A Project Report on

PLANT LEAF DISEASE DETECTION AND CLASSIFICATION USING DEEP LEARNING MODELS

Submitted in partial fulfillment for the award of

Bachelor of Technology

in

Computer Science and Engineering

By

J. Satvik(Y19ACS466) **CH. Shilthi Sailaja(Y19ACS433)**

G. Srikanth(Y19ACS446) **K. Naresh Gopi(Y19ACS492)**

Under the guidance of

Mr. K. Suman, Assistant Professor

Department of Computer Science and Engineering

Bapatla Engineering College

(Autonomous)

(Affiliated to Acharya Nagarjuna University)

BAPATLA – 522 102, Andhra Pradesh, INDIA

2022-2023

Department of Computer Science and Engineering

CERTIFICATE

This is to certify that the project report entitled **Plant Leaf Disease Detection and Classification using Deep Learning Models** that is being submitted by **J. Satvik (Y19ACS466), CH. Shilthi Sailaja (Y19ACS433), G. Srikanth (Y19ACS446), K. Naresh Gopi (Y19ACS492)** in partial fulfillment for the award of the Degree of Bachelor of Technology in Computer Science and Engineering to the Acharya Nagarjuna University is a record of bonafide work carried out by them under our guidance and supervision.

Date:

Mr. K. Suman
Assistant Professor

Dr. P. Pardhasaradhi
Professor & HoD

Declaration

We declare that this project work is composed by ourselves, and that the work contained herein is our own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

J. Satvik (Y19ACS466)

CH. Shilthi Sailaja(Y19ACS433)

G. Srikanth(Y19ACS446)

K. Naresh Gopi(Y19ACS492)

Acknowledgement

We sincerely thank the following distinguished personalities who have given their advice and support for successful completion of the work.

We are deeply indebted to our most respected guide **Mr. K. Suman**, Assistant Professor, Department of CSE, for his valuable and inspiring guidance, comments, suggestions and encouragement.

We extend our sincere thanks to **Dr. P. Pardhasaradhi**, Prof. & Head of the Dept. for extending his cooperation and providing the required resources.

We would like to thank our beloved Principal, **Dr. Sk. Nazeer** for providing the online resources and other facilities to carry out this work.

We would like to express our sincere thanks to the project coordination committee and our project coordinator **Dr. N. Sudhakar,** Prof., Dept. of CSE for their helpful suggestions throughout the project work and in presenting this document.

We extend our sincere thanks to all other teaching faculty and non-teaching staff of the department, who helped directly or indirectly for their cooperation and encouragement.

J. Satvik(Y19ACS466)

CH. Shilthi Sailaja(Y19ACS433)

G. Srikanth(Y19ACS446)

K. Naresh Gopi(Y19ACS492)

Abstract

Plant diseases are one of the major challenges facing agriculture, causing significant economic losses and affecting food security. They can also affect the aesthetic value of plants in gardens and landscapes, making them less attractive to consumers.

Cardamom is a prominent spice. It is indigenously grown in the evergreen forests of Karnataka, Kerala, Tamil Nadu, and the northeastern states of India. India is the third largest producer of cardamom. Thus, the early detection of plant diseases in these plants can reduce the harvesting loss in dreadful cases. The work focuses on maladies of cardamom plants such as Colletotrichum Blight and Leaf Spot.

Deep learning is chosen in various applications due to its profound discerning capability. In recent years, deep learning has extended its domain to plant pathology and has proven effective in different scenarios. The cardamom dataset is fed into U^2 Net for segmenting images and the obtained results are then compared with the classification outcomes such as Accuracy, F1-Score, Precision, recall of ResNet152V2 with other state-of-art methods such as Convolutional Neural Network (CNN) and EfficientNet models. The experimental results showed that the proposed approach achieved a detection accuracy of 98.438%.

Table of Contents

List of Figures

List of Tables

1 Introduction

Plant diseases are one of the major challenges facing agriculture, causing significant economic losses and affecting food security. They can also affect the aesthetic value of plants in gardens and landscapes, making them less attractive to consumers.

Traditional Methods of plant disease diagnosis involve visual inspection and laboratory tests, which can be time-consuming, expensive, and require specialized skills. The timely detection and accurate classification of plant diseases are critical for effective management and control of the diseases.

Plant diseases detection involves the identification of abnormal symptoms or signs in plants caused by pathogens, environment factors, or other stressors. In recent years, there has been an increasing interest in developing automated and reliable methods for plant diseases detection using various technologies, including image processing, machine learning, and deep learning.

Plant diseases classification refers to the process of identifying the type of disease present in plant based on the observed symptoms or signs. Accurate classification of plant disease can help farmers and researchers male informed decisions about treatment and management strategies.

Recent advancements in artificial intelligence and computer vision have led to the development of sophisticated algorithms for plant diseases detection and classification. Real time plant disease detection has some significant challenges, such as complex background and severity of the disease due to the images being captured in real-time scenarios from the farm field. Therefore recognizing the plant diseases

automatically in real-time using an alternative approach has become essential for farmers.

1.1 Cardamom Plant

Cardamom is a queen of spices. It is indigenously grown in the evergreen forests of Karnataka, Kerala, Tamil Nadu, and the northeastern states of India. India is the third largest producer of cardamom.

Cardamom leaves are a rich source of essential oils, which have various medicinal properties. In addition to its medicinal properties, cardamom leaves also have culinary uses. Small cardamom is affected by a host of pathogenic bacteria, which seriously damages the crop and is often harmful and emerged frequently as a result of poor crop management.

1.1.1 Cardamom Dataset

Data Set Url : https://github.com/sunilchinnahalli/Cardamom-Dataset-2021

This dataset contains 1724 images of Healthy Cardamom leaves along with two maladies of cardamom plants such as Colletotrichum Blight and Phyllosticta Leaf Spot of Cardamom

HEALTHY LEAVES = 781

COLLETOTRICHUM BLIGHT = 280

PHYLLOSTICTA LEAF SPOT = 663

1.1.2 Diseases of Cardamom Plant

Cardamom is an important spice crop grown in several countries such as India, Tanzania and Sri Lanka. The plant is prone to several diseases that can lead to reduced yield and quality of spice.

Here are some of the common diseases that affect cardamom plants. They are Colletotrichum Blight and Leaf Spot as shown in Figure 1.1 and Figure 1.2 respectively. These images are collected from a cardamom plantation using different electronic devices and set as a bench-mark dataset for the subsequent study.

Figure 1.1 Colletotrichum Blight

Figure 1.2 Leaf Spot

Figure 1.3 Cardamom Healthy

1.2 Deep Learning

Deep Learning is a subset of machine learning that involves training artificial neural networks to learn from large amounts of data and make predictions or decisions based on that learning. Deep learning models consist of multiple layers of interconnected nodes, each layer processing and transforming the input data until a final output is produced.

1.2.1 Advantages of Deep Learning

1. Feature extraction: Deep learning algorithms can automatically learn and extract relevant features from plant leaf images without the need for manual feature engineering. This can save time and effort compared to traditional machine learning approaches that require feature engineering.
2. Improved accuracy: Deep learning algorithms can learn complex patterns in data and identify subtle differences in plant leaf symptoms that are difficult for humans to detect. This can lead to more accurate disease detection and classification compared to traditional machine learning algorithms.
3. Scalability: Deep learning algorithms can be easily scaled up to handle large datasets, making it possible to analyze images from multiple locations and

plants simultaneously. This can save time and resources compared to traditional machine learning approaches that may not be scalable.

4. Transfer learning: Transfer learning is a technique that allows pre-trained deep learning models to be adapted to new tasks with minimal training data. This makes it possible to use pre-trained models to detect and classify new plant leaf diseases without the need for large amounts of training data. Traditional machine learning approaches may not have the same flexibility and adaptability.
5. Real-time processing: Deep learning algorithms can process large amounts of data quickly and accurately, making it possible to analyze large datasets of plant leaf images in real-time. This can be important for early detection and timely response to plant diseases.

1.3 Segmentation

Segmentation is the process of dividing an image into multiple segments or regions, each representing a different object or part of the image. In computer vision and image processing, segmentation is an important technique that is used to extract meaningful information from images, such as identifying objects or regions of interest, removing unwanted background noise, or separating foreground from the background.

There are several types of image segmentation methods, including thresholding, region-based methods, edge detection, and clustering. We also have deep learning-based segmentation methods those are capable of learning complex features and patterns in the images.

These methods can be broadly classified into two categories: instance segmentation and semantic segmentation. Individually both types of segmentation have their own advantages and disadvantages based on the requirement and real time need, we are going select one of them.

1.3.1 Semantic Segmentation

Semantic Segmentation is a pixel-level classification problem. Where the goal is to label each pixel in an image with a corresponding class label.

1.3.2 Advantages of Semantic Segmentation

While semantic segmentation is computationally less expensive and easy to implement.

1. Accurate object detection: Semantic segmentation enables accurate object detection in images identifying and segmenting different objects within an image.
2. Precise localization: By segmenting images at the pixel level, semantic segmentation provides precise localization of objects in an image.
3. Improved object recognition: Semantic segmentation improves object recognition by providing contextual information about the object, which can help in distinguishing between objects with similar appearances.
4. Automated processing: Semantic segmentation enables automated processing of images, which can be helpful in applications where large amounts of data need to be analysed.
5. Real-time processing: With the advancement in deep learning algorithms and hardware, semantic segmentation can be performed in real-time, making it suitable for applications such as leaf disease detection.

1.4 Transfer Learning

In transfer learning for plant leaf disease detection and classification, a pre-trained CNN model is often used as the starting point. The pre-trained model has already learned to extract and recognize basic image features, such as edges and shapes, from a large dataset. By using the pre-trained model as the starting point and fine-tuning it on a smaller dataset of plant leaf images, the model can learn to recognize more complex patterns specific to plant leaf diseases.

Transfer learning can significantly reduce the amount of labeled training data required to achieve high accuracy in plant leaf disease detection and classification. This is particularly useful in cases where labeled training data is scarce or difficult to obtain. In addition, transfer learning can help to overcome overfitting, which occurs when a model becomes too specialized to the training data and does not generalize well to new, unseen data.

Transfer learning methods for plant disease detection and classification can be broadly divided into two categories namely Fine-tuning and Feature Extraction.

Fine-tuning involves taking a pre-trained model and adapting it to the new task by training the last few layers of the network on the new dataset. This approach is useful when the new dataset is similar to the original dataset used to pre-train the model.

Feature extraction involves using the pre-trained model as a fixed feature extractor and training a new classifier on top of the extracted features. This approach is useful when the new dataset is significantly different from the original dataset used to pre-train the model.

1.5 Existing System

In the Existing system, They have used U^2 Net for Background Removal and different deep learning algorithm like CNN ,EfficientNetV2-S, EfficientNetV2-M, EfficientNetV2-L.

1.5.1 Convolution Neural Networks

In convolution layer network, we search patterns in the image. In the first few layer, it can detect corners and lines, and pass these patterns through the neural network to recognize complex features. There are different types of layers in CNN: convolutional, pooling and activation layers. In convolutional layer, to recognize patterns we apply some filters/kernel. When training an image, these weights will change based on loss.

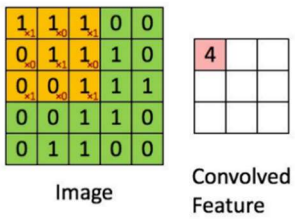

Figure 1.4 Image and Convolved Feature

Pooling layer is mainly used for reducing spatial dimensions, So that we have less parameters to train to avoid overfitting. Activation layer is used for squashes the value in the range. RELU: Rectified Linear Unit → returns x if the x is positive else 0. The first layers basically just encode direction and color. These direction and color filters then get combined into basic grid and spot textures. These textures gradually get combined into increasingly complex patterns. Feature map contains most information about the patterns in the images.

2 Literature Survey

Identification of the plant diseases is the key to preventing the losses in the yield and quantity of the agricultural product. The studies of the plant diseases mean the studies of visually observable patterns seen on the plant. Health monitoring and disease detection on plant is very critical for sustainable agriculture. It is very difficult to monitor the plant diseases manually. It requires tremendous amount of work, expertise in the plant diseases, and also require the excessive processing time.

Plant disease identification by visual way is more laborious task and at the same time, less accurate and can be done only in limited areas. Whereas if automatic detection technique is used it will take less efforts, less time and become more accurate. In plants, some general diseases seen are brown and yellow spots, early and late scorch, and others are fungal, viral and bacterial diseases.

In 2017, S. Zhang, X. Wu, et al., proposed a Leaf image based cucumber disease recognition using sparse representation classification [1] in their paper. They have used K- means clustering for segmentation, extracted features from lesian information and classified using Spare Representation (SR). Their work is able to effectively reduce the computation cost and improve the recognition accuracy.

In 2017, V. Singh, A. K Misra proposed a detection of plant leaf diseases using image segmentation and soft computing techniques [2] in their paper. They have used color co-occurrence methods for extracting features , MDC and SVM for classification. Their work eliminates the needs for user input during segmentation and enhanced accuracy.

In 2019, P. Ganesh, T. F. Burks, et al., proposed a Mask R-CNN based orange detection and segmentation [3] in their paper. They have used Mask R-CNN algorithm for improving mask segmentation performance.

In 2019, L. Yujian, S. Njuki, et al., proposed a comparative study of fine-tuning deep learning models for plant disease detection [4] in their paper. They have used VGG net model, ResNet, InceptionV4, DenseNet, Fine-tuning models and Batch Normalization for coherent increment in accuracy with rising number off epochs, with no manifestations of performance deterioration and overfitting. But their work had to improve on the computational time.

In 2020, Y. Zhong, M. Zhao proposed a research on deep learning in apple leaf disease recognition [5] in their paper. They have used DenseNet, Cross entropy loss function, Focal loss function, Regression for classification. The methods have achieved better recognition results on the unbalanced data set, compared with the traditional single-label multi classification. But their work is limited to single plant disease identification.

In 2020, C.-H. Yeh, M.-H. Lin, et al., proposed Enhanced Visual attention-guided deep neural networks for image classification [6] in their paper. They have used CNN and Huber loss function for improving classification performance with negligible extra overhead.

In 2021, L. M. Tassis, R. A. Krohling, et al., proposed a deep learning approach combining instance and semantic segmentation to identify diseases and pests of coffee leaves from in-field images [7] in their paper. They have used Mask R-CNN, U-Net, PSP Net, Res- Net for implementing the entire framework in an

embedded mobile platform. Their work lacks of capability to classify areas where two or more lesions overlap.

In 2021, J. Shin, Y. K. Chang, et al., proposed a deep learning approach for RGB image-based powdery mildew disease detection on strawberry leaves [8] in their paper. They have used AlexNet, SqueezeNet, GoogleNet, ResNet-50, SqueezeNet for better non-DL approaches. Their work had a limitation for transferability to field conditions.

In 2021, P. Singh, A. Verma, et al., proposed a disease and pests infection detection in coconut tree through deep learning techniques [9] in their paper. They have used K-means clustering segmentation, Keras pre-trained CNN models, custom 2D CNN for improving the accuracy with limited dataset. Their work had a limitation of detecting the severity level of the infection.

In 2021, C. K. Sunil proposed a cardamom plant disease detection approach using EfficientNetV2 [10] in their paper. They have used CNN, EfficientNetV2-S, EfficientNetV2-L, EfficientNetV2-M for classification. Their methods have achieved higher accuracy and better computational time but their work is limited to single plant disease identification and can be improved.

3 Design

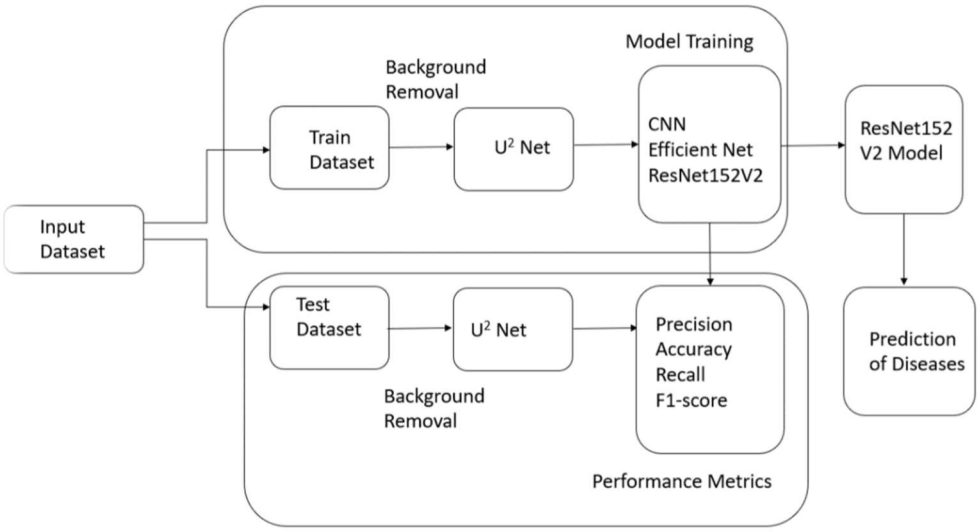

Figure 3.1 Workflow

3.1 Workflow Description

1. First, in training phase the complex background of the images removed by extracting the multiscale features with U²-Net.
2. Second, the segmented images are then passed to various Deep Learning models such as Convolutional Neural Network, EfficientNet, EfficientNetV2-S, EfficientNetV2-M, EfficientNetV2-L, ResNet152V2 and the resultant trained models are obtained.
3. Third the testing images are again given to U²-Net for background removal and is used to compute various performance metrics such as Accuracy, F1-Score, Precision and Recall.
4. Among all the trained models, use the best one for prediction of diseases.

4 Proposed Work & Methodology

4.1 Overview

The major contributions of the project are summarized as follows:

1. There are 2 phase that is followed in this approach, the training and testing phase.
2. To reduce the background complexity, the datasets in two phases are passed to U²-Net.
3. The classification models trained and used for finding performance metrics.
4. With the best obtained model used for prediction of diseases.

4.2 Background Removal

The background removal approach used in this study was U²-Net.

It Consists of 3 Stages:

Stage 1: Residual U-Block(RSU)

1. Extract Local Features
2. Input Convolution Layer - Generation activation map
3. Encoder-Decoder :- like that present in the U-Net

Stage 2 :- Five Stage Decoder using dilated version of RSU.

Stage 3 :- Saliency probability maps generation (attaching decoder with encoder stage)

Employing Deep supervision using the given loss function for reducing loss and manifest robust regularization.

4.3 U^2 Net

Semantic segmentation is to label each pixel of image with corresponding class of what is being represented. The encoder path captures the context of the image producing feature maps. Decoder path used to enable precise localization using transposed convolutions. Preserving the mapping of spatial information from each layer, image is concatenated with the corresponding image from the contracting path.

4.4 U^2 Net Architecture

U^2-Net is a two-level nested U-structure that is designed for SOD without using any pre-trained backbones from image classification. It can be trained from scratch to achieve competitive performance. The residual connections within each UNet block enables focus on local details while the overall residual U-Net architecture enables fusing these local details with global (multi scale) contextual information. RSU mainly consists of three components:

1. An input convolution layer,
2. U-Net like symmetric encoder-decoder structure with height of L (takes the intermediate feature map F1(x) as input and learns to extract and encode the multi-scale contextual information U(F1(x))
3. A residual connection which fuses local features and the multi-scale features by the summation: F1(x) +U(F1(x)).

U^2 Net is a subset of semantic segmentation models and can be used inwide variety of real time applications.

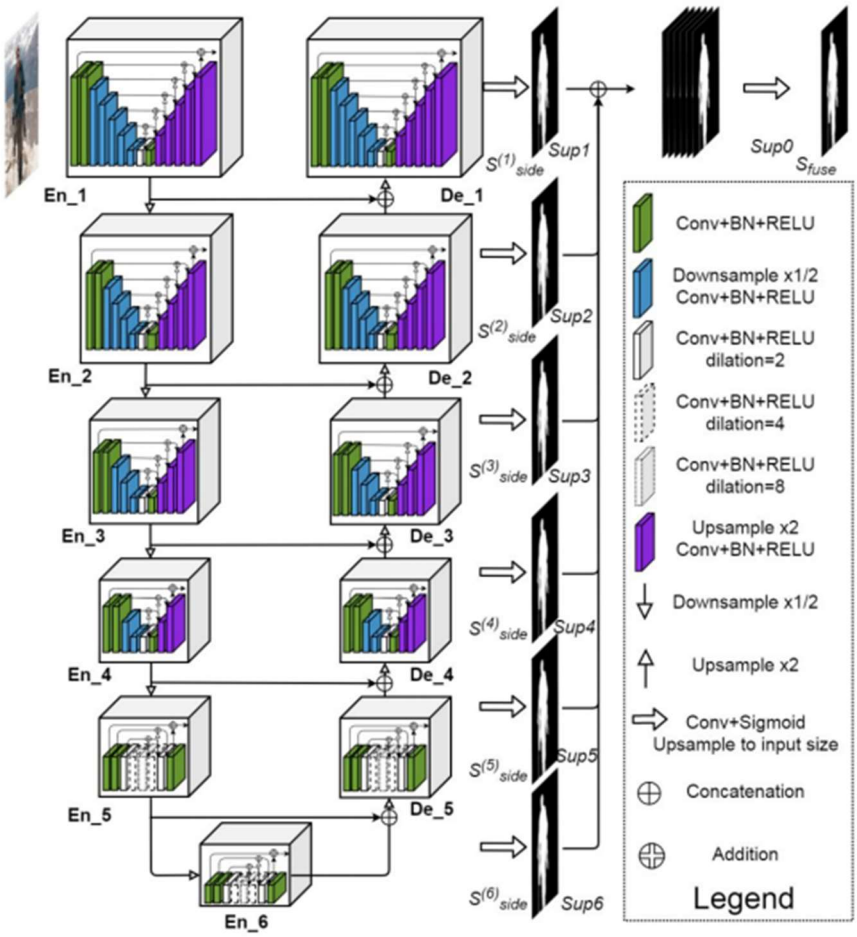

Figure 4.1 U2 Net Architecture

In encoder stages, En_1, En_2, En_3, and En_4, residual U-blocks RSU-7, RSU-6, RSU-5, and RSU-4 are used respectively, “7”, “6”, “5”, and “4” denote the heights (L) of RSU blocks. The resolution of feature maps in En_5 and En_6 is relatively low, further down sampling of these feature maps leads to loss of useful context. Hence, in both En_5 and En_6 stages, RSU-4F are used, where “F” means that the RSU is a dilated version, in this stage the pooling and up sampling operations are replaced with dilated convolutions (Dilated Conv → is basically a convolution with a wider kernel created by regularly inserting spaces between the kernel elements, it helps to expand the area of the image covered without pooling), so that RSU-4F have the same resolution as its input feature maps. U²-Net first generates six side

output saliency probability maps S (6) side, S (5) side, S (4) side, S (3) side, S (2) side, S (1) side from stages En_6, De_5, De_4, De_3, De_2 and De_1 by a 3 × 3 convolution layer and a sigmoid function(applies up sampling to input image size before sigmoid), after that fused together to form S_fuse.

4.5 ResNet

ResNet is a deep neural network that consists of a series of residual blocks. Each residual block contains two or more convolutional layers with shortcut connections that bypass some of the layers. These shortcut connections allow the gradients to flow directly from one layer to another, making it easier for the network to learn complex features. The residual connections also help to prevent the problem of vanishing gradients, which can occur in very deep neural networks. Some of the commonly used ResNet models include ResNet-18, ResNet-34, ResNet-50, ResNet-101, and ResNet-152. These models differ in the number of layers and the number of parameters they have, with ResNet-152 being the deepest and most complex.

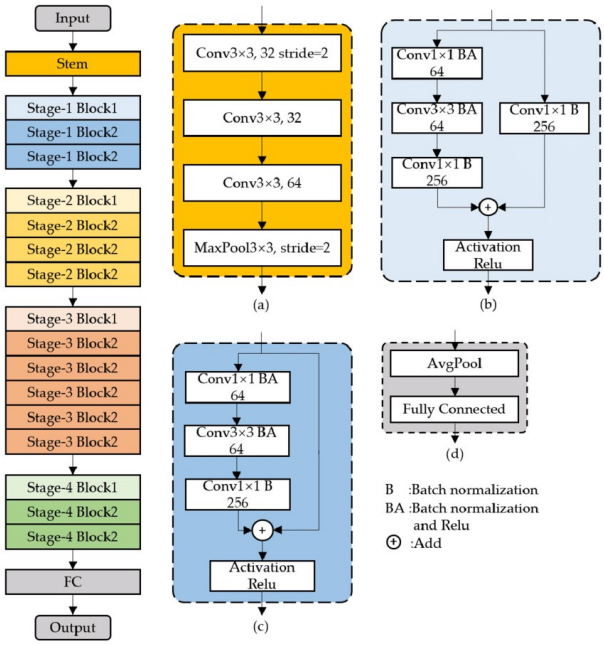

Figure 4.2 ResNet Architecture

4.6 ResNet152V2

ResNet-152v2 is a deep convolutional neural network (CNN) architecture with an advanced version of the ResNet architecture that consists of 152 layers, making it one of the deepest CNNs to date. The architecture of ResNet-152v2 is similar to that of the original ResNet architecture, but it includes several additional features that improve its performance.

The ResNet-152v2 architecture can be divided into several blocks, each of which includes a series of convolutional layers, batch normalization layers, and activation functions. The input to the network is a color image of a plant leaf, typically in the RGB format with a size of 224 x 224 pixels. The output of the network is a probability distribution over a set of plant leaf disease classes.

The ResNet-152v2 architecture includes four main types of blocks: the identity block, the convolutional block, the down-sampling block, and the up-sampling block.

The identity block is the simplest type of block and consists of two convolutional layers, each followed by a batch normalization layer and a ReLU activation function. The input to the block is added to its output, creating a shortcut connection that helps to propagate gradients more efficiently during training.

The convolutional block is similar to the identity block, but it includes an additional convolutional layer at the beginning of the block. This allows the block to learn more complex features.

The down-sampling block is used to reduce the spatial dimensions of the feature maps while increasing their depth. This is achieved through a combination of a

convolutional layer with a stride of 2 and a max pooling layer. The down-sampling block is used to reduce the computational cost of the network and increase its efficiency.

The up-sampling block is used to increase the spatial dimensions of the feature maps while decreasing their depth. This is achieved through a combination of a transposed convolutional layer with a stride of 2 and a batch normalization layer. The up-sampling block is used to recover the spatial information lost during the down-sampling process.

ResNet-152v2 also includes several other features that improve its performance, including global average pooling, weight decay, and dropout regularization.

4.6.1 Advantages of ResNet152V2

1. Deeper Architecture: ResNet152v2 is a deep neural network with 152 layers, which allows it to extract more complex and abstract features from input images compared to shallower models. This deeper architecture can help improve accuracy in plant leaf disease detection and classification tasks.
2. Pretrained Weights: ResNet152v2 is often pretrained on large-scale datasets such as ImageNet, which helps it to learn generalizable features that can be fine-tuned for specific tasks. This pretrained initialization can speed up the training process and improve accuracy.
3. Skip Connections: ResNet152v2 uses skip connections in its residual blocks, which allows the model to better propagate gradients through the network during training. This can help to reduce the vanishing gradient problem and improve convergence during training.

5 Hardware and Software Requirements

5.1 Hardware Requirements

Processor : I5/Intel Processor

RAM : 12 GB

Hard Disk : 128 GB

Key Board : Standard Windows Keyboard

Mouse : Two or Three Button Mouse

5.2 Software Requirements

Operating System : Windows 10

Language used : Python

IDE : GPU based environment

6 Coding

6.1 Dataset

```
data:
¦
¦
+---cardamom dataset
¦   +---test
¦   ¦   +---Blight1000
¦   ¦   +---Healthy_1000
¦   ¦   +---Phylosticta_LS_1000
¦   +---train
¦   ¦   +---Blight1000
¦   ¦   +---Healthy_1000
¦   ¦   +---Phylosticta_LS_1000
¦   +---validation
¦       +---Blight1000
¦       +---Healthy_1000
¦       +---Phylosticta_LS_1000
```

6.2 Complete Folder Structure

```
Cardamom_leaf_disease_Detection_via_U2net:
¦   Resnet152v2.ipynb
¦   app.py
¦   my_model.h5
+---code
¦   +---index.html
¦   MasterU2Net.ipynb
¦   sample_masked(s0).jpg
¦   sample_masked(s6).jpg
¦   data
¦   u2net_SegmentFolders.py
```

```
¦   u2net_util.py
+---models
¦   +---u2net.py
+---saved_models
¦   +---u2net
¦   ¦   +---u2net.pth
+---output
¦   +---224
¦   ¦   +---cardamom dataset
¦   ¦   ¦   +---test
¦   ¦   ¦   ¦   +---Blight1000
¦   ¦   ¦   ¦   +---Healthy_1000
¦   ¦   ¦   ¦   +---Phylosticta_LS_1000
¦   ¦   ¦   +---train
¦   ¦   ¦   ¦   +---Blight1000
¦   ¦   ¦   ¦   +---Healthy_1000
¦   ¦   ¦   ¦   +---Phylosticta_LS_1000
¦   ¦   ¦   +---validation
¦   ¦   ¦       +---Blight1000
¦   ¦   ¦       +---Healthy_1000
¦   ¦   ¦       +---Phylosticta_LS_1000
```

6.3 Modules

In this project, we have combined different modules like u2net.py, u2net SegmentFolders.py and u2net_util.py for process of segmentation.

6.3.1 u2net.py

This module mainly consists of code related to the working of U^2 Net Architecture. The code involves the step by step process how the process of segmentation takes place in algorithm. This module is further linked to other module for interlinking the image dataset.

```
import torch
import torch.nn as nn
import torch.nn.functional as F
class REBNCONV(nn.Module):
    def __init__(self,in_ch=3,out_ch=3,dirate=1):
        super(REBNCONV,self).__init__()
        self.conv_s1 =
nn.Conv2d(in_ch,out_ch,3,padding=1*dirate,dilation=1*dirate)
        self.bn_s1 = nn.BatchNorm2d(out_ch)
        self.relu_s1 = nn.ReLU(inplace=True)
    def forward(self,x):
        hx = x
        xout = self.relu_s1(self.bn_s1(self.conv_s1(hx)))
        return xout
## upsample tensor 'src' to have the same spatial size with
tensor 'tar'
def _upsample_like(src,tar):
    src =
F.interpolate(src,size=tar.shape[2:],mode='bilinear')
    return src
### RSU-7 ###
class RSU7(nn.Module): #UNet07DRES(nn.Module):
    def __init__(self, in_ch=3, mid_ch=12, out_ch=3):
        super(RSU7,self).__init__()
        self.rebnconvin = REBNCONV(in_ch,out_ch,dirate=1)
        self.rebnconv1 = REBNCONV(out_ch,mid_ch,dirate=1)
        self.pool1 = nn.MaxPool2d(2,stride=2,ceil_mode=True)
        self.rebnconv2 = REBNCONV(mid_ch,mid_ch,dirate=1)
        self.pool2 = nn.MaxPool2d(2,stride=2,ceil_mode=True)
        self.rebnconv3 = REBNCONV(mid_ch,mid_ch,dirate=1)
        self.pool3 = nn.MaxPool2d(2,stride=2,ceil_mode=True)
        self.rebnconv4 = REBNCONV(mid_ch,mid_ch,dirate=1)
        self.pool4 = nn.MaxPool2d(2,stride=2,ceil_mode=True)
        self.rebnconv5 = REBNCONV(mid_ch,mid_ch,dirate=1)
        self.pool5 = nn.MaxPool2d(2,stride=2,ceil_mode=True)
        self.rebnconv6 = REBNCONV(mid_ch,mid_ch,dirate=1)
        self.rebnconv7 = REBNCONV(mid_ch,mid_ch,dirate=2)
```

```
        self.rebnconv6d = REBNCONV(mid_ch*2,mid_ch,dirate=1)
        self.rebnconv5d = REBNCONV(mid_ch*2,mid_ch,dirate=1)
        self.rebnconv4d = REBNCONV(mid_ch*2,mid_ch,dirate=1)
        self.rebnconv3d = REBNCONV(mid_ch*2,mid_ch,dirate=1)
        self.rebnconv2d = REBNCONV(mid_ch*2,mid_ch,dirate=1)
        self.rebnconv1d = REBNCONV(mid_ch*2,out_ch,dirate=1)
    def forward(self,x):
        hx = x
        hxin = self.rebnconvin(hx)
        hx1 = self.rebnconv1(hxin)
        hx = self.pool1(hx1)
        hx2 = self.rebnconv2(hx)
        hx = self.pool2(hx2)
        hx3 = self.rebnconv3(hx)
        hx = self.pool3(hx3)
        hx4 = self.rebnconv4(hx)
        hx = self.pool4(hx4)
        hx5 = self.rebnconv5(hx)
        hx = self.pool5(hx5)
        hx6 = self.rebnconv6(hx)
        hx7 = self.rebnconv7(hx6)
        hx6d =  self.rebnconv6d(torch.cat((hx7,hx6),1))
        hx6dup = _upsample_like(hx6d,hx5)
        hx5d =  self.rebnconv5d(torch.cat((hx6dup,hx5),1))
        hx5dup = _upsample_like(hx5d,hx4)
        hx4d = self.rebnconv4d(torch.cat((hx5dup,hx4),1))
        hx4dup = _upsample_like(hx4d,hx3)
        hx3d = self.rebnconv3d(torch.cat((hx4dup,hx3),1))
        hx3dup = _upsample_like(hx3d,hx2)
        hx2d = self.rebnconv2d(torch.cat((hx3dup,hx2),1))
        hx2dup = _upsample_like(hx2d,hx1)
        hx1d = self.rebnconv1d(torch.cat((hx2dup,hx1),1))
        return hx1d + hxin
### RSU-6 ###
class RSU6(nn.Module): #UNet06DRES(nn.Module):
    def __init__(self, in_ch=3, mid_ch=12, out_ch=3):
        super(RSU6,self).__init__()
```

```
        self.rebnconvin = REBNCONV(in_ch,out_ch,dirate=1)
        self.rebnconv1 = REBNCONV(out_ch,mid_ch,dirate=1)
        self.pool1 = nn.MaxPool2d(2,stride=2,ceil_mode=True)
        self.rebnconv2 = REBNCONV(mid_ch,mid_ch,dirate=1)
        self.pool2 = nn.MaxPool2d(2,stride=2,ceil_mode=True)
        self.rebnconv3 = REBNCONV(mid_ch,mid_ch,dirate=1)
        self.pool3 = nn.MaxPool2d(2,stride=2,ceil_mode=True)
        self.rebnconv4 = REBNCONV(mid_ch,mid_ch,dirate=1)
        self.pool4 = nn.MaxPool2d(2,stride=2,ceil_mode=True)
        self.rebnconv5 = REBNCONV(mid_ch,mid_ch,dirate=1)
        self.rebnconv6 = REBNCONV(mid_ch,mid_ch,dirate=2)
        self.rebnconv5d = REBNCONV(mid_ch*2,mid_ch,dirate=1)
        self.rebnconv4d = REBNCONV(mid_ch*2,mid_ch,dirate=1)
        self.rebnconv3d = REBNCONV(mid_ch*2,mid_ch,dirate=1)
        self.rebnconv2d = REBNCONV(mid_ch*2,mid_ch,dirate=1)
        self.rebnconv1d = REBNCONV(mid_ch*2,out_ch,dirate=1)
    def forward(self,x):
        hx = x
        hxin = self.rebnconvin(hx)
        hx1 = self.rebnconv1(hxin)
        hx = self.pool1(hx1)
        hx2 = self.rebnconv2(hx)
        hx = self.pool2(hx2)
        hx3 = self.rebnconv3(hx)
        hx = self.pool3(hx3)
        hx4 = self.rebnconv4(hx)
        hx = self.pool4(hx4)
        hx5 = self.rebnconv5(hx)
        hx6 = self.rebnconv6(hx5)
        hx5d =  self.rebnconv5d(torch.cat((hx6,hx5),1))
        hx5dup = _upsample_like(hx5d,hx4)
        hx4d = self.rebnconv4d(torch.cat((hx5dup,hx4),1))
        hx4dup = _upsample_like(hx4d,hx3)
        hx3d = self.rebnconv3d(torch.cat((hx4dup,hx3),1))
        hx3dup = _upsample_like(hx3d,hx2)
        hx2d = self.rebnconv2d(torch.cat((hx3dup,hx2),1))
        hx2dup = _upsample_like(hx2d,hx1)
```

```
        hx1d = self.rebnconv1d(torch.cat((hx2dup,hx1),1))
        return hx1d + hxin
### RSU-5 ###
class RSU5(nn.Module): #UNet05DRES(nn.Module):
    def __init__(self, in_ch=3, mid_ch=12, out_ch=3):
        super(RSU5,self).__init__()
        self.rebnconvin = REBNCONV(in_ch,out_ch,dirate=1)
        self.rebnconv1 = REBNCONV(out_ch,mid_ch,dirate=1)
        self.pool1 = nn.MaxPool2d(2,stride=2,ceil_mode=True)
        self.rebnconv2 = REBNCONV(mid_ch,mid_ch,dirate=1)
        self.pool2 = nn.MaxPool2d(2,stride=2,ceil_mode=True)
        self.rebnconv3 = REBNCONV(mid_ch,mid_ch,dirate=1)
        self.pool3 = nn.MaxPool2d(2,stride=2,ceil_mode=True)
        self.rebnconv4 = REBNCONV(mid_ch,mid_ch,dirate=1)
        self.rebnconv5 = REBNCONV(mid_ch,mid_ch,dirate=2)
        self.rebnconv4d = REBNCONV(mid_ch*2,mid_ch,dirate=1)
        self.rebnconv3d = REBNCONV(mid_ch*2,mid_ch,dirate=1)
        self.rebnconv2d = REBNCONV(mid_ch*2,mid_ch,dirate=1)
        self.rebnconv1d = REBNCONV(mid_ch*2,out_ch,dirate=1)
    def forward(self,x):
        hx = x
        hxin = self.rebnconvin(hx)
        hx1 = self.rebnconv1(hxin)
        hx = self.pool1(hx1)
        hx2 = self.rebnconv2(hx)
        hx = self.pool2(hx2)
        hx3 = self.rebnconv3(hx)
        hx = self.pool3(hx3)
        hx4 = self.rebnconv4(hx)
        hx5 = self.rebnconv5(hx4)
        hx4d = self.rebnconv4d(torch.cat((hx5,hx4),1))
        hx4dup = _upsample_like(hx4d,hx3)
        hx3d = self.rebnconv3d(torch.cat((hx4dup,hx3),1))
        hx3dup = _upsample_like(hx3d,hx2)
        hx2d = self.rebnconv2d(torch.cat((hx3dup,hx2),1))
        hx2dup = _upsample_like(hx2d,hx1)
        hx1d = self.rebnconv1d(torch.cat((hx2dup,hx1),1))
```

```
        return hx1d + hxin
### RSU-4 ###
class RSU4(nn.Module): #UNet04DRES(nn.Module):
    def __init__(self, in_ch=3, mid_ch=12, out_ch=3):
        super(RSU4,self).__init__()
        self.rebnconvin = REBNCONV(in_ch,out_ch,dirate=1)
        self.rebnconv1 = REBNCONV(out_ch,mid_ch,dirate=1)
        self.pool1 = nn.MaxPool2d(2,stride=2,ceil_mode=True)
        self.rebnconv2 = REBNCONV(mid_ch,mid_ch,dirate=1)
        self.pool2 = nn.MaxPool2d(2,stride=2,ceil_mode=True)
        self.rebnconv3 = REBNCONV(mid_ch,mid_ch,dirate=1)
        self.rebnconv4 = REBNCONV(mid_ch,mid_ch,dirate=2)
        self.rebnconv3d = REBNCONV(mid_ch*2,mid_ch,dirate=1)
        self.rebnconv2d = REBNCONV(mid_ch*2,mid_ch,dirate=1)
        self.rebnconv1d = REBNCONV(mid_ch*2,out_ch,dirate=1)
    def forward(self,x):
        hx = x
        hxin = self.rebnconvin(hx)
        hx1 = self.rebnconv1(hxin)
        hx = self.pool1(hx1)
        hx2 = self.rebnconv2(hx)
        hx = self.pool2(hx2)
        hx3 = self.rebnconv3(hx)
        hx4 = self.rebnconv4(hx3)
        hx3d = self.rebnconv3d(torch.cat((hx4,hx3),1))
        hx3dup = _upsample_like(hx3d,hx2)
        hx2d = self.rebnconv2d(torch.cat((hx3dup,hx2),1))
        hx2dup = _upsample_like(hx2d,hx1)
        hx1d = self.rebnconv1d(torch.cat((hx2dup,hx1),1))
        return hx1d + hxin
### RSU-4F ###
class RSU4F(nn.Module): #UNet04FRES(nn.Module):
    def __init__(self, in_ch=3, mid_ch=12, out_ch=3):
        super(RSU4F,self).__init__()
        self.rebnconvin = REBNCONV(in_ch,out_ch,dirate=1)
        self.rebnconv1 = REBNCONV(out_ch,mid_ch,dirate=1)
        self.rebnconv2 = REBNCONV(mid_ch,mid_ch,dirate=2)
```

```
        self.rebnconv3 = REBNCONV(mid_ch,mid_ch,dirate=4)
        self.rebnconv4 = REBNCONV(mid_ch,mid_ch,dirate=8)
        self.rebnconv3d = REBNCONV(mid_ch*2,mid_ch,dirate=4)
        self.rebnconv2d = REBNCONV(mid_ch*2,mid_ch,dirate=2)
        self.rebnconv1d = REBNCONV(mid_ch*2,out_ch,dirate=1)
    def forward(self,x):
        hx = x
        hxin = self.rebnconvin(hx)
        hx1 = self.rebnconv1(hxin)
        hx2 = self.rebnconv2(hx1)
        hx3 = self.rebnconv3(hx2)
        hx4 = self.rebnconv4(hx3)
        hx3d = self.rebnconv3d(torch.cat((hx4,hx3),1))
        hx2d = self.rebnconv2d(torch.cat((hx3d,hx2),1))
        hx1d = self.rebnconv1d(torch.cat((hx2d,hx1),1))
        return hx1d + hxin
##### U^2-Net ####
class U2NET(nn.Module):
    def __init__(self,in_ch=3,out_ch=1):
        super(U2NET,self).__init__()
        self.stage1 = RSU7(in_ch,32,64)
        self.pool12 = nn.MaxPool2d(2,stride=2,ceil_mode=True)
        self.stage2 = RSU6(64,32,128)
        self.pool23 = nn.MaxPool2d(2,stride=2,ceil_mode=True)
        self.stage3 = RSU5(128,64,256)
        self.pool34 = nn.MaxPool2d(2,stride=2,ceil_mode=True)
        self.stage4 = RSU4(256,128,512)
        self.pool45 = nn.MaxPool2d(2,stride=2,ceil_mode=True)
        self.stage5 = RSU4F(512,256,512)
        self.pool56 = nn.MaxPool2d(2,stride=2,ceil_mode=True)
        self.stage6 = RSU4F(512,256,512)
        # decoder
        self.stage5d = RSU4F(1024,256,512)
        self.stage4d = RSU4(1024,128,256)
        self.stage3d = RSU5(512,64,128)
        self.stage2d = RSU6(256,32,64)
        self.stage1d = RSU7(128,16,64)
```

```
        self.side1 = nn.Conv2d(64,out_ch,3,padding=1)
        self.side2 = nn.Conv2d(64,out_ch,3,padding=1)
        self.side3 = nn.Conv2d(128,out_ch,3,padding=1)
        self.side4 = nn.Conv2d(256,out_ch,3,padding=1)
        self.side5 = nn.Conv2d(512,out_ch,3,padding=1)
        self.side6 = nn.Conv2d(512,out_ch,3,padding=1)
        self.outconv = nn.Conv2d(6*out_ch,out_ch,1)
    def forward(self,x):
        hx = x
        #stage 1
        hx1 = self.stage1(hx)
        hx = self.pool12(hx1)
        #stage 2
        hx2 = self.stage2(hx)
        hx = self.pool23(hx2)
        #stage 3
        hx3 = self.stage3(hx)
        hx = self.pool34(hx3)
        #stage 4
        hx4 = self.stage4(hx)
        hx = self.pool45(hx4)
        #stage 5
        hx5 = self.stage5(hx)
        hx = self.pool56(hx5)
        #stage 6
        hx6 = self.stage6(hx)
        hx6up = _upsample_like(hx6,hx5)
        #-------------------- decoder --------------------
        hx5d = self.stage5d(torch.cat((hx6up,hx5),1))
        hx5dup = _upsample_like(hx5d,hx4)
        hx4d = self.stage4d(torch.cat((hx5dup,hx4),1))
        hx4dup = _upsample_like(hx4d,hx3)
        hx3d = self.stage3d(torch.cat((hx4dup,hx3),1))
        hx3dup = _upsample_like(hx3d,hx2)
        hx2d = self.stage2d(torch.cat((hx3dup,hx2),1))
        hx2dup = _upsample_like(hx2d,hx1)
        hx1d = self.stage1d(torch.cat((hx2dup,hx1),1))
```

```
        #side output
        d1 = self.side1(hx1d)
        d2 = self.side2(hx2d)
        d2 = _upsample_like(d2,d1)
        d3 = self.side3(hx3d)
        d3 = _upsample_like(d3,d1)
        d4 = self.side4(hx4d)
        d4 = _upsample_like(d4,d1)
        d5 = self.side5(hx5d)
        d5 = _upsample_like(d5,d1)
        d6 = self.side6(hx6)
        d6 = _upsample_like(d6,d1)
        d0 = self.outconv(torch.cat((d1,d2,d3,d4,d5,d6),1))
        return (torch.sigmoid(d0), torch.sigmoid(d1),
torch.sigmoid(d2), torch.sigmoid(d3), torch.sigmoid(d4),
torch.sigmoid(d5), torch.sigmoid(d6))
```

6.3.2 u2net_SegmentFolders.py

This module contains the code to convert each image present in the raw data into segmented data in the output folder.

```
#for each directory in the image directory do corresponding
outcome in OUTPUT folder
import os
import gc
import glob
import shutil
import torch
import cv2
import argparse
import parser
import numpy as np
from PIL import Image
from pathlib import Path
```

```
from skimage import io, transform
from torch.autograd import Variable
from torchvision import transforms
from torch.utils.data import Dataset, DataLoader
from u2net_util import U2NetPrediction,LeafDataset,RescaleT
torch.cuda.empty_cache()
parser = argparse.ArgumentParser(description='Process
apply_mask')
parser.add_argument('--apply_mask')
parser.add_argument('--batch_size', type=int,
nargs='+',help='an integer for the accumulator')
parser.add_argument('--output_size', type=int,
nargs='+',help='output masked image size')
args = vars(parser.parse_args())
apply_mask = args['apply_mask']
batch_size = args['batch_size'][0]
output_size = args['output_size'][0]
mask_interpolation  = cv2.INTER_CUBIC
# output of u2net is 320
# if arg parameter outputsize < 320 we need to enlarge image
so we use cv2.INTER_CUBIC
if output_size:
  if output_size < 320:
    mask_interpolation  = cv2.INTER_CUBIC
  else:
    mask_interpolation  = cv2.INTER_AREA
if not batch_size:
  print("Auto assign batch_size = 1")
  batch_size = 1
def save_output(image_location,mask,segmented_dir,apply_mask):
    global mask_interpolation
    mask = mask.squeeze()
    mask_np = mask.cpu().data.numpy()
    if batch_size == 1:
      image_location = '/'.join(image_location)
    else:
      image_location = image_location
```

```
    image_name = os.path.basename(image_location)
    mask_np = mask_np*255
    image = cv2.imread(image_location)
    if not output_size:
      mask_image =
cv2.resize(mask_np,(image.shape[1],image.shape[0]),interpolati
on = mask_interpolation)
    else:
      if image.shape[1] < output_size:
        image =
cv2.resize(image,(output_size,output_size),interpolation =
cv2.INTER_CUBIC)
      else:
        image =
cv2.resize(image,(output_size,output_size),interpolation =
cv2.INTER_AREA)
      mask_image =
cv2.resize(mask_np,(output_size,output_size),interpolation =
mask_interpolation)
    mask_image = mask_image.astype(np.uint8)
    if apply_mask:
      masked_image = cv2.bitwise_and(image,image,mask =
mask_image)
    else:
      masked_image = mask_image

cv2.imwrite(os.path.join(segmented_dir,"masked_"+image_name),m
asked_image)
    del masked_image
    gc.collect()
semanticSegmenter = U2NetPrediction()
root_dir = 'data'
output_root_dir = 'output'
skipfolder = {"cardamom_dataset":"Blight1000"}  # already
segmented
if not apply_mask:
  output_root_dir = 'output_mask'
```

```
if output_size:
  output_root_dir = output_root_dir+'/'+str(output_size)
for datasetSplit in os.listdir(root_dir):
  print("Segmenting:-",datasetSplit)
  for dataset_dir in
os.listdir(os.path.join(root_dir,datasetSplit)):
    for labeled_image_dir in
os.listdir(os.path.join(root_dir,datasetSplit,dataset_dir)):
      image_dir =
os.path.join(root_dir,datasetSplit,dataset_dir,labeled_image_d
ir)
      segmented_dir =
os.path.join(output_root_dir,datasetSplit,dataset_dir,labeled_
image_dir)
      try:
        if skipfolder[datasetSplit] == labeled_image_dir:
          if os.path.exists(segmented_dir):
            shutil.rmtree(segmented_dir, ignore_errors=False,
onerror=None)
          shutil.copytree(image_dir, segmented_dir)
          # need to read all the images in the segmented_dir
and convert it to
          # argument parameter output_size shape
          if output_size:
            for file1 in os.listdir(segmented_dir):
              segmented_dir_img = segmented_dir+"/"+file1
              img = cv2.imread(segmented_dir_img)
              if img.shape[1] < output_size:
                temp_image_interpolation = cv2.INTER_CUBIC
              else:
                temp_image_interpolation = cv2.INTER_AREA
              img =
cv2.resize(img,(output_size,output_size),interpolation =
temp_image_interpolation)
              cv2.imwrite(segmented_dir_img,img)
              del img,segmented_dir_img
          print("Copied Already Segmented to
```

```
output",image_dir)
          continue
      except:
        pass
      Path(segmented_dir).mkdir(parents=True, exist_ok=True)
      img_name_list = glob.glob(image_dir + os.sep + '*')
      leaf_dataset = LeafDataset(imagelocationlist =
img_name_list,

transform=transforms.Compose([RescaleT(320),

transforms.ToTensor()]))
      leaf_dataloader = DataLoader(leaf_dataset,
                                   batch_size=batch_size,
                                   shuffle=False,
                                   num_workers=2)
      for image_data in leaf_dataloader:
        image = image_data['image']
        imagelocation = image_data['imagelocation']
        image = image.type(torch.FloatTensor)
        if torch.cuda.is_available():
              inputs_test = Variable(image.cuda())
        else:
              inputs_test = Variable(image)
        s1,s2,s3,s4,s5,s6,s7=
semanticSegmenter.u2net(inputs_test)
        #since we included batchsize we need to iterate within
batch
        del image
        gc.collect()
        torch.cuda.empty_cache()
        if batch_size !=1:
          for i in range(s1.shape[0]):
            mask = s1[i,0,:,:]
            mask = semanticSegmenter.normPRED(mask)

save_output(imagelocation[i],mask,segmented_dir,apply_mask)
```

```
        else:
          mask = s1[:,0,:,:]
          mask = semanticSegmenter.normPRED(mask)

save_output(imagelocation,mask,segmented_dir,apply_mask)
        del imagelocation,s1,s2,s3,s4,s5,s6,s7
        gc.collect()
      del leaf_dataloader,leaf_dataset
```

6.3.3 u2net_util.py

In the process of segmentation, This module contains utility functions for interlinking the input and output folders.

```
import gc
import os
import glob
from PIL import Image
from pathlib import Path
from skimage import io, transform
import cv2
import torch
import numpy as np
from models.u2net import U2NET
from torchvision import transforms
from torch.autograd import Variable
from torch.utils.data import Dataset, DataLoader
class U2NetPrediction:
  model_name = "u2net"
  model_dir = os.path.join(os.getcwd(), 'saved_models',
model_name, model_name + '.pth')
  u2net = U2NET(3,1)
  initialized = False
  def __init__(self,transform = None):
    if U2NetPrediction.initialized == False:
```

```
      if torch.cuda.is_available():

U2NetPrediction.u2net.load_state_dict(torch.load(U2NetPredicti
on.model_dir))
           U2NetPrediction.u2net.cuda()
      else:

U2NetPrediction.u2net.load_state_dict(torch.load(U2NetPredicti
on.model_dir,

map_location='cpu'))
      U2NetPrediction.initialized = True
    else:
      pass
    U2NetPrediction.u2net.eval()
    self.transform = transform
    self.prediction = None
  # normalize the predicted SOD probability map
  def normPRED(self,d):
    ma = torch.max(d)
    mi = torch.min(d)
    dn = (d-mi)/(ma-mi)
    del ma,mi
    gc.collect()
    return dn
  def semanticSegmentation(self,image =
None,apply_mask=False):
    height,width = image.shape[:2]
    original = image.copy()
    # apply the trainsformaton to image if transformation
present
    if self.transform:
      image = self.transform(image)
    image = image.type(torch.FloatTensor)
    image = image.unsqueeze(0)
    if torch.cuda.is_available():
      image = Variable(image.cuda())
```

```
      s1,s2,s3,s4,s5,s6,s7= U2NetPrediction.u2net(image)
    else:
      image = Variable(image)
      s1,s2,s3,s4,s5,s6,s7= U2NetPrediction.u2net(image)
    #seeing the result of each will give idea on how to
superimpose and generated mask
    self.S = (s1,s2,s3,s4,s5,s6,s7)
    mask = s1[:,0,:,:]
    mask = self.normPRED(mask)
    mask = mask.squeeze()
    mask_np = mask.cpu().data.numpy()
    mask_np = mask_np*255
    mask_image =
cv2.resize(mask_np,(width,height),interpolation =
cv2.INTER_AREA)
    mask_image = mask_image.astype(np.uint8)
    if apply_mask:
      masked_image =  cv2.bitwise_and(original,original,mask =
mask_image)
    else:
      masked_image = mask_image
    # return only the salicency map to create the mask alone
    # rest availableinthe d1,d2,d3 etc variables
    return masked_image
class RescaleT(object):
    def __init__(self,output_size):
        assert isinstance(output_size,(int,tuple))
        self.output_size = output_size
    def __call__(self,image):
        h, w = image.shape[:2]
        new_h, new_w = self.output_size,self.output_size
        new_h, new_w = int(new_h), int(new_w)
        image =
transform.resize(image,(new_h,new_w),mode='constant')
        return image
class LeafDataset(Dataset):
    """Plant Leaf dataset Generator"""
```

```
    def __init__(self, imagelocationlist, transform=None):
        """
        Args:
        imagelocationlist:- for getting list of images with
their location
        transform:- for applying transformation to image
        """
        self.imagelocationlist =imagelocationlist
        self.transform = transform
    def __len__(self):
        return len(self.imagelocationlist)
    def __getitem__(self, idx):
        if torch.is_tensor(idx):
            idx = idx.tolist()
        image = io.imread(self.imagelocationlist[idx])
        imagelocation = self.imagelocationlist[idx]
        if self.transform:
          image = self.transform(image)
        return {'image':image,'imagelocation':imagelocation}
```

6.4 Background Removal

In this project, U^2 Net approach is used for background removal.

6.4.1 MasterU2net.ipnyb

This is the file that need to be executed for segmenting whole raw dataset into output folder all at a time.

```
from google.colab import drive
drive.mount('/content/drive')
%cd /content/drive/Mydrive/Complete Project
!dir
!python -W ignore u2net_SegmentFolders.py –apply_mask True –
batch_size 4 –output_size 224
```

```
%matplotlib inline
import cv2
import torch
import u2net_util
import numpy as np
from PIL import Image
from skimage import io
import matplotlib.pyplot as plt
from torchvision import transforms
from u2net_util import U2NetPrediction, RescaleT
test_imageDir  =  'Healthy (139).jpg'
semanticSegmenter.transform = transforms.Compose([RescaleT(320
),transforms.ToTensor()])
image = io.imread(test_imageDir)
height,width = image.shape[:2]
mask = semanticSegmenter.semanticSegmentation(image = image,ap
ply_mask=False)
masked = semanticSegmenter.semanticSegmentation(image = image,
apply_mask=True)
S = semanticSegmenter.S
original = cv2.resize(image,(width,height))
mask = cv2.resize(mask,(width,height))
mask = cv2.cvtColor(mask,cv2.COLOR_GRAY2RGB)
masked = cv2.resize(masked,(width,height))
# s(1)mask and masked
temp = S[6]
temp = semanticSegmenter.normPRED(temp[:,0,:,:]).squeeze().cpu
().data.numpy() * 255
mask_s6 = cv2.resize(temp,(width,height)).astype(np.uint8)
masked_image_s6 =  cv2.bitwise_and(original,original,mask = ma
sk_s6)
fig = plt.figure(figsize=(20, 10),dpi=100)
rows = 2
columns = 3
fig.add_subplot(rows, columns, 1)
plt.imshow(image)
plt.axis('off')
```

```
plt.title("original")
fig.add_subplot(rows, columns, 2)
plt.imshow(mask)
plt.axis('off')
plt.title("mask(s0)")
fig.add_subplot(rows, columns, 3)
plt.imshow(masked)
plt.axis('off’)
plt.title("masked(s0)")
cv2.imwrite("sample_masked(s0).jpg", masked)
cv2.imwrite("sample_masked(s6).jpg", masked_image_s6)
fig.add_subplot(rows, columns, 4)
plt.imshow(image)
plt.axis('off')
plt.title("original")
mask_s1 = cv2.cvtColor(temp, cv2. COLOR_GRAY2RGB)
fig.add_subplot(rows, columns, 5)
plt.imshow(mask_s1)
plt.axis('off')
plt.title ("mask(s6) ")
fig.add_subplot(rows, columns, 6)
plt.imshow(masked_image_s6)
plt.axis('off')
plt.title("masked_image(s6)")
print("Visualization")
import matplotlib.pyplot as plt
fig = plt.figure(figsize=(20, 10),dpi=100)
rows = 3
columns = 3
j = 1
k = 0
for i in S:
  i = semanticSegmenter.normPRED(i[:,0,:,:]).squeeze().cpu().d
ata.numpy() * 255
  i = cv2.resize(i,(width,height),interpolation = cv2.INTER_AR
EA)
  i = i.astype(np.uint8)
```

```
  i = cv2.resize(i,(width//2,height//2))
  i = cv2.cvtColor(i, cv2. COLOR_GRAY2RGB)
  fig.add_subplot(rows, columns, j)
  plt.imshow(i)
  plt.axis('off')
  plt.title("S("+str(k)+")")
  if j==1:
    j=j+2
  j=j+1
  k=k+1
```

6.5 Training Model

After segmentation, we need to train the dataset using different deep learning models for building a model which can be used for further prediction of diseases and calculating performance metrics.

6.5.1 Resnet152v2.ipynb

This is file that need to be executed for building the model . In our proposed work, We have used ResNet152V2 Model for Model Training and saved the .h5 file for further prediction of diseases.

```
import tensorflow as tf
from tensorflow.keras.layers import Dense, Dropout, Flatten, G
lobalAveragePooling2D
from tensorflow.keras.models import Sequential
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping, ModelChe
ckpoint
from tensorflow.keras.preprocessing.image import ImageDataGene
rator
train_datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=20,
    zoom_range=0.2,
```

```
    shear_range=0.2,
    horizontal_flip=True,
    validation_split=0.2
)
test_datagen = ImageDataGenerator(rescale=1./255)
train_generator = train_datagen.flow_from_directory(
    '/content/drive/MyDrive/Complete Project/output/224/cardam
om_dataset/train',
    target_size=(224, 224),
    batch_size=32,
    class_mode='categorical',
    subset='training',
    shuffle=True
)
validation_generator = train_datagen.flow_from_directory(
    '/content/drive/MyDrive/Complete Project/output/224/cardam
om_dataset/val',
    target_size=(224, 224),
    batch_size=32,
    class_mode='categorical',
    subset='validation',
    shuffle=True)
base_model = tf.keras.applications.ResNet152V2(
    include_top=False,
    weights='imagenet',
    input_shape=(224, 224, 3)
)
for layer in base_model.layers:
    layer.trainable = False
model = Sequential([
    base_model,
    GlobalAveragePooling2D(),
    Dense(256, activation='relu'),
    Dropout(0.3),
    Dense(3, activation='softmax')
])
model.compile(optimizer=Adam(lr=0.001), loss='categorical_cros
```

```
sentropy', metrics=['accuracy', tf.keras.metrics.Precision(),
tf.keras.metrics.Recall()])
es = EarlyStopping(monitor='val_loss', mode='min', verbose=1,
patience=5)
mc = ModelCheckpoint('best_model.h5', monitor='val_accuracy',
mode='max', verbose=1, save_best_only=True)
history = model.fit(
    train_generator,
    steps_per_epoch=train_generator.n // train_generator.batch
_size,
    epochs=20,
    validation_data=validation_generator,
    validation_steps=validation_generator.n // validation_gene
rator.batch_size,
    callbacks=[es, mc]
)
# Load best model
model.load_weights('best_model.h5')
# Evaluate on test set
test_generator = test_datagen.flow_from_directory(
    '/content/drive/MyDrive/Complete Project/output/224/cardam
om_dataset/test',
    target_size=(224, 224),
    batch_size=32,
    class_mode='categorical',
    shuffle=False
)
loss, accuracy, precision, recall = model.evaluate(test_genera
tor, verbose=0)
f1_score = 2 * (precision * recall) / (precision + recall)
print('Test set performance:')
print('Loss:', loss)
print('Accuracy:', accuracy)
print('Precision:', precision)
print('Recall:', recall)
print('F1-score:', f1_score)
# Save model
```

```
model.save('my_model.h5')
```

6.6 Layout Code

This is user interface that we have designed for prediction of diseases in our project.

6.6.1 app.py

This is the file that is to be executed for launching the server side. And it opens the interface that has been designed for prediction.

```
#necessary libraries
import tensorflow as tf
from tensorflow.keras import utils
from PIL import Image
from tensorflow.keras.utils import load_img
from tensorflow.keras.utils import img_to_array
import numpy as np
from flask import Flask, render_template, request, redirect,
url_for
import os
model=tf.keras.models.load_model('D:/MY
project/Project/my_model.h5')
def preprocess_image(img_path):
     img = tf.keras.utils.load_img(img_path, target_size=(224,
224))
    img_array = tf.keras.utils.img_to_array(img)
    img_array = np.expand_dims(img_array, axis=0)
                                                          return
tf.keras.applications.resnet_v2.preprocess_input(img_array)
def predict_disease(img_path):
    img = preprocess_image(img_path)
    prediction = model.predict(img)
    return np.argmax(prediction)
app = Flask(__name__, template_folder='code')
@app.route('/', methods=['GET', 'POST'])
```

```
def home():
    if request.method == 'POST':
        # Get the uploaded file
        file = request.files['file']
        # Save the file to disk
        filename = file.filename
        file.save(filename)
        # Predict the disease
        prediction = predict_disease(filename)
        # Delete the file from disk
        os.remove(filename)
        # Return the prediction
        if prediction == 0:
           return "The leaf is infected with blight"
        elif prediction == 2:
            return "The leaf has a leaf spot"
        else:
            return "The leaf is healthy"
    else:
        return render_template('index.html')
if __name__ == '__main__':
    app.run(debug=True)
```

6.6.2 index.html

This is file designed for user interface.

```
<!DOCTYPE html>
<html>
<head>
    <title>Plant Leaf Disease Detection and
Classification</title>
</head>
<body>
    <h1>Plant Leaf Disease Detection and Classification</h1>
    <form method="POST" enctype="multipart/form-data">
```

```
        <input type="file" name="file" accept="image/*"
required>
        <br><br>
        <input type="submit" value="Submit">
    </form>
</body>
</html>
```

7 Results

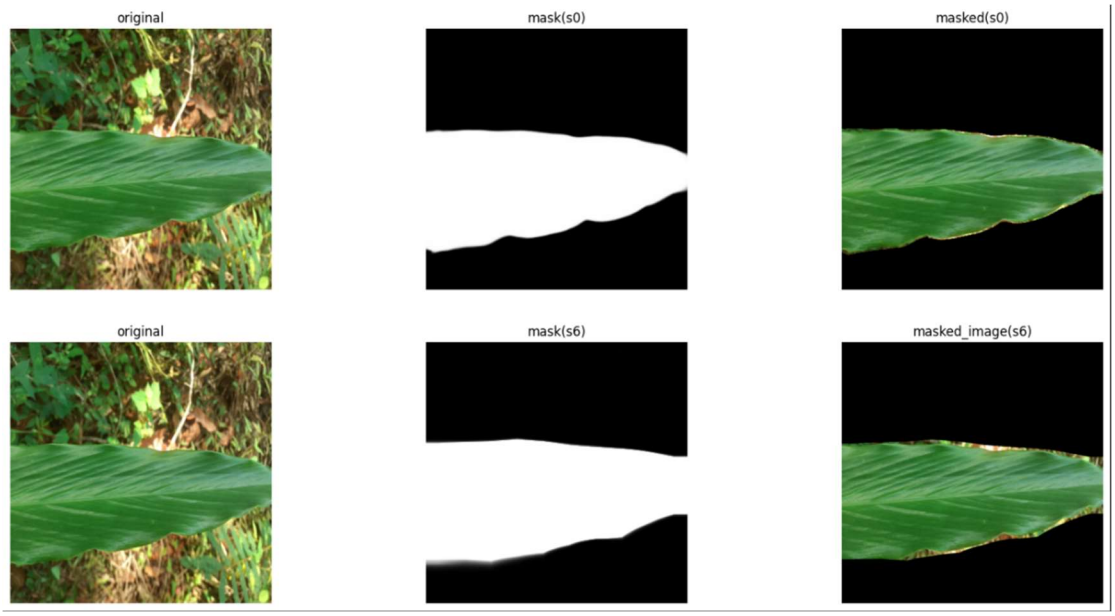

Figure 7.1 Healthy Leaf After Segmentation

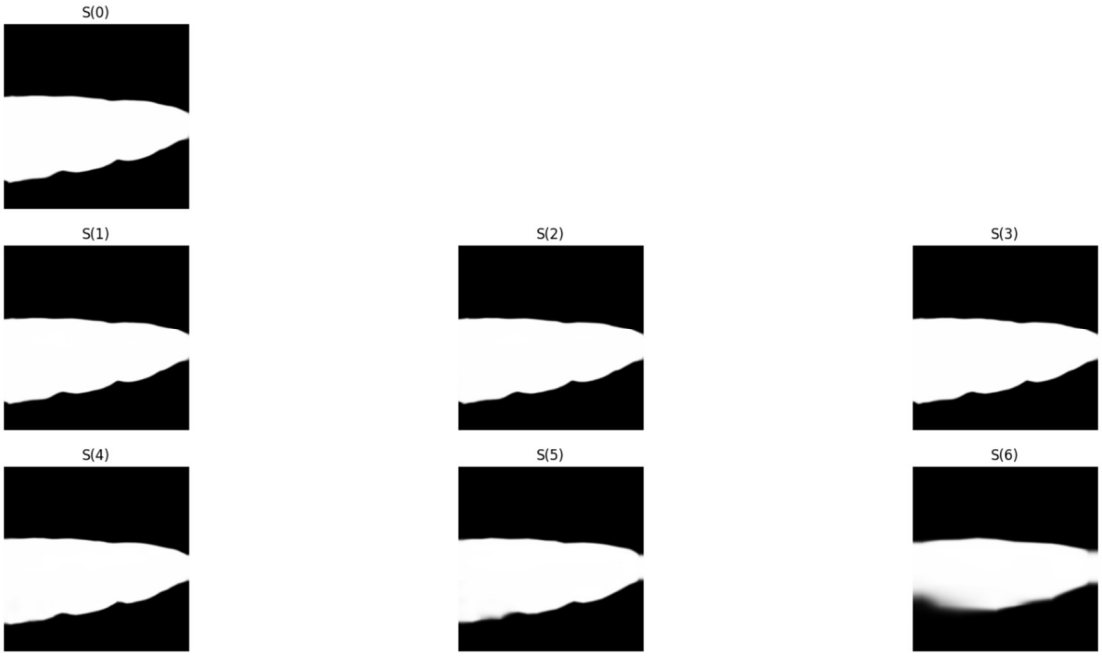

Figure 7.2 Saliency Map of Healthy Leaf

Table 7.1 Performance Metrics of Different Deep Learning Models

Dataset	Cardamom			
	Performance Metrics			
Models	Accuracy	Precision	Recall	F1-score
CNN	0.9592	0.9592	0.9592	0.9655
EfficientNet	0.91	0.92	0.938	0.908
EfficientNetV2-S	0.91	0.92	0.938	0.9049
EfficientNetV2-M	0.843	0.825	0.8198	0.8409
EfficientNetV2-L	0.843	0.8512	0.8314	0.8444
ResNet152V2	0.9844	0.9688	0.9841	0.9844

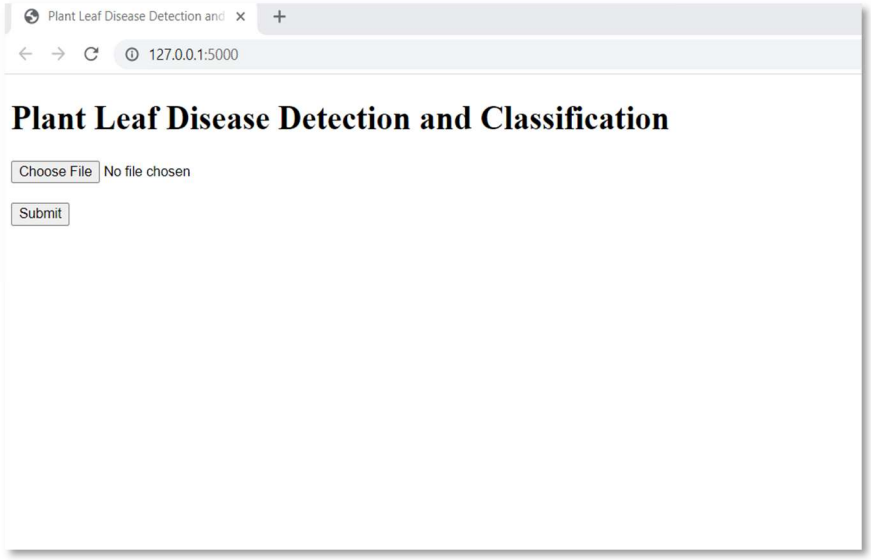

Figure 7.3 Home Page

Figure 7.3 depicts the first look of our front end. We have a text message called " No file chosen" so that a user can understand to click the button. It has a button called "Choose File" which can be used to browse the images on the system's hard disk.

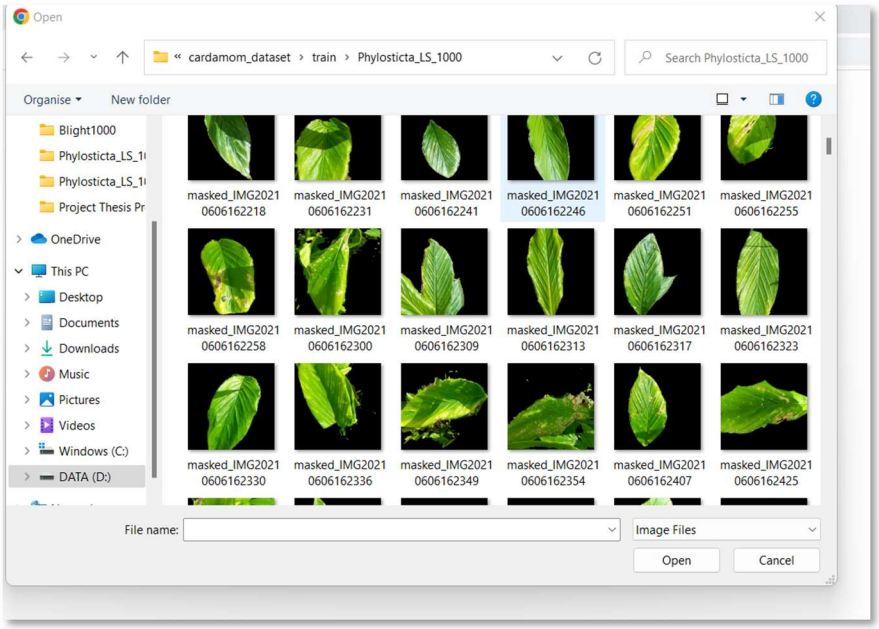

Figure 7.4 Selecting input from dataset

Figure 7.4 shows the popup which appears when user clicks on “Choose File” button. The popup window will be having number of input images to be selected, this action should be confirmed with a double click or an insert button.

Figure 7.5 Image Uploaded

Figure 7.5 shows the selected image from the system directory, there will be a button called “predict” for analyzing the input image to detect the disease of the leaf.

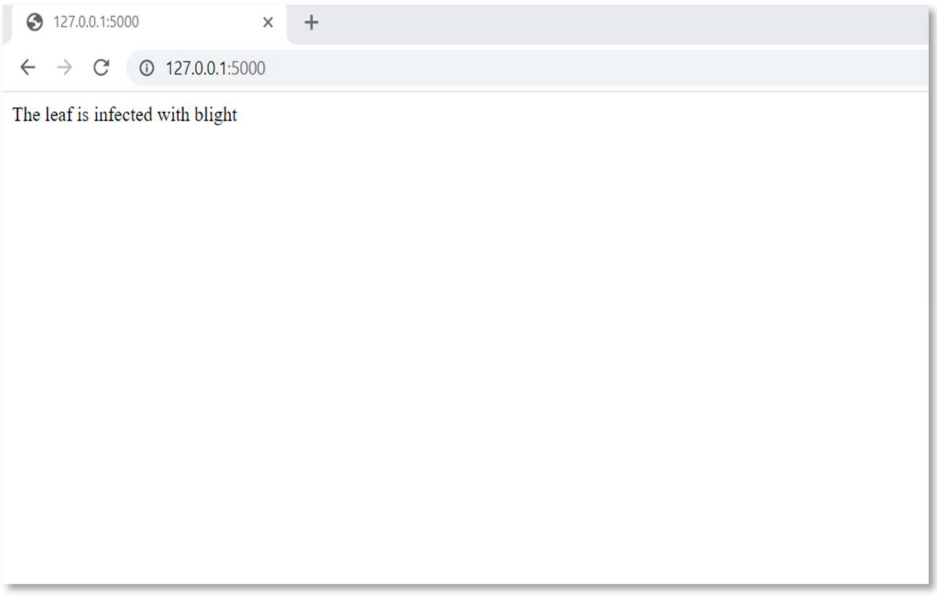

Figure 7.6 Result

Figure 7.6 displays the status of leaf i.e. Blight or Leaf Spot or Healthy, So It displays name of the disease in the next page. Below are the other occurrences of diseases.

8 Conclusion & Future Work

In this project, we propose a new plant leaf disease detection algorithm with consideration of various unnecessary factors such as the background noise and various environmental factors such as angle of capture, lightning, etc. Firstly, U²-Net architecture is utilized to remove the unnecessary and complex background, with which results are obtained without degrading the original image quality. After successful segmentation, ResNet152V2 model was trained and tested. Upon Testing, ResNet152V2 achieved 98.438% detection accuracy. Experimental results demonstrate that our methods consistently outperform the state-of-art-methods on the given data while keeping the real time performance.

In our Future Work, Mobile application can be developed which is handy and easy to use. An extension of this work will focus on automatically estimating the severity of the detected disease. As future enhancement of the project is to develop the open multimedia (Audio/Video) about the diseases and their solution automatically once the disease is detected. And You can also predict different plants and their diseases in a combined way.

9 References

[1] S. Zhang, X. Wu, Z. You and L. Zhang, "Leaf image based cucumber disease recognition using sparse representation classification," *Comput. Electron. Agricult.,* vol. 134, pp. 135-141, Mar. 2017.

[2] V. Singh and A. K. Misra, "Detection of plant leaf diseases using segmentation and soft computing techniques," *Inf. Process. Agricult.,* vol. 4, pp. 41-49, Mar. 2017.

[3] P. Ganesh, K. Volle, T. F. Burks and S. S. Mehta, "Deep Orange: Mask R-CNN based orange detection and segmentation," *IFAC-PapersOnline,* vol. 52, pp. 70-75, 2019.

[4] E. C. Too, L. Yujian, S. Njuki and L. Yingchum, "A comparative study of fine-tuning deep learning models for plant disease identification," *Comput. Electron. Agricult.,* vol. 161, pp. 272-279, Jun. 2019.

[5] Y. Zhong and M. Zhao, "Research on deep learning in apple leaf disease recognition," *Comput. Electron. Agricult.,* vol. 168, Jan. 2020.

[6] C.-H. Yeh, M.-H. Lin, P.-C. Chang and L.-W. Kang, "Enhanced Visual attention-guided deep neural networks for image classification," *IEEE Access,* vol. 8, pp. 163449-163457, 2020.

[7] L. M. Tassis, J. E. Tozzi de souza and R. A. Krohling, "A deep learning approach combining instance and semantic segmentation to identify diseases and pests of coffee leaves from in-field images," *Comput. Electron. Agricult.,* vol. 186, Jul. 2021.

[8] J. Shin, Y. K. Chang, B. Heung, T. Nguyen-Quang, G. W. Price and A. Al-Mallahi, "A deep learning approach for RGB image-based powdery mildew disease detection on strawberry leaves," *Comput. Electron. Agricult.,* vol. 183, Apr. 2021.

[9] P. Singh, A. Verma and J. S. R. Alex, "Disease and pest infection detection in coconut tree through deep learning techniques," *Comput. Electron. Agricult.,* vol.

182, Mar. 2021.

[10] C. K. Sunil, C. D. Jaidhar and P. Nagamma, "Cardamom Plant Disease Detection Approach Using EfficientNetV2," *IEEE Access,* vol. 10, Dec. 2021.