**Wheat Rust Guard**

|  |
| --- |
|  |

**By:**

**Jasim Sagheer**

**32676**

**Haider Ali Sammar**

**38192**

**Naqi Turab-Ul-Hassnain**

**36520**

**Supervised by:**

**Mr. Zeeshan Ali**

**Faculty of Computing**

**Riphah International University, Islamabad**

**Spring/Fall 2024-2025**

**A Dissertation Submitted To**

**Faculty of Computing,**

**Riphah International University, Islamabad**

**As a Partial Fulfillment of the Requirement for the Award of the Degree of**

**Bachelors of Science in Computer Science**

**Faculty of Computing**

**Riphah International University, Islamabad**

Date: [To Be Announced]

**Final Approval**

This is to certify that we have read the report submitted by ***Jasim Sagheer (32676), Haider Ali Sammar (38192), Naqi Turab-Ul-Hassnain (36520),*** for the partial fulfillment of the requirements for the degree of the Bachelors of Science in Computer Science (BSCS). It is our judgment that this report is of sufficient standard to warrant its acceptance by Riphah International University, Islamabad for the degree of Bachelors of Science in Computer Science (BSCS).

**Committee:**

|  |  |
| --- | --- |
| **1** | Mr. Zeeshan Ali  (Supervisor) |
|  |  |
| **2** | Mr. Musharraf  (Head of Department) |

**Declaration**

We hereby declare that this document “**Wheat Rust Guard**” neither as a whole nor as a part has been copied out from any source. It is further declared that we have done this project with the accompanied report entirely on the basis of our personal efforts, under the proficient guidance of our teachers, especially our supervisor **Mr. Zeeshan Ali**. If any part of the system is proved to be copied out from any source or found to be reproduction of any project from anywhere else, we shall stand by the consequences.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Jasim Sagheer**

**[32676]**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Haider Ali Sammar**

**[38192]**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Naqi Turab-Ul-Hassnain**

**[36520]**

**Dedication**

Our project is dedicated to our parents, seniors, friends and our supervisor “Mr. Zeeshan Ali” for their insightful guidance and patience throughout this journey. I also extend my gratitude to my friends and peers for their camaraderie and encouragement, which kept me motivated. Finally, I dedicate this work to everyone who inspired me to pursue excellence and to the pursuit of knowledge that continues to drive my aspirations.

**Acknowledgement**

First of all, we are obliged to Allah Almighty the Merciful, the Beneficent and the source of all Knowledge, for granting us the courage and knowledge to complete this Project.

We owe a debt of gratitude to “Mr. Zeeshan Ali,” our project supervisor. Without their individual oversight, counsel, and invaluable direction, this project’s conclusion would have been questionable. We are incredibly grateful to them for their support and ongoing assistance throughout the project. We are also grateful to our parents and family, who have always supported us and taught us the importance of integrity and diligence.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Jasim Sagheer**

**[32676]**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Haider Ali Sammar**

**[38192]**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Naqi Turab-Ul-Hassnain**

**[36520]**

**Abstract**

Wheat rust is among the most important agricultural threats affecting agricultural productivity and food security worldwide. Traditional diagnosis of wheat rust consumption involves much time and inefficiency; most of the time, human error may cause interventions that might be belated by farmers. This project, Wheat Rust Guard, detects wheat rust diseases through automation using state-of-the-art artificial intelligence techniques. Computer vision and deep learning have been applied here to categorize wheat crops into Healthy, Yellow Rust-affected and Brown Rust-affected with the help of image-based analysis. Further, the application of this system has been integrated on a mobile device for the convenience and ease of farmers. The application shall provide instantaneous results regarding the condition of the crop, along with the treatment recommendations, and has the capability to function offline in areas where internet access is poor or no internet. This, in turn, results in a scalable, efficient, user-friendly tool that will enable farmers to undertake timely action and thereby improve the health of the crop, ensuring sustainable agriculture.

Table of Contents

[Table of Contents i](#_Toc197952059)

[List of Tables iii](#_Toc197952060)

[List of Figures iv](#_Toc197952061)

[Abstract 1](#_Toc197952062)

[Chapter 1: Introduction 2](#_Toc197952063)

[1.1 Goals and Objectives 2](#_Toc197952064)

[1.2 Scope of the Project 3](#_Toc197952065)

[Chapter 2: Literature Review 4](#_Toc197952066)

[2.1 Introduction 4](#_Toc197952067)

[2.2 Background and Problem Elaboration 4](#_Toc197952068)

[2.3 Detailed Literature Review 5](#_Toc197952069)

[2.3.1 Definitions 5](#_Toc197952070)

[2.3.2 Related Research Work 1 5](#_Toc197952071)

[2.3.3 Related Research Work 2 6](#_Toc197952072)

[2.3.4 Related Research Work 3 6](#_Toc197952073)

[2.3.5 Related Research Work 4 7](#_Toc197952074)

[2.3.6 Related Research Work 5 7](#_Toc197952075)

[2.3.7 Related Research Work 6 7](#_Toc197952076)

[2.3.8 Related Research Work 7 8](#_Toc197952077)

[2.3.9 Related Research Work 8 8](#_Toc197952078)

[2.3.10 Related Research Work 9 8](#_Toc197952079)

[2.3.11 Related Research Work 10 8](#_Toc197952080)

[2.3.12 Related Research Work 11 9](#_Toc197952081)

[2.3.13 Related Research Work 12 9](#_Toc197952082)

[2.4 Literature Review Summary Table 9](#_Toc197952083)

[2.5 Research Gap 11](#_Toc197952084)

[2.6 Problem Statement 13](#_Toc197952085)

[Chapter 3: Requirements and Design 14](#_Toc197952086)

[3.1 Requirements 14](#_Toc197952087)

[3.1.1 Functional Requirements 14](#_Toc197952088)

[3.1.2 Non-Functional Requirements 15](#_Toc197952089)

[3.1.3 Hardware and Software Requirements 16](#_Toc197952090)

[3.2 Proposed Methodology 17](#_Toc197952091)

[3.2.1 Overview 17](#_Toc197952092)

[3.2.2 Step-by-Step Process 17](#_Toc197952093)

[3.3 Dataset 18](#_Toc197952094)

[3.3.1 Dataset (Unknown Class) 18](#_Toc197952095)

[3.4 Deep Learning Model 19](#_Toc197952096)

[3.5 System Architecture 20](#_Toc197952097)

[3.6 Use Cases 22](#_Toc197952098)

[3.6.1 Fully Dressed Use Case 25](#_Toc197952099)

[3.7 Class Diagram 33](#_Toc197952100)

[3.8 Database Schema 34](#_Toc197952101)

[Chapter 4: Implementation and Test Cases 36](#_Toc197952102)

[4.1 Implementation 36](#_Toc197952103)

[4.1.1 Model Performance Analysis: Mobile-Net with Relu Activation 36](#_Toc197952104)

[4.1.2 Model Performance Analysis: MobileNet with Swish Activation 37](#_Toc197952105)

[4.1.3 Model Performance Analysis: InceptionV3 using Relu Activation 38](#_Toc197952106)

[4.1.4 Model Performance Analysis: InceptionV3 using Swish Activation 39](#_Toc197952107)

[4.1.5 Model Performance Analysis: DenseNet121 using Relu Activation 40](#_Toc197952108)

[4.1.6 Model Performance Analysis: DenseNet121 using Swish Activation 41](#_Toc197952109)

[4.1.7 Model Performance Analysis: ResNet50 using Relu Activation 42](#_Toc197952110)

[4.1.8 Model Performance Analysis: ResNet50 using Swish Activation 43](#_Toc197952111)

[4.1.9 Model Performance Analysis: VGG16 using Relu Activation 44](#_Toc197952112)

[4.1.10 Model Performance Analysis: VGG16 using Swish Activation 45](#_Toc197952113)

[4.1.11 Model Performance Analysis: Hybrid Model using Relu Activation 46](#_Toc197952114)

[4.1.12 Model Performance Analysis: Hybrid Model using Leaky Relu Activation 47](#_Toc197952115)

[4.1.13 Model Performance Analysis: Hybrid Model with Unknown Class using Leaky Relu Activation 48](#_Toc197952116)

[4.2 Comparative Analysis 49](#_Toc197952117)

[4.3 Mobile Application Development 49](#_Toc197952118)

[4.3.1 UI/UX Design 50](#_Toc197952119)

[4.4 Test case Design and description 51](#_Toc197952120)

[4.4.1 Test Case 1-Image Upload and Detection 51](#_Toc197952121)

[4.4.2 Test Case 2-Offline Disease Detection 51](#_Toc197952122)

[4.4.3 Test Case 3-Profile Update and Firebase Sync 52](#_Toc197952123)

[4.4.4 Test Case 4-Result Logging in Previous History 53](#_Toc197952124)

[4.4.5 Test Case 5-UI/UX: Onboarding and Language Switching 53](#_Toc197952125)

[4.4.6 Test Case 6-Crop Diary Entry Creation and Display 54](#_Toc197952126)

[4.5 Summary 55](#_Toc197952127)

[4.6 Test Metrics 56](#_Toc197952128)

[4.6.1 Sample Test case Matric.No.1 56](#_Toc197952129)

[4.6.2 Sample Test case Metric.No.2 57](#_Toc197952130)

[4.6.3 Sample Test case Metric.No.3 57](#_Toc197952131)

[Chapter 5: Experimental Results and Analysis 58](#_Toc197952132)

[5.1 Introduction 58](#_Toc197952133)

[5.2 Experimental Setup 58](#_Toc197952134)

[5.3 Performance Results 58](#_Toc197952135)

[5.4 Analysis and Discussion 59](#_Toc197952136)

[5.5 Conclusion 60](#_Toc197952137)

[Chapter 6: Conclusion and Future Directions 61](#_Toc197952138)

[6.1 Conclusion 61](#_Toc197952139)

[6.2 Challenges Faced 61](#_Toc197952140)

[6.3 Future Directions 62](#_Toc197952141)

[6.4 Concluding Insights 63](#_Toc197952142)

[References 64](#_Toc197952143)

[Appendix 65](#_Toc197952144)

[Appendix A: Dataset Samples 65](#_Toc197952145)

[Appendix B: Hybrid Model Code Snippet 67](#_Toc197952146)

[Appendix C: Mobile Application (UI and Icons) 69](#_Toc197952147)

[Appendix D: Easy-Localization 70](#_Toc197952148)

List of Tables

[Table 3.1‑1 Functional Requirements for Farmer 14](#_Toc197952149)

[Table 3.1‑2 Functional Requirements for Admin 15](#_Toc197952150)

[Table 3.3‑1 Wheat Rust Dataset (Excluding Unknown Class) 18](#_Toc197952151)

[Table 3.3‑2 Wheat Rust Dataset (Including Unknown Class) 18](#_Toc197952152)

[Table 3.8‑1 Database Schema of User's Collection 34](#_Toc197952153)

[Table 3.8‑2 Database Schema of Detection Results Collection 34](#_Toc197952154)

[Table 3.8‑3 Database Schema of Feedback Collection 35](#_Toc197952155)

[Table 3.8‑4 Database Schema of Crop Diary Collection 35](#_Toc197952156)

[Table 4.1‑1 Performance Assessment MobileNet-Relu 36](#_Toc197952157)

[Table 4.1‑2 Performance Evaluation MobileNet-Relu 36](#_Toc197952158)

[Table 4.1‑3 Performance Assessment MobileNet-Swish 37](#_Toc197952159)

[Table 4.1‑4 Performance Evaluation MobileNet-Swish 37](#_Toc197952160)

[Table 4.1‑5 Performance Assessment InceptionV3-Relu 38](#_Toc197952161)

[Table 4.1‑6 Performance Evaluation InceptionV3-Relu 38](#_Toc197952162)

[Table 4.1‑7 Performance Assessment InceptionV3-Swish 39](#_Toc197952163)

[Table 4.1‑8 Performance Evaluation InceptionV3-Swish 39](#_Toc197952164)

[Table 4.1‑9 Performance Assessment DenseNet121-Relu 40](#_Toc197952165)

[Table 4.1‑10 Performance Evaluation DenseNet121-Relu 40](#_Toc197952166)

[Table 4.1‑11 Performance Assessment DenseNet121-Swish 41](#_Toc197952167)

[Table 4.1‑12 Performance Evaluation DenseNet121-Swish 41](#_Toc197952168)

[Table 4.1‑13 Performance Assessment ResNet50-Relu 42](#_Toc197952169)

[Table 4.1‑14 Performance Evaluation ResNet50-Relu 42](#_Toc197952170)

[Table 4.1‑15 Performance Assessment ResNet50-Swish 43](#_Toc197952171)

[Table 4.1‑16 Performance Evaluation ResNet50-Swish 43](#_Toc197952172)

[Table 4.1‑17 Performance Assessment VGG16-Relu 44](#_Toc197952173)

[Table 4.1‑18 Performance Evaluation VGG16-Relu 44](#_Toc197952174)

[Table 4.1‑19 Performance Assessment VGG16-Swish 45](#_Toc197952175)

[Table 4.1‑20 Performance Evaluation VGG16-Swish 45](#_Toc197952176)

[Table 4.1‑21 Performance Assessment Hybrid Model-Relu 46](#_Toc197952177)

[Table 4.1‑22 Performance Evaluation Hybrid Model-Relu 46](#_Toc197952178)

[Table 4.1‑23 Performance Assessment Hybrid Model-Leaky Relu 47](#_Toc197952179)

[Table 4.1‑24 Performance Evaluation Hybrid Model-Leaky Relu 47](#_Toc197952180)

[Table 4.1‑25 Performance Assessment Hybrid Model (Unknown Class)-Leaky Relu 48](#_Toc197952181)

[Table 4.1‑26 Performance Evaluation Hybrid Model (Unknown Class) -Leaky Relu 48](#_Toc197952182)

[Table 4.2‑1 Comparative Analysis of Each Model 49](#_Toc197952183)

[Table 4.4‑1 TC-Image Upload and Detection 51](#_Toc197952184)

[Table 4.4‑2 TC-Offline Disease Detection 51](#_Toc197952185)

[Table 4.4‑3 TC-Profile Update and Firebase Sync 52](#_Toc197952186)

[Table 4.4‑4 TC-Result Logging in Previous History 53](#_Toc197952187)

[Table 4.4‑5 TC-UI/UX: Onboarding and Language Switching 53](#_Toc197952188)

[Table 4.4‑6 TC-Crop Diary Entry Creation and Display 54](#_Toc197952189)

List of Figures

[Figure 1.1‑1 Objectives of our Project 3](#_Toc197952190)

[Figure 3.2‑1 Proposed Solution Diagram 18](#_Toc197952191)

[Figure 3.4‑1 Deep Learning (CNN-Model Architecture) 19](#_Toc197952192)

[Figure 3.5‑1 System Architecture Design of Wheat Rust Guard 21](file:///C:\Users\haide\OneDrive\Desktop\Report_v6.docx#_Toc197952193)

[Figure 3.6‑1 Use Case Diagram for Farmer 22](#_Toc197952194)

[Figure 3.6‑2 Use Case Diagram for Admin 23](#_Toc197952195)

[Figure 3.6‑3 Complete Use Case Diagram 24](#_Toc197952196)

[Figure 3.7‑1 Class Diagram 33](#_Toc197952197)

[Figure 4.1‑1 Confusion Matrix MobileNet-Relu 36](#_Toc197952198)

[Figure 4.1‑2 Confusion Matrix MobileNet-Swish 37](#_Toc197952199)

[Figure 4.1‑3 Confusion Matrix InceptionV3-Relu 38](#_Toc197952200)

[Figure 4.1‑4Confusion Matrix InceptionV3-Swish 39](#_Toc197952201)

[Figure 4.1‑5 Confusion Matrix DenseNet121-Relu 40](#_Toc197952202)

[Figure 4.1‑6 Confusion Matrix DenseNet121-Swish 41](#_Toc197952203)

[Figure 4.1‑7 Confusion Matrix ResNet50-Relu 42](#_Toc197952204)

[Figure 4.1‑8 Confusion Matrix ResNet50-Swish 43](#_Toc197952205)

[Figure 4.1‑9 Confusion Matrix VGG16-Relu 44](#_Toc197952206)

[Figure 4.1‑10 Confusion Matrix VGG16-Swish 45](#_Toc197952207)

[Figure 4.1‑11 Confusion Matrix Hybrid Model-Relu 46](#_Toc197952208)

[Figure 4.1‑12 Confusion Matrix Hybrid Model-Leaky Relu 47](#_Toc197952209)

[Figure 4.1‑13 Confusion Matrix Hybrid Model (Unknown Class)-Leaky Relu 48](#_Toc197952210)

[Figure 4.3‑1 Mobile Application Development Process 50](#_Toc197952211)

[Figure 4.5‑1 Key Tested Modules 56](#_Toc197952212)

# Abstract

The **Wheat Rust Guard** will provide the farmers community with an efficient, mobile-based platform for the detection and management of wheat rust diseases. The application will be made using **Flutter** and **Dart**,which will incorporate advance deep learning models in the detection of the diseases. It enables users to upload or scan images of wheat crops for the application of the identification process through AI-powered computer vision capable of identifying diseases like **Yellow Rust** and **Brown Rust**.

This contains a user-friendly interface that navigates farmers step by step. Onboarding is also designed to make usage easy for those who are not familiar with the technology. The app allows offline functionality in areas with low internet connectivity. Treatment recommendations will also be provided to help the farmer take timely and corrective action.

The app applies AI-driven analysis to guarantee high accuracy in detecting diseases of wheat rust, eliminating all inefficiencies that may characterize traditional manual diagnostics. But most importantly, Wheat Rust Guard empowers them with a scalable and efficient solution-meaning better crop health and truly sustainable agriculture.

# Introduction

Agriculture is the backbone of the economy in Pakistan and provides employment for day-to-day livelihood to millions. However, wheat rust-a highly destructive fungal diseases-seriously threatens wheat, the staple crop of the country. It affects the quality and quantity of the yield, causing losses in substantial economic earnings and imposing a hazard to food safety. Conventional diseases detection methods, however, are manual, which means time consumption, inaccuracies, and leaving farmers unprepared to take efficient actions against the disease accordingly.

Addressing these challenges, Wheat Rust Guard offers the integration of artificial intelligence and computer vision as a paradigm shift in wheat rust detection. The application enables farmers to detect wheat rust diseases – such as Brown Rust and Yellow Rust – by scanning or uploading images of affected crops. Advanced deep learning models applied within the app ensure a correct real-time diagnosis, while offering active insights to farmers for effective management.

The offline application of the system, other than detecting diseases, also suggests recommendations for treatment, hence making it accessible even in remote areas. Wheat Rust Guard propagates a junction between modern day technology and agricultural requirement in simplification of disease diagnosis and prevention, hence leading to sustainable farming and increasing crop yields.

## Goals and Objectives

The main goals and objectives of the Wheat Rust Guard project are:

1. Development of an AI-enabled mobile application that is capable of detecting wheat rust diseases, Brown Rust and Yellow Rust with high accuracy.
2. To provide effective, well-timed, and actionable insights to farmers for better disease management.
3. To implement user-friendly functionalities like offline access and treatment suggestions that could further improve the accessibility and usability.
4. To reduce the reliance on manual diagnostic methods, saving time and improving detection accuracy.
5. To contribute to environmentally friendly agriculture and to raise wheat production to achieve food security.

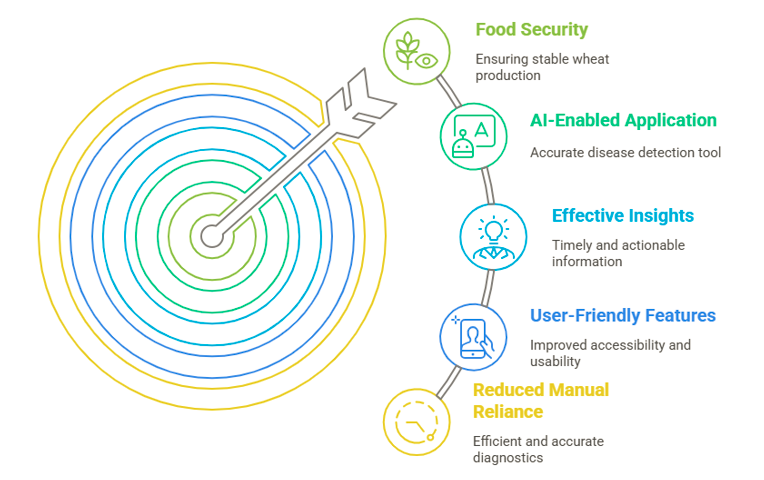


Figure 1.1‑1 Objectives of our Project

## Scope of the Project

The scope of Wheat Rust Guard will include:

1. Developing a mobile application using Flutter and Dart that employs AI and deep learning models.
2. It was aimed at the development of the capability to diagnose wheat rust diseases through uploading the image.
3. Integration of added features such as offline mode and suggestion of treatment that help to solve the accessibility related to farmers.
4. The target would be small to medium scale farmers in Pakistan, particularly in less accessible parts of the country for access to agricultural equipment or modern farming technologies.
5. Scalability and flexibility ensure the detection of other crop diseases in the future.

# Literature Review

## Introduction

This chapter focuses on a review pertaining to present studies and developments about early identification studies using artificial intelligence powered with computer vision for specific wheat rust diseases, precisely yellow and brown rust. Specific state-of-the-art AI tools related to agriculture, methodologies presented concerning the disease detection phases involved in this research area and deficiencies of the current resolution in the literature are debated herewith. It lays the foundation for the understanding of how the Wheat Rust Guard project innovates on existing approaches by offering a mobile-based, user-friendly platform, catering to farmers.

## Background and Problem Elaboration

Wheat rust diseases are considered some of the most feared diseases that attack wheat all over the world. These fungal infections will degrade not only the crop yields but also the quality of grains, thus impacting food security as a whole and farmers livelihoods. Management of any such disease requires early detection of the disease so that at least its impact could be minimized.

Traditional monitoring methods for wheat rust rely on either a manual inspection or the expert opinion of agricultural personnel. While effective to a degree, there are a number of limitations in these approaches:

* **Inefficiency and Cost:** Manual inspection is very time-consuming, requiring much labor, especially on large fields.
* **Inaccessibility:** Most small-scale farmers in developing regions face the challenge of a lack of accessibility to professional advice and improved equipment.
* **Delayed Diagnosis:** By the time symptoms are visible, the diseases has probably spread so extensively.

These challenges have transformative solutions through advances in AI and machine learning. In agriculture-food, AI has appeared to be effective in precision tasks concerning disease detection, yield estimation and soil analysis. Computer vision, a subcategory of AI, has proven especially apt, through visual data analysis of symptoms for the detection and identification of diseases with much accuracy. However, most of these tools driven by AI are either too costly, or require technical capability, or are beyond the reach and access of small-scale farmers. The Wheat Rust Guard Project fills this gap by providing an inexpensive, mobile-based platform for the early detection of wheat rust. Equipped with AI and computer vision, this tool enables farmers to quickly identify infections and reduces time and effort spent in monitoring while improving decision-making and response times.

## Detailed Literature Review

### Definitions

The following definitions are relevant and provide a basic understanding for the basis of this project:

* **Artificial Intelligence:** It is the ability of machines, especially computer systems, to simulate processes of human intelligence and, therefore, be able to carry out tasks that require making decisions, learning, and solving problems. Machine learning – subcategory of AI engaged in the development of algorithms through which systems can learn and improve processes from experience without explicit instructions. This can handle ML models learning from labeled datasets to recognize signs of wheat rust.
* **Computer Vision:** This is a subfield of AI, where machines have the ability to interpret, analyze and process images or videos. In this context, computer vision algorithms in Wheat Rust Guard analyze images of wheat leaves for rust symptoms.
* **Precision Agriculture:** This is the application of technologies in managing farming practices by monitoring field variability to apply intervention where exactly it is required. The project epitomizes precision agriculture, whereby the use of AI tools on mobile devices in the targeted detection of diseases was considered.
* **Yellow Rust / Stripe Rust:** A fungal disease, happening due to puccinia striiformis on wheat leaves in forms of yellow stripes thus offering considerable yield loss.
* **Brown Rust / Leaf Rust:** It is a fungal disease because of the fungus puccinia triticina; it shows reddish-brown-colored spots on the leaves of wheat, and it reduces the photosynthetic capability along with physical strength.

### Related Research Work 1

Romain Bebronne et al [1]. proposed a methodology for detecting fungal diseases within winter wheat, specifically stripe rust (SR), brown rust (BR), and septoria tritici blotch (STB). The authors applied reflectance and textural features from multispectral imagery. Their system integrated a ground-based platform fitted with a multispectral camera that captured wavebands over the entire visible and near-infrared spectrum. They conducted the estimation of disease severity based on spectral and textural data using ANN and PLSR models. Major steps involved in preprocessing were the segmentation of multispectral images and waveband-specific feature extraction. ANN outperformed PLSR in predicting disease severity with R^2 values of 0.72, 0.57, and 0.65 for STB, SR, and BR, respectively. STB disease severity classification accuracy at higher infection thresholds was 81%. The study showcased the potential of integrating reflectance data with machine learning for precision disease monitoring under natural conditions of the field.

### Related Research Work 2

Mamatha Mandava et al [2]. investigated a few deep learning models for yellow rust detection in wheat due to \*Puccinia striiformis\*. These studies compared the performance of three CNN architectures, namely ResNet50, DenseNet121, and VGG19, using a dataset comprising annotated images of healthy/infected wheat leaves. These models were trained using well-known techniques of data augmentation and transfer learning to enhance their generalizability. ResNet50 and DenseNet121 were very efficient, while VGG19 had a moderate performance in feature extraction. EfficientNetB3 gave the best accuracy among the models compared in this work and is suitable for large-scale precision agriculture real-time monitoring. Advanced CNN-based systems have been underlined in this work that achieve high classification accuracy to help in early disease detection.

### Related Research Work 3

Habib Khan et al[3]. have proposed a framework for automatically classifying wheat diseases that uses ML, with special emphasis on the classification of yellow and brown rust diseases.

It proposes field data collection from different regions in Pakistan with strong preprocessing like image segmentation and resizing to retain high input quality. The proposed framework compared several ML algorithms based on the above approach and achieved the maximum accuracy of 99.8%, which was beyond the results of the conventional methods. They highlighted the importance of efficient preprocessing and data preparation in ML applications, showcasing the effectiveness in real-world settings with such diverse environmental conditions.

### Related Research Work 4

Wang et al[4]. described an image-processing-based stripe rust- and leaf rust-infected identifying system on the wheat varieties under different environmental conditions. Its approach flowed by taking a single-leaf image in high resolution, scaling, doing some morphological reconstructions, and extracting these various median filters and lesion segmentations, with segmentation extracted onto 140 color, text, and shape features. Further, feature selection was undertaken via ReliefF, 1R, and correlation-based feature selection along with PCA. The implementation had been done on SVMs, BPNN, and random forest for both single- and multi-variety identification using ML models.

Under greenhouse conditions, the multi-variety models had identification accuracies as high as 100%, thus demonstrating their potential to monitor disease with accuracy across diverse wheat varieties.

### Related Research Work 5

Shafi et al[5]. have developed an embedded AI for the classification of wheat yellow rust infection types, including healthy, resistant, moderately resistant, and susceptible.

They collected their dataset indigenously and preprocessed it using the U2-Net model for background removal. Two deep learning classifiers were implemented: ResNet-50 and Xception. The best performance, with accuracy as high as 96%, was achieved by ResNet-50. Further, the model deployed on the edge computing device enables real-time monitoring. Thus, farmers are able to detect the rust severity in situ out in the field. This research highlights how the embedding of AI with portable devices has enormous potential to improve precision agriculture and, therefore, management of crop health.

### Related Research Work 6

Cuenca-Romero et al[6]. addressed the issue of unbalanced datasets in the identification of yellow and brown rust in wheat using hyperspectral imaging and machine learning. They used the SMOTE algorithm for augmentation in their dataset and then trained ANN, SVM, RF, and GNB models. RF performed best for yellow rust detection, giving an accuracy of 70% on raw datasets, while SVM was the best for brown rust detection on augmented data.

This study applied the integration of information of spectral reflectance owing to its efficiency in executing early detection of diseases, incorporating an investigation of machine learning-model performance into the dataset imbalance problem toward reliable identification.

### Related Research Work 7

Nguyen et al[7]. proposed a new system for early detection of wheat yellow rust using multispectral UAV imagery.Their approach combined VIs and GLCM texture features to capture the spectral and spatial dimensions of disease progress. They used a 3D-CNN for disease monitoring, which achieved detection accuracies of 60% at the early tillering stage up to 79% at flowering stages. Critical spectral bands included the red-edge (690-740 nm) and near-infrared (740-1000 nm) regions, which were key for disease differentiation.

In that sense, this approach showed that through an early detection method, 3-7% of crop yield could be conserved, which would correspondingly create added value for precision agriculture economically viable.

### Related Research Work 8

Liu et al[8]. presented StripeRust-Pocket, a smartphone-based deep learning app used in measuring stripe rust severity of wheat.The segmentation of a two-stage model in the system was realized using StripeRustNet, which combines MobileNetV2-DeepLabV3+ for leaf segmentation with ResNet50-DeepLabV3+ for lesion segmentation.

It was able to achieve 98.65% mean intersection over union for segmenting a leaf and 86.08% for lesion segmentations. Moreover, by using a custom labeling pipeline, the time spent annotating reduced from 20 minutes to 3 minutes per image. Also, on-site disease monitoring is made possible with the mobile application, letting the researcher and farmer have an easy implementable tool for timely treatment of the disease.

### Related Research Work 9

Alharbi et al[9]. The model suggested here for the classification of wheat disease through few-shot learning combined the concept of EfficientNet along with the attention mechanism for feature selection. The developed technique provided excellent results on 18 classes of wheat diseases with highly limited training data, which proved to yield 93.19% accuracy on custom images and 98.5% accuracy on the CGIAR dataset. This work tries to address the challenge of data availability and catastrophic forgetting using continual learning, hence showing its prospect in efficient disease identification in resource-constrained environments.

### Related Research Work 10

Kumar and Kukreja[10] designed a hybrid model, combining a generative adversarial network-STARGAN-with a convolutional neural network-CNN-for wheat yellow rust classification. Here, the STARGAN model was used for augmenting the data, while CNN was trained for multi-level severity detection. Their model had attained 95.6% accuracy at medium severity levels and outperformed some other deep learning models like a fully convolutional network and random forests.

The study also highlighted hybrid models as essential in improving the accuracy of disease prediction when conditions are unfavorable.

### Related Research Work 11

Tolba and Talal[11] proposed the Mobile-DNN-Net model, a hybrid of deep learning models combined taking the most important features from both feature extractors, namely MobilNet and DCNN. Recently, early detection and classification of diseases on wheat leaves with gradual class activation maps provided substantial explainability. Works also have been validated over different datasets of high-resolution images given a total of 15 classes of diseases that represented 14,155 images totally. Mobile-DNN-Net ensured better accuracy and more ex-plainability when using other compared models and makes robust tools within precision agriculture.

### Related Research Work 12

Jiang et al[12]. made a comparative assessment of the performance of VGG-16, Inception-v3, ResNet-50, and other architectures for wheat leaf disease using field-acquired photographs. "Transfer learning" approach adopted for the models with appropriate hyper-parameter tuning on Field-based Wheat Diseases Images-FWDI custom dataset. Among these models, the performance achieved varied from Inception-v3, the top among them with an accuracy of 92.5%, while MobileNetV3 showed the fastest process with poorer accuracy. It reflected how model architecture, training strategy, and input data quality interact in the process of diagnosing diseases.

## Literature Review Summary Table

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Name** | **Reference** | **Year** | **Input** | **Output** | **Description** |
| Romain Bebronne et al. | [1] | 2020 | Multispectral imagery (visible & NIR). | Disease severity R^2: STB (0.72), SR (0.57),BR (0.65); STB threshold classification (81%). | Methodology integrating reflectance and textural data with ANN / PLSR for fungal disease severity detection. |
| Mamatha Mandava et al. | [2] | 2024 | Annotated wheat leaf images. | EfficientNetB3 had the best accuracy. | Compared CNN models (ResNet50, DenseNet121, VGG19, EfficientNetB3) for yellow rust detection. |
| Habib khan et al. | [3] | 2022 | Field-collected wheat leaf images. | Classification accuracy of 99.8%. | ML models with segmentation and resizing; focused on yellow and brown rust classification. |
| Wang et al. | [4] | 2023 | High-resolution single-leaf images. | 100% accuracy (greenhouse conditions). | Feature selection (ReliefF, PCA) and ML models for rust detection across multiple wheat varieties. |
| Shafi et al. | [5] | 2023 | Indigenously collected images. | ResNet-50: 96% accuracy. | Edge AI for real-time monitoring; background removal with U2-Net. |
| Cuenca- Romero et al. | [6] | 2024 | Hyperspectral imaging. | RF: 70% for yellow rust; SYM: best for brown rust. | Addressed dataset imbalance using SMOTE; focused on spectral reflectance for early detection. |
| Nguyen et al. | [7] | 2023 | UAV multispectral imagery. | Detection: 60% (early), 79% (flowering). | 3D-CNN using VI and GLCM texture features for yellow rust monitoring; conserved crop yield (3-7%). |
| Liu et al. | [8] | 2024 | Smartphone images. | Leaf segmentation: 98.65%; lesions: 86.08%. | Developed StripeRust-Pocket app; reduced annotation time; enabled on-site rust monitoring. |
| Alharbi et al. | [9] | 2023 | Few-shot learning with CGIAR dataset. | 93.19% (custom), 98.5% (CGIAR dataset). | Combined EfficientNet with attention mechanism for wheat disease detection under data-constrained scenarios. |
| Kumar & Kukreja | [10] | 2023 | Augmented wheat leaf images. | 95.6% accuracy at medium severity levels. | Hybrid STARGAN-CNN model for wheat yellow rust detection. |
| Tolba et al. | [11] | 2024 | High resolution wheat leaf images. | High accuracy across 15 classes. | Mobile-DNN-Net combining MobilNet and DCNN for early disease classification and explainability. |
| Jiang et al. | [12] | 2022 | Field acquired wheat leaf photographs. | Inception-v3: 92.5% accuracy. | Transfer learning for wheat disease detection using multiple CNN architectures. |

## Research Gap

Despite significant advancements in wheat disease detection using multispectral imagery, machine learning (ML), and deep learning (DL) techniques, several challenges persist:

* **Early Detection at Subtle Stages**: Many studies, such as Nguyen et al[7]. and Romain Bebronne et al[1]., showed limited accuracy for early-stage disease detection (e.g., Nguyen et al.: 60% accuracy at tillering stage). There is a need to enhance the sensitivity of models for early disease progression stages.
* **Generalization Across Environments**: Most works, such as Habib Khan et al[3]. and Wang et al[4]., are region-specific and tested under controlled conditions (e.g., greenhouse). However, wheat diseases exhibit variability due to environmental, regional, and genetic differences. Research must focus on robust systems applicable across diverse regions and natural conditions.
* **Data Scarcity and Imbalance**: Studies like Cuenca-Romero et al[6]. have addressed dataset imbalance using SMOTE, but real-world datasets remain scarce and unbalanced, especially for rare disease cases. Techniques such as few-shot learning, continual learning, and synthetic data generation require further exploration.
* **Integration with Real-Time Monitoring Systems**: Although advancements like StripeRust-Pocket[8] and Shafi et al.'s[5] edge AI offer real-time solutions, they often lack scalability for large farms or regions with poor infrastructure. Further work is required to improve computational efficiency and accessibility of such technologies.
* **Explainability and Transparency**: While Tolba et al. explored class activation maps for explainability, the majority of DL approaches remain black-box models. There is a pressing need for interpretable AI models to enhance farmer trust and practical usability.
* **Integration of Multimodal Data**: Studies primarily focused on single modalities (e.g., hyperspectral or visual imagery). Future research can explore the fusion of multiple data types, such as thermal imagery, environmental data, and soil health, to improve disease prediction accuracy and reliability.
* **Economic and Environmental Viability**: Limited attention has been given to cost-effective and environmentally sustainable disease monitoring solutions for smallholder farmers. Technologies must address these aspects to ensure broader adoption.

These gaps underline the necessity for more comprehensive, scalable, and interpretable solutions in precision agriculture for wheat disease monitoring and management.

## Problem Statement

Current diagnostic methods face key challenges, including variations in lesion size and shape on wheat leaves and inconsistent lighting conditions in field environments, which hinder accurate disease detection. While deep neural networks (DNNs), especially CNNs, offer high accuracy, they require long training times, high computational power, and large, diverse datasets—making real-time field use difficult. This project aims to develop an efficient, accurate wheat rust detection system that adapts to lighting variations, simplifies model complexity, and supports real-time use, ultimately reducing environmental impact by minimizing pesticide misuse.

# Requirements and Design

The Wheat Rust Guard is a mobile based platform that was developed to support farmers in the early detection of wheat rust diseases, such as Yellow Rust and Brown Rust, by applying advanced AI and computer vision techniques. This platform will be user-friendly, very easy to use and will provide convenience and efficiency for farmers and administrators who interact with this system.

## Requirements

### Functional Requirements

* **User / Farmer:**

Table ‑ Functional Requirements for Farmer

|  |  |
| --- | --- |
| **ID** | **Requirements** |
| FR-1.1 | User shall be able to login. |
| FR-1.2 | User shall be able to create profile. |
| FR-1.3 | User shall be able to edit profile. |
| FR-1.4 | User shall be able to capture / upload image for detection of disease. |
| FR-1.5 | User shall be able to receive disease detection results. |
| FR-1.6 | User shall be able to receive treatment suggestions. |
| FR-1.7 | User shall be able to give feedback on treatment suggestion. |
| FR-1.8 | User shall be able to perform offline functionality. |

* **Admin:**

Table ‑ Functional Requirements for Admin

|  |  |
| --- | --- |
| **ID** | **Requirements** |
| FR-2.1 | Admin shall be able to login to the system. |
| FR-2.2 | Admin shall be able to manage AI model integration. |
| FR-2.3 | Admin shall be able to manage user accounts. |
| FR-2.4 | Admin shall be able to manage treatment recommendations. |
| FR-2.5 | Admin shall be able to access dashboard of firebase. |
| FR-2.6 | Admin shall be able to monitor and troubleshoot errors. |

### Non-Functional Requirements

* **User-Friendly Interface:** The User-Interface of the system is very simple and user-friendly, allowing users to navigate easily and access the desired features without any confusion.

### Hardware and Software Requirements

**Hardware Requirements**

**Mobile Device**

Processor: Quad-core (Snapdragon 665 or better)

RAM: 3GB (4GB Recommended)

Storage: 32GB (10MB free for App)

Camera: 8MP with autofocus

OS: Android 8+ / IOS 12+

Network: 3G / 4G / Wi-Fi

**Server**

Processor: Multi-core CPU

GPU: NVIDIA Tesla T4 or equivalent

RAM: 16GB (32GB Recommended)

Storage: 500GB SSD

Network: High-Speed

**Software Requirements**

**Mobile Application**

Framework: Flutter

Language: Dart

Database: Firebase

**AI model Processing**

An AI framework or library to implement wheat disease detection such as tensor-flow.

**Development Tools**

Integrated Development Environments (IDEs) like visual studio code, android studio for application development.

**Version Control**

A version control system like Git to manage source code and collaborate with multiple developers.

## Proposed Methodology

### Overview

Our system Wheat Rust Guard aims to detect and classify wheat rust diseases by leveraging a mobile application integrated with deep learning model. The approach provides a user-friendly solution for users / farmers to identify the wheat rust disease effectively.

### Step-by-Step Process

* **Data Collection:** Firstly, we collected dataset of Wheat Rust such as Yellow Rust, Brown Rust and Healthy from online source such as Kaggle.
* **Data Preprocessing:** This process involves removal of noise, cleaning data and resize dataset to cubic interpolation 224x224x3, also enhance the images quality for better applying deep learning model.
* **Data Augmentation:** It is applied to artificially increase the size and diversity of the dataset, which helps to prevent overfitting and improve the model’s ability to generalize to unseen images. Data augmentation enhances the model’s robustness and accuracy in classifying wheat leaf diseases.
* **Model Selection:** Choose an appropriate convolutional neural network (CNN) architecture for image classification, such as hybrid model to increase the accuracy and correctly identify the disease.
* **Model Training and Evaluation:** Train the selected CNN model using preprocessed dataset. Separate the dataset into train, test and validation with the ratio of 70%, 20% and 10% respectively. The evaluation of model is done through confusion matrix of that architecture on which the selected hybrid model is implemented.
* **Integration:** Developing a mobile app using flutter dart for front end, ensuring an intuitive user interface for users / farmers. Secondly, we implement our backend using tensor-flow lite to integrate our hybrid model with our app, enabling automated wheat rust diseases detection and diagnosis based on uploaded images or captured images.
* **Testing and Validation:** Conduct thorough testing and validation of our system to ensure its accuracy and reliability by going on-field testing of wheat rust diseases and training our dataset on different CNN architectures to ensure best performing model. It is mandatory to evaluate the confusion matrix in order to check for the best CNN model.

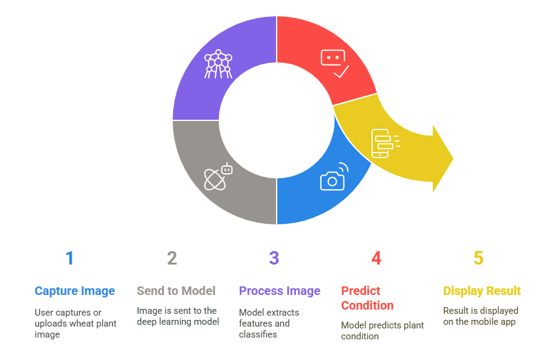


Figure ‑ Proposed Solution Diagram

## Dataset

Table ‑ Wheat Rust Dataset (Excluding Unknown Class)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Disease Class | Images Frequency | | | |
|  | Training | Validation | Testing | Total |
| Brown Rust | 1400 | 200 | 400 | 2000 |
| Yellow Rust | 1400 | 200 | 400 | 2000 |
| Healthy | 1444 | 206 | 413 | 2063 |

### Dataset (Unknown Class)

Table ‑ Wheat Rust Dataset (Including Unknown Class)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Disease Class | Images Frequency | | | |
|  | Training | Validation | Testing | Total |
| Brown Rust | 1400 | 200 | 400 | 2000 |
| Healthy | 1444 | 206 | 413 | 2063 |
| Unknown | 1750 | 256 | 500 | 2506 |
| Yellow Rust | 1400 | 200 | 400 | 2000 |

## Deep Learning Model

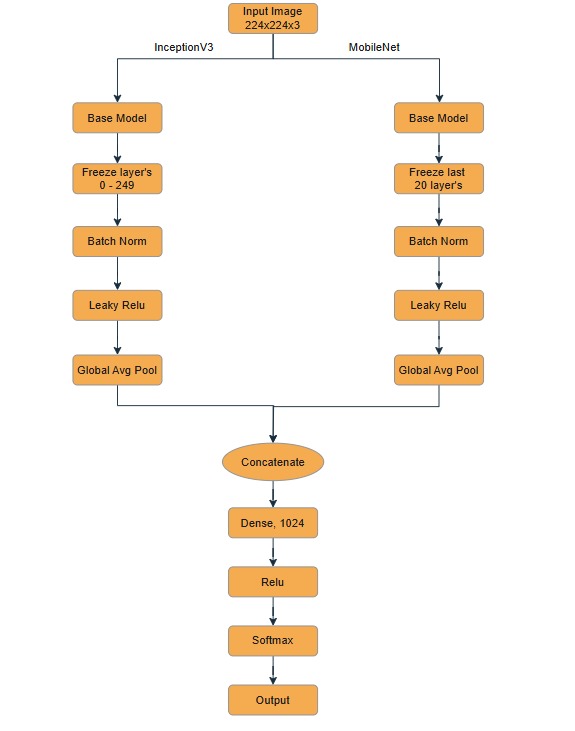


Figure ‑ Deep Learning (CNN-Model Architecture)

This diagram represents a hybrid deep learning model architecture for processing input images (224x224x3). It combines the Inception V3 architecture with a Mobile-Net architecture.

## System Architecture

The major components in the system architecture of the project are three in number and include Image Input, a Mobile Application, and Deep Learning Model Integration. These will classify the wheat rust diseases namely Yellow Rust and Brown Rust. The workflow is as follows:

* **Image Input:** In the interface of the mobile application, the user can upload an image of a corn leaf, which will be the input to the system for its processing and classification.
* **Mobile Application:** The mobile application provides the basic interface to the user. It receives the uploaded image and talks to the deep learning model. The app should be developed in such a way that it presents a very friendly user experience whereby communications between the user and the model can be as smooth as possible.
* **Deep learning model:** In this model, the hybrid CNN analyzes an uploaded image. Then it processes and classifies this image into three major groups, namely Wheat Rust, Brown Rust and Healthy, returning the processed result to the mobile phone application.
* **Output Results:** The results from classification, along with confirmation through visual means and, if possible, type of disease, are reflected in the mobile application. Moreover, the output from this system is in such a format that even a person with no technical knowledge about diseases can understand. The modular architecture ensures that the system provides real-time disease detection, smooth workflow, and classifies the findings with a high degree of accuracy based on deep learning.

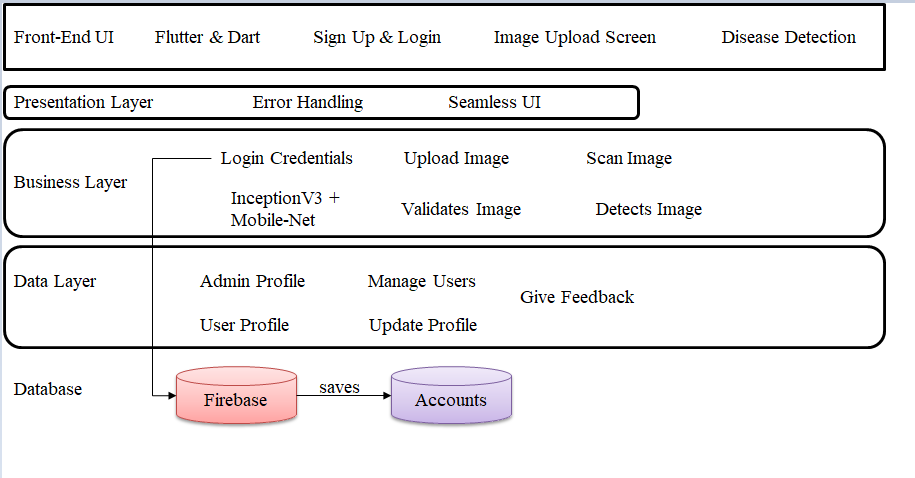


Figure ‑ System Architecture Design of Wheat Rust Guard

## Use Cases

* **User / Farmer:**

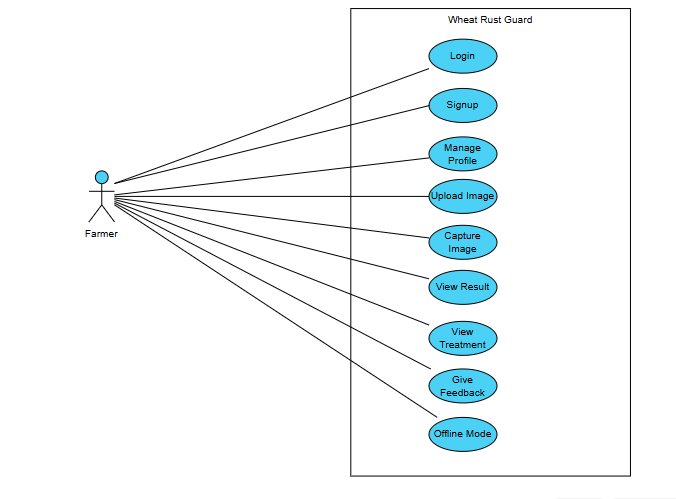


Figure ‑ Use Case Diagram for Farmer

* **Admin:**

A diagram of a person's face

AI-generated content may be incorrect.

Figure ‑ Use Case Diagram for Admin

* **Complete Use Case Diagram:**

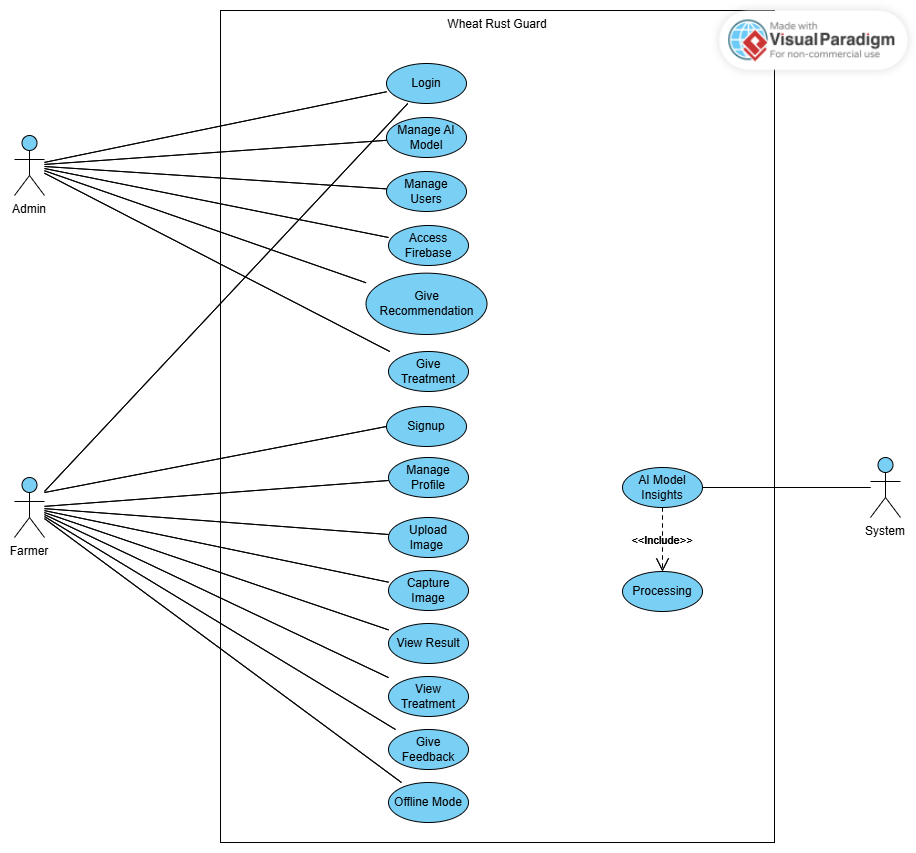


Figure ‑ Complete Use Case Diagram

### Fully Dressed Use Case

#### Login

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | | Login | | |
| Actor | | Admin | | |
| Summary | | Admin logs into the system using authorized credentials. | | |
| Pre-Conditions | | Admin must be registered. | | |
| Post-Conditions | | Admin gains access to the admin dashboard. | | |
| Basic Flow | | | | |
| Actor Action | | | **System Response** | |
| 1 | Enter phone number. | | 2 | System verifies the number and send the OTP on that number. |
| 3 | Enter OTP. | | 4 | System will log in as admin. |
| **Alternative Flow** | | | | |
| 1A | If credentials are invalid, the system shows an error message and prompts for retry. | |

#### Manage AI Model

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | | Manage AI Model | | |
| Actor | | Admin | | |
| Summary | | Allows the admin to view model insights such as accuracy and performance metrics. | | |
| Pre-Conditions | | AI model is embedded in the application. | | |
| Post-Conditions | | Admin has successfully viewed insights. | | |
| Basic Flow | | | | |
| Actor Action | | | **System Response** | |
| 1 | Navigate to “Manage AI Model” screen. | | 2 | Displays AI model name and version (read-only). |
| 3 | Select “View Insights”. | | 4 | Retrieves performance metrics. |
| 5 | Exit the screen. | | 6 | Return to admin dashboard. |

#### Manage Users

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | | Manage Users | | |
| Actor | | Admin | | |
| Summary | | Enables admin to view, search, or delete user accounts from database. | | |
| Pre-Conditions | | Admin is logged in with valid credentials. | | |
| Post-Conditions | | Selected user data is updated or deleted in firebase. | | |
| Basic Flow | | | | |
| Actor Action | | | **System Response** | |
| 1 | Navigate to “Manage Users” screen. | | 2 | Retrieves and displays user list from firebase. |
| 3 | Choose to update user data or delete. | | 4 | Opens editable form with current data or delete user as per requested. |
| 5 | Save changes. | | 6 | Updates user data in Firebase and confirms success. |
| **Alternative Flow** | | | | |
| 3A | Deletion canceled, returns to user list without making any changes. | |

#### Access Firebase

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | | Access Firebase | | |
| Actor | | Admin | | |
| Summary | | Allows the admin to access the firebase backend to read or write application data. This supports administrative maintenance and monitoring tasks. | | |
| Pre-Conditions | | Admin has the required firebase access permission. | | |
| Post-Conditions | | Admin successfully views or modifies firebase data. | | |
| Basic Flow | | | | |
| Actor Action | | | **System Response** | |
| 1 | Select a data type (user, feedback, etc.) | | 2 | Fetches and displays real-time data. |
| 3 | View or filter data records. | | 4 | Shows records in a readable format. |
| 5 | Exit the screen. | | 6 | Returns to admin dashboard. |
| **Alternative Flow** | | | | |
| 3A | Connection failure, shows network error. Failed to connect to firebase. | |

#### Give Recommendation

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | | Give Recommendation | | |
| Actor | | Admin | | |
| Summary | | Allows the admin to give personalized recommendations to farmers such as crop protection tips or preventive measures which are stored and displayed in the farmers app. | | |
| Pre-Conditions | | Detection results are available and stored in the system. | | |
| Post-Conditions | | Farmer can view the recommendation linked to their result in the app. | | |
| Basic Flow | | | | |
| Actor Action | | | **System Response** | |
|  |  | | 1 | System will display the result. |
| 2 | Farmer will be able to see the results according to the given image. | |  |  |
| 3 | Farmer will click on the treatment functionality on the app. | | 4 | The system will give the treatment according to the image provided. |

#### Signup

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | | Signup | | |
| Actor | | Farmer | | |
| Summary | | The farmer creates a new account in the system to access the Wheat Rust Guard features. | | |
| Pre-Conditions | | The use does not already have an account. | | |
| Post-Conditions | | The account is created, and the user is logged in. | | |
| Basic Flow | | | | |
| Actor Action | | | **System Response** | |
| 1 | The farmer selects the “Signup” option on the app. | | 2 | The system displays a form requiring user data. |
| 3 | The farmer fills out the form and submits it. | | 4 | The system validates the details and creates the account. |
| 5 | The farmer is redirected to the login page. | |

#### Give Treatment

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | | Give Treatment | | |
| Actor | | Admin | | |
| Summary | | Allows the Admin to assign a specific treatment plan for wheat rust disease based on image analysis and detection results. | | |
| Pre-Conditions | | Detection result is already available in the system. | | |
| Post-Conditions | | Treatment instructions are saved and linked to the relevant detection entry. | | |
| Basic Flow | | | | |
| Actor Action | | | **System Response** | |
| 1 | Admin picks a treatment from the list. | | 2 | App confirms the selected treatment. |
| 3 | Admin clicks on “Assign Treatment”. | | 4 | Farmer will now see this treatment in their app. |

#### Give Feedback

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | | Give Feedback | | |
| Actor | | Farmer | | |
| Summary | | The farmer provides feedback on the system or treatment recommendation. | | |
| Pre-Conditions | | The farmer has used the system and is logged in. | | |
| Post-Conditions | | Feedback is recorded in the system. | | |
| Basic Flow | | | | |
| Actor Action | | | **System Response** | |
| 1 | The farmer navigates to the ‘Give Feedback’ section. | | 2 | The system displays a feedback form with fields for comments and ratings. |
| 3 | The farmer fills out the form and submits it. | | 4 | The system saves the feedback and displays a thank-you message. |
|  | | | | |

#### Offline Mode

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | | Offline Mode | | |
| Actor | | Farmer | | |
| Summary | | The farmer uses the system in offline mode to process images without an internet connection. | | |
| Pre-Conditions | | The system has pre-downloaded in the user system. | | |
| Post-Conditions | | The farmer receives a diagnosis. | | |
| Basic Flow | | | | |
| Actor Action | | | **System Response** | |
| 1 | The farmer can use when there is no internet as our system does not rely on internet connectivity. | |  |  |
| 3 | The farmer can capture or upload image. | | 4 | The system will process the image and give the diagnosis. |
| 5 | The farmer can view the results. | |  |  |

#### Manage Profile

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | | Manage Profile | | |
| Actor | | Farmer | | |
| Summary | | Allows the farmer to view and update their personal profile details, including full name, email, and mobile number. | | |
| Pre-Conditions | | Farmer is logged in using phone number OTP. | | |
| Post-Conditions | | Changes are reflected in the user’s account across the app. | | |
| Basic Flow | | | | |
| Actor Action | | | **System Response** | |
| 1 | Opens “Profile” screen. | | 2 | Loads and displays current Full Name, Email, Mobile No. |
| 3 | Clicks “Edit” button. | | 4 | Shows editable form with existing profile details. |
| 5 | Updates any field. | | 6 | Validates inputs. |
| 7 | Clicks “Save”. | | 8 | Profile updated successfully. |
| **Alternative Flow** | | | | |
| 5A | If leaves any required field blank, shows error: All fields must be filled. | |

#### Upload Image

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | | Upload Image | | |
| Actor | | Farmer | | |
| Summary | | Allows the farmer to upload an image of a wheat leaf (from their device's gallery) for disease detection. | | |
| Pre-Conditions | | Farmer is logged in. | | |
| Post-Conditions | | Farmer can view result and treatment. | | |
| Basic Flow | | | | |
| Actor Action | | | **System Response** | |
| 1 | Navigates to the "Upload Image" screen. | | 2 | Shows options to select image from gallery. |
| 3 | Clicks “Choose Image”. | | 4 | Opens phone's gallery to pick an image. |
| 5 | Clicks on the image you want to detect. | | 6 | Sends image to AI model for processing. |
| **Alternative Flow** | | | | |
| 5A | No image selected, please select an image to proceed. | |

#### Capture Image

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | | Capture Image | | |
| Actor | | Farmer | | |
| Summary | | Enables the farmer to capture a live photo of a wheat leaf using the device camera. The image is used to detect disease via the AI model. | | |
| Pre-Conditions | | Device camera is functioning. | | |
| Post-Conditions | | Farmer can view disease result and treatment. | | |
| Basic Flow | | | | |
| Actor Action | | | **System Response** | |
| 1 | Navigates to “Capture Image” screen. | | 2 | Opens in-app camera interface. |
| 3 | Frames the wheat leaf in view. | | 4 | Shows live camera preview. |
| 5 | Clicks “Capture”. | | 6 | Takes photo and shows confirmation preview. |
| 7 | Clicks “Submit” or “Use Photo”. | | 8 | Passes image to AI model for disease prediction. |
| **Alternative Flow** | | | | |
| 1A | Camera permission denied, Camera access is required to capture image. | |

#### View Results

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | | View Results | | |
| Actor | | Farmer | | |
| Summary | | Allows the farmer to view past and recent detection results. | | |
| Pre-Conditions | | Internet connection is available (for online sync). | | |
| Post-Conditions | | Farmer can view a list of results. | | |
| Basic Flow | | | | |
| Actor Action | | | **System Response** | |
| 1 | Farmer upload or capture image. | |  |  |
| 2 | Farmer will press “Detect Disease”. | | 3 | System will show the disease results. |

#### View Treatment

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | | View Treatment | | |
| Actor | | Farmer | | |
| Summary | | Allows the farmer to view treatment advice based on the disease detected by the AI model. | | |
| Pre-Conditions | | The app has access to treatment data. | | |
| Post-Conditions | | Farmer can read treatment advice for the specific disease. | | |
| Basic Flow | | | | |
| Actor Action | | | **System Response** | |
| 1 | Farmer after uploading image press the detect disease button. | | 2 | System will show the disease accordingly. |
|  |  | | 3 | System will pop up a message about treatment. |
| 4 | Farmer can view the treatment according to the disease. | |  |  |

## Class Diagram

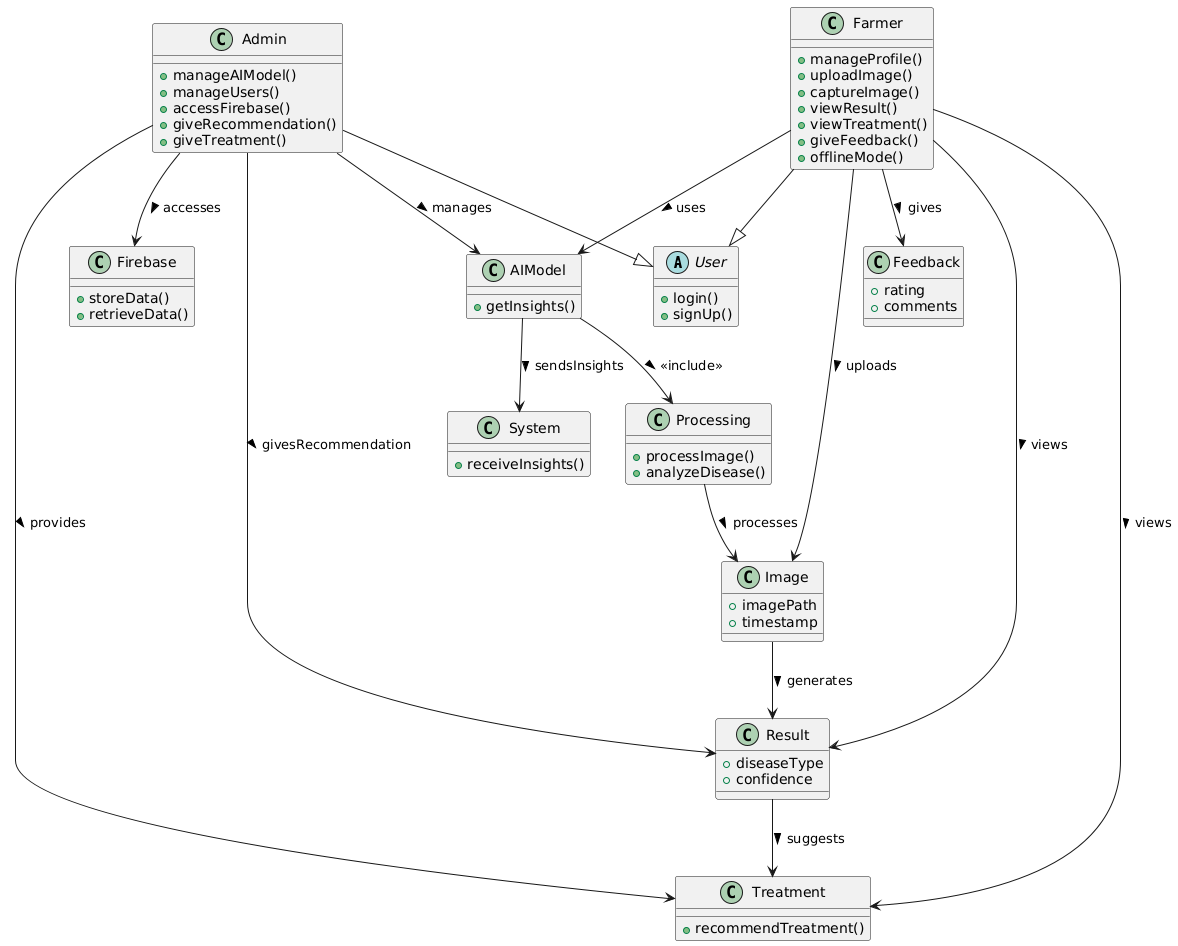


Figure ‑ Class Diagram

## Database Schema

Here is a Firebase Fire-store schema for our Wheat Rust Guard project.

**User’s Collection:**

Table ‑ Database Schema of User's Collection

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Field Name** | **Type** | **Required?** | **Description** | **Indexes** |
| uid | String | Yes | Firebase Auth UID (Primary key) | uid (default) |
| name | String | Yes | Full name of the user | — |
| phone | String | Yes | User's phone number (used for OTP) | — |
| profileImage | String | No | User's profile picture | — |
| createdAt | Timestamp | Yes | Account creation date | — |
| updatedAt | Timestamp | Yes | Last updated date | — |

**Detection Results Collection:**

Table ‑ Database Schema of Detection Results Collection

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Field Name** | **Type** | **Required?** | **Description** | **Indexes** |
| resultId | String | Yes | Unique ID for detection result (UUID/auto-ID) | resultId (default) |
| userId | String | Yes | UID of the user (farmer) | — |
| image | String | Yes | Uploaded image | — |
| result | String | Yes | Detected class (Healthy, Brown Rust, etc.) | — |
| timestamp | Timestamp | Yes | Date and time of detection | — |

**Feedback Collection:**

Table ‑ Database Schema of Feedback Collection

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Field Name** | **Type** | **Required?** | **Description** | **Indexes** |
| feedbackId | String | Yes | Unique ID for feedback | feedbackId (default) |
| userId | String | Yes | Reference to user who submitted feedback | — |
| rating | Number | Yes | Rating out of 5 | — |
| comment | String | No | Optional feedback text | — |
| timestamp | Timestamp | Yes | Date and time feedback was submitted | — |

**Crop Diary Collection:**

Table ‑ Database Schema of Crop Diary Collection

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Field Name** | **Type** | **Required?** | **Description** | **Indexes** |
| entryId | String | Yes | Unique ID for the diary entry (auto-generated by Firestore) | entryId (default) |
| userId | String | Yes | Firebase Auth UID of the user (part of the document path) | — |
| activity\_type | String | Yes | Type of activity (e.g., Sowing, Irrigation, Fertilizing) | — |
| crop\_stage | String | Yes | Growth stage of the crop (e.g., Germination, Tillering, etc.) | — |
| notes | String | Yes | User-provided notes related to the activity | — |
| date | Timestamp | Yes | Timestamp of when the entry was created (used for sorting) | date (optional) |

# Implementation and Test Cases

## Implementation

This chapter details the steps involved in implementing the proposed wheat disease detection system using (methodology/technology). It covers the model selection, confusion matrix and performance assessment.

### Model Performance Analysis: Mobile-Net with Relu Activation

**Confusion Matrix:**

Table ‑ Performance Assessment MobileNet-Relu

|  |  |  |  |
| --- | --- | --- | --- |
|  | Brown Rust | Healthy | Yellow Rust |
| Brown Rust | 318 | 37 | 45 |
| Healthy | 7 | 384 | 7 |
| Yellow Rust | 9 | 45 | 346 |

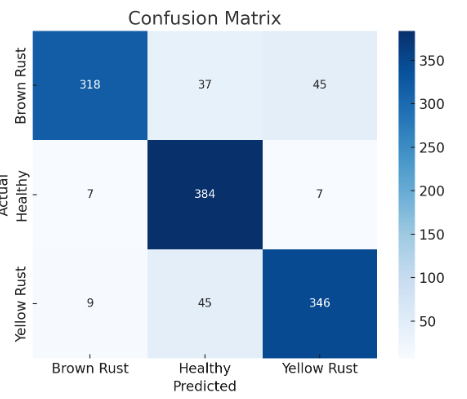


Figure ‑ Confusion Matrix MobileNet-Relu

Performance Evaluation:

Table ‑ Performance Evaluation MobileNet-Relu

|  |  |  |  |
| --- | --- | --- | --- |
| **Precision** | **Recall** | **F1-Score** | **Accuracy** |
| 88.19 | 87.47 | 87.41 | 87.47 |

### Model Performance Analysis: MobileNet with Swish Activation

**Confusion Matrix:**

Table ‑ Performance Assessment MobileNet-Swish

|  |  |  |  |
| --- | --- | --- | --- |
|  | Brown Rust | Healthy | Yellow Rust |
| Brown Rust | 364 | 19 | 17 |
| Healthy | 17 | 375 | 6 |
| Yellow Rust | 41 | 30 | 329 |

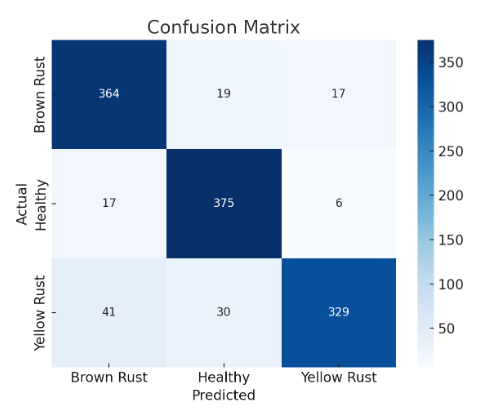


Figure ‑ Confusion Matrix MobileNet-Swish

Performance Evaluation:

Table ‑ Performance Evaluation MobileNet-Swish

|  |  |  |  |
| --- | --- | --- | --- |
| **Precision** | **Recall** | **F1-Score** | **Accuracy** |
| 89.38 | 89.14 | 89.09 | 89.14 |

### Model Performance Analysis: InceptionV3 using Relu Activation

**Confusion Matrix:**

Table ‑ Performance Assessment InceptionV3-Relu

|  |  |  |  |
| --- | --- | --- | --- |
|  | Brown Rust | Healthy | Yellow Rust |
| Brown Rust | 361 | 9 | 30 |
| Healthy | 6 | 380 | 12 |
| Yellow Rust | 15 | 9 | 376 |

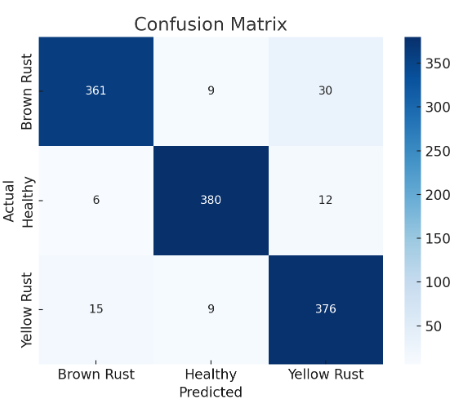


Figure ‑ Confusion Matrix InceptionV3-Relu

**Performance Evaluation:**

Table ‑ Performance Evaluation InceptionV3-Relu

|  |  |  |  |
| --- | --- | --- | --- |
| **Precision** | **Recall** | **F1-Score** | **Accuracy** |
| 93.30 | 93.23 | 93.24 | 93.23 |

### Model Performance Analysis: InceptionV3 using Swish Activation

**Confusion Matrix:**

Table ‑ Performance Assessment InceptionV3-Swish

|  |  |  |  |
| --- | --- | --- | --- |
|  | Brown Rust | Healthy | Yellow Rust |
| Brown Rust | 380 | 11 | 9 |
| Healthy | 3 | 391 | 4 |
| Yellow Rust | 10 | 15 | 375 |

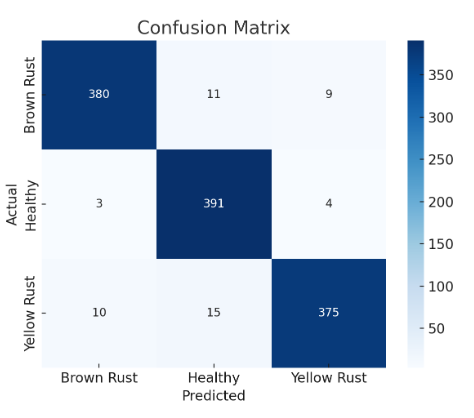


Figure ‑Confusion Matrix InceptionV3-Swish

**Performance Evaluation:**

Table ‑ Performance Evaluation InceptionV3-Swish

|  |  |  |  |
| --- | --- | --- | --- |
| **Precision** | **Recall** | **F1-Score** | **Accuracy** |
| 95.70 | 95.65 | 95.65 | 95.65 |

### Model Performance Analysis: DenseNet121 using Relu Activation

**Confusion Matrix:**

Table ‑ Performance Assessment DenseNet121-Relu

|  |  |  |  |
| --- | --- | --- | --- |
|  | Brown Rust | Healthy | Yellow Rust |
| Brown Rust | 317 | 15 | 68 |
| Healthy | 4 | 350 | 44 |
| Yellow Rust | 55 | 41 | 304 |

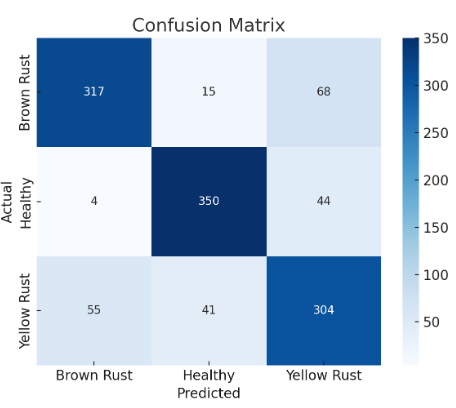


Figure ‑ Confusion Matrix DenseNet121-Relu

**Performance Evaluation:**

Table ‑ Performance Evaluation DenseNet121-Relu

|  |  |  |  |
| --- | --- | --- | --- |
| **Precision** | **Recall** | **F1-Score** | **Accuracy** |
| 81.18 | 81.05 | 81.08 | 81.05 |

### Model Performance Analysis: DenseNet121 using Swish Activation

**Confusion Matrix:**

Table ‑ Performance Assessment DenseNet121-Swish

|  |  |  |  |
| --- | --- | --- | --- |
|  | Brown Rust | Healthy | Yellow Rust |
| Brown Rust | 310 | 14 | 76 |
| Healthy | 3 | 337 | 58 |
| Yellow Rust | 58 | 42 | 300 |

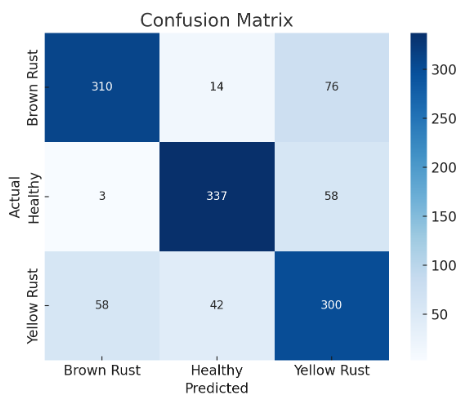


Figure ‑ Confusion Matrix DenseNet121-Swish

**Performance Evaluation:**

Table ‑ Performance Evaluation DenseNet121-Swish

|  |  |  |  |
| --- | --- | --- | --- |
| **Precision** | **Recall** | **F1-Score** | **Accuracy** |
| 79.46 | 79.04 | 79.17 | 79.04 |

### Model Performance Analysis: ResNet50 using Relu Activation

**Confusion Matrix:**

Table ‑ Performance Assessment ResNet50-Relu

|  |  |  |  |
| --- | --- | --- | --- |
|  | Brown Rust | Healthy | Yellow Rust |
| Brown Rust | 348 | 12 | 40 |
| Healthy | 30 | 326 | 42 |
| Yellow Rust | 52 | 26 | 322 |

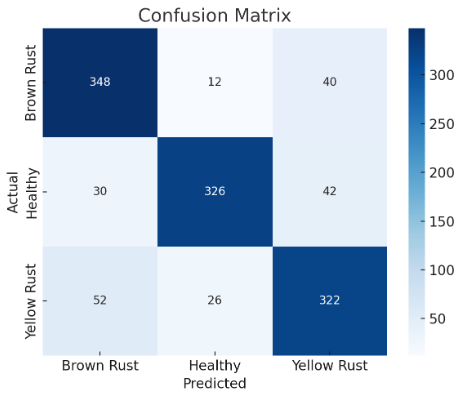


Figure ‑ Confusion Matrix ResNet50-Relu

**Performance Evaluation:**

Table ‑ Performance Evaluation ResNet50-Relu

|  |  |  |  |
| --- | --- | --- | --- |
| **Precision** | **Recall** | **F1-Score** | **Accuracy** |
| 83.38 | 83.13 | 83.16 | 83.13 |

### Model Performance Analysis: ResNet50 using Swish Activation

**Confusion Matrix:**

Table ‑ Performance Assessment ResNet50-Swish

|  |  |  |  |
| --- | --- | --- | --- |
|  | Brown Rust | Healthy | Yellow Rust |
| Brown Rust | 339 | 22 | 39 |
| Healthy | 24 | 340 | 34 |
| Yellow Rust | 52 | 33 | 315 |

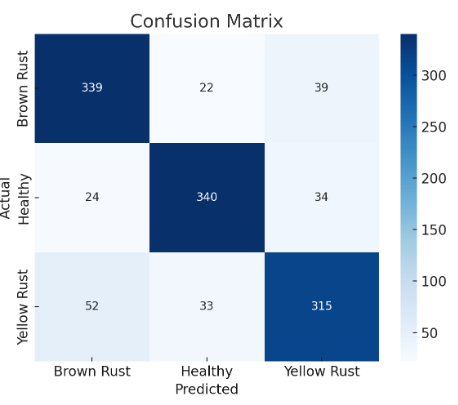


Figure ‑ Confusion Matrix ResNet50-Swish

**Performance Evaluation:**

Table ‑ Performance Evaluation ResNet50-Swish

|  |  |  |  |
| --- | --- | --- | --- |
| **Precision** | **Recall** | **F1-Score** | **Accuracy** |
| 82.97 | 82.97 | 82.95 | 82.97 |

### Model Performance Analysis: VGG16 using Relu Activation

**Confusion Matrix:**

Table ‑ Performance Assessment VGG16-Relu

|  |  |  |  |
| --- | --- | --- | --- |
|  | Brown Rust | Healthy | Yellow Rust |
| Brown Rust | 357 | 5 | 38 |
| Healthy | 7 | 365 | 26 |
| Yellow Rust | 10 | 6 | 384 |

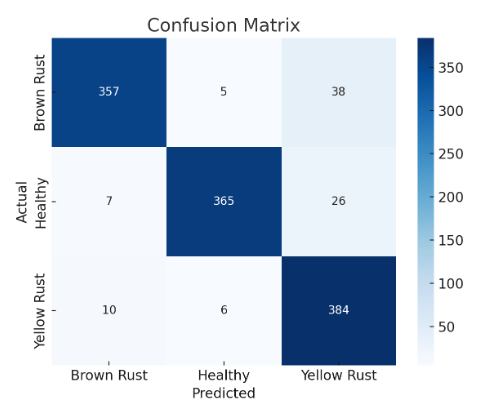


Figure ‑ Confusion Matrix VGG16-Relu

**Performance Evaluation:**

Table ‑ Performance Evaluation VGG16-Relu

|  |  |  |  |
| --- | --- | --- | --- |
| **Precision** | **Recall** | **F1-Score** | **Accuracy** |
| 92.74 | 92.32 | 92.37 | 92.32 |

### Model Performance Analysis: VGG16 using Swish Activation

**Confusion Matrix:**

Table ‑ Performance Assessment VGG16-Swish

|  |  |  |  |
| --- | --- | --- | --- |
|  | Brown Rust | Healthy | Yellow Rust |
| Brown Rust | 342 | 7 | 51 |
| Healthy | 2 | 366 | 30 |
| Yellow Rust | 6 | 6 | 388 |

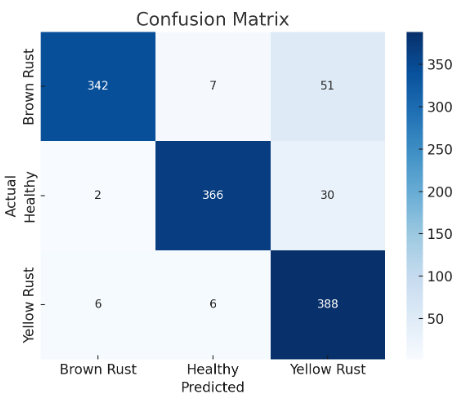


Figure ‑ Confusion Matrix VGG16-Swish

**Performance Evaluation:**

Table ‑ Performance Evaluation VGG16-Swish

|  |  |  |  |
| --- | --- | --- | --- |
| **Precision** | **Recall** | **F1-Score** | **Accuracy** |
| 92.23 | 91.48 | 91.56 | 91.48 |

### Model Performance Analysis: Hybrid Model using Relu Activation

**Confusion Matrix:**

Table ‑ Performance Assessment Hybrid Model-Relu

|  |  |  |  |
| --- | --- | --- | --- |
|  | Brown Rust | Healthy | Yellow Rust |
| Brown Rust | 387 | 5 | 8 |
| Healthy | 20 | 370 | 8 |
| Yellow Rust | 27 | 8 | 365 |

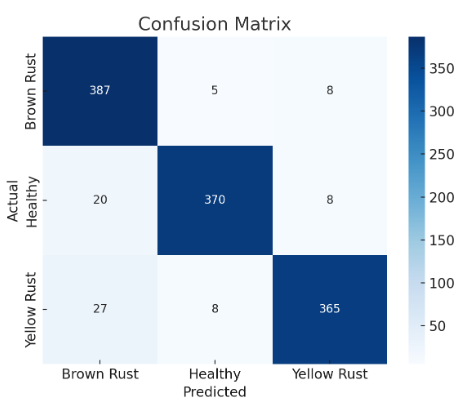


Figure ‑ Confusion Matrix Hybrid Model-Relu

**Performance Evaluation:**

Table ‑ Performance Evaluation Hybrid Model-Relu

|  |  |  |  |
| --- | --- | --- | --- |
| **Precision** | **Recall** | **F1-Score** | **Accuracy** |
| 93.85 | 93.65 | 93.67 | 93.65 |

### Model Performance Analysis: Hybrid Model using Leaky Relu Activation

**Confusion Matrix:**

Table ‑ Performance Assessment Hybrid Model-Leaky Relu

|  |  |  |  |
| --- | --- | --- | --- |
|  | Brown Rust | Healthy | Yellow Rust |
| Brown Rust | 368 | 7 | 25 |
| Healthy | 8 | 375 | 15 |
| Yellow Rust | 3 | 4 | 393 |

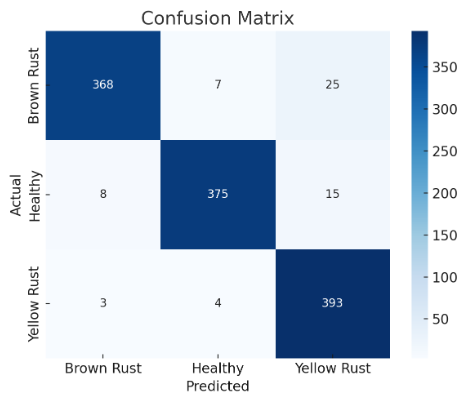


Figure ‑ Confusion Matrix Hybrid Model-Leaky Relu

**Performance Evaluation:**

Table ‑ Performance Evaluation Hybrid Model-Leaky Relu

|  |  |  |  |
| --- | --- | --- | --- |
| **Precision** | **Recall** | **F1-Score** | **Accuracy** |
| 94.99 | 94.82 | 94.83 | 94.82 |

### Model Performance Analysis: Hybrid Model with Unknown Class using Leaky Relu Activation

**Confusion Matrix:**

Table ‑ Performance Assessment Hybrid Model (Unknown Class)-Leaky Relu

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Brown Rust | Healthy | Unknown | Yellow Rust |
| Brown Rust | 395 | 4 | 0 | 1 |
| Healthy | 5 | 389 | 1 | 3 |
| Unknown | 1 | 1 | 498 | 0 |
| Yellow Rust | 13 | 1 | 0 | 386 |

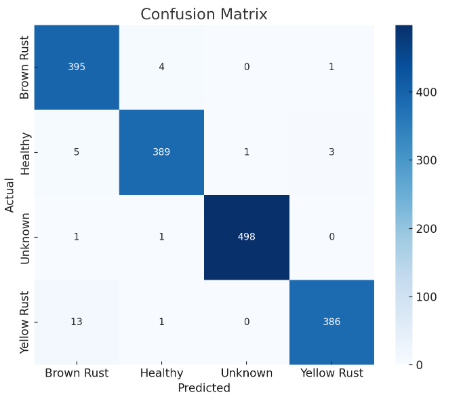


Figure ‑ Confusion Matrix Hybrid Model (Unknown Class)-Leaky Relu

**Performance Evaluation:**

Table ‑ Performance Evaluation Hybrid Model (Unknown Class) -Leaky Relu

|  |  |  |  |
| --- | --- | --- | --- |
| **Precision** | **Recall** | **F1-Score** | **Accuracy** |
| 98.26 | 98.23 | 98.23 | 98.23 |

## Comparative Analysis

Table ‑ Comparative Analysis of Each Model

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Model** | **Activation Function** | **Precision %** | **Recall %** | **F1-Score %** | **Accuracy %** |
| Inception-V3 | Relu | 93.30 | 93.23 | 93.24 | 93.23 |
| Swish | 95.70 | 95.65 | 95.65 | 95.65 |
| ResNet-50 | Relu | 83.38 | 83.13 | 83.16 | 83.13 |
| Swish | 82.97 | 82.97 | 82.95 | 82.97 |
| Mobile Net | Relu | 88.19 | 87.47 | 87.41 | 87.47 |
| Swish | 89.38 | 89.14 | 89.09 | 89.14 |
| VGG-16 | Relu | 92.74 | 92.32 | 92.37 | 92.32 |
| Swish | 92.33 | 91.48 | 91.56 | 91.48 |
| Dense Net-121 | Relu | 81.18 | 81.05 | 81.08 | 81.05 |
| Swish | 79.46 | 79.04 | 79.17 | 79.04 |
| Hybrid Model | Relu | 93.85 | 93.65 | 93.67 | 93.65 |
| Leaky Relu | 94.99 | 94.82 | 94.83 | 94.82 |
| Hybrid Model (Unknown Class) | Leaky Relu | 98.26 | 98.23 | 98.23 | 98.23 |

We choose the best performing model as our proposed solution which concatenates InceptionV3 and Mobile-Net out showing the best performance among all models.

## Mobile Application Development

This part offers an insight into the implementation phase of my Final Year Project, which involves developing a mobile app with Flutter and Dart. The app was built with a friendly user interface with functionality along with aesthetics in mind. During development, screens layouts, navigation, and responsive design were considered with care to provide an improved user experience across various devices. Firebase was included as a backend service for handling real-time data and as the main database to facilitate efficient data storage, authentication, as well as synchronization. The synergy of Flutter's dynamic UI framework and Firebase's powerful backend features allowed for developing a high-performance mobile app.

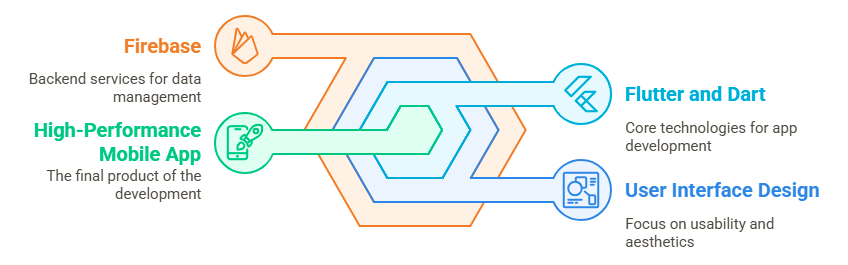


Figure ‑ Mobile Application Development Process

### UI/UX Design

Before developing the mobile app, there was careful planning and creation of the UI/UX design through Adobe XD. The design phase was important in giving shape to how the app looked overall, including in terms of structure, composition, and flow. Pre-designing screens and how they would work together allowed for easier detection of any usability problems and their correction early in development. The design allowed for creating an overall design language, structuring navigation routes, and providing smooth, intuitive interactions. Not only did all this save time in development, but it also ensured that the final product would fit closely with expectations from users as well as with contemporary design principles.

## **Test case Design and description**

In this section, we provide well-structured and in-depth test cases that ensure the functional response of the Wheat Rust Guard mobile app. Each test case confirms a critical element of the system. The style adopted is consistent with leading practices of software quality assurance processes.

### Test Case 1-Image Upload and Detection

Table ‑ TC-Image Upload and Detection

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Disease Detection Engine (Mobile)** | | | | | |
| **AI Classifier** | | | | | |
| Test Case ID: | | *TC-FARM-01* | Test Date: | | *05/04/2025* |
| Test case Version: | | *1.0* | Use Case Reference(s): | | *UC-FARMER-01(Scan/Upload for Detection)* |
| Revision History: | | *New test case* | | | |
| Objective | | *Validate that the system accurately classifies wheat rust based upon an uploaded image* | | | |
| Product/Ver/Module: | | *Wheat Rust Guard v1.0 / Detection Module* | | | |
| Environment: | | *Android 11, 12GB RAM / TensorFlow Lite, Dart SDK* | | | |
| Assumptions: | | *The pre-trained model is deployed, user grants camera/gallery permission* | | | |
| Pre-Requisite: | | *App installed and functional, ready to use* | | | |
| Step No. | Execution description | | | Procedure result | |
| 1 | *Launch app and go to detection screen* | | | *App loads detection interface* | |
| 2 | *Upload image from gallery or scan leaf* | | | *Image successfully loaded* | |
| 3 | *Tap “Detect Disease”* | | | *Classification triggered* | |
| 4 | *Display detected class* | | | *Displays: Correct disease* | |
| Comments:  Detection works accurately. Response time ~1.0s. | | | | | |
| *Passed* *Failed* *Not Executed* | | | | | |

### Test Case 2-Offline Disease Detection

Table ‑ TC-Offline Disease Detection

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Offline Mode** | | | | | |
| **Local AI Inference** | | | | | |
| Test Case ID: | | *TC-FARM-02* | Test Date: | | *05/04/2025* |
| Test case Version: | | *1.0* | Use Case Reference(s): | | *UC-FARMER-02(Offline Scan)* |
| Revision History: | | *New test case* | | | |
| Objective | | *Verify image classification in airplane/offline mode* | | | |
| Product/Ver/Module: | | *Wheat Rust Guard v1.0 / Offline* | | | |
| Environment: | | *Android 11, 12GB RAM / TensorFlow Lite, No internet* | | | |
| Assumptions: | | *TF Lite model installed within app bundle* | | | |
| Pre-Requisite: | | *App opened at least once with model loaded locally* | | | |
| Step No. | Execution description | | | Procedure result | |
| 1 | *Turn off Wi-Fi and mobile data* | | | *Device offline* | |
| 2 | *Upload a test image* | | | *Image loaded* | |
| 3 | *Tap “Detect Disease”* | | | *Classifier works locally* | |
| 4 | *Output result* | | | *Displays: Correct disease* | |
| Comments:  Detection works accurately. Response time ~1.1s. | | | | | |
| *Passed Failed Not Executed* | | | | | |

### Test Case 3-Profile Update and Firebase Sync

Table ‑ TC-Profile Update and Firebase Sync

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **User Profile** | | | | | |
| **Firebase Auth and Realtime DB** | | | | | |
| Test Case ID: | | *TC-03* | Test Date: | | *05/04/2025* |
| Test case Version: | | *1.0* | Use Case Reference(s): | | *UC-ADMIN-01*  *UC\_FARMER-03* |
| Revision History: | | *New test case* | | | |
| Objective | | *Ensure updates to profile are synced with firebase* | | | |
| Product/Ver/Module: | | *Wheat Rust Guard v1.0 / User Module* | | | |
| Environment: | | *Android 11, 12GB RAM / Firebase* | | | |
| Assumptions: | | *Authenticated user session* | | | |
| Pre-Requisite: | | *Active firebase project and user login* | | | |
| Step No. | Execution description | | | Procedure result | |
| 1 | *Go to profile screen* | | | *User details loaded* | |
| 2 | *Update name and phone number* | | | *Data entered* | |
| 3 | *Tap “Save”* | | | *Changes sent to firebase* | |
| 4 | *Refresh app* | | | *Updated info shown* | |
| Comments:  Firebase sync successful within 1.0s. No overwrite or data loss occurred. | | | | | |
| *Passed Failed Not Executed* | | | | | |

### Test Case 4-Result Logging in Previous History

Table ‑ TC-Result Logging in Previous History

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **History Logger** | | | | | |
| **Firebase Cloud Firestore** | | | | | |
| Test Case ID: | | *TC-04* | Test Date: | | *05/04/2025* |
| Test case Version: | | *1.0* | Use Case Reference(s): | | *UC-FARMER-04* |
| Revision History: | | *Second entry* | | | |
| Objective | | *Ensure new results are appended to user’s history* | | | |
| Product/Ver/Module: | | *Wheat Rust Guard v1.0 / Database Module* | | | |
| Environment: | | *Android 11, 12GB RAM / Firebase, Dart SDK* | | | |
| Assumptions: | | *Authenticated user and valid scan result* | | | |
| Pre-Requisite: | | *Detection result obtained successfully* | | | |
| Step No. | Execution description | | | Procedure result | |
| 1 | *Complete disease detection process* | | | *Detection successful* | |
| 2 | *Result saved to history* | | | *Logs data to firebase* | |
| 3 | *Navigate to precious history* | | | *Previous results appear* | |
| 4 | *Display detected class* | | | *Displays: Correct disease* | |
| Comments:  Accurate logging. Accessible from history view. | | | | | |
| *Passed Failed Not Executed* | | | | | |

### Test Case 5-UI/UX: Onboarding and Language Switching

Table ‑ TC-UI/UX: Onboarding and Language Switching

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **UI/Localization** | | | | | |
| **Easy Localization** | | | | | |
| Test Case ID: | | *TC-05* | Test Date: | | *05/04/2025* |
| Test case Version: | | *1.0* | Use Case Reference(s): | | *UC-FARMER-5* |
| Revision History: | | *New test case* | | | |
| Objective | | *Validate onboarding screens and language toggle* | | | |
| Product/Ver/Module: | | *Wheat Rust Guard v1.0* | | | |
| Environment: | | *Android 11, 12GB RAM / Flutter + Easy Localization* | | | |
| Assumptions: | | *Locale files correctly loaded* | | | |
| Pre-Requisite: | | *App installed* | | | |
| Step No. | Execution description | | | Procedure result | |
| 1 | *Launch app for first time* | | | *Onboarding shown* | |
| 2 | *Swipe through onboarding screens* | | | *All messages appear* | |
| 3 | *Change language to Urdu or Punjabi* | | | *Language updates across app* | |
| 4 | *Return to home* | | | *Interface remains localized* | |
| Comments:  Localization functional, Onboarding clear and helpful. | | | | | |
| *Passed Failed Not Executed* | | | | | |

### Test Case 6-Crop Diary Entry Creation and Display

Table ‑ TC-Crop Diary Entry Creation and Display

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **UI/Localization** | | | | | |
| **Easy Localization** | | | | | |
| Test Case ID: | | *TC-06* | Test Date: | | *05/04/2025* |
| Test case Version: | | *1.0* | Use Case Reference(s): | | *UC-FARMER-6* |
| Revision History: | | *New test case* | | | |
| Objective | | *Validate that a user can successfully create a crop diary entry and view it correctly.* | | | |
| Product/Ver/Module: | | *Wheat Rust Guard v1.0* | | | |
| Environment: | | *Android 11, 12GB RAM / Flutter + Easy Localization* | | | |
| Assumptions: | | *App has internet access.* | | | |
| Pre-Requisite: | | *App is installed and user is logged in.* | | | |
| Step No. | Execution description | | | Procedure result | |
| 1 | *Navigate to Crop Diary from the home menu.* | | | *Crop diary screen is displayed.* | |
| 2 | *Tap Add entry.* | | | *Entry creation form is shown.* | |
| 3 | *Fill in activity type, crop stage, notes and pick a date.* | | | *All fields accept input values.* | |
| 4 | *Tap Save entry.* | | | *Entry is saved to fire-store and confirmation shown.* | |
| 5 | *Return to Crop Diary main screen.* | | | *New entry is listed with correct values and sorted by date.* | |
| Comments:  Crop diary entry creation and retrieval worked as expected. Sorting by date verified. No UI lag. | | | | | |
| *Passed Failed Not Executed* | | | | | |

## Summary

The conducted test cases successfully validated the core functionalities of the Wheat Rust Guard v1.0 mobile application. These tests ensured a robust and reliable system that supports farmers in detecting and managing wheat rust diseases efficiently. The following key features were verified:

**Online Detection:** Real-time disease identification using uploaded or scanned leaf images with network connectivity.

**Offline Detection:** Local AI inference using TensorFlow Lite, enabling disease detection without internet access.

**Profile Synchronization:** Seamless user profile updates synced to Firebase Realtime Database for persistent user data management.

**Result History Logging:** Accurate and timely logging of detection results in Fire-store, enabling easy review of past scans.

**UI/UX and Localization:** Intuitive onboarding experience and dynamic language switching (Urdu/Punjabi) through Easy Localization.

These tests confirm that the application delivers a complete, user-friendly, and technically sound solution for wheat rust disease detection and management.

**Crop Diary:** They ensured that farmers are able to log activities such as planting, irrigation, and crop stages with notes. Entries are stored under each user's UID in Firestore and render correctly, sorted by date. This makes the feature usable, reliable, and efficiently helps farmers record and track their crop activities.



Figure ‑ Key Tested Modules

## Test Metrics

This section summarizes the attributes and performance of the executed test cases for the Wheat Rust Guard v1.0 system.

### Sample Test case Matric.No.1

|  |  |
| --- | --- |
| Metric: | Purpose |
| Number of Test Cases: | 5 (Detection, Profile sync, Offline mode, History logging and UI/localization) |
| Number of Test Cases Passed: | 5 |
| Number of Test Cases Failed: | 0 |
| Test Case Defect Density: | (0failed\*100)/5-executed = 0% |
| Test Case Effectiveness: | No defects were detected during testing |
| Traceability Matrix: | All test cases are traceable to their respective use cases |

### Sample Test case Metric.No.2

|  |  |
| --- | --- |
| Metric: | Purpose |
| Stability Across Environment | Ensure consistency across different platforms |
| Environment Tested | Android 11 (12GB RAM),TF Lite, Firebase |
| Observed Issues | None |
| Consistency Rate | 100% |

### Sample Test case Metric.No.3

|  |  |
| --- | --- |
| Metric: | Purpose |
| Execution Time Performance | Measure average response time per feature |
| Average Detection Time | 1.0sec |
| Average Firebase Sync Time | 1.0sec |
| Localization Switch Time | Instantaneous |
| Performance Status | Efficient and within acceptable limits |

# Experimental Results and Analysis

## Introduction

The current chapter shows an overall assessment of the detection system of Wheat Rust Guard with various deep learning models and activation functions. The performance of each model was measured with respect to its capability to classify the images of the wheat leaves under Healthy, Yellow Rust, and Brown Rust classes. The measurement parameters were the precision, the recall, F1-score, and overall accuracy that are utmost essential in agricultural real-world applications where accuracy is paramount to avoid losses.

## Experimental Setup

The experiments were performed with a labeled dataset with balanced samples of all three conditions of wheat. Images were pre-processed (resized to 224x224 pixels, normalized) and divided as:

* 70% training
* 20% test
* 10% validation

These models were trained on TensorFlow and optimized for mobile by utilizing TensorFlow Lite. Relu, Swish, and Leaky Relu were applied to measure how these functions contributed to classification accuracy. A Hybrid Model between InceptionV3 and Mobile-Net was also tested, in addition to one that incorporated adding an "Unknown Class" to examine real-world generalizability.

## Performance Results

It compares a variety of deep learning models employed in the classification of wheat rust disease. Three different activation functions (such as Relu, Swish, or Leaky Relu) were each applied to test the respective model’s performance. The comparison is made based on a set of essential metrics: Precision, Recall, F1-Score, and Accuracy in percentage.

**Key Insights:**

1. Hybrid model (Unknown Class) using Leaky Relu achieved the best performance:
   * + Accuracy 98.23%
2. InceptionV3 with Swish trailed closely behind with 95.65%, closely resembling its Relu counterpart with more than 2% advantage. Both ResNet50 and Mobile-Net performed moderately. Swish performed slightly better than Relu in both instances, but not by much.
3. Mobile-Net with Swish demonstrated significant improvement (89.14% vs. 87.47%) and retained speedy inference, perfect for mobile usage.
4. Hybrid model with Leaky Relu surpassed Relu by more than 1%, reaching accuracy of 94.82%.
5. DenseNet121 did worse, particularly with Swish, indicating model complexity didn’t necessarily yield better classification in this case.

**Regarding Activation Functions:**

The default and widely used activation is Relu (Rectified Linear Unit). Swish is a newer function known to improve deep model performance in many cases. Leaky Relu prevents the "dying Relu" issue and performed better in the Hybrid Model.

## Analysis and Discussion

The new experimental results emphasize the importance of model architecture and the activation function in ascertaining the performance of wheat rust disease classification. The findings explicitly indicate that model performance is greatly affected by activation functions, with Swish and Leaky Relu frequently surpassing the standard Relu function in most models that were tested.

The Hybrid Model that integrates InceptionV3 and Mobile-Net performed better in conjunction with Leaky Relu as the activation function. The model scored an impressive accuracy of 98.23% and F1 of 98.23% under the extended classification environment that involved an unseen class. This is indicative of the model's robustness and capacity for generalization, showing great promise for real-world agricultural settings where unseen data is prolific.

Among the individual models, InceptionV3 performed better consistently, particularly in combination with Swish activation and achieved from 93.23% to 95.65% accuracy. Mobile-Net, with its efficiency in computations, also achieved 89.14% accuracy with the help of Swish, and hence it is appropriate for real-time operations in low-powered mobile phones.

The Hybrid Model (excluding the unknown class) also did very well with Leaky Relu with an accuracy of 94.82%, verifying that Leaky Relu solves the problems of Relu, including the dying neuron problem, and improves gradient flow in training. However, DenseNet121 performed worse compared to the rest of the architectures, especially with Swish, in which accuracy fell to 79.04%. This indicates that although Dense-Net is strong, its deeper architecture is susceptible to overfitting or inefficient learning on less complex or small datasets like this one.

In short, the study finds that the Hybrid Model with appropriate activation function (Leaky Relu or Swish) yields major classification performance improvements, particularly in deployment-critical use cases.

## Conclusion

This chapter highlighted the in-depth performance analysis of the upgraded Wheat Rust Guard system with several deep learning models and activation functions. The experimental findings validated that the model architecture and the activation function are determinant elements in ensuring high classification accuracy in wheat rust detection. The Hybrid Model that coupled the power of InceptionV3 and Mobile-Net with Leaky Relu activation function attained the best overall performance and is able to generalize even in cases with unseen classes. It is thus suggested as the best model to deploy in real-world settings. The research further validated that Swish is overall better than Relu as an activation function, resulting in better model performance in the majority of architectures. Swish Mobile-Net became the optimum trade-off in resource-limited environments with comparatively high accuracy complemented by fast inference—perfect for offline usage in agriculture environments.

These findings prove that the Wheat Rust Guard system is technologically sound and deployable in that it has the ability to offer timely and precise disease detection feedback to farmers, supporting effective crop management and further helping reduce yield loss from unsuspected rust infections

# Conclusion and Future Directions

## Conclusion

Wheat Rust Guard was a pragmatic AI-based approach to identifying early indications of the wheat diseases Yellow Rust and Brown Rust to enable efficient management of healthy crops by the farmer. It successfully employs a powerful deep-learning models—most critically a hybrid model that blends InceptionV3 and Mobile-Net and a mobile application built on Flutter in Dart. The integration results in precise disease identification and real-time feedback, even in an offline mode that is imperative for rural or remote areas lacking or having minimal internet connectivity.

The project was guided by clearly defined objectives:

* Precise classification of wheat rust diseases
* Offline mode
* A user-friendly mobile interface (UI/UX)
* Consistent treatment advice

These goals were all successfully met. The hybrid model achieved a high accuracy rate of 94.82%, and the mobile application successfully underwent rigorous functional testing to prove its fluid performance speeds, ease of navigation, and accuracy in handling user input.

Moreover, the model was trained to detect not only the known classes (Healthy, Yellow Rust, Brown Rust), but also an unknown category that portrays unseen or novel disease patterns. The hybrid model was found to have a remarkable accuracy of 98.23% on the unknown category when using the Leaky Relu activation function and demonstrated good generalization and resistance. It is an important aspect for deployment in real-world settings when novel data or unknown diseases might happen.

## Challenges Faced

Multiple challenges arose during both development and implementation stages in both processes of developing an AI model as well as integrating with the mobile application:

**Data Scarcity and Imbalance:** It took considerable effort to procure an adequately diverse and balanced data set of healthy as well as diseased wheat leaves. Most datasets skewed toward one condition, which needed to undergo extensive preprocessing and augmentation.

**Generalization of Models:** To make sure the deep learning model worked effectively in unseen data with fluctuating lighting, background, and ambient surroundings, meticulous data preprocessing and augmentation techniques had to be implemented to enhance robustness.

**Mobile Integration:** The deployment of a lightweight but accurate deep neural network in a mobile app presented performance issues. Optimization using TensorFlow Lite occurred, but keeping in view smooth inference on the device with minimal use of memory and storage space needed rigorous testing.

**UI Design for Non-Tech Users:** The app is intended to be used by farmers with possibly minimal digital literacy. Designing an intuitive experience with minimal learning curve took many iterations and input from would-be end-users.

**Firebase Limitations:** Firebase provided real-time database and authentication functionality, but handling read/write permissions and synchronization of data in environments with poor connectivity posed challenges. The structuring of collections and document paths had to be carefully thought through to avoid performance issues in the database.

**Cross-platform compatibility:** Making the app function across various Android versions (particularly older ones) added problems with permissions, camera usage, and processing in real-time through hardware differences.

## Future Directions

Although Wheat Rust Guard is complete, there are various avenues along which the project can be expanded and enhanced:

**Expanded Disease Detection:** Future releases might include more crop diseases or incorporate more crops in addition to wheat.

**Multilingual Support:** To enhance accessibility, particularly in Pakistan, regional languages such as Pushto, or Sindhi might be included.

**Explainable AI:** Future models may include explain-ability functionality (e.g., heat maps) to assist with explanation to users about how the model diagnosed a disease.

**Portal for Data Collection:** Create a portal through which users can add images to continue enhancing the model and increasing dataset variability.

**Enterprise Scalability:** Working with departments of agriculture or non-governmental organizations might assist in making app-scaling possible to be used throughout larger geographical regions.

## Concluding Insights

The experience with Wheat Rust Guard illustrates how emerging AI and smartphone technologies can be useful in directly addressing actual-agriculture challenges. By tackling one specific and consequential issue - detection of rust diseases in wheat in resource-poor environments - Wheat Rust Guard illustrates how deep learning can have impact in everyday use beyond research labs. We have accumulated rich experience in data preprocessing, optimization of models, mobile integration, and user-centric design over the course of development. We realized how essential it is to balance technical correctness with usability, especially when dealing with non-technical clients such as farmers.

Most importantly, this work sets a solid foundation for developing innovations in the future. It showcases how technology, particularly when developed with accessibility and actual needs, can have far-reaching benefits in important areas such as farming. Wheat Rust Guard is more than an academic exercise; it is scalable technology with tangible ability to assist communities and aid food security in Pakistan and elsewhere.

# References

List all important sources of information which have been consulted for this project

[1] R. Bebronne *et al.*, “In-field proximal sensing of septoria tritici blotch, stripe rust and brown rust in winter wheat by means of reflectance and textural features from multispectral imagery,” *Biosyst Eng*, vol. 197, pp. 257–269, Sep. 2020, doi: 10.1016/j.biosystemseng.2020.06.011.

[2] M. Mandava, S. R. Vinta, H. Ghosh, and I. S. Rahat, “Identification and Categorization of Yellow Rust Infection in Wheat through Deep Learning Techniques,” *EAI Endorsed Transactions on Internet of Things*, vol. 10, 2024, doi: 10.4108/eetiot.4603.

[3] H. Khan, I. U. Haq, M. Munsif, Mustaqeem, S. U. Khan, and M. Y. Lee, “Automated Wheat Diseases Classification Framework Using Advanced Machine Learning Technique,” *Agriculture (Switzerland)*, vol. 12, no. 8, Aug. 2022, doi: 10.3390/agriculture12081226.

[4] H. Wang, Q. Jiang, Z. Sun, S. Cao, and H. Wang, “Identification of Stripe Rust and Leaf Rust on Different Wheat Varieties Based on Image Processing Technology,” *Agronomy*, vol. 13, no. 1, Jan. 2023, doi: 10.3390/agronomy13010260.

[5] U. Shafi *et al.*, “Embedded AI for Wheat Yellow Rust Infection Type Classification,” *IEEE Access*, vol. 11, pp. 23726–23738, 2023, doi: 10.1109/ACCESS.2023.3254430.

[6] C. Cuenca-Romero, O. E. Apolo-Apolo, J. N. Rodríguez Vázquez, G. Egea, and M. Pérez-Ruiz, “Tackling unbalanced datasets for yellow and brown rust detection in wheat,” *Front Plant Sci*, vol. 15, 2024, doi: 10.3389/fpls.2024.1392409.

[7] C. Nguyen, V. Sagan, J. Skobalski, and J. I. Severo, “Early Detection of Wheat Yellow Rust Disease and Its Impact on Terminal Yield with Multi-Spectral UAV-Imagery,” *Remote Sens (Basel)*, vol. 15, no. 13, Jul. 2023, doi: 10.3390/rs15133301.

[8] W. Liu *et al.*, “StripeRust-Pocket: A Mobile-Based Deep Learning Application for Efficient Disease Severity Assessment of Wheat Stripe Rust,” *Plant Phenomics*, vol. 6, 2024, doi: 10.34133/plantphenomics.0201.

[9] A. Alharbi, M. U. G. Khan, and B. Tayyaba, “Wheat Disease Classification Using Continual Learning,” *IEEE Access*, vol. 11, pp. 90016–90026, 2023, doi: 10.1109/ACCESS.2023.3304358.

[10] D. Kumar and V. Kukreja, “Combined CNN with STARGAN for Wheat Yellow Rust Disease Classification,” *International Journal of Computing and Digital Systems*, vol. 13, no. 1, pp. 1081–1095, 2023, doi: 10.12785/ijcds/130187.

[11] A. Tolba and N. Talal, “An Interpretable Deep Learning for Early Detection and Diagnosis of Wheat Leaf Diseases,” *Optimization in Agriculture*, vol. 1, pp. 81–93, May 2024, doi: 10.61356/j.oia.2024.1257.

[12] J. Jiang *et al.*, “Evaluation of Diverse Convolutional Neural Networks and Training Strategies for Wheat Leaf Disease Identification with Field-Acquired Photographs,” *Remote Sens (Basel)*, vol. 14, no. 14, Jul. 2022, doi: 10.3390/rs14143446.

# 

# Appendix

## Appendix A: Dataset Samples

The data employed in this work comprised three classes of wheat leaf images and one class for unknown dataset:

1. Yellow Rust
2. Brown Rust
3. Healthy
4. Unknown

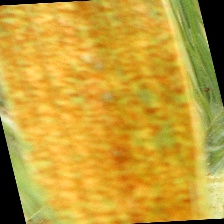
Each class contained images of different angles. Below are sample images from each class.



Appendix A- Yellow Rust Sample 1



Appendix A- Yellow Rust Sample 2



Appendix A- Brown Rust Sample 1



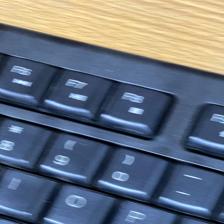
Appendix A- Brown Rust Sample 2



Appendix A- Healthy Sample 1



Appendix A- Healthy Sample 2



Appendix A- Unknown Sample 1



Appendix A- Unknown Sample 2

## Appendix B: Hybrid Model Code Snippet

data\_dir = dataset # Replace 'dataset' with actual dataset path (e.g., './dataset')

# Function to create InceptionV3 architecture with Transfer Learning

def inceptionv3\_base(input\_shape=(224, 224, 3)):

base\_model = tf.keras.applications.InceptionV3(weights='imagenet', include\_top=False, input\_shape=input\_shape)

for layer in base\_model.layers[:249]:

layer.trainable = False

for layer in base\_model.layers[249:]:

layer.trainable = True

x = base\_model.output

x = BatchNormalization()(x)

x = LeakyReLU(alpha=0.1)(x)

x = GlobalAveragePooling2D()(x)

return Model(base\_model.input, x)

# Function to create MobileNet architecture with Transfer Learning

def mobilenet\_base(input\_shape=(224, 224, 3)):

base\_model = tf.keras.applications.MobileNet(weights='imagenet', include\_top=False, input\_shape=input\_shape)

for layer in base\_model.layers[:-20]: # Freeze all but last 20 layers

layer.trainable = False

for layer in base\_model.layers[-20:]:

layer.trainable = True

x = base\_model.output

x = BatchNormalization()(x)

x = LeakyReLU(alpha=0.1)(x)

x = GlobalAveragePooling2D()(x)

return Model(base\_model.input, x)

# Build the feature extraction parts

inception\_base = inceptionv3\_base(input\_shape=(224, 224, 3))

mobilenet\_base = mobilenet\_base(input\_shape=(224, 224, 3))

# Define input

inputs = layers.Input(shape=(224, 224, 3))

inception\_features = inception\_base(inputs)

mobilenet\_features = mobilenet\_base(inputs)

x = Concatenate()([inception\_features, mobilenet\_features])

x = Dense(1024, activation='relu')(x)

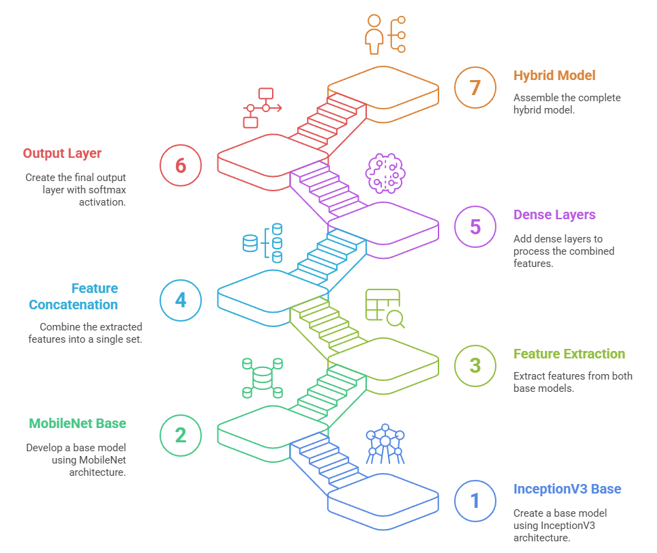
outputs = Dense(4, activation='softmax')(x)

# Create hybrid model

hybrid\_model = Model(inputs, outputs)

# Print model summary

hybrid\_model.summary()



Appendix B- Hybrid Model Overview

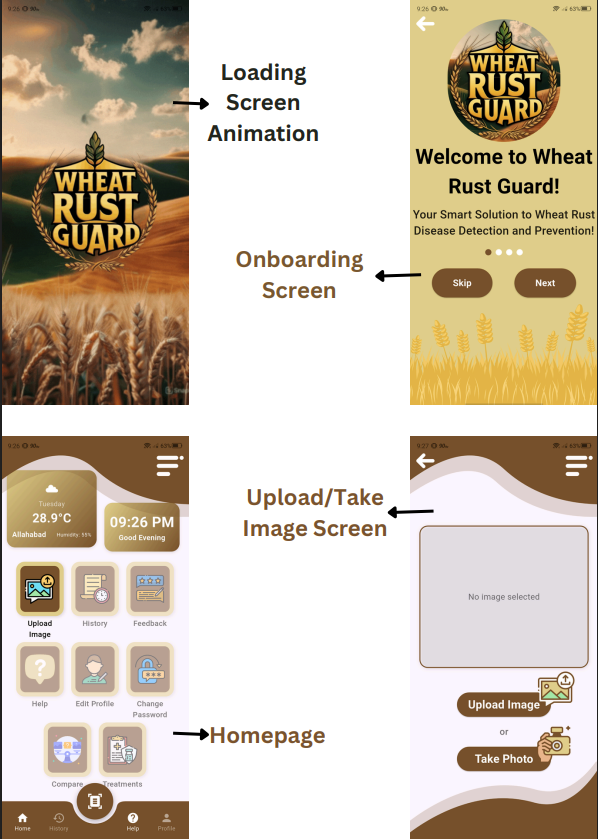
## Appendix C: Mobile Application (UI and Icons)

**LOGO:**

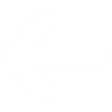


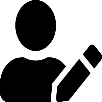
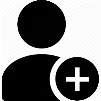
Appendix C- App Logo

App UI Screens:

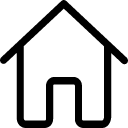


Appendix C- Mobile Screens Interface

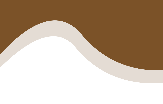
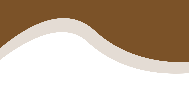
**Assets Used in Our App:**

****

****



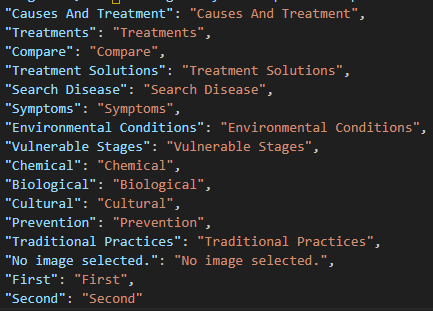




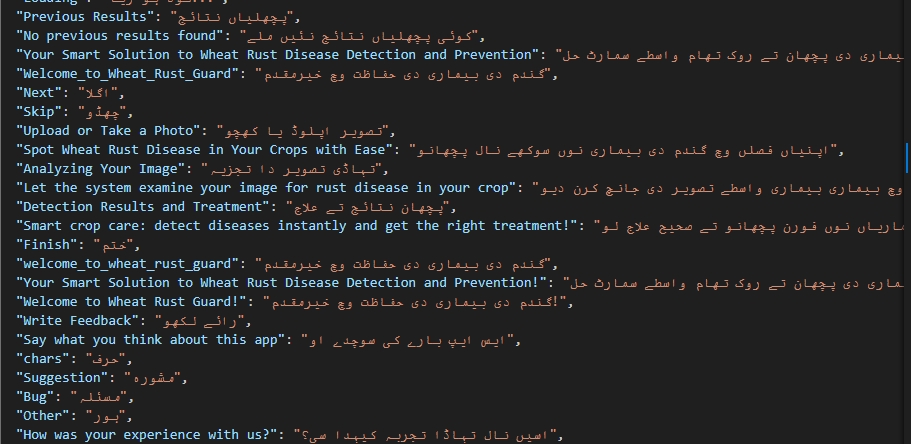
Appendix C- Assets

## Appendix D: Easy-Localization

This appendix contains a comprehensive overview for the implementation of the Easy Localization in the project. Easy Localization is a Flutter package that eases the internationalization of an application as it enables developers to manage and switch between various languages in a simple way. The appendix contains the steps in configuration, integration in the Material-App instance, language JSON file structure, and how the text in widgets is translated. This ensures efficient support for multilingual users within the application to increase usability and accessibility.



Appendix C- English.json

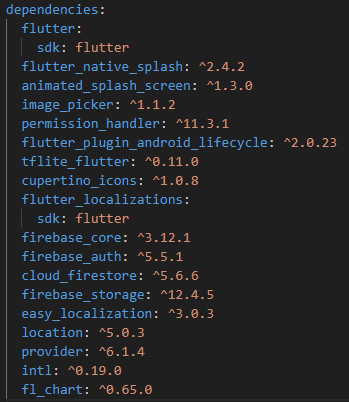
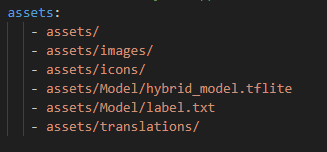


Appendix C- Punjabi.json



Appendix C- Urdu.json

To configure easy localization, first, we need to add dependency and declare the translation assets in the pubspec.yaml file.

Import the package name easy localization in each dart file and make all the text end with .tr()

For example:

Text(‘Wheat Rust Guard’.tr());