TRIBHUVAN UNIVERSITY
INSTITUTE OF ENGINEERING

Khwopa College Of Engineering
Libali, Bhaktapur
Department of Computer Engineering

A FINAL REPORT ON

RICE LEAF DISEASE DETECTION USING CNN-LSTM MODEL

Submitted in partial fulfillment of the requirements for the degree

BACHELOR OF COMPUTER ENGINEERING

Submitted by

Anuj Gaida	KCE076BCT007
Bikesh Manandhar	KCE076BCT014
Rahul Khatri	KCE076BCT029
Safal Raj Manandhar	KCE076BCT033

Under the Supervision of
Er.Dinesh Ghemosu
Department of Computer Engineering

Khwopa College of Engineering
Libali, Bhaktapur
2022-23

Certificate of Approval

The undersigned certifies that the minor project entitled **"Rice Leaf Disease Detection using CNN-LSTM model"** submitted by Anuj Gaida (KCE076BCT007), Bikesh Manandhar (KCE076BCT014), Rahul Khatri (KCE-076BCT029), and Safal Raj Manandhar (KCE076BCT033) to the Department of Computer Engineering in partial fulfillment of the requirement for the degree of Bachelor of Engineering in Computer Engineering. The project was carried out under special supervision and within the time frame prescribed by the syllabus.

.....
Er. Prakash Chandra Prasad
External Examiner
Assistant Professor,
Department of Electronic and
Computer Engineering,
IOE, Pulchowk Campus

.....
Er. Dinesh Ghemosu
Project Supervisor
Sr.Lecturer
Department of Electrical Engineering
Khwopa College of Engineering

.....
Er. Dinesh Gothe
Head of Department
Department of Computer Engineering
Khwopa College of Engineering

Copyright

Acknowledgement

We take this opportunity to express our deepest and most sincere gratitude to our supervisor Er. Dinesh Ghemosu, for his insightful advice, motivating suggestions, invaluable guidance, help, and support in the successful completion of this project, and also for his constant encouragement and advice throughout our Bachelor's program.
We express our deep gratitude to Er. Dinesh Gothe, HoD Department of Computer Engineering, Khwopa College of Engineering(KhCE) for his regular support, cooperation, and coordination. We are deeply indebted to the support of our lecturer Er. Bindu Bhandari. We are thankful for the support of Er. Sajit Pyakurel. We would also like to thank the teaching and non-teaching staff of the Department of Electronics and Communication and Computer Engineering, KhCE for their invaluable help and support for the project. We are also grateful to all our classmates for their help, encouragement, and suggestions.
Finally, yet, more importantly, we would like to express our deep appreciation to Er.Niranjan Bekoju for the guidance of the project

Anuj Gaida	KCE076BCT007
Bikesh Manandhar	KCE076BCT014
Rahul Khatri	KCE076BCT029
Safal Raj Manandhar	KCE076BCT033

Abstract

In recent years, automated image classification techniques have gained popularity in various sectors such as agriculture, medical, e-commerce, facial recognition, etc. In the agricultural sector, these techniques are used for weed identification, fruit categorization, and disease detection in plants and trees. project was developed as a mobile app, with a local server for user-server interaction. All classification activities were performed on the local server using two different neural networks and a segmentation process. The first neural network was used for binary classification to determine whether the inputted image was of a rice leaf or not. The second neural network was a combination of CNNs and LSTMs, used for the classification of the type of disease by utilizing the segmented image. In the segmentation section, the diseased area in the rice leaf was segmented out and forwarded back to the user in the mobile application along with the result, overview, and preventive measures for the specific diseased type. The final accuracy of the binary classification model was 97.86%, while the final accuracy of the disease classification model was 95.50%.

Keywords: *Convolution Neural Network,Deep Learning,Image Segmentation, Long Short Term Memory*

Contents

List of Tables

List of Figures

List of Symbols and Abbreviation

CNN	Convolution neural networks
Colab	Colaboratory
DCGAN	Deep Convolutional Generative Adversarial Networks
LSTM	Long short-term memory
MSAR	Maximum Segmentation Accuracy Rate
PCA	Principal component analysis
RGB	Red Green Blue
RNN	Recurrent neural Networks
SVM	Support Vector Machine
VGG	Visual Geometry Group

Chapter 1

Introduction

1.1 Background Introduction

Rice is one of the leading food crops around the world. Over 90% of the world's rice is produced and consumed in the Asia-Pacific Region. The majority of Asian countries, including Nepal, have a significant demand for rice. [agrifood-economics]. Because of the rising demand for rice, rice farming is now crucial to the agriculture industry and to people's livelihoods. However, rice plants are prone to a number of diseases. Some of the most frequently observed rice diseases are hispa, leaf blast, bacterial leaf blight, brown spot, and leaf smut.

The three rice diseases that are taken for this project are Brown spot, Hispa, and Leaf blast. They have their characteristic patterns and shapes. The features of the diseases is described and illustrated below [www.knowledgebank.irri.org]:

- Brownspot: Circular to oval with a light brown to gray center, surrounded by a reddish brown margin.
- Hispa: White streaks parallel to the midrib.
- Leafblast: Elliptical or spindle-shaped and whitish centers with red to brownish or necrotic border.

Brownspot Image | HispFor a Image | Leafblast Image

Figure 1.1: Sample Images of Diseased Rice leaves

For efficient and timely disease control, early disease detection is crucial. As, microscopic insects can quickly spread disease and ruin a whole field, an incorrect or delayed diagnosis could lead to disease transmission. These diseases can also be manually classified, which can be unnecessarily laborious, expensive, and time-consuming. There are various benefits of speeding up this process, including lower prices and less work. Different classification systems for plant diseases are developed to assist non-expert and non-botanist users in automatically identifying plant diseases. As a result, correct identification can save money and time.

So, here we have come up with a mobile application to help farmers and concerned parties to recognize the disease in time and make early preparation for its prevention. In this project, we use the image classification method, to detect and recognize the rice disease of 3 different class and provide details to the users about its cause, preventive measures, and solution.

1.2 Motivation

Motivated by the success of machine learning [1] [2] and deep learning [3] in automatic detection of diseases and pests in trees and plants. CNNs are frequently used for automatic crop disease detection. Additionally, a great deal of research on the detection of rice illness using trained CNN models has been published. This report proposes such an approach that makes disease detection and classification of the three mentioned rice diseases(Brownspot, Hispa, Leafblast). The novelty of the report lies in the detection of rice leaf diseases using deep learning approaches.

1.3 Problem Statement

Traditionally, agricultural diseases are manually detected by observing irregularity in plants, then the irregularity is classified as a disease by experts who propose appropriate treatments. When considering the vast amount of farmland, this set of responsibilities becomes incredibly difficult. Additionally, it requires more time and labor. Since machine learning models were applied to agriculture, the entire situation was revolutionized. Disease detection was much easier and cost and time efficient.

Numerous AI models have been employed for the diagnosis of plant diseases and have been successful in applications including plant identification and disease detection. Despite this, these models have the drawback of not being able to extract several important visual attributes from the input image [4]. However, the accuracy of these models has been an interest of the field to many researchers. CNN models alone are unable to calculate the dependency and continuity features of the intermediate layer output [4].

[1] `https://www.kaggle.com/datasets/shayanriyaz/riceleafs`

Therefore, the combination of CNN with the RNN model can help to improve the accuracy. Better classification accuracy will result from using the recurrent neural network to link the middle tire features to the final fully-connected network for classification [5]. Thus, we have developed a smartphone application to assist farmers in early illness detection and decision-making. In this study, we employ the image classification method to identify three different types of rice diseases and provide insights into their causes, mitigation strategies, and treatments.

1.4 Objective

The project intends to design a system that assists the farmers and other concerned parties in swiftly identifying the disease on rice leaves and makes their work more accessible, more efficient, and more effective.

The main intention of the project is:

- To integrate deep learning models(CNN and LSTM) that identify the three different diseases (Brownspot, Hispa, Leafblast) on rice leaf and recommend their solutions.
- To develop mobile application.

1.5 Scope and Application

This project can be used by farmers and other concerned parties who may or may not have specialized knowledge to quickly identify the disease in the rice leaf and treat it. This application will offer farmers and other concerned parties a remedy and preventative measures that may aid in reducing the spread of disease.

Chapter 2

Literature Review

When the present work on image classification was assessed, it was found that researchers had given careful consideration to mixing several models.
Islam, M.Z., et al. at 2020. [6] proposed a Combined deep CNN-LSTM network, where 4575 images of pneumonia and normal cases were used. They resized the images to ones with a resolution of 224*224 pixels. The network of 20 layers: 12 convolutional layers, 5 pooling layers, one fully connected layer, and one lstm layer was used to obtain 99.4% accuracy with an AUC score of 99.9%.
Another paper by Rangarajan, A.K., et al. at 2018. [7] used the pre-trained DL and its algorithm to accomplish the categorization of disease in tomato plants. The authors used AlexNet and VGG16 pre-trained architectures on PlantVillage consisting of 13262 pictures, both of which obtained 97.49% classification accuracy. For this dataset, however, additional transfer learning models must be assessed.
Karthik, R., et al. at 2020. [8],proposed embedded residual cnn where two distinct deep architectures are shown for identifying the type of infection in tomato leaves. Relative learning is used in the first architecture to identify important elements for categorization. The attention mechanism is added to the remaining deep network in the second architecture. The suggested work used the attention mechanism to take advantage of the characteristics that the CNN learnt at different levels of the processing hierarchy, and it was able to obtain an overall accuracy of 98% on the validation sets in the 5-fold cross-validation.
Hu, G., et al. at 2019 [9], in order to promptly prevent and control tea leaf diseases, they described a low-shot learning strategy for disease identification. The support vector machine (SVM) approach is used to segment disease spots on tea leaf disease images by extracting the color and texture attributes. The upgraded conditional deep convolutional generative adversarial networks (C-DCGAN) for data augmentation provide fresh training samples from segmented disease spot images, which are then fed into the VGG16 deep learning model for illness identification. The average identification accuracy of the proposed method reaches 90%, far exceeding classical low-shot learning methods.
In 2018 by Islam, T., et al. [10] A model was created to categorize the disease based solely on the extracted percentage of the RGB value of the damaged area of rice leaf using image processing. To finally classify the diseases into

three disease classes Bacterial leaf blight, Rice blast, and Brown spot the RGB percentages were fed into a Naive Bayes classifier. The model's disease classification accuracy is over 89%.
In 2022, Garg, D., et al. [4], The effectiveness of mixed CNN and RNN models was assessed by identifying pertinent visual features from pictures of diseased apple leaves. This paper suggests combining a specific kind of RNN called LSTM with a pre-trained CNN network. Transfer learning was used to separate the deep features from the Xception and VGG16 pre-trained deep models' several fully connected layers. To help the proposed model be more focused on identifying pertinent information in the input data, the recovered deep features from the CNN layer and RNN layer were concatenated and sent into the fully connected layer.
Another paper by Lamba, S., et al. at 2022 [11] proposed a brand-new hybrid model based on neural networks (GCL). Long-short-term memory (LSTM) and convolutional neural networks are combined in GCL to supplement datasets. The dataset is enhanced using GAN, CNN extracts the characteristics, and LSTM categorizes the various paddy diseases. To increase the precision and dependability of the classification model, the GCL model is being examined.
In 2021 Krishnamoorthy, N., et al. [12] utilized InceptionResNetV2 a type of CNN model with transfer learning approach for recognizing diseases in rice leaf images. The parameters of the proposed model is optimized for the classification task and obtained a good accuracy of 95.67%.
Al-Amin, M., et al. at 2019, December [13] developed a model that can predict rice diseases so that farmers can take appropriate action. This work presents a CNN based model that provides 97.40% accuracy in predicting various diseases of rice leaves. Using a data set of over 900 images of diseased and healthy leaves and following the technique of 10-fold cross validation(CV), the model was trained to identify 4 common rice diseases.

While the present work on image segmentation was assessed, by Niu, Xiaojing, et al. [14]K-means clustering is suggested as an automated and effective approach in this paper. The color picture is first converted from RGB to Lab color space. Then, in Lab color space, clustering is performed by measuring the exact difference between each pixel and the clustering center. Unlike conventional approaches, our technique does not require manual threshold value setting and is unaffected by the chosen channel. Their findings demonstrate the effectiveness of approach by demonstrating segmentation accuracy rates of more than 90% for three common diseases (powdery mildew, leaf rust, and stripe rust).
Zhou, Y., et al. [15] proposed Backpropagation Neural Network (BPNN) to image segmentation of rice disease spot. Firstly, the combination of different color feature parameters was selected as the input of the BPNN. Secondly, a BPNN with 5 input, 10 hidden neurons and 1 output was constructed to rice blast spots segmentation. Experimental results showed that the converging speed of training the improved BPNN was faster than that of the standard BPNN. To test the accuracy of the algorithm, manually segmented images were compared with those segmented automatically. Results show that the

highest accuracy of 99.8% was found.

S.N	Title	Model used	Accuracy
1	CNN-LSTM model for covid-19 detection [6]	Multi-layer perceptron-LSTM	99.9%
2	Tomato crop disease classification using pre-trained deep learning algorithm [7]	AlexNet and VGG16	97.49%
3	Attention embedded residual CNN for disease detection in tomato leaves [8]	Embedded residual CNN	98%
4	low shot learning method for tea leaf's disease [9]	VGG16	90%
5	Faster Technique on Rice Disease Detection using Image Processing [10]	Naive Bayes classifier	89%
6	Integration of Convolutional Neural Networks and Recurrent Neural Networks for Foliar Disease Classification in Apple Tree [4]	InceptionV3-LSTM model	99.8%
7	A novel GCL hybrid classification model for paddy disease [11]	GCL model	97%
8	Rice leaf diseases prediction using deep neural networks with transfer learning [12]	InceptionResNetv2	95.65%
9	Prediction of Rice Disease from Leaves using Deep Convolution Neural Network towards a Digital Agricultural System [13]	KNN,Decision tree, Logistic regression, NaiveBayes	97.91%

Table 2.1: Review Matrix for Classification

SN	Title	Method	MSAR
1	Image Segmentation Algorithm for Disease Detection of Wheat Leaves [14]	k-mean clustering in lab	98%
1	Segmentation of Rice Disease Spots Based on Improved BPNN [15]	BPNN	99%

Table 2.2: Review Matrix for Segmentation

Chapter 3

Feasibility Study

The feasibility studies of the project are described below:

3.1 Economic Feasibility

The project requires lots of data for training implemented models, which are taken from Kaggle [1] that are free and easy to access. So, the expenditure depends on computational consumption. The computational consumption for the project can be fed by our own laptops and Google Col-laboratory. So, the project is economically feasible.

3.2 Technical Feasibility

The project is to be trained with lots of labeled data. Preparing the data has its complexity like labeling the data. Creating the project will be challenging due to unstable training in modified models. In terms of availability, both datasets and computation power are available. So, the project is technically feasible.

3.3 Operational Feasibility

The project can be operational just after training from labeled data. After the models have been separately trained, they can be combined together. Then the system should be able to operate to implement given labeled data to distinguish between the types of rice disease. Thus, the project is operationally feasible.

[1] https://www.kaggle.com/datasets/shayanriyaz/riceleafs

Chapter 4

Requirement Analysis

4.1 Software Used

Various python libraries and platform were used for development of our system which are described below;

4.1.1 Python

Python is a high-level interpreted programming language for general-purpose programming created by Guido van Rossum which was released in 1991 [16]. It provides constructs that enable clear programming on small and large scales. The programming language features a dynamic type system and automatic memory management with support for multiple programming paradigms including Object Oriented Imperative, Functional and Procedural. The programming language has a large comprehensive standard library.

4.1.2 Google Colab

Google Colaboratory, or "Google Colab" for short, is a product from Google Research. Colab allows anybody to write and execute arbitrary python code through the browser and is especially well suited to machine learning, data analysis, and education.

4.1.3 Flutter

Flutter is an open-source, cross-platform mobile application development framework created by Google. It uses the Dart programming language and provides a fast, modern, and flexible development environment for building high-quality mobile apps for iOS and Android, as well as for web and desktop. Flutter allows for the creation of beautiful, fast, and responsive user interfaces with smooth animations and easy access to native features.

4.1.4 FastApi

FastAPI is a modern, fast (high-performance) web framework for building APIs with Python 3.7+ based on standard Python-type hints. It provides au-

tomatic documentation, validation, and serialization, making it easy to build APIs that can handle complex use cases, such as real-time data processing, machine learning, and web-socket communication. FastAPI also supports asynchronous programming, is built on top of Starlette for high performance, and has excellent support for modern web features such as OAuth2, WebSockets, and GraphQL.

4.1.5 Postman

Postman is a popular, free, and powerful API development tool used for testing and managing APIs. It allows developers to easily test, develop, and document APIs by providing a user-friendly interface for sending HTTP requests and analyzing responses. With Postman, you can build and send any HTTP request, including GET, POST, PUT, DELETE, and more, and view the response in real-time. It also supports various authentication methods and has features for testing, such as environment and global variables, and collection and folder organization.

4.1.6 Jupyter Lab

JupyterLab is an open-source web-based integrated development environment (IDE) for working with Jupyter notebooks, code, and data. It provides a modern, flexible, and powerful platform for data science, scientific computing, and machine learning, as well as for web development, teaching, and collaboration. JupyterLab supports multiple programming languages including Python, R, Julia, and more, and has features such as code execution, markdown and code cells, data visualization, and file management. It is an evolved version of the Jupyter Notebook and allows for a more integrated experience for working with multiple files, consoles, terminals, and notebooks in a single window.

4.1.7 Trello

Trello is a web-based project management tool that allows individuals or teams to organize and prioritize their tasks using a visual board. It uses cards and boards to represent tasks and projects, and users can move these cards across lists to indicate their progress. Trello also allows for collaboration among team members by allowing them to add comments, attachments, due dates, and labels to the cards. The tool is easy to use, highly customizable, and available on desktop and mobile platforms.

4.1.8 Microsoft Teams

Microsoft Teams is a collaboration platform that allows teams to chat, share files, and hold video conferences. It integrates with other Microsoft products, offers third-party app integration, and provides channels for organizing discussions and files.

4.1.9 Google Drive

Google Drive is a cloud-based storage platform that provides free storage and allows users to create, edit, and collaborate on files online. It offers features like offline access, automatic syncing, and the ability to share files with others.

4.1.10 LaTex

LaTeX is a document preparation system used for writing scientific and technical documents with high-quality output, particularly in mathematics, physics, and computer science. It provides commands for formatting and structuring the document, making it popular in academic and research environments.

4.1.11 StarUML

StarUML is a software modeling tool used for creating and designing UML diagrams. It provides a user-friendly interface and a variety of features for visual modeling, code generation, and documentation. StarUML supports multiple programming languages and is widely used in software engineering and development projects.

4.1.12 Visual Studio Code

VSCode (Visual Studio Code) is a free, open-source code editor developed by Microsoft. It provides a lightweight and customizable interface, with support for multiple programming languages, debugging, and source control integration. VSCode also offers a rich set of extensions and plugins that can be installed to enhance its functionality, making it a popular choice for software developers and programmers.

4.1.13 Microsoft Azure

Microsoft Azure is a cloud computing platform that provides a wide range of services for building, deploying, and managing applications and services through Microsoft-managed data centers. Microsoft provides 100-dollar credit for students. So, we can grab the service to create a virtual machine on azure for the project server deployment.

4.2 Hardware Requirements

Some of the requirements are:

- CPU: Intel core i5(or above) or AMD Ryzen 5
- RAM: 8 GB or above
- Android device: version 5.0 Lollipop or above

4.3 Functional Requirements

It defines what the system should do in order to meet the user's needs or expectations. Functional requirements can be thought of as features that the user detects.

4.3.1 Image Processing

With the help of the system user can easily select region of interest by cropping the image to be inputted which is later used by system for noise reduction and segmentation.

4.3.2 Disease Detection

The system should be able to detect above mentioned three types of disease with high accuracy.

4.3.3 Recommendation

The system should be able to recommend the solution according tho the type of disease detected.

4.3.4 Select Image

The user should be able to select image from their android devices.

4.3.5 Upload Image

The user should be able to upload image to the server for detection.

4.3.6 Display Report

The user should be able to see the disease condition, its symptoms, and the control measures through the report.

4.4 Non-Functional Requirements

Although the system does not require these specifications, they are necessary for our systems to operate more effectively. The following points concentrate on the system's non-functional necessity.

4.4.1 Reliability

The developed model for classification has high accuracy around 95%. Hence, the outcome of the system is reliable.

4.4.2 Maintainability

The project is decomposed into modules. It is simpler to assign tasks, monitor progress, and spot possible issues or blockages when a project is divided into smaller pieces. Decomposition has made it easier to maintain the project.

4.4.3 Performance

The overall computation for the binary classification and disease classification is within 5 seconds and with higher accuracy, we can say the project has better performance.

4.4.4 Portability

The application was built using flutter. Flutter provides flexibility to change platforms without any modification to the actual code. Hence, the project is portable.

Chapter 5

System Design and Architecture

We focused to develop a system that takes an image as its input and then first classifies whether the input image is rice leaf or not then the image is again processed to detect disease and provide solutions along with overview, symptoms, and preventive measures as system output.

5.1 Use Case Diagram

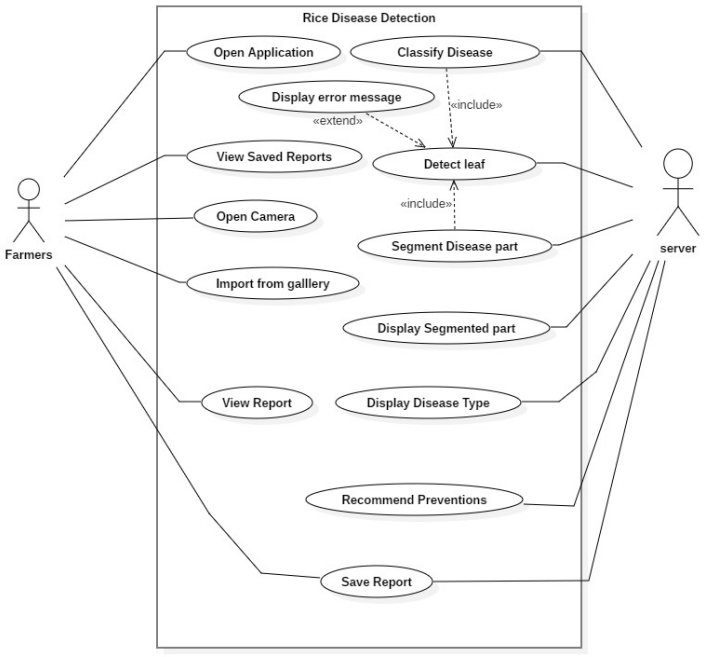


Figure 5.1: System Use Case Diagram

The farmer can open the application. Farmers can open the camera to click an image or select the image from the gallery. The selected image can be forwarded to the server where the disease is classified. To classify the disease, the image must be the image of a rice leaf and further segmented. The server responds with the report with prevention and segmented image. Finally, the report can also be saved.

5.2 Data Flow Diagram

The Data Flow Diagram shows the flow of the data between the subsystem of the classification system. The DFD level-1 of the classification system is shown below:

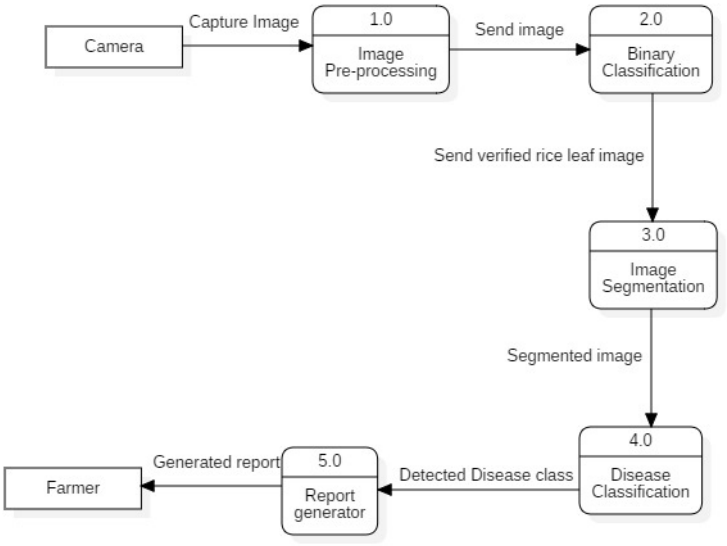


Figure 5.2: Data Flow Diagram Level 1 of Rice Leaf Disease Detection

The image selected is forwarded for image pre-processing before binary classification. Once the image selected is of a rice leaf, the image is then forwarded for image segmentation. The image is again passed for disease classification and the generated report is finally displayed to the farmer through mobile application.

5.3 Sequence Diagram

The Sequence Diagram shows the flow of the process in-between the subsystem. And the state when the subsystem is active and when the subsystem is passive. The Sequence Diagram of the classification system is shown below:

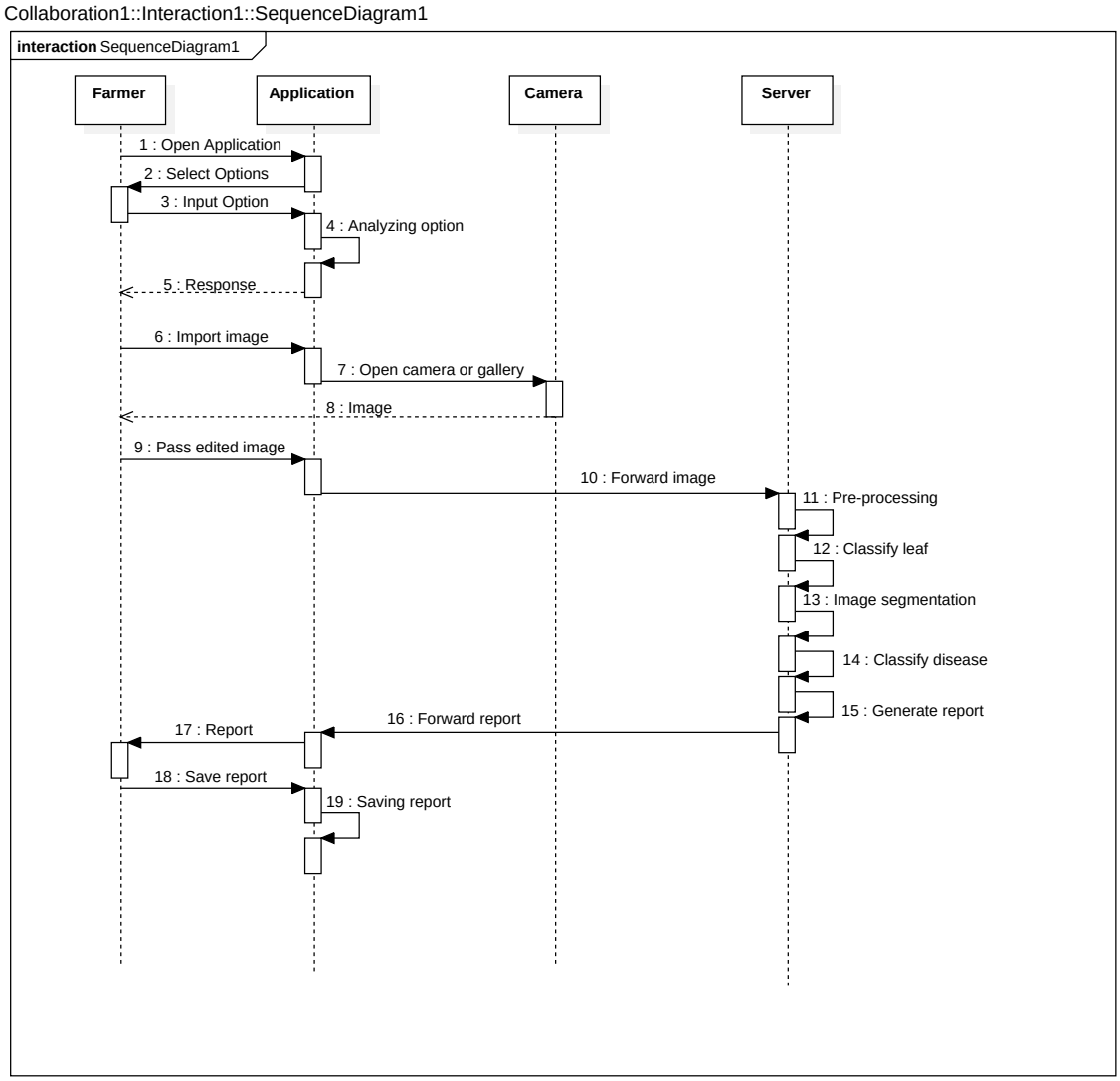


Figure 5.3: Sequence Diagram of Rice Leaf Disease Detection

Farmer opens the program first, then chooses an option(click image, import image, view reports). Then the application analyzes the choice and provides the appropriate services. The program receives images from the user's camera or photo gallery. The image is sent to the server by the program, which pre-processes it before classifying it as a rice leaf. The classified picture is then subjected to image segmentation. And, forwarded to the disease classification model. The server generates a report based on classification and responds to the application with a report that includes the disease type, its diagnosis, prevention advice, treatment options, and the segmented and original picture. The user can save the report after it is displayed by the program.

5.4 System Block Diagram

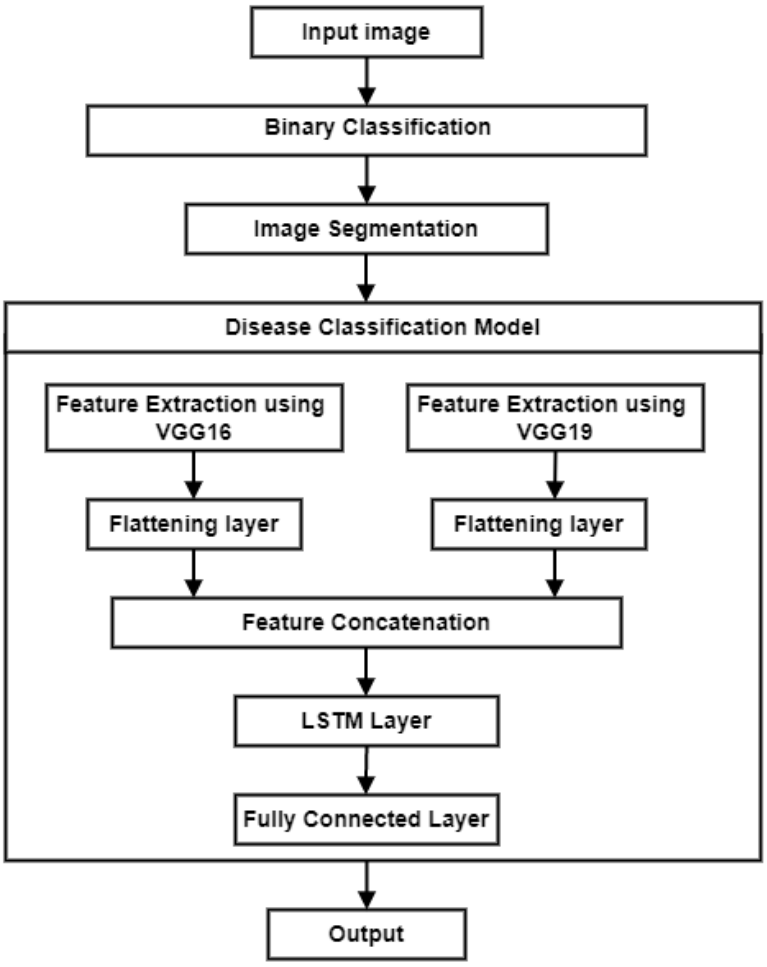


Figure 5.4: System Block Diagram of Rice Leaf Disease Detection

The inputted image is first classified as rice leaf or not using a binary classification model. After verification of the rice leaf the image is segmented, then the segmented image is fed into the multi-class classification model integrated using VGG16, VGG19, and LSTM for final classification.

Chapter 6

Methodology

6.1 Software Development Approach

There are various types of software development models available such as the waterfall model, iterative and incremental development model, V-Model, prototype model, agile model, and many more. Among these, we prefer the Agile model for developing our classification system. This software development approach is based on iterative development, where tasks are fragmented into smaller iterations, or parts do not directly involve long-term planning. Under the Agile model, the SCRUM framework will be applied to the project. There are other frameworks under agile. They are KANBAN, XP, and so on.

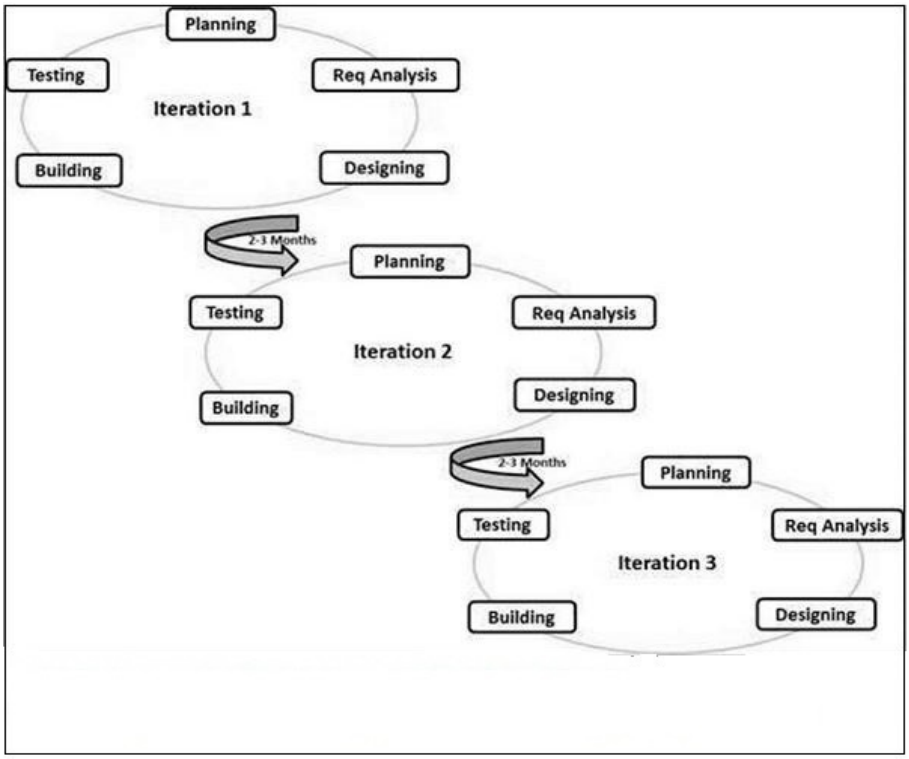


Figure 6.1: Agile Model for Software Development
[1]

[1] https://www.tutorialspoint.com/sdlc/sdlc_agile_model%20htm

6.1.1 Why SCRUM?

SCRUM framework provides the facility for stand-up meetings which are conducted daily (Daily SCRUM) for daily updates among team members. Also, SCRUM encourages the concept of a sprint where the product is presented to the client, in this case, our supervisor. In the course of the sprint review, developments are coded and tested. Teams may accomplish project deliverables more rapidly and effectively with the aid of Scrum.

6.2 Trello as Kanban Board

Trello was used as a kanban board to organize, distribute, manage, and track all the work and tasks assigned to each member of our team. All the work done at the base level as well as documentation was recorded in Trello. The project was managed in Trello in the following sessions:

6.2.1 Project Plan

Planning for further steps was done in detail in this session. All the tasks to be done, their sub-tasks and the deadline for the tasks and subtasks were recorded here. Then the task was linked to the respective iteration session by the assigned member before doing the task

6.2.2 Sprint

The works decided to be done, being done and the ones that were done recently were recorded here.

6.2.3 WIKKI

Utility tasks that are useful and informative but unrelated to the actual project were listed here. For example tutorials and guidelines to use GitHub, Trello, etc.

6.2.4 Iteration1

Everything related to the first iteration of our project was recorded here including testing, reviews, and reports.

6.2.5 Iteration2

Everything related to the second iteration of our project was recorded here including testing, improvements, reviews, and reports.

6.3 Phase Followed

The overall project has been completed in three main phases which are:

- Planning Phase
- Development Phase
- Testing Phase
- Integration

6.3.1 Planning Phase

In the planning phase, necessary work to be done and planning of the overall project was discussed and the decisions were documented for further procedures. Initially, the project was divided into the following parts:

- Data Collection
- Data Preparation
- Binary Classification
- Disease part Segmentation
- Disease Classification
- Recommendation System
- Integrating model to the server
- Mobile App

6.3.2 Development Phase

In this phase, the divided parts were well studied and developed.

6.3.3 Testing Phase

In this phase, the developed models were tested and evaluated separately and necessary optimization was done.

6.3.4 Integration

In this phase, the separately developed parts were integrated and testing was done on the integrated model.

6.4 Task Workflow

Each task of every session was done and recorded in a procedural manner and the codes were recorded in Trello and Google drive. The workflow of a task follows the following steps:

a. Create and assign the task in Trello.

b. Mark the task assigned to To REVIEW in Trello.

6.5 Work Breakdown

A work breakdown structure is given below:

Task	**Anuj**	**Bikesh**	**Rahul**	**Safal**
Perform feasibility test	✓	✓	✓	✓
Study about different rice disease	✓	✗	✓	✗
Data Collection	✓	✓	✓	✓
Study different thresholding technique	✓	✓	✗	✗
Familiarization with ways of implementing selected algorithm	✓	✓	✗	✗
Study about classification model	✗	✗	✓	✓
Select model	✗	✗	✓	✓
Study about CNN's architecture	✓	✓	✓	✓
Familiarization with ways of implementing model	✓	✓	✓	✓
Familiarization with the training of models	✓	✓	✓	✓
Binary Classification model	✓	✗	✓	✗
Study the CNN-LSTM combination	✗	✓	✗	✓
Development of CNN-LSTM classifier one half	✓	✗	✓	✗
Development of CNN-LSTM classifier other half	✗	✓	✗	✓
Study about feature reduction	✓	✓	✗	✗
Evaluation of binary classifier	✓	✗	✓	✗
Evaluation of segmentation	✓	✓	✗	✓
Evaluation of CNN-LSTM classifier	✗	✓	✗	✗
Models comparison for the best selection	✓	✓	✗	✗
UI design	✗	✗	✓	✓
Android app development	✗	✗	✓	✓
Integration of models in server	✗	✗	✗	✓
Documentation	✓	✓	✓	✓

Table 6.1: Work Breakdown Structure

6.6 Data Collection

We collected Rice leaf images from different platforms like kaggle.com,and mendeley.com. The images collected from different platforms are further described in detail below:

6.6.1 Kaggle Dataset

The main dataset used for disease classification was downloaded from kaggle [17] This dataset was originally 9.12 GB containing high-quality images of 3 different diseases and healthy classes. The size of the images is not constant throughout the dataset(min size = 1200*1200, max size = 2163*2163). The overview of the dataset is shown below:

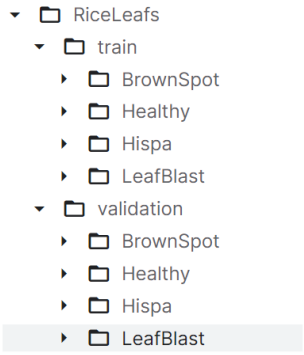


Figure 6.2: Structure of Dataset

Data sets		
SN	**Class**	**Original**
1	BrownSpot	523
2	Healthy	1488
3	Hispa	565
4	LeafBlast	799

Table 6.2: Number of Images in Each Class

The images of non-rice leaves that were required for binary classification were also downloaded from kaggle.com, such as apple leaf [18], potato leaf [19], wheat leaf [20]etc. Finally, This data set contains 5865 images of rice leaves and 5467 images of non-rice leaves. Some of the many non-rice classes are shown in the table below.

SN	Name	Images
1	Apple	419
2	Potato	4072
3	Wheat	407

Table 6.3: Non-Rice Leaf Images

6.6.2 Mendeley Dataset

The additional images required were also collected from Mendeley. The following table shows the class and the number of images collected in each class.

SN	Name	Images
1	Wheat [21]	407
2	Rice [22]	5932

Table 6.4: Mendeley Dataset

6.6.3 Sample Images

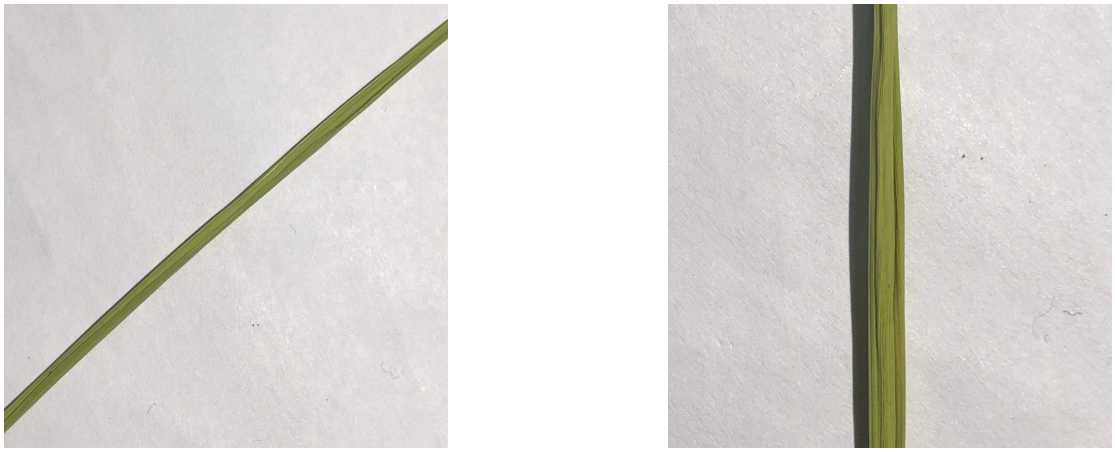


Figure 6.3: Healthy Images

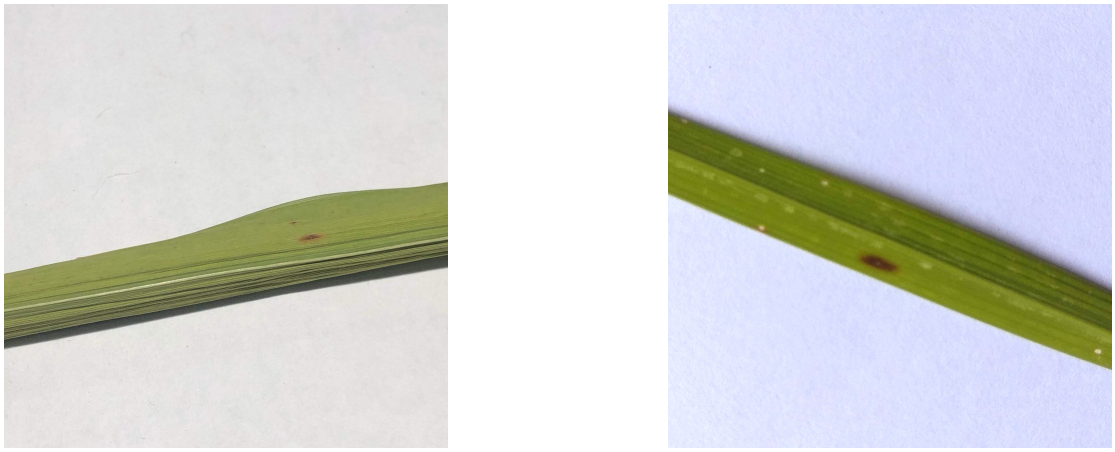


Figure 6.4: BrownSpot Images

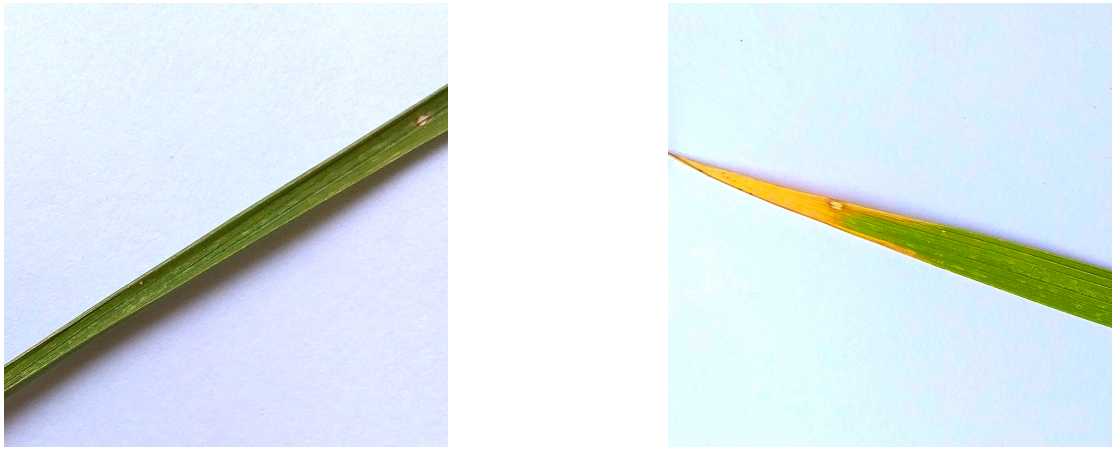


Figure 6.5: LeafBlast Images

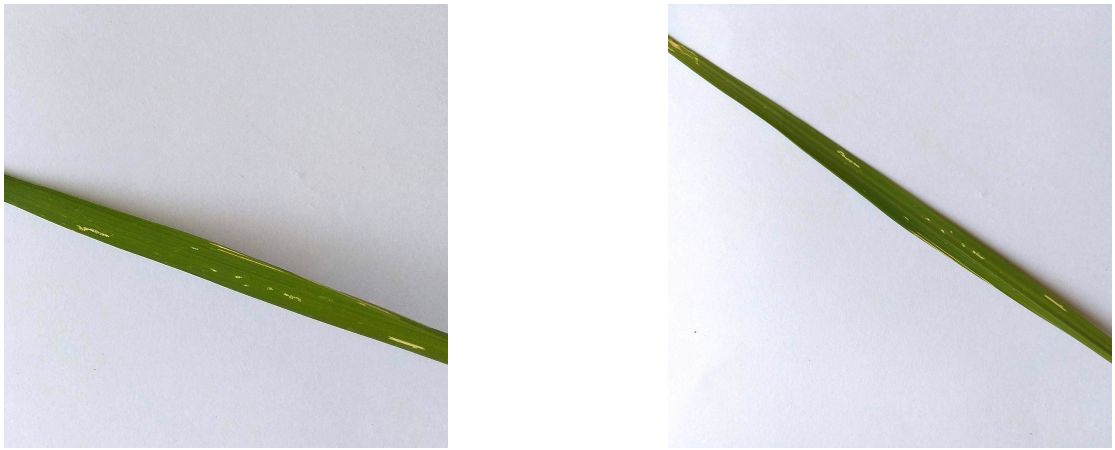


Figure 6.6: Hispa Images

6.6.4 Data for Preventive Measures and Solutions

Regarding the report suggesting a treatment for the disease in the mobile application, we gathered the information by reading the articles and websites listed below.

Leaf blast:

- https://extension.missouri.edu/publications/mp645
- https://www.nepjol.info/index.php/jpps/article/view/47294
- https://en.wikipedia.org/wiki/Magnaporthe_grisea

Hispa:

- https://plantix.net/en/library/plant-diseases/600098/rice-hispa
- http://riceportal.in/content/symptoms-and-nature-of-damage-of-rice-hispa

Brown spot:

- https://link.springer.com/article/10.1007/s10658-013-0195-6
- Brown spot of rice: an overview [23]

6.7 Data Preparation

After the manual evaluation of the entire dataset, lots of blur images were found in both the rice leaf and the non-rice leaf dataset. So those blur or not clear were deleted manually. Some examples are shown below.

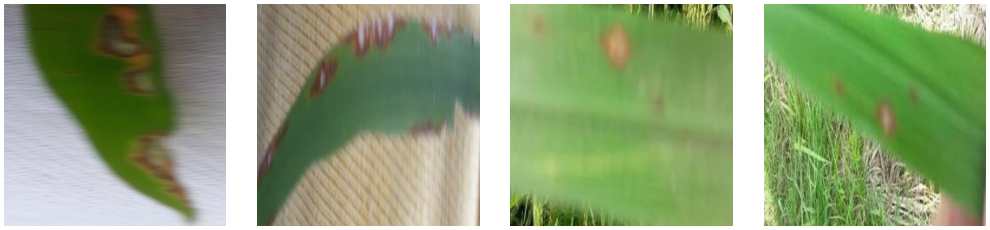


Figure 6.7: Some Unwanted Images Examples

For the model's training and evaluation, pixel scaling and image data augmentation were used. The constructed model uses image input of size 256×256. So, the data was first converted into 256x256 size.
Data Augmentation was made possible by using the Keras library in python. The loaded data for training were resized and normalized. The data were re-arranged in a random fashion and new images were produced by simple transformations such as rotation, horizontal flip, vertical flip, image cropping, etc to further increase the size of the data set the newly formed data set has size (min size = 88*102, max size = 2163*2163).

Data sets				
SN	**Class**	**Original**	**After Deletion**	**After Augmentation**
1	BrownSpot	523	435	640
2	Healthy	1488	1083	1083
3	Hispa	565	361	544
4	LeafBlast	779	504	701

Table 6.5: Number of Images in Each Class

The dataset used for binary classification is shown below, which was prepared by, rice leaf containing images of both healthy and non-healthy rice leaves. And non-rice leaves containing other types of leaves and objects and some human faces. Image size (256*256)

SN	**Class**	**Images**
1	Rice	5865
2	Non-rice	5467

Table 6.6: Number of Images in Each Class

6.8 Algorithms Used

Various algorithms were studied for building the modified CNN_LSTM image classifier. The following topics describe all the algorithms which are used in this project in detail:

6.8.1 Algorithms Used in Image Segmentation

Algorithms used in Image Segmentation are described below:-

6.8.1.1 Thresholding Algorithm

Thresholding is a technique in image processing that is used to separate an image into two parts, i.e., foreground and background, based on a threshold value. Various thresholding approaches such as adaptive thresholding, color thresholding, binary thresholding, and otsu thresholding were tested among which the result of the combination of binary thresholding and otsu thresholding was most suited for our case. So, the combination of otsu thresholding and binary thresholding is used in this project for image segmentation.

6.8.1.2 Binary Thresholding Algorithm

Binary thresholding is a simple and most effective image processing technique in our case which is used to convert a grayscale or color image into a binary image. In binary thresholding, each pixel in the input image is compared to a threshold value which is auto-calculated by the otsu thresholding. If the pixel value is greater than or equal to the threshold, the output pixel is set to a maximum value (e.g., 255 for an 8-bit image) and if the pixel value is less than the threshold, the output pixel is set to a minimum value (e.g., 0 for an 8-bit image). The result is a binary image that contains only two values: 0 (black) or 255 (white), based on whether the pixel value was above or below the threshold.

Step 1: For each pixel in the input image, compare its intensity value to the threshold value.

Step 2: If the pixel intensity is greater than or equal to the threshold value, set the corresponding pixel to 1(white).

Step 3: If the pixel intensity is less than the threshold value, set the corresponding pixel to 0(black).

Step 4: Return the binary image.

6.8.1.3 OTSU Thresholding Algorithm

Otsu's thresholding is a global image thresholding technique that automatically calculates an optimal threshold value based on the histogram of a grayscale image. The technique separates the image into two classes, foreground, and background, by choosing a threshold that maximizes the between-class variance of the pixel intensities. The threshold value is used to create a binary image, where the foreground pixels are assigned a value of 1 and the background pixels are assigned a value of 0. Otsu's thresholding is commonly used for image segmentation tasks, where the goal is to separate an object of interest from its background.

Step 1: Compute histogram and normalize
- Compute histogram h_i of pixel intensities $i \in [0, 255]$ in image I
- Normalize histogram

$$p_i = \frac{h_i}{NM}$$

Step 2: Compute cumulative distribution function

$$P_i = \sum_{j=0}^{i} p_j$$

Step 3: Compute the mean gray level of the image

$$\mu = \sum_{i=0}^{255} i p_i$$

Step 4: Compute global variance of the image

$$\sigma^2 = \sum_{i=0}^{255} (i - \mu)^2 p_i$$

Step 5: Iterate over all possible threshold values - For $t = 0$ to 255, do the following:

- Compute class probabilities and mean gray levels:

$$w_0 = P_t,\ w_1 = 1 - w_0$$
$$\mu_0 = \frac{\sum_{i=0}^{t-1} i p_i}{w_0},\ \mu_1 = \frac{\sum_{i=t}^{255} i p_i}{w_1}$$

- Compute between-class variance:

$$V_b = w_0 w_1 (\mu_0 - \mu_1)^2$$
$$V_b(t) = V_b$$

Step 6: Select threshold value that maximizes between-class variance:

$$t_{opt} = \text{argmax}_t \, V_b(t)$$

Step 7: Create binary image by thresholding original image:

$$B_{i,j} = \begin{cases} 1, & \text{if } I_{i,j} > t_{opt} \\ 0, & \text{otherwise} \end{cases}$$

6.8.1.4 Steps Taken for Segmentation

The following steps were taken for the segmentation of the diseased part for the images.

Step 1: Resizing the image to 256*256 pixel size.

if (image size > 256 * 256) {

resizing using inter_nearest interpolation (the value of each

pixel in the output image is simply equal to the value of

the nearest pixel in the input image).

} else {

border padding is performed with padding of 1.

}

Step 2: Background removal using " remove " function for rmbg library.

Step 3: Conversion to hsv channel and generating mask to remove background in case of failure of removal of background noise in Step 2.

Step 4: Edges detection using Canny edge detection by the method of otsu thresholding method.

kernel type = elliptical

Kernel1		
0	1	0
1	1	1
0	1	0

Table 6.7: Elliptical Kernel Used for Edge Detection

Step 5: Erode and Dilate operation to generate the first mask using a kernel.

kernel type = elliptical

Kernel2				
0	0	1	0	0
1	1	1	1	1
1	1	1	1	1
1	1	1	1	1
0	0	1	0	0

Table 6.8: Elliptical Kernel Used for Erode and Dilate Operation

Step 6: Creating first segmentation using generated, the first mask.

Step 7: Lab color conversion to image output of Step 2.

Step 8: Using ' a channel ' to create a second mask using the otsu thresholding method. The first kernel of 3*3 was used.

Step 9: Creating second segmentation using generated the second mask.

Step 10: Merging first segmentation and second segmentation to create the final segmented result.

6.8.2 Algorithms Used in Classification

6.8.2.1 Adam Optimizer

Adam (short for Adaptive Moment Estimation) is a stochastic gradient descent optimization algorithm that combines ideas from both the Adagrad and RMSProp algorithms. The basic idea behind Adam is to compute adaptive learning rates for each parameter based on the first and second moments of the gradients.

Step 1: Initialize the parameters of the model.

Step 2: Initialize the first and second moments of the gradients to zero.

Step 3: For each iteration:

a. Compute the gradients of the loss with respect to the parameters.

b. Update the first moment of the gradients using a moving average.

c. Update the second moment of the gradients using a moving average.

d. Compute the adaptive learning rate for each parameter.

e. Update the parameters using the adaptive learning rates and the first and second moments of the gradients.

Step 4: Repeat until convergence or for a fixed number of iterations.

6.8.3 Different Parameters for Evaluating Neural Network

The primary purpose of evaluating a neural network is to assess its performance on a given task and to identify any weaknesses or areas for improvement. The various parameters used to evaluate our model performance are accuracy, loss, confusion matrix, precision, recall, roc curve, and f1 score.

6.8.3.1 Cross-Entropy Loss

The cross-entropy loss is a commonly used loss function in machine learning, especially in classification problems. Here is an overview of the algorithm used to calculate the cross-entropy loss:

Step 1: Compute the predicted probability distribution over the classes for a given input data point using the model.

Step 2: Compute the true probability distribution over the classes for the input data point.

Step 3: Compute the cross-entropy loss as the negative log-likelihood of the true probability distribution under the predicted probability distribution.

Step 4: Repeat steps 1-3 for all input data points in the training set.

Step 5: Calculate the average cross-entropy loss over the training set.

The formula for cross-entropy loss for a single data point is:

$$L(y, \hat{y}) = -\sum_{i=1}^{n} y_i \log(\hat{y}_i) \tag{6.1}$$

where:

y is the true probability distribution over the classes

ŷ is the predicted probability distribution over the classes

i is the index of the classes

6.8.3.2 Accuracy

The 'accuracy' metric is a commonly used evaluation metric in classification problems. It measures the proportion of correctly classified samples among all the samples in the test set. Here is an overview of the algorithm used to calculate accuracy:

Step 1: Feed the test data through the trained model to get the predicted class labels.

Step 2: Compare the predicted class labels with the true class labels.

Step 3: Calculate the proportion of correctly classified samples as the accuracy.

Step 4: Repeat steps 1-3 for all the test samples.

Step 5: Calculate the average accuracy over the test set.

The formula for accuracy is:

$$\text{Accuracy} = \frac{\text{Number of correctly classified samples}}{\text{Total number of samples}}$$

6.8.3.3 Precision

Precision is a statistical metric that measures the proportion of true positive results among the total positive results in a classification or detection task. In other words, it is a measure of the accuracy of positive predictions or decisions made by a model or a system. Mathematically, it can be expressed as:

$$\text{Precision} = \frac{TP}{(TP+FP)}$$

where:
TP = True Positive (it is the number of cases where the model correctly identifies a sample as belonging to a specific class when it does indeed belong to that class.)
FP = False Positive (it is a case where the model mistakenly identifies a sample as belonging to a specific class when it actually belongs to a different class.)

6.8.3.4 Recall

Recall, also known as sensitivity or true positive rate (TPR), is a statistical metric that measures the proportion of true positive instances that are correctly identified as positive by a classification or detection system. In other words, it is a measure of how well the system is able to identify all instances of the positive class. Mathematically, it can be expressed as:

$$\text{Recall} = \frac{TP}{(TP+FN)}$$

where:
TP = True Positive (it is the number of cases where the model correctly identifies a sample as belonging to a specific class when it does indeed belong to that class.)
FN = False Negative (it is a case where the model mistakenly identifies a sample as not belonging to a specific class when it actually does belong to that class.)

6.8.3.5 F1 Score

The F1 score is a statistical metric used to evaluate the performance of a classification or detection system. It is a harmonic mean of precision and recall, and provides a single score that balances the trade-off between these two metrics. Mathematically, it can be expressed as:

$$\text{F1 Score} = \frac{2 * (precision * recall)}{(precision + recall)}$$

6.8.4 Different Parameters Used for Evaluation Segmentation

The various parameters used to evaluate our segmentation are Jaccard Index, Dice coefficient, precision, recall, and f1 score.

6.8.4.1 Jaccard Index

The Jaccard index, also known as the Jaccard similarity coefficient or Jaccard coefficient, is a statistic used for comparing the similarity or dissimilarity of two sets of data. It is commonly used in data science, information retrieval, and image processing.
The Jaccard index is defined as the size of the intersection of two sets divided by the size of the union of the sets. Mathematically, it can be expressed as:

$$\text{Jaccard Index} = \frac{(A \cap B)}{(A \cup B)}$$

where:

A = number of non-zeros in segmented image

B = number of non-zeros in ground truth image

6.8.4.2 Dice Coefficient

The Dice coefficient, also known as the Sørensen–Dice coefficient, is a statistical measure of the similarity between two sets of data. It is commonly used in image segmentation, medical image analysis, and natural language processing. The Dice coefficient is defined as twice the size of the intersection of two sets divided by the sum of the sizes of the two sets. Mathematically, it can be expressed as:

$$\text{Dice Coefficient} = \frac{2 * (A \cap B)}{(A) + (B)}$$

$$= \frac{2 * (TruePositive)}{(2 * TruePositive + FalsePositive + FalseNegative)}$$

where:

A = number of non-zeros in segmented image

B = number of non-zeros in ground truth image

6.8.4.3 Precision

Precision is a statistical metric that measures the proportion of true positive results among the total positive results in a classification or detection task. In other words, it measures the accuracy of positive predictions or decisions made by a model or a system. Mathematically, it can be expressed as:

$$\text{Precision} = \frac{TP}{(TP + FP)}$$

where:

TP = True Positive (number of pixels that are correctly classified as diseased pixels.)

FP = False Positive (number of pixels that are classified as diseased pixels but in actuality they are non-diseased pixels.)

6.8.4.4 Recall

Recall, also known as sensitivity or true positive rate (TPR), is a statistical metric that measures the proportion of true positive instances that are correctly identified as positive by a classification or detection system. In other words, it is a measure of how well the system is able to identify all instances of the positive class. Mathematically, it can be expressed as:

$$\text{Recall} = \frac{TP}{(TP + FN)}$$

where:

TP = True Positive (number of pixels that are correctly classified as diseased pixels.)

FN = False Negative (number of pixels that belong to diseased pixels in the ground truth image, but were incorrectly labeled as background or not belonging to diseased pixels.)

6.8.4.5 F1 Score

The F1 score is a statistical metric used to evaluate the performance of a classification or detection system. It is a harmonic mean of precision and recall, and provides a single score that balances the trade-off between these two metrics. Mathematically, it can be expressed as:

$$\text{F1 Score} = \frac{2 * (precision * recall)}{(precision + recall)}$$

6.9 Disease Part Segmentation

Image segmentation is the process of segmenting the image into various segments, that could be used for further applications such as Image understanding models, Robotics, Image analysis, Medical diagnosis, etc [24]. Among different segmentation processes, the thresholding approach was used for the disease part segmentation. Before the segmentation of the diseased part " remove " function from rembg was used to reduce the background noises. Then combination of masks generated by canny edge detection and lab color transformation was used to generate a segmented image. The method of thresholding used in the project is the combination of binary and OTSU thresholding [25] methods to segment disease spots only which are visualized below. The steps taken for the segmentation are described in subsubsection 6.8.1.4. The evaluation of the segmentation is shown in section 7.2 along with sample calculation.

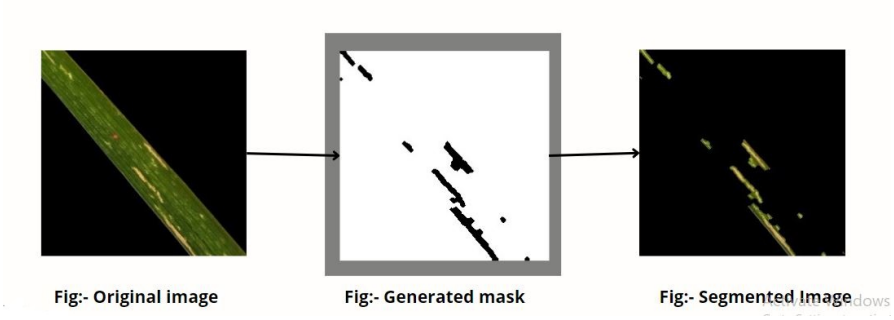


Figure 6.8: Segmentation Using Canny Edge Detection

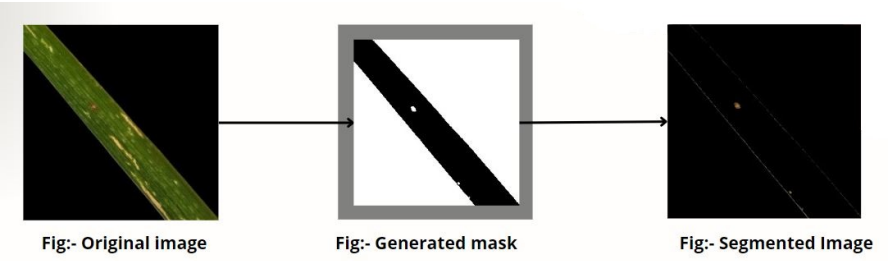


Figure 6.9: Segmentation Using Lab Color Space

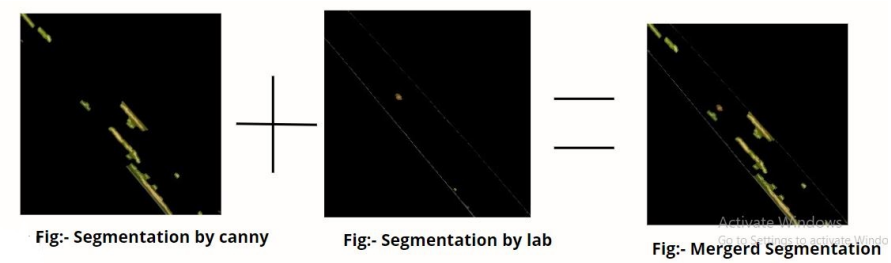
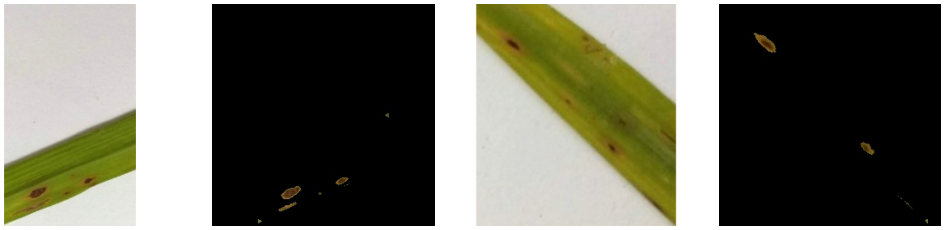
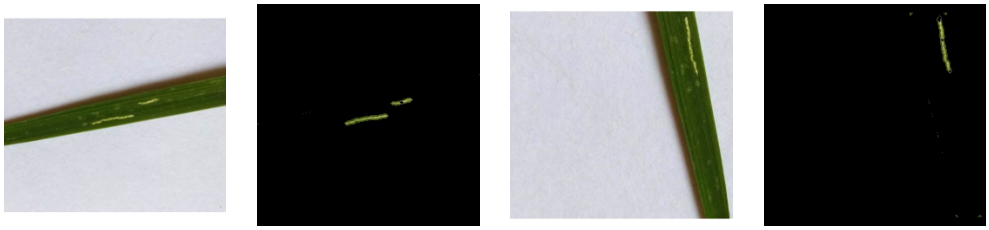


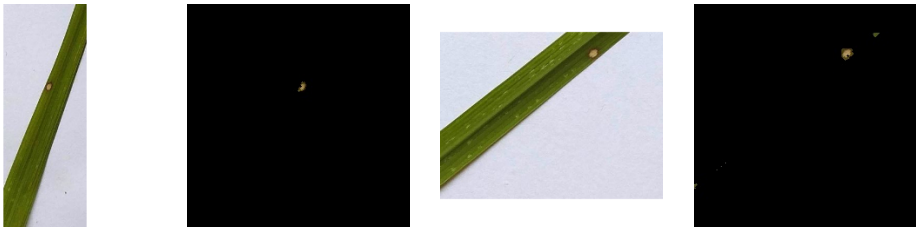
Figure 6.10: Merging Segmented Image



Brownspot Segmentation



Hispa Segmentation



Leafblast Segmentation

Figure 6.11: Some Segmentation Images Examples

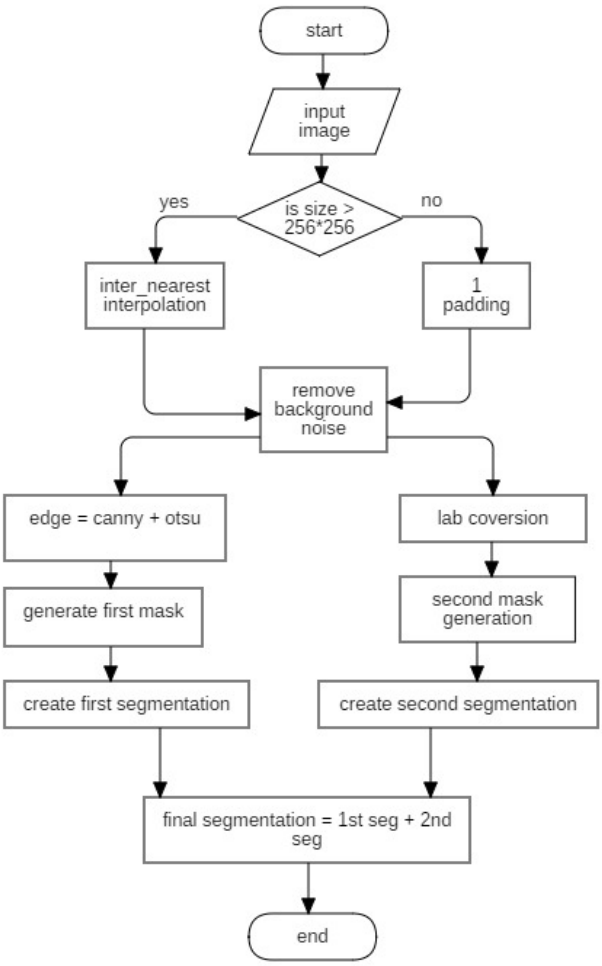


Figure 6.12: Segmentation Flowchart Diagram

The step by step procedure of the segmentation process is described in subsubsection 6.8.1.4.

6.10 Models

6.10.1 VGG-16 Model

We used a pre-trained VGG-16 model from the Keras library. Fine-tuning was performed on the imported model by removing all dense layers then the convolutional layers were freezed except for the last convolutional layer, then two fully connected layers were added between with a batch normalization layer and a dropout layer before training the model. The architecture of the pre-trained VGG-16 and the summary of the constructed model is shown below. And, the final accuracy of the model is shown in Table 6.5

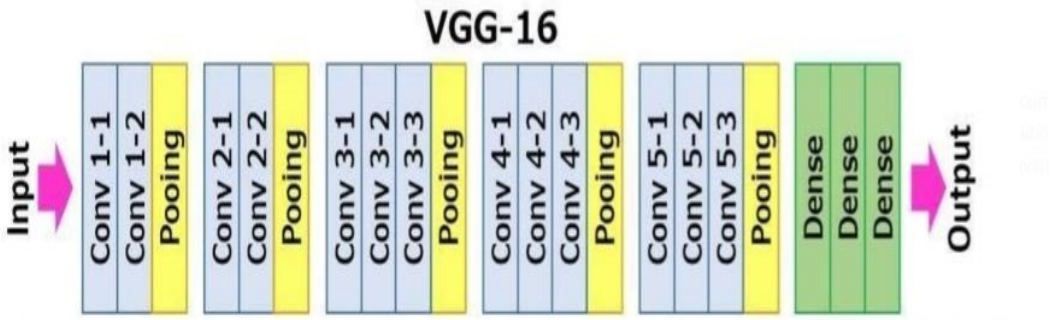


Figure 6.13: Architecture of VGG-16.
[2]

[2] https://medium.com/@mygreatlearning/everything-you-need-to-know-about-vgg16-7315defb5918

Model: VGG-16 Model		
Layer	**Output Shape**	**Param #**
input_1 (InputLayer)	[(None, 256, 256, 3)]	0
block1_conv1 (Conv2D)	(None, 256, 256, 64)	1792
block1_conv2 (Conv2D)	(None, 256, 256, 64)	36928
block1_pool (MaxPooling2D)	(None, 128, 128, 64)	0
block2_conv1 (Conv2D)	(None, 128, 128, 128)	73856
block2_conv2 (Conv2D)	(None, 128, 128, 128)	147584
block2_pool (MaxPooling2D)	(None, 64, 64, 128)	0
block3_conv1 (Conv2D)	(None, 64, 64, 256)	295168
block3_conv2 (Conv2D)	(None, 64, 64, 256)	590080
block3_conv3 (Conv2D)	(None, 64, 64, 256)	590080
block3_pool (MaxPooling2D)	(None, 32, 32, 256)	0
block4_conv1 (Conv2D)	(None, 32, 32, 512)	1180160
block4_conv2 (Conv2D)	(None, 32, 32, 512)	2359808
block4_conv3 (Conv2D)	(None, 32, 32, 512)	2359808
block4_pool (MaxPooling2D)	(None, 16, 16, 512)	0
block5_conv1 (Conv2D)	(None, 16, 16, 512)	2359808
block5_conv2 (Conv2D)	(None, 16, 16, 512)	2359808
block5_conv3 (Conv2D)	(None, 16, 16, 512)	2359808
block5_pool (MaxPooling2D)	(None, 8, 8, 512)	0
flatten (Flatten)	(None, 32768)	0
dense (Dense)	(None, 1024)	33555456
batch_normalization (BatchNormalization)	(None, 1024)	4096
dropout (Dropout)	(None, 1024)	0
dense_1 (Dense)	(None, 4)	4100
Total params: 48,278,340		
Trainable params: 35,921,412		
Non-trainable params: 12,356,928		

Table 6.9: Fine Tuned VGG-16 Model Summary

6.10.2 VGG-19 Model

As in the VGG-16 model, we used a pre-trained VGG-19 layer from the Keras library. All, other actions were performed as in the VGG-16 model. The architecture of the pre-trained VGG-16 and the summary of the constructed model is shown below. And, the accuracy of the model is shown in Table 6.5.

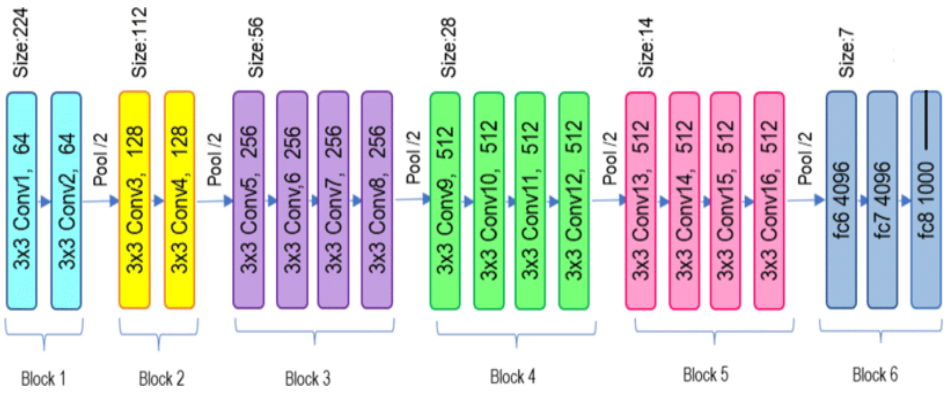


Figure 6.14: Architecture of VGG-19.
[3]

[3] https://www.researchgate.net/figure/VGG-19-Architecture-39-VGG-19-has-16-convolution-laye fig5_359771670

Model: VGG-19 Model		
Layer	**Output Shape**	**Param #**
input_1 (InputLayer)	[(None, 256, 256, 3)]	0
block1_conv1 (Conv2D)	(None, 256, 256, 64)	1792
block1_conv2 (Conv2D)	(None, 256, 256, 64)	36928
block1_pool (MaxPooling2D)	(None, 128, 128, 64)	0
block2_conv1 (Conv2D)	(None, 128, 128, 128)	73856
block2_conv2 (Conv2D)	(None, 128, 128, 128)	147584
block2_pool (MaxPooling2D)	(None, 64, 64, 128)	0
block3_conv1 (Conv2D)	(None, 64, 64, 256)	295168
block3_conv2 (Conv2D)	(None, 64, 64, 256)	590080
block3_conv3 (Conv2D)	(None, 64, 64, 256)	590080
block3_conv4 (Conv2D)	(None, 64, 64, 256)	590080
block3_pool (MaxPooling2D)	(None, 32, 32, 256)	0
block4_conv1 (Conv2D)	(None, 32, 32, 512)	1180160
block4_conv2 (Conv2D)	(None, 32, 32, 512)	2359808
block4_conv3 (Conv2D)	(None, 32, 32, 512)	2359808
block4_conv4 (Conv2D)	(None, 32, 32, 512)	2359808
block4_pool (MaxPooling2D)	(None, 16, 16, 512)	0
block5_conv1 (Conv2D)	(None, 16, 16, 512)	2359808
block5_conv2 (Conv2D)	(None, 16, 16, 512)	2359808
block5_conv3 (Conv2D)	(None, 16, 16, 512)	2359808
block5_conv4 (Conv2D)	(None, 16, 16, 512)	2359808
block5_pool (MaxPooling2D)	(None, 8, 8, 512)	0
flatten (Flatten)	(None, 32768)	0
dense (Dense)	(None, 1024)	33555456
batch_normalization (BatchNormalization)	(None, 1024)	4096
dropout (Dropout)	(None, 1024)	0
dense_1 (Dense)	(None, 4)	4100
Total params: 53,588,036		
Trainable params: 35,921,412		
Non-trainable params: 17,666,624		

Table 6.10: Fine Tuned VGG-19 Model Summary

6.10.3 LSTM Model

Long short-term memory (LSTM) is an artificial neural network used in the fields of artificial intelligence and deep learning. Unlike standard feedforward neural networks, LSTM has feedback connections. Such a recurrent neural network (RNN) can process not only single data points (such as images) but also entire sequences of data (such as speech or video).
A common LSTM unit is composed of a cell, an input gate, an output gate, and a forget gate. The cell remembers values over arbitrary time intervals and the three gates regulate the flow of information into and out of the cell. LSTM networks are well-suited to classifying, processing, and making predictions. An LSTM network is the same as a standard RNN, except those summation units in the hidden layer are replaced by memory blocks. The multiplicative gates allow LSTM memory cells to store and access information over long periods of time [26].

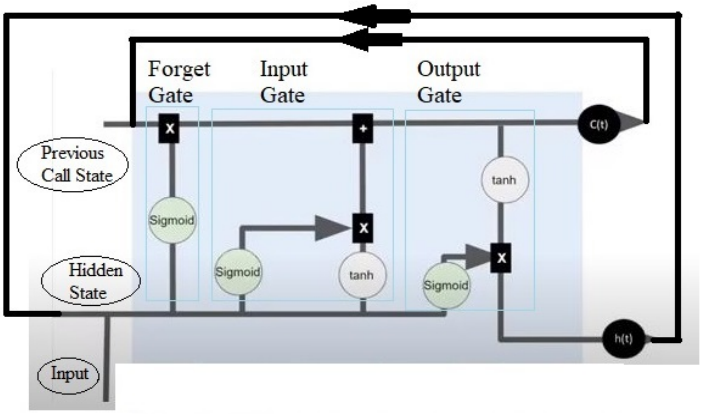


Figure 6.15: Architecture of LSTM.
[4]

[4]Image source: `https://www.youtube.com/watch?v=-qpXVJnRC1w`

6.10.4 Binary Classification Model

Before the classification of the diseased type of rice leaf, it was crucial to determine whether the input image is a rice leaf or not. so, various methods such as SVM, PCA algorithm, CNN were etc experimented whose details are shown in Table 7.1, among experimented various models one with the best output was chosen as the final model. The summary of the used model is shown below.

Model: Sequential Model		
Layer (type)	**Output Shape**	**Param #**
conv2d_22 (Conv2D)	(None, 254, 254, 16)	448
max_pooling2d (MaxPooling2D)	(None, 127, 127, 16)	0
conv2d_23 (Conv2D)	(None, 125, 125, 32)	4640
max_pooling2d_1 (MaxPooling2D)	(None, 62, 62, 32)	0
conv2d_24 (Conv2D)	(None, 60, 60, 64)	18496
max_pooling2d_2 (MaxPooling2D)	(None, 30, 30, 64)	0
flatten (Flatten)	(None, 57600)	0
dense (Dense)	(None, 256)	14745856
batch_normalization (BatchNormalization)	(None, 256)	1024
dropout (Dropout)	(None, 256)	0
dense_1 (Dense)	(None, 1)	257
Total params: 14,770,721		
Trainable params: 14,770,209		
Non-trainable params: 512		

Table 6.11: Binary Classification Model Summary

The first layer is a 2D convolutional layer with 16 filters, a filter size of 3*3, a stride of 1, a RELU activation function, and an input shape of (256,256,3). The convolutional layer is followed by a max pooling layer with a default pool size of 2*2. It is then followed by the 2D convolutional layer with 32 filters, a filter size of 3*3, a stride of 1, and a RELU activation function. Then is followed by another max pooling layer with the default pool size of 2*2. Then it is again followed by the 2d convolutional layer with 64 filters, a filter size of 3*3, a stride of 1, and a RELU activation function. Then is followed by another max pooling layer with the default pool size of 2*2. Then the output of the max pooling layer is flattened to 1D array and is connected to a fully connected dense layer with 256 neurons and a RELU activation function. A batch normalization layer is added to normalize the outputs from the previous layer. A dropout layer is added to randomly drop out the 20% of the neurons in the previous layer during the training to prevent overfitting. And finally, a fully connected layer with a single neuron and a sigmoid activation function output the binary classification output(probability score) of the model. Overall this CNN model consists of multiple layers of convolutional layer, max pooling, and fully connected layer with batch normalization and dropout layers to improve performance and prevent overfitting.

6.10.5 Multi Class Classification Model

6.10.5.1 CNNs-LSTM Model

The proposed model is a combination of CNN (VGG model) and RNN (LSTM model). The model is composed of two CNN models, a single-layered LSTM model, and fully connected layers whose block diagram is shown in Figure 5.4. The output of the CNN model is used to produce sequential data to help LSTM for making predictions so that it can easily extract highly accurate image features. The evaluation of the Integrated model is shown in section 7.3 along with sample calculation.

Model: VGG16-VGG19-LSTM Model			
Layer	**Output Shape**	**Param #**	**Connected to**
input_1 (InputLayer)	(None, 256, 256, 3)	0	
model (Functional)	(None, 8, 8, 512)	14,714,688	input_1[0][0]
model_1 (Functional)	(None, 8, 8, 512)	20,024,384	input_1[0][0]
flatten (Flatten)	(None, 32768)	0	model[0][0]
flatten_1 (Flatten)	(None, 32768)	0	model_1[0][0]
concatenate (Concatenate)	(None, 65536)	0	flatten[0][0], flatten_1[0][0]
lambda (Lambda)	(1, 128, 512)	0	concatenate[0][0]
lstm (LSTM)	(1, 128)	278,784	lambda[0][0]
dense (Dense)	(1, 4)	260	dropout[0][0]
Total params: 35,067,780			
Trainable params: 328,708			
Non-trainable params: 34,739,072			

Table 6.12: Integrated Model Summary

It consist of single input layer taking RGB image of size 256*256 as input. Two models, model(VGG16) and model_1(VGG19) are connected to same input layer. A flatten layer is added to both CNNs model which are then concatenate together. Lambda layer is used to reshape concatenated feature to desired input shape for LSTM. Here the LSTM output(1, 128) represents, LSTM model is trained single image at a time and have 128 memory cells. Input to LSTM is provided in such a way that in each time step comparision of 512 features is performed and stored in a single memory cell. Finally, the LSTM layer is connected to fully connected dense layer with four nodes for final prediction.

6.11 Training

After data has been prepared the data is fed into various models described in Section section 6.10. Various evaluations were carried out in the trained model which is shown in chapter 7. Various hyperparameter tuning actions were performed to increase the accuracy of the models which are described in

Table 7.4. Besides the listed hyperparameter tuning other tuning operations were also performed, such as tuning the learning rate, stopping model training, etc. Since, manual reduction of the learning rate, and defining the number of epochs to train the model was not effective and optimal way of training the model. Concepts of auto-learning rate reduction and early stopping approach were implemented for both binary classification and multi-class classification models.
Following conditions were defined for reducing the learning rate and stopping model training.

- **Learning Rate Reduction.**
  If the value of val_loss (validation loss) is not deviated by a minimum of 0.001 in the 5 epoch cycles then reduce the learning rate by 0.1 factor (i.e reduce the learning rate by 10 times).

- **Early Stopping**
  If the maximum values of val_accuracy (validation accuracy) is not deviated by a minimum of 0.001 in the 10 epoch cycles then stop training the model.

As for training the classification model, the following actions were performed.

- Model architecture was prepared.

- Since datasets were pre-splitted into training and testing sets, the model was trained using the training datasets, and evaluations were performed using the testing sets.

- Various graphs were plotted for visualization of results.

6.12 Description of User Interface

We live in an age where almost everyone owns a mobile phone especially android phones with cameras and storage available. However, the computation power of budget android phones may not be enough for executing our models. So, assuming the user with internet access we have built a flutter-based android application that communicates with the server via API. With the use of an image picker, the application makes it easy to choose, crop, and click images for input. It also gives users access to reports that servers have generated and can be stored or deleted using hives in Flutter.

6.13 Description of Server

All computation is handled by the server. For the project, we have chosen a FastAPI Python-based server. FastAPI supports HTTP requests, the request received from users through the mobile application. The server first checks if the provided image is a rice leaf with the help of a trained binary classification model. if rice leaf, the image is further passed for image segmentation. The output of segmentation is then passed to the disease classification model that returns the disease detected by the model. Based on the disease detected

the server returns an API as the report in JSON format which includes the disease, overview, symptoms, preventive measures, and solution to the mobile application.

Chapter 7

Experiments And Results

7.1 Evaluation of Binary Classification Model

For the first model, the features were extracted using pre-trained vgg19, then the support vector machine was used for classification with tuned hyper-parameters using cross-validation(5 fold) and grid search.

For the second model, the extracted features were reduced using the PCA algorithm. The number of the principal components were chosen by repeated hit and trial, followed by the observing plot between the percentage of variance explained by a principal component and principle components. Then, the reduced (x, 100) data and corresponding labels were used to train the SVM machine. The evaluation of the Binary Classification model is shown in section 7.1 along with sample calculation

For the third model a new neural network was constructed, the summary of the model is shown in Table 6.11. Overall, this architecture is designed to extract relevant features from images and classify them into one of two classes, while also including regularization techniques to prevent overfitting. The model comparison is shown in Table 7.1.

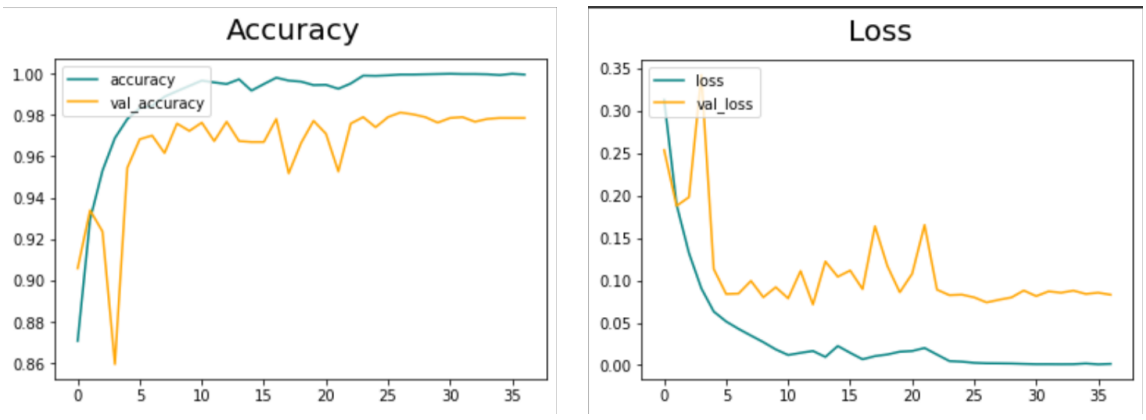


Figure 7.1: Accuracy and Loss Curve for Binary Classification

From the above graph we can observe that the training and testing accuracy of our model is gradually increasing over time, which is a good indication that

our model is learning from the training data. Generally, when the validation accuracy curve of the model is lower than the training accuracy curve, such a condition is known as data leakage. As the difference in training and testing data accuracy is only 2.1% which is a very small difference. So, we can conclude that our model is training appropriately.
Further evaluation of this binary classification model was done on the basis of the following confusion metric:

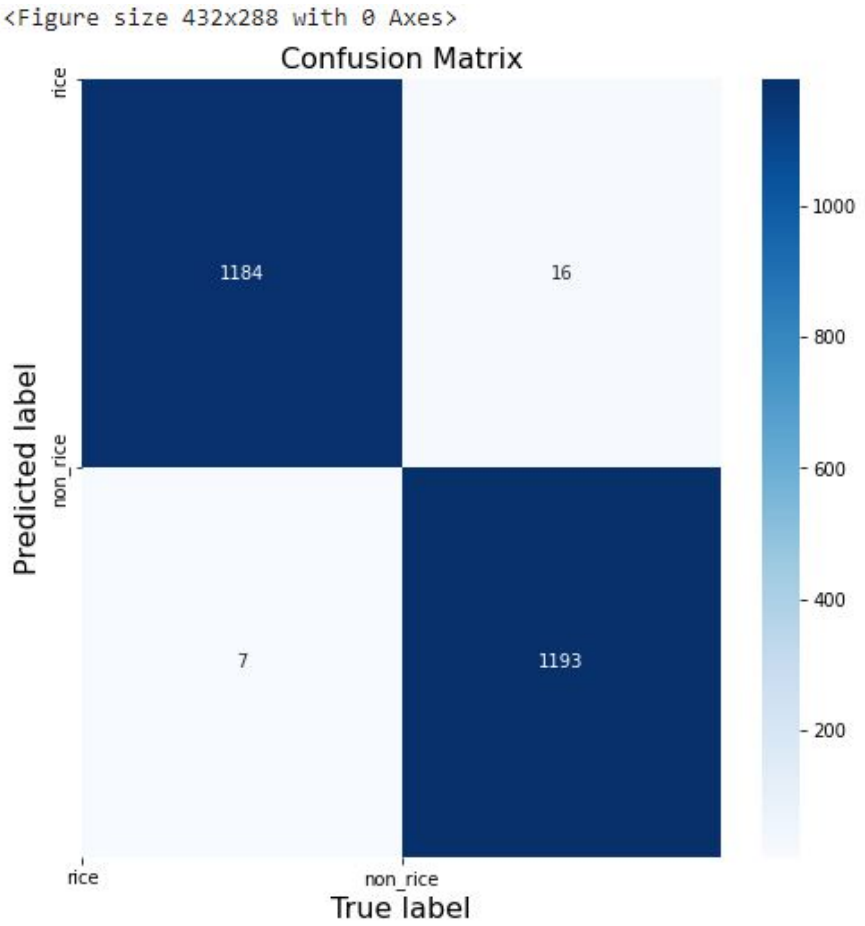


Figure 7.2: Confusion Matrix of Binary Classification

The above-mentioned confusion metric is plotted with 1200 test data of each case, the top left data represents the true positive values, the top right data represents the false negative values, the bottom left data represent the false positive values and the bottom right value represent the true negative value, which were utilized to calculate evaluation parameters such as Precision, Recall and F1 score.

Example of Final Average Result for Evaluation Of Binary Classifier Model:

Determined Average Values:

True Positive(TP) = 1193.0

False Positive(FP) = 16.0

False Negative(FN) = 7.0

1. Precision $= \frac{TP}{(TP+FP)} = \frac{1193.0}{1193.0+16.0} = 0.9867659222497932$

2. Recall $= \frac{TP}{(TP+FN)} = \frac{1193.0}{1193.0+7.0} = 0.9941666666666666$

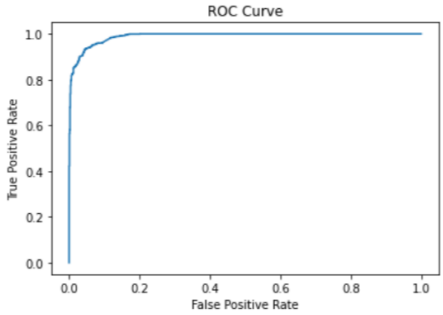


Figure 7.3: ROC Curve for Binary Classification Model

SN	Models	Train ACC	Test ACC
1	VGG19-SVM	91.6%	90.7%
2	VGG19-PCA-SVM	94.33%	93.66%
3	CNN	99.96%	97.86%

Table 7.1: Binary Classification Model Comparision

7.2 Evaluation of Segmentation

The Evaluation of our Segmentation was a very difficult task in our case due to the inaccessibility of the ground truth of the segmented region. So, in order to tackle this problem we created our own ground truth for some samples of our dataset with the help of the " Sefexa " desktop tool. The segmentation was based on the thresholding method. The final evaluations of our segmentation are mentioned below:

SN	Evaluation Type	Brownspot	Hispa	Leafblast
1	Jaccard Index	0.6355	0.5412	0.4596
2	Dice Coefficient	0.7720	0.6900	0.6148
3	Positive Predicted Value (PPV or Precision)	0.7285	0.6651	0.7454
4	Sensitivity(Recall)	0.8422	0.7431	0.5725
5	F1 Score	0.7720	0.6900	0.6148

Table 7.2: Evaluation of Segmentation

From the above table we can observe the value of evaluation of segmentation is very small, the main reason for this is due to the small number of pixels occupied by segmentation compared to the total pixels of the image. As Jaccard index is not very suitable for the evaluation of overlapping regions of small dimensions. Others, segmentation evaluation parameters were also calculated to evaluate the segmentation.
The mathematical expressions for evaluating parameters are described in chapter 7.

Example of Final Average Result for Evaluation of Brownspot:

Determined Average Values:

Intersection = 1.237335205078125

Union = 1.805419921875

True Positive(TP) = 318.0

False Positive(FP) = 71.0

False Negative(FN) = 75.0

1. Jaccard Index $= \frac{Intersection}{Union} = \frac{1.237335205078125}{1.805419921875}$

   $= 0.7720131661328153$

2. Dice Coefficient $= \frac{2 * TP}{(2 * TP) + FP + FN} = \frac{2 * 318.0}{(2 * 318) + 71.0 + 75.0}$

   $= 0.7720131661328153$

3. Precision $= \frac{TP}{(TP + FP)} = \frac{318.0}{318.0 + 71.0}$

   $= 0.7285028628277521$

4. Recall $= \frac{TP}{(TP + FN)} = \frac{318.0}{318.0 + 75.0} = 0.8422416659266592$

5. F1 Score $= \frac{2 * (precision * recall)}{(precision + recall)} =$

   $\frac{2 * (0.7285028628277521 * 0.8422416659266592)}{0.7285028628277521 + 0.8422416659266592}$

   $= 0.7720131661328153$

7.3 Evaluation of Integrated model

Various approaches were taken for the evaluation of our integrated model which is described in chapter 7. Along with mentioned above method, the final training and testing accuracy were compared among different models trained using the available datasets as mentioned in Table 7.4 for evaluation.

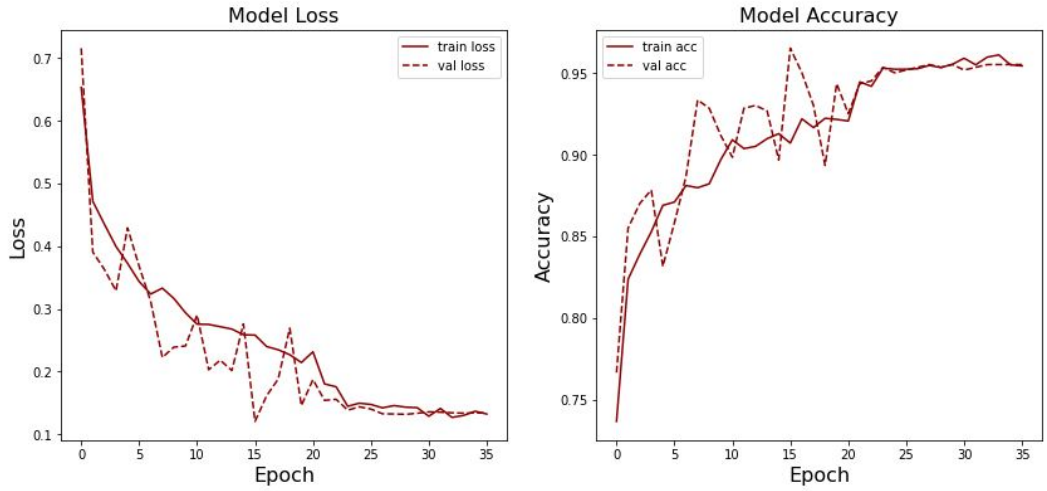


Figure 7.4: Loss and Accuracy Plot of Integrated Model

From the above graph we can observe that initially up to 20 epochs our model is very unstable, as it was training with an early learning rate of 0.001. As the first learning rate reduction was first observed at 21 epochs, after the reduction in the learning rate we can observe that our model is undergoing a stable state. From the above graph, we can observe that the training and testing accuracy of our model is gradually increasing over time, which is a good indication that our model is learning from the training data. As the difference in training and testing data accuracy is only 0.08% which is a very small difference can be considered a negligible value. So, we can conclude that our model is trained appropriately.

Final Train Accuracy	95.42%
Final Test Accuracy	95.50%
Final Train Loss	0.1321
Final Test Loss	0.1338

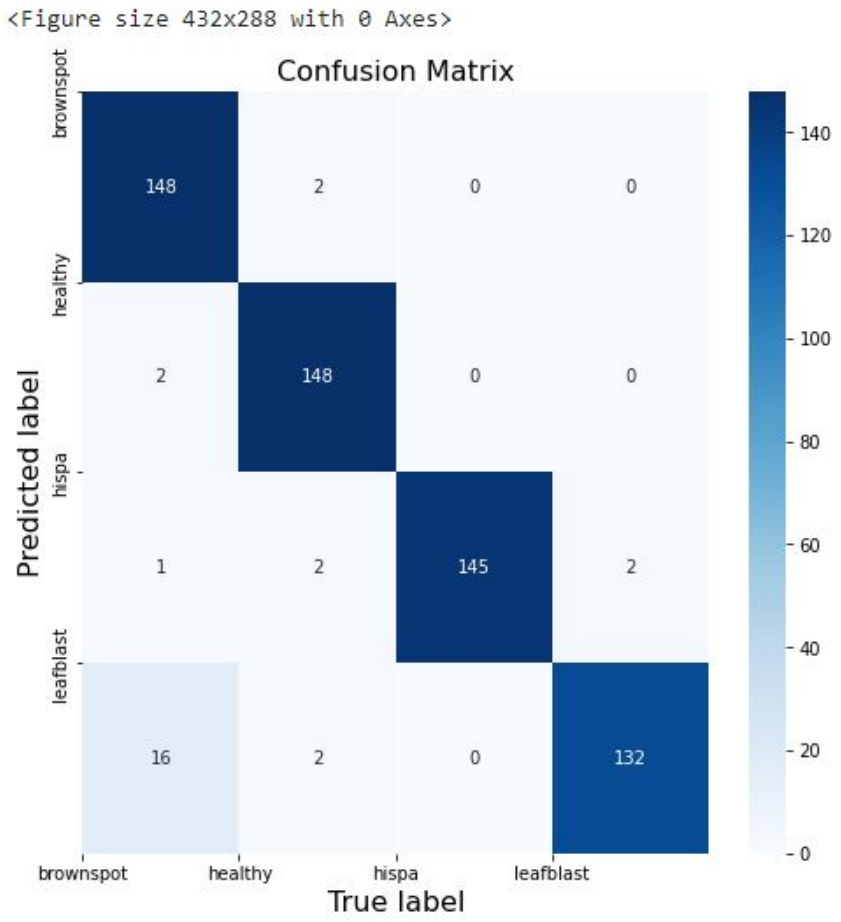


Figure 7.5: Confusion Matrix of Integrated Model

The above-mentioned confusion metric is plotted with 150 test data of each case, the diagonal data represents the true positive value of each class, the sum of elements in the column except the true positive value represents the false positive value of each class and the sum of elements in a row except true positive value represents the false negative value of each class which was utilized to calculate evaluation parameters such as Precision, Recall and F1 score.

The mathematical expressions for evaluating parameters are described in chapter 7.

Classes	**Brownspot**	**Healthy**	**Hispa**	**Leafblast**
Precision score per class	0.8862	0.9610	1.0	0.9851
Recall score per class	0.9867	0.9867	0.9667	0.88
F1 score per class	0.9338	0.9737	0.9830	0.9296

Example of Final Average Result For Evaluation Of Brownspot:

Determined Average Values:

True Positive(TP) = 132.0

False Positive(FP) = 2.0

False Negative(FN) = 18.0

1. Precision = $\frac{TP}{(TP+FP)} = \frac{132.0}{132.0+2.0} = 0.8862275449101796$

2. Recall = $\frac{TP}{(TP+FN)} = \frac{132.0}{132.0+18.0} = 0.9866666666666667$

3. F1 Score = $\frac{2*(precision*recall)}{(precision+recall)}$

   $= \frac{2*(0.8862275449101796*0.9866666666666667)}{0.8862275449101796+0.9866666666666667}$

   $= 0.9337539432176656$

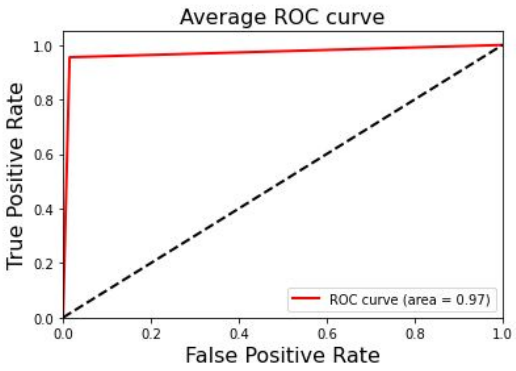


Figure 7.6: Precision-Recall Plot and Average_ROC_Curve of Integrated Model

7.3.1 Model Tuning

Various model tuning actions were performed to increase the accuracy of the model and to increased stability of the model.
For the fine-tuning operation of the VGG16 and VGG19 existing layer of the pre-trained model was removed then new layers were added to it. The summary of the fine-tuned model of vgg16 and vgg19 are shown in Table 6.9 and Table 6.10 respectively. Both binary classification model and multi-class classification models were trained under a similar environment as described in section 6.11. Both models start learning at the rate of 0.001, but the final learning rate of the binary classification model was 1.0000e-05 and the final learning rate of multi-class classification model was 1.0000e-06.

VGG_16 + VGG_19 + LSTM							
SN	**Hyperparameters**	**Test1**			**Test2**		
		Train Acc	**Test Acc**	**Epoch**	**Train Acc**	**Test Acc**	**Epoch**
"8" different real time data augmentation were performed							
rescale: 1/255, horizontal_flip, vertical_flip, rotation_range: 30, zoom_range: 0.2, width_shift_range: 0.1, height_shift_range: 0.2, shear_range: 0.2							
1	64 memory cell	83.89%	86.50%	19	87.94%	88.50%	40
2	128 memory cell	89.09%	88.83%	46	89.29%	89.67%	39
3	256 memory cell	89.52%	89.17%	40	89.15%	88.67%	45
Real time data augmentation was reduced to "6"							
rescale: 1/255, rotation_range: 30, zoom_range: 0.2, width_shift_range: 0.1, height_shift_range: 0.2, shear_range: 0.2							
4	128 memory cell	92.22%	93.17%	52	93.40%	93.50%	49
Real time data augmentation was reduced to "4"							
rescale: 1/255, rotation_range: 30, zoom_range: 0.2, shear_range: 0.2							
5	128 memory cell	92.99%	93.17%	40	95.42%	95.50%	36

Table 7.3: Hyper parameters Changes Experiments of Integrated Model

SN	**Model**	**Original Image**			**Segmented Image**		
		Train ACC	**Test ACC**	**Epoch**	**Train ACC**	**Test ACC**	**Epoch**
1	VGG16_ VGG19_ LSTM	91.11%	93.17%	75	95.42%	95.50%	36
2	VGG16	86.15%	80%	22	88.24%	87.83%	44
3	VGG19	84.70%	74%	41	87.26%	88.83%	66
4	Densenet_ 121	88.85%	91.83%	31	81.77%	84.50%	26
5	VGG16_ LSTM	88.75%	91.50%	30	86.76%	88.50%	31
6	VGG19_ LSTM	85.58%	84.83%	25	86.73%	89.33%	36

Table 7.4: Multi Class Classification Model Comparision

Chapter 8

Limitation And Future Enhancement

The following are the limitations of the projects are:

- The project can only classify three different classes of disease.
- A leaf can suffer from multiple diseases but the model developed can only detect one disease on the leaf.
- Usually parallel-veined leaves have a similar appearance, due to this reason parallel-veined leaves are considered to be rice leaves.
- The application we build is in only the English language so, the user must be familiar with the English language.
- Internet access is required to communicate with the server for classification.

The following points can be taken for future enhancement:

- More disease datasets can be implemented to increase disease detection.
- Improvements can be made to the segmentation quality through various means.
- Multi-disease in a leaf can be implemented.

Chapter 9

Conclusion

In conclusion, we developed the model to classify if the provided image is the image of a rice leaf or not. Multiple models were compared. CNN models were finally used for the binary classification. The disease part is segmented out and finally forwarded to the disease classification model. Our models have a very high accuracy of 97.86% for binary classification and 95.50% for multi-classification model and are very feasible for real-life environments. Early detection of rice leaf diseases can help farmers take necessary preventive measures before the disease spreads and causes significant crop loss.

Bibliography

[1] U. Mokhtar, N. E. Bendary, A. E. Hassenian, E. Emary, M. A. Mahmoud, H. Hefny, and M. F. Tolba, "Svm-based detection of tomato leaves diseases," in *Intelligent Systems' 2014*. Springer, 2015, pp. 641–652.

[2] S. Ramesh, R. Hebbar, M. Niveditha, R. Pooja, N. Shashank, P. Vinod *et al.*, "Plant disease detection using machine learning," in *2018 International conference on design innovations for 3Cs compute communicate control (ICDI3C)*. IEEE, 2018, pp. 41–45.

[3] A. Fuentes, S. Yoon, S. C. Kim, and D. S. Park, "A robust deep-learning-based detector for real-time tomato plant diseases and pests recognition," *Sensors*, vol. 17, no. 9, p. 2022, 2017.

[4] D. Garg and M. Alam, "Integration of convolutional neural networks and recurrent neural networks for foliar disease classification in apple trees," *International Journal of Advanced Computer Science and Applications*, vol. 13, no. 4, 2022.

[5] P. Li, H. Tang, J. Yu, and W. Song, "Lstm and multiple cnns based event image classification," *Multimedia Tools and Applications*, vol. 80, pp. 30 743–30 760, 2021.

[6] M. Z. Islam, M. M. Islam, and A. Asraf, "A combined deep cnn-lstm network for the detection of novel coronavirus (covid-19) using x-ray images," *Informatics in medicine unlocked*, vol. 20, p. 100412, 2020.

[7] A. K. Rangarajan, R. Purushothaman, and A. Ramesh, "Tomato crop disease classification using pre-trained deep learning algorithm," *Procedia computer science*, vol. 133, pp. 1040–1047, 2018.

[8] R. Karthik, M. Hariharan, S. Anand, P. Mathikshara, A. Johnson, and R. Menaka, "Attention embedded residual cnn for disease detection in tomato leaves," *Applied Soft Computing*, vol. 86, p. 105933, 2020.

[9] G. Hu, H. Wu, Y. Zhang, and M. Wan, "A low shot learning method for tea leaf's disease identification," *Computers and Electronics in Agriculture*, vol. 163, p. 104852, 2019.

[10] T. Islam, M. Sah, S. Baral, and R. R. Choudhury, "A faster technique on rice disease detectionusing image processing of affected area in agro-field," in *2018 Second International Conference on Inventive Communication and Computational Technologies (ICICCT)*. IEEE, 2018, pp. 62–66.

[11] S. Lamba, A. Baliyan, and V. Kukreja, "A novel gcl hybrid classification model for paddy diseases," *International Journal of Information Technology*, pp. 1–10, 2022.

[12] N. Krishnamoorthy, L. N. Prasad, C. P. Kumar, B. Subedi, H. B. Abraha, and V. Sathishkumar, "Rice leaf diseases prediction using deep neural networks with transfer learning," *Environmental Research*, vol. 198, p. 111275, 2021.

[13] M. Al-Amin, D. Z. Karim, and T. A. Bushra, "Prediction of rice disease from leaves using deep convolution neural network towards a digital agricultural system," in *2019 22nd International Conference on Computer and Information Technology (ICCIT)*. IEEE, 2019, pp. 1–5.

[14] X. Niu, M. Wang, X. Chen, S. Guo, H. Zhang, and D. He, "Image segmentation algorithm for disease detection of wheat leaves," in *Proceedings of the 2014 International Conference on Advanced Mechatronic Systems*. IEEE, 2014, pp. 270–273.

[15] Y. Zhou, Y. Wang, and Q. Yao, "Segmentation of rice disease spots based on improved bpnn," in *2010 International Conference on Image Analysis and Signal Processing*. IEEE, 2010, pp. 575–578.

[16] L. Tulchak and rchuk, "History of python," Ph.D. dissertation, , 2016.

[17] S. RIYAZ, "Riceleafdataset," https://www.kaggle.com/datasets/shayanriyaz/riceleafs, 2018.

[18] A. M. HASHAN, "Appleaf," https://www.kaggle.com/datasets/mhantor/apple-leaf-diseases, 2020.

[19] J. RASHID, "potatoleaf," https://www.kaggle.com/datasets/rizwan123456789/potato-disease-leaf-datasetpld, 2021.

[20] O. GETCH, "Wheatleaf," https://www.kaggle.com/datasets/olyadgetch/wheat-leaf-dataset, 2020.

[21] H. Getachew, "Wheatleaf," https://data.mendeley.com/datasets/wgd66f8n6h/1, 2021.

[22] prabira Kumar sethy, "Riceleaf," https://data.mendeley.com/datasets/fwcj7stb8r/1, 2020.

[23] S. Sunder, R. Singh, and R. Agarwal, "Brown spot of rice: an overview," *Indian Phytopathology*, vol. 67, no. 3, pp. 201–215, 2014.

[24] S. Saini and K. Arora, "A study analysis on the different image segmentation techniques," *International Journal of Information & Computation Technology*, vol. 4, no. 14, pp. 1445–1452, 2014.

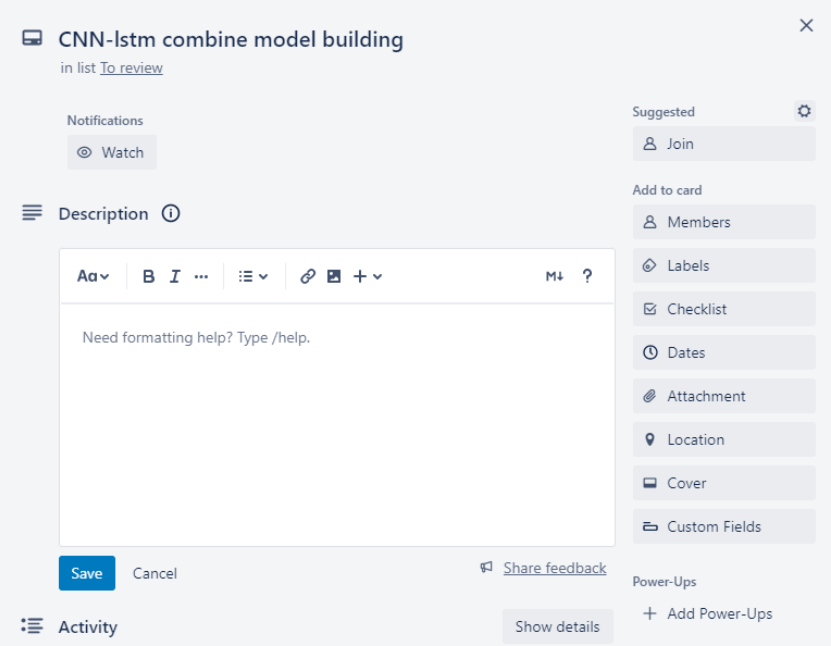
[25] J. Yousefi, "Image binarization using otsu thresholding algorithm," *Ontario, Canada: University of Guelph*, vol. 10, 2011.

[26] A. Graves, "Long short-term memory," *Supervised sequence labelling with recurrent neural networks*, pp. 37–45, 2012.

Appendix

A Snapshot

A.1 Create Task in Trello



B App Screenshots

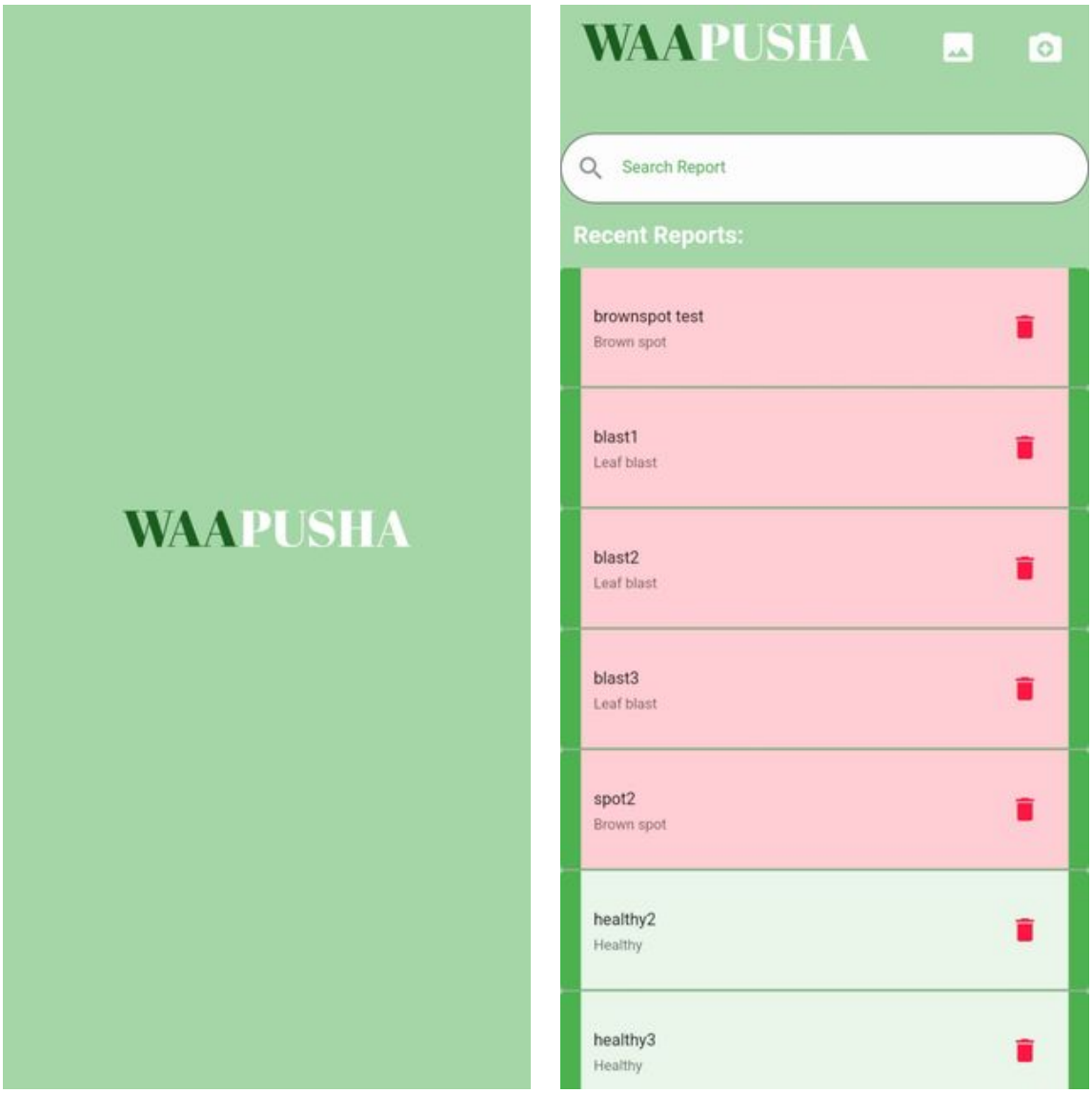


Figure 9.1: App Screenshot

C Unit Test with Healthy Leaf

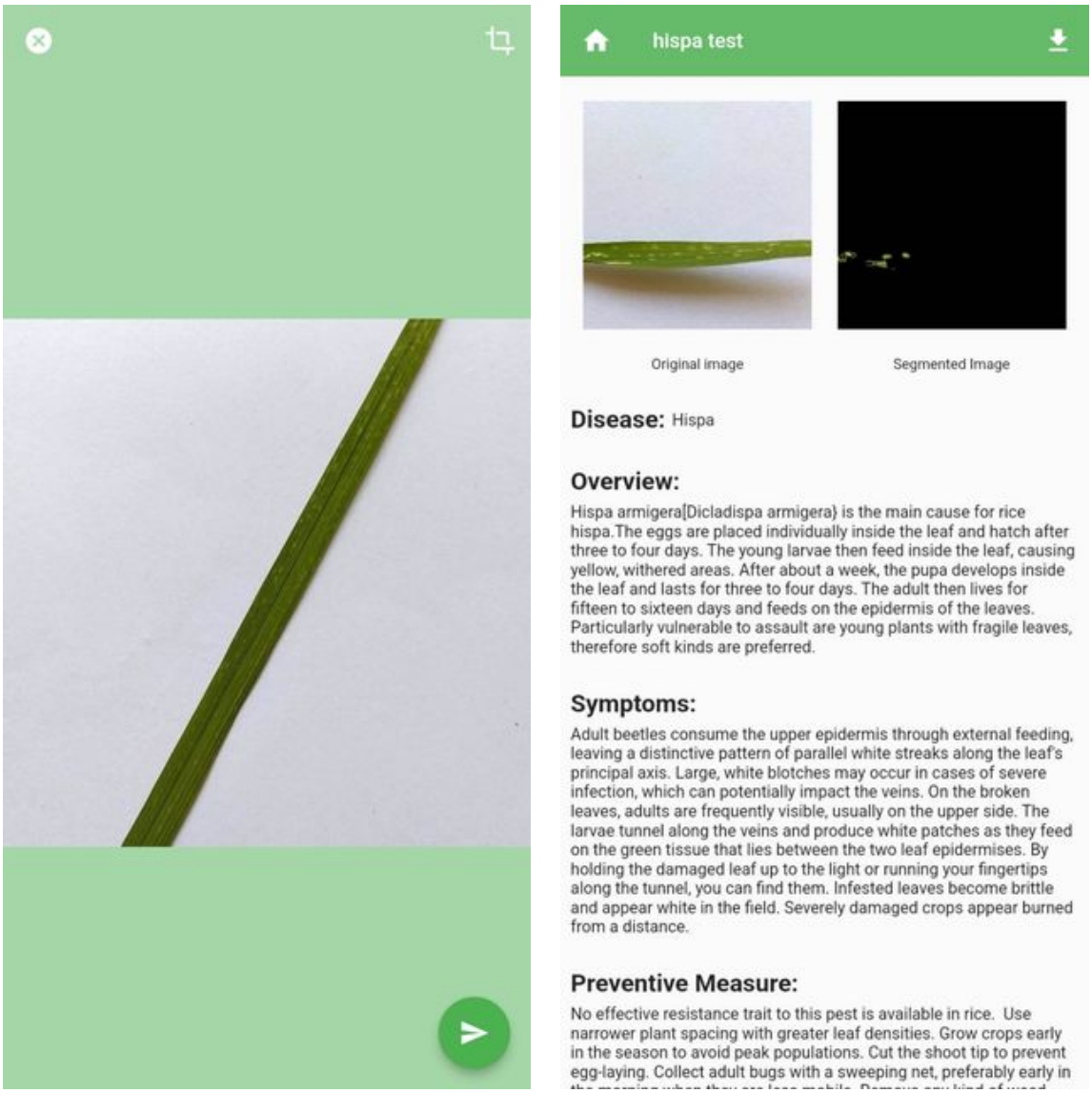


Figure 9.2: Unit Test 1

D Unit Test with Brownspot Leaf

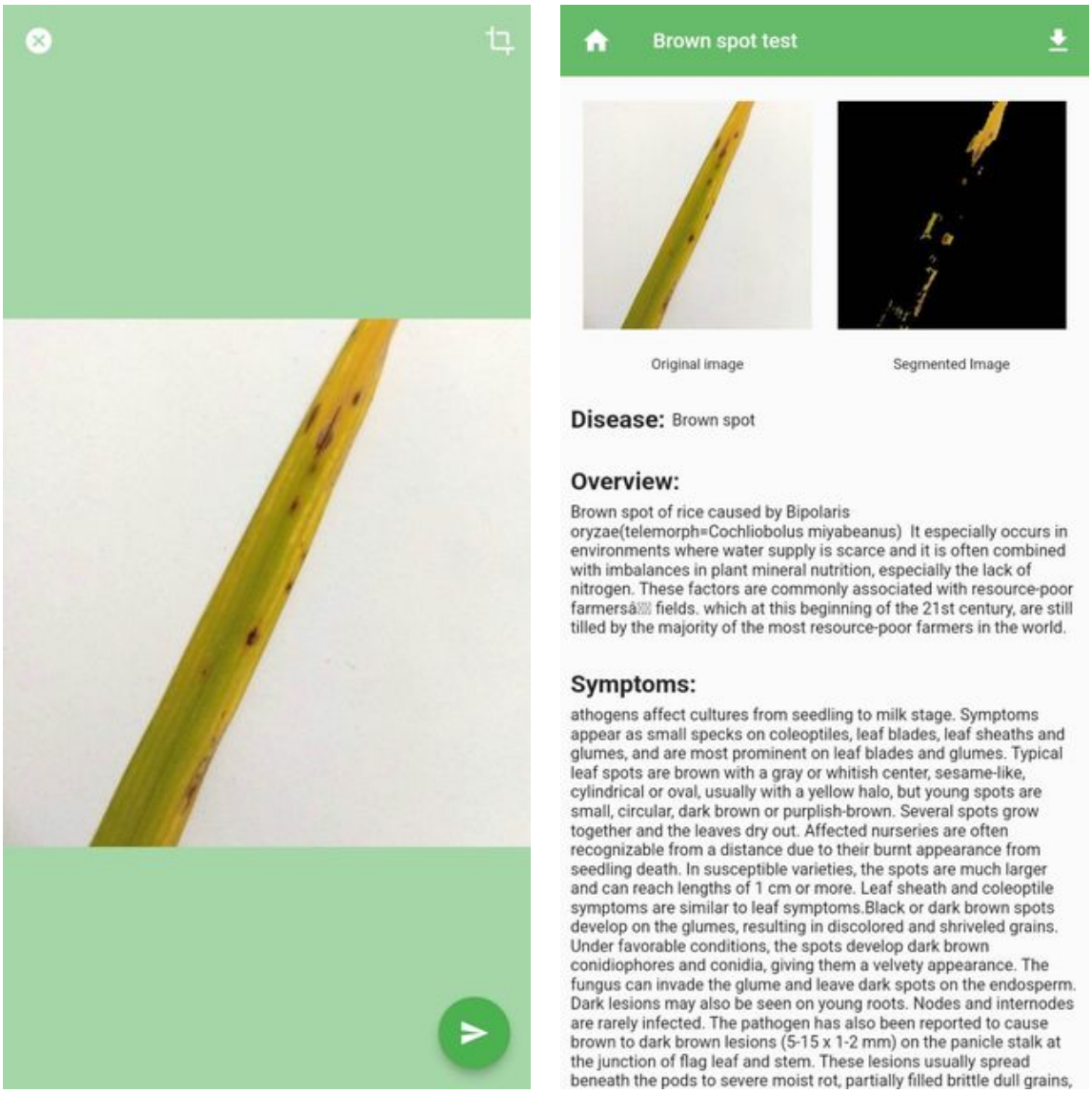


Figure 9.3: Unit Test 2

E Unit Test with Hispa Leaf

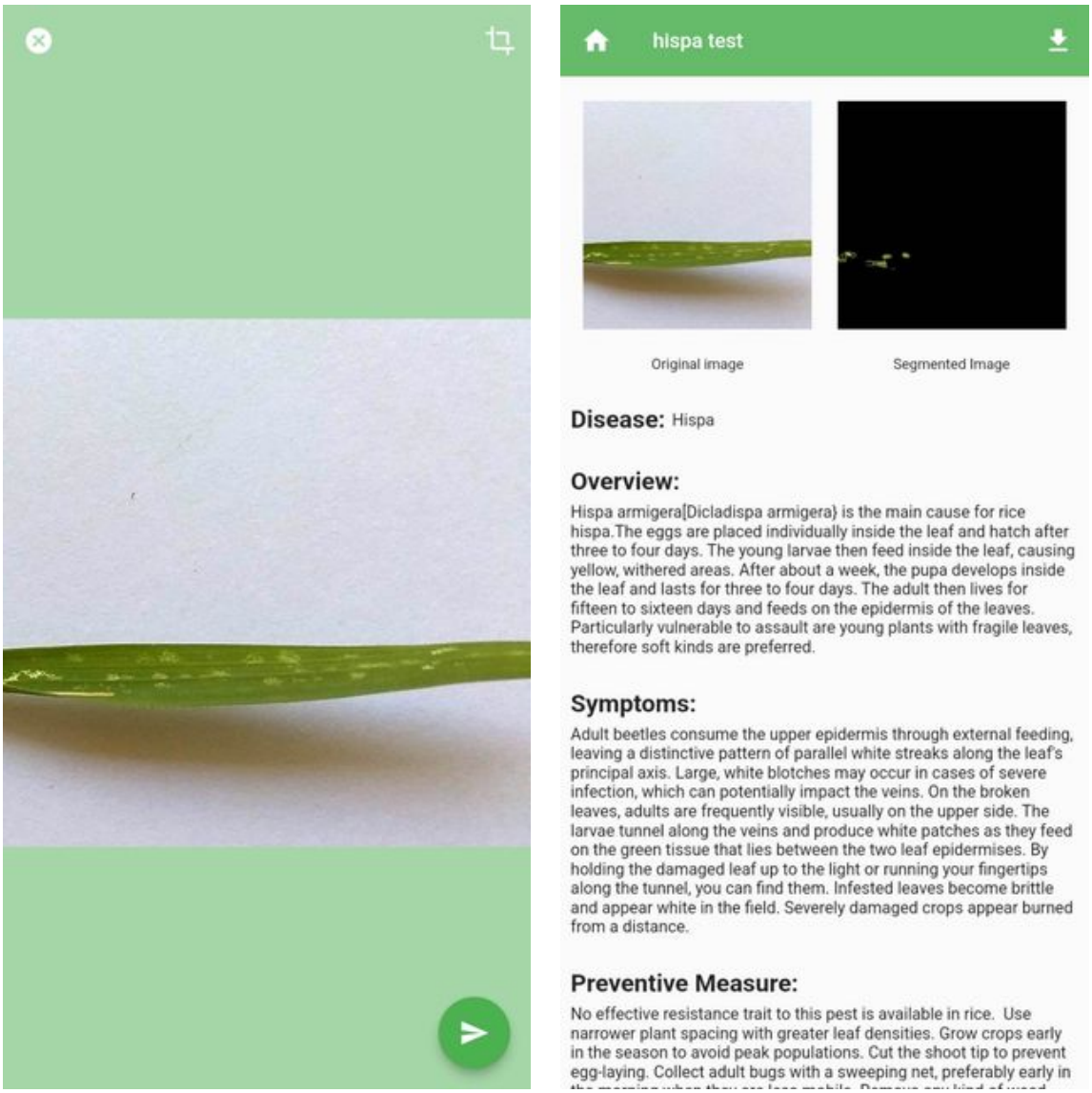


Figure 9.4: Unit Test 3

F Unit Test with Leaf Blast Leaf

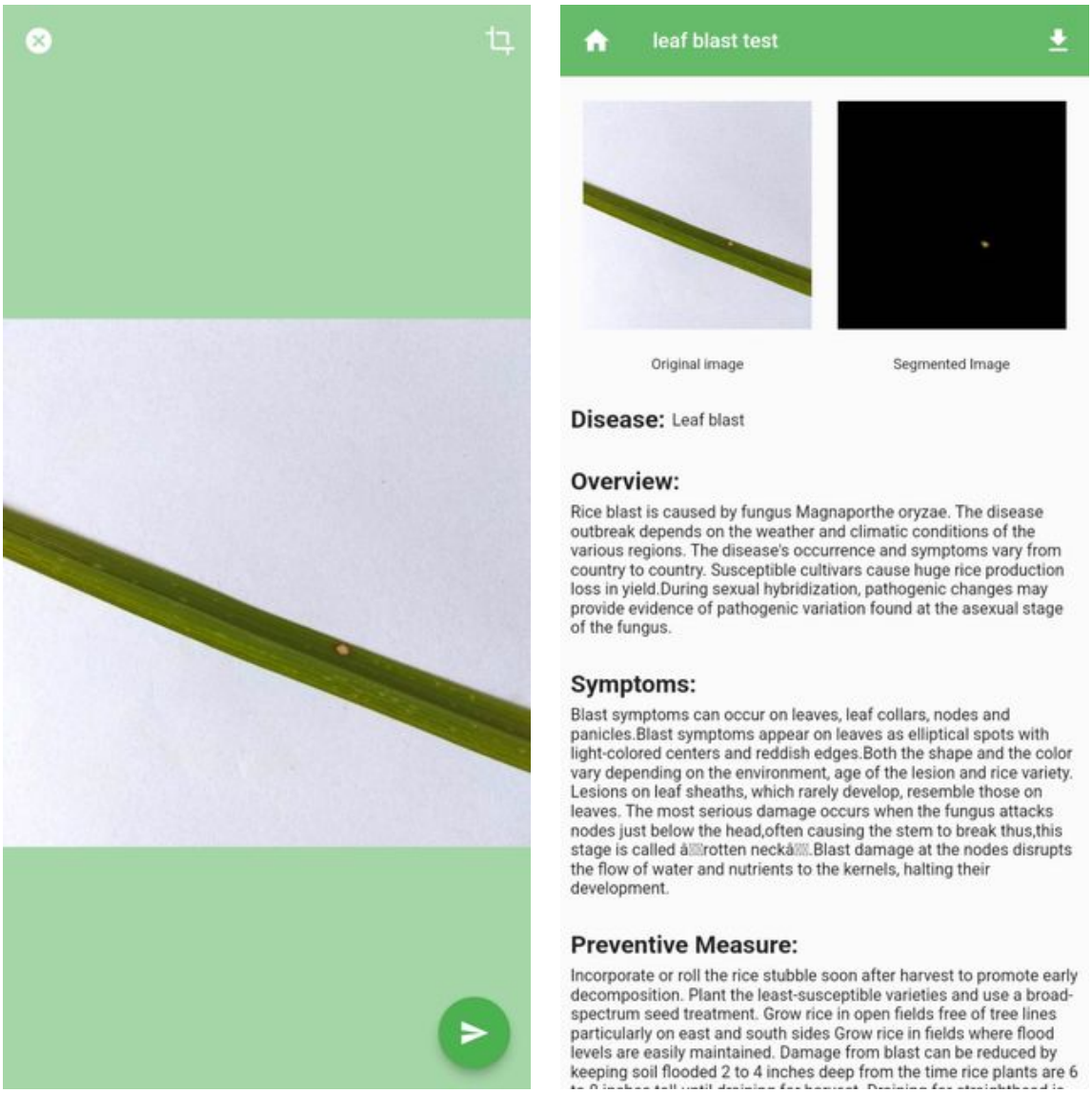


Figure 9.5: Unit Test 4

G Azure Screenshot

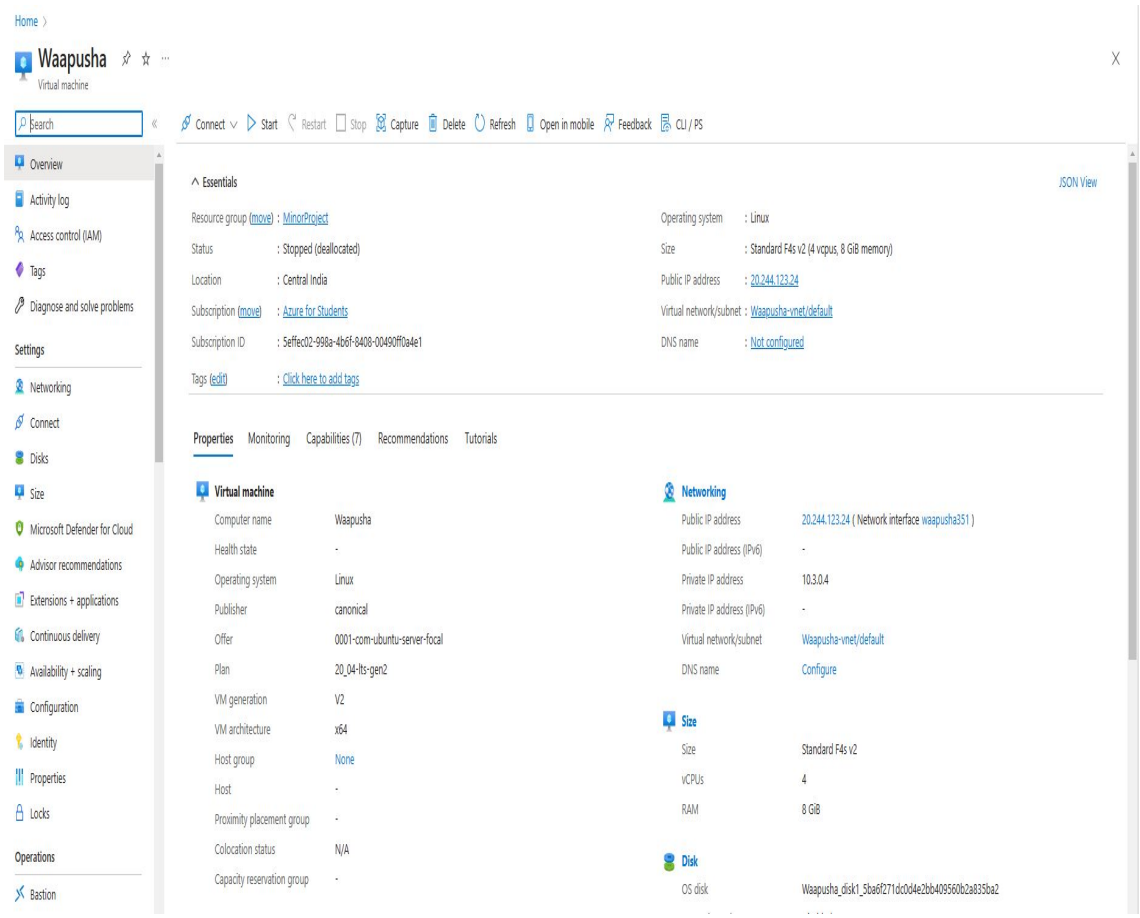


Figure 9.6: Azure Screenshot

H Server Screenshot

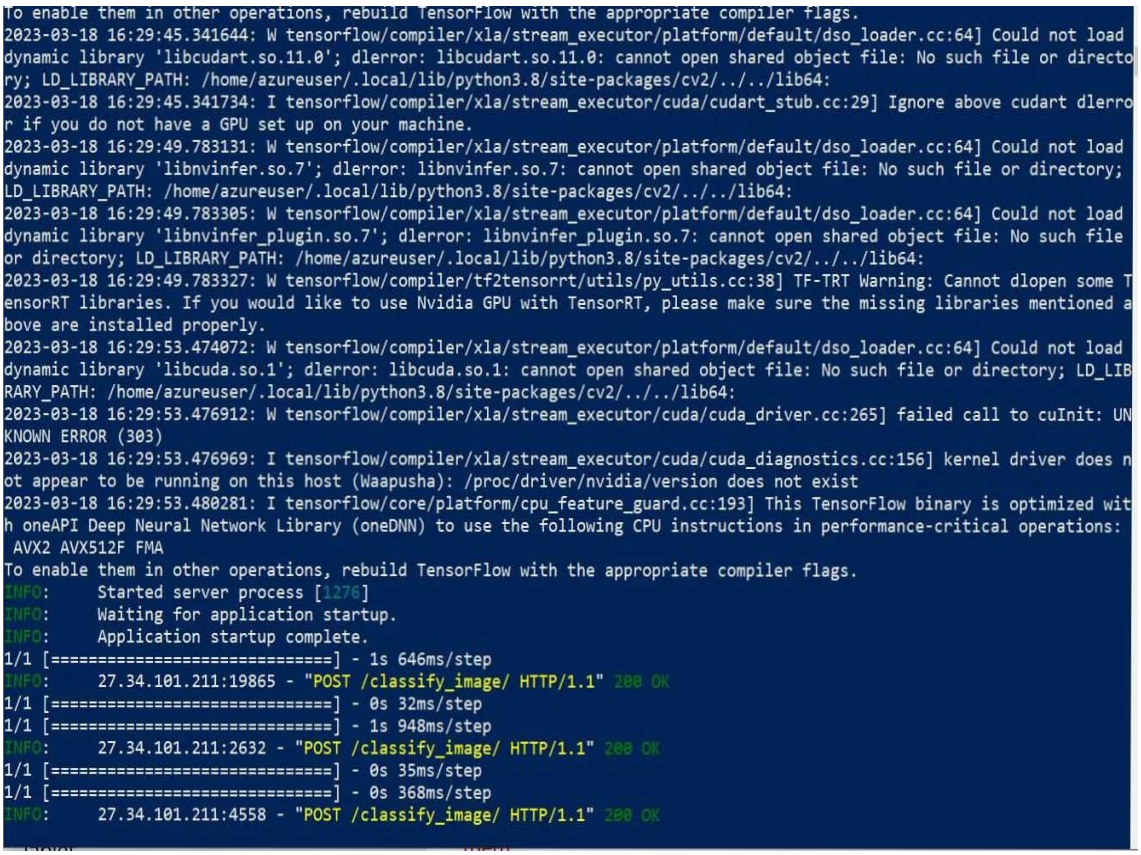


Figure 9.7: Server Screenshot