Journal of Management, Social Sciences and Humanities, 2024, Vol. 05, Issue 02, 71 – 106
DOI: https://doi.org/10.4038/jmsh.v5i2.34

Anthurium Leaf Diseases Identification Using Deep Learning

By Wickramanayaka, D.M.R.[1]

ABSTRACT

Tropical ornamental plants such as anthurium are cultivated for their attractive foliage and long-lasting flowers and have gained importance as export-quality cut flowers. Healthy growth and high flower yield depend on suitable physical and chemical conditions of the growing medium and the use of high-quality planting materials. However, anthurium cultivation is significantly affected by plant diseases, particularly due to difficulties in accurate disease identification and limited farmer awareness. Some diseases, such as bacterial infections, have no effective cure and must be detected early to prevent widespread damage.

This study presents a deep learning–based approach for identifying anthurium leaf diseases using Convolutional Neural Networks (CNNs). CNNs are powerful image-processing models capable of extracting complex features from leaf images while efficiently utilizing GPU-based computation. Pre-trained CNN models were employed to classify images of healthy anthurium leaves and two major bacterial diseases—bacterial blight and bacterial wilt.

Approximately 400 leaf images were used for training and testing the models. Model performance was evaluated by adjusting key hyperparameters, including the number of epochs, batch size, and learning rate. The optimal configuration consisted of 25 training epochs, a batch size of 32, and a learning rate of 0.001, achieving a test accuracy of 98.44%. Further evaluation using advanced deep learning architectures, namely ResNet-50 and VGG-16, resulted in classification accuracies of 100%, demonstrating the effectiveness of deeper network structures.

The final model was implemented in an Android mobile application that enables disease detection using images captured via a mobile camera or image gallery. The high accuracy and practical deployment of the system highlight its potential as a reliable tool for early disease diagnosis, supporting farmers in effective disease management and promoting healthier anthurium cultivation.

KEYWORDS: disease detection, image acquisition, pre-processing, learning rate, epoch

[1] Department of Computing and Information Systems, Wayamba University of Sri Lanka, Sri Lanka
methmirasara1997@gmail.com

INTRODUCTION

Background

Anthuriums are popular and widely grown in Asia and Europe. They are popular in Japan, the Netherlands, Thailand, and Sri Lanka. Anthurium business is huge in these countries. Anthuriums are available in an extensive array of hues and forms. There are both native and imported anthuriums that are grown in Sri Lanka. It is a popular commercial flower that comes in a range of colours and shapes. Anthuriums are grown for a variety of reasons, including cut flowers and potted plants. Local Anthurium types are mostly produced in Sri Lanka for cut flowers, which are cut and sold for commercial purposes, as well as for beauty or ornamental purposes. Anthuriums are ornamental plants whose flowers are not harvested. Foreign cultivars are mostly used as flowering plants in plastic pots. Figure 1.1 shows some examples of how diseases affect an anthurium plant.

Figure 1: A representation of the damage caused by diseases in anthurium plants

Source: By Google

They can be sold as cut flowers if done as a business. In addition, Anthuriums can be sold as huge flowering plants as well as little flowerless plants. It is a popular crop for both flowering potted plants and exotic cut flowers due to its eye-catching and durable blooms. In their commercial greenhouse conditions, growers often report two fungal infections and three bacterial diseases.

Since anthurium is highly prone to bacterial and fungal infections, commercial output may be severely constrained. The most dangerous kind is most likely Xanthomonas-induced bacterial blight. The formation of anthurium is also associated with root rots brought on by Rhizoctonia, Pythium, and Phytophthora. Farmers should have a good understanding of perceptions of crop diseases, pest practices, and pest management, as it is critical for the development of management strategies supported by government and non-governmental organizations that cater to farmers' needs and are likely to be adopted by the intended users. [1]

Image processing in agriculture has been used in a variety of applications, including sorting, grading, and detecting flaws such as dark spots, cracks, and bruising in fruits and seeds, among others. These categorization tasks were completed by utilizing semantic features like as corners, edges, forms, and so on. Image processing for disease detection entails stages such as image acquisition, image pre-processing, image segmentation, feature extraction, and classification.

Deep learning technology has made more progress in the area of plant disease identification. Deep Learning (DL) technology in the face of the user is transparent; the professional level of plant protection and statistics researchers is not high; it can automatically extract image features and classify plant disease spots, eliminating a lot of work from traditional image recognition technology of feature extraction and classifier design. CNNs, a subset of deep learning techniques, are swiftly gaining popularity. [2].

RESEARCH PROBLEM

Farmers confront significant hurdles due to a lack of understanding about diseases and incorrect disease detection. Furthermore, certain infections have no cure but must be eradicated before they spread. The techniques and means of acquiring critical support, such as from responsible parties i.e. the Department of Floriculture or Agriculture in Sri Lanka, are limited because there is no flexible framework in place to engage their resource personnel in an efficient manner.

Therefore, this study proposes to construct a detection system using the images of the Anthurium leaves of the plants with CNN techniques. The findings of this study will aid farmers in recognizing various anthurium illnesses, which might be difficult for them to identify otherwise.

OBJECTIVES OF THE PROJECT

Aim

The primary goal of this research is to propose an accurate and automated model for identifying and classifying pre-identified particular illnesses of anthurium plant leaves using photographs. Using image processing and CNN approaches, the suggested methodology would improve the accuracy of identifying anthurium leaf illnesses, resulting in a more cost-effective, precise, user-friendly, and reasonable solution for detecting those diseases.

Objectives

• To collect the images for the dataset.

• To train and evaluate the model using Open-Source Computer Vision Library (OpenCV).

• To label the images to use in You Only Look Once version 5 (YOLOv5).

• To train and evaluate the model using YOLOv5.

• To deploy the model.

• To identify the diseases using a mobile camera.

Significance

Anthurium is a commercial crop for most Sri Lankan farmers. There are several diseases that can happen to anthurium plants in roots (Rhizoctonia root rot disease) and flowers (black nose disease). Out of the diseases, mainly anthurium leaf diseases were targeted [3]. When considering leaf diseases, there are mainly two types: bacterial diseases and Miata. Here, mainly bacterial wilt, bacterial blight, and other diseases were identified, and a mobile app to detect those diseases was utilised, which is very useful because it is very difficult to identify those diseases in the enormous farms using the naked eye, and the latter is not an effective way. On the other hand, it requires expertise in plant diseases. Many farmers now lack expertise and awareness of those illnesses. Early detection of these diseases is very important because they can spread very quickly all over the country when treatments are delayed. Through early detection, there is a high possibility of preventing the disease from spreading to the cultivation and reducing harvest loss. Therefore, it is critical to seek an autonomous, precise, rapid, and less expensive technique for illness identification. This study's findings will enhance the number of correct decisions and would be an effective technique for lowering harvest losses owing to these diseases.

Figure 2: Anthurium Plants

Source: By Google

MATERIALS AND TECHNOLOGY ADOPTED

Materials

Convolutional Neural Networks (CNNs)

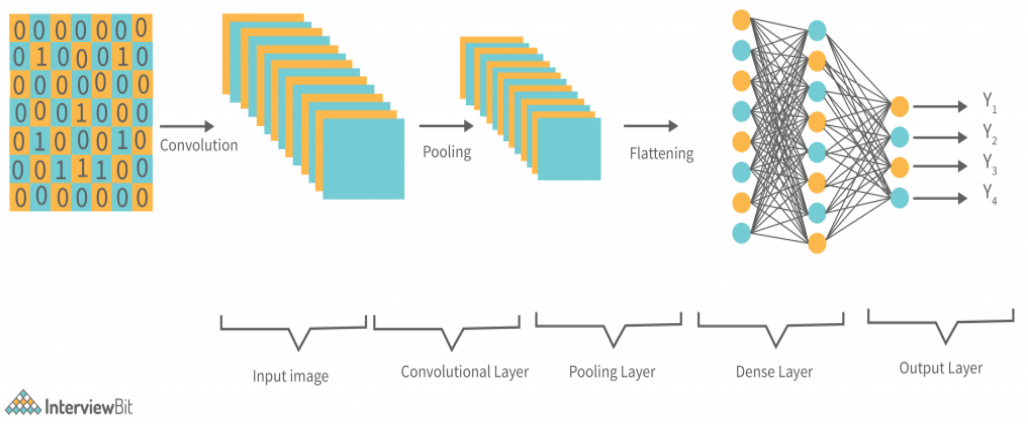
Deep learning techniques are based on neural networks, which are a subset of machine learning. They are made up of node layers, which have an output layer, an input layer, and one or more hidden levels. Every node has a threshold and weight that are connected to one another. A node is activated and sends data to the following layer of the network if its output exceeds a given threshold value. If not, no data is transferred to the network's subsequent tier.

The term "computer vision" is becoming more and more fashionable in the field of image processing. The need for automatic object recognition has grown significantly

with the advent of computer vision applications. Excellent achievements in video processing, object recognition, picture classification and segmentation, natural language processing, speech recognition, and many other domains have been made possible by deep CNNs. This has been beneficial to the computer vision community. Moreover, the advent of copious amounts of data and easily accessible hardware has created new opportunities for CNN research. Numerous thought-provoking ideas for CNN's advancement have been researched, including regularization, parameter optimization, alternative activation functions, and architectural breakthroughs. Moreover, obtaining architectural changes leads to a massive improvement in the deep CNN's capacity.

How do neural networks with convolutions operate?

Figure 3: CNN Architecture



Source: By Google

CNNs function better with picture, speech, or audio signal inputs than other types of neural networks. There are three primary categories of layers in them:

- Convolutional layer
- Pooling layer
- Fully connected (FC) layer

The convolutional layer is the initial layer of a convolutional network. The fully connected layer is the last layer, even if convolutional layers might be followed by pooling layers or further convolutional layers. The CNN becomes more complicated with each layer, recognizing a larger area of the image.

Instead of them, there are more layers

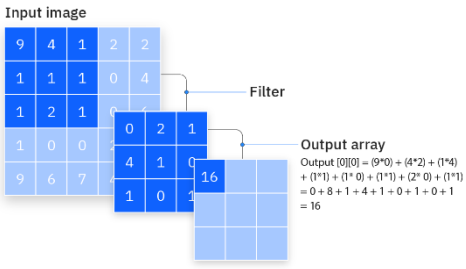
- Input Layer
- Flatten Layer
- SoftMax/Logistic Layer
- Output Layer

Convolution Layer

The majority of computation in a CNN takes place in the convolutional layer, which is its fundamental building component. Three things are needed: a feature map, a filter, and input data. Assume for the moment that the input will be a colour image composed of a three-dimensional (3D) matrix of pixels. This indicates that the three dimensions of the input—height, width, and depth—will match the RGB values in an image. Additionally, we have a feature detector—also referred to as a kernel or filter—that will scan the image's receptive fields to determine whether the feature is there. We call this process a convolution.

A two-dimensional (2D) array of weights that represents a portion of the image serves as the feature detector. The filter size is usually a 3x3 matrix; however, they can vary in size; this also establishes the size of the receptive field. After applying the filter to a portion of the image, the dot product between the input pixels and the filter is computed. An output array is then supplied with this dot product. The filter then moves one step forward and backward until the kernel has scanned the entire image. A feature map, activation map, or convolved feature is the result of the series of dot products created from the input and filter.

Figure 4: Convolution Layer



Source: By Google

Prior to the neural network starting its training phase, three hyperparameters that impact the output's volume size must be established. These consist of

- Kernel/Filter

The output's depth is influenced by the number of filters. For instance, obtaining three different feature maps with three different filters would result in a depth of three.

- Stride

The number of pixels the kernel moves across the input matrix is known as its stride. A bigger stride results in a reduced output, albeit stride values of two or higher are uncommon.

- Padding

When the filters do not match the input image, zero-padding is typically utilized. This results in an output that is larger or equal in size by setting all elements that are outside of the input matrix to zero. Three different kinds of padding exist:

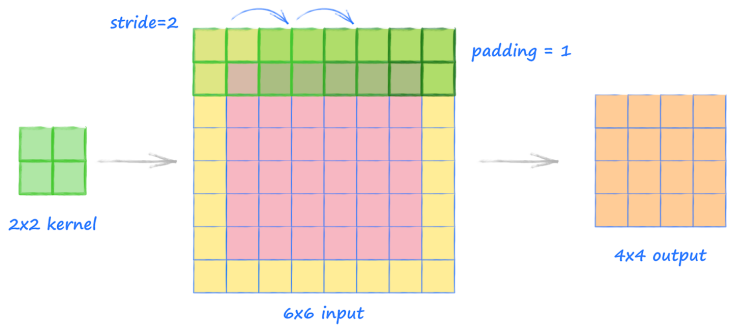
Valid padding**:** No padding is sometimes referred to as valid padding. In this instance, if dimensions are not aligned, the last convolution is dropped.

Same padding: With the same cushioning by using padding, the output layer's size is guaranteed to match that of the input layer.

Full padding**:** Complete padding by adding zeros to the input's border; this kind of padding expands the output's size.

A CNN applies a Rectified Linear Unit (ReLU) adjustment to the feature map following each convolution operation, which adds nonlinearity to the model.

Figure 5: Kernel, Stride, Padding



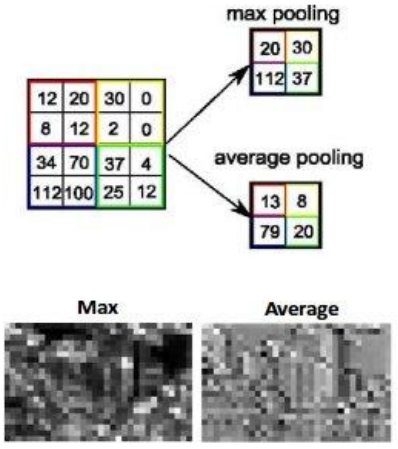
Source: By Google

Pooling Layer

Pooling layers or down sampling does dimensionality reduction by lowering the number of parameters in the input. While the pooling operation sweeps a filter across the entire input, it differs from the convolutional layer in that the filter is weightless. Rather, the values in the receptive field are subjected to an aggregation function by the kernel, which then fills the output array. Two primary categories of pooling exist:

- Max pooling: The filter chooses the pixel with the highest value to send to the output array as it passes through the input. In addition, this method is typically applied more frequently than ordinary pooling.
- Average pooling: The filter determines the average value in the receptive field to send to the output array as it passes through the input.

Figure 6: Max Pooling & Average Pooling



Source: By Google

Flatten Layer

The core responsibility of the flatten layer is to flatten the image into a column vector.

Fully Connected Layer

In partially linked layers, the pixel values of the input image are not directly connected to the output layer. Every output layer node in the fully connected layer, on the other hand, is directly connected to every other layer node.

SoftMax/Logistic Layer

The final layer of CNN is this one. After the input passes through each of the aforementioned levels, it is in charge of providing the categorized output. SoftMax is used for multi-classification, and logistic is used for binary classification.

Technology Stack

Figure 7: OpenCV & Photoshop

Source: By Google

Figure 8: Tensorflow & Keras

Source: By Google

Figure 9: Python (Google Colab & Jupyter Notebook)

Source: By Google

Figure 10: Android Studio

Source: By Google

Keras and TensorFlow

Python-written Keras is a high-level Application Programming Interface (API) that TensorFlow uses to create and train deep learning models. Keras can be used for cutting-edge research and production, as well as quick prototyping. It may be extended and is easy to use. Keras gives users unambiguous feedback. Instead of developing it from scratch, customers find it simpler to import several APIs from Keras. TensorFlow is an open-source machine learning platform. It includes versatile tools, extensive libraries, and other things.

Python

Python, a high-level, interpreted, object-oriented programming language with dynamic semantics. The name "Python" is a reference to the British comedy troupe Monty Python and is meant to be simple and enjoyable. Due to its reputation as a beginner-friendly language, Python has displaced Java as the most popular introductory language. This is because it takes care of a lot of the complexity for the user, freeing them up to concentrate on understanding programming concepts rather than specifics.

Jupyter Notebook

The IPython project, which formerly had its own IPython Notebook project, gave rise to the Jupyter Notebooks project. The programming languages Julia, Python, and R are the three main ones that it supports, hence the name Jupyter. You can write Python applications because Jupyter comes with the IPython kernel.

Google Colab

Colab is a cloud-based notebook environment that is available for free. Google Docs functions similarly to Google Colab. A hosted Jupyter notebook service developed by Google Research is called Google Colab. Python routines can be run through a web

browser. This is ideal for machine learning applications. Up to a specific number of GPUs, it offers the service for free; if we require more, we can purchase Colab Pro.

The performance of Google Colab is as follows.

- Write and run Python code
- Provide documentation for any programming that handles mathematical expressions
- Notebooks can be created, uploaded, shared, and imported into or out of Google Drive.
- Import/Publish notebooks from GitHub
- Integrate PyTorch, TensorFlow, Keras, and OpenCV
- Import external datasets, such as those from Kaggle
- Take advantage of free cloud services with free GPU

Adobe Photoshop

The raster graphics editor Adobe Photoshop was created and released by Adobe Inc. supports both macOS and Windows. With regard to professional digital art, the program has emerged as the most popular tool, particularly for manipulating raster graphics. Photoshop allows masks, alpha compositing, and different colour models in addition to being able to edit and compose raster pictures on numerous layers.

Android Studio

The official Android Studio Integrated Development Environment (IDE) is used to create Android apps. Android Studio provides additional capabilities that increase your efficiency when creating Android apps, and it is built on top of the robust code editor and developer tools from IntelliJ IDEA.

Libraries Used

OpenCV

A software library for computer vision and machine learning that is available for free is called OpenCV (Open-Source Computer Vision Library). In order to facilitate the use of machine perception in commercial goods and to give computer vision applications a common foundation, OpenCV was developed. OpenCV's license for Apache 2 makes it simple for companies to use and alter the code. More than 2500 optimized algorithms are available in the collection, including a wide range of both traditional and cutting-edge computer vision and machine learning techniques.

NumPy

In Python, NumPy can be introduced as a stand-alone library. This library is a general-purpose array processing package. The NumPy library is a utility for working with multidimensional arrays and offers a high-attainment multidimensional array object.

A fundamental library package for Python used in scientific computing is called NumPy.

It consists of various important features, like the following:

- A strong N-dimensional array object
- Sophisticated (broadcasting) functions
- Tool for integrating C/C++ and the Fortran code
- Useful linear algebra, Fourier transform, and random number capabilities. Apart from these obvious scientific functionalities, the NumPy library also can be used as an efficient multi-dimensional container of generic data. Arbitrary data types also can be illustrated using the Numpy library which allows NumPy to seamlessly and quickly integrate with a large variety of databases.

Matplotlib

Python programming language and its NumPy numerical mathematics extension, Matplotlib, is a graphing library. It offers an object-oriented API that may be used with general-purpose GUI toolkits such as Tkinter, wxPython, Qt, or GTK to embed plots into applications.

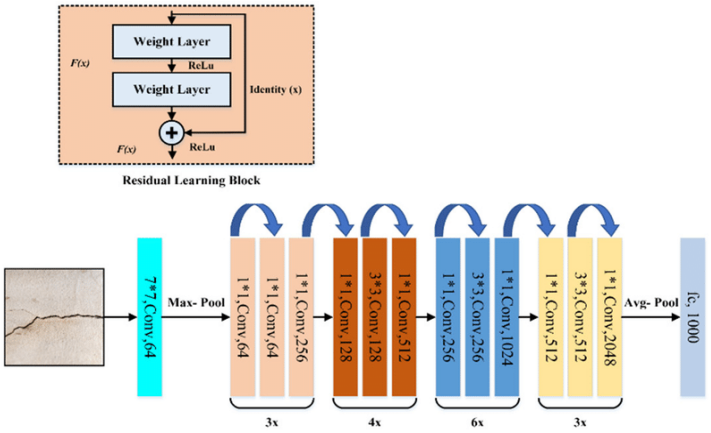
CNN Algorithm and Models

This research is mainly based on CNN, which is a part of DL. During this study, 2 types of CNN models that are trained and tested using a subset of the PlantVillage dataset were used.

ResNet50 Architecture

- Introduces residual connections, also known as skip connections, to address the vanishing gradient issue and enables very deep networks.
- There are many depths of ResNet architectures, including ResNet-50, ResNet-101, and ResNet-152.

Figure 11: ResNet - 50

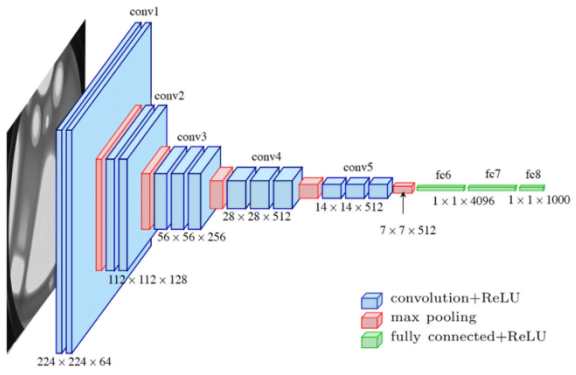


Source: By Google

VGGNet Architecture

- Well-known for its deep layer application of 3x3 convolutional filters and simplicity.
- Two well-liked variations with 16 and 19 weight layers, respectively, are VGG16 and VGG19.
- A common starting point for image classification problems is the VGG network.

Figure 12: VGG-16



Source: By Google

Dataset

In the PlantVillage dataset, images of anthurium leaves are used. The Anthurium subset has 400 (326) images of anthurium leaves and is classified into three classes. There is one class of healthy anthurium leaf and another class of anthurium leaf diseases. Types of anthurium leaf diseases include bacterial blight, bacterial wilt, and others. Here, a healthy leaf was used. In this area, there are mainly only two leaf diseases: anthurium and others spread in anthurium flowers and roots. This research is mainly focused on target leaf, so those diseases are obtained.

Figure 13: Example of colour anthurium leaf images 1) Bacterial Light 2) Bacterial Wilt 3) Healthy

Source: Developed by Author

RESEARCH METHODOLOGY AND IMPLEMENTATION

Methodology

Data Loading

The researcher's local PC has 8 GB of RAM and an i7 processor; hence the training process could be done using it because this process requires a high capacity of RAM and a higher processing power. Because of that, it was much easier to train a model with Anaconda in a Jupyter notebook. But it needs some high epochs and time. After doing all the model creation using a Jupyter notebook, Google Colab. It allows the free use of GPUs. The dataset was stored in the researcher's Google Drive folder. This shows the zip folder file uploaded and how to unzip it. Jupyter notebook was used on the researcher's local PC for training, but it took some time as it used CPU power. Instead, it was decided to use Google Colab because GPUs or TPUs can be used.

Import Dataset

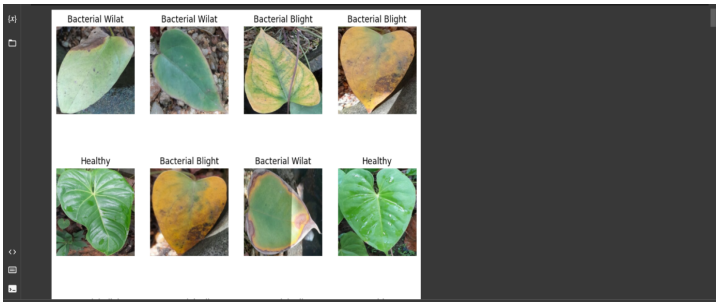
When the main model trains, a separate dataset was not obtained. The entire dataset was divided into three parts. Train (80%), test (10%), and validate (10%). After comparing whether this affects it or not, the first three data types were obtained: train, test, and validate, and then trained the same model code.

Figure 14: Import Dataset

```
[ ] len(dataset)

    11

[ ] plt.figure(figsize=(10, 10))
    for image_batch, label_batch in dataset.take(1):
        for i in range(12):
            ax = plt.subplot(3, 4, i+1)
            plt.imshow(image_batch[i].numpy().astype("uint8"))
            plt.title(class_names[label_batch[i]])
            plt.axis("off")
```

Source: Developed by Author

Figure 15: Data



Source: Developed by Author

Import Libraries

At the beginning, Python libraries are imported as follows: Before entering libraries, the data set is entered. Then I will imp the Keras, TensorFlow, Matplot, and NumPy libraries.

Figure 16: Import Libraries

```
[ ] import tensorflow as tf
    import matplotlib.pyplot as plt

[ ] import keras
    import tensorflow.keras as keras

[ ] import numpy as np
```

Source: Developed by Author

Existing Models Implementation

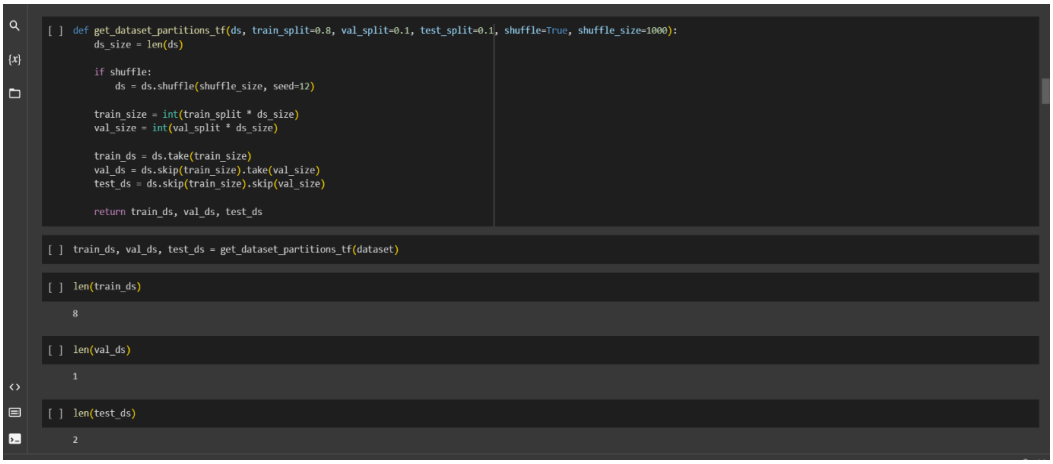
Splitting the Dataset

Three subsets of the dataset should be split, specifically:

- *Training:* Dataset to be used while training
- *Validation:* Training dataset used for testing
- *Test:* After training a model, this dataset will be used for testing.

Here we get the length of the whole dataset. Because it is needed to measure final accuracy, separate datasets for training, testing, and validating are not obtained. All datasets should be divided into three parts.

Figure 17: Split Dataset



Source: Developed by Author

Cache, Shuffle, and Prefetch the Dataset

- Cache - By enabling rapid dataset access, caching shortens the time required to read and load data from slower storage, like disk or network.
- Shuffle - By rearranging the samples, the model is unable to identify patterns in the data that may have been caused by the order in which they were gathered or organized.
- Prefetch - process of loading and getting ready the following batch of data in the background as the one being processed is being processed.

Figure 18: Cache, Shuffle, Prefetch

```
[ ] train_ds = train_ds.cache().shuffle(1000).prefetch(buffer_size=tf.data.AUTOTUNE)
    val_ds = val_ds.cache().shuffle(1000).prefetch(buffer_size=tf.data.AUTOTUNE)
    test_ds = test_ds.cache().shuffle(1000).prefetch(buffer_size=tf.data.AUTOTUNE)
```

Source: Developed by Author

Creating a Layer for Resizing and Normalization

Figure 19: Creating Layers for Resizing & Normalization

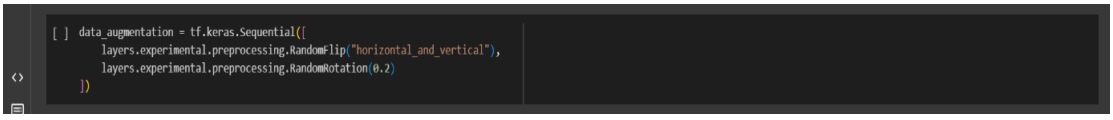
```
[ ] resize_and_rescale = tf.keras.Sequential([
        layers.experimental.preprocessing.Resizing(256, 256),
        layers.experimental.preprocessing.Rescaling(1.0/255)
    ])
```

Source: Developed by Author

Data Augmentation

By adding new variants to the current data, data augmentation techniques enable the model to learn from a larger set of samples. This can be very helpful if a lot of original data is not available. Adding more data can aid in avoiding overfitting.

Figure 20: Data Augmentation



```
[ ] data_augmentation = tf.keras.Sequential([
        layers.experimental.preprocessing.RandomFlip("horizontal_and_vertical"),
        layers.experimental.preprocessing.RandomRotation(0.2)
    ])
```

Source: Developed by Author

Model Architecture

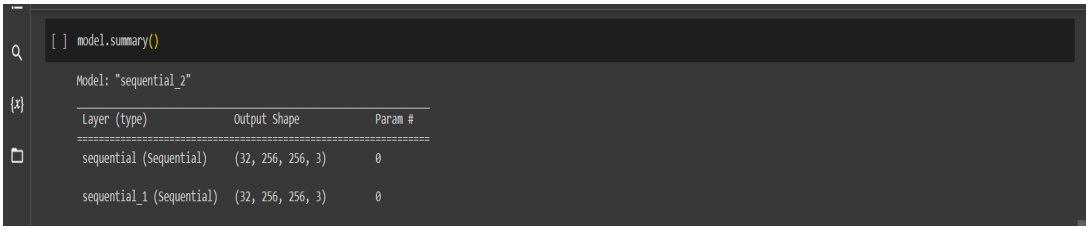
Here, 6 convolutional layers are used along with 6 max pooling layers. Batch size is 32 and the image sizes are 256 x 256. Then they are flattened, and ‘SoftMax’ is used as a fully connected layer, as it is multiple classification.

Figure 21: Model Architecture

```
[ ] input_shape = (32, 256, 256, 3)
    n_classes = 3
    model = models.Sequential([
        resize_and_rescale,
        data_augmentation,
        layers.Conv2D(32, (3,3), activation='relu', input_shape = (256,256)),
        layers.MaxPooling2D((2,2)),
        layers.Conv2D(64, kernel_size = (3,3), activation='relu'),
        layers.MaxPooling2D((2,2)),
        layers.Conv2D(64, kernel_size = (3,3), activation='relu'),
        layers.MaxPooling2D((2,2)),
        layers.Conv2D(64, (3,3), activation='relu'),
        layers.MaxPooling2D((2,2)),
        layers.Conv2D(64, (3,3), activation='relu'),
        layers.MaxPooling2D((2,2)),
        layers.Conv2D(64, (3,3), activation='relu'),
        layers.MaxPooling2D((2,2)),
        layers.Flatten(),
        layers.Dense(64, activation='relu'),
        layers.Dense(n_classes, activation='softmax')
    ])
    model.build(input_shape)
```

Source: Developed by Author

Figure 22: Model Summary



```
[ ] model.summary()

Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 sequential (Sequential)     (32, 256, 256, 3)         0

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

Source: Developed by Author

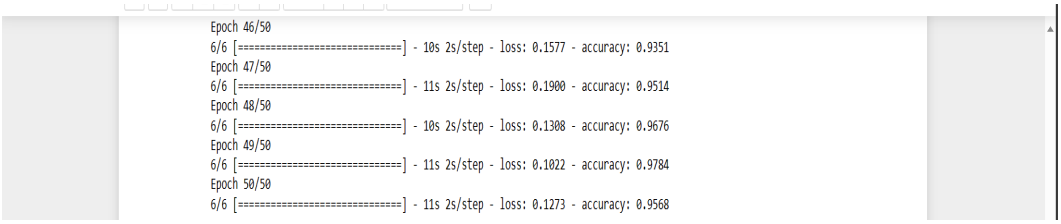
Compiling the Model

Figure 23: Compiling Model

```
[ ] model.compile(
        optimizer='adam',
        loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=False),
        metrics=['accuracy']
    )

[ ] Epochs=50

[ ] history = model.fit(
        train_ds,
        epochs=Epochs,
        batch_size=32,
        verbose=1,
        validation_data=val_ds
    )

    Epoch 1/50
    8/8 [==============================] - 16s 132ms/step - loss: 1.0522 - accuracy: 0.4648 - val_loss: 0.8952 - val_accuracy: 0.7500
    Epoch 2/50
    8/8 [==============================] - 0s 62ms/step - loss: 0.7621 - accuracy: 0.6523 - val_loss: 0.6544 - val_accuracy: 0.7188
    Epoch 3/50
    8/8 [==============================] - 0s 62ms/step - loss: 0.6284 - accuracy: 0.7500 - val_loss: 0.6936 - val_accuracy: 0.6562
    Epoch 4/50
```

Source: Developed by Author

Figure 24: Compile



Source: Developed by Author

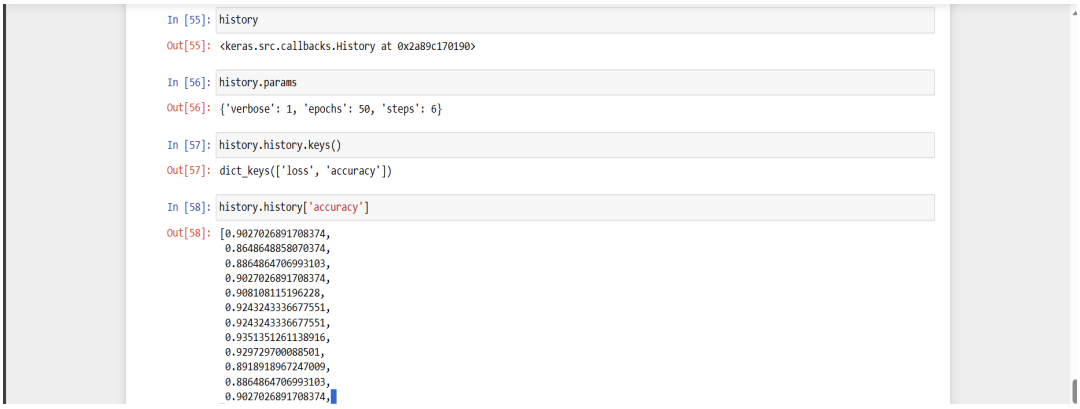
Plotting the Accuracy and Loss Curves

Final test accuracy comes as 98.44%

Figure 25: Test Accuracy

Source: Developed by Author

Figure 26: History



Source: Developed by Author

Figure 27: Test

```
In [ ]: acc = history.history['accuracy']
        val_acc = history.history['val_accuracy']

        loss = history.history['loss']
        val_loss = history.history['val_loss']

In [ ]: plt.figure(figsize=(8,8))
        plt.subplot(1, 2, 1)
        plt.plot(range(Epochs), acc, label='Training Accuracy')
        plt.plot(range(Epochs), val_acc, label='Validation Accuracy')
        plt.legend(loc='lower right')
        plt.title('Training and Validation Accuracy')

        plt.subplot(1, 2, 2)
        plt.plot(range(Epochs), loss, label='Training Loss')
        plt.plot(range(Epochs), val_loss, label='Validation Loss')
        plt.legend(loc='lower right')
        plt.title('Training and Validation Loss')
        plt.show()

In [ ]: np.argmax
```

Source: Developed by Author

Saving the Model

It is required to save the model as *model.tflite* model because it is needed to save the model as a folder and then used for android app.

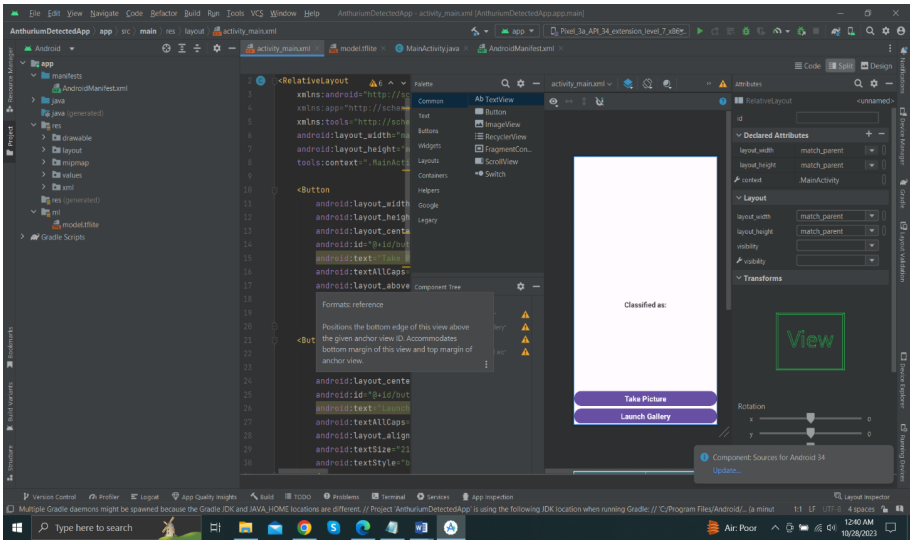
Figure 28: Save Model

```
[ ] converter = tf.lite.TFLiteConverter.from_keras_model(model)
    tflite_model = converter.convert()
    with open("model.tflite", 'wb')as f:
      f.write(tflite_model)
```

Source: Developed by Author

Export the Train Model & Develop Using Android Studio

Figure 29: Develop Using Android Studio



Source: Developed by Author

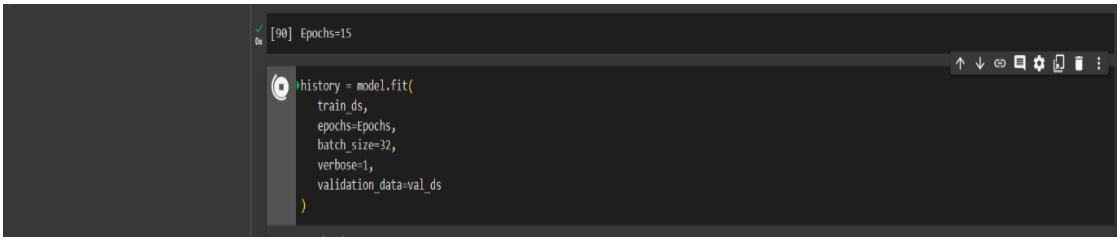
How the results vary with parameters

Now it is required to check how to implement performance or what happens when the parameters and values are changed. It is known that epoch, learning rate, dense layers, and normalization steps are involved to improve performance.

Steps per Epoch

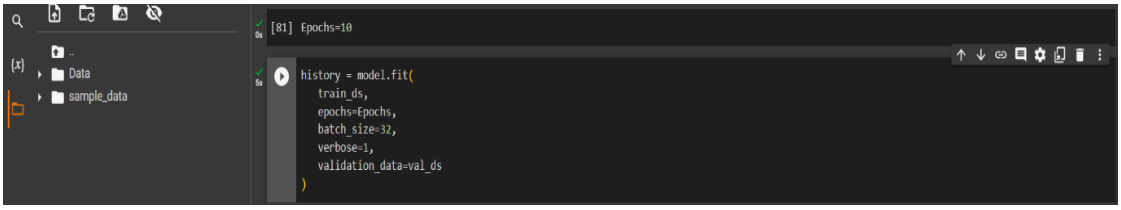
Here, the result variation of model with different Steps Per Epoch values was checked.

Figure 30: Epoch 15



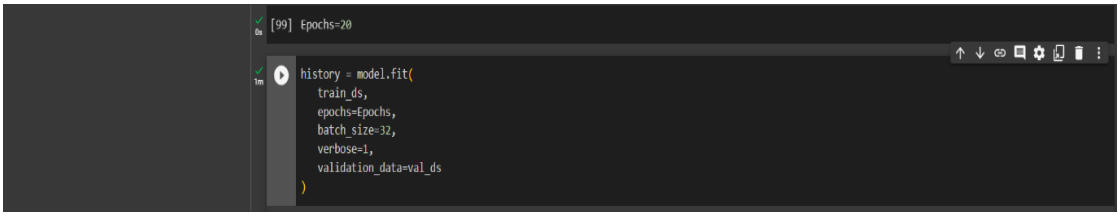
Source: Developed by Author

Figure 31: Epoch 10



Source: Developed by Author

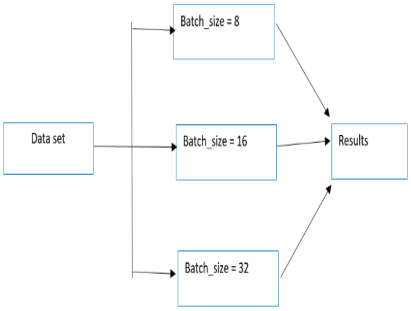
Figure 32: Epoch 20



Source: Developed by Author

Batch Size

Figure 33: Batch Sizes



Source: Developed by Author

Learning Rate

This study considers changing the learning rate from 0.1 to 0.0001.

Figure 34: Learning Rate

```
learning_rate = 0.001

custom_optimizer = tf.keras.optimizers.Adam(learning_rate=learning_rate)

model.compile(
   optimizer=custom_optimizer,
   loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=False),
   metrics=['accuracy']
)
```

Source: Developed by Author

1.1.1 How Results Change When Train, Test, Validate Data are Separately Given

Here, it is required to check which results are uplifted or not and whether it affects the accuracy of the results.

Model Architectures

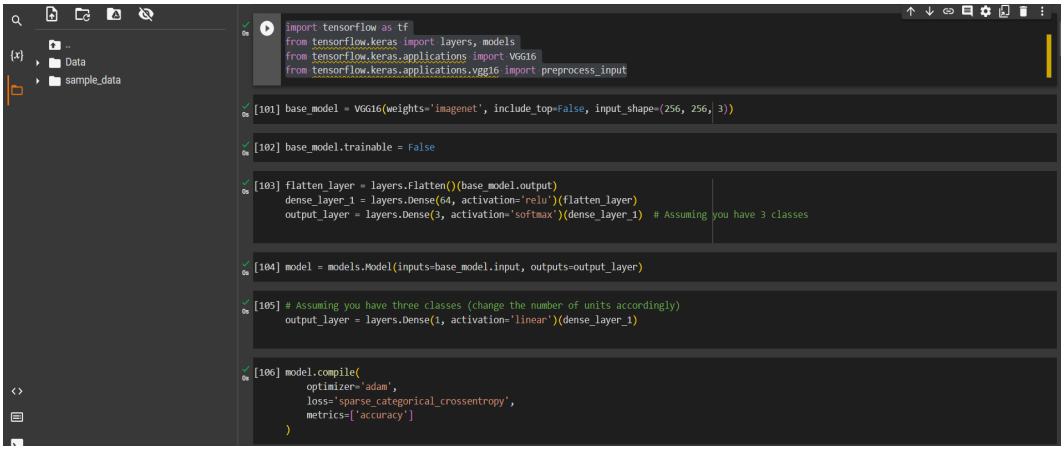
ResNet-50

Figure 35: ResNet-50

Source: Developed by Author

VGG-16

Figure 36: VGG-16



Source: Developed by Author

RESULTS AND FINDINGS

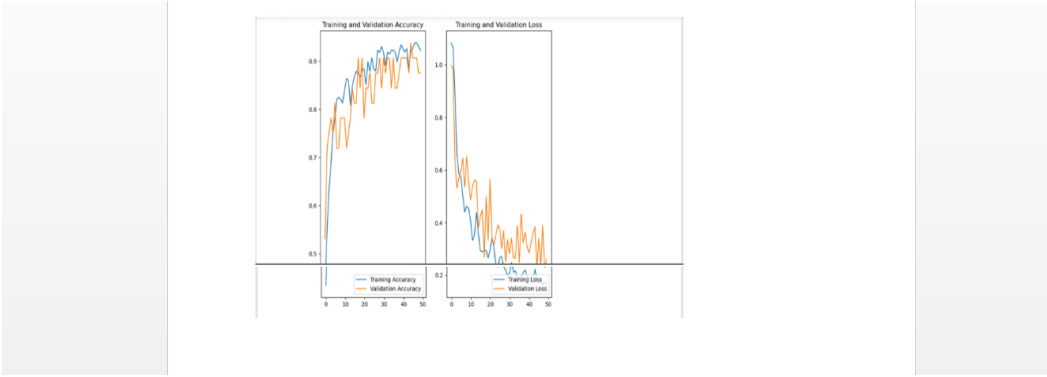
As described in Chapter 4, the present study has used a plant village dataset to train the model and test its accuracy. At first, the whole dataset is obtained, and the model is separated as train, test, and validate. Secondly, the researcher checked with a separate dataset using train, test, and validate to see if the results changed. Then data set was checked for parameter changes, such as epoch, batch size, and learning rate, to see how those parameter changes affected accuracy. It was attempted to

understand how model architectures affect accuracy when layers change; for that, the ResNet-50 and VGG-16 architectures were used.

Output of the Application

Here are the training and validation accuracy and loss diagrams. After predicting the accuracy, it shows the test accuracy and predicts the leaf by the model, and it checks using Android Studio app.

Figure 37: Testing & Validation Accuracy & Loss

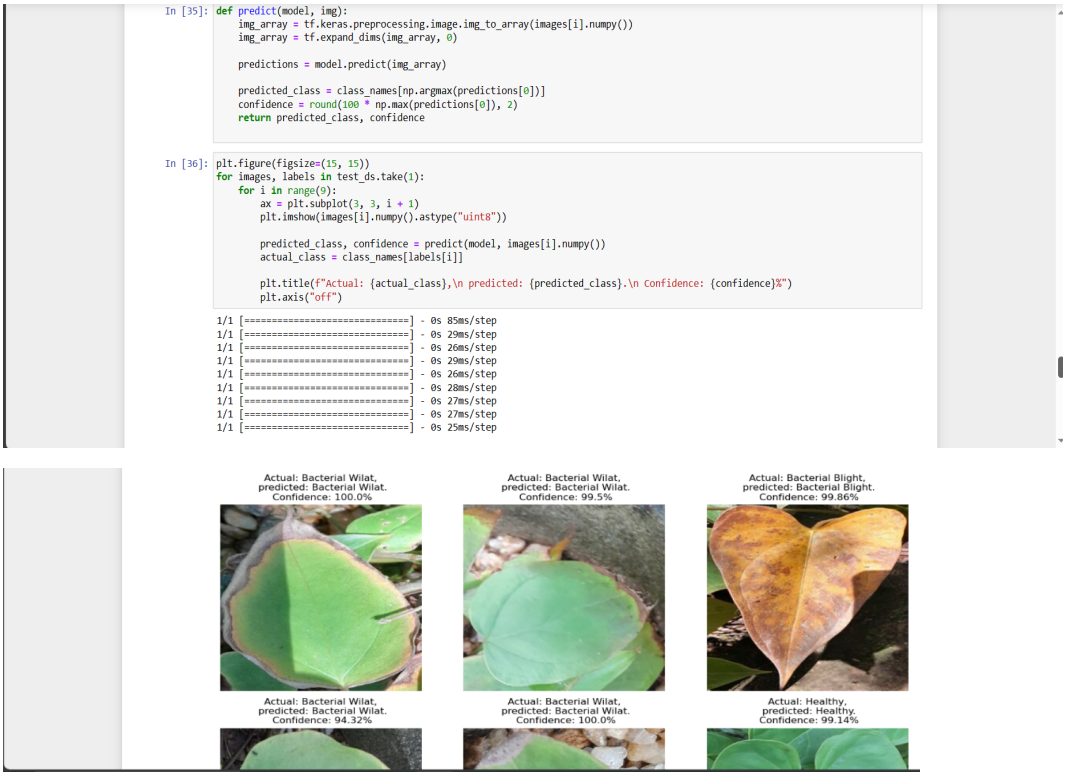


Source: Developed by Author

Figure 38: Prediction

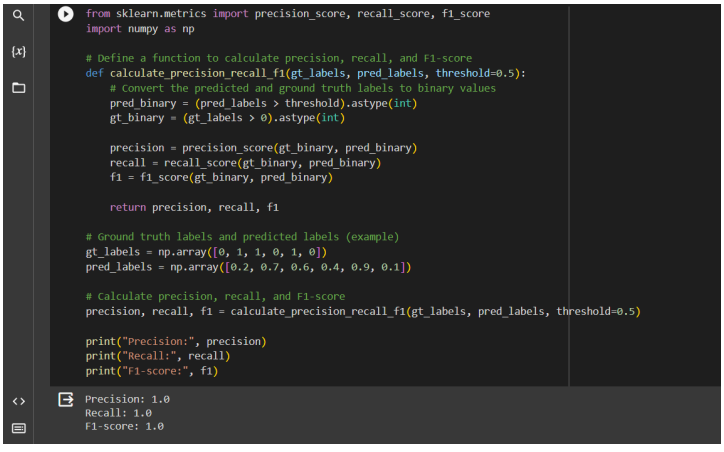
Source: Developed by Author

Figure 39: Predict for 15 images



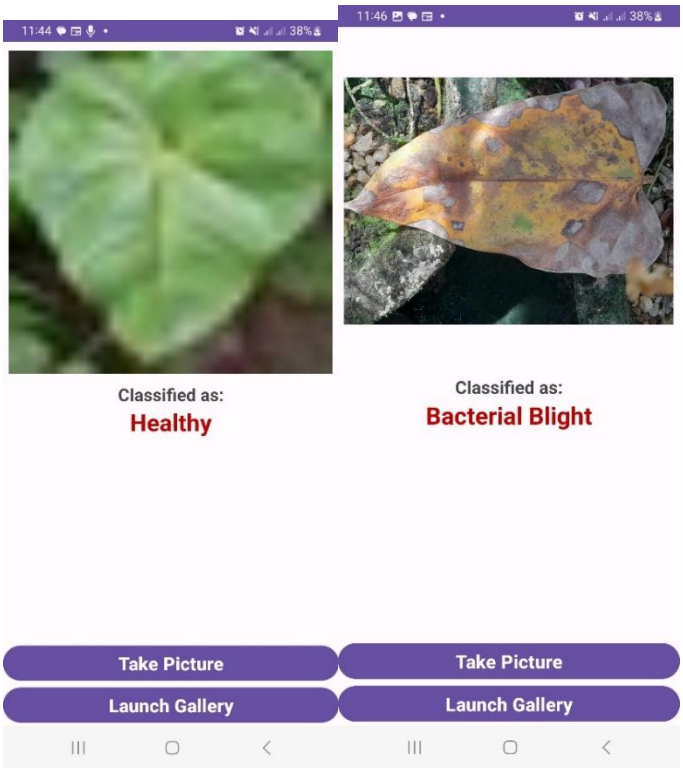
Source: Developed by Author

Figure 40: Precision, Recall, F-Score



Source: Developed by Author

Figure 41: Mobile Application Detecting



Source: Developed by Author

How the Results vary with Parameters

Steps per Epoch

Figure 42: Variation of Epoch

Epoch	Test Accuracy	Compilation Time
5	76.56%	17s
10	81.25%	5s
15	89.06%	8s
20	93.0%	10s
25	95.0%	14s
25	98.0%	13s

Source: Developed by Author

Epoch change is mainly involved in accuracy changes. But compilation time (train accuracy) can change not only the quantity of the epoch; the researcher believes GPU support and other tasks are also affected.

Batch Size

Figure 43: Variation of Batch size

(Consider Epoch = 25)

Batch Size	Test Accuracy	Compilation Time
8	85%	15s
16	92.19%	16s
32	82.81%	16s

Source: Developed by Author

Train accuracy and compilation time are given. It does not affect batch size, but test accuracy is changed according to batch size.

Learning Rate

Figure 44: Variation of Learning Rate

Learning Rate (consider Epoch = 25)

Learning Rate	Test Accuracy	Compilation Time
0.1	59.38%	13s
0.01	59.38%	16s
0.001	85.94%	15s
0.0001	73.44%	17s

Source: Developed by Author

Learning rate mainly involves test accuracy; 0.001 <= 0.0001 is better. Because normal learning rates are between 0.1 and 0.0001.

How Results Change when Train, Test, Validate Data are Given Separately

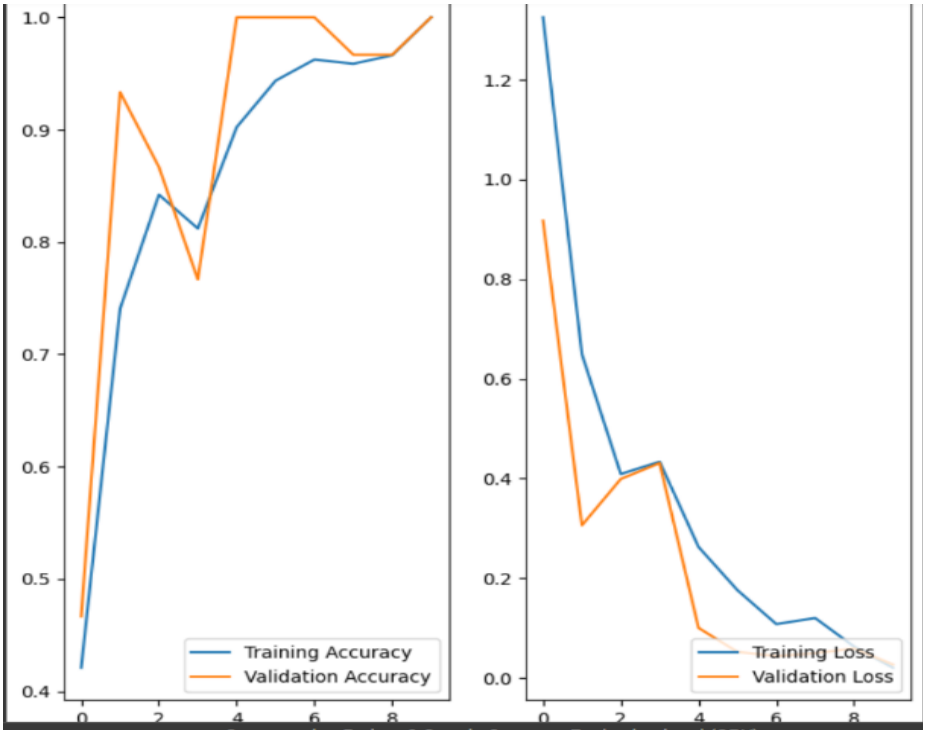
Figure 45: Data divided into 3 Parts

```
[ ]  import tensorflow as tf

     batch_size = 32
     image_size = (256, 256)

     train_ds = tf.keras.utils.image_dataset_from_directory(
         "Data1/Train",
         image_size=image_size,
         batch_size=batch_size
     )

     val_ds = tf.keras.utils.image_dataset_from_directory(
         "Data1/Validation",
         image_size=image_size,
         batch_size=batch_size
     )

     test_ds = tf.keras.utils.image_dataset_from_directory(
         "Data1/Test",
         image_size=image_size,
         batch_size=batch_size
     )

     Found 266 files belonging to 3 classes.
     Found 30 files belonging to 3 classes.
     Found 30 files belonging to 3 classes.
```

Source: Developed by Author

Figure 46: Accuracy & Loss for Separated Dataset

```
[ ] history = model.fit(
        train_ds,
        epochs=10,
        validation_data=val_ds
    )

    Epoch 1/10
    9/9 [==============================] - 3s 86ms/step - loss: 1.3255 - accuracy: 0.4211 - val_loss: 0.9174 - val_accuracy: 0.4667
    Epoch 2/10
    9/9 [==============================] - 1s 64ms/step - loss: 0.6495 - accuracy: 0.7406 - val_loss: 0.3060 - val_accuracy: 0.9333
    Epoch 3/10
    9/9 [==============================] - 1s 76ms/step - loss: 0.4090 - accuracy: 0.8421 - val_loss: 0.3991 - val_accuracy: 0.8667
    Epoch 4/10
    9/9 [==============================] - 1s 67ms/step - loss: 0.4337 - accuracy: 0.8120 - val_loss: 0.4310 - val_accuracy: 0.7667
    Epoch 5/10
    9/9 [==============================] - 1s 63ms/step - loss: 0.2626 - accuracy: 0.9023 - val_loss: 0.1002 - val_accuracy: 1.0000
    Epoch 6/10
    9/9 [==============================] - 1s 61ms/step - loss: 0.1760 - accuracy: 0.9436 - val_loss: 0.0520 - val_accuracy: 1.0000
    Epoch 7/10
    9/9 [==============================] - 1s 64ms/step - loss: 0.1079 - accuracy: 0.9624 - val_loss: 0.0436 - val_accuracy: 1.0000
    Epoch 8/10
    9/9 [==============================] - 1s 62ms/step - loss: 0.1200 - accuracy: 0.9586 - val_loss: 0.0493 - val_accuracy: 0.9667
    Epoch 9/10
    9/9 [==============================] - 1s 61ms/step - loss: 0.0628 - accuracy: 0.9662 - val_loss: 0.0590 - val_accuracy: 0.9667
    Epoch 10/10
    9/9 [==============================] - 1s 62ms/step - loss: 0.0204 - accuracy: 1.0000 - val_loss: 0.0266 - val_accuracy: 1.0000

[ ] model.evaluate(test_ds)

    1/1 [==============================] - 0s 67ms/step - loss: 0.0273 - accuracy: 1.0000
    [0.027268672361969948, 1.0]
```

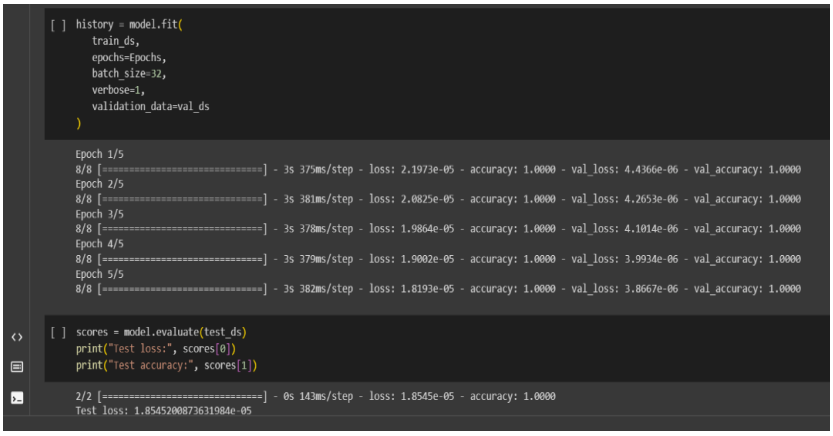


Source: Developed by Author

Model Architectures

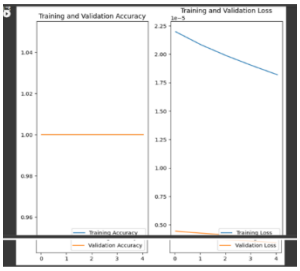
Resnet-50

Figure 47: Resnet-50 Test Accuracy



Source: Developed by Author

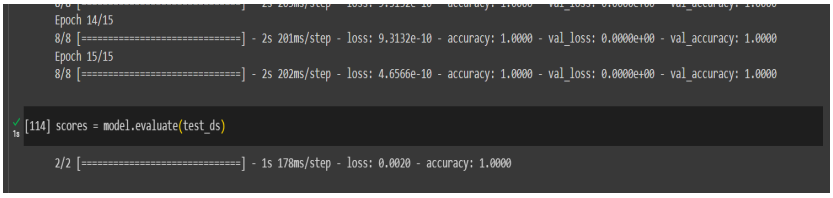
Figure 48: Accuracy & Loss Diagram for Resnet-50



Source: Developed by Author

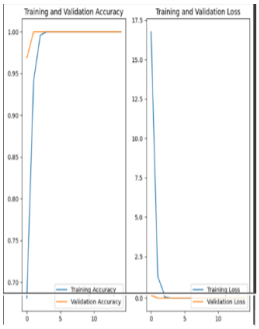
VGG-16

Figure 49: VGG-16 Test Accuracy



Source: Developed by Author

Figure 50: Accuracy & Loss Diagram for VGG-16



Source: Developed by Author

DISCUSSION

In this research, data was obtained from plant villages or cultivation areas to identify those diseases. In addition, help was obtained from some experts in those diseases in the agricultural department and some farmers. After obtaining those datasets, they were categorized into three types of diseases. Then the model was trained using OpenCV and tested for its accuracy, which was at 98.44%. This was predicted using the model and checked more by using the Android mobile app, which can access mobile cameras and galleries.

After finishing the main research, it was required to uplift accuracy more. At first, in the main research, the dataset was obtained as a whole, not divided into train, test, and validate, which is separated into train, test, and validate by using the code randomly. Hence, in this case, the first dataset was divided into train, test, and test validate, and then trained the model as before to check accuracy. Then it is required to achieve 100% accuracy.

Next, more experiments are conducted to see how parameter changes affect the accuracy. For that, epoch, batch size, and learning rate were utilized. Here, the researcher checked how to implement not only test accuracy but also the time spent compiling the train accuracy. It was then understood that the epoch value is mainly involved in test accuracy uplift, but the compilation time can be changed according to GPU power or other reasons. In addition, the best learning rate came in at 0.001. Because 0.1 and 0.01 do not change the accuracy value, high accuracy comes when batch sizes of 32 are used instead of 8 and 16.

Then it is required to study which type of change can happen when model architecture is used for the main model. For this, the ResNet-50 and VGG-16 architectures were used. Both models had a test accuracy of 100%.

ISSUES AND LIMITATIONS

Dataset seeking seems to be a difficult task for the researcher because if requires finding expertise in a particular field. For training the models, the performance of the local PC did not fulfil the sufficient requirements. After training the model, it is the intension of the researcher to create an Android mobile app, but again, the researcher finds some issues when setting up the Android app on mobile. Some runtime issues come up. Moreover, it was attempted to train this model for anthurium flower disease and “black nose” disease identification, but there was no sufficient data when training it.

At first, it was required to build the model by using Tabaco, but only a small dataset was available, and because of security or their rules, it was prohibited to visit those plantations. In addition, during training, the RAM allocation went beyond the limit, and run time was interrupted.

CONCLUSION AND FUTURE WORKS

In the agricultural sector, anthurium is widely cultivated and is a popular edible plant that contains rich nutrition and a unique taste. Anthurium usually provides a good harvest, and it plays an important role in agricultural production and trade around the world. Anthurium diseases are often caused by various factors like bacteria and fungus. Most farmers are unaware of such diseases. Therefore, these diseases can pose an influential threat to cultivation. Hence, the identification of these leaf diseases plays an important role in diseasing control and increasing the quality and quantity of crop yield. Addressing this practical scenario, this research aims to detect and classify the diseases of anthurium plants’ leaves.

A computerized system can dramatically increase the accuracy of the disease detection process and simultaneously improve its throughput. With the development of deep learning, implementing a convolutional neural network with image processing techniques can be considered a better solution for the above-mentioned problem. Throughout this study, the nearest 400 datasets were taken for implementation by using mobile cameras and galleries.

Although CNN has achieved great success in image classification, neural networks are still hard to design. It still requires both human expertise and labour for the development process. A separate dataset and a shuffled dataset were used. Then it is required to check performance by using epoch, batch size, and learning rate. Finally, the model architecture types, which are ResNet-50 and VGG-16, and how to implement them for accuracy were also checked.

Future Work

Those proposed solutions can only be used for trained diseases, and there are only bacterial light and bacterial wilt. Mainly, anthurium leaves have those two diseases only, but this study can be extended to classify any type of plant disease, such as Miata diseases.

REFERENCES

Ahmad, A., Saraswat, D. and El Gamal, A., 2023. A survey on using deep learning techniques for plant disease diagnosis and recommendations for development of appropriate tools. Smart Agricultural Technology, 3, p.100083.

Ahmed, K., Shahidi, T.R., Alam, S.M.I. and Momen, S., 2019, December. Rice leaf disease detection using machine learning techniques. In 2019 International Conference on Sustainable Technologies for Industry 4.0 (STI) (pp. 1-5). IEEE.

Alvarez, A.M., Toves, P.J. and Vowell, T.S., 2006. Bacterial blight of anthuriums: Hawaii's experience with a global disease. APSnet features, 2006.

Archana, K.S. and Sahayadhas, A., 2018. Automatic rice leaf disease segmentation using image processing techniques. Int. J. Eng. Technol, 7(3.27), pp.182-185.

Uchida, J.Y., Sipes, B.S. and Kadooka, C.Y., 2003. Burrowing nematode on anthurium: recognizing symptoms, understanding the pathogen, and preventing disease.

Dou, B., Zhu, Z., Merkurjev, E., Ke, L., Chen, L., Jiang, J., Zhu, Y., Liu, J., Zhang, B. and Wei, G.W., 2023. Machine learning methods for small data challenges in molecular science. Chemical Reviews, 123(13), pp.8736-8780.

Eunice, J., Popescu, D.E., Chowdary, M.K. and Hemanth, J., 2022. Deep learning-based leaf disease detection in crops using images for agricultural applications. Agronomy, 12(10), p.2395.

Gavhale, K.R. and Gawande, U., 2014. An overview of the research on plant leaves disease detection using image processing techniques. Iosr journal of computer engineering (iosr-jce), 16(1), pp.10-16.

Geetha, G., Samundeswari, S., Saranya, G., Meenakshi, K. and Nithya, M., 2020. Plant leaf disease classification and detection system using machine learning. In Journal of Physics: Conference Series (Vol. 1712, No. 1, p. 012012). IOP Publishing.

Hassan, S.M. and Maji, A.K., 2022. Plant disease identification using a novel convolutional neural network. IEEE access, 10, pp.5390-5401.

Holder, A.W., Elibox, W. and Umaharan, P., 2015. Status of bacterial leaf spot disease of Anthurium in Trinidad and characterization of native isolates of the causal organism, Acidovorax anthurii. HortScience, 50(7), pp.1023-1027.

Holder, A.W., Elibox, W., Avey, C. and Umaharan, P., 2017. Identification of field resistance to bacterial leaf spot disease of anthurium under natural epiphytotics in Trinidad. HortScience, 52(1), pp.89-93.

Holder-John, A.W., Elibox, W. and Umaharan, P., 2021. A rapid leaf-disc vacuum-infiltration screening for assessing resistance to bacterial leaf spot disease in anthurium. Scientia Horticulturae, 288, p.110344.

Kumar, V.S., Jaganathan, M., Viswanathan, A., Umamaheswari, M. and Vignesh, J.J.E.R.C., 2023. Rice leaf disease detection based on bidirectional feature attention pyramid network with YOLO v5 model. Environmental Research Communications, 5(6), p.065014.

Li, L., Zhang, S. and Wang, B., 2021. Plant disease detection and classification by deep learning—a review. IEEE access, 9, pp.56683-56698

Mahum, R., Munir, H., Mughal, Z.U.N., Awais, M., Sher Khan, F., Saqlain, M., Mahamad, S. and Tlili, I., 2023. A novel framework for potato leaf disease detection using an efficient deep learning model. Human and Ecological Risk Assessment: An International Journal, 29(2), pp.303-326.

Menghani, G., 2023. Efficient deep learning: A survey on making deep learning models smaller, faster, and better. ACM Computing Surveys, 55(12), pp.1-37.

Mohanty, S.P., Hughes, D.P. and Salathé, M., 2016. Using deep learning for image-based plant disease detection. Frontiers in plant science, 7, p.215232.

Muraleedharan, A., Ramesh Kumar, S., Kousika, S. and Joshi, J.L., 2018. Growth and flowering on Anthurium (Anthurium andreanum) plants treated with foliar application of growth regulators cv. Tropical. Journal of Emerging Technologies and Innovative Research (JETIR), 5(5), pp.1348–1353.

Norman, D.J. and Ali, G.S., 2015. Anthurium diseases: identification and control in commercial greenhouse operations. University of Florida, Institute of Food and Agricultural Sciences Extension, pp292.

R. S. P. K. Alka Singh, “Evaluation of anthurium varieties in greenhouse under South Gujarat condition,” Article in Annals of Biology • July 2018, pp. 1-5, 2018.

Ranasinghe, K.M.N.S. and Liyanage, U.P., 2021. Anthurium disease detector using machine learning techniques. International Conference on Applied and Pure Sciences, 2021.

Sethy, P.K., Barpanda, N.K., Rath, A.K. and Behera, S.K., 2020. Deep feature based rice leaf disease identification using support vector machine. Computers and Electronics in Agriculture, 175, p.105527.

Shelar, N., Shinde, S., Sawant, S., Dhumal, S. and Fakir, K., 2022. Plant disease detection using CNN. In ITM web of conferences (Vol. 44, p. 03049). EDP Sciences.

Shelar, N., Shinde, S., Sawant, S., Dhumal, S. and Fakir, K., 2022. Plant disease detection using CNN. In ITM web of conferences (Vol. 44, p. 03049). EDP Sciences.

Soeb, M.J.A., Jubayer, M.F., Tarin, T.A., Al Mamun, M.R., Ruhad, F.M., Parven, A., Mubarak, N.M., Karri, S.L. and Meftaul, I.M., 2023. Tea leaf disease detection and identification based on YOLOv7 (YOLO-T). Scientific reports, 13(1), p.6078.

Sujatha, R., Chatterjee, J.M., Jhanjhi, N.Z. and Brohi, S.N., 2021. Performance of deep learning vs machine learning in plant leaf disease detection. Microprocessors and Microsystems, 80, p.103615.

Sujatha, R., Chatterjee, J.M., Jhanjhi, N.Z. and Brohi, S.N., 2021. Performance of deep learning vs machine learning in plant leaf disease detection. Microprocessors and Microsystems, 80, p.103615.’;