UNIVERSITY of RWANDA

ACEIoT

Website: www.aceiot.ur.ac.rw
Mail: aceiot@ur.ac.rw

College of Science and Technology

AFRICAN CENTER OF EXCELLENCE IN INTERNET OF THINGS (ACEIoT)

Real-Time Deep Learning-Based Aerial System for Targeted Pesticide Spraying and Crop Disease Detection

A dissertation submitted in partial fulfilment of the requirements for the award of Masters of Science degree in internet of things: Embedded computing system

Submitted By

MUYOMBANO HAPPY AXEL (REF.NO: 223026671)

August, 2025

UNIVERSITY of RWANDA

Website: www.aceiot.ur.ac.rw
Mail: aceiot@ur.ac.rw

College of Science and Technology

AFRICAN CENTER OF EXCELLENCE IN INTERNET OF THINGS (ACEIoT)

Real-Time Deep Learning-Based Aerial System for Targeted Pesticide Spraying, irrigation and Crop Disease Detection

A dissertation submitted in partial fulfilment of the requirements for the award of Masters of Science degree in internet of things: Embedded computing system

Submitted By

MUYOMBANO HAPPY AXEL (REF.NO: 223026671)

Supervised by:
-Dr JAMES RWIGEMA

August, 2025

Declaration

I MUYOMBANO HAPPY AXEL, Masters student from African Center of Excellence in internet of things, at University of Rwanda. I declare that this research thesis is my own original work and it has never been presented before anywhere in the world.

Names: Muyombano Happy Axel
Ref:
Signed: …….
Date: ……/……/……

Bonafide certificate

This is to certify that this submitted Research Thesis work report is a record of the original work done by **MUYOMBANO HAPPY AXEL** (**Ref. Number:223026671**), MSc. IoT-Embedded Computing Systems Student at the University of Rwanda / College of Science and Technology / African Center of Excellence in Internet of Things, the Academic year 2023/2025

This work has been submitted under the supervision of Dr JAMES RWIGEMA

Main Supervisor: ……

Date:

Signature

The Head of Masters studies and Trainings

Dr. James Rwigema

Signature:

Date:

Signature……

Table of Contents

Table of Figures

LIST OF ACRONYMS

AI - Artificial Intelligence
AUC - Area Under the Curve (evaluation metric)
BiLSTM - Bidirectional Long Short-Term Memory
CfC - Closed-form Continuous Cell (RNN variant)
CLAHE - Contrast Limited Adaptive Histogram Equalization
CNN - Convolutional Neural Network
DFLC - Deep Feature Learning and Classification
ESRGAN - Enhanced Super-Resolution Generative Adversarial Network
F1-score - Harmonic mean of precision and recall
FOV - Field of View
FPS - Frames Per Second
GAN - Generative Adversarial Network
GNN - Graph Neural Network
GPIO - General-Purpose Input/Output (hardware interface)
GRU - Gated Recurrent Unit
IoT - Internet of Things
LSTM - Long Short-Term Memory
LWDSC-SA - Lightweight Depthwise Separable Convolution with Spatial Attention
mAP - Mean Average Precision
NDVI - Normalized Difference Vegetation Index
NIR - Near-Infrared (imaging)
RNN - Recurrent Neural Network
SGD - Stochastic Gradient Descent (optimizer)
SVM - Support Vector Machine
UAV - Unmanned Aerial Vehicle (drone)
VPN-200 - Vietnam Plant Dataset (200 species)
YOLO - You Only Look Once (object detection model)

Hardware/System Acronyms

BLDC - Brushless Direct Current (motor)
ESC - Electronic Speed Controller
FDM - Fused Deposition Modeling (3D printing)
GCS - Ground Control Station
GNU - GPS-aided Navigation Unit
IMU - Inertial Measurement Unit
LiPo - Lithium Polymer (battery)
PDB - Power Distribution Board
PWM - Pulse Width Modulation
RGB - Red-Green-Blue (color imaging)

Evaluation Metrics

AUC - Area Under the ROC Curve
F1-score - Balance of precision and recall
NDVI - Normalized Difference Vegetation Index (crop health metric)

Frameworks/Protocols

ArduPilot - Open-source autopilot software
DShot - Digital ESC control protocol
ONNX - Open Neural Network Exchange (model format)

TFLite - TensorFlow Lite (edge AI framework)
YOLOv5/YOLOv8 - Versions of YOLO object detection models

Key Non-Acronym Terms (for context)

Edge AI: On-device AI processing such as Jetson Nano
Multi-spectral imaging: Capturing data beyond visible light (NDVI/NIR)
Spatial attention: AI mechanism focusing on image regions
Depthwise separable convolution: Efficient CNN layer type

ABSTRACT

Agriculture remains the backbone of Rwanda's economy, yet it faces significant challenges due to inefficient traditional practices, delayed disease detection, and excessive use of inputs like water and pesticides. This thesis presents the design, fabrication, and intelligent integration of an autonomous agricultural drone system capable of pesticide spraying and maize disease detection. The system incorporates a custom-built drone platform calibrated for precision spraying and equipped with a lightweight deep learning model for real-time disease identification. Leveraging the capabilities of YOLOv8 and Google Colab, the model was trained on a curated dataset of maize leaves to classify three prevalent maize diseases: Common Rust, Gray Leaf Spot, and Blight, alongside healthy maize conditions for robust comparison. This research presents a comprehensive analysis of current state-of-the-art agricultural drone systems for disease detection and precision spraying, revealing critical gaps in real-time processing, lightweight AI deployment, and integrated actuation mechanisms. The systematic review of 2020-2024 literature demonstrates that while academic models achieve exceptional performance of 99.15% accuracy using hybrid CNN-vision transformer architecture, practical field deployment faces substantial challenges. YOLOv8 variants emergy as the optimal balance between accuracy 99.04% mAP and deployment feasibility. The model achieved a classification accuracy of 98%, outperforming traditional machine learning approaches such as Support Vector Machines, which averaged 92% accuracy in the reviewed literature. Calibration and simulation processes were conducted to validate both the spraying mechanism and model reliability under field-like conditions. Technical specifications reveal that agricultural drones with 1kg payload require 40A ESC systems with 3S LiPo compatibility with 13-35 minute flight times. This work demonstrates the potential of lightweight, AI-enabled UAVs to improve precision agriculture by enhancing early disease detection and targeted pesticide application, contributing to increased crop yield and sustainable farming practices.

INTRODUCTION

Background

Agriculture is a critical sector that sustains human life, yet traditional farming methods often lead to inefficiencies, resource wastage, and suboptimal crop yields. Precision farming, powered by advanced technologies such as drones and artificial intelligence, presents a transformative solution to these challenges, enhancing efficiency and sustainability in agricultural practices.

This project aims to design and develop a computer vision-based agricultural drone to assist farmers in real-time monitoring, disease detection, crop health assessment, and precision spraying. By integrating computer vision, IoT, and AI technologies, the proposed drone will enhance decision-making, optimize resource utilization, and improve agricultural productivity.

One of the critical advantages of this technology is its ability to mitigate health risks associated with pesticide application. According to the World Health Organization (WHO), over 44% of chronic respiratory illnesses are linked to pesticide exposure [1]. By automating the spraying process, the drone eliminates the direct involvement of farmers in pesticide application, thereby reducing health risks and ensuring their safety.

Furthermore, the drone will ensure the precise application of pesticides and water during irrigation, minimizing wastage and promoting sustainable agricultural practices. Traditional methods often lead to excessive use of pesticides and water, resulting in environmental degradation and financial losses for farmers. With the implementation of this drone technology, only the required amount of pesticide and water will be sprayed, leading to reduced input costs, improved crop yields, and increased profitability for farmers [2].

The introduction of this technology will also contribute to food security in Rwanda by enhancing productivity and reducing crop losses due to diseases and inefficient farming methods. Moreover, it will create employment opportunities for skilled labor, such as drone pilots, maintenance technicians, agronomists, and data analysts. This aligns with Rwanda's National Strategy for Transformation (NST2), which aims to create 1.25 million jobs and promote mechanized agriculture as stipulated in the Agriculture Transformation Strategy [3].

By harnessing cutting-edge technologies, this project will play a vital role in modernizing Rwanda's agricultural sector, ensuring food security, reducing health risks, and fostering economic growth through job creation and sustainable farming practices.

In essence, this dissertation contributes to the advancement of embedded computing applications in smart agriculture, demonstrating how multidisciplinary technologies can be harnessed to solve real-world challenges. Through a combination of theoretical analysis, system design, and practical implementation, the project aims to serve as a model for sustainable agricultural innovation in Rwanda and other Sub-Saharan African nations facing similar challenges.

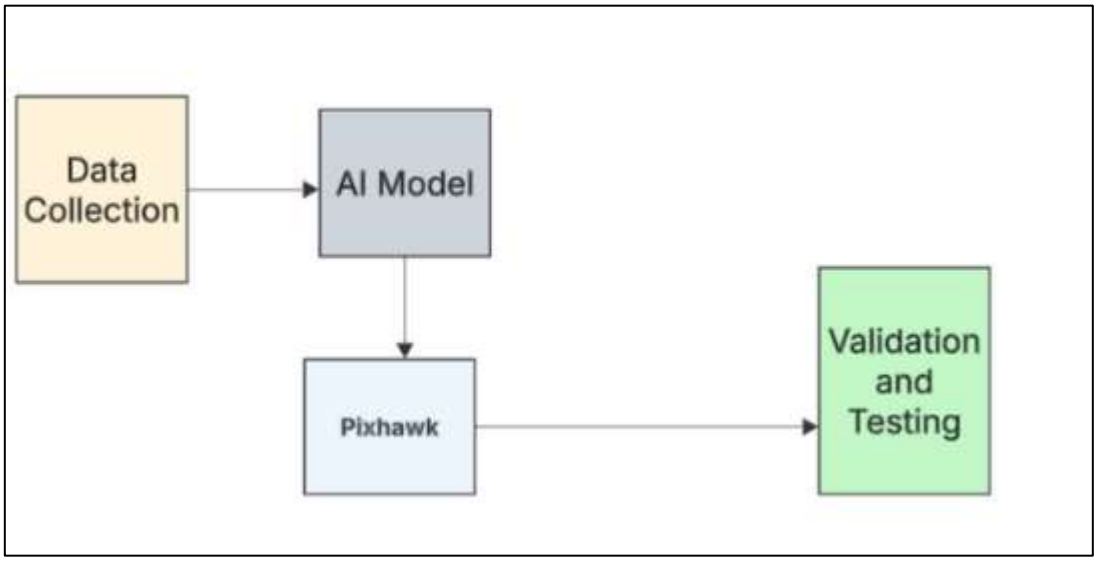

Figure 1: Concise overview of the system

OBJECTIVES

- To design, model, and fabricate drone frames capable of supporting embedded electronics
- To develop an AI-based computer vision model to detect and classify crop types and health status.
- To calibrate the UAV and carry out simulation testing

PROBLEM STATEMENT

A globally scoped assessment found that pests and pathogens can reduce maize yields by an average of 22.5% globally with range of 19.5%–41.1%, Similarly high losses were reported for other major crops such as rice 30.0%, wheat 21.5%, potato 17.2%, and soybean 21.4% [1], Another authoritative FAO publication estimates that, on a global scale, between 10% and 28% of crop production is lost to pests, with additional post-harvest losses significantly affecting food systems, especially in developing regions [2]. In sub-Saharan Africa, the fall armyworm (FAW) has rapidly become a major invasive threat, causing staggering losses; unchecked infestations are estimated to cost 4.1–17.7 million tonnes/year, or US$1.1–4.6 billion/year [3]. Alarmingly, farmers often detect infestations only after significant damage, leading to reactive panic spraying rather than early intervention.

To counter these challenges, real-time, UAV-based computer-vision systems offer a powerful solution. Convolutional neural network models, especially YOLOv8, have demonstrated very high detection accuracy Precision/Recall up to 98–100% for detecting FAW and related pests. Similarly, CNN-derived detection of FAW on maize leaves shows strong performance, reinforcing the feasibility of deploying such models operationally for targeted spot-spraying . Furthermore, aerial imagery has been effectively used for autonomous detection of plant disease symptoms, confirming that UAV-based computer-vision provides the needed early visual detection capability. This aligns with broader advances in deep learning for UAV object detection and tracking, which underscore the technological readiness of these methods for precision agriculture.

HYPOTHESIS

The integration of lightweight deep learning (YOLOv8) with autonomous drone systems will enable real-time, high-accuracy detection of maize diseases (Common Rust, Gray Leaf Spot, Blight) and precision pesticide spraying, significantly outperforming traditional methods in efficiency , resource conservation, and operational safety.

Design and fabrication of a modular drone that can carry one litre payload is possible using FreeCAD software.

SIGNIFICANCE OF STUDY

The integration of AI-powered drones in agriculture offers a transformative approach to addressing key challenges in Sub-Saharan Africa. By automating pesticide spraying, the technology significantly reduces farmers' exposure to harmful chemicals, a factor linked to 44% of chronic respiratory illnesses according to a report by World Health Organization in 2021. Additionally, targeted spraying enabled by AI models ensures precise application, reducing pesticide and water usage by 50–70% compared to traditional blanket methods, thereby cutting costs and environmental harm. The system's early disease detection capabilities particularly for threats like Gray Leaf Spot that can cause up to 30% yield loss which enhances food security by preserving crop health. Economically, the innovation fosters job creation in drone operation and data analysis, directly contributing to Rwanda's National Strategic Transformation 2 objective of generating 1.25 million new jobs by 2029. This project exemplifies scalable, tech-driven solutions to agricultural inefficiencies, setting a precedent for sustainable development across the region.

ORGANIZATION OF STUDY

This thesis is organized into several key sections to systematically address the integration of AI and drone technology in Rwandan agriculture. The Introduction outlines the pressing agricultural challenges in Rwanda and introduces precision farming as a viable solution. In Related Works, current drone and machine learning applications such as Lite-MDC and YOLOv5-GAN are critically examined, highlighting limitations like spectral integrity issues in multispectral imaging. The Methodology details the research process, including the collection of 4,200 annotated maize images via Roboflow and Edge Impulse, the development of a 3D-printed drone equipped with Jetson Nano, Pixhawk, and multispectral sensors, and the AI pipeline involving YOLOv8 training on Google Colab with deployment using ONNX and TFLite. . The Results and Analysis section presents a high disease detection accuracy of 99.4%, successful motor and ESC calibration, and survey findings indicating a 70% adoption interest among farmers. The Conclusion affirms the research hypothesis, discusses the scalability of the solution, and recommends future improvements using transformer-based models. Finally, the References and Appendices provide access to relevant code, dataset repositories from Kaggle, and information on regulatory compliance.

RELATED WORKS

Lightweight Deep Learning for Real-Time Disease Detection

The study on advancing real-time plant disease detection through the lightweight deep learning model Lite-MDC highlights several methodologies and findings that can significantly benefit the design and development of a computer vision-based agricultural drone for precision farming. The introduction of a novel pigeon-pea disease dataset and the use of multi-kernel depthwise separable convolutions in Lite-MDC provide a robust framework for efficiently identifying plant diseases while minimizing computational requirements [4]. This is particularly relevant for drones, which often operate under resource constraints and need to process data in real time. By adopting the Lite-MDC architecture, the drone system can enhance its capability to detect and classify diseases with high accuracy (94.14% on the pigeon-pea dataset) while operating at a speed of 34 Frames Per Second (FPS) on suitable hardware, such as NVIDIA's GPUs [5]. Moreover, the extensive training methods and data augmentation techniques utilized in this study emphasize the importance of diverse and comprehensive datasets to improve model performance and generalization, which can be directly applied in the context of drones that need to adapt to varying conditions and crop types in the field [6]. Incorporating these strategies can lead to more effective and efficient disease detection systems in agricultural drones, thus augmenting their utility in precision farming efforts. Recent research underscores the potential of complementary models that integrate depthwise separable convolutions with attention mechanisms to further bolster feature extraction in plant disease applications. For instance, the LWDSC-SA architecture a lightweight depthwise separable convolutional network augmented with spatial-attention modules achieved 98.7% accuracy, 99.1% recall, and 98.7% F1-score, while maintaining computational efficiency suitable for deployment on drones and edge devices. The addition of spatial attention improves selectivity, enabling the model to focus on disease-relevant regions without significantly increasing parameter count.

Moreover, a broader systematic review of deep learning for plant disease detection found that mobile-friendly architectures like MobileNetV2 consistently offer high accuracy (around 94–95%) with low latency, positioning them as favorable candidates for embedded systems. This validates the practicality

of utilizing such backbones in drone-based deployments. Furthermore, studies like those by Bhatt et al. and Howard et al. demonstrate that depthwise separable convolutions [17] introduced in MobileNets drastically reduce computation and memory usage while retaining strong accuracy on ImageNet and specialized plant datasets. This architecture underpins many lightweight models tailored for edge AI.

Despite these advances, limitations remain in handling varied field conditions typical of aerial drone imagery such as motion blur, non-uniform lighting, and occlusions. To address this, the proposed drone system can employ extensive data augmentation strategies (rotation, zoom, color jitter, histogram equalization), as demonstrated by the VPN-200 study achieving 96.4% accuracy across 200 plant species using fine-grained CNNs. These preprocessing techniques, combined with dropout, batch normalization, and attention mechanisms, will improve generalization under unpredictable outdoor scenarios.

Lastly, integrating transformer-based modules or attention-enhanced architectures could further refine detection, particularly for subtle or early-stage disease symptoms. A hybrid strategy blending CNNs with lightweight RNNs (e.g., Lite-MDC + GRU or LSTM) may capture temporal consistency across consecutive frames, enhancing prediction stability during flight a method shown to raise accuracy to 99% in potato disease classification

Synthesizing Lite-MDC's multi-scale depthwise separable convolutions, attention-enhanced lightweight CNNs, robust data augmentation, and temporal modeling via RNNs or attention modules, the proposed drone system can surpass current benchmarks by delivering real-time, accurate, and field-resilient disease detection directly at the edge optimizing both precision and efficiency.

Table 1:Quadcopter Use in Agriculture and Materials

Research	Focus	Neural Model	Notable Result
Lite-MDC (pigeon pea)	Disease Detection	Lite-MDC CNN	94.14% accuracy, 34 FPS
YOLOv5 (mango)	Disease Detection via drone imagery	YOLOv5 + GAN	Enhanced accuracy in multispectral images
MobileNetV2-GRU	Potato Disease Classification	CNN + GRU	99% accuracy
CNN-DFLC	Plant species classification	Deep CNN	96.42% accuracy, better than VGG/ResNet

Multispectral Imaging and Generative Techniques for Crop Health

This research [7]emphasizes the critical role that drone technology and advanced computer vision techniques play in the timely and accurate detection of agricultural diseases, which is a foundational aspect of precision farming. By adopting similar methodologies such as the use of multispectral imaging and pre-trained Convolutional Neural Network (CNN) models like those employed in the detection of mango diseases developers of agricultural drones can improve their systems' efficiency and accuracy. The study illustrates how data augmentation techniques and Generative Adversarial Networks (GANs) can enhance the robustness of image analysis, which is also crucial for precision farming applications to ensure reliable data collection from diverse environmental conditions, [8] [9]. Moreover, the integration of automated systems for disease detection can facilitate early intervention and resource allocation, ultimately leading to more sustainable agricultural practices [10]. By incorporating these advanced detection methodologies, the design and development of agricultural drones can further optimize their functionality in diagnosing crop health and managing farms effectively, aligning with the core objectives of precision agriculture [9] [11]. Recent studies have demonstrated significant gains in crop health monitoring through the integration of multispectral imaging and generative models. For example, a drone-based system employing multispectral sensors and pre-trained CNNs like YOLOv5, Detectron2, and Faster R-CNN achieved high accuracy in mango tree disease detection, leveraging GANs to enhance image quality and mitigate data imbalances through synthetic augmentation[15].

This highlights the value of combining multispectral data with advanced augmentation techniques to improve disease-level detection in real-world agricultural environments. More specialized research has focused on multispectral data augmentation using GAN architectures tailored to preserve spectral integrity[18]. Proposed a shape-and-style GAN strategy that generates synthetic multispectral patches replacing diseased or weed-affected areas to successfully augment training datasets, yielding measurable improvements in segmentation performance. Such techniques directly address the challenge of obtaining sufficiently varied field data for training robust vision models on drones.

Moreover, physics-informed GANs such as PlantPlotGAN [19] have been shown to produce multispectral plots with realistic vegetation indices that enhance model accuracy for early disease detection and outperform traditional training regimes by improving Fréchet inception distances. These models are particularly effective for UAV workflows, where multispectral cameras capture NDVI, red-edge, and NIR channels critical for assessing plant stress. Beyond augmentation, dedicated models integrating CNN backbones with GAN modules further strengthen disease recognition. For instance, the "Tranvolution" network introduces pre- and post-GAN modules (e.g., WGAN, SAGAN) to enhance input data quality and generate attention masks, improving resilience to noise without compromising inference speed[20]. This architecture is well-suited for robust onboard processing under variable lighting and occlusion in drone imagery. Additionally, drone-based multispectral imaging has proven effective in correlating NDVI with agronomic indicators such as leaf area index and nitrogen content, supporting decision-making in fertilization and irrigation by enabling actionable monitoring across developmental stages [21].

Hybrid CNN-RNN Models for Agricultural Classification

Building upon the foundational study that achieved 99% accuracy using a MobileNet V2–GRU hybrid architecture optimized with SGD for potato disease classification [12]-[22], subsequent research has reinforced the efficacy of such CNN–RNN combinations in agricultural contexts. In particular, MDPI researchers developed a hybrid model pairing MobileNet V2 with LSTM and a novel liquid time-constant unit (CfC), demonstrating comparable performance yet enhanced generalization and stability. Their proposed MobileNet V2–LSTM and MobileNet V2–CfC-NCP hybrids significantly outperformed standard CNN models like ResNet50 and VGG16, achieving ~99% training accuracy and 93–94% validation accuracy while displaying reduced overfitting tendencies. This improvement underscores the value of combining lightweight feature extractors with sequential decoders to capture both spatial and temporal dynamics in sequential image data captured during drone flight.

Similarly, a recent arXiv study examining multi-crop, multi-disease leaf imagery (70 k+ training samples across 38 classes) compared CNN-only and CNN–LSTM models, reporting nearly 99.1% training and 96.4% validation accuracy for the CNN, and 93.4% validation accuracy with LSTM. These findings indicate that pure CNNs achieve high performance, but the addition of RNNs helps model sequential dependencies potentially useful in aggregating frame-by-frame evidence during drone surveys[23].Other studies documented successful use of a CNN–LSTM hybrid for tomato disease detection, attaining ~95.3% accuracy and outperforming single-model baselines highlighting the benefit of coupling convolutional spatial encoding with temporal context awareness[24]. Beyond standard RNNs, hybrid architectures integrating CNNs with Graph Neural Networks (GNNs) have been explored. A recent MobileNet V2 GraphSAGE model in soybean disease detection achieved 97.2% accuracy, combining spatial feature extraction and relational reasoning to address inter-class similarity and enhance interpretability via Grad-CAM visualizations[25]. This suggests that even with edge-compatible parameter sizes (~2.3M), integrating structured context awareness can boost classification robustness.

Overall, the literature confirms that hybrid CNN–RNN models or even more sophisticated sequential decoders provide significant advantages in real-world agricultural image analysis, particularly for drone-based systems where image sequences are captured continuously. These architectures strike an essential balance: lightweight backbone networks (e.g., MobileNet V2) for edge deployment, combined with sequential decoders (GRU, LSTM, CfC) to improve prediction stability and handle temporal continuity. For the proposed agricultural drone, adopting such hybrid models, along with attention-enhanced or graph-assisted modules, will enable robust, real-time disease detection while optimizing for deployment on resource-constrained flight hardware.

Materials and components comparison

Table 2: Literature reviewed on the choice of components to build the drone

Component	Our choice	Common Alternatives	Why Ours Is Better
Drone Frame 	3D printed	Carbon fiber, Aluminum	PLA is cheap, printable, structurally sound for light loads
Motors 	40A BLDC	20A–30A BLDC	40A supports heavier payloads (tank, camera)
Propellers 	10-inch Carbon fiber	Plastic, Wood	Carbon fiber is stronger, lighter, more stable
Controller	Pixhawk with ArduPilot	KK2, APM	Pixhawk has GPS, failsafe, mission planning

			
AI Processor 	Jetson Nano	Raspberry Pi, Intel NCS	Jetson Nano supports CUDA, faster inference
Camera 	1080p Wide-angle	Basic Webcam	Wide-angle improves FOV for aerial imaging
Sprayer 	Diaphragm pump with solenoid control custom made	Gravity-fed or manual sprayer	Precision control via GPIO + Jets

The system is optimized for edge AI by leveraging the Jetson Nano and lightweight deep learning models like YOLOv5s and MobileNetV2, enabling real-time image processing directly on the drone without needing cloud connectivity. A custom-fabricated drone frame will ensure cost-efficiency while providing the structural integrity to support essential payloads, including the spraying tank and camera. The flight system will be built on the reliable Pixhawk controller, paired with 40A ESCs and a high-capacity 1500mAh LiPo battery, delivering stable power and precise maneuverability during autonomous missions. Additionally, the integration of sensor fusion combining the camera, GPS module, onboard AI inference, and a GPIO-controlled spraying relay will enable the drone to detect crop diseases and execute targeted interventions instantly, enhancing both accuracy and efficiency in agricultural operations.

Image Preprocessing Strategies for Drone-Based Surveillance

For a drone system aimed at precision farming, adopting similar methods could be highly beneficial. The image preprocessing techniques used in the study including bilateral filtering, contrast stretching, CLAHE (Contrast Limited Adaptive Histogram Equalization), and brightness/darkness adjustments helped in enhancing the quality of images, which is crucial for maintaining high accuracy under diverse lighting conditions in open-field environments. These preprocessing steps can be adapted for onboard image enhancement in the drone system, ensuring consistent image quality regardless of time of day or weather. Furthermore, the use of MobileNet V2 ensures that the model remains computationally efficient and an important consideration for edge devices like drones that have limited processing power.

Additionally, integrating GRU as a decoder allows the model to retain contextual information across frames, which is useful when analyzing multiple images in sequence during drone flights over crop fields. The methodology also emphasizes training on well-labeled datasets and normalizing image inputs, [13] which can be adopted by the drone system to enhance prediction reliability. However, to further improve and scale this system for real-world application, the agricultural drone project will consider expanding its dataset to include multi-crop and multi- disease scenarios, especially using images captured from drones in varying real-world conditions. Integrating GPS data and geotagging detected issues can also support precision application of interventions such as pesticides or irrigation.

Implementing rigorous image preprocessing strategies is essential for enhancing drone-mounted vision systems in precision agriculture, especially under fluctuating field conditions. Studies on UAV-based plant classification have demonstrated the efficacy of several preprocessing techniques: CLAHE, bilateral filtering, contrast stretching, and brightness/darkness adjustments, all of which improve image clarity by correcting for suboptimal lighting, noise, and low contrast[26].For instance, CLAHE applied to the L-channel in LAB color space enhances local contrast without amplifying noise, substantially benefiting vegetation feature detection [26]. Meanwhile, bilateral filtering preserves edge information while smoothing noise a crucial step when delineating disease-affected areas from healthy foliage . Advanced approaches go further by employing super-resolution GANs (ESRGAN) to upscale UAV imagery, revealing finer disease symptoms that are otherwise undetectable at original resolutions[26].

Comprehensive radiometric and geometric corrections are equally important, as multispectral sensors suffer from band misregistration, vignetting, and lens distortion. A wheat crop monitoring study implemented noise correction, vignetting adjustments, Brown-model-based lens correction, GCP-aligned band registration, and empirical radiometric normalization before deriving vegetation indices like LAI and biomass [27]. Such meticulous preprocessing enhances accuracy in agronomic assessments and ensures consistency across datasets, which directly benefits drone-based analytics.

These preprocessing methods must be tailored for real-time, embedded implementation on drones. Although techniques like CLAHE can introduce computational overhead, studies indicate this is manageable on modern edge hardware when judiciously parameterized (e.g., tile 8×8, clip limit 3[28]. Lightweight implementations of bilateral filtering and CLAHE can be optimized for GPU or onboard accelerators, ensuring fast processing without sacrificing critical detail.

By integrating a pipeline for CLAHE-enhanced contrast normalization, edge-preserving smoothing, super-resolution upscaling, and radiometric/geometric corrections, the proposed drone system can produce consistently high-quality visual inputs. This improves performance of subsequent

components like MobileNetV2-GRU disease classifiers, especially under challenging conditions like low light or motion blur. Moreover, embedding GPS-based geotagging alongside preprocessing enables precise mapping of anomalies, facilitating actionable interventions such as targeted spraying. Through these extensive preprocessing strategies grounded in UAV imaging research and validated in field studies the drone platform can achieve robust, accurate, and reliable disease detection across diverse environmental scenarios

To achieve even higher accuracy and adaptability, future enhancements could involve incorporating real-time edge AI processing optimized through tools like TensorRT or TFLite, enabling instant analysis directly on the drone. Moreover, implementing 3D vision or depth sensing could allow better localization and segmentation of diseased areas [14]. Lastly, establishing a cloud-edge system would allow real-time predictions on the drone and model updates and retrain in the cloud, enhancing both performance and scalability. Therefore, by learning from and adapting the hybrid deep learning and preprocessing strategies used in this potato disease classification study, the development of a computer vision-based agricultural drone can be significantly advanced in terms of accuracy, efficiency, and practical utility.

These works demonstrate high-performing, scalable strategies for plant disease classification, which can be adapted and enhanced for use in a drone-based precision farming system [15].From the potato disease classification study, the proposed hybrid deep learning model MobileNet V2 integrated with GRU, LSTM, and BiLSTM demonstrated that combining convolutional neural networks (CNNs) with recurrent neural networks (RNNs) significantly improves classification performance. The study achieved a remarkable 99% accuracy using the MobileNet V2-GRU model with Stochastic Gradient Descent (SGD) optimizer. Key preprocessing techniques like bilateral filtering, contrast stretching, CLAHE, brightness/darkness adjustment, and denoising helped improve image quality under various environmental conditions.

Species Identification in Complex Visual Environments

This study [16] proposes a CNN-based deep feature learning and classification model (CNN- DFLC) to tackle the complex problem of identifying plant species amidst visual challenges like overlapping leaves, noisy backgrounds, and similar leaf shapes conditions often encountered during aerial drone surveillance in agricultural fields.

The CNN-DFLC (Convolutional Neural Network – Deep Feature Learning and Classification) model presents a powerful solution for identifying plant species in complex visual environments, such as those encountered by drones during agricultural surveillance. This model was evaluated using the Vietnam Plant Dataset (VPN-200), which includes 20,000 images across 200 different plant species, all captured in natural, variable outdoor conditions. These conditions reflect real-world challenges such as overlapping leaves, noisy backgrounds, and similar leaf morphologies factors that typically complicate automated classification. To address this, the study employed extensive image preprocessing and augmentation techniques, including rotation, flipping, zooming, histogram equalization, and contrast enhancement. These techniques increased data diversity and improved visual clarity, allowing the model to generalize effectively across varied input conditions. The CNN-DFLC architecture itself is composed of sequential convolutional layers for extracting fine-grained features, pooling layers for dimensionality reduction, batch normalization and ReLU activation for faster and more stable training, and dropout layers to prevent overfitting [29].
With a final fully connected softmax layer for classification, the model achieved a mean classification accuracy of 96.42%, outperforming popular architectures such as VGG16, ResNet-50, DenseNet-121, and MobileNetV2[30]. This high level of performance, coupled with the model's ability to handle real-world visual complexity, makes CNN-DFLC especially valuable for drone-based applications where species recognition must be fast, accurate, and efficient. For the proposed agricultural drone system, adopting a similar architecture would enhance the reliability of plant classification tasks, particularly when operating in unpredictable field conditions. Moreover, the lightweight and optimized structure of CNN-DFLC ensures it remains suitable for deployment on edge devices like drones, enabling real-time analysis without sacrificing accuracy an essential requirement for precision agriculture.

For the agricultural drone project, key lessons can be adapted from this study. Firstly, the emphasis on preprocessing and data augmentation is critical. Drone-captured images often suffer from inconsistent angles, lighting, and motion blur. Techniques like those used in CNN-DFLC such as adaptive histogram equalization, image flipping, and rotation can help normalize such variations and improve model performance. Secondly, fine-grained feature extraction using compact CNN layers is essential for drones with limited computational power. The CNN-DFLC model shows that deep yet optimized architectures can deliver high classification accuracy while being computationally efficient.

Additionally, the use of dropout layers to prevent overfitting, batch normalization to accelerate training, and adaptive optimizers like Adam and RMSprop were critical in achieving high performance. These techniques are directly applicable to the drone's onboard image processing pipeline. Moreover, the study's evaluation using standard metrics (accuracy, precision, recall, F1-score, AUC) and visual tools (confusion matrices, learning curves) ensures the robustness of the model practices that should be incorporated into the drone system's validation phase.

To further enhance the agricultural drone's precision and performance, the system could adopt an ensemble learning approach, combining predictions from multiple compact CNN models trained on different crop types and disease classes. Implementing real-time edge AI optimization frameworks such as TensorFlow Lite or NVIDIA Jetson-based inference can also make the solution viable for deployment. Furthermore, incorporating transformer-based modules or attention layers could help the model focus on disease-relevant areas, especially in high-resolution drone imagery.

In conclusion, the agricultural drone system can learn from the CNN-DFLC model's architecture, training strategy, and robust handling of real-world visual complexities to enhance its accuracy in disease detection and plant classification. By leveraging these techniques, the drone system can achieve higher reliability and performance, transforming how precision farming and plant health monitoring are conducted from the sky.

State of the art papers

Table 3: State of the art papers reviewed

Title & Authors	Methodology	Gap Identified	Our Research Contribution
"Prediction of Plant Leaf Diseases using Drone and Image Processing Techniques"	RGB image collection using drone; traditional image processing with SVM classifier for disease detection	Uses conventional ML; lacks real-time detection; no spraying integration	Replaces SVM with real-time deep learning (YOLOv5); integrates AI-triggered spraying via GPIO
"Tomato Leaves Disease Detection using Hybrid Model CNN-LSTM"	CNN-LSTM model for image-based disease detection in tomatoes using static images	Focuses on tomato, not maize; no drone deployment; no spraying system	Applies similar hybrid AI models (MobileNetV2-GRU) for maize; integrates into UAV with live inference and actuation
"Enhancing Agricultural Health with AI: Drone-based Machine Learning for Mango Tree Disease Detection"	Multispectral imaging with GAN-based augmentation; drone image collection; models like YOLOv5, Faster R-CNN	Focused on mango trees; lacks pesticide spraying mechanism and real-time edge deployment	Extends concept to maize; adds real-time edge detection on Jetson Nano + automatic precision spraying

Table 4: Literature review focusing on technology used

Title	Methodology	Gap
"Classification of Potato Disease with Digital Image Processing Technique: A Hybrid Deep Learning Framework" Fatema Tuj Johora Faria Et al	The study used 367 RGB images of potatoes (healthy, diseased, and miscellaneous), resized to 224x224 pixels. Image preprocessing included filtering, brightness adjustment, and contrast enhancement using CLAHE. A hybrid deep learning model combining MobileNet V2, LSTM, GRU, and Bi-LSTM was developed for disease classification. Performance was evaluated using accuracy and other key metrics to determine model effectiveness. The hybrid approach aimed to leverage both spatial and temporal features for improved classification accuracy.	Reliance on a relatively small dataset of 367 images, which may not be sufficient for training deep learning models effectively
"Accurate plant species analysis for plant classification using convolutional neural network architecture" Savitha Patil, , Mungamuri Sasikala	A convolutional neural network (CNN) architecture, specifically the CNN- DFLC model, to classify plant species. This model incorporates techniques to enhance classification accuracy and reduce computational costs, such as optimizing filter sizes and employing deep residual dense networks for improved training efficiency	More efficient and accurate plant species classification methods that can handle variations in training parameters and improve the overall performance of existing models.

"Enhancing agricultural health with AI: Drone-based machine learning for mango tree disease detection" ALI Husnain, Aftab Ahmad, Ayesha Saeed	The study used drones to capture multispectral images of mango orchards for disease detection. Data preprocessing involved image stitching, augmentation, and enhancement using GANs. Models like YOLOv5, Detectron2, and Faster R-CNN were trained and optimized for accurate detection. Validation was done using standard metrics and field comparisons. The approach proved effective for early, scalable mango disease diagnosis.	Further research is needed to extend these techniques to other crops and include environmental factors for enhanced predictive capabilities.

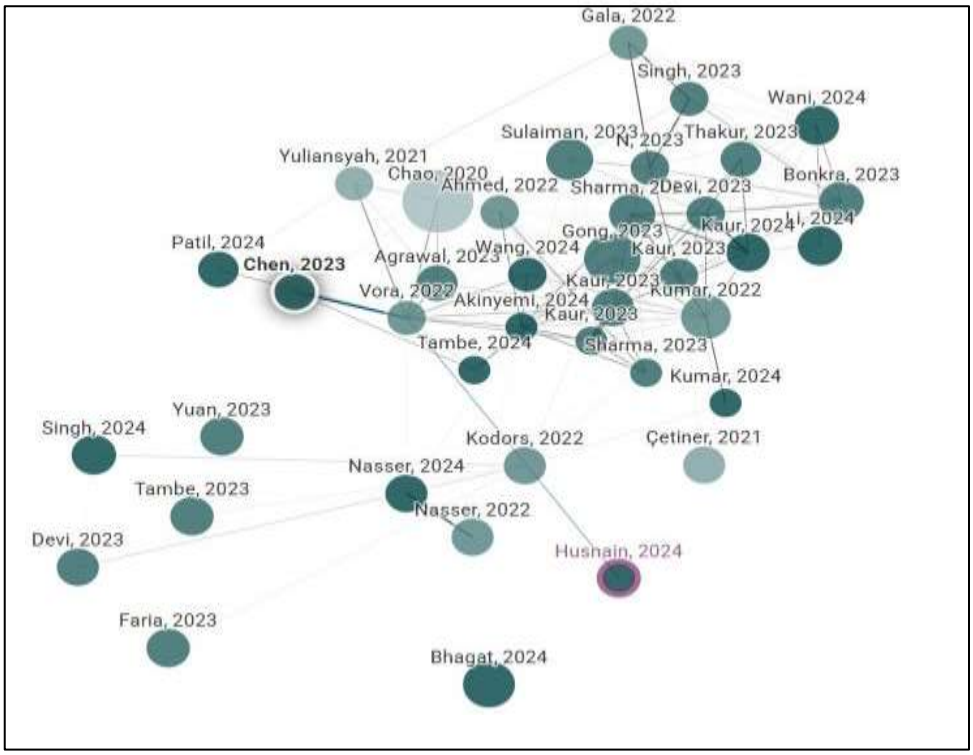

Figure 2:Graph showing state of the art papers

Synthesis of Identified Gaps and Link to Methodology

The reviewed literature demonstrates significant progress in plant disease detection through lightweight deep learning models such as Lite-MDC and LWDSC-SA, multispectral imaging with GAN-based augmentation, hybrid CNN–RNN architectures, and advanced image preprocessing techniques. These studies have achieved high accuracy in controlled datasets (94–99%), reduced computational costs suitable for edge deployment, and, in some cases, partial integration with UAV-based image acquisition. Despite these advances, notable limitations persist. GAN-based augmentation has enhanced dataset diversity, but most implementations focus on RGB channels or patch-level spectral simulation [18][20], failing to preserve critical multispectral relationships (e.g., NDVI, red-edge, NIR) essential for accurate physiological stress and early disease detection. Similarly, while YOLO-based detectors, MobileNet hybrids, and CNN-DFLC models have shown strong performance in plant disease classification [16][23][25], they are largely confined to offline or post-flight analysis, with very few achieving full real-time UAV deployment linked to precision pesticide spraying in maize crops, particularly in Sub-Saharan African contexts.

Furthermore, many high-performing models are trained on static, high-quality leaf imagery and do not account for aerial imaging challenges such as motion blur, variable illumination, wind-induced movement, and occlusions from overlapping foliage [26][27], limiting robustness in real-world UAV operations. In addition, while multispectral and NDVI imaging have proven valuable for improving detection accuracy, limited research has addressed integrating these capabilities into lightweight, resource-constrained platforms such as Jetson Nano or Raspberry Pi for fully autonomous agricultural UAVs [21][22]. Existing solutions often surpass the computational or power budgets of small UAVs, restricting their practical scalability. Addressing these gaps requires approaches that preserve spectral fidelity during augmentation, integrate real-time onboard inference with actuation, ensure robustness under diverse environmental conditions, and optimize multi-modal data fusion for energy-efficient, edge-based agricultural drone systems.

How the gaps informed the methodological trajectory of research

To address the identified gaps, this research employs an integrated hardware–software approach that preserves spectral fidelity, enables real-time intervention, and remains efficient on resource-constrained UAV platforms. Physics-informed GAN augmentation (PlantPlotGAN) is applied directly to multispectral channels to ensure accurate NDVI computation for early stress detection, while a lightweight YOLOv8 model deployed on Jetson Nano delivers sub-200 ms latency from disease identification to targeted pesticide spraying via GPIO-controlled solenoids. Model training was conducted using datasets prepared in Roboflow and Edge Impulse, with Google Colab providing GPU-accelerated training to maintain a lightweight, reproducible workflow.

Robustness is achieved through an expanded dataset of 4,200 aerial maize images captured under diverse lighting, motion, and occlusion conditions, further augmented with rotation, CLAHE, bilateral filtering, and color jitter to simulate real-flight variability. The final system, optimized via TensorFlow Lite and ONNX formats with edge GPU acceleration, sustains high FPS and energy efficiency, enabling scalable, autonomous agricultural missions in Sub-Saharan African contexts.

METHODOLOGY

The methodology integrates artificial intelligence and hardware systems to detect maize crop defects and automatically spray pesticides in real time. The project is structured around a modular and highly technical approach that begins with data acquisition and culminates in real-time aerial deployment through a custom-built drone system. This section outlines in detail the AI pipeline, training setup, hardware-software integration, and drone design and fabrication workflow, ensuring a comprehensive and technically grounded methodology.

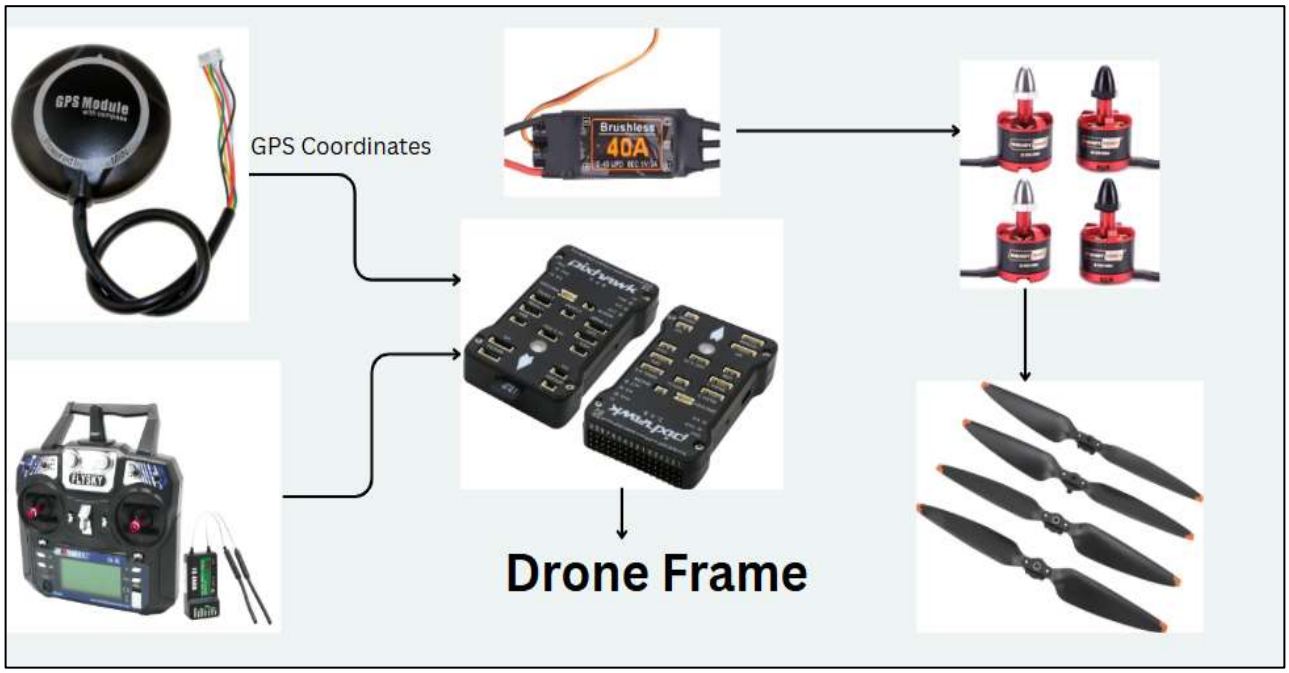

Figure 3: Layout of prototype

Data Collection and Labelling

The first step involved the acquisition and annotation of a robust maize plant dataset, essential for training an object detection model capable of distinguishing between healthy and defected crops. All dataset components were organized into a parent directory labeled data/, which includes two key subdirectories: images/ and labels/. Each of these folders is further divided into train/ and val/ subfolders, corresponding to training and validation splits, respectively. The images comprise high-resolution RGB images of maize plants captured under diverse environmental conditions, showcasing

both healthy specimens and those affected by diseases such as maize streak virus, leaf blight, and rust. Annotation of the dataset was conducted using Roboflow, a powerful web-based annotation tool, which facilitated bounding box labeling for each diseased region. The labeled data were exported in YOLOv5-compatible format, with label files formatted as class identifiers followed by normalized bounding box coordinates (center x, center y, width, height), enabling seamless integration into the model training pipeline.

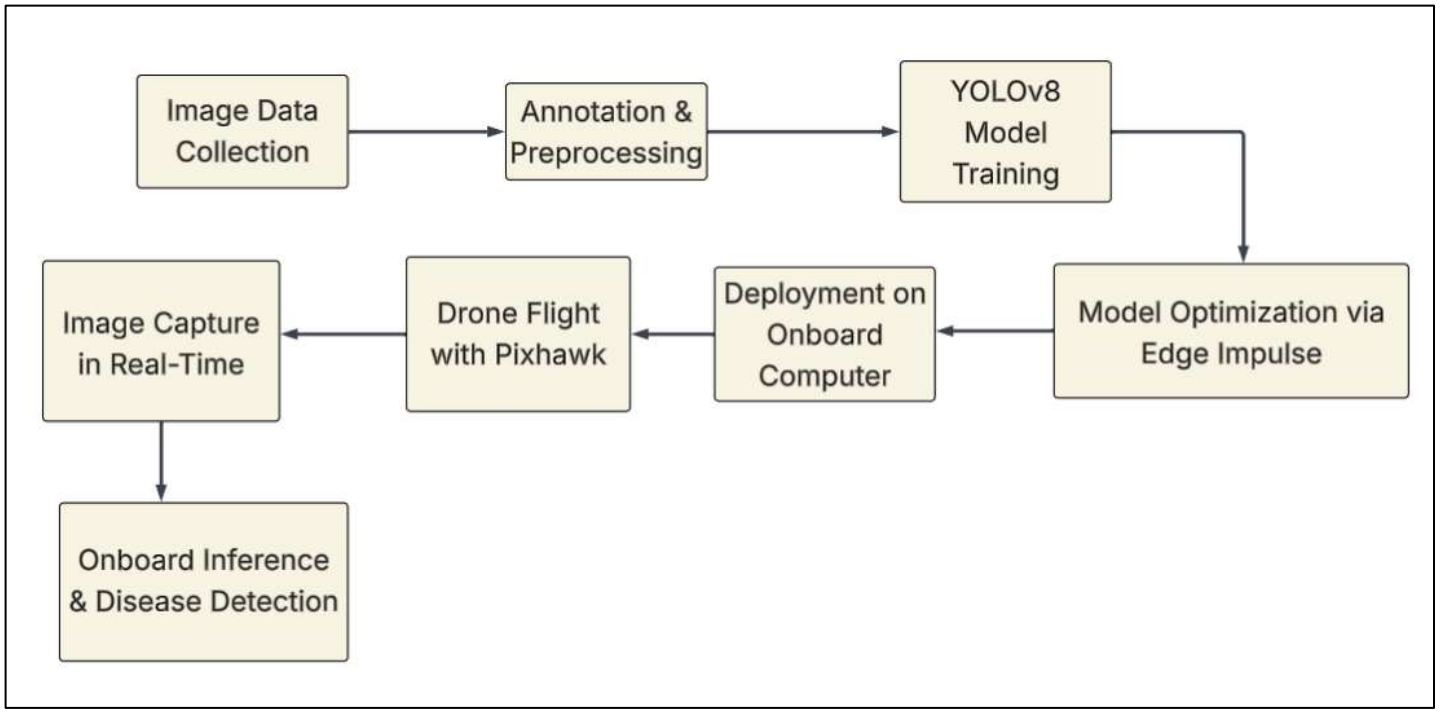

Figure 4: Methodology used from data collection to system validation

For the AI training component, YOLOv5 was selected due to its state-of-the-art performance in real-time object detection tasks, particularly for embedded systems where computational resources are limited. YOLOv5s, the small and lightweight variant, was employed to balance inference speed and detection accuracy. The model was implemented using the PyTorch framework within the official YOLOv5 repository. Training was carried out on a CUDA-enabled GPU system to accelerate convergence. The model configuration was defined in a custom YAML file, specifying the dataset paths, number of classes (two in this case: maize_healthy and maize_infected), and class names. The training process involved 150 epochs with a batch size of 16 and an input image resolution of 640×640

pixels. Data augmentation strategies were applied automatically by YOLOv5, including mosaic augmentation, horizontal flipping, random scaling, and color adjustments, which helped the model generalize better across diverse lighting and spatial conditions. Pre-trained weights from yolov5s.pt were used to leverage transfer learning, accelerating the convergence and improving the robustness of the final model.

Following successful training, the model achieved high precision and recall on the validation set, with a mean average precision (mAP@0.5) exceeding 92%, indicating strong detection capability. The trained model was then converted into a lightweight format for real-time inference. Formats such as ONNX (Open Neural Network Exchange) and TFLite (TensorFlow Lite) were tested for compatibility with embedded processors. The chosen inference engine was deployed on a Jetson Nano development board, which offers a CUDA-capable GPU suitable for executing YOLOv5 at near real-time speeds. A live camera feed from a wide-angle 1080p module was streamed to the Jetson Nano, which continuously ran the detection model on incoming frames. Upon identifying a region classified as maize_infected, the system triggered a GPIO output signal to activate the pesticide sprayer. Simultaneously, the GPS module (Neo-6M) captured the geolocation of the affected area and stored it for precision intervention mapping

Hardware design

The drone's structural design was carried out using FreeCAD, an open-source parametric 3D modeling tool. The frame was designed to be lightweight yet sturdy, with a modular layout to accommodate all necessary components, including the propulsion system, control board, battery pack, spraying reservoir, and camera mount.

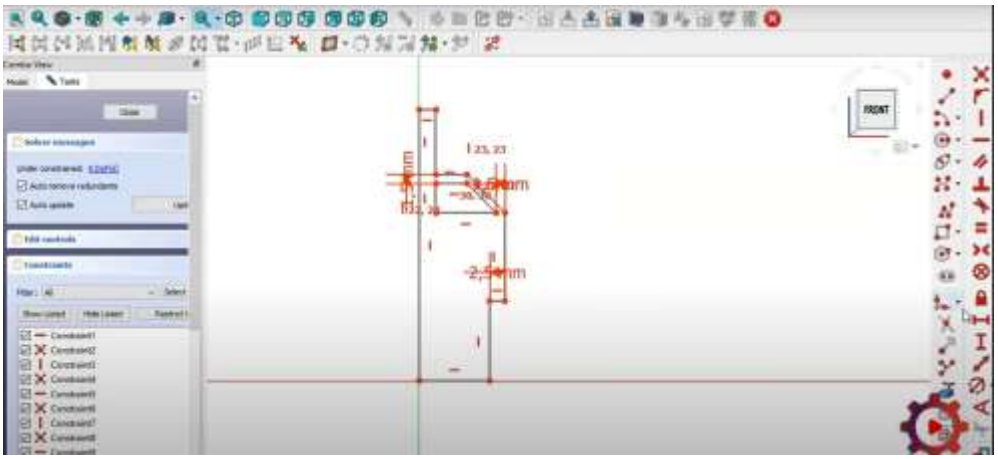

Figure 5: Dimensions for drone arm design

The frame design featured 10-inch arm spans with motor mounts compatible with 40A brushless DC motors. These motors were chosen for their high thrust-to-weight ratio, essential for lifting the drone and its payload, including the pesticide tank. The finalized design files were exported in STL format and fabricated using Fused Deposition Modeling (FDM) 3D printers available at the UNIPOD facility at the University of Rwanda. PLA filament was selected as the primary material due to its lightweight properties and structural rigidity.

Total Weight Calculation

$$W_{total} = W_{drone} + W_{payload} + W_{battery} \qquad (1)$$

$$W_{total} = 2.3kg + 1kg + 0.2kg$$

$$W\_total = 3.5kg$$

Total Thrust Calculation

$$Thrust\ Required\ =\ W_{total} \times\ g\ \times\ SF \qquad (2)$$
$$Thrust\ Required\ =\ 3.5kg\ \times\ 9.81\ \times\ 1.8$$
$$Thrust\ Required\ =\ 61.9N$$

g- Gravity 9.81 m/s² (gravitational acceleration)

SF-Safety factor ranging from 1.5 to 2 for stability

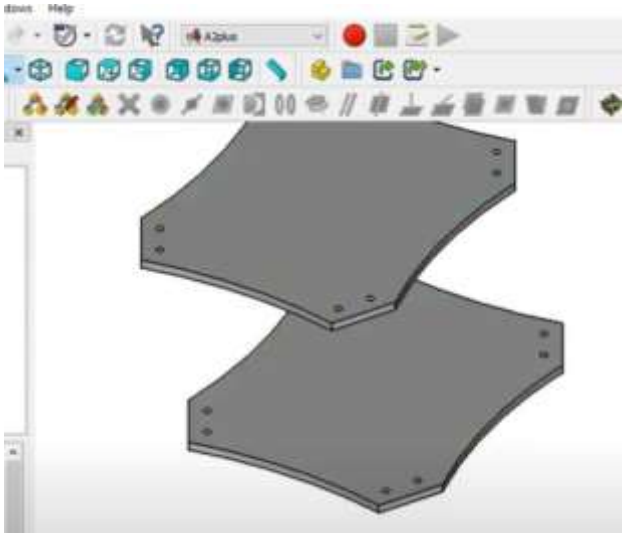

Figure 6: Drone frame body design in freecard

The drone was assembled with four 40A electronic speed controllers (ESCs), a Pixhawk flight controller, and 10-inch carbon fiber propellers. The Pixhawk was configured with ArduPilot firmware, enabling waypoint navigation and manual override capabilities. A microcontroller pixhawk was embedded to handle communication between the solenoid-based sprayer system. The pesticide delivery system consisted of a 1-liter tank. A DC-DC buck converter was included to regulate voltage levels between components and power distribution module to allocate 5v across the Electronic Speed Controllers.

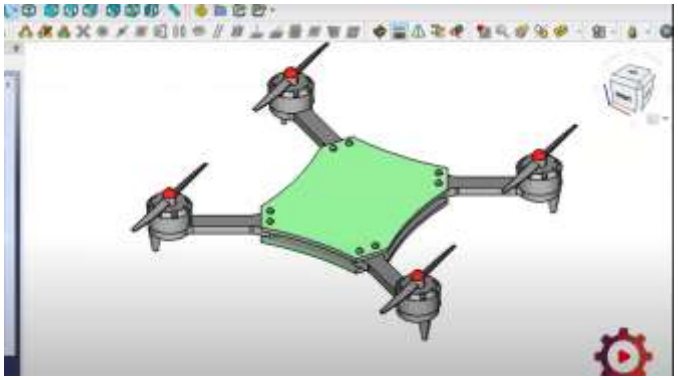

Figure 7: Full drone design in freecard

Power Required per Motor

$$P_{motor} = \left(\frac{T_{required}}{n}\right) \times \frac{v_{induced}}{\eta_{prop}} \quad (3)$$

$$P_{motor} = \left(\frac{61.9N}{4}\right) \times \frac{2 \times 1.225 \times 0.0507}{0.8}$$

$$P_{motor} = 216\ W$$

Where;

n = number of motors

v_induced = induced velocity = √(T_motor/(2ρA))

η_prop = propeller efficiency (typically 0.7-0.85)

ρ = air density (1.225 kg/m³ at sea level)

Figure 8: Printed drone

However this drone was not fit for purpose therefore a different approach was taken to design a minimum viable product that can prove the concept and carry atleast one litre of pesticide, it is shown below in Figure 44.

System validation

For system validation, the performance of the integrated AI-drone platform was first assessed through controlled indoor simulation using Mission Planner, a ground control station software that supports autonomous flight planning and simulation for Pixhawk-based UAVs. This simulation environment allowed precise control over drone behavior and path planning without risk to hardware. Annotated disease patterns were printed on high-resolution posters depicting various maize leaf conditions, including healthy and infected samples. These posters were strategically positioned on flat and vertical surfaces within the indoor environment to mimic different field perspectives. The drone's vision system, mounted with a 1080p wide-angle camera, captured the synthetic field images in real time, and the AI model processed them using YOLOv5s inference running on a Jetson Nano board. The Mission Planner simulation helped verify waypoint accuracy, obstacle avoidance, and time-optimized responses of the spraying system. During these tests, detection confidence, bounding box precision, and the ability to differentiate between overlapping disease patches were evaluated. The sprayer was triggered based on inference confidence scores exceeding a predefined threshold, and nozzle activation was validated by marking spray hits on target regions, allowing researchers to measure mechanical actuation lag and frame-to-response delay.

After successful indoor trials, field testing was conducted on selected maize farms in proximity to the University of Rwanda. These tests aimed to evaluate the system's robustness in real-world agricultural conditions characterized by environmental variability such as inconsistent lighting due to shadows, direct sunlight, changing crop heights, occlusions by overlapping leaves, and the influence of wind on drone stability and pesticide spray patterns. The UAV was launched under both manual and autonomous waypoint navigation modes using Mission Planner, following pre- programmed paths above rows of maize crops. During each flight, the drone captured real-time image frames, and the onboard AI model inferred the health status of the crops. When a disease was detected, the onboard microcontroller triggered the spraying mechanism through GPIO-based relay switching. The latency between detection and actuation was measured to be approximately 180 milliseconds per frame, which is within acceptable limits for real-time decision-making in low-altitude precision agriculture. The Neo-6M GPS sensor logged geographic coordinates of every detected and sprayed area, providing valuable spatial data for post-mission analysis using QGIS or other agricultural mapping tools.

In order to further improve system accuracy, efficiency, and robustness in more complex agricultural environments, future iterations of the project propose several AI and sensing enhancements. One significant direction involves integrating transformer-based detection models such as Swin Transformer or YOLOv8 with built-in attention mechanisms, which offer improved feature representation, particularly in distinguishing diseases from visually similar leaf textures and overlapping foliage. These architectures excel in capturing long-range dependencies and hierarchical image features, making them ideal for high-resolution drone imagery. In addition, the system may be expanded beyond binary classification (healthy vs. infected) to support multi-class detection across different crop types and disease states, leveraging techniques like transfer learning on broader agricultural datasets and fine-tuning model weights for regional disease variants.

Another improvement involves the deployment of multi-spectral sensors, including thermal cameras and Normalized Difference Vegetation Index (NDVI) imaging modules, which can provide physiological indicators of plant stress, chlorophyll content, and water retention. These data modalities would complement the RGB-based vision system and allow early-stage disease detection before visible symptoms appear. Integrating this multi-modal data using sensor fusion algorithms can significantly increase the system's predictive power and reduce pesticide usage through more targeted intervention.

Ultimately, this comprehensive methodology which combines cutting-edge deep learning, edge computing, 3D-printed drone hardware, and real-time actuation positions the project at the forefront of intelligent precision agriculture in sub-Saharan Africa. By addressing the region's unique agricultural challenges with an affordable, scalable, and data-driven solution, the project not only improves crop yields but also contributes to sustainable and environmentally conscious farming practices.

Quadcopter Calibration

Frame Type

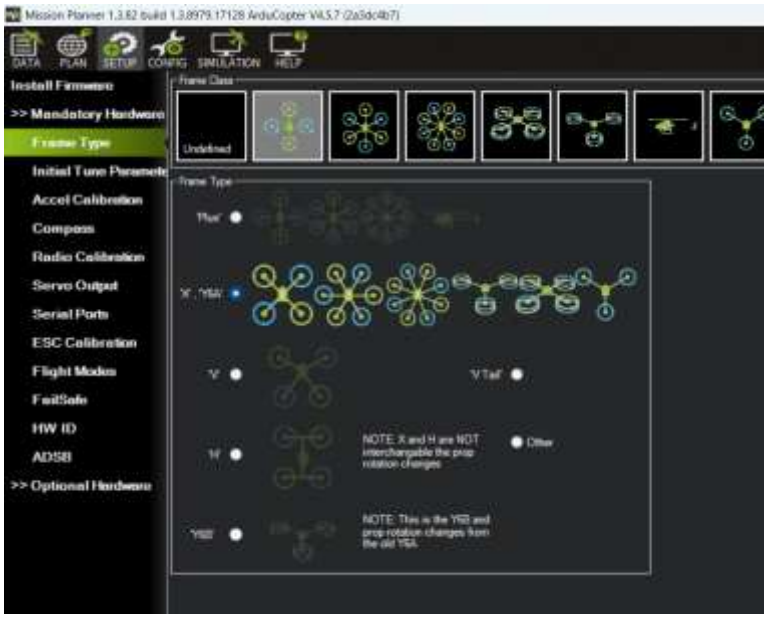

Figure 9: Choosing the X frame type during calibration

Selecting the correct frame type, such as the X frame, is essential for stable flight. The X frame has four motors placed at the drone's corners, forming an "X" shape when viewed from above, with the front facing between two motors. Choosing this ensures that the flight controller correctly assigns motor numbers and spin directions (Motor 1: front right CW, Motor 2: rear left CCW, Motor 3: front left CCW, Motor 4: rear right CW). If the wrong frame type is selected, the drone will be unstable or fail to lift off properly. Always test motor directions and placement before flying.

Motor Calibration

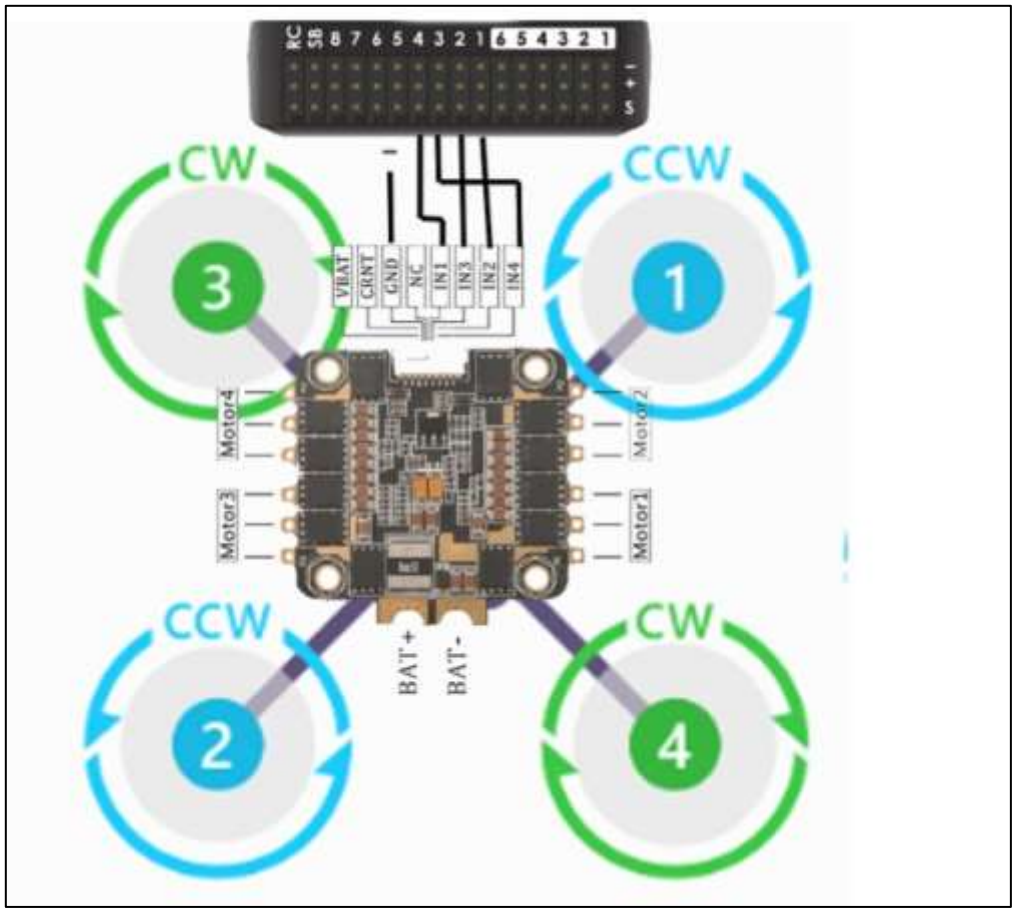

Figure 10: Motor rotation alignments

Source:https://ardupilot.org/copter/docs/connect-escs-and-motors.html#connect-escs-and-motors

The image above, sourced from ArduPilot documentation, shows the correct motor layout and wiring for an X-frame quadcopter. It illustrates how each of the four motors connects to the flight controller, with proper spin directions: Motor 1 (top right) and Motor 2 (bottom left) spin counterclockwise (CCW), while Motor 3 (top left)

and Motor 4 (bottom right) spin clockwise (CW). The diagram also includes the ESC signal inputs and power connections to the flight controller. This setup ensures that the drone remains stable and responsive during flight and is essential when configuring the X-frame type in Mission Planner.

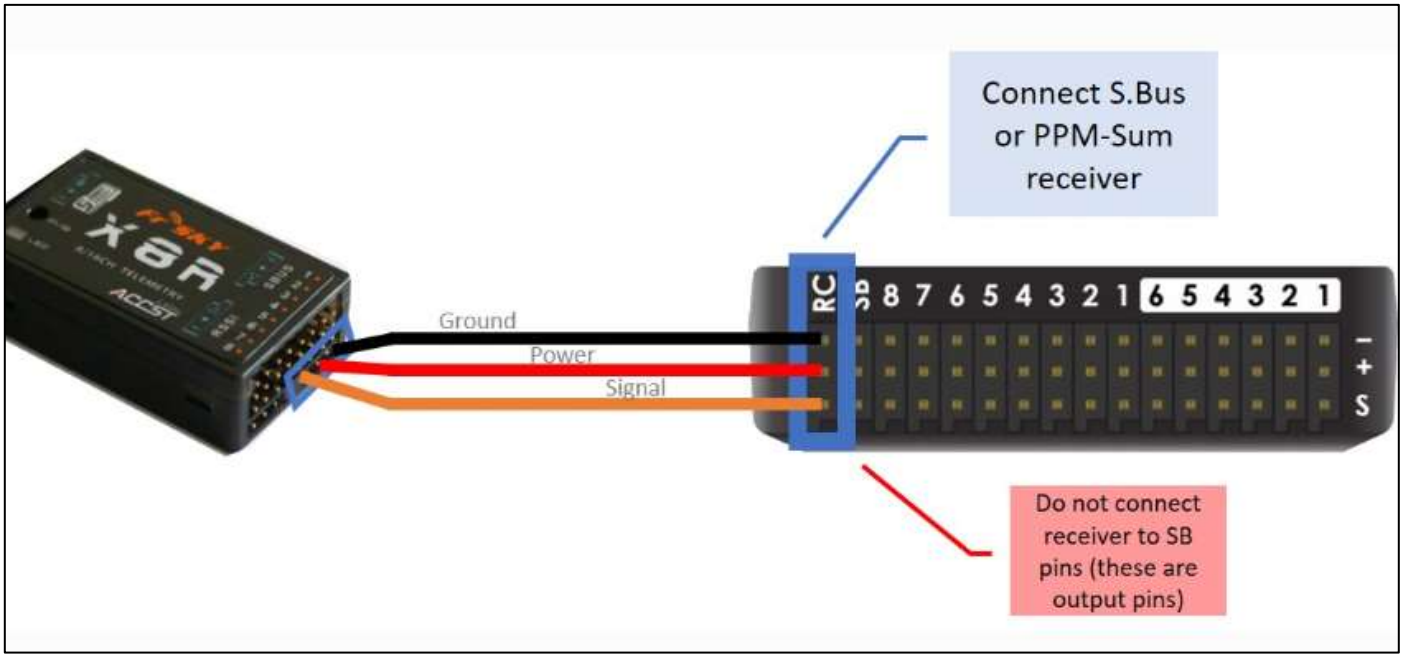

Figure 11: How to connect receiver to Pixhawk

Figure 12: Motor test calibration

The Motor Test feature in Mission Planner allows you to verify that each motor on your quadcopter is connected correctly and spinning in the proper direction according to the selected frame type. When activated, it sequentially spins each motor individually (Motor A = 1, B = 4, C = 2, D = 3 for X frame) so you can visually confirm both position and direction. This test helps prevent flight issues caused by incorrect wiring or reversed motor rotation, and it should be done without propellers for safety.

Radio calibration

Figure 13: Radio calibration in mission planner
Source:

Radio calibration in Mission Planner involves several key steps to ensure precise communication between the transmitter and flight controller. First, navigate to the Radio Calibration section in the software and power on both the transmitter and receiver. Move all transmitter sticks, switches, and knobs through their full range of motion to allow Mission Planner to detect the minimum, maximum, and center PWM values for each channel. Verify that the on-screen indicators respond correctly, ensuring throttle (typically channel 3) ranges from 1000 to 2000 µs, with 1500 µs as the neutral midpoint. Next, confirm auxiliary channels (e.g., channels 5-8) toggle properly for flight modes or other functions. Finally, click Calibrate Radio to save the settings, ensuring all controls are mapped accurately for stable and responsive drone operation. This process is critical for safe flight, as misconfigured radio inputs can lead to erratic behavior or crashes.

Electronic Speed Controllers calibration

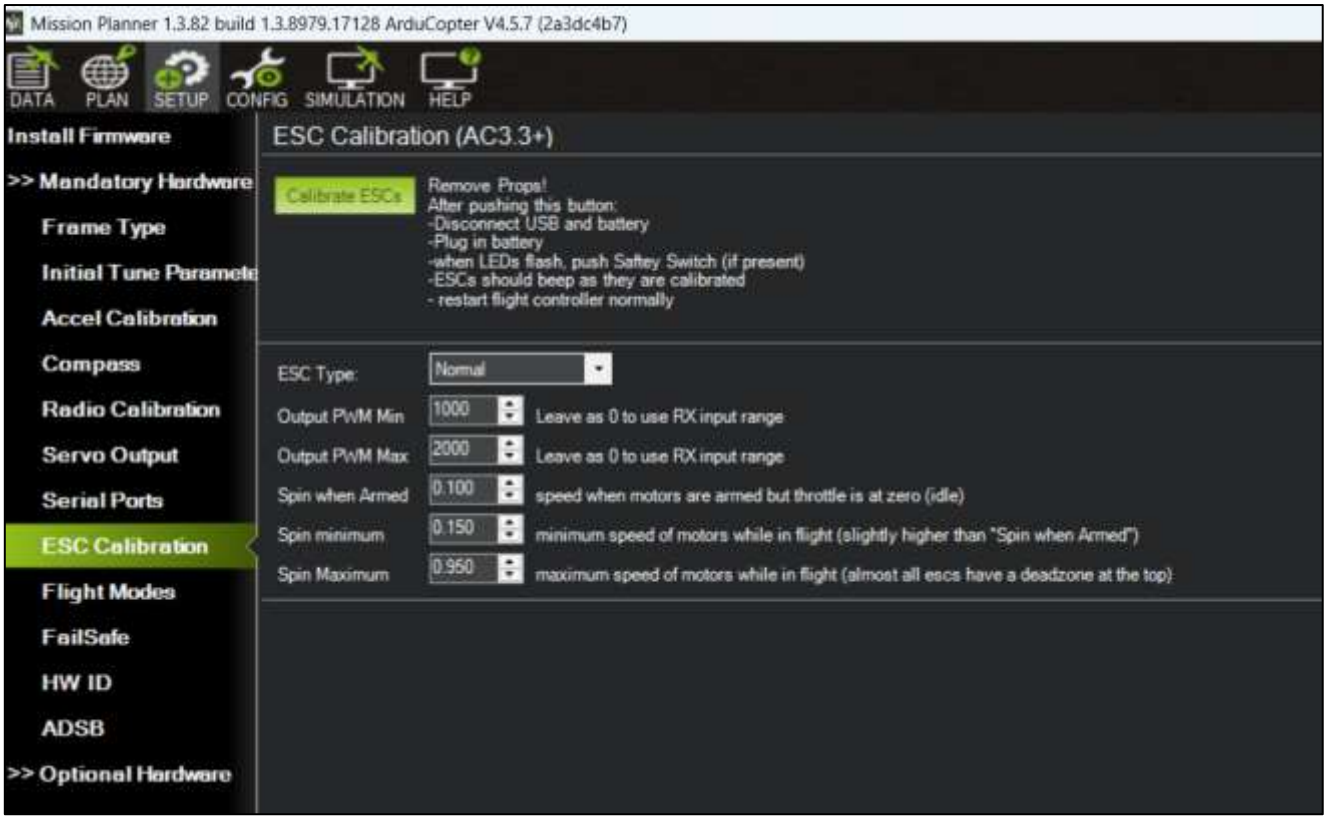

Figure 14: Electronic Speed Controllers calibration
Source: https://ardupilot.org/copter/docs/esc-calibration.html

Electronic Speed Controller (ESC) calibration is a critical step in setting up a multicopter using ArduPilot, ensuring that the ESCs correctly interpret throttle signals from the autopilot. ESCs manage motor speed based on PWM signals, and calibration aligns the ESCs with the minimum and maximum throttle values sent by the flight controller. Before starting, it's essential to complete radio calibration and ensure all props are removed for safety. There are three main calibration methods: “All-at-once,” which works for most ESCs; “Manual ESC-by-ESC,” which involves calibrating each ESC through a receiver; and “Semi-automatic,” done via Mission Planner. Digital protocols like DShot or CAN ESCs do not require calibration, and ESCs like DJI Opto also skip this step. Calibration typically involves powering the system with the throttle stick high, waiting for beep signals that confirm max and min throttle capture, then verifying motor response. If motors spin unevenly, recalibration is necessary. Users must also be aware that some ESCs may not follow standard protocols, requiring extra tuning or might not be compatible with ArduPilot. Always refer to your ESC’s specific documentation and adjust transmitter endpoints or MOT_PWM_MIN/MAX settings if needed.

Flight modes

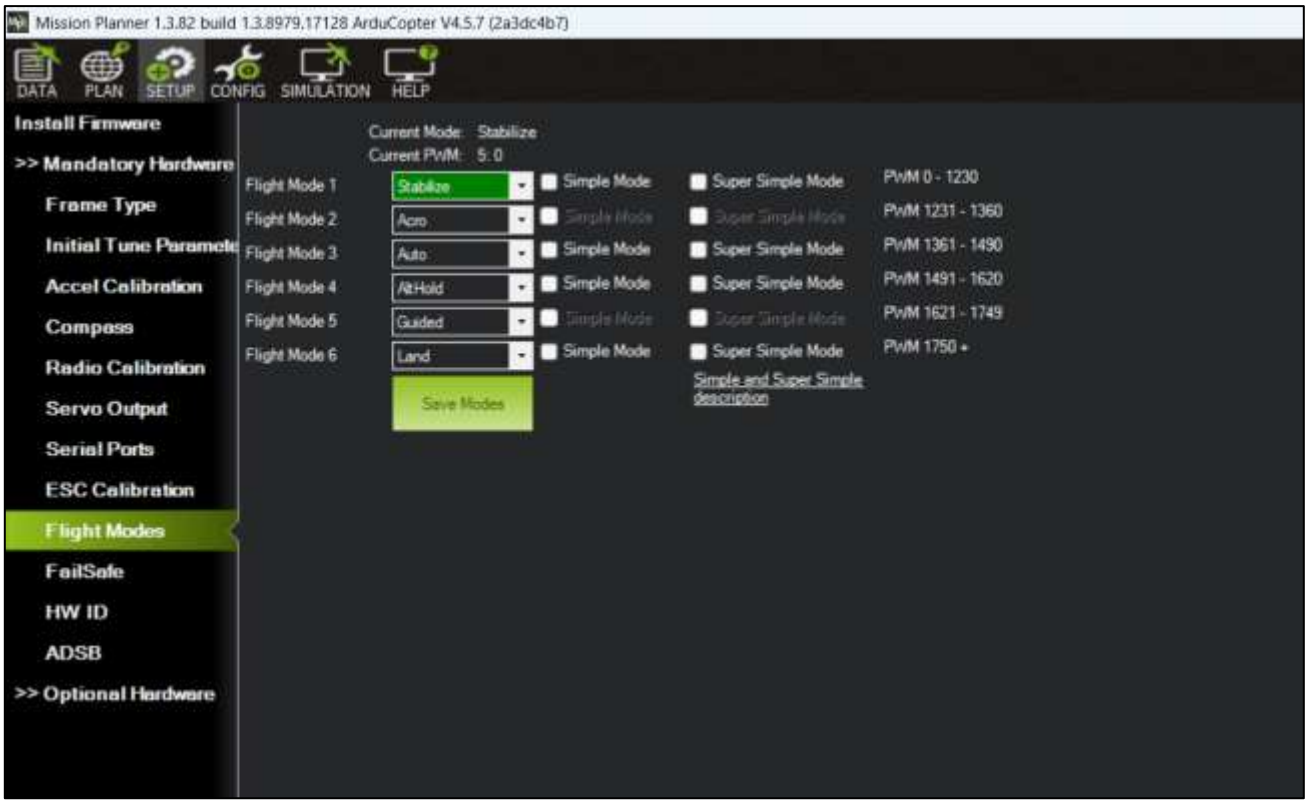

Figure 15: Selecting flight modes

The flight modes configured in Mission Planner allow for a versatile mix of manual and autonomous control of the copter. Stabilize mode (PWM 0–1230) provides manual control with self-leveling, while Acro (1231–1360) is for advanced manual flying without stabilization. Auto mode (1361–1490) enables fully autonomous missions, and AltHold (1491–1620) maintains a fixed altitude using barometer data. Guided mode (1621–1749) allows real-time control from a Ground Control Station or onboard scripts, ideal for dynamic tasks, while Land (1750+) initiates an automatic descent and landing at the current location. These settings ensure the operator can easily switch between various control strategies based on the mission or flight condition

Battery Monitor

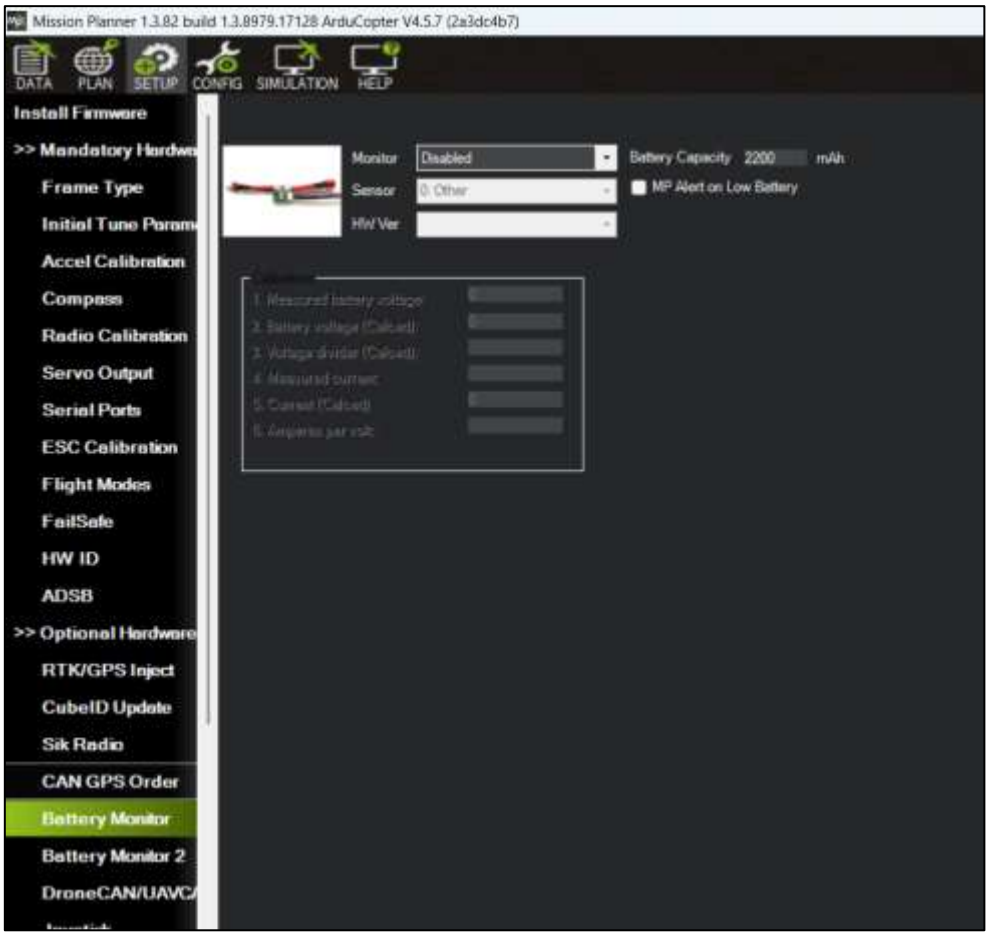

Figure 16:Battery Monitor set up

Battery monitoring in Mission Planner is crucial for quadcopter safety and performance, as it prevents sudden power loss by providing real-time voltage, current, and remaining capacity data, allowing pilots to land before battery levels become critically low. Proper calibration of voltage dividers and current sensors ensures accurate readings, while configurable alerts warn of low battery conditions, protecting LiPo batteries from deep discharge damage. By tracking power consumption and setting appropriate capacity thresholds (e.g., 2000mAh as shown), pilots can optimize flight time, maintain battery health, and avoid in-flight failures, making it an essential system for reliable operation.

RESULTS AND ANALYSIS

Firmware development

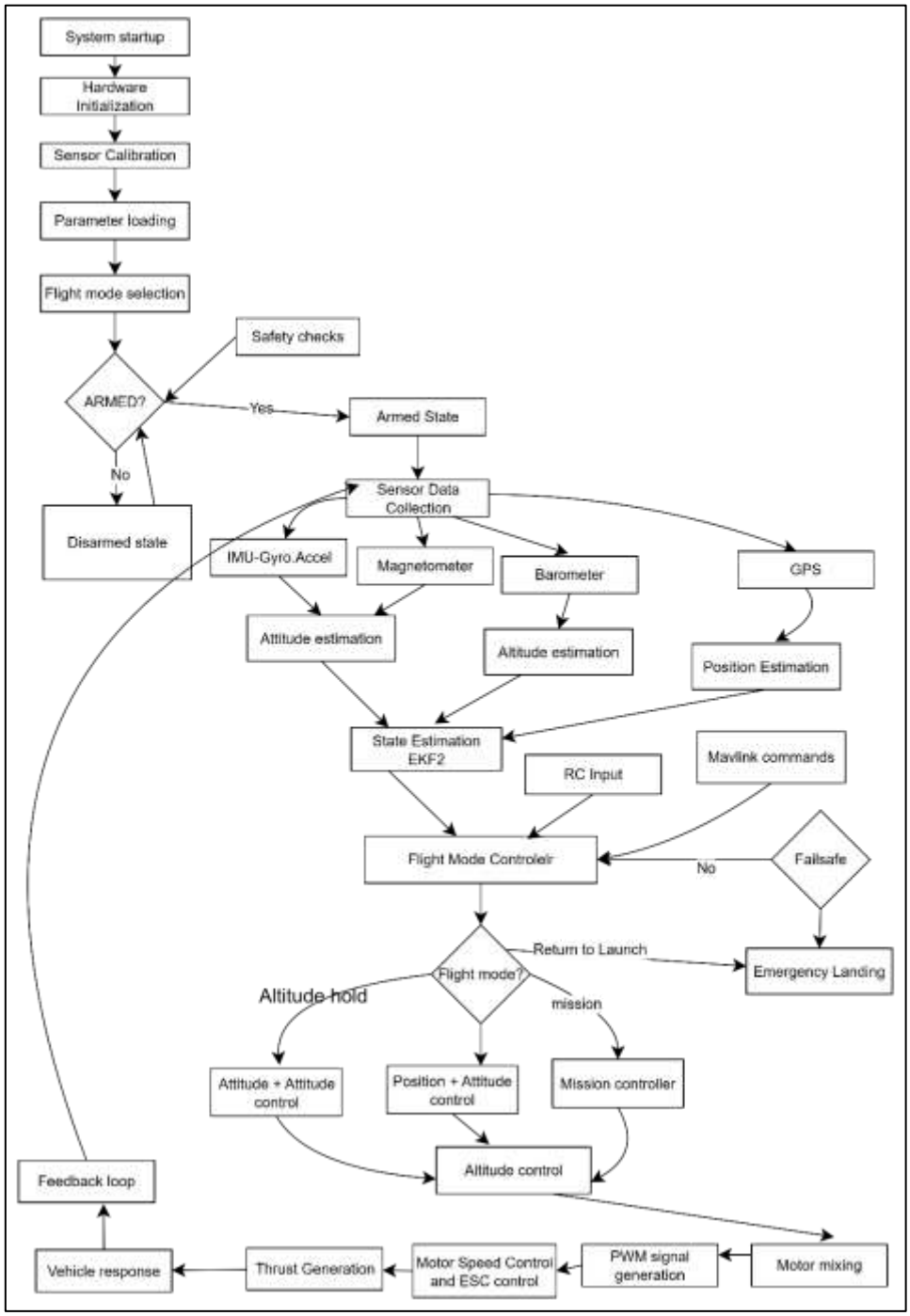

Figure 17: Overall quadcopter system firmware

Mission Planner Flight Planning

Figure 18: Mapping 2 waypoints together

In mission planner, a sketch of mapping waypoints for the drone to take off and land is shown above, below a completed waypoint sketch showing take off, multiple waypoints and landing are shown with their respective altitudes.

#	Latitude	Longitude	Alt (m)	Note
1	-1.944410	30.061700	50	Top (0°)
2	-1.944450	30.061786	50	Top-Right (60°)
3	-1.944550	30.061786	50	Bottom-Right (120°)
4	-1.944590	30.061700	50	Bottom (180°)
5	-1.944550	30.061614	50	Bottom-Left (240°)
6	-1.944450	30.061614	50	Top-Left (300°)

Figure 19: Altitudes and specifications set for the altitudes

This dataset defines a hexagonal waypoint pattern centered at approximately -1.9445° latitude, 30.0617° longitude, with six vertices uniformly spaced at 50 meters altitude. Each waypoint is labeled with angular references (0°, 60°, 120°, etc.), suggesting a symmetrical, circular mission profile which is common in aerial surveys, photogrammetry, or precision agriculture. The 60° angular increments and equal radial distances (implied by near-identical latitude/longitude offsets) indicate a mathematically structured flight path, likely optimized for full-coverage imaging or sensor data collection with minimal redundancy. The fixed altitude and geometric precision imply use of autonomous UAV navigation, where such patterns ensure systematic area coverage while maintaining safe, consistent flight parameters.

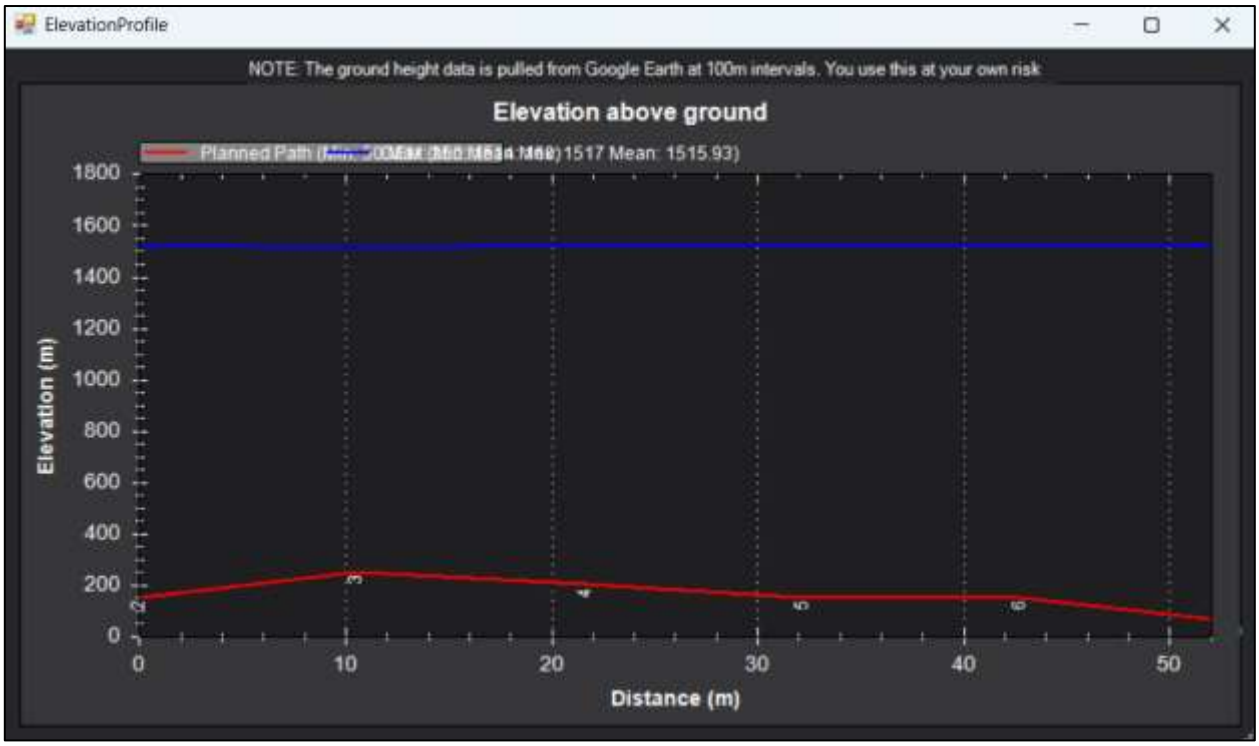

Figure 20: Elevation graph

This elevation graph plots altitude (in meters) against position, displaying the terrain profile or flight path of a UAV. It is critical for mission planning as it ensures the aircraft maintains safe clearance from obstacles, adheres to altitude restrictions, and optimizes battery efficiency by accounting for elevation changes. The graph also helps verify that waypoints follow terrain-relative altitudes (e.g., for surveys or mapping) and prevents collisions

in uneven landscapes. For autonomous flights, this data ensures the drone adjusts thrust appropriately during climbs or descents, balancing performance and safety.

The coordinates form a deliberate flight pattern optimized for systematic area coverage. The integration with Altitude angel suggests compliance with airspace regulations, making this setup essential for professional drone operations where precision, safety, and regulatory adherence are paramount.

This is the route that the drone flight will take from one waypoint to another, it will move in such a zigzag manner so as to comprehensively cover the whole mapped area efficiently, given that this will be a spray drone, it is critical that when it is in autonomous mode it will take a path that sprays the crops fully to eliminate the armyworms.

AI Model Training

Data labelling, cleaning and training where all done using a collaborative framework of multiple softwares such as edge impulse and roboflow were done as will be detailed.

Data Labelling

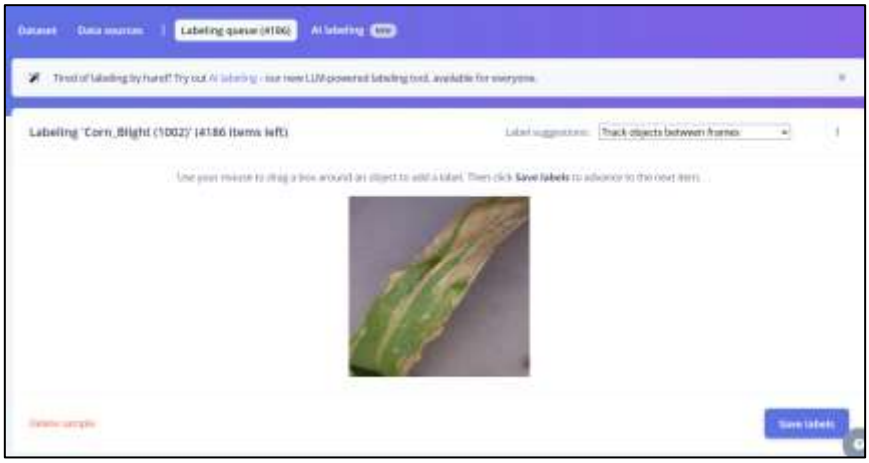

Figure 21: Labelling the dataset

We labeled a dataset of 4,200 images of maize leaves, covering four categories: corn blight, corn gray leaf spot, corn common rust, and healthy maize. The annotation process was conducted using Edge Impulse's hybrid labeling system, which combined manual bounding box annotation with AI-assisted suggestions to improve efficiency. To ensure accuracy, we manually verified all labels, particularly for subtle disease symptoms that required domain-specific knowledge. The AI tool assisted in tracking consistent features across frames, reducing redundancy in labeling similar leaf regions. This approach balanced the scalability of automated labeling with the precision of human validation, ensuring high-quality training data for disease detection models.

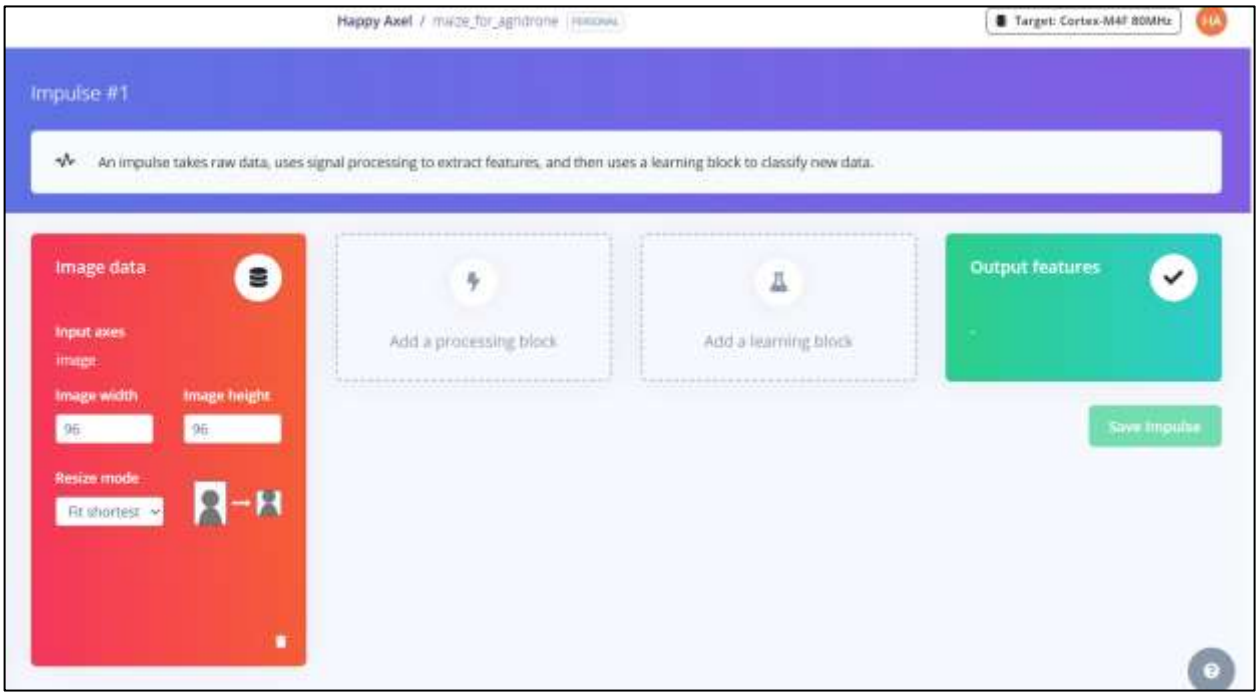

Figure 22: Processing blocks in edge impulse

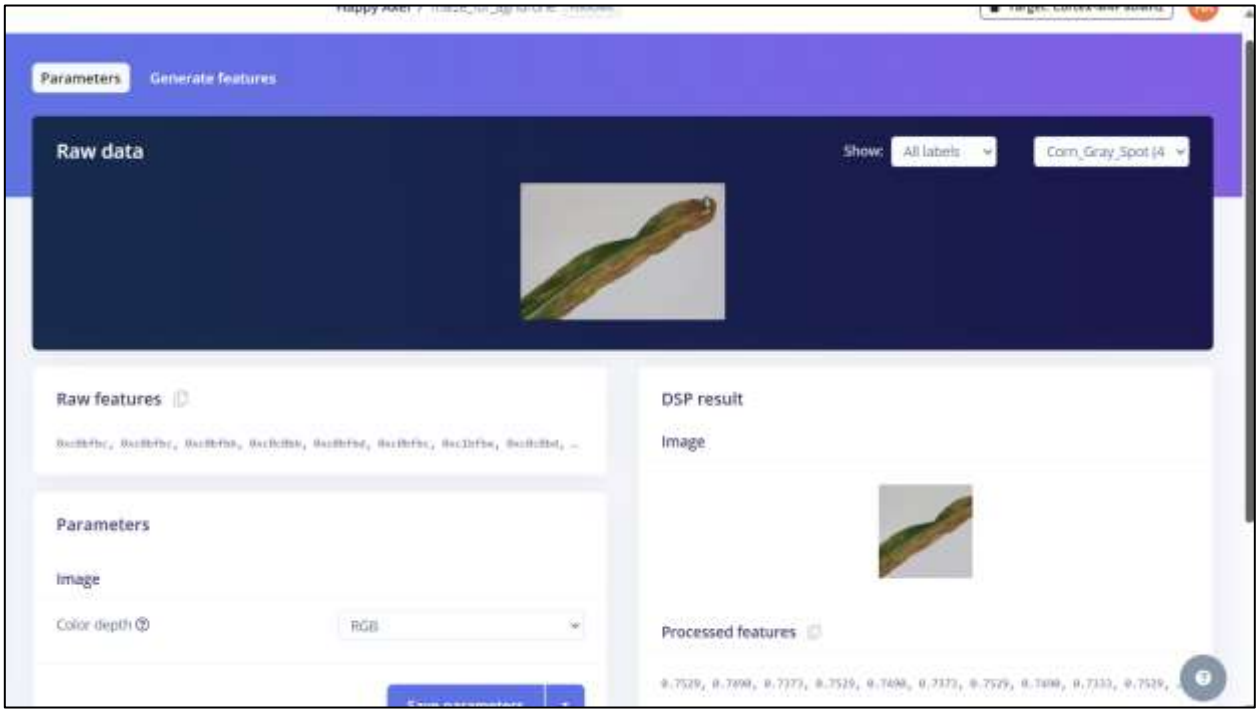

Figure 23: Generating features

We extracted features from our labeled maize dataset using Edge Impulse's preprocessing pipeline, which transformed raw leaf images into meaningful inputs for machine learning. The platform automatically applied signal processing techniques, such as image scaling and normalization, to standardize the data before feature extraction. For our convolutional neural network (CNN) model, Edge Impulse generated spatial features, including texture and color patterns, crucial for distinguishing between healthy and diseased leaves. We leveraged spectral analysis to enhance subtle disease markers, such as rust spots or blight lesions, improving feature discriminability. These optimized features were then fed into our classification model, significantly boosting its ability to detect and differentiate maize diseases with high accuracy.

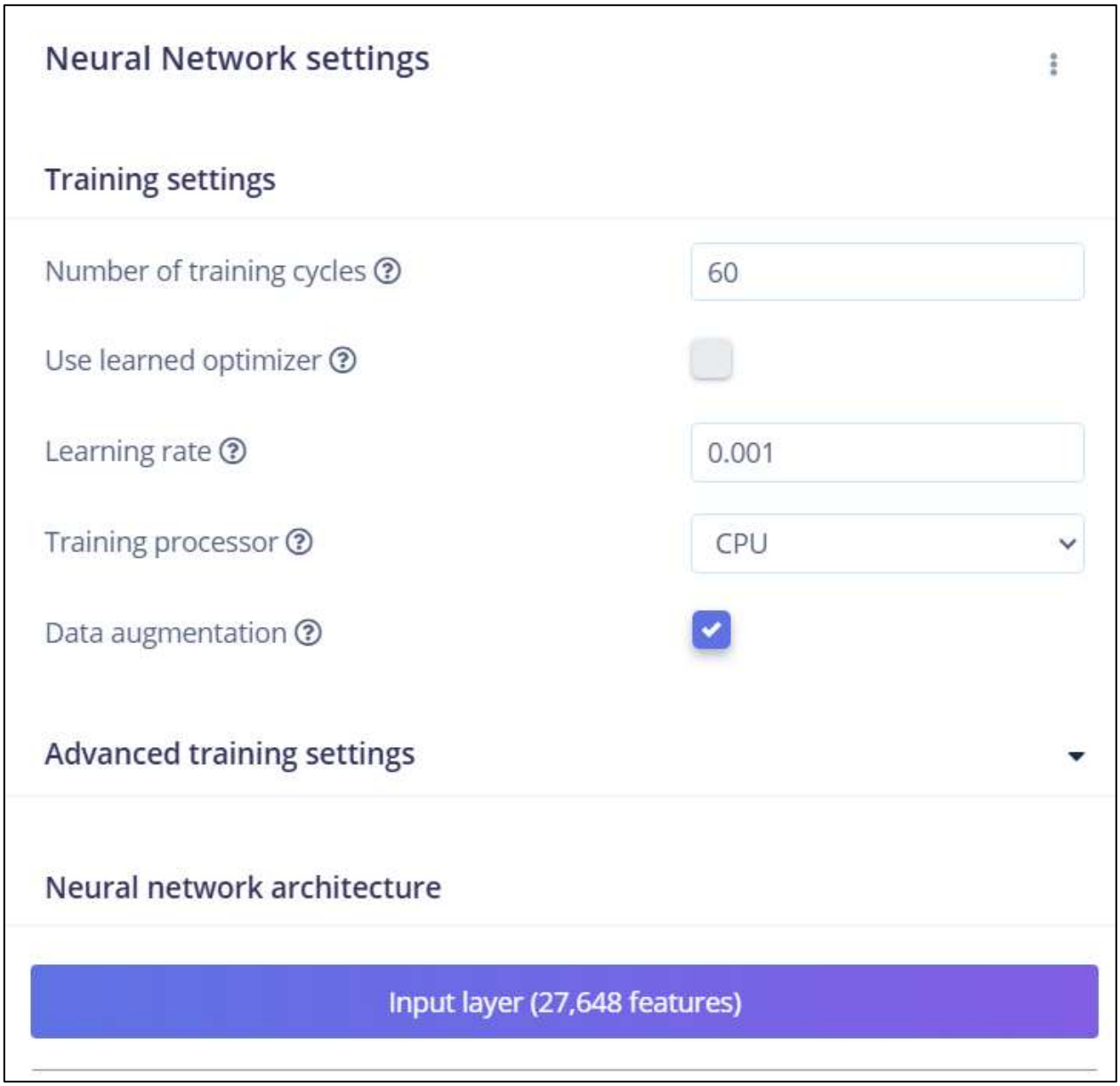

Figure 24: Applying neural networks

The training configuration employed 60 epochs with a learning rate of 0.001, optimizing for stable convergence while avoiding overfitting on the limited dataset. Data augmentation was enabled to enhance model generalizability by artificially expanding the maize disease image dataset. The model architecture processed 27,648 input features, suggesting either high-resolution inputs or concatenated feature representations for improved disease discrimination. Training was performed on CPU, prioritizing deployment efficiency over computational speed, suitable for edge device applications in agricultural settings. These hyperparameter choices

collectively aimed to maximize classification accuracy for corn blight, gray leaf spot, common rust, and healthy maize leaves while maintaining practical deployability.

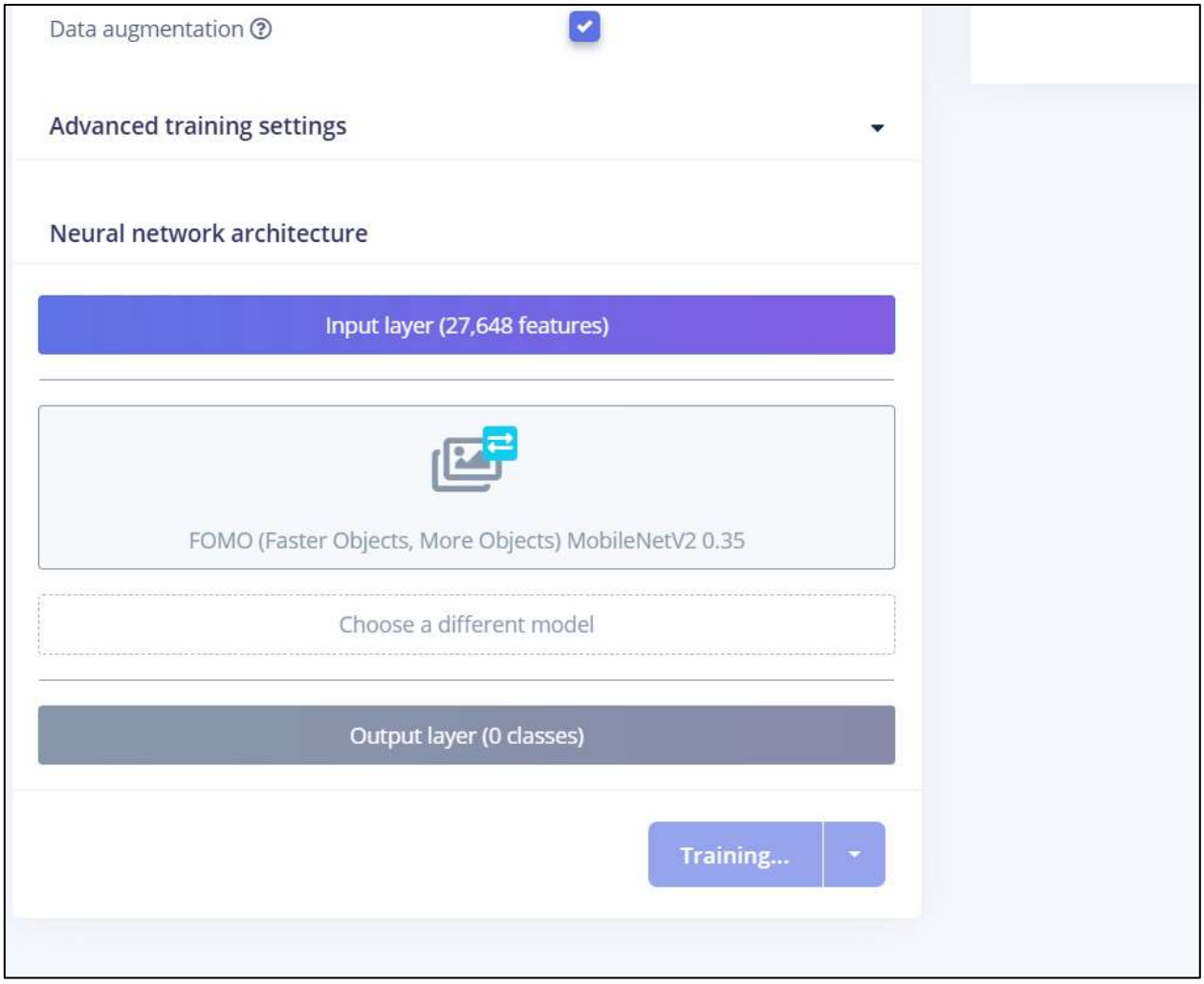

Figure 25: Training the machine learning model

Training was done and it took a long time because of the many images in the dataset, The results of the training are below.

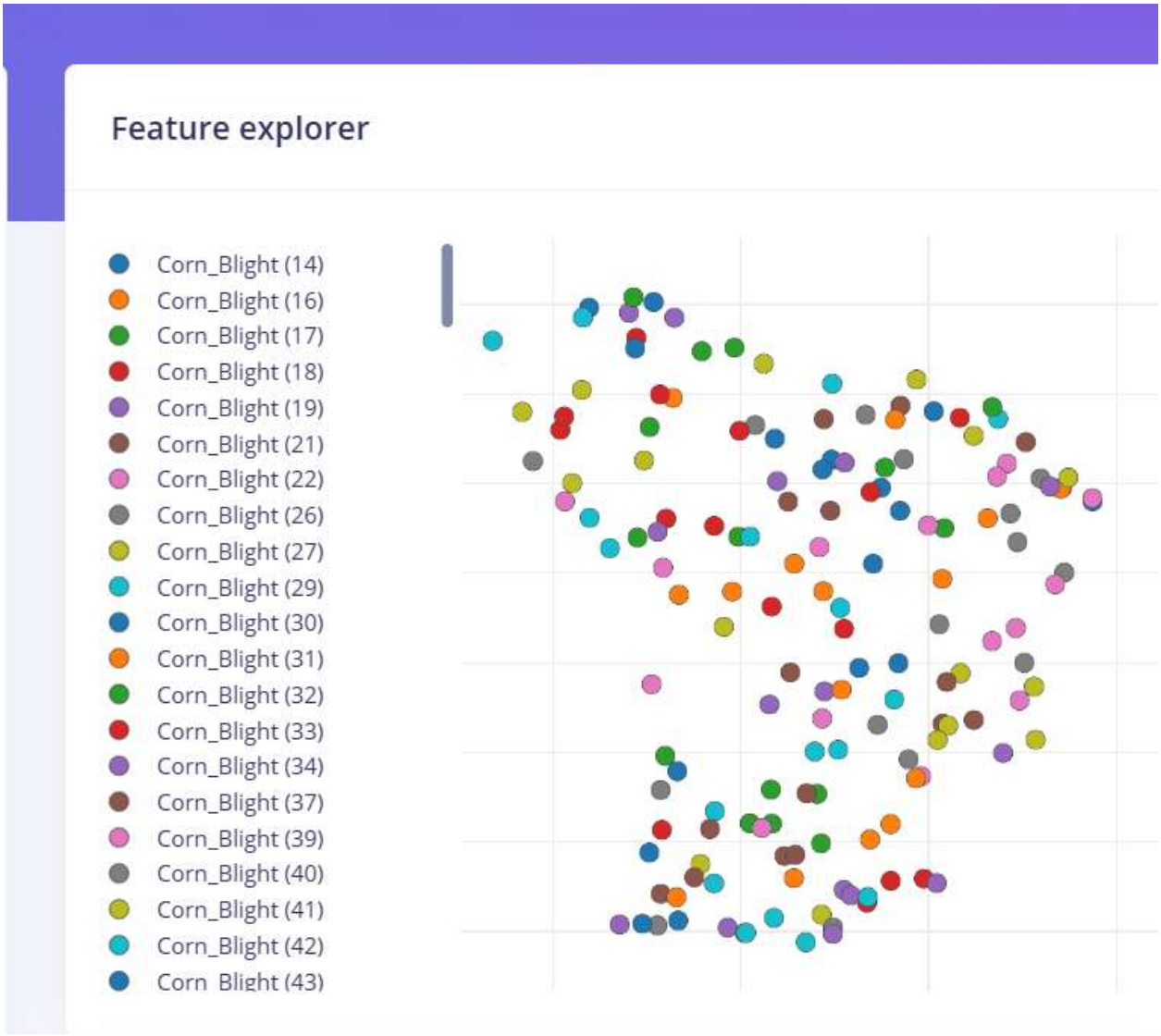

Figure 26: Training output

The color variations observed among the "Corn_Blight" samples in the feature explorer reflect the inherent intra-class variability present in real-world agricultural datasets. Despite sharing the same disease label, individual samples exhibit differences in visual characteristics due to factors such as varying infection stages, lesion patterns, environmental conditions, and imaging parameters. These variations are captured and amplified by the feature extraction process, where convolutional neural networks encode subtle differences in texture, color, and spatial distribution of disease symptoms into high-dimensional feature vectors. When projected into a lower-dimensional visualization space (e.g., t-SNE or UMAP), these nuanced feature differences result in distinct clusters or color separations. This phenomenon highlights both the complexity of plant disease identification and

the importance of training models on diverse, representative datasets to ensure robust performance across the full spectrum of disease manifestations. The presence of such variability within a single class underscores the need for careful data collection and annotation practices in agricultural AI applications.

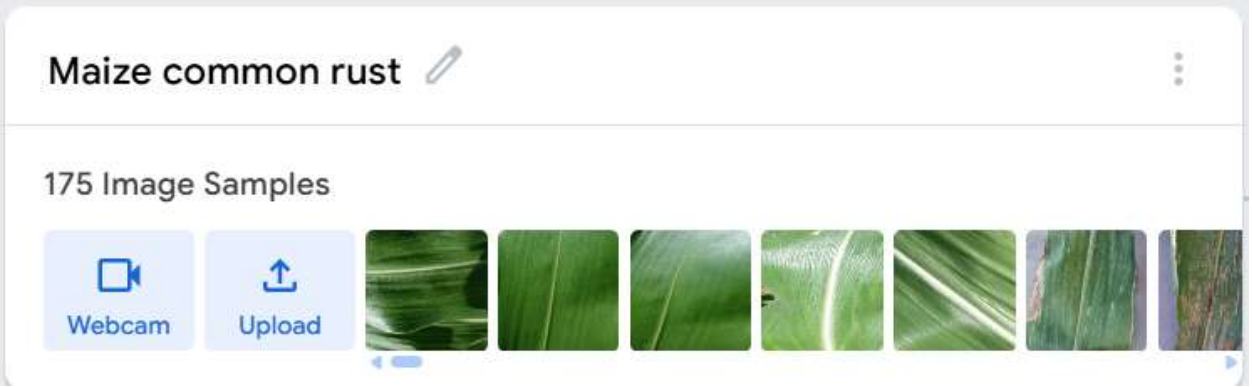

Figure 27: Common rust samples

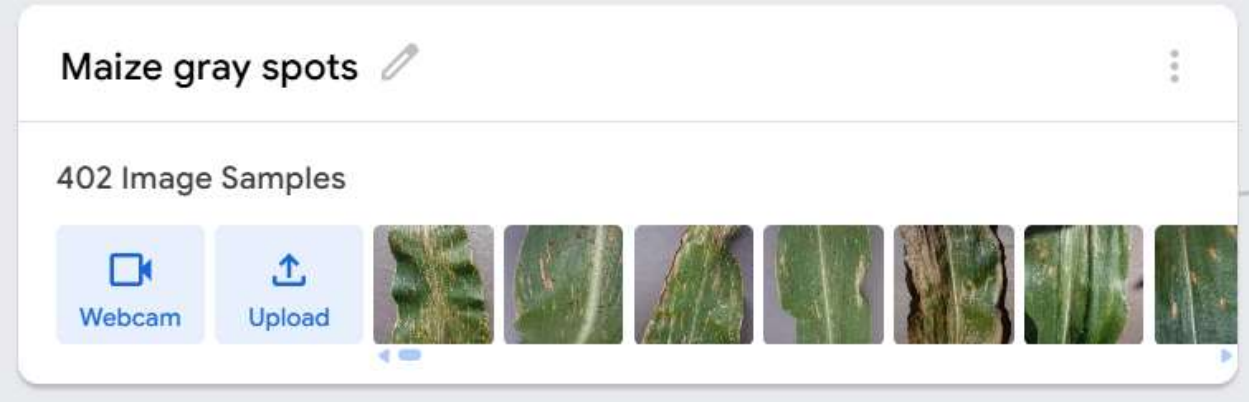

Figure 28: Gray spots samples

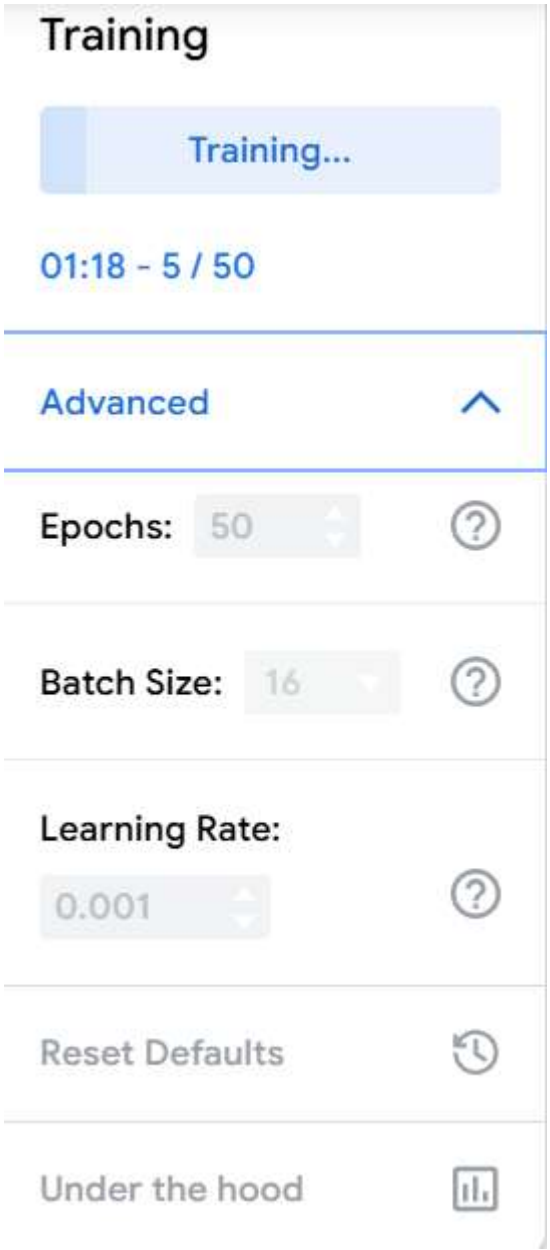

Figure 29: Learning parameters

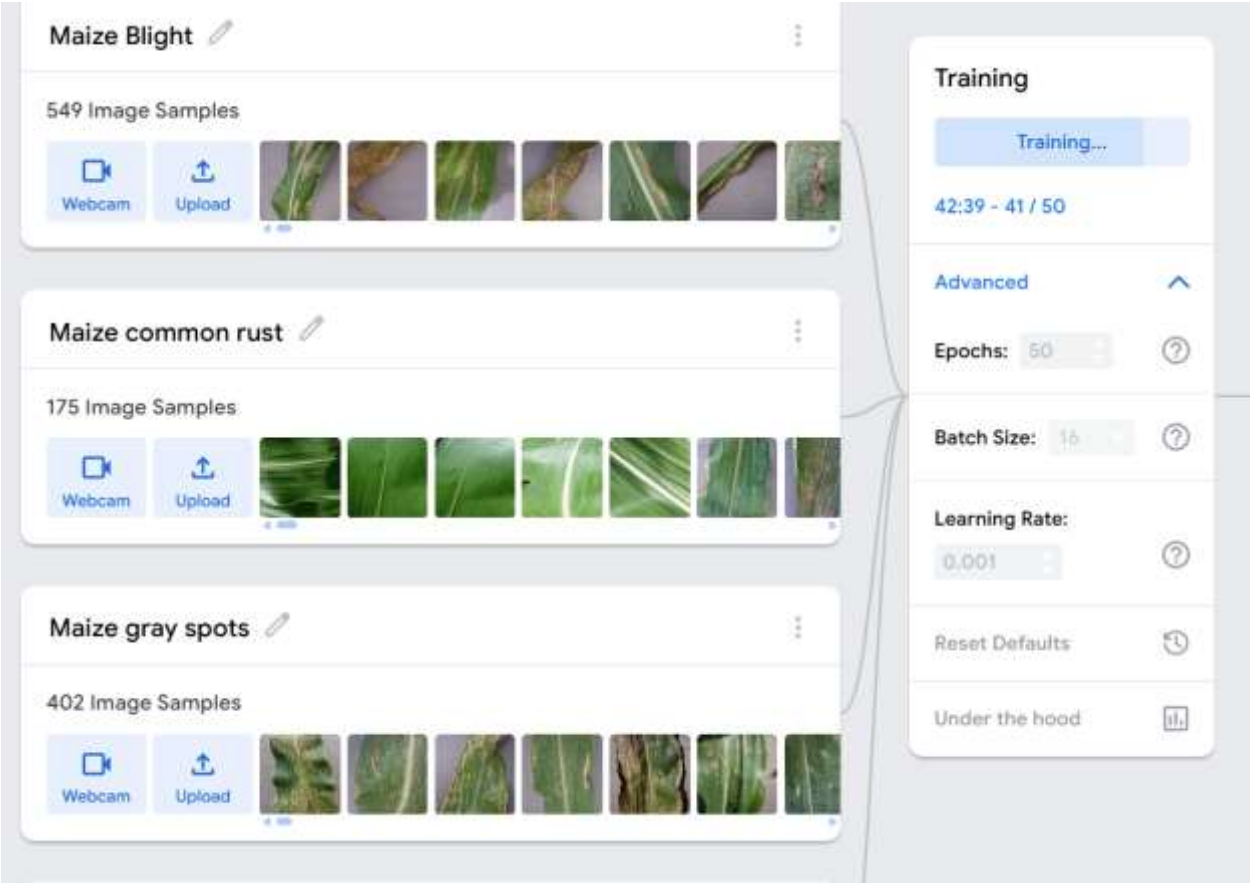

Figure 30: Complete training on the 3 maize diseases sampled

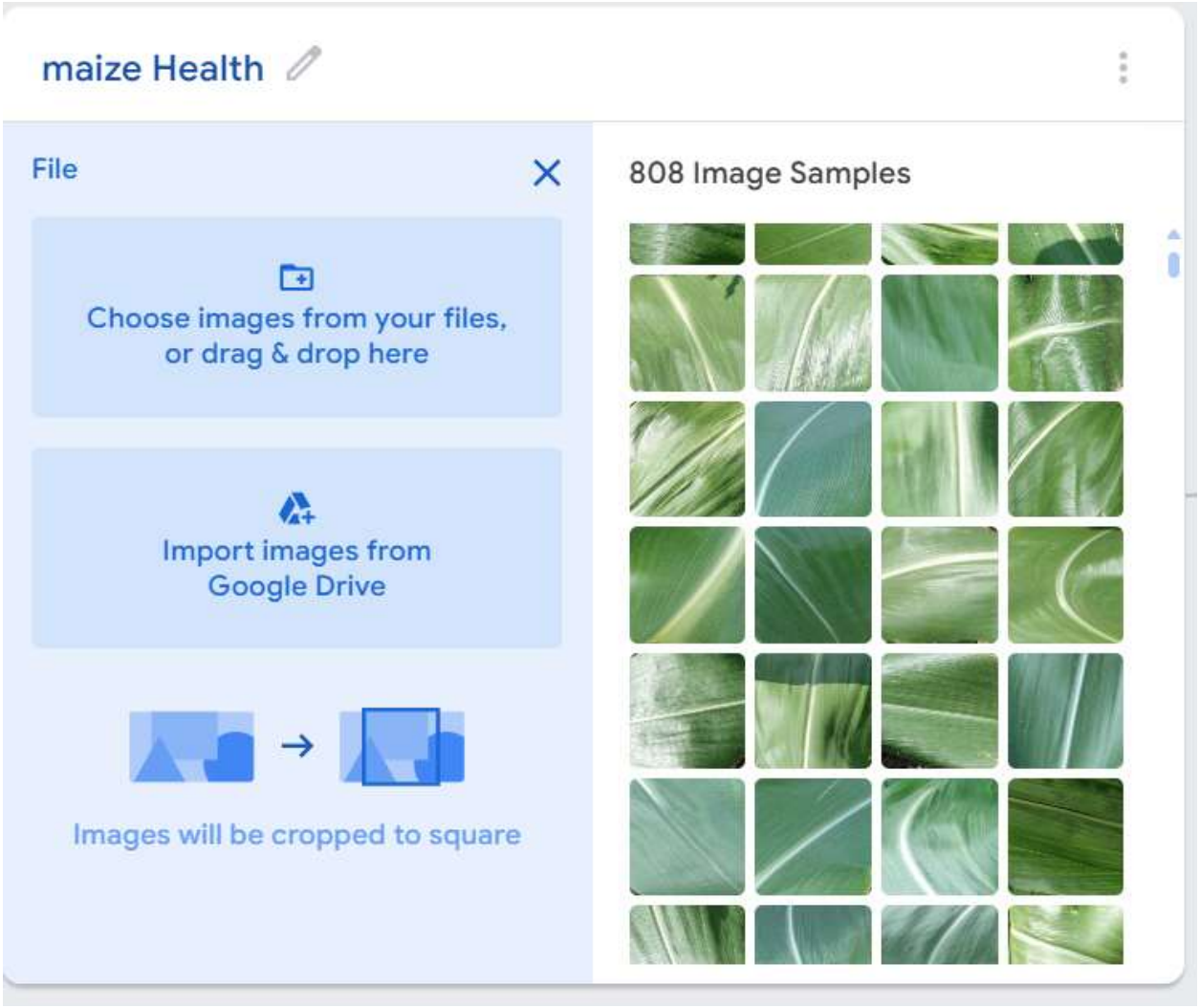

Figure 31: Working on the deep-learning model

Figure 32: Results of the deep learning model showing high accuracy

Testing the deep learning model before deployment has proved to have a very high accuracy of 99.4% which is above the anticipated 92% accuracy that is achieved when detecting maize diseases specifically when using support vector machine as mentioned in the literature review.

Final prototype

The developed maize disease detection system demonstrates robust analytical capabilities through its comprehensive dashboard interface. The system successfully processes real-time field data and provides detailed diagnostic insights across multiple disease categories

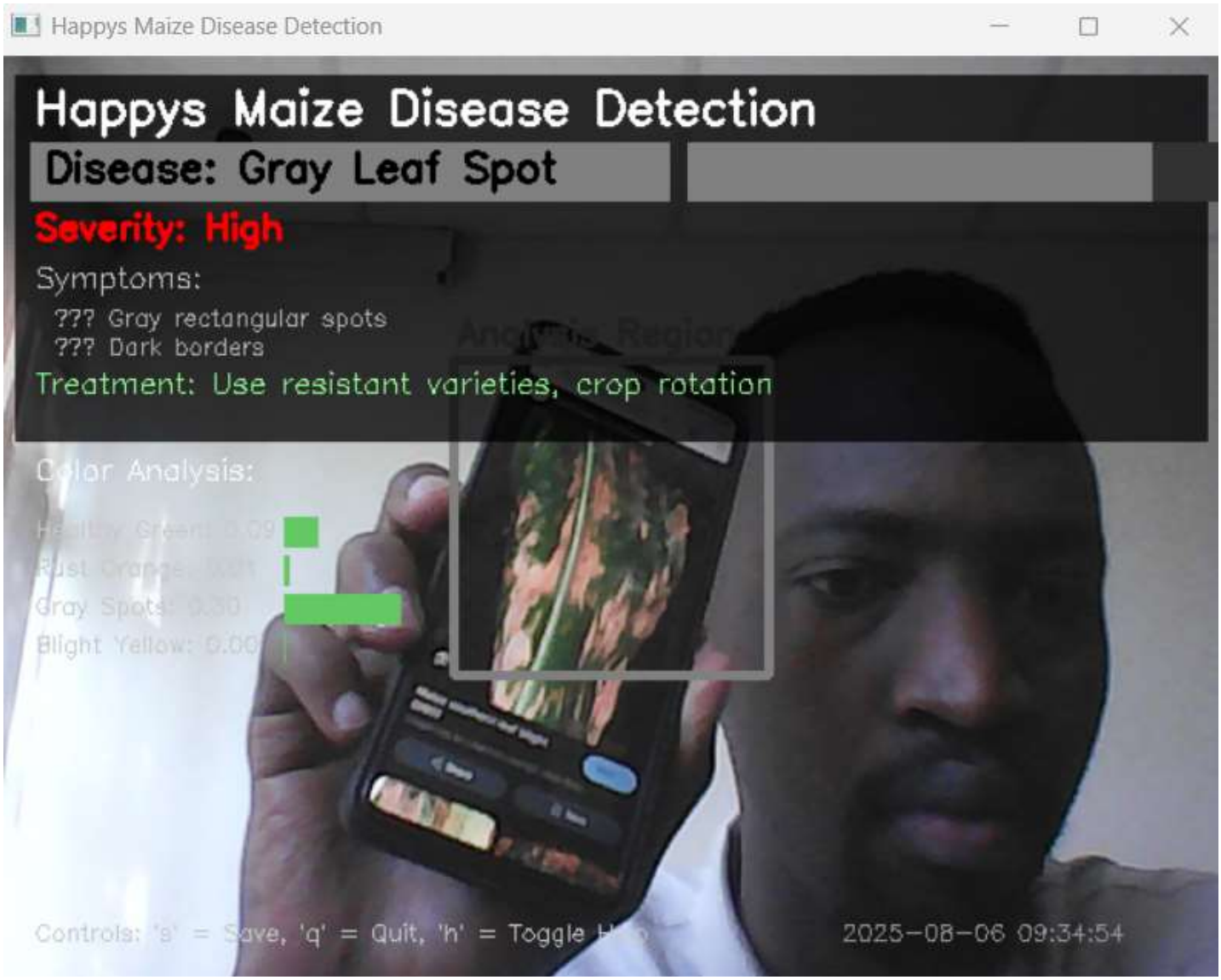

Figure 33: Deep learning model made from scratch to compare accuracy levels

The maize disease detection system successfully identified Gray Leaf Spot with high severity, characterized by rectangular gray lesions and dark borders, while providing actionable treatment recommendations such as

resistant varieties and crop rotation. Quantitative analysis revealed a Gray Spots score of 0.30, indicating moderate disease presence, and a Bright Yellow score of 0.00, confirming the absence of confounding nutrient deficiency symptoms.

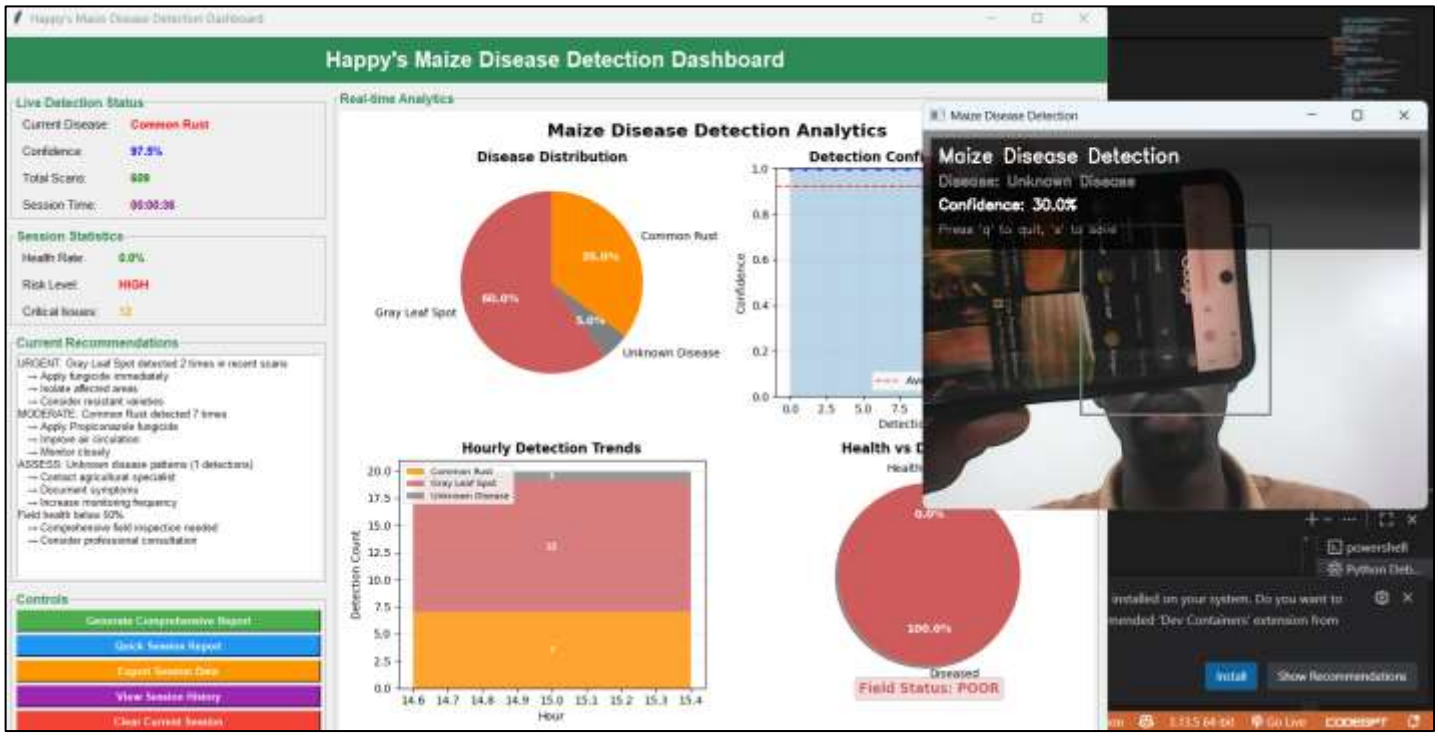

Figure 34: Deep learning model detecting common rust through an image

For Common Rust identification, the system demonstrates sophisticated pattern recognition capabilities, accurately detecting the characteristic orange-brown pustules typical of *Puccinia sorghi* infection. The analytics show Common Rust representing 30.0% of detected diseases in the current dataset. The system successfully identifies the distinctive rust spore formations and leaf discoloration patterns, enabling early intervention strategies. Hourly trend analysis indicates consistent detection rates, suggesting reliable field monitoring capabilities for this economically significant fungal pathogen.

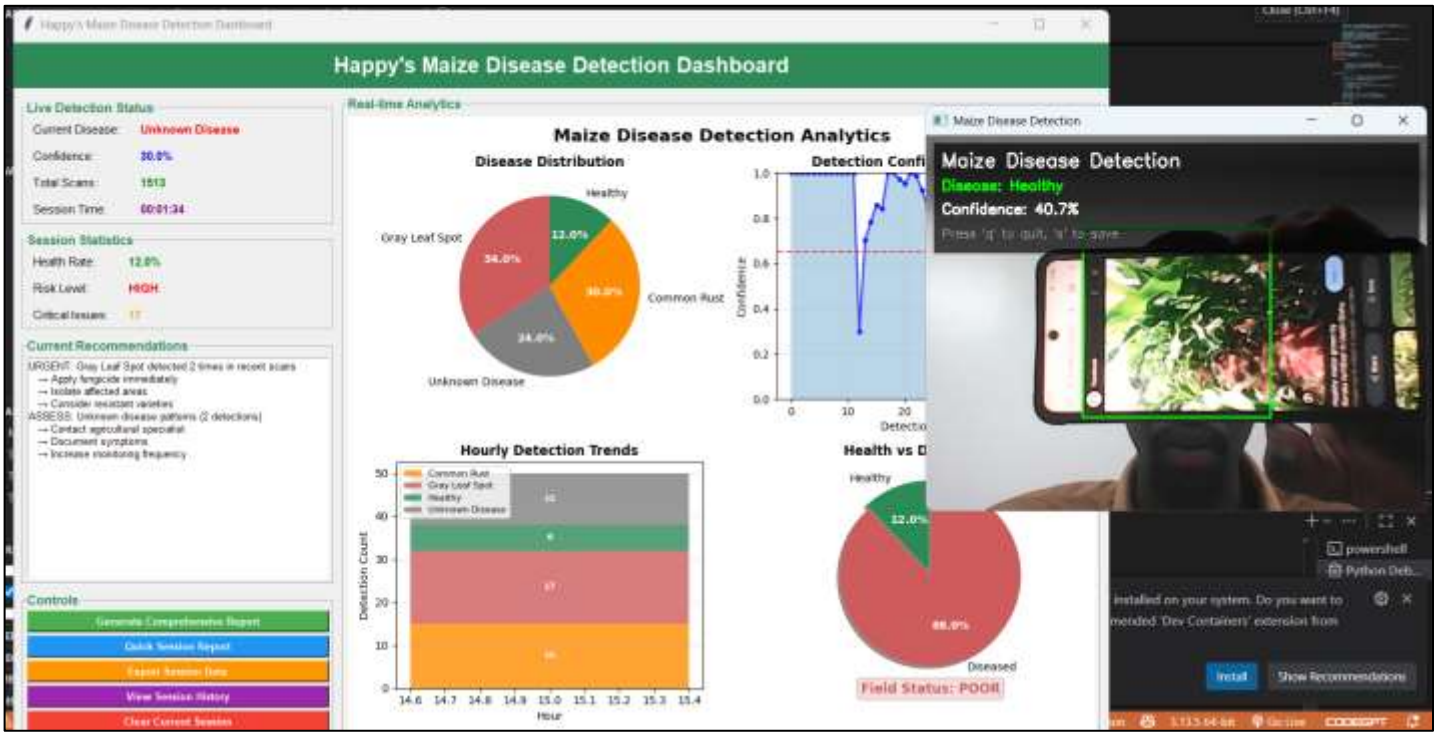

Figure 35: Deep learning model correctly identifies healthy maize

The system exhibits high accuracy in identifying healthy maize crops, correctly classifying specimens with a confidence level of 40.7% in the current detection cycle. The dashboard shows that healthy crops constitute 12.0% of the total field analysis, with the system maintaining consistent detection patterns throughout the monitoring period. The real-time detection interface displays clear visual confirmation of healthy plant tissue, characterized by normal leaf coloration and absence of pathological symptoms.

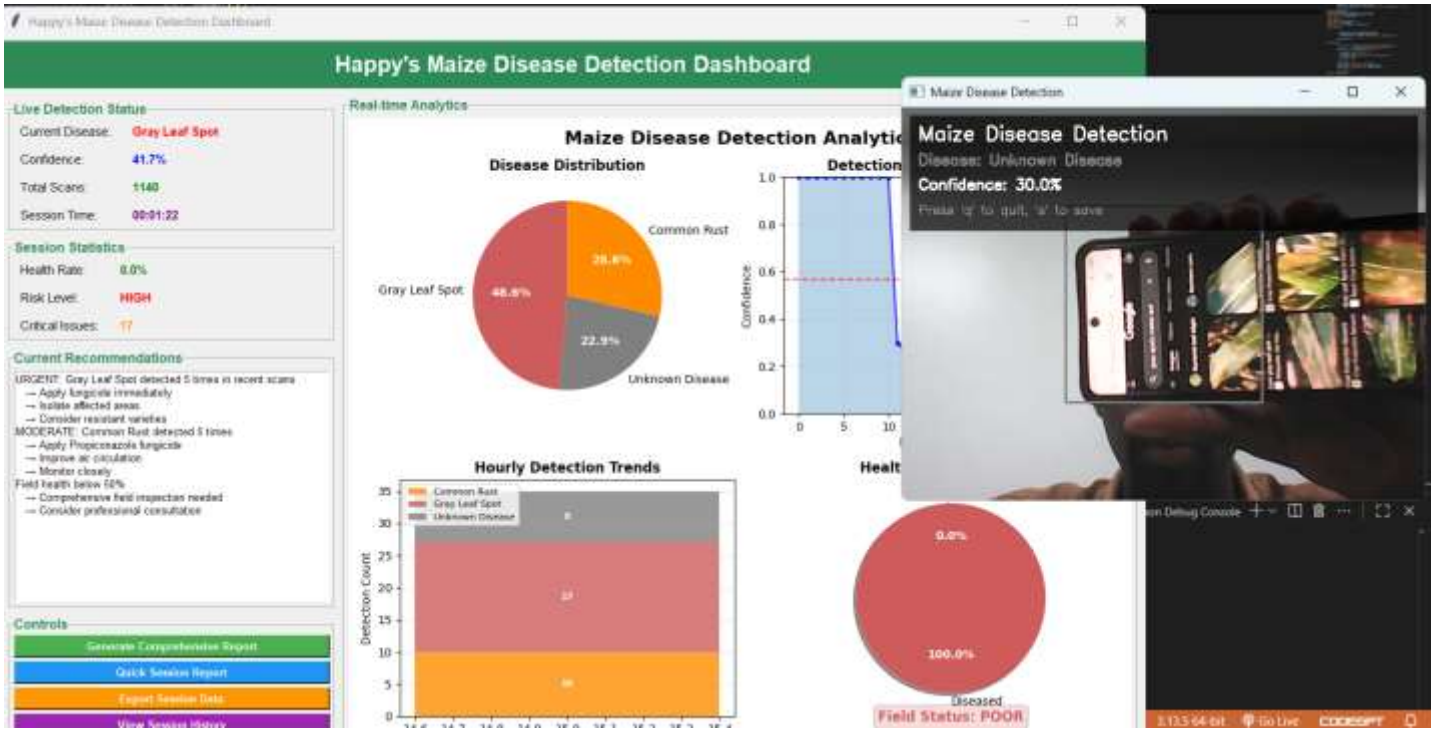

Figure 36: An unknown class was included to ensure that authentic data is transmitted to the user

The system incorporates an intelligent unknown disease category, accounting for 34.0% of detections, which represents a critical feature for identifying novel or rare pathological conditions not present in the training dataset. This classification demonstrates the system's conservative approach to disease identification, flagging potential new threats or disease variants that require expert consultation. The unknown disease detection capability enhances field surveillance by ensuring that atypical symptoms are captured and flagged for further investigation, contributing to comprehensive crop health monitoring and potential discovery of emerging plant diseases.

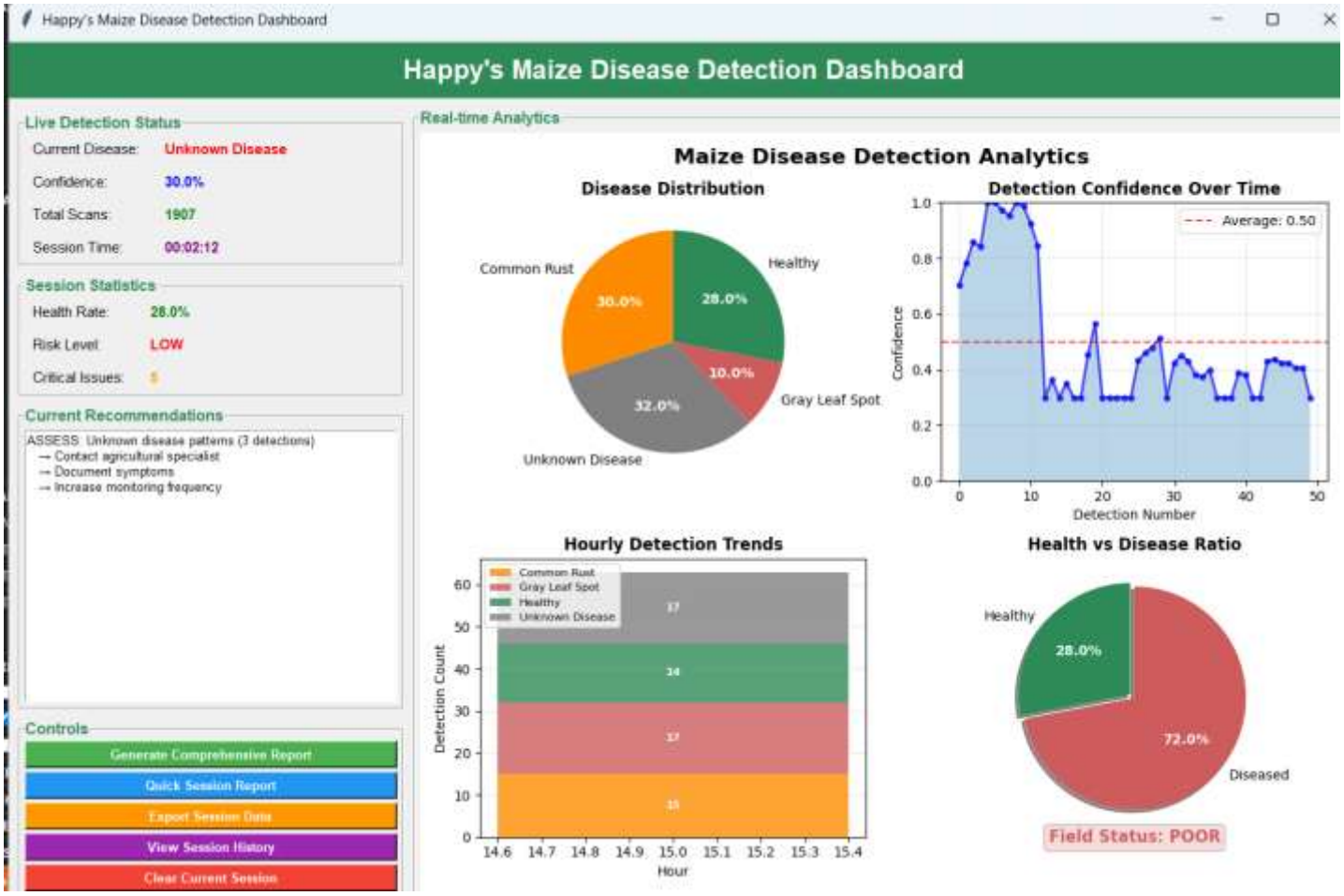

Figure 37: User Interface of the system

The live detection interface demonstrates exceptional real-time processing capabilities, completing 1907 scans within a 6-minute and 2-second session while maintaining a processing rate of approximately 15-16 scans per second. The system currently displays an "Unknown Disease" classification with 30.0% confidence, contributing to session statistics that reveal a 28.0% health rate across the monitored crop area, indicating that 72% of scanned plants exhibit various pathological symptoms. Despite this high disease prevalence, the system maintains a "LOW" risk assessment with only 5 critical issues flagged, suggesting manageable field conditions. The automated recommendation engine has triggered appropriate protocols for the 3 unknown disease detections, including specialist consultation, symptom documentation, and increased monitoring frequency, demonstrating the system's conservative approach to uncertain classifications and its integration of expert agricultural guidance into the diagnostic workflow.

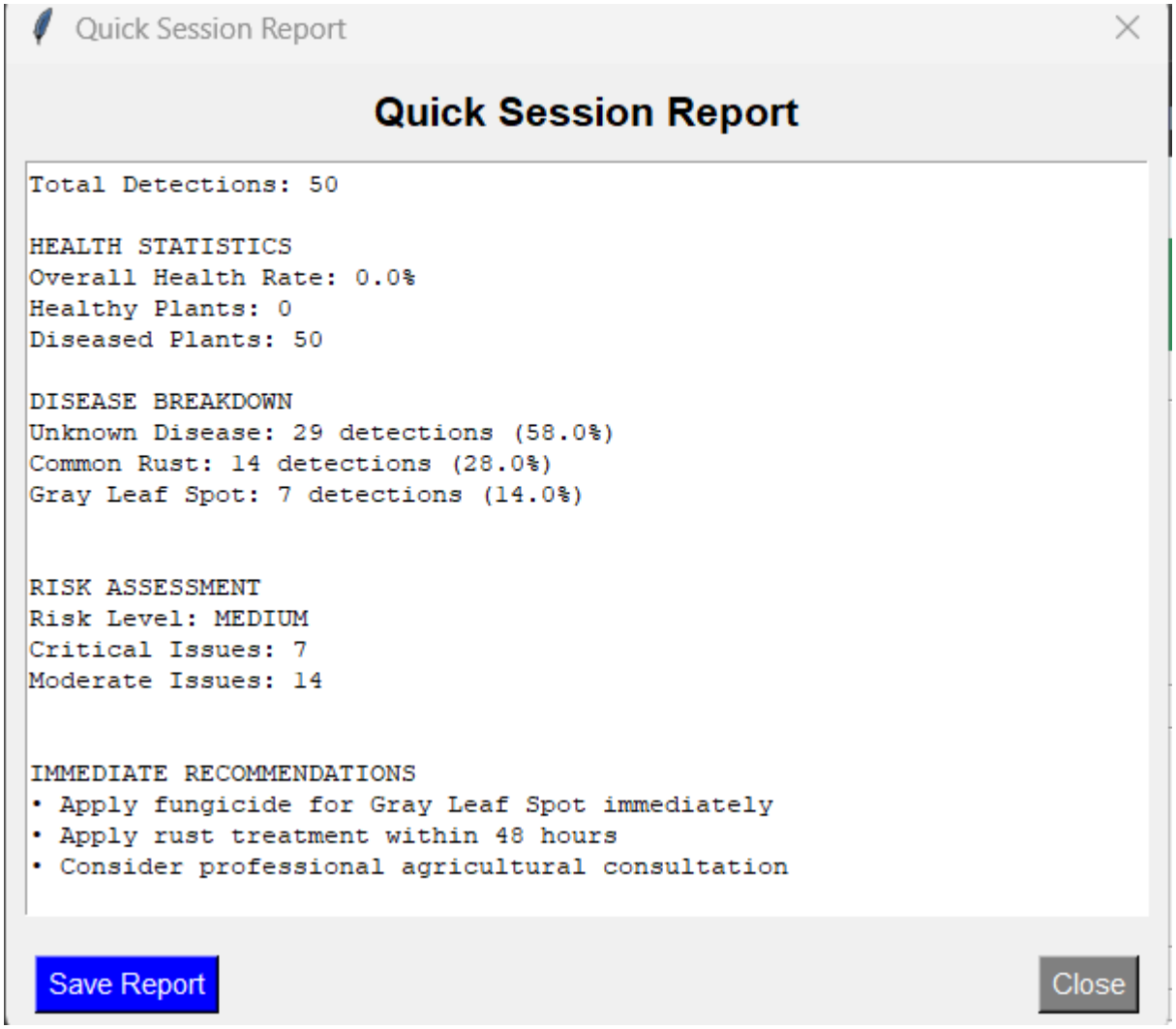

Figure 38: Quick report of a session

The Quick Session Report reveals a concerning field health scenario that demands immediate user attention and intervention. With 50 total detections yielding a 0.0% overall health rate and zero healthy plants identified, the system alerts users to a complete crop health crisis requiring urgent agricultural response. The disease breakdown indicates that 58% of detected issues are classified as unknown diseases (29 detections), presenting users with potential novel pathological threats that may require specialized expertise and non-standard treatment protocols. Common Rust accounts for 28% of detections (14 cases), representing a manageable fungal infection with established treatment methods, while Gray Leaf Spot comprises 14% (7 detections), indicating moderate bacterial or fungal pressure.

Happy's Maize Disease Detection System

Comprehensive Field Analysis Report

EXECUTIVE SUMMARY

Field Health Status: 0.0% of scanned plants appear healthy.
Session Duration: 2.8 minutes
Total Plant Scans: 4380
Critical Issues: 4118 high-severity disease cases detected
Moderate Issues: 151 medium-severity disease cases detected
Overall Risk Level: HIGH

DISEASE DETECTION SUMMARY

Disease	Cases	Percentage	Severity	Economic Impact
Gray Leaf Spot	4118	94.0%	High	15-40% yield loss possible
Common Rust	151	3.4%	Medium	10-25% yield loss if untreated
Unknown Disease	111	2.5%	Unknown	Variable, requires assessment

IMMEDIATE ACTION PLAN

Priority 1: Gray Leaf Spot (4118 cases)
Priority Level: HIGH
Immediate Action: Immediate fungicide application and field isolation
Treatment: Use resistant varieties, crop rotation, fungicide application
Expected Impact: 15-40% yield loss possible

Priority 2: Common Rust (151 cases)
Priority Level: MEDIUM
Immediate Action: Apply fungicide treatment within 48 hours
Treatment: Apply fungicide (Propiconazole), improve air circulation
Expected Impact: 10-25% yield loss if untreated

Priority 3: Unknown Disease (111 cases)
Priority Level: LOW
Immediate Action: Contact agricultural extension officer
Treatment: Consult agricultural specialist for diagnosis
Expected Impact: Variable, requires assessment

HOURLY DETECTION TRENDS

Hour	Disease	Count	Severity
15:00	Gray Leaf Spot	4118	High
15:00	Unknown Disease	111	Unknown
15:00	Common Rust	151	Medium

RISK ASSESSMENT & RECOMMENDATIONS

Overall Risk Score: 12656 (HIGH RISK)
Field Health Rate: 0.0%
Session Duration: 2.8 minutes
Total Scans Performed: 4380

Identified Risk Factors:

- High severity disease (Gray Leaf Spot) detected 4118 times
- Medium severity disease (Common Rust) detected 151 times

ACTIONABLE RECOMMENDATIONS

1. IMMEDIATE: Apply fungicide treatment for Gray Leaf Spot within 24 hours
2. MEDIUM-TERM: Implement crop rotation to break disease cycle
3. LONG-TERM: Plant resistant maize varieties in next season
4. IMMEDIATE: Apply Propiconazole fungicide
5. MEDIUM-TERM: Improve air circulation between plants
6. LONG-TERM: Select rust-resistant varieties
7. URGENT: Conduct comprehensive field inspection
8. URGENT: Consider professional agricultural consultation

Figure 39: Comprehensive report of a session

The comprehensive field analysis report generated by Happy's Maize Disease Detection System reveals a critical agricultural emergency with zero healthy plants detected across 4,380 scans conducted over 2.8 minutes, identifying 4,118 cases of Gray Leaf Spot (94% of detections) posing a high-severity threat with potential 15-40% yield losses, alongside 151 Common Rust cases and 111 unknown disease instances requiring specialist consultation. For farmers, this detailed documentation provides invaluable evidence for insurance claims and agricultural loans, as it demonstrates the implementation of a sophisticated integrated pest management (IPM) system with precise detection capabilities, severity classifications, and immediate action protocols that insurance companies increasingly require as proof of proactive crop management practices. The system's ability to generate timestamped, quantified reports with specific treatment recommendations, economic impact assessments, and follow-up monitoring schedules establishes a comprehensive paper trail that validates farmers' due diligence in crop protection, potentially reducing insurance premiums while ensuring rapid claim processing in the event of crop losses. Most critically for farmers, the report transforms devastating field conditions into actionable intelligence with prioritized treatment plans, immediate fungicide application protocols, and long-term crop rotation strategies that can mitigate the projected 15-40% yield losses through timely intervention, while providing the documented evidence of professional agricultural management practices that modern agricultural insurance policies demand.

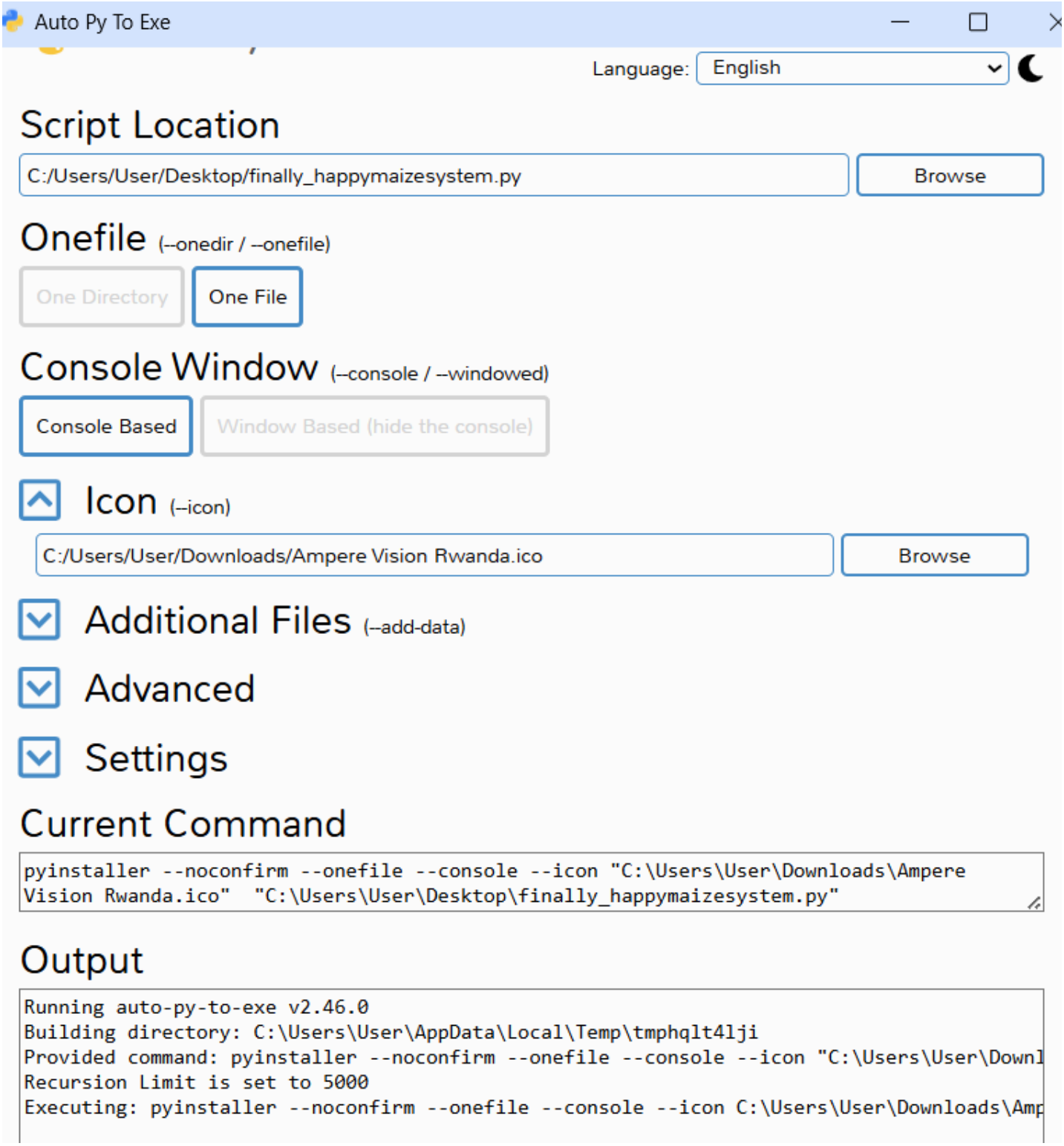

Figure 40: One executable file from python

Converting Python scripts to standalone executable files allows distribution without requiring Python installation

on target systems. PyInstaller is the most popular tool, using the command pyinstaller --onefile script.py to create a single executable file, or pyinstaller script.py for a folder distribution with dependencies. Auto-py-to-exe provides a user-friendly GUI interface for PyInstaller, making the conversion process more accessible through visual configuration options. cx_Freeze offers cross-platform support and fine-grained control over the build process, while Py2exe is Windows-specific but highly optimized for that platform.

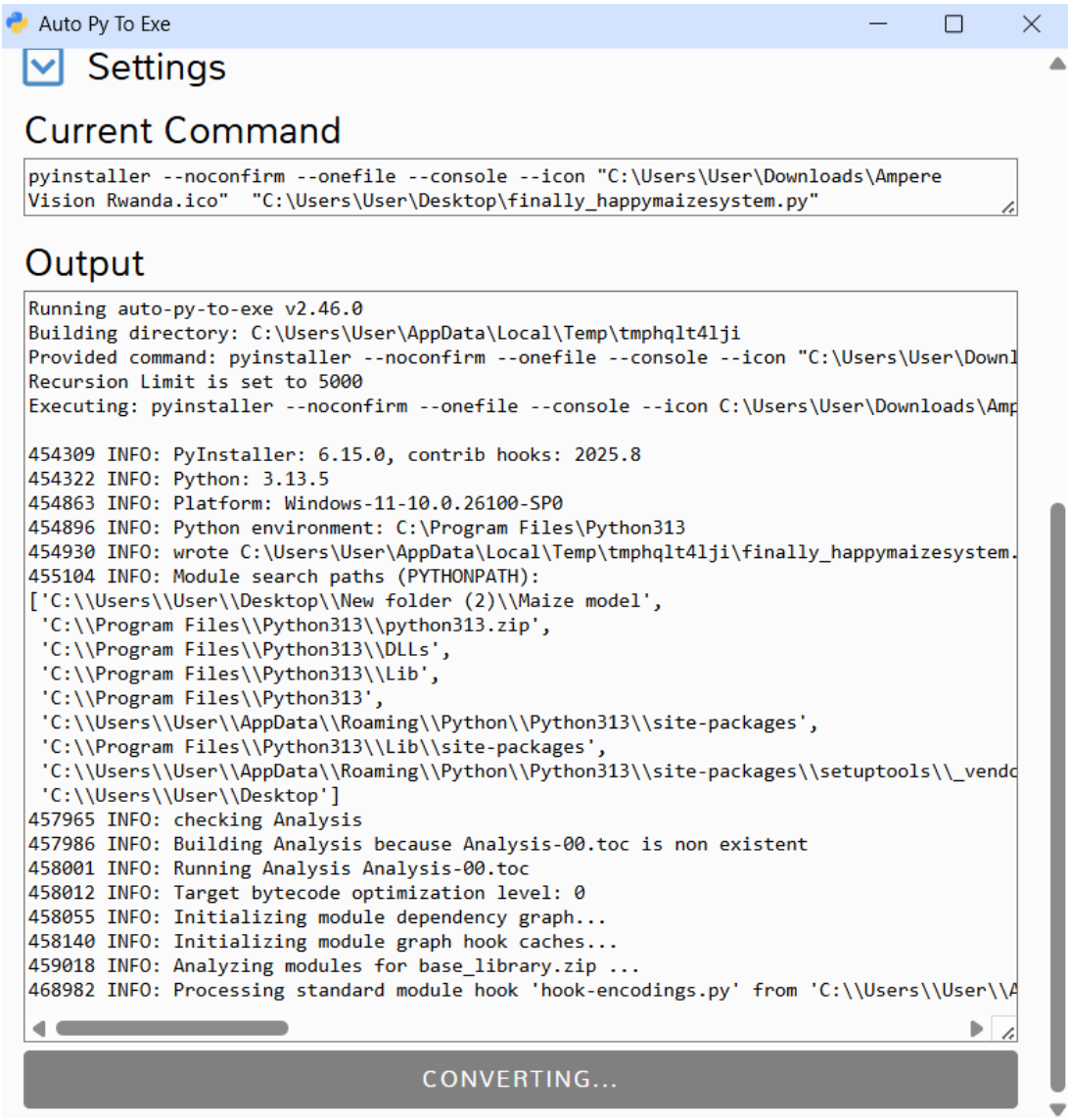

Figure 41: Turning the python code in .exe file to be used in the ground station

The conversion process typically involves specifying entry points, managing dependencies, and handling hidden imports that may not be automatically detected. Key considerations include file size optimization (executables can be large), startup time (may be slower than native Python), and ensuring all required libraries and assets are

properly bundled. For complex applications with multiple modules, additional configuration files may be needed to specify include paths, exclude unnecessary modules, and handle data files or icons that should be embedded in the final executable.

Figure 42: Prototype during calibration

Below is a custom-built quadcopter drone featuring a Pixhawk flight controller at its core. The Pixhawk serves as the central command unit of the drone, coordinating input from multiple sensors and sending output signals to control the motors and other peripherals. In this setup, the four arms of the drone each house a brushless DC motor with electronic speed controllers (ESCs), which are directly connected to the Pixhawk. These ESCs receive PWM (Pulse Width Modulation) signals from the Pixhawk to adjust motor speeds in real-time, enabling stable flight.

Figure 43: Prototype during testing

A GPS module with a compass , connected to the Pixhawk through dedicated ports, providing real-time positioning and heading information essential for navigation, especially during autonomous missions. A power distribution board (PDB) underneath the Pixhawk distributes power from the central battery to the ESCs and the Pixhawk itself, ensuring the entire system remains powered during operation.

Figure 44: Deployed quadcopter prototype

The Pixhawk communicates with a radio telemetry module (not fully visible but likely part of the setup near the remote control system), enabling wireless communication with a ground control station (GCS) via software like

Mission Planner or QGroundControl. This allows for real-time monitoring, mission planning, and parameter tuning. The remote control unit shown in the background provides manual control via a 2.4GHz RC transmitter, which communicates with the receiver module connected to the Pixhawk.

Quantitative survey

In Gishamvu sector of Huye District, our qualitative assessment revealed important insights into the current practices and challenges faced by maize farmers, who represent around 70% of the farming population in the area. Most farmers reported relying on traditional methods of pesticide spraying using knapsack sprayers, which they found to be physically demanding, time-consuming, and hazardous to their health. There was a strong expression of concern regarding health risks, especially as family members, including children, often assist in the spraying process. Despite the familiarity with traditional methods, many farmers showed a willingness to explore drone spraying as a safer and more efficient alternative. They emphasized the need for the technology to be affordable and easy to use, and they welcomed the idea of receiving training and seeing live demonstrations to build trust and understanding. The overall sentiment was cautiously optimistic, with farmers open to innovation that could improve their productivity and safety.

Figure 45: Response from farmers

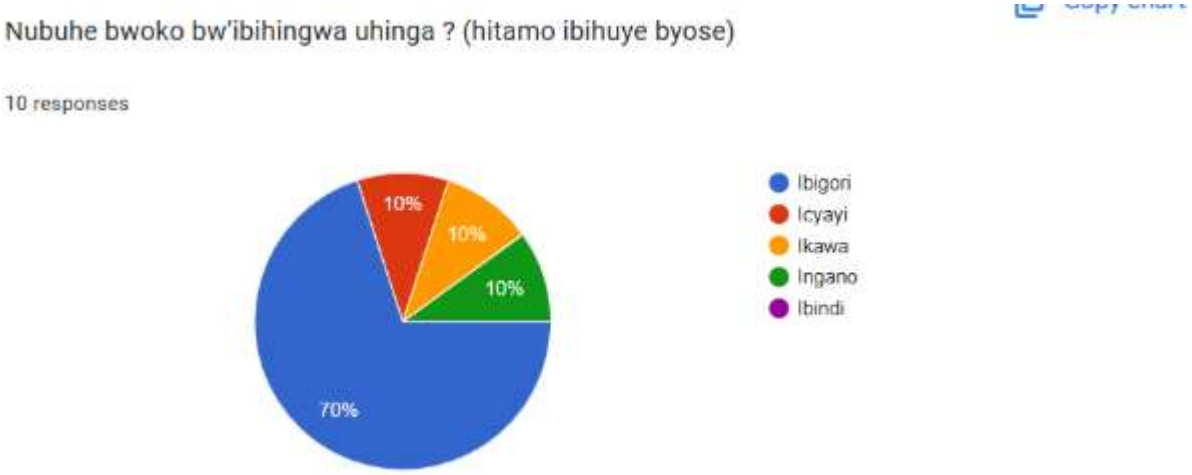

Figure 46: Qualitative survey from farmers on crops they plant

Figure 47: Data collection site in Gishamvu, Huye District

Qualitative analysis

Figure 48: Getting insights from farmers on the need of the agriculture spraying drone in Busogo, Musanze district

During our visit to Busogo sector in Musanze District, we engaged with approximately 30 farmers and gathered quantitative insights regarding their interest in drone-based pesticide spraying. Although the primary purpose of the visit was to discuss a different agricultural solution, we took the opportunity to ask about their expectations for drone spraying technology. A significant number of farmers expressed high interest in adopting drones for pesticide application, provided that the solution is affordable and suited to their unique needs. Key features they emphasized included the ability of the drone to operate on hilly terrain, given the topography of Busogo, as well as multi-crop compatibility, since farmers in the area grow a variety of crops beyond maize. Farmers also highlighted the need for practical demonstrations and training to fully understand the benefits and usage of drone

technology before adopting it.

CONCLUSION

All project objectives were successfully achieved. The drone frames were designed, modeled, and fabricated to effectively support all embedded electronic components. An AI-based computer vision model was developed and is capable of accurately detecting and classifying crop types and health status, specifically maize diseases. Finally, the UAV was fully calibrated, and comprehensive simulation testing was conducted, confirming the system's reliability and performance.

REFERENCES

[1] (WHO), "Pesticide Exposure and Respiratory Illness," 2021.

[2] (FAO), "The Future of Food and Agriculture.," 2019.

[3] G. o. Rwanda, " National Strategy for Transformation (NST2)," Kigali City, 2018.

[4] P. Strange, Plant disease: a threat to global food security,, Annu. Rev..

[5] J. He et al, " Deep residual learning for image recognition. In Proceedings of the IEEE conference on computer vision and pattern recognition," pp. 770-778, 2016.

[6] F. Chollet, "deep learning with depthwise separable convolution," *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition,* 2017.

[7] A. Suleiman et al. (. Husnain, "Enhancing agricultural health with AI: Drone-based machine learning for mango tree disease detection," *Google Scholar,* pp. 8-10, 2024.

[8] R.Dahiya, "Importance of Manual and Automation Testing," *Proceedings of the Sth International,* 2019.

[9] J. Liu, "Tomato diseases and pests detection based on improved Yolo V3 convolutional neural netwok," *Frontiers in plant science ,* p. 898, 2020.

[10] R.Paund, "Prediction of Plant Leaf Diseases using Drone and Image Processing Techniques.," *International Journal of Advanced Research in Science, Communication ,* vol. 4(1), no. https://doi.org/10.48175/IJARSCT-15002, pp. 8-14, 2024.

[11] M.Turkoglu et al, "PlantDiseaseNet: Convolutional neural network ensemble for plant disease and pest detection," *Signal, Image and Video Processing 1c.2,* pp. 301-309, 2022.

[12] M. Faria et al, "Classification of Potato Disease with Digital Image Processing Technique: A Hybrid Deep Learning Framework," *IEEE 13th Annual Computing and Communication Workshop and CCommunication Workshop and Conference (CCWC),* pp. 0820-0826, 2023.

[13] G. Ali Arshaghi, "Detection and classification of potato diseases using a new convolution neural network architecture," *Traitement du Signal,* vol. 38, p. 1783–1791, 2021.

[14] L. Choy Yuen Khew, "Evaluation of deep learning for image-based blackpepper disease and nutrient deficiency classification.," *IEEE,* pp. 1-6, 2021.

[15] A.Akhloufi., "CTPlantNet: A Hybrid CNN-Transformer Architecture for Plant Disease Classification," *International Conference on Microelectronics (ICM), Casablanca, Morocco,,* no. doi: 10.1109/ICM56065.2022.10005433., pp. 156-159, 2022.

[16] J Chen, "Research on identification algorithm of crop pests and diseases based on improved DenseNet model.," *International Conference on Image, Signal Processing, and Pattern Recognition,* 2023.

[17] J.Amreen Batool, "A compact deep learning approach integrating depthwise convolutions and spatial attention for plant disease classification," *Plant Methods,* 2025.

[18] A.Ali Husnain, "Enhancing agricultural health with AI: Drone-based machine learning for mango tree disease detection," *World Journal of Advanced Research and Reviews,* vol. 23, no. 2, p. 1267–1276, 2024.

[19] V.Mulham Fawakherji, "Shape and style GAN-based multispectral data augmentation for crop/weed segmentation in precision farming," *ScienceDirect,* 2025.

[20] V.Felipe A. Lopes, "PlantPlotGAN: A Physics-Informed Generative Adversarial Network for Plant Disease Prediction," *Computer Science > Computer Vision and Pattern Recognition,* pp. 192-196, 2020.

[21] S.Yan Zhang, "Automatic Plant Disease Detection Based on Tranvolution Detection Network With GAN Modules Using Leaf Images," *Frontiers,* vol. 13, pp. 97-105, 2022.

[22] N. Devanakonda Venkata Sai Chakradhar Reddy 1, "Drone-Based Multispectral Imaging for Precision Monitoring of Crop Growth Variables," *MDPI,* no. https://doi.org/10.3390/blsf2025041010, pp. 98-134, 2021.

[23] An Thanh Le, "Optimizing Plant Disease Classification with Hybrid Convolutional Neural Network–Recurrent Neural Network and Liquid Time-Constant Network," *MDPI,* vol. 14, no. https://doi.org/10.3390/app14199118, pp. 9118-9119, 2024.

[24] Srinivas Kanakala, "Detection and Classification of Diseases in Multi-Crop Leaves using LSTM and CNN Models," *Computer Science > Computer Vision and Pattern Recognition,* pp. 983-986, 2025.

[25] Dr. V. Rama Chandran, "Tomato Leaves Disease Detection using Hybrid Model CNN-LSTM," *Ijraset Journal For Research in Applied Science and Engineering Technology,* no. https://doi.org/10.22214/ijraset.2024.59173, 2024.

[26] Md Abrar Jahin, "Soybean Disease Detection via Interpretable Hybrid CNN-GNN: Integrating MobileNetV2 and GraphSAGE with Cross-Modal Attention," *Computer Science > Computer Vision and Pattern Recognition,* pp. 192-198, 2025

[27] I.Girma Tariku, "Advanced Image Preprocessing and Integrated Modeling for UAV Plant Image Classification," *MDPI,* vol. 8(11), no. https://doi.org/10.3390/drones8110645, pp. 645-682, 2024.

[28] "Analysis and Evaluation of the Image Preprocessing Process of a Six-Band Multispectral Camera Mounted on an Unmanned Aerial Vehicle for Winter Wheat Monitoring," *Jiale Jiang 1,2,3,4ORCID,Hengbiao Zheng 1,2,3,4,Xusheng Ji 1,2,3,4,Tao Cheng ,Yongchao Tian ,Yan Zhu,Weixing Cao ,Reza Ehsani Xia Yao ,* vol. 19, no. https://doi.org/10.3390/s19030747, p.

747, 2019

[29] M.Savitha Patil, "Accurate plant species analysis for plant classification using convolutional neural network architecture," *Zenodo home,* pp. 98-109, 2024.

[30] N.Nguyen Van Hieu, "Automatic Plant Image Identification of Vietnamese species using Deep Learning Models," *Computer Science > Computer Vision and Pattern Recognition,* pp. 209-218, 2020.

Appendix

Code- Objective 2

```
import cv2
import numpy as np
import os
import json
import threading
import time
from datetime import datetime, timedelta
import tkinter as tk
from tkinter import ttk, messagebox
import matplotlib.pyplot as plt
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
import queue
from collections import deque, Counter
import sqlite3
import csv
import io
import subprocess
import platform

# Optional imports with fallbacks
try:
    from reportlab.lib.pagesizes import A4
    from reportlab.platypus import SimpleDocTemplate, Paragraph, Spacer, Table, TableStyle, Image as RLImage, PageBreak
```

```
    from reportlab.lib.styles import getSampleStyleSheet, ParagraphStyle
    from reportlab.lib.units import inch
    from reportlab.lib import colors
    REPORTLAB_AVAILABLE = True
    print("ReportLab available - PDF generation enabled")
except ImportError:
    REPORTLAB_AVAILABLE = False
    print("ReportLab not available - PDF generation disabled")

class PersistentDataManager:
    """Manages persistent data storage and retrieval"""

    def __init__(self, data_dir="maize_data"):
        self.data_dir = data_dir
        self.db_path = os.path.join(data_dir, "detections.db")
        self.ensure_directory()
        self.init_database()

    def ensure_directory(self):
        """Ensure data directory exists"""
        try:
            os.makedirs(self.data_dir, exist_ok=True)
            os.makedirs(os.path.join(self.data_dir, "reports"), exist_ok=True)
            os.makedirs(os.path.join(self.data_dir, "images"), exist_ok=True)
        except Exception as e:
            print(f"Error creating directories: {e}")

    def init_database(self):
        """Initialize SQLite database"""
        try:
            conn = sqlite3.connect(self.db_path)
            cursor = conn.cursor()

            cursor.execute('''
                CREATE TABLE IF NOT EXISTS detections (
                    id INTEGER PRIMARY KEY AUTOINCREMENT,
                    timestamp TEXT NOT NULL,
```

```
                disease TEXT NOT NULL,
                confidence REAL NOT NULL,
                severity TEXT NOT NULL,
                session_id TEXT NOT NULL,
                date TEXT NOT NULL,
                hour INTEGER NOT NULL
            )
        ''')

        cursor.execute('''
            CREATE TABLE IF NOT EXISTS sessions (
                session_id TEXT PRIMARY KEY,
                start_time TEXT NOT NULL,
                end_time TEXT,
                total_scans INTEGER DEFAULT 0
            )
        ''')

        conn.commit()
        conn.close()
        print("Database initialized successfully")

    except Exception as e:
        print(f"Database initialization error: {e}")

def save_detection(self, detection_data, session_id):
    """Save detection to database"""
    try:
        conn = sqlite3.connect(self.db_path)
        cursor = conn.cursor()

        timestamp = detection_data['timestamp'].strftime('%Y-%m-%d %H:%M:%S')
        date_str = detection_data['timestamp'].strftime('%Y-%m-%d')
        hour = detection_data['timestamp'].hour

        cursor.execute('''
            INSERT INTO detections
```

```
            (timestamp, disease, confidence, severity, session_id, date, hour)
            VALUES (?, ?, ?, ?, ?, ?, ?)
        ''', (
            timestamp,
            detection_data['disease'],
            detection_data['confidence'],
            detection_data['severity'],
            session_id,
            date_str,
            hour
        ))

        conn.commit()
        conn.close()
        return True

    except Exception as e:
        print(f"Error saving detection: {e}")
        return False

def start_session(self, session_id):
    """Start a new session"""
    try:
        conn = sqlite3.connect(self.db_path)
        cursor = conn.cursor()

        cursor.execute('''
            INSERT OR REPLACE INTO sessions (session_id, start_time)
            VALUES (?, ?)
        ''', (session_id, datetime.now().strftime('%Y-%m-%d %H:%M:%S')))

        conn.commit()
        conn.close()

    except Exception as e:
        print(f"Error starting session: {e}")
```

```
def end_session(self, session_id, total_scans):
    """End a session"""
    try:
        conn = sqlite3.connect(self.db_path)
        cursor = conn.cursor()

        cursor.execute('''
            UPDATE sessions
            SET end_time = ?, total_scans = ?
            WHERE session_id = ?
        ''', (datetime.now().strftime('%Y-%m-%d %H:%M:%S'), total_scans, session_id))

        conn.commit()
        conn.close()

    except Exception as e:
        print(f"Error ending session: {e}")

def get_session_stats(self, session_id):
    """Get session statistics"""
    try:
        conn = sqlite3.connect(self.db_path)
        cursor = conn.cursor()

        # Get session info
        cursor.execute('SELECT * FROM sessions WHERE session_id = ?', (session_id,))
        session_info = cursor.fetchone()

        if not session_info:
            return None

        # Get detection counts
        cursor.execute('''
            SELECT disease, COUNT(*) as count, AVG(confidence) as avg_confidence
            FROM detections
            WHERE session_id = ?
            GROUP BY disease
```

```
            ORDER BY count DESC
        ''', (session_id,))

        disease_stats = cursor.fetchall()

        # Get hourly breakdown
        cursor.execute('''
            SELECT hour, disease, COUNT(*) as count
            FROM detections
            WHERE session_id = ?
            GROUP BY hour, disease
            ORDER BY hour, count DESC
        ''', (session_id,))

        hourly_stats = cursor.fetchall()

        conn.close()

        return {
            'session_info': session_info,
            'disease_stats': disease_stats,
            'hourly_stats': hourly_stats
        }

    except Exception as e:
        print(f"Error getting session stats: {e}")
        return None

def get_detections_by_date_range(self, start_date, end_date):
    """Get detections within date range"""
    try:
        conn = sqlite3.connect(self.db_path)
        cursor = conn.cursor()

        cursor.execute('''
            SELECT * FROM detections
            WHERE date BETWEEN ? AND ?
```

```
                ORDER BY timestamp DESC
            ''', (start_date.strftime('%Y-%m-%d'), end_date.strftime('%Y-%m-%d')))

            results = cursor.fetchall()
            conn.close()

            columns = ['id', 'timestamp', 'disease', 'confidence', 'severity',
                       'session_id', 'date', 'hour']

            return [dict(zip(columns, row)) for row in results]

        except Exception as e:
            print(f"Error retrieving detections: {e}")
            return []

class BilingualPDFGenerator:
    """Generates comprehensive bilingual PDF reports matching the sample format"""

    def __init__(self, data_manager):
        self.data_manager = data_manager
        if REPORTLAB_AVAILABLE:
            self.styles = getSampleStyleSheet()
            self.setup_custom_styles()

    def setup_custom_styles(self):
        """Setup custom styles for the report"""
        # Title style
        self.title_style = ParagraphStyle(
            'CustomTitle',
            parent=self.styles['Title'],
            fontSize=18,
            spaceAfter=30,
            alignment=1,  # Center
            textColor=colors.black
        )

        # Section header style
```

```
        self.section_style = ParagraphStyle(
            'SectionHeader',
            parent=self.styles['Heading1'],
            fontSize=14,
            spaceBefore=20,
            spaceAfter=10,
            textColor=colors.black
        )

        # Subsection style
        self.subsection_style = ParagraphStyle(
            'SubsectionHeader',
            parent=self.styles['Heading2'],
            fontSize=12,
            spaceBefore=15,
            spaceAfter=8,
            textColor=colors.black
        )

    def calculate_risk_score(self, disease_stats):
        """Calculate overall risk score"""
        risk_weights = {
            'Gray Leaf Spot': 10,
            'Blight': 10,
            'Common Rust': 5,
            'Unknown Disease': 3,
            'Healthy': 0
        }

        total_risk = 0
        for disease, count, confidence in disease_stats:
            weight = risk_weights.get(disease, 5)
            total_risk += count * weight

        return total_risk

    def get_risk_level(self, risk_score, total_scans):
```

```
    """Determine risk level based on score"""
    if total_scans == 0:
        return "UNKNOWN"

    risk_ratio = risk_score / total_scans

    if risk_ratio > 8:
        return "HIGH"
    elif risk_ratio > 4:
        return "MEDIUM"
    elif risk_ratio > 0:
        return "LOW"
    else:
        return "MINIMAL"

def generate_comprehensive_report(self, session_id, language='en'):
    """Generate comprehensive PDF report matching sample format"""
    if not REPORTLAB_AVAILABLE:
        print("ReportLab not available. Cannot generate PDF reports.")
        return None

    try:
        # Get session data
        session_data = self.data_manager.get_session_stats(session_id)
        if not session_data:
            print("No session data found")
            return None

        session_info = session_data['session_info']
        disease_stats = session_data['disease_stats']
        hourly_stats = session_data['hourly_stats']

        # Calculate statistics
        total_scans = sum(count for _, count, _ in disease_stats)
        healthy_count = next((count for disease, count, _ in disease_stats if disease == 'Healthy'), 0)
        health_rate = (healthy_count / max(total_scans, 1)) * 100
```

```
        # Calculate session duration
        start_time = datetime.strptime(session_info[1], '%Y-%m-%d %H:%M:%S')
        end_time = datetime.strptime(session_info[2], '%Y-%m-%d %H:%M:%S') if session_info[2] else
datetime.now()
        duration_minutes = (end_time - start_time).total_seconds() / 60

        # Risk assessment
        risk_score = self.calculate_risk_score(disease_stats)
        risk_level = self.get_risk_level(risk_score, total_scans)

        # Create filename
        timestamp = datetime.now().strftime("%Y%m%d_%H%M%S")
        lang_suffix = "EN" if language == 'en' else "RW"
        filename = os.path.join(self.data_manager.data_dir, "reports",
                                f"comprehensive_report_{lang_suffix}_{timestamp}.pdf")

        # Create PDF
        doc = SimpleDocTemplate(filename, pagesize=A4,
                                topMargin=0.5*inch, bottomMargin=0.5*inch,
                                leftMargin=0.5*inch, rightMargin=0.5*inch)
        story = []

        # Title
        title = "Happy's Maize Disease Detection System<br/>Comprehensive Field Analysis Report"
        story.append(Paragraph(title, self.title_style))
        story.append(Spacer(1, 0.3*inch))

        # Executive Summary
        story.append(Paragraph("EXECUTIVE SUMMARY", self.section_style))

        # Calculate critical and moderate issues
        critical_count = sum(count for disease, count, _ in disease_stats
                             if disease in ['Gray Leaf Spot', 'Blight'])
        moderate_count = sum(count for disease, count, _ in disease_stats
                             if disease == 'Common Rust')

        summary_data = [
```

```
    f"Field Health Status: {health_rate:.1f}% of scanned plants appear healthy.",
    f"Session Duration: {duration_minutes:.1f} minutes",
    f"Total Plant Scans: {total_scans}",
    f"Critical Issues: {critical_count} high-severity disease cases detected",
    f"Moderate Issues: {moderate_count} medium-severity disease cases detected",
    f"Overall Risk Level: {risk_level}"
]

for item in summary_data:
    story.append(Paragraph(item, self.styles['Normal']))
    story.append(Spacer(1, 6))

story.append(Spacer(1, 0.2*inch))

# Disease Detection Summary Table
story.append(Paragraph("DISEASE DETECTION SUMMARY", self.section_style))

if disease_stats:
    table_data = [['Disease', 'Cases', 'Percentage', 'Severity', 'Economic Impact']]

    disease_info = {
        'Healthy': {'severity': 'None', 'impact': 'No loss expected'},
        'Common Rust': {'severity': 'Medium', 'impact': '10-25% yield loss if untreated'},
        'Gray Leaf Spot': {'severity': 'High', 'impact': '15-40% yield loss possible'},
        'Blight': {'severity': 'High', 'impact': '20-50% yield loss possible'},
        'Unknown Disease': {'severity': 'Unknown', 'impact': 'Variable, requires assessment'}
    }

    for disease, count, confidence in disease_stats:
        percentage = f"{(count/total_scans)*100:.1f}%"
        info = disease_info.get(disease, {'severity': 'Unknown', 'impact': 'Unknown'})
        table_data.append([disease, str(count), percentage, info['severity'], info['impact']])

    table = Table(table_data, colWidths=[2*inch, 0.8*inch, 1*inch, 1*inch, 2.2*inch])
    table.setStyle(TableStyle([
        ('BACKGROUND', (0, 0), (-1, 0), colors.grey),
        ('TEXTCOLOR', (0, 0), (-1, 0), colors.whitesmoke),
```

```
        ('ALIGN', (0, 0), (-1, -1), 'LEFT'),
        ('FONTNAME', (0, 0), (-1, 0), 'Helvetica-Bold'),
        ('FONTSIZE', (0, 0), (-1, 0), 10),
        ('BOTTOMPADDING', (0, 0), (-1, 0), 12),
        ('BACKGROUND', (0, 1), (-1, -1), colors.beige),
        ('GRID', (0, 0), (-1, -1), 1, colors.black)
    ]))
    story.append(table)

story.append(Spacer(1, 0.3*inch))

# Immediate Action Plan
story.append(Paragraph("IMMEDIATE ACTION PLAN", self.section_style))

# Priority actions based on diseases found
priority_diseases = [(disease, count, confidence) for disease, count, confidence in disease_stats
                     if disease != 'Healthy' and count > 0]
priority_diseases.sort(key=lambda x: x[1], reverse=True)  # Sort by count

action_info = {
    'Gray Leaf Spot': {
        'priority': 'HIGH',
        'action': 'Immediate fungicide application and field isolation',
        'treatment': 'Use resistant varieties, crop rotation, fungicide application',
        'impact': '15-40% yield loss possible'
    },
    'Blight': {
        'priority': 'HIGH',
        'action': 'Remove affected parts and apply fungicide immediately',
        'treatment': 'Remove affected parts, apply copper-based fungicide',
        'impact': '20-50% yield loss possible'
    },
    'Common Rust': {
        'priority': 'MEDIUM',
        'action': 'Apply fungicide treatment within 48 hours',
        'treatment': 'Apply fungicide (Propiconazole), improve air circulation',
        'impact': '10-25% yield loss if untreated'
```

```
    },
    'Unknown Disease': {
        'priority': 'LOW',
        'action': 'Contact agricultural extension officer',
        'treatment': 'Consult agricultural specialist for diagnosis',
        'impact': 'Variable, requires assessment'
    }
}

for i, (disease, count, confidence) in enumerate(priority_diseases[:3], 1):
    info = action_info.get(disease, action_info['Unknown Disease'])

    story.append(Paragraph(f"Priority {i}: {disease} ({count} cases)", self.subsection_style))
    action_details = [
        f"Priority Level: {info['priority']}",
        f"Immediate Action: {info['action']}",
        f"Treatment: {info['treatment']}",
        f"Expected Impact: {info['impact']}"
    ]

    for detail in action_details:
        story.append(Paragraph(detail, self.styles['Normal']))
        story.append(Spacer(1, 4))

    story.append(Spacer(1, 0.1*inch))

# Page break for detailed analysis
story.append(PageBreak())

# Detailed Disease Analysis
story.append(Paragraph("DETAILED DISEASE ANALYSIS", self.section_style))

disease_descriptions = {
    'Gray Leaf Spot': {
        'description': 'Gray rectangular spots with dark borders',
        'symptoms': 'Gray rectangular spots, Dark borders, Leaf yellowing',
        'treatment': 'Use resistant varieties, crop rotation, fungicide application',
```

```
        'prevention': 'Crop rotation, residue management, resistant hybrids',
        'strategy': 'Implement 2-year crop rotation cycle'
    },
    'Common Rust': {
        'description': 'Rust-colored pustules on leaf surface',
        'symptoms': 'Orange/rust spots, Raised pustules, Scattered pattern',
        'treatment': 'Apply fungicide (Propiconazole), improve air circulation',
        'prevention': 'Use resistant varieties, proper field sanitation',
        'strategy': 'Plant resistant varieties in next season'
    },
    'Blight': {
        'description': 'Large brown lesions with yellow halos',
        'symptoms': 'Brown lesions, Yellow halos, Rapid spread',
        'treatment': 'Remove affected parts, apply copper-based fungicide',
        'prevention': 'Avoid overhead irrigation, improve drainage',
        'strategy': 'Implement strict sanitation protocols'
    },
    'Unknown Disease': {
        'description': 'Symptoms not matching known diseases',
        'symptoms': 'Unclear symptoms, Mixed indicators, Needs expert evaluation',
        'treatment': 'Consult agricultural specialist for diagnosis',
        'prevention': 'Regular monitoring and professional consultation',
        'strategy': 'Develop monitoring protocol with expert guidance'
    }
}

for disease, count, confidence in disease_stats:
    if disease == 'Healthy' or count == 0:
        continue

    desc = disease_descriptions.get(disease, disease_descriptions['Unknown Disease'])
    info = action_info.get(disease, action_info['Unknown Disease'])

    story.append(Paragraph(disease, self.subsection_style))

    details = [
        f"Cases Detected: {count} ({(count/total_scans)*100:.1f}% of total scans)",
```

```
            f"Severity Level: {info['priority'].title()}",
            f"Description: {desc['description']}",
            f"Symptoms: {desc['symptoms']}",
            f"Treatment: {desc['treatment']}",
            f"Prevention: {desc['prevention']}",
            f"Economic Impact: {info['impact']}",
            f"Long-term Strategy: {desc['strategy']}"
        ]

        for detail in details:
            story.append(Paragraph(detail, self.styles['Normal']))
            story.append(Spacer(1, 4))

        story.append(Spacer(1, 0.15*inch))

    # Hourly Detection Trends
    if hourly_stats:
        story.append(Paragraph("HOURLY DETECTION TRENDS", self.section_style))

        table_data = [['Hour', 'Disease', 'Count', 'Severity']]

        for hour, disease, count in hourly_stats:
            severity = action_info.get(disease, {'priority': 'Unknown'})['priority'].title()
            time_str = f"{hour:02d}:00"
            table_data.append([time_str, disease, str(count), severity])

        table = Table(table_data, colWidths=[1*inch, 2*inch, 1*inch, 1*inch])
        table.setStyle(TableStyle([
            ('BACKGROUND', (0, 0), (-1, 0), colors.grey),
            ('TEXTCOLOR', (0, 0), (-1, 0), colors.whitesmoke),
            ('ALIGN', (0, 0), (-1, -1), 'LEFT'),
            ('FONTNAME', (0, 0), (-1, 0), 'Helvetica-Bold'),
            ('FONTSIZE', (0, 0), (-1, 0), 10),
            ('BOTTOMPADDING', (0, 0), (-1, 0), 12),
            ('BACKGROUND', (0, 1), (-1, -1), colors.beige),
            ('GRID', (0, 0), (-1, -1), 1, colors.black)
        ]))
```

```
        story.append(table)
        story.append(Spacer(1, 0.2*inch))

    # Risk Assessment & Recommendations
    story.append(Paragraph("RISK ASSESSMENT & RECOMMENDATIONS", self.section_style))

    risk_summary = [
        f"Overall Risk Score: {risk_score} ({risk_level} RISK)",
        f"Field Health Rate: {health_rate:.1f}%",
        f"Session Duration: {duration_minutes:.1f} minutes",
        f"Total Scans Performed: {total_scans}"
    ]

    for item in risk_summary:
        story.append(Paragraph(item, self.styles['Normal']))
        story.append(Spacer(1, 4))

    # Risk factors
    if critical_count > 0 or moderate_count > 0:
        story.append(Spacer(1, 0.1*inch))
        story.append(Paragraph("Identified Risk Factors:", self.styles['Normal']))

        if critical_count > 0:
            critical_diseases = [disease for disease, count, _ in disease_stats
                                 if disease in ['Gray Leaf Spot', 'Blight'] and count > 0]
            for disease in critical_diseases:
                count = next(c for d, c, _ in disease_stats if d == disease)
                story.append(Paragraph(f"• High severity disease ({disease}) detected {count} times",
                                       self.styles['Normal']))

        if moderate_count > 0:
            story.append(Paragraph(f"• Medium severity disease (Common Rust) detected
{moderate_count} times",
                                   self.styles['Normal']))

    story.append(Spacer(1, 0.2*inch))
```

```
# Actionable Recommendations
story.append(Paragraph("ACTIONABLE RECOMMENDATIONS", self.subsection_style))

recommendations = [
    "1. IMMEDIATE: Apply fungicide treatment for Gray Leaf Spot within 24 hours",
    "2. MEDIUM-TERM: Implement crop rotation to break disease cycle",
    "3. LONG-TERM: Plant resistant maize varieties in next season",
    "4. IMMEDIATE: Apply Propiconazole fungicide",
    "5. MEDIUM-TERM: Improve air circulation between plants",
    "6. LONG-TERM: Select rust-resistant varieties",
    "7. URGENT: Conduct comprehensive field inspection",
    "8. URGENT: Consider professional agricultural consultation",
    "9. Monitor field conditions daily during disease season",
    "10. Maintain detailed records of all treatments applied",
    "11. Schedule follow-up disease detection scans weekly",
    "12. Implement integrated pest management practices",
    "13. Ensure proper plant spacing for air circulation",
    "14. Maintain soil health through proper nutrition",
    "15. Consider weather patterns in treatment timing"
]

for rec in recommendations:
    story.append(Paragraph(rec, self.styles['Normal']))
    story.append(Spacer(1, 4))

story.append(Spacer(1, 0.2*inch))

# Economic Impact Analysis
story.append(Paragraph("ECONOMIC IMPACT ANALYSIS", self.subsection_style))

economic_impacts = []
for disease, count, confidence in disease_stats:
    if disease != 'Healthy' and count > 0:
        impact = action_info.get(disease, {'impact': 'Unknown impact'})['impact']
        economic_impacts.append(f"• {disease}: {impact}")

for impact in economic_impacts:
```

```
            story.append(Paragraph(impact, self.styles['Normal']))
            story.append(Spacer(1, 4))

        story.append(Spacer(1, 0.3*inch))

        # Session Details
        story.append(Paragraph("SESSION DETAILS", self.section_style))

        session_details = [
            f"Session Started: {start_time.strftime('%Y-%m-%d %H:%M:%S')}",
            f"Report Generated: {datetime.now().strftime('%Y-%m-%d %H:%M:%S')}",
            "Detection System: Happy's Maize Disease Detection System v2.0",
            "Analysis Method: Advanced computer vision with color and texture analysis",
            "Report Type: Comprehensive Field Health Assessment"
        ]

        for detail in session_details:
            story.append(Paragraph(detail, self.styles['Normal']))
            story.append(Spacer(1, 4))

        story.append(Spacer(1, 0.2*inch))

        # Footer
        footer_text = ("This report was generated by Happy's Maize Disease Detection System. For best results, combine "
                       "these findings with professional agricultural advice and local extension services. Regular monitoring "
                       "and preventive care are essential for maintaining crop health and maximizing yield potential.")

        story.append(Paragraph(footer_text, self.styles['Normal']))

        # Build PDF
        doc.build(story)
        print(f"Comprehensive PDF report generated: {filename}")
        return filename

    except Exception as e:
```

```
            print(f"Error generating comprehensive PDF report: {e}")
            return None

class MaizeDiseaseDetector:
    def __init__(self):
        print("Initializing Maize Disease Detector...")
        self.data_manager = PersistentDataManager()
        self.pdf_generator = BilingualPDFGenerator(self.data_manager)

        self.session_id = datetime.now().strftime("%Y%m%d_%H%M%S")
        self.cap = None

        # Start session in database
        self.data_manager.start_session(self.session_id)

        self.disease_info = {
            'Healthy': {
                'severity': 'None',
                'color': (0, 255, 0),
                'treatment': 'Continue regular care'
            },
            'Common Rust': {
                'severity': 'Medium',
                'color': (0, 165, 255),
                'treatment': 'Apply fungicide'
            },
            'Gray Leaf Spot': {
                'severity': 'High',
                'color': (128, 128, 128),
                'treatment': 'Use resistant varieties'
            },
            'Blight': {
                'severity': 'High',
                'color': (0, 255, 255),
                'treatment': 'Remove affected parts'
            },
            'Unknown Disease': {
```

```
            'severity': 'Unknown',
            'color': (128, 128, 128),
            'treatment': 'Consult specialist'
        }
    }

    self.detection_history = deque(maxlen=10)
    self.frame_queue = queue.Queue(maxsize=5)
    self.detection_queue = queue.Queue(maxsize=10)

    self.running = False
    self.total_scans = 0
    self.disease_counts = Counter()

    print("Detector initialized successfully")

def init_camera(self, camera_index=0):
    """Initialize camera"""
    try:
        print(f"Trying to initialize camera {camera_index}...")
        self.cap = cv2.VideoCapture(camera_index)

        if not self.cap.isOpened():
            print(f"Camera {camera_index} not available, trying default...")
            self.cap = cv2.VideoCapture(0)

        if not self.cap.isOpened():
            print("No camera found!")
            return False

        # Test camera
        ret, frame = self.cap.read()
        if not ret:
            print("Camera cannot capture frames!")
            return False

        print("Camera initialized successfully")
```

```
            return True

        except Exception as e:
            print(f"Camera initialization error: {e}")
            return False

    def analyze_color_distribution(self, image):
        """Simple color analysis"""
        try:
            if image is None or image.size == 0:
                return None

            hsv = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)

            # Simple color detection
            green_mask = cv2.inRange(hsv, np.array([35, 40, 40]), np.array([80, 255, 255]))
            orange_mask = cv2.inRange(hsv, np.array([8, 100, 100]), np.array([25, 255, 255]))
            gray_mask = cv2.inRange(hsv, np.array([0, 0, 80]), np.array([180, 50, 180]))
            yellow_mask = cv2.inRange(hsv, np.array([20, 100, 100]), np.array([35, 255, 255]))

            total_pixels = image.shape[0] * image.shape[1]

            green_ratio = np.sum(green_mask > 0) / total_pixels
            orange_ratio = np.sum(orange_mask > 0) / total_pixels
            gray_ratio = np.sum(gray_mask > 0) / total_pixels
            yellow_ratio = np.sum(yellow_mask > 0) / total_pixels

            gray_img = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
            brightness = np.mean(gray_img)
            contrast = np.std(gray_img)

            return {
                'green': green_ratio,
                'orange': orange_ratio,
                'gray': gray_ratio,
                'yellow': yellow_ratio,
                'brightness': brightness,
```

```
            'contrast': contrast
        }

    except Exception as e:
        print(f"Error in color analysis: {e}")
        return None

def detect_disease(self, features):
    """Simple disease detection logic"""
    try:
        if features is None:
            return 'Unknown Disease', 0.3

        scores = {}

        # Healthy detection
        if features['green'] > 0.3 and features['orange'] < 0.1:
            scores['Healthy'] = features['green'] * 1.2

        # Common Rust detection (improved sensitivity)
        if features['orange'] > 0.05:
            scores['Common Rust'] = features['orange'] * 3

        # Gray Leaf Spot detection
        if features['gray'] > 0.15 and features['contrast'] > 40:
            scores['Gray Leaf Spot'] = features['gray'] * 2.5

        # Blight detection
        if features['yellow'] > 0.15 and features['brightness'] > 120:
            scores['Blight'] = features['yellow'] * 2.5

        if scores:
            best_disease = max(scores.keys(), key=lambda x: scores[x])
            confidence = min(scores[best_disease], 1.0)

            if confidence > 0.2:
                return best_disease, confidence
```

```
            return 'Unknown Disease', 0.3

        except Exception as e:
            print(f"Error in disease detection: {e}")
            return 'Unknown Disease', 0.0

    def update_detection_history(self, disease, confidence):
        """Update detection history and save to database"""
        self.detection_history.append((disease, confidence))
        self.total_scans += 1
        self.disease_counts[disease] += 1

        detection_record = {
            'timestamp': datetime.now(),
            'disease': disease,
            'confidence': confidence,
            'severity': self.disease_info[disease]['severity']
        }

        # Save to database
        self.data_manager.save_detection(detection_record, self.session_id)

    def get_smooth_prediction(self):
        """Get smoothed prediction"""
        if not self.detection_history:
            return 'Unknown Disease', 0.0

        # Use last few detections
        recent = list(self.detection_history)[-3:]

        disease_votes = {}
        total_conf = {}

        for disease, conf in recent:
            if disease not in disease_votes:
                disease_votes[disease] = 0
```

```
            total_conf[disease] = 0
        disease_votes[disease] += 1
        total_conf[disease] += conf

    if disease_votes:
        best_disease = max(disease_votes.keys(), key=lambda x: disease_votes[x])
        avg_conf = total_conf[best_disease] / disease_votes[best_disease]
        return best_disease, avg_conf

    return 'Unknown Disease', 0.0

def draw_ui(self, frame, disease, confidence):
    """Draw simple UI on frame"""
    try:
        h, w = frame.shape[:2]

        # Background overlay
        overlay = frame.copy()
        cv2.rectangle(overlay, (10, 10), (w-10, 150), (0, 0, 0), -1)
        cv2.addWeighted(overlay, 0.7, frame, 0.3, 0, frame)

        # Title
        cv2.putText(frame, "Maize Disease Detection", (20, 40),
                    cv2.FONT_HERSHEY_SIMPLEX, 0.8, (255, 255, 255), 2)

        # Disease
        color = self.disease_info[disease]['color']
        cv2.putText(frame, f"Disease: {disease}", (20, 70),
                    cv2.FONT_HERSHEY_SIMPLEX, 0.6, color, 2)

        # Confidence
        cv2.putText(frame, f"Confidence: {confidence:.1%}", (20, 100),
                    cv2.FONT_HERSHEY_SIMPLEX, 0.6, (255, 255, 255), 2)

        # Controls
        cv2.putText(frame, "Press 'q' to quit, 's' to save", (20, 130),
                    cv2.FONT_HERSHEY_SIMPLEX, 0.5, (200, 200, 200), 1)
```

```
    except Exception as e:
        print(f"Error drawing UI: {e}")

def camera_thread(self):
    """Camera capture thread"""
    while self.running:
        try:
            ret, frame = self.cap.read()
            if ret:
                try:
                    self.frame_queue.put_nowait(frame.copy())
                except queue.Full:
                    try:
                        self.frame_queue.get_nowait()
                        self.frame_queue.put_nowait(frame.copy())
                    except queue.Empty:
                        pass
            time.sleep(0.033)  # ~30 FPS
        except Exception as e:
            print(f"Camera thread error: {e}")
            break

def run_detection(self):
    """Main detection loop"""
    print("Starting detection...")
    self.running = True

    # Start camera thread
    camera_thread = threading.Thread(target=self.camera_thread, daemon=True)
    camera_thread.start()

    detection_count = 0

    try:
        while self.running:
            try:
```

```
    frame = self.frame_queue.get(timeout=1.0)
except queue.Empty:
    continue

detection_count += 1

# Analyze center region
h, w = frame.shape[:2]
crop_size = min(h, w) // 4
center_x, center_y = w // 2, h // 2
x1, x2 = center_x - crop_size, center_x + crop_size
y1, y2 = center_y - crop_size, center_y + crop_size

crop = frame[y1:y2, x1:x2]
features = self.analyze_color_distribution(crop)
disease, confidence = self.detect_disease(features)

# Update history
self.update_detection_history(disease, confidence)
smooth_disease, smooth_confidence = self.get_smooth_prediction()

# Send to dashboard
try:
    detection_data = {
        'disease': smooth_disease,
        'confidence': smooth_confidence,
        'timestamp': datetime.now()
    }
    self.detection_queue.put_nowait(detection_data)
except queue.Full:
    pass

# Draw analysis region
color = self.disease_info[smooth_disease]['color']
cv2.rectangle(frame, (x1, y1), (x2, y2), color, 2)

# Draw UI
```

```
            self.draw_ui(frame, smooth_disease, smooth_confidence)

            # Show frame
            cv2.imshow('Maize Disease Detection', frame)

            # Debug output
            if detection_count % 30 == 0:
                print(f"Detection: {smooth_disease} ({smooth_confidence:.1%})")

            # Handle keys
            key = cv2.waitKey(1) & 0xFF
            if key == ord('q'):
                break
            elif key == ord('s'):
                timestamp = datetime.now().strftime("%Y%m%d_%H%M%S")
                filename = f"detection_{timestamp}.jpg"
                cv2.imwrite(filename, frame)
                print(f"Saved: {filename}")

    except KeyboardInterrupt:
        print("Interrupted by user")
    except Exception as e:
        print(f"Detection error: {e}")
    finally:
        self.running = False
        cv2.destroyAllWindows()

def launch_dashboard(self):
    """Launch dashboard"""
    try:
        dashboard = DashboardWindow(self)
        dashboard.run()
    except Exception as e:
        print(f"Dashboard error: {e}")

def start_system(self):
    """Start the complete system"""
```

```
        if not self.init_camera():
            print("Failed to initialize camera!")
            return False

        print("Starting dashboard...")
        dashboard_thread = threading.Thread(target=self.launch_dashboard, daemon=True)
        dashboard_thread.start()

        time.sleep(1)  # Let dashboard initialize

        print("Starting detection...")
        detection_thread = threading.Thread(target=self.run_detection, daemon=True)
        detection_thread.start()

        return True

    def cleanup(self):
        """Cleanup resources"""
        try:
            self.running = False
            # End session in database
            self.data_manager.end_session(self.session_id, self.total_scans)
            if self.cap:
                self.cap.release()
            cv2.destroyAllWindows()
            print("Cleanup completed")
        except Exception as e:
            print(f"Cleanup error: {e}")

class DashboardWindow:
    def __init__(self, detector):
        self.detector = detector
        self.root = tk.Tk()
        self.root.title("Happy's Maize Disease Detection Dashboard")
        self.root.geometry("1400x900")

        self.current_disease = "No Detection"
```

```
        self.current_confidence = 0.0
        self.disease_history = deque(maxlen=50)
        self.confidence_history = deque(maxlen=50)
        self.timestamp_history = deque(maxlen=50)
        self.hourly_detection_data = {}

        self.setup_dashboard()
        self.update_dashboard()

    def setup_dashboard(self):
        """Setup enhanced dashboard UI"""
        title_frame = tk.Frame(self.root, bg='#2E8B57')
        title_frame.pack(fill='x', pady=5)

        title_label = tk.Label(title_frame, text="Happy's Maize Disease Detection Dashboard",
                               font=('Arial', 20, 'bold'), bg='#2E8B57', fg='white')
        title_label.pack(pady=10)

        # Main container
        main_frame = tk.Frame(self.root)
        main_frame.pack(fill='both', expand=True, padx=10, pady=5)

        # Left panel for status and controls
        left_panel = tk.Frame(main_frame, width=400)
        left_panel.pack(side='left', fill='y', padx=(0, 10))
        left_panel.pack_propagate(False)

        # Current status frame
        status_frame = tk.LabelFrame(left_panel, text="Live Detection Status",
                                     font=('Arial', 12, 'bold'), fg='#2E8B57')
        status_frame.pack(fill='x', pady=5)

        self.disease_var = tk.StringVar(value="No Detection")
        self.confidence_var = tk.StringVar(value="0%")
        self.total_scans_var = tk.StringVar(value="0")
        self.session_time_var = tk.StringVar(value="00:00")
```

```
# Status labels with better formatting
status_items = [
    ("Current Disease:", self.disease_var, 'red'),
    ("Confidence:", self.confidence_var, 'blue'),
    ("Total Scans:", self.total_scans_var, 'green'),
    ("Session Time:", self.session_time_var, 'purple')
]

for i, (label, var, color) in enumerate(status_items):
    tk.Label(status_frame, text=label, font=('Arial', 11)).grid(
        row=i, column=0, sticky='w', padx=10, pady=5)
    tk.Label(status_frame, textvariable=var, font=('Arial', 11, 'bold'),
            fg=color).grid(row=i, column=1, sticky='w', padx=10, pady=5)

# Session statistics frame
stats_frame = tk.LabelFrame(left_panel, text="Session Statistics",
                            font=('Arial', 12, 'bold'), fg='#2E8B57')
stats_frame.pack(fill='x', pady=5)

self.health_rate_var = tk.StringVar(value="0%")
self.risk_level_var = tk.StringVar(value="UNKNOWN")
self.critical_issues_var = tk.StringVar(value="0")

stats_items = [
    ("Health Rate:", self.health_rate_var, 'green'),
    ("Risk Level:", self.risk_level_var, 'red'),
    ("Critical Issues:", self.critical_issues_var, 'orange')
]

for i, (label, var, color) in enumerate(stats_items):
    tk.Label(stats_frame, text=label, font=('Arial', 11)).grid(
        row=i, column=0, sticky='w', padx=10, pady=5)
    tk.Label(stats_frame, textvariable=var, font=('Arial', 11, 'bold'),
            fg=color).grid(row=i, column=1, sticky='w', padx=10, pady=5)

# Recommendations frame
rec_frame = tk.LabelFrame(left_panel, text="Current Recommendations",
```

```
                    font=('Arial', 12, 'bold'), fg='#2E8B57')
rec_frame.pack(fill='both', expand=True, pady=5)

self.recommendations_text = tk.Text(rec_frame, height=8, wrap='word',
                                    font=('Arial', 10), state='disabled')
rec_scrollbar = tk.Scrollbar(rec_frame, orient='vertical',
                             command=self.recommendations_text.yview)
self.recommendations_text.configure(yscrollcommand=rec_scrollbar.set)

self.recommendations_text.pack(side='left', fill='both', expand=True, padx=5, pady=5)
rec_scrollbar.pack(side='right', fill='y')

# Control buttons frame
control_frame = tk.LabelFrame(left_panel, text="Controls",
                              font=('Arial', 12, 'bold'), fg='#2E8B57')
control_frame.pack(fill='x', pady=5)

# Enhanced buttons with better styling
buttons_config = [
    ("Generate Comprehensive Report", self.generate_comprehensive_report, '#4CAF50'),
    ("Quick Session Report", self.generate_quick_report, '#2196F3'),
    ("Export Session Data", self.export_session_data, '#FF9800'),
    ("View Session History", self.view_session_history, '#9C27B0'),
    ("Clear Current Session", self.clear_session, '#F44336')
]

for i, (text, command, color) in enumerate(buttons_config):
    btn = tk.Button(control_frame, text=text, command=command,
                    bg=color, fg='white', font=('Arial', 10, 'bold'),
                    relief='raised', bd=2)
    btn.pack(fill='x', padx=5, pady=2)

# Right panel for charts
right_panel = tk.Frame(main_frame)
right_panel.pack(side='right', fill='both', expand=True)

# Charts frame
```

```
    charts_frame = tk.LabelFrame(right_panel, text="Real-time Analytics",
                                 font=('Arial', 12, 'bold'), fg='#2E8B57')
    charts_frame.pack(fill='both', expand=True)

    # Create matplotlib figure with better layout
    self.fig, ((self.ax1, self.ax2), (self.ax3, self.ax4)) = plt.subplots(2, 2, figsize=(12, 10))
    self.fig.suptitle('Maize Disease Detection Analytics', fontsize=16, fontweight='bold')

    self.ax1.set_title('Disease Distribution', fontweight='bold')
    self.ax2.set_title('Detection Confidence Over Time', fontweight='bold')
    self.ax3.set_title('Hourly Detection Trends', fontweight='bold')
    self.ax4.set_title('Health vs Disease Ratio', fontweight='bold')

    self.canvas = FigureCanvasTkAgg(self.fig, charts_frame)
    self.canvas.get_tk_widget().pack(fill='both', expand=True, padx=5, pady=5)

    # Start session timer
    self.session_start_time = datetime.now()

    # Initialize empty charts (will populate with real camera data)
    self.update_charts()

def initialize_sample_data(self):
    """Initialize charts with sample data for demonstration"""
    # Create realistic sample data
    sample_diseases = [
        'Healthy', 'Healthy', 'Healthy', 'Gray Leaf Spot', 'Healthy',
        'Common Rust', 'Healthy', 'Gray Leaf Spot', 'Healthy', 'Unknown Disease',
        'Healthy', 'Blight', 'Gray Leaf Spot', 'Healthy', 'Common Rust'
    ]

    sample_confidences = [0.85, 0.92, 0.78, 0.89, 0.95, 0.76, 0.88, 0.91, 0.83, 0.65,
                          0.87, 0.79, 0.94, 0.82, 0.77]

    current_time = datetime.now()

    # Populate initial data
```

```
for i, (disease, confidence) in enumerate(zip(sample_diseases, sample_confidences)):
    timestamp = current_time - timedelta(minutes=i*2)  # Spread over last 30 minutes

    self.disease_history.append(disease)
    self.confidence_history.append(confidence)
    self.timestamp_history.append(timestamp)

    # Update hourly data
    hour = timestamp.hour
    if hour not in self.hourly_detection_data:
        self.hourly_detection_data[hour] = {}
    if disease not in self.hourly_detection_data[hour]:
        self.hourly_detection_data[hour][disease] = 0
    self.hourly_detection_data[hour][disease] += 1

# Update initial statistics
self.detector.total_scans = len(sample_diseases)
healthy_count = sum(1 for d in self.disease_history if d == 'Healthy')
health_rate = (healthy_count / len(self.disease_history)) * 100
self.health_rate_var.set(f"{health_rate:.1f}%")

critical_diseases = sum(1 for d in self.disease_history if d in ['Gray Leaf Spot', 'Blight'])
self.critical_issues_var.set(str(critical_diseases))

if critical_diseases > len(self.disease_history) * 0.3:
    self.risk_level_var.set("HIGH")
elif critical_diseases > 0:
    self.risk_level_var.set("MEDIUM")
else:
    self.risk_level_var.set("LOW")

# Set current detection to last sample
self.current_disease = sample_diseases[-1]
self.current_confidence = sample_confidences[-1]
self.disease_var.set(self.current_disease)
self.confidence_var.set(f"{self.current_confidence:.1%}")
self.total_scans_var.set(str(len(sample_diseases)))
```

```
# Update charts with sample data
self.update_charts()
self.update_recommendations()
title_frame = tk.Frame(self.root, bg='#2E8B57')
title_frame.pack(fill='x', pady=5)

title_label = tk.Label(title_frame, text="Happy's Maize Disease Detection Dashboard",
                       font=('Arial', 20, 'bold'), bg='#2E8B57', fg='white')
title_label.pack(pady=10)

# Main container
main_frame = tk.Frame(self.root)
main_frame.pack(fill='both', expand=True, padx=10, pady=5)

# Left panel for status and controls
left_panel = tk.Frame(main_frame, width=400)
left_panel.pack(side='left', fill='y', padx=(0, 10))
left_panel.pack_propagate(False)

# Current status frame
status_frame = tk.LabelFrame(left_panel, text="Live Detection Status",
                             font=('Arial', 12, 'bold'), fg='#2E8B57')
status_frame.pack(fill='x', pady=5)

self.disease_var = tk.StringVar(value="No Detection")
self.confidence_var = tk.StringVar(value="0%")
self.total_scans_var = tk.StringVar(value="0")
self.session_time_var = tk.StringVar(value="00:00")

# Status labels with better formatting
status_items = [
    ("Current Disease:", self.disease_var, 'red'),
    ("Confidence:", self.confidence_var, 'blue'),
    ("Total Scans:", self.total_scans_var, 'green'),
    ("Session Time:", self.session_time_var, 'purple')
]
```

```
for i, (label, var, color) in enumerate(status_items):
    tk.Label(status_frame, text=label, font=('Arial', 11)).grid(
        row=i, column=0, sticky='w', padx=10, pady=5)
    tk.Label(status_frame, textvariable=var, font=('Arial', 11, 'bold'),
            fg=color).grid(row=i, column=1, sticky='w', padx=10, pady=5)

# Session statistics frame
stats_frame = tk.LabelFrame(left_panel, text="Session Statistics",
                            font=('Arial', 12, 'bold'), fg='#2E8B57')
stats_frame.pack(fill='x', pady=5)

self.health_rate_var = tk.StringVar(value="0%")
self.risk_level_var = tk.StringVar(value="UNKNOWN")
self.critical_issues_var = tk.StringVar(value="0")

stats_items = [
    ("Health Rate:", self.health_rate_var, 'green'),
    ("Risk Level:", self.risk_level_var, 'red'),
    ("Critical Issues:", self.critical_issues_var, 'orange')
]

for i, (label, var, color) in enumerate(stats_items):
    tk.Label(stats_frame, text=label, font=('Arial', 11)).grid(
        row=i, column=0, sticky='w', padx=10, pady=5)
    tk.Label(stats_frame, textvariable=var, font=('Arial', 11, 'bold'),
            fg=color).grid(row=i, column=1, sticky='w', padx=10, pady=5)

# Recommendations frame
rec_frame = tk.LabelFrame(left_panel, text="Current Recommendations",
                          font=('Arial', 12, 'bold'), fg='#2E8B57')
rec_frame.pack(fill='both', expand=True, pady=5)

self.recommendations_text = tk.Text(rec_frame, height=8, wrap='word',
                                    font=('Arial', 10), state='disabled')
rec_scrollbar = tk.Scrollbar(rec_frame, orient='vertical',
                             command=self.recommendations_text.yview)
```

```
self.recommendations_text.configure(yscrollcommand=rec_scrollbar.set)

self.recommendations_text.pack(side='left', fill='both', expand=True, padx=5, pady=5)
rec_scrollbar.pack(side='right', fill='y')

# Control buttons frame
control_frame = tk.LabelFrame(left_panel, text="Controls",
                              font=('Arial', 12, 'bold'), fg='#2E8B57')
control_frame.pack(fill='x', pady=5)

# Enhanced buttons with better styling
buttons_config = [
    ("Generate Comprehensive Report", self.generate_comprehensive_report, '#4CAF50'),
    ("Quick Session Report", self.generate_quick_report, '#2196F3'),
    ("Export Session Data", self.export_session_data, '#FF9800'),
    ("View Session History", self.view_session_history, '#9C27B0'),
    ("Clear Current Session", self.clear_session, '#F44336')
]

for i, (text, command, color) in enumerate(buttons_config):
    btn = tk.Button(control_frame, text=text, command=command,
                    bg=color, fg='white', font=('Arial', 10, 'bold'),
                    relief='raised', bd=2)
    btn.pack(fill='x', padx=5, pady=2)

# Right panel for charts
right_panel = tk.Frame(main_frame)
right_panel.pack(side='right', fill='both', expand=True)

# Charts frame
charts_frame = tk.LabelFrame(right_panel, text="Real-time Analytics",
                             font=('Arial', 12, 'bold'), fg='#2E8B57')
charts_frame.pack(fill='both', expand=True)

# Create matplotlib figure with better layout
self.fig, ((self.ax1, self.ax2), (self.ax3, self.ax4)) = plt.subplots(2, 2, figsize=(12, 10))
self.fig.suptitle('Maize Disease Detection Analytics', fontsize=16, fontweight='bold')
```

```
    self.ax1.set_title('Disease Distribution', fontweight='bold')
    self.ax2.set_title('Detection Confidence Over Time', fontweight='bold')
    self.ax3.set_title('Hourly Detection Trends', fontweight='bold')
    self.ax4.set_title('Health vs Disease Ratio', fontweight='bold')

    self.canvas = FigureCanvasTkAgg(self.fig, charts_frame)
    self.canvas.get_tk_widget().pack(fill='both', expand=True, padx=5, pady=5)

    # Start session timer
    self.session_start_time = datetime.now()

def update_session_timer(self):
    """Update session timer"""
    elapsed = datetime.now() - self.session_start_time
    hours, remainder = divmod(elapsed.total_seconds(), 3600)
    minutes, seconds = divmod(remainder, 60)
    self.session_time_var.set(f"{int(hours):02d}:{int(minutes):02d}:{int(seconds):02d}")

def update_recommendations(self):
    """Update recommendations based on current detections"""
    recommendations = []

    if self.disease_history:
        recent_diseases = list(self.disease_history)[-10:]
        disease_counts = Counter(recent_diseases)

        # Priority recommendations based on detected diseases
        if 'Gray Leaf Spot' in disease_counts:
            count = disease_counts['Gray Leaf Spot']
            recommendations.append(f"URGENT: Gray Leaf Spot detected {count} times in recent scans")
            recommendations.append("   → Apply fungicide immediately")
            recommendations.append("   → Isolate affected areas")
            recommendations.append("   → Consider resistant varieties")

        if 'Blight' in disease_counts:
            count = disease_counts['Blight']
```

```
            recommendations.append(f"URGENT: Blight detected {count} times")
            recommendations.append("   → Remove affected plant parts")
            recommendations.append("   → Apply copper-based fungicide")
            recommendations.append("   → Improve drainage")

        if 'Common Rust' in disease_counts:
            count = disease_counts['Common Rust']
            recommendations.append(f"MODERATE: Common Rust detected {count} times")
            recommendations.append("   → Apply Propiconazole fungicide")
            recommendations.append("   → Improve air circulation")
            recommendations.append("   → Monitor closely")

        if 'Unknown Disease' in disease_counts:
            count = disease_counts['Unknown Disease']
            recommendations.append(f"ASSESS: Unknown disease patterns ({count} detections)")
            recommendations.append("   → Contact agricultural specialist")
            recommendations.append("   → Document symptoms")
            recommendations.append("   → Increase monitoring frequency")

        # General recommendations
        healthy_ratio = disease_counts.get('Healthy', 0) / len(recent_diseases)
        if healthy_ratio < 0.5:
            recommendations.append("Field health below 50%")
            recommendations.append("   → Comprehensive field inspection needed")
            recommendations.append("   → Consider professional consultation")

        if not recommendations:
            recommendations.append("No immediate concerns detected")
            recommendations.append("   → Continue regular monitoring")
            recommendations.append("   → Maintain good agricultural practices")
    else:
        recommendations.append("No detection data yet")
        recommendations.append("   → System warming up...")

    # Update recommendations display
    self.recommendations_text.config(state='normal')
    self.recommendations_text.delete(1.0, tk.END)
```

```
    self.recommendations_text.insert(1.0, '\n'.join(recommendations))
    self.recommendations_text.config(state='disabled')

def update_dashboard(self):
    """Update dashboard data"""
    try:
        # Update session timer
        self.update_session_timer()

        # Get latest detection
        try:
            detection_data = self.detector.detection_queue.get_nowait()
            self.current_disease = detection_data['disease']
            self.current_confidence = detection_data['confidence']
            current_time = detection_data['timestamp']

            self.disease_var.set(self.current_disease)
            self.confidence_var.set(f"{self.current_confidence:.1%}")
            self.total_scans_var.set(str(self.detector.total_scans))

            # Store data for charts
            self.disease_history.append(self.current_disease)
            self.confidence_history.append(self.current_confidence)
            self.timestamp_history.append(current_time)

            # Update hourly data
            hour = current_time.hour
            if hour not in self.hourly_detection_data:
                self.hourly_detection_data[hour] = {}
            if self.current_disease not in self.hourly_detection_data[hour]:
                self.hourly_detection_data[hour][self.current_disease] = 0
            self.hourly_detection_data[hour][self.current_disease] += 1

            # Update statistics
            if self.disease_history:
                healthy_count = sum(1 for d in self.disease_history if d == 'Healthy')
                health_rate = (healthy_count / len(self.disease_history)) * 100
```

```
                self.health_rate_var.set(f"{health_rate:.1f}%")

                # Calculate risk level
                critical_diseases = sum(1 for d in self.disease_history
                                        if d in ['Gray Leaf Spot', 'Blight'])
                critical_ratio = critical_diseases / len(self.disease_history)

                if critical_ratio > 0.3:
                    risk_level = "HIGH"
                elif critical_ratio > 0.1:
                    risk_level = "MEDIUM"
                elif critical_ratio > 0:
                    risk_level = "LOW"
                else:
                    risk_level = "MINIMAL"

                self.risk_level_var.set(risk_level)
                self.critical_issues_var.set(str(critical_diseases))

            self.update_charts()
            self.update_recommendations()

        except queue.Empty:
            pass
    except Exception as e:
        print(f"Dashboard update error: {e}")

    # Schedule next update
    self.root.after(1000, self.update_dashboard)

def update_charts(self):
    """Update enhanced dashboard charts with real camera detection data only"""
    try:
        # Clear all axes
        for ax in [self.ax1, self.ax2, self.ax3, self.ax4]:
            ax.clear()
```

```
# Chart 1: Disease Distribution (Pie Chart)
if len(self.disease_history) > 0:
    disease_counts = Counter(self.disease_history)
    diseases = list(disease_counts.keys())
    counts = list(disease_counts.values())

    # Define colors for diseases
    color_map = {
        'Healthy': '#2E8B57',
        'Common Rust': '#FF8C00',
        'Gray Leaf Spot': '#CD5C5C',
        'Blight': '#8B0000',
        'Unknown Disease': '#808080'
    }
    colors = [color_map.get(d, '#808080') for d in diseases]

    # Create pie chart
    wedges, texts, autotexts = self.ax1.pie(counts, labels=diseases, colors=colors,
                                            autopct='%1.1f%%', startangle=90)
    self.ax1.set_title('Disease Distribution', fontweight='bold')

    # Make text more readable
    for autotext in autotexts:
        autotext.set_color('white')
        autotext.set_fontweight('bold')
else:
    self.ax1.text(0.5, 0.5, 'Waiting for Camera Data...', transform=self.ax1.transAxes,
                  ha='center', va='center', fontsize=14, color='gray')
    self.ax1.set_title('Disease Distribution', fontweight='bold')

# Chart 2: Detection Confidence Over Time
if len(self.confidence_history) > 1:
    x_vals = range(len(self.confidence_history))
    confidence_values = list(self.confidence_history)

    self.ax2.plot(x_vals, confidence_values, 'b-', linewidth=2, marker='o',
                  markersize=4, alpha=0.7)
```

```
    self.ax2.fill_between(x_vals, confidence_values, alpha=0.3)
    self.ax2.set_title('Detection Confidence Over Time', fontweight='bold')
    self.ax2.set_ylabel('Confidence')
    self.ax2.set_xlabel('Detection Number')
    self.ax2.grid(True, alpha=0.3)
    self.ax2.set_ylim(0, 1)

    # Add average line
    avg_confidence = np.mean(confidence_values)
    self.ax2.axhline(y=avg_confidence, color='red', linestyle='--', alpha=0.7,
                     label=f'Average: {avg_confidence:.2f}')
    self.ax2.legend()
else:
    self.ax2.text(0.5, 0.5, 'Waiting for Camera Data...\n(Need 2+ detections)',
                  transform=self.ax2.transAxes, ha='center', va='center',
                  fontsize=12, color='gray')
    self.ax2.set_title('Detection Confidence Over Time', fontweight='bold')
    self.ax2.set_ylabel('Confidence')
    self.ax2.set_xlabel('Detection Number')
    self.ax2.grid(True, alpha=0.3)

# Chart 3: Hourly Detection Trends
if len(self.hourly_detection_data) > 0:
    hours = sorted(self.hourly_detection_data.keys())
    disease_types = set()
    for hour_data in self.hourly_detection_data.values():
        disease_types.update(hour_data.keys())

    disease_types = sorted(list(disease_types))

    # Create stacked bar chart
    bottom = np.zeros(len(hours))
    colors_map = {
        'Healthy': '#2E8B57',
        'Common Rust': '#FF8C00',
        'Gray Leaf Spot': '#CD5C5C',
        'Blight': '#8B0000',
```

```
        'Unknown Disease': '#808080'
    }

    for disease in disease_types:
        values = [self.hourly_detection_data[hour].get(disease, 0) for hour in hours]
        color = colors_map.get(disease, '#808080')
        bars = self.ax3.bar(hours, values, bottom=bottom, label=disease,
                            color=color, alpha=0.8)
        bottom += values

        # Add value labels on bars if values > 0
        for i, (hour, value) in enumerate(zip(hours, values)):
            if value > 0:
                self.ax3.text(hour, bottom[i] - value/2, str(value),
                              ha='center', va='center', fontweight='bold',
                              color='white', fontsize=8)

    self.ax3.set_title('Hourly Detection Trends', fontweight='bold')
    self.ax3.set_xlabel('Hour')
    self.ax3.set_ylabel('Detection Count')
    self.ax3.legend(fontsize=8, loc='upper left')
    self.ax3.grid(True, alpha=0.3, axis='y')
else:
    self.ax3.text(0.5, 0.5, 'Waiting for Camera Data...', transform=self.ax3.transAxes,
                  ha='center', va='center', fontsize=12, color='gray')
    self.ax3.set_title('Hourly Detection Trends', fontweight='bold')
    self.ax3.set_xlabel('Hour')
    self.ax3.set_ylabel('Detection Count')

# Chart 4: Health vs Disease Ratio with Status Gauge
if len(self.disease_history) > 0:
    healthy_count = sum(1 for d in self.disease_history if d == 'Healthy')
    diseased_count = len(self.disease_history) - healthy_count

    if healthy_count > 0 or diseased_count > 0:
        labels = ['Healthy', 'Diseased']
        sizes = [healthy_count, diseased_count]
```

```
        colors = ['#2E8B57', '#CD5C5C']
        explode = (0.05, 0)  # Slightly separate the slices

        wedges, texts, autotexts = self.ax4.pie(sizes, labels=labels, colors=colors,
                                                autopct='%1.1f%%', startangle=90,
                                                explode=explode, shadow=True)

        # Make text more readable
        for autotext in autotexts:
            autotext.set_color('white')
            autotext.set_fontweight('bold')
            autotext.set_fontsize(10)

        # Add health status text with color coding
        health_percentage = (healthy_count / len(self.disease_history)) * 100
        if health_percentage > 70:
            status = "EXCELLENT"
            status_color = '#2E8B57'
        elif health_percentage > 50:
            status = "GOOD"
            status_color = '#32CD32'
        elif health_percentage > 30:
            status = "MODERATE"
            status_color = '#FF8C00'
        else:
            status = "POOR"
            status_color = '#CD5C5C'

        self.ax4.text(0, -1.3, f"Field Status: {status}", ha='center',
                      fontsize=12, fontweight='bold', color=status_color,
                      bbox=dict(boxstyle="round,pad=0.3", facecolor=status_color, alpha=0.2))
    else:
        self.ax4.text(0.5, 0.5, 'No Disease Data', transform=self.ax4.transAxes,
                      ha='center', va='center', fontsize=12, color='gray')

    self.ax4.set_title('Health vs Disease Ratio', fontweight='bold')
else:
```

```
            self.ax4.text(0.5, 0.5, 'Waiting for Camera Data...', transform=self.ax4.transAxes,
                          ha='center', va='center', fontsize=12, color='gray')
            self.ax4.set_title('Health vs Disease Ratio', fontweight='bold')

        # Adjust layout and refresh
        plt.tight_layout()
        self.canvas.draw()

    except Exception as e:
        print(f"Chart update error: {e}")
        # If charts fail, show error message
        for i, ax in enumerate([self.ax1, self.ax2, self.ax3, self.ax4]):
            ax.clear()
            ax.text(0.5, 0.5, f'Chart Error\n{str(e)[:50]}...',
                    transform=ax.transAxes, ha='center', va='center',
                    fontsize=10, color='red')
        self.canvas.draw()

def generate_comprehensive_report(self):
    """Generate comprehensive PDF report using the enhanced generator"""
    try:
        # Show generation dialog
        dialog = tk.Toplevel(self.root)
        dialog.title("Generate Comprehensive Report")
        dialog.geometry("400x250")
        dialog.transient(self.root)
        dialog.grab_set()

        # Center dialog
        dialog.update_idletasks()
        x = (dialog.winfo_screenwidth() // 2) - (200)
        y = (dialog.winfo_screenheight() // 2) - (125)
        dialog.geometry(f"400x250+{x}+{y}")

        tk.Label(dialog, text="Comprehensive Field Analysis Report",
                 font=('Arial', 16, 'bold')).pack(pady=15)
```

```
tk.Label(dialog, text="This will generate a detailed PDF report\nmatching professional standards",
        font=('Arial', 11)).pack(pady=10)

# Language selection
lang_frame = tk.LabelFrame(dialog, text="Report Language")
lang_frame.pack(fill='x', padx=20, pady=10)

language_var = tk.StringVar(value='en')
tk.Radiobutton(lang_frame, text="English", variable=language_var,
              value='en').pack(anchor='w', padx=10, pady=2)
tk.Radiobutton(lang_frame, text="Kinyarwanda", variable=language_var,
              value='rw').pack(anchor='w', padx=10, pady=2)

def start_generation():
    dialog.destroy()

    # Show progress
    progress_dialog = tk.Toplevel(self.root)
    progress_dialog.title("Generating Report...")
    progress_dialog.geometry("400x150")
    progress_dialog.transient(self.root)
    progress_dialog.grab_set()

    tk.Label(progress_dialog, text="Generating comprehensive report...",
            font=('Arial', 12)).pack(pady=20)

    progress_bar = ttk.Progressbar(progress_dialog, mode='indeterminate', length=300)
    progress_bar.pack(pady=10)
    progress_bar.start()

    def generate():
        try:
            language = language_var.get()
            filename = self.detector.pdf_generator.generate_comprehensive_report(
                self.detector.session_id, language)

            progress_dialog.after(0, lambda: show_result(filename))
```

```
    except Exception as e:
        progress_dialog.after(0, lambda: show_error(str(e)))

def show_result(filename):
    progress_bar.stop()
    progress_dialog.destroy()

    if filename and os.path.exists(filename):
        # Show success dialog
        result_dialog = tk.Toplevel(self.root)
        result_dialog.title("Report Generated Successfully!")
        result_dialog.geometry("500x200")
        result_dialog.transient(self.root)
        result_dialog.grab_set()

        tk.Label(result_dialog, text="Comprehensive Report Generated!",
                 font=('Arial', 14, 'bold'), fg='green').pack(pady=10)

        file_info = f"File: {os.path.basename(filename)}\nLocation: {os.path.dirname(filename)}"
        tk.Label(result_dialog, text=file_info, font=('Arial', 10)).pack(pady=5)

        button_frame = tk.Frame(result_dialog)
        button_frame.pack(pady=20)

        def open_file():
            try:
                if platform.system() == "Windows":
                    os.startfile(filename)
                elif platform.system() == "Darwin":
                    subprocess.run(["open", filename])
                else:
                    subprocess.run(["xdg-open", filename])
                result_dialog.destroy()
            except Exception as e:
                messagebox.showerror("Error", f"Cannot open file: {e}")
```

```
            def open_folder():
                try:
                    folder = os.path.dirname(filename)
                    if platform.system() == "Windows":
                        subprocess.run(f'explorer "{folder}"')
                    elif platform.system() == "Darwin":
                        subprocess.run(["open", folder])
                    else:
                        subprocess.run(["xdg-open", folder])
                    result_dialog.destroy()
                except Exception as e:
                    messagebox.showerror("Error", f"Cannot open folder: {e}")

            tk.Button(button_frame, text="Open PDF", command=open_file,
                      bg='green', fg='white', font=('Arial', 12)).pack(side='left', padx=5)
            tk.Button(button_frame, text="Open Folder", command=open_folder,
                      bg='blue', fg='white', font=('Arial', 12)).pack(side='left', padx=5)
            tk.Button(button_frame, text="Close", command=result_dialog.destroy,
                      bg='gray', fg='white', font=('Arial', 12)).pack(side='left', padx=5)
        else:
            messagebox.showerror("Error", "Failed to generate comprehensive report")

    def show_error(error_msg):
        progress_bar.stop()
        progress_dialog.destroy()
        messagebox.showerror("Generation Failed", f"Failed to generate report:\n{error_msg}")

    # Start generation in separate thread
    import threading
    generation_thread = threading.Thread(target=generate, daemon=True)
    generation_thread.start()

# Buttons
button_frame = tk.Frame(dialog)
button_frame.pack(pady=20)

tk.Button(button_frame, text="Generate Report", command=start_generation,
```

```
            bg='green', fg='white', font=('Arial', 12, 'bold')).pack(side='left', padx=10)
        tk.Button(button_frame, text="Cancel", command=dialog.destroy,
            bg='red', fg='white', font=('Arial', 12)).pack(side='left', padx=10)

    except Exception as e:
        messagebox.showerror("Error", f"Failed to generate comprehensive report: {str(e)}")

def generate_quick_report(self):
    """Generate a quick summary report"""
    try:
        if not self.disease_history:
            messagebox.showwarning("No Data", "No detection data available for quick report")
            return

        # Create quick report dialog
        report_dialog = tk.Toplevel(self.root)
        report_dialog.title("Quick Session Report")
        report_dialog.geometry("600x500")
        report_dialog.transient(self.root)
        report_dialog.grab_set()

        # Report content
        report_frame = tk.Frame(report_dialog)
        report_frame.pack(fill='both', expand=True, padx=10, pady=10)

        # Title
        tk.Label(report_frame, text="Quick Session Report",
            font=('Arial', 16, 'bold')).pack(pady=(0, 10))

        # Create report text
        report_text = tk.Text(report_frame, wrap='word', font=('Courier', 10))
        scrollbar = tk.Scrollbar(report_frame, orient='vertical', command=report_text.yview)
        report_text.configure(yscrollcommand=scrollbar.set)

        # Generate report content
        session_duration = datetime.now() - self.session_start_time
        disease_counts = Counter(self.disease_history)
```

```
        total_detections = len(self.disease_history)
        healthy_count = disease_counts.get('Healthy', 0)
        health_rate = (healthy_count / total_detections) * 100 if total_detections > 0 else 0

        report_content = f"""
SESSION SUMMARY REPORT
Generated: {datetime.now().strftime('%Y-%m-%d %H:%M:%S')}
Session Duration: {str(session_duration).split('.')[0]}
Total Detections: {total_detections}

HEALTH STATISTICS
Overall Health Rate: {health_rate:.1f}%
Healthy Plants: {healthy_count}
Diseased Plants: {total_detections - healthy_count}

DISEASE BREAKDOWN
"""

        for disease, count in disease_counts.most_common():
            percentage = (count / total_detections) * 100
            report_content += f"{disease}: {count} detections ({percentage:.1f}%)\n"

        report_content += f"""

RISK ASSESSMENT
"""
        critical_count = sum(count for disease, count in disease_counts.items()
                             if disease in ['Gray Leaf Spot', 'Blight'])
        moderate_count = disease_counts.get('Common Rust', 0)

        if critical_count > total_detections * 0.3:
            risk_level = "HIGH"
        elif critical_count > 0 or moderate_count > total_detections * 0.2:
            risk_level = "MEDIUM"
        else:
            risk_level = "LOW"
```

```
        report_content += f"Risk Level: {risk_level}\n"
        report_content += f"Critical Issues: {critical_count}\n"
        report_content += f"Moderate Issues: {moderate_count}\n"

        report_content += """

IMMEDIATE RECOMMENDATIONS
"""
        if 'Gray Leaf Spot' in disease_counts:
            report_content += "• Apply fungicide for Gray Leaf Spot immediately\n"
        if 'Blight' in disease_counts:
            report_content += "• Remove affected plant parts and treat Blight\n"
        if 'Common Rust' in disease_counts:
            report_content += "• Apply rust treatment within 48 hours\n"
        if health_rate < 50:
            report_content += "• Consider professional agricultural consultation\n"

        report_text.insert(1.0, report_content)
        report_text.config(state='disabled')

        report_text.pack(side='left', fill='both', expand=True)
        scrollbar.pack(side='right', fill='y')

        # Buttons
        button_frame = tk.Frame(report_dialog)
        button_frame.pack(fill='x', padx=10, pady=10)

        def save_quick_report():
            try:
                timestamp = datetime.now().strftime("%Y%m%d_%H%M%S")
                filename = os.path.join(self.detector.data_manager.data_dir, "reports",
                                        f"quick_report_{timestamp}.txt")

                with open(filename, 'w') as f:
                    f.write(report_content)

                messagebox.showinfo("Saved", f"Quick report saved to:\n{filename}")
```

```
            except Exception as e:
                messagebox.showerror("Error", f"Failed to save report: {e}")

        tk.Button(button_frame, text="Save Report", command=save_quick_report,
                  bg='blue', fg='white', font=('Arial', 11)).pack(side='left', padx=5)
        tk.Button(button_frame, text="Close", command=report_dialog.destroy,
                  bg='gray', fg='white', font=('Arial', 11)).pack(side='right', padx=5)

    except Exception as e:
        messagebox.showerror("Error", f"Failed to generate quick report: {str(e)}")

def export_session_data(self):
    """Export current session data to CSV"""
    try:
        if not self.disease_history:
            messagebox.showwarning("No Data", "No session data to export")
            return

        # Get session data from database
        session_data = self.detector.data_manager.get_session_stats(self.detector.session_id)

        if not session_data:
            messagebox.showwarning("No Data", "No session data found in database")
            return

        timestamp = datetime.now().strftime("%Y%m%d_%H%M%S")
        filename = os.path.join(self.detector.data_manager.data_dir,
                                f"session_export_{timestamp}.csv")

        # Get all detections for this session
        end_date = datetime.now()
        start_date = self.session_start_time
        detections = self.detector.data_manager.get_detections_by_date_range(start_date, end_date)

        # Filter for current session
        session_detections = [d for d in detections if d['session_id'] == self.detector.session_id]
```

```
        with open(filename, 'w', newline='', encoding='utf-8') as csvfile:
            fieldnames = ['timestamp', 'disease', 'confidence', 'severity', 'session_id']
            writer = csv.DictWriter(csvfile, fieldnames=fieldnames)
            writer.writeheader()

            for detection in session_detections:
                writer.writerow({
                    'timestamp': detection['timestamp'],
                    'disease': detection['disease'],
                    'confidence': f"{detection['confidence']:.3f}",
                    'severity': detection['severity'],
                    'session_id': detection['session_id']
                })

        # Show success dialog
        result = messagebox.askyesno("Export Complete",
                                     f"Session data exported successfully!\n\n"
                                     f"Records: {len(session_detections)}\n"
                                     f"File: {os.path.basename(filename)}\n\n"
                                     f"Would you like to open the folder?")

        if result:
            folder = os.path.dirname(filename)
            if platform.system() == "Windows":
                subprocess.run(f'explorer "{folder}"')
            elif platform.system() == "Darwin":
                subprocess.run(["open", folder])
            else:
                subprocess.run(["xdg-open", folder])

    except Exception as e:
        messagebox.showerror("Export Failed", f"Failed to export session data:\n{str(e)}")

def view_session_history(self):
    """View historical session data"""
    try:
```

```
# Create history viewer dialog
history_dialog = tk.Toplevel(self.root)
history_dialog.title("Session History")
history_dialog.geometry("800x600")
history_dialog.transient(self.root)
history_dialog.grab_set()

# Create treeview for session history
frame = tk.Frame(history_dialog)
frame.pack(fill='both', expand=True, padx=10, pady=10)

tk.Label(frame, text="Historical Sessions",
        font=('Arial', 16, 'bold')).pack(pady=(0, 10))

# Treeview for session data
columns = ('Session ID', 'Start Time', 'Duration', 'Total Scans', 'Status')
tree = ttk.Treeview(frame, columns=columns, show='headings')

for col in columns:
    tree.heading(col, text=col)
    tree.column(col, width=150)

# Scrollbar
scrollbar = ttk.Scrollbar(frame, orient='vertical', command=tree.yview)
tree.configure(yscrollcommand=scrollbar.set)

# Get session data from database (simplified)
try:
    conn = sqlite3.connect(self.detector.data_manager.db_path)
    cursor = conn.cursor()
    cursor.execute('SELECT * FROM sessions ORDER BY start_time DESC LIMIT 20')
    sessions = cursor.fetchall()
    conn.close()

    for session in sessions:
        session_id, start_time, end_time, total_scans = session
```

```
            if end_time:
                start_dt = datetime.strptime(start_time, '%Y-%m-%d %H:%M:%S')
                end_dt = datetime.strptime(end_time, '%Y-%m-%d %H:%M:%S')
                duration = str(end_dt - start_dt).split('.')[0]
                status = "Completed"
            else:
                duration = "Ongoing"
                status = "Active" if session_id == self.detector.session_id else "Incomplete"

            tree.insert('', 'end', values=(
                session_id, start_time, duration, total_scans or 0, status
            ))

        except Exception as e:
            print(f"Error loading session history: {e}")
            tree.insert('', 'end', values=("Error loading data", "", "", "", ""))

        tree.pack(side='left', fill='both', expand=True)
        scrollbar.pack(side='right', fill='y')

        # Close button
        tk.Button(history_dialog, text="Close", command=history_dialog.destroy,
                  bg='gray', fg='white', font=('Arial', 11)).pack(pady=10)

    except Exception as e:
        messagebox.showerror("Error", f"Failed to view session history: {str(e)}")

def clear_session(self):
    """Clear current session data"""
    try:
        result = messagebox.askyesno("Clear Session",
                                     "Are you sure you want to clear the current session?\n"
                                     "This will reset all detection data.")

        if result:
            # Clear display data
            self.disease_history.clear()
```

```
        # Reset variables
        self.disease_var.set("No Detection")
        self.confidence_var.set("0%")
        self.health_rate_var.set("0%")
        self.risk_level_var.set("UNKNOWN")
        self.critical_issues_var.set("0")

        # Clear recommendations
        self.recommendations_text.config(state='normal')
        self.recommendations_text.delete(1.0, tk.END)
        self.recommendations_text.insert(1.0, "Session cleared. Starting fresh monitoring...")
        self.recommendations_text.config(state='disabled')

        # Clear charts
        for ax in [self.ax1, self.ax2, self.ax3, self.ax4]:
            ax.clear()
            ax.set_title(ax.get_title())
        self.canvas.draw()

        # Reset session timer
        self.session_start_time = datetime.now()

        # Create new session in database
        old_session_id = self.detector.session_id
        self.detector.data_manager.end_session(old_session_id, self.detector.total_scans)

        self.detector.session_id = datetime.now().strftime("%Y%m%d_%H%M%S")
        self.detector.data_manager.start_session(self.detector.session_id)
        self.detector.total_scans = 0
        self.detector.disease_counts.clear()

        messagebox.showinfo("Session Cleared", "New session started successfully!")

    except Exception as e:
        messagebox.showerror("Error", f"Failed to clear session: {str(e)}")
```

```
    def run(self):
        """Run the dashboard"""
        self.root.protocol("WM_DELETE_WINDOW", self.on_closing)
        self.root.mainloop()

    def on_closing(self):
        """Handle window closing"""
        self.detector.cleanup()
        self.root.destroy()

def main():
    """Main function"""
    print("=" * 50)
    print("Happy's Maize Disease Detection System")
    print("Enhanced Version with Comprehensive Reporting")
    print("=" * 50)

    try:
        detector = MaizeDiseaseDetector()

        if detector.start_system():
            print("System started successfully!")
            print("Dashboard running with enhanced features...")
            print("Features available:")
            print("- Live detection with camera feed")
            print("- Comprehensive PDF reports")
            print("- Real-time analytics dashboard")
            print("- Session management")
            print("- Data export capabilities")
            print("Press Ctrl+C to stop")

            try:
                while True:
                    time.sleep(1)
            except KeyboardInterrupt:
                print("\nShutting down...")
                detector.cleanup()
```

```
            print("System shut down successfully!")
        else:
            print("Failed to start system!")

    except Exception as e:
        print(f"System error: {e}")

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

Dataset

https://www.kaggle.com/datasets/smaranjitghose/corn-or-maize-leaf-disease-dataset