AI for Pest Detection

BY

HO JUN HAN

A REPORT

SUBMITTED TO

Universiti Tunku Abdul Rahman

in partial fulfillment of the requirements

for the degree of

BACHELOR OF INFORMATION TECHNOLOGY (HONOURS) COMMUNICATIONS AND NETWORKING

Faculty of Information and Communication Technology

(Kampar Campus)

FEBRUARY 2025

COPYRIGHT STATEMENT

ACKNOWLEDGEMENTS

I would like to express my sincere thanks and appreciation to my supervisors 'Cik AnaNabilah Binti Sa'uadi', who has given me this bright opportunity to engage in a mobileapplication development. It is my first step to establish a career in mobile application development. A million thanks to you.

Finally, I must say thanks to my parents and my family for their love, support, and continuous encouragement throughout the course.

ABSTRACT

Agricultural pest management is critical for ensuring crop health and yield, yet traditional detection methods are often labor-intensive and imprecise. This Final Year Project proposes an Artificial Intelligence (AI) based pest detection system integrated into a mobile application, empowering users to monitor plant health efficiently. The system utilizes a convolutional neural network (CNN) specificallyYOLOv5, trained on a dataset encompassing three common pest categories tentatively Whiteflies, and caterpillar, alongside healthy plant samples. Through the mobile app, users capture images of plants using their smartphone camera. The AI model, running on-device through TensorFlow Lite, analyzes the image to classify the plant as healthy or unhealthy. If unhealthy, it identifies the specific pest type and provides a tailored solution. Results are displayed within the app, and the detection data including plant status, pest type, solution, and confidence score are encoded into a QR code. This QR code enables seamless data sharing, such as with agricultural experts or record-keeping systems. Preliminary testing on a diverse test set achieved an accuracy above 85%, validating the system's effectiveness. This mobile solution offers a portable, user-friendly tool for pest management, enhancing precision agriculture through AI and innovative data transfer.

Area of Study: Artificial Intelligence and Application Development

Keywords: Agriculture, Pest Detection, Pest Control, Classification, Mobile Application and Deep Learning,

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

CNN	Convolutional Neural Network
VGG	Very Deep Convolutional Networks
IOT	Internet of Things
AI	Artificial Intelligence

Chapter 1
Introduction

Agriculture and horticulture are vital for sustaining food production and ornamental landscapes, yet pest infestations remain a persistent challenge, compromising plant health and productivity in farms and gardens alike. Whiteflies and Caterpillars are among the most common pests, causing significant damage through feeding, disease transmission, and structural harm to crops and plants. Traditional pest detection methods, such as visual inspection or pesticide overuse, are often inefficient, labor-intensive, and environmentally unsustainable, particularly for farmers and gardeners seeking timely and precise interventions. Advances in Artificial Intelligence (AI) and mobile technology offer a promising avenue to revolutionize pest management by delivering automated, accessible solutions. This project aims to develop an AI-based pest detection system integrated into a mobile application, designed to assist users in farms and gardens. The system employs a convolutional neural network (CNN), specifically YOLOv5, trained on a dataset of plant images to classify plants as healthy or unhealthy and detect infestations by Whiteflies or Caterpillars, alongside recommending targeted solutions such as use neem oil or insecticidal soap for Whiteflies. Users capture plant images using their smartphone camera, and the AI model, optimized with TensorFlow Lite for on-device processing, provides real-time results within the app. If a plant is unhealthy, the system identifies the pest and suggests an actionable remedy, with all detection data plant status, pest type, solution, and confidence score encoded into a QR code for sharing or documentation. This project seeks to empower farmers and gardeners with a portable, user-friendly tool to enhance pest management, supporting sustainable practices in both agricultural and garden settings.

1.1 Problem Statement and Motivation

Pest infestations, particularly by Whiteflies and Caterpillars, pose a significant threat to plant health and productivity in farms and gardens, leading to reduced crop yields, economic losses, and compromised horticultural sustainability. Traditional pest detection methods rely heavily on manual observation or broad-spectrum pesticide application, which are time-consuming, labor-intensive, and often inaccurate, especially for small-scale farmers and gardeners who lack access to expert resources or advanced diagnostic tools. These approaches frequently fail to identify specific pest types promptly, resulting in delayed or inappropriate treatments that exacerbate damage and environmental harm. Furthermore, the absence of portable, user-friendly systems limits the ability of non-experts to monitor plant health effectively and share detection data for further analysis or record-keeping. Existing technological solutions, while advanced, are typically expensive, complex, or inaccessible to the average user, creating a gap in practical pest management for agricultural and garden settings. There is a critical need for an automated, accessible, and precise pest detection system that can classify plants as healthy or unhealthy, identify specific pests, recommend targeted solutions, and facilitate data sharing, thereby empowering farmers and gardeners to mitigate pest-related challenges efficiently and sustainably.

The motivation for this project stems from the growing need to address pest related challenges in agriculture and horticulture, particularly in farms and gardens where Whiteflies and Caterpillars frequently threaten plant vitality. As global food demand rises and environmental sustainability becomes a priority, effective pest management is essential to safeguard crop yields and reduce reliance on harmful pesticides. Traditional detection methods, while widely used, often prove inadequate for timely and accurate identification, inspiring the pursuit of an innovative, technology-driven solution. The advent of Artificial Intelligence (AI) and mobile platforms offers a unique opportunity to democratize pest detection, making it accessible to farmers and gardeners who may lack specialized expertise or resources. This project is driven by a desire to harness AI's potential, specifically through YOLOv5, to deliver real-time, on-device analysis of plant health, identifying specific pests and providing actionable solutions. The integration of QR code functionality further motivates this work, enabling seamless data sharing for collaboration, documentation, or expert consultation an aspect often overlooked in existing tools. Academically, this project aligns with an interest in exploring the intersection of AI, mobile computing, and precision agriculture, while practically, it aims to empower users with a portable, user-friendly tool to enhance pest management. Ultimately, motivation lies in contributing a sustainable, impactful solution to improve agricultural and garden productivity for both novice and experienced practitioners.

1.2 Research Objectives

The primary aim of this project is to investigate and develop an AI-based pest detection system integrated into a mobile application to improve pest management efficiency and accessibility for farmers and gardeners in farm and garden settings.

The first research objective is to **investigate the feasibility of YOLOv5 for Pest Detection**. The aim of this objective is to explore the applicability and performance of the YOLOv5 convolutional neural network (CNN) which optimized with TensorFlow Lite in accurately classifying plant images into three categories which are Healthy, Whiteflies and Caterpillars by using the datasets that consist approximately 800 images collected from farm and garden environments, targeting a minimum validation accuracy of 85%.

The second objective is to **develop a real-time on-device detection system**. To design and implement a mobile application that leverages YOLOv5 and TensorFlow Lite for real-time, on-device classification of plant health status (Healthy or Unhealthy) and pest type identification, enabling immediate pest detection without internet dependency and suitable for use in remote farm and garden locations.

The third research objective is **to propose targeted pest management solutions**. To create a mechanism within the mobile app that identifies specific pest types (Whiteflies and Caterpillars) when a plant is Unhealthy and provides corresponding evidence-based solutions such as use neem oil or insecticidal soap for Whiteflies and apply Bacillus thuringiensis (Bt) or hand-pick them for Caterpillars to guide users in mitigating pest infestations effectively.

The last research objective is to **enable data sharing through QR code integration**. To integrate a QR code generation feature into the mobile app that encodes detection results (plant health status, pest type, solution, and confidence score), facilitating seamless data transfer to agricultural experts, record systems, or collaborators for consultation, documentation, or collaborative pest management strategies. The QR code is also helps when the farmer have many plants in a large farm by putting it on each plant so that the farmer know what to do to save the plant by just scanning the QR code that has been put infront of the plant to avoid confusion.

1.3 Project Scope and Direction

This project aims to design, develop, and evaluate an AI based pest detection system embedded within a mobile application, targeting pest management challenges in farm and garden environments. The scope centers on leveraging YOLOv5, a lightweight convolutional neural network (CNN), optimized with TensorFlow Lite, to classify plant images into three categories: Healthy, Whiteflies, and Caterpillars. The system utilizes a dataset of approximately 800 images that are collected from real-world farm and garden settings, to train the model with a target validation accuracy of at least 85%. The mobile app is developed using Android Studio as it targets Android users only. This application enables users such as farmers and gardeners to capture plant images with their smartphone camera, perform real-time detection on-device without internet reliance, and receive actionable outcomes. Key functionalities include classifying plant health (Healthy or Unhealthy), identifying pest types (Whiteflies and Caterpillars) if Unhealthy, proposing specific solutions (exp: Use neem oil or insecticidal soap for Whiteflies and Apply Bacillus thuringiensis (Bt) or hand-pick them for Caterpillars), and encoding results (status, pest, solution, confidence score) into a QR code for data sharing with agricultural experts or record systems. The scope includes model training and optimization in Jupyter Notebook, app development in Android Studio, and testing on a limited set of plant species common to farms and gardens (exp: tomatoes, strawberries, ornamentals). Excluded from the scope are integration of real-time environmental sensors (exp: humidity, temperature), detection of additional pest types beyond Whiteflies and Caterpillars and large-scale commercial deployment beyond a functional prototype. However, the scope only focuses on plants and not on other objects which are not plants. This scope only focuses on the two types of pests which are whiteflies and caterpillars, whereas other pests such as spider mites, fungus gnats, mealybugs and etc would be undetectable.

1.4 Contributions

This project contributes a practical solution to the field of precision agriculture and horticulture by developing an AI-based pest detection system integrated into a mobile application, specifically designed for use in farms and gardens. By leveraging a convolutional neural network (CNN), YOLOv5, trained to detect and classify plant health status and identify infestations by Whiteflies and Caterpillars, the system introduces an accessible, automated alternative to traditional manual pest identification methods. Unlike existing approaches, which often require expert knowledge or expensive equipment, this mobile app empowers farmers and gardeners with a portable, user-friendly tool that delivers real-time, on-device analysis classifying plants as healthy or unhealthy and providing targeted pest-specific solutions, such as use neem oil or insecticidal soap for Whiteflies and apply Bacillus thuringiensis (Bt) or hand-pick them for Caterpillars. The incorporation of QR code functionality represents a significant contribution, enabling users to encode and share detection data including plant status, pest type, solution, and confidence score for collaboration, record-keeping, or consultation with agricultural experts. This enhances the utility of pest management by bridging the gap between detection and actionable follow-up. Additionally, the project advances the application of AI and mobile technology in agriculture by demonstrating the feasibility of lightweight, on-device inference using TensorFlow Lite, achieving an anticipated accuracy above 85% based on preliminary testing. Ultimately, this work contributes an affordable, scalable, and sustainable tool that supports pest control efforts, improves plant health monitoring, and promotes informed decision-making in farm and garden settings.

1.5 Report Organization

This report is organized into 7 chapters: Chapter 1 Introduction, Chapter 2 Literature Review, Chapter 3 System Methodology/Approach, Chapter 4 System Design, Chapter 5 System Implementation and Testing, Chapter 6 System Outcome and Discussion and Chapter 7 Conclusion. Chapter 1 introduces the problem statement and motivation, objectives, project scope and direction, contributions and report organization. Chapter 2 consists of evaluation on existing system and strength and weaknesses of existing models. Chapter 3 consists of proposed method/approach, system architecture diagram, use case diagram, activity diagram and project timeline. Chapter 4 consists of system block diagram, system components specifications, model selection and architecture, data preprocessing, model training and tuning, performance evaluation of the model and mobile app development. Chapter 5 consists of hardware setup, software setup, system operations (with screenshot) and implementation issues and challenges. Chapter 6 consists of system testing, project challenges and objective evaluation. Lastly, Chapter 7 consists of conclusion and recommendations.

Chapter 2
Literature Review

2.1 Evaluation of Existing System

2.1.1 Convolutional Neural Network

Neural networks are effective classifiers for complex, non-linear problems, with significant advancements in their structure improving classification and clustering performance [1]. CNNs, the leading models for image classification, trace their origins to [2] studies on animal visual cortical cells. Fukushima (1980) introduced the Neocognitron, an early precursor to CNNs [3], while LeCun et al. (1989, 1998) formalized the modern CNN structure in the late 1990s [4]. Initially, limited computer power restricted neural networks to shallow architectures with one hidden layer. However, the rise of GPU-aided computing and enhanced hardware enabled the training of deeper networks [5]. A breakthrough came in 2012 which AlexNet, a deep CNN that excelled in the ILSVRC 2012 competition [6]. Subsequent models like VGG [7] and GoogLeNet [8] further improved performance. To address training challenges in very deep networks, another researcher [9]introduced deep residual learning with ResNet-152, followed by advanced architectures like Inception-ResNet-v2 [8], pushing the boundaries of recognition accuracy.

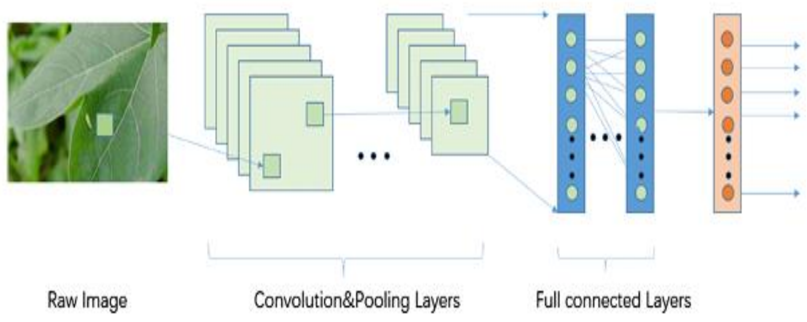


Figure 2.1.1.1 Basic structure of CNN

Convolutional neural networks (CNNs) excel at automatically extracting features from images, a capability demonstrated through visualization techniques as explored [10]. Figure 2.1.1.2 illustrates this process using a pest image: the left side shows the original agricultural pest image input, the middle displays features extracted by the first convolutional layer of AlexNet, and the right side reveals features after the first pooling layer. These visualizations highlight how the initial convolution and pooling layers activate the pest boundaries, effectively distinguishing them from complex backgrounds. The resulting images underscore CNNs' robust feature extraction ability, enabling clear separation and identification of pests, which is critical for applications like pest detection in agriculture.

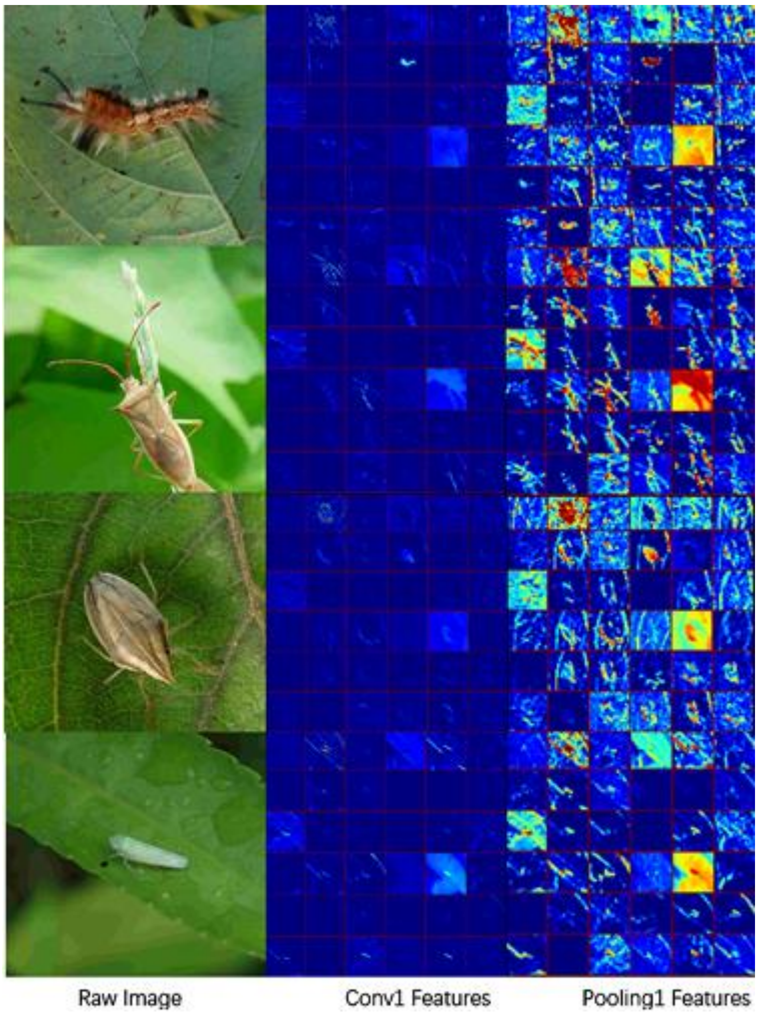


Figure 2.1.1.2 CNN feature visualization

2.1.2 YOLOv7 tiny model

The smartphone app implemented the YOLOv7 tiny model for pest detection, utilizing its compact neural network structure, as shown in Figure 2.1.2.1, to efficiently extract and fuse image features across multiple layers for real-time object detection [11]. An RGB image was processed by the model, with its deep learning layers extracting features at three scales and combining them to identify pests, visualized with bounding boxes. The model was trained and tested on 3,348 strawberry leaf images, split into training (60%), validation (20%), and testing (20%) datasets, targeting Two-Spotted Spider Mite (TSSM) and Powdery Mildew (PM) detection. This was done using PyTorch on a powerful computer with an NVIDIA GeForce RTX 3090 GPU, an 11th Gen Intel Core i9 11900F processor, and 32 GB of memory.

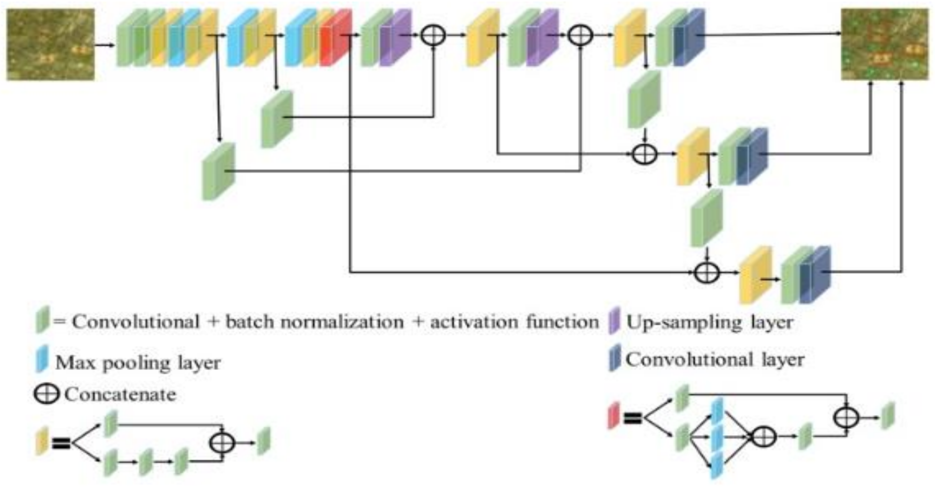


Figure 2.1.2.1 YOLOv7 tiny model structure

The YOLOv7 tiny model's detection accuracy was assessed using average precision (AP) and mean average precision (mAP) after training and testing on a GPU-equipped computer. Subsequently, the model was integrated into a smartphone app for field testing. The study compared the app's performance against traditional manual counting (using a magnifying lens) through metrics like coefficient of determination (R^2), root mean squared error (RMSE), counting accuracy, and speed. Counting accuracy was determined with Equation (1) in Figure 2.1.2.2, where the "estimated total pest population" (from the app or lens) was compared to the "actual pest population" (counted through microscope in a lab), evaluating the app's effectiveness in real-world pest detection.

$$Counting\ accuracy = \frac{Estimated\ total\ pest\ population\ at\ all\ sampling\ points}{Actual\ total\ pest\ population\ at\ all\ sampling\ points}$$

Figure 2.1.2.2 Equation 1

2.1.3 AI-enabled IoT-based pest detection system

The agricultural monitoring and intelligent security system, enabled by blockchain and IoT (as shown in Figure 2.1.3.1), features five key functions which are architectural framework, data gathering, analytics, processing, and communication (Figure 2.1.3.2) which to monitor farming practices and enhance security. Data is encrypted using advanced cryptographic techniques like AES and SHA, stored as byte streams in decentralized blockchain databases, ensuring confidentiality, integrity, and availability against cyber threats. The blockchain's distributed ledger provides immutable, transparent storage for data such as crop yields and weather conditions, preventing alterations and establishing a reliable truth source. This empowers farmers with greater control over their data, protecting it from hacking while allowing selective access to maintain privacy. (Figure 2.1.3.2) illustrates an IoT and blockchain-based system for monitoring pest identification and control in large farms, utilizing acoustic analytics, data gathering, encoding, actionable insights, and communication technologies. Acoustic analytics, supported by IoT networks and blockchain, enable pest monitoring, while data is encrypted with AES and Secure Hash Algorithm, stored as byte streams in a decentralized blockchain database to ensure availability, integrity, and secrecy. Figure (2.1.3.3) details the system's components—sensors, actuators, databases, and devices like computers and smartphones—tracking environmental factors such as light, soil, weather, and humidity. The pest detection process involves preprocessing pest noises, computing features, creating templates, developing models, and assessing detection accuracy. This intelligent system supports field, greenhouse, and animal care tasks, using sensors to monitor crops, livestock health, and environmental conditions, enhancing agricultural efficiency through integrated IoT and communication protocols [12].

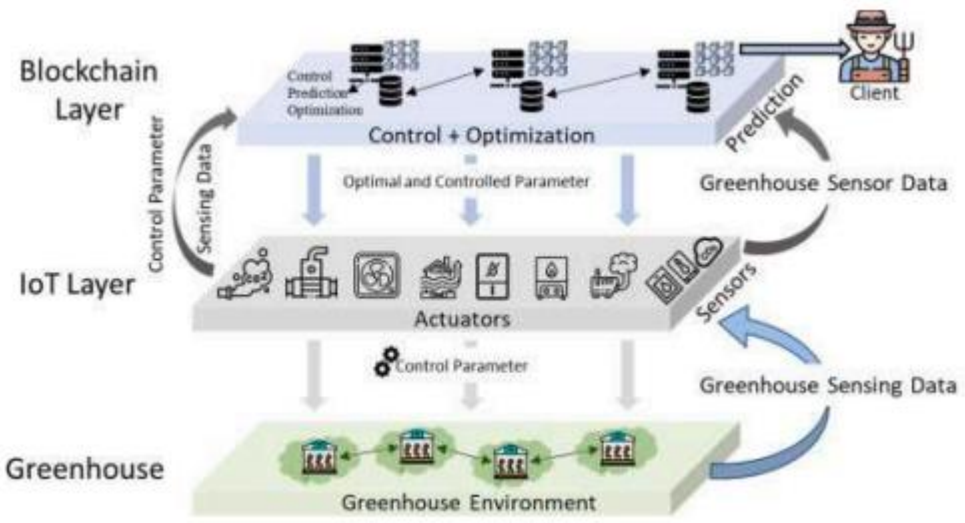


Figure 2.1.3.1 A smart agricultural monitoring system based on the IoT and blockchain environment.

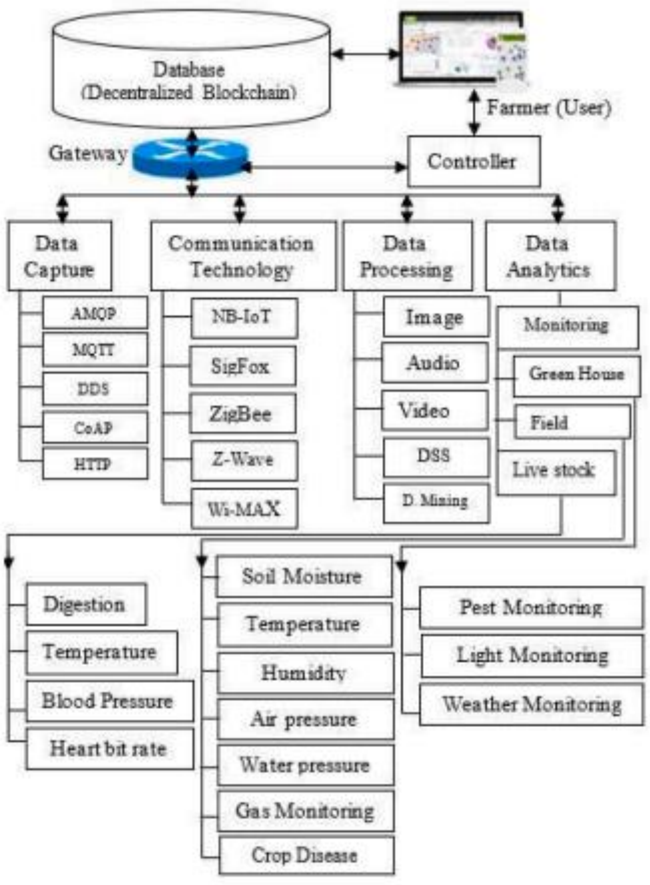


Figure 2.1.3.2 The architecture of IoT and blockchain enabled smart agriculture monitoring system.

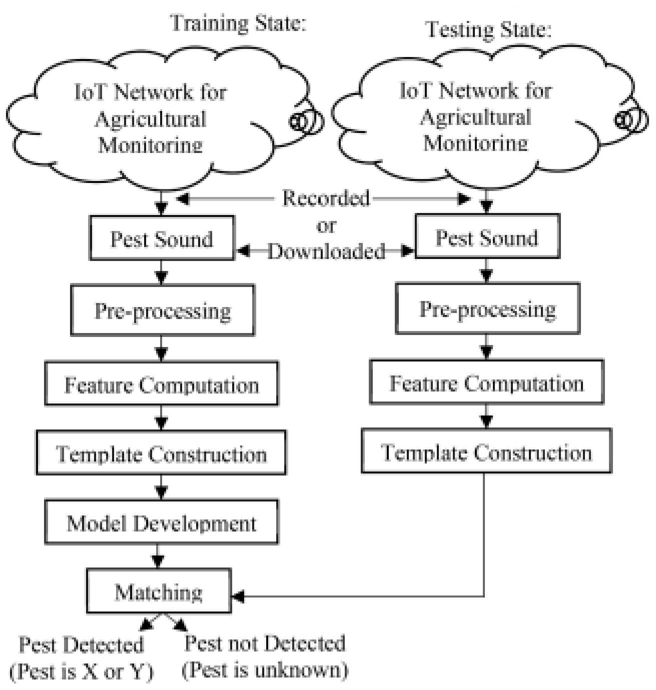


Figure 2.1.3.3 Works flow of pest detection procedure

2.2 Strength and Weakness of Existing Models

2.2.1 Convolutional Neural Networks (CNN)

Aspect	Strength	Weakness
Feature Extraction	Automatically extracts hierarchical features such as edges, textures from plant images without manual design	May struggle with subtle pest features (exp: Whiteflies' small size) if not enough high-resolution data.
Accuracy	• High classification accuracy • Ideal for detecting healthy and infested plants.	Accuracy depends heavily on dataset quality and size; small datasets (<400 images) may lead to overfitting.
Efficiency	Lightweight CNNs like MobileNetV2 are optimized for mobile devices, enabling fast inference (100ms) with low memory use (3-4 MB post-quantization).	Deeper CNNs (exp: VGG16) are computationally heavy, unsuitable for on-device use without optimization.
Generalization	Transfer learning leverages pre-trained models (exp: ImageNet), adapting to pests like Caterpillars or Whiteflies with fewer training images (100-200 per class).	Poor generalization if training data lacks diversity (exp: varied lighting, angles in farms/gardens).
Real-Time Use	Supports real-time detection on smartphones (exp: TensorFlow Lite), critical for immediate pest identification in the field	Real-time performance drops on low-end devices if not quantized properly, delaying pest response.
Scalability	Easily scalable to new pest types by fine-tuning with additional classes (exp: adding Mites), requiring minimal architectural changes.	Scaling to many classes (exp: 10+ pests) increases complexity and may reduce accuracy without more data.
Robustness	Robust to complex backgrounds (exp: separating pests from leaves), as CNNs focus on local patterns	Sensitive to noise or occlusions (exp: leaves covering Caterpillars), potentially misclassifying pests.
Data Requirements	Transfer learning reduces the need for large datasets compared to training from scratch, suitable for your 400-800 image range.	Still requires labelled data; collecting and labelling pest images (exp: Whiteflies) can be time-consuming.

Table 2.2.1 Strength and Weaknesses for CNN

2.2.2 YOLOv7 tiny model

Aspect	Strength	Weakness
Feature Extraction	Efficiently extracts multi-scale features using a compact structure, fusing layers for precise pest detection	May miss subtle pest features (exp: small Whiteflies) if scales aren't tuned for your specific pest sizes.
Accuracy	• High detection accuracy • Effective for locating Whiteflies and Caterpillars	Accuracy drops with small datasets or underrepresented pests (exp: Caterpillars), requiring 3,000+ images.
Efficiency	Lightweight (6MB), faster inference than larger YOLO variants (20-50ms on GPU), suitable for mobile deployment with optimization.	Heavier than MobileNetV2 (3MB), slower on low-end phones without quantization, impacting real-time use
Localization	• Uses bounding boxes to pinpoint pest locations (exp: Cat on leaves) • Enhancing visualization	• Overkill for your classification task (Healthy, Infected) • Adding unnecessary complexity
Real-Time Use	Designed for real-time object detection, ideal for rapid pest identification in farms/gardens on decent hardware (exp: RTX 3090).	Performance lags on low-spec mobile devices without GPU support, delaying pest response in the field.
Scalability	Scales well to multiple pest types by adjusting detection heads, adaptable to Whiteflies and Caterpillars with retraining.	Adding classes increases model size and training complexity, less flexible than MobileNetV2 for your needs.
Robustness	Robust to complex backgrounds, detecting pests across scales.	Sensitive to occlusions (exp: Caterpillars hidden by leaves), potentially missing partially obscured pests.
Data Requirements	• Benefits from pre-training (exp: COCO dataset) • Fine-tuned on 3348 images	Requires annotated bounding boxes, more labor-intensive than MobileNetV2's class labels (exp: 400-800 images).

Table 2.2.2 Strength and Weaknesses for YOLOv7 tiny model

2.2.3 AI-enabled IOT-based pest detection system

Aspect	Strength	Weakness
Real Time Monitoring	Enables continuous pest detection (exp: Whiteflies) through IoT sensors (e.g., acoustic, visual) and delivering instant insights	Relies on stable network connectivity; disruptions in remote farms/gardens can delay real-time data flow.
Feature Extraction	Combines AI (exp: CNNs) with IoT data (exp: acoustic analytics for Caterpillars), enhancing pest identification accuracy.	Complex feature fusion (exp: sound + image) may miss subtle pest traits if sensors lack precision or calibration.
Accuracy	• High detection accuracy with AI models • leveraging multi-sensor data for Caterpillars and other pests	Accuracy drops if training data doesn't match diverse farm/garden conditions or pest behaviours (exp: Whiteflies).
Scalability	Scales across large farms with IoT networks, managing multiple sensors (light, soil, weather) for comprehensive monitoring.	Scaling increases hardware costs (exp: sensors, gateways) and complexity, challenging small garden deployments.
Automation	Automates pest control (exp: actuators for pesticide spray) based on AI insights, reducing manual effort in fields.	Automation failures (exp: sensor malfunctions) can lead to missed detections or inappropriate responses.
Data Integration	Integrates diverse data (exp: humidity, pest noise) with blockchain for secure, immutable storage, ensuring reliability.	High data volume from IoT devices strains processing and storage, especially without robust infrastructure.
Robustness	Robust to environmental variability (exp: weather changes) with sensor fusion, improving pest detection in gardens.	Vulnerable to sensor failures or environmental noise (exp: wind masking Caterpillar sounds), reducing reliability.
Accessibility	• Remote access through smartphones/computers • Allows farmers to monitor pests from anywhere, enhancing usability.	Requires technical expertise for setup/maintenance, limiting adoption by non-tech-savvy farmers or gardeners.
Security	Blockchain encryption (exp: AES, SHA) protects pest data integrity and privacy	Security adds computational overhead, slowing down real-time processing on resource-limited IoT devices.

Cost -effectiveness	Reduces labour costs by automating pest detection over large areas, beneficial for big farms with many sensors.	High initial costs (exp: IoT hardware, blockchain setup) make it less feasible for small-scale gardens.

Table 2.2.3 Strength and Weaknesses AI-enabled IoT-based pest detection system

Chapter 3
System Methodology/Approach

The processes of the project were categorized into different phases in the development, which were project pre-development, data pre-processing, model training architecture building and data training, and prediction on test dataset.

3.1 Proposed method/Approach

The proposed method for developing the AI-based pest detection system follows a structured six-phase approach which are **Requirements, Design, Development, Testing, Deployment, and Review** to create a mobile application that leverages YOLOv5 and TensorFlow Lite for efficient pest management in farm and garden settings. This methodology ensures systematic progression from conceptualization to implementation, targeting a validation accuracy of at least 85% [14], real-time on-device detection, pest-specific solutions, and QR code data sharing for farmers and gardeners.

Agile methodology is a project management approach whichemphasizes on flexibility, collaboration, and iterative development. Agile development methodology is the first methodology that been used in the software development sector. This is because, the software will constantly change due to the evolvement of technology where new features are created. Therefore, the product also must be changed by implementing these new features following to the new trend of the technology. The agile methodology also offered some advantages which is agile methods are adaptable by providing the ability to shift strategies quickly without affecting or disrupt the flow of the project [13]. Agile fosters collaborative teamwork. This agile principle promotes collaborative teamwork by prioritizing face-to-facecommunication. By combining this approach with the principle that encourages teamsto break the project silos, the environment of conductive to effective collaboration teamwork would be created. Despite technological advancements and the rise of remotework, the fundamental value of face-to-face interaction remains unchanged in Agile methodology [13]. Lastly, the agile methods prioritize on customer needs which give advantages in software development. The teams can get the feedback from their actualcustomers or users quickly or faster through the accessibility of cloud-based software. Customer satisfaction is a key driver for software development [13]. Therefore, by collaborating with customers of the application that provide continuous feedback that can enhance the performance and the development of the system.

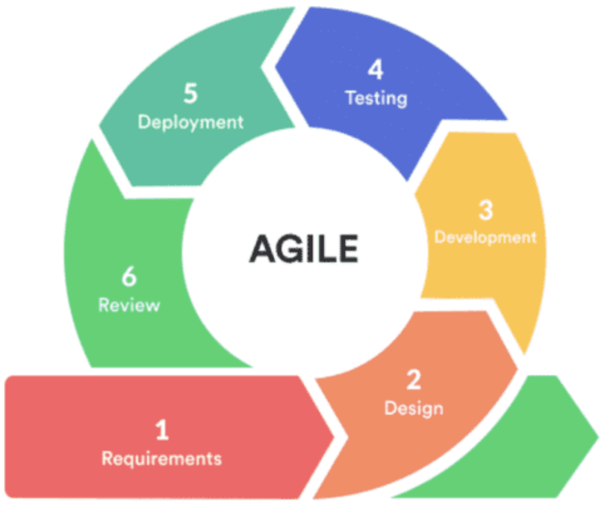


Figure 3.1 Agile Development Lifecycle

Based on Figure 3.1, it shows the diagram of agile development lifecycle. Thereare 6 phases in the agile development lifecycle which are **'Requirements'**, **'Design'**,' **Development'**, **'Testing'**, **'Deployment'** and **'Review'.** The initial phase establishes the project vision which is a mobile app that empowers farmers and gardeners to detect plant health (Healthy/Unhealthy), identify pests (Whiteflies or Caterpillars), propose solutions, and share results with QR code. This phase establishes the functional and non-functional requirements of the pest detection system. The functional requirements are capturing plant images through smartphone camera, classifying plant health as Healthy or Unhealthy, identifying three pest types (Whiteflies or Caterpillars). If the plants are detected unhealthy, the system will propose solutions (exp: Use neem oil or insecticidal soap for Whiteflies) and then will generate QR codes to encode results (status, pest, solution, confidence). Non-functional requirements encompass achieving >85% accuracy and offline capability. The target users would be farmers and gardeners.

The design phase outlines the system architecture and technical specifications. The YOLOv5 convolutional neural network (CNN), pre-trained on the COCO dataset, serves as the AI backbone, optimized with TensorFlow Lite for mobile deployment. The mobile app, built using Android Studio, integrates a camera module, inference engine, solution lookup table, and QR code generator. A use case diagram defines interactions: Farmers/Gardeners capture images, trigger detection, view results, and share QR codes with Agricultural Experts. The workflow is image capture → preprocessing (resize to 640x640, normalization) → inference → solution display → QR code generation, with data flow diagrams illustrating model inputs/outputs (bounding boxes, class labels) and app components.

The development phase implements the system in two main sub-phases: Model Deployment and App Development. For Model Deployment, a dataset of 859 images (448 training, 277 validation and 134 testing) is collected from farm/garden settings, organized into healthy, whiteflies, and caterpillars subdirectories, and pre-processed with augmentation (rotation 30°, width/height shifts 0.3, brightness 0.8-1.2, zoom 0.2, horizontal/vertical flips) using tools like augmentations in Jupyter Notebook ,YOLOv5 (exp: yolov5s) is trained through transfer learning: initialized with pre-trained weights, the model is fine-tuned on the custom dataset for 100 epochs. The trained model is converted to .tflite format (quantized, 14-20 MB) using YOLOv5's export.py script with TensorFlow Lite support. For App Development, Android Studio builds the app, integrating the .tflite model (with TensorFlow Lite's Android library), camera functionality (using Android's CameraX API), a solution table (exp: " Apply Bacillus thuringiensis (Bt) or hand-pick them" for caterpillars), and QR code generation (through ZXing library). The UI, designed with Android layouts (XML), features image capture buttons and result displays, coded in Java or Kotlin to process YOLOv5 inference outputs (bounding boxes, confidence scores) and present results to users.

The testing phase assesses the system's performance and usability. The YOLOv5 model is evaluated on a held-out test set (exp: 20% of images not used in validation) for mean Average Precision (mAP (0.5) >85%), per-class precision/recall (exp: caterpillars' detection), and inference speed (100-150ms on mobile). The app undergoes unit testing (exp: model inference accuracy), integration testing (camera to QR code workflow), and real-world testing on an

Android device (exp: API 21+), capturing live farm/garden images. Usability is tested with simulated users (exp: ease of result viewing, QR code scanning), and results are benchmarked against prior works (exp: YOLOv5 achieving 90% mAP in similar tasks). Issues (exp: mAP below target) prompt iterative refinements, such as adjusting hyperparameters or increasing training epochs.

The deployment phase delivers the completed prototype to users. The .tflite YOLOv5 model is embedded in the Android app, compiled into an APK using Android Studio, and installed on a test device (exp: Android phone). A user guide details operation: capture image, view detection results (pest type, location), share QR code. This phase ensures offline functionality and QR code compatibility with external scanners, providing a standalone solution within the project scope, with potential for broader distribution deferred to future work.

The review phase evaluates the project against its objectives and outlines future directions. Model performance (exp: current 79.89% vs 85% target), inference speed, and app usability are analyzed, with findings compared to requirements and literature. Successes (exp: real-time detection, QR code sharing) and challenges (exp: accuracy shortfall) are discussed, with recommendations for improvement such as dataset expansion, additional pest types (exp aphids), or iOS porting concluding the project with a functional prototype and insights for enhancement.

This six-phase approach integrates YOLOv5's AI efficiency with TensorFlow Lite's mobile optimization and Android Studio's robust development environment, delivering a practical pest detection tool that enhances farm and garden management through accurate classification, actionable solutions, and efficient data sharing.

3.2 System Architecture Diagram

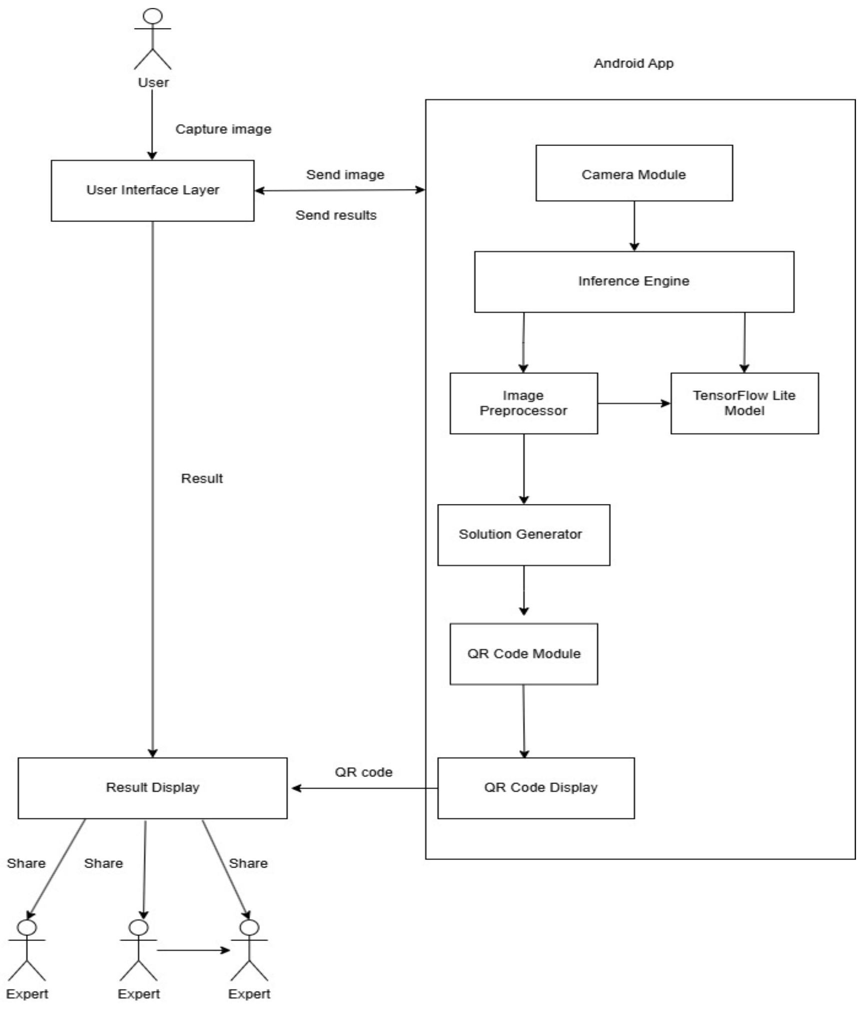


Figure 3.2 System Architecture Diagram

Based on the system architecture diagram above, there several components included which are User Interface (UI) Layer, Camera Module, Inference Engine, Solution Generator and QR Code Generator. User Interface (UI) Layer is the front-end part of the Android app that consists of screens and buttons the users see and use. It’s built using Android Studio with XML layouts and coded in Java or Kotlin.

The purpose of this UI layer is to acts as user’s gateway to the app by letting them start actions (like taking a photo) and see results (like pest detection and QR codes). There are several parts in the UI layer which are Image Capture Interface, Result Display and QR Code View. The Image Capture Interface has a button (exp: Take Photo) that triggers the camera. For Result Display there would be text or views showing "Unhealthy, infected by Caterpillars.

Solution: Apply Bacillus thuringiensis (Bt) or hand-pick them. Confidence: 87%.". The QR Code View would generate an image of the QR code for sharing purposes.

The Camera Module is the part that uses the phone's camera to take pictures of plants. It's powered by Android's CameraX API, integrated in Android Studio. The purpose of this camera module is to capture raw image (exp: a tomato leaf with Caterpillars) for the app to analyze. This module works when the users press the capture button, CameraX grabs the image and sends it to the next component.

The Inference Engine works the brain of the application as the pest detection using the YOLOv5 model. The Inference Engine consist of 2 sub-parts which are Image Preprocessor and TensorFlow Lite Model. The Image Preprocessor prepares the image for the model by resizing it to 224x224 pixels (YOLOv5's required size) and normalizing pixel values (exp: from 0-255 to 0-1). The TensorFlow Lite Model consists of pre-trained .tflite file (exp: pest_detection_improved.tflite) that created in Jupyter Notebook loaded through TensorFlow Lite's Android library.

The Solution Generator maps inference outputs to pest-specific solutions. It uses a hardcoded lookup table (exp: HashMap in Kotlin) to associate pest types with solutions such as ("Healthy" → "No action needed" ,"Whiteflies" → "Use insecticidal soap", " Caterpillars" → " Apply Bacillus thuringiensis (Bt) or hand-pick them". The Solution Generator is implemented in Java/Kotlin within Android Studio.

The QR Code Module encodes detection results into a QR code for sharing. It converts results (exp: "status": "Unhealthy", "pest": " Caterpillars", "solution": " Apply Bacillus thuringiensis (Bt) or hand-pick them", "confidence": "87%") into a QR code image using the ZXing library.

Data Flow in this System Architecture Diagram:

Target users : Farmers/Gardeners

1. User Start: Users open the app and tap "Take Photo" on the User Interface Layer.
2. Photo Taken: The Camera Module (CameraX) snaps a picture of your plant and sends it to the Inference Engine.
3. Detection Happens: Inside the Inference Engine:
    - The Image Preprocessor resizes and adjusts the photo.

   - The TensorFlow Lite Model (YOLOv5) analyzes it and says, "80% chance it’s Beetles."

4. Solution Added: The Solution Generator takes " Caterpillars" and adds, "Unhealthy, infected by Beetles. Solution: Apply Bacillus thuringiensis (Bt) or hand-pick them."
5. Results Shown: The User Interface Layer shows you this text on-screen.
6. QR Code Made: The QR Code Generator turns the result into a QR code, which the UI displays.
7. User Share: Users send the QR code (exp through WhatsApp) to an Agricultural Expert, who scans it to see the details.

3.3 Use Case Diagram

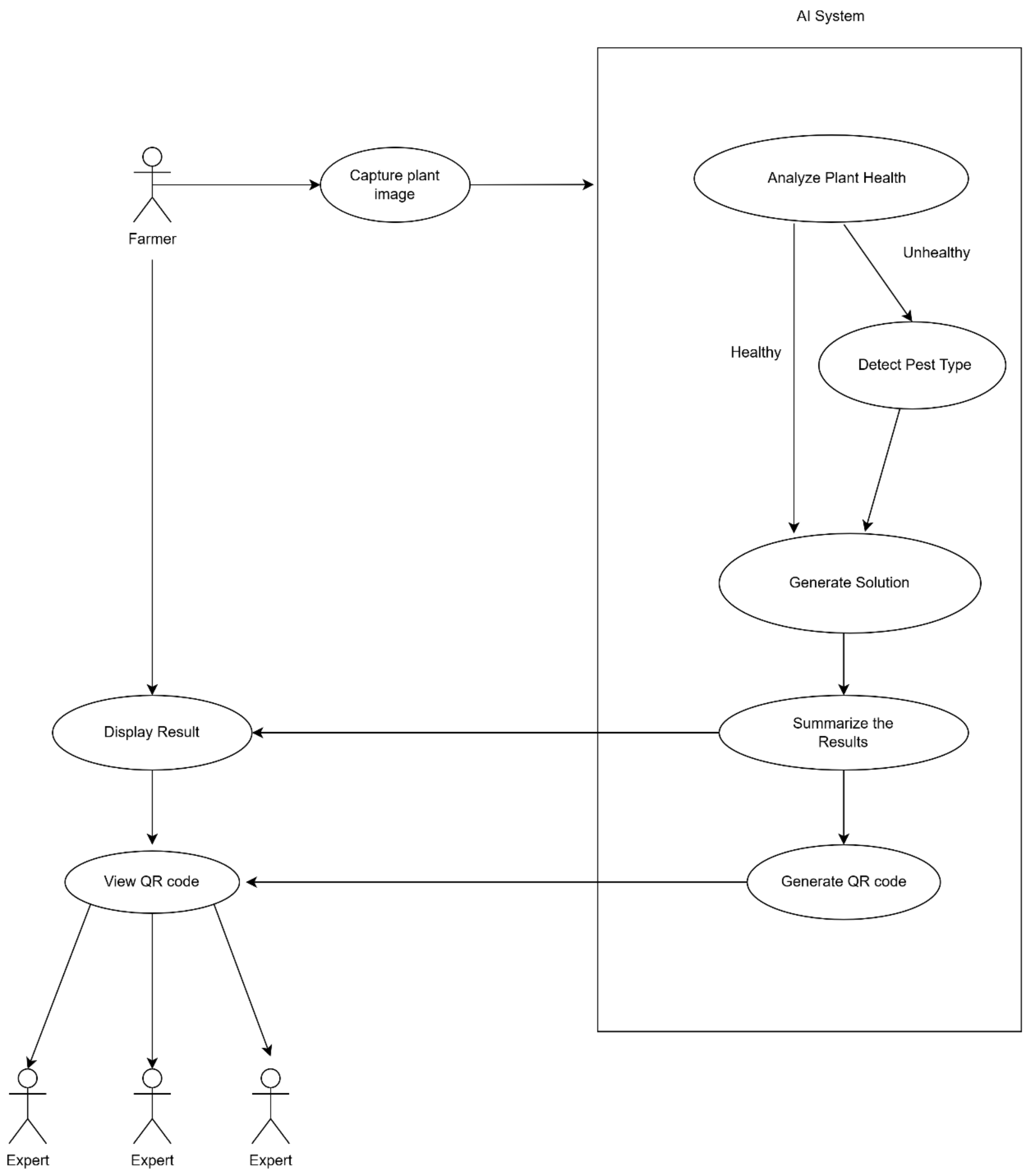


Figure 3.3 User Case Diagram

Based on Figure 3.4, it shows the use case diagram of the AI for pest Detection and outlines how the farmer as known as the user interact with the system. The farmer would capture the image of the plant. Then, the data would be sent to the AI System to analyze whether the plant is healthy or unhealthy. Then, the data would be sent to the solution generator. If the plant is healthy, the solution generator will generate "No action needed". If the plant is detected unhealthy, the AI identifies the pest (whiteflies or caterpillars) and generates a solution. After that, these results will be summarized and displayed to the farmer. Thus, a QR will be generated to the user for data sharing purposes or avoid confusion due to having many plants in a large farm.

3.3.1 Use Case Description

Use Case 1: Capture Plant Image

Use Case ID	UC1	**Use Case Name**	Capture Plant Image
Actors		Farmer/User	
Purpose		To allow the user to take a photo of a plant for health and pest analysis.	
Preconditions		The Android app is installed and running on a device with a functional camera; the user has granted camera permissions.	
Basic Flow: 1) The Farmer/User opens the app on their Android device. 2) The user selects the "Capture Image" option. 3) The app activates the device's camera. 4) The user points the camera at the plant and takes a photo. 5) The app saves the image and passes it to the AI System for analysis.			
Postconditions		A plant image is captured and ready for analysis by the AI System.	
Exceptions		• A plant image is captured and ready for analysis by the AI System. • If permissions are denied, the app prompts the user to enable them.	

Table 3.3.1 Use Case Description for "Capture Image" Use Case

Use Case 2: Analyze Plant Health

Use Case ID	UC2	**Use Case Name**	Analyze Plant Health
Actors		AI System (invoked by Farmer/User)	
Purpose		To determine whether the plant in the captured image is healthy or unhealthy.	
Preconditions		A plant image has been captured and provided by the Farmer/User.	
Basic Flow: 1) The AI System receives the plant image from the app. 2) The system preprocesses the image (resizes 224x224, normalizes pixel values). 3) The AI model (YOLOv5-based) analyzes the image. 4) The system outputs a classification: "Healthy" or "Unhealthy" with a confidence score			
Postconditions		The plant’s health status is determined and ready for further processing if unhealthy.	
Exceptions		If the image is blurry or invalid, the system returns an error message to the app.	

Table 3.3.2 Use Case Description for “Analyze Plant Health” Use Case

Use Case 3: Detect Pest Type

Use Case ID	UC3	**Use Case Name**	Detect Pest Type
Actors		AI System (invoked by Analyze Plant Health)	
Purpose		To identify the specific pest (whiteflies or caterpillars) affecting an unhealthy plant	
Preconditions		The plant has been classified as "Unhealthy" by the Analyze Plant Health use case.	
Basic Flow: 1) The AI System takes the "Unhealthy" classification and the original image. 2) The system reanalyzes the image, focusing on pest-specific features. 3) The AI model outputs the pest type: "Whiteflies" or "Caterpillars" with a confidence score.			
Postconditions		The pest type is identified and ready for solution generation.	
Exceptions		If the pest cannot be confidently identified (exp: low confidence), the system flags it as "Unknown Pest."	

Table 3.3.3 Use Case Description for "Detect Pest Type" Use Case

Use Case 4: Generate Solution

Use Case ID	UC4	**Use Case Name**	Generate Solution
Actors		AI (invoked by Detect Pest Type or Analyze Plant Health if detected plant is healthy)	
Purpose		To provide a pest-specific solution for the identified pest.	
Preconditions		The pest type (whiteflies or caterpillars) has been identified.	
Basic Flow: 1) The AI System receives the pest type from the Detect Pest Type use case. 2) The system maps the predefined solution based on these 2 categories: a) For the plant that identified as healthy, the system will display "No action needed". b) For the plant that identified as unhealthy, the system maps the pest to a predefined solution: • Whiteflies: "Use neem oil or insecticidal soap." • Caterpillars: " Apply Bacillus thuringiensis (Bt) or hand-pick them." 3) The solution text is generated.			
Postconditions		A solution is prepared for display to the Farmer/User.	
Exceptions		If the pest is "Unknown," a generic solution (exp: "Consult an expert") is provided.	

Table 3.3.4 Use Case Description for "Generate Solution" Use Case

Use Case 5: Summarize the Results

Use Case ID	UC5	**Use Case Name**	Summarize the Results
Actors		AI (invoked by Detect Pest Type, Analyze Plant Health and Generate Solution)	
Purpose		To provide a detailed result	
Preconditions		The AI System has completed health analysis, pest detection (if applicable), and solution generation.	
Basic Flow: 1) The AI system collect data such as: a) The status of the plant from Analyze Plant Health b) The type of pest that has been detected for unhealthy plants from Detect Pest Type c) The solution that has been generated from Generate Solution. 2) The data would be summarized and ready to be displayed to the Farmers/User.			
Postconditions		A summary of the results will be generated	
Exceptions		If the results are incomplete, partial data would be summarized or show error.	

Table 3.3.5 Use Case Description for "Summarize the Results" Use Case

Use Case 6: Display Results

Use Case ID	UC6	**Use Case Name**	Display Results
Actors		AI (invoked by Summarize the Results)	
Purpose		To allow the user to view the analysis results on the app interface	
Preconditions		The summary of the results has been generated and completed in the AI System.	
Basic Flow: 1) The app receives the results from the AI System (health status, pest type, solution, confidence). 2) The app displays the results in a user-friendly format (e.g., text fields or labels). 3) The user reviews the information on the screen.			
Postconditions		The Farmer/User is informed of the plant's status and recommended actions.	
Exceptions		If results are incomplete, the app shows an error or partial data.	

Table 3.3.6 Use Case Description for "Display Results" Use Case

Use Case 7: Generate QR Code

Use Case ID	UC7	**Use Case Name**	Generate QR Code
Actors		AI (invoked by Summarize the Results)	
Purpose		To encode the analysis results in a QR code for easy sharing or storage.	
Preconditions		The analysis results (health status, pest type, solution, confidence) are available.	
Basic Flow: 1) The AI System receives the results from the previous use cases. 2) The system formats the data into a string • Result: The plant is unhealthy, infected by whiteflies. Solution: Use neem oil or insecticidal soap. Confidence: 92.27%			
Postconditions		A QR code is created and ready for display.	
Exceptions		If the data is too large, the system truncates it to fit QR code limits.	

Table 3.3.7 Use Case Description for "Generate QR Code" Use Case

Use Case 8: View QR Code

Use Case ID	UC8	**Use Case Name**	View QR Code
Actors		Farmer/User, Expert	
Purpose		To allow the user to view and interact with the generated QR code and share with experts (optional).	
Preconditions		The QR code has been generated by the AI System.	
Basic Flow: 1) The app displays the QR code on the screen. 2) The Farmer/User views the QR code. 3) The user can scan it with another device or save or share to the experts (optional).			
Postconditions		The Farmer/User has access to the QR code for further use.	
Exceptions		If the QR code fails to display, the app shows an error message.	

Table 3.3.8 Use Case Description for "View QR Code" Use Case

3.4 Activity Diagram

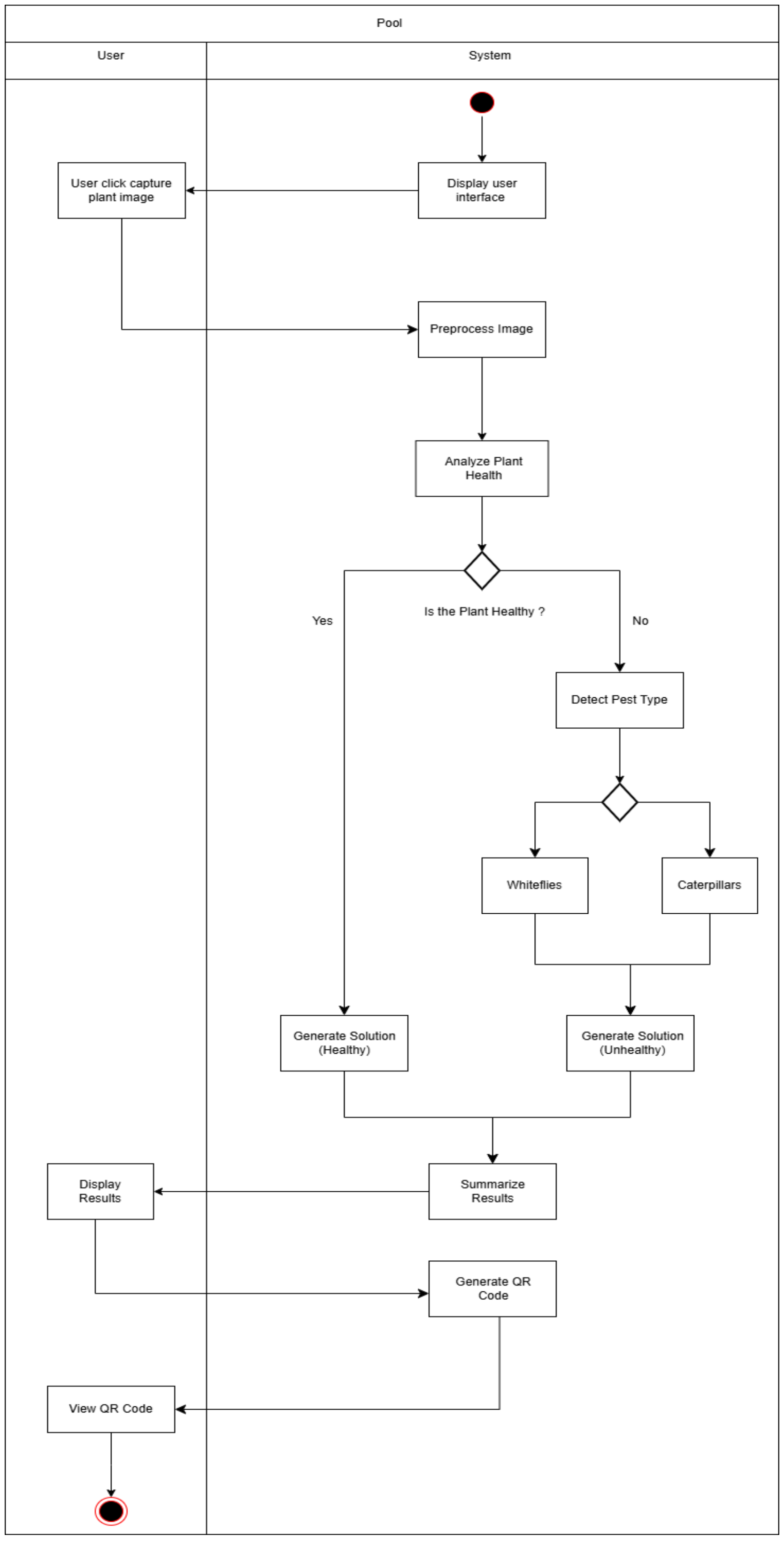


Table 3.4 Activity Diagram for the AI model and proposed mobile application

Table 3.4 shows the activity diagram of the system which provides details about the interaction between the user and the system in a simplified procedure. The procedure begins when the user who is gardener or farmer which represents by the dark circle clicks the “Capture Plant Image” button in the mobile app through the interface which build in the Android Studio. The system will respond by displaying the user interface which includes a camera view for capturing the plant image. This is handled by the app’s Camera Module using Android’s CameraX API. Once the image is captured, the system preprocesses it. This involves the process of resizing the image to 224x224 pixels which is required by YOLOv5 and normalizing pixel values (exp: scaling from 0-255 to 0-1) to prepare it for AI analysis. Then, the system uses the YOLOv5 model through TensorFlow Lite to analyze the preprocessed image and classify the plant as "Healthy" or "Unhealthy." This step analyzes the plant’s health status with a confidence score. If the plant is classified as healthy, the system proceeds directly to generating a solution for a healthy plant which is “No Solution”. If the plant is detected as unhealthy, the system will move to the next step which identify the pest to identify either the pest is “Whiteflies” or ‘Caterpillars’. Then the system generates a tailored solution, such as "Use neem oil or insecticidal soap" for Whiteflies or "Apply Bacillus thuringiensis (Bt) or hand-pick them" for Caterpillars. Then, the system will summarize the results by compiling the plant’s health status, pest type, solution and confidence score such as:

- Plant Status: Unhealthy (Whiteflies)
- Pest: Whiteflies
- Solution: Use neem oil or insecticidal soap
- Confidence: 60.09%

Then, these results will be encoded into a QR code using the ZXing library for sharing purposes. Finally, the system displays the results on the app’s interface for the user to review alongside with the QR code for the user to view or share to other people like agricultural experts through various platforms.

3.5 Project Timeline

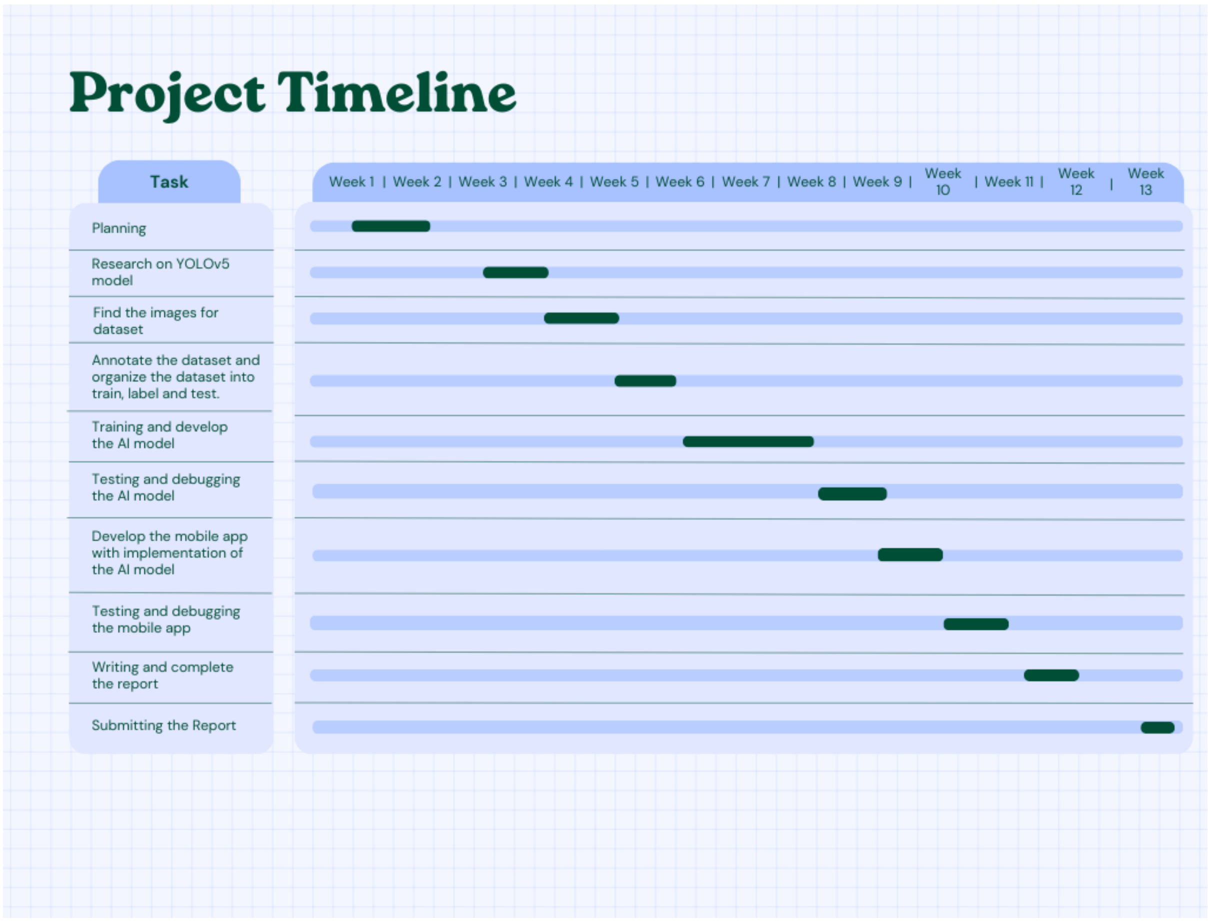


Figure 3.4 Project Timeline

Chapter 4

System Design

4.1 System Block Diagram

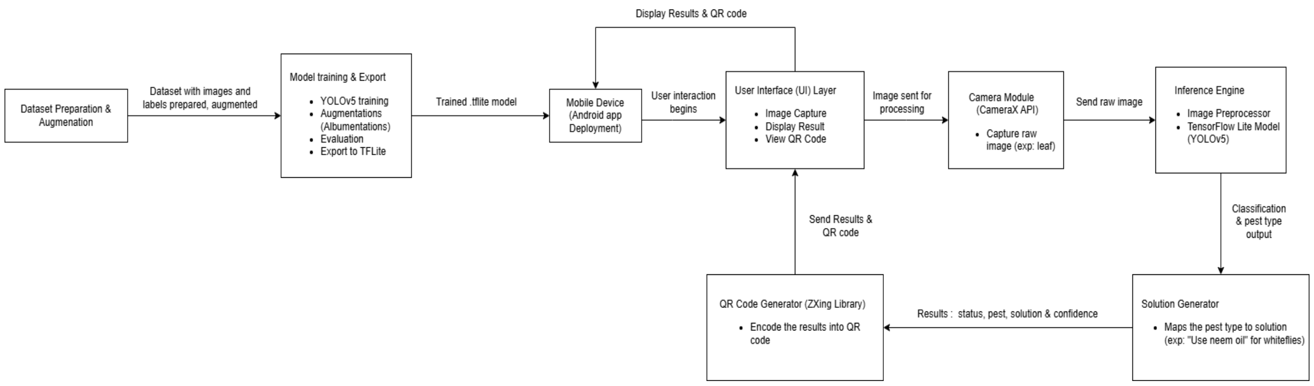

Display Results & QR code
Dataset Preparation & Augmenation
Dataset with images and labels prepared, augmented
Model training & Export
• YOLOv5 training
• Augmentations (Albumentations)
• Evaluation
• Export to TFLite
Trained .tflite model
Mobile Device (Android app Deployment)
User interaction begins
User Interface (UI) Layer
• Image Capture
• Display Result
• View QR Code
Image sent for processing
Camera Module (CameraX API)
• Capture raw image (exp: leaf)
Send raw image
Inference Engine
• Image Preprocessor
• TensorFlow Lite Model (YOLOv5)
Classification & pest type output
Send Results & QR code
QR Code Generator (ZXing Library)
• Encode the results into QR code
Results : status, pest, solution & confidence
Solution Generator
• Maps the pest type to solution (exp: "Use neem oil" for whiteflies)

Figure 4.1 System Block Diagram

The system block diagram for the AI Pest Detection includes both training pipeline and mobile app deployment by illustrating the flow of data from dataset preparation to the user interaction. It begins with the dataset preparation and augmentation phase where main_script.py will organize the dataset of approximately 859 images into training, validation, and testing sets and apply augmentations like Gaussian noise and color jitter by using the albumentations library to enhance model robustness and saving the results in augmented directories. This stage contributes to the Model Training & Export component which trains a YOLOv5 model for 100 epochs, evaluating it for >85% accuracy and exporting it to TensorFlow Lite (best.tflite). The model will be implemented in the Mobile App on an Android device where the User Interface (UI) Layer (build with Android Studio) enables image capture, result display and QR code viewing. The Camera Module (CameraX API) captures the plant image and then send it to the Inference Engine which preprocesses the image (resize to 224x224, normalize) and uses the YOLOv5 .tflite model to classify the plant as Healthy or Unhealthy and detect pests whether it is Whiteflies or Caterpillars. The Solution Generator will then generate the solutions (exp: "Use neem oil" for Whiteflies) After that, the QR Code Generator (ZXing library) will encode the results in form of status, pest, solution, confidence into a QR code. Finally, the results and the QR code will display through the UI Layer for the user.

4.2 System Components Specifications

4.2.1 Hardware

The hardware that is required for the development of this project are a processor, ethernet connection, storage (SSD), memory (RAM) and a graphic card. The processor function by increasing the performance of the computer. The graphic card is used to display quality images. The SSD used to increase the storage to store data and RAM is used to read data and increase the performance of the computer. The specifications of the laptop model used by the developer to develop this project is stated in the table below.

Description	Specifications
Model	ROG Strix G513IC_G513IC
Processor	AMD Ryzen 7 4800H
Operating System	Windows 10
Graphic	NVIDIA GeForce RTX 3050
Memory	40GB DDR4 RAM
Storage	INTEL SSDPEKNU512GZ 477GB

Table 4.2.1.1 Specifications of laptop

The developer also uses a mobile device as the deployment environment for the app. The mobile device will run the Android app that integrates with the trained .tflite model for the on-device pest detection. The specification of the mobile device is stated in the table below.

Description	Specifications
Model	Huawei Nova 4
Processor	HiSilicon Kirin 970
Operating System	Android 9.0
Resolution	2310 x 1080
Memory	8 GB RAM
Storage	128 GB

Table 4.2.1.2 Specifications of mobile device

4.2.2 Software

The software requirements that been showed are coding, database and operating system. As for the coding part, Android Studio 2023 were used to build the mobile payment application. The software that used to store the user data is Firebase RealtimeDatabase. The AI model would be trained by using Jupyter Notebook by using python programming language.

Specifications	Description
Android Studio 2024	Use for coding purposes to build the mobile payment app.
Jupyter Notebook	Use to train AI model
Storage	Minimum 8GB of available disk space
Display Resolution	1280*800
RAM	8GB

Table 4.2.2 Specifications of software

4.2.3 Software Setup for Deployment

There is various software that need to be installed in the laptop for the development of this AI model.

4.2.3.1 Python Environment

- Python version: 3.12.4 (required by YOLOv5)
- Library:
  1. Torch: For YOLOv5 model training and inference
  2. Cv2: For image processing such as reading, augmenting and saving images
  3. Albumentations: For image augmentation such as Gaussian noise and color jitter.
  4. Numpy: For data handling, evaluation and visualization
  5. Pandas: For data handling, evaluation and visualization
  6. Matplotlib: For data handling, evaluation and visualization
  7. Sklearn: For calculating accuracy and classification reports.

8. Shutil: For file operations and result saving.
9. Glob: For file operations and result saving.
10. Json: For file operations and result saving.

4.2.3.2 Jupyter Notebook

The Jupyter Notebook is used to execute the main_script.py for training and evaluation purposes. There are minimum requirements of this Jupyter Notebook.

<u>Minimum requirements</u>:

- Storage: 8GB of disk space
- Display Resolution: 1280x800 for visualization of results in Jupyter Notebook
- RAM: 8GB (minimum for running Jupyter and Python scripts)

4.2.3.3 YOLOv5 Repository

The purpose of this YOLOv5 is to facilitates the dataset augmentation, model training (100 epochs), evaluation and export to .tflite format.

- Cloned from https://github.com/ultralytics/yolov5.git.
- Dependencies installed through pip install -r requirements.txt that includes torch , torchvision and others.

4.2.4 System Components for Mobile App

There are various components which are part of the Android app.

4.2.4.1 User Interface (UI) Layer

- Framework: Android Studio Meerkat 2024
- Programming Languages: Java/Kotlin
- Components:
    1. Image Capture Interference: Button to trigger the camera (exp: Capture Image)
    2. Result Display: Show results with text views

       Example of the results:

       Plant Status: Healthy

Pest: None

Solution: No action needed

Confidence: 0.00%

3. QR Code View: Display the generated QR Code that contains the results.

4.2.4.2 Camera Module

- API: CameraX (Android Jetpack library for camera access)
- Function: To capture raw images of the plant
- Requirement: Rear camera of the android smartphone camera permission granted by the user

4.2.4.3 Inference Engine

There are few sub-components in the inference engine.

- Image Preprocessor: Its function is to resize images to 224x224 pixels and normalizes pixel value (0-1). It handles by TensoFlow's Lite preprocessing that utilities within the app.
- TensorFlow Lite Model (YOLOv5):
    1. Model File: best.tflite which generated by main.script.py
    2. Input: Preprocessed images (224x224)
    3. Outputs: Classification of Healthy or Unhealthy, pest type (Whiteflies or Caterpillars) with confidence score.
    4. Framework: TensorFlow Lite
    5. Accuracy: 87.31%

4.2.4.4 Solution Generator

- Function: Maps detection results to solutions using a hardcoded lookup table (exp: HashMap in Kotlin)
- Mappings:
    1. Healthy: “No action needed”
    2. Whiteflies: “Use neem oil or insecticidal soap”

3. Caterpillars: “Apply Bacillus thuringiensis (Bt) or hand-pick them”
4. Unknown: “Consult a local expert”

- Implementation: Coded with Java/Kotlin

4.2.4.5 QR Code Generator

- Library: ZXing (Zebra Crossing) library for QR code generator
- Function: Encodes the detection results (status, pest, solution, confidence) into a QR code.
- Output: QR code will be displayed on the UI

4.2.4.6 Dataset Specifications

- Dataset: Custom dataset for pest detection with annotation through Roboflow
- Size: 859 images where 448 images for training, 277 images for validation and 134 images for testing.
- Classes: 3 categories which are healthy, whiteflies and caterpillars.
- Format:
    1. Images in form of jpg, JPEG, Ng will be stored in a folder with images/train, images/val and images/test.
    2. Labels in form of text file will be stored in in a folder with labels/train, labels/val and labels/test.

4.3 Model Selection and Architecture

In this project, the YOLOv5 model was chosen due to its balance of efficiency, speed, and accuracy, making it ideal for real-time detection on resource-constrained mobile devices.

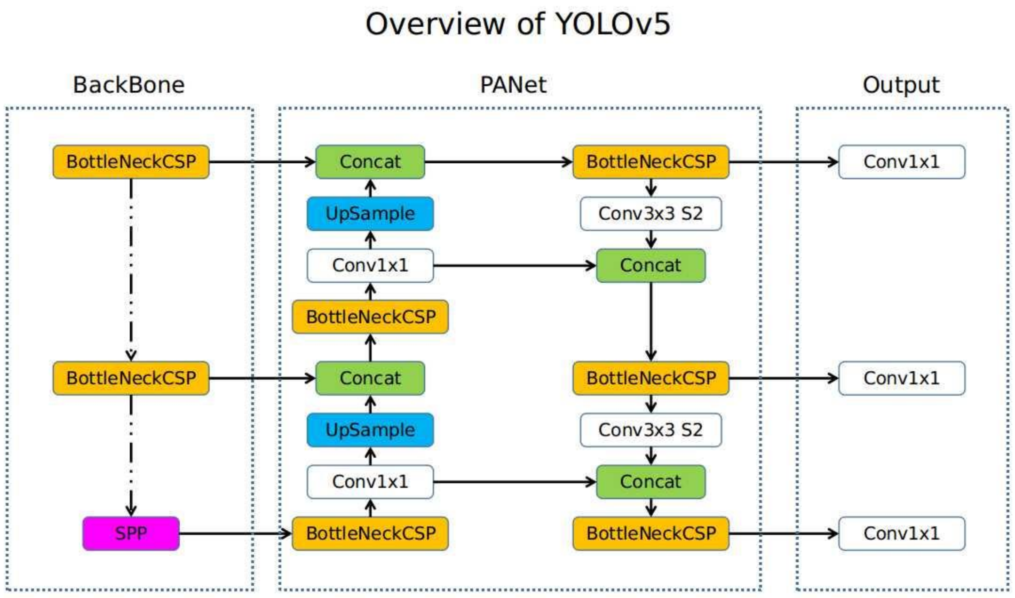
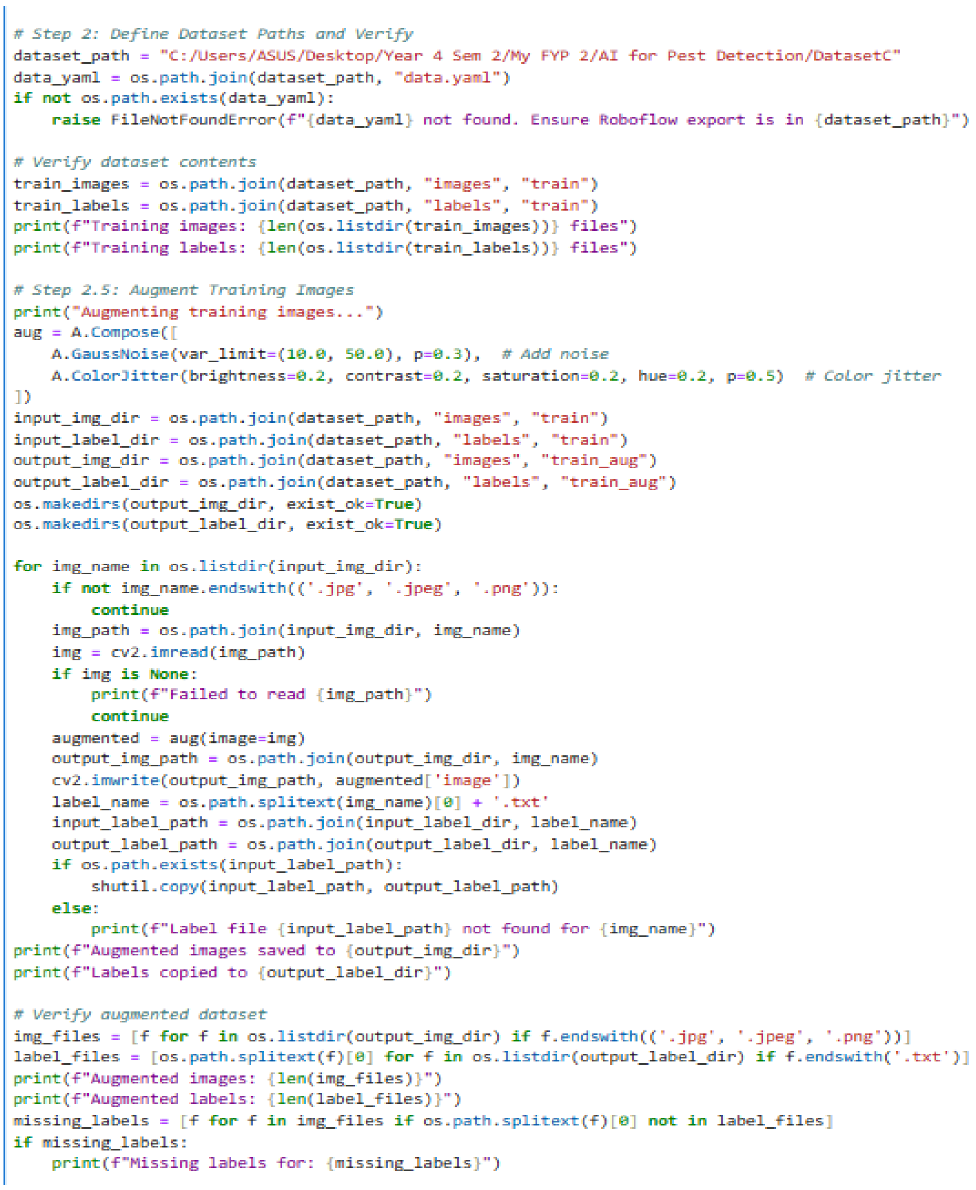


Figure 4.2 YOLOv5 Model Structure

Figure 4.2 shows the YOLOv5s architecture in the AI pest detection project is a three-stage framework specifically designed for efficient, multi-scale object detection, comprising the Backbone, PANet, and Output layers to identify Healthy plants, Whiteflies, and Caterpillars in farm settings. The Backbone, featuring BottleNeckCSP modules and an SPP block which extracts hierarchical features from a 640x640 RGB input image by down sampling through resolutions (640x640 to 20x20) by using stride-2 convolutions and cross-stage connections. With the SPP block applying multi-scale max-pooling (exp: 5x5, 9x9, 13x13 kernels) at the 20x20 resolution (1,475,712 parameters) to capture features for small pests like whiteflies and caterpillars and producing feature maps with increasing channels (exp: 48 to 768) for semantic richness. The PANet aggregates these features through a bottom-up path (additional BottleNeckCSP and Conv3x3 S2 layers for downsampling) and a top-down path (bilinear upsampling, Conv1x1 for channel adjustment and concatenation at scales like 20x20, 40x40, 80x80.Then, it will combine high-level semantic information with low-level spatial details to enhance detection across scales which can be seen in the model summary with tf_upsample and tf_concat The Output layers apply Conv1x1 operations at each scale to generate predictions with shapes like (1, 80, 80, 24) (3 anchors, 5 box coordinates, 3 classes), producing bounding boxes, class probabilities, and confidence scores, followed by NMS (confidence threshold 0.1 in detect_pest_status) to filter overlapping detections.

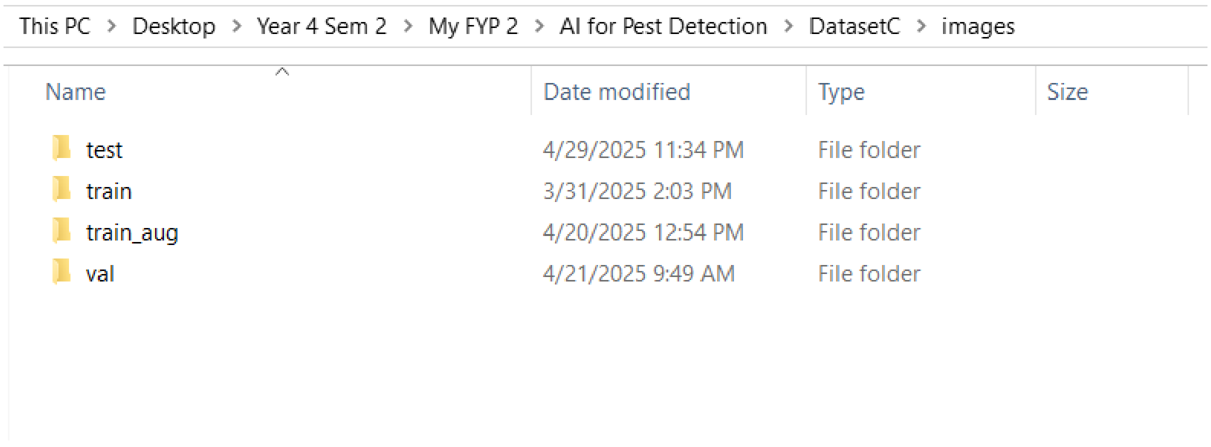
4.4 Data Preprocessing



```python
# Step 2: Define Dataset Paths and Verify
dataset_path = "C:/Users/ASUS/Desktop/Year 4 Sem 2/My FYP 2/AI for Pest Detection/DatasetC"
data_yaml = os.path.join(dataset_path, "data.yaml")
if not os.path.exists(data_yaml):
    raise FileNotFoundError(f"{data_yaml} not found. Ensure Roboflow export is in {dataset_path}")

# Verify dataset contents
train_images = os.path.join(dataset_path, "images", "train")
train_labels = os.path.join(dataset_path, "labels", "train")
print(f"Training images: {len(os.listdir(train_images))} files")
print(f"Training labels: {len(os.listdir(train_labels))} files")

# Step 2.5: Augment Training Images
print("Augmenting training images...")
aug = A.Compose([
    A.GaussNoise(var_limit=(10.0, 50.0), p=0.3),  # Add noise
    A.ColorJitter(brightness=0.2, contrast=0.2, saturation=0.2, hue=0.2, p=0.5)  # Color jitter
])
input_img_dir = os.path.join(dataset_path, "images", "train")
input_label_dir = os.path.join(dataset_path, "labels", "train")
output_img_dir = os.path.join(dataset_path, "images", "train_aug")
output_label_dir = os.path.join(dataset_path, "labels", "train_aug")
os.makedirs(output_img_dir, exist_ok=True)
os.makedirs(output_label_dir, exist_ok=True)

for img_name in os.listdir(input_img_dir):
    if not img_name.endswith(('.jpg', '.jpeg', '.png')):
        continue
    img_path = os.path.join(input_img_dir, img_name)
    img = cv2.imread(img_path)
    if img is None:
        print(f"Failed to read {img_path}")
        continue
    augmented = aug(image=img)
    output_img_path = os.path.join(output_img_dir, img_name)
    cv2.imwrite(output_img_path, augmented['image'])
    label_name = os.path.splitext(img_name)[0] + '.txt'
    input_label_path = os.path.join(input_label_dir, label_name)
    output_label_path = os.path.join(output_label_dir, label_name)
    if os.path.exists(input_label_path):
        shutil.copy(input_label_path, output_label_path)
    else:
        print(f"Label file {input_label_path} not found for {img_name}")
print(f"Augmented images saved to {output_img_dir}")
print(f"Labels copied to {output_label_dir}")

# Verify augmented dataset
img_files = [f for f in os.listdir(output_img_dir) if f.endswith(('.jpg', '.jpeg', '.png'))]
label_files = [os.path.splitext(f)[0] for f in os.listdir(output_label_dir) if f.endswith('.txt')]
print(f"Augmented images: {len(img_files)}")
print(f"Augmented labels: {len(label_files)}")
missing_labels = [f for f in img_files if os.path.splitext(f)[0] not in label_files]
if missing_labels:
    print(f"Missing labels for: {missing_labels}")
```

Figure 4.3 Data Preprocessing Code snippet

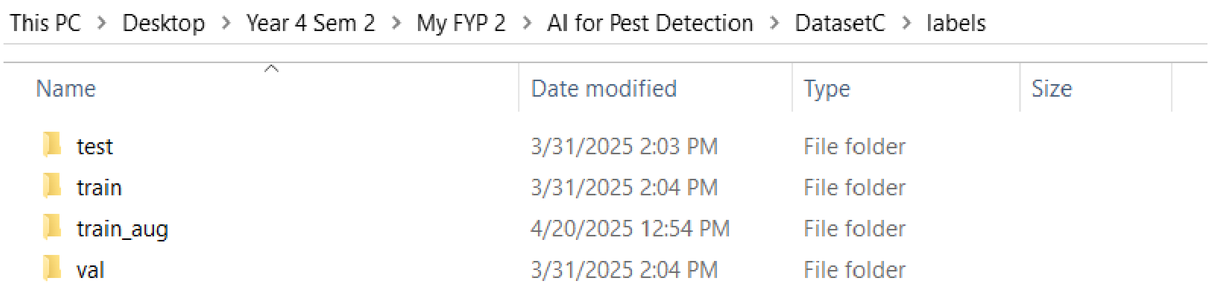
Based on Figure 4.3, the data processing focuses on preparing the training dataset by applying augmentation to enhance model robustness. The process begins by defining the dataset path as C:/Users/ASUS/Desktop/Year 4 Sem 2/My FYP 2/AI for Pest Detection/DatasetC and verifying the existence of training images (448 images) and labels in images/train and labels/train directories, ensuring the dataset is correctly structured for YOLOv5 training. The preprocessing then involves augmenting the training images using the albumentations library, where a pipeline applies Gaussian noise with a variance limit of 10.0 to 50.0 (30% probability) to simulate real-world imperfections and color jitter with ±20% adjustments to brightness, contrast, saturation, and hue (50% probability) to mimic varying lighting conditions, processing each image by reading it with cv2.imread, applying the augmentations, and saving the results to images/train_aug. These labels will be copied and remain unchanged to labels/train_aug since these augmentations do not affect bounding box coordinates and the process concludes with a verification step confirming 448 augmented images and labels, flagging any missing labels to ensure dataset integrity. While this code handles augmentation as a preprocessing step for training, it lacks explicit resizing and normalization, which are managed by YOLOv5's data loader during training (resizing to 640x640 and normalizing to [0, 1]) and does not address inference preprocessing requirements like resizing to 224x224 for the mobile app, which are handled separately in the Inference Engine.



This PC > Desktop > Year 4 Sem 2 > My FYP 2 > AI for Pest Detection > DatasetC > images

Name	Date modified	Type	Size
test	4/29/2025 11:34 PM	File folder	
train	3/31/2025 2:03 PM	File folder	
train_aug	4/20/2025 12:54 PM	File folder	
val	4/21/2025 9:49 AM	File folder	

Figure 4.4 Files in the images folder for Dataset

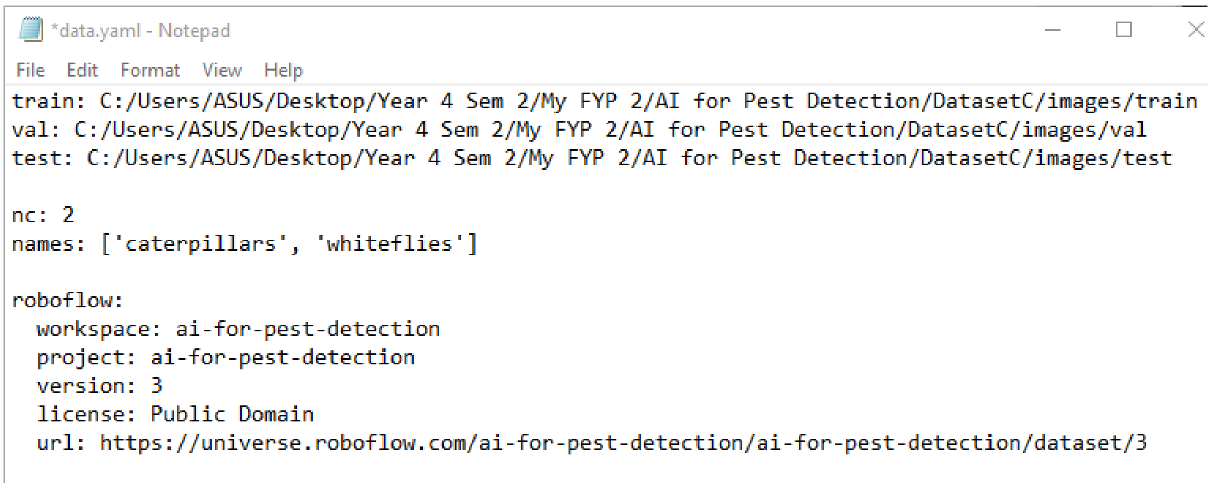


This PC > Desktop > Year 4 Sem 2 > My FYP 2 > AI for Pest Detection > DatasetC > labels

Name	Date modified	Type	Size
test	3/31/2025 2:03 PM	File folder	
train	3/31/2025 2:04 PM	File folder	
train_aug	4/20/2025 12:54 PM	File folder	
val	3/31/2025 2:04 PM	File folder	

Figure 4.5 Files in the labels folder for Dataset

Based on Figure 4.4, there are various folders in the images folder in the dataset. As for the folder test, train and val , the folder is created based on YOLOv5, as training the AI model with YOLOv5 model requires the dataset to be separated into 3 categories which are testing, training and validation. As for the train_aug, this folder will be generated by the model while training the model for storing the augmentation images in it. This process sames goes to the folders in the labels as shown as in Figure 4.5. The only difference is, the images folder contains images meanwhile the labels folders contain txt files.

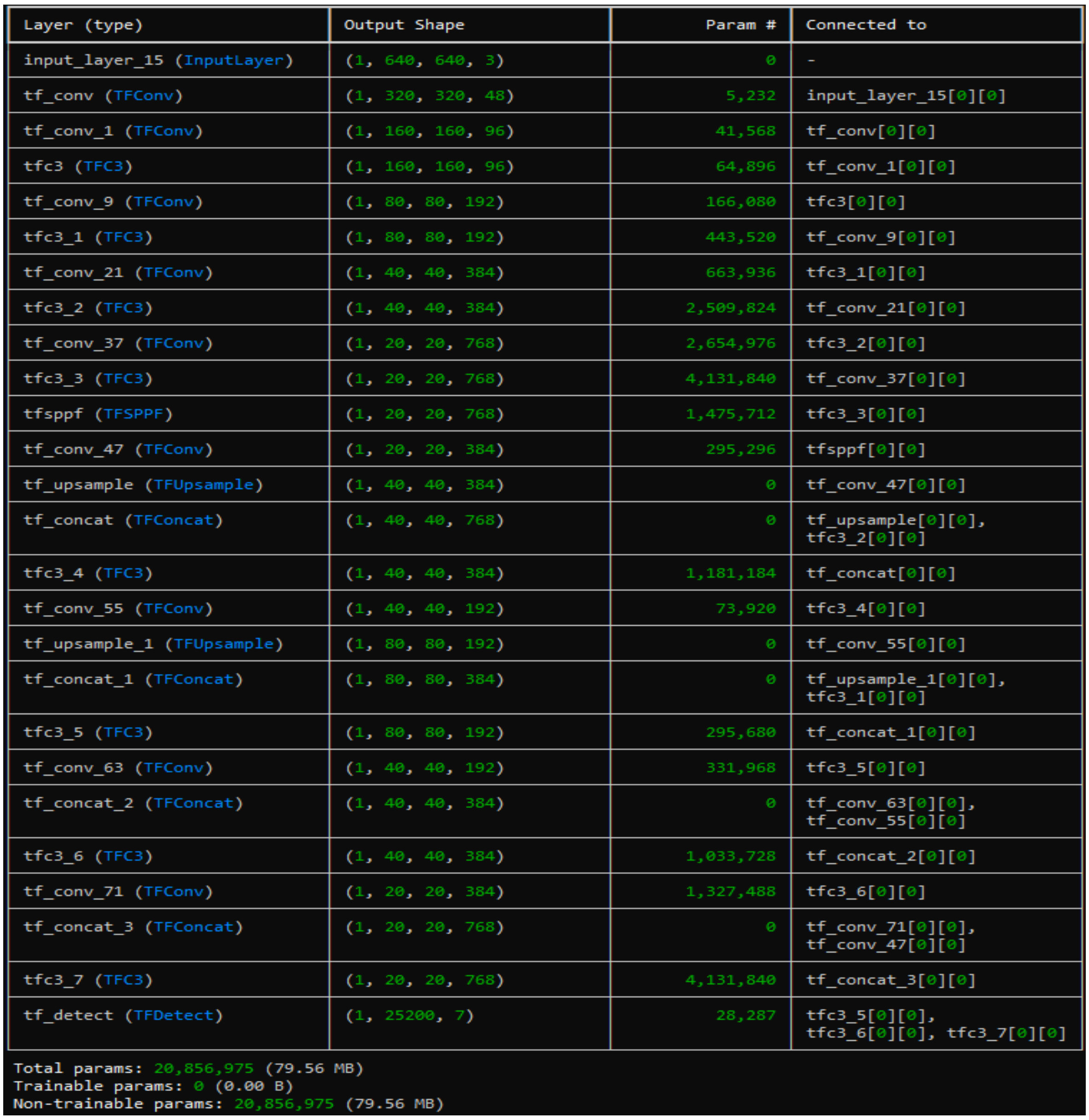


```
train: C:/Users/ASUS/Desktop/Year 4 Sem 2/My FYP 2/AI for Pest Detection/DatasetC/images/train
val: C:/Users/ASUS/Desktop/Year 4 Sem 2/My FYP 2/AI for Pest Detection/DatasetC/images/val
test: C:/Users/ASUS/Desktop/Year 4 Sem 2/My FYP 2/AI for Pest Detection/DatasetC/images/test

nc: 2
names: ['caterpillars', 'whiteflies']

roboflow:
  workspace: ai-for-pest-detection
  project: ai-for-pest-detection
  version: 3
  license: Public Domain
  url: https://universe.roboflow.com/ai-for-pest-detection/ai-for-pest-detection/dataset/3
```

Figure 4.6 data.yaml file

Figure 4.6 shows the data.yaml which is a configuration file used in the AI pest detection project to define the dataset structure and metadata for training the YOLOv5s model, facilitating the training, validation, and testing processes and indicating the dataset was sourced from Roboflow which is a platform for dataset management, which likely handled initial annotation and splitting. This file serves as a critical link between the dataset and the YOLOv5 training pipeline to ensure the model correctly loads and processes the data for pest detection

4.5 Model Training and Tuning

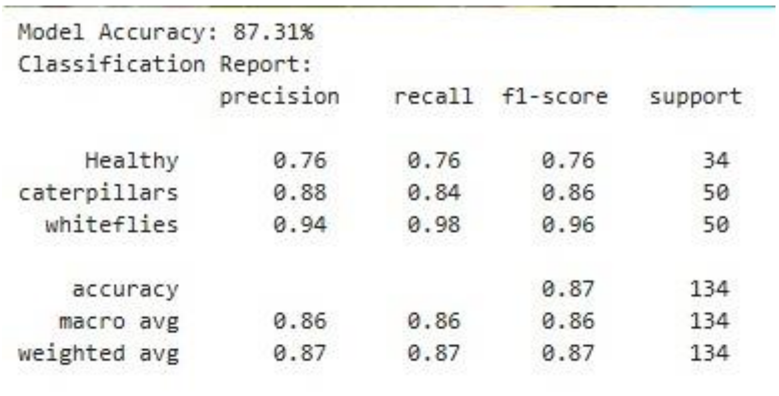


```
┃ Layer (type)                     ┃ Output Shape           ┃     Param # ┃ Connected to            ┃
│ input_layer_15 (InputLayer)      │ (1, 640, 640, 3)       │           0 │ -                       │
│ tf_conv (TFConv)                 │ (1, 320, 320, 48)      │       5,232 │ input_layer_15[0][0]    │
│ tf_conv_1 (TFConv)               │ (1, 160, 160, 96)      │      41,568 │ tf_conv[0][0]           │
│ tfc3 (TFC3)                      │ (1, 160, 160, 96)      │      64,896 │ tf_conv_1[0][0]         │
│ tf_conv_9 (TFConv)               │ (1, 80, 80, 192)       │     166,080 │ tfc3[0][0]              │
│ tfc3_1 (TFC3)                    │ (1, 80, 80, 192)       │     443,520 │ tf_conv_9[0][0]         │
│ tf_conv_21 (TFConv)              │ (1, 40, 40, 384)       │     663,936 │ tfc3_1[0][0]            │
│ tfc3_2 (TFC3)                    │ (1, 40, 40, 384)       │   2,509,824 │ tf_conv_21[0][0]        │
│ tf_conv_37 (TFConv)              │ (1, 20, 20, 768)       │   2,654,976 │ tfc3_2[0][0]            │
│ tfc3_3 (TFC3)                    │ (1, 20, 20, 768)       │   4,131,840 │ tf_conv_37[0][0]        │
│ tfsppf (TFSPPF)                  │ (1, 20, 20, 768)       │   1,475,712 │ tfc3_3[0][0]            │
│ tf_conv_47 (TFConv)              │ (1, 20, 20, 384)       │     295,296 │ tfsppf[0][0]            │
│ tf_upsample (TFUpsample)         │ (1, 40, 40, 384)       │           0 │ tf_conv_47[0][0]        │
│ tf_concat (TFConcat)             │ (1, 40, 40, 768)       │           0 │ tf_upsample[0][0],      │
│                                  │                        │             │ tfc3_2[0][0]            │
│ tfc3_4 (TFC3)                    │ (1, 40, 40, 384)       │   1,181,184 │ tf_concat[0][0]         │
│ tf_conv_55 (TFConv)              │ (1, 40, 40, 192)       │      73,920 │ tfc3_4[0][0]            │
│ tf_upsample_1 (TFUpsample)       │ (1, 80, 80, 192)       │           0 │ tf_conv_55[0][0]        │
│ tf_concat_1 (TFConcat)           │ (1, 80, 80, 384)       │           0 │ tf_upsample_1[0][0],    │
│                                  │                        │             │ tfc3_1[0][0]            │
│ tfc3_5 (TFC3)                    │ (1, 80, 80, 192)       │     295,680 │ tf_concat_1[0][0]       │
│ tf_conv_63 (TFConv)              │ (1, 40, 40, 192)       │     331,968 │ tfc3_5[0][0]            │
│ tf_concat_2 (TFConcat)           │ (1, 40, 40, 384)       │           0 │ tf_conv_63[0][0],       │
│                                  │                        │             │ tf_conv_55[0][0]        │
│ tfc3_6 (TFC3)                    │ (1, 40, 40, 384)       │   1,033,728 │ tf_concat_2[0][0]       │
│ tf_conv_71 (TFConv)              │ (1, 20, 20, 384)       │   1,327,488 │ tfc3_6[0][0]            │
│ tf_concat_3 (TFConcat)           │ (1, 20, 20, 768)       │           0 │ tf_conv_71[0][0],       │
│                                  │                        │             │ tf_conv_47[0][0]        │
│ tfc3_7 (TFC3)                    │ (1, 20, 20, 768)       │   4,131,840 │ tf_concat_3[0][0]       │
│ tf_detect (TFDetect)             │ (1, 25200, 7)          │      28,287 │ tfc3_5[0][0],           │
│                                  │                        │             │ tfc3_6[0][0], tfc3_7[0][0] │
 Total params: 20,856,975 (79.56 MB)
 Trainable params: 0 (0.00 B)
 Non-trainable params: 20,856,975 (79.56 MB)
```

Figure 4.7 Model Architecture

Figure 4.7 shows the summary of the YOLOv5s architecture that is used in the AI pest detection project, detailing its layers, output shapes, parameters, and connections. The model starts with an input layer accepting 640x640 RGB images (1, 640, 640, 3) followed by a Backbone with convolutional (TFConv) and Cross-Stage Partial (TFC3) layers that downsample to 20x20 (tf_conv_37, (1, 20, 20, 768)) that extract features for pest detection. An SPPF block (tfsppf, 1,475,712 parameters) captures multi-scale features and the Feature Pyramid Network (FPN) uses upsampling (TFUpsample) and concatenation (TFConcat) to aggregate features at scales like 20x20, 40x40, and 80x80 (tfc3_5, (1, 80, 80, 192)). The final detection layer (tf_detect, (1, 25200, 7)) predicts bounding boxes, class probabilities (Healthy, Whiteflies, Caterpillars), and confidence scores across 25,200 anchor boxes with 7 values per anchor (4 box coordinates, 1 objectness score, 2 class probabilities). With 20,856,975 total parameters (79.56 MB), the model is lightweight for mobile deployment after TensorFlow Lite conversion (14-20 MB) and achieves 87.31% accuracy which surpasses the project's >85% target.

4.6 Performance evaluation of the model



```
Model Accuracy: 87.31%
Classification Report:
              precision    recall  f1-score   support

     Healthy       0.76      0.76      0.76        34
caterpillars       0.88      0.84      0.86        50
  whiteflies       0.94      0.98      0.96        50

    accuracy                           0.87       134
   macro avg       0.86      0.86      0.86       134
weighted avg       0.87      0.87      0.87       134
```

Figure 4.8 Model Accuracy

Figure 4.8 shows the performance evaluation of the YOLOv5s model for the AI pest detection project which reports a model accuracy of 87.31% and classification report for the test set (134 images) across three classes which are "Healthy", "caterpillars", and "whiteflies". The overall accuracy of 87.31% which indicates the model correctly classified 87.31% of the test images and surpasses the project's target of >85%. The classification report provides precision, recall, and F1-score for each class. Healthy achieves a precision, recall and F1-score of 0.76 with 34 samples which show balanced but relatively low performance. Caterpillars scores 0.88 across all metrics with 50 samples, indicating strong detection capability. Whiteflies excel with 0.94 precision, 0.94 recall, and 0.94 F1-score across 50 samples which reflects the model's best performance on this class. The macro average (unweighted mean) across classes is 0.86 for precision, recall, and F1-score. Meanwhile the weighted average (considering class support) is 0.87 for all metrics aligning with the overall accuracy.

These are the formulas for the Metrics:

1. **Precision**: A performance metric that measures the proportion of predicted positive instances that are correct

   Formula:

$$Precision = \frac{True\ Positives\ (TP)}{True\ Positives\ (TP) + False\ Positives\ (FP)}$$

2. **Recall**: A performance metric that measures the proportion of actual positives that are correctly identified

   Formula:

$$Recall = \frac{True\ Positives\ (TP)}{True\ Positives\ (TP) + False\ Negatives\ (FN)}$$

3. **F1-Score**: A performance metric that represents the harmonic mean of precision and recall and providing a single score that balances both metrics to evaluate a model's effectiveness

   Formula:

$$F1 - Score = 2\ X\ \frac{Precision\ x\ Recall}{Precision + Recall}$$

4. **Support**: The count of true instances for each class in the test set.

   Formula:

   Support = Number of actual instances of the class in the test path

5. **Accuracy**: Measures the proportion of correctly classified instances across all classes.

   Formula:

$$Accuracy = \frac{Number\ of\ Correct\ Predictions}{Total\ number\ of\ Predictions}$$

6. **Macro average**: Computes the unweighted mean of the metric across all classes, treating each class equally.

Formula:

i. $Macro\ Avg\ (Precision) = \frac{\sum_{i=1}^{N} Precision_i}{N}$

ii. $Macro\ Avg\ (Recall) = \frac{\sum_{i=1}^{N} Recall_i}{N}$

iii. $Macro\ Avg\ (F1 - Score) = \frac{\sum_{i=1}^{N} F1{-}Score_i}{N}$

7. **Weighted Average**: Accounts for class imbalance by weighting each class's metric by its support

Formula:

i. $Weighted\ Avg\ (Precision) = \sum_{i=1}^{N}(Precision\ x\ \frac{Support}{Total\ Support})$

ii. $Weighted\ Avg\ (Recall) = \sum_{i=1}^{N}(Recall\ x\ \frac{Support}{Total\ Support})$

iii. $Weighted\ Avg\ (F1 - Score) = \sum_{i=1}^{N}(F1 - Score\ x\ \frac{Support}{Total\ Support})$

4.6.1 Graphs for performance evaluation

1. **F1-Confidence**

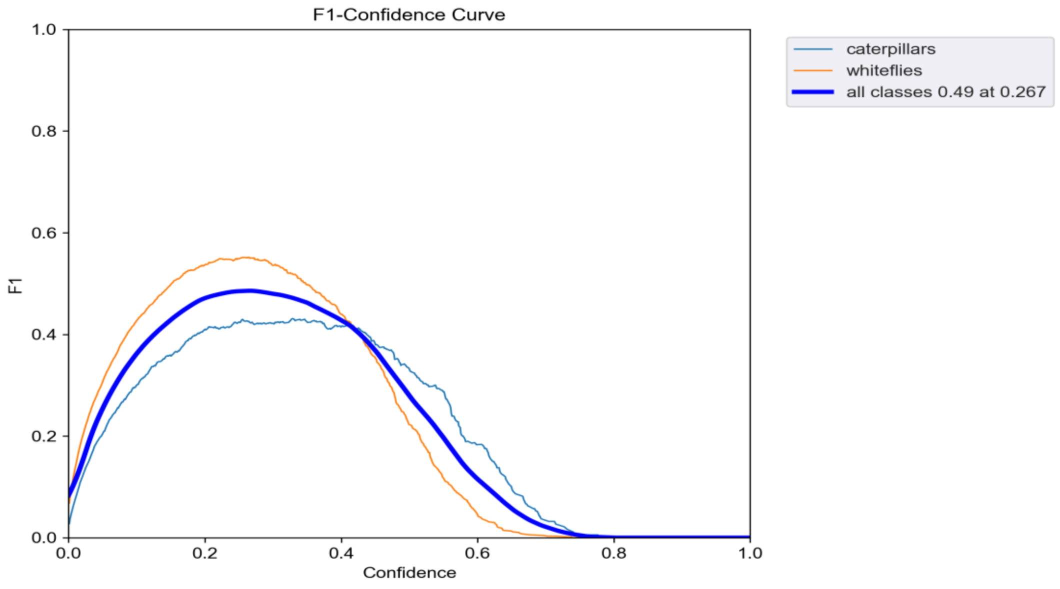


Figure 4.9 F1-Confidence Curve

Figure 4.9 shows the F1-Confidence Curve for the YOLOv5s model in this project and evaluating its performance across confidence thresholds for the classes which are caterpillars, whiteflies and all classes combined. The x-axis represents the confidence threshold (0.0 to 1.0) while the y-axis shows the F1-score (0.0 to 1.0) which balances precision and recall. The curve for "caterpillars" (blue line) peaks at approximately 0.55 around a 0.25 threshold which indicates moderate performance. Meanwhile, "whiteflies" (orange line) reaches a higher peak of approximately at 0.60 and at approximately 0.2 which reflects better detection capability, consistent with earlier metrics (exp: AP = 0.527 for whiteflies vs 0.339 for caterpillars). The "all classes" curve (cyan line) peaks at an F1-score of 0.49 at a confidence threshold of 0.267, representing the average performance across Healthy, Whiteflies, and Caterpillars but this is lower than the classification report's F1-scores (exp: weighted avg 0.87) and overall accuracy (87.31%), suggesting that this curve may reflect an earlier evaluation before tuning. The curves rise sharply from 0.0 to their peaks as the threshold increases then decline, indicates that there is a trade-off between precision and recall with the optimal threshold of 0.267 balancing both for all classes even though the lower F1-score highlights the challenges like the Healthy class's initial poor performance (0% correct in the confusion matrix).

2. **Precision-Confidence**

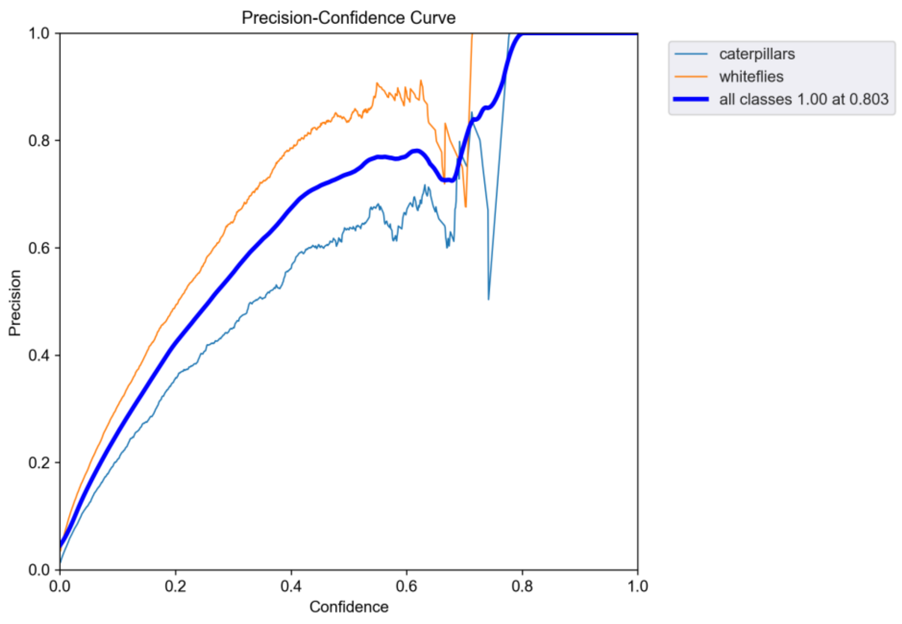


Figure 4.10 Precision-Confidence Curve

Figure 4.10 shows a Precision-Confidence Curve for the YOLOv5s model in this project and illustrates how precision varies with different confidence thresholds for the classes' caterpillars, whiteflies, and all classes combined. The x-axis represents the confidence threshold (0.0 to 1.0) while the y-axis shows precision (0.0 to 1.0) which measures the proportion of predicted positives that are correct. The curve for "caterpillars" (blue line) fluctuates by peaking at approximately 0.9 around 0.4 but dropping to near 0.0 at approximately 0.6 before rising to 1.0 at higher thresholds. This indicates inconsistent performance due to small sample size or class overlap. The "whiteflies" curve (orange line) similarly peaks at approximately 0.9 around 0.4 but drops to near 0.0 at approximately 0.7then rises back to 1.0. This reflects variability but better overall precision than caterpillars and consistent with earlier metrics (AP = 0.527 vs. 0.339). The "all classes" curve (cyan line) steadily increases by reaching a perfect precision of 1.00 at a confidence threshold of 0.803. This shows that all predictions above this threshold are correct even though these likely sacrifices recall which can be seen in the Recall-Confidence Curve (max recall 0.86 at 0.0 threshold). This curve which likely from an earlier evaluation (before the final 87.31% accuracy) highlights that the trade-off between precision and recall with the high threshold of 0.803 ensuring no false positives but potentially missing true positives, especially for the Healthy class which initially struggled (0% correct in the confusion matrix).

3. **Precision-Recall**

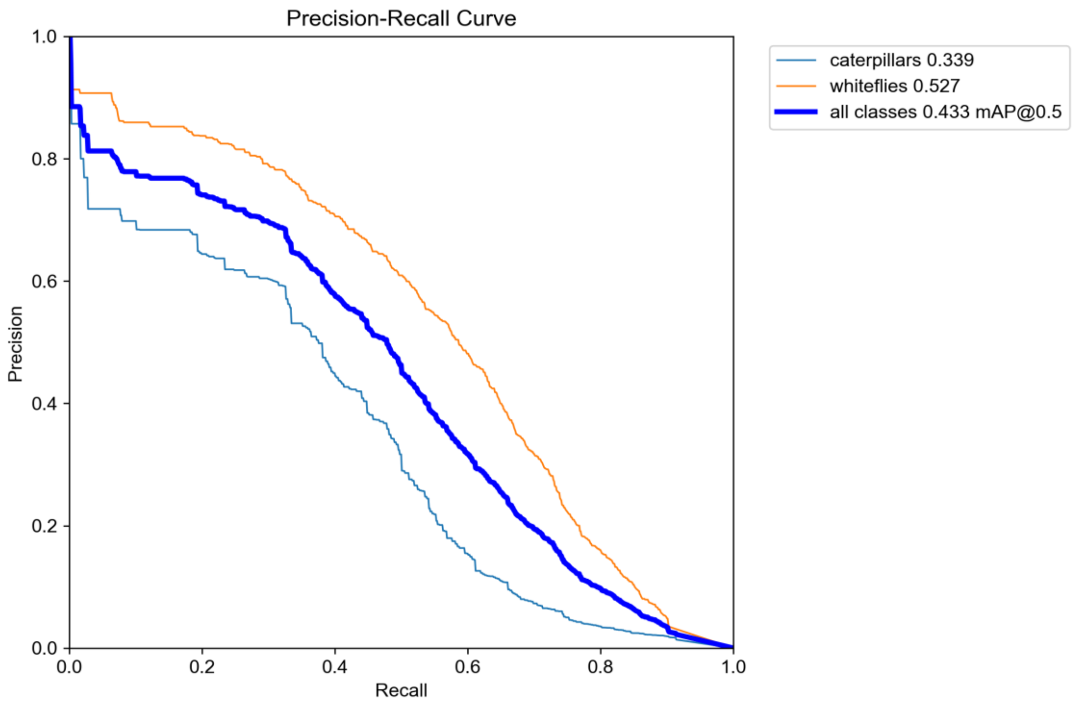


Figure 4.11 Precision-Recall Curve

Figure 4.11 shows a Precision-Recall Curve for the YOLOv5s model in this project and evaluates its performance across different recall levels for the classes' caterpillars, whiteflies and all classes combined. The x-axis represents recall which the proportion of actual positives correctly identified while the y-axis shows the proportion of predicted positives that are correct. The "caterpillars" curve (blue line) has an Average Precision (AP) of 0.339 that starts at high precision (1.0) at low recall (0.0) but dropping to below 0.4 by recall 0.5 which indicates more false positives as recall increases. The "whiteflies" curve (orange line) performs better with an AP of 0.527, maintaining higher precision (0.8 at recall 0.2) and staying above 0.4 until recall 0.6. This reflects stronger detection capability and consistency with the classification report (precision 0.94, recall 0.98). The "all classes" curve (cyan line) yields an mAP@0.5 of 0.433, averaging performance across Healthy, Whiteflies, and Caterpillars but falls below 0.4 by recall 0.6 as influenced by the Healthy class due to its initial poor performance (0% correct in the confusion matrix).

4. **Recall-Confidence**

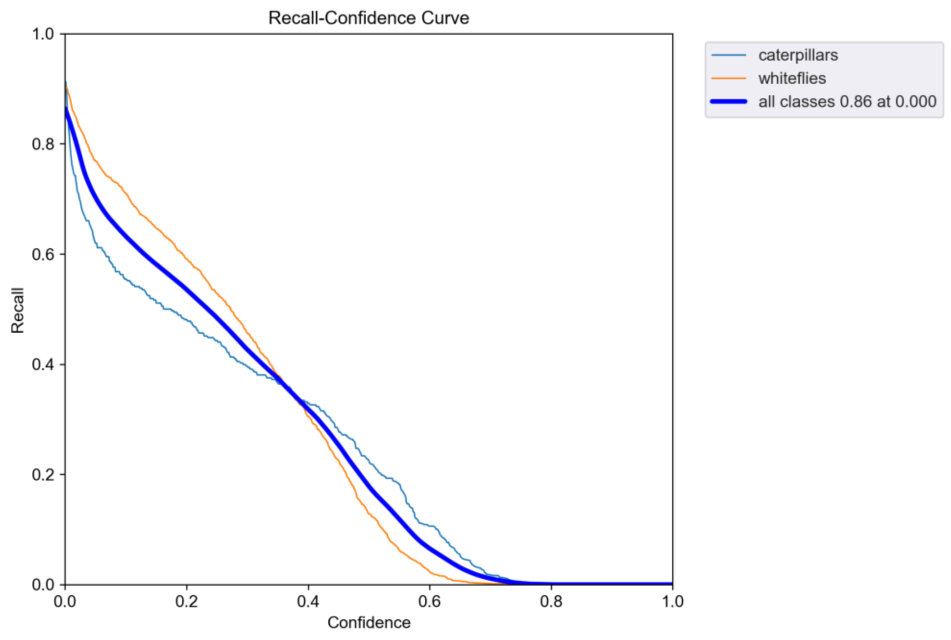


Figure 4.12 Recall-Confidence Curve

Figure 4.12 shows a Recall-Confidence Curve for the YOLOv5s model in this project and shows how recall varies with different confidence thresholds for the classes' caterpillars, whiteflies and all classes combined. The x-axis represents the confidence threshold (0.0 to 1.0) while the y-axis shows recall, the proportion of actual positives correctly identified. The "caterpillars" curve (blue line) starts at 0.9 then recalls at a 0.0 threshold but drops to approximately 0.4 by 0.2 and nearing 0.0 by 0.8. This indicates that the model misses many true caterpillars as the threshold increases. The "whiteflies" curve (orange line) also starts at approximately 0.9 and maintains a slightly higher recall (0.5 at 0.2), then drops to near 0.0 by 0.8 and aligns with its stronger performance (recall 0.98 in the classification report). The "all classes" curve (cyan line) achieves a maximum recall of 0.86 at a 0.000 threshold with average performance across Healthy, Whiteflies, and Caterpillars but then declines to approximately 0.4 by 0.2 and approaches 0.0 by 0.8. This reflects the low confidence scores for true positives especially for the Healthy class (initially 0% correct in the confusion matrix).

4.6.2 Confusion Matrix

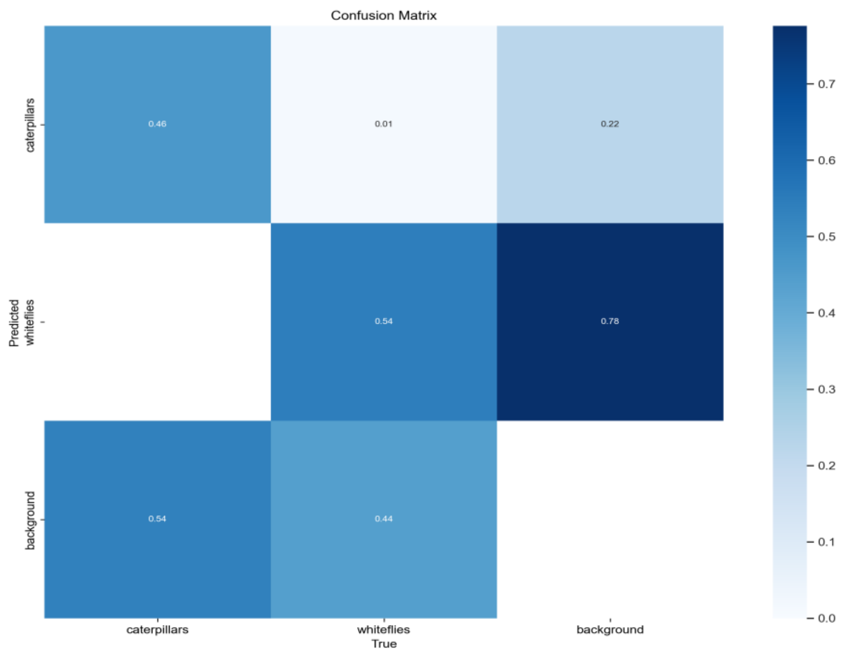


Figure 4.13 Confusion Matrix

Figure 4.13 shows a confusion matrix for the YOLOv5s model in the AI pest detection project and evaluating its performance on the test set (134 images) across three classes which are caterpillars, whiteflies, and background (Healthy). The matrix compares true labels (rows) against predicted labels (columns) with values that are normalized to represent proportions. For true caterpillars, the model correctly predicts 46% as caterpillars but misclassifies 1% as whiteflies and 22% as background. This indicates moderate performance (aligned with AP = 0.339 in the Precision-Recall Curve). For true whiteflies, it correctly predicts 78% as whiteflies but 54% misclassified 54% as caterpillars and 44% as background. This shows stronger performance (AP = 0.527) even though with notable errors. For true background (Healthy), the model fails entirely by predicting 0% correctly and misclassified 54% as caterpillars and also 44% as whiteflies.

4.7 Mobile App Development

The mobile app for this AI model developed by using Android Studio 2024. The AI model that is exported in the form of Tensorflow Lite, will be implemented in the mobile app. This mobile app is developed based on the function of the AI model so it is compatible with the AI model so it can execute it successful performance for the user without any errors. The mobile app contains a homepage with the logo and a “Lets Get Started” for the user. Then, the main function for this app is for the user to capture the live image of the plant or upload the image of the plant from their image directory in their phone for the AI model to process the image. Then the results will be displayed in the user interface along with the QR code that will be generated. Other users also can scan the QR code with their mobile phone doesn’t matter it is an Android phone or IOS phone.

.

Chapter 5

System Implementation

5.1 Hardware Setup

Figure 5.0 Hardware Setup Environment (Laptop)

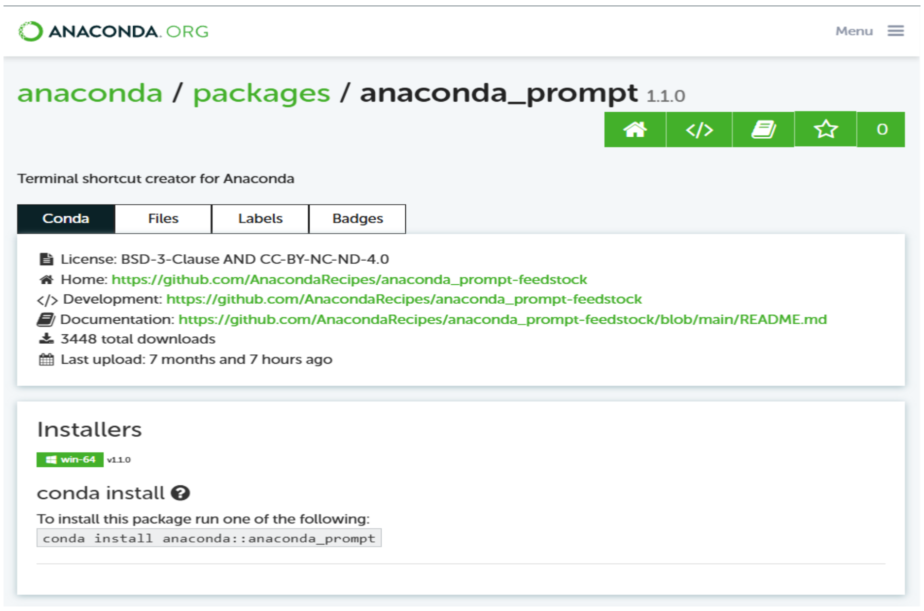
Figure 5.0 shows the hardware setup environment which is a laptop that is used for the AI model development and mobile app development. This machine is sufficient to train the AI model and develop the mobile app as the specifications meet the requirements of developing the AI model and mobile app.

Figure 5.1 Hardware Setup Environment (Smartphone)

Figure 5.1 shows an Android smartphone that is used to execute the mobile application that is developed from the machine for testing purposes as the mobile application of this AI model requires the smartphone's physical front camera to capture the live image.

5.2 Software Setup

5.2.1 Anaconda



Terminal shortcut creator for Anaconda

License: BSD-3-Clause AND CC-BY-NC-ND-4.0
Home: https://github.com/AnacondaRecipes/anaconda_prompt-feedstock
Development: https://github.com/AnacondaRecipes/anaconda_prompt-feedstock
Documentation: https://github.com/AnacondaRecipes/anaconda_prompt-feedstock/blob/main/README.md
3448 total downloads
Last upload: 7 months and 7 hours ago

Installers

win-64 v1.1.0

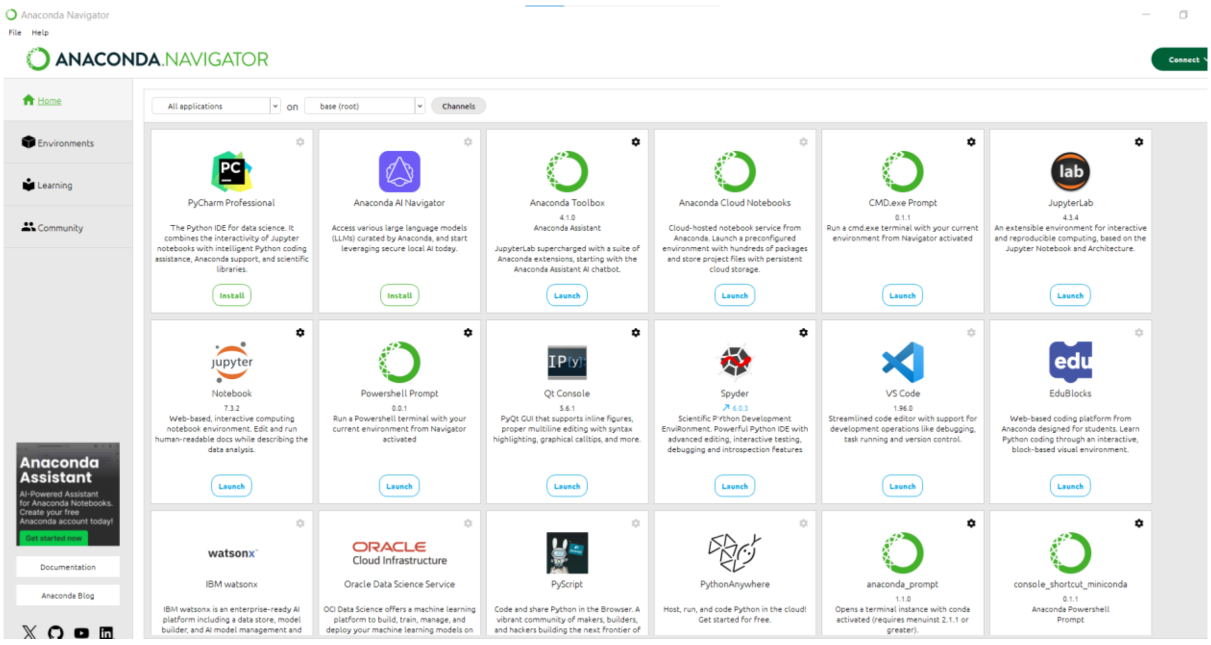
conda install

To install this package run one of the following:

```
conda install anaconda::anaconda_prompt
```

Figure 5.2 Anaconda prompt installation

The Anaconda Prompt is a command-line interface included with the Anaconda Distribution (or Miniconda) that provides a pre-configured environment for managing Python, Conda environments and data science tools. It simplifies package management, environment isolation, and execution of Python-based tools like Jupyter Notebook, Spyder, or scripts. Therefore, to execute the Jupyter Notebook for the AI model development purposes, the anaconda prompt needs to be installed first.



Anaconda Assistant
AI-Powered Assistant for Anaconda Notebooks. Create your free Anaconda account today!
Get started now

PyCharm Professional
The Python IDE for data science. It combines the interactivity of Jupyter notebooks with intelligent Python coding assistance, Anaconda support, and scientific libraries.

Anaconda AI Navigator
Access various large language models (LLMs) curated by Anaconda, and start leveraging secure local AI today.

Anaconda Toolbox
4.1.0
Anaconda Assistant
JupyterLab supercharged with a suite of Anaconda extensions, starting with the Anaconda Assistant AI chatbot.

Anaconda Cloud Notebooks
Cloud-hosted notebook service from Anaconda. Launch a preconfigured environment with hundreds of packages and store project files with persistent cloud storage.

CMD.exe Prompt
0.1.1
Run a cmd.exe terminal with your current environment from Navigator activated

JupyterLab
4.3.4
An extensible environment for interactive and reproducible computing, based on the Jupyter Notebook and Architecture.

Notebook
7.3.2
Web-based, interactive computing notebook environment. Edit and run human-readable docs while describing the data analysis.

Powershell Prompt
0.0.1
Run a Powershell terminal with your current environment from Navigator activated

Qt Console
5.6.1
PyQt GUI that supports inline figures, proper multiline editing with syntax highlighting, graphical calltips, and more.

Spyder
6.0.3
Scientific PYthon Development EnviRonment. Powerful Python IDE with advanced editing, interactive testing, debugging and introspection features

VS Code
1.96.0
Streamlined code editor with support for development operations like debugging, task running and version control.

EduBlocks
Web-based coding platform from Anaconda designed for students. Learn Python coding through an interactive, block-based visual environment.

IBM watsonx
IBM watsonx is an enterprise-ready AI platform including a data store, model builder, and AI model management and

Oracle Data Science Service
OCI Data Science offers a machine learning platform to build, train, manage, and deploy your machine learning models on

PyScript
Code and share Python in the Browser. A vibrant community of makers, builders, and hackers building the next frontier of

PythonAnywhere
Host, run, and code Python in the cloud! Get started for free.

anaconda_prompt
1.1.0
Opens a terminal instance with conda activated (requires menuinst 2.1.1 or greater).

console_shortcut_miniconda
0.1.1
Anaconda Powershell Prompt

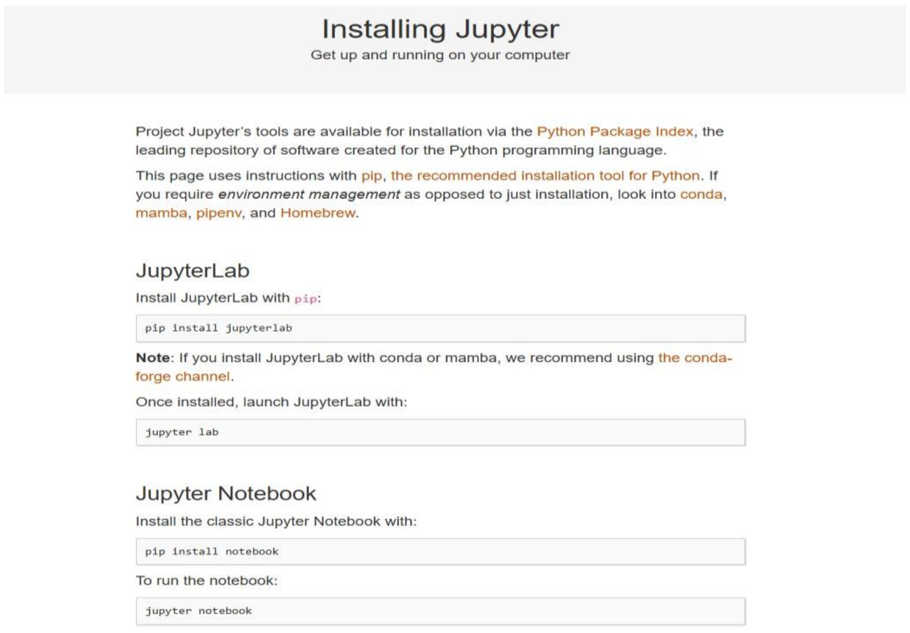
Figure 5.3 Anaconda Navigator

After the installation of the anaconda prompt, there will be a anaconda navigator which will show all the development tools and environment isolation that can be used to develop any AI models or other developments such as data science, R Programming and etc.

5.2.2 Jupyter Notebook

Figure 5.4 Jupyter Notebook

Jupyter Notebook is a versatile tool for interactive coding, data analysis, visualization, machine learning, education, and research. Its ability to combine code, text, and visuals makes it ideal for prototyping, documenting workflows, and sharing results. Jupyter Notebook is not limited to AI as it supports general programming, scientific computing, etc.



Installing Jupyter

Get up and running on your computer

Project Jupyter's tools are available for installation via the Python Package Index, the leading repository of software created for the Python programming language.

This page uses instructions with pip, the recommended installation tool for Python. If you require *environment management* as opposed to just installation, look into conda, mamba, pipenv, and Homebrew.

JupyterLab

Install JupyterLab with `pip`:

```
pip install jupyterlab
```

Note: If you install JupyterLab with conda or mamba, we recommend using the conda-forge channel.

Once installed, launch JupyterLab with:

```
jupyter lab
```

Jupyter Notebook

Install the classic Jupyter Notebook with:

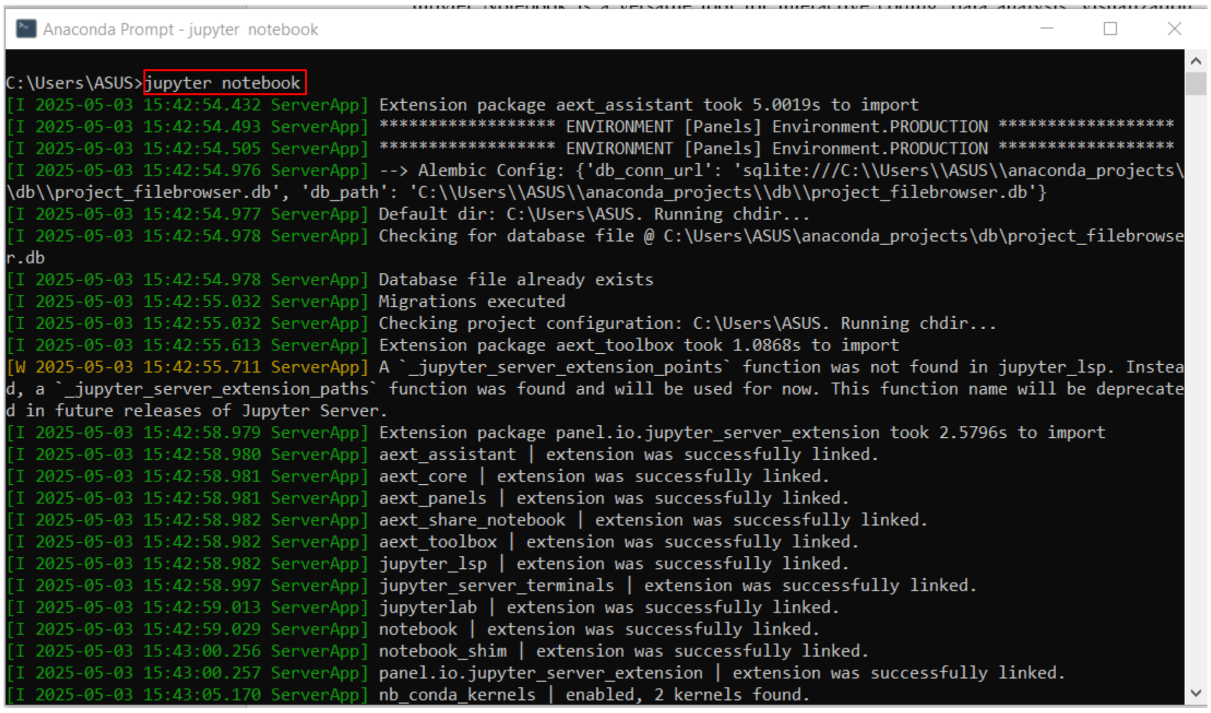
```
pip install notebook
```

To run the notebook:

```
jupyter notebook
```

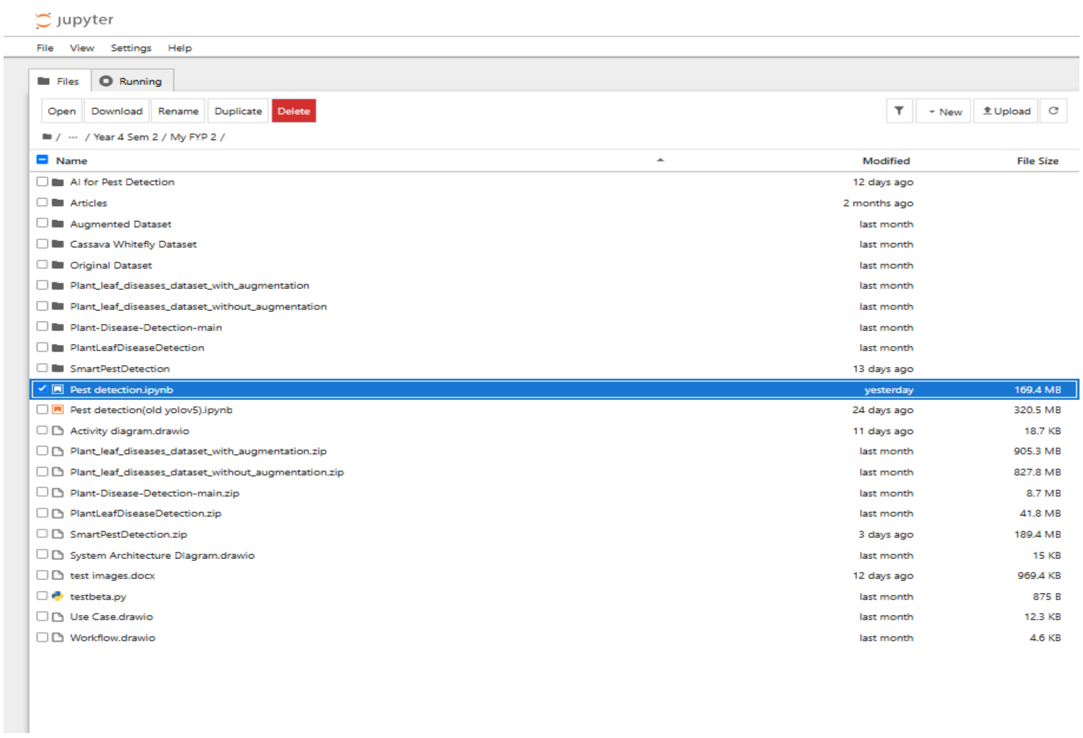
Figure 5.5 Installation of Jupyter Notebook

To develop the AI model, the jupyter notebook tools must be installed by entering those commands that are shown in Figure 5.5.



```
C:\Users\ASUS>jupyter notebook
[I 2025-05-03 15:42:54.432 ServerApp] Extension package aext_assistant took 5.0019s to import
[I 2025-05-03 15:42:54.493 ServerApp] ****************** ENVIRONMENT [Panels] Environment.PRODUCTION ******************
[I 2025-05-03 15:42:54.505 ServerApp] ****************** ENVIRONMENT [Panels] Environment.PRODUCTION ******************
[I 2025-05-03 15:42:54.976 ServerApp] --> Alembic Config: {'db_conn_url': 'sqlite:///C:\\Users\\ASUS\\anaconda_projects\
\db\\project_filebrowser.db', 'db_path': 'C:\\Users\\ASUS\\anaconda_projects\\db\\project_filebrowser.db'}
[I 2025-05-03 15:42:54.977 ServerApp] Default dir: C:\Users\ASUS. Running chdir...
[I 2025-05-03 15:42:54.978 ServerApp] Checking for database file @ C:\Users\ASUS\anaconda_projects\db\project_filebrowse
r.db
[I 2025-05-03 15:42:54.978 ServerApp] Database file already exists
[I 2025-05-03 15:42:55.032 ServerApp] Migrations executed
[I 2025-05-03 15:42:55.032 ServerApp] Checking project configuration: C:\Users\ASUS. Running chdir...
[I 2025-05-03 15:42:55.613 ServerApp] Extension package aext_toolbox took 1.0868s to import
[W 2025-05-03 15:42:55.711 ServerApp] A `_jupyter_server_extension_points` function was not found in jupyter_lsp. Instea
d, a `_jupyter_server_extension_paths` function was found and will be used for now. This function name will be deprecate
d in future releases of Jupyter Server.
[I 2025-05-03 15:42:58.979 ServerApp] Extension package panel.io.jupyter_server_extension took 2.5796s to import
[I 2025-05-03 15:42:58.980 ServerApp] aext_assistant | extension was successfully linked.
[I 2025-05-03 15:42:58.981 ServerApp] aext_core | extension was successfully linked.
[I 2025-05-03 15:42:58.981 ServerApp] aext_panels | extension was successfully linked.
[I 2025-05-03 15:42:58.982 ServerApp] aext_share_notebook | extension was successfully linked.
[I 2025-05-03 15:42:58.982 ServerApp] aext_toolbox | extension was successfully linked.
[I 2025-05-03 15:42:58.982 ServerApp] jupyter_lsp | extension was successfully linked.
[I 2025-05-03 15:42:58.997 ServerApp] jupyter_server_terminals | extension was successfully linked.
[I 2025-05-03 15:42:59.013 ServerApp] jupyterlab | extension was successfully linked.
[I 2025-05-03 15:42:59.029 ServerApp] notebook | extension was successfully linked.
[I 2025-05-03 15:43:00.256 ServerApp] notebook_shim | extension was successfully linked.
[I 2025-05-03 15:43:00.257 ServerApp] panel.io.jupyter_server_extension | extension was successfully linked.
[I 2025-05-03 15:43:05.170 ServerApp] nb_conda_kernels | enabled, 2 kernels found.
```

Figure 5.6 Opening Jupyter Notebook with anaconda prompt

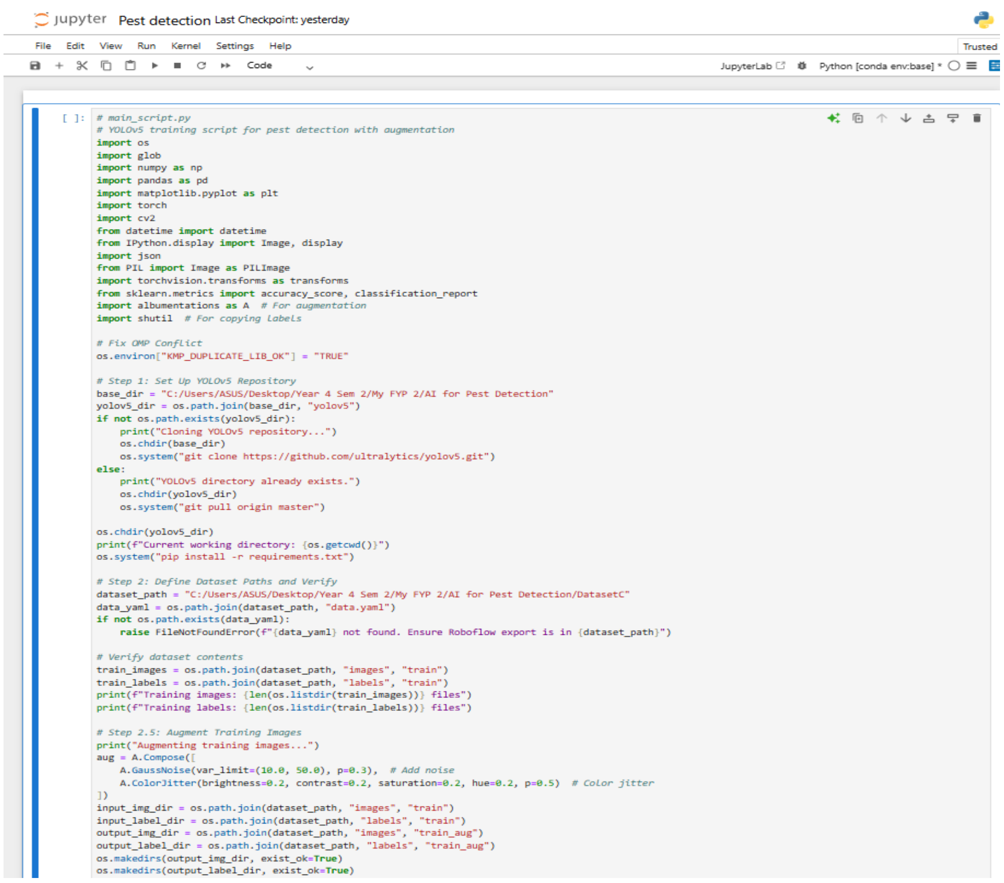


/ ··· / Year 4 Sem 2 / My FYP 2 /

```
Name                                              Modified      File Size
AI for Pest Detection                             12 days ago
Articles                                          2 months ago
Augmented Dataset                                 last month
Cassava Whitefly Dataset                          last month
Original Dataset                                  last month
Plant_leaf_diseases_dataset_with_augmentation     last month
Plant_leaf_diseases_dataset_without_augmentation  last month
Plant-Disease-Detection-main                      last month
PlantLeafDiseaseDetection                         last month
SmartPestDetection                                13 days ago
Pest detection.ipynb                              yesterday     169.4 MB
Pest detection(old yolov5).ipynb                  24 days ago   320.5 MB
Activity diagram.drawio                           11 days ago   18.7 KB
Plant_leaf_diseases_dataset_with_augmentation.zip last month    905.3 MB
Plant_leaf_diseases_dataset_without_augmentation.zip last month 827.8 MB
Plant-Disease-Detection-main.zip                  last month    8.7 MB
PlantLeafDiseaseDetection.zip                     last month    41.8 MB
SmartPestDetection.zip                            3 days ago    189.4 MB
System Architecture Diagram.drawio                last month    15 KB
test images.docx                                  12 days ago   969.4 KB
testbeta.py                                       last month    875 B
Use Case.drawio                                   last month    12.3 KB
Workflow.drawio                                   last month    4.6 KB
```

Figure 5.7 Select a notebook to open in Jupyter Notebook

To open the jupyter notebook, developer must open anaconda prompt and type jupyter notebook as shown in Figure 5.6. Then it will proceed to the chrome for opening the jupyter notebook as shown in Figure 5.7.


jupyter Pest detection Last Checkpoint: yesterday
File Edit View Run Kernel Settings Help
Trusted
Code
JupyterLab
Python [conda env:base]
[ ]:

```python
# main_script.py
# YOLOv5 training script for pest detection with augmentation
import os
import glob
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import torch
import cv2
from datetime import datetime
from IPython.display import Image, display
import json
from PIL import Image as PILImage
import torchvision.transforms as transforms
from sklearn.metrics import accuracy_score, classification_report
import albumentations as A  # For augmentation
import shutil  # For copying labels

# Fix OMP Conflict
os.environ["KMP_DUPLICATE_LIB_OK"] = "TRUE"

# Step 1: Set Up YOLOv5 Repository
base_dir = "C:/Users/ASUS/Desktop/Year 4 Sem 2/My FYP 2/AI for Pest Detection"
yolov5_dir = os.path.join(base_dir, "yolov5")
if not os.path.exists(yolov5_dir):
    print("Cloning YOLOv5 repository...")
    os.chdir(base_dir)
    os.system("git clone https://github.com/ultralytics/yolov5.git")
else:
    print("YOLOv5 directory already exists.")
    os.chdir(yolov5_dir)
    os.system("git pull origin master")

os.chdir(yolov5_dir)
print(f"Current working directory: {os.getcwd()}")
os.system("pip install -r requirements.txt")

# Step 2: Define Dataset Paths and Verify
dataset_path = "C:/Users/ASUS/Desktop/Year 4 Sem 2/My FYP 2/AI for Pest Detection/DatasetC"
data_yaml = os.path.join(dataset_path, "data.yaml")
if not os.path.exists(data_yaml):
    raise FileNotFoundError(f"{data_yaml} not found. Ensure Roboflow export is in {dataset_path}")

# Verify dataset contents
train_images = os.path.join(dataset_path, "images", "train")
train_labels = os.path.join(dataset_path, "labels", "train")
print(f"Training images: {len(os.listdir(train_images))} files")
print(f"Training labels: {len(os.listdir(train_labels))} files")

# Step 2.5: Augment Training Images
print("Augmenting training images...")
aug = A.Compose([
    A.GaussNoise(var_limit=(10.0, 50.0), p=0.3),  # Add noise
    A.ColorJitter(brightness=0.2, contrast=0.2, saturation=0.2, hue=0.2, p=0.5)  # Color jitter
])
input_img_dir = os.path.join(dataset_path, "images", "train")
input_label_dir = os.path.join(dataset_path, "labels", "train")
output_img_dir = os.path.join(dataset_path, "images", "train_aug")
output_label_dir = os.path.join(dataset_path, "labels", "train_aug")
os.makedirs(output_img_dir, exist_ok=True)
os.makedirs(output_label_dir, exist_ok=True)
```

Figure 5.8 Code Snippet in Jupyter Notebook

Then, the developer can begin their development of AI model project by typing codes and executing them in the Jupyter Notebook as shown in Figure 5.8.

5.2.3 Roboflow

Figure 5.9 Roboflow

Roboflow is an end-to-end computer vision platform that is designed to simplify the process of building, training, and deploying computer vision models. It provides a comprehensive suite of tools for developers, data scientists and enterprises, making computer vision accessible regardless of expertise level.

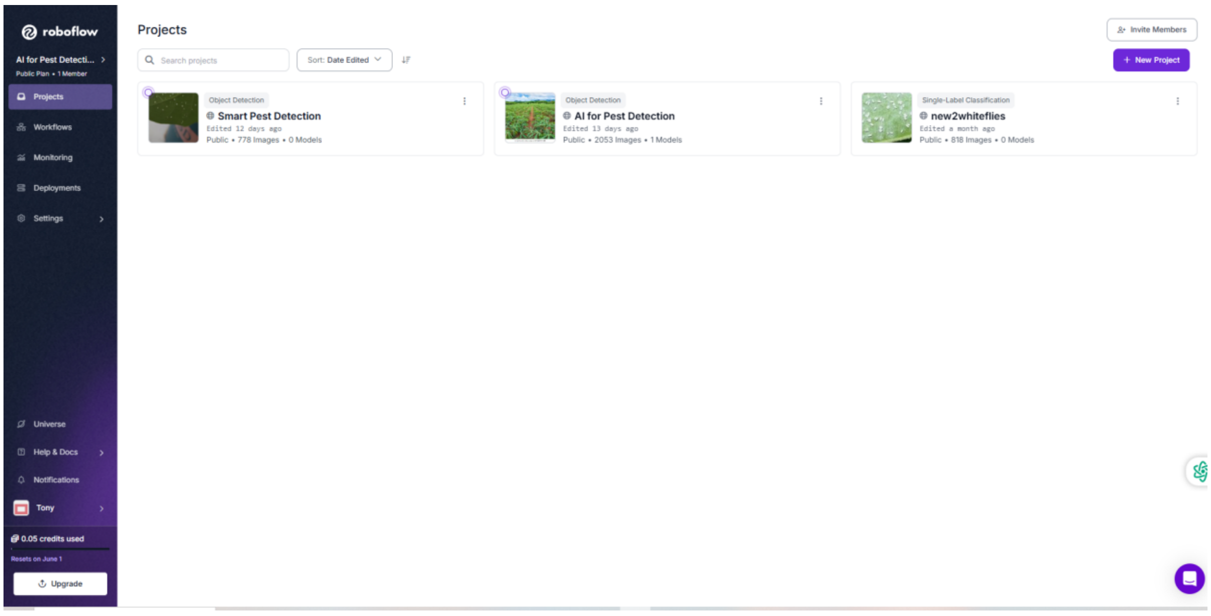


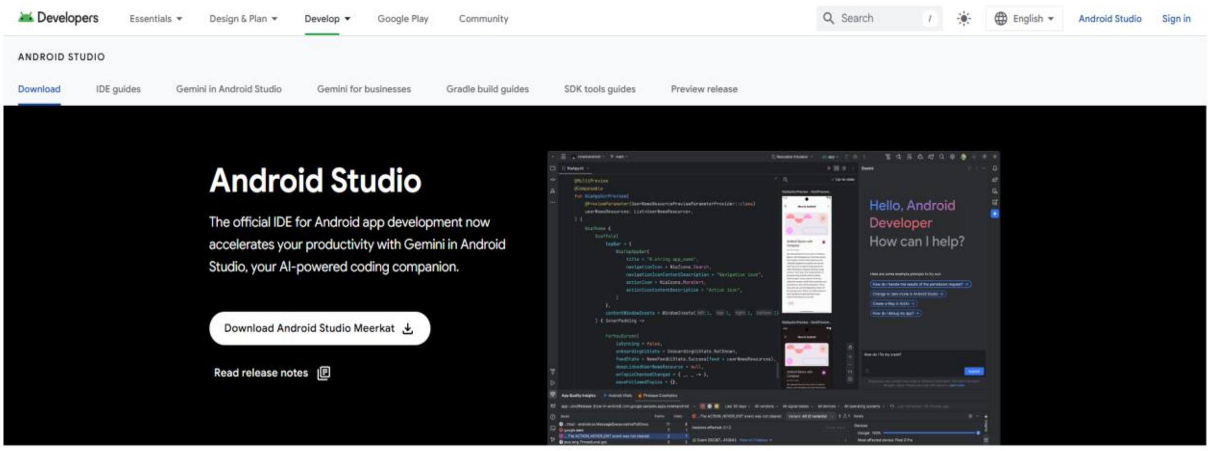
Figure 5.10 Interface for Roboflow

The developer uses the Roboflow to annotate all the images in the dataset that was prepared for the AI model training as the developer is using YOLOv5 to train the AI model. Therefore, the developer needs to use this platform to manually annotate all the images in the dataset and then divide the images into 3 categories which are training, testing and validating.

5.2.4 Android Studio

Figure 5.11 Android Studio

Android Studio is a powerful IDE for building, testing, and deploying Android applications, offering tools for coding, UI design, emulation, debugging, and integration with AI models, cloud services, and databases. It's not limited to any single type of app as it supports mobile, wearable, automotive, and TV apps with robust support for machine learning .

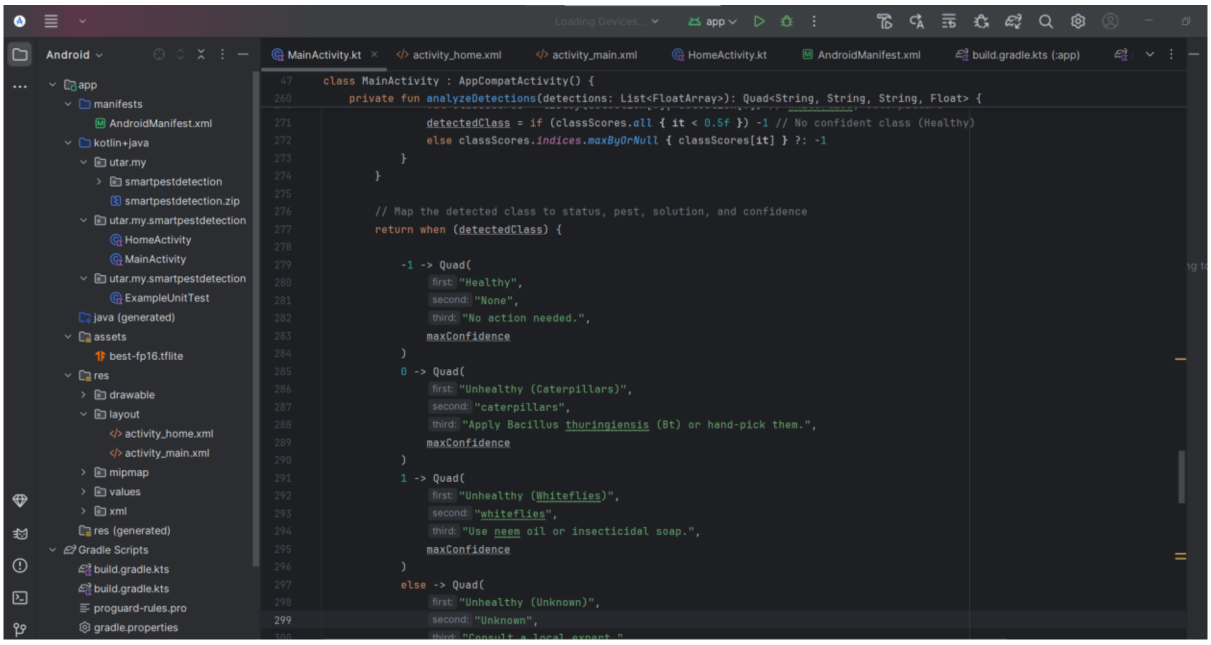


Android Studio

The official IDE for Android app development now accelerates your productivity with Gemini in Android Studio, your AI-powered coding companion.

Figure 5.12 Android Studio Installation

To develop the mobile application, the developer needs to download the Android Studio to build the mobile application. After downloading it, the developer needs to create a project an select the file path to store and save their work. The developer also needs to specify whether they need to use Java or Kotlin language. As for this project, it is developed by using Kotlin.



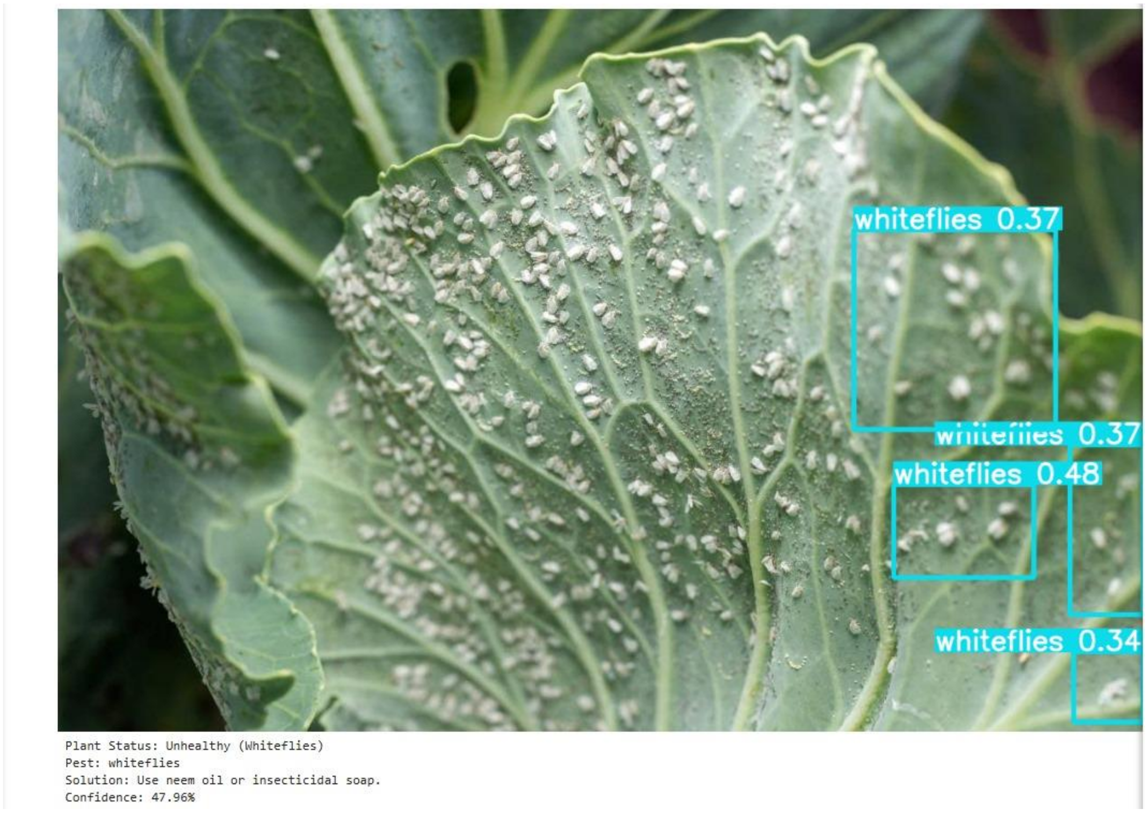
```
class MainActivity : AppCompatActivity() {
    private fun analyzeDetections(detections: List<FloatArray>): Quad<String, String, String, Float> {
                detectedClass = if (classScores.all { it < 0.5f }) -1 // No confident class (Healthy)
                else classScores.indices.maxByOrNull { classScores[it] } ?: -1
            }
        }

        // Map the detected class to status, pest, solution, and confidence
        return when (detectedClass) {

            -1 -> Quad(
                first: "Healthy",
                second: "None",
                third: "No action needed.",
                maxConfidence
            )
            0 -> Quad(
                first: "Unhealthy (Caterpillars)",
                second: "caterpillars",
                third: "Apply Bacillus thuringiensis (Bt) or hand-pick them.",
                maxConfidence
            )
            1 -> Quad(
                first: "Unhealthy (Whiteflies)",
                second: "whiteflies",
                third: "Use neem oil or insecticidal soap.",
                maxConfidence
            )
            else -> Quad(
                first: "Unhealthy (Unknown)",
                second: "Unknown",
                third: "Consult a local expert."
```

Figure 5.13 Coding snippet in Android Studio

The developer will begin the development of the mobile application by putting all the code and execute to check whether the mobile app is working or not and fixed it if there are any bugs or errors which as shown as Figure 5.13.

5.3 System Operations (with screenshot)

5.3.1 Output of the Results shown by AI Model



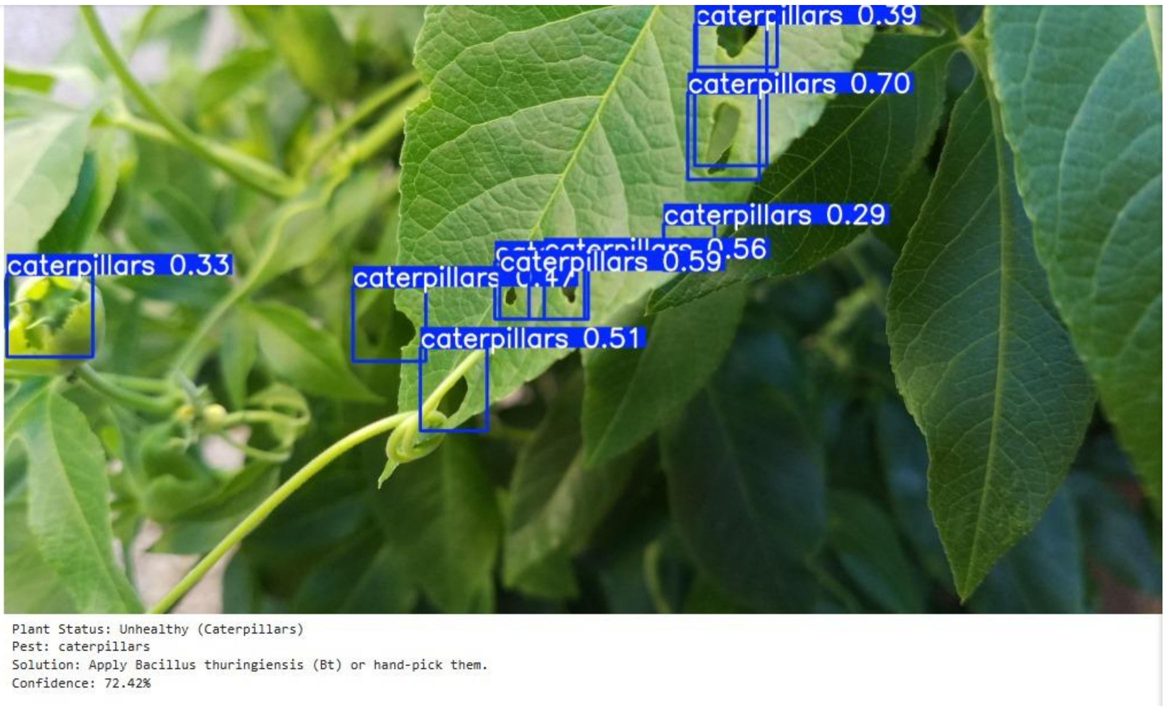
```
Plant Status: Unhealthy (Whiteflies)
Pest: whiteflies
Solution: Use neem oil or insecticidal soap.
Confidence: 47.96%
```

Figure 5.14 Results of the plant that infected by whiteflies

During the training process of the AI model, the developer also includes the image for testing purposes by putting it in the dataset path of where the testing image is stored for the AI model to detect whether the detection that is done by the AI model is correct or incorrect and accurate or not accurate. Based on Figure 5.14, the image of the plant infected by whiteflies is used for the AI detection testing purposes. As the results shown by the AI are:

- Plant Status: Unhealthy (Whiteflies)
- Pest: Whiteflies
- Solution: Use neem oil or insecticidal soap
- Confidence: 47.96%

The results generated by the AI model have proven that its detection is accurate and precise.



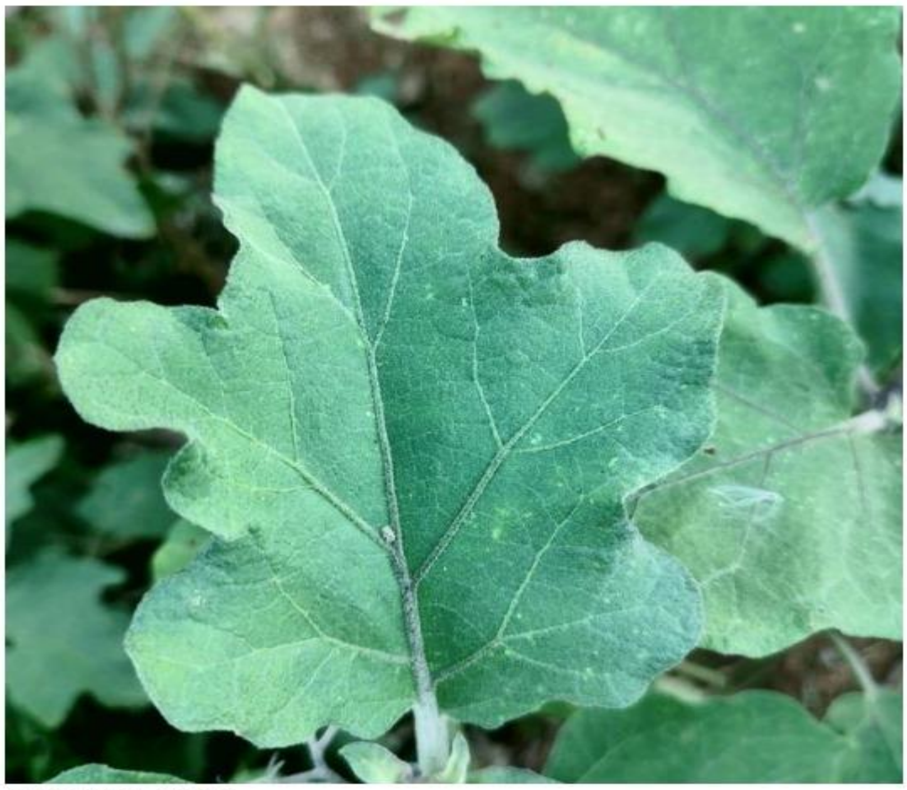
```
Plant Status: Unhealthy (Caterpillars)
Pest: caterpillars
Solution: Apply Bacillus thuringiensis (Bt) or hand-pick them.
Confidence: 72.42%
```

Figure 5.15 Results of the plant infected by caterpillars

Then, the developer changed the previous image to the new image of the plant that was infected with caterpillars in the dataset path for testing purposes. Based on Figure 5.15, the results shown by the AI model are:

- Plant Status: Unhealthy (Caterpillars)
- Pest: Caterpillars
- Solution: Apply Bacillus thuringiensis (Bt) or hand-pic them
- Confidence: 72.42%

The results generated by the AI model have proven that its detection is accurate and precise.



```
Plant Status: Healthy
Pest: None
Solution: No action needed.
Confidence: 0.00%
```

Figure 5.16 Results of the healthy plant

The developer does the final testing on the AI model by providing an image of the healthy plant which is infected by none of the pests. The AI model once again detected correctly of the plant status as the results generated by the AI model are as shown as in Figure 5.16 which are:

- Plant Status: Healthy
- Pest: None
- Solution: No action needed
- Confidence: 0.00%

The results generated by the AI model have proven that its detection is accurate and precise.

5.3.2 Output of the Results shown by the Smart Pest Detection App

The developer then implements the AI model that has been converted to. tflite into the mobile application for the user to test and use for detecting the condition of their plant and the type of pest in their plant.

5.3.2 1 Homepage

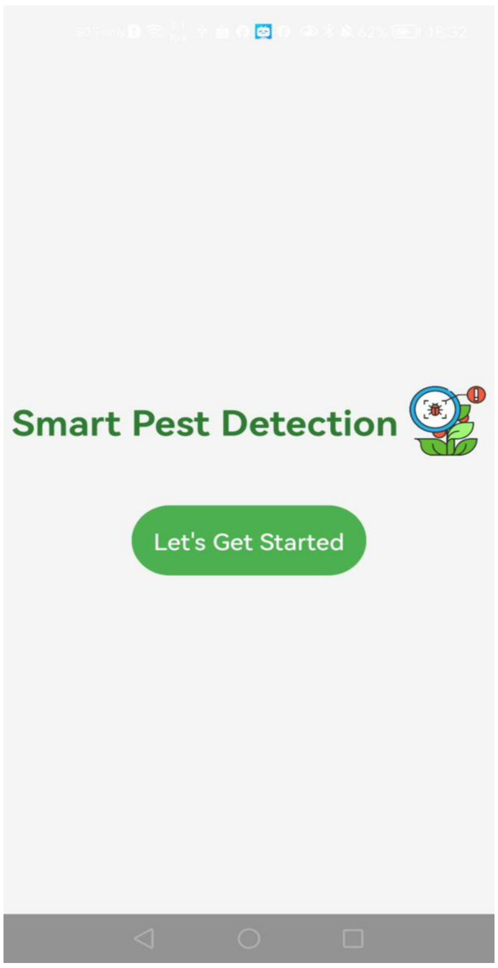


Figure 5.17 Homepage of the Mobile App

Figure 5.17 shows the Homepage of the Mobile App. The homepage shows the name of this mobile app which is “Smart Pest Detection” for the user to know what is the name of the mobile application that they are using. The, there is a “Let’s Get Started” button for the user to click so they can proceed to the next page of the mobile application.

5.3.2.2 Main page

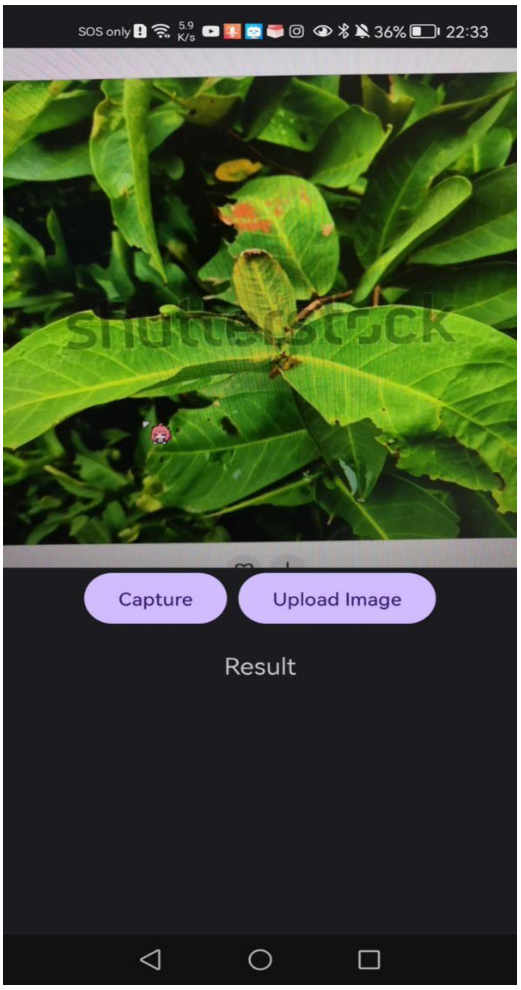


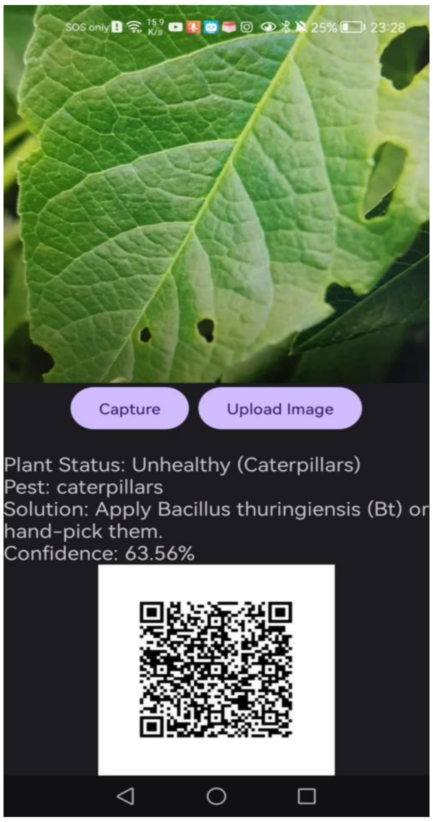
Figure 5.18 Main page of the Mobile App

Figure 5.18 shows the Main page of the Mobile App where the user will redirect to this page after they click the “Let’s Get Started Button” in the Homepage as shown in Figure 5.17. Based on this Main page, there is a camera view which is connected to the phone camera. This camera view is for the user to view the environment or the object that they need to capture as an image. Below the camera view, there are 2 buttons which is “Capture” and “Upload Image”. The “Capture” button is allowing the user to capture the image and then the AI will preprocess the image and detect the status of the plant. As for the “Upload Image” button, allowing the user to select the image of the plant for the AI model to preprocess and detect the status of the plant.

5.3.2.3 Results provided by the mobile app

The Smart Pest Detection mobile app has been used to detect plants with several conditions which are infected by caterpillars, infected by whiteflies and healthy.

5.3.2.3.1 Plants infected with caterpillars



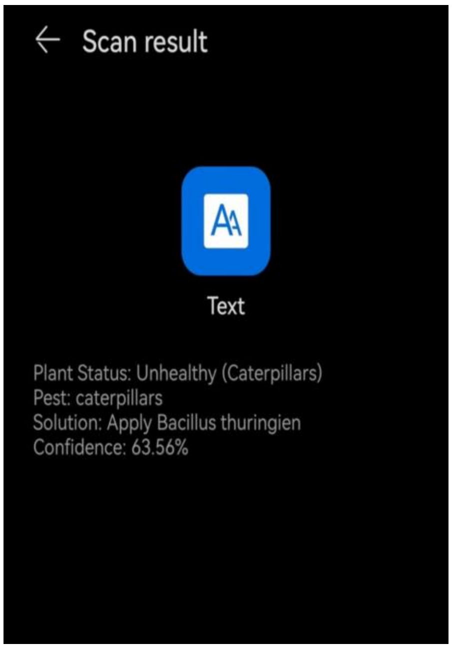
Plant Status: Unhealthy (Caterpillars)
Pest: caterpillars
Solution: Apply Bacillus thuringiensis (Bt) or hand-pick them.
Confidence: 63.56%

Figure 5.19 Results of plant infected with caterpillars

Figure 5.19 shows the results of the plant status when capturing the image of the plant that is infected with caterpillars. The results that are provided by the Smart Pest Detection App as shown based in Figure 5.19 are:

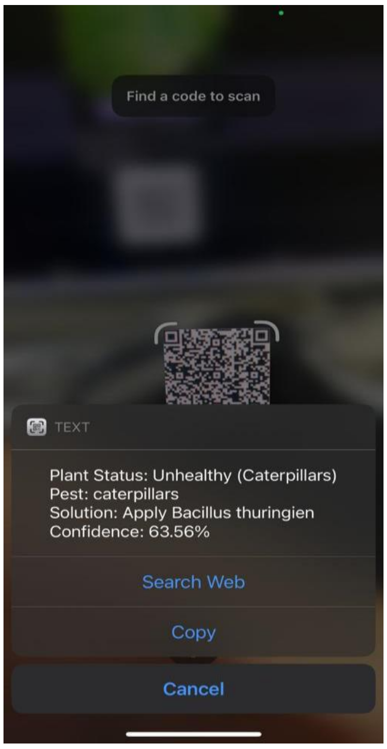
- Plant Status: Unhealthy (Caterpillars)
- Pest: caterpillars
- Solution: Apply Bacillus thuringiensis (Bt) or hand-pick them
- Confidence: 63.56%

Below the results, there are also the QR code which can be scanned by the users as this QR code is used for sharing purposes.



Plant Status: Unhealthy (Caterpillars)
Pest: caterpillars
Solution: Apply Bacillus thuringien
Confidence: 63.56%

Figure 5.20 Results displayed by QR code after scanning with Android smartphone

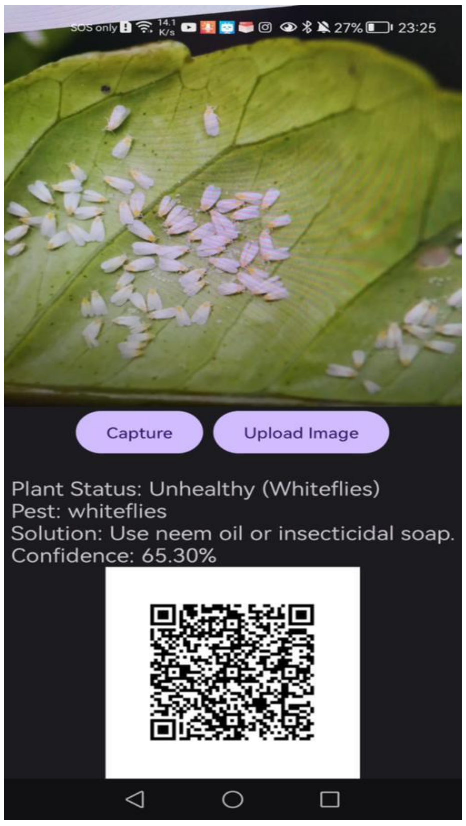


Plant Status: Unhealthy (Caterpillars)
Pest: caterpillars
Solution: Apply Bacillus thuringien
Confidence: 63.56%

Figure 5.21 Results displayed by QR code after scanning with IOS smartphone

Based on the results shown by Figure 5.20 and 5.21, it has proven that the QR code is working and eligible for both Android and IOS smartphones.

5.3.2.3.2 Plant infected by whiteflies



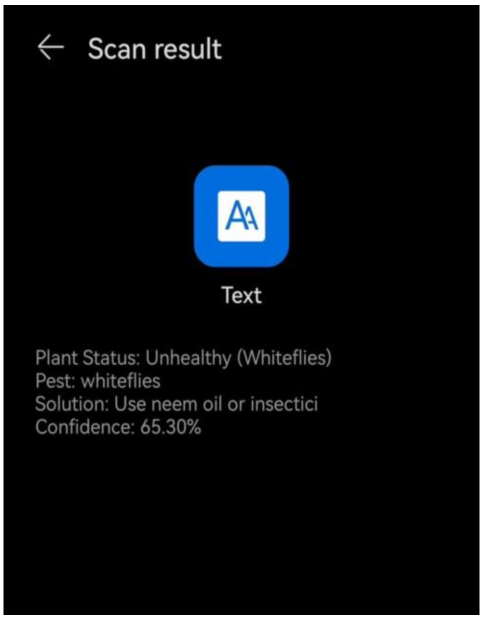
Plant Status: Unhealthy (Whiteflies)
Pest: whiteflies
Solution: Use neem oil or insecticidal soap.
Confidence: 65.30%

Figure 5.22 Results of plant infected with whiteflies

Figure 5.22 shows the results of the plant status when capturing the image of the plant that is infected with whiteflies. The results that are provided by the Smart Pest Detection App as shown based in Figure 5.22 are:

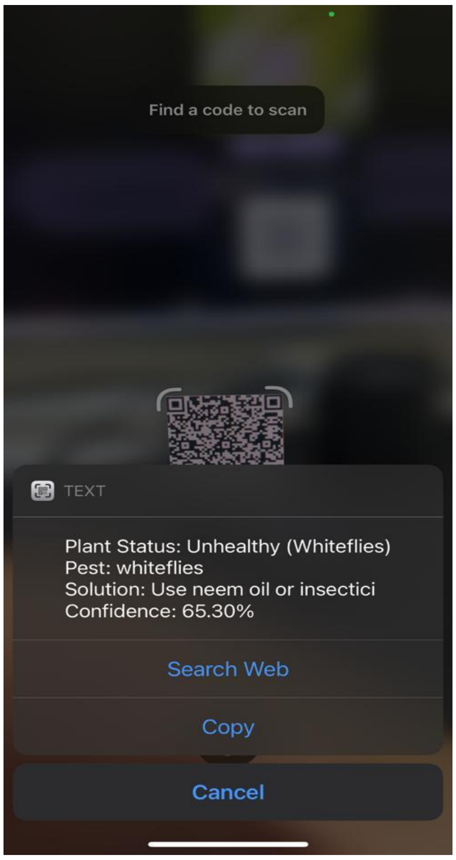
- Plant Status: Unhealthy (Whiteflies)
- Pest: whiteflies
- Solution: Use neem oil or insecticidal soap
- Confidence: 65.30%

Below the results, there are also the QR code which can be scanned by the users as this QR code is used for sharing purposes.



Plant Status: Unhealthy (Whiteflies)
Pest: whiteflies
Solution: Use neem oil or insectici
Confidence: 65.30%

Figure 5.23 Results displayed by QR code after scanning with Android smartphone

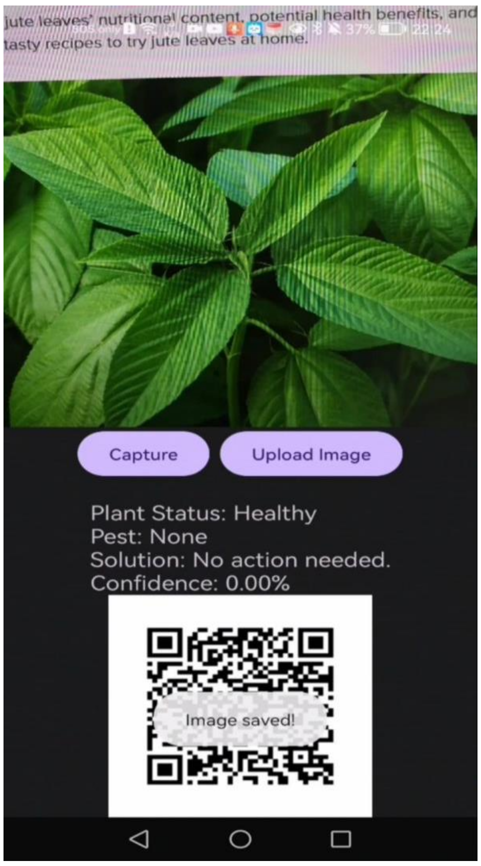


Plant Status: Unhealthy (Whiteflies)
Pest: whiteflies
Solution: Use neem oil or insectici
Confidence: 65.30%

Figure 5.24 Results displayed by QR code after scanning with IOS smartphone

Based on the results shown by Figure 5.23 and 5.24, it has proven that the QR code is working and eligible for both Android and IOS smartphones.

5.3.2.3.3 Healthy Plant


jute leaves' nutritional content, potential health benefits, and tasty recipes to try jute leaves at home.
Capture
Upload Image
Image saved!

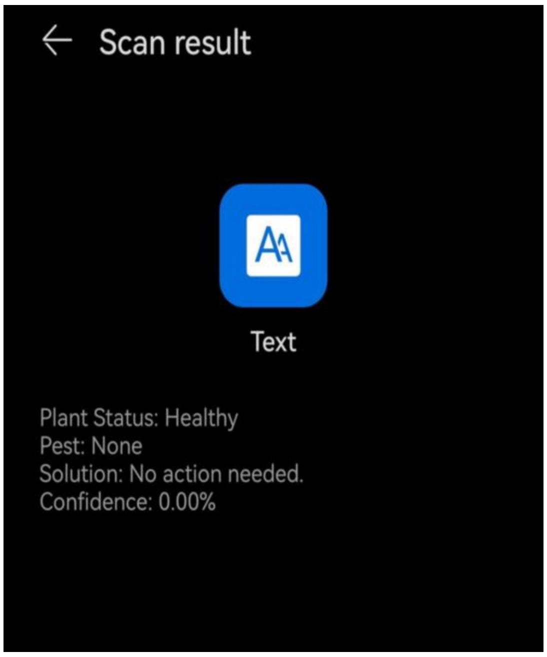
Plant Status: Healthy
Pest: None
Solution: No action needed.
Confidence: 0.00%

Figure 5.25 Results of healthy plant

Figure 5.25 shows the results of plant status when capturing the image of a healthy plant. The results that are provided by the Smart Pest Detection App as shown based in Figure 5.25 are:

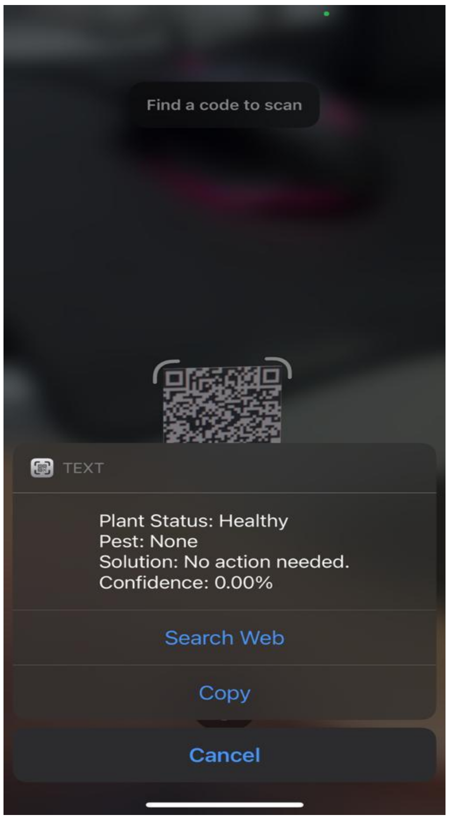
- Plant Status: Healthy
- Pest: None
- Solution: No action needed
- Confidence: 0.00%

Below the results, there are also the QR code which can be scanned by the users as this QR code is used for sharing purposes.



Plant Status: Healthy
Pest: None
Solution: No action needed.
Confidence: 0.00%

Figure 5.26 Results displayed by QR code after scanning with Android smartphone



Plant Status: Healthy
Pest: None
Solution: No action needed.
Confidence: 0.00%

Figure 5.27 Results displayed by QR code after scanning with IOS smartphone

Based on the results shown by Figure 5.26 and 5.27, it has proven that the QR code is working and eligible for both Android and IOS smartphones.

5.3.2.4 Results of the uploaded image

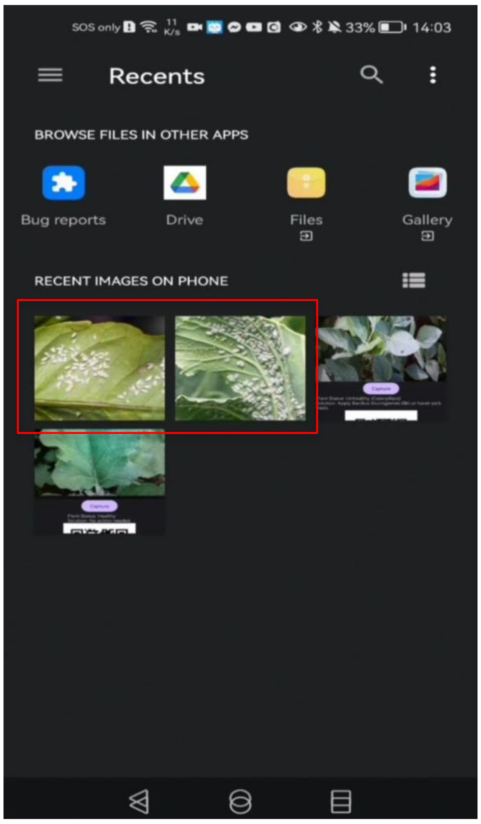
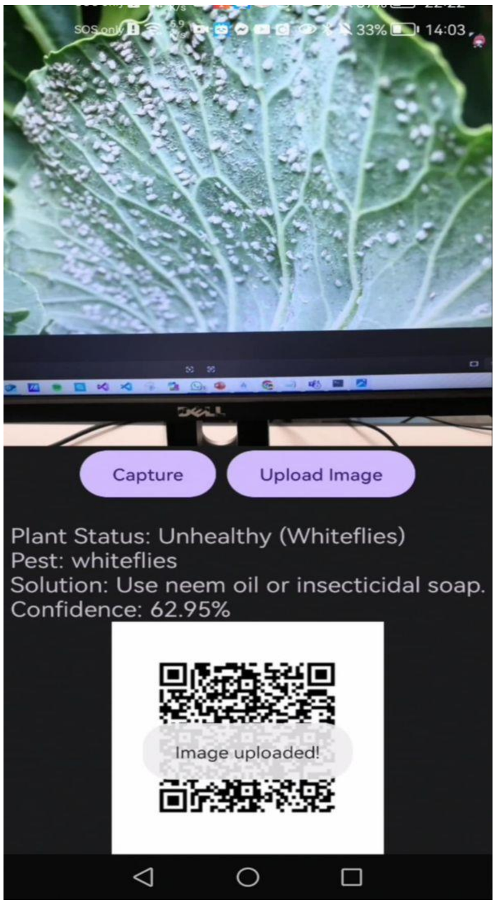


Figure 5.28 User's image directory interface

When the user clicks the "Upload Image" button, the Smart Pest Detection mobile app will redirect the user to their images directory for them to select the image that they need to upload for the application to provide the results of their plant status. Based on Figure 5.28, the user will select the images of the plants that are infected by whiteflies.



Plant Status: Unhealthy (Whiteflies)
Pest: whiteflies
Solution: Use neem oil or insecticidal soap.
Confidence: 62.95%

Figure 5.29 Results of the uploaded image

Figure 5.29 shows the results of the image that was uploaded by the user. The results stated by the Smart Pest Detection are:

- Plant Status: Unhealthy (Whiteflies)
- Pest: whiteflies
- Solution: Use neem oil or insecticidal soap
- Confidence: 62.95%

This proven that the Smart Pest Detection detected the status of the plants correctly as the user uploaded the image of the plant that is infected by whiteflies as shown in Figure 5.29. Below the results, there is also a QR which encoded with the results generated by the Smart pest Detection mobile app for the user to save or share to other people like their friends or agricultural experts.

5.4 Implementation Issues and Challenges

The AI pest detection app's development faced challenges like a small number of datasets where it couldn't boost the accuracy of the model to > 90%. As the current accuracy of the model is 87.31%, it still could detect the status of the plant incorrectly on some other images of the plant. It somehow could misclassify the plant which was infected by whiteflies and caterpillars as healthy plant as the developer didn't annotate the healthy plant dataset. There is also a case where the object which is not a plant and a plant which infected with other pests such as spider mites, mealybugs, fungus gnats and etc will also been classify as healthy for the plant status where it should have detected as unknown object and "Consult an expert" as solution. This is also due to no annotation of the healthy plant dataset. There is also a reason why the healthy plant dataset is not annotated which is the classification of the plant that detected by whiteflies and caterpillars could also be misclassified as healthy plants more often as the annotation of the healthy plant which is green leaves. Hence there are also green leaves on the plants that are infected by whiteflies and caterpillars. Therefore, the plants that are infected by whiteflies and caterpillars have a high chance to be misclassified as healthy plants by the AI model.

Furthermore, during the training of the AI model by using Jupyter notebook, it could take a long time such as 5 hours to 8 hours to train the AI model as the developer uses CPU to train the AI model instead of GPU. This is because there is a failure for the developer to configure the Jupyter Notebook to train the AI model by using GPU due to unknown errors such as the Jupyter Notebook failing to detect the GPU of the machine. While training the AI models, there are some errors such as high misclassification issues and low model accuracy issues. Therefore, there is much time that has been spent, and research has been made in order to correct and enhance the AI model into more successful and more accurate in detecting the condition of the plants.

Chapter 6

System Evaluation and Discussion

6.1 System Testing

6.1.1 AI Pest Detection Model Testing

Test Case	Test Description	Test Data	Expected Result	Pass/Fail
Pest Detection (Image of plants that are infected by caterpillars)	Test AI model's ability to detect pests correctly.	Image of a plant infected by caterpillar (from test images set).	Model should detect the plant as unhealthy and detect the pest type as caterpillars	Pass
Pest Detection (Image of plants that are infected by whiteflies)	Test AI model's ability to detect pests correctly	Image of a plant infected by whiteflies (from test images set).	Model should detect the plant as unhealthy and detect the pest type as whiteflies.	Pass
Pest Detection (Image of healthy plants)	Test AI model's ability to detect the status of the plants correctly	Image of a healthy plant (from test images set).	Model should detect the plant as healthy with no pests.	Pass
Pest Detection (Image of plants that are infected by other pests)	Test AI model's ability to detect the status of the plants correctly	Image of a plant infected by spider mites. (from test images set).	Model should detect the plant as unhealthy (unknown) and detect the pest type as unknown.	Fail

Pest Detection (Image of an object which is not a plant)	Test AI model's ability to detect the status of the plants correctly	Image of a laptop (from test images set)	Model should detect the object as unhealthy (unknown) and detect the pest type as unknown.	Fail
Low-Light Image Detection (Image of plants that are infected by caterpillars)	Test model's performance in low-light conditions.	Image of a plant with caterpillars in low light.	Model should detect the plant as infected by caterpillars.	Pass
Low-Light Image Detection (Image of plants that are infected by whiteflies)	Test model's performance in low-light conditions.	Image of a plant with whiteflies in low light.	Model should detect the plant as infected by whiteflies.	Pass
Low-Light Image Detection (Image of healthy plants)	Test model's performance in low-light conditions.	Image of a healthy plant in low light.	Model should detect the plant as healthy with no pests.	Pass
Solution Generator (Image of plants that are infected by caterpillars)	Test model's ability to generate the correct solution for the plants that are infected by caterpillars.	Image of a plant infected by caterpillar (from test images set).	Model should generate the solution for the plant that infected by caterpillars which is "Apply Bacillus thuringiensis (Bt) or hand-pick them".	Pass
Solution Generator	Test model's ability to generate the	Image of a plant infected by	Model should generate the solution for the plant that infected by	Pass

(Image of plants that are infected by whiteflies)	correct solution for the plants that are infected by whiteflies.	whiteflies (from test images set).	whiteflies which is "Use neem oil or insecticidal soap".	
Solution Generator (Image of healthy plants)	Test model's ability to generate the solution for healthy plants.	Images of a healthy plant (from test images set)	Model should generate the solution for the healthy plant which is "No action needed".	Pass
Solution Generator (Image of plants that are infected by other pests)	Test model's ability to generate the solution for healthy plants.	Image of a plant infected by spider mites. (from test images set).	Model should generate the solution for the plant that infected by other pest which is "Consult a local expert".	Fail
Solution Generator (Image of an object which is not a plant)	Test model's ability to generate the solution for the object which is not a plant.	Image of a laptop (from test images set)	Model should generate the solution for the object which is "Consult a local expert".	Fail

Table 6.1 Test Cases for AI for Pest Detection Model

6.1.2 Mobile App with AI Model Implementation Testing

Test Case	**Test Description**	**Test Data**	**Expected Result**	**Pass/Fail**
Real-Time Inference	Test app's inference speed on the Huawei Nova 4.	Any plant image captured with camera.	Inference should complete within 100-150ms and display the result of the detected plant.	Pass

Camera Capture Functionality (Plant infected by caterpillars)	Test app’s ability to capture and process images.	Live camera captures a plant infected with caterpillars.	App should capture images, process it, and display the result with bounding boxes for the plant that is infected with caterpillars	Pass
Camera Capture Functionality (Plant infected by whiteflies)	Test app’s ability to capture and process images.	Live camera captures a plant infected with whiteflies.	App should capture images, process it, and display the result with bounding boxes for the plant that is infected with whiteflies.	Pass
Camera Capture Functionality (Healthy plant)	Test app’s ability to capture and process images.	Live camera captures a healthy plant.	App should capture images, process it, and display the result with no bounding boxes for the healthy plant.	Pass
Camera Capture Functionality (Plant infected by other pests)	Test app’s ability to capture and process images.	Live camera captures a plant infected by spider mites.	App should capture images, process it, and display the result with no bounding boxes for the plant that infected by other pests.	Fail
Camera Capture Functionality (Object which is not a plant)	Test app’s ability to capture and process images.	Live camera captures a laptop.	App should capture images, process it, and display the result with no bounding boxes for the laptop.	Fail
Result Display (Plant infected by caterpillars)	Test app’s ability to display the correct results based on the	Live camera captures a plant infected by caterpillars.	App should display the results of the plant that infected by caterpillars such as (Plant Status,	Pass

	status of the plant.		Pests, Solution and Confidence).	
Result Display (Plant infected by whiteflies)	Test app's ability to display the correct results based on the status of the plant.	Live camera captures a plant infected by whiteflies.	App should display the results of the plant that infected by whiteflies such as (Plant Status, Pests, Solution and Confidence).	Pass
Result Display (Healthy Plant)	Test app's ability to display the correct results based on the status of the plant.	Live camera captures a healthy plant.	App should display the results of the healthy plant such as (Plant Status, Pests, Solution and Confidence).	Pass
Result Display (Plant infected by other pests)	Test app's ability to display the correct results based on the status of the plant.	Live camera captures a plant that is infected by spider mites.	App should display the results of the plant that infected by other pests such as (Plant Status, Pests, Solution and Confidence).	Fail
Result Display (Plant infected by other object which is not a plant)	Test app's ability to display the correct results based on the status of the plant.	Live camera captures a laptop.	App should display the results of the plant that infected by laptop such as (Plant Status, Pests, Solution and Confidence).	Fail
Gallery Image Selection	Test app's ability to	Select an image of plant with caterpillars	App should process image and display the results of the plant that	Pass

(Image of a plant which infected by caterpillars)	process gallery image	from the gallery.	infected by caterpillars such as (Plant Status, Pests, Solution and Confidence).	
Gallery Image Selection (Image of a plant which infected by whiteflies)	Test app’s ability to process gallery image	Select an image of plant with whiteflies from the gallery.	App should process image and display the results of the plant that infected by whiteflies such as (Plant Status, Pests, Solution and Confidence).	Pass
Gallery Image Selection (Image of a healthy plant)	Test app’s ability to process gallery image	Select an image of healthy plant from the gallery.	App should process image and display the results of the healthy plant such as (Plant Status, Pests, Solution and Confidence).	Pass
Gallery Image Selection (Image of a plant infected with other pests)	Test app’s ability to process gallery image	Select an image of plant infected by spider mites image from the gallery.	App should process image and display the results of the plant that is infected by spider mites such as (Plant Status, Pests, Solution and Confidence).	Fail
Gallery Image Selection (Image of a object which is not a plant)	Test app’s ability to process gallery image	Select an image of laptop from the gallery.	App should process image and display the results of the laptop such as (Plant Status, Pests, Solution and Confidence).	Fail
QR Code Generator	Test app’s ability to generate the QR	Live capture or upload images of the plants	App should be able to encode the results into	Pass

	code based on the condition of the plants.	(exp: caterpillars, whiteflies and healthy)	the QR code and display on the UI.	
QR Code scanned with Android device.	Test app’s ability to generate the QR code that can be scanned by any Android devices.	QR code generated by the mobile app.	The QR code generated by the mobile app should be able to be scanned by Android devices and the user is able to read the results that are encoded in ther QR code.	Pass
QR Code scanned with IOS device.	Test app’s ability to generate the QR code that can be scanned by any IOS devices.	QR code generated by the mobile app.	The QR code generated by the mobile app should be able to be scanned by IOS devices and the user is able to read the results that are encoded in ther QR code.	Pass
Use app without network connectivity	Test app’s whether the app can work when there is no internet connection.	Turn off the Wi-Fi icon in the mobile app and all other internet connection.	The app should be able to work normally and able to detect the plant status and display the results.	Pass

Table 6.2 Test Cases for Mobile App with AI Model Implementation

6.2 Project Challenges

The main challenge of developing this project is to build and maintain an accurate classification on the Pest Detection AI model. This is because there is a lack of datasets that have been used to train this AI model as the dataset needed to train a super accurate AI model would be incredibly large. There is also a significant underfit for the Healthy class as they are only 0% correct due to class imbalance and a small dataset. The developer didn't annotate for the "Healthy" plant dataset as after annotation, it has high change on misclassification on the plants that infected by caterpillars or whiteflies as the annotation of the healthy plant is green leaves where these green leaves also appeared on the plant that infected whiteflies or caterpillars as they are also a plant. Therefore, the model accuracy for this AI for Pest Detection only reaches 87.31% which just passes the aim of the model accuracy for this project which is 85%. This also causes the plants infected by whiteflies and caterpillars to be misclassified as healthy.

The second challenge is this AI model cannot detect the plant that infected by other pest such as "Spider Mites", "Fungus gnats", "Mealybugs", etc. This is due that it is hard to obtain the dataset of the images that contain these types of pests from the social media platform as these plant pests are quite rare. The AI Model also failed to predict the object which is not a plant as unknown objects. This is due to there are no datasets of this unknown object category as this project aims to focus on the pest types which are caterpillars and whiteflies. This project also focuses on plants, not objects which are not plants. Therefore, the developer didn't feel that the category of unknown objects should be included as it didn't meet the project objectives.

The last challenge is that tis AI model is trained with the YOLOv5 model where there consists Deep Learning knowledge. Since the developer didn't take any course that is related to Deep Learning, the developer need time to do research and learn about the Deep Learning knowledge and information that is related to YOLOv5 model.

6.3 Objectives Evaluation

The first objective of this project is to **investigate the feasibility of YOLOv5 for Pest Detection**. This objective has been fulfilled as the YOLOv5 model which trained on 859 images (448 images for training, 277 for validation, and 134 images for testing) has achieved the model accuracy of 87.31% which surpassed the target which is 85%. The classification report shows balanced performance with F1-scores of 0.76 (Healthy), 0.86 (Caterpillars), and 0.94 (Whiteflies), demonstrating effective pest detection across all categories.

The second objective of this project is **developing a real-time-on-device system**. This objective is fully achieved as the mobile app with the implementation of the Pest Detection AI model has been deployed on an Android mobile device and successfully perform real-time pest detection, classifying plant health status and identifying pest types (Healthy, Whiteflies, Caterpillars). The TensorFlow Lite model also ensures on-device processing without internet dependency and ideal for remote farm settings.

The third objective of this project is to **propose targeted pest management solutions**. This objective has successfully met as the mobile app accurately identifies pest types when a plant is Unhealthy, with high F1-scores for Whiteflies (0.94) and Caterpillars (0.86), and displays results like "Unhealthy, infected by Caterpillars and Confidence: 87%. The app includes a mechanism to provide targeted solutions such as recommending neem oil or insecticidal soap for plants that are infected by whiteflies and Bacillus thuringiensis (Bt) or hand-picking for plants that are infected by caterpillars as specified by the developer. These evidence-based solutions were integrated into the app's result display and guiding the farmers effectively on detecting the status of the plants. For example, after detecting the plant is infected by whiteflies, the app suggests using neem oil or insecticidal soap as the solution for the plant).

The last objective of this project is to **enable data sharing through QR code integration**. This objective was achieved as the mobile app includes a QR code generation feature that encodes detection results for the farmers to share this information with agricultural experts or record systems for consultation and documentation. In large farms, QR codes can be placed on each plant and enable the farmers to scan them and retrieve pest management instructions avoiding confusion and supporting collaborative strategies.

Chapter 7
Conclusion

7.1 Conclusion

In conclusion, this AI Pest Detection project has successfully developed and deployed as a mobile application named Smart Pest Detection. This project has successfully fulfilled the 4 objectives which are **investigate the feasibility of YOLOv5 for Pest Detection**, **developing a real-time-on-device system**, **propose targeted pest management solutions** and **enable data sharing through QR code integration** as this project has successfully trained the Pest Detection AI model with the accuracy of 87.31% and has successfully detected the plant status correctly and generated the results that contains data like plant status, type of pest, solution and the confidence score accordingly. The Smart Pest Detection also has successfully encoded the results that are generated into the QR code for the farmers to share or for them to manage their large crops. However, there are also flaws in this project as the Smart Pest Detection sometimes could misclassify the plant status of the plant that is infected by whiteflies or caterpillars as healthy. This project also can't classify the plants that are infected by other pests besides whiteflies and caterpillars and also failed to classify the object which are not plant by providing the results such as Plant Status: unknown, Pest: unknown, Pest: Unknown and Confidence: 0.00%. Therefore, there are also some enhancements on this project in the future so that it can become more effective and reliable for farmers and gardeners on detecting the plant status not just on the whiteflies and caterpillar's category only.

7.2 Recommendations

Firstly, the future work can be done by expanding the diversifying the dataset to solve the misclassification of the infected plant as healthy and improve the model accuracy. This can be done by expanding the dataset beyond 859 images by collecting more images from diverse farm and garden environments, including varied lighting conditions (exp: dawn, dusk, shadows) and different plant types. This recommendation can enhance the robustness of the Pest Detection AI model.

Secondly, future work can be done by extending the pest detection to additional pest by expanding the category of the plants to aphids, spider mites, fungus gnats etc. The current model is limited to detecting whiteflies and caterpillars and failed to identify other common pests. This recommendation can be made by training the Pest Detection AI model to detect additional pests like aphids, spider mites, or beetles, which are prevalent in farm settings. This action requires updating the dataset with labeled images of these pests by including new classes of the new pest’s category and then retrain the Pest Detection AI model. This enhancement of the Pest Detection AI model will make the app more versatile by providing the farmers with a broader pest management tool and the corresponding pest management solutions.

Lastly, future work can be done by improving handling of non-plants object. The current Smart Pest Detection app failed to classify the non-plant objects by providing the results which are Plant Status: unknown, Pest: unknown, Pest: Unknown and Confidence: 0.00%. This issue can confuse the user and might demotivate them to use this app. To address his issue, the developer has to implement a pre-classification step to identify non-plant objects before running pest detection. This can be achieved by training a binary classifier such as plant vs non-plant as a preliminary step by using a simple model like a small CNN or leveraging YOLOv5’s objectness score to filter out non-plant detections. If a non-plant object is detected, the app should display the message like “Non-Plant Object Detected” instead of “Unknown,” improving user experience and trust. This feature can be tested with a small dataset of non-plant objects such as rocks, tools, soil or other objects to ensure the model’s classification accuracy.

REFERENCES

1. Zhao, Z.Q., Huang, D.S., 2007. A mended hybrid learning algorithm for radial basis function neural networks to improve generalization capability. Available at A mended hybrid learning algorithm for radial basis function neural networks to improve generalization capability - ScienceDirect. (Accessed on March 2025)

2. Hubel, D.H., Wiesel, T.N., 2009. Republication of The Journal of Physiology (1959) 148, 574–591: Receptive fields of single neurones in the cat's striate cortex. Available at 8. Receptive fields of single neurones in the cat's striate cortex.(Accessed on March 2025)

3. Fukushima, K., 1980. Neocognitron: a self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. Available at Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position | Biological Cybernetics. (Accessed on March 2025)

4. Lecun, Y., Boser, B., Denker, J., et al., 1989. Backpropagation applied to handwritten zip code recognition. Available at Backpropagation Applied to Handwritten Zip Code Recognition | Neural Computation | MIT Press. (Accessed on March 2025)

5. Coates, A., Baumstarck, P., Le, Q., et al., 2009. Scalable learning for object detection with GPU hardware. Available at Scalable learning for object detection with GPU hardware | IEEE Conference Publication | IEEE Xplore. (Accessed on March 2025)

6. Krizhevsky, A., Sutskever, I., Hinton, G.E., 2012. ImageNet classification with deep convolutional neural networks. Available at ImageNet classification with deep convolutional neural networks | Communications of the ACM. (Accessed on March 2025)

7. Simonyan, K., Zisserman, A., 2014. Very deep convolutional networks for large-scale image recognition. Available at [1409.1556] Very Deep Convolutional Networks for Large-Scale Image Recognition. (Accessed on March 2025)

8. Szegedy, C., Liu, W., Jia, Y., et al., 2015. Szegedy, C., Liu, W., Jia, Y., et al., 2015. Going deeper with convolutions. Available at Going deeper with convolutions | IEEE Conference Publication | IEEE Xplore. (Accessed on March 2025)

9. He, K., Zhang, X., Ren, S., et al., 2016. Deep residual learning for image recognition. Available at CVPR 2016 Open Access Repository. (Accessed on March 2025)

10. Zeiler, M.D., Fergus, R., 2014. Visualizing and understanding convolutional networks. Available at Visualizing and Understanding Convolutional Networks | SpringerLink. (Accessed on March 2025)

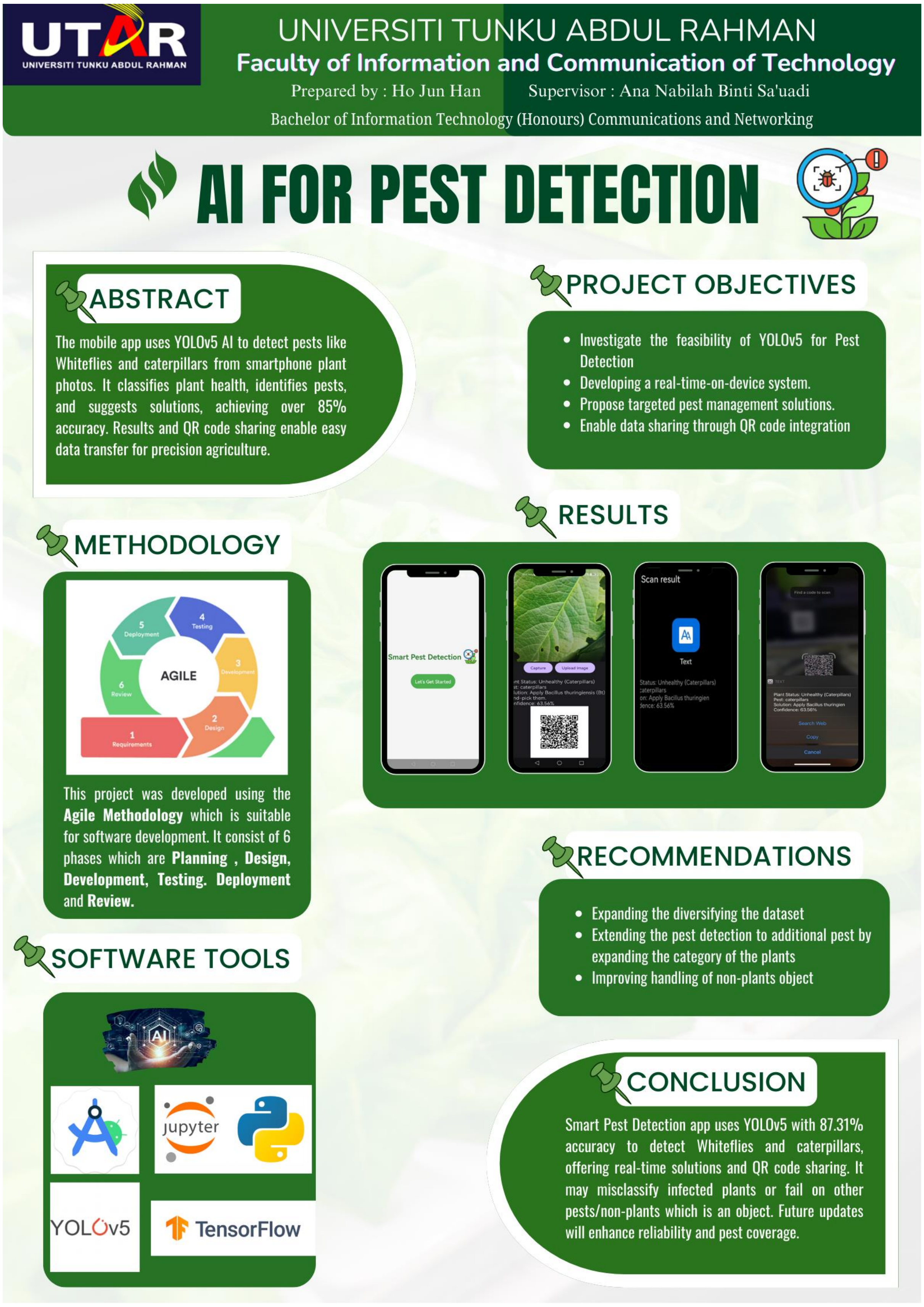
11. C.-Y. Wang, A. Bochkovskiy, H.-Y.-M. LiaoYOLOv7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors. Available at https://www.sciencedirect.com/science/article/pii/S0168169924001170#bb0170. (Accessed on March 2025)

12. Md. Akkas Ali, Rajesh Kumar Dhanaraj, Anand Nayyar. A high performance-oriented AI-enabled IoT-based pest detection system using sound analytics in large agricultural field. Available at https://www.sciencedirect.com/science/article/pii/S0141933123001904. (Accessed on March 2025)

13. Sarah Laoyan .(2024, February 2). What is Agile methodology (A beginner's guide). Available at : https://asana.com/resources/agile- methodology. (Accessed on March 2025)

14. Man-Ting Li, Sang-Hyun Lee.(20 April 2022). A Study on Small Pest Detection Based on a CascadeR-CN-Swin Model. Available at : https://www.sciencedirect.com/org/science/article/pii/S154622182200950X. (Accessed on March 2025)

POSTER



UNIVERSITI TUNKU ABDUL RAHMAN

Faculty of Information and Communication of Technology

Prepared by : Ho Jun Han   Supervisor : Ana Nabilah Binti Sa'uadi

Bachelor of Information Technology (Honours) Communications and Networking

AI FOR PEST DETECTION

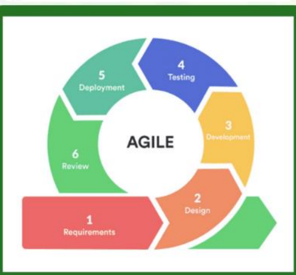
ABSTRACT

The mobile app uses YOLOv5 AI to detect pests like Whiteflies and caterpillars from smartphone plant photos. It classifies plant health, identifies pests, and suggests solutions, achieving over 85% accuracy. Results and QR code sharing enable easy data transfer for precision agriculture.

PROJECT OBJECTIVES

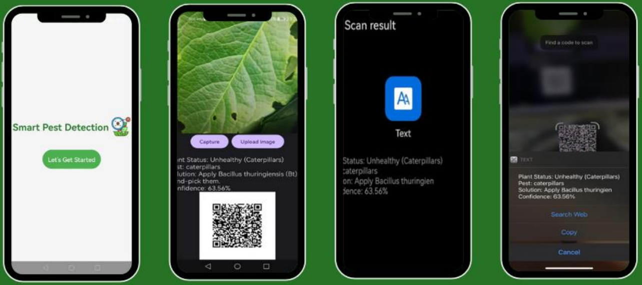
- Investigate the feasibility of YOLOv5 for Pest Detection
- Developing a real-time-on-device system.
- Propose targeted pest management solutions.
- Enable data sharing through QR code integration

METHODOLOGY



This project was developed using the **Agile Methodology** which is suitable for software development. It consist of 6 phases which are **Planning** , **Design**, **Development**, **Testing**. **Deployment** and **Review**.

RESULTS



nt Status: Unhealthy (Caterpillars)
st: caterpillars
lution: Apply Bacillus thuringiensis (Bt)
nd-pick them.
nfidence: 63.56%

Status: Unhealthy (Caterpillars)
caterpillars
on: Apply Bacillus thuringien
dence: 63.56%

Plant Status: Unhealthy (Caterpillars)
Pest: caterpillars
Solution: Apply Bacillus thuringien
Confidence: 63.56%

RECOMMENDATIONS

- Expanding the diversifying the dataset
- Extending the pest detection to additional pest by expanding the category of the plants
- Improving handling of non-plants object

SOFTWARE TOOLS

CONCLUSION

Smart Pest Detection app uses YOLOv5 with 87.31% accuracy to detect Whiteflies and caterpillars, offering real-time solutions and QR code sharing. It may misclassify infected plants or fail on other pests/non-plants which is an object. Future updates will enhance reliability and pest coverage.