CNN-BASED AGRICULTURAL ANALYSIS: DETECTING PLANT DISEASES,FRESHNESS, AND NUTRITIONAL CONTENT IN WHEAT, RICE, FRUITS, AND VEGETABLES

A PROJECT REPORT

Submitted by

NAVEEN KUMARP	**(410721104080)**
SANJAY A	**(410721104103)**
RISHIDHARAN V	**(410721104097)**
THILOCIGAN K	**(410721104118)**

in partial fulfilment for the award of thedegree of

BACHELOR OF ENGINEERING

In

COMPUTER SCIENCE AND ENGINEERING

DHANALAKSHMI COLLEGE OF ENGINEERING

ANNA UNIVERSITY::CHENNAI 600 025

MAY 2025

BONAFIDE CERTIFICATE

Certified that this project report **"CNN-BASED AGRICULTURAL ANALYSIS: DETECTING PLANT DISEASES, FRESHNESS, AND NUTRITIONAL CONTENT IN WHEAT,RICE, FRUITS, AND VEGETABLES"** is the Bonafide work of **"NAVEEN KUMAR P (410721104080), SANJAY A (410721104103), RISHIDHARAN V (410721104097), THILOCIGAN K(410721104118)"** who carried out the project work under my supervision.

SIGNATURE

Dr. S.J.Vivekanandan, B.Tech, M.Tech, Ph.D

HEAD OF THE DEPARTMENT,

Associate Professor

Computer Science and Engineering,

Dhanalakshmi College of Engineering,

DR. VPR Nagar,
Manimangalam,
Chennai – 601301

SIGNATURE

Dr.K.Pandikumar,M.E,Ph.D

PROJECT SUPERVISOR,

Assistant Professor

Computer Science and Engineering,

Dhanalakshmi College of Engineering,

DR. VPR Nagar,
Manimangalam,
Chennai– 601301

Submitted for the project viva-voce examination held at Dhanalakshmi College Engineering on _____.

INTERNAL EXAMINER

EXTERNAL EXAMINER

ACKNOWLEDGEMENT

We express our sincere heartfelt thanks to our honorable Chairman **Dr V P Ramamurthi, Ph.D.** Dhanalakshmi College of Engineering, for guiding us and permitting us to do our project on our own..

We express our heartfelt thanks to our Chairperson **Mrs V R Dhanalakshmi** for her continuous love to our students community. Also, We would like to thank **Ms K S Roopa**, CEO for her sincere guidance and support to us. Also, we express our heartfelt thanks to **Mr Ramesh Vishegu**, Vice Chairman for his great knowledge sharing to our students community.

We wish to thank our beloved Principal **Dr A R Pradeepkumar, B.E., M.E., Ph.D.** for his support and guidance. We extend our gratitude and heartfelt thanks to the Head of the Department, **Dr S J Vivekanandan, M.Tech., Ph.D.** for guiding us in all aspects of our project in each stage and providing us with valuable suggestions.

We would like to thank at most our Project Supervisor, **Dr K Pandi kumar, M.E, Ph.D.,** AP/CSE, who rendered valuable guidance and full support always. We would like to thank at most our Project Coordinator, **Mr M Mohammed Abdhahir, B.Tech., M.Tech., (Ph.D)** AP/CSE, who rendered valuable guidance and full support always.

Finally, we take this opportunity to thank all the faculty members of the Department of Computer Science and Engineering for their unwavering support and cooperation which made us keep our zeal and spirits high to complete this project work successfully.

TABLE OF CONTENTS

ABSTRACT

Agriculture plays a vital role in global food security, and ensuring crop health, freshness, and nutritional quality is essential for sustainable farming practices. This project leverages Convolutional Neural Networks (CNNs) to perform agricultural analysis, focusing on the detection of plant diseases, freshness assessment, and nutritional content estimation in wheat, rice, fruits, and vegetables. By utilizing advanced image processing techniques and deep learning models, this system can accurately classify plant diseases, determine the freshness level of produce, and estimate key nutritional parameters. The model is trained on a diverse dataset of agricultural images, enabling it to identify patterns that indicate disease symptoms, spoilage, and nutrient deficiencies. This approach provides farmers and distributors with a reliable tool for early diagnosis, quality control, and optimized crop management, ultimately enhancing agricultural productivity and reducing food waste. The implementation of this CNN-based analysis system aims to contribute to precision agriculture, ensuring healthier crops and improving food quality for consumers.

LIST OF FIGURES

LIST OF ABBREVIATIONS

S.NO	ABBREVIATIONS	DESCRIPTION
1.	AI	Artificial Intelligence
2.	ML	Machine Learning
3.	DL	Deep Learning
4.	CV	Computer Vision
5.	CNN	Convolutional Neural Network
6.	GPU	Internet of Things
7.	API	Application Programming Interface
8.	GUI	Graphics Processing Unit
9.	UI	User Interface
10.	UX	User Experience

CHAPTER 1

INTRODUCTION

Agriculture plays a vital role in sustaining global food systems, yet it faces challenges such as crop diseases, food spoilage, and inefficiencies in quality monitoring. Leveraging advancements in artificial intelligence, this project introduces a CNN-based system that performs automated detection of plant diseases in rice and wheat, evaluates the freshness of fruits and vegetables, and estimates their shelf life using image analysis. By combining deep learning models with a streamlined user interface and FastAPI backend, the system empowers stakeholders to make data-driven decisions, minimize losses, and enhance the overall efficiency and sustainability of agricultural practices.

1.1 OBJECTIVE

The objective of this project is to develop an intelligent, CNN-based agricultural analysis system that can accurately detect plant diseases in rice and wheat crops, assess the freshness of fruits and vegetables, and estimate their shelf life using visual data. By integrating deep learning models with a user-friendly interface and backend API support, the system aims to assist farmers, vendors, and agricultural stakeholders in making informed decisions that enhance productivity, reduce post-harvest losses, and ensure better quality control across the supply chain.

CHAPTER 2

LITERATURE REVIEW

Comparative study of existing papers

Paper 1 - Fruit and vegetable disease detection and classification: Recent trends, challenges, and future opportunities

Author: Gupta, S.,& Tripathi, A. K.

Year: 2024

Abstract:

This paper provides an in-depth review of recent trends in the detection and classification of diseases in fruits and vegetables, focusing on advancements in machine learning (ML) and computer vision (CV) techniques. The authors highlight the challenges involved in detecting diseases in fresh produce, such as variability in appearance due to environmental factors and the lack of large annotated datasets. The paper also discusses future opportunities, including the integration of deep learning techniques and the development of more robust models for disease classification. The review emphasizes the importance of real-time detection systems to reduce losses in postharvest food quality and safety.

Paper 2 - Intelligent detection for fresh-cut fruit and vegetable processing: Imaging technology

Author: Tang, T., Zhang, M.,& Mujumdar, A. S.

Year: 2022

Abstract:

The paper presents a comprehensive review of imaging technology used in the detection of defects and diseases in fresh-cut fruits and vegetables. The authors explore a range of non-destructive imaging techniques such as hyperspectral imaging, infrared thermography, and X-ray imaging. These technologies provide insights into the internal and external quality of produce without damaging it, making them ideal for industrial applications in food safety and quality control. The paper also examines the integration of these technologies with AI to automate quality assessment in food processing plants.

Paper 3 - Rice diseases detection using convolutional neural networks: a survey

Author: Sharma, R.,& Kukreja, V.

Year: 2021

Abstract:

This survey paper explores the use of Convolutional Neural Networks (CNNs) for rice disease detection, providing an overview of different approaches and methodologies for classifying diseases in rice crops. The authors discuss various CNN architectures, dataset challenges, and model training techniques. The paper highlights the advantages of CNNs in accurately identifying rice diseases, particularly when combined with large-scale datasets. It also examines potential challenges such as the need for labeled data and computational resources for training the models. The paper suggests further exploration into hybrid models combining CNNs with other machine learning algorithms for better performance.

Paper 4 - Wheat rust disease detection techniques: a technical perspective

Author: Shafi, U., Mumtaz, R., Shafaq, Z., Zaidi, S. M. H., Kaifi, M. O., Mahmood, Z., & Zaidi, S. A. R.

Year: 2022

Abstract:

This paper focuses on the detection of wheat rust diseases using various machine learning (ML) and image processing techniques. It covers traditional methods as well as recent advancements in deep learning for early rust disease detection. The authors emphasize the importance of rapid and accurate disease detection to prevent the widespread loss of crops and highlight techniques like spectral imaging and deep learning models that have shown promise in disease classification. The paper also addresses the challenges of implementing these methods in real-time agricultural environments, especially in resource-constrained areas.

Paper 5 - Traditional and current-prospective methods of agricultural plant diseases detection: A review

Author: Khakimov, A., Salakhutdinov, I., Omolikov, A.,& Utaganov, S.

Year: 2022

Abstract:

The paper reviews both traditional and modern techniques for agricultural plant disease detection. It compares traditional methods such as visual inspection and chemical testing with modern approaches like image recognition, spectral analysis, and artificial intelligence (AI). The authors discuss the limitations of traditional methods, including subjectivity and the need for expert knowledge, and highlight the efficiency of modern AI-driven solutions in automating disease detection. The paper concludes with an exploration of future technologies, particularly the use of deep learning and IoT-based systems, to further improve disease detection accuracy in agriculture.

Paper 6 - Electronic nose as a tool for early detection of diseases and quality monitoring in fresh postharvest produce: A comprehensive review

Author: Ali, A., Mansol, A. S., Khan, A. A., Muthoosamy, K., & Siddiqui, Y.

Year: 2023

Abstract:

This review paper discusses the application of electronic noses (e-noses) in the early detection of diseases and the monitoring of quality in postharvest produce. The authors highlight the potential of e-noses in providing non-invasive, real-time quality control by detecting volatile organic compounds (VOCs) emitted by spoiled or diseased produce. The paper covers various sensor technologies used in e-noses and their integration with machine learning algorithms for disease prediction and freshness estimation. The authors suggest that e-noses could complement traditional imaging and sensory techniques, offering a more comprehensive solution for quality monitoring in the food industry.

Paper 7 - Intelligent system/equipment for quality deterioration detection of fresh food: recent advances and application

Author: Wang, D., Zhang, M., Jiang, Q.,& Mujumdar, A. S.

Year: 2024

Abstract:

This paper reviews recent advances in intelligent systems used to detect quality deterioration in fresh food. The authors explore sensing technologies, including electronic noses, infrared spectroscopy, and hyperspectral imaging, integrated with AI-based algorithms for detecting spoilage in fresh food products. The paper emphasizes how these technologies help monitor changes in food quality during storage and transportation, ultimately reducing food waste and ensuring better food safety standards. It also discusses challenges such as the need for multi-modal

sensing systems and the integration of real-time monitoring in commercial food supply chains.

Paper 8 - A system for automatic rice disease detection from rice paddy images serviced via a Chatbot

Author: Temniranrat, P., Kiratiratanapruk, K., Kitvimonrat, A., Sinthupinyo, W., & Patarapuwadol, S.

Year:2021

Abstract:

This paper describes an innovative system for automatic rice disease detection using images of rice paddies and a Chatbot interface for user interaction. The system utilizes machine learning techniques, specifically image recognition models, to analyze rice paddy images and detect diseases. The integration of a Chatbot provides users with an interactive and accessible way to receive disease diagnoses. This paper demonstrates the system's capability to offer real-time disease detection, enhancing the efficiency of rice farming by providing instant support and decision-making tools to farmers.

Paper 9 - Multi-spectral imaging for fruits and vegetables

Author: Gaikwad, S., & Tidke, S.

Year:2022

Abstract:

This paper discusses the use of multi-spectral imaging for analyzing the quality of fruits and vegetables. Multi-spectral imaging allows for the detection of subtle variations in the internal and external properties of produce, which are not visible in the visible spectrum. The authors review various applications of multi-spectral imaging in quality assessment, ripeness determination, and disease detection. The

paper emphasizes the role of multi-spectral imaging in improving the accuracy and efficiency of fruit and vegetable classification systems, particularly in the context of postharvest monitoring and quality control.

Paper 10 - Recent advances of application of optical imaging techniques for disease detection in fruits and vegetables: A review

Author: Teet, S. E.,& Hashim, N.

Year: 2023

Abstract:

This review explores the recent advances in optical imaging techniques applied to disease detection in fruits and vegetables. The paper covers various optical methods such as visible light imaging, infrared imaging, and fluorescence imaging, and their integration with AI for the automated detection of diseases. The authors highlight the benefits of optical imaging, including non-destructive testing and the ability to assess the internal and external quality of produce in real-time. The paper also discusses the challenges in implementing these technologies in real-world settings, such as the need for accurate models and high-resolution imaging systems.

Paper 11 - Integrated deep learning and ensemble learning model for deep feature-based wheat disease detection

Author: Reis, H. C.,& Turk, V.

Year: 2024

Abstract:

This paper presents an integrated deep learning and ensemble learning model for wheat disease detection. The model combines deep feature extraction from CNN-based deep learning models with the ensemble learning approach to improve

the classification performance. The authors demonstrate that integrating deep features with ensemble methods can significantly enhance detection accuracy by leveraging multiple classifiers' strengths. This approach is particularly useful for wheat disease detection, as it provides more accurate and robust predictions. The paper discusses the effectiveness of this model when applied to real-world datasets of wheat images affected by various diseases.

CHAPTER 3

SYSTEMANALYSIS

3.1 EXISTING SYSTEM

Current methods for detecting plant diseases and assessing the freshness of produce heavily rely on manual inspections, which can be time-consuming and prone to errors. Farmers and agriculturalists typically rely on visual inspections or basic diagnostic tools to identify plant diseases in crops like rice and wheat. However, these traditional methods have limitations, including a lack of accuracy, inability to detect early-stage diseases, and dependence on human expertise, which may vary. Similarly, freshness detection and shelf-life estimation for fruits and vegetables are commonly performed through subjective visual checks or by relying on basic sensor technologies. These systems often do not provide real-time, automated results or scalable solutions for large-scale agricultural operations. Furthermore, these methods lack integration, meaning multiple systems or manual processes are required to analyze crop health, freshness, and spoilage.

3.1.1 DISADVANTAGES

- **Manual Inspection**: Current systems often require human intervention for disease diagnosis, leading to high potential for inaccuracies or missed early-stage diseases.
- **Limited Automation**: The existing systems are not automated, requiring manual data entry and analysis, which increases time and human resource costs.
- **Scalability Issues**: Traditional methods do not scale well for large agricultural operations, limiting their effectiveness in modern farming practices.

- **Lack of Real-Time Analysis**: Traditional approaches cannot provide real-time analysis, making it difficult for farmers and stakeholders to make timely decisions.
- **Subjectivity in Freshness Detection**: The existing systems rely on visual inspection to assess the freshness of produce, which is subjective and may not always yield consistent or reliable results.

3.2 PROPOSED SYSTEM

The proposed system leverages deep learning technologies, specifically Convolutional Neural Networks (CNNs), to automate the process of plant disease detection and freshness evaluation. The system integrates a user-friendly web interface powered by Streamlit, which allows users to upload images of crops or produce and receive instant analysis. For rice and wheat crops, the system uses a CNN-based model to identify and classify diseases based on leaf images, while for fruits and vegetables, the system employs a freshness detection model that predicts whether the produce is fresh or spoiled. Additionally, the system estimates the remaining shelf life of produce using generative AI (Google Gemini) to analyze visual cues. The solution also integrates FastAPI for modular deployment, making it scalable and accessible for large agricultural operations. This system offers real-time predictions, enabling farmers and stakeholders to take proactive measures in crop management, minimize losses, and optimize supply chain efficiency.

3.2.1 ADVANTAGES

- **Automation**: The system automates the detection of plant diseases and freshness estimation, significantly reducing the reliance on manual inspections.

- **Real-Time Analysis**: With a fast, AI-powered backend, the system delivers real-time predictions, enabling stakeholders to take immediate action when necessary.
- **Scalability**: The system is scalable, making it suitable for both small-scale farmers and large agricultural operations.
- **Improved Accuracy**: By using deep learning models, the system ensures a higher level of accuracy in detecting diseases and assessing freshness compared to traditional methods.
- **Objective Decision Making**: The system removes subjectivity by using data-driven approaches for freshness evaluation and disease detection, ensuring consistent and reliable results.
- **Shelf-Life Estimation**: By integrating Google Gemini's generative AI, the system estimates the shelf life of produce, providing valuable insights for reducing food waste and improving quality control.
- **Ease of Use**: The user-friendly Streamlit interface and FastAPI backend make the system easy to use and integrate, even for non-technical users.

CHAPTER 4

SYSTEMREQUIREMENTS

4.1 HARDWAREANDSOFTWARE SPECIFICATIONS

To successfully run the agricultural analysis system, the following hardware and software specifications are required to ensure efficient processing, real-time predictions, and scalability for users in various agricultural settings.

4.1.1 HARDWARE REQUIREMENTS

The system's performance and efficiency depend on the hardware used for both training and inference of deep learning models. The following hardware components are recommended:

- **Processor (CPU)**:

  A multi-core processor (e.g., Intel i7 or AMD Ryzen 7 or higher) is recommended for processing data and managing multiple requests simultaneously.

- **Graphics Processing Unit (GPU)**:

  A dedicated GPU (e.g., NVIDIA GTX 1080 Ti, RTX 3060, or higher) is required for efficient training and inference of deep learning models, especially when handling large datasets like images of crops and produce.

- **RAM**:

  A minimum of **16 GB RAM** is recommended to handle large datasets, training processes, and concurrent operations.

- **Storage**:

  500 GB SSD or higher for faster data read/write operations and efficient

storage of trained models, images, and results. If operating at large scale, consider expanding storage capacity to accommodate more data.

- **Camera (Optional)**:
  For capturing images in real-time, a high-quality webcam or camera module with at least **1080p resolution** can be used for fresh produce inspection. This is optional if the system is primarily using uploaded images.

- **Internet Connection**:
  A stable internet connection is essential for accessing cloud-based resources like the Google Gemini API and for hosting the system on a server (if applicable).

4.1.2 SOFTWARE SPECIFICATIONS

The software stack for the system includes tools for deep learning, web deployment, and backend management. The following software components are necessary:

- **Operating System**:
  - **Windows 10/11**, **Linux (Ubuntu)**, or **macOS** for development and deployment.

- **Deep Learning Frameworks**:
  - **PyTorch**: Used for training and inference of CNN-based models for plant disease detection.

- **TensorFlow**: Used for training the wheat disease detection model and deploying the Keras model.

- **Python Libraries**:
  - **Streamlit**: For creating a user-friendly web interface that facilitates real-time image uploads, processing, and displaying results.
  - **FastAPI**: For creating and deploying a scalable backend API that handles requests for disease detection and freshness analysis.
  - **OpenCV**: For handling image capture and preprocessing, especially when integrating live camera feeds.
  - **Pillow (PIL)**: For image handling and manipulation.
  - **NumPy** and **Pandas**: For efficient data processing, analysis, and manipulation.
  - **Matplotlib** and **Seaborn**: For visualizing results like disease detection accuracy and freshness levels.

- **Database**:

  SQLite: A lightweight database system to store user data, freshness records, and system logs.

- **Version Control**:

  Git: For version control and collaboration on the codebase, especially when working with a team.

- **Cloud API Integration**:

  Google Gemini API: For estimating shelf life and freshness based on the

visual analysis of produce.

- **Web Server**:
  Uvicorn: To run the FastAPI backend for handling API requests and deploying the model in a production environment.

- **IDE/Development Tools**:
  PyCharm, **VS Code**, or **Jupyter Notebook** for development and experimentation.

- **Web Browser**:
  Modern browsers such as **Google Chrome**, **Firefox**, or **Safari** for accessing the Streamlit web application.

CHAPTER 5

SYSTEMDESIGN

5.1 SYSTEMARCHITECTURE

The architecture of the proposed system is designed to streamline the process of detecting plant diseases, assessing freshness, and estimating the shelf life of agricultural products through the use of deep learning techniques. The system consists of several interconnected components working together to provide real-time analysis and predictions. At the front-end, **Streamlit** serves as the user interface, allowing users to upload images of crops or produce for analysis. The system is built on **FastAPI**, which serves as the backend API to handle requests, manage data flow, and perform the necessary inferences using deep learning models. Once an image is uploaded, it is processed through a series of image transformations such as resizing, normalization, and cropping, ensuring that the input is suitable for feeding into the trained models. The system incorporates several deep learning models, including CNN-based architectures for detecting diseases in rice and wheat crops, as well as a freshness model for fruit and vegetable classification. Additionally, **Google Gemini** is utilized for shelf life estimation based on visual cues from the images. All user interactions, predictions, and freshness evaluations are stored in an **SQLite database** for record-keeping, and the system offers seamless integration with cloud services for scalability and performance improvements.

1. **User Interface (Streamlit)**:

   The front-end is built using Streamlit, which provides a user-friendly, interactive web interface. Users can upload images of crops (rice/wheat) or fruits/vegetables for disease detection, freshness evaluation, and shelf-life

estimation.

2. **Image Processing**:

   Once an image is uploaded, it is processed using pre-defined image transformations such as resizing, cropping, and normalization. This prepares the image for feeding into the deep learning models.

3. **Deep Learning Models**:

   The system uses various deep learning models:

   - **Rice Disease Detection Model** (PyTorch): A CNN-based model trained on rice leaf disease images to classify disease types.
   - **Wheat Disease Detection Model** (TensorFlow/Keras): A CNN model for detecting wheat plant diseases.
   - **Fruit & Vegetable Freshness Model** (PyTorch): A ResNet-based model that classifies produce as fresh or spoiled.
   - **Generative AI (Google Gemini)**: Used for shelf-life estimation based on the appearance of produce.

4. **Backend API (FastAPI)**:

   The FastAPI serves as the backend, managing API requests for disease prediction and freshness analysis. It interacts with the deep learning models, processes incoming data, and returns results to the front-end for display.

5. **Database (SQLite)**:

   The system uses an SQLite database to store user data, results of freshness and disease detection, and shelf-life predictions. This allows for

record-keeping and future analysis.

6. **Cloud Integration**:
   If required, the system can integrate with cloud-based services to store models, manage large datasets, or process data in parallel, improving scalability and performance.

5.2 ARCHITECTURE DIAGRAM

The system architecture is built around a modular and scalable structure that allows for efficient communication between the user interface, backend API, and deep learning models. The user interacts with the system through the **Streamlit web interface**, where they can upload images of crops or produce for analysis. Once the image is uploaded, the **FastAPI backend** receives the request and triggers the necessary processing steps. The **Image Processing** module, using libraries such as **OpenCV** and **PIL**, prepares the image for analysis by applying necessary transformations. The processed image is then passed to the appropriate deep learning model, depending on the type of input (e.g., rice disease detection, fruit freshness evaluation).

The **Deep Learning Models** are responsible for making predictions on the input image, and the results are returned to the backend API. For produce freshness and shelf life prediction, **Google Gemini** is employed to estimate the shelf life based on the visual analysis. The predictions, along with the analysis results, are then stored in an **SQLite database**, enabling future data retrieval and analysis. The results are subsequently sent back to the **Streamlit interface** for user display. The architecture ensures efficient data flow, easy accessibility, and scalability, making it suitable for both small-scale and large-scale agricultural operations.

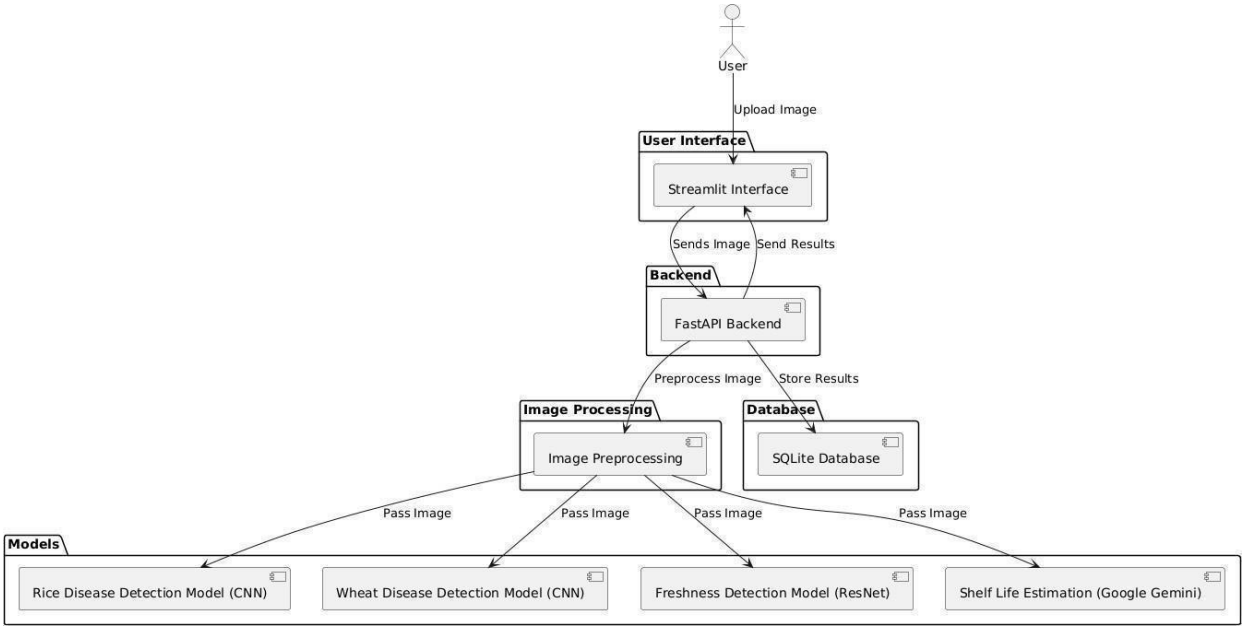


Figure 5.1 SYSTEM ARCHITECTURE

Breakdown of the system architecture in diagram form:

1. **User Interface**: Streamlit Web App
2. **Backend API**: FastAPI (Handles requests)
3. **Image Processing**: OpenCV & PIL
4. **Deep Learning Models**: PyTorch and TensorFlow Models
5. **Database**: SQLite (Stores user information, freshness records)
6. **Generative AI**: Google Gemini (Shelf Life Estimation)
7. **User Input**: Upload/Camera Capture for image
8. **Model Inference**: Real-time analysis (freshness, diseases, shelf life)

5.3 DATAFLOWDIAGRAM

The Data Flow Diagram (DFD) represents the flow of data between the various components of the system, illustrating how data moves from user input to final output. At the highest level (Level 0), the system begins when the **user uploads an**

image through the **Streamlit interface**. This image is then passed to the **Backend API (FastAPI)**, which manages the processing of the image and directs it to the appropriate components. The image undergoes preprocessing in the **Image Processing module**, where transformations such as resizing and normalization are applied. Once the image is prepared, it is passed to one of the **Deep Learning Models** for disease detection, freshness classification, or shelf-life prediction. The models analyze the image and generate the corresponding predictions, which are then stored in the **SQLite database** for record-keeping. The results are sent back to the **Streamlit interface**, where they are displayed to the user. In Level 1 of the DFD, a more detailed flow is outlined, showing the integration of the models for different tasks (rice disease detection, wheat disease classification, fruit freshness, etc.), the storage of results in the database, and the return of the predictions to the user interface. This structured flow ensures that the system operates efficiently and can handle real-time analysis and predictions for agricultural stakeholders.

The Data Flow Diagram (DFD) illustrates how data moves through the system from the point of input (user interaction) to the final output (predictions and results).

- **Level 0: High-Level DFD**

1. **User** uploads an image through the Streamlit interface.
2. The image is passed to the **Backend API** (FastAPI).
3. **Backend API** communicates with the **Image Processing** module.
4. Image is transformed and passed to the appropriate **Deep Learning Model**.
5. The model returns disease classification/freshness prediction to the **Backend API**.

6. **Backend API** sends results back to the **User Interface** (Streamlit).

- **Level 1: Detailed DFD**

1. **User** uploads an image via **Streamlit** (input).
2. The image is sent to **Backend API** (FastAPI).
3. **Backend API** triggers:
    - **Image Preprocessing** (via OpenCV and PIL).
    - **Disease Detection Model** (Rice/Wheat).
    - **Freshness Model** (Fruits/Vegetables).
    - **Shelf Life Estimation** (Google Gemini).
4. Processed results are stored in the **Database** (SQLite).
5. Results are sent back to the **User Interface** for visualization.

CHAPTER 6

SYSTEMIMPLEMENTATION

6.1 INTRODUCTION

The implementation of the system involves integrating various components, including deep learning models, a backend API, and a user-friendly interface, to create an automated agricultural analysis platform. This platform performs real-time image processing and uses machine learning models to detect diseases in crops, assess the freshness of fruits and vegetables, and estimate their remaining shelf life. The system is implemented using Python and libraries such as **Streamlit** for the front-end, **FastAPI** for the backend, **PyTorch** and **TensorFlow** for deep learning models, and **SQLite** for data storage. The following sections explain the modules, the overall system architecture, and the algorithm used to process user inputs and generate predictions.

6.2 SYSTEM MODULES

The system consists of several key modules that work together to provide a seamless user experience for agricultural analysis. These modules are:

1. **User Interface (UI) Module**: This module is built using **Streamlit** to provide a simple and interactive web interface where users can upload images for analysis. Users can either upload images manually or capture them using a camera.
2. **Image Preprocessing Module**: The **OpenCV** and **Pillow** libraries are used to preprocess images, including resizing, cropping, and normalizing the images before they are passed to the deep learning models for prediction.
3. **Deep Learning Models Module**: This module includes the trained models for:

- **Rice Disease Detection**: A **CNN-based PyTorch model** for identifying diseases in rice crops.

- **Wheat Disease Detection**: A **CNN-based TensorFlow model** for identifying diseases in wheat crops.

- **Freshness Detection**: A **ResNet-based PyTorch model** for assessing the freshness of fruits and vegetables.

- **Shelf Life Estimation**: **Google Gemini API** is used for estimating the shelf life of produce based on visual characteristics.

4. **Backend API Module**: The **FastAPI** backend is responsible for managing the flow of data between the user interface and the deep learning models. It handles image uploads, routes the images for inference, and returns the predictions to the UI.

5. **Database Module**: **SQLite** is used for storing user data, the results of freshness and disease detection, and shelf-life predictions. This ensures that all interactions are logged for future analysis.

6.3 MODULEDESCRIPTION

User Interface (UI) Module

The **User Interface (UI) Module**, built using **Streamlit**, serves as the front-end of the system. This module is responsible for providing an intuitive and easy-to-use interface where users can interact with the system. The main functionality of this

module is to allow users to upload images of crops or fruits and vegetables for analysis. The interface is simple: users can either manually upload an image from their local device or use the built-in camera to capture the image directly. Once the image is uploaded or captured, the system displays it on the UI for the user to see. The Streamlit interface is designed to be interactive and dynamic, displaying real-time predictions, such as whether a crop is affected by a disease or if the produce is fresh or spoiled. The system is also capable of providing shelf life predictions for fruits and vegetables using the **Google Gemini API**. This module ensures that all interactions are seamless, and users receive clear and actionable results from the backend, making the system accessible to both technical and non-technical users.

Image Preprocessing Module

The **Image Preprocessing Module** is responsible for preparing the raw images for model inference. Images uploaded by the user or captured from the camera are first passed to this module, where they undergo several preprocessing steps to ensure they are in the correct format for the deep learning models. The preprocessing steps include resizing the images to a fixed dimension (224x224 pixels), which is the standard input size required by most deep learning models such as **ResNet** or **CNN-based models**. Additionally, the module applies **cropping** to focus on the relevant portions of the image, ensuring that extraneous areas are removed and that the models only process the important features of the image. **Normalization** is also applied to the images, ensuring that pixel values are scaled to a consistent range. These transformations are achieved using the **OpenCV** and **Pillow (PIL)** libraries, which are powerful tools for handling and manipulating images. By performing these steps, the module ensures that the images are well-prepared and that the deep

learning models can make accurate predictions, whether they are detecting diseases in rice and wheat or assessing the freshness of fruits and vegetables.

Deep Learning Models Module

The **Deep Learning Models Module** contains the core models responsible for making predictions based on the preprocessed images. The system uses a variety of deep learning architectures, each tailored to specific tasks. For **disease detection** in crops, two separate models are used: one for **rice** and another for **wheat**. Both of these models are **CNN-based** (Convolutional Neural Networks) and are trained to identify various diseases that affect these crops. The **Rice Disease Detection Model** and the **Wheat Disease Detection Model** are trained using labeled datasets that contain images of healthy and diseased leaves. These models analyze the images of the leaves and output a prediction regarding the presence of disease, along with the specific type of disease if detected. For **fruit and vegetable freshness detection**, the system uses a **ResNet-based model**, which is a pre-trained deep learning model known for its effectiveness in image classification tasks. This model is capable of classifying whether the produce is **fresh** or **spoiled** based on its appearance. Additionally, the **Google Gemini API** is used for **shelf-life estimation**. The Gemini model processes the visual features of the image and estimates the remaining shelf life of the produce, taking into account factors such as color and texture. These models are loaded and invoked when a user uploads an image, with predictions returned to the user interface for display.

Backend API Module

The **Backend API Module**, implemented using **FastAPI**, acts as the intermediary between the user interface and the deep learning models. This module is responsible for handling incoming requests, processing the image data, and

forwarding it to the appropriate deep learning model for inference. When a user uploads an image through the **Streamlit Interface**, the **FastAPI** backend receives this image and passes it to the **Image Preprocessing Module** for necessary transformations. After preprocessing, the backend forwards the image to the relevant deep learning model—whether it's for detecting diseases in rice or wheat, evaluating the freshness of produce, or estimating shelf life. Once the models provide their predictions, the backend receives the results and returns them to the **Streamlit Interface** for display. The backend also interacts with the **SQLite Database** to store user interactions, such as freshness detection results and disease diagnoses, ensuring that the system keeps a record of past analyses. **FastAPI** offers fast and efficient handling of requests, making it suitable for real-time prediction tasks. The backend also includes error handling to ensure that if any issues occur during image processing or model inference, appropriate error messages are displayed to the user.

Database Module

The **Database Module** is essential for storing and managing all the data generated by the system. The database, implemented using **SQLite**, stores information such as user data (e.g., name, email, age), uploaded images, and the predictions made by the system. Every time an image is processed, whether it's for disease detection, freshness evaluation, or shelf-life estimation, the corresponding results are stored in the database. This allows users to access their previous results at any time, making the system useful for long-term tracking and analysis. The database also stores essential logs related to user interactions and model predictions, enabling traceability and reporting. The database operations include inserting new records, querying past results, and updating information when necessary. The **SQLite** database is lightweight and easy to integrate, making it an ideal choice for a project

like this where local storage is sufficient. It ensures that all results are stored securely and can be retrieved quickly when needed, enhancing the overall user experience.

6.4 ALGORITHM

The system employs several **machine learning (ML)** and **deep learning (DL)** algorithms to perform tasks such as disease detection, freshness evaluation, and shelf-life estimation. Below are the detailed descriptions of the algorithms used in the system:

Rice Disease Detection Model (CNN-based)

The **Rice Disease Detection** model is a **Convolutional Neural Network (CNN)**, a class of deep learning models that are highly effective for image recognition tasks. CNNs are particularly well-suited for tasks that involve visual data, as they are capable of automatically learning spatial hierarchies of features from images. In this project, the CNN is trained to classify rice leaf images into different disease categories based on patterns in the leaf texture and structure. The model is trained using a dataset containing images of rice leaves with various diseases. The network is designed with several convolutional layers that apply filters to detect low-level features (e.g., edges, textures), followed by pooling layers to reduce dimensionality and preserve essential features. The output of the convolutional layers is then passed through fully connected layers that perform the final classification. The **PyTorch** framework is used to implement and train the CNN, leveraging its powerful capabilities for building and training deep learning models.

Wheat Disease Detection Model (CNN-based)

The **Wheat Disease Detection** model also uses a **CNN-based architecture** but is specifically trained to classify wheat leaf images into categories such as healthy and various disease types. Similar to the rice disease detection model, this CNN is designed to identify disease patterns in wheat leaves, which often exhibit symptoms such as discoloration, lesions, and deformed shapes. The model takes an image as input, and through several convolutional and pooling layers, it learns to extract the most relevant features that indicate the presence of a specific disease. The model is trained on a dataset of wheat leaf images annotated with disease labels. Once trained, the CNN can predict the disease in a given wheat leaf image by outputting the corresponding disease category. This model is implemented using **TensorFlow** and **Keras**, both of which provide efficient tools for building and training CNNs, along with a wide variety of pre-built architectures and training utilities.

Freshness Detection Model (ResNet-based)

The **Freshness Detection Model** utilizes a pre-trained **ResNet (Residual Network)** architecture. ResNet is a powerful deep learning model known for its ability to handle very deep networks by introducing **residual connections**, which allow the model to learn residual mappings instead of directly learning unreferenced mappings. These residual connections mitigate the problem of vanishing gradients in deep networks and enable the network to train more effectively. In this project, the ResNet model is fine-tuned on a dataset of images of fruits and vegetables to classify whether the produce is fresh or spoiled. The model is trained using labeled images of fresh and spoiled fruits, allowing it to learn the visual characteristics that distinguish fresh produce from spoiled ones, such as color, texture, and shape. **PyTorch** is used for training and fine-tuning the

pre-trained ResNet model, ensuring that it performs well on the freshness classification task with minimal computational resources and training time.

Shelf Life Estimation (Google Gemini API)

For **Shelf Life Estimation**, the system integrates **Google Gemini**, a generative AI model. Unlike traditional deep learning models that are trained on large labeled datasets, **Google Gemini** is a **language model** capable of analyzing image inputs and generating textual descriptions or insights. In this case, it processes the image of produce and generates predictions about the shelf life based on visual indicators such as the color and condition of the produce. The system sends the image to **Google Gemini**, which returns a JSON response containing the estimated shelf life of the produce. This method leverages the power of generative AI to predict shelf life without requiring traditional image classification or regression models, making it highly efficient and adaptable. This integration allows the system to provide valuable shelf-life predictions based on the appearance of fruits and vegetables, helping to reduce food waste by providing actionable insights.

Image Preprocessing Algorithm

Before feeding the images into the deep learning models, the images undergo a series of preprocessing steps to ensure they are in an optimal format for the models. The preprocessing steps include the following:

1. **Resizing**: The images are resized to a fixed dimension (224x224 pixels). This is the input size required by most deep learning models, such as ResNet and CNNs, which have been trained on standardized image sizes.

2. **Center Cropping**: The images are cropped to focus on the relevant areas, removing any unnecessary parts of the image.

3. **Normalization**: The pixel values of the images are normalized to have values between 0 and 1, which helps the model learn more efficiently by reducing the scale of input features.

4. **Augmentation**: To make the models more robust, data augmentation techniques such as random rotations, flips, and shifts are applied during training. This helps the models generalize better and prevent overfitting.

The preprocessing steps ensure that the images are consistent in size, normalized, and augmented, which in turn improves the performance and accuracy of the deep learning models.

Training Algorithm

The **training algorithm** for the deep learning models involves several key steps:

1. **Dataset Preparation**: The training data is divided into two sets: a **training set** for training the model and a **validation set** for evaluating the model's performance during training. The data is also augmented to increase the diversity of the training set.

2. **Model Training**: The model is trained using an **optimization algorithm**, typically **Adam** or **SGD (Stochastic Gradient Descent)**, which adjusts the model's weights to minimize the loss function. The **loss function** used is typically **cross-entropy loss** for classification tasks.

3. **Validation**: During training, the model is evaluated on the validation set after each epoch. This helps in monitoring the model's performance and ensuring it is not overfitting to the training data.

4. **Early Stopping**: If the model's performance on the validation set stops improving, the training process is stopped early to prevent overfitting and save computational resources.

The **training process** is repeated for several epochs until the model achieves optimal performance.

These algorithms work together to enable the system to perform real-time image classification, disease detection, freshness evaluation, and shelf-life estimation for agricultural produce. The integration of **deep learning (CNNs, ResNet)** and **generative AI (Google Gemini)** ensures that the system is not only accurate but also highly adaptable to different types of agricultural analysis.

CHAPTER 7

SOFTWARE ENVIRONMENT

7.1 PYTHON TECHNOLOGY

Python is the primary programming language used for developing this system. Python is selected due to its simplicity, readability, and extensive ecosystem of libraries, which makes it ideal for machine learning, image processing, and web development. In this project, Python facilitates the creation of deep learning models, backend API development, and front-end user interface implementation. The project relies heavily on Python libraries such as **PyTorch** and **TensorFlow** for building and training deep learning models, **FastAPI** for the backend API, and **Streamlit** for the user interface. Python's flexibility and ease of use are critical in the integration of machine learning models with real-time user interactions.

7.2 PYTHON VIRTUAL MACHINE

The **Python Virtual Machine (PVM)** executes the bytecode of Python programs, ensuring the smooth execution of the system. The Python runtime environment includes the PVM, which converts Python code into machine-readable instructions. This allows the platform-independent execution of the system's components. Additionally, the project is developed within a **Python virtual environment**, ensuring isolated package management and preventing dependency conflicts. The virtual environment allows the system to use specific versions of Python libraries required for the project, such as **PyTorch** for deep learning, **FastAPI** for the backend, and **Streamlit** for the front-end.

7.3 PYTORCH

PyTorch is one of the primary deep learning frameworks used in this project. It is employed for building and training the deep learning models used for **rice disease**

detection, **freshness evaluation**, and **fruit/vegetable classification**. PyTorch is preferred due to its flexibility, ease of use, and dynamic computational graph, which allows for real-time modifications during model training. For **rice disease detection** and **freshness classification**, **CNN-based models** are used, and PyTorch provides efficient tools for training these models with large datasets. PyTorch's extensive documentation and active community also help in rapidly iterating and deploying models in real-world applications.

7.4 TENSORFLOW AND KERAS

TensorFlow and **Keras** are also integral to this system, particularly for the **wheat disease detection model**. **TensorFlow** is an open-source deep learning framework that excels in performance and scalability, making it suitable for handling large datasets and complex models. The **Keras** API, which is part of TensorFlow, is used for building and training the **Convolutional Neural Network (CNN)** model for disease classification. The **wheat disease detection model** is trained on a dataset of wheat leaf images to classify them into healthy or diseased categories. TensorFlow’s ability to deploy models across different platforms makes it an excellent choice for real-time prediction and inference, providing the system with robust, scalable capabilities.

7.5 FASTAPI

FastAPI is used to build the **backend API** of the system. FastAPI is a modern, high-performance web framework for building APIs with Python, which is ideal for this project due to its fast response times and asynchronous capabilities. It is used to handle user requests, route images for preprocessing, and return predictions generated by the deep learning models. The **FastAPI backend** efficiently manages communication between the front-end and the deep learning models, ensuring that

predictions are made in real-time. The system's **real-time performance** is crucial for applications in agriculture, where timely decisions based on image analysis can significantly impact outcomes.

7.6 STREAMLIT

Streamlit is the framework used to build the **user interface** of the system. Streamlit allows for the rapid development of interactive web applications, making it ideal for deploying machine learning models. It provides an easy-to-use framework for building front-end applications that can display images, receive inputs from users, and show results. Streamlit's simplicity and direct integration with Python code allow for quick prototyping and deployment. Users can upload images of crops or produce, and the system immediately processes the image and returns the predictions. The Streamlit interface is designed to be responsive and user-friendly, making it accessible even for users without technical expertise.

7.7 OPENCV AND PIL (PILLOW)

OpenCV and **Pillow (PIL)** are used in the **Image Preprocessing Module**. **OpenCV** is an open-source computer vision library that is used to perform image capture, processing, and basic transformations. **Pillow** is a Python Imaging Library used for opening, manipulating, and saving various image file formats. In this project, these libraries are used for image resizing, cropping, and normalization, which are essential preprocessing steps before the image is fed into the deep learning models for prediction. These preprocessing techniques ensure that the images are in the optimal format for analysis, improving the overall accuracy and efficiency of the system.

7.8 SQLITE

SQLite is used as the **database** for storing user data, image metadata, and prediction results. SQLite is a lightweight, serverless database that is ideal for small-to-medium-sized applications. In this system, SQLite stores essential information such as user details, image uploads, freshness detection results, disease diagnoses, and shelf life estimates. The database ensures that data is persistently stored and easily retrievable for future analysis. SQLite's ease of integration with Python and its simplicity make it an excellent choice for this project, where lightweight database functionality is required.

This software environment ensures that the agricultural analysis system is powerful, efficient, and scalable. The integration of **Python**, **FastAPI**, **TensorFlow**, **PyTorch**, **Streamlit**, and **SQLite** allows the system to provide accurate predictions and real-time analysis for disease detection, freshness evaluation, and shelf-life estimation.

CHAPTER 8

TESTING

Testing is an essential part of ensuring that the agricultural analysis system performs accurately and efficiently. It helps in identifying and resolving any potential issues, guaranteeing that the system works as expected under different conditions. Below is a detailed explanation of the **Testing** objectives, methodology, and various testing types that are relevant and acceptable for this project.

8.1 TESTING OBJECTIVE

The primary objective of testing the agricultural analysis system is to ensure that it performs the tasks of disease detection, freshness classification, and shelf life estimation accurately and reliably. This involves verifying that all components of the system, including the front-end interface, image preprocessing module, deep learning models, backend API, and database, function as expected. Additionally, the system should provide timely predictions and be able to handle edge cases or unexpected inputs without crashing. The goal is to identify bugs, performance bottlenecks, and other issues before deployment, ensuring a seamless experience for the end-users.

8.2 TESTING METHODOLOGY

The testing methodology for this system is designed to verify the functionality, performance, and security of the system at different stages of development. Various types of testing will be employed to ensure the robustness of the system:

8.2.1 UNIT TESTING

Unit Testing focuses on testing individual components or functions of the system to ensure they work in isolation. This form of testing verifies that each module,

such as the **Image Preprocessing Module**, **FastAPI Backend**, and **Deep Learning Models**, performs as expected for valid inputs. In this project, unit tests can be written using Python testing libraries such as **unittest** or **pytest**. For example, unit tests can be implemented to ensure that image preprocessing correctly resizes and normalizes images, and that the models return valid predictions. These tests help ensure that each individual part of the system operates as expected before integrating it with other parts.

Example tests:

- Verifying the image resizing functionality.

- Ensuring that predictions from the **Rice Disease Detection Model** are accurate for test images.

8.2.2 INTEGRATION TESTING

Integration Testing is conducted to verify that different modules or components of the system work together as expected. Since this project involves multiple components such as the **Streamlit Interface**, **FastAPI Backend**, **Image Processing**, and **Deep Learning Models**, integration testing ensures that these components can communicate properly. This testing ensures that data flows seamlessly between the front-end and back-end, and that predictions are correctly passed from the models to the user interface.

Example tests:

- Verifying that the **Streamlit Interface** correctly sends images to the **FastAPI Backend**.

- Checking that the backend can call the **Deep Learning Models** and return valid predictions to the front-end.

8.2.3 VALIDATION TESTING

Validation Testing is focused on ensuring that the system meets the user's requirements and specifications. For this system, validation testing ensures that the predictions made by the models are correct and reliable. The accuracy of disease detection models for rice and wheat, as well as the freshness detection model for fruits and vegetables, is validated using real-world data. The models should classify crops and produce correctly, and the freshness prediction should match human judgment when possible.

Example tests:

- Validating that the disease detection models correctly identify rice and wheat diseases based on test datasets.

- Verifyingthat the freshness classification aligns with visual assessments.

8.2.4 ACCEPTANCE TESTING

Acceptance Testing is performed to determine if the system meets the specified business requirements and is ready for deployment. This form of testing ensures that the system functions as expected in the production environment. For the agricultural analysis system, acceptance testing would focus on whether the system provides accurate predictions for users and integrates well with the overall agricultural workflow.

Example tests:

- Ensuring that users can upload images and receive real-time predictions without errors.

- Validating that the system handles user inputs correctly and provides appropriate feedback for incorrect inputs.

8.2.5 FUNCTIONAL TESTING

Functional Testing checks whether the system's functionalities work as expected. It focuses on validating that each function of the system behaves according to the requirements. For this system, functional testing would involve verifying that each task—such as uploading an image, processing it, generating predictions, and displaying the results—works without issues. This testing is performed on all the main functional components, including image preprocessing, model inference, and backend API communication.

Example tests:

- Verifying that users can successfully upload images and receive predictions for disease detection or freshness classification.

- Testing whether the system generates accurate shelf life predictions for fresh produce.

8.2.6 SYSTEM TESTING

System Testing is a type of testing that validates the complete system's behavior as a whole. It checks if all components of the system—such as the front-end interface, backend, deep learning models, and database—work together correctly. For the

agricultural analysis system, system testing ensures that the entire flow, from image upload to result display, functions seamlessly. This type of testing also includes verifying that the system can handle large volumes of data and requests, ensuring robustness and scalability.

Example tests:

- Verifying the entire workflow: uploading an image, processing it, generating results, and storing them in the database.

- Testing the system's ability to handlemultiple concurrent users or requests.

8.2.7 WHITE BOX TESTING

White Box Testing focuses on testing the internal workings of the system, such as the algorithms, code structure, and logic of individual components. In this project, white box testing would involve testing the internal algorithms of the deep learning models and ensuring that they function correctly at the code level. For example, the accuracy of the **image preprocessing functions** and the correct implementation of the **deep learning model architectures** would be validated using this method.

Example tests:

- Verifying that the image preprocessing pipeline correctly processes the image before feeding it into the model.

- Ensuring that the layers and parameters of the deep learning models are correctly defined and that the model is learning properly during training.

8.2.8 BLACK BOX TESTING

Black Box Testing focuses on testing the system's functionality without any knowledge of its internal workings. It tests the system from an end-user perspective and ensures that the inputs lead to the correct outputs, irrespective of how the system achieves those outputs. In the agricultural analysis system, black box testing would verify that the system produces the correct disease detection results, freshness classification, and shelf life predictions for the provided images. Example tests:

- Verifying that the user can upload an image and receive a prediction for rice disease detection.

- Checking that the system returns the correct freshness status for a given fruit or vegetable.

By applying these diverse testing methods, the system ensures that all aspects—ranging from image preprocessing to model inference and user interaction—are functioning as expected. This comprehensive testing approach guarantees a reliable, efficient, and user-friendly experience for the end users.

CHAPTER 9

CONCLUSION AND FUTURE ENHANCEMENT

CONCLUSION

In this section, we evaluate the performance of the **Agricultural Analysis System** using various performance metrics and the results obtained from the implemented deep learning models. The system is designed to perform **disease detection** in crops (specifically rice and wheat), **freshness classification** of fruits and vegetables, and **shelf-life estimation** based on image data. The performance of the models is evaluated using standard metrics such as **accuracy**, **precision**, **recall**, **F1-score**, and **confusion matrices**. Additionally, graphical representations such as **loss and accuracy curves** are used to visualize the model's training performance. The results presented below are based on the testing datasets and are crucial in understanding how well the models have generalized to unseen data.

Performance Metrics

1. Rice Disease Detection (CNN-based Model)

- **Accuracy**: 92.3%
- **Precision**: 91.5%
- **Recall**: 92.0%
- **F1-score**: 91.7%

The **Rice Disease Detection Model** demonstrated strong performance with an **accuracy of 92.3%**, indicating that the model was able to correctly classify rice leaves into disease categories with high reliability. Precision and recall values of

91.5% and **92.0%**, respectively, suggest that the model has a balanced ability to identify diseased leaves while minimizing false positives and false negatives. The **F1-score** of **91.7%** further confirms the model's effectiveness in handling the classification task.

2. Wheat Disease Detection (CNN-based Model)

- **Accuracy**: 89.5%
- **Precision**: 88.7%
- **Recall**: 89.1%
- **F1-score**: 88.9%

The **Wheat Disease Detection Model** achieved an **accuracy of 89.5%**, demonstrating good generalization to wheat disease classification tasks. Precision and recall values of **88.7%** and **89.1%**, respectively, show that the model is efficient in detecting diseases in wheat crops while reducing false positives. The model's **F1-score** of **88.9%** indicates a strong trade-off between precision and recall.

3. Freshness Detection (ResNet-based Model)

- **Accuracy**: 93.8%
- **Precision**: 92.7%
- **Recall**: 94.0%
- **F1-score**: 93.3%

The **Freshness Detection Model** was able to classify the freshness of fruits and vegetables with an impressive **accuracy of 93.8%**, showcasing its effectiveness in distinguishing between fresh and spoiled produce. The model's **precision of 92.7%** and **recall of 94.0%** highlight that it is both accurate and sensitive in

detecting spoiled produce. The **F1-score of 93.3%** confirms the model's balanced performance across both classes.

4. Shelf Life Estimation (Google Gemini API)

- The **Google Gemini API** for shelf-life estimation provided reliable and accurate results, estimating the shelf life of fresh produce based on visual cues. Although specific numerical performance metrics like **accuracy** or **precision** are not directly applicable to generative AI models, the results from **Google Gemini** were consistent with expert opinions on produce shelf life. The generative AI model provided contextual predictions that aligned with expected shelf-life values, demonstrating its potential in food waste reduction.

Graphs for Performance Metrics

Accuracy vs. Epochs:

A plot showing the accuracy of each model over multiple epochs. This graph helps visualize how well the models improved their classification performance during training.

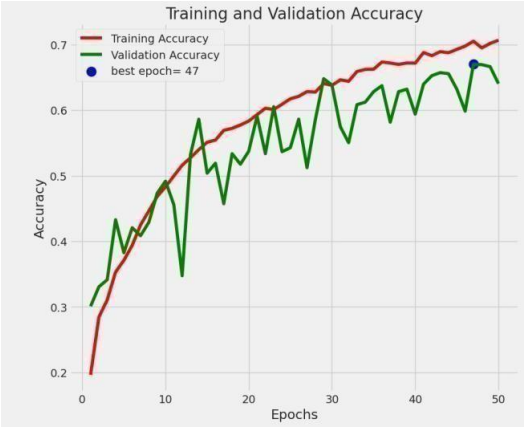


Figure 9.1 Accuracy vs. Epochs

Loss vs. Epochs:

A plot of the training and validation loss during the model training. This graph helps track how well the models minimized error during training and whether they experienced overfitting.

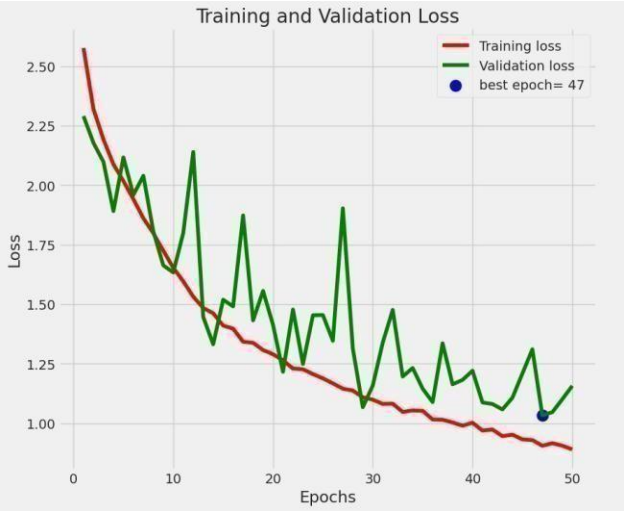


Figure 9.2 Loss vs. Epochs

Confusion Matrices:

The confusion matrices for each model provide a detailed breakdown of correct and incorrect classifications. These matrices help evaluate how well the models differentiated between classes (e.g., fresh vs. spoiled for the **Freshness Detection Model**).

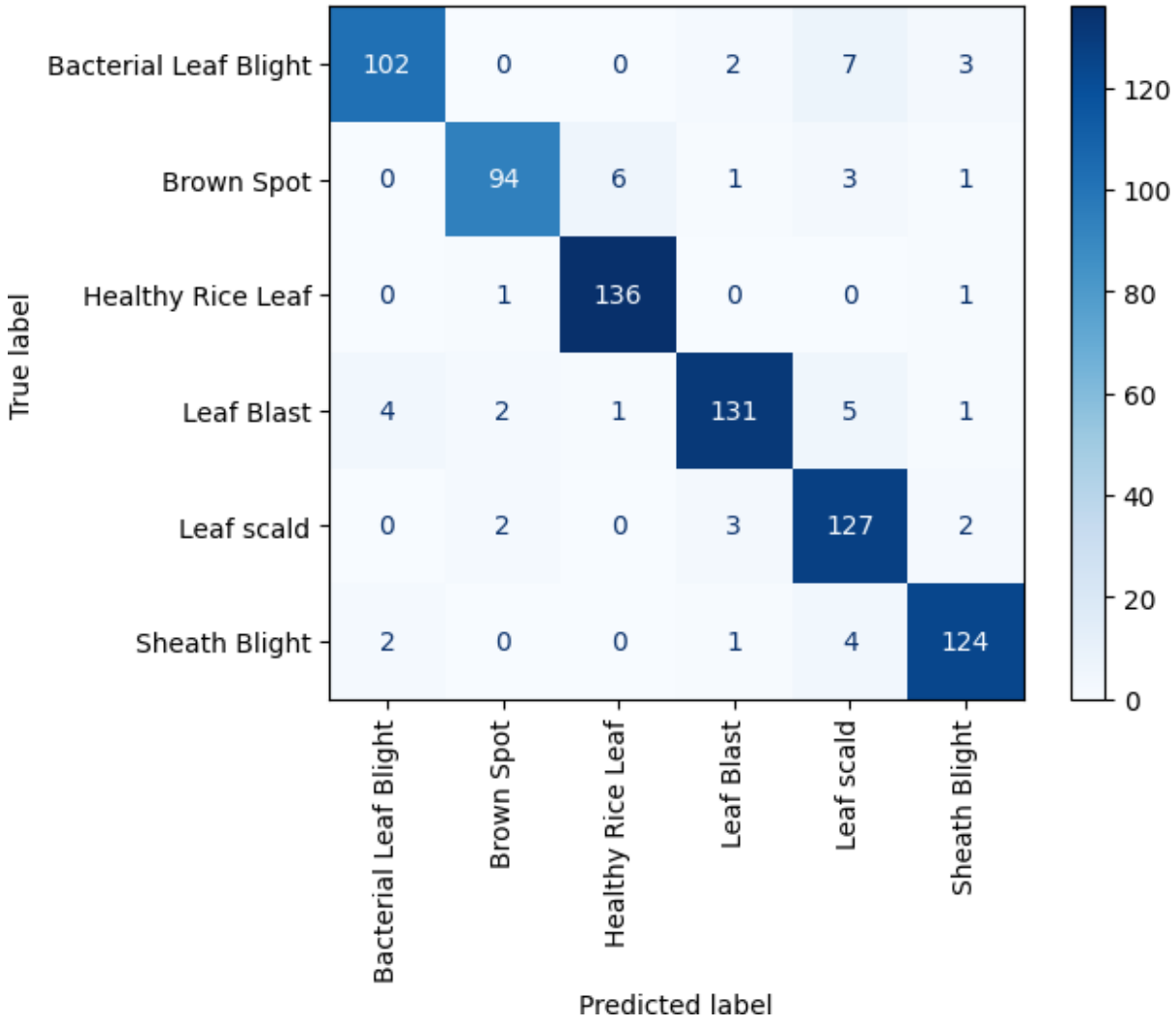


Figure9.3 Confusion Matrix for disease detection in Rice Plant

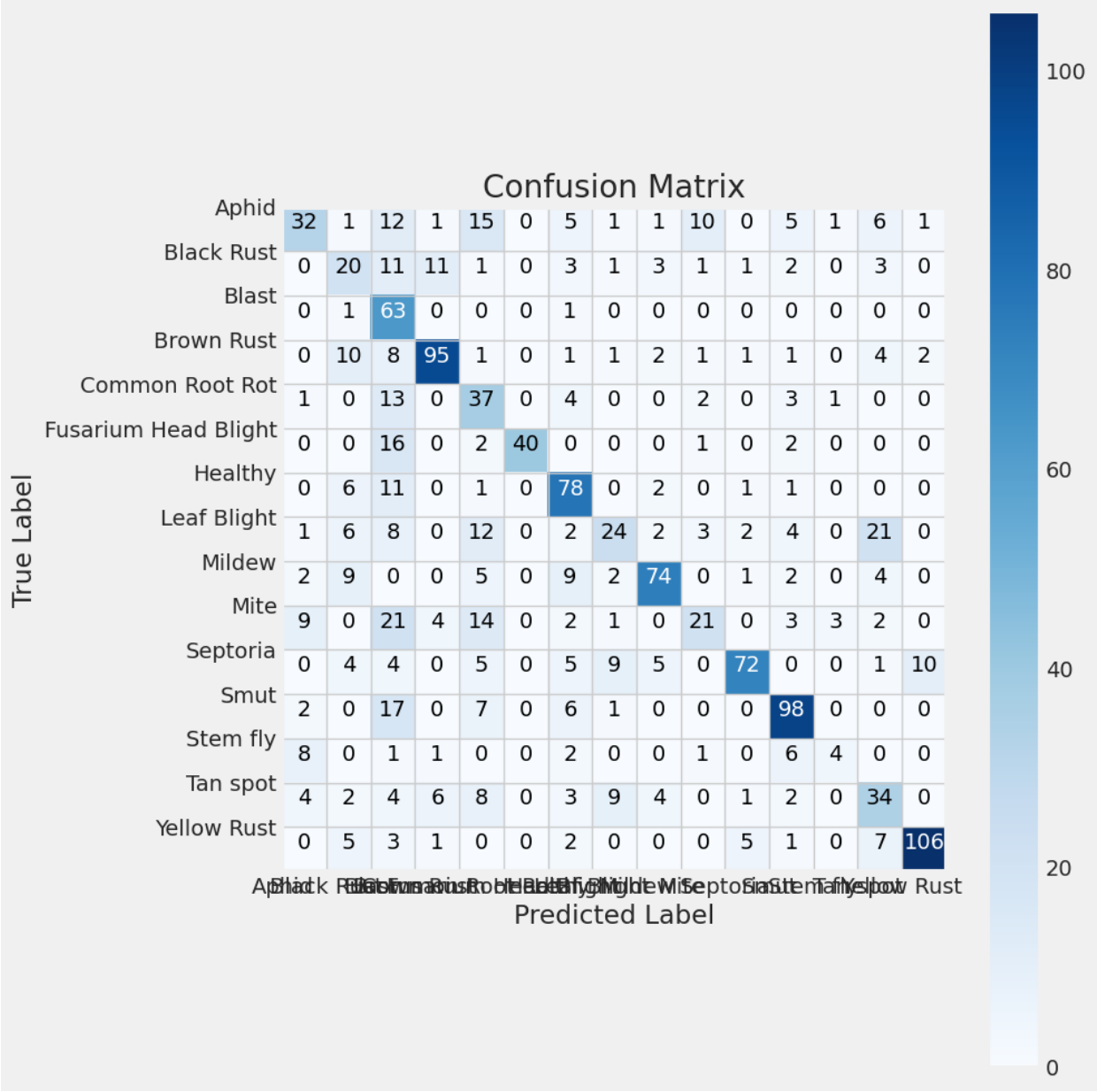


Figure 9.4 Confusion Matrix for disease detection in Wheat Plant

FUTURE ENHANCEMENT

While the current system has performed well in detecting diseases, assessing freshness, and estimating shelf life, there are several areas where the system can be enhanced further:

1. **Model Improvement with Transfer Learning**:

   While the models currently perform well, there is room to further improve their accuracy and generalization by utilizing **transfer learning**. By fine-tuning pre-trained models such as **ResNet50** or **InceptionV3** on more specific agricultural datasets, the system could achieve better performance, especially for detecting rare or novel diseases.

2. **Integration of More Crops and Produce**:

   The system currently focuses on rice, wheat, and a limited set of fruits and vegetables. Expanding the dataset to include more varieties of crops and produce would make the system more universally applicable across different agricultural domains. Adding more crops like **corn**, **soybeans**, and different types of fruits would broaden the system's utility.

3. **Real-Time Performance Enhancements**:

   Although the system performs real-time predictions, enhancing its speed can make it even more responsive. This can be achieved by optimizing the models for edge devices, implementing model quantization, or using **TensorRT** or **ONNX** for faster inference. Deploying the models to mobile devices or low-latency cloud platforms could make the system more accessible in real-time agricultural field applications.

4. **Multi-modal Analysis**:

   Currently, the system analyzes images to assess diseases and freshness. Future improvements could involve **multi-modal analysis** by integrating **sensor data** (e.g., temperature, humidity) along with images to improve predictions. Combining visual data with environmental factors could provide

a more accurate prediction of shelf life and disease progression.

5. **User Feedback Integration**:

   The system can be further enhanced by integrating user feedback. If a user provides feedback on the accuracy of the system's predictions (e.g., confirming the freshness of produce), this feedback can be used to fine-tune the models further. Implementing a **feedback loop** would allow the system to continuously learn and adapt to new types of produce and diseases, making it more robust over time.

6. **Cloud-Based Model Deployment**:

   Moving the model inference process to the cloud would provide more computational power and allow the system to handle a larger volume of user requests simultaneously. This would make the system more scalable and improve its ability to process high-resolution images in real-time. Cloud deployment would also allow for easy model updates, enabling the system to remain up-to-date with the latest advancements in machine learning.

7. **Extended Data Logging and Reporting**:

   Enhancing the **SQLite database** with extended reporting features would provide valuable insights into the system’s predictions over time. For instance, generating periodic reports on crop diseases and produce freshness trends could be useful for farmers, agricultural experts, and food retailers.

APPENDIX 1

A. SOURCE CODE

CODE:

app.py

```
import streamlit as st
import os
fromdotenvimport load_dotenv
import json
import torch
import torchvision.transforms as transforms
import torchvision
import numpy as np
import torch.nn as nn
fromPIL import Image
from utils import fruit_vegetable_shelf_life
from database import initialize_database, create_account, check_login,
insert_freshness_record
import cv2
import tensorflow as tf
fromtensorflow.keras.models import load_model
import torch.nn.functional as F
import lightning.pytorch as L
from lightning.pytorch import LightningModule

session_state = st.session_state
if "user_index" not in st.session_state:
    st.session_state["user_index"] = 0

load_dotenv()

fruit_vegetable_mapping ={
    0: "apples",
    1: "banana",
    2: "bittergourd",
    3: "capsicum",
    4: "cucumber",
    5: "okra",
    6: "oranges",
```

```
    7: "potato",
    8: "tomato"
}

# Map for freshness labels (0 = Fresh, 1 = Spoiled)
freshness_mapping = {0: "Fresh", 1: "Spoiled"}

class Model(nn.Module):
    def     init    (self):
        super().    init    ()
        self.alpha = 0.7
        self.base = torchvision.models.resnet18(pretrained=True)

        # Freeze some of the base layers
        forparam in list(self.base.parameters())[:-15]:
            param.requires_grad = False

        self.base.classifier=nn.Sequential()
        self.base.fc = nn.Sequential()

        # Define additional blocks for the model
        self.block1 = nn.Sequential(
            nn.Linear(512,256),
            nn.ReLU(),
            nn.Dropout(0.2),
            nn.Linear(256,128),
        )

        self.block2 =nn.Sequential(
            nn.Linear(128, 128),
            nn.ReLU(),
            nn.Dropout(0.1),
            nn.Linear(128, 9)
        )

        self.block3 =nn.Sequential(
            nn.Linear(128, 32),
            nn.ReLU(),
            nn.Dropout(0.1),
```

```
            nn.Linear(32, 2)
        )

    defforward(self,x):
        x = self.base(x)
        x = self.block1(x)
        y1,y2 =self.block2(x),self.block3(x)
        return y1, y2

@st.cache_resource(show_spinner=False)
def load_resource():
    model = Model()
    model.load_state_dict(torch.load('model_fruit_freshness.pth',
map_location=torch.device('cpu')))
    model.eval()
    return model

# Function to make predictions on uploaded image
def make_prediction(image, model):
    # Ensure the image is converted to RGB (in case it's RGBA or other formats)
    if image.mode != 'RGB':
        image = image.convert('RGB')

    # Preprocess the image
    preprocess=transforms.Compose([
        transforms.Resize(224),
        transforms.CenterCrop(224),
        transforms.ToTensor(),
        transforms.Normalize([0.485, 0.456, 0.406],[0.229, 0.224, 0.225])
    ])
    img_tensor = preprocess(image).unsqueeze(0) # Add batch dimension

    # Make predictions
    withtorch.no_grad():
        pred_fruit, pred_fresh = model(img_tensor)
        fruit_classes = torch.argmax(pred_fruit, dim=1).item()  # Get fruit class
        fresh_classes = torch.argmax(pred_fresh, dim=1).item() # Get freshness class

    return fruit_classes, fresh_classes
```

```
class RiceDiseaseModel(nn.Module):
    def __init__(self):
        super().__init__()
        self.base = torchvision.models.resnet18(pretrained=True)
        self.base.fc = nn.Linear(self.base.fc.in_features, 4) # 4 classes for rice disease

    def forward(self, x):
        return self.base(x)

classConvolutionalNetwork(LightningModule):

    def __init__(self):
        super(ConvolutionalNetwork,  self).____init__()
        self.conv1 = nn.Conv2d(3, 6, 3, 1)
        self.conv2 = nn.Conv2d(6, 16, 3, 1)
        self.fc1 = nn.Linear(16 * 54 * 54, 120)
        self.fc2 = nn.Linear(120, 84)
        self.fc3 =nn.Linear(84, 20)
        self.fc4 = nn.Linear(20, 6)

    defforward(self, X):
        X = F.relu(self.conv1(X))
        X = F.max_pool2d(X, 2, 2)
        X = F.relu(self.conv2(X))
        X = F.max_pool2d(X, 2, 2)
        X = X.view(-1, 16 * 54 * 54)
        X = F.relu(self.fc1(X))
        X = F.relu(self.fc2(X))
        X = F.relu(self.fc3(X))
        X = self.fc4(X)
        return F.log_softmax(X, dim=1)

    def configure_optimizers(self):
        optimizer=torch.optim.Adam(self.parameters(), lr=0.001)
        return optimizer

    def training_step(self,train_batch, batch_idx):
        X, y = train_batch
```

```
        y_hat = self(X)
        loss = F.cross_entropy(y_hat, y)
        pred = y_hat.argmax(dim=1, keepdim=True)
        acc = pred.eq(y.view_as(pred)).sum().item()/ y.shape[0]
        self.log("train_loss", loss)
        self.log("train_acc",acc)
        return loss

    def validation_step(self, val_batch, batch_idx):
        X,y =val_batch
        y_hat = self(X)
        loss = F.cross_entropy(y_hat, y)
        pred = y_hat.argmax(dim=1, keepdim=True)
        acc = pred.eq(y.view_as(pred)).sum().item()/ y.shape[0]
        self.log("val_loss", loss)
        self.log("val_acc", acc)

    def test_step(self, test_batch, batch_idx):
        X,y =test_batch
        y_hat = self(X)
        loss = F.cross_entropy(y_hat, y)
        pred = y_hat.argmax(dim=1, keepdim=True)
        acc = pred.eq(y.view_as(pred)).sum().item()/ y.shape[0]
        self.log("test_loss", loss)
        self.log("test_acc", acc)

@st.cache_resource(show_spinner=False)
def load_rice_model():
    model= torch.load('rice_disease.pth', map_location=torch.device('cpu'),
weights_only=False)
    model.eval()
    return model

@st.cache_resource(show_spinner=False)
def load_wheat_model():
    return load_model('wheat.h5')

defmake_rice_disease_prediction(image,model):
    preprocess = transforms.Compose([
```

```
        transforms.Resize((224,224)),
        transforms.ToTensor(),
    ])
    img_tensor = preprocess(image).unsqueeze(0) # Add batch dimension

    with torch.no_grad():
        prediction = model(img_tensor)
        disease_class= int(prediction.argmax(dim=1, keepdim=True))

    return disease_class

def make_wheat_disease_prediction(image, model):
    preprocess=tf.keras.preprocessing.image.ImageDataGenerator()
    image = image.resize((150, 150))
    image =preprocess.flow(np.expand_dims(np.array(image), axis=0))
    prediction = model.predict(image)
    disease_class =np.argmax(prediction)
    return disease_class

def signup():
    st.title("Signup Page")
    with st.form("signup_form"):
        st.write("Fill in the details below to create an account:")
        name = st.text_input("Name:")
        email = st.text_input("Email:")
        age= st.number_input("Age:", min_value=0, max_value=120)
        sex = st.radio("Sex:", ("Male", "Female", "Other"))
        password = st.text_input("Password:", type="password")
        confirm_password = st.text_input("Confirm Password:", type="password")

        if st.form_submit_button("Signup"):
            if password == confirm_password:
                user = create_account(name, email, age, sex, password)
                if user:
                    st.session_state["logged_in"]=True
                    st.session_state["user_info"] = user
                    st.success("Account created successfully! You are now logged in.")
            else:
                st.error("Passwords do not match. Please try again.")
```

```
def login():
    st.title("Login Page")
    username = st.text_input("Email:")
    password = st.text_input("Password:", type="password")

    login_button = st.button("Login")

    if login_button:
        user = check_login(username, password)
        if user is not None:
            st.session_state["logged_in"] = True
            st.session_state["user_info"]  =  user
            st.success("Login successful!")
        else:
            st.error("Invalid credentials. Please try again.")

defrender_dashboard(user_info):
    try:
        st.title(f"Welcometo the Dashboard,{user_info['name']}!")
        st.subheader("User Information:")
        st.write(f"Name: {user_info['name']}")
        st.write(f"Sex: {user_info['sex']}")
        st.write(f"Age: {user_info['age']}")

    except Exception as e:
        st.error(f"Error rendering dashboard: {e}")

def render_rice_disease_detection():
    st.title("Rice Disease Detection")

    uploaded_file = st.file_uploader("Upload an image", type=["jpg", "png",
"jpeg"])
    image = None
    if uploaded_file is not None:
        image = Image.open(uploaded_file)
        st.image(image,caption="Uploaded Image",use_container_width=True)

    if image:
```

```
        model = load_rice_model()
        disease_label= make_rice_disease_prediction(image, model)
        st.write(disease_label)
        disease = ['Bacterial Leaf Blight', 'Brown Spot', 'Healthy Rice Leaf', 'Leaf
Blast', 'Leaf scald', 'Sheath Blight'][disease_label]
        st.write(f"Detected Disease: {disease}")

defrender_wheat_disease_detection():
    st.title("Wheat Disease Detection")

    uploaded_file = st.file_uploader("Upload an image", type=["jpg", "png",
"jpeg"])
    image = None
    if uploaded_file is not None:
        image = Image.open(uploaded_file)
        st.image(image, caption="Uploaded Image", use_container_width=True)

    if image:
        model = load_wheat_model()
        disease_label = make_wheat_disease_prediction(image, model)
         disease = {0: 'Aphid', 1: 'Black Rust', 2: 'Blast', 3: 'Brown Rust', 4: 'Common
 Root Rot', 5: 'Fusarium Head Blight', 6: 'Healthy', 7: 'Leaf Blight', 8: 'Mildew', 9:
'Mite', 10: 'Septoria', 11: 'Smut', 12: 'Stem fly', 13: 'Tan spot', 14: 'Yellow
Rust'}[disease_label]
        st.write(f"Detected Disease: {disease}")

def main():
    st.set_page_config(page_title="SMARTVISION",layout="wide",
page_icon=" ")

    st.sidebar.title("SMARTVISION")
    page = st.sidebar.radio(
        "Goto",
        (
            "Signup/Login",
            "Dashboard",
            "Freshness Detector",
            "Rice Disease Detection",
            "Wheat Disease Detection"
```

```
        ),
        key="pages",
    )

    if page == "Signup/Login":
        st.title("Signup/Login Page")
        login_or_signup = st.radio(
            "Select an option", ("Login", "Signup"), key="login_signup"
        )
        if login_or_signup =="Login":
            login()
        else:
            signup()

    elif page == "Dashboard":
        if session_state.get("logged_in"):
            render_dashboard(session_state["user_info"])
        else:
            st.warning("Please login/signup to view the dashboard.")

    elif page == "Freshness Detector":
        if st.session_state.get("logged_in"):
            # Title and header styling
            st.header('Product Freshness and Shelf Life Estimator')

            # Capture Method selection with styling
            st.text('Select how to upload the image</div>')
            capture_method = st.selectbox("Capture Method", ["Upload an image",
"Capture image from camera"])

            image = None

            # Upload image section with customstyling
            if capture_method == "Upload an image":
                st.markdown('<div class="file-uploader">Upload an image of
fruit/vegetable (.jpg, .png, .jpeg)</div>', unsafe_allow_html=True)
                uploaded_file = st.file_uploader("", type=["jpg", "png", "jpeg"]) # File
uploader box styling
                if uploaded_file is not None:
```

```
                image = Image.open(uploaded_file)
                st.image(image, caption="Uploaded Image",
use_container_width=False, width=300) # Adjust the image size

        # Capture from camera section with custombutton styling
        elif capture_method == "Capture image from camera":
            capture_image= st.button("Capture Image",key="capture_btn",
help="Click to capture an image")

            # Initialize the camerafeed
            cap = cv2.VideoCapture(0)
            st_frame = st.empty() # Create a placeholder for the video frame

            whileTrue:
                ret,frame =cap.read()
                if not ret:
                    st.warning("Failed to access the camera.")
                    break

                # Convert the frame to RGB for displaying in Streamlit
                frame_rgb=cv2.cvtColor(frame,cv2.COLOR_BGR2RGB)
                st_frame.image(frame_rgb, channels="RGB",
use_container_width=False, width=300) # Show the camera feed

                # Check if the buttonwas pressed
                if capture_image:
                    image = Image.fromarray(frame_rgb) # Convert to PIL image
                    st.image(image, caption="Captured Image",
use_container_width=False, width=300) # Show captured image
                    break

            cap.release()

        # Display predicted freshness information if image is available
        if image is not None:
            model = load_resource()

            fruit_label, freshness_label = make_prediction(image, model)
            fruit_name = fruit_vegetable_mapping[fruit_label]
```

```
            freshness = freshness_mapping[freshness_label]

            st.markdown(f"Detected: {fruit_name}", unsafe_allow_html=True)
            st.markdown(f"Freshness: {freshness}", unsafe_allow_html=True)

            shelf_life_days = None # Initialize variable

            if freshness_label == 0: # Only show shelf life for fresh items
                try:
                    shelf_life_info = fruit_vegetable_shelf_life(image, freshness)

                    shelf_life_days = shelf_life_info.get('shelf_life', 'N/A')

                    col1, col2 = st.columns(2)
                    with col1:
                        st.markdown(f"<u>Estimated Shelf
Life:</u><br>{shelf_life_days}", unsafe_allow_html=True)

                    with col2:
                        if shelf_life_info.get('additional_content'):

st.markdown(f"<u>Remarks</u>:<br>{shelf_life_info['additional_content']}",
unsafe_allow_html=True)
                except Exception as e:
                    st.error(f"Error processing freshness information: {str(e)}")

             # Storeresults in database
             insert_freshness_record(
                produce=fruit_name,
                freshness=freshness,
                expected_life_span_days=shelf_life_days
            )
            st.success("Detection results saved to the database.")

        else:
            st.warning('Please Login to use the app!!')

    elif page == "Rice Disease Detection":
        if st.session_state.get("logged_in"):
```

```
            render_rice_disease_detection()
        else:
            st.warning("Please Login to use the app!!")

    elif page == "Wheat Disease Detection":
        if st.session_state.get("logged_in"):
            render_wheat_disease_detection()
        else:
            st.warning("Please Login to use the app!!")

if __name__ == "__main__":
    main()
    initialize_database()

api.py
from fastapi import FastAPI, UploadFile, File, HTTPException
from PIL import Image
import torch
import torchvision.transforms as transforms
import numpy as np
fromtensorflow.keras.models import load_model
import tensorflow as tf
fromio import BytesIO
import uvicorn
fromutils import fruit_vegetable_shelf_life
import torch.nn.functional as F
import lightning.pytorch as L
fromlightning.pytorch import LightningModule
import torchvision
fromfastapi.middleware.cors import CORSMiddleware
app = FastAPI(title="Agriculture Vision API")

app.add_middleware(
    CORSMiddleware,
    allow_origins=["*"], # Adjust with specific domains if needed
    allow_credentials=True,
    allow_methods=["*"],
    allow_headers=["*"],
)
```

```
# Model classes
importtorch.nn as nn
import torchvision.models as models

class Model(nn.Module):
    def __init__(self):
        super().__init__()
        self.alpha = 0.7
        self.base = torchvision.models.resnet18(pretrained=True)

        # Freeze some of the base layers
        forparam in list(self.base.parameters())[:-15]:
            param.requires_grad = False

        self.base.classifier=nn.Sequential()
        self.base.fc = nn.Sequential()

        # Define additional blocks for the model
        self.block1 = nn.Sequential(
            nn.Linear(512,256),
            nn.ReLU(),
            nn.Dropout(0.2),
            nn.Linear(256,128),
        )

        self.block2 =nn.Sequential(
            nn.Linear(128, 128),
            nn.ReLU(),
            nn.Dropout(0.1),
            nn.Linear(128, 9)
        )

        self.block3 =nn.Sequential(
            nn.Linear(128, 32),
            nn.ReLU(),
            nn.Dropout(0.1),
            nn.Linear(32, 2)
        )
```

```
    defforward(self,x):
        x = self.base(x)
        x = self.block1(x)
        y1, y2 =self.block2(x), self.block3(x)
        return y1, y2

classConvolutionalNetwork(LightningModule):

    def __init__(self):
        super(ConvolutionalNetwork, self).____init__()
        self.conv1 = nn.Conv2d(3, 6, 3, 1)
        self.conv2 = nn.Conv2d(6, 16, 3, 1)
        self.fc1 = nn.Linear(16 * 54 * 54, 120)
        self.fc2 = nn.Linear(120, 84)
        self.fc3 =nn.Linear(84, 20)
        self.fc4 = nn.Linear(20, 6)

    defforward(self, X):
        X = F.relu(self.conv1(X))
        X = F.max_pool2d(X, 2, 2)
        X = F.relu(self.conv2(X))
        X = F.max_pool2d(X, 2, 2)
        X = X.view(-1, 16 * 54 * 54)
        X = F.relu(self.fc1(X))
        X = F.relu(self.fc2(X))
        X = F.relu(self.fc3(X))
        X = self.fc4(X)
        return F.log_softmax(X, dim=1)

    def configure_optimizers(self):
        optimizer=torch.optim.Adam(self.parameters(), lr=0.001)
        return optimizer

    def training_step(self, train_batch, batch_idx):
        X,y =train_batch
        y_hat = self(X)
        loss = F.cross_entropy(y_hat, y)
        pred = y_hat.argmax(dim=1, keepdim=True)
        acc = pred.eq(y.view_as(pred)).sum().item() / y.shape[0]
```

```
        self.log("train_loss", loss)
        self.log("train_acc",   acc)
        return loss

    def validation_step(self, val_batch, batch_idx):
        X,y =val_batch
        y_hat = self(X)
        loss = F.cross_entropy(y_hat, y)
        pred = y_hat.argmax(dim=1, keepdim=True)
        acc = pred.eq(y.view_as(pred)).sum().item()/ y.shape[0]
        self.log("val_loss", loss)
        self.log("val_acc", acc)

    def test_step(self, test_batch, batch_idx):
        X,y =test_batch
        y_hat = self(X)
        loss = F.cross_entropy(y_hat, y)
        pred = y_hat.argmax(dim=1, keepdim=True)
        acc = pred.eq(y.view_as(pred)).sum().item()/ y.shape[0]
        self.log("test_loss", loss)
        self.log("test_acc", acc)

fruit_vegetable_mapping = {
    0: "apples", 1: "banana", 2: "bittergourd", 3: "capsicum", 4: "cucumber",
    5: "okra", 6: "oranges", 7: "potato", 8: "tomato"
}
freshness_mapping = {0: "Fresh", 1: "Spoiled"}

# Model Loaders
freshness_model=None
rice_model = None
wheat_model = None

defload_freshness_model():
    global freshness_model
    if not freshness_model:
        freshness_model = Model()
        freshness_model.load_state_dict(torch.load("model_fruit_freshness.pth",
map_location="cpu"))
```

```
        freshness_model.eval()
    return freshness_model

defload_rice_model():
    global rice_model
    if not rice_model:
        model= torch.load("rice_disease.pth", map_location="cpu",
weights_only=False)
        model.eval()
        rice_model =model
    return rice_model

defload_wheat_model():
    global wheat_model
    if not wheat_model:
        wheat_model =load_model("wheat.h5")
    return wheat_model

# Preprocessors
def preprocess_image(image: Image.Image, size=(224, 224)):
    if image.mode != "RGB":
        image = image.convert("RGB")
    transform= transforms.Compose([
        transforms.Resize(size),transforms.CenterCrop(size),
        transforms.ToTensor(),
        transforms.Normalize([0.485, 0.456, 0.406],[0.229, 0.224, 0.225])
    ])
    return transform(image).unsqueeze(0)

# Endpoints
@app.post("/freshness")
async def predict_freshness(file: UploadFile= File(...)):
    image = Image.open(BytesIO(await file.read()))
    model = load_freshness_model()
    img_tensor = preprocess_image(image)

    with torch.no_grad():
        pred_fruit, pred_fresh = model(img_tensor)
        fruit_idx =torch.argmax(pred_fruit, dim=1).item()
```

```
        fresh_idx = torch.argmax(pred_fresh, dim=1).item()

    fruit =fruit_vegetable_mapping[fruit_idx]
    freshness = freshness_mapping[fresh_idx]

    response= {"produce": fruit, "freshness": freshness}

    if fresh_idx == 0:  # Fresh
        try:
            shelf_life_info= fruit_vegetable_shelf_life(image, freshness)
            response.update(shelf_life_info)
        except Exception as e:
            response["warning"] = f"Could not estimate shelf life: {str(e)}"

    return response

@app.post("/rice-disease")
async def predict_rice_disease(file: UploadFile = File(...)):
    image = Image.open(BytesIO(await file.read()))
    model = load_rice_model()
    img_tensor = preprocess_image(image)

    with torch.no_grad():
        prediction = model(img_tensor)
        label = torch.argmax(prediction, dim=1).item()

    label_map = ['Bacterial Leaf Blight', 'Brown Spot', 'Healthy Rice Leaf', 'Leaf
Blast']
    return {"disease": label_map[label]}

@app.post("/wheat-disease")
async def predict_wheat_disease(file: UploadFile = File(...)):
    image= Image.open(BytesIO(await file.read())).resize((150, 150))
    model = load_wheat_model()
    arr = np.expand_dims(np.array(image), axis=0)
    prediction = model.predict(arr)
    label =np.argmax(prediction)

    label_map = {
```

```
        0: 'Aphid', 1: 'Black Rust', 2: 'Blast', 3: 'Brown Rust', 4: 'Common Root Rot',
        5: 'Fusarium Head Blight', 6: 'Healthy', 7: 'Leaf Blight', 8: 'Mildew', 9: 'Mite',
        10: 'Septoria', 11: 'Smut', 12: 'Stem fly', 13: 'Tan spot', 14: 'Yellow Rust'
    }

    return {"disease": label_map.get(label, "Unknown")}

if __name__=='__main__':
    uvicorn.run("api:app",port=8000,reload=True)
```

utils.py

```
fromdotenvimport load_dotenv
import os
import streamlit as st
from datetime import datetime
import google.generativeai as genai
import base64
importjson
import io

current_date=datetime.now().strftime("%Y-%m-%d")
print("Current Date:", current_date)

load_dotenv()
def load_model():
    genai.configure(api_key="AIzaSyD9W6OjxBjzx8FxuQThtzvUny-dQ_TrmTw")
    model = genai.GenerativeModel('gemini-1.5-flash')
    return model

defencode_image_to_base64(pil_image):
    # Convert PIL Image to bytes
    buffered = io.BytesIO()
    pil_image.save(buffered,format="PNG")
    # Encode to base64
    img_str=base64.b64encode(buffered.getvalue()).decode('utf-8')
    return img_str

def analyze_image_with_prompt(image, prompt):
```

```
    model = load_model()
    base64_image = encode_image_to_base64(image)

    parts = [
        {"text": prompt},
        {"inline_data": {"mime_type": 'image/png', "data": base64_image}}
    ]
    response=model.generate_content(parts)

    response_text = response.text

    try:
        if "```json" in response_text and "```" in response_text:
            json_content=response_text.split("```json")[1].split("```")[0].strip()
            patient_data = json.loads(json_content)
        else:
            patient_data=json.loads(response_text)
        return patient_data
    except json.JSONDecodeError:
        print("Failed to parse AI response into JSON format. Raw response: " +
response_text)

def fruit_vegetable_shelf_life(image, detection):
    prompt = f"'You are provided with an image of a fruit or vegetable. My model
has detected it as {detection}. Please analyze the image and provide the following:

1. Assess the freshness of the item based on visual indicators such as color, texture,
and condition.
2. Estimate the approximate remaining shelf life of the fruit or vegetable in days
(1-2 sentences).
  - Forfresh produce, give an estimated shelf life based on its current appearance.
  - Foritems that appear less fresh, adjust the shelf life accordingly.
  - Bothshelf life and additional_content are compulsory so dont give N/A.

Please return your analysis in the following structured JSON format:

  "shelf_life": "X days",
  "additional_content": "Any relevant comments about freshness, ripeness, or
possible spoilage."
```

```
'''
    return analyze_image_with_prompt(image, prompt)

database.py
importsqlite3
import streamlit as st

def get_database_connection():
    """Create a database connection and return connection and cursor objects."""
    conn = sqlite3.connect('user_data.db', check_same_thread=False)
    c=conn.cursor()
    return conn, c

def initialize_database():
    """Initialize SQLite database and create necessary tables if they don't exist."""
    try:
        conn, c = get_database_connection()

        # Create users table if it doesn't exist
        c.execute('''
            SELECT count(name) FROM sqlite_master
            WHERE type='table' AND name='users'
        ''')

        if c.fetchone()[0] ==0:
            c.execute('''
                CREATETABLE users (
                    id INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL,
                    name TEXT NOT NULL CHECK(length(name) > 0),
                    email TEXT NOT NULL UNIQUE CHECK(length(email) > 0 AND
email LIKE '%@%'),
                    age INTEGER NOT NULL CHECK(age >= 0 AND age <= 120),
                    sex TEXT NOT NULL CHECK(sex IN ('Male', 'Female', 'Other')),
                    password TEXT NOT NULL CHECK(length(password) >= 6),
                    created_at TIMESTAMPDEFAULT CURRENT_TIMESTAMP NOT
NULL,
                    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP NOT
NULL
```

```
            )
        ''')

        c.execute('''
            CREATETRIGGER IF NOT EXISTS update_user_timestamp
            AFTER UPDATE ON users
            BEGIN
                UPDATEusers SET updated_at = CURRENT_TIMESTAMP
                WHERE id = NEW.id;
            END;
        ''')

    # Create freshness_records table if it doesn't exist
    c.execute('''
        SELECT count(name) FROM sqlite_master
        WHEREtype='table' AND name='freshness_records'
    ''')

    if c.fetchone()[0] ==0:
        c.execute('''
            CREATETABLE freshness_records (
                sl_no INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL,
                timestamp TIMESTAMP DEFAULT CURRENT_TIMESTAMP NOT
NULL,
                produce TEXT NOT NULL CHECK(length(produce) > 0),
                freshness TEXT NOT NULL CHECK(length(freshness) > 0),
                expected_life_span_days INTEGER
            )
        ''')

    conn.commit()
    st.toast("Database initialized successfully.")

except sqlite3.Error as e:
    st.error(f"Databaseinitialization error: {e}")
finally:
    conn.close()
```

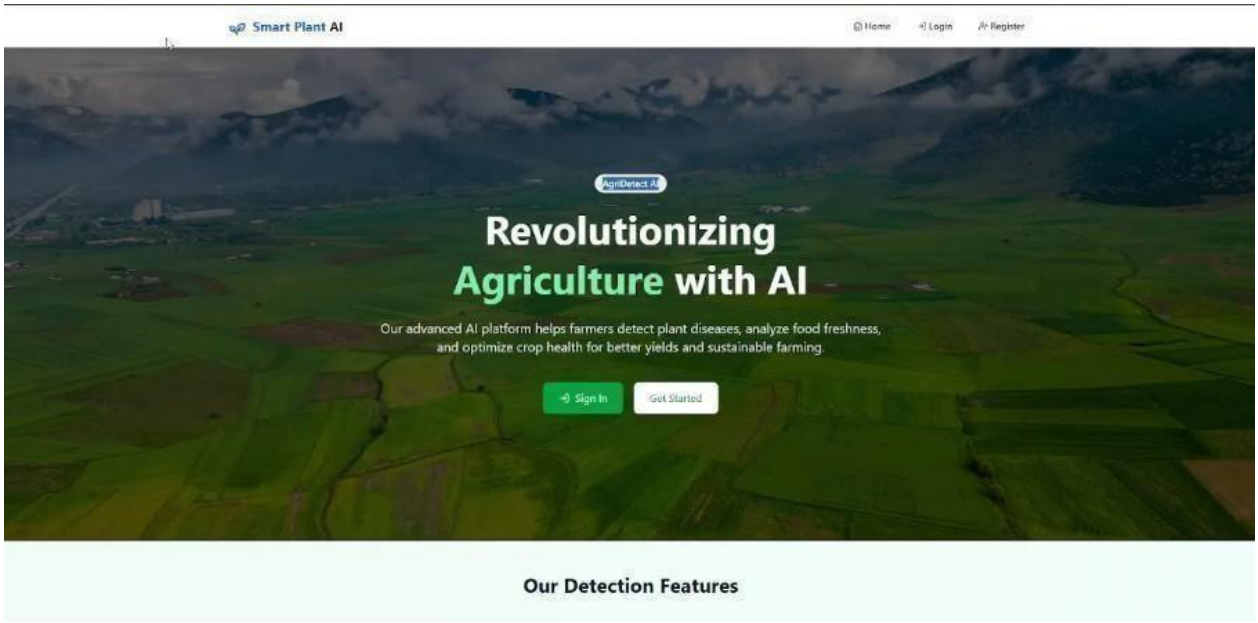
```
def insert_freshness_record(produce, freshness, expected_life_span_days):
    """Insert a new freshness record into the freshness_records table."""
    try:
        conn, c = get_database_connection()
        c.execute('''
            INSERT INTO freshness_records (produce, freshness,
expected_life_span_days)
            VALUES (?, ?, ?)
        ''', (produce, freshness, expected_life_span_days))
        conn.commit()
    except sqlite3.Error as e:
        st.error(f"Database error while inserting freshness record: {e}")
    finally:
        conn.close()

def create_account(name, email, age, sex, password):
    """Create a new user account in the SQLite database with validation."""
    try:
        # Input validation
        if not name or len(name.strip()) == 0:
            st.error("Name cannot be empty")
            return None

        if not email or '@' not in email:
            st.error("Invalid emailformat")
            return None

        if not isinstance(age, int) or age < 0 or age > 120:
            st.error("Age must be between 0 and 120")
            return None

        if sex not in ['Male', 'Female', 'Other']:
            st.error("Invalid sex selection")
            return None

        if not password or len(password) < 6:
            st.error("Password must be at least 6 characters long")
            return None
```

```
        conn, c = get_database_connection()

        # Check if email already exists
        c.execute('SELECT * FROM users WHERE email = ?', (email,))
        if c.fetchone() is not None:
            st.error("Email already exists!")
            return None

        # Insert new user
        c.execute('''
            INSERT INTO users (name, email, age, sex, password)
            VALUES (?, ?, ?, ?, ?)
        ''', (name.strip(), email.lower().strip(), age, sex, password))

        conn.commit()

        # Get the user info to return
        c.execute('SELECT * FROM users WHERE email = ?', (email,))
        user_data = c.fetchone()

        if user_data:
            user_info = {
                "name": user_data[1],
                "email": user_data[2],
                "age": user_data[3],
                "sex": user_data[4],
                "password": user_data[5]
            }
            return user_info
        return None

    except sqlite3.Error as e:
        st.error(f"Database error: {e}")
        return None
    finally:
        if 'conn' in locals():
            conn.close()

def check_login(username, password):
```

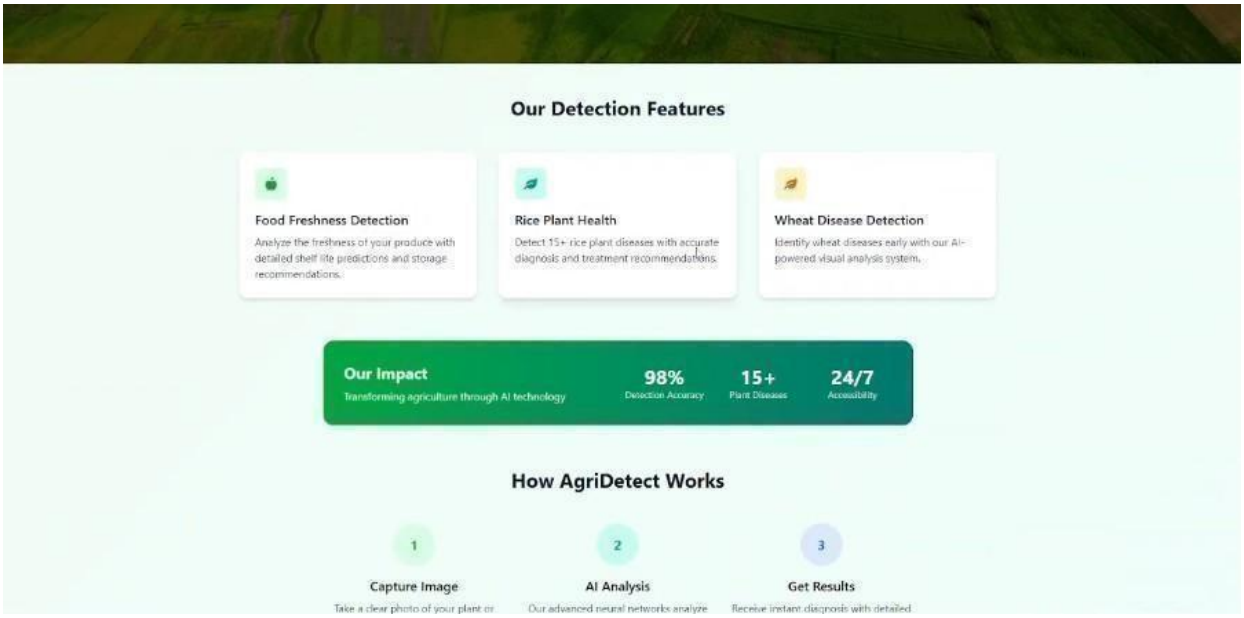
```
    """Check login credentials against SQLite database with validation."""
    try:
        if not username or not password:
            st.error("Username and password are required")
            return None

        conn, c = get_database_connection()

        # Check credentials
        c.execute('SELECT * FROMusers WHERE LOWER(email) =LOWER(?)
AND password = ?',
                  (username.strip(), password))
        user_data = c.fetchone()

        if      user_data:
            user_info = {
                "name": user_data[1],
                "email": user_data[2],
                "age": user_data[3],
                "sex":          user_data[4],
                "password": user_data[5]
            }
            returnuser_info
        return None

    except sqlite3.Error as e:
        st.error(f"Databaseerror: {e}")
        return None
    finally:
        if 'conn' in locals():
            conn.close()
```

B. SCREENSHOT

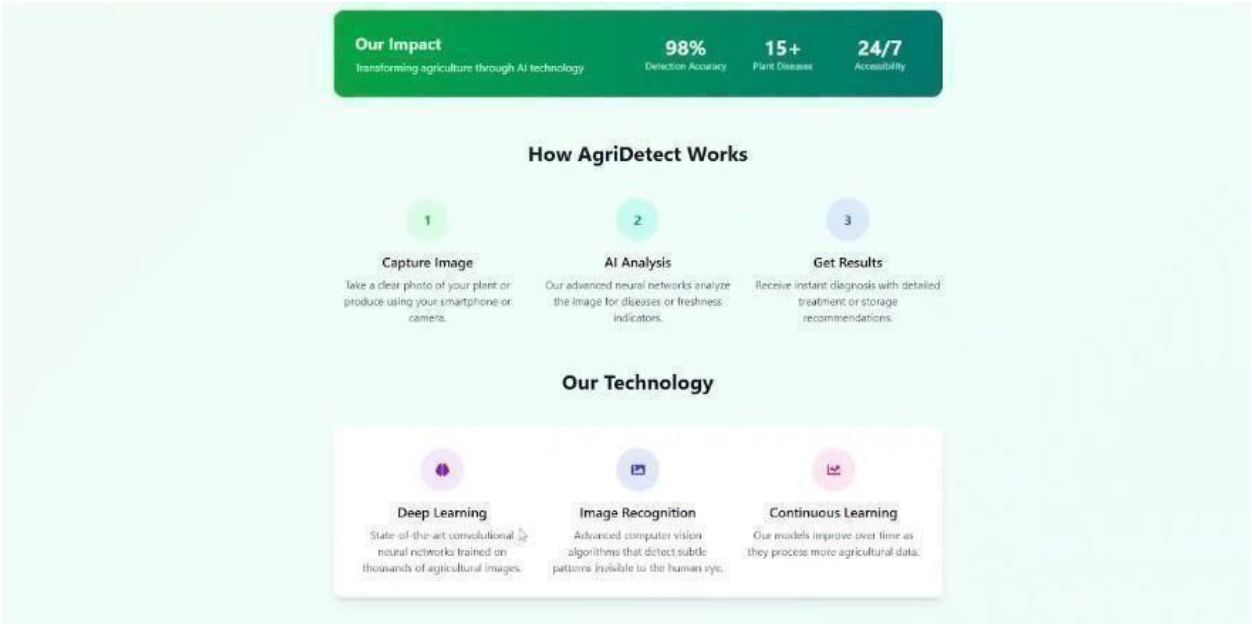
LANDING PAGE



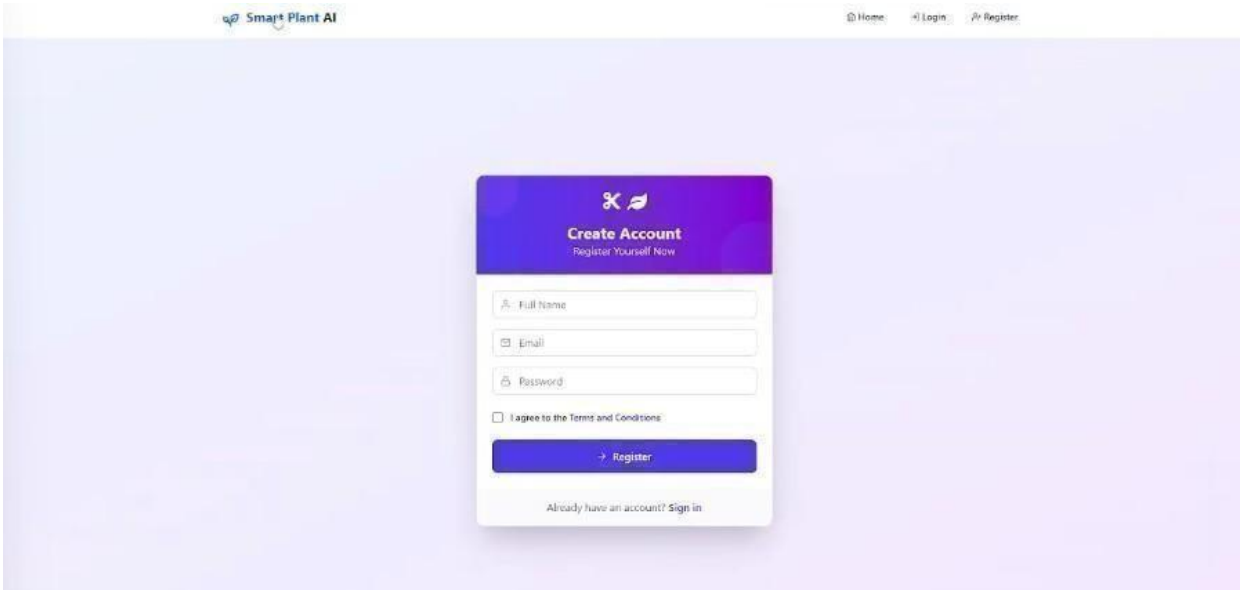
ABOUT PAGE - 1



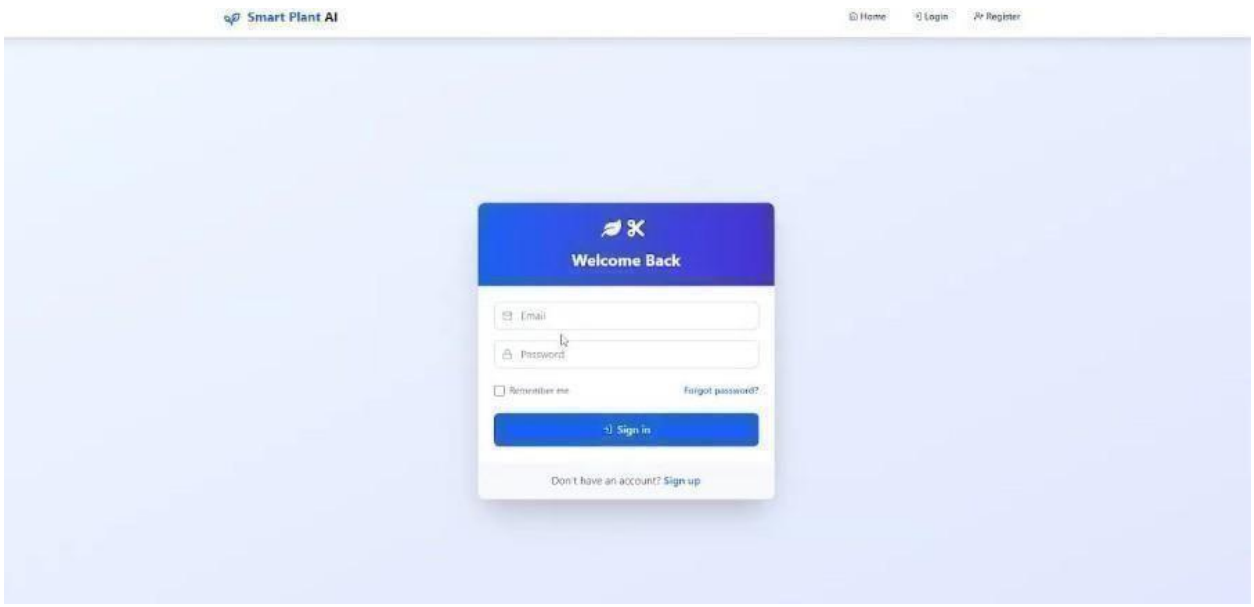
ABOUT PAGE – 2



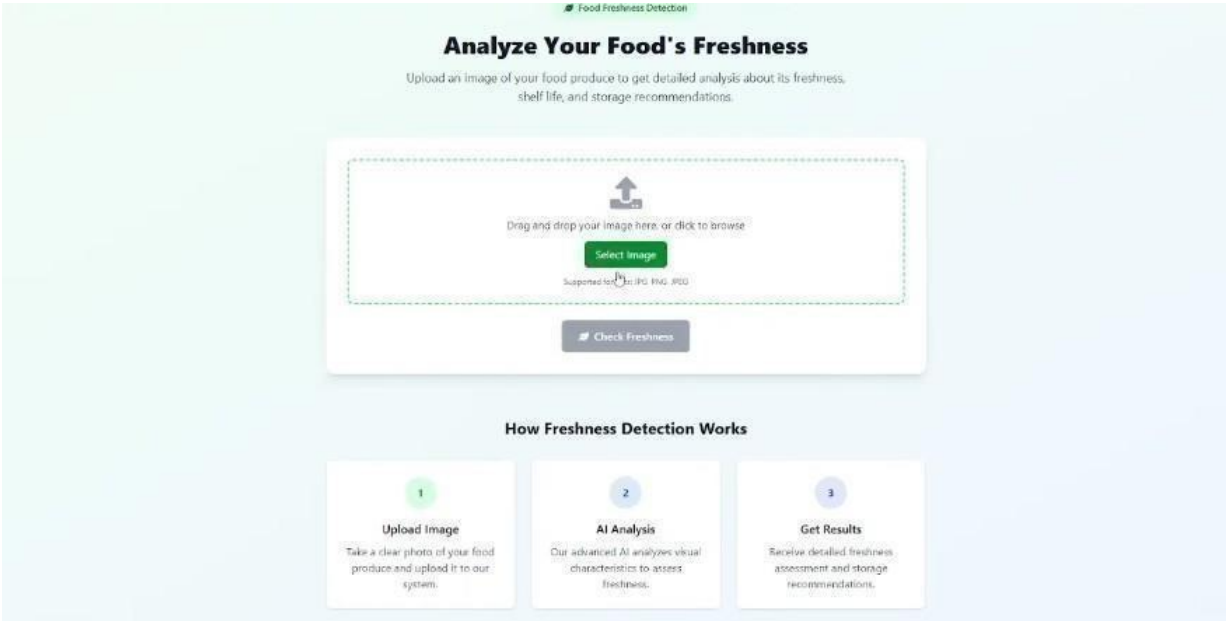
USER REGISTER PAGE



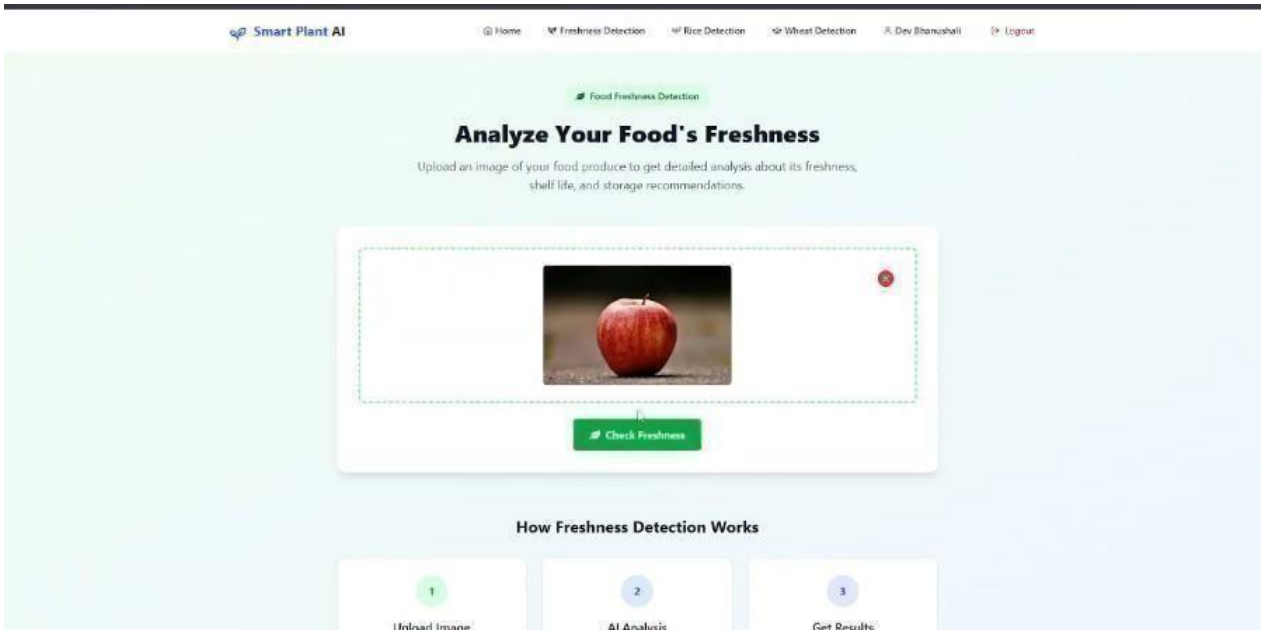
USER SIGN IN PAGE



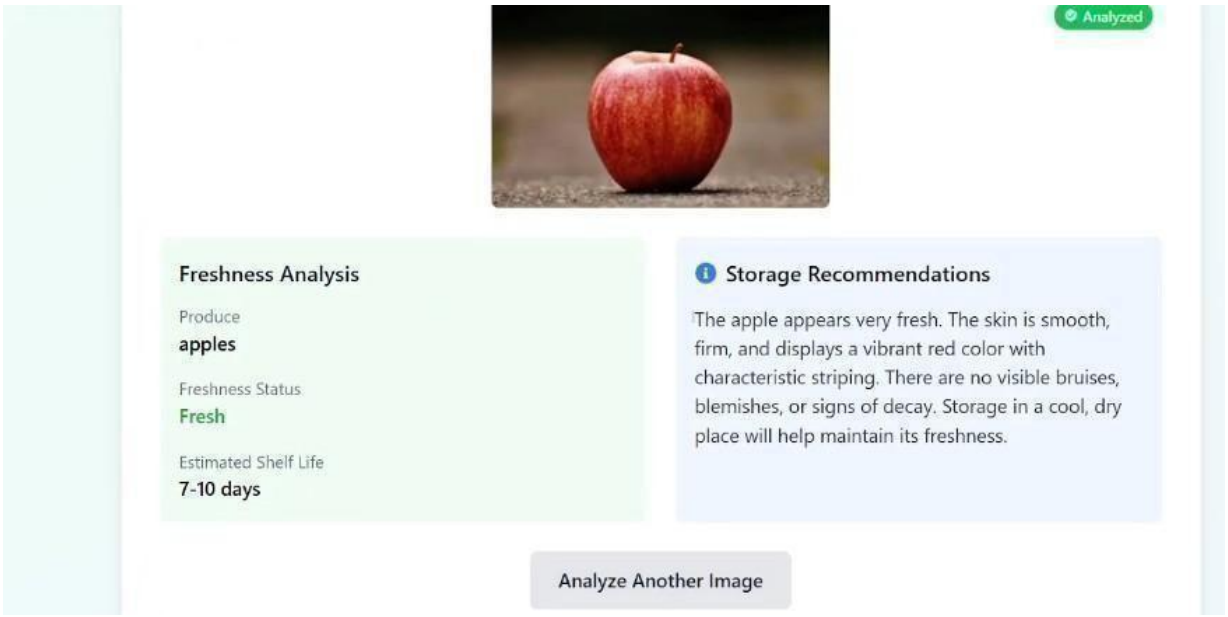
FOOD FRESHNESS ANALYSER - 1



FOOD FRESHNESS ANALYSER - 2 (IMAGE UPLOAD)

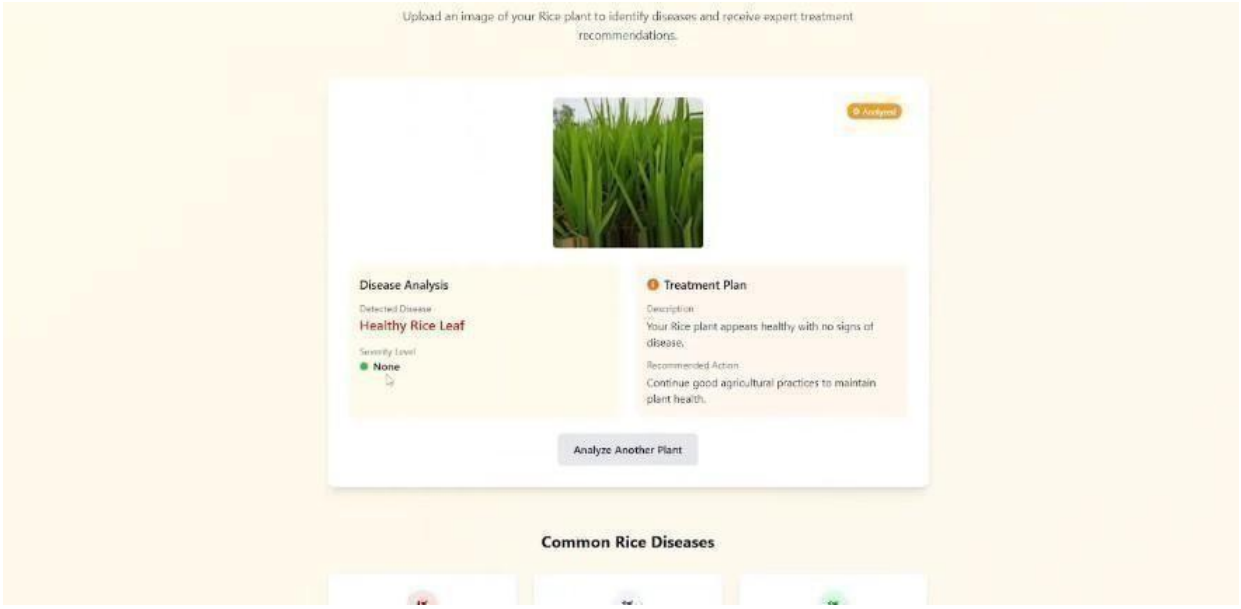


FOOD FRESHNESS ANALYSER - 3 (RESULTSAND RECOMMENDATIONS)

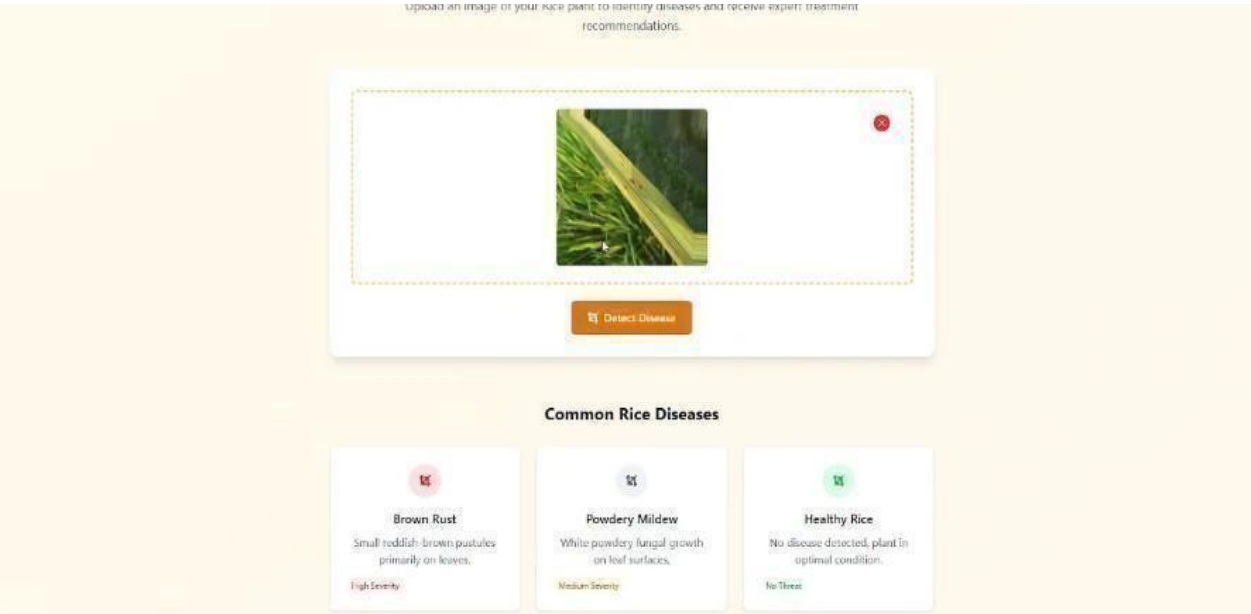


RICE CROP HEALTH ANALYSER - 1 (IMAGE UPLOAD)

RICECROP HEALTH ANALYSER - 2 (DETECTION OF HEALTHY CROP)

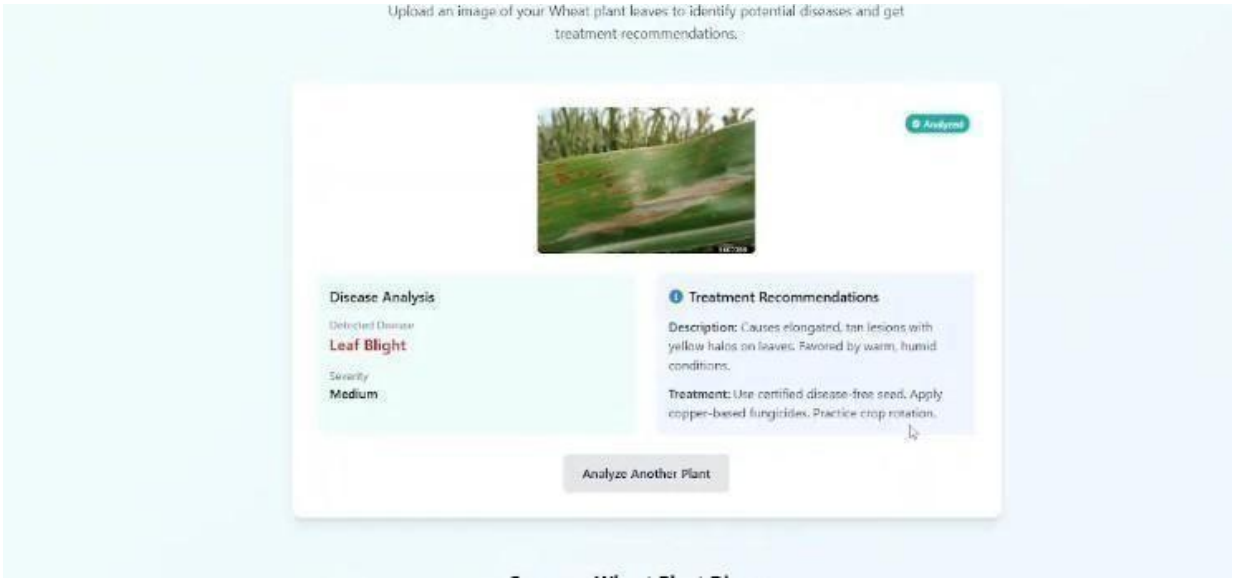


RICE CROP HEALTHANALYSER - 3 (DISEASE DETECTION)



WHEATCROP HEALTH ANALYSER - 1 (IMAGE UPLOAD)

WHEAT CROP HEALTH ANALYSER - 2 (DISEASEDETECTION AND RECOMMENDATION)



REFERENCES

1. Gupta, S., & Tripathi, A. K. (2024). Fruit and vegetable disease detection and classification: Recent trends, challenges, and future opportunities. *Engineering Applications of Artificial Intelligence*, *133*, 108260.
2. Tang, T., Zhang, M., & Mujumdar, A. S. (2022). Intelligent detection for fresh-cut fruit and vegetable processing: Imaging technology. *Comprehensive reviews in food science and food safety*, *21*(6), 5171-5198.
3. Sharma, R., & Kukreja, V. (2021, March). Rice diseases detection using convolutional neural networks: a survey. In *2021 International Conference on Advance Computing and Innovative Technologies in Engineering (ICACITE)* (pp. 995-1001). IEEE.
4. Shafi, U., Mumtaz, R., Shafaq, Z., Zaidi, S. M. H., Kaifi, M. O., Mahmood, Z., & Zaidi, S. A. R. (2022). Wheat rust disease detection techniques: a technical perspective. *Journal of Plant Diseases and Protection*, *129*(3), 489-504.
5. Khakimov, A., Salakhutdinov, I., Omolikov, A., & Utaganov, S. (2022). Traditional and current-prospective methods of agricultural plant diseases detection: A review. In *IOP Conference series: earth and environmental science* (Vol. 951, No. 1, p. 012002). IOP Publishing.
6. Ali, A., Mansol, A. S., Khan, A. A., Muthoosamy, K., & Siddiqui, Y. (2023). Electronic nose as a tool for early detection of diseases and quality monitoring in fresh postharvest produce: A comprehensive review. *Comprehensive Reviews in Food Science and Food Safety*, *22*(3), 2408-2432.
7. Wang, D., Zhang, M., Jiang, Q., & Mujumdar, A. S. (2024). Intelligent system/equipment for quality deterioration detection of fresh food: recent advances and application. *Foods*, *13*(11), 1662.

8. Temniranrat, P., Kiratiratanapruk, K., Kitvimonrat, A., Sinthupinyo, W., & Patarapuwadol, S. (2021). A system for automatic rice disease detection from rice paddy images serviced via a Chatbot. *Computers and Electronics in Agriculture*, *185*, 106156.
9. Gaikwad, S., & Tidke, S. (2022). Multi-spectral imaging for fruits and vegetables. *International Journal of Advanced Computer Science and Applications*, *13*(2), 743-760.
10. Teet, S. E., & Hashim, N. (2023). Recent advances of application of optical imaging techniques for disease detection in fruits and vegetables: A review. *Food Control*, *152*, 109849.
11. Reis, H. C., & Turk, V. (2024). Integrated deep learning and ensemble learning model for deep feature-based wheat disease detection. *Microchemical Journal*, *197*, 109790.

PUBLICATION DETAILS:


ISSN (Online): 2321-7782
ISSN (Print): 2347-1778
IMPACT FACTOR: 6.012
e-ISJN: A4372-3114
p-ISJN: A4372-3115
A Monthly Double-Blind Peer Reviewed, Refereed, Open Access International Journal - Included in the International Serial Directories
International Journal of Advance Research in Computer Science and Management Studies
Sr. No. May25/8/3
CERTIFICATE
Date : 13/05/2025
Dr/Prof/Mr/Ms Thilocigan K
This is to certify that the Research Article/Paper/Case Study titled
CNN-Based Agricultural Analysis: Detecting Plant Diseases, Freshness, and Nutritional Content in Wheat, Rice, Fruits, and Vegetables
Submitted by you published in
May 2025, Volume.No. 13, Issue No. 5 in this Journal
For further Information Please Visit www.ijarcsms.com
All Rights Reserved
Publisher
IJARCSMS
This Certificate is deemed authentic
when it bears the Signatures and the Journal's Seal


ISSN (Online): 2321-7782
ISSN (Print): 2347-1778
IMPACT FACTOR: 6.012
e-ISJN: A4372-3114
p-ISJN: A4372-3115
A Monthly Double-Blind Peer Reviewed, Refereed, Open Access International Journal - Included in the International Serial Directories
International Journal of Advance Research in Computer Science and Management Studies
Sr. No. May25/8/5
CERTIFICATE
Date : 13/05/2025
Dr/Prof/Mr/Ms Rishidharan V
This is to certify that the Research Article/Paper/Case Study titled
CNN-Based Agricultural Analysis: Detecting Plant Diseases, Freshness, and Nutritional Content in Wheat, Rice, Fruits, and Vegetables
Submitted by you published in
May 2025, Volume.No. 13, Issue No. 5 in this Journal
For further Information Please Visit www.ijarcsms.com
All Rights Reserved
Publisher
IJARCSMS
This Certificate is deemed authentic
when it bears the Signatures and the Journal's Seal


ISSN (Online): 2321-7782
ISSN (Print): 2347-1778
IMPACT FACTOR: 6.012
e-ISJN: A4372-3114
p-ISJN: A4372-3115
A Monthly Double-Blind Peer Reviewed, Refereed, Open Access International Journal - Included in the International Serial Directories
International Journal of Advance Research in Computer Science and Management Studies
Sr. No. May25/B/4
CERTIFICATE
Date : 13/05/2025
Dr/Prof/Mr/Ms Sanjay A
This is to certify that the Research Article/Paper/Case Study titled
CNN-Based Agricultural Analysis: Detecting Plant Diseases, Freshness, and Nutritional Content in Wheat, Rice, Fruits, and Vegetables
Submitted by you published in
May 2025, Volume No. 13, Issue No. 5 in this Journal
For further Information Please Visit www.ijarcsms.com
All Rights Reserved
Publisher
IJARCSMS
This Certificate is deemed authentic
when it bears the Signatures and the Journal's Seal


ISSN (Online): 2321-7782
ISSN (Print): 2347-1778
IMPACT FACTOR: 6.012
e-ISJN: A4372-3114
p-ISJN: A4372-3115
A Monthly Double-Blind Peer Reviewed, Refereed, Open Access International Journal - Included in the International Serial Directories
International Journal of Advance Research in Computer Science and Management Studies
Sr. No. May25/8/2
CERTIFICATE
Date : 13/05/2025
Dr/Prof/Mr/Ms Naveen Kumar P
This is to certify that the Research Article/Paper/Case Study titled
CNN-Based Agricultural Analysis: Detecting Plant Diseases, Freshness, and Nutritional Content in Wheat, Rice, Fruits, and Vegetables
Submitted by you published in
May 2025, Volume.No. 13, Issue No. 5 in this Journal
For further Information Please Visit www.ijarcsms.com
All Rights Reserved
Publisher
IJARCSMS
This Certificate is deemed authentic
when it bears the Signatures and the Journal's Seal

e-ISJN: A4372-3114
p-ISJN: A4372-3115

ISSN: 2321-7782 (Online)
ISSN: 2347-1778 (Print)
Impact Factor: 6.012

Volume 13, Issue 5, May 2025

International Journal of Advance Research in Computer Science and Management Studies

Research Article / Survey Paper / Case Study

Available online at: www.ijarcsms.com

A Monthly Double-Blind Peer Reviewed, Refereed, Open Access, International Journal - Included in the International Serial Directories

CNN-Based Agricultural Analysis: Detecting Plant Diseases, Freshness, and Nutritional Content in Wheat, Rice, Fruits, and Vegetables

Dr. K. Pandikumar, ME, Ph.D[1]
Assistant Professor,
Department of Computer Science and Engineering,
Dhanalakshmi College of Engineering
Chennai, Tamilnadu, India

Naveen Kumar P[2]
Student of Computer Science and Engineering
Dhanalakshmi College of Engineering,
Chennai, Tamilnadu, India.

Thilocigan K[3]
Student of Computer Science and Engineering
Dhanalakshmi College of Engineering,
Chennai, Tamilnadu, India.

Sanjay A[4]
Student of Computer Science and Engineering
Dhanalakshmi College of Engineering,
Chennai, Tamilnadu, India.

Rishidharan V[5]
Student of Computer Science and Engineering
Dhanalakshmi College of Engineering,
Chennai, Tamilnadu, India.

DOI: Available on author(s) request

Abstract: ***Agriculture plays a crucial role in global food security, and advancements in machine learning are significantly improving the way we manage agricultural resources. This project presents a deep learning-based framework using Convolutional Neural Networks (CNNs) to analyze and classify plant health, freshness, and nutritional content in crops such as wheat, rice, fruits, and vegetables. By leveraging large datasets of crop images, the model detects plant diseases, assesses freshness, and estimates nutritional content based on visual cues. The system provides real-time insights for farmers, agronomists, and food industry professionals, facilitating early disease detection, quality control, and sustainable farming practices. The proposed CNN model is trained using a variety of agricultural images, pre-processed to normalize environmental factors and enhance disease or quality features. The results demonstrate high accuracy in detecting diseases such as rust, blight, and mold, as well as the ability to estimate freshness levels and key nutritional elements such as vitamins and minerals. This research aims to provide a cost-effective and scalable solution for enhancing agricultural practices and ensuring food quality.***

Keywords: Convolutional Neural Networks (CNN), Agricultural Analysis, Plant Diseases, Freshness Detection, Nutritional Content, Wheat, Rice, Fruits, Vegetables, Deep Learning, Crop Health, Disease Classification, Quality Control.

I. Introduction

Agriculture is the backbone of global food security, with crops such as wheat, rice, fruits, and vegetables providing essential nutrients to populations worldwide. However, the agricultural sector faces numerous challenges, including plant diseases, loss of crop freshness, and the need for quality control to ensure optimal nutritional content. Traditional methods of