Detection and Classification of Rice Plant Diseases

by
Md. Noor Hossain
ID: CSE1701010002

Mustafizur Rahman Helal
ID: CSE2003021021

Fahima Akter
ID: CSE2003021017

Md. Biplob Mia
ID: CSE2003021046

Tariqul Islam
ID: CSE1703012020

Supervised by
Md. Ashfakur Rahman

Submitted in partial fulfillment of the requirements for the degree of Bachelor of Science in Computer Science and Engineering



DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
SONARGAON UNIVERSITY (SU)

May 2024

APPROVAL

The project titled "**Detection and Classification of Rice Plant Diseases**" submitted by Md Noor Hossain (CSE1701010002), Mustafizur Rahman Helal (CSE2003021021), Fahima Akter (CSE2003021017), Md. Biplob Mia (CSE2003021046) and Tariqul Islam (CSE1703012020) to the Department of Computer Science and Engineering, Sonargaon University (SU), has been accepted as satisfactory for the partial fulfillment of the requirements for the degree of Bachelor of Science in Computer Science and Engineering and approved as to its style and contents.

Board of Examiners

Md. Ashfakur Rahman **Supervisor**
Lecturer
Department of Computer Science and Engineering
Sonargaon University (SU)

(Examiner Name & Signature) **Examiner 1**
Department of Computer Science and Engineering
Sonargaon University (SU)

(Examiner Name & Signature) **Examiner 2**
Department of Computer Science and Engineering
Sonargaon University (SU)

(Examiner Name & Signature) **Examiner 3**

Department of Computer Science and Engineering
Sonargaon University (SU)

DECLARATION

We, hereby, declare that the work presented in this report is the outcome of the investigation performed by us under the supervision of **Md. Ashfakur Rahman**, Lecturer, Department of Computer Science and Engineering, Sonargaon University, Dhaka, Bangladesh. We reaffirm that no part of this project has been or is being submitted elsewhere for the award of any degree or diploma.

Countersigned

(**Md. Ashfakur Rahman**)
Supervisor

Signature

Md. Noor Hossain
ID: CSE1701010002

Mustafizur Rahman Helal
ID: CSE2003021021

Fahima Akter
ID: CSE2003021017

Md. Biplob Mia
ID: CSE2003021046

Tariqul Islam
ID: CSE1703012020

ABSTRACT

Rice diseases are the cause of the continuous decline in rice production. To date, the Bangladesh Rice Research Institute has identified a total of 32 diseases in Bangladesh so far in different areas and varieties of rice at different times. Out of these 10 are major diseases and the remaining 22 are minor. These diseases reduce the yield of rice by 10-15 percent. Among the diseases, 2 are viral, 3 are bacterial, 22 are fungal, 5 are helminthic. [1] Rice diseases are the cause of the continuous decline in rice production. To date, Bangladesh Rice Research Institute has so far detected a total of 32 diseases in Bangladesh in different areas and varieties of rice at different times. Out of these 10 are major diseases and the remaining 22 are minor. These diseases reduce the yield of rice by 10-15 percent. Among the diseases, 2 are viral, 3 are bacterial, 22 are fungal, 5 are helminthic. 232 species of rice pests have been identified in Bangladesh and more than 100 species are considered pests in rice production systems worldwide, but only 20 - 33 species can cause significant economic losses. Various rice diseases cause an average yield loss of 20% in Asia, which accounts for more than 90% of the world's rice production. [2] The main objective of this study is to explore all possible ways of improved and proposed technologies for rice disease detection for farmers and to reduce economic losses. Early and accurate identification is crucial to ensure effective management and abundant harvest. This is where advances in machine learning and image processing come to the rescue, providing powerful tools for disease detection and classification of rice plants in Bangladesh.

ACKNOWLEDGMENT

At the very beginning, we would like to express my deepest gratitude to the Almighty Allah for giving us the ability and the strength to finish the task successfully within the scheduled time.

We are auspicious that we had the kind association as well as supervision of **Md. Ashfakur Rahman**, Lecturer, Department of Computer Science and Engineering, Sonargaon University whose hearted and valuable support with best concern and direction acted as necessary recourse to carry out our project.

We would like to convey our special gratitude to **Prof. Bulbul Ahamed,** Head of the Department of Computer Science and Engineering, Sonargoan University and **Brig. Gen. (Retd) Prof. Habibur Rahman Kamal, NDC, PSC,** Dean, Faculty of Science and Engineering for their kind concern and precious suggestions.

We are also thankful to all our teachers during our whole education, for exposing us to the beauty of learning.

Finally, our deepest gratitude and love to my parents for their support, encouragement, and endless love.

LIST OF ABBREVIATIONS

SVM	Support Vector Machine
BRRI	Bangladesh Rice Research Institute
VS	Visual Studio
HTML	Hypertext Markup Language
CSS	Cascading Style Sheets
OS	Operating System
API	Application Programming Interface
E-R	Entity Relationship
CNN	Convolutional Neural Network

TABLE OF CONTENTS

Title		Page No.
DECLARATION …….....		iii
ABSTRACT …….		iv
ACKNOWLEDGEMENT ……..		v
LIST OF ABBREVIATIONS …….		vi
CHAPTER 1		**1-2**
INTRODUCTION TO RICE PLANTS DISEASES		
1.1	Introduction…….	1
1.2	Motivation…….....	1
1.3	Problem Statement……..	1
1.4	What is “Rice Plants Diseases?.....	2
1.5	Objectives……...	2
1.6	Chapter Review……..	2
CHAPTER 2		**3-4**
BACKGROUND OF THE PROJECT		
2.1	Introduction……	3
2.2	Background Study…….	3
2.3	Design Requirements……	3
2.4	Challenges…….	4
CHAPTER 3		**5-9**
TOOLS AND TECHNOLOGY		
3.1	Introduction…….	5
3.2	Development Tools, Technology & Resource……	5
3.3	Compatible Tools……	5-6
3.4	Backend Resource……	6-7
3.5	FrontEnd……..	7-9

CHAPTER 4 **10-18**

LOGICAL SYSTEM SPECIFICATION

4.1 Introduction…….. 10

4.2 Modules Overview…… 10

4.3 E-R Diagram……. 10-11

4.4 Use case Diagram…….. 12-13

4.5 Data-Flow Diagram…… 13-14

4.6 Database Table Folder…… 14-18

CHAPTER 5 **19-42**

USER INTERFACE AND IMPLEMENTATION

5.1 Introduction…… 19

5.2 Import Library…… 19-20

5.3 Loading Dataset……. 21

5.4 Helper Function……..... 22

5.5 Image Transfer……... 23-24

5.6 Image plot Display……. 24

5.7 Data Frame Data, Train, Validation……... 25

5.8 Data Frame Combined, Train, Validation…….. 26

5.9 Create the Train and Validation Sets…….. 27

5.10 Dataframe Train and Validation……. 28-29

5.11 Save the data frame as Compressed Files……... 30

5.12 Augmentation Image…….. 31

5.13 Display the First Images……. 32

5.14 First image in X-Train…… 33

5.15 Validation Generator…….. 34-35

5.16 X-Validation Image Show…….. 36

5.17 Train Generator…….. 37-39

5.18 X-Test Image Show…… 40

5.19 MobileNet Model Predictions……. 41

5.20 Data File…….. 42

CHAPTER 6 **43**

CONCLUSION, LIMITATIONS AND FUTURE WORKS

6.1 Conclusion…… 43

6.2 Limitations…… 43

6.3 Future Works…… 43

REFERENCES……... 44-45

LIST OF FIGURES

Table No.	Title	Page No.
Fig 4.1	E-R Diagram ……...	11
Fig 4.2	Use case Diagram ……	12
Fig 4.3	Data-flow Diagram ……..……	14
Fig 4.4	Validation & Train Dataset Folder……..	14
Fig 4.5	Validation Distinct Categories Folder…….	15
Fig 4.6	Validation Healthy Data……..	15
Fig 4.7	Validation Brown Spot Diseases Dataset……	16
Fig 4.8	Validation Hispa Diseases Data……	16
Fig 4.9	Validation LeafBlast Data…….	17
Fig 4.10	Train Distinct Categories……...	17
Fig 4.11	Train Brown Spot Data…….....	18
Fig 4.12	Train Healthy Data…….	18
Fig 4.13	Train Hispa Diseases Data……..	19
Fig 4.14	Train LeafBlast Data…….....	19
Fig 5.1	Import Library…….	18
Fig 5.2	Loading Dataset…...……..…	19
Fig 5.3	Helper Function…….……..…..…….	20
Fig 5.4	Image Transfer……..	21
Fig 5.5	Image Transfer …….…….	22
Fig 5.6	Image plt Display ……..	22
Fig 5.7	Data Frame, Data Tradin, Validation ……	23
Fig 5.8	Data Frame Combined, Train, Validation ……	24
Fig 5.9	Create the Train and Validation Sets…….…….	25
Fig 5.10	Dataframe Train and Validation ……...	26
Fig 5.11	Dataframe Train and Validation …….....	27
Fig 5.12	Save the dataframe as a Compressed Files …….	28
Fig 5.13	Augment Image …….……..	29

Fig 5.14	Display the First image …….....	30
Fig 5.15	First image in X-Train …….....	31
Fig 5.16	Validation Generator ……	32
Fig 5.17	Validation Generator …….……	33
Fig 5.18	X-Validation Image Show ……..	34
Fig 5.19	Train Generator ……...…….	35
Fig 5.20	Train Generator …….……..	36
Fig 5.21	Test Genarator …….……	37
Fig 5.22	X-Test Image Show …….……	38
Fig 5.23	MobileNet Model Predictions …….	39
Fig 5.24	Data File …….....	40

CHAPTER 1

INTRODUCTION TO RICE PLANTS DISEASES

1.1 Introduction

Rice is a staple food in Bangladesh, and diseases can significantly affect crop yields. Early and accurate detection of these diseases is crucial for farmers to take appropriate measures and reduce losses. We are making advances in **image processing and machine learning** to develop techniques to automatically detect and classify rice plant diseases based on digital images of affected plants[2].

1.2 Motivation

Rice is the staple food for a large part of the world's population, especially in Bangladesh. Rice plant diseases can significantly reduce crop yield and farmers' economic income. Early detection and classification of rice plant diseases is crucial for implementing appropriate control measures to minimize yield losses. Traditional methods [3] of disease detection rely on visual inspection by agricultural experts, which can be time-consuming, labor-intensive, and subjective. Rice plant disease detection and classification using **machine learning and image processing techniques** offers several advantages over traditional methods. Early detection of rice plant diseases allows farmers to take timely measures to control disease spread and reduce yield losses.

1.3 Problem Statement

Rice production is significantly affected by various plant diseases. These diseases can cause substantial yield losses, reduce crop quality, and threaten food security. Early and accurate identification and classification of rice diseases is crucial for implementing effective disease management strategies and minimizing crop losses. Some of the major problems that may be faced in our country are[4]:

1. Traditional methods of detecting rice diseases rely on visual inspection by farmers or agronomists. This method is time-consuming, subjective, and prone to human error, especially in the early stages of infection when symptoms are subtle.

2. The appearance of disease symptoms can be influenced by environmental factors such as light conditions, moisture levels, and the growth stage of the rice plant. This variability makes it difficult to develop a universal identification and classification system.

3. Existing diagnostic tools can be expensive or require specialized equipment, making them inaccessible to small-scale farmers in developing countries.

1.4 What is "Detection and Classification of Rice Plant Diseases"?

Detection and Classification of Rice Plant Diseases is a technical process to apply image recognition and machine learning techniques to develop methods for automatic detection and classification of Rice Plant diseases[5]. Early and accurate detection and classification of these diseases are crucial for implementing appropriate control measures and minimizing yield losses.

1.5 Objectives

The main objectives of detecting and classifying rice plant diseases are:

1. Reduce yield loss
2. Improve crop management
3. Enhance food security
4. Increased Rice Production [6]

1.6 Chapter Review

In Chapter 1: Chapter 1 explains why early rice disease detection is vital for Bangladesh. Traditional methods are slow and inaccurate. Machine learning offers a solution for automatic disease identification using image recognition. This improves food security by minimizing crop losses.

In Chapter 2: This chapter explores the need for an AI-powered rice disease detection system. It highlights how image processing and machine learning can overcome traditional method limitations, enabling early disease diagnosis for improved crop management and food security.

In Chapter 3: It constitutes both a super and physical corporation. The deep learning model runs the analysis in the cloud platforms such as Google Colab, on the other hand, the experimental phase requires machine learning libraries (TensorFlow, Keras) for training the disease-identifying image analysis model. A web interface is made with technologies that are ever so familiar to the users, CSS, JavaScript, and HTML.

In chapter 4: The fourth chapter describes the plan and documentation for the rice disease detector model of farmers and researchers. The system saves uploaded rice plant images and performs the identification of diseases accompanied by recommendations and guidelines. Deep learning utilizes images for classification purposes. The chapter includes data flow and folder requirements to be trained and tested.

In chapter 5: Chapter 5 builds the rice disease app. It covers setting up the environment, data loading, and splitting for training, validation, and testing. It also details data augmentation and shows sample image visualizations.

CHAPTER 2

BACKGROUND OF THE PROJECT

2.1 Introduction

Disease detection from a plant image is an interesting research area in the agricultural field, for which machine learning concepts from the computer field can be applied. This article presents a prototype system for rice disease detection and classification based on images of infected rice plants. This prototype system has been developed after a detailed experimental analysis of various techniques used in image processing operations. We consider three diseases of rice plants namely bacterial leaf blight, brown spot, and leaf smut. We capture images of infected rice plants using a digital camera from a paddy field. We experimentally evaluate four background removal techniques and three segmentation techniques. To enable accurate extraction of features, we propose K-means clustering based on centroid feeding to segment disease areas from leaf images. We increase the output of K-means clustering by removing green pixels in disease areas. We extract different features under three categories: color, shape, and texture. We use a Support Vector Machine (SVM) for multi-class classification[7].

2.2 Background Study

This background study will explore the current state of the art in detecting and classifying rice plant diseases. We are discussing the different types of rice diseases, the challenges involved in their identification, and the techniques employed for automated identification and classification. We will examine the major rice diseases caused by bacteria, fungi and viruses. Each disease will be characterized by causative agents, symptoms, and effects on rice plants. Through Detection and Classification Techniques we will explore various techniques used for automatic detection and classification of rice plant diseases[8]. These may include image processing, machine learning, and deep learning approaches.

This background study aims to gain a comprehensive understanding of the existing knowledge and challenges in rice plant disease identification and classification. This background study will be essential to further explore the potential of improved strategies and contribute to the development of robust and efficient disease management strategies.

2.3 Design Requirements

When we design software or systems, we consider certain requirements to make the project efficient. We have tried to build our model properly. From which the most accurate results of rice disease problems are identified. Detailed information on how our developed model works, images, model training, model training results, case diagrams, activity diagrams are provided for ease of understanding. We used real images to build our database. Besides this, I have collected two databases online to train the data. We used Google Colab to build our entire model. Here the detection of rice disease problems is done through image processing through convolution neural networks of machine learning.

2.4 Challenges

Every work has some challenges. So, we also have to face some challenges.

- The main data processing in our model is image processing which requires an updated environment, strong data flow, large storage capacity, and efficient date processing capability. But we don't have any such local M/E. So we don't work in the local environment.

- So we use the virtual environment, Google Club. Google Colab's main limitation is temporary memory/sessional memory. All data will be deleted after each session ends. To overcome this problem we introduced Google Drive with Colab.

- Horsni load shedding and network service are currently a major problem due to the natural environment. So we constantly face this kind of problem during our work.

- With the continuous help, encourg and teamwork of our supervisors we overcame all the errors and finally completed the work.

- As of now, Bangladesh Rice Research Institute has developed 106 varieties of rice and has identified 18 of the main diseases of rice including fungal and non-fungal diseases. It is difficult and time-consuming to collect actual images of all the above varieties of rice diseases.

- Our model is not suitable for use as a real project at this time. To implement it as a real project, we have to work for a long time.

CHAPTER 3

TOOLS AND TECHNOLOGY

3.1 Introduction

The next paragraph deals with a discussion of the core software and hardware that is vital for implementing the system intended for the diagnosis and classification of rice diseases. With this conversation, the components of hardware, software, and methods that were significant as part of our research will be up for discussion.

3.2 Development Tools, Technology & Resource

Development tools, technologies, and resources are used to bring machine-learning projects to life. By using these elements effectively in machine learning projects, we can streamline our development process, build robust models, and achieve successful project outcomes.

3.3 Compatible Tools

- **Google Colab:**



Google Colab (short for collaboration) is a game changer for aspiring data scientists and machine learning enthusiasts. It offers a cloud-based Jupyter Notebook environment that eliminates the need for heavy software installation or powerful local machines. Google Colab empowers anyone with an internet connection to explore the fascinating world of machine learning. Its free cloud resources, user-friendly interface, and collaborative features make it an ideal platform for testing, learning, and even professional machine-learning projects.

- **Visual Studio:**



While Visual Studio (VS) is traditionally known for its back-end development capabilities, it also offers a surprisingly powerful tool for front-end web development. This makes it a compelling choice for developers who want a unified environment for both sides of the web development coin.

VS supports the main front-end languages: HTML, CSS, and JavaScript. The IntelliSense feature provides intelligent code completion, helping you write code faster and with fewer errors. For CSS, class names and attributes pop up as you type, while HTML benefits from

context-aware suggestions for tags and attributes. Enjoying similar behavior to JavaScript, IntelliSense offers function definitions, variables, and an object member based on your project and includes libraries.

- **GitHub**



GitHub projects offer a powerful way to streamline your workflow by connecting different elements of your development process. This functionality goes beyond simple linking and encourages a more organized and collaborative environment.

3.4 Backend Resource

A brief description of the resources imported and what they are used for when our system works through Google Collab:-

- **OS:** for interacting with the operating system (file paths etc.). Python uses the import OS statement to access operating system (OS) functionality from our Python code. OS modules provide a collection of functions that allow us to interact with the underlying operating system, be it Windows, macOS or Linux.
- **Random:** to generate randomness (useful for shuffling data). In Python, we usually use import random to access functionality provided by the random module. This module offers a collection of functions to generate pseudo-random numbers and perform operations that rely on randomness.
- **Zip file:** for working with zip files (if our data is compressed). Google Colab is a cloud-based environment, so we can upload our data as zip files for convenience. The zipfile module allows us to unzip this data and access the files within the archive. This is useful for working with datasets or code repositories stored in zip format
- **Glob:** to find all files matching a specific pattern in a directory. We'll use glob import in Python for Google Colab, especially when we want to work with multiple files that follow a specific pattern within a directory. This provides a convenient way to find these files without having to list them individually in our code
- **Matplotlib.pyplot:** for data visualization (making pictures). Using matplotlib.pyplot empowers us to create informative and visually appealing data visualizations within Google Colab notebooks, facilitating data exploration, analysis and communication.
- **As TensorFlow TF:** The core TensorFlow library. Using import tensorflow as tf in Colab provides a convenient, readable, and standard way to work with TensorFlow libraries while avoiding potential naming conflicts.
- **Keras.layers:** Provides building blocks for neural network layers (dense, flat, etc.). Effectively using keras.layers, we can use the combined power of Keras and TensorFlow to build powerful and flexible deep learning models in Google Colab. The keras.layers module of TensorFlow's Keras API provides a comprehensive set of pre-built layers that we can use to build our models. These layers offer a high level of

abstraction, which we can focus on designing our model architecture without getting bogged down in the low-level details of TensorFlow operations.

- **Keras.models:** for creating and managing Keras models. By using keras.models, we gain a user-friendly way to interact with the core building blocks of our deep learning models in Colab, making the development process more efficient and readable.
- **Keras.preprocessing:** utility for image preprocessing and data augmentation. The keras.preprocessing module of Keras (often used with TensorFlow) provides the tools we need to prepare our data before feeding it to our deep learning model.
- **Tensorflow.keras.preprocessing.image:** Image specific preprocessing functions from TensorFlow. Effectively using tensorflow.keras.preprocessing.image (or its alternatives), we can streamline our image preprocessing workflow in Google Colab and increase the performance of our deep learning models working with image data.
- **Keras.utils:** Helper functions like one-hot encoding labels. The keras.utils import in TensorFlow provides valuable utilities that make deep learning development in Google Colab more efficient, maintainable, and consistent. By incorporating these functions into our projects, we can streamline our workflow and focus on building effective neural network models.
- **Sklearn.metrics:** Evaluation metrics like accuracy and classification reports. We use the sklearn.metrics import in Python for Google Colab to access a collection of functions designed to evaluate the performance of machine learning models. Scikit-learn (often abbreviated as sklearn) is a popular Python library that provides a variety of tools and techniques for machine learning, and sklearn.metrics is a submodule of it.
- **Sklearn.model_selection:** function to split data into training and test sets. The sklearn.model_selection import is commonly used for machine learning tasks in Python, and Google Colab is a popular environment for running machine learning code. By using sklearn.model_selection, we can ensure that our machine learning workflow in Google Colab is robust and statistically sound.
- **Sklearn.preprocessing:** data preprocessing technique (in this case OneHotEncoder). In machine learning, data preprocessing is an important step that prepares our data for model training. This ensures consistency and improves the performance of machine learning algorithms. scikit-learn's sklearn.preprocessing module offers a variety of tools for data preprocessing tasks in Python.

3.5 FrontEnd

- **Hypertext Markup Language (HTML)**



Hypertext Markup Language (HTML) serves as the basic building block of web pages. It is not a programming language in the traditional sense, but a markup language that provides

structure and organization of content. HTML is a blueprint, which outlines the various sections (headings, paragraphs, images, lists, etc.) that make up a web page.

- **Cascading Style Sheets (CSS)**



Cascading Style Sheets (CSS) is the language of aesthetics, which dictates the visual presentation of your web pages. Here you transform basic HTML structure into a visually appealing experience. Bring your website to life with effects like CSS, colors, fonts, layouts and artist palettes.

- **JavaScript**



JavaScript, unlike HTML and CSS, is a true programming language. It injects interactivity, dynamism and responsiveness into your web pages. Think of JavaScript as an engine that makes things happen, allowing you to create features like forms, animations, interactive elements, and more. JavaScript code can be embedded directly into HTML pages or included as a separate file.

CHAPTER 4

LOGICAL SYSTEM SPECIFICATION

4.1 Introduction

Identification and classification of rice plant diseases is one of the major problems not only in Bangladesh but also in the world. Various diseases of rice plants hurt the growth of the crop. Rice plant diseases are not detected in time, but they have the potential to have disastrous effects on food security. If the diseases of rice plants can be detected in the early stages, the unnecessary use of pesticides will be reduced under normal conditions. As a result, it can play a special role in ensuring effective yield from rice plants.[9]

4.2 Modules Overview

Since there is no end user in our model module, here the actor can see the output by inputting the data. In between our system will work as data processing and analysis. So, there are 2 parts in our system. That is: -

1. Actor
2. System or model

Actor:

The actors here are farmers, agriculturists, and scholars. Those who will use our system as actors will not need to register or log in separately. You can use it completely Hassle.

System or Model:

Our system or model provides results considering a total of 6 aspects of a rice plant.

1. Add observations
2. Find possible diseases
3. Compose observations
4. Look for possible communicable diseases
5. Find potential controls
6. Find possible abnormalities of the given disease [10]

4.3 E-R Diagram

Describes the entire process of research. Firstly, the user will upload a picture of a defective Rice plant to the system. Then the system will search the image-related information in the database, if it matches the image uploaded to the system database, an output will show the disease name and if no image matches, then it will exit. [11]

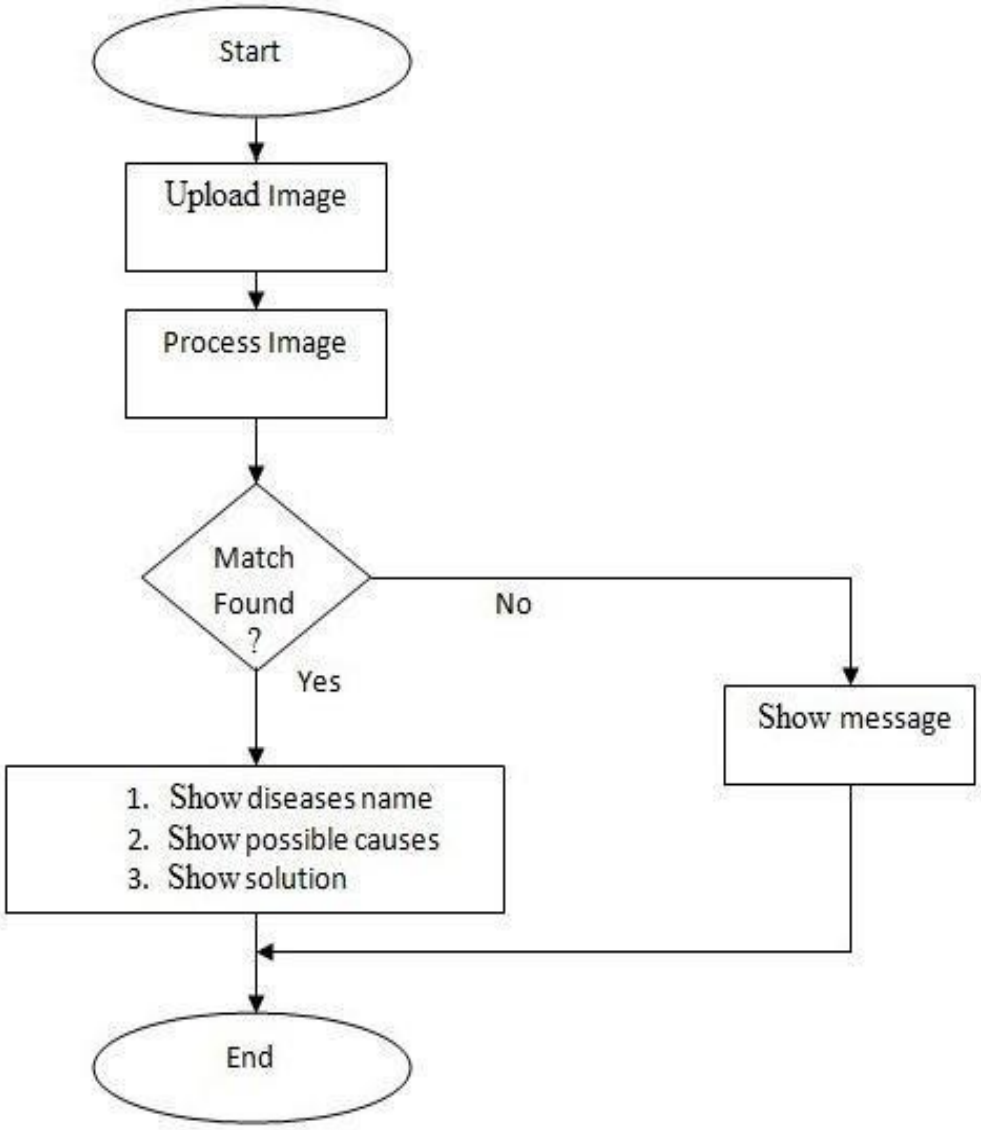


Fig 4.1: E-R Diagram

4.4 Use Case Diagram

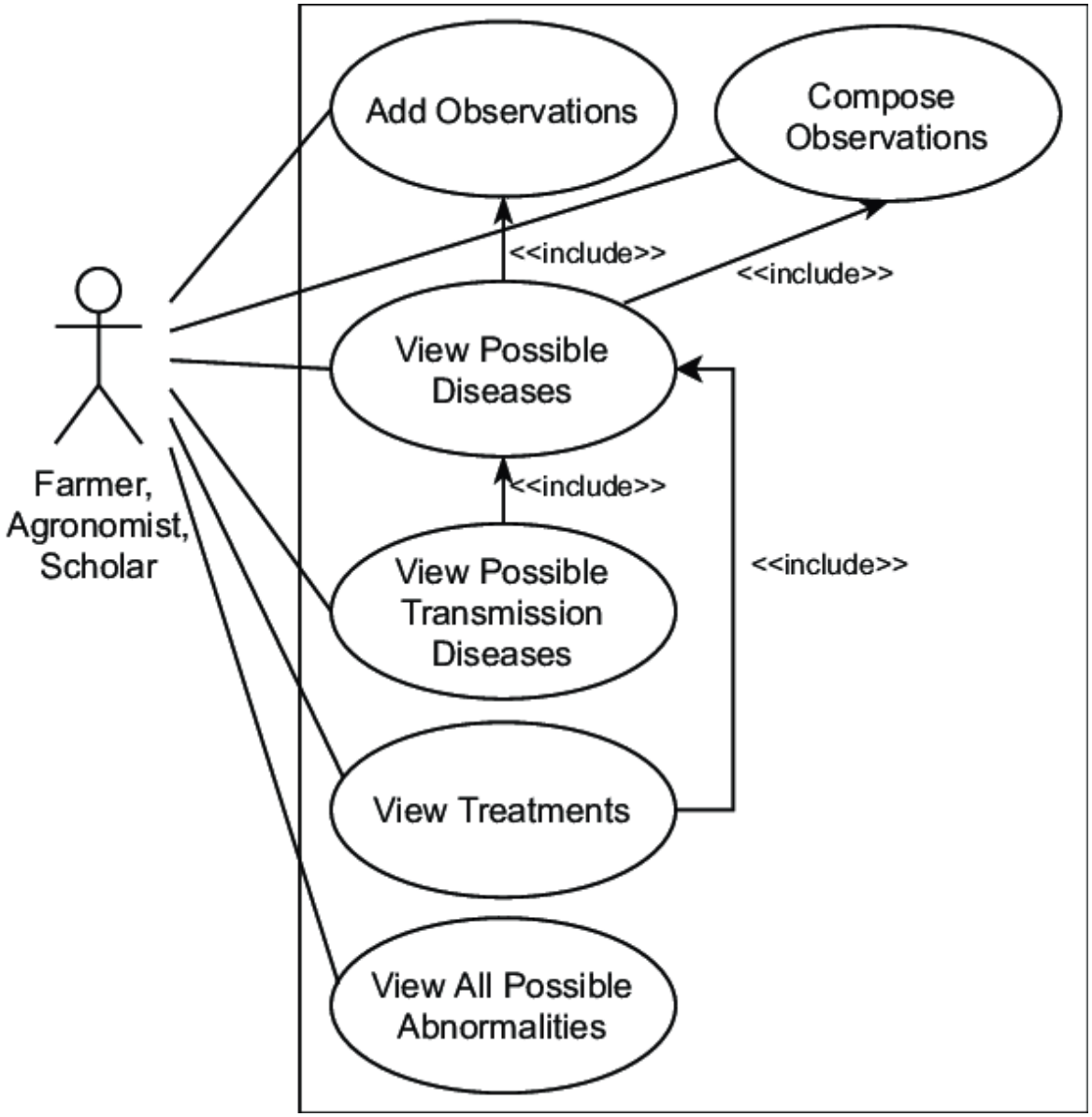


Fig 4.2: Use Case Diagram

Add observation: Our system will be very helpful for easy digital monitoring of paddy plant diseases. By uploading pictures, you can easily understand the problem of rice plants.

Find possible diseases: Our model will help to find potential diseases of both apparent and latent diseases related to rice plant image observation.

Record observations: Identifying diseases of rice plants is easy to record. By using this system, observations are composed. Which can be an easy way to inform the nearby farmer. It is often seen that rice diseases and rice pests spread to nearby areas. As a result, farmers can take precautions in advance.

Find possible communicable diseases: Image helps detect potential communicable diseases. As a result, users or farmers discussing potentially transmissible diseases can quickly identify diseases in rice plants and inform others about them. This will also reduce the number of rice diseases.

Find potential controls: By comparing the image of the tree with other images, the system will help to find possible diseases and possible controls. If the correct disease of the rice plant can be identified, it can be taken care of quickly. By doing this, the possibility of disease in rice plants will be reduced.

Detect possible abnormalities of a given disease: Rice plants can return all possible abnormal appearances only when the rice plant is infected with a particular disease. This enables users to compare their observations with the retrieved results and make a better decision.

4.5 Data-Flow Diagram

The model we proposed for rice foliar disease classification in our work was able to identify and classify a total of four distinct categories. These four distinct categories are Brown Spot, Healthy, Hispa, and Leaf Blast. In the proposed deep CNN transfer learning-based approach, the images will go to the preprocessing stage, where the images will undergo background removal, resizing, and enhancement. In this work, we apply data augmentation, which applies slight changes to original images to create new distinct images. Minor changes may include rotation, scale-in/scale-out, and translation. Features are then extracted using CNN. CNN uses feature reduction using smooth, dense, and SoftMax layers, and the last layers perform classification. We evaluate our proposed approach using the following metrics; Accuracy, precision, and F1-measure. [12]

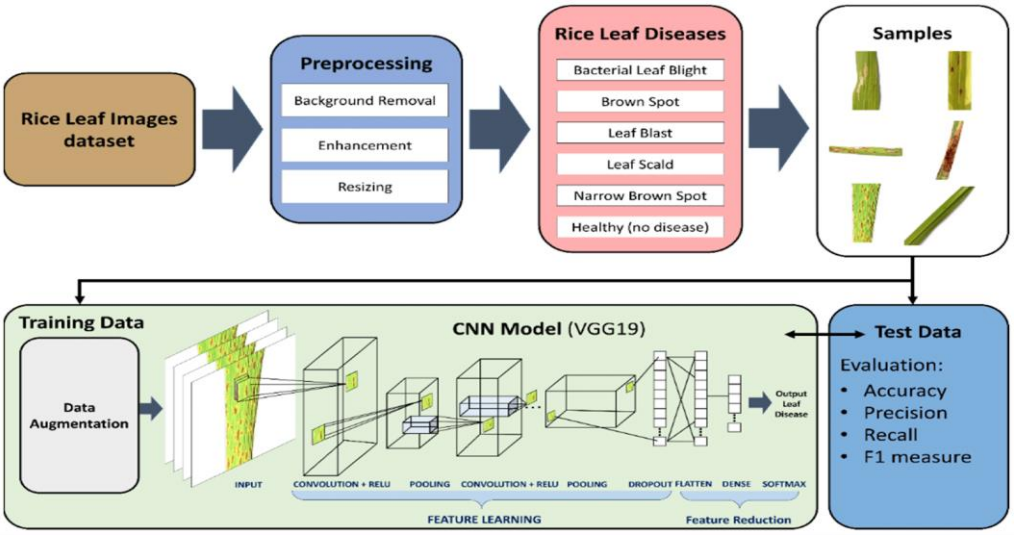


Fig 4.3: Data-Flow Diagram

4.6 Database Table Folder

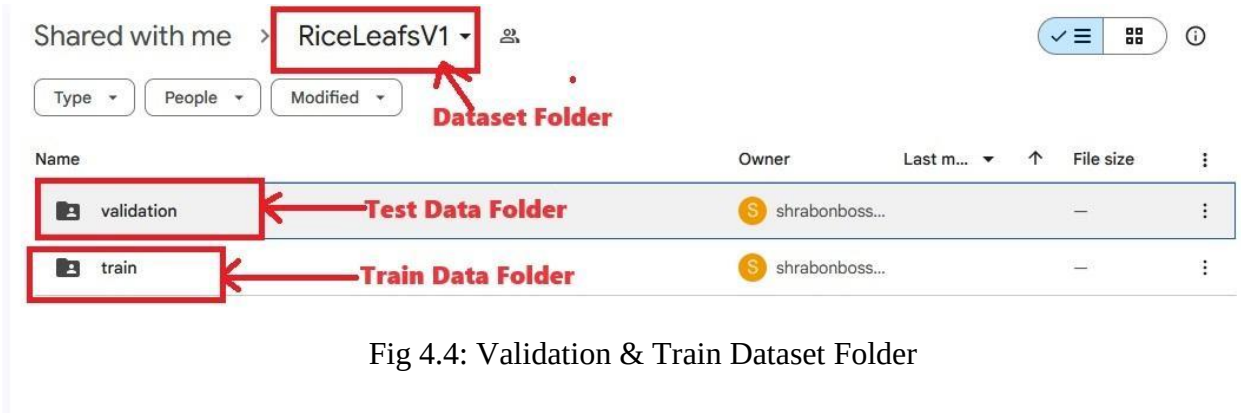


Fig 4.4: Validation & Train Dataset Folder

4.7 Validation Distinct Categories Folder

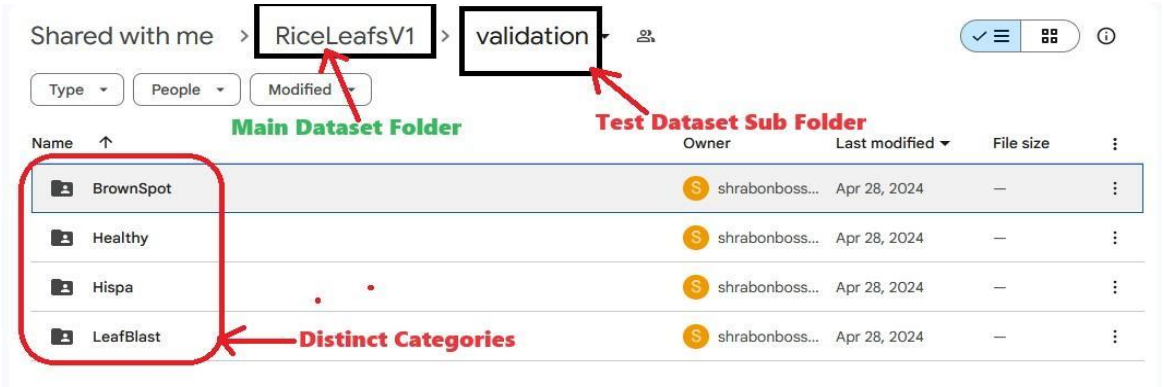


Fig 4.5: Validation Distinct Categories Folder

4.8 Validation Healthy Data:

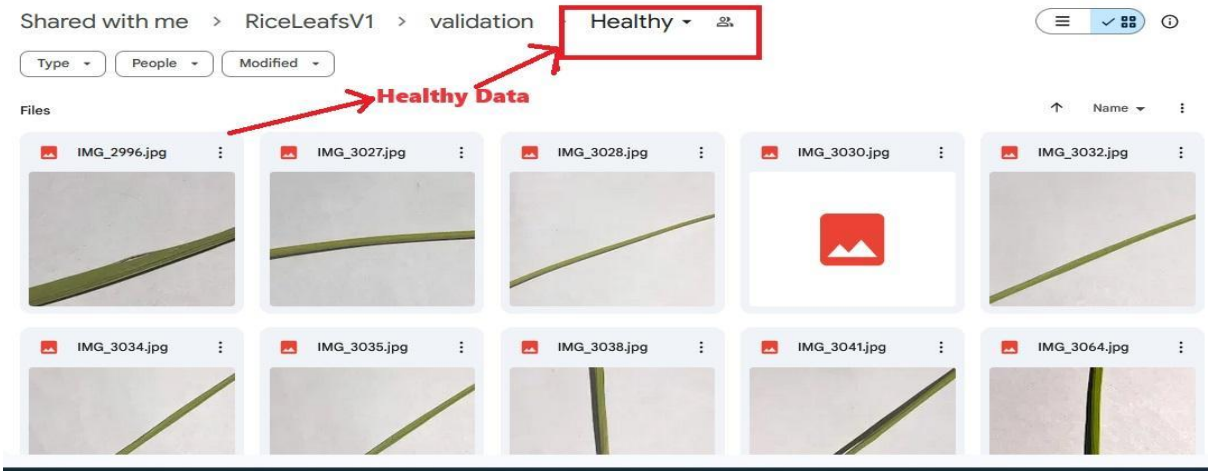


Fig 4.6: Validation Healthy Data

4.9 Validation Brown Spot Diseases Dataset:

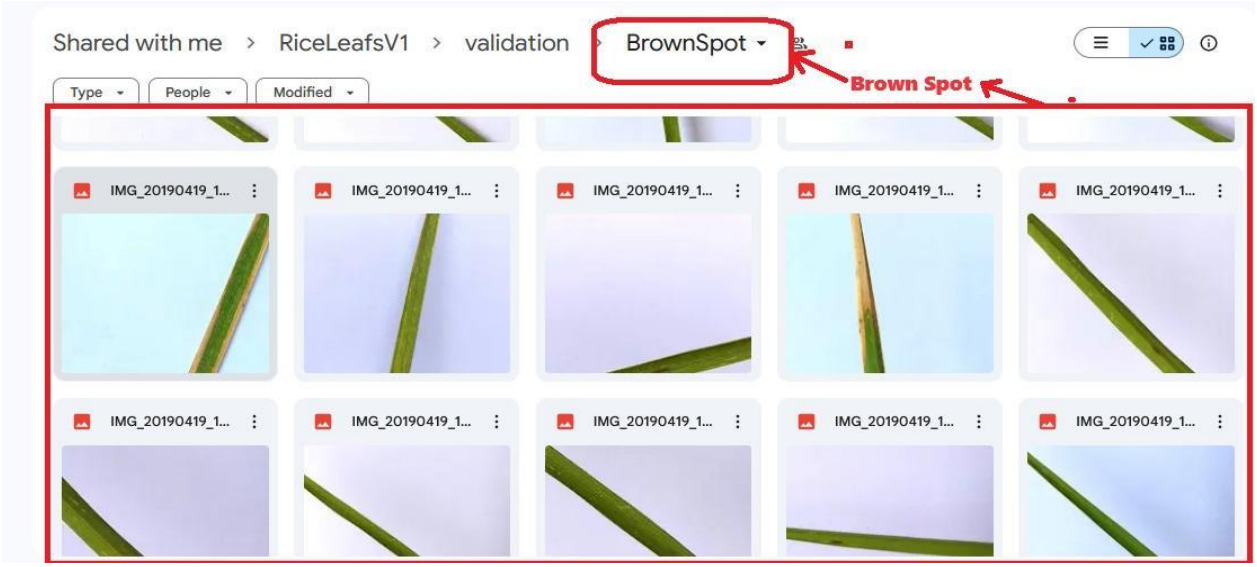


Fig 4.7: Validation Brown Spot Diseases Dataset

4.10 Validation Hispa Diseases Data:



Fig 4.8: Validation Hispa Diseases Data

4.11 Validation LeafBlast Dataset:

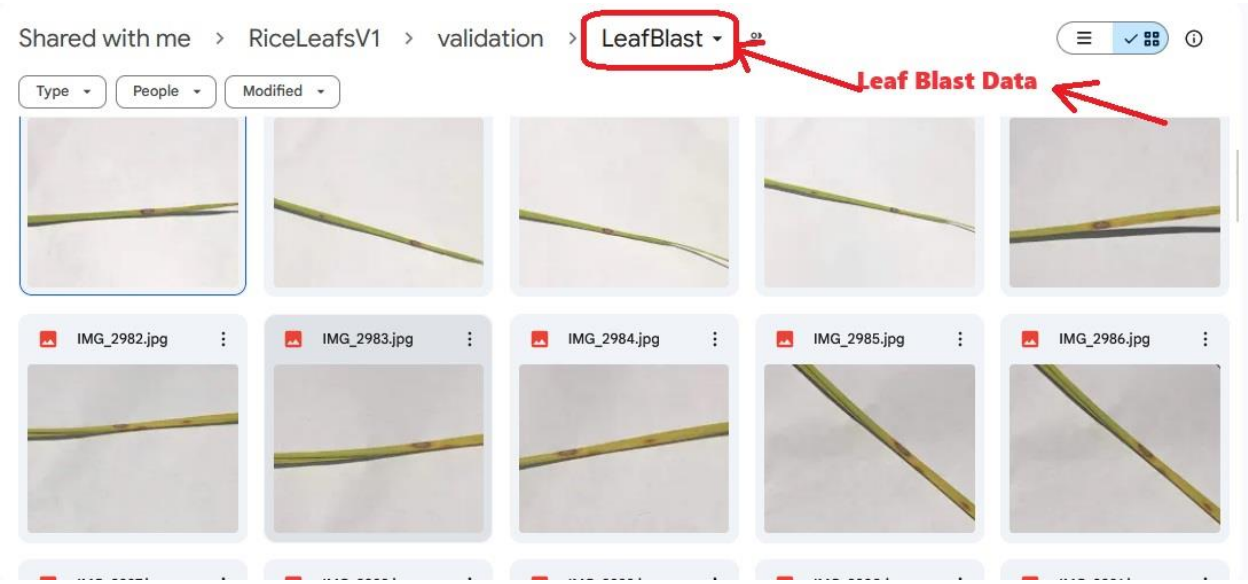


Fig 4.9: Validation LeafBlast Data

4.12 Train Distinct Dataset:



Fig 4.10: Train Distinct Categories

4.13 Train Brown Spot Data:

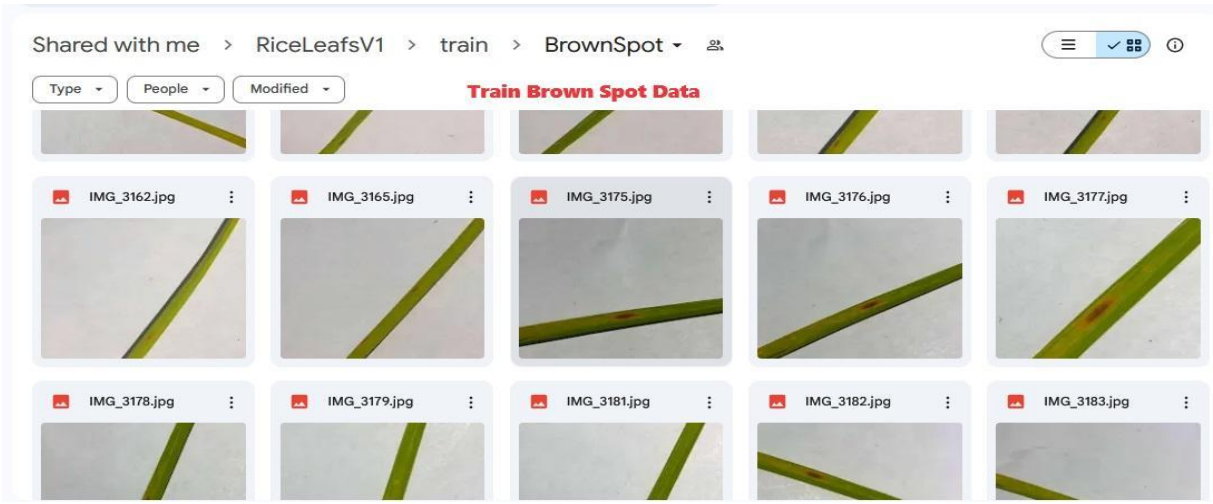


Fig 4.11: Train Brown Spot Data

4.14 Train Healthy Data:



Fig 4.12: Train Healthy Data

4.15 Train Hispa Diseases Data:

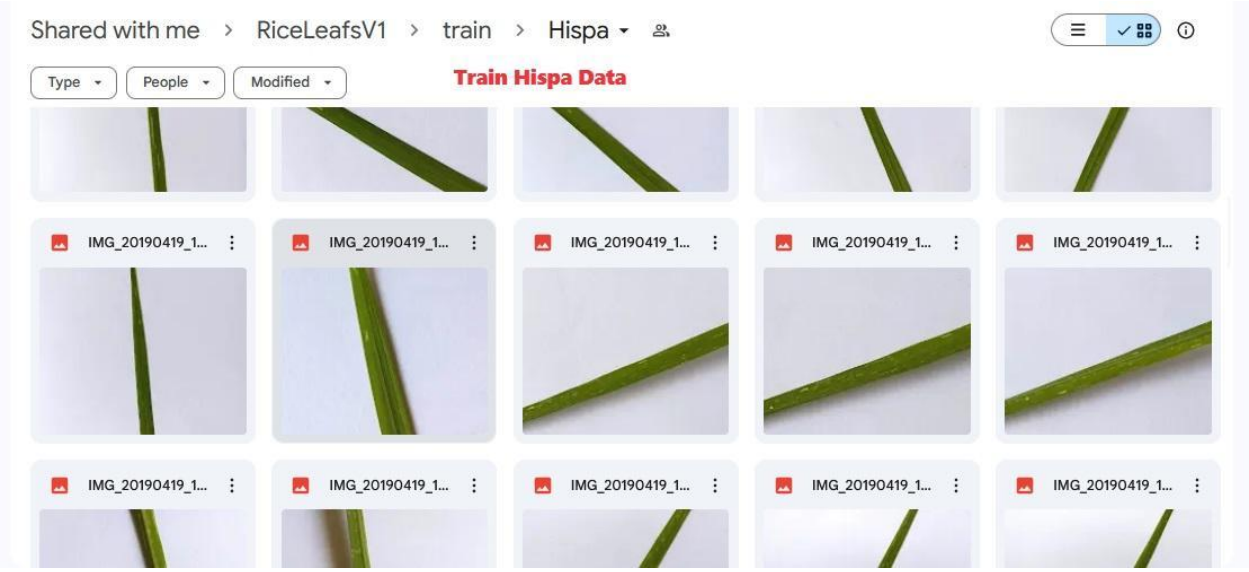


Fig 4.13: Train Hispa Diseases Data

4.16 Train Leaf Blast Data:

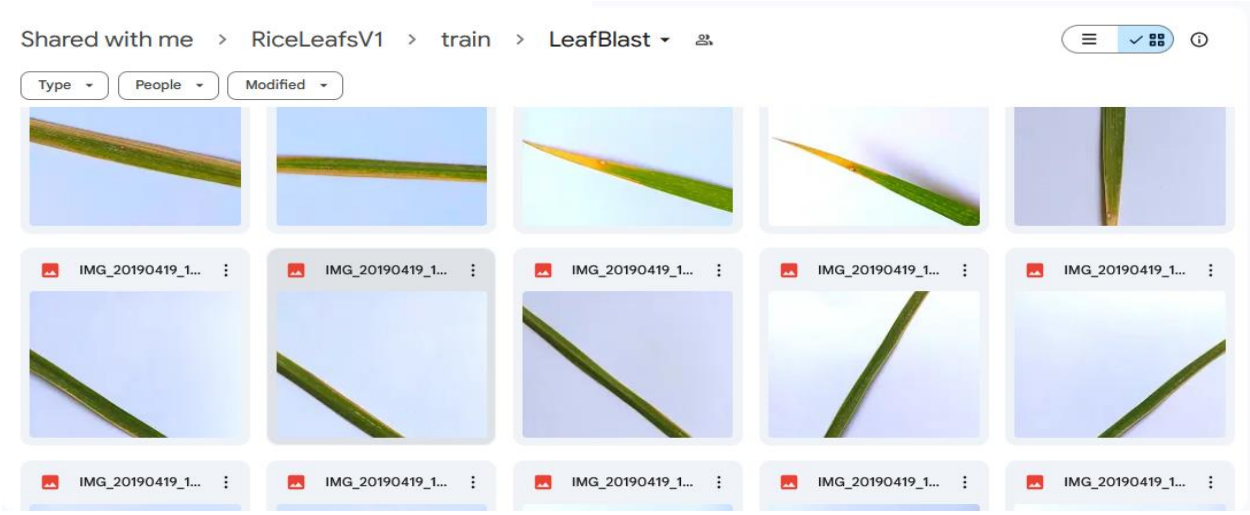


Fig 4.14: Training Leaf Blast Data

CHAPTER 5

USER INTERFACE AND IMPLEMENTATION

5.1 Introduction

In this chapter, we will create and develop our project, review, run, and test our application. Testing results will be displayed to verify the eligibility and quality of the application from Google Colab.

5.2 Import Library

Importing Libraries

```
[6]  # Check the tensorflow version

     import tensorflow as tf
     tf.__version__
```

```
'2.16.1'
```

```
import pandas as pd
import numpy as np
import os

import cv2

import albumentations as albu
from albumentations import Compose, ShiftScaleRotate, Resize
import albumentations as A
from albumentations.pytorch import ToTensorV2

from sklearn.model_selection import train_test_split
from sklearn.utils import shuffle

from sklearn.metrics import confusion_matrix
import itertools
from sklearn.metrics import classification_report

import shutil

import matplotlib.pyplot as plt
%matplotlib inline
```

Fig 5.1: Import Library

5.3 Loading Dataset

Loading Dataset

```
[10] from google.colab import drive
     drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

```
import os
from google.colab import drive  # Mount Google Drive if necessary

# Assuming mounted drive (uncomment if needed)
# drive.mount('/content/drive')

data_dir = '/content/drive/MyDrive/RiceLeafsV1/train'

brown_spot_dir = os.path.join(data_dir, 'BrownSpot')
healthy_dir = os.path.join(data_dir, 'Healthy')  # Assuming a "Healthy" folder
hispa_dir = os.path.join(data_dir, 'Hispa')
leaf_blast_dir = os.path.join(data_dir, 'LeafBlast')

# Create directories if they don't exist (optional)
# os.makedirs(brown_spot_dir, exist_ok=True)  # Uncomment if needed
# ... (similar for other directories)

brown_spot_list = os.listdir(brown_spot_dir)
healthy_list = os.listdir(healthy_dir)
hispa_list = os.listdir(hispa_dir)
leaf_blast_list = os.listdir(leaf_blast_dir)

print(len(brown_spot_list))
print(len(healthy_list))
print(len(hispa_list))
print(len(leaf_blast_list))
```

```
100
100
100
100
```

Fig 5.2: Loading Dataset

5.4 Helper Function

```
[8]  IMAGE_HEIGHT = 224
     IMAGE_WIDTH = 224
     IMAGE_CHANNELS = 3
```

Helper Functions

```
def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):
    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    """
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        print("Normalized confusion matrix")
    else:
        print('Confusion matrix, without normalization')

    print(cm)

    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    fmt = '.2f' if normalize else 'd'
    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, format(cm[i, j], fmt),
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

    plt.ylabel('True label')
    plt.xlabel('Predicted label')
    plt.tight_layout()
```

Fig 5.3: Helper Function

5.5 Image Transfer

```
brown_spot_list = os.listdir(brown_spot_dir)
healthy_list = os.listdir(healthy_dir)
hispa_list = os.listdir(hispa_dir)
leaf_blast_list = os.listdir(leaf_blast_dir)

# Transfer the brown_spot images
for fname in brown_spot_list:

    path = '/content/drive/MyDrive/RiceLeafsV1/train/BrownSpot'

    # source path to image
    src = os.path.join(path, fname)
    # destination path to image
    dst = os.path.join(image_dir, fname)
    # copy the image from the source to the destination
    shutil.copyfile(src, dst)


# Transfer the healthy images
for fname in healthy_list:

    path = '/content/drive/MyDrive/RiceLeafsV1/train/Healthy'

    # source path to image
    src = os.path.join(path, fname)
    # destination path to image
    dst = os.path.join(image_dir, fname)
    # copy the image from the source to the destination
    shutil.copyfile(src, dst)


# Transfer the hispa images
for fname in hispa_list:

    path = '/content/drive/MyDrive/RiceLeafsV1/train/Hispa'

    # source path to image
    src = os.path.join(path, fname)
    # destination path to image
    dst = os.path.join(image_dir, fname)
    # copy the image from the source to the destination
    shutil.copyfile(src, dst)
```

Fig 5.4: Image Transfer

```
# Create the train set for each class

# brown_spot
# get a list of val images
val_list = list(df_brown_spot_val['image'])
# filter out the val images
df_brown_spot_train = df_brown_spot[~df_brown_spot['image'].isin(val_list)] # ~ means notin

# healthy
# get a list of val images
val_list = list(df_healthy_val['image'])
# filter out the val images
df_healthy_train = df_healthy[~df_healthy['image'].isin(val_list)] # ~ means notin

# hispa
# get a list of val images
val_list = list(df_hispa_val['image'])
# filter out the val images
df_hispa_train = df_hispa[~df_hispa['image'].isin(val_list)] # ~ means notin


# leaf_blast
# get a list of val images
val_list = list(df_leaf_blast_val['image'])
# filter out the val images
df_leaf_blast_train = df_leaf_blast[~df_leaf_blast['image'].isin(val_list)] # ~ means notin

print(len(df_brown_spot_train))
print(len(df_healthy_train))
print(len(df_hispa_train))
print(len(df_leaf_blast_train))
```

```
80
80
80
80
```

Fig 5.5: Image Transfer

5.6 Image plt Display

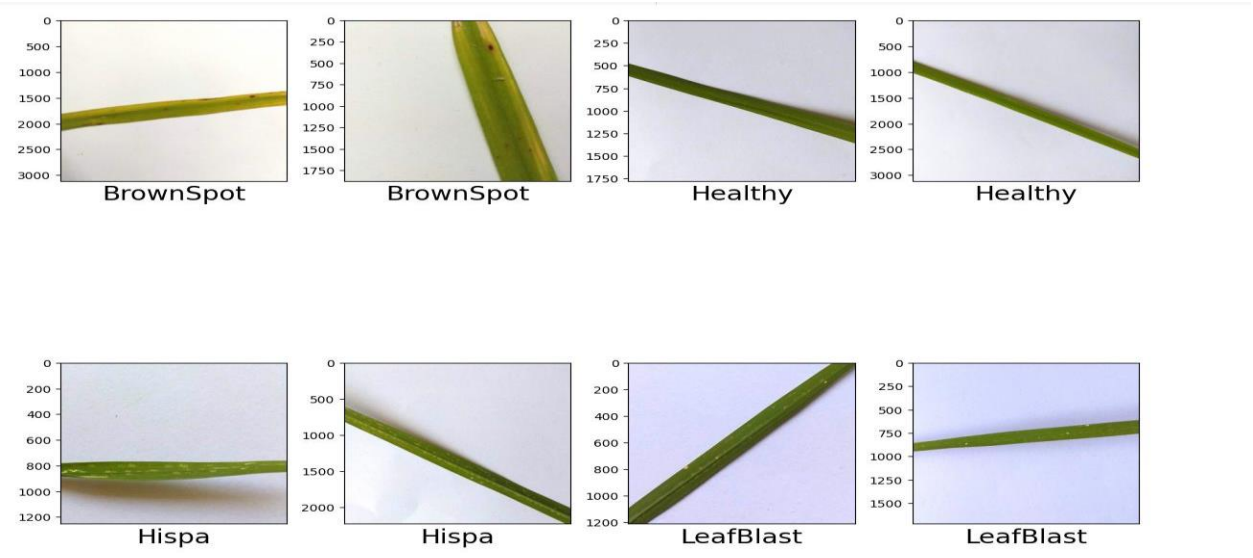


Fig 5.6: Image plt Display

5.7 Data Frame, Data Train, Validation

```
# Create df_data, df_train and df_val

df_data = pd.concat([ df_brown_spot, df_healthy,df_hispa, df_leaf_blast], axis=0).reset_index(drop=True)

df_train = \
pd.concat([ df_brown_spot_train, df_healthy_train, df_hispa_train, df_leaf_blast_train], axis=0).reset_index(drop=True)

df_val = \
pd.concat([ df_brown_spot_val, df_healthy_val, df_hispa_val, df_leaf_blast_val], axis=0).reset_index(drop=True)

df_data = shuffle(df_data)
df_train = shuffle(df_train)
df_val = shuffle(df_val)

print(df_data.shape)
print(df_train.shape)
print(df_val.shape)
```

```
(400, 2)
(320, 2)
(80, 2)
```

```
df_data['target'].value_counts()
```

```
target
leaf_blast    100
brown_spot    100
hispa         100
healthy       100
Name: count, dtype: int64
```

Fig 5.7: Data Frame, Data Tradin, Validation

5.8 Data Frame Combined, Train, Validation

```
[19] df_combined.head()
```

	image	target_brown_spot	target_healthy	target_hispa	target_leaf_blast
0	IMG_20190419_130033.jpg	False	True	False	False
1	IMG_20190419_095439.jpg	False	False	False	True
2	IMG_20190420_190623.jpg	True	False	False	False
3	IMG_20190419_144610.jpg	False	False	True	False
4	IMG_5508.jpg	False	False	False	True

Next steps: Generate code with df_combined View recommended plots

```
[20] df_train.head()
```

	image	target_brown_spot	target_healthy	target_hispa	target_leaf_blast
80	IMG_3046.jpg	True	False	False	False
81	IMG_20190420_190255.jpg	True	False	False	False
82	IMG_20190419_134311.jpg	False	False	True	False
83	IMG_3227.jpg	True	False	False	False
84	IMG_2992.jpg	True	False	False	False

Next steps: Generate code with df_train View recommended plots

```
[21] df_val.head()
```

	image	target_brown_spot	target_healthy	target_hispa	target_leaf_blast
0	IMG_20190419_130033.jpg	False	True	False	False
1	IMG_20190419_095439.jpg	False	False	False	True
2	IMG_20190420_190623.jpg	True	False	False	False
3	IMG_20190419_144610.jpg	False	False	True	False

Fig 5.8: Data Frame Combined, Train, Validation

5.9. Create the Train and Validation Sets

```
# Create the train and val sets

df_brown_spot = pd.DataFrame(brown_spot_list, columns=['image'])
df_brown_spot['target'] = 'brown_spot'

df_healthy = pd.DataFrame(healthy_list, columns=['image'])
df_healthy['target'] = 'healthy'

df_hispa = pd.DataFrame(hispa_list, columns=['image'])
df_hispa['target'] = 'hispa'

df_leaf_blast = pd.DataFrame(leaf_blast_list, columns=['image'])
df_leaf_blast['target'] = 'leaf_blast'


# Create a val set for each class

# Sample 5 validation images from each class
df_brown_spot_val = df_brown_spot.sample(n=20, random_state=101)
df_healthy_val = df_healthy.sample(n=20, random_state=101)
df_hispa_val = df_hispa.sample(n=20, random_state=101)
df_leaf_blast_val = df_leaf_blast.sample(n=20, random_state=101)


print(len(df_brown_spot_val))
print(len(df_healthy_val))
print(len(df_hispa_val))
print(len(df_leaf_blast_val))
```

```
20
20
20
20
```

Fig 5.9: Create the Train and Validation Sets

5.10 Dataframe Train and Validation

```
[16] df_train['target'].value_counts()
```

```
target
brown_spot    80
hispa         80
healthy       80
leaf_blast    80
Name: count, dtype: int64
```

```
[17] df_val['target'].value_counts()
```

```
target
healthy       20
leaf_blast    20
brown_spot    20
hispa         20
Name: count, dtype: int64
```

Transform the target

```
# combine val, train and test
val_len = len(df_val)
train_len = len(df_train)
df_combined =  pd.concat(objs=[df_val, df_train], axis=0).reset_index(drop=True)

# create the dummy variables
df_combined = pd.get_dummies(df_combined, columns=['target'])

# separate the train and val sets
df_val = df_combined[:val_len]
df_train = df_combined[val_len:]


print(df_train.shape)
print(df_val.shape)
```

```
(320, 5)
(80, 5)
```

Fig 5.10: Dataframe Train and Validation

Display some images by class

```
import matplotlib.pyplot as plt

# Set up the canvas for the subplots
plt.figure(figsize=(12, 12))  # Adjust figsize for desired dimensions

# Define image data (replace with your actual data)
image_data = [
    ('BrownSpot', '/content/drive/MyDrive/RiceLeafsV1/train/BrownSpot/IMG_20190420_190157.jpg'),
    ('BrownSpot', '/content/drive/MyDrive/RiceLeafsV1/train/BrownSpot/IMG_20190420_190524.jpg'),
    ('Healthy', '/content/drive/MyDrive/RiceLeafsV1/train/Healthy/IMG_20190419_134409.jpg'),
    ('Healthy', '/content/drive/MyDrive/RiceLeafsV1/train/Healthy/IMG_20190419_133948.jpg'),
    ('Hispa', '/content/drive/MyDrive/RiceLeafsV1/train/Hispa/IMG_20190419_144411.jpg'),
    ('Hispa', '/content/drive/MyDrive/RiceLeafsV1/train/Hispa/IMG_20190419_140219.jpg'),
    ('LeafBlast', '/content/drive/MyDrive/RiceLeafsV1/train/LeafBlast/IMG_20190419_110303.jpg'),
    ('LeafBlast', '/content/drive/MyDrive/RiceLeafsV1/train/LeafBlast/IMG_20190419_110347.jpg')
]

# Loop through image data and display in subplots
rows, cols = 2, 4  # 2 rows, 4 columns
for i, (label, path) in enumerate(image_data):
    plt.subplot(rows, cols, i + 1)  # Start indexing from 1 for subplots
    image = plt.imread(path)
    plt.imshow(image)
    plt.xlabel(label, fontsize=20)
    plt.xticks([])  # Turn off x-axis ticks for better layout

# Adjust layout (optional)
plt.tight_layout()

# Display the plot
plt.show()
```

Fig 5.11: Dataframe Train and Validation

5.11 Save the Dataframe as a Compressed Files

Save the dataframes as compressed csv files

```
[22] # Assuming mounted drive (uncomment if necessary)
     # from google.colab import drive
     # drive.mount('/content/drive')

     output_dir = '/content/drive/MyDrive/output'
     os.makedirs(output_dir, exist_ok=True)  # Creates the directory if it doesn't exist

     # Now you can save your CSV files:
     df_combined.to_csv(os.path.join(output_dir, 'df_combined.csv.gz'), compression='gzip', index=False)
     df_train.to_csv(os.path.join(output_dir, 'df_train.csv.gz'), compression='gzip', index=False)
     df_val.to_csv(os.path.join(output_dir, 'df_val.csv.gz'), compression='gzip', index=False)
```

Move all images to the same folder

```
[23] # Assuming mounted drive (uncomment if necessary)
     # from google.colab import drive
     # drive.mount('/content/drive')

     output_dir = '/content/drive/MyDrive/output'
     image_dir = os.path.join(output_dir, 'image_dir')  # Combine paths for the full directory

     # Create the directory 'image_dir' within 'output' if it doesn't exist
     os.makedirs(image_dir, exist_ok=True)
```

```
[ ] !ls /content/drive/MyDrive/output

df_combined.csv.gz  df_train.csv.gz  df_val.csv.gz  image_dir
```

Fig 5.12: Save the dataframe as a Compressed Files

5.12 Augment Image

```
[31] # Test the transformation setup.
     # The image will be different each time this cell is run.

     aug_image = augment_image(aug_types, image)

     plt.imshow(aug_image)
```

```
<matplotlib.image.AxesImage at 0x7dec768a7fa0>
```

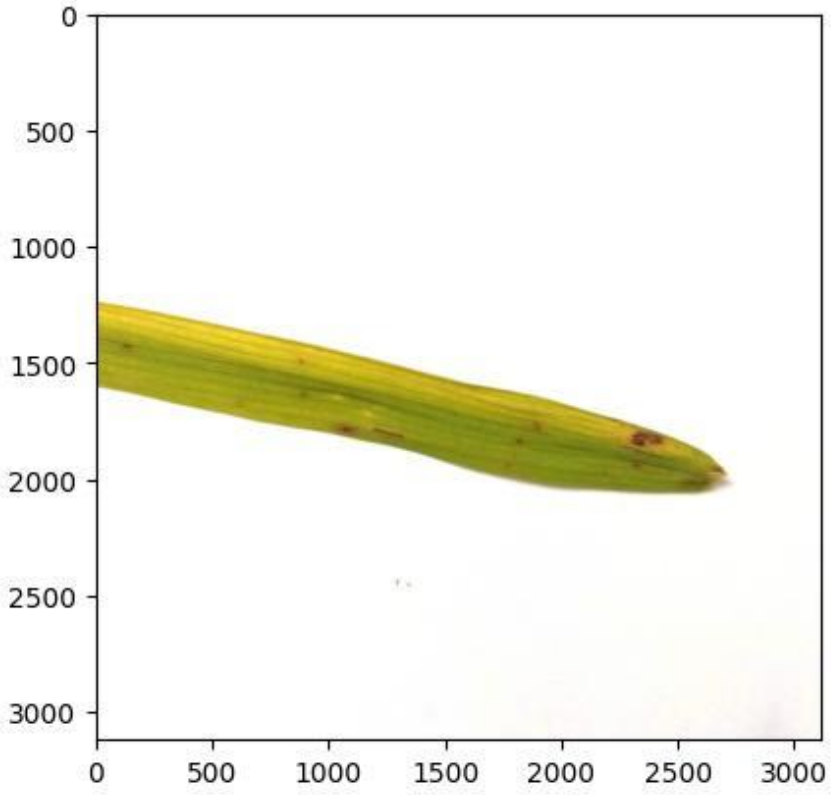


Fig 5.13: Augment Image

5.13 Display the First image

```
import os
import matplotlib.pyplot as plt

# Get an image to test transformations

# Get the path to the image directory
path = '/content/drive/MyDrive/output/image dir'

# Check if the directory exists (optional)
if not os.path.exists(path):
  print("Error: Directory", path, "does not exist.")
  exit()

# Loop through all images in the directory
for fname in os.listdir(path):
  image_path = os.path.join(path, fname)
  image = plt.imread(image_path)
  plt.imshow(image)
  plt.show()  # Add plt.show() to display each image
  break  # Uncomment to only process the first image
```

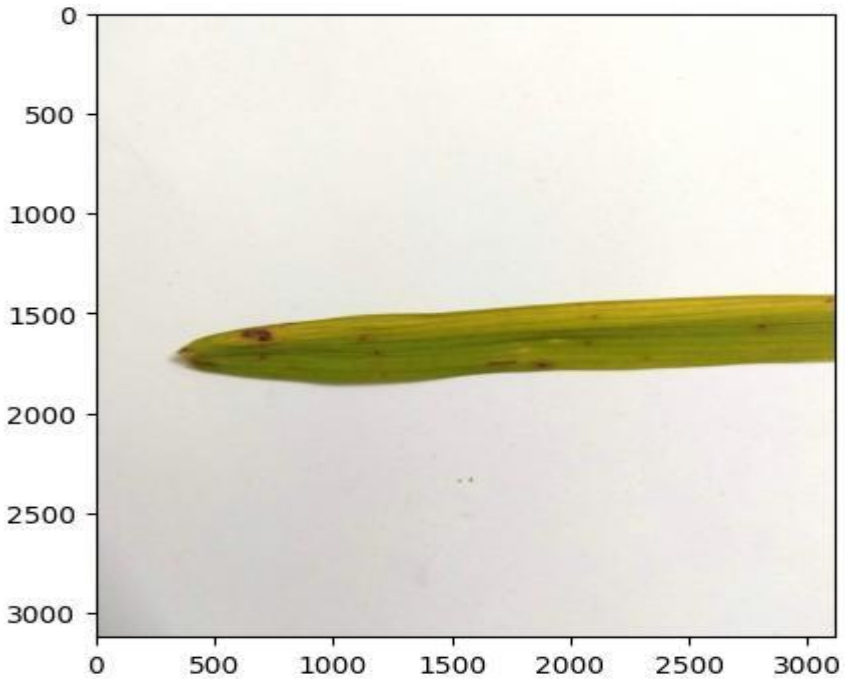


Fig 5.14: Display the First image

5.14 First image in X-Train

```
[34] y_train

     array([[ True, False, False, False],
            [ True, False, False, False],
            [False, False,  True, False],
            [ True, False, False, False],
            [ True, False, False, False],
            [False, False,  True, False],
            [False,  True, False, False],
            [False,  True, False, False]])
```

```
# Print the first image in X_train
# Remember that train images have been augmented.

image = X_train[0,:,:,:]
plt.imshow(image)
```

```
<matplotlib.image.AxesImage at 0x7dec75f87070>
```

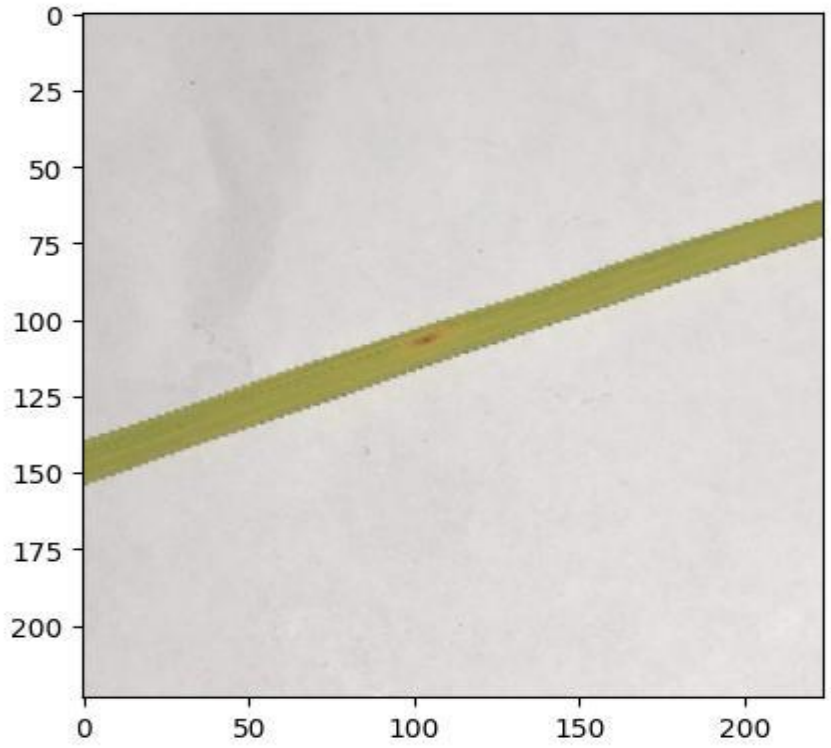


Fig 5.15: First image in X-Train

5.15 Validation Generator

[2] Val Generator

```
[36] def val_generator(batch_size=5):

       while True:
         val_path = '/content/drive/MyDrive/output/df_val.csv.gz'

         # Load the data in chunks (batches)
         for df in pd.read_csv(val_path, chunksize=batch_size):

           # Get the list of images
           image_id_list = list(df['image'])

           # Create empty X matrix - 3 channels
           X_val = np.zeros((len(df), IMAGE_HEIGHT, IMAGE_WIDTH, IMAGE_CHANNELS), dtype=np.uint8)

           # Create X_val
           # ===============
           for i in range(0, len(image_id_list)):

             # Get the image and mask
             image_id = image_id_list[i]

             # Set the path to the image (assuming 'image_dir/' is the correct directory)
             path = '/content/drive/MyDrive/output/image_dir/' + image_id

             # Print the image path for debugging
             print(path)

             try:
               # Read the image using OpenCV (handle potential errors)
               image = cv2.imread(path)

               if image is None:
                 print(f"Error reading image: {path}")
                 continue  # Skip this image if reading failed

               # Convert from BGR to RGB
               image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

               # Resize the image
               image = cv2.resize(image, (IMAGE_HEIGHT, IMAGE_WIDTH))

               # Insert the image into X_val
```

✓ Connected to Python 3 Google Compute Engine backend

Fig 5.16: Validation Generator

```
[36]            image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

                # Resize the image
                image = cv2.resize(image, (IMAGE_HEIGHT, IMAGE_WIDTH))

                # Insert the image into X_val
                X_val[i] = image

              except Exception as e:  # Catch generic exceptions for broader error handling
                print(f"Error processing image {path}: {e}")
                continue  # Skip this image if an error occurs

            # Create y_val
            # ===============
            cols = ['target_brown_spot', 'target_healthy', 'target_hispa', 'target_leaf_blast']
            y_val = df[cols]
            y_val = np.asarray(y_val)

            # Change the shape to (batch_size, 1) (optional, depending on your needs)
            # y_val = y_val.reshape((-1, 1))  # -1 tells numpy to automatically detect the batch size

            # Normalize the images
            X_val = X_val / 255

            yield X_val, y_val

    # Test the generator

    # Initialize
    val_gen = val_generator(batch_size=5)

    # Run the generator
    X_val, y_val = next(val_gen)

    print(X_val.shape)
    print(y_val.shape)
```

```
/content/drive/MyDrive/output/image_dir/IMG_20190419_130033.jpg
/content/drive/MyDrive/output/image_dir/IMG_20190419_095439.jpg
/content/drive/MyDrive/output/image_dir/IMG_20190420_190623.jpg
/content/drive/MyDrive/output/image_dir/IMG_20190419_144610.jpg
/content/drive/MyDrive/output/image_dir/IMG_5508.jpg
(5, 224, 224, 3)
(5, 4)
```

Fig 5.17: Validation Generator

5.16 X-Validation Image Show

```
[37] array([[False,  True, False, False],
            [False, False, False,  True],
            [ True, False, False, False],
            [False, False,  True, False],
            [False, False, False,  True]])
```

```
# print the image from X_val
image = X_val[0,:,:,:]
plt.imshow(image)
```

```
<matplotlib.image.AxesImage at 0x7dec7601e2f0>
```

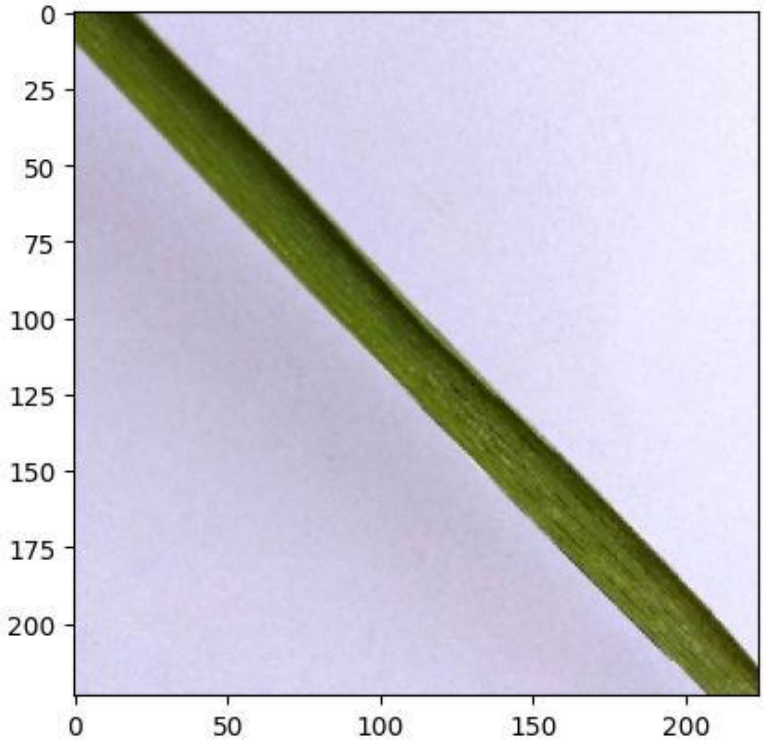


Fig 5.18: X-Validation Image Show

5.17 Train Generator

Build the Data Generators

[1] Train Generator

```
df_train.head()
```

	image	target_brown_spot	target_healthy	target_hispa	target_leaf_blast
80	IMG_3046.jpg	True	False	False	False
81	IMG_20190420_190255.jpg	True	False	False	False
82	IMG_20190419_134311.jpg	False	False	True	False
83	IMG_3227.jpg	True	False	False	False
84	IMG_2992.jpg	True	False	False	False

Next steps: Generate code with df_train View recommended plots

```
[33] def train_generator(batch_size=8):
        output_path = '/content/drive/MyDrive/output/df_train.csv.gz'

        while True:
            # load the data in chunks (batches)
            for df in pd.read_csv(output_path, chunksize=batch_size):

                # get the list of images
                image_id_list = list(df['image'])

                # Create empty X matrix - 3 channels
                X_train = np.zeros((len(df), IMAGE_HEIGHT, IMAGE_WIDTH, IMAGE_CHANNELS), dtype=np.uint8)

                # Create X_train
                for i in range(0, len(image_id_list)):

                    # get the image and mask
                    image_id = image_id_list[i]

                    # set the path to the image
                    path = '/content/drive/MyDrive/output/image_dir/' + image_id
```

Fig 5.19: Train Generator

```
                image = cv2.imread(path)

                # convert to from BGR to RGB
                image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

                # resize the image
                image = cv2.resize(image, (IMAGE_HEIGHT, IMAGE_WIDTH))

                # Create y_train
                cols = ['target_brown_spot', 'target_healthy', 'target_hispa', 'target_leaf_blast']
                y_train = df[cols]
                y_train = np.asarray(y_train)

                # change the shape to (batch_size, 1)
                #y_train = y_train.reshape((-1, 1)) # -1 tells numpy to automatically detect the batch size

                # Augment the image and mask
                aug_image = augment_image(aug_types, image)

                # insert the image into X_train
                X_train[i] = aug_image

            # Normalize the images
            X_train = X_train/255

            yield X_train, y_train



# Test the generator

# initialize
train_gen = train_generator(batch_size=8)

# run the generator
X_train, y_train = next(train_gen)

print(X_train.shape)
print(y_train.shape)
```

```
(8, 224, 224, 3)
(8, 4)
```

Fig 5.20: Train Generator

5.18 Test Genarator

[3] Test Generator

```
def test_generator(batch_size=1):

        while True:
          test_path = '/content/drive/MyDrive/output/df_val.csv.gz'

            # load the data in chunks (batches)
          for df in pd.read_csv(test_path, chunksize=batch_size):

              # get the list of images
              image_id_list = list(df['image'])

              # Create empty X matrix - 3 channels
              X_test = np.zeros((len(df), IMAGE_HEIGHT, IMAGE_WIDTH, IMAGE_CHANNELS), dtype=np.uint8)



              # Create X_test
               #================

          for i in range(0, len(image_id_list)):


             # get the image and mask
             image_id = image_id_list[i]


             # set the path to the image
             path = '/content/drive/MyDrive/output/image_dir/' + image_id

             # read the image
             image = cv2.imread(path)

             # convert to from BGR to RGB
             image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

             # resize the image
             image = cv2.resize(image, (IMAGE_HEIGHT, IMAGE_WIDTH))

             # insert the image into X_train
             X_test[i] = image
```

Fig 5.21: Test Genarator

5.19 X-Test Image Show

```
# print the image from X_test

image = X_test[0,:,:,:]
plt.imshow(image)
```

```
<matplotlib.image.AxesImage at 0x7dec75e8f6d0>
```

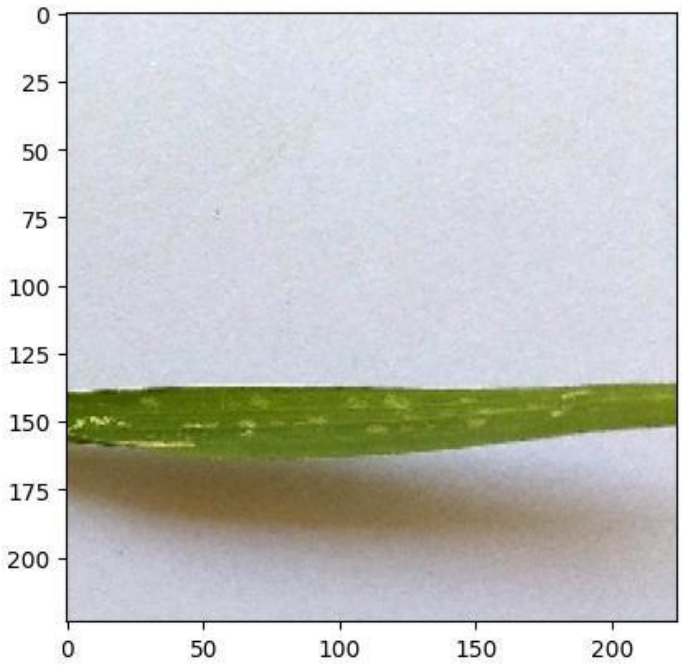


Fig 5.22: X-Test Image Show

5.20 MobileNet Model Predictions

```
from tensorflow.keras.applications.mobilenet import MobileNet

model = MobileNet(weights='imagenet')

# Exclude the last 2 layers of the above model.
x = model.layers[-2].output

# Create a new dense layer for predictions
# 3 corresponds to the number of classes
predictions = Dense(3, activation='softmax')(x)

# inputs=model.input selects the input layer, outputs=predictions refers to the
# dense layer we created above.

model = Model(inputs=model.input, outputs=predictions)

model.summary()
```

Model: "functional_5"

Layer (type)	Output Shape	Param #
input_layer_2 (InputLayer)	(None, 224, 224, 3)	0
conv1 (Conv2D)	(None, 112, 112, 32)	864
conv1_bn (BatchNormalization)	(None, 112, 112, 32)	128
conv1_relu (ReLU)	(None, 112, 112, 32)	0
conv_dw_1 (DepthwiseConv2D)	(None, 112, 112, 32)	288
conv_dw_1_bn (BatchNormalization)	(None, 112, 112, 32)	128
conv_dw_1_relu (ReLU)	(None, 112, 112, 32)	0
conv_pw_1 (Conv2D)	(None, 112, 112, 64)	2,048
conv_pw_1_bn (BatchNormalization)	(None, 112, 112, 64)	256
conv_pw_1_relu (ReLU)	(None, 112, 112, 64)	0
conv_pad_2 (ZeroPadding2D)	(None, 113, 113, 64)	0
conv_dw_2 (DepthwiseConv2D)	(None, 56, 56, 64)	576

Fig 5.23: MobileNet Model Predictions

5.21 Data File

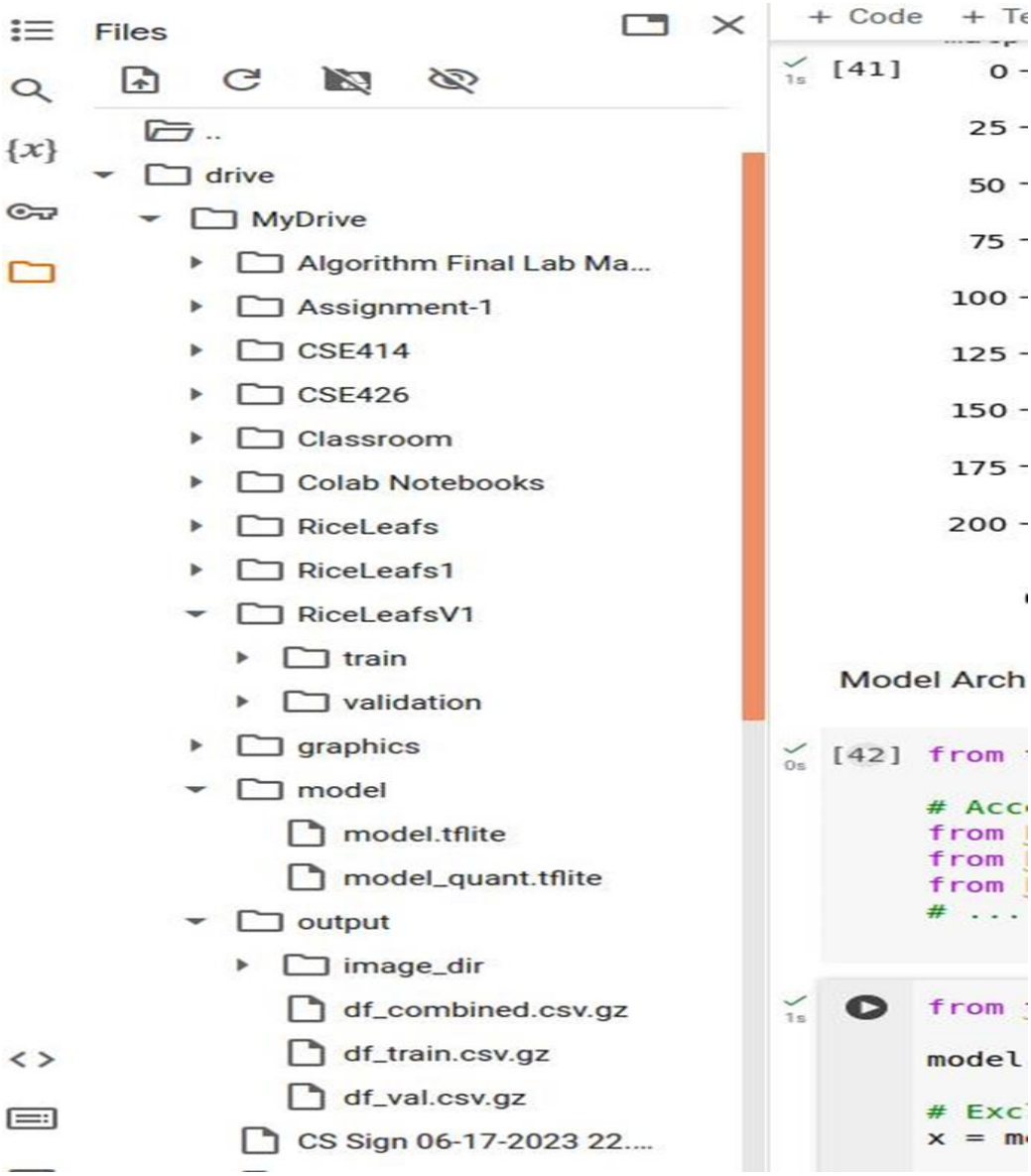


Fig 5.24: Data File

CHAPTER 6

CONCLUSION, LIMITATIONS AND FUTURE WORKS

6.1 CONCLUSION

The system provides an apps-based application which is very useful for emergency services. It will be a helpful and very useful application for the farmers of Bangladesh. Although the system has not yet been fully developed. Backend work is almost done, front end work is left. If all these tasks are done successfully then we hope that it will create a very good quality user friendly interface. Built using machine learning technology running within our system. By using our system, the farmer will be able to identify the disease of his main crop rice plant and take quick action. As a result, besides reducing the rice crisis of rice in the country, the effect of importing rice from foreign countries will be reduced to some extent. This will reduce the dependence on other countries and our country will be self-sufficient in food.

6.2 LIMITATIONS

The limitations of our project are minimal and manageable, they are:

- It is not suitable for detecting all types of rice diseases common in Bangladesh.
- Diseases of rice plants can be detected but no information is given on the cause of this disease.
- There is no alternative to only detecting diseases of rice plants by imaging.
- Our system does not yet have an option to save any reports.
- Some menu bars/buttons are required for user conveniences that have not been developed so far.
- As the device configuration used by us is not of good quality, there is no opportunity to work with very large databases.

6.3 FUTURE WORKS

For more fertile research, various paths are available. Next, we could extend our model to identify more different rice diseases by training the classifier model on more data. Moreover, this can easily be put on operation in the field by integrating the system with portable devices for real-time rice pest detection in the fields. Ultimately, the model should be felicitated with decision support systems to help farmers apply disease management tactics directly needed for the determined disease.

REFERENCE

[1] Khatun, M.T., Nessa, B., Salam, M.U. and Kabir, M.S., 2021. Strategy for rice disease management in Bangladesh. Bangladesh Rice Journal, 25(1), pp.23-36.

[2] Arnal Barbedo, J.G., 2013. Digital image processing techniques for detecting, quantifying, and classifying plant diseases. *SpringerPlus*, *2*(1), p.660.

[3] John, S. and Rose, A.L., 2021. Machine learning techniques in plant disease detection and classification-a state of the art. *INMATEH-Agricultural Engineering*, *65*(3)

[4] Prajapati, Harshadkumar B., Jitesh P. Shah, and Vipul K. Dabhi. "Detection and classification of rice plant diseases." Intelligent Decision Technologies 11, no. 3 (2017): 357-373.

[5] Chen, J., Zhang, D., Zeb, A. and Nanehkaran, Y.A., 2021. Identification of rice plant diseases using lightweight attention networks. *Expert Systems with Applications*, *169*, p.114514.

[6] Lu, Yang, Shujuan Yi, Nianyin Zeng, Yurong Liu, and Yong Zhang. "Identification of rice diseases using deep convolutional neural networks." Neurocomputing 267 (2017): 378-384.

[7] Hasan, M.J., Mahbub, S., Alom, M.S. and Nasim, M.A., 2019, May. Rice disease identification and classification by integrating support vector machine with deep convolutional neural network. In *2019 1st International Conference on Advances in Science, engineering, and Robotics Technology (ICASERT)* (pp. 1-6). IEEE.

[8] Wang, Y., Wang, H. and Peng, Z., 2021. Rice disease detection and classification using attention-based neural network and Bayesian optimization. *Expert Systems with Applications*, *178*, p.114770.

[9] Chen, Junde, Defu Zhang, Adnan Zeb, and Yaser A. Nanehkaran. "Identification of rice plant diseases using lightweight attention networks." Expert Systems with Applications 169 (2021): 114514.

[10] Jearanaiwongkul, Watanee, Chutiporn Anutariya, Teeradaj Racharak, and Frederic Andres. 2021. "An Ontology-Based Expert System for Rice Disease Identification and Control Recommendation" Applied Sciences 11, no. 21: 10450.

[11] Alam, Mohammad Jahangir, Md Abdul Awal, and Md Nurul Mustafa. "Crops diseases detection and solution system." International Journal of Electrical and Computer Engineering 9, no. 3 (2019): 2112.

[12] Latif, Ghazanfar, Sherif E. Abdelhamid, Roxane Elias Mallouhy, Jaafar Alghazo, and Zafar Abbas Kazimi. 2022. "Deep Learning Utilization in Agriculture: Detection of Rice Plant Diseases Using an Improved CNN Model" Plants 11, no. 17: 2230.

[13] Khatun, M.T., Nessa, B., Salam, M.U. and Kabir, M.S., 2021. Strategy for rice disease management in Bangladesh. Bangladesh Rice Journal, 25(1), pp.23-36.

[14] Haque, M.E., Rahman, A., Junaeid, I., Hoque, S.U. and Paul, M., 2022. Rice leaf disease classification and detection using yolov5. arXiv preprint arXiv:2209.01579.