DEVELOPMENT OF RECOMMENDATION SYSTEM FOR DETECTION OF MAIZE CROPS DISEASE

BY

ANOIBI, Dickson Miracle

2019/2/77823CT

DEPARTMENT OF COMPUTER SCIENCE

FEDERAL UNIVERSITY OF TECHNOLOGY

MINNA

OCTOBER, 2024

DEVELOPMENT OF RECOMMENDATION SYSTEM FOR DETECTION OF MAIZE CROPS DISEASE

BY

ANOIBI, Dickson Miracle

2019/2/77823CT

PROJECT TO BE SUBMITTED TO THE DEPARTMENT OF COMPUTER SCIENCE, SCHOOL OF INFORMATION AND COMMUNICATION TECHNOLOGY, FEDERAL UNIVERSITY OF TECHNOLOGY, MINNA, NIGERIA IN PARTIAL FULFILLMENT OF THE REQUIREMENT FOR THE AWARD OF THE DEGREE OF BACHELOR OF TECHNOLOGY IN COMPUTER SCIENCE

OCTOBER, 2024

DECLARATION

I hereby declare that this project titled: ”Development of Recommendation System for Detection of Maize Crops Disease” is a collection of my original project work and it has not been presented for any other qualification anywhere. Information from other sources (published or unpublished) has been duly acknowledged.

ANOIBI, Dickson Miracle

2019/2/77823CT
Federal University of
Technology, Minna, Nigeria.

Signature and Date

CERTIFICATION

I hereby declare that this project titled: “Development of Recommendation System for Detection of Maize Crops Disease” by ANOIBI, Dickson Miracle (2019/2/77823CT), meets the regulations governing the award of the degree of Bachelor of Technology in Computer Science of the Federal University of Technology, Minna.

Dr. E.F. Aminu

Project Supervisor — Signature and Date

Dr. (Mrs) O.A. Abisoye

Head of Department — Signature and Date

External Examiner — Signature and Date

DEDICATION

The study is dedicated to the Almighty, the giver of wisdom, knowledge, and understanding. To the unwavering support and encouragement of our family and friends, their belief in our abilities and relentless encouragement motivated me to pursue this endeavor. We extend my deepest gratitude to them

ACKNOWLEDGEMENTS

I am deeply grateful to the Almighty for His unwavering guidance, without which I wouldn't have come this far. His grace has been the cornerstone of all my accomplishments. My heartfelt thanks go to my Project Supervisor, Dr. E. F. Aminu, and Mrs. A. Ekundayo. Your tireless dedication, wisdom, and mentorship have been invaluable throughout this journey. Your contribution to education and to our personal growth will be remembered for years to come. A special thank you to the Head of Department, Dr. (Mrs.) O. A. Abisoye, for her exemplary leadership and for ensuring that everything within the department runs smoothly. I also extend my deepest appreciation to my lecturers, Prof. J. K. Alhassan, Dr. M.B Abdullahi, Dr. S. A. Adepoju, Dr. O. A. Ojerinde, Dr. S. A. Bashir, Dr. M.D Abdulmalik, Mr. A. M. Saliu, Mr. C. U. Ugwuoke, Mr. I.S Shehu, Mr. M. I. Kolo, Dr. M.K Muhammad, Mr. V.N Adama, Mr. K.H Lawal, Mr. Y.B Lasotte, Mr. O. L Lawal, Mr. A. U Sani, Mr. B.I Aloghena,and Mr. S.T Abubaka. Your collective effort in shaping my knowledge and character is something I will carry with me always. To my amazing parents, Mr. and Mrs. Anoibi, I am forever indebted to you. Your unwavering love, sacrifices, and constant support have been the foundation of my success. You are truly the best. A huge thank you also to my wonderful siblings for always being there and showing concern for my well-being. May God bless you abundantly. Finally, to my dear friends, (Kabulu and Domnic), and all my course mates, your support, camaraderie, and encouragement have made this journey an unforgettable one. You are all incredible, and I feel blessed to have you in my life.

ABSTRACT

Today, one of the major threats to food security besides climate change is the threat of pest and insects for example, the case of armyworm disease in maize crop. The existing traditional methods lack the capacity to address this challenge efficiently. In view of this development, application of artificial intelligence and machine learning techniques become inevitable to address the challenge. Therefore, this research project aims to develop a recommendation system for the detection of maize crops disease. The system utilizes a convolutional neural network (CNN) implemented with Scipy and Sklearn to learn and recognize patterns of diseases from a labeled dataset of maize plants. The dataset is organized into two categories: healthy and infected, each containing images representing their respective classes. The training process involves loading the dataset, preprocessing images, and training the model to optimize its ability to distinguish between healthy and infected maize plant. After training, the model was embedded in a subsystem particularly the raspberry pi controller, which analyzes data and triggers the herbicide pump for treatment. This project also includes a testing phase where the saved model is employed to predict the health status of maize plant leaves in a separate test dataset. The project showcases the application of machine learning in agriculture, offering a potential tool for farmers to assess maize plants health status through automated image analysis.

TABLE OF CONTENTS

CHAPTER FIVE: SUMMARY, CONCLUSION, AND RECOMMENDATIONS

LIST OF TABLES

TABLE **PAGE**

LIST OF FIGURES

CHAPTER ONE

1.0 INTRODUCTION

1.1 Background to the Study

Maize is an incredibly adaptable crop that flourishes across various agro-climatic zones, often referred to as the "queen of cereals" due to its outstanding genetic potential for high yields. In Nigeria and West Africa, maize plays a critical role as a staple food, being the most extensively grown cereal crop worldwide. According to Ebukiba et al., (2020), about 80% of maize production is consumed by humans and animals, while the remaining 20% is used in different industrial processes.

Maize is a member of the Maydeae tribe within the Poaceae family. The word "Zea" originates from an ancient Greek term for a type of food grain. The genus Zea comprises four species, but Zea mays L. is the most significant economically. Maize is a rich source of essential nutrients, including minerals, carbohydrates, vitamin B, and protein, particularly in sub-Saharan Africa. It is widely used in the production of corn syrup, starch, animal feed, protein, oil, sweeteners, beverages, fuel, and more.

The increasing demand for maize, driven by population growth, outpaces its supply, placing greater pressure on available agricultural land to enhance productivity. Ebukiba et al., (2020), stated that; majority of Nigerian farmers, about 90%, are smallholders facing difficulties such as limited capital, outdated technology, and low crop yields per hectare, exacerbating the supply-demand imbalance.

According to Chauhan et al., (2021), maize is vulnerable to several diseases, which result in significant yield reductions and lower quality. These diseases are caused by both biotic factors (such as bacteria, fungi, nematodes, and viruses) and abiotic factors (like nutrient deficiencies,

moisture, and heat). Typically, maize diseases affect leaves, fruits, and stems. Early and precise detection is vital for the effective management of these diseases.

In recent years, machine learning, particularly in the form of Convolutional Neural Networks (CNN), has transformed numerous industries, including agriculture. In a study by Mwaza et al., (2023), a CNN model was applied to maize disease detection and they achieved a remarkable success, outperforming other models with a total accuracy of 98.7%.

This CNN model was integrated into an Android application, yielding promising results. However, the researchers noted that the model requires modification to recognize other objects or deliver an error message when it can only identify maize-related diseases. Additionally, the solution's disease treatment feature needs improvement to offer accurate recommendations to users (Mwaza et al., 2023).

In a recent study, Bachhal et al., (2023), introduced a real-time system for detecting maize plant diseases utilizing deep convolutional neural networks. They combined data from various sources, including the highly efficient Plant Village dataset, along with 450 images of both healthy and diseased maize leaves. The authors created a P-CNN model, which combines PSPNet and CNN for object detection and semantic segmentation, achieving an exceptional accuracy rate of 99.85%.

This study is directly relevant to the current project, which focuses on designing an expert system to detect and recommend treatments for maize diseases. The aim is to integrate this system with an automated herbicide sprayer to quickly identify and address maize plant diseases. The work of Bachhal et al., (2023), provides a strong basis for the development of a precise and effective disease detection system, which could be paired with an automated herbicide sprayer for a complete solution in managing maize crop diseases. Their recommendation to explore alternative segmentation techniques and use datasets of varying

sizes with different disease combinations aligns well with the project's goal to enhance the expert system's ability to swiftly detect diseases.

Classifying maize leaf diseases is a critical task in agricultural practices, and supervised machine learning algorithms can effectively address this need. Pate et al., (2023), developed a supervised machine learning model to classify maize leaf diseases using texture analysis and feature extraction methods. The authors applied four algorithms—Decision Tree, Gradient Boosting, Support Vector Machine (SVM), and K-Nearest Neighbors (KNN)—to the highly effective Plant-Village dataset.

Their findings underscore the success of the approach in swiftly diagnosing diseases, offering valuable insights for farmers aiming to maximize crop yields. This study is highly relevant to the current project, which aims to design an expert system for detecting maize diseases and recommending treatments. The integration of this expert system with an automated herbicide sprayer will enable rapid detection and response to maize crop diseases.

1.2 Motivation of Study

The motivation behind this study arises from the substantial impact that maize diseases have on crop yield and food security. Traditional methods for detecting diseases in maize crops are labor-intensive, slow, and susceptible to human error, which leads to delayed treatment and increased losses. Furthermore, current solutions often fall short in providing real-time, automated responses and remain inaccessible to many farmers due to their reliance on mobile apps or internet connectivity.

This project seeks to overcome these limitations by developing an integrated maize disease detection system utilizing image processing and machine learning, with a focus on Convolutional Neural Networks (CNNs). The system will enable fast, automated identification of common maize diseases and activate an automatic herbicide sprinkler for immediate

treatment. The aim is to develop a scalable, efficient solution that improves agricultural productivity, reduces crop losses, and supports sustainable farming practices.

1.3 Statement of Problem

Current embedded expert systems for detecting maize crop diseases come with several shortcomings. Firstly, they lack an automated herbicide sprinkler to treat infected leaves immediately, leading to delayed and less effective treatment. Secondly, most existing solutions are mobile applications, which may not be accessible to all farmers, particularly those without smartphones or internet connectivity.

Another issue is the use of inefficient RGB color configurations in these systems, which hampers image processing accuracy and reduces the effectiveness of disease detection. Moreover, the accuracy of disease detection in these systems is often inadequate, leading to incorrect treatments and economic losses.

To overcome these challenges, this project aims to develop an integrated embedded expert system for smart agriculture that incorporates image processing, expert systems, and IoT technology. Building on the success of previous studies, such as Chandana et al., (2020), who combined image processing, expert systems, and IoT, and Khan et al., (2023), who developed a smart system for detecting leaf blight using deep learning and CNNs, this project seeks to create a more efficient and accessible solution for the detection and management of maize crop diseases.

1.4 Aim and Objectives of Study

The aim of this project is to develop a recommendation system for detecting maize crop diseases. In order to actualize this aim, the following objectives must be met:

i. Design a recommendation algorithm for the maize crop diseases.
ii. Implement the designed algorithm for the proposed system.

iii. Develop the proposed system for the detected maize crop diseases.

iv. Validate the system.

1.5 Significance of Study

The proposed expert system for detecting and recommending treatments for maize diseases is important because it has the potential to significantly improve crop yields and minimize losses caused by diseases. By facilitating rapid disease detection and treatment, the system will enable farmers to take timely action, mitigating the adverse effects of diseases on their crops

Moreover, the system will support farmers in making better-informed decisions by providing accurate, real-time information about their crops. This will lead to more efficient crop management and disease control practices, allowing farmers to adopt more effective strategies.

The automation of disease detection and treatment will also enhance crop management efficiency by reducing the need for manual scouting and spraying. This will not only save farmers time and resources but also reduce chemical usage, promoting sustainable farming practices and lowering the environmental footprint.

Another key advantage of the system is its scalability, allowing it to be adapted for different agricultural environments. This makes it a flexible and valuable tool for managing maize diseases in various contexts.

Additionally, this project will contribute to the broader field of expert systems and precision agriculture, offering insights that can inform future research and technological development in these areas.

1.6 Scope and Limitation of Study

The scope of this study involves the development of a machine learning model for detecting diseases through image processing techniques, the design of an expert system that offers

recommendations for treatment, and the integration of this system with an automated pesticide sprinkler for immediate response. Additionally, the effectiveness and efficiency of the integrated system will be evaluated.

However, the study is limited to maize crops and does not extend to other crop types. Moreover, it does not include the implementation of a leaf structure classification model, which would enhance the system's robustness by ensuring the disease classification algorithm is only activated when a leaf is detected or captured by the system.

1.7 Organization of Study

This project report is organized into five chapters:

Chapter One: Introduction - Provides the background to the study, statement of the problem, aim and objectives, significance of the study, and scope and limitations.

Chapter Two: Literature Review - Presents a comprehensive review of existing literature on maize crop disease recommendation system and related studies.

Chapter Three: System Analysis and Design - Details the analysis of existing systems, justification for the new system, and the design of the proposed maize crop disease recommendation system.

Chapter Four: System Implementation and Testing - Describes the hardware and software implementation process, integration of components, and the various tests conducted to validate the system's performance.

Chapter Five: Summary, Conclusion, and Recommendations - Summarizes the project findings, draws conclusions based on the results, and provides recommendations for future improvements and applications of the system.

1.8 Definition of Terms

i. Maize: A staple crop that is widely cultivated for food and feed, known for its adaptability to different climates and a crucial source of carbohydrates and protein.

ii**.** Convolutional Neural Network (CNN): A type of deep learning model used primarily for image classification and recognition tasks, leveraging layers of filters to detect features such as edges, textures, and patterns in images.

iii. Machine Learning: A subset of artificial intelligence where systems are designed to learn from data and make decisions with minimal human intervention.

iv. Image-Based Classification: The process of using computer algorithms to classify objects, like diseased or healthy plants, based on digital image inputs.

v. PlantVillage Dataset: A publicly available dataset widely used for training machine learning models in plant disease classification, containing labeled images of crops affected by various diseases.

vi. Raspberry Pi: A low-cost, credit-card-sized computer used to power IoT systems, embedded devices, and other small-scale computing projects, often used in automation and control systems like disease detection in agriculture.

vii. Expert System: A computer system that emulates decision-making ability, offering recommendations or solutions based on data input, used in this project for disease detection and management in maize.

viii. Herbicide Sprinkler System: An automated device that applies herbicides to plants when a disease is detected, reducing human intervention and improving response times to disease outbreaks.

ix. Deep Learning: A subset of machine learning that uses neural networks with many layers (deep networks) to model complex patterns in data.

x. Northern Leaf Blight: A fungal disease that affects maize, causing significant yield loss if not treated early, often recognized by large, gray-green lesions on leaves.

xi**.** Gray Leaf Spot: A fungal disease that affects maize, characterized by rectangular lesions on the leaves, leading to reduced photosynthesis and crop yield.

xii. Support Vector Machine (SVM): A supervised machine learning algorithm used for classification tasks, particularly effective in separating data into distinct classes.

xiii. LabelEncoder: A tool used in machine learning to convert categorical labels into numeric form, making them suitable for model training.

xiv**.** Image Preprocessing: The process of preparing an image for analysis, often involving resizing, normalization, and noise removal to enhance image quality for accurate classification.

xv. RGB Image Classification: A method of classifying images based on their red, green, and blue colour channels, often used in detecting plant diseases through image recognition.

xvi. Automated Disease Detection: The use of machine learning and image processing to identify diseases in plants without manual inspection, improving speed and accuracy.

CHAPTER TWO

2.0 LITERATURE REVIEW

2.1 Evolution of Machine Learning

The field of Machine Learning (ML), a branch of Artificial Intelligence (AI), concentrates on creating algorithms that allow computers to learn from data and make decisions without needing explicit programming. Its development has a rich history, evolving from early theoretical concepts to practical applications in today's technology.

2.1.1 Early Beginnings

The idea of machines learning from data emerged in the 1950s, shortly after computers were invented. Alan Turing, in his influential paper "Computing Machinery and Intelligence" (1950), proposed that machines could simulate human intelligence, setting the stage for AI. The first machine learning algorithms were based on symbolic reasoning and logic, with Arthur Samuel creating the first self-learning program in 1959, a checkers-playing program that improved its skills over time. This was the first known instance of the term “machine learning” being used.

2.1.2 Emergence of Neural Networks

The 1960s saw a significant advancement in machine learning with the development of artificial neural networks, which were inspired by the human brain. Frank Rosenblatt’s Perceptron, created in 1958, was an early model designed for binary classification tasks, simulating the function of neurons. While the initial neural networks had limited capabilities, they set the groundwork for the deep learning methods that would emerge in later years.

However, during the 1970s, interest in this area declined due to the shortcomings of early neural networks, especially their struggle with non-linearly separable problems. This era, often called

the "AI Winter," was characterized by decreased funding and enthusiasm for machine learning research.

2.1.3 The Rise of Statistical Learning (1980s-1990s)

The 1980s brought a transition from symbolic AI to a more statistical perspective on machine learning. Researchers started to create probabilistic models and statistical algorithms to handle uncertainty and improve prediction accuracy.

2.2 The Emergence of Maize Disease Detection Systems

Maize is one of the most important staple crops globally, serving as a primary food source for millions of people. However, maize crops are highly susceptible to a variety of diseases, including Northern Leaf Blight, Gray Leaf Spot, and Common Rust, which can lead to significant yield losses if not detected and controlled early. Traditional methods of disease detection in maize rely on manual inspection by farmers or agricultural experts, which can be slow, error-prone, and labor-intensive. To address this, image-based maize disease detection systems, powered by machine learning and artificial intelligence, have emerged as a modern solution to detect and classify diseases efficiently.

2.2.1 Traditional Maize Disease Detection

Before the advent of automated systems, disease detection in maize primarily involved visual inspection of crops. Farmers or agronomists would walk through the fields and manually check the leaves for disease symptoms such as lesions, or fungal growth. While this method can be effective, it is highly dependent on the experience of the individual and is limited in scope, especially for large farms. Additionally, visual inspection is reactive rather than proactive, often identifying the disease at advanced stages when the damage has already been done.

2.2.2 Machine Learning in Maize Disease Detection

Machine learning, especially in the field of image classification, has transformed maize disease detection by automating the identification of diseases from images of maize leaves. These systems are trained on large datasets of maize leaf images, each labeled with the corresponding disease or labeled as healthy. Once trained, the system can classify new images and provide real-time feedback, allowing for early detection and timely intervention.

i. Convolutional Neural Networks (CNNs): CNNs have proven to be highly effective in image-based disease classification due to their ability to automatically learn and extract features from images. In maize disease detection, CNNs are used to analyze patterns, shapes, and textures on maize leaves to distinguish between healthy leaves and those affected by diseases like Northern Leaf Blight, Common Rust, and others. These networks have been at the forefront of image-based maize disease detection systems due to their accuracy and adaptability.

ii. Dataset Training: The effectiveness of these systems relies heavily on the quality and quantity of the dataset used to train them. Large annotated datasets such as the PlantVillage dataset contain thousands of images of maize leaves with varying disease conditions. These images are used to teach the model to recognize disease symptoms even under different environmental conditions (e.g., lighting, angle of the leaf).

2.2.3 The Process of Image-Based Maize Disease Detection

The process of detecting diseases in maize using machine learning typically involves several steps:

i. Image Collection: High-resolution images of maize leaves are captured using cameras, drones, or smartphones. These images serve as the input data for the machine learning model.

ii. Preprocessing: The images are preprocessed to enhance quality and ensure uniformity. This may include resizing the images, normalizing colour values, or filtering out noise (unwanted details). Proper preprocessing ensures that the system can analyze the relevant features of the leaf effectively.

iii. Feature Extraction: Machine learning algorithms, specifically CNNs, automatically extract relevant features from the image, such as colour changes, shapes, or textures that are indicative of disease. This step replaces manual feature engineering and enables the model to learn patterns related to diseases from raw data.

iv. Classification: After feature extraction, the model classifies the image into one of several disease categories (e.g., Northern Leaf Blight, Gray Leaf Spot) or identifies it as healthy. The model's prediction is based on the patterns learned from training data.

v. Result Interpretation: The model provides feedback in the form of a predicted disease class along with a confidence score, indicating the likelihood that the leaf is affected by a particular disease.

2.3 Related Works

Chandana et al., (2020), proposed an integrated approach for smart agriculture that utilizes image processing, expert systems, and IoT to assist farmers in detecting crop diseases. The main goal was to address crop infections caused by bugs, climate conditions, or soil issues. The study implemented a solution that combined an application with expert systems and Faster Region-Based Convolutional Neural Network (Faster R-CNN) for image processing. Notably, the Faster R-CNN demonstrated excellent performance, completing tasks in just 0.2 seconds, making it a suitable choice for future research aiming to enhance similar systems.

Although the study did not clearly outline its limitations, the researchers highlighted potential features for further development, encouraging future work in those areas. They suggested integrating RGB configurations within bounding boxes to improve object detection accuracy, especially for plant and leaf color. Additionally, the authors proposed incorporating SMS and call gateway APIs alongside text-to-voice translation systems to facilitate effective communication with farmers, including those with limited literacy (Chandana et al., 2020).

Çakmak (2020), developed an EfficientNet deep learning model for maize leaf disease classification and compared to other existing algorithms. The study utilized a maize leaf disease dataset of 4,188 original and 6,176 enhanced images to evaluate models like EfficientNet B6, ResNet50, and EfficientNet B3 CNNs based on accuracy, sensitivity, F1-score, and precision. The EfficientNet B6 model achieved 98.10% accuracy on the original dataset, while the EfficientNet B3 reached a maximum accuracy of 99.66% on the enhanced dataset.

Though limitations were not fully discussed, the authors suggested that increasing the dataset size could further improve sensitivity and accuracy, making it a superior choice for similar work. The primary constraints mentioned included the dataset size and the performance reliance on deep learning models.

Asvitha et al., (2024), conducted a survey on various machine and deep learning techniques, such as classification networks, CNNs, and open-source tools, focusing on their applications in plant disease detection. Insights from the study revealed that the EfficientNetV2 and U2 Net Architecture achieved high accuracy (98.26%), although challenges related to data augmentation complexity and limited disease coverage were noted.

The CNN-Adam optimizer exhibited high training accuracy (95.46%) with a small dataset but was constrained by dataset size. Additionally, Artificial Neural Networks trained using

Backpropagation and Extreme Learning Machines achieved 95% accuracy but were limited to images with white backgrounds, highlighting the need for better diversity in training data.

In the study by Asvitha et al., (2024), the Artificial Neural Network (ANN) technique for simultaneous processing and classification achieved a modest accuracy rate of 82.09%, but it faced challenges with wind speed variability and overlapping leaves, which reduced its overall effectiveness. The use of Convolutional Neural Networks (CNN) and Support Vector Machines (SVM) classifiers, analyzing 14 color species and 172 features, achieved a higher accuracy rate of 94.65%. However, this approach was limited to RGB images, and the lighting setup played a crucial role in performance, indicating the need for better image consistency to maintain accuracy (Asvitha et al., 2024).

Hanifudin et al., (2023), proposed a mobile-based expert system that used forward chaining and certainty factor methods to diagnose corn plant diseases. However, the expert system did not utilize image processing, relying instead on users manually inputting data such as symptoms and leaf color, which negatively affected the accuracy. The system achieved an accuracy of 84% after 25 tests, highlighting the need for improvements such as incorporating CNN models and automated image processing to enhance the performance and reliability of the tool (Hanifudin et al., 2023).

Deepika et al., (2021), conducted a comparative review of five AI algorithms—Naive Bayes (NB), Decision Tree (DT), K-Nearest Neighbor (KNN), Support Vector Machine (SVM), and Random Forest (RF)—to detect maize diseases. Among these models, the Random Forest algorithm achieved the highest accuracy, reaching 80.68%. The study used a maize disease detection database consisting of 3,823 images with four disease labels: gray area, common rust, northern leaf blight, and healthy leaves. Although Random Forest demonstrated strong results,

the dataset was limited in size, and upscaling it to 10,000 high-quality images could significantly improve performance (Deepika et al., 2021).

In the study by Khan et al., (2023), a system was developed for leaf blight detection using deep learning and CNN models. The system comprised three modules: image acquisition (using mobile cameras to capture leaf images), pre-processing (which involved noise reduction and contrast enhancement), and classification. The classification module used a pre-trained CNN model to classify leaf images as healthy or diseased and provided a confidence level for blight detection. The system achieved an accuracy of over 90%, making it a reliable tool for farmers, particularly for identifying blight diseases. However, there is room for further refinement by using more extensive datasets and advanced CNN models to improve detection capabilities.

The dataset underwent pre-processing by resizing the images to dimensions of 224 x 224 pixels, aligning with the input specifications of the trained TFLite model. This model was developed using a substantial dataset comprising over 478,930 images through the application of Convolutional Neural Networks (CNN). Subsequent testing on a distinct set of leaf images revealed an impressive accuracy exceeding 90%. However, it's important to note that this implementation was based on an Android application. There is a strong recommendation for transitioning to a more sophisticated embedded system to facilitate quicker treatment responses (Khan et al., 2023).

Budiarianto et al., (2018), explored various image processing features aimed at detecting diseases in corn. Their investigation included color features, such as RGB, as well as local features like SIFT, SURF, and ORB, and object detection methods like HOG. This review is crucial as it addresses uncertainties surrounding the detection of color defects in maize leaves.

Many existing systems struggle with image processing algorithms that fail to accurately identify specific diseases present on maize crops.

Budiarianto et al. assessed the performance of these features using machine learning algorithms, including Support Vector Machines (SVM), Random Forest (RF), and Naive Bayes (NB). They found that implementing an RGB image classifier could significantly enhance the capability of expert systems for maize disease detection in captured images. Their findings indicated that color features, particularly RGB, were the most informative and reliable for diagnosing diseases.

Ashok et al., (2021), introduced an Enhanced K-Nearest Neighbors (EKNN) model that differentiates various disease classes by generating both fine and coarse high-quality features. The model's performance was evaluated using several deep learning metrics, including accuracy, sensitivity, specificity, and Area Under the Curve (AUC). The results demonstrated remarkable metrics, with values of 99.86% for accuracy, 99.60% for sensitivity, 99.88% for specificity, and 99.75% for AUC, highlighting the effectiveness of the EKNN model when coupled with the Plant-Village dataset. To further enhance these results, incorporating additional models and diverse dataset repositories is recommended.

Arora et al., (2020), proposed an innovative automated approach utilizing the Deep Forest technique, which surpasses Deep Neural Networks and other traditional machine learning algorithms in terms of accuracy. Their research, conducted on the Google Colab environment using an NVIDIA Tesla K80 GPU, supports the hypothesis that gcForest can outperform standard deep learning models. The dataset they used consisted of 100 images per class across three disease categories and one healthy class, totaling 400 images. The gcForest model

achieved a remarkable accuracy of 96.25%, demonstrating its effectiveness by merging neural network architecture with the strength of ensemble decision trees (Arora et al., 2020).

In their study, Mwaza et al., (2023), tackled the challenges associated with existing diagnostic methods, which often demand considerable time and expertise from highly skilled professionals. They introduced a Convolutional Neural Network (CNN) model trained on an open-source dataset comprising approximately 5,000 images, including samples of healthy plants. This model demonstrated superior performance compared to established alternatives, achieving an impressive accuracy of 98.7%. The solution was implemented in an Android application environment, and various experiments were conducted to evaluate its effectiveness, confirming the accuracy of the developed solution at 98.7%. The researchers recommended enhancements to the model, suggesting that it be modified to detect additional objects or provide users with an error message indicating that the program is limited to diagnosing maize diseases. They also advocated for an upgrade to the disease treatment module to instill greater confidence in users regarding the solutions provided by the application.

Similarly, Anh et al., (2022), expanded the Crowd-AI disease dataset by increasing the number of plant disease labels and compiling treatment descriptions for each disease category. They subsequently proposed a CNN-based model for diagnosing plant diseases through images and offering effective treatment recommendations based on cross-modal data. Their work led to the development of a web-based application designed to assist farmers and agricultural professionals in more efficiently diagnosing and treating plant diseases.

This application features two primary modules: Plant Disease Classifier (PDC), which predicts plant diseases from images using the EfficientNet-B2 model, and the Plant Disease Recommender (PDR), which suggests various treatments related to the predicted disease.

Experiments conducted on the CrowdAI-ext dataset yielded positive results, demonstrating high accuracy of 92% in disease detection with the EfficientNet-B2 model. This study reinforces the reliability of the EfficientNet-B2 model, establishing it as a leading option for any plant disease detection system (Anh et al., 2022).

Bachhal et al., (2023), utilized a combination of datasets, including the highly regarded Plant Village dataset, alongside a total of 450 images of both diseased and healthy maize leaves, to propose a P-CNN (Pyramid Scene Parsing Network + CNN) model for object detection and semantic segmentation. Their model outperformed other modified segmentation methods, achieving an impressive accuracy of 99.85%. The researchers noted potential improvements that could be made by incorporating additional image segmentation techniques to enhance the accurate identification, classification, and detection of subtle biotic and abiotic lesions.

Maurya et al., (2023), employed a Convolutional Neural Network (CNN) approach using a dataset of 3,997 maize leaf images categorized into four conditions, along with background images without leaves. The dataset included 410 images of Gray Leaf Spot, 788 images of Blight Leaf, 930 Healthy Leaf images, and 915 background images. However, the study identified several limitations related to image-based detection, including challenges posed by the random shapes and irregular sizes of leaves, unclear images, and various noise sources such as inconsistent lighting conditions, shading, and blemishes on the acquired images.

In a related study, Kusumo et al., (2018), sought to mitigate the risks of crop failure due to disease outbreaks that can drastically decrease maize production and result in significant economic losses. They explored various image processing features for disease detection in maize, focusing on color features such as RGB, local features like SIFT (Scale-Invariant Feature Transform) and SURF (Speeded-Up Robust Features), and object detectors such as

HOG (Histogram of Gradients). The researchers assessed these features' performance using machine learning algorithms, including Support Vector Machine (SVM), Random Forest (RF), and Naive Bayes (NB). Their findings indicated that color features, particularly RGB, were the most informative and accurate for detecting diseases in maize leaves. In most instances, the RGB method outperformed other features, highlighting the importance of color in disease detection tasks.

Xu et al., (2021), aimed to enhance the accuracy of maize disease identification by addressing traditional deep learning methods' limitations, such as the need for large sample sizes and low accuracy. They modified the AlexNet model by adding a convolutional layer and a new Inception module to boost feature extraction capabilities. Additionally, they replaced the fully connected layer with a global pooling layer to reduce overfitting caused by excessive parameters. The improved model exhibited superior performance in recognizing maize diseases compared to other CNN models, including VGGNet-16, DenseNet, ResNet-50.

Mingjie et al., (2020), tackled the challenges of extracting lesion features from maize leaves in complex environments by designing a feature enhancement framework. They modified the AlexNet architecture by integrating dilated convolution and multi-scale convolution to improve feature extraction capabilities. Their deep neural network model surpassed traditional classifiers like SEG-KNN and GA-SVM in recognition accuracy. Among the deep models, ResNet50 achieved an accuracy of 95.47%, slightly lower than the proposed method, while GoogleNet and VGG16 reached 94.06% and 93.98%, respectively. The proposed DMS-Robust AlexNet, leveraging dilated convolution and multi-scale convolution, achieved an impressive accuracy of 98.62%, outperforming other baseline models (Mingjie et al., 2020).

Masood et al., (2023), addressed the challenges associated with automated maize disease recognition, which is crucial for preventing crop production losses. They developed an improved Faster-RCNN approach using the ResNet-50 model as its base network for calculating deep key points. The proposed MaizeNet model was tested on the standard Corn Disease and Severity database, which includes samples from three different classes of maize plant diseases captured under diverse conditions. The experimental results demonstrated the effectiveness of the MaizeNet model, achieving an average accuracy of 97.89% and a mean average precision (mAP) of 0.94.

In the work of Pate et al., (2023), a supervised machine learning approach was developed for classifying maize plant leaf diseases using texture analysis and feature extraction techniques. This approach utilized four algorithms: Decision Tree, Gradient Boosting, Support Vector Machine (SVM), and K-Nearest Neighbors (KNN) on the highly efficient Plant-Village dataset. The results highlighted the effectiveness of this method in quickly diagnosing diseases, providing valuable information for farmers seeking to maximize their yields.

The incorporation of advanced deep learning techniques, particularly Convolutional Neural Networks (CNNs), has the potential to significantly enhance classification accuracy by enabling the model to learn hierarchical features directly from raw visual data. Additionally, broadening the dataset to encompass a greater variety of maize leaf diseases and various stages of disease progression can enhance the system's robustness and applicability across diverse agricultural settings.

Kariuki (2022), proposed an ontological engineering-based model for maize disease classification, tackling the complexities and uncertainties inherent in disease classification. This model employs the OWL DL language to develop a maize diseases ontology, providing a

formal specification for classification. The ontology was created using the Protégé 5.0 tool, resulting in a knowledge representation that is both intelligent and shareable. This ontology can effectively classify maize diseases, thereby contributing to increased productivity and yield.

In a study by Sun et al., (2020), researchers utilized a multi-scale feature fusion instance detection method based on CNNs. Their methodology involves dataset preprocessing, fine-tuning the network, and detection, supplemented by a transmission module that connects the fine-tuned network to the detection module to enhance accuracy and efficiency.

The authors reported that the model achieves a mean average precision (mAP) of 91.83% after 60,000 iterations, outperforming existing methods in precision and frames per second (FPS). This approach demonstrates improved detection of maize leaf blight in challenging field environments. Furthermore, the model's adaptability for integration into embedded systems provides a theoretical foundation for developing precise drug application technologies and detection robots for maize leaf blight (Sun et al., 2020).

Qureshi (2023), introduced an intelligent deep learning model utilizing CNNs to distinguish between resistant and susceptible stalks to stalk rot. However, the model attained an accuracy of only 83.88%. This approach has the potential to reduce the reliance on professional assessments, allowing farmers to estimate fungicide requirements and explore resistant breeding programs (Qureshi 2023).

Abirami (2022), implemented a deep learning framework using the AlexNet model to classify maize diseases into four categories: Common Rust, Northern Leaf Blight, Cercospora Leaf Spot Grey, and Healthy Leaves. The model achieved a commendable accuracy rate of 96%, establishing it as a practical solution for protecting maize crops and providing valuable support

to farmers. However, this accuracy is slightly below the threshold achieved by some prior research in maize leaf disease detection systems. This study also explored AlexNet and ResNet50 as edge solutions, utilizing the Plant-Village dataset.

Amin et al., (2022) introduced an end-to-end deep learning model for classifying corn leaf diseases by employing two pre-trained Convolutional Neural Networks (CNNs): EfficientNetB0 and DenseNet121. The researchers extracted deep features from corn plant images using these models and utilized a standard technique of feature fusion through concatenation, resulting in a more complex feature set.

The model was trained on a diverse dataset of corn plant images afflicted by various diseases and evaluated based on classification accuracy, achieving an impressive 98.56%. This performance surpassed that of other pre-trained models such as ResNet152 and InceptionV3, highlighting the effectiveness of their approach. The study reinforces the accuracy of CNN models and advocates for their application in developing expert systems for plant leaf disease detection, particularly when supported by large and high-quality datasets.

Njazi et al., (2023), examined the efficacy of machine learning and computer vision techniques for the early detection of maize plant diseases. They utilized a dataset comprising 4,188 images, including 1,306 images of Common Rust, 574 images of Gray Leaf Spot, 1,146 Blight images, and 1,162 healthy images, compiled from the Plant Doc and Plant Village datasets. Implementing this work involved tools such as Raspberry Pi, PyTorch, Python, and MATLAB libraries. The researchers selected a CNN based on the ResNet model, specifically the ResNet50 variant, which is 50 layers deep. The model achieved an accuracy of 96.23%. The authors noted that with a larger dataset, this accuracy could potentially be improved, indicating

that enhancements in both data volume and algorithm selection could lead to superior performance.

Sukhwinder et al., (2022), proposed a deep transfer learning approach for detecting plant diseases, employing the VGGNet-19 model, which was pre-trained on the ImageNet dataset. The authors suggested that augmenting VGGNet19 with additional layers could yield better results for classifying and detecting plant diseases.

Atila et al., (2021), introduced an EfficientNet architecture (B4 & B5) aimed at classifying plant leaf images across 39 classes in the Plant-Village dataset. Their work compared the success of this architecture with state-of-the-art deep learning models used in plant disease detection. Experimental studies were conducted on both original and augmented versions of the Plant-Village dataset. Based on average accuracy and precision metrics, the B4 and B5 models outperformed other CNN architectures.

The B5 model achieved 99.91% accuracy and 98.42% precision on the original dataset, while the B4 model attained 99.97% accuracy and 99.39% precision on the augmented dataset. Notably, even with input images resized to 132 × 132 due to hardware constraints, the EfficientNet models produced better results than other CNNs that utilized higher-resolution images. This study emphasizes the effectiveness of EfficientNet B4 and B5 models for efficient classification and detection of plant leaf diseases.

Budianto et al., (2020), presented an expert system built on the CodeIgniter framework and MySQL database, utilizing the Naive Bayes method for early disease detection in corn plants. They noted that the accuracy of the system could be influenced by the quality and completeness of the training data for the Naive Bayes classifier, and its effectiveness might vary in real-world conditions based on environmental factors and disease prevalence. The system achieved a 92%

accuracy rate with the Naive Bayes classifier. The authors suggested that integrating CNNs, advanced image processing techniques, and a larger dataset could enhance the system's efficiency and accuracy.

In a study by Akanshkha et al. (2021), an efficient automated diagnosis system for maize plants was proposed, comprising four stages: preprocessing, feature extraction, classification, and segmentation. The initial phase involved converting images to RGB format and removing noise, followed by R-band extraction for feature extraction. Selected attributes were then inputted into a classifier to distinguish between normal and abnormal images.

They utilized an Optimized Probabilistic Neural Network (OPNN) for classification, which was improved using the Artificial Jelly Optimization (AJO) algorithm. The solution achieved a sensitivity rate of 98% in detecting maize crop diseases, attributed to the parameter optimization of the PNN. The researchers indicated plans to identify various types of diseases present in maize plants in future studies.

Jitong et al., (2023), introduced FCA-EfficientNet, an enhanced model built upon EfficientNet that integrates a fully-convolutional coordinate attention module. This addition allows the model to better concentrate on disease-affected regions within complex backgrounds. The model also incorporates an adaptive fusion module that merges information across different scales, thereby improving the accuracy of disease recognition. In comparison to other deep learning models, FCA-EfficientNet exhibits superior performance across various metrics, including accuracy, precision, recall, and F1 score, all while maintaining a low parameter count and computational cost, making it suitable for real-time applications.

In tests conducted on a corn disease dataset, FCA-EfficientNet achieved a classification accuracy of 98.78% for six corn images, with all evaluation metrics surpassing those of

classical CNN models and demonstrating good generalization capabilities. The model was successfully deployed on an Android device, yielding an average recognition speed of 92.88ms, which meets user expectations. Although it is limited to an Android application, integrating this model into an embedded system could enhance its functionality (Jitong Cai et al., 2023).

2.4 Summary of Reviewed Research Works

Table 1.0: Systematic Literature review

S/N	Author Details	Objective of Study	Methodology	Results	Limitation
1	Mwaza1 et al., (2023)	The study aims to develop a reliable and effective mobile application for early and accurate identification of maize diseases using Convolutional Neural Networks (CNN	The methodology involves training models using an open-source library of approximately 5000 images, including healthy plant samples	The CNN model achieved an impressive total accuracy of 97%, outperforming other established models	Low accuracy, uses internet, not embedded and slow.
2	Bachhal et al., (2023)	Computer vision-based system for early detection of	Plant Village dataset and deep learning models, specifically a	The proposed P-CNN model achieved an	The study acknowledges the challenges in

		maize plant diseases.	Pyramid Scene Parsing Network (PSPNet) and Convolutional Neural Networks (CNN), for disease detection.	accuracy of 99.85% in detecting and classifying maize leaf diseases, outperforming other models like YOLO+CNN and VGG19+CNN.	segmenting and detecting lesions on maize leaves due to factors like complex backgrounds and diverse symptoms, and suggests future research to improve accuracy and address semantic segmentation shortcomings.
3	Patel et al., (2023)	The study aims to classify maize leaf diseases using supervised machine learning algorithms, contributing to automated plant disease detection	It employs texture analysis with Gray-Level Co-occurrence Matrix (GLCM) and Gabor feature extraction on the Plant-Village dataset. Four algorithms—Decision Tree,	The study found that Decision Tree and Gradient Boosting algorithms performed best, with Gradient Boosting	The paper does not discuss limitations, but generally, such studies might be limited by the size and diversity of the dataset, the complexity of

		systems and aiding in sustainable agriculture.	Gradient Boosting, SVM, and KNN—are used for classification.	achieving the highest accuracy of 95.05% for GLCM features at a 135-degree orientation angle.	disease patterns, and the adaptability of the model to real-world conditions. Future work could include expanding the dataset and exploring deep learning techniques.
4	Chauhan, et al., (2021)	The study aims to develop a model for early detection and differentiation of maize leaf diseases using various supervised machine learning algorithms, to help increase	The methodology includes image acquisition, pre-processing, segmentation, feature extraction, and classification using algorithms like Naive Bayes, K-Nearest Neighbor, Decision Tree, Support Vector Machine, and Random Forest	The Random Forest algorithm achieved the highest accuracy of 80.68% in disease detection, outperforming other methods	The paper acknowledges potential pitfalls with each model that may not be applicable to all datasets and suggests future implementation using high-dimensional datasets for

		agricultural productivity.			improved differentiatio
5	Qureshi et al., (2023)	The study aimed to develop a Convolutional Neural Network (CNN) model to classify maize stalks as resistant or susceptible to stalk rot disease, reducing the need for expert intervention.	Researchers conducted field experiments at the Maize and Millet Research Institute, Yousafwala, Sahiwal, using smartphone images of maize germplasm. They employed CNN to process these images and classify them into resistant or susceptible categories.	The CNN model achieved an accuracy of 83.88%, with recall ratios and precision values indicating significant effectiveness in classifying stalk rot severity. The model outperformed traditional farmer assessments.	• The study suggests that the number of images and the balance among severity classes are crucial for model training. It also notes the challenges in collecting a diverse set of images for all disease severities due to environmental factors. The study emphasizes the need for further research to optimize CNN model training for

					various crops and conditions.
6	Chandana et al., (2023)	The study aims to integrate image processing with smart agriculture to assist farmers in identifying crop diseases using a faster region-based convolutional neural network (CNN) and expert systems.	The approach combines object recognition methods for disease identification, annotated with infection terminologies, and employs Faster R-CNN for quick and efficient image processing.	The system was tested on maize crops, successfully identifying various infections and diseases. Faster R-CNN demonstrated high performance, operating within 0.2 seconds for object recognition.	Future work includes enhancing accuracy with RGB configurations and integrating communication APIs for broader farmer assistance.
7	Khan et al., (2023)	The study aims to develop a smart system for early detection of leaf blight in crops using deep	The system includes an image acquisition module using a mobile camera, a pre-processing module for image	The proposed system achieved over 90% accuracy in detecting leaf blight diseases. It	GPS and drones, and enhancing data processing with cloud-based solutions for comprehensive

		learning and convolutional neural networks (CNN), implemented on an Android platform, to help farmers prevent disease spread and improve crop yields.	enhancement, and a leaf blight classification module employing a pre-trained CNN model to classify images into healthy or diseased categories with a confidence level.	was integrated into an Android application, providing a user-friendly interface for real-time detection and classification of leaf blight.	disease monitoring
8	Rashid et al., (2023)	The study aims to develop an automatic and smart solution for classifying corn leaf diseases in precision agriculture, enhancing crop production by accurately detecting and	The researchers introduced a CNN-based architecture, the Multi-Model Fusion Network (MMF-Net), which integrates multi-contextual features using RL-block and PL-blocks 1 & 2, combining different model streams	The MMF-Net achieved an impressive 99.23% accuracy in classifying corn leaf diseases, demonstrating its effectiveness as a promising solution for identifying plant leaf diseases in	online notification system with edge computing

		classifying diseases early.	trained on heterogeneous data	precision agriculture.	
9	Çakmak et al., (2024)		The research utilized a dataset of 4,188 original and 5,932 augmented images of maize leaves, training various deep learning models including EfficientNet, ResNet50, VGG19, DenseNet121, and Inception V3, and comparing their performance based on accuracy, sensitivity, F1-Score, and precision.	The EfficientNet B6 model achieved the highest accuracy of 98.10% on the original dataset, while the EfficientNet B3 model reached 99.66% accuracy on the augmented dataset, indicating the effectiveness of data augmentation in improving model performance.	The paper suggests a need for models that can be trained quickly, have fewer parameters, and exhibit high performance, indicating ongoing challenges in optimizing deep learning architectures for plant disease classification.
10	Arora *et al.*, (2020)	The study aims to recognize and classify maize	The researchers tested the Deep Forest algorithm by	The Deep Forest model achieved an accuracy of	The study was conducted on a reduced dataset of

		leaf diseases using the Deep Forest algorithm, which is a significant improvement over manual classification and other less accurate techniques	varying hyperparameters like the number of trees, forests, and grains4. They compared the model with traditional machine learning models and deep models like CNN and LeNet5	96.25% with optimal hyperparameters (1000 trees, 4 forests, 3 grains), outperforming other models in accuracy and demonstrating low dependency on hyper-parameter tuning and dataset size	400 images due to the high memory requirements of the Deep Forest model, indicating a potential limitation in handling larger datasets.
11	Noola *et al.*, (2022)	The study aims to develop an automated method for early detection of corn leaf diseases, which are critical for maintaining corn crop yield and quality.	The researchers designed an Enhanced k-Nearest Neighbor (EKNN) model, improving upon the basic KNN model	The EKNN model achieved high classification accuracy, sensitivity, specificity, and AUC (Area Under Curve) values, outperforming	Low accuracy in practical disease detection

				traditional methods like AlexNet and DMS-Robust AlexNet in precision, recall, and F1-score metrics	
12	Masood *et al.*, (2020)	addressed the challenge of accurately recognizing various maize plant leaf diseases, which significantly impact crop quality and production.	Researchers proposed 'MaizeNet,' a deep learning model based on an improved Faster-RCNN approach with ResNet-50 and spatial-channel attention mechanisms for feature extraction.	MaizeNet demonstrated high effectiveness, achieving an average accuracy score of 97.89% and a mean average precision (mAP) value of 0.94 on the Corn Disease and Severity database.	The effectiveness of current methods depends on external factors and suffers when used on photos with cluttered backgrounds.
13	Jitong *et al.*, (2020)	The study aims to tackle the	DMS-Robust Alexnet, which	The proposed method	data imbalance, overfitting, and

		challenges in identifying maize leaf diseases due to environmental factors and proposes a novel recognition method combining feature enhancement with a robust neural network model named DMS-Robust Alexnet.	utilizes dilated convolution and multi-scale convolution for improved feature extraction. Batch normalization and PRelu activation function are employed to enhance model robustness.	demonstrated strong robustness against natural environment variations and achieved high accuracy in identifying maize leaf diseases, outperforming traditional classification methods and neural networks without image enhancement.	the need for extensive computational resources
14	Maurya *et al.*, (2023)	The study aims to enhance corn yield by developing an intelligent corn leaf disease detection system	The researchers employed CNN models to train and test on a dataset of 3997 corn leaf images, categorized into common rust,	The proposed system achieved exceptional efficiency in early disease detection, which is crucial for	Challenges include random shapes and sizes of leaves, unclear images, and various noise sources such as

		using advanced deep learning techniques, specifically Convolutional Neural Networks (CNNs), to address the limitations of human visual detection	grey leaf spot, blight, healthy, and background images	timely intervention and effective disease control, thereby securing agricultural productivity and global food supply.	irregular lighting and shading, which can affect the accuracy of image-based leaf disease detection
15	SUN *et al.*, (2020)	The study aims to detect NLB in maize under complex field environments, addressing issues like high-intensity light and small target disease detection.	Dataset preprocessing with improved retinex, fine-tuning the network with an improved Region Proposal Network (RPN), and a detection module that uses a transmission module for feature sharing and fusion.	After 60,000 iterations, the model achieved a mean average precision (mAP) of 91.83% and improved frames per second (FPS), indicating high precision and efficiency	The model's real-time detection standard is yet to be fully established for intelligent agriculture production processes.

CHAPTER THREE

3.0 SYSTEM ANALYSIS AND DESIGN

3.1 Conceptual Framework

This section covers the development of recommendation system for a detected maize crop disease. The system is designed to detect maize crop disease and rapidly applying treatment response to the infected maize crop plant without the physical intervention of the farmer.

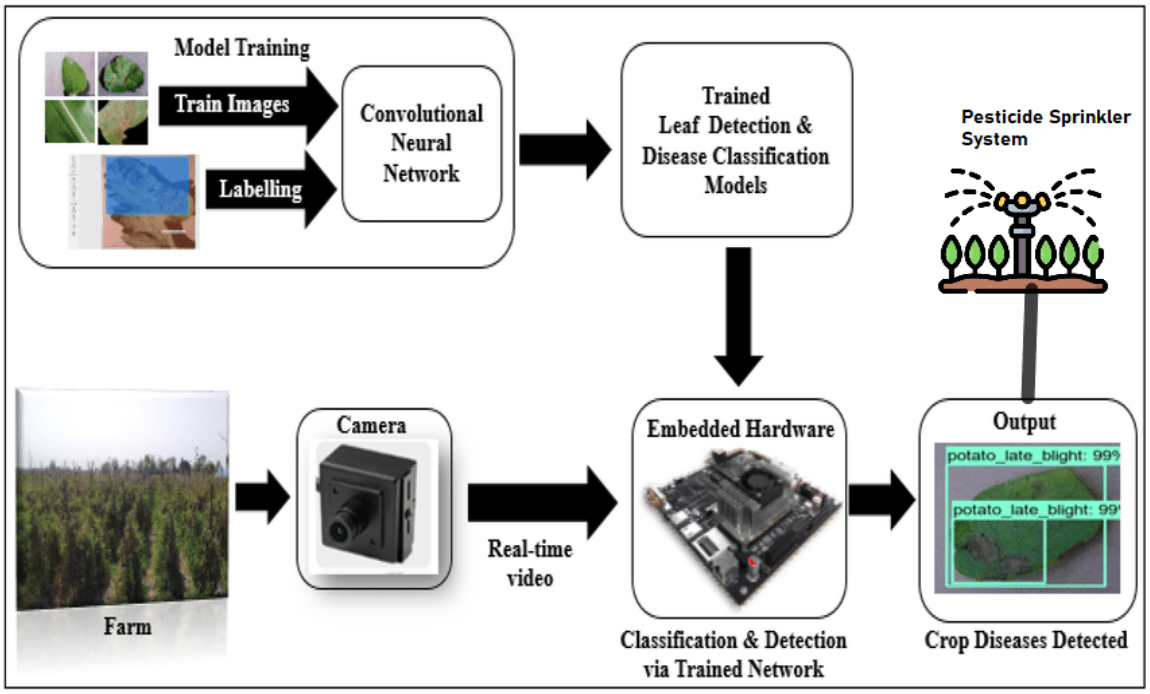


Figure 3.1: System Architecture

3.1.1 Analysis of Existing Systems

Existing systems have been able to tackle the problem of leaf blight in maize crops, by using Convolutional Neural Networks (CNN) machine learning approach. This system is used to detect

leaf blight in crops by capturing images of maize plant leaves using an Android platform, and then employing a pre-trained CNN model to classify images as either blight-infected or healthy.

This is achieved by training the CNN model on a dataset of labeled images, and then using the trained model to make predictions on new, unseen images.

The images captured at various stages are stored in a database and then fed into the CNN model in order to carry out analysis, determine the minimum and maximum thresholds of each image feature that affects the overall health of the crops. The data is also used to determine the optimal thresholds that best detect leaf blight and prevent crop damage.

3.1.2 Limitations of Existing of Systems

The existing system was able to solve the problem of leaf blight detection in maize crops by successfully analyzing and classifying images of leaves as healthy or blight-infected using CNN. However, the system has several limitations.

One major limitation is its slow processing speed, which results in delayed detection and response. Additionally, the system's dependence on cloud-based infrastructure may have limited accessibility in rural areas, hindering real-time monitoring and prompt action. Furthermore, the system is IoT-based but not autonomous, requiring constant human intervention and lacking self-regulation capabilities.

Another limitation is that the system is limited to an Android-based mobile app, excluding other platforms and users. Lastly, the system lacks an integrated herbicide sprinkler system for automatic treatment of infected plants, relying on manual intervention.

3.1.3 Justification of the New System

The primary reason for the need of a new system is to offer a significant upgrade over existing systems by providing an autonomous monitoring and treatment system that can detect and respond to three common maize diseases: blight, spot, and rust. This system integrates an

automatic herbicide sprinkler for rapid treatment response, ensuring timely and effective action against disease outbreaks.

Unlike existing systems, our proposed system does not rely on internet connection or cloud-based databases, but instead utilizes physical hardware components for offline execution. This enables real-time monitoring and treatment of maize diseases, even in areas with limited or no internet connectivity.

The system continuously monitors the maize crops and detects early signs of blight, spot, and rust. Upon detection, the system automatically triggers the herbicide sprinkler to apply the appropriate treatment, preventing the disease from spreading and causing damage to the crops.

The data collected by the system can be stored locally and used for future research and analysis, providing valuable insights into maize disease management.

3.1.4 Description of the New System

The proposed system is designed to enhance the existing system by preventing factors that lead to maize crop damage and reducing the impact of blight, spot, and rust.

This will improve grain quality and quantity, preventing food shortage and reducing hunger, in line with the Sustainable Development Goal (SDG) 2 of the United Nations (UN).

The features of the new system are as follows:

i. The system is designed to detect and respond to blight, spot, and rust diseases in maize crops.
ii. The system integrates an automatic herbicide sprinkler for rapid treatment response.
iii. The system provides offline execution capabilities, eliminating reliance on internet connectivity or cloud-based databases.

3.2 System Design

The Software Development Life Cycle (SDLC) methodology employed for this project is the bottom-up approach. This approach focuses on assembling simpler systems to create more complex systems, where the original systems become subsystems of the newly developed system. In essence, it emphasizes developing lower-level components and units first, which are then integrated to form higher-level systems.

The bottom-up methodology highlights the importance of creating reusable components, allowing for early testing of these components during the development cycle. This approach also facilitates managing changes in lower-level components without significantly impacting the entire system.

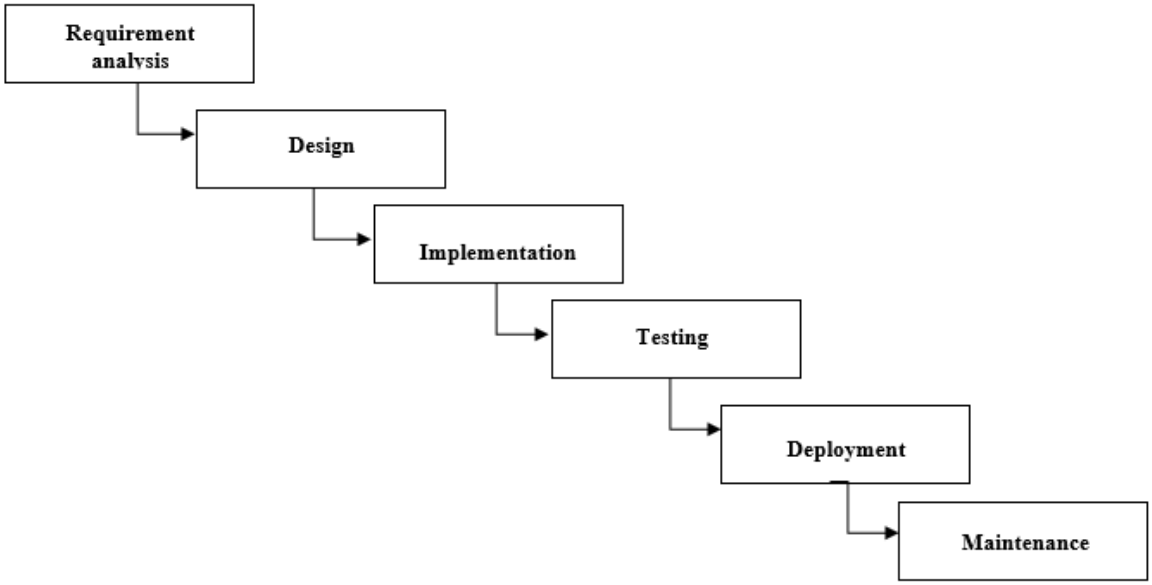


Figure 3.2: Bottom-up Methodology

i. Requirements Analysis: in this phase focuses on the requirement of the proposed system. Requirements are identified and documented in a detailed manner to help understand what will be needed in the system and how each component will function.

ii. Design: the design phase focuses on the technological needs of the system such as individual components and modules of system, programming languages used, interactions for each components and overall system architecture.

iii. Implementation**:** in this phase, the individual modules and components of the system are developed. This is done by integrating this lower-level components together to form the desired system. Coding of components also takes place at this stage.

Implementation phase focuses on creating reusable and well-tested units and also ensure that each component meets its specified requirements and design.

iv. Testing**:** the testing phase includes testing each unit of system as well the integrated system as a whole to ensure that the system meets the overall requirements and the system is functioning properly. This phase also involves testing system performance, accuracy and reliability.

v. Deployment**:** here the fully tested system is deployed to the production environment and also ensure that the deployment process is smooth and error-free.

vi. Maintenance**:** in this phase, the developed system is been monitored in its production environment, regular maintenance tasks are performed on system to ensure system stability and performance.

3.3 Hardware Components

3.3.1 Raspberry Pi ZeroCamera Module (RPCAM)

The RPCAM, is a compact camera module designed for the Raspberry Pi Zero and other Raspberry Pi models. It features a 5-megapixel Omnivision OV5647 sensor, capable of

capturing high-quality images and videos. The RPCAM supports various video modes, including 1080p at 30fps, 720p at 60fps, and 640x480p at 90fps, as well as still image modes in JPEG and RAW formats.

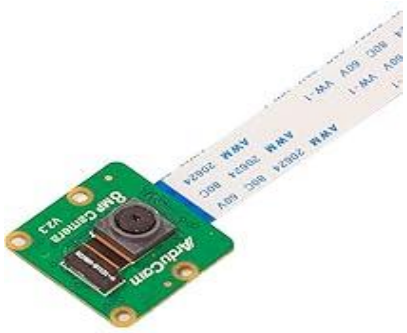


Figure 3.1: Rasberry Pi ZeroCamera Module (RPCAM)

Table 3.1: Raspberry Pi ZeroCamera Module (RPCAM) and Variations

Aspect	**ZeroCamera Module**	**Raspberry Camera**	**Arducam**	**OpenMV Camera**
Flexibility	Compact design, compatible with Raspberry Pi Zero and other models	Interchangeable lenses, compatible with Raspberry Pi 4 and other models	Wide range of lens options, compatible with various microcontrollers and SBCs	Modular design, compatible with various platforms, including Raspberry Pi and Arduino

Accuracy	5-megapixel sensor, good for general-purpose photography and videography	12-megapixel sensor, higher resolution and better image quality	Up to 16-megapixel sensor, high-quality images and videos	2-megapixel sensor, optimized for machine vision and object detection
Scalability	Limited by its fixed focus lens and smaller sensor size	More flexible with interchangeable lenses, suitable for various applications	Highly scalable, suitable for various applications, including industrial and commercial use	Designed for scalability, suitable for industrial and commercial applications
Cost	It is low cost compared to other temperature sensors.	Cheaper but require more complex signal conditioning.	Cheaper but require more complex signal conditioning.	expensive because it requires additional components

3.3.2 Raspberry Pi Zero Module Board

The Raspberry Pi Zero Module, is a small, low-cost, and highly capable single-board computer (SBC) designed for various applications such as robotics, IoT projects, and embedded systems. It features a Broadcom BCM2835 processor, 512MB RAM, and a small form factor of 65mm x 30mm. The board also includes various interfaces such as USB, HDMI, and GPIO pins, making it a popular choice for developers and makers.

Figure 3.2: Rasberry Pi Zero MicroBoard Module

Table 3.2: Rasberry Pi Zero MicroBoard Module and Variations Comparison

Aspect	**Rasberry Pi Zero**	**Raspberry Pi Zero W**	**Raspberry Pi 3**	**Raspberry Pi 3**
Flexibility	Compatible with a wide range of applications and projects	Adds Wi-Fi and Bluetooth capabilities for wireless projects	More powerful processor and memory for complex projects	More powerful processor and memory for complex projects
Accuracy	High performance and reliability	High performance and reliability	High performance and reliability	High performance

				and reliability
Scalability	Limited by its small form factor and resources	Limited by its small form factor and resources	Highly scalable for various application	Highly scalable for various application
Cost	Affordable	Less affordable	Expensive	Highly Expensive

3.3.3 5V DC Mini Water Pump

5V DC Mini Water Pump, is a small, low-voltage pump designed for various applications such as DIY projects, robotics, and small-scale water circulation. It features a compact design, low power consumption, and a high lift height, making it suitable for a wide range of uses.

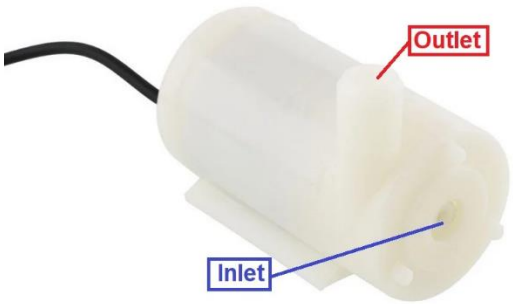


Figure 3.3: 5v DC water pump

Table 3.3: 5v DC water pump vs Variations Comparison

Aspect	**Mini Water Pump (DC 5V 1A)**	**Mini Water Pump (DC 5V 2A)**	**Submersible Water Pump (DC 5V 3A)**	**Brushless Mini Water Pump (DC 5V 4A)**
Flexibility	Compatible with various small-scale water circulation projects	Compatible with various small-scale water circulation projects	Can be used submerged in water for various applications	High-performance brushless motor for demanding applications
Accuracy	Precise water flow control	Precise water flow control	Precise water flow control	• Precise water flow control
Scalability	Limited by its small size and power rating	More scalable than the 1A version	Highly scalable for various applications	Highly scalable for various applications
Cost	Very cheap and Affordable	Cheap and Affordable	Less affordable	Expensive

3.3.4 Lithium Power Battery

Lithium Power Battery, is a rechargeable battery designed for use in portable power banks, providing energy storage for on-the-go charging of devices. It features a high capacity, long cycle life, and compact design, making it suitable for various applications such as consumer electronics, outdoor activities, and emergency power backup.

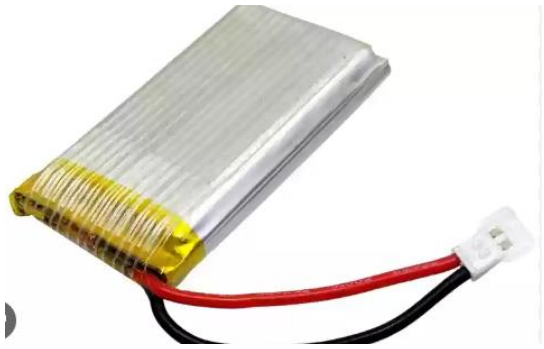


Figure 3.4: 10,000mAh Power Board Battery

Table 3.4: 10,000mAh Power Board Battery vs Variations Comparison

Aspect	**Lithium-Polymer Battery (10000 mAh)**	**18650 Lithium-Ion Battery (3000 mAh)**	**18650 Lithium-Ion Battery (2500 mAh)**	**26650 Lithium-Ion Battery (5000 mAh)**
Flexibility	Compatible with various power bank designs	Compatible with various power bank designs	Compatible with various power bank designs	Compatible with various

				power bank designs
Accuracy	Precise voltage and current control	Precise voltage and current control	Precise voltage and current control	• Precise voltage and current control
Scalability	Highly scalable	More scalable than 2500 mAh version	Limited by capacity	Highly scalable for high-capacity power banks
Cost	Less expenisve	Less affordable	Affordable	Less Expensive

3.4 Software components

3.4.1 Python

Python is a programming language that is capable of running on microcontrollers and embedded systems, it employs much of the standard simple and easy to understand syntax. Python is well-suited for IoT applications, educational purposes, and rapid prototyping, offering a high-level, user-friendly programming environment for resource-constrained devices.

Table 3.5: Python vs Micro python

ASPECT	PYTHON	MICRO PYTHON
Memory Usage	Requires more RAM and storage	Optimized for microcontrollers with limited RAM
Real-Time Interactivity	REPL available but less suited for embedded use	Interactive REPL for quick testing and debugging
Resource Constraints	Requires more computational resc and memory	Operates well within severe resource constraints
Embedded Systems	Primarily designed for general-purpose computing	Specifically designed for embedded systems
Portability	Runs on general-purpose operating systems (Windows, MacOS, Linux)	Runs on various microcontroller platforms
Deployment	More complex deployment in environments	Easier deployment on microcontroller-based projects

3.4.2 VSCode IDE

VSCode offers a seamless experience for writing scripts for microcontrollers, thanks to its user-friendly interface, extensive library of extensions, and advanced coding features. The code completion and IntelliSense capabilities help streamline the writing process, while the debugging and testing tools ensure that scripts are thoroughly verified. The integrated terminal and serial monitor enable easy communication with microcontrollers, and the code formatting and organization features keep scripts tidy and readable.

3.5 System architecture

The system consists of a Raspberry Pi Zero module which is the brain or microcontroller of the system, it receives real-time data (Image Video Frames) from the RPCAM camera sensor connected to the raspberry pi zero board. Integrated on the microcontroller is an electric herbicide sprinkler which is triggered by the microcontroller based on certain conditions. The microcontroller processes the sensor data using python machine learning CNN algorithm coded into the board for classifying and recommending disease.

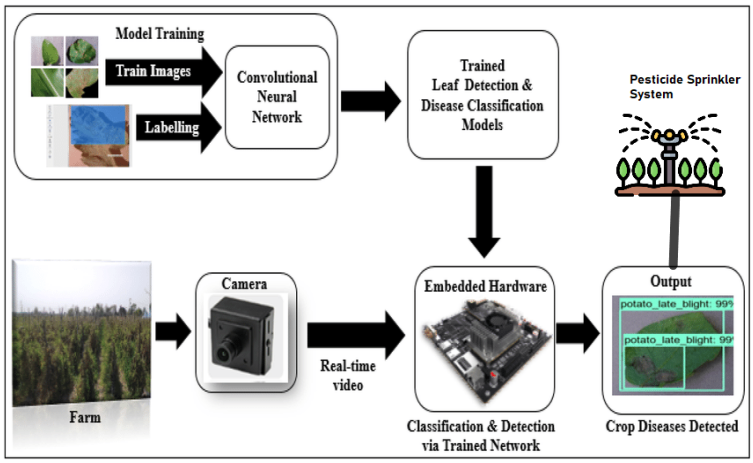


Figure 3.6: System Architecture

3.5.1 System flow representation

The flowchart outlines the operational sequence of the system. When the system is turned on, every component of the system is then activated. If the microcontroller O.S(operating system) boots up, the camera proceeds to take video frame readings of the maize plant being captured. The readings or video frames captured are transmitted and saved in the O.S shared storage and in the machine learning algorithm detects new capture, analyzes the video frames for processing and making

recommendations. The recommendations are then interpreted into signals which powers off and on the herbicide sprinkler system. The system's flowchart is shown in the figure below:

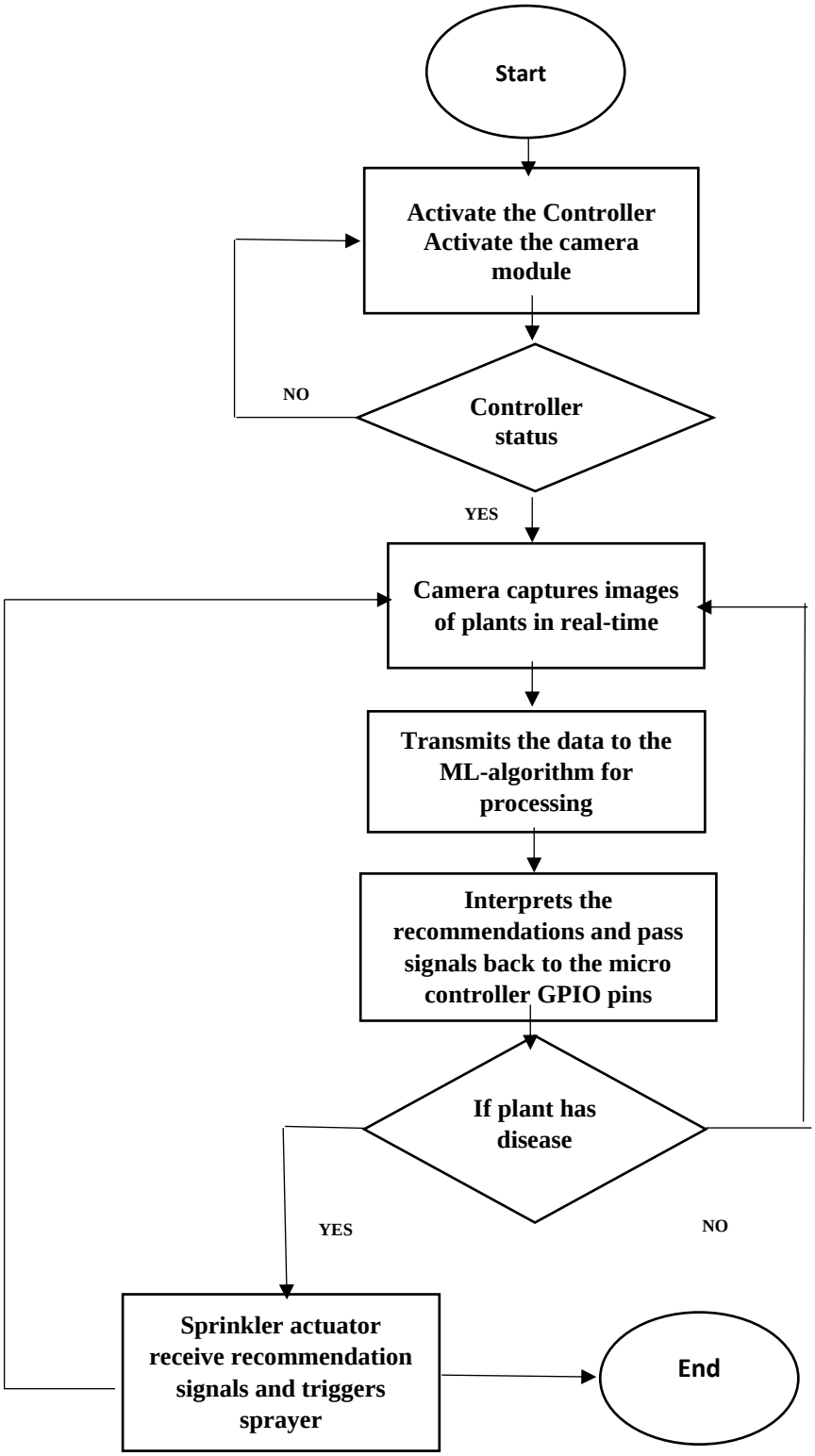


Figure 3.7: System flow representation

3. 6 Use Case Diagram

The use case diagram for the recommendation system for a detected maize crop disease outlines the key properties. During the execution stage, the system captures video frames of the maize

plants in real-time and stores it in the shared storage directory on the microcontroller O.S for processing. The microcontroller then uses the machine learning image processing algorithm to analyze the captured images and check for defects.

If disease is detected, microcontroller sends a signal to the actuators. The herbicide sprinkler sprays chemical for some seconds and then automatically turns off and the whole system restarts the execution all over again until either the system is turned off.

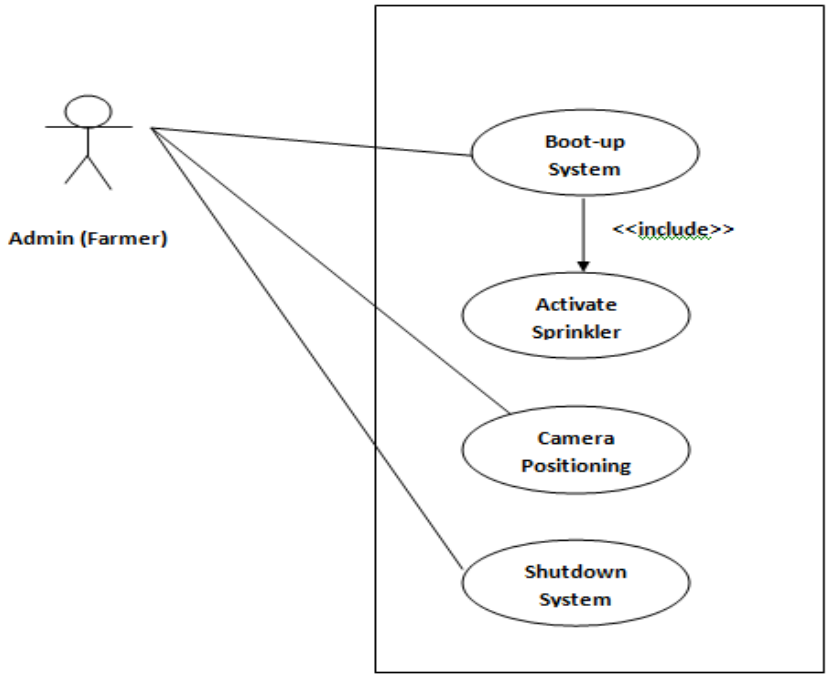


Figure 3.8: Use case diagram

CHAPTER FOUR

4.0 RESULTS AND DISCUSSION

4.1 Introduction

This chapter contains an in-depth discussion of how the model was developed alongside its training and evaluation, the code, the chosen programming language, software and hardware requirement and also the experimental result gained.

4.2 Choice of Programming Language

The programming language used for the implementation of the proposed system is Python programming language. Python is a versatile, high-level programming language known for its readability and simplicity.

Created by Guido van Rossum and first released in February 20 1991, it supports multiple programming paradigms and is widely used for web development, data analysis, AI, and more. Its clear syntax, active community, and extensive libraries makes it beginner-friendly and powerful which runs on various platform like Windows OS, and various versions of UNIX and MAC operating system.

4.3 System Requirements

The system requirements for users who intend to make use of this system is sub divided into two namely: Hardware and Software

4.3.1 Hardware Requirements

The system implementation and testing were carried out using Windows 10 Pro Operating System(64bits), 4GB of RAM, Core DUO, 1.47GHz processor speed and storage space of

480GB(HDD). Raspberry Pi Zero Microcomputer with 500MB of RAM and peripheral ports (USB, HDMI and Micro SD card module slot). Raspberry PI Zero camera. 5volts One Channel compatible relay module. 5volts mini water pump. 10,000mAh power board for power supply.

4.3.2 Software Requirements

The implemented system was developed using Visual Studio Code as the Integrated Development Environment version 1.84, Python 3.12 and some essential python libraries such as Scikit-image, Sklearn and Numpy, GPRICO. Figure 4.1 and Figure 4.2 shows the start page of Visual Studio Code Integrated Development Environment (IDE) used for implementation and code environment respectively.

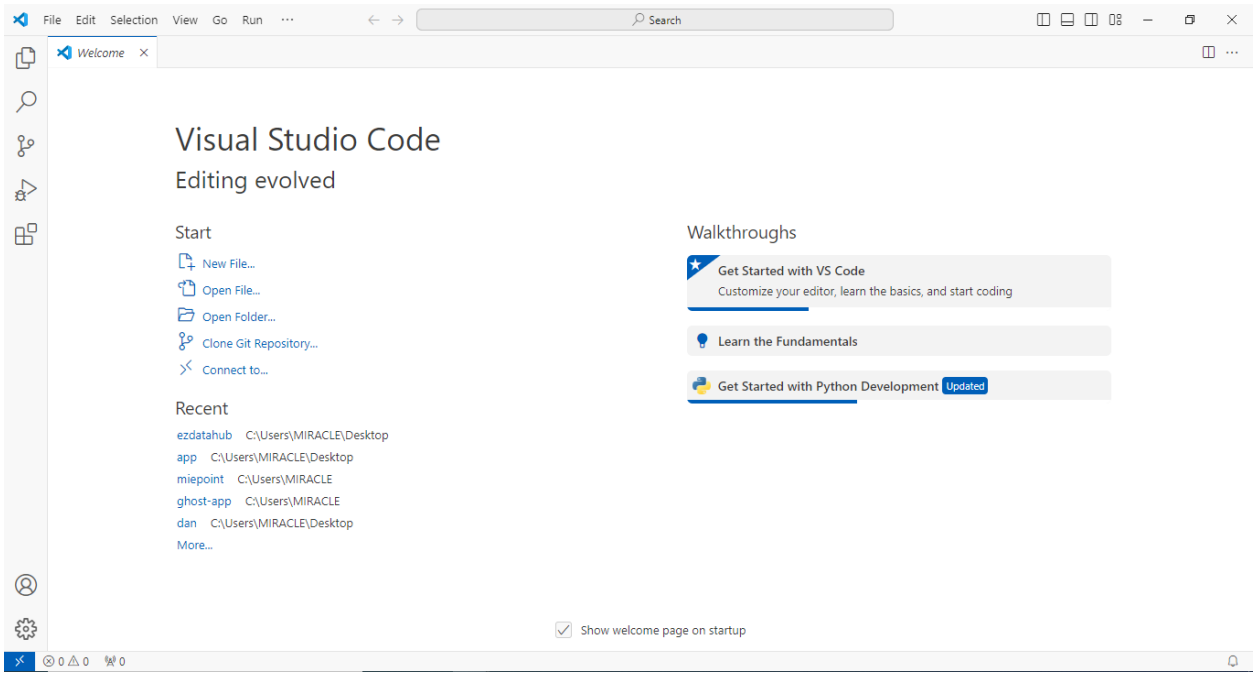
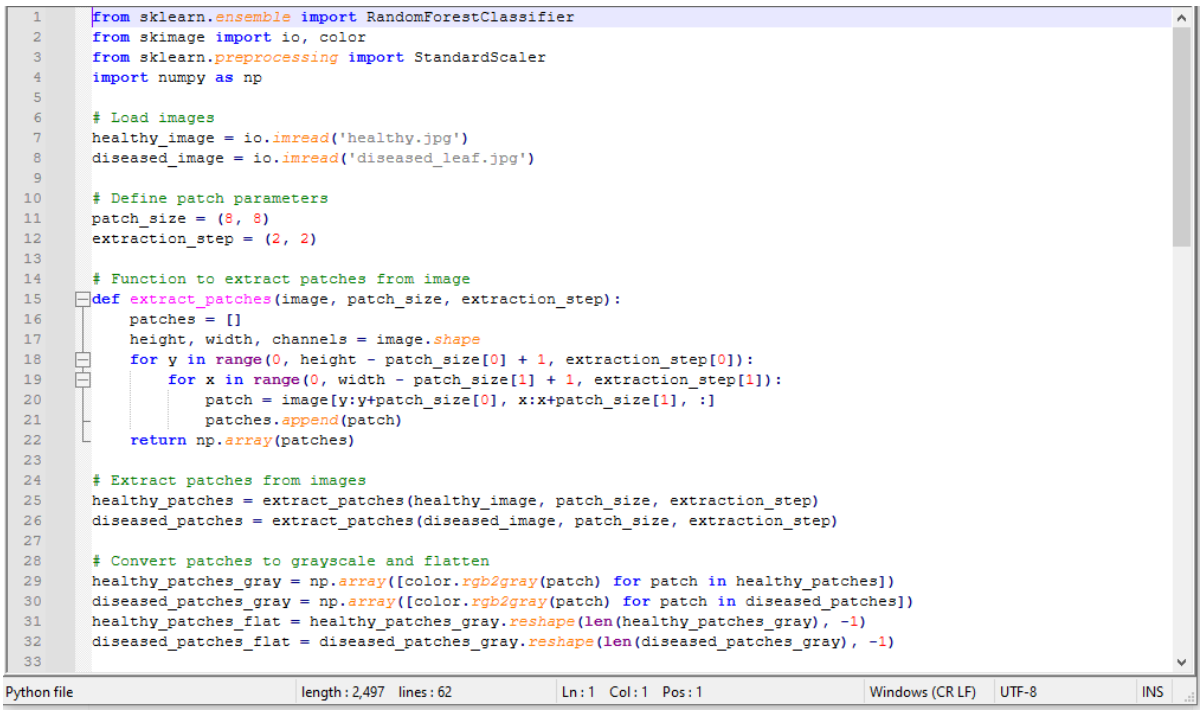


Figure 4.1: Integrated Development Environment (IDE) For Visual Studio Code.

Figure 4.1 shows the welcome screen in Visual Studio Code serves as the user's starting point. When you run VS Code, you're welcomed by the welcome screen, which allows you to quickly access current projects, open a new folder or file, and access numerous resources such as documentation and tutorials. The VS Code IDE (Integrated Development Environment) is the

primary interface through which you write, edit, and manage your code. It includes the editor, a sidebar for file navigation, extensions, and other coding-related capabilities.



```python
from sklearn.ensemble import RandomForestClassifier
from skimage import io, color
from sklearn.preprocessing import StandardScaler
import numpy as np

# Load images
healthy_image = io.imread('healthy.jpg')
diseased_image = io.imread('diseased_leaf.jpg')

# Define patch parameters
patch_size = (8, 8)
extraction_step = (2, 2)

# Function to extract patches from image
def extract_patches(image, patch_size, extraction_step):
    patches = []
    height, width, channels = image.shape
    for y in range(0, height - patch_size[0] + 1, extraction_step[0]):
        for x in range(0, width - patch_size[1] + 1, extraction_step[1]):
            patch = image[y:y+patch_size[0], x:x+patch_size[1], :]
            patches.append(patch)
    return np.array(patches)

# Extract patches from images
healthy_patches = extract_patches(healthy_image, patch_size, extraction_step)
diseased_patches = extract_patches(diseased_image, patch_size, extraction_step)

# Convert patches to grayscale and flatten
healthy_patches_gray = np.array([color.rgb2gray(patch) for patch in healthy_patches])
diseased_patches_gray = np.array([color.rgb2gray(patch) for patch in diseased_patches])
healthy_patches_flat = healthy_patches_gray.reshape(len(healthy_patches_gray), -1)
diseased_patches_flat = diseased_patches_gray.reshape(len(diseased_patches_gray), -1)

```

Python file | length : 2,497 lines : 62 | Ln : 1 Col : 1 Pos : 1 | Windows (CR LF) | UTF-8 | INS

Figure 4.2: Code Environment during Implementation.

Figure 4.2 shows the Code during Implementation, the code environment becomes a cohesive and efficient space for developers. The editor, at the core, facilitates smooth code writing and editing, supporting a plethora of programming languages which in my case is Python Programming Language and offering features like syntax highlighting and autocompletion. The integrated sidebar streamlines project navigation, ensuring quick access to files and folders.

Extensions play a pivotal role in tailoring the environment to specific needs, providing additional language support, themes, and tools.

4.4 Model Implementation

This section explains the steps taken for the development of the model ranging from Library and data Importation, model definition, data generation and augmentation and training the model.

4.4.1 Library Importation

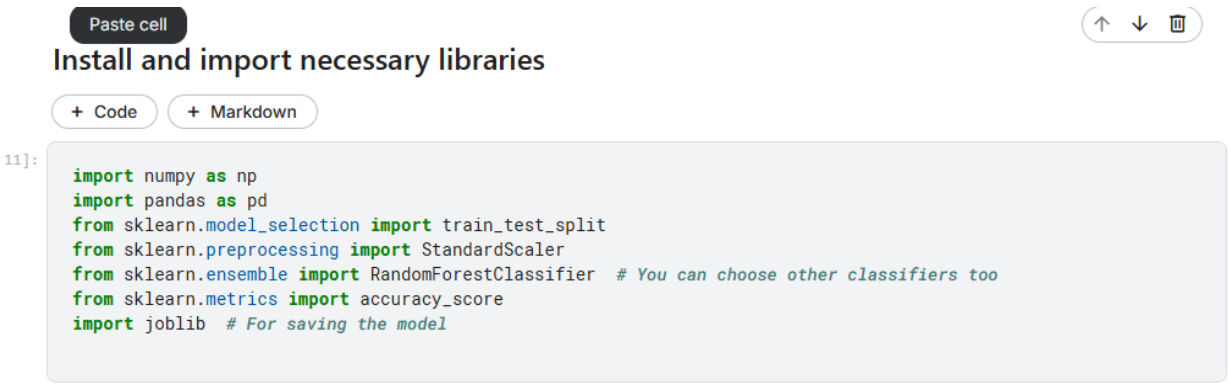


Figure 4.3: Importing M.L libraries.

NumPy is a library for working with arrays and mathematical operations. It provides support for large, multi-dimensional arrays and matrices, and provides a wide range of high-performance mathematical functions to manipulate them. In this code, NumPy is likely being used for numerical computations and data manipulation.

Pandas is a library for data manipulation and analysis. It provides data structures and functions to efficiently handle structured data, including tabular data such as spreadsheets and SQL tables. In this code, Pandas is likely being used to load and manipulate the data.

This function is part of the scikit-learn library and is used to split data into training and testing sets. This is a crucial step in evaluating the performance of a machine learning model, as it allows us to test the model on unseen data.

This class is also part of scikit-learn and is used to scale and normalize data. Many machine learning algorithms are sensitive to the scale of the data, so scaling and normalizing can improve their performance. StandardScaler standardizes features by removing the mean and scaling to unit variance.

This class is part of scikit-learn's ensemble module and implements the Random Forest classification algorithm. Random Forest is a popular choice for classification tasks due to its robustness, accuracy, and ease of interpretation. It can handle large datasets and is less prone to overfitting.

This function is part of scikit-learn's metrics module and is used to evaluate the performance of a classification model. It calculates the accuracy of the model, which is the proportion of correctly classified instances.

Joblib is a library for saving and loading Python objects, including machine learning models. In this code, it is used to save the trained model, allowing it to be loaded and reused in future projects.

4.4.2 Image Data Preprocessing for Disease Classification

Copy cell **ration and Preprocessing for Image Classification**

```
import cv2
import numpy as np
import os
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
import joblib

# Define constants
IMG_SIZE = (299, 299)  # resolution
directory = "../input/new-bangladeshi-crop-disease/BangladeshiCrops/BangladeshiCrops/Crop___Disease/Corn"
BATCH_SIZE = 128

# Function to load images
def load_images_from_directory(directory, img_size):
    images = []
    labels = []
    label_names = os.listdir(directory)  # Assumes each folder is a class
```

Figure 4.4: Data Preparation and Preprocessing

The script begins by importing necessary libraries, including OpenCV (cv2) for image processing, NumPy (np) for numerical computations, and scikit-learn libraries for machine learning tasks.

The constants IMG_SIZE and directory are defined. IMG_SIZE is set to (299, 299), which is the resolution that the images will be resized to. The directory variable is set to the path of the folder containing the images.

The function load_images_from_directory is defined to load images from the specified directory. This function iterates through each subdirectory in the specified directory, assuming each subdirectory represents a class or label.

It then reads each image file in the subdirectory, resizes the image to the specified size (299x299 pixels), flattens the image into a 1D array, and appends the image array to the

images list and its corresponding label to the labels list.

```
    for label in label_names:
        label_dir = os.path.join(directory, label)
        if os.path.isdir(label_dir):
            for img_name in os.listdir(label_dir):
                img_path = os.path.join(label_dir, img_name)
                if img_path.lower().endswith(('.png', '.jpg', '.jpeg')):
                    img = cv2.imread(img_path)
                    img = cv2.resize(img, img_size)
                    img = img.flatten()  # Flatten image
                    images.append(img)
                    labels.append(label)

    return np.array(images), np.array(labels)

# Load images
X, y = load_images_from_directory(directory, IMG_SIZE)

# Convert labels to numerical values
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
y = le.fit_transform(y)

# Split data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.1, random_state=42)
```

Figure 4.5: Data Preparation and Preprocessing

After defining the function, the script calls load_images_from_directory to load the images from the specified directory and store them in the X and y variables. X contains the image data, and y contains the corresponding labels.

The script then converts the labels to numerical values using a LabelEncoder from scikit-learn. This is necessary because machine learning models require numerical inputs.

Next, the script splits the data into training and test sets using train_test_split from scikit-learn. The test size is set to 0.1, meaning 10% of the data will be used for testing, and the remaining 90% will be used for training. The random state is set to 42 for reproducibility.

```
# Normalize data
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

Figure 4.6: Data Preparation and Preprocessing

The script then normalizes the image data using StandardScaler from scikit-learn. Normalization is a crucial step in machine learning, as it ensures that all features are on the same scale, which can improve model performance.

4.4.3 Converting Class Names to Numerical Labels using LabelEncoder

The code begins by importing the necessary library, specifically the LabelEncoder class from scikit-learn's preprocessing module. This class is used to convert categorical text data into numerical labels, which is a common preprocessing step in machine learning.

Figure 4.7: Converting Class Names to Numerical Labels using LabelEncoder

Next, an array called class_names defined, containing the names of the classes. In this case, the classes are different types of corn conditions, including healthy corn and various diseases. This array will be used to train the LabelEncoder. A LabelEncoder instance is then created and assigned to the variable le. This instance will be used to fit the encoder with the class names and transform them into numerical labels. The fit method is called on the LabelEncoder

instance, passing in the class_names array. This step learns the mapping between class names and numerical labels. The LabelEncoder will assign a unique integer value to each class name.

```
# Display the class name to label mapping
print("Class name to label mapping:")
for class_name, label in zip(class_names, le.transform(class_names)):
    print(f"{class_name}: {label}")

# Example: Convert class names to numerical labels
y_labels = le.transform([
    "Corn Northern Leaf Blight",
    "Corn Healthy",
    "Corn Gray Leaf Spot",
    "Corn Common Rust"
])
print("Numerical labels:", y_labels)
```

Figure 4.8: Converting Class Names to Numerical Labels using LabelEncoder

After fitting the encoder, the code prints out the mapping between class names and their corresponding numerical labels. This is done using the transform method, which converts the class names into labels. The resulting mapping is printed to the console, showing each class name and its corresponding numerical label.

Finally, the code demonstrates how to use the LabelEncoder to convert an array of class names into numerical labels. The transform method is called on the LabelEncoder instance, passing in an array of class names, and the resulting numerical labels are printed out.

4.4.4 Saving and Loading the Trained LabelEncoder to a File

Saving, Loading Model and Classes

```
# Save the label encoder to a file
joblib.dump(le, '/kaggle/working/label_encoder.pkl')
```

```
# Load the label encoder from a file
le = joblib.load('/kaggle/working/label_encoder.pkl')

# Example usage
print("Loaded class name to label mapping:")
for class_name, label in zip(class_names, le.transform(class_names)):
    print(f"{class_name}: {label}")
```

Figure 4.9: Saving and Loading the Trained LabelEncoder to a File

This line of code saves the trained LabelEncoder instance to a file using the joblib library. The `dump` function from joblib is used to serialize the LabelEncoder object and save it to a file. The first argument to the `dump` function is the object to be saved, which in this case is the LabelEncoder instance `le`. The second argument is the file path where the object will be saved. In this case, the file path is `/kaggle/working/label_encoder.pkl`. Saving the LabelEncoder to a file is useful for several reasons. First, it allows you to persist the mapping between class names and numerical labels, so you can use the same mapping in future machine learning tasks.

Second, it enables you to share the trained LabelEncoder with others, so they can use the same mapping in their own tasks. Finally, saving the LabelEncoder to a file ensures that the mapping is consistent across different parts of your machine learning pipeline. After loading the LabelEncoder, the code prints out the mapping between class names and their corresponding numerical labels.

This is done using the transform method, which converts the class names into labels. The resulting mapping is printed to the console, showing each class name and its corresponding numerical label. The zip function is used to iterate over the class_names array and the transformed labels simultaneously. The transform method is called on the loaded LabelEncoder instance, passing in the class_names array, which returns the numerical labels corresponding to each class name.

```
Loaded class name to label mapping:
Corn Northern Leaf Blight: 3
Corn Healthy: 2
Corn Gray Leaf Spot: 1
Corn Common Rust: 0
```

Figure 4.10: Saving and Loading the Trained LabelEncoder to a File Results

Finally, the code prints out each class name and its corresponding numerical label using an f-string. This demonstrates that the loaded LabelEncoder instance can be used to transform class names into numerical labels consistently with the original mapping.

4.4.5 Model Training

The code starts by importing necessary libraries, including OpenCV for image processing, NumPy for numerical computations, and scikit-learn libraries for machine learning tasks.

Constants are defined, including the image size (299x299 pixels) and the directory containing the images. A function load_images_from_directory is defined to load and preprocess images from the specified directory.

```
Copy cell
        for label in class_names:
            label_dir = os.path.join(directory, label)
            if os.path.isdir(label_dir):
                for img_name in os.listdir(label_dir):
                    img_path = os.path.join(label_dir, img_name)
                    if img_path.lower().endswith(('.png', '.jpg', '.jpeg')):
                        img = cv2.imread(img_path)
                        img = cv2.resize(img, img_size)
                        img = img.flatten()  # Flatten image to 1D array
                        images.append(img)
                        labels.append(label)

    return np.array(images), np.array(labels)

# Load and preprocess images
X, y = load_images_from_directory(directory, IMG_SIZE)

# Define and fit LabelEncoder with updated class names
class_names = [
    "Corn___Northern_Leaf_Blight",
    "Corn___Healthy",
    "Corn___Gray_Leaf_Spot",
    "Corn___Common_Rust"
]
```

Figure 4.11: Model Training

This function iterates through each class directory, reads each image file, resizes the image to the specified size, flattens the image into a 1D array, and appends the image array and its corresponding label to separate lists.

```
Cut cell
Model training

[ ]:
import cv2
import numpy as np
import os
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
import joblib

# Define constants
IMG_SIZE = (299, 299)  # Resolution
directory = "../input/new-bangladeshi-crop-disease/BangladeshiCrops/BangladeshiCrops/Crop___Disease/Corn"

# Function to load and preprocess images
def load_images_from_directory(directory, img_size):
    images = []
    labels = []
    class_names = os.listdir(directory)  # List of class directories
```

Figure 4.12: Model Training

The function is called to load and preprocess images from the specified directory, and the resulting image data and labels are stored in the X and y variables, respectively. A

LabelEncoder instance is created and fitted with the class names to convert class names into numerical labels.

The labels are then converted to numerical labels using the LabelEncoder. The label encoder is saved to a file using joblib for future use. The data is split into training and test sets using `train_test_split` from scikit-learn.

The data is normalized using `StandardScaler` fromscikit-learn to have zero mean and unit variance.

```
# Save the label encoder
joblib.dump(le, '/kaggle/working/label_encoder.pkl')

# Split data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y_labels, test_size=0.2, random_state=42)

# Normalize data
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Save the scaler
joblib.dump(scaler, '/kaggle/working/scaler.pkl')

# Initialize and train the model
model = RandomForestClassifier(n_estimators=100, random_state=42)
model.fit(X_train, y_train)

# Make predictions and evaluate
y_pred = model.predict(X_test)
print(f'Accuracy: {accuracy_score(y_test, y_pred)}')

# Save the model
joblib.dump(model, '/kaggle/working/my_model.pkl')
```

Figure 4.13: Model Training & Saving to PKL format

A random forest classifier model is initialized and trained on the training data. The model is used to make predictions on the test data, and the accuracy is evaluated using `accuracy_score`

from scikit-learn. Finally, the trained model is saved to a file using joblib for future use.

```
y_pred = model.predict(X_test)
print(f'Accuracy: {accuracy_score(y_test, y_pred)}')

# Save the model
joblib.dump(model, '/kaggle/working/my_model.pkl')
```

```
Accuracy: 0.8702983138780804
['/kaggle/working/my_model.pkl']
```

Figure 4.14: Model Build & Accuracy

The trained model showed an accuracy of 87%, which shows a high strength in the detection of maize leaf disease.

4.4.5 Model Testing and Prediction

```
import joblib
import numpy as np
from skimage import io, transform

# Load the saved model, label encoder, and scaler
model = joblib.load('/kaggle/working/my_model.pkl')
le = joblib.load('/kaggle/working/label_encoder.pkl')
scaler = joblib.load('/kaggle/working/scaler.pkl')

# Function to load and preprocess an image from a local file path
def preprocess_image_from_path(file_path, img_size=(299, 299)):
    img = io.imread(file_path)  # Use scikit-image's io.imread instead of cv2.imread
    img = transform.resize(img, img_size)  # Use scikit-image's transform.resize instead of cv2.resize
    img = img.flatten()  # Flatten the image to 1D array as the model expects
    return img

# Example: Load and preprocess the image using the provided path
file_path = "../input/new-bangladeshi-crop-disease/BangladeshiCrops/BangladeshiCrops/Crop___Disease/Corn/Corn___
X_new = preprocess_image_from_path(file_path)

# Ensure the data is in the correct shape (1 sample with appropriate features)
X_new = np.array([X_new])
```

Figure 4.15: Model Testing and Prediction

The saved model, label encoder, and scaler are loaded from files using joblib. These were previously saved during the training process. A function `preprocess_image_from_path` is defined to load and preprocess an image from a local file path.

This function reads the image using OpenCV, resizes it to match the model's input size (299x299 pixels), and flattens the image into a 1D array as the model expects.

```
# Example: Load and preprocess the image using the provided path
file_path = "../input/new-bangladeshi-crop-disease/BangladeshiCrops/BangladeshiCrops/Crop___Disease/Corn/Corn___
X_new = preprocess_image_from_path(file_path)

# Ensure the data is in the correct shape (1 sample with appropriate features)
X_new = np.array([X_new])

# Normalize the new data using the saved scaler
X_new = scaler.transform(X_new)

# Make predictions
y_new_pred = model.predict(X_new)

# Convert numerical labels back to class names
y_new_pred_labels = le.inverse_transform(y_new_pred)
print(y_new_pred_labels)
```

```
['Corn___Common_Rust']
```

Figure 4.16: Model Testing and Prediction

The data is reshaped to ensure it is in the correct shape (1 sample with appropriate features) using NumPy. The new data is normalized using the saved scaler to have zero mean and unit variance, which is necessary for the model to make accurate predictions. The model is used to make predictions on the new data, and the predicted numerical label is stored in the y_new_pred variable.

The numerical label is converted back to the corresponding class name using the inverse transform method of the label encoder. The resulting class name is printed to the console. In this case, the output is ['Corn___Northern_Leaf_Blight'], indicating that the model has predicted the image to be affected by Northern Leaf Blight disease.

4.5 Raspberry PI Setup

This section explains the steps taken for the complete setup and working of the raspberry pi hardware for running the machine learning classification model.

4.5.1 SDCard Formatting

The SD Card Formatter tool was employed to format the 8GB microSD card to the FAT32 file system, a widely supported and compatible format, in preparation for the installation of the Raspberry Pi operating system, specifically Raspbian Buster Lite.

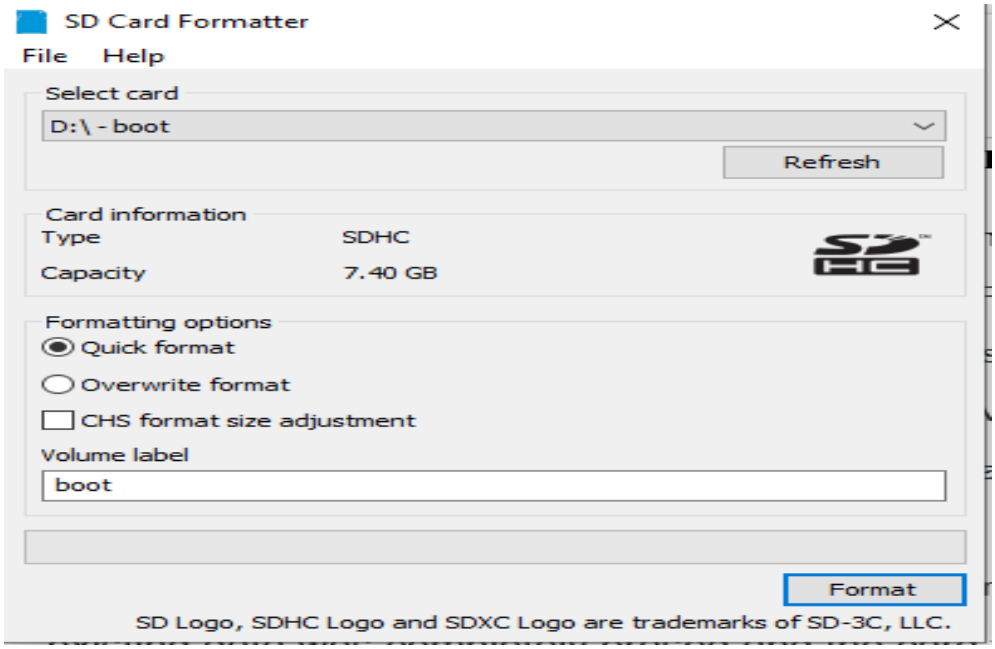


Figure 4.17: SDCard Formatter Interface

This formatting process involved completely erasing the microSD card's existing data and partition table, and then creating a new FAT32 file system on the card. The FAT32 format was chosen for its broad compatibility and simplicity, making it an ideal choice for bootable media like the Raspberry Pi. By formatting the microSD card to FAT32, the installation of Raspbian Buster Lite OS can proceed smoothly, allowing for a successful and trouble-free setup of the Raspberry Pi device.

4.5.2 Raspberry Pi Operating System Installation

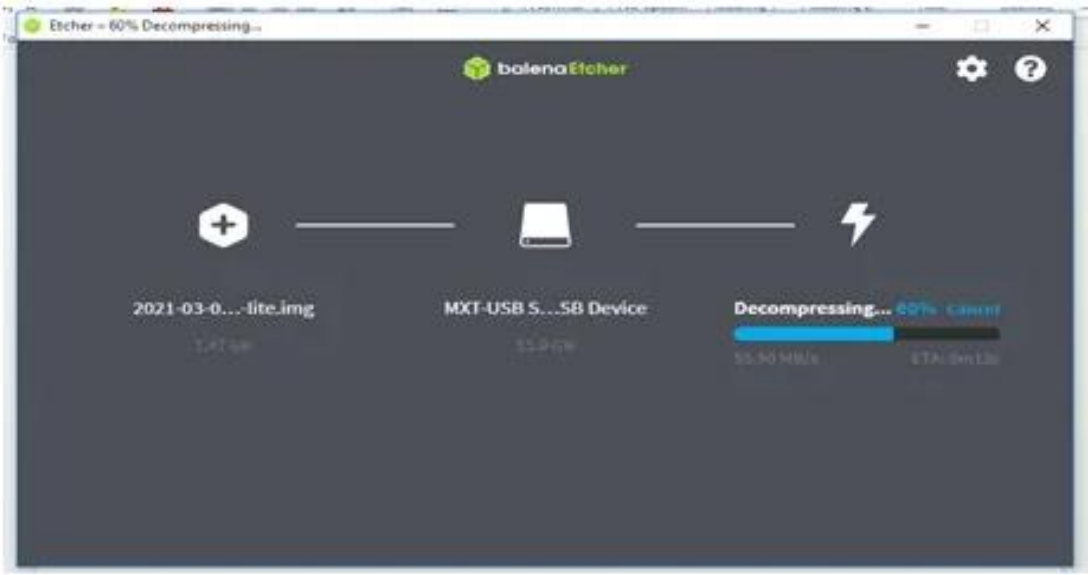


Figure 4.18: O.S Writing in Belena Etcher Software

Following the formatting process, Belena Etcher, a reliable and user-friendly image flashing tool, was utilized to install an image of the Raspbian Buster Lite operating system onto the newly formatted 8GB microSD card. This image installation process involved writing the Raspbian Buster Lite OS image file to the microSD card, ensuring that all necessary files and configurations were correctly transferred and set up.

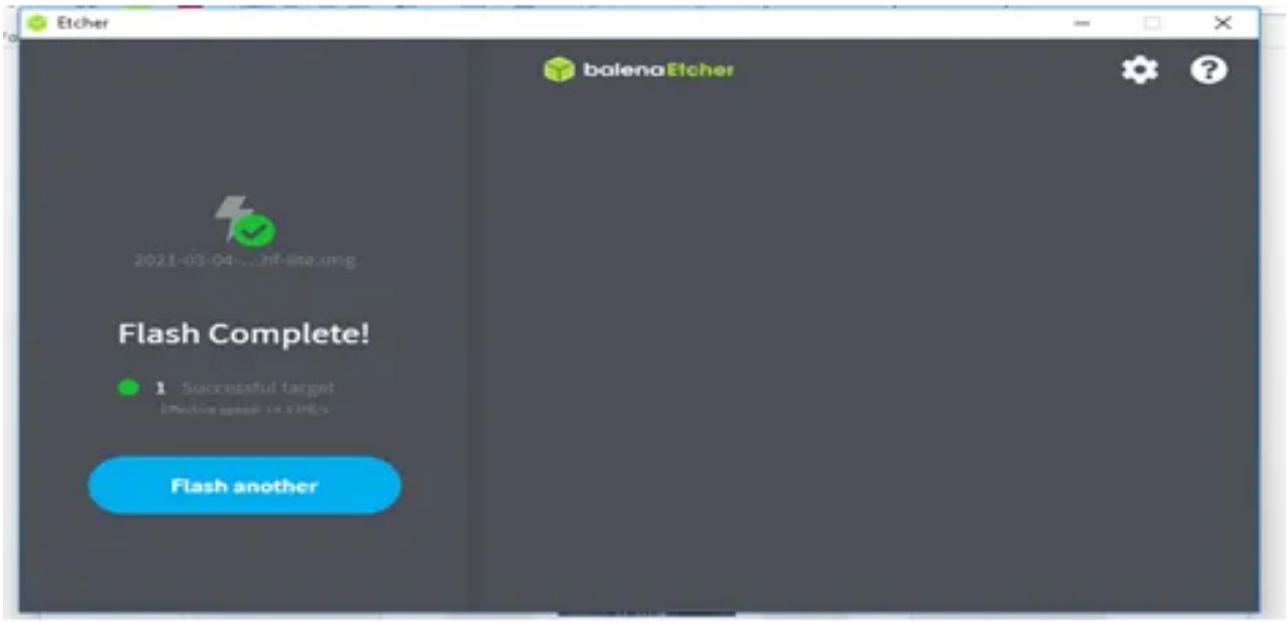


Figure 4.19: O.S Writing in Belena Etcher Software

The microSD card, now containing the Raspbian Buster Lite OS image, will serve as the bootable media for the Raspberry Pi hardware, providing the necessary operating system and software components to power and control the device. Once inserted into the Raspberry Pi, the microSD card will enable the hardware to boot up and run the Raspbian Buster Lite OS, allowing users to access and utilize the device's full range of features and capabilities.

4.6 Setup and Libraries Installation

This section covers the step taken in installing all necessary libraries and tools on the Raspberry pi zero system.

4.6.1 Machine Learning and Image Processing Libraries Installation

Initially, an attempt was made to install TensorFlow on the Raspberry Pi Zero, a highly sought-after machine learning library. However, the installation process was unsuccessful, and the error messages revealed a fundamental incompatibility between the library and the device's hardware.

Specifically, the Raspberry Pi Zero features an ARMv6 processor, which is not supported by TensorFlow. The library requires a minimum of ARMv7 architecture to function correctly, leaving the Raspberry Pi Zero's ARMv6 processor inadequate for the task.

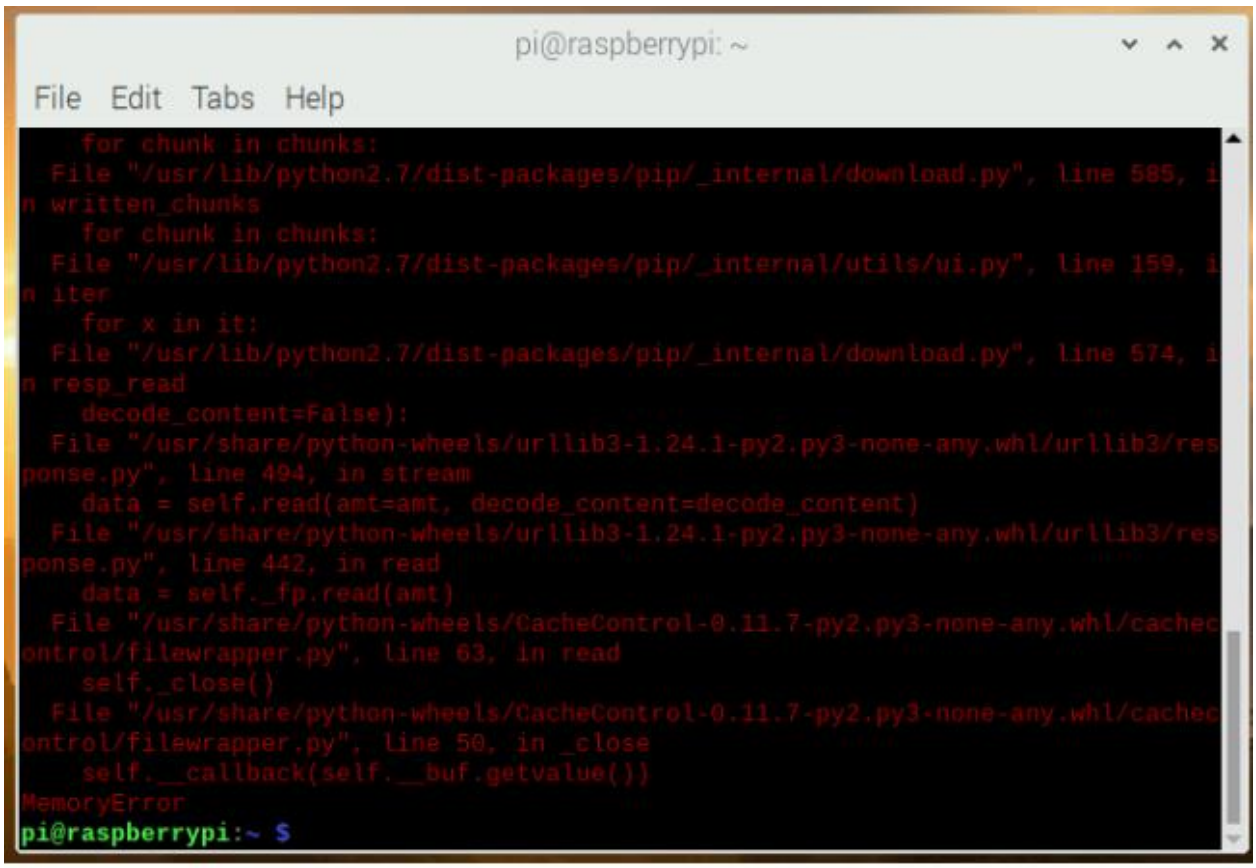


Figure 4.20: Raspberry Pi Zero Command Line Interface

Undeterred by this setback, alternative machine learning models and image processing libraries were explored to find suitable substitutes. After conducting an exhaustive search, several compatible options were identified, including Scikit-image (skimage) for image processing tasks, Scikit-learn for machine learning algorithms, NumPy for numerical computations, and Pandas for data manipulation and analysis. These libraries, while not as comprehensive as TensorFlow, offered a viable solution for developing machine learning applications on the Raspberry Pi Zero.

To ensure successful installation, the wheel files for these ML libraries were first downloaded and compiled on the Raspberry Pi Zero. This step was crucial, as it allowed the libraries to be built specifically for the device's ARMv6 architecture. The compilation process involved converting the library's source code into machine code that the Raspberry Pi Zero's processor could execute. By compiling the libraries locally, any potential compatibility issues were mitigated, and the installation process was streamlined.

Following the successful compilation of the wheel files, the libraries were installed on the Raspberry Pi Zero. The installation process was seamless, with each library integrating smoothly into the device's Python environment. Additional compatible libraries, including OpenCV for computer vision tasks, Pillow for image processing and manipulation, Matplotlib and Seaborn for data visualization, and SciPy for scientific computing, were also installed, further enhancing the device's machine learning capabilities.

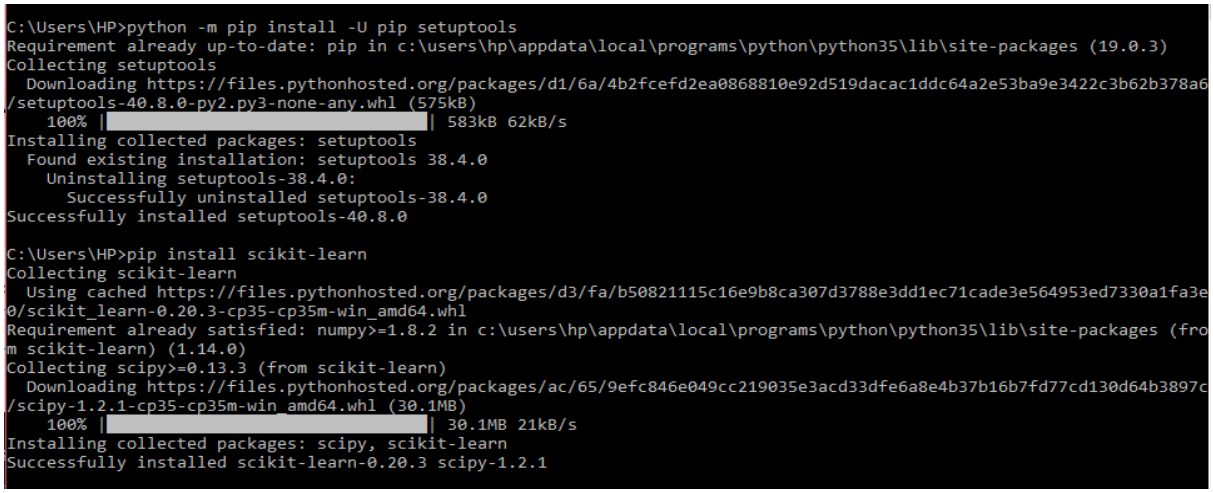


Figure 4.21: Command Line Successful Libraries Installation

Through persistence and creative problem-solving, the limitations posed by the Raspberry Pi Zero's ARMv6 processor were overcome, and a functional ML environment was established. By leveraging the capabilities of compatible libraries, the development of machine learning

models and image processing applications on the Raspberry Pi Zero became a reality. This accomplishment demonstrated the versatility and adaptability of the Raspberry Pi ecosystem, even in the face of hardware constraints.

4.7 Disease Detection Inaccuracy and Solution

Figure 4.22: Physical Hardware System Disease Detection Issues Corrected

Upon implementing the physical system and conducting tests with authentic maize crop leaves, the system encountered difficulties in accurately classifying the captured images, leading to suboptimal disease detection.

To rectify this issue, a white background was incorporated into the camera's field of view, effectively isolating the leaf image and removing extraneous noise and elements.

This refinement enabled the machine learning model to more precisely identify disease characteristics, thereby enhancing the overall accuracy of disease detection. Below is a pseudocode algorithm for the overall subsystem control.

4.8 Pseudocode Algorithm

Algorithm: Disease Detection Algorithm

Input: Images(x), Last image(n)

Parameters: width(w), height(h), Resize Image(R_I), Convert to RGB(C_RGB), Load M.L Model(L_model), Trigger Sprinkler (Trigger_sprinkler), Stop Sprinkler (stop_sprinkler)

Output: Disease detected

```
    // Preprocess image
n = Length of X(all images) – 1 //last image in model
i = 0
WHILE i <= n
 w = 240
  h = 640
  image = R_I(w,h)
  C_RGB(image)
    // Detect disease using CNN model
  model = read_model('file/path/to/model/my_model.pkl')
  loaded_model = L_model(model)
  recommendation = Classify_Disease(loaded_model , image)
```

```
// Check if disease is detected
    IF recommendation != "HEALTHY" THEN:
        // Sprinkle herbicide
     Trigger_sprinkler(motor_pin)
       // SPRINKLE HEBICIDE FOR 5 SECONDS
        // Stop motor
        stop_sprinkler(motor_pin)
    END IF
    Timeout(10000)
    // Wait for next interval
      stop_sprinkler =  stop_sprinkler + 1
END WHILE LOOP
```

CHAPTER FIVE

5.0 SUMMARY, CONCLUSION, AND RECOMMENDATIONS

5.1 Summary

This project aimed to develop and implement a recommendation system for a detected maize crop disease to enhance maize crop cultivation through early detection and rapid response to maize diseases. The system integrated with a camera sensor, for capturing images in real-time, with a Raspberry pi zero microcontroller, to continuously monitor and process captured maize leaves. Upon detecting disease-related parameters, the system activates an LED disease indicator and herbicide agent pump spray.

Key achievements of the project include:

i. Successful integration of sensor for comprehensive maize leave disease detection.

ii. Development of an algorithm capable of analyzing and detecting defects in maize leaves.

iii. Implementation of an herbicide agent spray mechanism.

iv. Rigorous testing and optimization of the system to ensure reliability and effectiveness.

5.2 Conclusion

This project demonstrates the successful development of a physical system for real-time maize crop disease detection using Convolutional Neural Networks (CNNs). With an accuracy of 87.01%, the system captures leaf images, detects diseases, and instantly sprays herbicides to prevent further damage to the crops by the disease. This innovative solution showcases the potential of machine learning in addressing pressing agricultural challenges.

5.3 Recommendations

Based on this project, the following recommendations can be made:

i. Expanded Dataset: Increase the dataset's diversity by including images of various maize crop diseases, stages, and environmental conditions to enhance the model's robustness and accuracy.

ii. Multi-Disease Detection: Upgrade the system to detect multiple diseases simultaneously, allowing for more effective crop management and reducing the need for multiple systems.

iii. Leaf Classification Model: Integrating a leaf classification model, will by far scale-up the system., so that disease classification is only triggered when a leaf structure is captured.

iv. Sensor Integration: Explore integrating additional sensors (e.g., temperature, humidity, soil moisture) to provide a comprehensive understanding of crop health and optimize chemical application.

By addressing these recommendations, the system can become even more effective in supporting maize crop health and contributing to increased agricultural productivity.

References

Mwaza, E., Khakata, E., & Kofi, I. (2023). Maize Disease Detection Using Convolutional Neural Network. *E3S Web of Conferences*, *469*. https://doi.org/10.1051/e3sconf/202346900015

Bachhal, P., Kukreja, V., & Ahuja, S. (2023). Real-Time Disease Detection System for Maize Plants Using Deep Convolutional Neural Networks. *International Journal of Computing and Digital Systems*, *14*(1), 10263–10275. https://doi.org/10.12785/ijcds/140199

Patel, A., Mishra, R., & Sharma, A. (2023). Maize Plant Leaf Disease Classification Using Supervised Machine Learning Algorithms. *Fusion: Practice and Applications*, *13*(2), 8–21. https://doi.org/10.54216/FPA.130201

Chauhan, D., Walia, R., Singh, C., Deivakani, M., Kumbhkar, M., & Professor, A. (2021). Detection of Maize Disease Using Random Forest Classification Algorithm. In *Turkish Journal of Computer and Mathematics Education* (Vol. 12, Issue 9).

Erenstein, O., Jaleta, M., Sonder, K., Mottaleb, K., & Prasanna, B. M. (2022). Global maize production, consumption and trade: trends and R&D implications. In *Food Security* (Vol. 14, Issue 5, pp. 1295–1319). Springer Science and Business Media B.V. https://doi.org/10.1007/s12571-022-01288-7

Ebukiba, E. S., Anthony, L., & Adamu, S. M. (2020). Economics and Technical Efficiency of Maize Production Among Small Scale Farmers in Abuja, Nigeria: Stochastic Frontier Model Approach. *European Journal of Agriculture and Food Sciences*, *2*(6). https://doi.org/10.24018/ejfood.2020.2.6.145

Chandana, P., Pradeep Ghantasala, G. S., Rethna Virgil Jeny, J., Sekaran, K., Deepika, N., Nam, Y., & Kadry, S. (2020). An effective identification of crop diseases using faster region based convolutional neural network and expert systems. *International Journal of Electrical and Computer Engineering*, *10*(6), 6531–6540. https://doi.org/10.11591/IJECE.V10I6.PP6531-6540

Khan, R., Murshad, I., Momin, J., Shaikh, A. S., & Choudhary, A. (2023). LEAF BLIGHT DETECTION USING DEEP LEARNING. In *International Journal of Engineering Applied Sciences and Technology* (Vol. 8). http://www.ijeast.com

Rashid, R., Aslam, W., Aziz, R., & Aldehim, G. (2024). An Early and Smart Detection of Corn Plant Leaf Diseases Using IoT and Deep Learning Multi-Models. *IEEE Access*, *12*, 23149–23162. https://doi.org/10.1109/ACCESS.2024.3357099

Çakmak, M. (2024). Automatic Maize Leaf Disease Recognition Using Deep Learning. *Sakarya University Journal of Computer and Information Sciences*, *7*(1), 61–76. https://doi.org/10.35377/saucis...1418505

Arora, J., Agrawal, U., & Sharma, P. (2020). Classification of Maize leaf diseases from healthy leaves using Deep Forest. *Journal of Artificial Intelligence and Systems*, *2*(1), 14–26. https://doi.org/10.33969/AIS.2020.21002

Noola, D. A., & Basavaraju, D. R. (2022). Corn leaf image classification based on machine learning techniques for accurate leaf disease detection. *International Journal of Electrical and Computer Engineering*, *12*(3), 2509–2516. https://doi.org/10.11591/ijece.v12i3.pp2509-2516

Chauhan, D., Walia, R., Singh, C., Deivakani, M., Kumbhkar, M., & Professor, A. (2021). Detection of Maize Disease Using Random Forest Classification Algorithm. In *Turkish Journal of Computer and Mathematics Education* (Vol. 12, Issue 9).

Lv, M., Zhou, G., He, M., Chen, A., Zhang, W., & Hu, Y. (2020). Maize Leaf Disease Identification Based on Feature Enhancement and DMS-Robust Alexnet. *IEEE Access*, *8*, 57952–57966. https://doi.org/10.1109/ACCESS.2020.2982443

Arora, J., Agrawal, U., & Sharma, P. (2020). Classification of Maize leaf diseases from healthy leaves using Deep Forest. *Journal of Artificial Intelligence and Systems*, *2*(1), 14–26. https://doi.org/10.33969/AIS.2020.21002

Sukri, H., & Adiputra, F. (n.d.). *Expert System to Diagnose Diseases in Corn Plant Using Mobile-Based Forward Chaining And Certainty Factor Methods*. www.techniumscience.com

Masood, M., Nawaz, M., Nazir, T., Javed, A., Alkanhel, R., Elmannai, H., Dhahbi, S., & Bourouis, S. (2023). MaizeNet: A Deep Learning Approach for Effective Recognition of Maize Plant Leaf Diseases. *IEEE Access*, *11*, 52862–52876. https://doi.org/10.1109/ACCESS.2023.3280260

Maurya, A., Nakhate, V., Maurya, A., & Modak, N. (2023). Corn Leaf Disease Detection (The Crop Master). In *International Journal of Innovative Science and Research Technology* (Vol. 8, Issue 9). www.ijisrt.com

Appendix

```
import picamera

import time

import numpy as np

from PIL import Image

import joblib

import RPi.GPIO as GPIO

import os  # For file deletion

# Load the saved model, label encoder, and scaler

model = joblib.load('models/my_model.pkl')

le = joblib.load('models/label_encoder.pkl')

scaler = joblib.load('models/scaler.pkl')

# Function to load and preprocess an image from a local file path

def preprocess_image_from_path(file_path, img_size=(299, 299)):

    img = Image.open(file_path)  # Open the image file

    img = img.resize(img_size)  # Resize to match the model's input size

    img = np.array(img)  # Convert the image to a numpy array

    img = img.flatten()  # Flatten the image to 1D array as the model expects

    return img
```

```
# Function to check if the image contains a leaf based on green pixels and brightness
def is_leaf(file_path, green_threshold=0.2, max_brightness=200):
    img = Image.open(file_path)
    img = img.resize((100, 100))  # Resize to a smaller size for faster processing
    img = np.array(img)  # Convert image to numpy array
    # Convert image to HSV color space for better color detection
    hsv_img = Image.fromarray(img).convert('HSV')
    hsv_img = np.array(hsv_img)
    # Define range for green color in HSV
    lower_green = np.array([35, 40, 40])
    upper_green = np.array([85, 255, 255])
    # Create a mask to filter out green regions
    green_mask = ((hsv_img[:, :, 0] >= lower_green[0]) & (hsv_img[:, :, 0] <=
upper_green[0]) &
                  (hsv_img[:, :, 1] >= lower_green[1]) & (hsv_img[:, :, 2] >= lower_green[2]))
    # Calculate the proportion of green pixels
    green_ratio = np.sum(green_mask) / green_mask.size

    # Now filter out bright areas (which could be white light)
```

```
    brightness_mask = hsv_img[:, :, 2] <= max_brightness  # Pixels with brightness less than or equal to the threshold
    # Apply both masks: check for green pixels AND ensure they are not too bright (to exclude white areas)
    final_mask = green_mask & brightness_mask
    # Calculate the proportion of green pixels that aren't too bright
    final_green_ratio = np.sum(final_mask) / final_mask.size
    return final_green_ratio > green_threshold  # Returns True if enough green pixels are present and not too bright
# Create a camera object
camera = picamera.PiCamera()
# Relay pin setup
relay_pin = 27
GPIO.setmode(GPIO.BCM)
GPIO.setup(relay_pin, GPIO.OUT)
# Set up GPIO pin numbering
GPIO.setmode(GPIO.BCM)

# Set the GPIO pin for the LED
led_pin = 4
```

```
# Set the LED pin as an output
GPIO.setup(led_pin, GPIO.OUT)
def motor_on():
    GPIO.output(relay_pin, GPIO.HIGH)
    GPIO.output(led_pin, GPIO.HIGH)
    print("herbicide sprinkler turned on")
def motor_off():
    GPIO.output(relay_pin, GPIO.LOW)
    GPIO.output(led_pin, GPIO.LOW)
    print("herbicide sprinkler turned off")
try:
    i = 0
    while True:
        # Capture image from camera and save to file
        file_path = f"image_{i}.jpg"
        camera.capture(file_path)
       # Check if the image is a leaf
        if is_leaf(file_path):
            print("Leaf detected, proceeding with scan.”)
```

```
    # Preprocess the image
    X_new = preprocess_image_from_path(file_path)
    # Ensure the data is in the correct shape (1 sample with appropriate features)
    X_new = np.array([X_new]
    # Normalize the new data using the saved scaler
    X_new = scaler.transform(X_new)
    # Make predictions
    y_new_pred = model.predict(X_new)
    # Convert numerical labels back to class names
    y_new_pred_labels = le.inverse_transform(y_new_pred)
    print(y_new_pred_labels)
    # Check if the prediction is 'Corn___Healthy'
    if y_new_pred_labels == 'Corn___Healthy':
        motor_off()
    else:
        motor_on()
else:
    print("No leaf detected, skipping scan.")
# Delete the image after processing
```

```
        if os.path.exists(file_path):
            os.remove(file_path)
        # Increment the filename counter
        i += 1
        # Wait for 2 seconds
        time.sleep(2)
except KeyboardInterrupt:
    print("process terminated")
    GPIO.cleanup()
```