SAVITRIBAI PHULE PUNE UNIVERSITY, PUNE

A

PROJECT STAGE-I REPORT ON

"Sugarcane Disease Identification Using Deep Learning"

SUBMITTED TOWARDS THE

PARTIAL FULFILLMENT OF THE AWARD OF THE DEGREE OF

BACHELOR OF TECHNOLOGY (COMPUTER ENGINEERING)

BY

MR. ANSARI MOHAMMED ANAS	[UCS20M1011]
MR. ADARSH BORDE	[UCS20M1028]
MR. PRANAV JOSHI	[UCS20M1057]
MS. SANSKRUTI KEKAN	[UCS20F1068]

UNDER THE GUIDANCE OF

Dr. P. N. Kalavadekar

Department of Computer Engineering

SANJIVANI COLLEGE OF ENGINEERING

KOPARGAON-423603

(AN AUTONOMOUS INSTITUTE)

2023-2024

[07/2023-2024]

Sanjivani College of Engineering, Kopargaon-423603

(An Autonomous Institute)

DEPARTMENT OF COMPUTER ENGINEERING

CERTIFICATE

This is to certify that the project entitled

"Sugarcane Disease Identification Using Deep Learning"

Submitted by

MR. ANSARI MOHAMMED ANAS	**[UCS20M1011]**
MR. ADARSH BORDE	**[UCS20M1028]**
MR. PRANAV JOSHI	**[UCS20M1057]**
MS. SANSKRUTI KEKAN	**[UCS20F1068]**

is a bonafide work carried out by students under the supervision of Dr. P. N. Kalavadekar and it is submitted towards the partial fulfillment of the requirement of Bachelor of Technology (Computer Engineering).

During the Academic Year 2023-24

Dr. P. N. Kalavadekar
[Internal Guide]

Dr. S.R. Deshmukh
[Project Co-Ordinator]

Dr. D. B. Kshirsagar
[Head of Dept.]

Dr. A. G. Thakur
[Director]

Signature of Internal Examiner

Signature of External Examiner

PROJECT APPROVAL SHEET

A

PROJECT STAGE- I REPORT

ON

"“Sugarcane Disease Identification Using Deep Learning”

Is Successfully Completed By

MR. ANSARI MOHAMMED ANAS	**[UCS20M1011]**
MR. ADARSH BORDE	**[UCS20M1028]**
MR. PRANAV JOSHI	**[UCS20M1057]**
MS. SANSKRUTI KEKAN	**[UCS20F1068]**

At

DEPARTMENT OF COMPUTER ENGINEERING

SANJIVANI COLLEGE OF ENGINEERING,KOPARGAON – 423603

(AN AUTONOMOUS INSTITUTE)

SAVITRIBAI PHULE PUNE UNIVERSITY, PUNE

ACADEMIC YEAR 2023-24

Dr. P. N. Kalavadekar	**Dr. S.R. Deshmukh**
[Internal Guide]	[Project Co-Ordinator]
Dr. D. B. Kshirsagar	**Dr. A. G. Thakur**
[Head of Dept.]	[Director]

ACKNOWLEDGEMENT

"Sugarcane Disease Identification Using Deep Learning" has been a subject with tremendous scope to research, which leads one's mind to explore new heights in the field of Computer Engineering, and its miscellaneous applications. We dedicate all our project work to our esteemed guide **Dr. P. N. Kalavadekar** whose interest and guidance helped us to complete the work successfully as well as he has provided facilities to explore the subject with more enthusiasm.

This experience will always encourage us to do our work perfectly and professionally. We also extend our gratitude to **Dr. D. B. Kshirsagar (H.O.D. Computer Department).**

We express our immense pleasure and thankfulness to all the teachers and staff of the Department of Computer Engineering, Sanjivani College of Engineering, Kopargaon for their cooperation and support. Last but not least, we thank all others, and especially our parents and friends, who in one way or another, helped us in the successful completion of this project.

MR. ANSARI MOHAMMED ANAS
MR. ADARSH BORDE
MR. PRANAV JOSHI
MS. SANSKRUTI KEKAN

(B.TECH. COMPUTER)

ABSTRACT

Sugarcane is a vital crop, contributing significantly to the global sugar industry. However, the health of sugarcane plants is often threatened by various diseases, which can have devastating effects on crop yield. Early detection and accurate identification of these diseases are crucial for effective disease management and sustainable crop production. In this study, we propose a novel approach to tackle this challenge by harnessing the power of deep learning techniques.

Our research focuses on the development of a deep-learning model capable of automatically identifying and segmenting sugarcane diseases in images. We have curated a comprehensive dataset of sugarcane plant images with a variety of disease manifestations. Through the application of convolutional neural networks (CNNs) and advanced image segmentation algorithms, our model effectively identifies the presence of diseases and provides pixel-level segmentation, enabling precise localization of infected regions. The results of our experiments demonstrate the model's high accuracy in disease detection and segmentation, making it a promising tool for sugarcane disease monitoring.

By automating this process, our approach can significantly reduce the time and labor required for disease assessment, thus aiding farmers in making timely interventions. This research contributes to the advancement of precision agriculture and offers a potential solution to mitigate the impact of sugarcane diseases on crop productivity.

Contents

List of Figures

List of Tables

Chapter 1

INTRODUCTION

1.1 Problem Definition

Our final year project addresses the complex challenge of sugarcane disease detection, which poses significant difficulties for farmers. In many cases, farmers invest resources in disease management without sufficient technical support, resulting in suboptimal disease control. Our proposed approach offers a solution to improve disease detection accuracy while minimizing farmer effort and reducing detection time.

Sugarcane is susceptible to various diseases caused by factors such as climate change, fungal attacks, and leaf drying due to excessive heat. Detecting these diseases accurately is pivotal in enhancing farmers' crop yields and mitigating disease outbreaks. Our project seeks to develop a user-friendly, efficient system that leverages technology, specifically Deep Learning, to empower farmers with a more precise and timely disease identification tool. This will ultimately contribute to improved sugarcane crop management and better disease control in the agricultural sector.

1.2 Literature Review/Relevant Theory

Literature survey is the most important step in software development process. Before developing the tool, it is necessary to determine the time factor, economy and company strength. Once these things are satisfied, ten next steps are to determine which operating system and language can be used for developing the tool. Once the programmers start building the tool the programmers need lot of external support. This support can be obtained from senior programmers, from book or from websites. Before building the system, the above consideration is taken into account for developing the proposed system.

Conference/Journal: Published in 2019 International Journal Of Scientific Technology Research Volume 8, Issue 11.

Paper Title: Integrated Management System For Sugarcane Disease Using Deep Learning Techniques-A Review

Author: Rutuja Kadam, Aniket Jagtap, Rahul Joshi

Agriculture produces a civilized world, in India it is one of the agricultural countries, and the economy of India is mainly based on crop production. That is why agriculture is the backbone of business. Agriculture in India is influenced by various factors like geography, climate, historical, institutional, political, biological development, geographical, for all, and socio-economic factors. Agricultural production is mainly influenced by environmental factors. The weather affects the growth of crops, and the seasonal yield fluctuation is large. The spatial variability of soil properties interacting with weather causes spatial yield fluctuations. Crop cultivation is Agricultural Management, fertilizer spraying, irrigation; it can be used to offset harvest losses due to weather effects.

As a result, harvest prediction represents an important tool for optimizing crop yields and assessing crop area insurance policies. Plant diseases and pests affect food crops, resulting in huge losses to farmers and threaten food security. In rural India sugarcane is one of the commercial crops and the sugar industry is the largest agro-based industries. From sugarcane forms the variety of products such as Jagger, Sugar, Molasses, Green Top, Filter Cake, and Bagasse. The disease in the cultivation of sugar cane is not only the reduction in yield, but also the deterioration of the variety. Most of the time, the quality of sugar cane production depends on its robustness from the disease.

1.3 Scope

1. **Image Dataset Collection:** Gathering a diverse dataset of sugarcane images containing healthy and diseased samples from various sources.
2. **Deep Learning Model Development:** Building and fine-tuning convolutional neural networks (CNNs) or other suitable architectures for disease identification and segmentation.

3. **Real-world Deployment:** Considering the feasibility of implementing the system in real-world agricultural settings.
4. **Performance Evaluation:** Evaluating the accuracy and efficiency of the model in disease identification and segmentation.

1.4 Objectives

- **Automated Disease Detection:** Develop a deep learning model capable of automatically detecting sugarcane diseases, distinguishing between healthy and infected crops.
- **Accurate Segmentation:** Implement an image segmentation component within the system to precisely outline disease-affected areas on sugarcane leaves.
- **Multi-Disease Recognition:** Enable the model to identify multiple sugarcane diseases, providing a comprehensive solution for farmers.
- **Scalability:** Consider the potential for scalability and adaptation of the system for different regions and disease types to maximize its impact in the agricultural sector.

Chapter 2

REQUIREMENT ANALYSIS

Requirements Analysis or requirement engineering is a process of determining user expectations for new software or providing updates for previous products. These core points must be measurable, relevant and detailed. In the software engineering field this term is also called functional specifications. Requirements analysis mainly deals with communication with users or customers to determine system feature expectations, requirements and reduce convicts as demanded by various software users. Energy should be directed towards ensuring that the system or product conforms to user needs rather than attempting to turn user expectations to the requirements.

2.1 Requirement Specifications

Requirement specification describes the function and performance of the computer based system and constraints which govern its development. It can be a written document, a set of graphical models, a collection of scenarios, or any combination of above. These are of 3 types:

1. NR: Normal Requirements
2. ER: Expected Requirements
3. XR: Exciting Requirements

2.1.1 Normal Requirements

These are the requirements which are clearly stated by the customer so all these requirements will be present in the project for user satisfaction.

- NR1: System to correctly identify and classify sugarcane diseases to facilitate timely and accurate interventions.
- NR2: A user-friendly interface that allows easy interaction with the system.
- NR3: Identify multiple diseases affecting sugarcane.

2.1.2 Expected Requirements

These requirements are expected by the customer but not clearly stated by the customer. These are implicit types of requirements.

- ER1: System should be fast and reliable.
- ER2: System should have a neat and clean user interface.
- ER3: System should provide instruction of usage to the user.

2.1.3 Exciting Requirements

These requirements are not stated by the customer but externally provided by the developer in order to maintain a good relationship with the customer.

- XR1: Implement an autonomous monitoring system that uses drones equipped with cameras to capture images of sugarcane fields.
- XR2: Develop a predictive analytics feature that uses historical data and environmental factors to forecast potential disease outbreaks in sugarcane crops

2.2 Validation of Requirements

Requirements validation is the process of checking that requirements defined for development, define the system that the customer really wants. To check issues related to requirements, we perform requirements validation. We usually use requirements validation to check error at the initial phase of development as the error may increase excessive rework when detected later in the development process.

In the requirements validation process, we perform a different type of test to check the requirements mentioned in the Software Requirements Specification (SRS), these checks include:

- Completeness checks
- Consistency checks
- Validity checks
- Realism checks
- Ambiguity checks
- Verifiability

The output of requirements validation is the list of problems and agreed on actions of detected problems. The lists of problems indicate the problem detected during the process of requirement validation. The list of agreed action states the corrective action that should be taken to fix the detected problem.

The development of software begins once the requirements document is ready. One of the objectives of this document is to check whether the delivered software system is acceptable. For this, it is necessary to ensure that the requirements specification contains no errors and that it specifies the users requirements correctly. Also, errors present in the SRS will adversely affect the cost if they are detected later in the development process or when the software is delivered to the user. Hence, it is necessary to detect errors in the requirements before the design and development of the software begins. To check all the issues related to requirements, requirements validation is performed. In the validation phase, the work products produced as a consequence of requirements engineering are examined for consistency, omissions, and ambiguity. The basic objective is to ensure that the SRS reflects the actual requirements accurately and clearly. The requirement checklist as follows:

1. Are all requirements consistent?
2. Are the requirements really necessary?
3. Is each requirement testable?
4. Does the requirement model properly reflect the information function and behaviour of the system to be built?

2.3 Functional Requirement

- Users should be able to upload images of sugarcane plants, either through a web interface or a mobile application.
- The system should be capable of identifying and distinguishing between multiple sugarcane diseases.
- The system should support autonomous monitoring of sugarcane fields using drones equipped with cameras.
- The system should include a predictive analytics feature to forecast potential disease outbreaks.
- Develop a mobile application with augmented reality capabilities for on-the-go disease identification.

2.4 Non-Functional Requirement

- The system should be capable of processing and identifying diseases in images within a reasonable timeframe, aiming for real-time or near-real-time performance.
- The system should be scalable to accommodate a growing number of users, an increasing volume of uploaded images, and the addition of new features.
- The system should be available and accessible to users at all times, with minimal downtime for maintenance or updates.

2.5 System Requirements

Requirements specify what features a product should include and how those features should work. They help to define the test criteria, which is vital for verification and validation.

2.5.1 Hardware Requirements

1. RAM: 8GB (min)

2. Laptop
3. Nvidia Graphics card (Min. 2 GB)
4. Ubuntu¿=18

2.5.2 Software Requirements

- GitHub accounts for repository management
- Dockerized models (Basic expertise needed)
- OpenCV
- Convolutional Neural Networks (CNNs)
  a. MaskRCNN
  b. U-Net, etc.
  c. Tensorflow/PyTorch
  d. Python3
- Operating system – windows 7, windows XP, windows 8, windows 10, windows 11, Linux, MacOS
- Browser – Any of Mozilla, Chrome
- Language – python
- Framework- Tensorflow

Chapter 3

SYSTEM MODEL

Software process model is an intellectual demonstration of a process. It presents an explanation of a process. Process models may contain activities that are part of the software process. All software process models work on the five generic framework activities such as communication, planning, modelling, construction, and deployment. Each and every activity has its own functionality. The goal of the process model is to provide guidance for systematically coordinating and monitoring the tasks that must be accomplished in order to achieve the end product and the project objective.

3.1 Process Model

Software process model is an abstract representation of a process. The goal of process model is to provide guidance for systematically coordinating and controlling the tasks that must be performed in order to achieve the end product and the project objective. Incremental model is used as the process model in our system.

3.1.1 Incremental Model

Incremental model in software engineering is a one which associates the elements of waterfall model in an iterative manner. It delivers a series of releases called increments which provide progressively more functionality for the user as each increment is delivered. In the incremental model of software engineering, the waterfall model is frequently applied in each increment. The incremental model applies linear sequences in a required pattern as calendar time passes. Each linear sequence produces an increment in the work.

A, B, C are modules of Software Product that are incrementally developed and delivered. Every subsequent release of a module adds function to the previous release. This process is

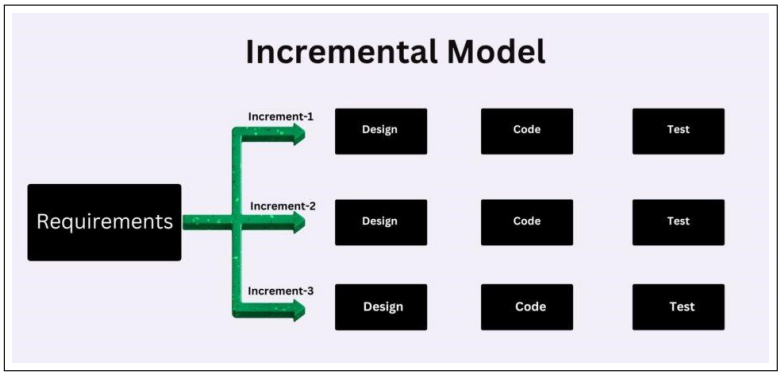


Figure 3.1: Incremental Model

continued until the complete system is achieved

Phases of Incremental Model

1. Requirement analysis: In the very first phase of the incremental model, the product analysis expertise identifies the requirements. And the system functional requirements are understood by the requirement analysis team. To develop the software under the incremental model, this phase performs a crucial role.

2. Design Development: In this phase of the development, the design of the system functionality and the development method are finished with success. When software develops new practicality, the incremental model uses style and development phase.

3. Testing: In the incremental model, the testing phase checks the performance of each and every existing function as well as additional functionality. In the testing phase, the various methods are used to test the behaviour of each task.

4. Deployment: Once the product is tested, it is deployed in the production environment or first User Acceptance Testing (UAT) is done depending on the customer expectation. In the case of UAT, a replica of the production environment is created and the customer along with the developers does the testing.

5. Maintenance: After the deployment of a product on the production environment, maintenance of the product i.e., if any issue comes up and needs to be fixed or any enhancement is to be done is taken care by the developers.

3.1.2 Why to use Incremental Model

We used Incremental model for our system design because of reasons mention below:

- Major requirements must be defined: however, some detail can evolve with time.
- There is a need to get a product to market early.
- The software will be produced quickly during the software life cycle.
- It is flexible and less luxurious to change requirements and scope.
- Though the development stages changes can be done.
- This model is less costly than others.
- A customer can answer back to each building.

3.1.3 Advantages

1. Initial product delivery is faster.
2. Lower initial delivery cost.
3. Core product is developed leading i.e., main functionality is added in the first increment.
4. After each iteration, regression testing should be conducted. During this testing, faulty elements of the software can be quickly recognized because few changes are made within any single iteration.
5. It is generally easier to test and debug than other methods of software development because comparatively smaller changes are made during each iteration. This allows for more targeted and difficult testing of each element within the overall product.
6. With each release, a new article is added to the product.
7. Customers can reply to features and review the product.
8. Risk of changing requirements is reduced.
9. Workload is less.

3.1.4 Disadvantages

1. It requires good planning and designing.
2. Problems might be caused due to system architecture as such not all requirements are collected up front for the entire software lifecycle.
3. Each iteration phase is rigid and does not overlap each other.

Rectifying a problem in one unit requires correction in all the units and consumes a lot of time. It may arise affecting to system architecture because not all necessities are gathered up front for the entire software life cycle.

3.2 Module Details

These are the following modules which will be executed in this system.

1. Users
2. System

3.2.1 User

- Allow users, such as farmers, agronomists, and researchers, to register for an account on the platform.
- Allow users to upload images of sugarcane plants for disease identification.

1. Parameter Input:

   When designing the parameters for the Users module in a sugarcane disease identification system, you need to consider various aspects related to user registration, authentication, roles, and interaction with the platform.

2. Upload Images:

   Image upload functionality with drag-and-drop or file selection. Gallery or history view for users to review their uploaded images.

3. Disease Prediction:

   The output for sugarcane identification using deep learning involves presenting the results of the disease identification process based on the input images.

3.2.2 Admin Module

- Implement a user management system to handle administrator accounts. Include features for adding, removing, and modifying administrator accounts.
- Allow administrators to upload new datasets for retraining the deep learning model.
- nable administrators to monitor the performance of the deep learning model.

1. Data Collection:

   Data collection is the process of gathering and measuring information from countless different sources.

2. Pre-processing:

   Preprocessing involves adding the missing values, the correct set of data, and extracting the functionality. Data set form is important to the process of analysis.

3. Model Building:

   (a) Data Splitting: Data splitting is when data is divided into two or more subsets. Typically, with a two-part split, one part is used to evaluate or test the data and the other to train the model. Data splitting is an important aspect of data science, particularly for creating models based on data.

   (b) Training Prediction Model: Training data is the data you use to train an algorithm or CNN model to predict the outcome you design your model to predict.

   (c) Testing Prediction Model: Test data is used to measure the performance, such as accuracy or efficiency,of the algorithm you are using to train the CNN model.

4. Performance analysis of CNN algorithm:

   The performance of any CNN algorithm may improve with the utilization of a distinct set of features in the same training dataset.

3.3 Project Estimation

In this section various calculation and estimations related to project has been calculated. The figure shows the system modules. The number of lines required for implementation of various modules can be estimated as follows

3.3.1 Estimation in KLOC

Estimation is the process of finding an estimate, or approximation, which is a value that can be used for some purpose even if input data may be incomplete, uncertain, or unstable. Estimation determines how much money, effort, resources, and time it will take to build a Specific system or product. The number of lines required for implementation of various modules can be estimated as follows.

Table 3.1: Estimation of KLOC

Sr.no	Task	Estimated KLOC
1	Data Preprocessing	0.6
2	Training prediction model	0.5
3	Performance Analysis of DL Algorithm	0.6
4	Parameter Input	0.4
5	Uploading Images	0.6
6	Disease prediction	0.6
	Total	3.3

3.3.2 Efforts

The Efforts required in person/month for implementation can be estimated as follows:

$$Effort = a * (LOC)^b$$

Where, $a = 3.2$ and $b = 1.05$.

Hence, we get:

$$Effort = 3.2 * (KLOC)^{1.05}$$

$$\Rightarrow Effort = 3.2 * (3.3)^{1.05}$$

$$\Rightarrow Effort = 11.2 \quad persons/month$$

3.3.3 Development time in months

We know that:

$$Time\ required = \frac{Efforts}{No.\ of\ Developers}$$

$$\Rightarrow Time\ required = \frac{11.2}{4}$$

$$\Rightarrow Time\ required = 2.8 \quad months$$

3.3.4 Total Time Required for Project Development

The total time required can be calculated as follows:

Table 3.2: Time Required for Project Development

Task	**Time Required**
Requirement Analysis and Design	3 months
Implementation and Testing	2.8 months
Total	5.8 months

Hence, total time required is nearly 5.8 months.

3.3.5 Number of Developers (N)

The project is assigned to a group of 4 people (developers). Hence, the number of developers is taken 4.

The developers are as follows:

- D1: Ansari Mohammed Anas.
- D2: Adarsh Borde.
- D3: Pranav Joshi.
- D4: Sanskruti Kekan.

Chapter 4

SYSTEM ANALYSIS

4.1 Project Scheduling and Tracking

Project Preparation and Tracing is important because in order to build a complex system, many software engineering tasks occur in parallel, and the result of work performed during one task may have a reflective effect on work to be conducted in another task. The inter dependencies are very difficult to understand without a detailed schedule.

4.1.1 Project Work and Breakdown Structure (Analysis)

The project work is decomposed into the following work breakdown structure as a part of the analysis phase.

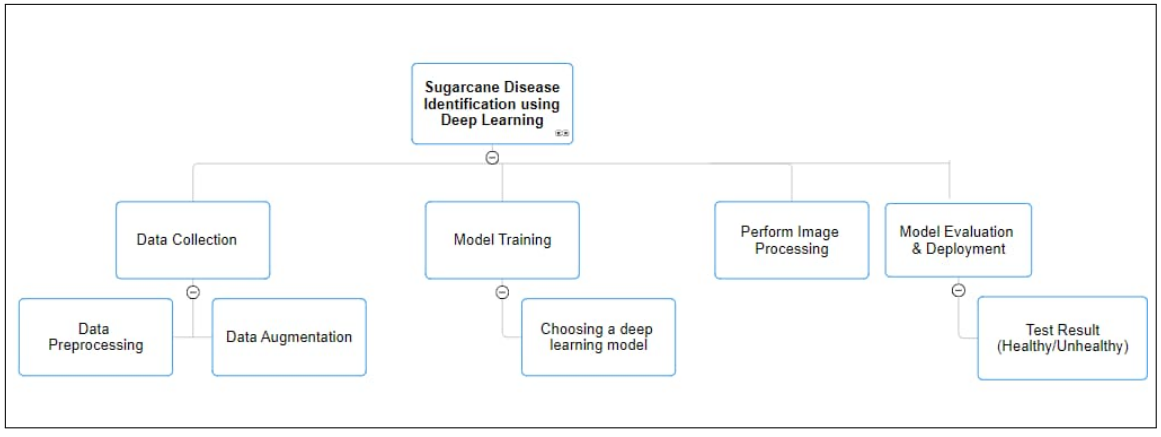


Figure 4.1: System Breakdown Structure

Various tasks have been mentioned in the above diagram.

- **T1 : Communication** Software development process starts with the communication

between customer and developer. According to need of project, we gathered the requirements related to project. Requirement gathering is an important aspect as the developer will come to know what customer expects from the project and also he can help a customer to know more features that can be added to project as he is a technical person. The most important thing needed is that communication should be smooth and clear that means developer should easily understand the demands of customer.

- **T2 : Planning** It includes complete estimation and scheduling (complete time line chart for project development). Before starting the project tasks should be scheduled that means there should be starting and ending date assigned for each and every task and developer should work harder to complete the required task within time chosen at the time of scheduling.

- **T3 : Modeling** It includes detailed requirements analysis and project design (algorithm, flowchart etc). Flowchart shows complete pictorial view of the project and algorithm is step by step solution of problem. Both flowchart and algorithm will be helpful in knowing the overall view of project and serve as a base for development of whole project.

- **T4 : Risk Management** It is a process of identifying, organizing, assessing and controlling threats to some organizations' capitals and earnings which assets overall or partial software product or performance. These threats, or risk, could stem from a wide variety of sources, including financial uncertainty, legal liabilities, strategies, management errors, accidents and natural disasters.

4.2 Project work breakdown structure (Implementation)

Implementation is the stage of the project when the theoretical design is turned out into a working system. Thus, it can be considered to be the most critical stage in achieving a successful new system and in giving the user, confidence that the new system will work and be effective.

Implementation is the stage of the project when the theoretical design is turned out into a working system. Thus, it can be considered to be the most critical stage in achieving a successful new system and in giving the user, confidence that the new system will work and be effective. The modules included in the system are:

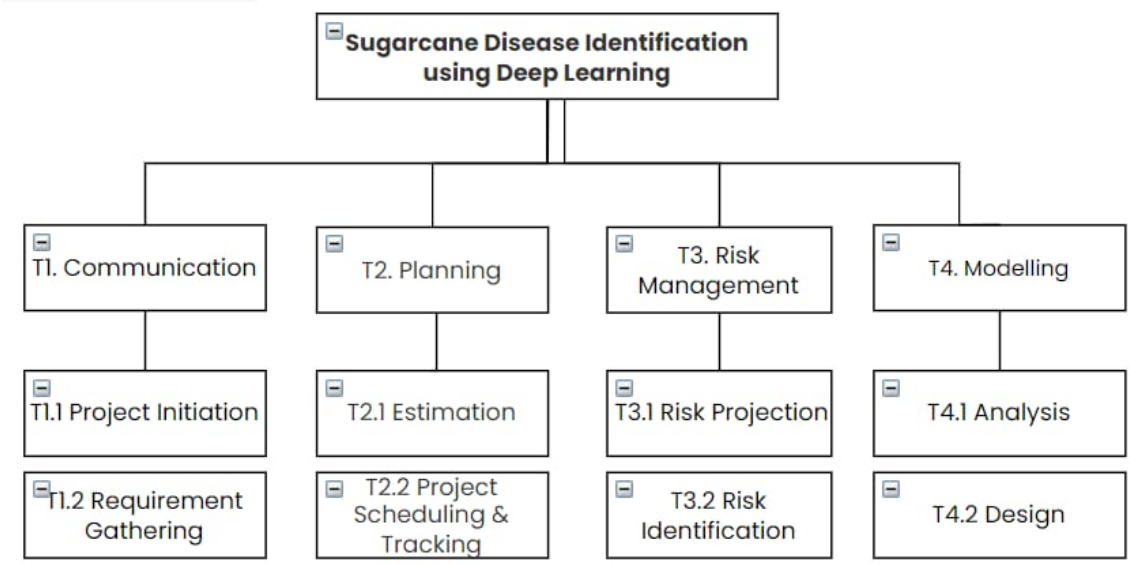


Figure 4.2: Breakdown Structure (Implementation)

User

- Allow users, such as farmers, agronomists, and researchers, to register for an account on the platform.
- Allow users to upload images of sugarcane plants for disease identification.

1. Parameter Input: When designing the parameters for the Users module in a sugarcane disease identification system, you need to consider various aspects related to user registration, authentication, roles, and interaction with the platform
2. Upload Images: Image upload functionality with drag-and-drop or file selection. Gallery or history view for users to review their uploaded images
3. Disease Prediction: The output for sugarcane identification using deep learning involves presenting the results of the disease identification process based on the input images

Admin

- Implement logging and monitoring tools to track the system's usage, performance, and potential errors.

- Implement authentication and authorization mechanisms to ensure that only authorized administrators can access the admin module.

1. Data Collection: Data collection is the process of gathering and measuring information from countless different sources. In order to use the data we collect to develop practical artificial intelligence (AI) and ma- chine learning solutions, it must be collected and stored in a way that makes sense for the business problem at hand.

2. Pre-processing: Preprocessing the data is considered as a significant step in the CNN learning phase. Preprocessing involves adding the missing values, the correct set of data, and extracting the functionality. Data set form is important to the process of analysis. The data collected in this step will be induced in Google Colab platform in the form of python programming in order to get the desired output. In data pre-processing we are going to use the different libraries like pandas, Numpy, matplotlib, tensorflow, keras to perform operations and analyze the data. Using these libraries, we are performing the different significant operation such as Extracting dependent and independent variables, handling the missing data, Feature scaling.

3. Feature Analysis: In this Module we are analyzing feature that how it affects the output on different values. We will analyse all the features.

4. Training Prediction Model: Training data is the data you use to train an algorithm or CNN learning model to predict the outcome you design your model to predict. If you are using supervised learning or some hybrid that includes that approach, your data will be enriched with data labeling or annotation. In order to train CNN learning model, we have to split our data into two parts, splitting of data will be like 70% train set and 30% test set or 80% train set and 20% test set etc. Use the train test split() function in sklearn to split the sample set into a training set, which we will use to train the model, and a test set, to evaluate the model.

5. Testing Prediction Model: Test data is used to measure the performance, such as accuracy or efficiency, of the algorithm you are using to train the CNN. Test data will help you see how well your model can predict new answers, based on its training. Both training and test data are important for improving and validating CNN learning models.

6. Performance analysis of CNN model: Analyzing the performance of a Convolutional Neural Network (CNN) model involves assessing various metrics to understand how well the model is performing on a given task, such as image classification for sugarcane disease identification. Here are key aspects and metrics for performance analysis:

   (a) Confusion Matrix:

   A confusion matrix provides a detailed breakdown of true positives, true negatives, false positives, and false negatives. It helps in understanding where the model is making errors.

   (b) Accuracy:

   Accuracy:

   Accuracy is a fundamental metric that measures the overall correctness of the model. It is the ratio of correctly predicted instances to the total instances.

   (c) Recall (Sensitivity):

   Recall, also known as sensitivity or true positive rate, measures the ability of the model to capture all the relevant instances. It is the ratio of true positive predictions to the sum of true positives and false negatives.

   (d) DRobustness to Perturbations:

   Evaluate how well the model performs when faced with slight variations or distortions in the input data. This is particularly important for real-world scenarios where input data may not be perfectly clean.

4.3 Task Identification

Following analysis and design tasks are to be carried out in the process of analysis and design of the project. All project modules are divided into following tasks.

- T1: Project Definition Searching
- T2: Project Definition Preparation
- T3: Literature Collection
- T4: Project Definition Finalization

- T5: Dataset Collection
- T6: Synopsis Preparation
- T7: Requirement Analysis and Validation
- T8: Determine Process Model (Incremental Model)
- T9: System Breakdown Structure
- T10: Project Estimation in KLOC
- T11: Project Scheduling and Tracking
- T12: Analysis Modelling using Behavioral, Functional and Architectural Modelling
- T13: Feasibility Management of Project using Mathematical Modelling
- T14: Risk Analysis, Project Management and Risk Management
- T15: Report Preparation
- T16: Data Preprocessing using Preprocessing Techniques
- T17: Extracting the Features from the Preprocessed Dataset
- T18: Creating Graphical User Interface
- T19: Model Training using CNN model.
- T20: Predicting whether the Sugarcane Leaf have disease or not.
- T21: Testing of the Generated Model
- T22: Deployment of Project

4.3.1 Project Schedule

Table 4.1 describes the schedule for project development and also highlight all the task to be carried out along with their duration, dependency and developer(s) assign to accomplish the task.

Table 4.1: Project Task Table

Task	Days	Dependencies	Developer Assigned
T1	5	-	D1,D2,D3,D4
T2	7	T1	D1,D2,D3,D4
T3	3	T2	D1,D2,D3,D4
T4	6	T1, T2,T3	D1,D2,D3,D4
T5	3	T2	D1,D2,D3,D4
T6	8	T3,T4	D1,D2,D3,D4
T7	9	T6	D1,D2,D3,D4
T8	5	-	D1,D2,D3,D4
T9	4	T5	D1,D2,D3,D4
T10	3	-	D1,D2,D3,D4
T11	3	-	D1,D2,D3,D4
T12	4	T7, T8, T9	D1,D2,D3,D4
T13	5	T11	D1,D2,D3,D4
T14	6	T8	D1,D2,D3,D4
T15	14	T7,T8	D1,D2,D3,D4
T16	14	T14	D1,D2,D3,D4
T17	12	T15	D1,D2,D3,D4
T18	20	-	D1,D2,D3,D4
T19	15	T17	D1,D2,D3,D4
T20	15	T18	D1,D2,D3,D4
T21	7	T18, T19	D1,D2,D3,D4
T22	4	T15,T19,T21	D1,D2,D3,D4
Total	**172**		

4.4 Project Table and Time-line Chart

4.4.1 Project Schedule Time

Table 4.2: Project Schedule Time Table

Test ID	Exp. Start Time	Act. Start Time	Exp. End Time	Act. End Time	Developers
T1	22/08/23	22/08/23	25/08/23	25/08/23	D1,D2,D3,D4
T2	26/08/23	26/08/23	30/08/23	30/08/23	D1,D2,D3,D4
T3	01/09/23	01/09/23	04/09/23	04/09/23	D1,D2,D3,D4
T4	05/09/23	05/09/23	09/09/23	10/09/23	D1,D2,D3,D4
T5	10/09/23	11/09/23	13/09/23	13/09/23	D1,D2,D3,D4
T6	14/09/23	14/09/23	20/09/23	21/09/23	D1,D2,D3,D4
T7	20/09/23	21/09/23	30/09/23	30/09/23	D1,D2,D3,D4
T8	01/10/23	01/10/23	06/10/23	07/10/23	D1,D2,D3,D4
T9	07/10/23	08/10/23	12/10/23	12/10/23	D1,D2,D3,D4
T10	13/10/23	13/10/23	16/10/23	16/10/23	D1,D2,D3,D4
T11	17/10/23	17/10/23	21/10/23	21/10/23	D1,D2,D3,D4
T12	22/10/23	22/10/23	28/10/23	28/10/23	D1,D2,D3,D4
T13	29/10/23	29/10/23	04/11/23	04/11/23	D1,D2,D3,D4
T14	05/11/23	05/11/23	09/11/23	09/11/23	D1,D2,D3,D4
T15	10/11/23	10/11/23	14/11/23	14/11/23	D1,D2,D3,D4
T16	15/11/23	15/11/23	19/11/23	19/11/23	D1,D2,D3,D4
T17	20/11/23	20/11/23	23/11/23	23/11/23	D1,D2,D3,D4
T18	23/11/23	23/11/23	25/11/23	25/11/23	D1,D2,D3,D4
T19	26/11/23	26/11/23	28/11/23	28/11/23	D1,D2,D3,D4
T20	28/11/23	28/11/23	30/11/23	30/11/23	D1,D2,D3,D4
T21	27/12/23	27/12/23	21/02/24	28/02/24	D1,D2,D3,D4
T22	22/02/24	01/03/24	10/04/24	10/04/24	D1,D2,D3,D4

4.4.2 Time-Line Chart

Timeline chart shows the progress of project development in various phases.

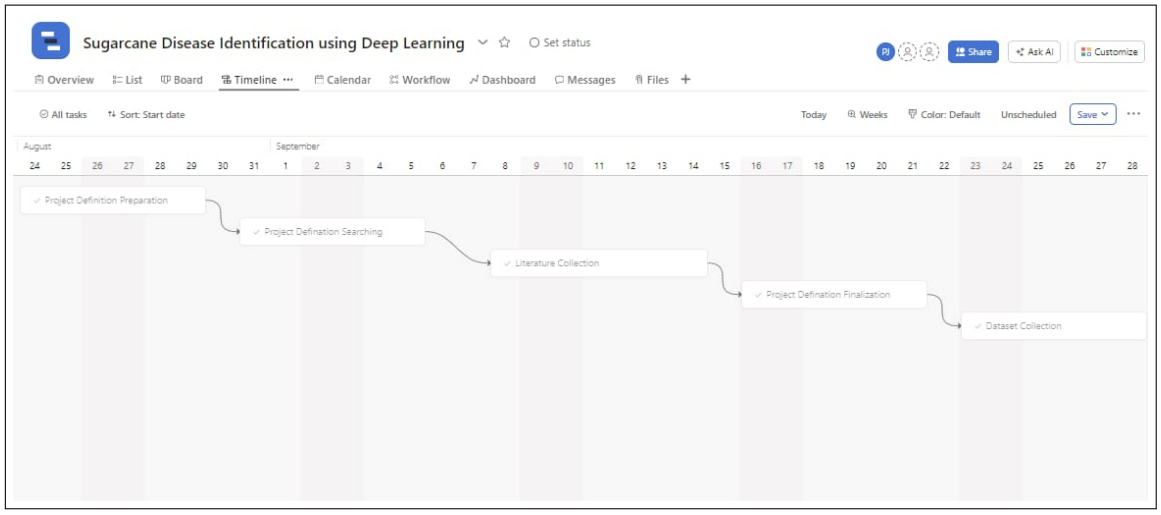


Figure 4.3: Expected Timeline Chart for Sem-1

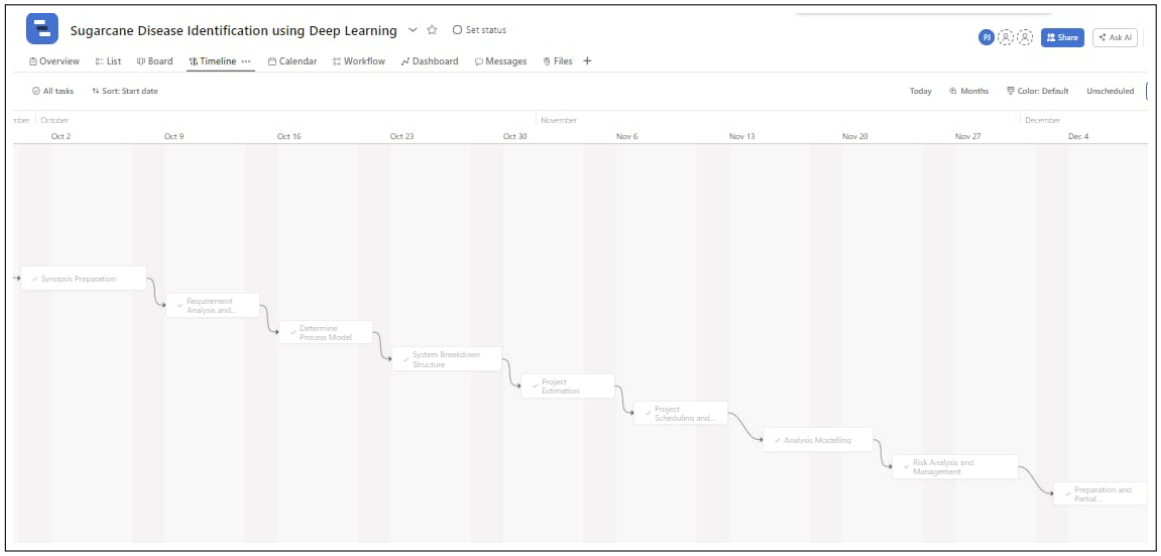


Figure 4.4: Expected Timeline Chart for Sem-2

4.5 Analysis Modeling

The system analysis model is made up of class diagram, sequence or collaboration diagrams and state chart diagrams. Between them they constitute a logical, implementation free view of computer system that includes a detail definition of every aspect of functionality. Analysis model contains following modeling:

1. Behavioral modeling.

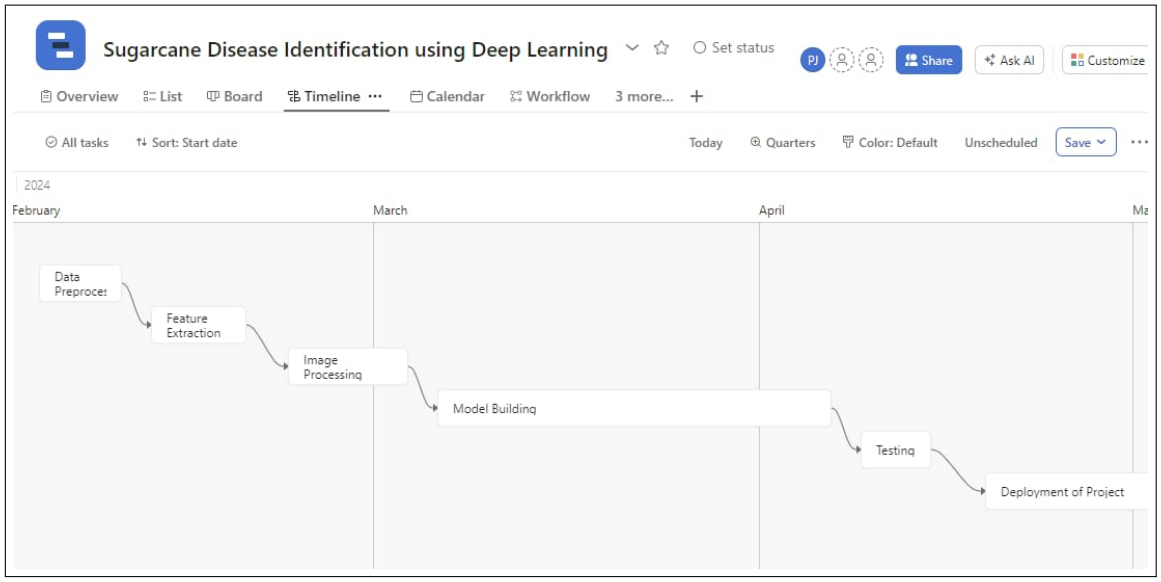


Figure 4.5: Expected Timeline Chart for Sem-2

2. Functional modeling.

3. Architectural modeling.

Analysis modeling uses a combination of text and diagrammatic form to depict requirement for data, function and behaviour in a way that is relatively easy to understand and more important, straightforward to review for correctness, completeness and consistency.

4.5.1 Behavioral modeling

Use case diagram

A use case involves a sequence of interactions between the motivator and the system, possibly including other actors.

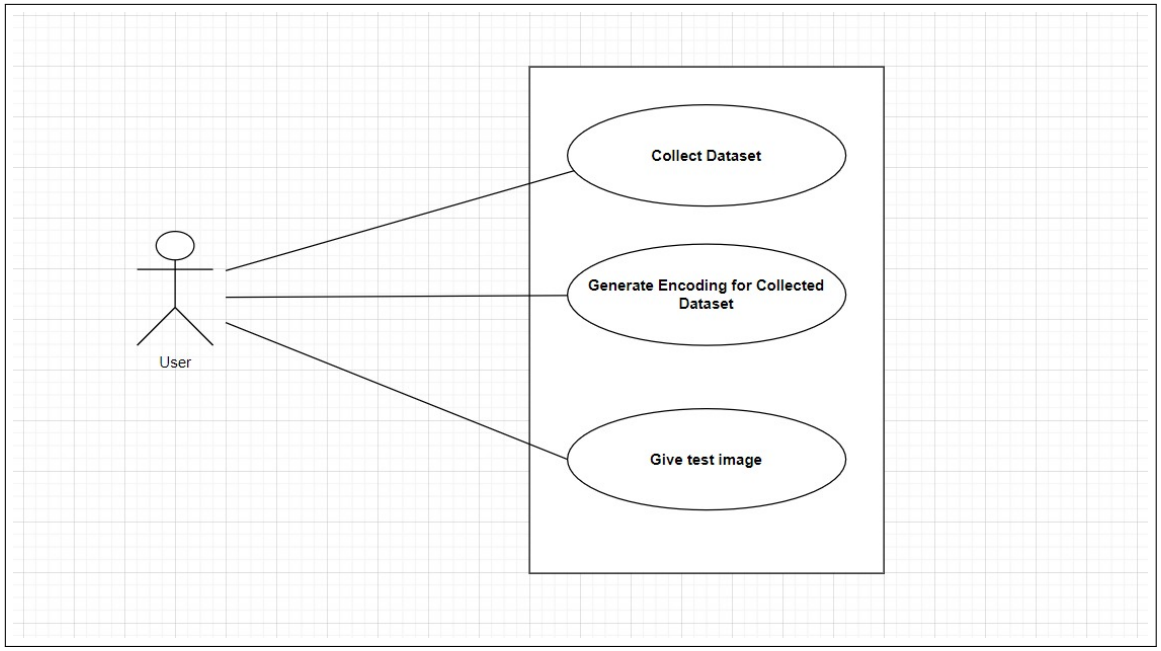


Figure 4.6: Use Case Diagram

Sequence Diagram

A sequence diagram is a graphical view of a state that shows object interface in a time-based sequence.

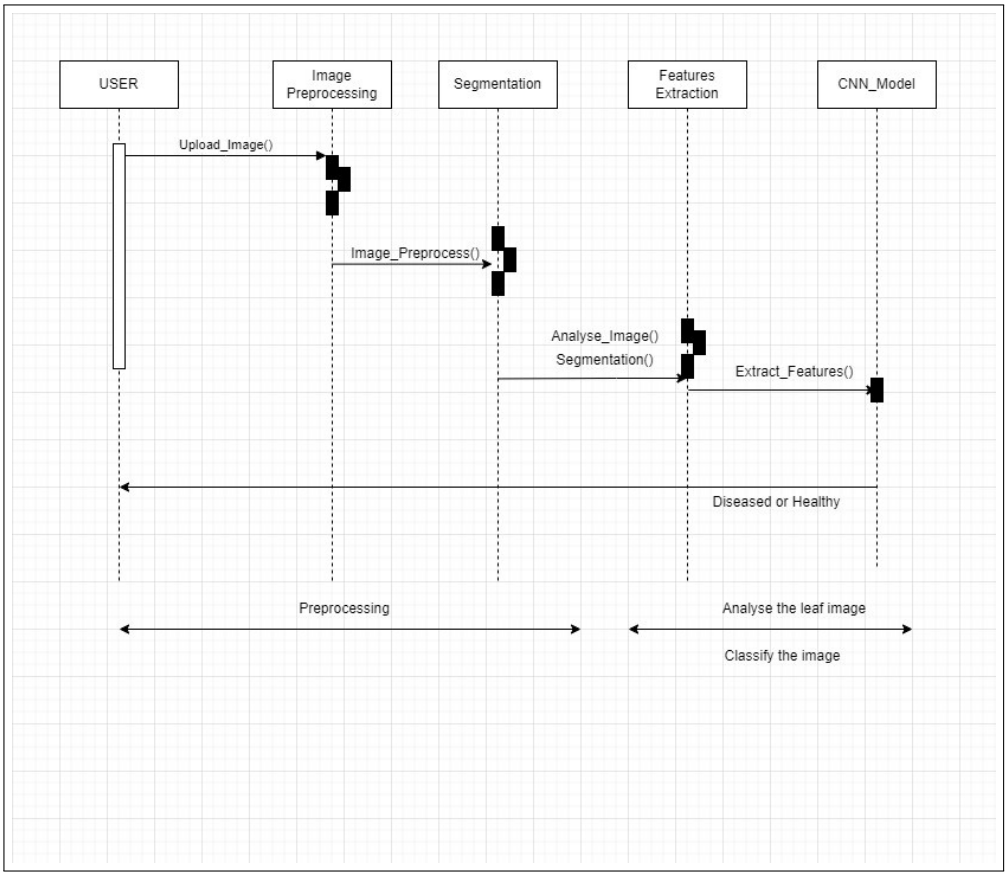


Figure 4.7: Sequence Diagram

Class Diagram

A Class is an Explanation of a set of objects that have the same attributes and methods Attribute.

- Attribute
- Operation
- Relationship

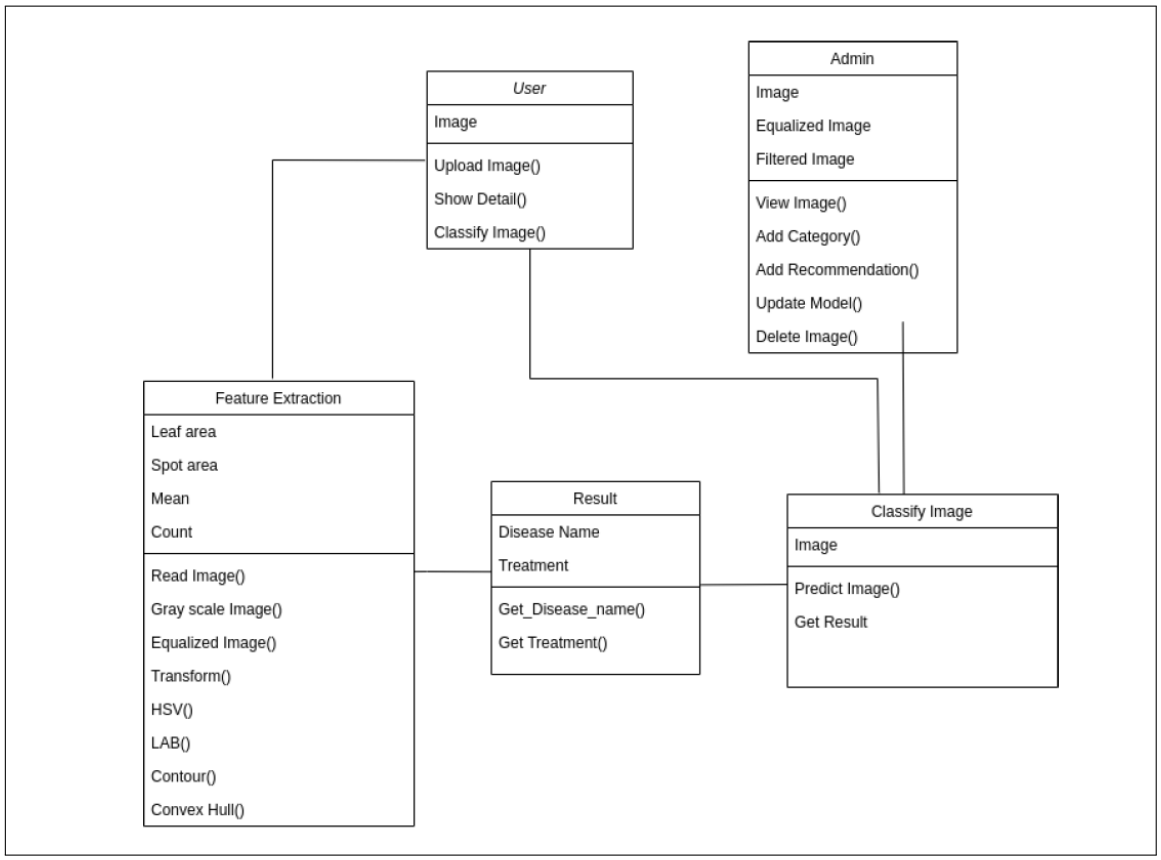


Figure 4.8: Class Diagram

Activity Diagram

Activity diagram is an important diagram to describe the dynamic aspects of the system. Activity diagram is essentially a flowchart to represent the flow from one activity to another activity.

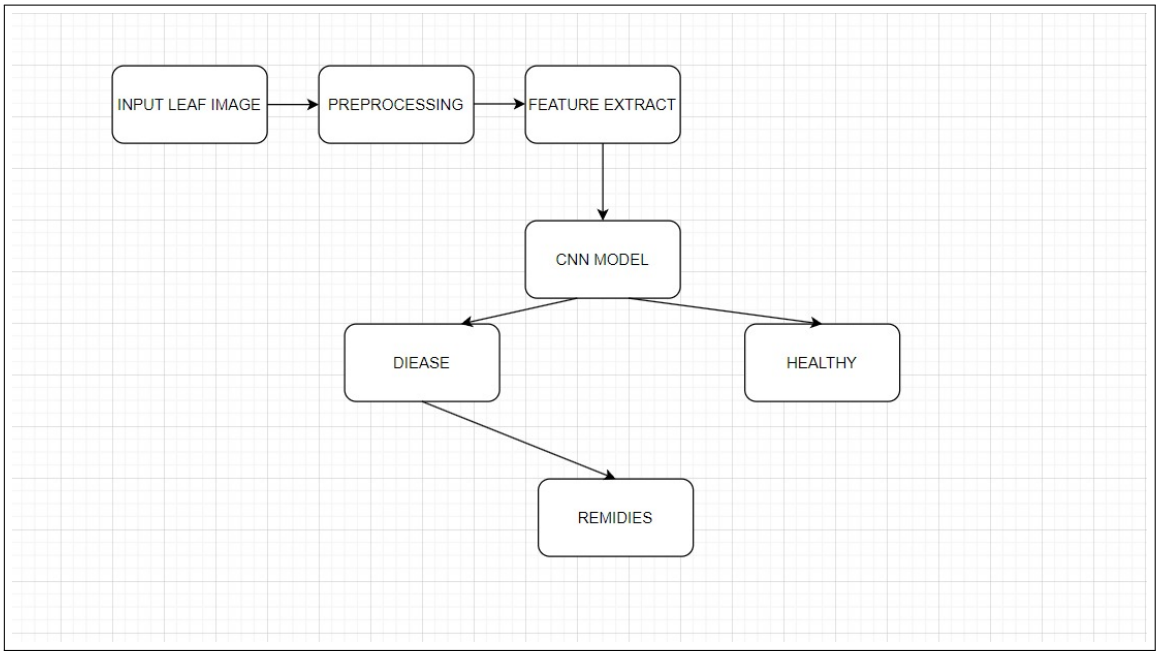


Figure 4.9: Activity Diagram

State Chart Diagram

State chart Diagram shows the state CNN that consists of states, transitions, events and activities. This diagram shows how the system makes the transition from one state to another state on occurrence of a particular event.

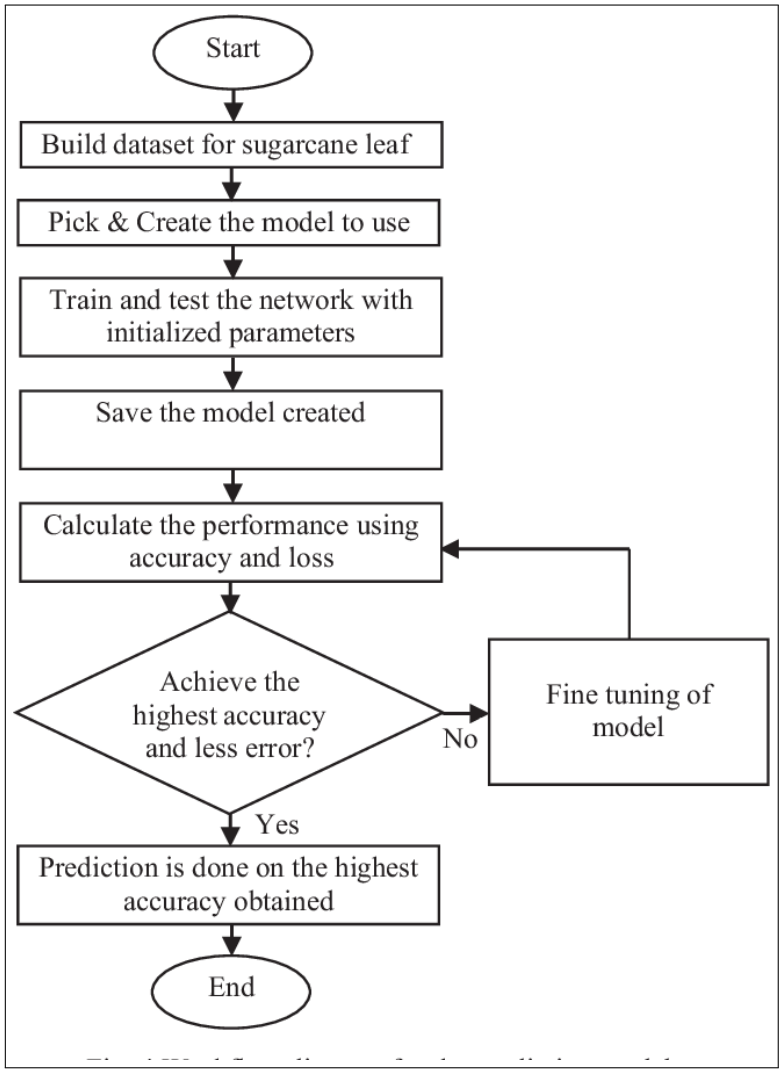


Figure 4.10: State Chart Diagram

4.5.2 Functional Modeling

Data Flow Diagram

Data Flow Diagram: Data flow diagram is also called Bubble Chart is a graphical technique, which is used to represent information flow, and converters are applied when data moves from input to output. DFD represents system requirements clearly and identifies modifiers that become programs in design. DFD may be further separated into different levels to show detailed information flow.

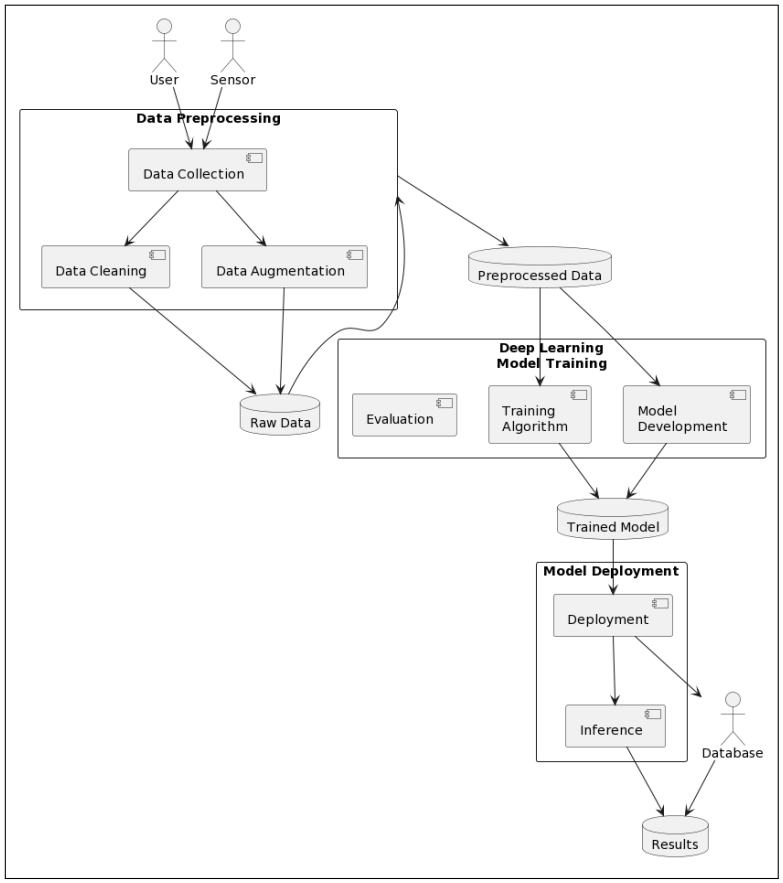


Figure 4.11: Data Flow Diagram level 0

Control Flow Diagram

The large class of applications having following characteristics requires control flow modeling. The applications that are driven by events rather than data. The applications that produce control flow material rather than reports or Displays.

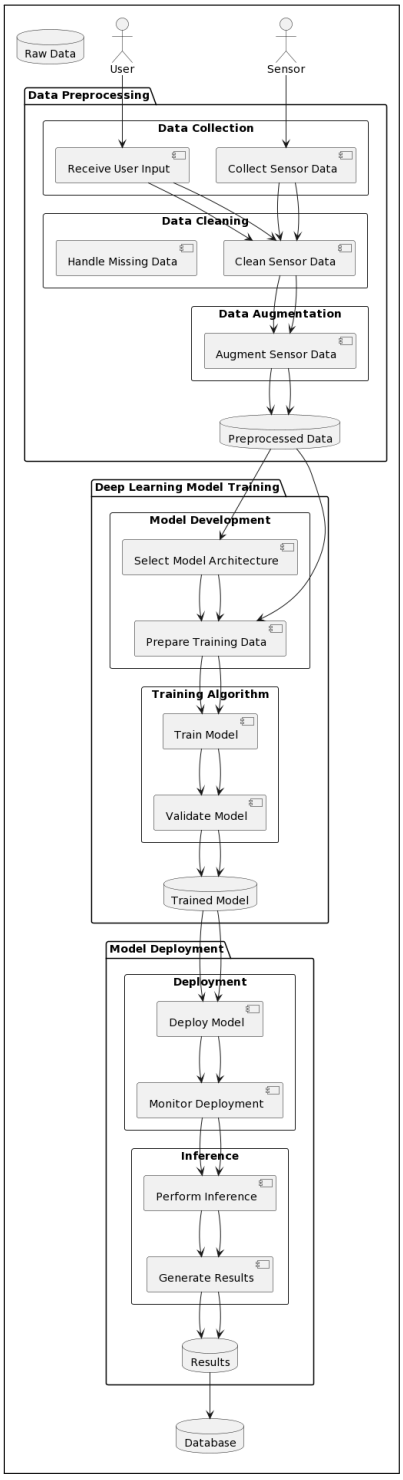


Figure 4.12: Data Flow Diagram level 1

4.5.3 Architectural Modeling

Component Diagram

A Component diagram shows a set of components with their relationships. The component diagrams are useful to represent the static implementation view of a system.

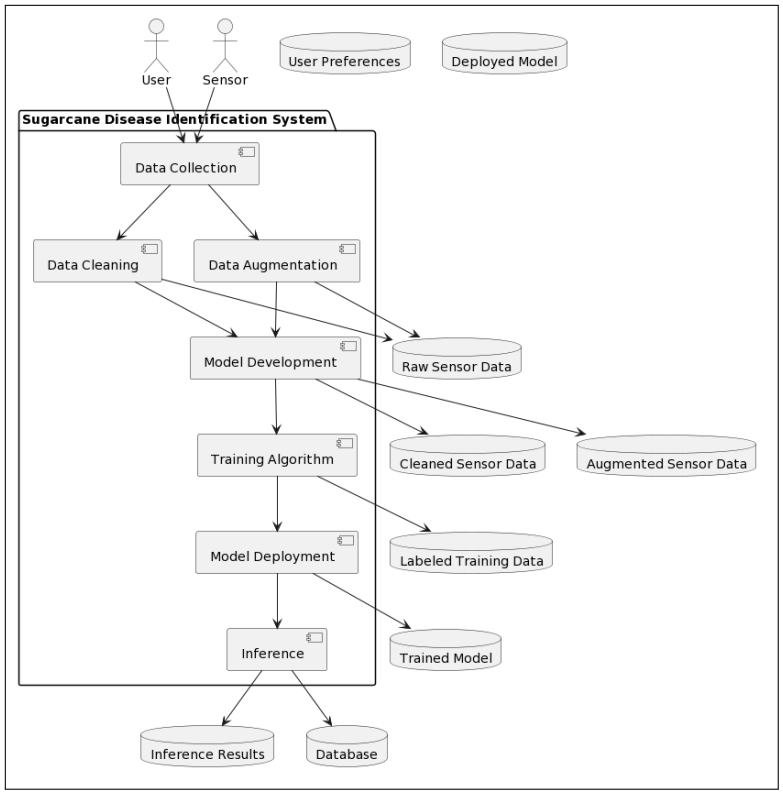


Figure 4.13: Data Flow Diagram level 0

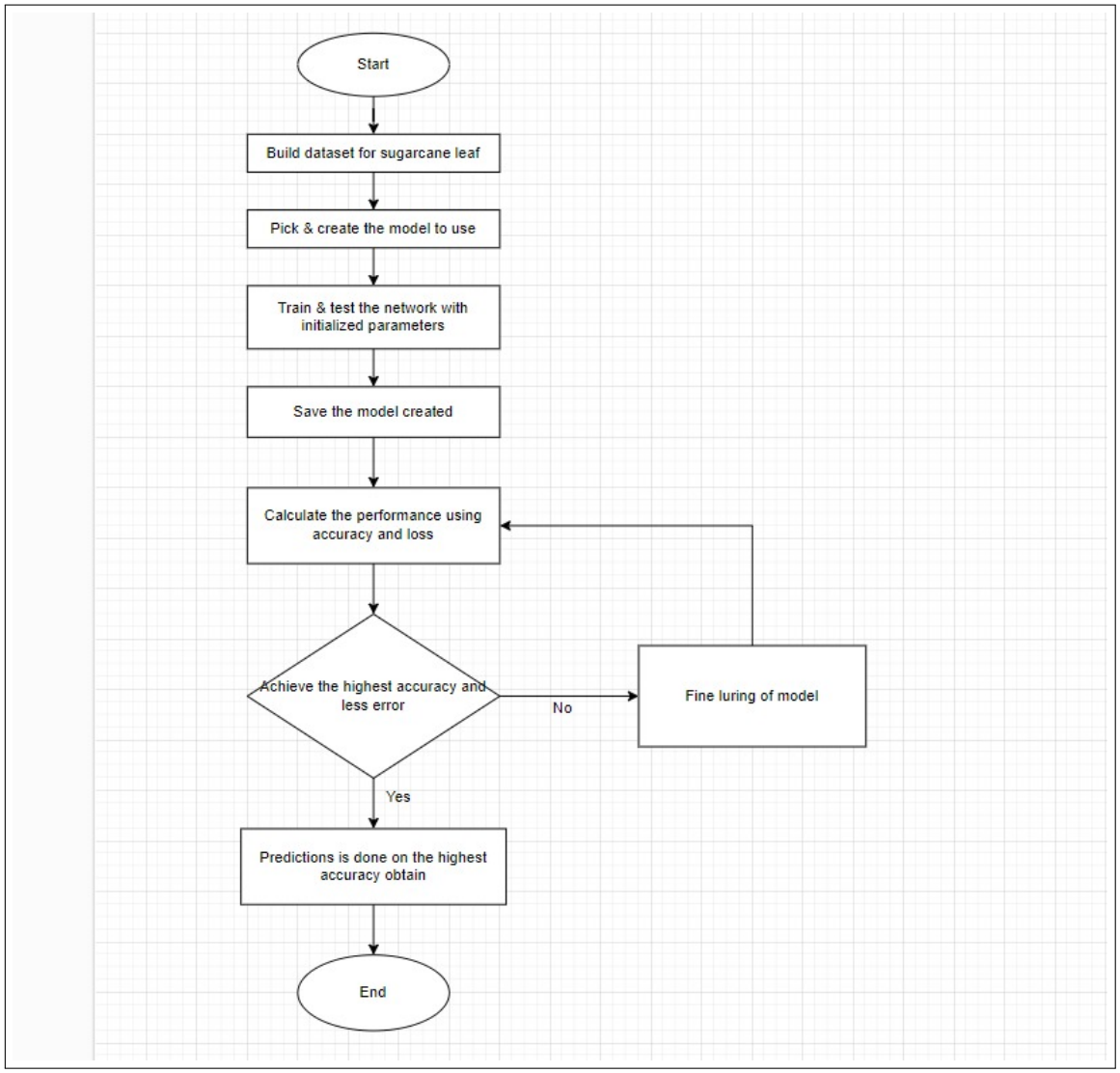


Figure 4.14: Control Flow Diagram

Deployment Diagram

Deployment figures are used to visualize the topology of the physical components of a system where the software components are organized. Deployment diagrams are used for describing the hardware components where software components are deployed.

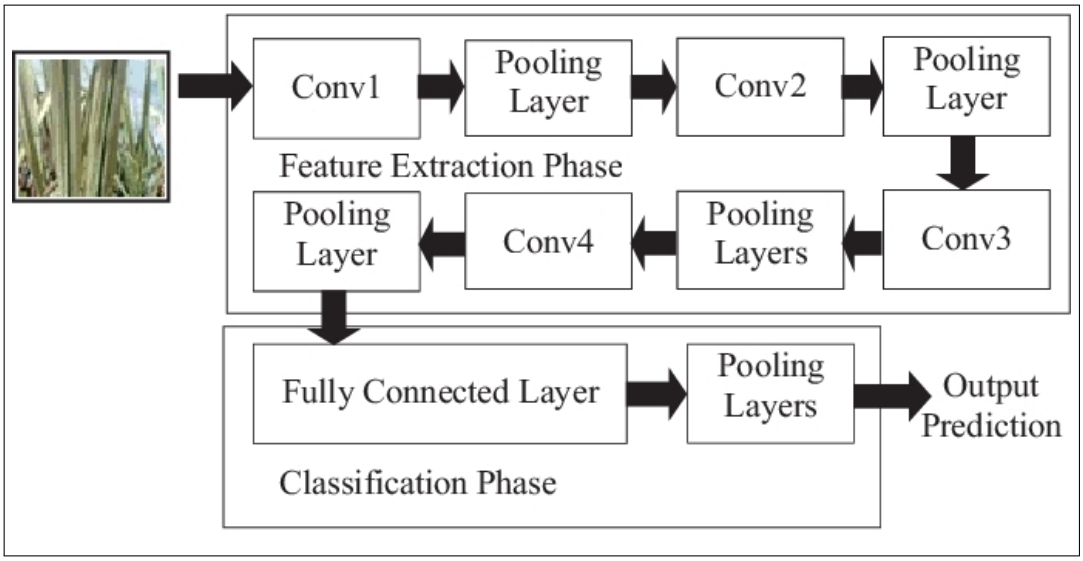


Figure 4.15: Component Diagram

4.6 Mathematical Modeling

NP-Hard Problem

These problems need not have any certainty on their running time. Intuitively, these are the problems that are even harder than the NP- complete problems Note that NP-hard problems do not have to be in NP, and they do not have to be decision problems. The precise definition here is that a problem X is NP- hard, if there is an NP-complete problem Y, such that Y is reducible to X in polynomial time. But since any NP- complete problem can be reduced to any other NP-complete problem in polynomial time, all NP-complete problems can be reduced to any NP-hard problem in polynomial time. Then, if there is a solution to one NP-hard problem in polynomial time, there is a solution to all NP problems in polynomial time. From the above theory we can conclude that our problem definition comes under the 'P' class.

Set Theory

Set theory is a branch of calculated logic that studies set, which informally are collections of objects. Although any type of object can be composed into a set, set theory is applied most often to objects that are appropriate to mathematics. The language of set theory can be used in the meanings of nearly all mathematical objects.

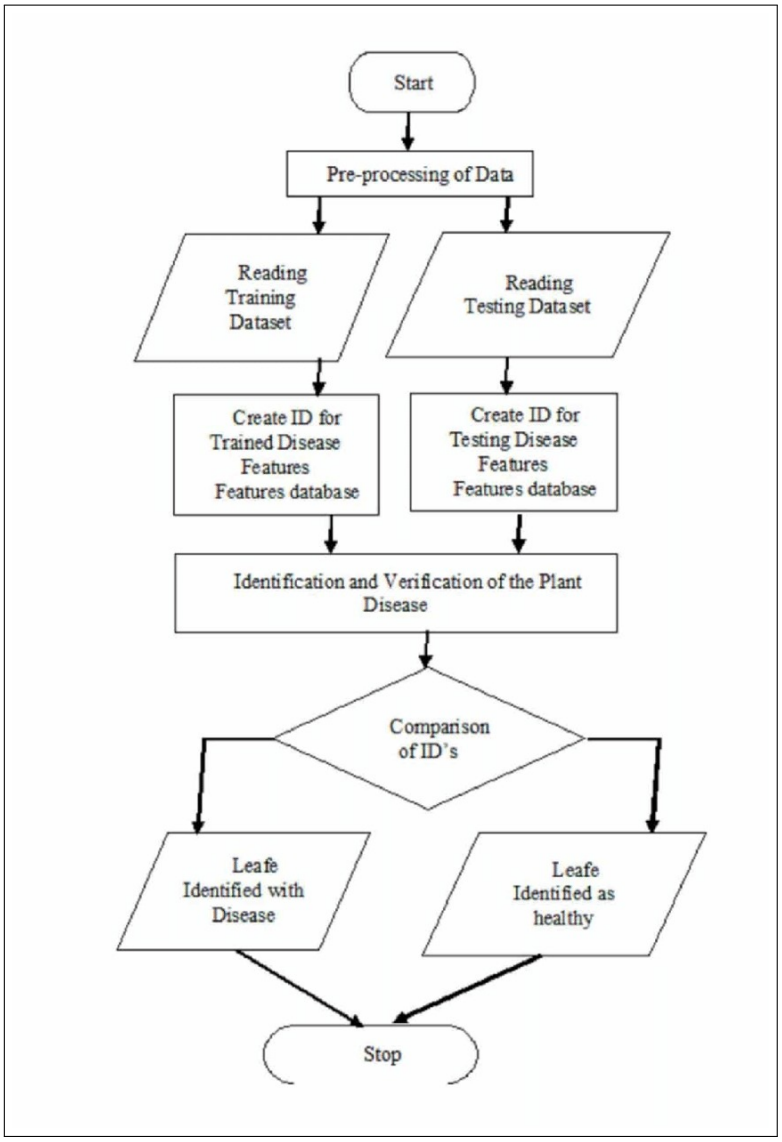


Figure 4.16: Deployment Diagram

System representation is given as,

$$S = \{I, P, R, O\}$$

Where,

$\{I\}$ is set of all inputs given to system.

$\{P\}$ is set of all processes in system.

$\{R\}$ is set of rules that drives your input set.

$\{O\}$ is set of output expected from system.

Input $\{I\}$ is set of inputs giving to system.

$$I = \{I1\}$$

Where,

I1 = Required user information.

Process

$$P = \{P1, P2, P3, P4\}$$

Where,

P1 = Taking input from user.

P2 = Storing users information in database.

P3 = Predicting IPL Matches Score and Win/Loss using ML algorithm.

P4 = Storing result in database. P5 = Display result to user.

Rules

$$R = \{R1\}$$

Where,

R1 = User must provide all the required input data.

Output

$$O = \{O1\}$$

Where,

O1 = Provide generated result to the user.

Chapter 5

RISK MANAGEMENT

Project risk management is the process of identifying, analyzing and then responding to any risk that arises over the life cycle of a project to help the project remain on track and meet its goal. Managing risk isn't reactive only, it should be part of the planning process to figure out risk that might happen in the project and how to control that risk if it in fact occurs. A risk is anything that could potentially impact your projects timeline, performance or budget. Risks are potentialities, and in a project management context, if they become realities, they then become classified as issues that must be addressed. So risk management, then, is the process of identifying, categorizing, prioritizing and planning for risks before they become issues .This article gives us ten golden rules to apply risk management successfully in our project.

5.1 Risk Identification

- **Product size risk**

  R1: Is the development team able to decompose the required program into smaller, highly cohesive modules.

  R2: Are there enough development team members available relative to the size of the project.

- **Business impact**

  R3: Delay in project delivery (violation in time constraints) can hamper the customer economically.

  R4: If system is not more efficient than the existing system, it will cause economic losses.

- **Customer related risk**

  R5: User or service provider is a non-technical person; if proper guide- lines were not mentioned then it will create ambiguity.

  R6: If a user wants any modifications that lead to changing the entire System.

- **Process risk** R7: The risk of technology errors or security incidents that disrupt or invalid processes.

  R8: Quality of a process itself that leads to failures. A low quality process may not properly work and may break down the system.

- **Technical Risk**

  R9: Lack of database stability and concurrency.

  R10: Module integration fails.

  R11: Wrong trained data may lead to wrong results.

- **Development Environment Related risk**

  R12: Lack of proper training and less knowledge of programming leads to a moderate risk. It will delay product development and deployment.

5.2 Strategies Used to Manage Risk

- S1: Formulation and follow up of the project plan on a regular basis.
- S2: Keep assigned work under certain deadlines.
- S3: Web Development cycle should be used.
- S4: Regular meetings with users reduce the risk to some extent, design systems with flexibility and maintain necessary documentation for the same.
- S5: Re-defined software process at higher degree.
- S6: Proper training on required technical tools for development of projects reduces risk.
- S7: Make certain rules that each one the members are taking part in the design.

- S8: Study and understanding of project definition, programming language.
- S9: Take a look at model development and all its associated used software.
- S10: Time constraints must be followed to avoid economical risks.
- S11: Each and every module must be tested for its functioning.
- S12: After unit testing, the system must be integrated and validated accordingly.
- S13: Integration testing for authentication hierarchy.
- S14: Use of standard database technology which supports concurrency more.

5.3 Risk Projection

5.3.1 Preparing Risk Table

The risk table shown below lists all possible risks which may occur at any stage during development of a project. Table also clearly shows the impact of the risks and RMMM (Risk Mitigation Monitoring and Management) planto deal with any such risks.

Table 5.1: Risk Management Table

Risk	Category	Probability	Impact	Plan
R1	Product Size Risk	More	High	S1,S3,S4
R2	Product Size Risk	Less	Less	S4,S6,S1
R3	Business Impact Risk	More	High	S2
R4	Business Impact Risk	More	High	S4
R5	Customer Related Risk	More	High	S11,S12,S13
R6	Customer Related Risk	More	Less	S11,S12,S13
R7	Process Risk	More	High	S14
R8	Process Risk	More	High	S14
R9	Technical Risk	Less	High	S1,S6
R10	Technical Risk	More	High	S7
R11	Development Environment Related Risk	Less	Less	S4,S5,S7,S8
R12	Product Size Risk	More	High	S9,S10

5.4 Feasibility

Feasibility is defined as an evaluation or analysis of the potential impact of a proposed project.

- Technical feasibility: - It is disturbed with specifying equipment and software that will successfully preserve the task required.
- SAT (Satisfiability): - Boolean formula is satisfiability if there exists at least one way of assigning value to its variable so as to make it true and we denote it by using SAT. The problem of deciding whether a given formula is satisfiability or not.
- Facility to produce output in given times.
- Response time under certain conditions.
- Operational Feasibility: -It is related to human organization.
- What changes will be brought in with the system.
- How organizational Structure will be distributed.
- What new skills are required.
- Economic Feasibility: -It is the most frequently used technique for evaluating the effectiveness of proposed systems. Most usually identified as cost/benefit analysis.

Chapter 6

TECHNICAL SPECIFICATION

6.1 Software requirement specification

- GitHub accounts for repository management
- Dockerized models (Basic expertise needed
- OpenCV
- Convolutional Neural Networks (CNNs) such as MaskRCNN, Tensorflow/PyTorch, Python3

6.2 Hardware Requirement Specifications

- Hard Disk: 120 GB
- Ram: 8GB
- Nvidia Graphics card (Min. 2 GB)
- Ubuntu ¿=18

Chapter 7

IMPLEMENTATION DETAILS

In this chapter implementation details are explained as follow:

Code of data preprocessing for Sugarcane Disease Identification Using Deep Learning

```
In [ ]: # Downloading Necessary Libraries
        !pip install tensor-dash

In [ ]: from google.colab import drive
        drive.mount('/content/drive')

        Mounted at /content/drive

In [ ]: !ls

        drive  sample_data

In [ ]: # Reading the Training Data
        dataset = pd.read_csv('/content/drive/MyDrive/Colab Notebooks/plant/train.csv')

In [ ]: # Importing Necessary Libraries
        import tensorflow as tf
        import tensorflow.keras as keras
        import matplotlib.pyplot as plt
        from sklearn.model_selection import train_test_split
        from tensordash.tensordash import Tensordash
        import plotly.express as px
        import json
        import skimage.io as io
        #user_secrets = UserSecretsClient()
        #secret_value_0 = user_secrets.get_secret("email")
        #secret_value_1 = user_secrets.get_secret("pin")

In [ ]: dataset
```

Figure 7.1: Preprocessing

Library:

1. import tensorflow as tf: Imports the TensorFlow library, which is a popular open-source deep learning library developed by Google.

2. import tensorflow.keras as keras: Imports the Keras API, which is now integrated into TensorFlow. Keras is a high-level neural networks API that simplifies the process of building and training deep learning models.

3. import matplotlib.pyplot as plt: Imports Matplotlib, a data visualization library, to create plots and charts.

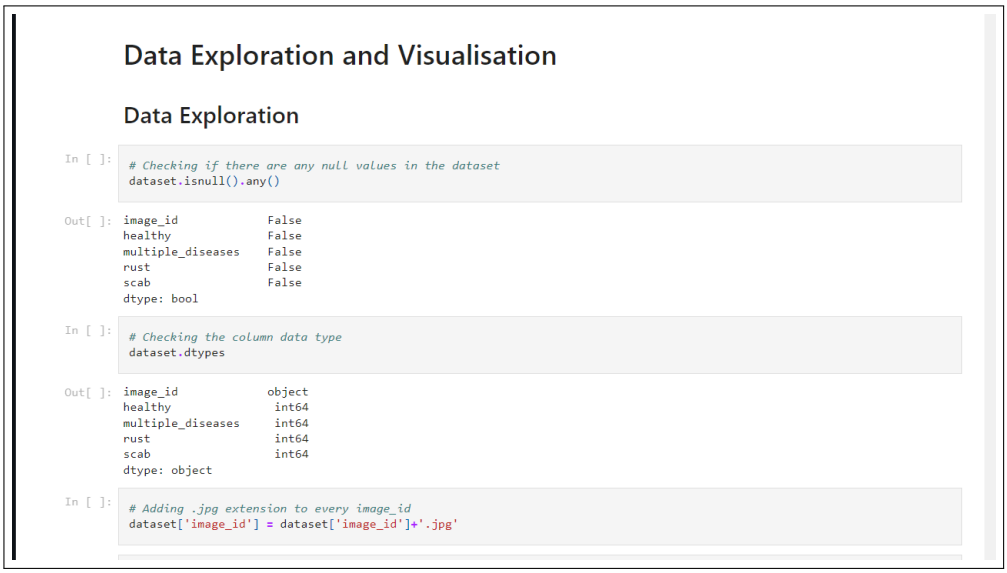
4. from sklearn.model selection import train test split: Imports the train test split function from scikit-learn, which is commonly used to split datasets into training and testing sets.

5. from tensordash.tensordash import Tensordash: Attempts to import the Tensordash module for TensorBoard integration. Note that the Tensordash module needs to be installed for this import to work.

6. import plotly.express as px: Imports Plotly Express, a high-level interface for creating interactive plots.

7. import json: Imports the JSON module, which provides methods for working with JSON data.

8. import skimage.io as io: Imports the image processing module io from scikit-image.



Data Exploration and Visualisation

Data Exploration

```
In [ ]: # Checking if there are any null values in the dataset
        dataset.isnull().any()

Out[ ]: image_id            False
        healthy             False
        multiple_diseases   False
        rust                False
        scab                False
        dtype: bool

In [ ]: # Checking the column data type
        dataset.dtypes

Out[ ]: image_id            object
        healthy              int64
        multiple_diseases    int64
        rust                 int64
        scab                 int64
        dtype: object

In [ ]: # Adding .jpg extension to every image_id
        dataset['image_id'] = dataset['image_id']+'.jpg'
```

Figure 7.2: Data Exploration

Image Visualisation :

A loop is used to iterate through the grid of subplots (4 rows by 4 columns).

1. img = plt.imread: Reads an image file (presumably in JPEG format) using Matplotlib's imread function. The image file path is constructed using the loop variable i.

2. fig.add subplot(rows, columns, i): Adds a subplot to the figure with the specified number of rows and columns and places it at position i. The code then checks the labels of the dataset (dataset.healthy[i], dataset.multiple diseases[i], dataset.rust[i]) to determine the class of the image and sets the title accordingly.

3. plt.imshow(img): Displays the image in the subplot.

4. plt.axis('off'): Turns off the axis labels and ticks for each subplot. Finally, plt.show() is used to display the entire figure containing the grid of images and their corresponding titles.

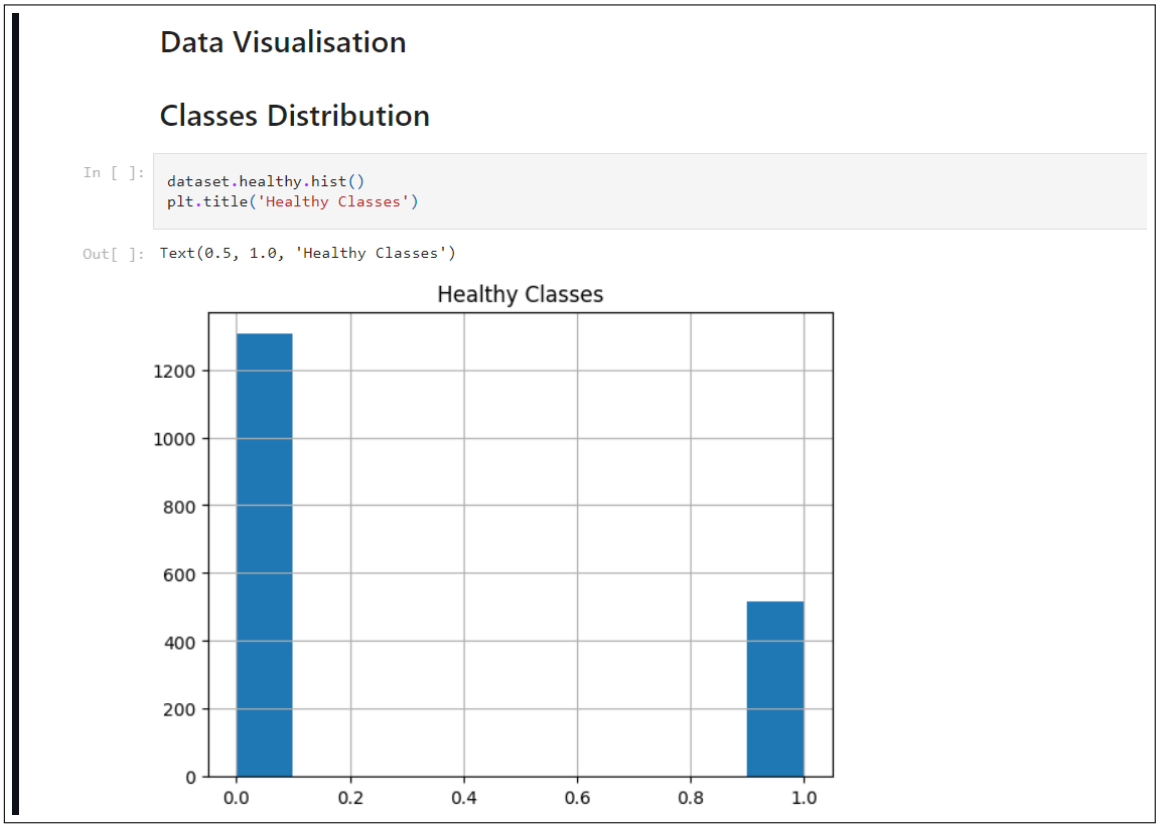


Figure 7.3: Data Visualisation

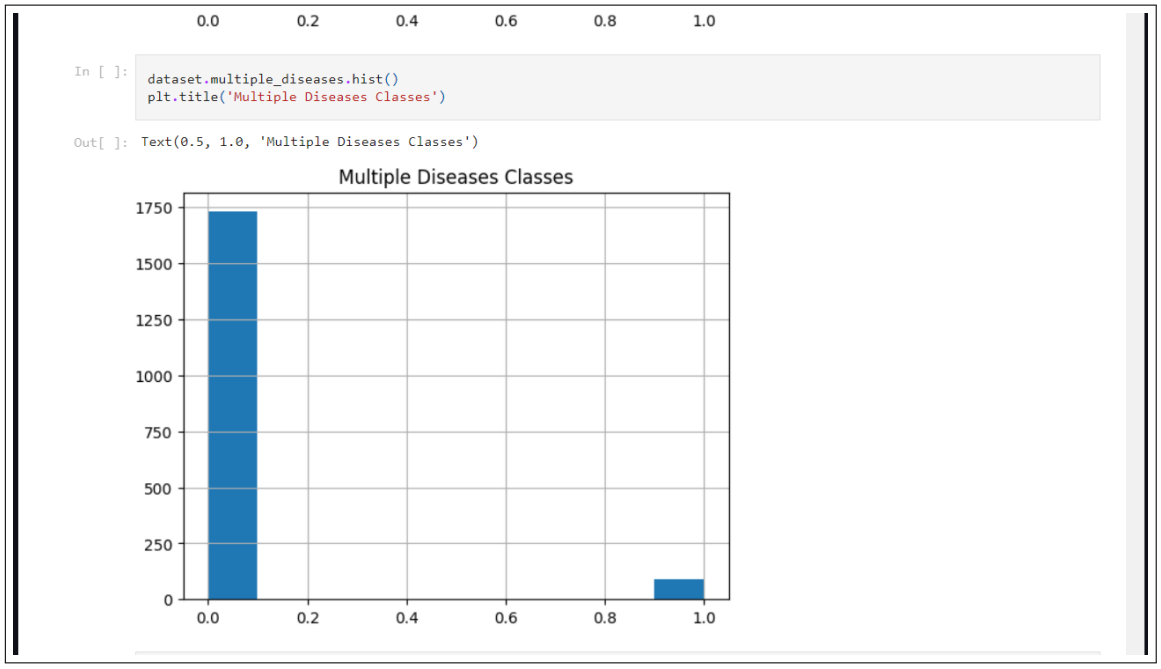


Figure 7.4: Multiple Diseases Classes

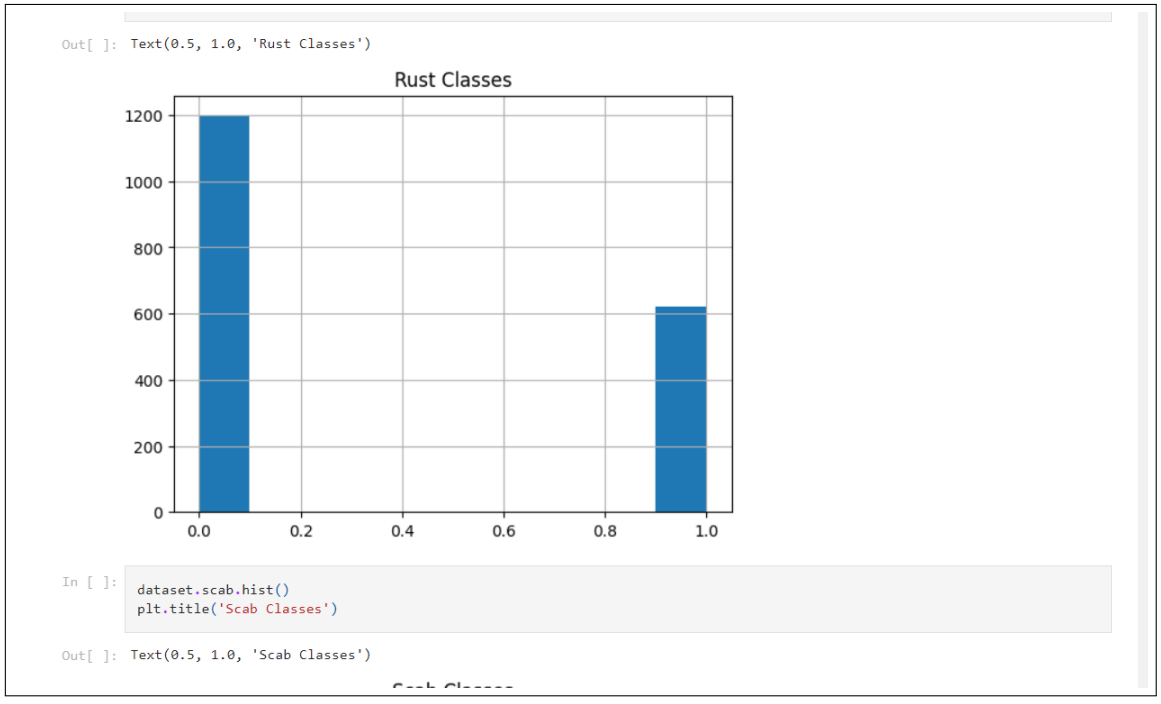


Figure 7.5: Rust Classes

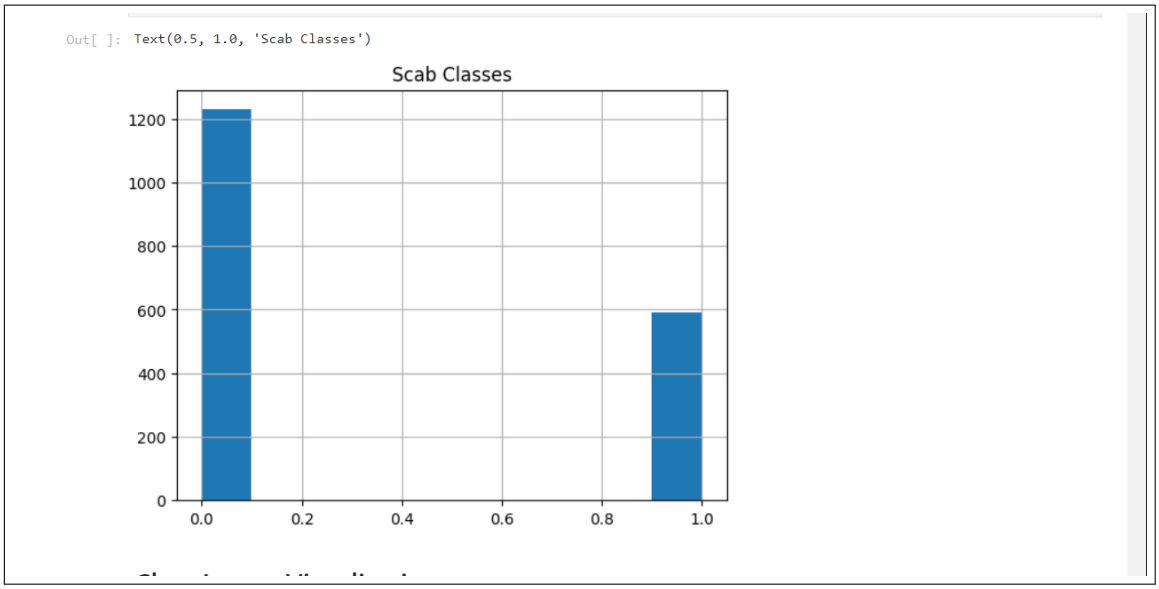


Figure 7.6: Scab Classes

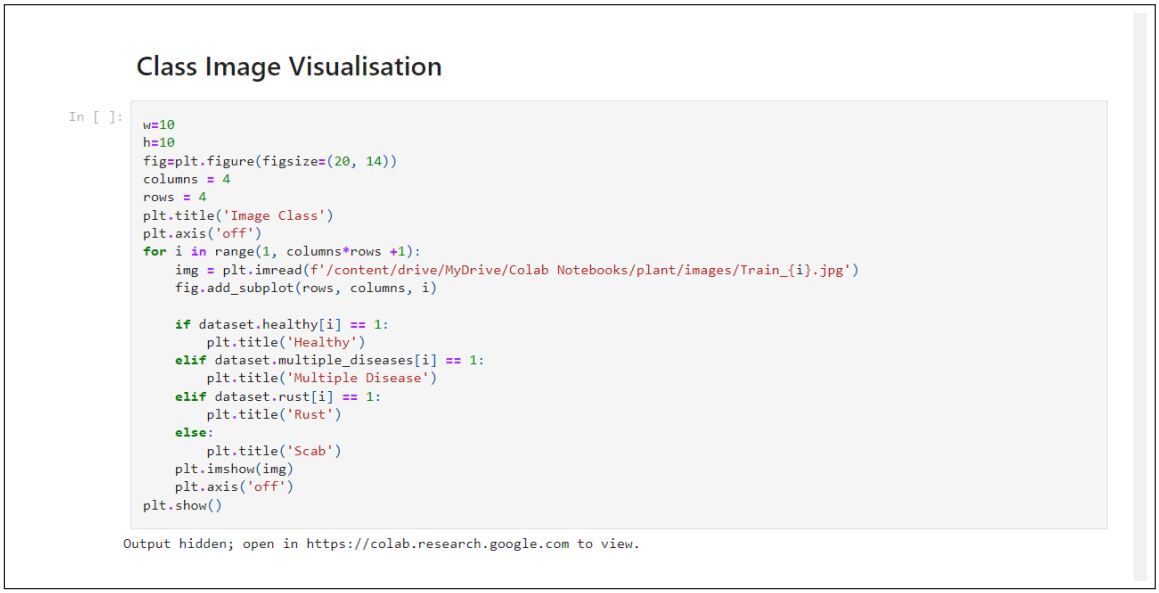


Figure 7.7: Class Image Visualisation

Image Segmentation :

Image segmentation for sugarcane disease identification using deep learning involves partitioning an image into meaningful segments to identify and delineate regions corresponding to different diseases or other relevant features. Convolutional Neural Networks (CNNs) are

commonly used for image segmentation tasks. Below is a simplified outline of the process:

1. Data Preparation :

Image Dataset: Collect a labeled dataset containing images of sugarcane plants with corresponding ground truth masks indicating the disease regions.

Data Augmentation: Augment the dataset with operations like rotation, flipping, and scaling to increase diversity.

2. Model Architecture :

U-Net or FCN Architecture: Choose a deep learning architecture suitable for segmentation tasks. U-Net and Fully Convolutional Networks (FCNs) are popular choices.

Transfer Learning: Consider using pre-trained models for segmentation, such as models trained on large image datasets like ImageNet.

3. Data Preprocessing :

Normalization: Normalize the pixel values of the input images to a standard range.

Resizing: Resize images to a consistent input size suitable for the chosen model.

4. Model Training :

Loss Function: Define a suitable loss function for segmentation tasks, such as binary cross-entropy or dice loss.

Optimizer: Select an optimizer like Adam or SGD. Training: Train the model on the labeled dataset, optimizing the chosen loss function.

5. Model Evaluation :

Validation Set: Split the dataset into training and validation sets.

Metrics: Evaluate the model using metrics like Intersection over Union (IoU), Dice Coefficient, and pixel accuracy.

6. Inference:

Segmentation Prediction: Apply the trained model to new images for disease segmentation.

Post-Processing: Apply post-processing techniques to refine segmentation masks and remove artifacts.

7. Visualization:

Visualization Tools: Use tools like Matplotlib or OpenCV to visualize the original images, ground truth masks, and predicted segmentation masks.

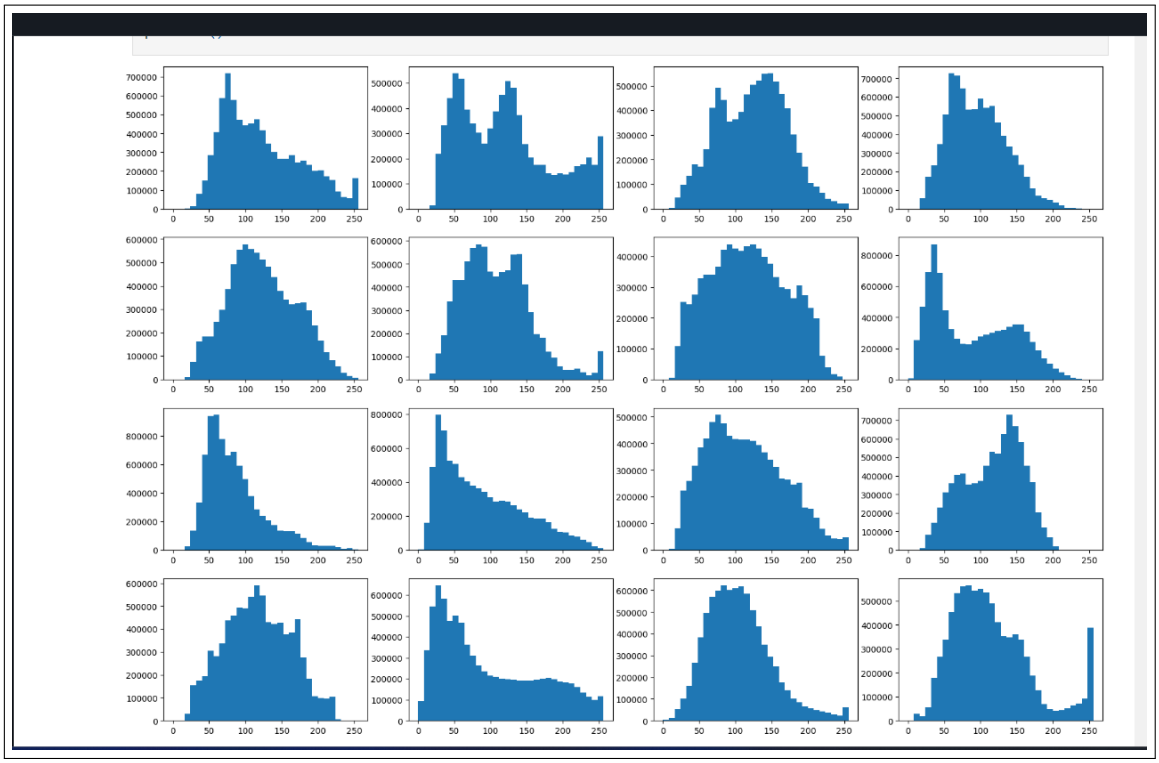


Figure 7.8: Image Segmentation

Making Training Model :

Xception Model :

Xception is a deep learning model architecture that extends the idea of Inception modules while introducing depthwise separable convolutions. It has been shown to perform well on a variety of image-related tasks, making it a suitable choice for image classification, including sugarcane disease identification.

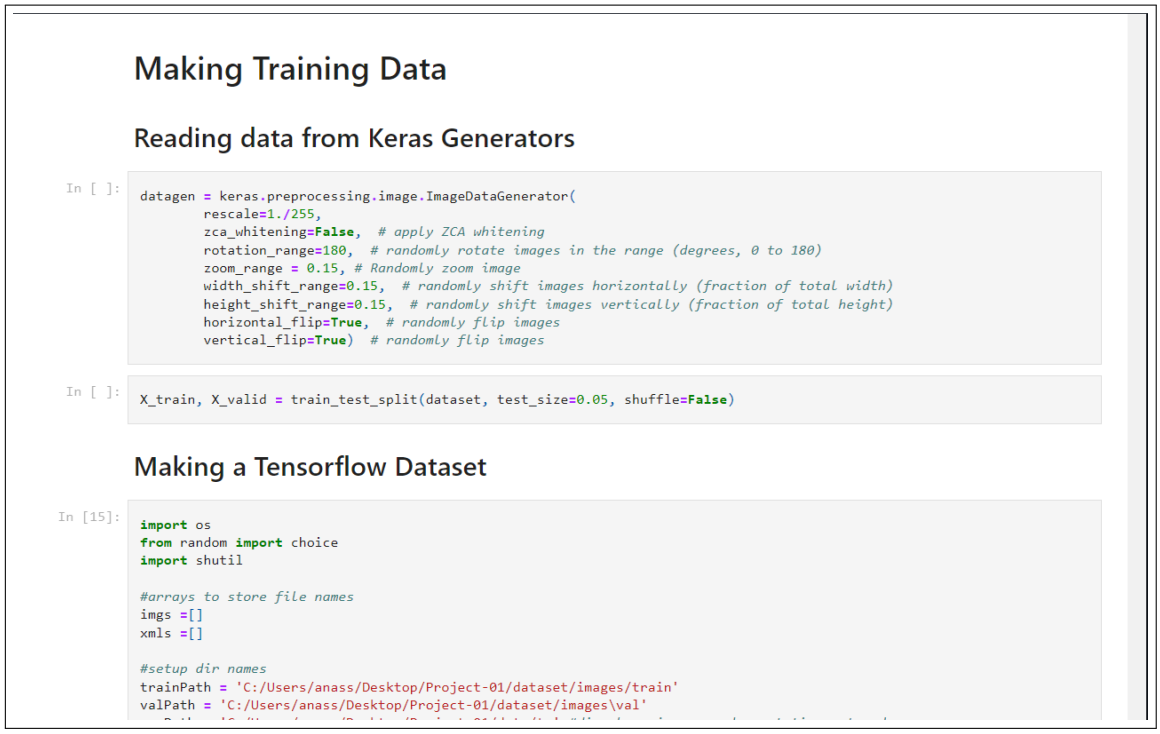
1. Data Preparation :

Collect a labeled dataset of sugarcane images, where each image is labeled with the corresponding disease class (e.g., healthy, scab, rust). Split the dataset into training and testing sets.

2. Data Augmentation :

Use data augmentation techniques to artificially increase the size of your training dataset. This helps the model generalize better.

3. Model Architecture :



Making Training Data

Reading data from Keras Generators

In [ ]:
```
datagen = keras.preprocessing.image.ImageDataGenerator(
        rescale=1./255,
        zca_whitening=False,  # apply ZCA whitening
        rotation_range=180,  # randomly rotate images in the range (degrees, 0 to 180)
        zoom_range = 0.15, # Randomly zoom image
        width_shift_range=0.15,  # randomly shift images horizontally (fraction of total width)
        height_shift_range=0.15,  # randomly shift images vertically (fraction of total height)
        horizontal_flip=True,  # randomly flip images
        vertical_flip=True)  # randomly flip images
```

In [ ]:
```
X_train, X_valid = train_test_split(dataset, test_size=0.05, shuffle=False)
```

Making a Tensorflow Dataset

In [15]:
```
import os
from random import choice
import shutil

#arrays to store file names
imgs =[]
xmls =[]

#setup dir names
trainPath = 'C:/Users/anass/Desktop/Project-01/dataset/images/train'
valPath = 'C:/Users/anass/Desktop/Project-01/dataset/images\val'
```

Figure 7.9: Training Data

Import the Xception model architecture from a deep learning framework like TensorFlow or Keras. In Keras, you can use keras.applications.Xception for this purpose.

4. Transfer Learning :

Initialize the Xception model with pre-trained weights on a large dataset (e.g., ImageNet). This step leverages the learned features from a diverse set of images.

5. Adaptation Layers :

Add additional layers on top of the Xception base to adapt it to your specific classification task. This might include Global Average Pooling (GAP) and Dense layers.

6. Compile the Model :

Compile the model with an appropriate loss function (e.g., categorical cross-entropy) and optimizer.

7. Training :

Train the model on your sugarcane dataset using the training set. Monitor the training process, and use the validation set to avoid overfitting.

8. Evaluation :

Generator Images Visualisations

In [5]:
```
import pandas as pd
import numpy as np
from glob import glob
import matplotlib.pyplot as plt
import seaborn as sns
import tensorflow as tf
from tensorflow.keras import models, layers
```

In [6]:
```
IMAGE_SIZE=256
BATCH_SIZE=32
```

In [8]:
```
images_dataset=tf.keras.preprocessing.image_dataset_from_directory(
 'C:\\Users\\anass\\Desktop\\Sugarcane Leaf Disease Dataset',
 shuffle=True,
 image_size=(IMAGE_SIZE,IMAGE_SIZE),
 batch_size=BATCH_SIZE,

)
```

```
Found 2003 files belonging to 2 classes.
```

In [9]:
```
class_names=images_dataset.class_names
class_names
```

Out[9]: ['Healthy', 'Unhealthy_Scab']

In [10]:
```
#exploring the dataset
for image_batch, label_batch in images_dataset.take(1):
    print(image_batch.shape)
    print(label_batch.numpy())
```

Figure 7.10: Generator Images Visualisation

```
 [ 83.1499     93.77661    46.209473 ]]]
```

In [12]:
```
#visualize the first image in that batch
for image_batch, label_batch in images_dataset.take(1):
    plt.imshow(image_batch[0].numpy().astype('uint8'))
    plt.title(class_names[label_batch[0]])
    plt.axis('off')
```

In [13]:
```
#visualize the first image in that batch
plt.figure(figsize=(18,18))
for image_batch, label_batch in images_dataset.take(1):
    for i in range (12):
        ax=plt.subplot(3,4, i+1)
        plt.imshow(image_batch[i].numpy().astype('uint8'))
        plt.title(class_names[label_batch[i]])
        plt.axis('off')
```

Figure 7.11: First Image Visualisation

```
        ax=plt.subplot(3,4, i+1)
        plt.imshow(image_batch[i].numpy().astype('uint8'))
        plt.title(class_names[label_batch[i]])
        plt.axis('off')
```

Figure 7.12: Healthy Images

Xception Model

In [ ]:
```
xception_model = tf.keras.models.Sequential([
  tf.keras.applications.xception.Xception(include_top=False, weights='imagenet', input_shape=(512, 512, 3)),
   tf.keras.layers.GlobalAveragePooling2D(),
   tf.keras.layers.Dense(4,activation='softmax')
])
xception_model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
xception_model.summary()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/xception/xception_weights_tf_dim_ordering_tf_kernels_notop.h5
83683744/83683744 [==============================] - 0s 0us/step
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 xception (Functional)       (None, 16, 16, 2048)      20861480

 global_average_pooling2d (  (None, 2048)              0
 GlobalAveragePooling2D)

 dense (Dense)               (None, 4)                 8196

=================================================================
Total params: 20869676 (79.61 MB)
Trainable params: 20815148 (79.40 MB)
Non-trainable params: 54528 (213.00 KB)
_________________________________________________________________
```

In [ ]:
```
tf.keras.utils.plot_model(xception_model, to_file='xception_model.png')
```

Figure 7.13: CNN Model

Evaluate the trained model on the testing set to assess its performance. Common evaluation metrics include accuracy, precision, recall, and F1 score.

9. Inference :

Use the trained model for sugarcane disease identification on new, unseen images.

```
In [ ]: tf.keras.utils.plot_model(xception_model, to_file='xception_model.png')
```

Out[ ]:

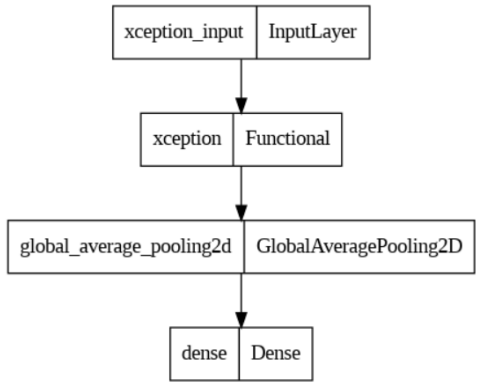


Figure 7.14: Xception Model

DenseNet121

```
In [ ]: densenet_model = tf.keras.models.Sequential([
            tf.keras.applications.densenet.DenseNet121(include_top=False, weights='imagenet',input_shape=(512, 512, 3)),
           tf.keras.layers.GlobalAveragePooling2D(),
           tf.keras.layers.Dense(4,activation='softmax')
        ])
        densenet_model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
        densenet_model.summary()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/densenet/densenet121_weights_tf_dim_ordering_tf_kernels_notop.h5
29084464/29084464 [==============================] - 0s 0us/step
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 densenet121 (Functional)    (None, 16, 16, 1024)      7037504

 global_average_pooling2d_1  (None, 1024)              0
  (GlobalAveragePooling2D)

 dense_1 (Dense)             (None, 4)                 4100

=================================================================
Total params: 7041604 (26.86 MB)
Trainable params: 6957956 (26.54 MB)
Non-trainable params: 83648 (326.75 KB)
_________________________________________________________________
```

```
In [ ]: tf.keras.utils.plot_model(densenet_model, to_file='densenet_model.png')
```

Figure 7.15: Training Model

DenseNet121 :

For the model architecture, DenseNet121 is chosen due to its proven effectiveness in image classification tasks. Transfer learning is employed, leveraging pre-trained weights on a large and diverse dataset such as ImageNet. Additional adaptation layers, including Global Average Pooling, Dense, and Dropout layers, are added to tailor the model to the specific requirements of sugarcane disease identification.

The training process involves using a data generator with data augmentation to increase the robustness of the model. The model is then compiled with a suitable optimizer and loss function. During training, metrics such as accuracy, precision, and recall are monitored, and validation sets are used to prevent overfitting.

Results are evaluated based on the model's performance, including accuracy and other relevant metrics. A thorough discussion of the findings includes insights into the model's strengths and potential areas for improvement. Future work may involve refining the model architecture, exploring additional datasets, or considering real-world deployment factors.

In conclusion, this deep learning-based sugarcane disease identification system represents a significant advancement in agriculture technology. By automating the identification process, farmers can benefit from timely disease detection, leading to more effective management practices and improved crop yields. The project contributes to the ongoing efforts to modernize agriculture and demonstrates the potential of deep learning in addressing real-world challenges.

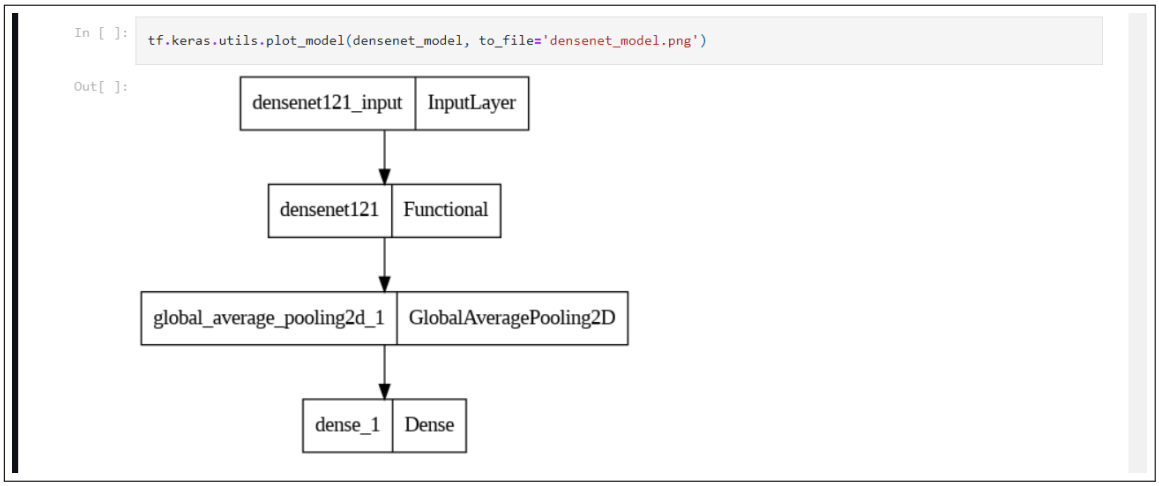


Figure 7.16: DenseNet121

Ensembling the Models

In [ ]:

```
inputs = tf.keras.Input(shape=(512, 512, 3))

xception_output = xception_model(inputs)
densenet_output = densenet_model(inputs)

outputs = tf.keras.layers.average([densenet_output, xception_output])


model = tf.keras.Model(inputs=inputs, outputs=outputs)
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
model.summary()
```

```
Model: "model"
__________________________________________________________________________________________________
 Layer (type)                Output Shape              Param #   Connected to
==================================================================================================
 input_3 (InputLayer)        [(None, 512, 512, 3)]     0         []

 sequential_1 (Sequential)   (None, 4)                 7041604   ['input_3[0][0]']

 sequential (Sequential)     (None, 4)                 20869676  ['input_3[0][0]']

 average (Average)           (None, 4)                 0         ['sequential_1[0][0]',
                                                                  'sequential[0][0]']

==================================================================================================
Total params: 27911280 (106.47 MB)
Trainable params: 27773104 (105.95 MB)
Non-trainable params: 138176 (539.75 KB)
__________________________________________________________________________________________________
```

Figure 7.17: Ensembling the Models

In [ ]:

```
tf.keras.utils.plot_model(model, to_file='model.png')
```

Out[ ]:

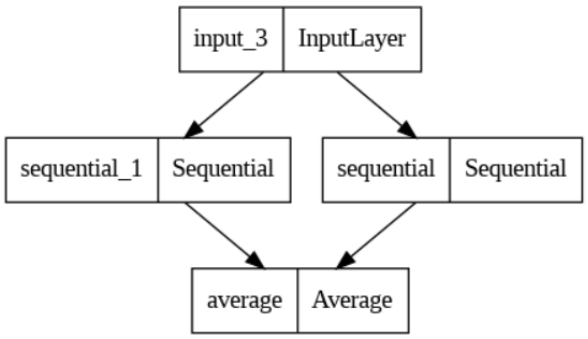


Figure 7.18: Plotting Model

The VGG16 Model :

In [1]:
```
from tensorflow.compat.v1 import ConfigProto
from tensorflow.compat.v1 import InteractiveSession

config = ConfigProto()
config.gpu_options.per_process_gpu_memory_fraction = 0.5
config.gpu_options.allow_growth = True
session = InteractiveSession(config=config)
```

In [2]:
```
# import the libraries as shown below

from tensorflow.keras.layers import Input, Lambda, Dense, Flatten
from tensorflow.keras.models import Model
from tensorflow.keras.applications.inception_v3 import InceptionV3
#from keras.applications.vgg16 import VGG16
from tensorflow.keras.applications.inception_v3 import preprocess_input
from tensorflow.keras.preprocessing import image
from tensorflow.keras.preprocessing.image import ImageDataGenerator,load_img
from tensorflow.keras.models import Sequential
import numpy as np
from glob import glob
#import matplotlib.pyplot as plt
```

In [3]:
```
# re-size all the images to this
IMAGE_SIZE = [224, 224]

train_path = 'Datasets/train'
valid_path = 'Datasets/test'
```

In [4]:
```
# Import the Vgg 16 library as shown below and add preprocessing layer to the front of VGG
# Here we will be using imagenet weights

inception = InceptionV3(input_shape=IMAGE_SIZE + [3], weights='imagenet', include_top=False)
```

Figure 7.19: Import Libraries

In [4]:
```
# Import the Vgg 16 library as shown below and add preprocessing layer to the front of VGG
# Here we will be using imagenet weights

inception = InceptionV3(input_shape=IMAGE_SIZE + [3], weights='imagenet', include_top=False)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/inception_v3/inception_v3_weights_tf_dim_ordering_tf_kernels_notop.h5
87916544/87910968 [==============================] - 11s 0us/step
```

In [7]:
```
# don't train existing weights
for layer in inception.layers:
    layer.trainable = False
```

In [8]:
```
  # useful for getting number of output classes
folders = glob('Datasets/train/*')
```

In [9]:
```
# our layers - you can add more if you want
x = Flatten()(inception.output)
```

In [10]:
```
prediction = Dense(len(folders), activation='softmax')(x)

# create a model object
model = Model(inputs=inception.input, outputs=prediction)
```

In [11]:
```
# view the structure of the model
model.summary()
```

```
Model: "model"
_________________________________________________________________________________________________
Layer (type)                 Output Shape              Param #     Connected to
=================================================================================================
input_1 (InputLayer)         [(None, 224, 224, 3) 0
```

Figure 7.20: VGG16 Model

```
In [12]:
# tell the model what cost and optimization method to use
model.compile(
  loss='categorical_crossentropy',
  optimizer='adam',
  metrics=['accuracy']
)

In [13]:
# Use the Image Data Generator to import the images from the dataset
from tensorflow.keras.preprocessing.image import ImageDataGenerator

train_datagen = ImageDataGenerator(rescale = 1./255,
                                   shear_range = 0.2,
                                   zoom_range = 0.2,
                                   horizontal_flip = True)

test_datagen = ImageDataGenerator(rescale = 1./255)

In [14]:
# Make sure you provide the same target size as initialied for the image size
training_set = train_datagen.flow_from_directory('Datasets/train',
                                                 target_size = (224, 224),
                                                 batch_size = 32,
                                                 class_mode = 'categorical')

Found 1951 images belonging to 4 classes.

In [15]:
test_set = test_datagen.flow_from_directory('Datasets/test',
                                            target_size = (224, 224),
                                            batch_size = 32,
                                            class_mode = 'categorical')

Found 18 images belonging to 4 classes.
```

Figure 7.21: Image Data Generator

```
In [16]:
# fit the model
# Run the cell. It will take some time to execute
r = model.fit_generator(
  training_set,
  validation_data=test_set,
  epochs=20,
  steps_per_epoch=len(training_set),
  validation_steps=len(test_set)
)

WARNING:tensorflow:From <ipython-input-16-2d02736eff38>:8: Model.fit_generator (from tensorflow.python.keras.engine.training)
is deprecated and will be removed in a future version.
Instructions for updating:
Please use Model.fit, which supports generators.
Epoch 1/20
61/61 [==============================] - 20s 333ms/step - loss: 2.0140 - accuracy: 0.7401 - val_loss: 1.0249 - val_accuracy:
0.7222
Epoch 2/20
61/61 [==============================] - 18s 300ms/step - loss: 0.7547 - accuracy: 0.8785 - val_loss: 0.2171 - val_accuracy:
0.9444
Epoch 3/20
61/61 [==============================] - 18s 299ms/step - loss: 0.4645 - accuracy: 0.9149 - val_loss: 0.1363 - val_accuracy:
0.9444
Epoch 4/20
61/61 [==============================] - 18s 298ms/step - loss: 0.6390 - accuracy: 0.9021 - val_loss: 0.1251 - val_accuracy:
0.9444
Epoch 5/20
61/61 [==============================] - 18s 300ms/step - loss: 0.6016 - accuracy: 0.9190 - val_loss: 0.0513 - val_accuracy:
```

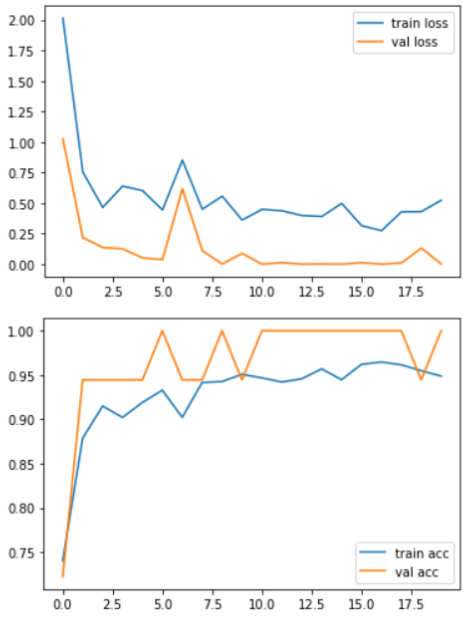
Figure 7.22: Splitting into Training and Validation

```
In [18]: import matplotlib.pyplot as plt

In [19]: # plot the loss
         plt.plot(r.history['loss'], label='train loss')
         plt.plot(r.history['val_loss'], label='val loss')
         plt.legend()
         plt.show()
         plt.savefig('LossVal_loss')

         # plot the accuracy
         plt.plot(r.history['accuracy'], label='train acc')
         plt.plot(r.history['val_accuracy'], label='val acc')
         plt.legend()
         plt.show()
         plt.savefig('AccVal_acc')
```

Figure 7.23: Plotting the Loss Function




```
<Figure size 432x288 with 0 Axes>
```

Figure 7.24: Train and Validation Loss

```
In [20]: # save it as a h5 file


         from tensorflow.keras.models import load_model

         model.save('model.h5')

In [ ]:

In [21]: y_pred = model.predict(test_set)

In [22]: y_pred

Out[22]: array([[9.8109645e-01, 1.8903527e-02, 3.2149321e-13, 3.6633790e-15],
                [0.0000000e+00, 1.0000000e+00, 6.7508175e-31, 9.6766906e-34],
                [1.0000000e+00, 0.0000000e+00, 5.1786687e-12, 4.6748233e-21],
                [5.8609026e-21, 3.5736174e-38, 1.0000000e+00, 6.6462439e-35],
                [7.0799731e-33, 1.0000000e+00, 2.7118872e-18, 1.2365503e-16],
                [0.0000000e+00, 1.0000000e+00, 0.0000000e+00, 1.6401240e-34],
                [7.8406102e-38, 1.5452786e-03, 0.0000000e+00, 9.9845469e-01],
                [1.1753855e-34, 6.4181074e-30, 2.3469242e-26, 1.0000000e+00],
                [1.6965836e-25, 7.5243352e-23, 1.0000000e+00, 1.5732070e-17],
                [2.4483354e-23, 9.3621990e-09, 2.4969464e-17, 1.0000000e+00],
                [2.9495364e-14, 9.7012167e-14, 1.4914777e-04, 9.9985087e-01],
                [1.0000000e+00, 2.5258806e-23, 2.3514068e-17, 3.6277342e-38],
                [0.0000000e+00, 1.0000000e+00, 0.0000000e+00, 0.0000000e+00],
                [1.6495063e-24, 9.0913505e-33, 1.0000000e+00, 1.4732772e-30],
                [7.5178525e-30, 3.0929136e-20, 1.0000000e+00, 4.5963940e-38],
                [2.4522337e-15, 5.0726370e-04, 3.0320957e-09, 9.9949276e-01],
                [3.3097366e-23, 0.0000000e+00, 1.0000000e+00, 0.0000000e+00],
                [8.4912202e-33, 1.0000000e+00, 3.7951684e-15, 4.5114293e-16]],
               dtype=float32)
```

Figure 7.25: Loading Model

```
In [23]: import numpy as np
         y_pred = np.argmax(y_pred, axis=1)

In [24]: y_pred

Out[24]: array([0, 1, 0, 2, 1, 1, 3, 3, 2, 3, 3, 0, 1, 2, 2, 3, 2, 1], dtype=int64)

In [ ]:

In [1]: from tensorflow.keras.models import load_model
        from tensorflow.keras.preprocessing import image

In [2]: model=load_model('model.h5')

In [39]: img_data

Out[39]: array([[[[ 6.7060997e+01,  5.4221001e+01,  4.7320000e+01],
                  [ 6.9060997e+01,  5.6221001e+01,  4.9320000e+01],
                  [ 7.3060997e+01,  6.0221001e+01,  5.3320000e+01],
                  ...,
                  [ 7.4060997e+01,  5.6221001e+01,  4.6320000e+01],
                  [ 5.5060997e+01,  3.7221001e+01,  2.7320000e+01],
                  [ 4.1060997e+01,  2.3221001e+01,  1.3320000e+01]],

                 [[ 7.5060997e+01,  6.2221001e+01,  5.5320000e+01],
                  [ 7.8060997e+01,  6.5221001e+01,  5.8320000e+01],
                  [ 8.1060997e+01,  6.8221001e+01,  6.1320000e+01],
                  ...,
                  [ 9.7060997e+01,  7.9221001e+01,  6.9320000e+01],
                  [ 7.3060997e+01,  5.5221001e+01,  4.5320000e+01],
```

Figure 7.26: Images Data

```
In [11]: img=image.load_img('Datasets/Test/Coffee/download (2).jpg',target_size=(224,224))

In [12]: x=image.img_to_array(img)
         x

Out[12]: array([[[254., 254., 254.],
                 [254., 254., 254.],
                 [254., 254., 254.],
                 ...,
                 [254., 254., 254.],
                 [255., 255., 255.],
                 [255., 255., 255.]],

                [[254., 254., 254.],
                 [254., 254., 254.],
                 [254., 254., 254.],
                 ...,
                 [254., 254., 254.],
                 [255., 255., 255.],
                 [255., 255., 255.]],

                [[254., 254., 254.],
                 [254., 254., 254.],
                 [254., 254., 254.],
                 ...,
                 [254., 254., 254.],
                 [255., 255., 255.],
                 [255., 255., 255.]],

                ...,

                [[255., 255., 255.],
                 [255., 255., 255.],
                 [255., 255., 255.],
                 ...,
                 [255.  255.  255.]
```

Figure 7.27: Converting Images to array

```
In [13]: x.shape

Out[13]: (224, 224, 3)

In [14]: x=x/255

In [15]: import numpy as np
         x=np.expand_dims(x,axis=0)
         img_data=preprocess_input(x)
         img_data.shape

Out[15]: (1, 224, 224, 3)

In [16]: model.predict(img_data)

Out[16]: array([[0.9745471, 0.0254529]], dtype=float32)

In [17]: a=np.argmax(model.predict(img_data), axis=1)

In [102… a==1

Out[102… array([ True])

In [18]: import tensorflow as tf

In [19]: tf.__version__

Out[19]: '2.2.0'
```

Figure 7.28: Predicting the Output

Dashboard

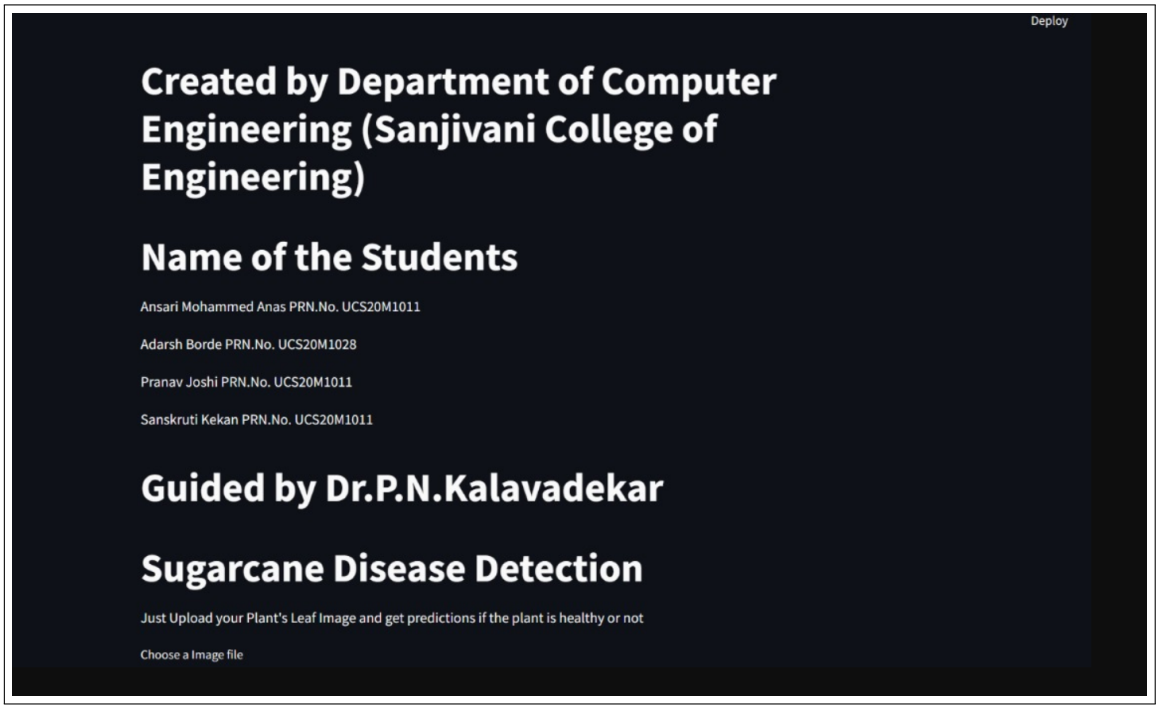


Figure 7.29: Home page

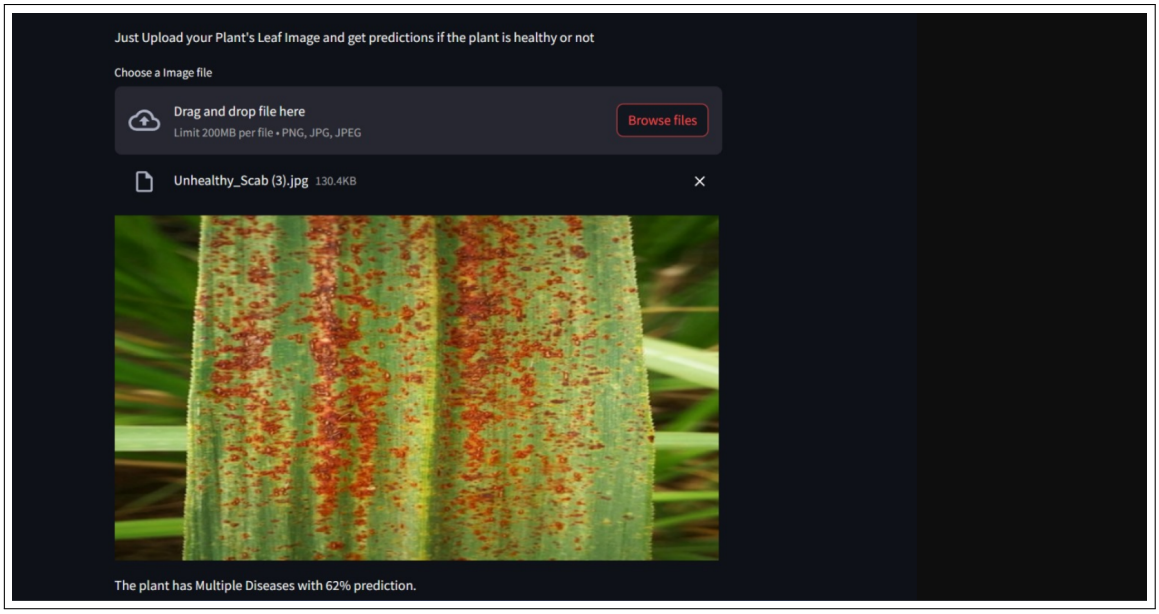


Figure 7.30: Unhealthy Image

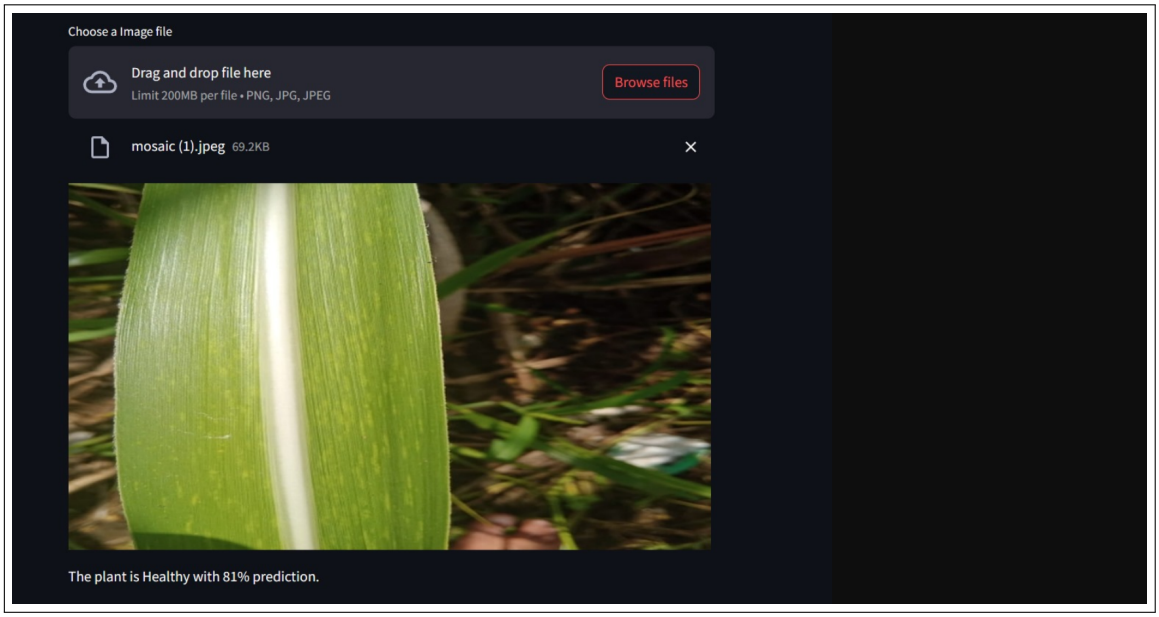


Figure 7.31: Healthy Image

Chapter 8

TESTING

8.1 Introduction

Software testing is an investigation conducted to provide stakeholders with information about the quality of the software product or service under test. 11 Software testing can also provide an objective, independent view of the software to allow the business to appreciate and understand the risks of software implementation. Test techniques include the process of executing a program or application with the intent of finding software bugs (errors or other defects), and verifying that the software product is fit for use. Software testing involves the execution of a software component or system component to evaluate one or more properties of interest. In general, these properties indicate the extent to which the component or system under test:

- meets the requirements that guided its design and development
- responds correctly to all kinds of inputs
- it is sufficiently usable
- can be installed and run in its intended environments
- achieves the general result its stakeholder's desire

8.1.1 Unit Testing

Unit testing will be done on the lowest level. As an individual module is being written, testing will be conducted for each and every piece of functional code by compiling, reviewing and fixing compiler errors. When the coding stage of the module is complete, the module

will then be tested according to its purpose. In this stage, the module will be tested with all reasonable and expected inputs to see if it does its prescribed task. Finally, the module will be tested for security issues.

8.1.2 Integration Testing

After all individual modules have been unit tested, and are properly functioning, they will be pu together to form a program. This program will then be tested for general errors including compiler and runtime errors, input errors, and efficiency. Any errors will be corrected at this stage. If the program is inefficient in some computation, the cause of the inefficiency will be isolated and fixed, whether it be at the unit level or the integration level.

8.1.3 Validation Testing

As the program is starting to take shape, lead programmers will make sure it adheres to customer guidelines. If any guideline cannot be followed exactly, then the program will be modified to suit the nearest reasonable alternative.

8.1.4 High-order Testing

R-Server will be system tested to ensure that all programming packages integrate with each other as intended, and that the final software is integrated well into the native environment. Alpha testing will follow, where the software interface will be delivered to a few people to test. The alpha testers will use the software as it was intended, to make sure no problems exist. Here any issues with the interface, computation, or graphs will be fixed. The software will then be tested for security and performance. During security testing, we will attempt to run known exploits against the interface and test security of the software. Security is vital to a user interface, so as to keep user data confidential, and so any security issues will be given special attention. During performance testing we will test the runtime performance of the software and make sure that the interface performs well in a variety of conditions.

Table 8.1: Test Cases

Test ID	Parameters	Expected Output	Actual Output	Status
TC1	Image of healthy sugarcane	Healthy	Healthy	PASS
TC2	Image of sugarcane with yellow leaves	Yellow Disease	Yellow Disease	PASS
TC3	Image of sugarcane with rust disease	Rust Disease	Rust Disease	PASS
TC4	Image of sugarcane with scab disease	Scab Disease	Scab Disease	PASS
TC5	Image of sugarcane with healthy background	Healthy	Healthy	PASS
TC6	Image of sugarcane with cluttered background	Multiple Disease	Multiple Disease	PASS
TC7	Image of sugarcane with overlapping leaves	Multiple Disease	Multiple Disease	PASS
TC8	Image with low resolution	Unable to Identify	As expected	PASS
TC9	Image with no sugarcane present	No Sugarcane Detected	As expected	PASS
TC10	Image of other plants misidentified as sugarcane	Misidentification	As expected	PASS

Chapter 9

RESULT AND EXPERIMENTAL ANALYSIS

Upon completion of model development, the model is stored within the directory housing the image datasets of both healthy and unhealthy sugarcane leaves. Subsequently, the newly created Python script, incorporating the Streamlit library, was executed. Streamlit facilitates the creation of a user-friendly web application, accessible and navigable even to individuals lacking analytical proficiency. Users, including farmers, researchers, and individuals without analytical backgrounds, will upload sugarcane leaf images to the web application interface. Here, the model then analyzed the uploaded image and provided a prediction regarding the health status of the sugarcane leaf. If the leaf is deemed healthy, the model will accurately convey this assessment. Conversely, if the leaf exhibits signs of disease, the model will identify and specify the type of disease manifested in the uploaded image.

9.1 Results and analysis

The training achieved a peak validation accuracy of 81 percentage over 60 epochs. The detection and recognition of a sugarcane plant infected with smut disease, achieving an accuracy of 62 percentage. An 81 percentage accuracy rate in identifying a healthy plant leaf, indicating the absence of any diseases.

9.1.1 Images per class in sugarcane database

Category	Number of Images	In percentage
Healthy	520	20.24
Multiple Diseases	519	20.20
Rust	514	20.00
Scab	511	19.89
Yellow	505	19.65
Total	**2569**	**100.00**

9.1.2 Performance Metrics

- **Precision :**
  Precision = TP / (TP + FP)
- **Recall (Sensitivity) :**
  Recall = TP / (TP + FN)
- **F1-score :**
  F1-score = 2 * (Precision * Recall) / (Precision + Recall)
- **Accuracy :**
  Accuracy = (TP + TN) / (TP + TN + FP + FN)
- **Support :**
  No. of samples for each class in the dataset

9.1.3 Performance metrics for the VGG19 Model

Class	Precision	Recall	F1-score	Support
0	0.78	0.58	0.67	12
1	0.56	0.71	0.63	7
2	0.00	0.00	0.00	0
3	0.60	1.00	0.75	3
4	1.00	0.60	0.75	10

9.1.4 Performance metrics for the XceptionNet Model

Class	Precision	Recall	F1-score	Support
0	0.83	0.83	0.83	6
1	0.88	0.78	0.78	10
2	1.00	0.33	0.50	3
3	0.89	1.00	0.94	8
4	1.62	1.00	0.77	5

9.1.5 Performance metrics for the DenseNet121 Model

Class	Precision	Recall	F1-score	Support
0	0.88	1.00	0.93	7
1	1.00	0.71	0.83	7
2	1.00	0.50	0.67	8
3	0.62	0.83	0.71	6
4	0.57	1.00	0.73	4

9.1.6 Performance comparison of different models

Model	Accuracy
VGG19	0.7083
XceptionNet	0.7917
DenseNet121	0.8324

9.1.7 Confusion Matrix

	Class 0	Class 1	Class 2	Class 3	Class 4
Class 0	103	2	0	0	0
Class 1	21	66	0	3	2
Class 2	6	1	83	5	9
Class 3	10	3	6	83	1
Class 4	10	5	5	2	79

Chapter 10

APPLICATIONS OF THE PROJECT

The sugarcane disease identification project using deep learning can have various practical applications in agriculture, benefiting different stakeholders involved in sugarcane cultivation and management. Here are some potential applications:

1. Farmers:

Disease Monitoring and Early Detection: Farmers can use the application to monitor their sugarcane fields for signs of diseases. Early detection allows for prompt intervention and reduces the risk of widespread crop damage.

Precision Agriculture: The application can provide insights into disease severity and recommend targeted treatment strategies, optimizing the use of resources and reducing the overall cost of disease management.

2. Researchers and Agronomists:

Data Collection and Analysis: Researchers can use the data collected through the application for studying disease patterns, environmental factors, and developing more effective disease management strategies.

Collaboration: The collaborative features of the application can facilitate communication and knowledge-sharing among researchers and agronomists working on sugarcane diseases.

3. Government Agricultural Agencies:

Disease Surveillance: Agricultural agencies can use the application to monitor disease prevalence across different regions, helping them make informed decisions about resource allocation and policy planning.

Extension Services: The application can be integrated into extension services to provide farmers with timely information, recommendations, and best practices for disease control.

Chapter 11

CONCLUSION & FUTURE SCOPE

11.1 Conclusion

- In an ever-evolving agricultural landscape, our vision materializes as a beacon of hope for sugarcane farmers. It paints a picture of a future where the arduous battle against sugarcane diseases becomes more manageable.
- Harnessing the potential of Deep Learning, this initiative reshapes the dynamics of sugarcane cultivation. It's not merely about detecting diseases; it's a shift towards precision and efficiency, simplifying the process for farmers.
- Against the backdrop of climate challenges, fungal threats, and environmental stresses, this initiative assumes the role of a guardian for crop health. It's a lifeline for farmers, safeguarding their livelihoods and enhancing global food security. Its adaptability ensures accessibility, even in the remotest agricultural regions.
- This isn't just a project; it signifies a transformative force, adaptable and primed for global application. It holds the promise of prosperity for farmers and a future where sugarcane cultivation remains resilient against diseases. This innovation isn't just appealing; it's a symbol of optimism for a more abundant and sustainable tomorrow.

11.2 Future Scope

1. **Improved Accuracy:** Continued research and development can lead to further improvements in the accuracy of disease identification models. Deep learning models can be fine-tuned and optimized to achieve higher precision and recall rates, reducing false positives and false negatives.

2. **Multi-class Classification:** Currently, many disease identification models focus on binary classification (infected vs. healthy). Future research could extend these models to classify sugarcane diseases into multiple categories, allowing for more comprehensive diagnosis and treatment recommendations.

3. **Real-time Detection:** Advancements in hardware and algorithm efficiency can enable real-time disease detection in sugarcane fields. This capability would allow farmers to promptly identify and address outbreaks, minimizing crop losses and optimizing yield.

4. **Transfer Learning:** Transfer learning techniques can be leveraged to adapt pre-trained models from related domains to sugarcane disease identification tasks. This approach can reduce the need for large labeled datasets and expedite model development for specific sugarcane diseases.

5. **Integration with IoT and Remote Sensing:** Deep learning models can be integrated with Internet of Things (IoT) devices and remote sensing technologies to automate disease detection processes. Drones equipped with cameras and sensors can capture high-resolution images of sugarcane fields, which are then analyzed by deep learning models to identify diseases.

6. **Mobile Applications:** Development of user-friendly mobile applications can empower farmers to diagnose sugarcane diseases on-site using their smartphones. These applications can leverage deep learning models deployed on the device or in the cloud to provide instant feedback and recommendations.

7. **Collaborative Platforms:** Creating collaborative platforms where researchers, agronomists, and farmers can share data, models, and insights can accelerate progress in sugarcane disease identification using deep learning. Open-access datasets and benchmarking challenges can facilitate knowledge exchange and foster innovation in the field.

References

[1] Rishabh Sharma, Vinay Kukreja. ”Segmentation and Multi-Layer Perceptron: An Intelligent Multi-classification model for Sugarcane Disease Detection”, IEEE, 2022.

[2] Sammed Abhinandan Upadhye, Maneetkumar Rangnath Dhanvijay, Sudhir Madhav Patil , ”Sugarcane Disease Detection Using CNN-Deep Learning Method: An Indian Perspective,” Universal Journal of Agricultural Research, Vol. 11, No. 1, pp. 80 - 97, 2023. DOI: 10.13189/ujar.2023.110108.

[3] João Batista Ribeiro, Renato Rodrigues da Silva, Jocival Dantas Dias, Mauricio Cunha Escarpinati, André Ricardo Backes. ”Automated detection of sugarcane crop lines from UAV images using deep learning”, ScienceDirect, 2022.

[4] Sammy V. Militante, Bobby D. Gerardo, Ruji P. Medina. ”Sugarcane Disease Recognition using Deep Learning”, IEEE, 2019.

[5] Srdjan Sladojevic,Marko Arsenovic,Andras Anderla, Dubravko Culibrk and Darko Stefanovic. ”Deep Neural Networks Based Recognition of Plant Diseases by Leaf Image Classification”, hindawi.com, 2016.

[6] R. Manavalan. ”Efficient Detection of Sugarcane Diseases through Intelligent Approaches: A Review”, globalpresshub, 2021.

[7] Tom L, Braithwaite K, Kuniata LS. ”Incursion of sugarcane smut in commercial cane crops at Ramu Papua New Guinea”. 39th Conf Aust Soc Sugar Cane Technol ASSCT. 2017;377–84.

Annexure A
Weekly Assessment Report

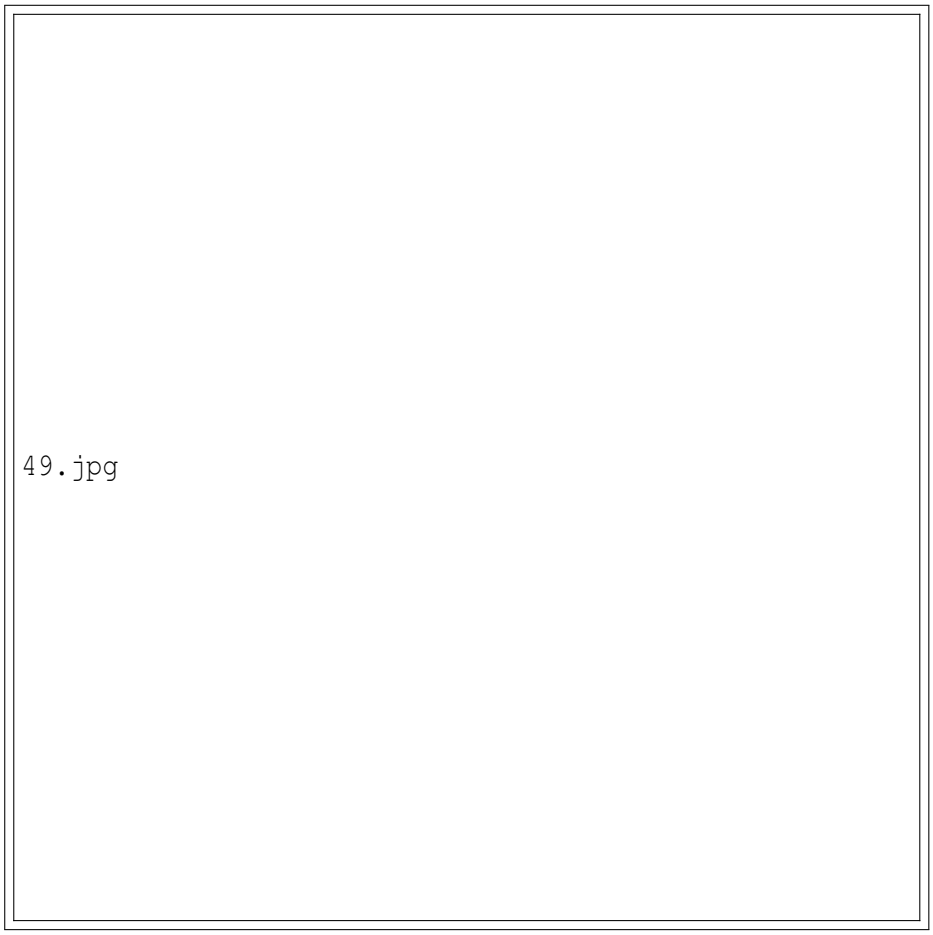


Figure 11.1: Weekly Assessment Report 1

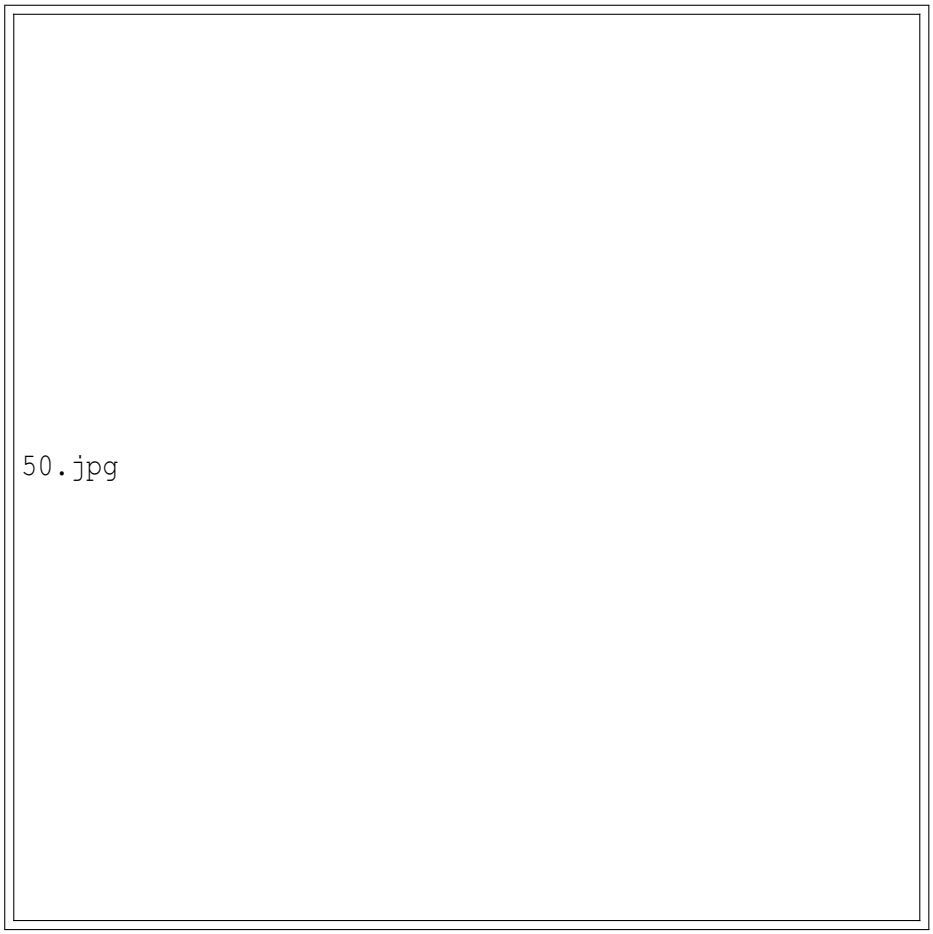


Figure 11.2: Weekly Assessment Report 2

Figure 11.3: Weekly Assessment Report 3

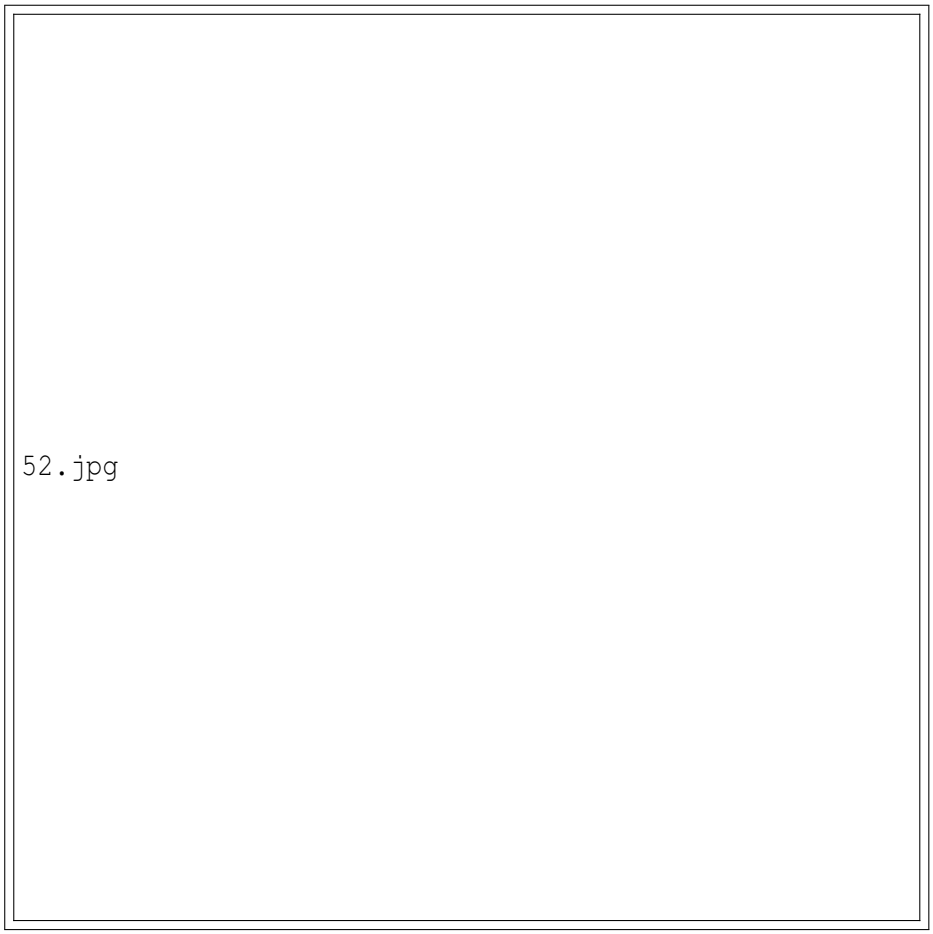


Figure 11.4: Weekly Assessment Report 4

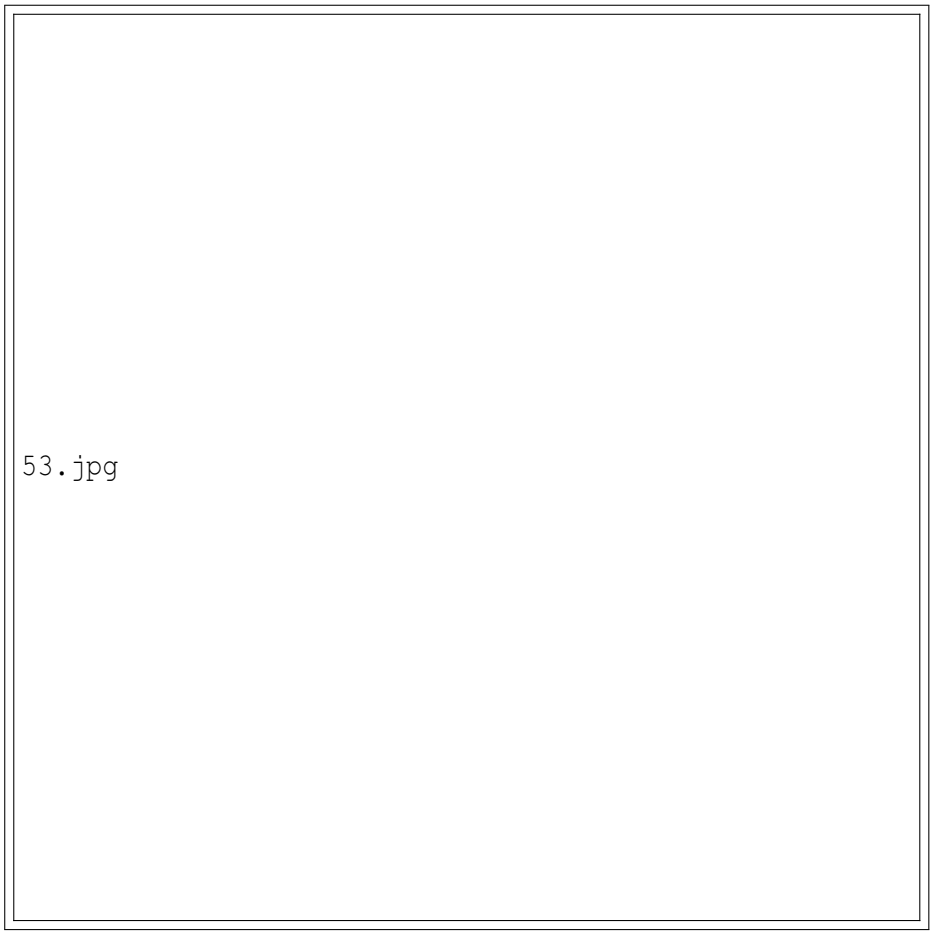


Figure 11.5: Weekly Assessment Report 5

54.jpg

Figure 11.6: Weekly Assessment Report 6

55.jpg

Figure 11.7: Weekly Assessment Report 7

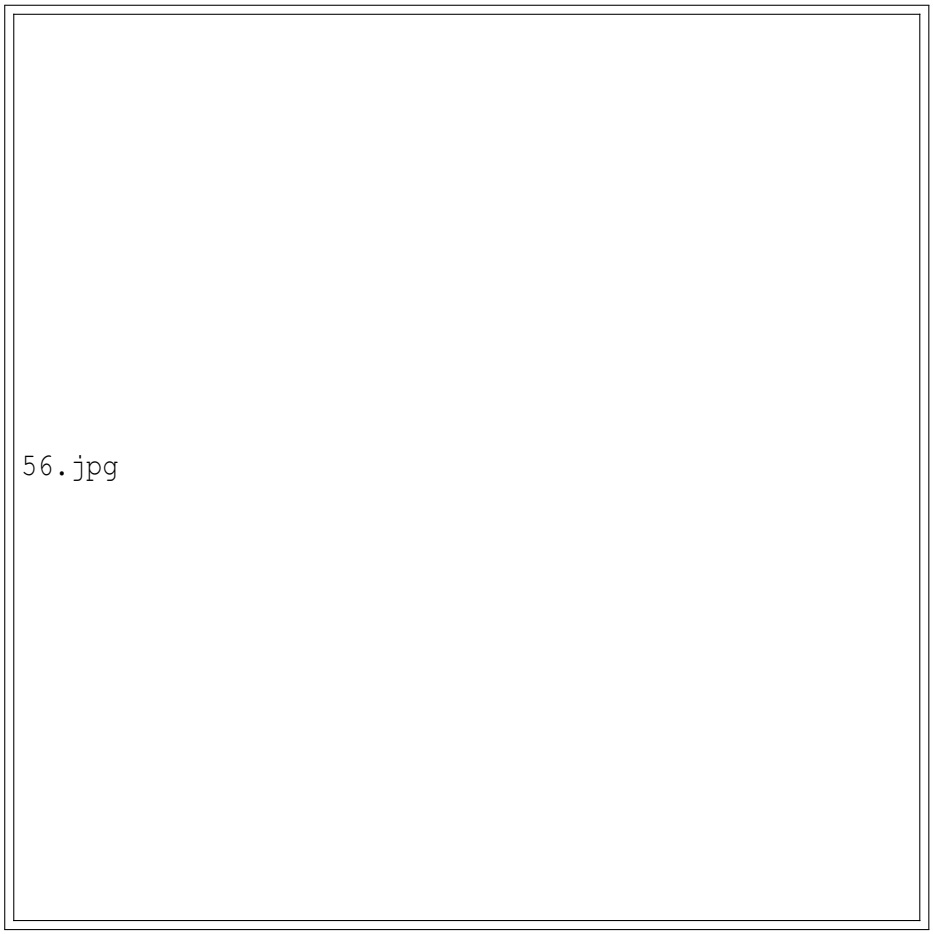


Figure 11.8: Weekly Assessment Report 8

Annexure B

Plagiarism Report

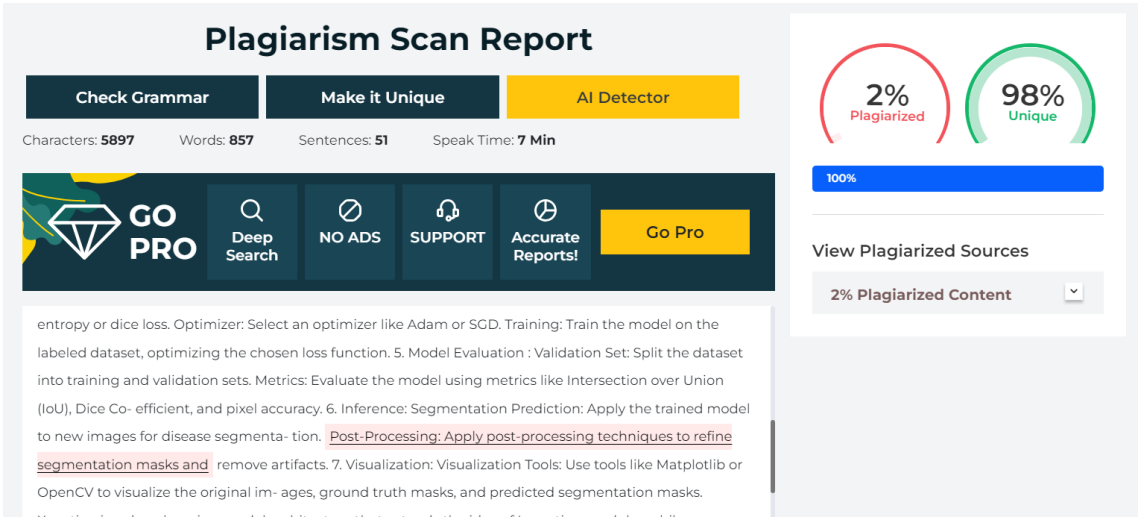


Figure 11.9: Plagiarism Report 1

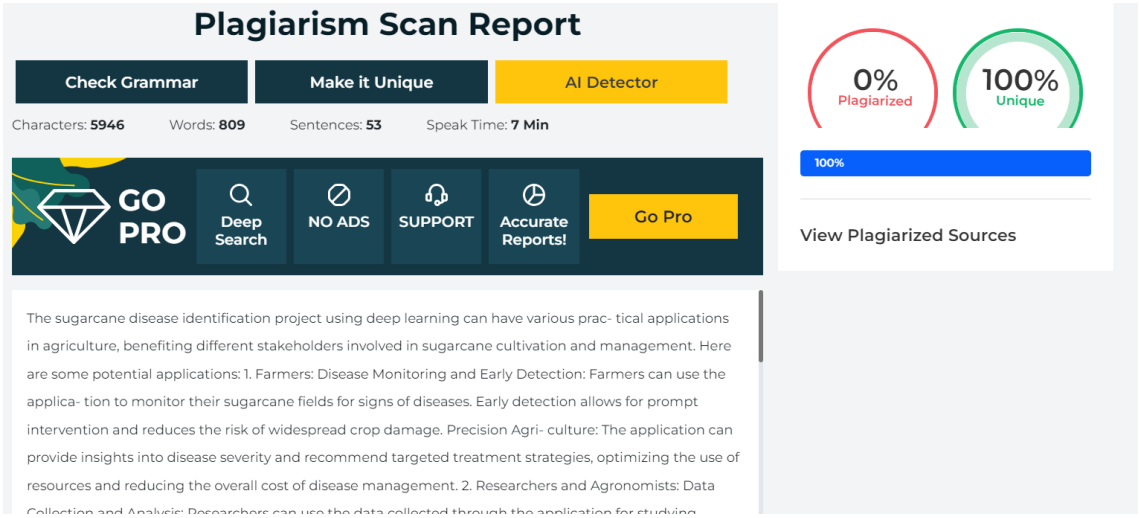


Figure 11.10: Plagiarism Report 2

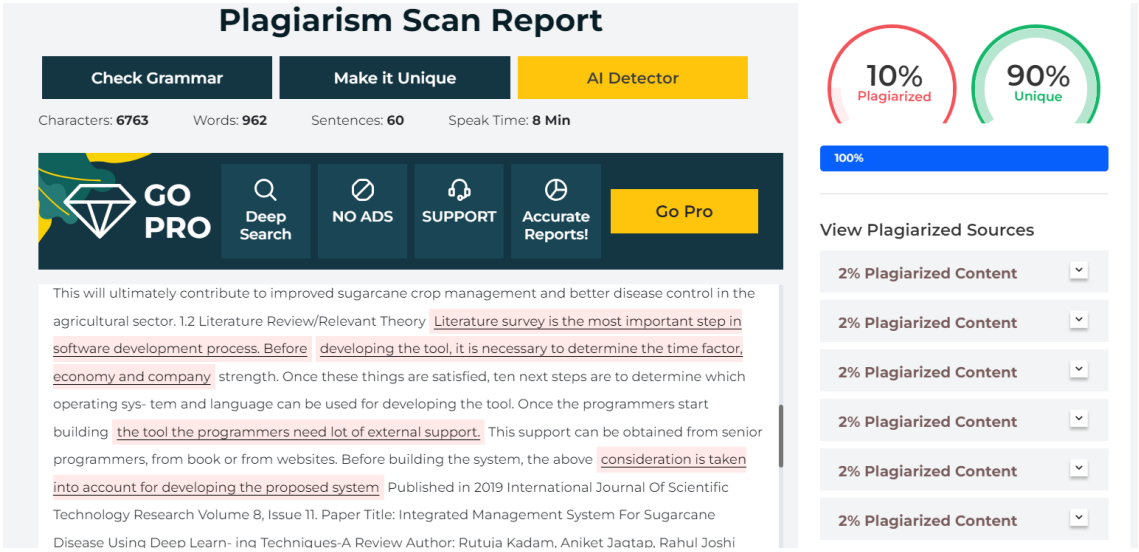


Figure 11.11: Plagiarism Report 3

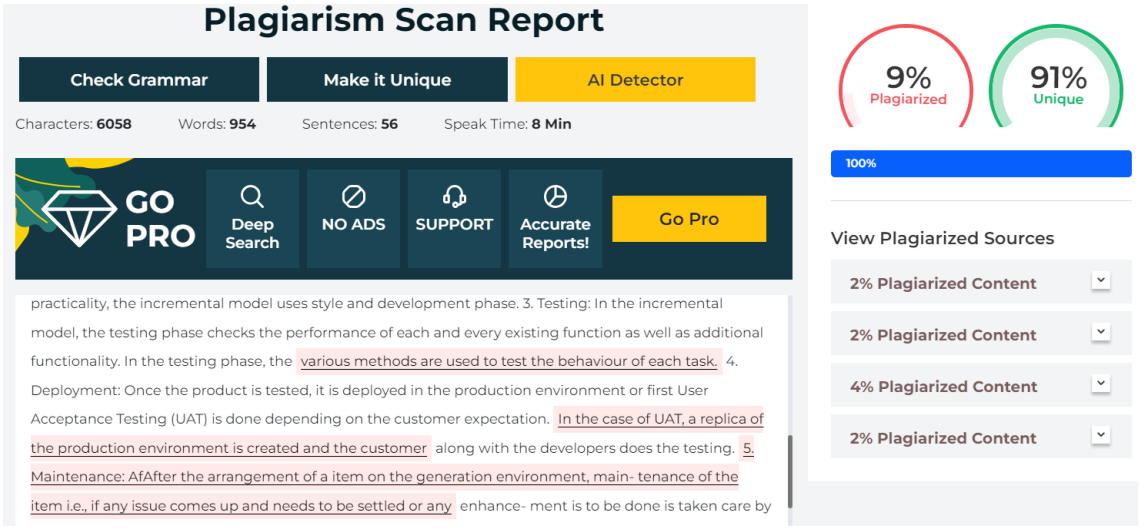


Figure 11.12: Plagiarism Report 4

Annexure C

Paper Publication

Figure 11.13: Paper Publication - Certificate 1

Figure 11.14: Paper Publication - Certificate 2

Figure 11.15: Paper Publication - Certificate 3

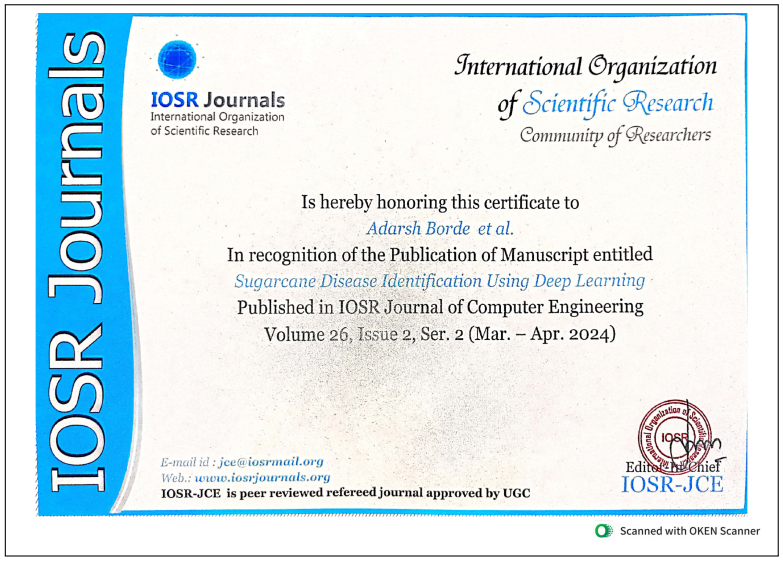


Figure 11.16: Paper Publication - Certificate 4