As a future work, we are trying to modify the model with a greater number of images with some other crop. Moreover, we are also in process to improve the same model on same dataset as testing accuracy is less

.

BIBLIOGRAPHY

[1] Brahimi, M., Arsenovic, M., Laraba, S., Sladojevic, S., Boukhalfa, K., Moussaoui, A., 2018. Deep learning for plant diseases: Detection and saliency map visualisation, in: Human and Machine Learning. Springer, pp. 93–117.

[2] Brahimi, M., Boukhalfa, K., Moussaoui, A., 2017. Deep learning for tomato diseases: classification and symptoms visualization. Applied Artificial Intelligence 31, 299–315
.

[3] DeChant, C., Wiesner-Hanks, T., Chen, S., Stewart, E.L., Yosinski, J., Gore, M.A., Nelson, R.J., Lipson, H., 2017. Automated identification of northern leaf blight-infected maize plants from field imagery using deep learning. Phytopathology 107, 1426–1432.

[4] Fujita, E., Kawasaki, Y., Uga, H., Kagiwada, S., Iyatomi, H., 2016. Basic investigation on a robust and practical plant diagnostic system, in: 2016 15th IEEE International Conference on Machine Learning and Applications (ICMLA), IEEE. pp. 989–992.

[5] Howard, A.G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., Adam, H., 2017. Mobilenets: Efficient convolutional neural networks for mobile vision applications. arXiv preprint arXiv:1704.04861.

[6] Huang, G., Liu, Z., Van Der Maaten, L., Weinberger, K.Q., 2017. Densely connected convolutional networks, in: Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 4700–4708.

[7] Jia, Y., Shelhamer, E., Donahue, J., Karayev, S., Long, J., Girshick, R., Guadarrama, S., Darrell, T., 2014. Caffe: Convolutional architecture for fast feature embedding, in: Proceedings of the 22nd ACM international conference on Multimedia, ACM. pp. 675–678.

[8] Kawasaki, Y., Uga, H., Kagiwada, S., Iyatomi, H., 2015. Basic study of automated diagnosis of viral plant diseases using convolutional neural networks, in: International Symposium on Visual Computing, Springer. pp. 638–645.

[9] Krizhevsky, A., Sutskever, I., Hinton, G.E., 2012. Imagenet classification with deep convolutional neural networks, in: Advances in neural information processing systems, pp. 1097–1105.

[10] Mohanty, S.P., Hughes, D.P., Salathe, M., 2016. Using deep learning for image-based plant disease detection. Frontiers in plant science 7, ´ 1419