SMART PEST DETECTION FOR AN AGRICULTURAL FIELD CROP BASED ON DEEP LEARNING

By

NAVYA MARIAM PRASAD (2020-02-008)

FATHIMA HIBA K (2020-02-040)

VISHNUPRIYA V (2020-02-044)

VAISHNAVI AJAYAN A (2020-02-053)

DEPARTMENT OF IRRIGATION AND DRAINAGE ENGINEERING

KELAPPAJI COLLEGE OF AGRICULTURAL ENGINEERING AND FOOD TECHNOLOGY

TAVANUR- 679573, MALAPPURAM

KERALA, INDIA

2024

SMART PEST DETECTION FOR AN AGRICULTURAL FIELD CROP BASED ON DEEP LEARNING

By

NAVYA MARIAM PRASAD (2020-02-008)

FATHIMA HIBA K (2020-02-040)

VISHNUPRIYA V (2020-02-044)

VAISHNAVI AJAYAN A (2020-02-053)

PROJECT REPORT

Submitted in partial fulfilment of the requirement for the degree

Bachelor of technology

In

Agricultural Engineering

Faculty of Agricultural Engineering and Technology

KERALA AGRICULTURAL UNIVERSITY

DEPARTMENT OF IRRIGATION AND DRAINAGE ENGINEERING

KELAPPAJI COLLEGE OF AGRICULTURAL ENGINEERING AND FOOD TECHNOLOGY

TAVANUR- 679573, MALAPPURAM

KERALA, INDIA

2024

DECLARATION

We hereby declare that this project entitled “SMART PEST DETECTION FOR AN AGRICULTURAL FIELD CROP BASED ON DEEP LEARNING” is a bonafide record of project work done by us during the course of study and that the report has not previously formed the basis for the award to us of any degree, diploma, associateship, fellowship or other similar title of another university or society.

Place: Tavanur

Date:

Navya Mariam Prasad

(2020-02-008)

Fathima Hiba K

(2020-02-040)

Vishnupriya V

(2020-02-044)

Vaishnavi Ajayan A

(2020-02-053)

CERTIFICATE

Certified that the project entitled **"SMART PEST DETECTION FOR AN AGRICULTURAL FIELD CROP BASED ON DEEP LEARNING"** is a record of project work done jointly by **Ms. Navya Mariam Prasad (2020-02-008), Ms. Fathima Hiba K (2020-02-040), Ms. Vishnupriya V (2020-02-044), Ms. Vaishnavi Ajayan A (2020-02-053)** under my guidance and supervision and that it has not previously formed the basis for the award of any degree, diploma, fellowship or associateship to them.

Place: Tavanur

Date:

Guide: **Dr. Asha Joseph**
Professor
Dept. of IDE
KCAE&FT, Tavanur

Co-Guide: **Dr. Aiswarya L**
Assistant Professor (C)
Dept. of IDE
KCAE&FT Tavanur

Acknowledgment

ACKNOWLEDGEMENT

First of all, with an open heart, we thank Almighty for his invisible helping hand that guided us through the right way to pursue our journey to the completion of this project.

It is our prerogative to express profound gratitude and respect to our guide, **Dr. Asha Joseph**, Professor, Department of Irrigation and Drainage Engineering, KCAE&FT, Tavanur for her inexplicable support and guidance throughout our endeavor.

We are thankful to **Dr. Jayan P. R**., Dean (Agrl. Engg), and **Dr. Rema K.P.**, Professor & Head, Department of IDE, KCAE&FT, Tavanur, for his support and encouragement during the course of the project work.

We are also indebted to our co-guide, **Dr. Aiswarya L**, Assistant Professor (C), Department of Irrigation and Drainage Engineering, KCAE&FT, Tavanur, for providing us with all the guidance and support during the project work.

We remain indebted to all our teachers of Dept. of IDE of KCAE&FT, Tavanur for their support and encouragement throughout our study.

Our words of thanks to **Dr. Josephina Paul**, Assistant Professor, Dept. of BEAS, for her timely support during our project work.

We also wish to remember and gratify our Parents, who always bless us for our betterment and pray for our success.

Finally, we thank all those directly or indirectly involved in our project work.

NAVYA MARIAM PRASAD (2020-02-008)

FATHIMA HIBA K (2020-02-040)

VISHNUPRIYA V (2020-02-044)

VAISHNAVI AJAYAN A (2020-02-053)

Dedicated to the Profession of Agricultural Engineering

CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF PLATES

SYMBOLS AND ABBREVIATIONS

AI	Artificial Intelligence
AP	Average Precision
CNN	Convolutional Neural Network
CSS	Cascading Style Sheet
FN	False Negative
FP	False Positive
HTML	HyperText Markup Language
IoU	Intersection over Union
mAP	mean Average Precision
ML	Machine Learning
P	Precision
R	Recall
TN	True Negative
TP	True Positive
YOLO	You Only Look Once
YOLOv8	You Only Look Once Version 8
YOLOv8s	You Only Look Once Version 8 small
YOLOv8n	You Only Look Once Version 8 nano
YOLOv8l	You Only Look Once Version 8 large

Introduction

CHAPTER - I

INTRODUCTION

Agriculture plays an important role in the economy, and it is the backbone of the economic system for a developing country like India. Ending hunger and undernutrition are also important goals of agricultural modernization and economic transformation. However, agriculture may lose its quantity and quality when it is attacked by different insect pests. Pests and insects can damage crops, reduce yields, and increase the cost of production, which can have a significant economic impact on farmers and the overall agricultural sector. In agriculture, pest control has always been considered as the most challenging task for farmers.

India ranks second in fruit and vegetable production in the world after China (Paul *et al.*,2024). According to production data, India is one of the world's top producers of pumpkin. Cucurbita maxima or pumpkins are a member of the Cucurbitaceae family, which is widely cultivated worldwide (Syed *et al*., 2019). Plant diseases of pumpkins greatly impact production, growth, and economic well-being. Identifying pumpkin disease on the plant is a critical step in preventing a significant loss of productivity and quantity of agricultural products (Bezabh *et al*., 2024). The quantity and quality of pumpkins are seriously threatened by the spread of pests, especially the red pumpkin beetle (Aulacophora foveicollis). Red pumpkin beetle pests are very challenging to manage in different Cucurbitaceae crops.

Traditionally, farmers have relied on manual scouting and visual inspections to detect and identify these issues. However, this approach is time-consuming, subjective, and often prone to errors. So, in developing countries, where agriculture is often a key component of the economy, the damage can be devastating. About 30 to 40% of the global agricultural production is being destroyed by pests (Junaid and Gokce., 2024). The cost of controlling pests can be high, and the use of chemical pesticides can have environmental and health impacts. Although there are many sophisticated technologies in the field of agriculture, there is still no proper technology to detect the problems related to pests. Achieving insect classification with higher accuracy in real-time fields is a challenging issue in major agriculture field crops, in the presence of shadows, leaves, dirt, branches, flower buds, etc.

Traditional monitoring methods have cost implications because the variables cannot be measured simultaneously at all monitoring points. These pests may go undetected by the labourers as they are hard to locate during nighttime. So, as a preventative measure, farmers spray pesticides in bulk which is not only harmful for the crops but also harmful for the environment. Additionally, insects and bugs become resistant to pesticides with continuous exposure, resulting in heavier pesticide usage. Extreme use of pesticides can result in severe water & soil contamination and can also intoxicate plants with harmful chemicals. To avoid the drawbacks of traditional monitoring, emerging techniques have been adopted that involve the application of computer vision techniques to automatically detect and identify insect pests (Lello *et al*., 2023).

The AI based pest detection system is an effective method to provide support to farmers in order to reduce pests. With the advent of image processing techniques, computer vision, and machine learning, a new era of automated pest and disease identification in agriculture has emerged, revolutionizing the way farmers manage these challenges (Ngugi *et al*., 2021). Image processing techniques leverage the capabilities of computer vision algorithms to analyze and interpret images. By harnessing this technology, automated pest and disease identification systems can provide rapid and accurate diagnoses, enabling timely interventions and minimizing crop losses (Nagar and Sharma, 2020). The automatic detection of insects and pests from poor photos is now possible because of advances in machine learning and deep learning (Anwar *et al*., 2023).

Image processing, in combination with advanced algorithms and machine learning, has opened up a world of possibilities for accurate and efficient pest and disease identification. By analyzing digital images of plants or crops, these automated systems can rapidly and accurately detect and classify pests, diseases, or symptoms of stress. This technology offers farmers a powerful tool to make informed decisions, implement targeted interventions, and ultimately safeguard their crops against potential losses (Kasinathan and Uyyala, 2021). The application of automated pest and disease identification systems using image processing in agriculture thus offers several significant benefits: 1. Early Detection and Timely Intervention 2. Increased Accuracy and Efficiency 3. Enhanced Sustainable Farming Practices 4. Integration with Precision Agriculture.

In recent years, deep learning a machine learning technique based on artificial neural networks (ANN) has revolutionised the field of machine vision. Convolutional neural networks (CNNs), a subset of artificial neural networks (ANNs) with convolutional layers, are one of the main forces behind machine vision since they can extract features straight from the raw values of pixels without the need for hand-engineered features. Deep learning seems to be more successful in diagnosing crop problems in agricultural production. Deep learning using CNN is widely used in agricultural fields for plant pest detection. The CNN with a traditional machine learning algorithm was a significant make attempt to detect plant diseases (Li *et al*., 2020). Ai *et al*. (2019) proposed that CNN was used effectively to detect crop bugs automatically. Ferentinos *et al*., (2018) discovered deep learning approaches by healthy leaf images and diseased plants. CNN models were created to conduct plant disorder identification and diagnosis. (Rustia *et al*., 2020) implemented deep learning models for the automatic object detection of pests from traps. Many research studies have proved that the CNN model provides the highest classification accuracy of more than 90%. (Chen *et al*., 2020)

Since its introduction by Joseph Redmon (Redmon *et al*., 2016), YOLO (You Only Look Once) has grown to be one of the most well-liked real-time machine vision algorithms. It does this by instantly identifying specific objects in videos, live feeds, or images using features learned from a deep convolutional neural network, all while operating much faster than other networks and maintaining accuracy. YOLOv8, the highest performing YOLO model released very recently, setting new standards in real-time detection and segmentation, will soon become a hot topic in the research world for solving complex issues by providing simple and effective AI solutions.

Thus, by leveraging the power of computer vision and deep learning, automated pest and disease identification systems offer accurate and timely identification, enabling farmers to implement appropriate intervention measures promptly. With the ability to detect pests and diseases at their early stages, farmers can reduce crop losses, optimize resource utilization, and embrace sustainable farming practices. The continued advancement of image processing technology, thus, holds great promise for a more resilient and productive agricultural industry in the future. By optimizing pest and disease control measures, farmers can mitigate environmental impacts, enhance crop quality, and ensure the safety of agricultural produce. Furthermore, automated pest and disease identification systems seamlessly integrate with other precision agriculture technologies like variable rate application and site-specific management. This integration facilitates precise and localized interventions, optimizing resource utilization and maximizing agricultural productivity (Abiri *et al.*, 2023).

In view of all the above facts, the present study entitled "Smart pest detection for an agricultural field crop based on deep learning" was undertaken with the following specific objectives:

- Development of a deep learning-based object detection model for pest detection in an agriculture field crop.
- Performance and accuracy assessment of the model.
- Development of a web application for real-time pest detection.

Review Of Literature

CHAPTER – II

REVIEW OF LITERATURE

Insect pests are one of the main causes affecting agricultural crop yield and quality all over the world. Various methods have existed for pest detection since ancient times. Most of these methods are time-consuming and expensive. Pest detection using image processing is a novel practice in this sector. Deep learning models can make the tedious pest detection process effortless. Hence, in this chapter, a review of literature referring to an analysis of various deep learning techniques for pest detection, application of Roboflow in dataset creation & annotation, and application of YOLO for pest detection carried out by various researchers all over the world were briefly explained under the following subheads.

1. Smart pest detection using different Machine Learning(ML) techniques
2. Application of Roboflow in dataset creation and annotation
3. Application of YOLO for pest detection

2.1 SMART PEST DETECTION USING DIFFERENT ML TECHNIQUES:

Chithambarathanu and Jeyakumar (2023) studied crop pest detection as a crucial step in precision agriculture; they used Convolutional Neural Networks (CNNs) and transfer learning that showed promising results in detecting pests from images, with accuracy rates exceeding 90%. Researchers have utilized datasets such as PlantVillage and Crop Pest Dataset for training and testing CNN models. Machine learning algorithms like Support Vector Machines (SVM) and random forests have also been employed for feature extraction and classification. Studies have also emphasized the importance of data augmentation, hyperparameter tuning, and ensemble methods for improving model performance. While these approaches have demonstrated potential, challenges persist, including data quality, class imbalance, and scalability.

Kasinathan *et al.,* (2021) studied the modern machine-learning techniques that have revolutionized insect classification and detection in field crops, enhancing precision. They studied deep learning architectures like CNNs and Recurrent Neural Networks (RNNs) to classify insects from images, achieving high accuracy rates (>95%). Transfer learning and fine-tuning pre-trained models have also been successfully employed. They utilized datasets like IP102 and Crop Insect Dataset for training and testing models. Additionally, machine learning algorithms like SVM and Random Forests have been used for feature extraction and classification. Studies have emphasized the importance of data augmentation, hyperparameter tuning, and ensemble methods for improving model performance.

Hadipour *et al.,* (2023) studied the intelligent detection of citrus fruit pests using machine vision and CNNs through transfer learning. By leveraging pre-trained CNN models and fine-tuning them for citrus pest detection, researchers got high accuracy rates above 95%. Machine vision systems capture images of citrus fruits, and CNNs extract features and classify pests. Transfer learning adapts pre-trained models to specific citrus pest detection tasks, reducing training time and improving performance. This integrated approach enables automated and efficient citrus pest detection, allowing for timely pest management decisions.

Wang *et al*., (2021) studied field detection of tiny pests from sticky trap images using deep learning in agricultural greenhouses has been proposed as a method for pest detection and control. Insect traps are essential for monitoring and controlling pest populations in greenhouses, and sticky traps are a common type of trap used to capture and monitor insect pests. Deep learning techniques, such as CNNs, can be used to automatically detect and classify insects from images of sticky traps, learning features and patterns that distinguish between insects and non-insects. Once trained, CNNs can classify new images of sticky traps and detect the presence of insects, improving the efficiency and effectiveness of pest detection and control. This method can potentially reduce the use of chemical pesticides and minimize harm to the environment and human health.

Singh *et al.,* (2021) studied detecting diseases and pest infections in coconut trees using deep learning. They employed CNNs and transfer learning to classify images of coconut trees into healthy or infected categories. Studies have demonstrated high accuracy in detecting diseases such as root wilt, leaf blight, and pest infestations like rhinoceros beetles and red palm weevils. The study shows an accuracy of 96.5% in detecting root wilt disease using a CNN model. Similarly, detection of rhinoceros beetle infestations with an accuracy of 93.2% using transfer learning. These approaches can potentially revolutionize coconut tree disease management, enabling early detection and prompt action to prevent crop damage. Moreover, deep learning-based systems can be integrated with drone technology and IoT sensors for large-scale monitoring, making them a valuable tool for sustainable agriculture and food security.

2.2 APPLICATION OF ROBOFLOW IN DATASET CREATION AND ANNOTATION

Brucal *et al*., (2023) studied that the development of tomato leaf disease detection using YOLOv8 via Roboflow 2.0 has significantly advanced precision agriculture. Tomato leaf diseases, such as septoria leaf spot and early blight, can significantly reduce crop yields and affect fruit quality. They proposed deep learning techniques, specifically YOLOv8, to detect tomato leaf diseases with high accuracy. Roboflow 2.0, a machine learning platform, has been employed to streamline the development process. By leveraging YOLOv8's object detection capabilities and Roboflow 2.0's automation features, researchers have achieved efficient and accurate disease detection, enabling timely interventions and improved crop management.

2.3 APPLICATION OF YOLO FOR PEST DETECTION

Melo *et al.*, (2024) developed a lightweight model on YOLOv8 has been developed for detecting the Neotropical brown stink bug, Euschistus heros, in soybean fields. This model leverages transfer learning and fine-tuning to achieve high accuracy and efficiency in detecting this significant soybean pest. Compared to existing models, this approach demonstrates improved performance, with an accuracy of 97.3% and a detection speed of 35 frames per second. The model's lightweight architecture and optimized computational requirements make it suitable for deployment on edge devices, enabling real-time monitoring and control of

Euschistus heros in soybean fields. This innovation has significant implications for precision agriculture and integrated pest management, allowing farmers to quickly identify and respond to pest infestations, reducing crop damage and pesticide use.

Onler (2021) studied real-time pest detection using YOLOv5, enabling farmers to swiftly identify and respond to pest infestations. YOLOv5, an advanced object detection algorithm, has been successfully adapted for pest detection in various crops, including soybeans, tomatoes, and potatoes. This approach leverages deep learning techniques to detect pests with high accuracy and speed, outperforming traditional methods. YOLOv5's real-time capabilities, with detection speeds of up to 30 frames per second, enable farmers to monitor fields continuously, facilitating prompt action against pest outbreaks. The study concluded that the effectiveness of YOLOv5 in detecting various pests, including aphids, whiteflies, and spider mites, with accuracy rates exceeding 90%. This technology has the potential to significantly reduce crop damage, pesticide use, and economic losses, making it a valuable tool for sustainable agriculture and food security.

Slim *et al.,* (2023) studied smart insect monitoring using YOLOV5 for detecting Mediterranean fruit fly (Ceratitis capitata) and Peach fruit fly (Bactrocera zonata). It was found that YOLOV5-based systems can accurately identify and classify these pests in real-time, achieving detection accuracies exceeding 95%. This innovative approach has been praised for its ability to process images at a rate of 30 frames per second, making it suitable for large-scale monitoring applications. Researchers have highlighted the potential of YOLOV5-based monitoring to revolutionize integrated pest management strategies, enabling farmers to respond promptly to fruit fly infestations and minimize crop damage. Moreover, this technology has been shown to reduce labor costs and enhance precision, promoting sustainable agriculture practices.

Yang *et al.,* (2024) studied a YOLOv8-based method has been proposed for rice pest recognition, demonstrating enhanced accuracy and efficiency. This approach leverages transfer learning and fine-tuning to detect various rice pests, including planthoppers, leafrollers, and rice borers. Studies have shown that the improved YOLOv8 model outperforms existing methods, achieving an accuracy of 97.3% and a detection speed of 35 frames per second. The model's improved performance is attributed to its ability to learn robust features and adapt to complex environments. Additionally, this method is effective in detecting pests at various growth stages and under different lighting conditions. The real-time detection capability and high accuracy of this approach make it a valuable tool for rice crop management, enabling farmers to take prompt action against pest infestations and reduce crop damage.

Zhu *et al.*, (2024) studied a significant research study on developing a detection model for common soybean pests in complex environments using CBF-YOLO (Class Balanced Focal Loss-YOLO). They studied the challenge of detecting pests in traditional methods. The CBF-YOLO model has been proposed to address class imbalance and improve detection accuracy. The study contributes to the existing literature by developing a CBF-YOLO model for detecting common soybean pests in complex environments, evaluating the model's performance on a diverse dataset, and demonstrating the effectiveness of the CBF-YOLO model in improving detection accuracy and addressing class imbalance issues. Overall, the study advances the field of agricultural computer vision and pest detection, offering a valuable tool for farmers and researchers to monitor and manage soybean pests more efficiently.

Tian *et al.*, (2023) conducted an approach to small target pest detection using a multi-scale dense YOLO (MD-YOLO) model, addressing the challenge of detecting small pests in images, crucial for agricultural monitoring and management. Existing methods struggle with small targets, and rely on manual annotation, limited by dataset size and quality. The MD-YOLO model integrates multi-scale features and dense connections into a YOLO framework, demonstrating improved detection performance on a pest dataset, showcasing its potential for small target pest detection in agricultural applications. The result showed an 88.1 IoU and 79.1 F1 score. This study contributes to the development of more accurate and efficient pest detection methods, essential for sustainable agriculture and food security. By leveraging MD-YOLO, researchers and practitioners can better detect and manage pests, reducing crop damage and promoting environmentally friendly farming practices.

Materials And Methods

CHAPTER - III

MATERIALS AND METHODS

The materials used and methodology adopted for the development of smart pest detection in an agricultural field crop using deep-learning object detection are explained in this chapter. The proposed methodology comprises three main parts, which are respectively responsible for data collection, developing a deep learning model, and creating a user-friendly web application for real-time pest detection. The details are explained under the following subheads

3.1 DETAILS OF STUDY AREA, CROP AND PEST

The site is situated on the cross point of 10˚51' 23" latitude75˚59' 18" longitude at an altitude of 10m above mean sea level. This study area comes under the farm area of the Instructional Farm KCAE&FT, Tavanur. The field data collection was conducted in the pumpkin field of Instructional Farm, where there was a high count of red pumpkin beetle (Aulacophora torticollis) during the pumpkin cultivation.

Table 3.1 Details of the pumpkin field in the KCAE&FT Instructional Farm

Area	1350 m^2
Length	50m
Width	27m
Crop	Pumpkin
Pest	Red Pumpkin Beetle

The Red Pumpkin Beetle was a significant agricultural pest primarily found in Asia and Australia. These beetles were notorious for infesting cucurbit crops, including pumpkins, squash, cucumbers, and melons. Red pumpkin beetles cause damage to crops by feeding on leaves, stems, and fruits, leading to defoliation, stunted growth, and reduced yield. Both adult beetles and larvae contribute to crop damage, with larvae often tunnelling into plant tissues and causing additional harm.

Identifying features of the red pumpkin beetle include its vibrant red colouration, elongated body shape, and distinctive black markings. Adult beetles typically measure around 6 to 8 millimeters in length and have prominent antennae and legs. They were active during the day and feed voraciously on plant foliage, stems, and fruits, causing damage to the host plants. By observing these identifying features, individuals can recognize and distinguish red pumpkin beetles from other beetles.

3.2. AN OVERVIEW OF PROPOSED DEEP LEARNING-BASED APPROACH FOR PEST DETECTION

The overall workflow, as shown in Fig. 3.1 consists of the following components: 1. Data acquisition 2. Dataset preprocessing 3. Data split into training, validation, and testing set 4. Model development 5. Model validation and testing 6. Performance and accuracy assessment. 7. Creation of an HTML webpage for real time pest detection. The data were acquired by taking actual images from the field using a mobile camera and from the website ‘Shutterstock’, a free website that can download the best royalty-free images, including photos, vectors, and illustrations. The data preprocessing and data augmentation were done in Roboflow. Also, data training, validation, testing, feature extraction, feature fusion, and object detection were done using YOLOv8, a deep-learning model designed for real-time object detection in computer vision applications. ‘Flask’, a lightweight, modular, and flexible web development library for Python, was used for creating web applications for real-time pest detection. The details of each component in the overall conceptual framework were explained under the following subheadings.

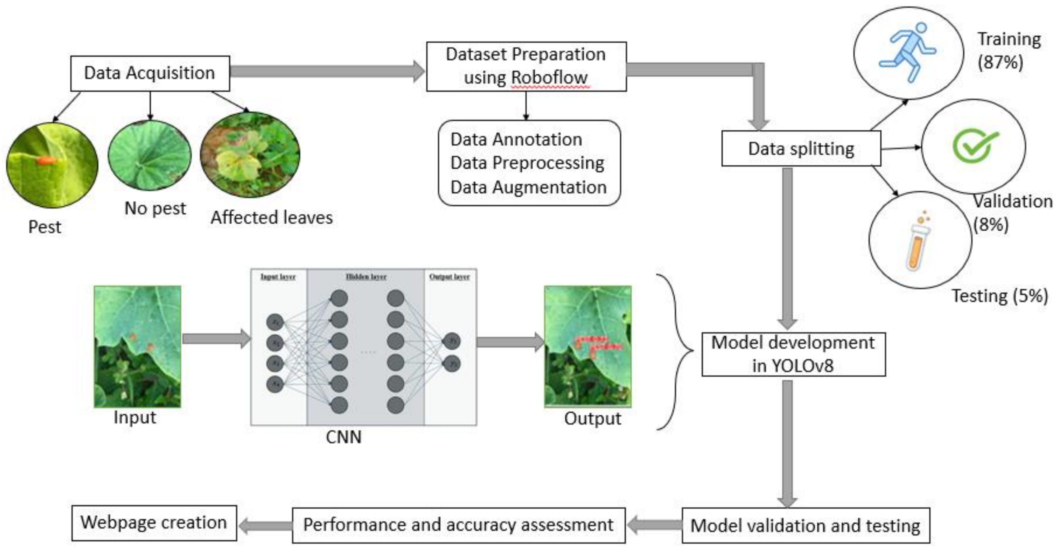


Fig. 3.1 An illustration of the proposed smart pest detection approach.

3.3 DATA ACQUISITION AND DATA SET USED

Data collection lies at the heart of developing robust and accurate object detection systems, serving as the foundation upon which models are trained and evaluated. The quality and diversity of the dataset significantly influence the performance and generalization ability of the trained models. The collection of a maximum number of photos was the first step in creating a well-defined dataset. Since these beetles were active in the early morning, for the first 8 weeks (from 22-02-2024 to 19-04-2024), a survey was conducted every three days, and images were collected using a smartphone (iPhone 7 plus) camera (Dual 12MP wide-angle and telephoto cameras) with a median image ratio of 4032 × 3024 pixels.

Images were captured from different angles and resolutions to increase the dataset's accuracy. The images were also captured at different times of the day, such as morning (6.30 AM), noon (12.00 PM), and afternoon (4.30 PM), under a variety of lighting conditions, including direct sunlight, indirect sunlight, shade, and mixed lighting conditions. Most images were taken in the morning time because these pests were plenty in number during this time. Images of beetles resting on leaves, stems and flower buds were taken. Along with this, pictures of healthy and unhealthy leaves were also captured to avoid false detection by the model. The captured images were saved daily in a Google Drive folder for easy access. The capturing of images was acquired by keen observation and meticulous planning. Thus, a total of 570 images were captured from the field.

Apart from the field, 40 images were collected from an online website called ‘Shutterstock’. The accuracy of training increases with the increase in the number of data sets. Thus a total of 610 images were added to Google Drive for easy access and storage and data is available in the following link https://drive.google.com/drive/folders/1GcwdZPiTA5BE46UcSr2A7NQRMRHHdRF?usp=sharing.

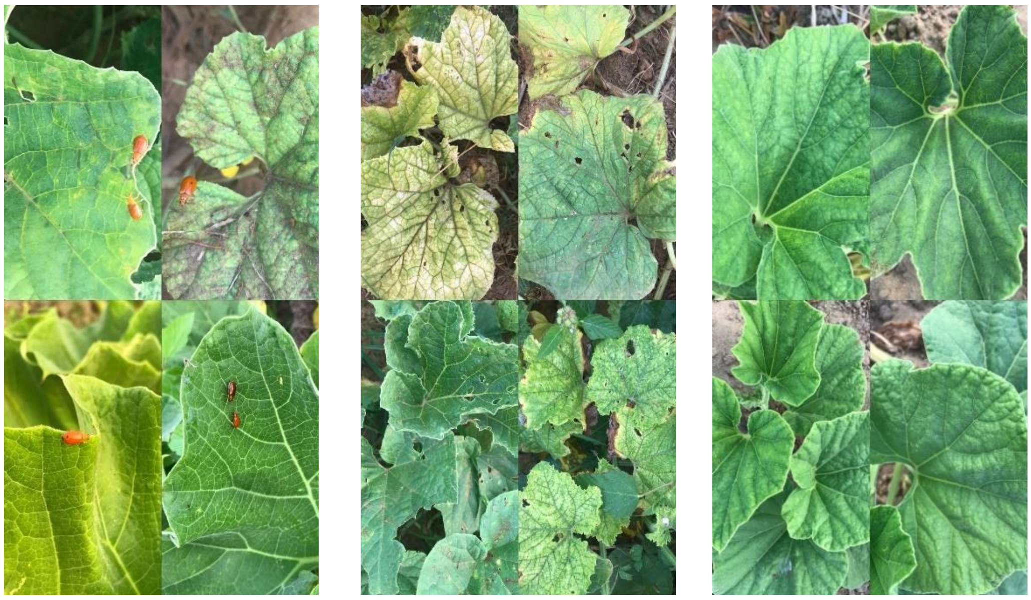


Plate 3.1 Sample images captured from the KCAE&FT Instructional Farm

3.4 CUSTOMIZED DATASET PREPARATION

The customized data set preparation consists of data annotation, data preprocessing and data augmentation. ‘Roboflow’ was used as the customized data set preparation conversion tool.

3.4.1 Roboflow

Roboflow offers a full range of tools and services with the goal of streamlining and accelerating the development of computer vision models. It is a useful tool for programmers and companies who want to use computer vision in their applications and projects. Roboflow key features and capabilities include data annotation and preparation for labelling movies and images, both of which are necessary for teaching computer vision models to identify and locate objects in images (https://Roboflow.com/). Hence, Roboflow was used to prepare and manage image datasets for the machine-learning process. Roboflow offers a variety of data augmentation strategies that help machine learning models become more resilient and general. Roboflow supports data sets in a format compatible with Computer vision models, which are usually trained using deep learning frameworks like TensorFlow, PyTorch, YOLO, etc. These frameworks offer a wide range of pre-built components and optimizations specifically tailored for computer vision tasks, which significantly accelerates the model development process. Hence, Roboflow eases the computer vision task in the field of deep learning. It empowers developers to build their computer vision applications, no matter their skill set or experience. It supports object detection and classification models. Hence Roboflow was selected for this study

3.4.1.1 Dataset preparation using Roboflow

The first step in creating a customized dataset using Roboflow is to gather the raw data. For this, the images of red pumpkin beetles were captured from the KCAE&FT Instructional Farm. Once the data is collected, it can be uploaded to Roboflow's platform for further processing. A total of 610 images were uploaded to Roboflow.

3.4.1.2 Data annotation using Roboflow

After uploading the images, it must be annotated to provide labels or bounding boxes for objects of interest within the images or frames. The annotation process consists of marking the object to be detected on each image by taking it into a rectangle. More than one object can also be marked on the image. Information about the objects marked on each image was stored in a text file with the same name as the related image. The Roboflow annotated file data includes the class of the marked objects, which drawn as a bounding box (x and y-axis, and width and height). The rectangle coordinates were normalized between 0 and 1 to be independent of the image size. Thus, each image was evaluated according to its size. Images with different sizes were used in the object detection system (Önler, 2021).

Roboflow offers a range of annotation tools to facilitate this process, including bounding boxes, polygon, and segmentation tools. In this study, the object to be detected was marked with bounding boxes. For easy object detection, the data was classified into 3 classes, viz. pest (Yellow box), which represents the red pumpkin beetle infestation on the pumpkin, no pest (purple box), which represents a healthy pumpkin leaf and pest-affected leaf (red box) which represents a pumpkin leaf affected by red pumpkin beetle.

3.4.1.3 Dataset preprocessing using Roboflow

In image pre-processing, image enhancement techniques were applied to reduce noise in the images and sharpen the images for better accuracy. It improves the image quality for better detection and classification of insects (Kasinathan *et al.*, 2021). Before fitting the model, the dataset was pre-processed to get good training, validation, and testing results and to avoid overfitting the model. Hence the following two preprocessing steps were applied using Roboflow.

*i. Converting to Grayscale***.**

Grayscale conversion is applied to reduce complexity and decrease the amount of time to process data. It is helpful for activities like image analysis or model training as it makes colour information easier to understand. Using the grayscale conversion option in Roboflow, the uploaded datasets were converted to grayscale.

ii. Resizing.

Resizing is done to enhance the raw image data. To resize the uploaded images using Roboflow, the preferred resizing option (800x800) was selected. 610 images were resized from 4032x3024 px to 800x800 px. Thus, the images were automatically downsized by Roboflow, which helped to standardize the dataset's dimensions for further processing or model training.

3.4.1.4 Data augmentation using Roboflow

Since fewer insect images were available in the dataset, image augmentation was applied to enrich the training dataset. Data augmentation is done to enhance the diversity and robustness of the dataset, which is essential for accurate training of machine learning models. Roboflow provides a comprehensive set of data augmentation techniques that users can apply to their dataset with just a few clicks. Thus, to enrich the dataset, the images were further augmented.

The various augmentation techniques applied in this study include:

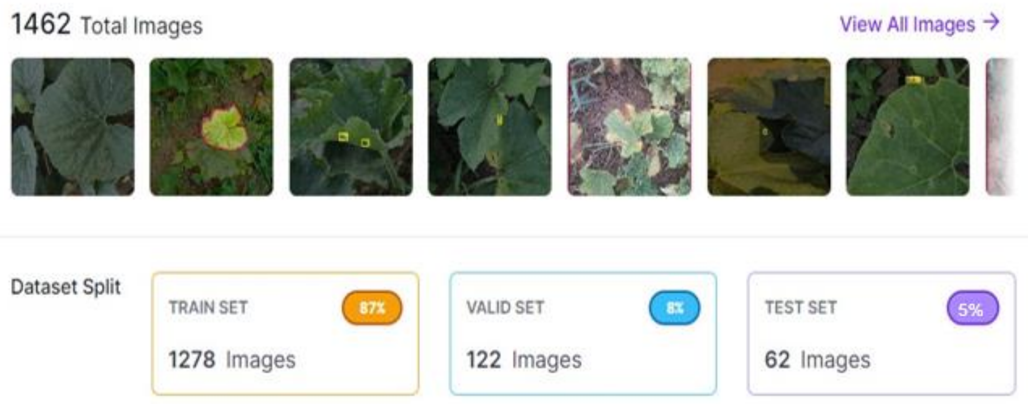
- Rotation: Rotate images by a specified angle to introduce variation which makes the model more resilient to camera roll.
- Flipping: Flip images horizontally or vertically to simulate different perspectives which adds horizontal or vertical flips to make the model less sensitive to subject orientation
- Noise Addition: Add random noise to images to make them more resilient to noise in real-world scenarios.
- Colour Adjustment: Adjust brightness, contrast, and saturation levels to account for variations in lighting conditions.
- Shearing: Shearing is a geometric transformation that skews or tilts an image by a specified angle, adding variability to perspective to make the model more resilient to the camera and subject pitch and yaw.
- Crop: Crop an image refers to extracting a rectangular Region Of Interest (ROI) from the original image.

- Bounding Box Flip: This flips the bounding box coordinates (x, y, w, h) of objects in an image, simulating mirror-like reflections.
- Mosaic: Mosaic images that combine multiple images into a single image, creating a mosaic pattern that is essential to divide the resultant dataset into training, validation, and test sets.
- Hue: Adjusts the colour tone of an image by shifting the hue value in the HSV (Hue, Saturation, Value) colour space which randomly modifies the vibrancy of the colours in the images
- Blur: This technique simulates real-world image degradation by applying a blur filter to images, reducing their sharpness and clarity which adds random Gaussian blur to help your model be more resilient to camera focus.
- Exposure: Adjusts the brightness and contrast of images to simulate varying lighting conditions.
- Cutout: Randomly removes rectangular regions of an image, simulating real-world occlusions and object cutouts.

The combined multiple augmentation techniques were applied to create a diverse set of training examples to improve the model's generalization capabilities. Roboflow gave us the ability to define the ideal split ratio. One-tenth went towards testing, one-fifth went towards validation, and seven-tenth went towards the training set. Finally, Roboflow facilitates data export in various formats that are compatible with popular YOLO frameworks. After applying augmentation on the training set, the datasets created 852 more augmented trained images besides the original. Thus, the combined dataset consists of a total of 1462 images for training the model.

3.4.1.5 Splitting dataset using Roboflow

The entire data was divided into 3 sets, having a training set (87%) containing 1278 images, a Validation set (8%) containing 122 images, and a testing set (5%) containing 62 images by Roboflow. The ratio between the number of images in each class were made the same in order to detect pests more accurately and reduce the impact of class imbalance (a dataset within which one or some classes have much greater number of datasets than others) on the model's performance.


1462 Total Images

Dataset Split

TRAIN SET 87%
1278 Images

VALID SET 8%
122 Images

TEST SET 5%
62 Images

Fig. 3.2 Splitting of Dataset into train set, valid set, and test set in Roboflow

i. Training Set

This is the actual dataset from which a model trains. i.e. the model sees and learns from this data to predict the outcome or to make the right decisions. Most of the training data was collected from several resources and then pre-processed and organized to provide proper performance of the model. The training data type hugely determines the model's ability to generalize. i.e., the better the quality and diversity of training data, the better the performance of the model. As per Roboflow this training data set should be 87% of the total data available for the project. Hence this study selected 1278 images for training the model.

ii. Validation Set

The validation set is used to fine-tune the hyperparameters of the model and is considered a part of the training of the model. The model only sees this data for evaluation but does not learn from this data, providing an objective unbiased evaluation of the model. The validation dataset can also be utilized for regression by interrupting the training of the model when the loss of the validation dataset becomes greater than the loss of the training dataset. i.e. reducing bias and variance.

This data should be approximately 10-15 % of the total data available for the project, but this can change depending on the number of hyperparameters. i.e., if the model has quite many hyperparameters, then a large validation set gives better results. However, this study selected 8% (122 images) for validation. Whenever the accuracy of the model on validation data is greater than that on training data, then the model is said to be generalized well.

iii. Testing Set

This dataset is independent of the training and validation sets, but has a similar probability distribution of classes. It is used as a benchmark to evaluate the model and is applied only after completing the model training. A testing set is usually a properly organized dataset with all kinds of data for scenarios, the model would probably face when used in the real world. If the model's accuracy on training data is greater than that on testing data then the model is said to be overfitting. If the model has not learned the patterns in the training data well and is unable to generalize well on the new data then the model is said to be underfitting. If both the training data error and the testing data error are minimal then the model is said to be good fitting. The data required should be approximately 5-10% of the total data available for the project. However, the Roboflow study selected 5% of images (62 images), which were the data taken from the actual field to test the model. The details of the data and its split up were shown in Table 3.2.

Table 3.2 Details of data and its split up for training, validation and testing

Data source	No. of images	Data set Split		
		Training (87%)	Validation (8%)	Testing (5%)
From crop field	570	1278	122	62
From Shutterstock	40			
By data augmentation	852			

3.4.1.6 Dataset health check

Roboflow offers a comprehensive data health checkup to ensure data integrity, consistency, quality, label quality, data balance, distribution, size, and storage. Running a data health checkup can identify and address potential issues, ensuring the dataset is reliable, efficient, and optimized for machine learning model training. This study adopted the following data health checkup methods.

- Annotation heatmap
- Histogram equalization

i. *Annotation heatmap*

In Roboflow, the annotation heat map is a powerful visualization tool that reveals the distribution and density of annotations in your dataset, helps to optimize annotation strategy and improves model performance. By analysing the annotation heat map, one can refine the annotation process, ensure more accurate and balanced labels, and enhance the overall quality of machine learning models, leading to better decision-making and outcomes. In the heatmap, the blue region (cool colour) has a low annotation density, and the green region (warm colour has a high annotation density allowing us to quickly identify hotspots, class imbalance issues, and potential biases (Jiang *et al*., 2024).

ii. *Histogram equalization*

Histogram equalization is a visualization tool that provides insights into the distribution of object instances in a dataset. This histogram displays the count of objects on the y-axis and the object classes (viz. pest, affected, no pest) on the x-axis, represented as a bar chart, enabling understanding of the object class distribution, identifying class imbalance issues, and visualizing the frequency of each object class. By analysing the histogram, the dataset can be refined to address class imbalance, adjust model hyperparameters for improved performance, identify rare or underrepresented classes, and enhance object detection and classification accuracy (Jiang *et al*., 2024).

3.5 EXPERIMENTAL ENVIRONMENT CONFIGURED FOR DEEP LEARNING OBJECT DETECTION MODEL DEVELOPMENT

3.5.1 Hardware used

Table 3.3 displays the configuration of the experimental environment used in the study. The hardware primarily comprises a high-performance computer. The mainframe computer is equipped with an Intel(R) Core (TM) i5-12450H processor and a graphic card GeForce RTX 3050 6GB. GPU is a specialized processor, commonly used to accelerate graphics rendering and computational tasks in machine learning. CUDA, a parallel computing platform by NVIDIA that enables developers to use GPUs for general-purpose processing, significantly speeding up tasks that involve heavy computation.

Table 3.3 Experimental environment configuration needed for model building.

Item	Category	Description
Hardware	Central Processing Unit	Intel(R) Core (TM) i5-12450H
	Random Access Memory	8.00 GB
	Solid State Drive	SAMSUNG MZVL2512HCJQ-00BH1
	Graphics card (GPU)	NVIDIA GeForce RTX 3050 6GB
	CUDA	12.5
Software	Operating System	Windows 11
	Programming Language	Python 3.10.11
	Coding Software platforms	Colab PyCharm 2024.1 Pytorch
	Software	Open CV NumPy Ultralytics Flask
	Object detection model	YOLO

3.5.2 Details of software used

i. Python

It is a high-level, interpreted programming language that is easy to learn and understand. It is a versatile language that can be used for various purposes such as web development, data analysis, artificial intelligence, automation, and scientific computing. Python is known for its simplicity, readability, and large community support, making it an ideal language for beginners and experienced programmers a like.

Python has a vast collection of libraries and frameworks that make it easy to perform various tasks, such as data analysis, web development, and machine learning. Some popular libraries include NumPy, pandas, and scikit-learn for data analysis, and TensorFlow and Keras for machine learning. Python is also a cross-platform language, meaning that programs written in Python can run on multiple operating systems, including Windows, macOS, and Linux. Overall, Python is a powerful and versatile language that is well-suited for a wide range of applications. Hence, we selected python as the programming language, also YOLO is supported by python for object detection using a Convolutional Neural Network.

Libraries in Python are a collection of code that makes everyday tasks more efficient. In this study, we used library functions, such as OpenCV, NumPy and Flask.

ii. Google Colab

Google Colab is a free, web-based platform for data science and machine learning that offers a Jupyter notebook environment with unlimited storage and computing resources including GPUs (Graphics Processing Units) and TPUs (Tensor Processing Units). The Google Colab platform in Python programming was used to run the custom-trained model. With Colab, users can write and execute Python code for data cleaning, visualization, and modeling without setting up a local environment. It features free access to GPU acceleration for fast training of machine learning models, integration with Google Drive for seamless access to data and files, and support for popular libraries like TensorFlow, PyTorch, and scikit-learn. Additionally, Colab allows for real-time collaboration and sharing of notebooks, automatic saving and versioning of work. Thus it is a collaborative platform that offers a multitude of resources for machine learning endeavors to learn and improve its object detection capabilities.

iii. PyCharm

PyCharm is a popular Integrated Development Environment (IDE) for Python programming, offering a comprehensive set of tools for coding, debugging, and testing Python applications. It features intelligent code completion, advanced debugging and testing tools, support for web development frameworks like Django and Flask, support for scientific computing and data science libraries like NumPy and Pandas. PyCharm comes in two editions: Community (free) and Professional (paid), with the latter adding additional features like web development, database support, and scientific computing tools. Using PyCharm, developers can improve coding efficiency and productivity, enhance code quality and maintainability, streamline debugging and testing processes, and support a wide range of Python libraries and frameworks, all while enjoying cross-platform compatibility on Windows, macOS, and Linux.

iv. PyTorch

PyTorch is an open-source deep learning framework developed by Facebook's AI Research Lab (FAIR). It provides a flexible and efficient platform for developing and deploying machine learning and deep learning models. PyTorch is installed for supporting GPU and repository uses in YOLOv8.

v.Open CV

OpenCV (Open-Source Computer Vision Library) is an open-source computer vision and machine learning software library. OpenCV is used to process frames from the camera and annotate the detected objects with bounding boxes

It is the fundamental building block of the library, providing low-level operations for manipulating and processing images and matrices. It provides a wide range of tools and algorithms for tasks such as object detection, facial recognition, and image transformation in image processing. It supports multiple languages, including Python, Java, and C++.

vi.NumPy

NumPy is the essential package for computation in Python. It's a Python library that provides a multidimensional array object, multihued deduced objects (masked arrays and matrices), and a variety of routines for fast operations on arrays, including mathematical, logical, shape manipulation, sorting, opting, I/ O, discrete Fourier transforms, basic linear algebra, basic statistical operations, random simulation and much more. NumPy is an input format for YOLOv8 and can load and save images as JPEG or TIFF. The images were stored as a multi-dimensional array. Image resizing was done by changing the dimensions of the NumPy array.

The predictions are evaluated with the help of Numpy, NumPy library plays a crucial role in the object detection process. It enables efficient data preprocessing, and feature extraction from images, allowing for the generation of object proposals and computation of bounding box overlap and Intersection Over Union (IoU). NumPy's array operations facilitate the calculation of loss functions during model training. Additionally, it stores model weights and biases, enabling efficient matrix multiplications and neural network computations. By leveraging NumPy's capabilities, object detection algorithms can process and analyze large datasets, leading to accurate and efficient object detection in images and videos.

vii.Ultralytics

Ultralytics is a Python library for machine learning and computer vision tasks, particularly designed for YOLO (You Only Look Once) object detection models. It provides a simple and efficient interface for tasks like image processing, object detection, and model training, making it a popular choice for applications like object detection, image processing, model training and evaluation.

viii.Flask

Flask is a lightweight, modular, and flexible web development library for Python that enables the building of scalable and efficient web applications. It provides a simple and easy-to-use API for building web services, APIs, and web applications, focusing on flexibility and extensibility. Flask is ideal for building small to medium-sized web applications, prototyping, and proof-of-concepts.

3.5.3 YOLO object detection model

The foundational architecture for training the computer vision models in this research is the You Only Look Once (YOLO) architecture. Renowned for its high degree of accuracy while maintaining a compact model size, YOLO has become prominent in the field of computer vision. YOLO is a new wave of AI in the field of object detection. It is one of the most popular model architectures and object detection algorithms. YOLO is a real-time object detection algorithm developed by Joseph Redmon and Ali Farhadi in 2015.

Object detection using YOLO is performed by deploying convolutional neural networks. In order to detect and classify objects, YOLO uses a neural network with single forward propagation. The algorithm has exceptional speed and accuracy for detecting and classifying multiple objects in an image, video, or in real-time. The YOLO algorithm has many variants. In this study, YOLO is applied to detect multiple pests. It is a single-stage object detector that employs a single Convolutional Neural Network (CNN) to predict the bounding boxes and class probabilities of objects in input images. YOLO divides the input image into a grid of cells, and, for each cell, predicts the probability of an object's presence and the object's bounding box coordinates. The algorithm uses end-to-end neural networks that make predictions of bounding boxes and class probabilities all at once and provides real-time object detection. It is a very popular algorithm because of its speed and accuracy. YOLO has been developed in several versions such as YOLOv1, YOLOv2, YOLOv3, YOLOv4, YOLOv5, YOLOv6, YOLOv7 and YOLOv8. Ultralytics, the entity responsible for the inception of YOLO ushered in a new era with the introduction of YOLOv8 in January 2023. Because of the following Key features such as accuracy, speed, backbone, and supervised learning (class label), YOLOv8 was selected in this study for model development (insert table)

3.5.3.1 YOLO-v8

There are many algorithms and models for pest detection. This study selected the YOLOv8 algorithm developed by Ultralytics recently on 10 January 2023. YOLO is currently the most popular real-time object detector, which can be widely accepted for the following reasons: a) Lightweight network architecture. b) Effective feature fusion methods. c) Multiple backbone d) Improved accuracy and enhanced speed. d) Multiscale prediction, e) Customizable architecture, f) Adaptive training g) Supervised learning It represents the cutting-edge advancements in the YOLO series, showcasing outstanding detection accuracy and speed. A noteworthy aspect of YOLOv8 is the inclusion of an apparatus for self-attention in the network's head. This feature enables the model to concentrate on various areas of the imagery and change the value of elements according to relevance. Another noteworthy aspect of YOLOv8 is its capacity to recognize objects on many scales, which is accomplished via a characteristic hierarchy network. The model can reliably recognize things of various sizes inside an image because of the network's numerous layers that detect objects at various scales.

i. Working principle and key features of YOLOv8

YOLO architecture operates on the principle of performing object detection in a single forward pass of the network, making them exceptionally fast and suitable for real-time applications.

ii. Key features of using YOLOv8

The multiple backbones in YOLOv8 support various backbones, which is part of the architecture that utilizes EfficientNet, ResNet, and CSPDarknet, giving users the flexibility to choose the best model for their specific use case. YOLOv8 uses adaptive training to optimize the learning rate and balance the loss function during training, leading to better model performance. The customizable architecture in YOLOv8 allows users to easily modify the model's structure and parameters to suit their needs.

The YOLOv8 comes in five different variants depending on the network width and depth, viz. nano(n), small(s), medium(m), large(l), and extra-large(x).

3.5.3.2 Architecture of YOLOv8

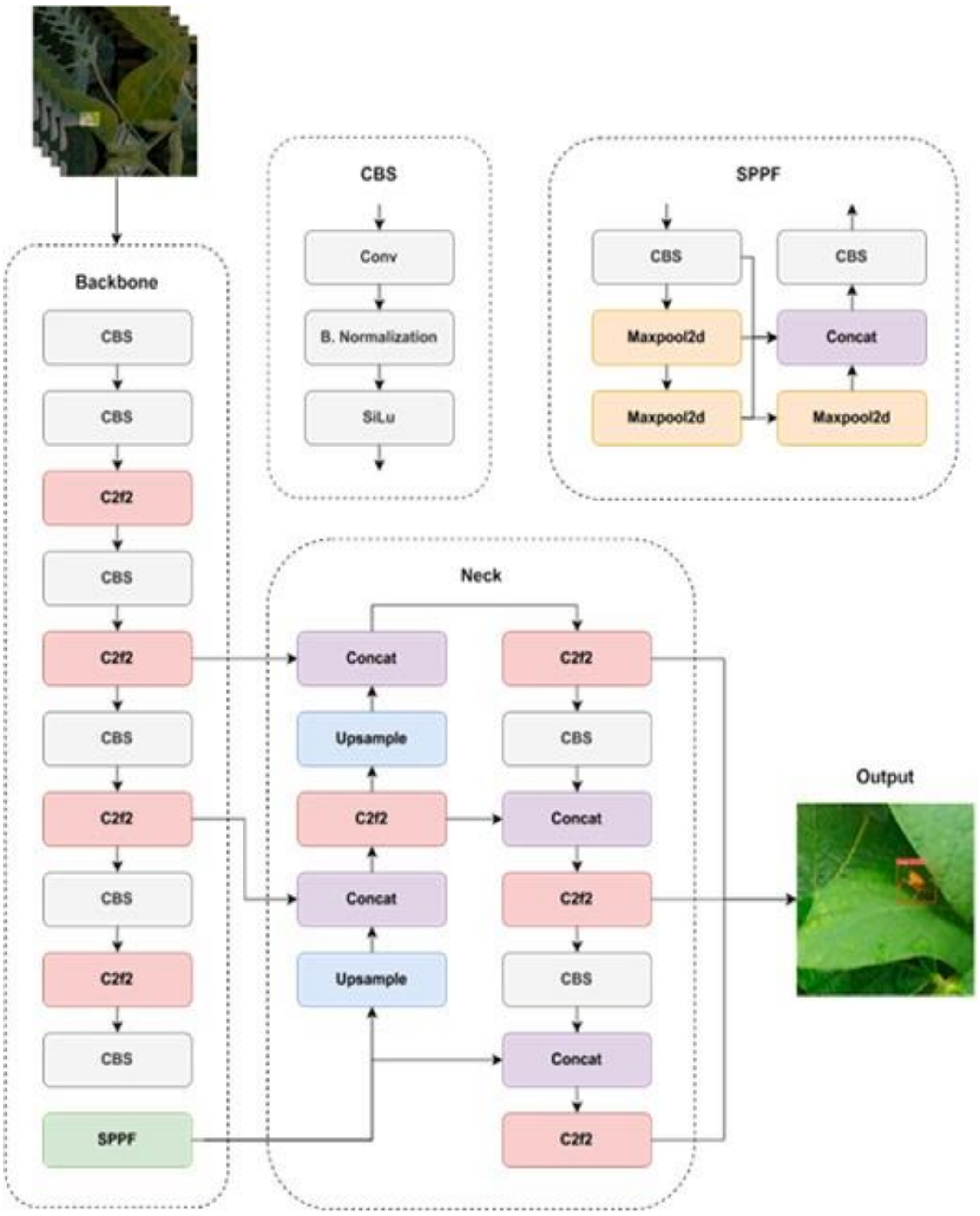


Fig. 3.3 The architecture of YOLO-v8 model

3.5.3.3 YOLOV8 model architecture description

YOLOV8 utilizes a convolutional neural network that can be divided into three main components: the backbone network (feature extraction layer), neck network (fusion layer), and head (prediction layer).

i. Backbone

The backbone, also known as the feature extractor, is responsible for extracting meaningful features from the input. YOLOv8 incorporates an enhanced CSPDarknet53 as its backbone network for efficient feature extraction. The architecture known as Darknet-53, which comprises fifty-three convolutional layers, is partitioned into several smaller convolutional modules based on varying phases of information transmission, directing the model gradient flow during propagation, thus mitigating the vanishing gradient issue.

In YOLOv8 architecture, CBS stands for operations such as Convolutional layer, Batch Normalization, and Swish activation. The convolutional layer performs convolution operation and batch normalization normalizes the inputs of each mini-batch, stabilizing and accelerating the training process. The swish activation option in YOLO is a smooth function to introduce non-linearity. Activation functions like SiLU (Sigmoid Linear Unit) are applied for activation functions for neural networks. C2F2 stands for convolutional layer, followed by two fused residual blocks. The C2F2 module is to enhance feature extraction by leveraging the strengths of convolutional layers and residual connections. SPPF (Spatial Pyramid Pooling – Fast) module enhance the feature extraction process by aggregating features at multiple spatial scales efficiently. SPPF is composed of multiple Maxpool2d and CBS. MaxPool2d operation is used to reduce the spatial dimensions of feature maps while retaining the most significant features. The CBS is replaced by C2F2 to enhance feature fusion across different stages of the network.

iii. Neck

The neck is a bridge between the backbone and the head, performing feature fusion. It is responsible for building feature pyramids. Activities include performing concatenation or fusion of features of different scales. The "neck" in the architecture of YOLOv8 typically refers to the part of the network that comes after the backbone (which extracts features from the input image) and before the head (which makes predictions).

In the architecture of the neck, the "upsample" operation involves increasing the spatial resolution of feature maps. Concat (concatenation) operation is used to involve concatenating feature maps from different scales or levels of abstraction. It helps in integrating multi-scale information and enhancing the model's ability to detect objects of various sizes and complexities. Transitioning from "Concat to C2F2" enhances the model's capability to extract and integrate features across different scales or stages, potentially leading to improved object detection performance. From "C2F2 to upsample" suggests a shift using advanced feature fusion techniques to primarily focus on adjusting the spatial resolution of feature maps through upsampling operations. Transitioning from "Concat to C2F2" enhances the model's capability to extract and integrate features across different scales or stages, potentially leading to improved object detection performance.

iii. Head

The head is the final part of the network and is responsible for generating the network’s outputs, such as bounding boxes and confidence scores for object detection. i.e. The head receives the output from the neck and utilizes it to generate predictions for both classes and bounding boxes. Activities include generating bounding boxes associated with possible objects in the image, assigning confidence scores to each bounding box to indicate how likely an object is present, sorts the objects in the bounding boxes according to their categories. For this, YOLOv8 adopts a detection module. Head generates bounding boxes associated with possible objects in the image and assigns confidence scores to each bounding box to indicate how likely an object is present. It also sorts the objects in the bounding boxes according to their categories. Finally, detection is done in the head part.

3.5.3.4 Different Layers in YOLOv8

YOLOv8l is based on a deep convolutional neural network. The exact architecture and number of layers can vary slightly depending on the specific implementation and any modifications made by individual researchers or developers. But generally, it has the following layers.

1. Input Layer: Accepts the input image

2. Convolutional Layers: These layers consist of convolutional filters that extract features from the input image. They form the backbone of the network.

3. Normalization Layers: Techniques like batch normalization may be employed to improve the training stability and convergence speed of the network.

4. Activation Layers: Activation functions SiLU (Sigmoid Linear Unit) are applied after convolutional and fully connected layers to introduce non-linearity into the network.

5. Pooling Layers: These layers downsample the spatial dimensions of the feature maps, reducing computational complexity and increasing the receptive field.

6. Fully Connected (FC) Layers: In FC layers, each input is connected to all neurons. In some versions of YOLO, fully connected layers are used for tasks such as bounding box regression and class prediction.

7. Detection Layers: These layers are responsible for detecting objects in the input image. They typically consist of convolutional layers and a final detection layer that outputs bounding boxes, object, and class predictions.

8. Output Layer: Produces the final output of the network, typically bounding boxes along with associated class probabilities.

3.5.3.5 Selection of best variant of YOLOv8 for model training

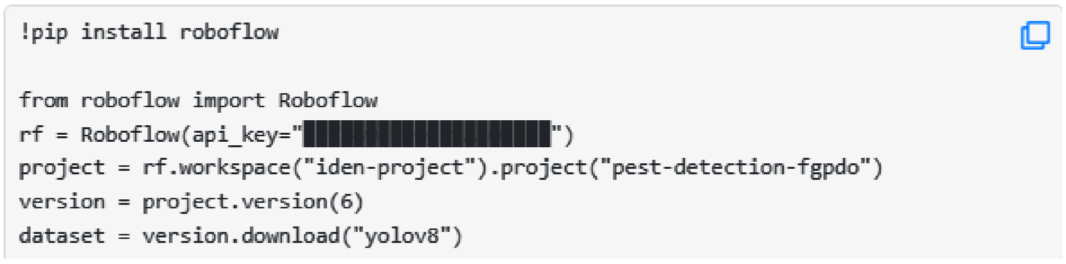
Relevant comparison experiments were performed on different variants of YOLOv8 using the same validation dataset to verify the improved model's effectiveness, and the results were compared. Since the dataset is not very large, a study was conducted to train the model in the different architectures, such as YOLOv8n, YOLOv8s, and YOLOv8l. The performance was compared, and based on the performance, the best variant was selected for this study.

3.5.4 Dataset export

Once the data is annotated and augmented, it can be exported in various formats suitable for training the machine learning models. Roboflow supports exporting datasets in formats compatible with popular deep learning frameworks, including TensorFlow, PyTorch, and Yolo. In this study, YOLO was selected.

3.5.4.1 Source code for customized dataset export

The network architecture of the model was trained using source code on the Google Colab platform, a platform provided by Google LLC in Menlo Park, CA, U.S.A. This platform is useful for accessing GPUs, which are necessary for training deep learning models. The required computing environment was set up by adjusting the runtime type to GPU and acquiring the NVIDIA GeForce RTX 3050 6GB from NVIDIA in Santa Clara, CA, U.S.A. The YOLOv8n, YOLOv8s, and YOLOv8l repository are cloned from the official GitHub repository. The customized dataset was exported using the following source code.



```
!pip install roboflow

from roboflow import Roboflow
rf = Roboflow(api_key="[redacted]")
project = rf.workspace("iden-project").project("pest-detection-fgpdo")
version = project.version(6)
dataset = version.download("yolov8")
```

Fig. 3.4 Source code for customized dataset

3.5.4.2 Raw URL of customised dataset

The customized data set was converted into a URL link. Using the following URL, users can access and download the customized dataset used in this study. https://app.Roboflow.com/ds/7g8BbZsBry?key=tKihxgRuEp. The dataset was exported in the YOLOv8 PyTorch/TXT, formats from the Roboflow project and structured with the necessary directory hierarchy and annotation files compatible with the YOLO architecture.

3.6 EXPERIMENTAL SETUP FOR YOLOV8 MODEL BUILDING

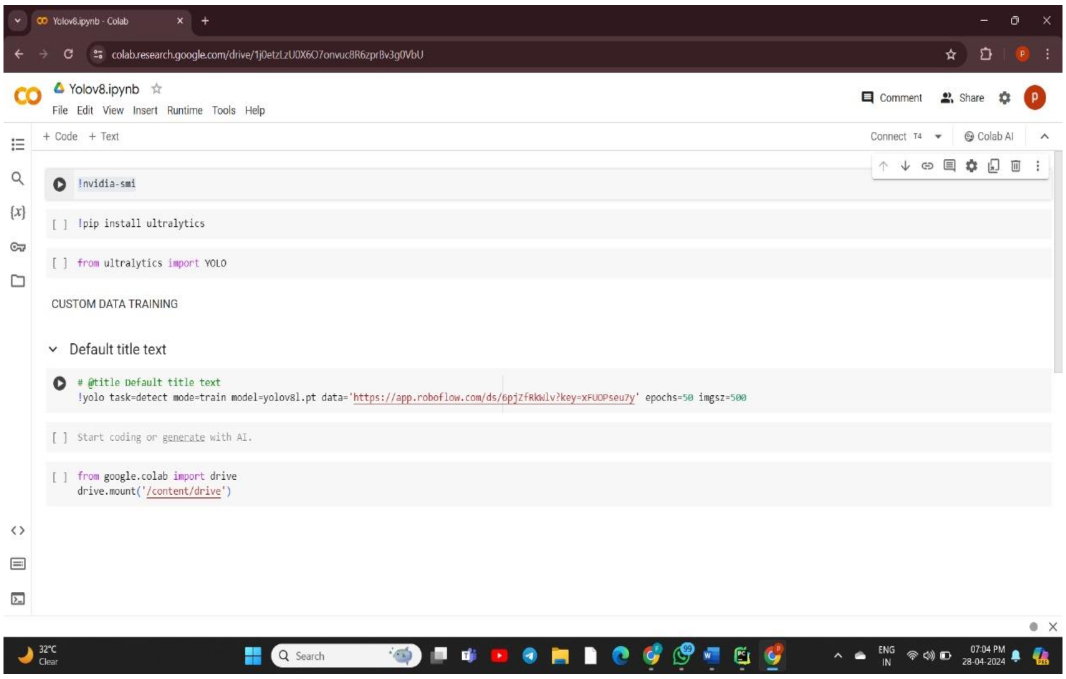
Dataset preparation, network training, model validation & testing, and model performance & accuracy assessment were the fundamental processes adopted to develop the model for the object identification task. Dataset preparation has already been explained. The object detection deep learning model YOLOV8 was deployed for training. The dataset was separated into three parts to conduct the experiment: 87% for training, 8% for validation, and 5% for testing. The model is trained from training images, followed by an evaluation of the validation images, and when the experiment is ready to accomplish the predicted results, the final evaluation is done on the testing set.

3.6.1 YOLOv8 Training, Validation and Testing

i. Model training strategy

The Yolo object detection model was configured and trained on the images of red pumpkin beetles. This study selected 1278 images for training using YOLOv8. Before training begins, the model is configured concerning the dataset and configuration according to GPU, CUDA, and OpenCV. The data set prepared in Roboflow was used for training of YOLOv8 model. The Google Colab platform in Python programming was used to run the custom-trained model. The YOLOv8 model was imported to Google Colab. The training was carried out by installing Ultralytics, which is widely recognized for its speed and accuracy in real-time computer vision tasks. Roboflow with the dataset was also attached to Google Colab. Usually training a model typically involves gathering a labeled dataset, configuring the model architecture, and setting hyperparameters. This study selected the hyperparameters epoch, learning rate, batch size and image size for training the model. The optimizer used was Adam W.

The training process was meticulously executed using a specialized source code specifically designed for the task shown in Fig 3.5. This source code was integrated into the Google Colab environment. During the training phase, the model underwent numerous iterations, commonly referred to as epochs. Epoch indicates the number of times the entire training dataset is passed through the CNN network. Epoch is a hyperparameter that determines the machine learning model's training process. The learning rate, image size, and batch size were also adjusted. Batch size defines the number of samples taken to work through a particular machine-learning model before updating its internal model parameters. Thus, network architecture, hyperparameters, lost function parameters, etc, were adjusted to train the model. The loss function is the difference between the actual value and the predicted value, which varies between 0 and 1. As the value of the loss function decreases better accurate the model for object detection. The model was trained using the training dataset until it achieved satisfactory performance indicators in detecting objects within images. Thus, weights are obtained as a result of training the model, which is represented in the file name ‘last. pt’. As the training progressed, the model iteratively adjusted its internal parameters to optimize its ability to detect and localize objects of interest within the images accurately. This learning process involved minimizing a defined bounding loss function that quantified the disparity between the predicted bounding boxes and the ground truth annotations.


Yolov8.ipynb - Colab
colab.research.google.com/drive/1j0etzLzU0X6O7onvuc8R6zprBv3g0VbU
Yolov8.ipynb
File Edit View Insert Runtime Tools Help
Comment
Share
+ Code + Text
Connect T4
Colab AI

```
!nvidia-smi
```

```
!pip install ultralytics
```

```
from ultralytics import YOLO
```

CUSTOM DATA TRAINING

Default title text

```
# @title Default title text
!yolo task=detect mode=train model=yolov8l.pt data='https://app.roboflow.com/ds/6pjZfRkWlv?key=xFUOPseu7y' epochs=50 imgsz=500
```

```
Start coding or generate with AI.
```

```
from google.colab import drive
drive.mount('/content/drive')
```

Fig. 3.5 The code used for training YOLOv8 in Google Collab

ii. Model validation and testing strategy

Validation is a critical process used to evaluate the performance and accuracy of a model. The detection model was validated and tested on the images of red pumpkin beetles. The validation and testing process was executed using the source code specifically designed for the task. This study selected 122 images for validation and 62 images for testing. Validation and testing of a YOLOv8 model estimate its accuracy and reliability. Validation adjusts hyperparameters to optimize performance and select the best-performing model, while testing involves evaluating the model's performance on a separate, unseen dataset to ensure that it generalizes well to new data and calculates metrics like precision, recall, MAP, false positives, and false negatives on a test dataset. Key metrics include mAP and AP, which evaluate object detection, class-specific performance, and recall. The file ‘last.pt’ is uploaded for validation and testing of the model. Add Dataset for validation and testing. Validation is done with the validation dataset and testing is done with the photos and videos taken from the field. Run the model evaluate the prediction and analyse the performance until satisfactory results are obtained. The final results are then documented.

3.7 PERFORMANCE EVALUATION AND ACCURACY ASSESSMENT

Performance evaluation and accuracy assessment of an object detection model are crucial steps in ensuring the model's reliability and effectiveness in identifying and localizing objects within images. These assessments typically involve a combination of quantitative metrics and qualitative analysis. Key metrics include the Confusion matrix, precision, recall, and the F1-score, which together provide insights into the model's accuracy in detecting true positives while minimizing false positives and negatives. The mean Average Precision (mAP) is often used as a comprehensive measure, evaluating the precision-recall curve across different threshold levels. Continuous evaluation using these methods helps fine-tune the model, enhancing its generalization capability and robustness in real-world applications.

3.7.1 Confusion matrix

Confusion matrix, a popular tool for evaluating the performance of deep learning-based models, is utilized in this study. It serves as a visual representation and quantitative assessment of the Pest detection in the YOLOv8 model's predictive performance, allowing the evaluation, analysis, and improve the model's accuracy and effectiveness. Additionally, the confusion matrix enables the calculation of class-specific performance metrics, providing a more detailed understanding of the model's strengths and weaknesses across different pest detection classes. The confusion matrix is organized as a grid, with the Y-axis representing the true classifications and the X-axis representing the predicted classifications. Table 3.4 represents a basic confusion matrix. There are 3 primary classes in this case study. For three primary classes, the matrix will be 4x4 as shown in table 3.5. The main diagonal of the confusion matrix corresponds to the correctly classified instances, where the predicted class matches the true class. Off-diagonal cells represent the misclassified instances, where the predicted class differs from the true class.

The breakdown of a confusion matrix is as follows:

- True Positive (TP): The cases where the model correctly predicts the positive class.
- True Negative (TN): The cases where the model correctly predicts the negative class.
- False Positive (FP): The cases where the model incorrectly predicts the positive class (Type I error).
- False Negative (FN): The cases where the model incorrectly predicts the negative class (Type II error).

Table 3.4 A basic confusion matrix

	Predicted Positive	Predicted Negative
True Positive	TP	FN
True Negative	FP	TN

By dissecting these elements, various performance metrics can be derived, such as accuracy, precision, recall, and F1 score, each providing unique insights into the model's effectiveness. Additionally, confusion matrices are invaluable for diagnosing specific issues within a model, such as imbalances between classes or misclassification patterns. In essence, confusion matrices serve as a cornerstone for understanding and refining classification models, offering a structured framework for assessing their efficacy and identifying areas for improvement.

Table 3.5 A 4x4 matrix formed for the three primary classes considered in this study.

	Predicted Class "Pest"	Predicted Class "No Pest"	Predicted Class "Affected"
True Class "Pest"	True Positive (Pest)	False Negative (No Pest)	False Negative (Affected)
True Class "No Pest"	False Positive (Pest)	True Positive (No Pest)	False Negative (Affected)
True Class "Affected"	False Positive (Pest)	False Positive (No Pest)	True Positive (Affected)

The confusion matrix offers four different and individual matrics, as already seen. Based on these four metrics, other metrics can be calculated that offer more information about how the model's behavior and performance. They are

1. Accuracy
2. Precision
3. Recall

3.7.1.1 Accuracy

Accuracy is the ratio of correctly predicted observations (both positive and negative) to the total observations. It provides an overall measure of the model's performance

$$\text{Accuracy}=\frac{TP+TN}{TP+TN+FP+FN}$$

3.7.1.2 Precision

Precision is the ratio of correctly predicted positive observations to the total predicted positives. It tells us how many of the predicted positives were actually correct.

$$\text{Precision} = \frac{TP}{TP+FP}$$

3.7.1.3 Recall

Recall is a metric that measures how often a machine learning model correctly identifies positive instances (true positives) from all the actual positive samples in the dataset.

$$\text{Recall} = \frac{TP}{TP + FN}$$

3.7.2 Curves derived from confusion matrix

3.7.2.1 Precision-confidence curve:

The precision-confidence curve is a visual representation of the relationship between precision and confidence threshold. The precision-confidence curve plots the precision value at different confidence thresholds. Starting from a high confidence threshold, the precision will be relatively high because only confident predictions are considered valid.

3.7.2.2 Recall-confidence curve

The recall-confidence curve illustrates how the recall metric changes as the confidence threshold varies. As the confidence threshold decreases, more predictions are considered valid, including both true positives and potential false positives

3.7.2.3 Precision-recall curve

The precision-recall curve is another visual representation that provides insights into the performance and efficiency. The curve showcases the relationship between precision and recall, two important metrics used to evaluate the model's predictive accuracy.

It is desired that the algorithm should have both high precision and high recall. However, most deep learning algorithms often involve a trade-off between the two. A good PR curve has a greater AUC (area under the curve). It is important to note that the classifier has a higher AUC. Here, we can consider an algorithm that classifies whether or not a picture contains a red pumpkin beetle. Assume there are 12 images, with the following ground truth (actual) and classifier output class labels.

By setting different thresholds, we get multiple such precision, recall pairs. By plotting multiple such P-R pairs with either value ranging from 0 to 1, we get a PR curve.

3.7.3 OTHER PERFORMANCE INDICATORS

3.7.3.1 Average precision

To compute AP, precision and recall values are first obtained by varying a threshold that determines whether a predicted instance is correct. Then, precision is calculated at each recall level. These precision-recall pairs are plotted on a graph, and the area under this curve represents AP.

The AP metric is particularly valuable in scenarios where the class distribution is imbalanced or when ranking predictions by their confidence scores. In binary classification tasks, AP quantifies the model's ability to effectively distinguish between positive and negative instances.

$$\text{AP}=\int P(r)dr$$

3.7.3.2 Mean Average Precision(mAP)

The mean Average Precision (mAP) is often used when dealing with multi-class classification or object detection tasks. mAP calculates the AP for each class and then computes their average. This provides a more comprehensive evaluation of the model's performance across all classes.

$$\text{mAP}=\frac{1}{n}\sum_{i}^{n} APi \quad \text{where n is the No.of classes}$$

3.7.3.3 F1 Confidence Curve

The F1-score curve demonstrates how the F1-score changes as the threshold varies. The curve illustrates the relationship between the F1-score and a varying threshold used for predictions. The X-axis of the F1-score curve represents the threshold, which is the minimum confidence score required for a prediction to be considered valid. The Y-axis represents the F1-score, which is a metric that combines both precision and recall into a single value. The F1-score provides a balanced evaluation of the model's performance, considering both the ability to correctly identify positive instances (precision) and capture all actual positive instances (recall). As the threshold decreases, more predictions are considered valid, potentially increasing the recall of the model.

$$\text{F1 Score} = 2 \times \frac{precision \times recall}{precision + recall}$$

3.7.3.4 Loss Function Curves

A loss function curve is a crucial tool for visualizing the training process of a machine learning model. This graph typically plots the loss values on the y-axis against epochs or iterations on the x-axis. The curve usually includes two lines: one for the training loss and another for the validation loss. As the model learns, the training loss should decrease, indicating improved performance on the training data. Concurrently, the validation loss should decrease if the model generalizes well to unseen data.

i. *Underfitting of the model*

   If both training loss and validation loss are high and similar in magnitude, the model is not trained for enough time. This represents the Underfitting of the model.

ii. *Overfitting of the model*

   If validation loss increases and training loss decreases, the model becomes too tailored to training data and loses its ability to generalize. This represents the Overfitting of the model.

iii. *Goodfit model*

   If both training loss and validation loss are decreasing model is Goodfit. It indicates that the model learns effectively from the training data and would perform well on new, unseen data.

Analyzing the loss function curve helps diagnose issues like overfitting and underfitting and guides decisions on adjustments to hyperparameters or training strategies.

Types of Losses

i. Box Loss.

Box loss is a critical component in training object detection models, which need to identify and localize objects within an image accurately. It measures the discrepancy between the predicted bounding boxes and the ground truth boxes. By minimizing box loss during training, the model becomes more adept at detecting and localizing objects with high precision, which is essential for tasks such as autonomous driving, image recognition, and various computer vision applications. Analysing and optimizing box loss helps improve object detection systems' overall performance and reliability.

ii. Class Loss

Class loss is a fundamental component in the training of machine learning models for classification tasks, particularly in object detection. It quantifies the error between the predicted class probabilities and the true class labels of the objects being detected. Class loss helps the model to learn and distinguish between different categories or classes accurately. During training, the model assigns a probability to each possible class for each detected object, and the class loss measures how well these predictions match the actual labels. Minimizing class loss is crucial for improving the model's ability to correctly classify objects, ensuring high accuracy in identifying various classes. This process involves iterative adjustments to the model's parameters, guided by the loss function, to enhance its predictive performance. Effective management of class loss, in conjunction with other losses like box loss, is essential for developing robust and reliable object detection systems that perform well across diverse datasets and real-world scenarios.

iii. DFL (Distribution Focal loss)

Distribution Focal Loss (DFL): Distribution Focal Loss extends the idea of focal loss by incorporating the distribution of the dataset into the loss function. The goal is to ensure that the loss function considers the distribution of classes, giving more importance to classes that are under-represented in the training dataset. This can help the model to learn better representations for minority classes, improving the overall detection performance, especially for rare pests. Distribution Focal Loss in the context of deep learning for pest detection is a loss function that helps to mitigate the impact of class imbalance by dynamically adjusting the focus on different classes based on their distribution in the dataset. This leads to improved detection of rare pest species, which are often the most challenging to identify accurately.

3.7.3.5 Frames Per Second (FPS)

Frames Per Second (FPS) measures how many video frames the model can analyze in one second. A higher FPS indicates that the model could process more frames per second, making it faster and more efficient in real-time scenarios. FPS detection speed in the performance evaluation matrix for YOLOv8 refers to the number of frames per second that the model can process and make predictions. This metric is crucial for evaluating the real-time performance of the model. This speed is influenced by factors such as:

1. Model Complexity: More complex models with higher accuracy often require more computational power and time to process each frame, resulting in lower FPS.

2. Hardware: The type and capability of the hardware (CPU, GPU, memory) significantly affect the FPS. More powerful hardware can process frames faster.

3. Input Resolution: Higher resolution inputs require more computation, which can reduce FPS.

4. Batch Size: Batch size is the number of training examples utilized in one iteration of model training. The number of frames processed simultaneously can impact FPS.

Larger batch sizes can utilize hardware more efficiently but might introduce latency.

3.7.3.6 Inference Time

Inference time in YOLOv8 refers to the amount of time it takes for the model to process a single image or frame and produce output, such as bounding boxes and class probabilities. This metric is crucial for real-time object detection applications. The inference time depends on various factors, including the hardware used (such as GPU, CPU, or TPU), the model architecture and size, input image resolution, batch size, and the framework and implementation used (like PyTorch or TensorFlow).

3.7.3.7 Detection speed

Detection speed in YOLOv8 refers to the rate at which the model can detect objects in images or videos, measured in frames per second (FPS). It is a critical performance metric for real-time object detection applications, where the goal is to quickly and accurately identify objects in a stream of images or video frames. The YOLOv8 detection speed is influenced by factors such as the hardware used, model architecture, input image resolution, and optimization techniques employed. A higher detection speed indicates better performance, with YOLOv8 aiming to achieve detection speeds of 30 FPS or higher, depending on the specific use case and hardware configuration.

3.8 WEB APPLICATION DEVELOPMENT FOR REAL-TIME MODEL TESTING

Following training the YOLOv8 model for pest detection, the model was converted into a format that could be readily integrated into a web application. This was accomplished by the following steps

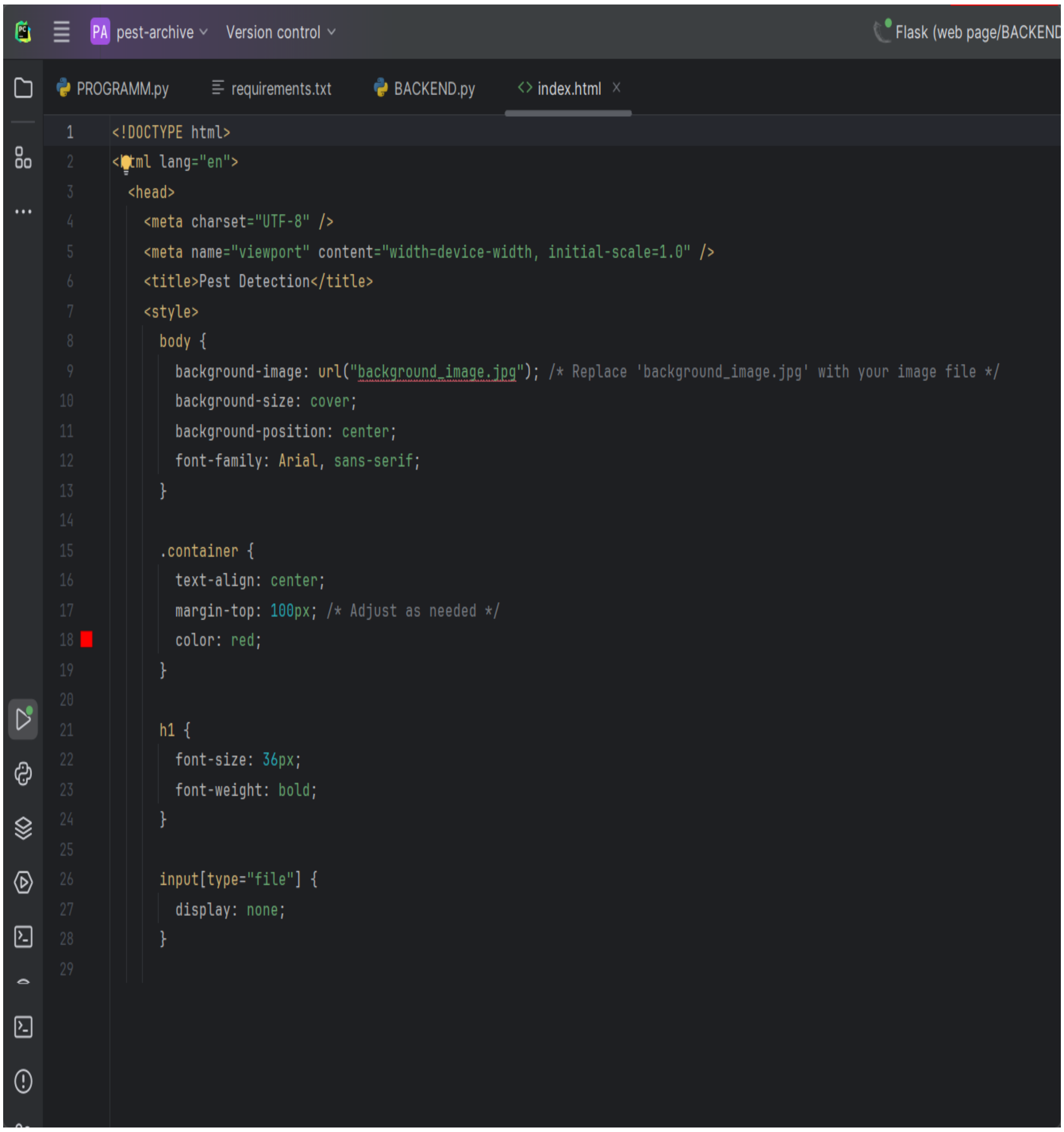
3.8.1 WEB PAGE CREATION

In this study, a web application is developed to detect red pumpkin bettles in real-time field conditions. For this, 'Notepad' in Windows is used, which is a basic text editor that can be used to create web pages. Web pages are created with 1. ‘HyperText Markup Language ’ (HTML) is used to create structure content, define layout, add multimedia, tables, etc, 2. ‘Cascading Style Sheets (CSS)’ used for styling and layout purposes and 3. ‘JavaScript’ helps to create buttons, animations, special effects, menus, etc. In assessing real-world detection of pests on the web pages, three procedures were adopted: 1. By uploading real field photos, 2. by uploading videos taken from the field, and 3. by uploading real-time field data. Using various media sources, the detection system's effectiveness could be enriched in the evaluation process.

Since buying a field web camera was beyond the scope of this study, the webcam on a laptop was used for real-time pest detection. The pumpkin leaf and dead pests taken from the field were framed on cardboard to access it through the laptop's webcam. Thus, real-time pest detection was achieved via the web application.

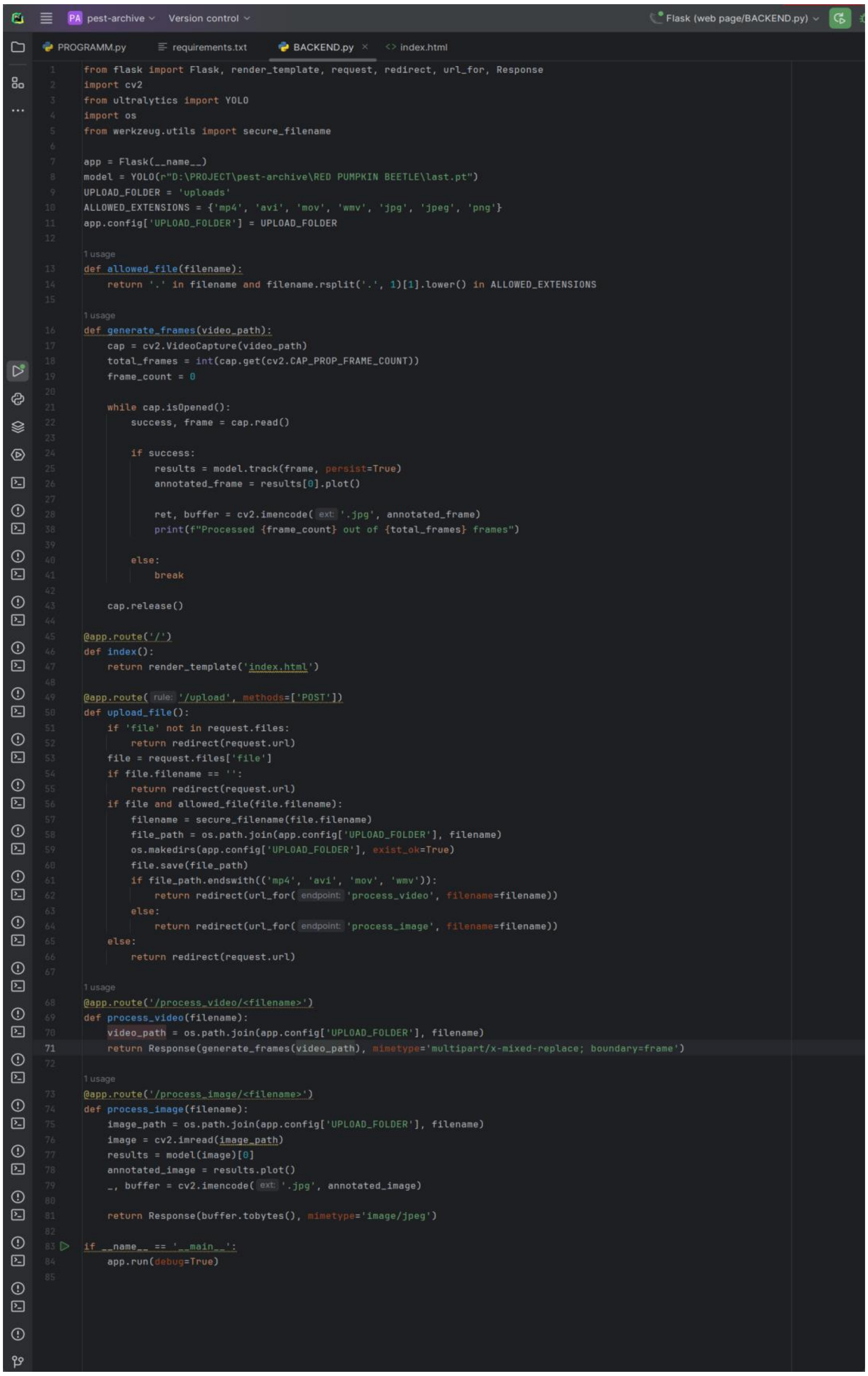
3.8.1.1 Web page creation strategy

Webpage is created by using ‘Notepad’. The HTML document is structured to include a 'head' section that contains metadata, styling, and CSS used for a clean layout and design. The 'body' section defines the content, featuring a page title, buttons for uploading photos and videos, a video element for the webcam feed, and a button to start and stop the webcam. Additionally, custom-styled buttons are created to trigger file input elements for uploading photos and videos. Overall, the HTML document effectively incorporates functionality to stream video from the webcam and provides user-friendly options for uploading photos and videos, making it a comprehensive and interactive web page. JavaScript functions are incorporated with Python language to create the web page's backend. The backend (server side) is the portion of the website responsible for storing and organizing data and ensuring everything. The backend communicates with the frontend, sending and receiving information to be displayed as a web page. It contains the algorithms developed during the model's Training, Validation and Testing. Finally, the detection takes place.



```html
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Pest Detection</title>
    <style>
      body {
        background-image: url("background_image.jpg"); /* Replace 'background_image.jpg' with your image file */
        background-size: cover;
        background-position: center;
        font-family: Arial, sans-serif;
      }

      .container {
        text-align: center;
        margin-top: 100px; /* Adjust as needed */
        color: red;
      }

      h1 {
        font-size: 36px;
        font-weight: bold;
      }

      input[type="file"] {
        display: none;
      }

```

Fig. 3.6 The code developed for the front interference of the webpage



```python
from flask import Flask, render_template, request, redirect, url_for, Response
import cv2
from ultralytics import YOLO
import os
from werkzeug.utils import secure_filename

app = Flask(__name__)
model = YOLO(r"D:\PROJECT\pest-archive\RED PUMPKIN BEETLE\last.pt")
UPLOAD_FOLDER = 'uploads'
ALLOWED_EXTENSIONS = {'mp4', 'avi', 'mov', 'wmv', 'jpg', 'jpeg', 'png'}
app.config['UPLOAD_FOLDER'] = UPLOAD_FOLDER

1 usage
def allowed_file(filename):
    return '.' in filename and filename.rsplit('.', 1)[1].lower() in ALLOWED_EXTENSIONS

1 usage
def generate_frames(video_path):
    cap = cv2.VideoCapture(video_path)
    total_frames = int(cap.get(cv2.CAP_PROP_FRAME_COUNT))
    frame_count = 0

    while cap.isOpened():
        success, frame = cap.read()

        if success:
            results = model.track(frame, persist=True)
            annotated_frame = results[0].plot()

            ret, buffer = cv2.imencode( ext: '.jpg', annotated_frame)
            print(f"Processed {frame_count} out of {total_frames} frames")

        else:
            break

    cap.release()

@app.route('/')
def index():
    return render_template('index.html')

@app.route( rule: '/upload', methods=['POST'])
def upload_file():
    if 'file' not in request.files:
        return redirect(request.url)
    file = request.files['file']
    if file.filename == '':
        return redirect(request.url)
    if file and allowed_file(file.filename):
        filename = secure_filename(file.filename)
        file_path = os.path.join(app.config['UPLOAD_FOLDER'], filename)
        os.makedirs(app.config['UPLOAD_FOLDER'], exist_ok=True)
        file.save(file_path)
        if file_path.endswith(('mp4', 'avi', 'mov', 'wmv')):
            return redirect(url_for( endpoint: 'process_video', filename=filename))
        else:
            return redirect(url_for( endpoint: 'process_image', filename=filename))
    else:
        return redirect(request.url)

1 usage
@app.route('/process_video/<filename>')
def process_video(filename):
    video_path = os.path.join(app.config['UPLOAD_FOLDER'], filename)
    return Response(generate_frames(video_path), mimetype='multipart/x-mixed-replace; boundary=frame')

1 usage
@app.route('/process_image/<filename>')
def process_image(filename):
    image_path = os.path.join(app.config['UPLOAD_FOLDER'], filename)
    image = cv2.imread(image_path)
    results = model(image)[0]
    annotated_image = results.plot()
    _, buffer = cv2.imencode( ext: '.jpg', annotated_image)

    return Response(buffer.tobytes(), mimetype='image/jpeg')

if __name__ == '__main__':
    app.run(debug=True)
```

Fig. 3.7 Code developed for the back end of the webpage for pest detection

Results And Discussion

CHAPTER – IV
RESULTS AND DISCUSSION

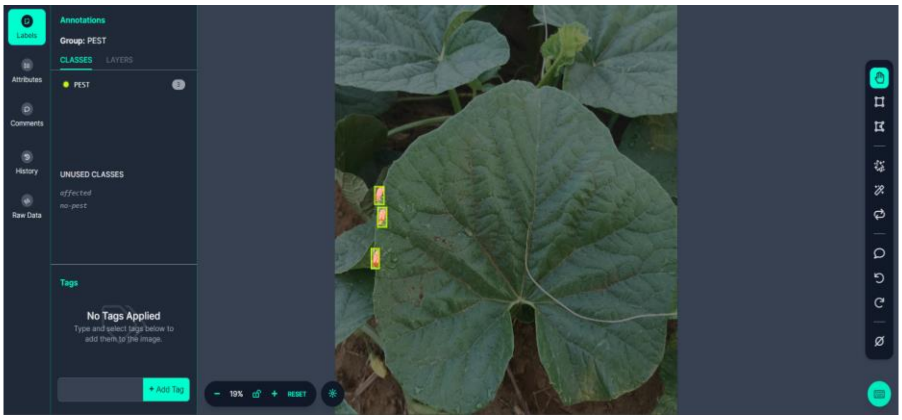
A smart pest detection model based on deep learning architecture was developed for the detection of the pest, red pumpkin beetles in the field. Roboflow was used as a conversion tool for data annotation, preprocessing, and augmentation. YOLOv8l was the object detection model used to develop the model. It is a CNN-based architecture. The results pertaining to data preparation, model development, performance assessment, and web application development for real-time pest detection were explained under the following subtitles.

4.1 PREPARATION OF CUSTOMISED DATASET

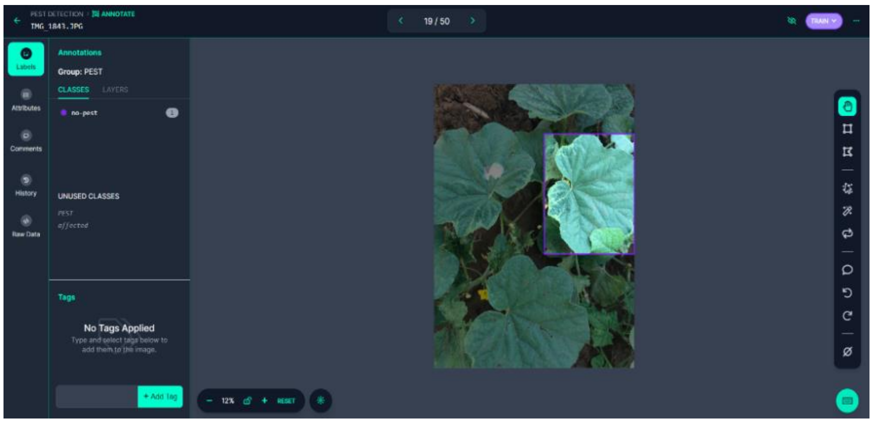
Roboflow was used as the conversion tool to prepare a customized dataset from the collected raw data. The images were annotated using Roboflow to provide labels or bounding boxes for objects of interest within the images or frames. The data was annotated manually and classified into 3 classes viz. pest (Yellow box), no pest (purple box), and affected (red box). The dataset was pre-processed by two preprocessing methods, viz. 1. resizing images and 2. converting to greyscale. The data was augmented to increase the number of images for training. The total images were split randomly into 3 sets by Roboflow: the training set, the validation set and the test set. Finally, the annotated and augmented data sets were converted into various formats in order to train the YOLOv8l model.

4.1.1 Data annotation using Roboflow

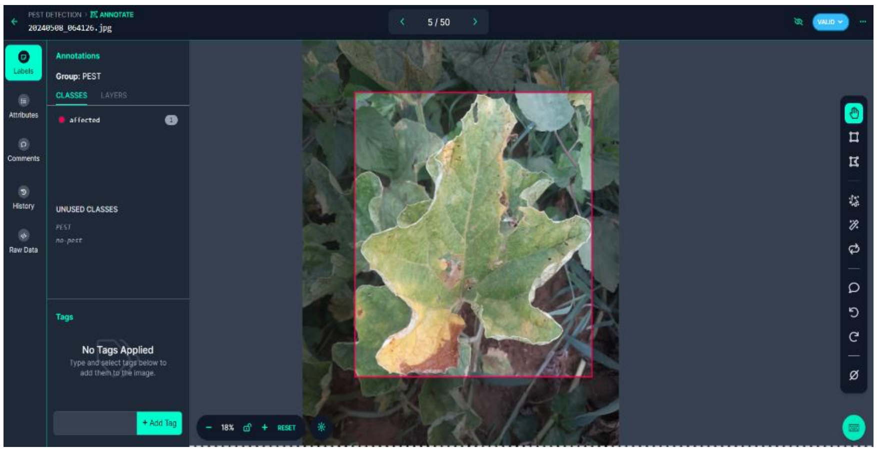
A total of 610 images collected from the field and the website Shutterstock were annotated and categorized into three classes. Fig 4.1 shows the images after annotation a. with pest infestation on a leaf (class named as ‘pest’) represented by yellow box b. healthy leaves without pest (class named as ‘no pest’) represented by purple box and c. leaves affected by pest (class named as ‘affected’) represented by red box



(a)



(b)



(c)

Fig. 4.1 Annotated images (a) with pest) (b) without pest (c) affected

4.1.2 Dataset pre-processing using Roboflow

By Pre-processing the dataset, the data was cleaned and standardized to make them ready for the model. The results of various pre-processing steps are explained as follows

4.1.2.1 Converting to Grayscale

Converting images to grayscale is a common preprocessing step in deep learning tasks, especially when dealing with tasks where colour information might not be relevant or could potentially introduce noise to the model. simplifies the data, reduces computational load, and emphasizes important structural features. This can lead to more efficient and effective training of machine learning models The image after converting to grayscale is shown in Fig. 4.2.

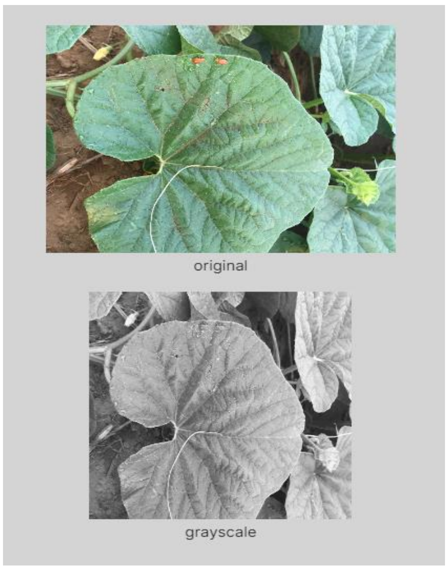


Fig. 4.2 Image converted to Gray scale

4.1.2.2 Resizing

Resizing images is an essential step in the data processing pipeline. Because YOLOv8 models require fixed-size input images, resizing ensures that all input images are of the same size before being fed into the model, which simplifies the training process. This will allow efficient processing by the model, reducing computational load and improving training speed. Hence the image was resized from 4032x3024 px to 800x800 px. The image after resizing is shown in Fig. 4.3.

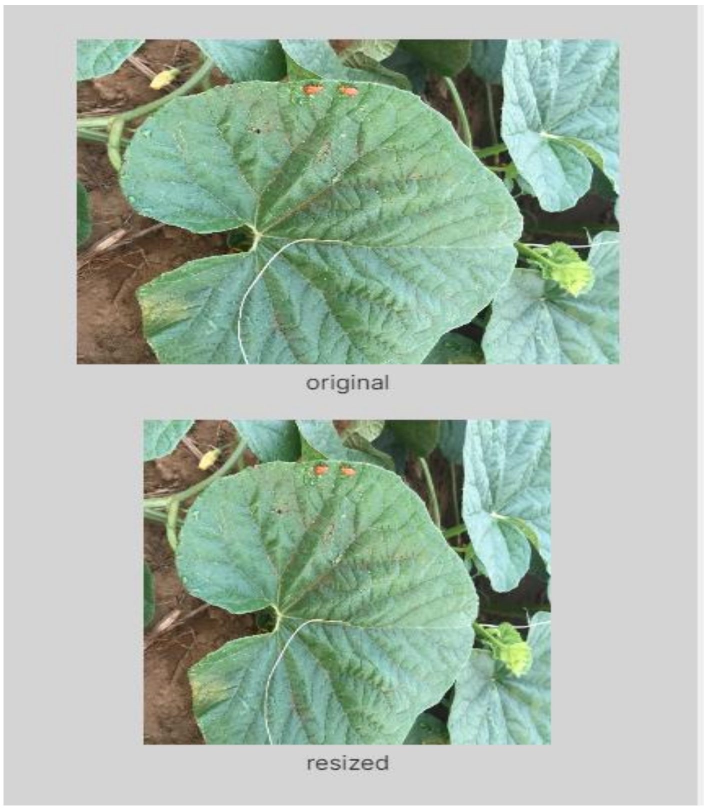


Fig. 4.3 Image Resized

4.1.3 Data augmentation using Roboflow

The various data augmentation techniques commonly used in deep learning, such as Rotation, flipping, Noise addition, colour adjustment, etc, were done.

4.1.3.1 Rotation

Training the model on rotated images makes it more robust and less sensitive to orientation changes, improving its ability to detect objects regardless of their rotation. This helps in detecting pests in diverse and varied real-world conditions. The image after rotation is shown in Fig. 4.4

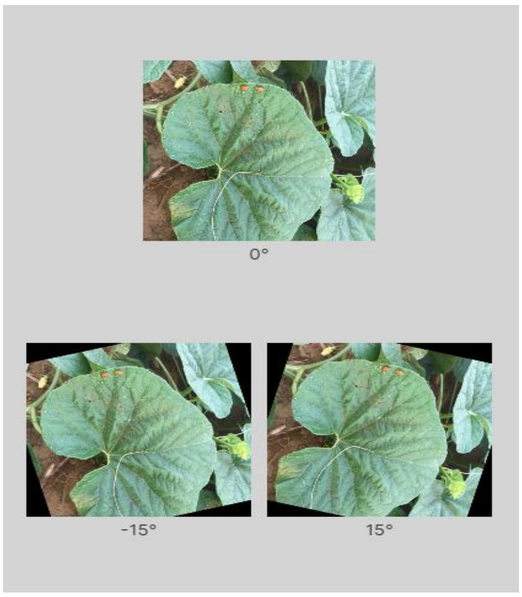


Fig. 4.4 Image Rotated 15°

4.1.3.2 Flipping

Flipping an image horizontally (left-to-right) or vertically (top-to-bottom) helps the model generalize better by providing additional training data with minimal additional labelling effort. Flipping images adds more variety to the training dataset, helping to reduce overfitting and improve the model's ability to generalize. The image after flipping is shown in Fig. 4.5.

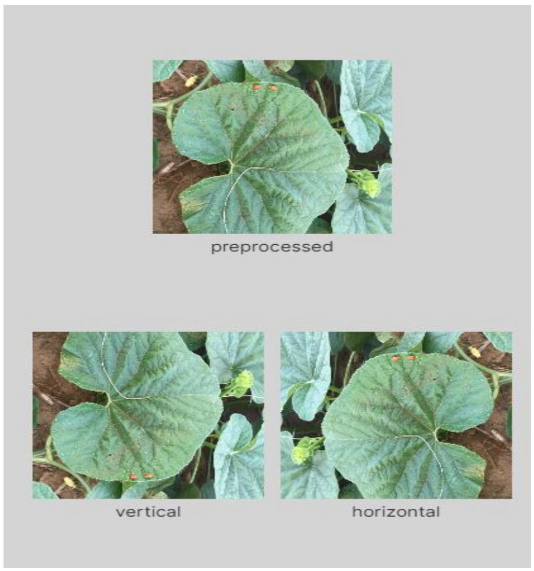


Fig. 4.5 Image Flipped vertically and horizontally

4.1.3.3 Noise Addition

The purpose of adding noise is to introduce variability into the training data, which can help the model to detect and identify pests even when the images are not perfect. Noise addition simulates real-world scenarios where images may be corrupted by noise. The image after noise addition is shown in Fig. 4.6.

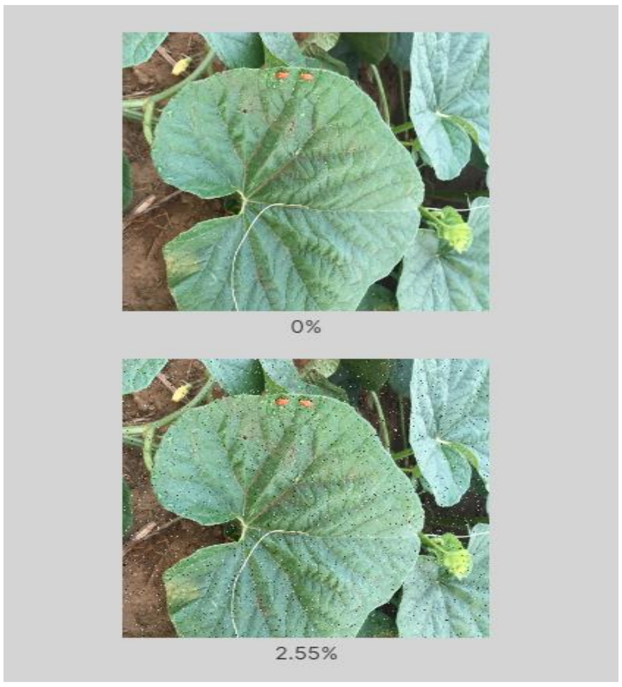


Fig. 4.6 Image after Noise addition

4.1.3.4 Colour Adjustment

By adjusting the colours of images, it is possible to diversify the training data, making the model more robust to variations in lighting conditions, camera settings, and other factors. Colour adjustment simulates real-world scenarios where objects may appear in different lighting conditions or colours, making the model more robust. The image after colour adjustment is shown in Fig. 4.7.

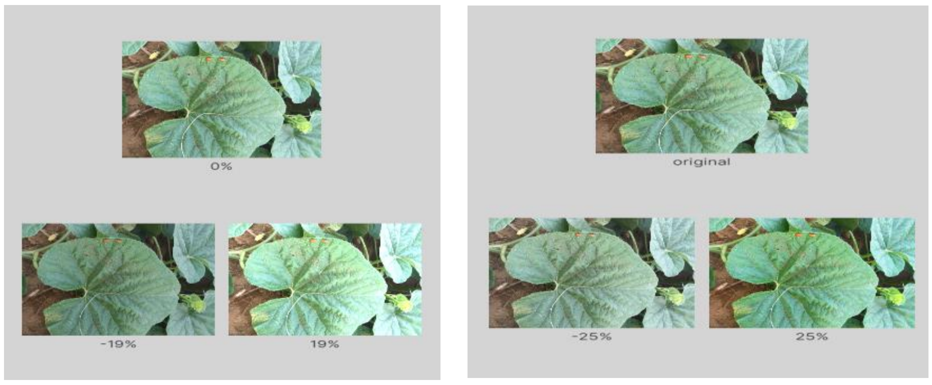


Fig. 4.7 Image after Colour adjustment

4.1.3.5 **Shearing**

Shearing helps the model to detect pests in real field even when the leaves are viewed from oblique angles or when the camera is not perfectly aligned. The image after shearing is shown in Fig. 4.8.

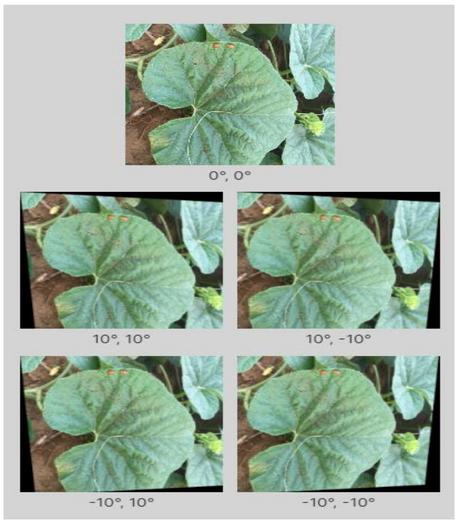


Fig. 4.8 Image after Shear augmentation

4.1.3.6 **Cropping**

Cropping involves removing portions of an image, which can help the model learn to focus on relevant parts of the image and improve its generalization capabilities. Cropping simulates how objects may appear in different environments, such as cropped views or partial occlusions. It helps the model to detect pests even if some part of the pest is visible. The image after cropping is shown in Fig. 4.9.

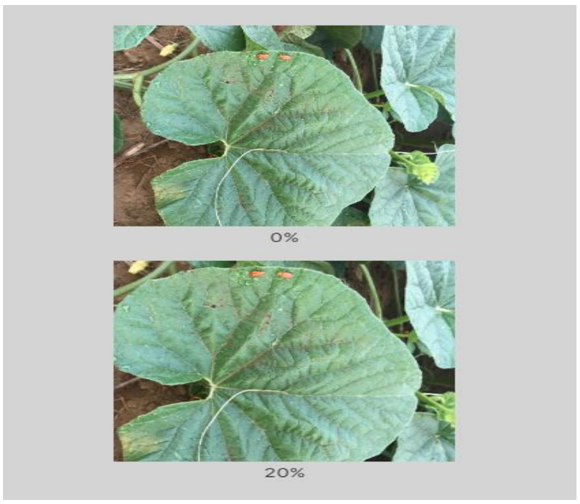


Fig. 4.9 Image after Cropping

4.1.3.7 Hue

Hue is a form of colour augmentation that can be beneficial in data processing for YOLOv8. Real-time images captured in real-time in different lighting conditions (e.g., sunlight, shade, artificial light) can cause colour shifts in images. Hue augmentation helps the model focus on object features rather than their colour, leading to improved object detection. The image after Hue augmentation is shown in Fig. 4.10.

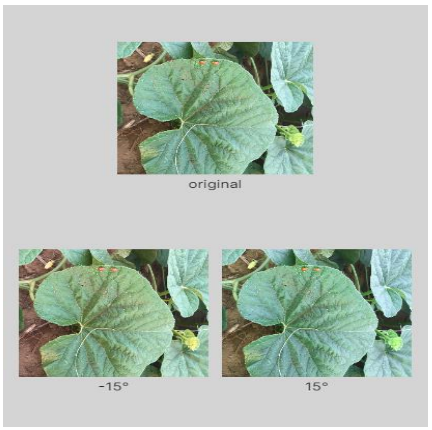


Fig. 4.10 Image after Hue Augmentation

4.1.3.8 Blur

Blur augmentation helps the model learn to detect objects even in low-quality or blurry images, making it more robust to image degradation. It can detect pests in real-time even when they become blurred due to camera movement or falling raindrops. The image after Blur augmentation is shown in Fig. 4.11

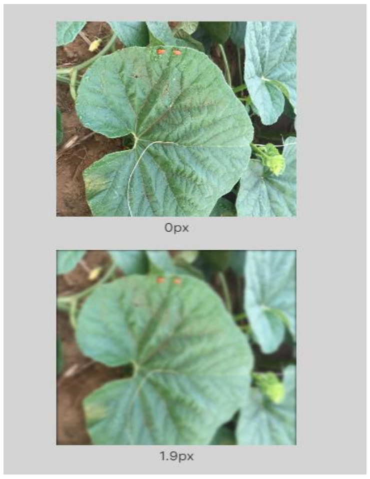


Fig. 4.11 Image Blurred

4.1.3.9 Exposure

Adjusting exposure is another form of data augmentation that can enhance the robustness and generalization of object detection models like YOLOv8. Exposure augmentation helps the model generalize better to new, unseen images by exposing them to various lighting conditions. Exposure adjustment involves modifying an image's overall brightness or darkness, which can simulate variations in lighting conditions and improve the model's ability to handle

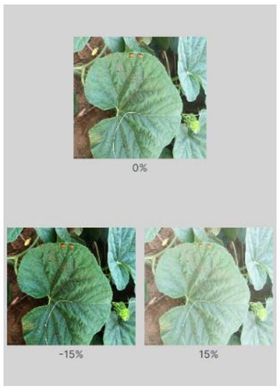


Fig. 4.12 Image after Exposure

different environments. The image after Exposure augmentation is shown in Fig. 4.12

4.1.3.10 Cutout

Cutout augmentation helps the model learn to detect objects even when they are partially occluded or cut off, making it more robust to real-world scenarios. Cutout augmentation involves randomly removing patches of pixels from an image during training. This technique helps to improve the model's robustness by encouraging it to focus on different parts of the image, preventing overfitting and enhancing generalization. The image after Cutout augmentation is shown in Fig. 4.13

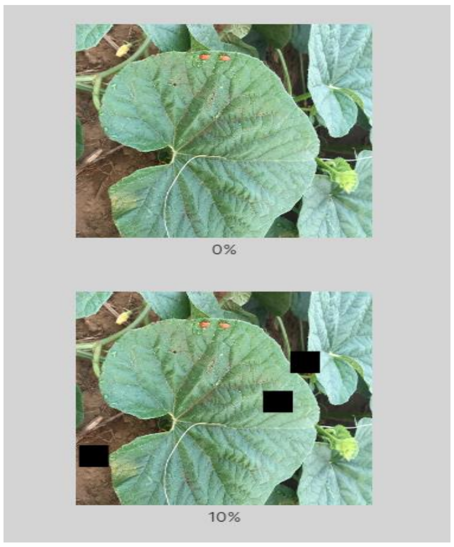


Fig. 4.13 Image after Cutout augmentation

Thus, after data augmentation, a total of 1462 images (i.e., 610 raw images + 852 augmented images) were created to train the model. The result of augmentation is a richer and more varied training dataset, which leads to a model that is more accurate, reliable, and capable of performing well on new, unseen data.

4.1.2 Data health checkup

After the data preparation, a data health checkup was done by two methods 1. Annotation heatmap and 2. Histogram equalization.

4.1.2.1 Annotation heatmap

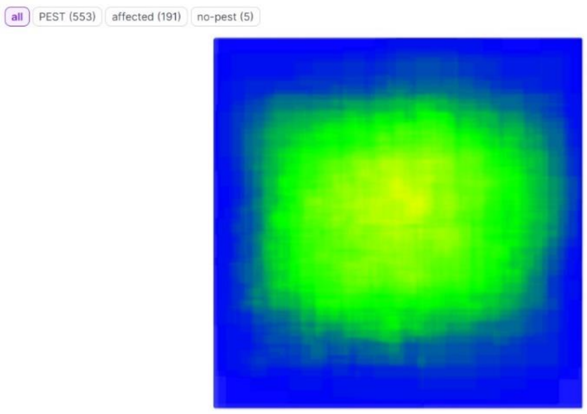


Fig.4.14 Annotation Heatmap of the model

It’s a map where different colours show how much pest is present in different areas. Blue colour indicates there's low pest density, green colour indicates there's medium pest density, and yellow colour indicates there is high pest density. This heatmap shown in Fig. 4.14 revealed that the pest has the most amount present right in the middle of the image and less as it moves away from the center of the image.

4.1.2.2 Histogram equalization

This graph displays the number of objects present in various images. The majority of photos have exactly 1 PEST object (341 photos). Some photos have 2 PEST objects (130 photos), and very few numbers of photos have 3 PEST objects as shown in the histogram (Fig.4.15).

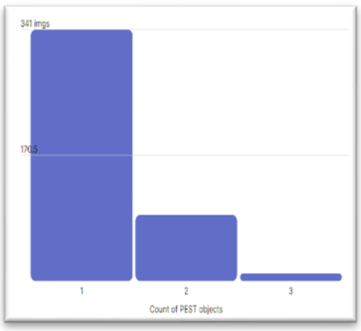


Fig.4.15 Histogram

After a thorough health check of the dataset, it was confirmed that the annotated, pre-processed, and augmented data are suitable for modeling.

4.2 COMPARISON OF DIFFERENT VARIANTS OF YOLOV8 AND SELECTION OF BEST VARIANT FOR MODEL TRAINING

Compared the different variants of the YOLO network to select the best one for the current data. The comparison study is shown in Table 4.1. YOLOv8n had very low precision (0.19), meaning it incorrectly identifies many objects, but it has decent recall (0.80), meaning it finds most of the objects. The overall scores (PR curve and F1 curve) are somewhat moderate. YOLOv8s had much higher precision (0.84) and recall (0.83) than YOLOv8n, leading to better overall scores. The YOLOv8l version performed the best with the highest precision (0.92), recall (0.88), and overall scores (PR curve 0.94, F1 Score 0.86). Hence YOLOv8l was found best at correctly detecting objects and finding all the objects in an image, making it the most accurate and reliable version among the three for the current data. Therefore, it proved that the training speed of the network had indeed increased in YOLOv8l during the early training state itself. Hence YOLOv8l was selected for model training in this study.

Table 4.1 Comparison experiment of YOLOv8 versions of different complexity for the current data

	P curve	R curve	PR curve	F1 curve
YOLOv8n	0.19	0.80	0.74	0.92
YOLOv8s	0.84	0.83	0.88	0.84
YOLOv8l	0.92	0.88	0.94	0.86

YOLOv8l (Larger) architectures can detect more complex patterns and extract better features from images. Pest detection often demands real-time results for swift decision-making and timely implementation of corresponding measures. YOLOv8l has better precision and recall, improved detection of small and difficult-to-distinguish objects, has a deeper network with more convolutional layers,

processes high-resolution images more effectively, and handles a larger number of object classes efficiently. Hence, YOLOv8large was selected in this study.

4.3 MODEL DEVELOPMENT

When training the network model for red pumpkin beetle object detection, the dimensions of the input image are uniformly modified to 800X800px. The ‘Adam W optimizer’ in YOLOv8 is utilized with a total of 50 epochs (the number of epochs refers to the number of times the entire training dataset is passed forward and backward through the neural network during the training process) with a learning rate of 0.002 and batch size of 16. The loss values gradually decrease over epochs, indicating that the model is run effectively. Additionally, the precision (P) and recall (R) values are improving, suggesting better detection performance. The mean Average Precision (mAP) is also increasing, which is a positive sign of the model's overall effectiveness. The size of the custom weight file generated was 65 MB. The number of convolutional layers in the trained model was 56.

4.4 PERFORMANCE EVALUATION AND ACCURACY ASSESSMENT

4.4.1 Confusion Metrix:

A confusion matrix is a powerful tool for evaluating the performance of a detection model. It provides detailed insight into how well the model is performing in terms of correctly and incorrectly classified instances for each class. In the context of a confusion matrix, especially for a classification problem involving multiple classes, "background" typically refers to a class that represents all categories that are not of primary interest where the model must distinguish between foreground objects (the primary classes of interest) and the background (everything else).

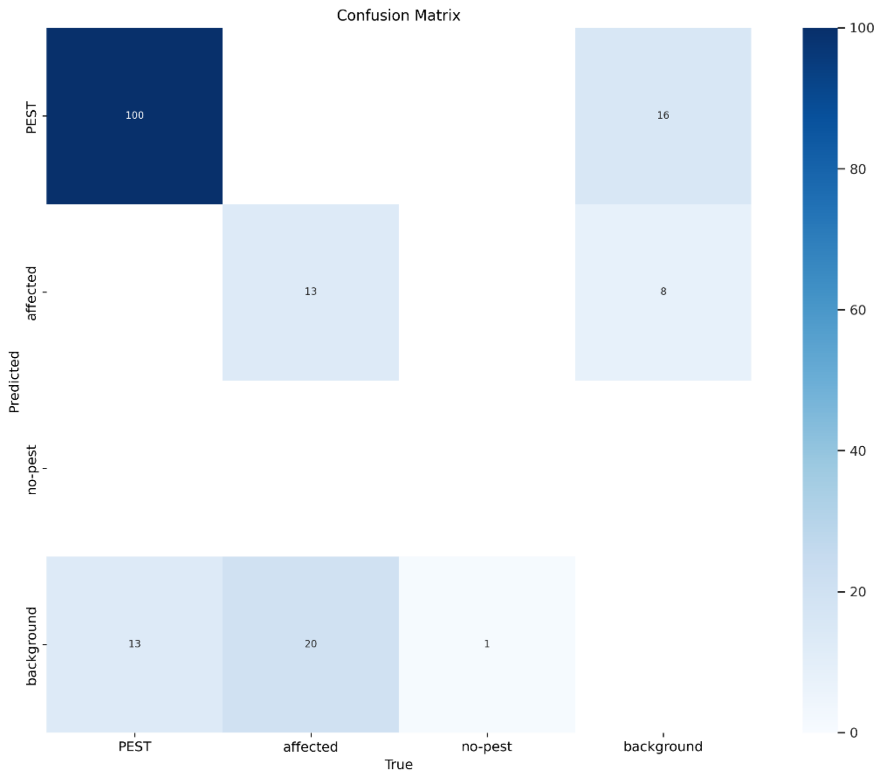


Fig. 4.16 Confusion matrix of validation of YOLOv8l model

The confusion matrix has been reformatted into a 2x2 matrix, with the class "pest" considered as the positive class and the other three classes ("no pest," " affected," and "background") grouped together as the negative class. This simplification facilitates calculating accuracy, precision, recall, and other performance metrics specifically for the "pest" class. Below (Table 4.2) is the resulting 2x2 matrix representing the model's validation performance.

Table 4.2 Confusion matrix of validation of YOLOv8l model

	Predicted Positive	Predicted Negative
Actual Positive	100	16
Actual Negative	13	42

True Negatives (TN), False Positives (FP), and False Negatives (FN) in the context of classification were as follows:

- True Positives (TP): Images that were accurately detected and categorized as "pest" in this study TP=100
- True Negatives (TN): True Negatives are those cases in which the document was appropriately labelled as "no pest" even though it had nothing to do with pest. In this study TN=42
- False Positives (FP): Images that were mistakenly categorised as "pest present" even though they had nothing to do with pest. In this study FP=16
- False Negatives (FN): Examples of documents that were mistakenly labelled as "No pest," but in reality, they were pest present. In this study FN=13
- Thus, obtained counts are : TP=100, TN=42, FP=16, FN=13
- Finally, accuracy, precision and recall were calculated from this as follows

4.4.1.1 Accuracy

The model's accuracy represents the proportion of correct predictions (both pests detected when a pest is present and no pest detected when no pest is present) among the total number of cases. The model has an accuracy of 83%, meaning it correctly identifies the presence or absence of pests in 83% of the cases. 83% accuracy of model is generally considered to be a fairly good model.

I.e., Accuracy=$\frac{TP+TN}{TP+TN+FP+FN}$ =$\frac{100+42}{100+42+16+13}$ =0.830=83%

4.4.1.2 Precision

The precision of YOLOv8, based on the assumed detection results, is 0.86 (or 86%). This means that 86% of the objects detected by YOLOv8 are true positives, while the remaining 14% are false positives. An 86% accuracy is generally considered a solid performance of model.

I.e., $Precision$=$\frac{TP}{TP+FP}$ =$\frac{100}{100+16}$ = 0.86=86%

4.4.1.3 Recall

The recall of YOLOv8, based on the assumed detection results, is approximately 0.88 (or 88%). This means that YOLOv8 correctly identifies 88% of all actual objects, with the remaining 12% being missed (false negatives). These metrics indicated that YOLOv8 has a high recall, meaning it is effective at detecting most of the objects present in the dataset, with a good, but slightly lower precision.

$$\text{I.e., } Recall=\frac{TP}{TP+FN}=\frac{100}{100+13}=0.88=88\%$$

Hence it can be concluded that the model is fit for prediction.

4.4.1.4 FI score

The F1 score is a single metric that combines both precision and recall, giving a balanced measure of the system’s accuracy.

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

$$=2\times\frac{86\times 88}{86+88}$$

$$=.869=86.9\%$$

F1 score of 86.9% indicated that the model has a good balance between precision and recall.

4.4.2 Precision Confidence curve

The graph shown in Fig. 4.16 is a precision-confidence curve, which shows how precision changes with varying confidence thresholds for different classes.

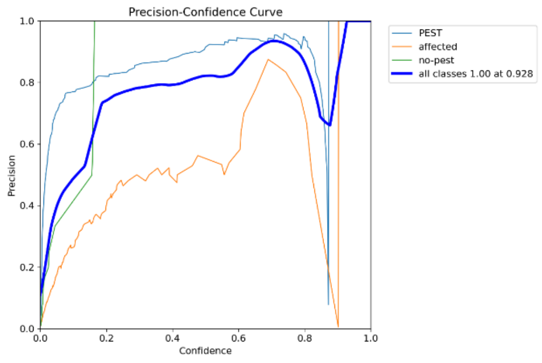


Fig. 4.17 Precision confidence curve for validation of model.

4.4.2.1 Axes:

X-axis Represents the confidence level of predictions. This typically ranges from 0 to 1, where 0 indicates no confidence and 1 indicates full confidence in the prediction. Y-axis represents the precision of the model. Precision is the ratio of true positive predictions to the total number of positive predictions (true positives + false positives).

4.4.2.2 Curves:

Different Coloured Lines: Each coloured line represents the precision-confidence relationship for different classes. Light Blue (PEST), Orange (affected), Green (no-pest), and thick blue line represent the precision across all classes combined at a specific threshold of 0.928.

i. PEST (Light Blue Line):

As confidence increases from 0 to about 0.2, precision increases sharply, indicating that at low confidence levels, there are many false positives. Precision continues to increase, peaking around a confidence level of 0.6 to 0.8. After this peak, precision drops, suggesting that extremely high-confidence predictions may not always be precise.

ii. Affected (Orange Line):

Initially, precision increases with confidence, but at a slower rate compared to PEST. Precision peaks at a lower value (around 0.6) and then decreases more erratically. This suggested that the model has more difficulty in making precise predictions for the 'affected' class as confidence increases.

iii.No-pest (Green Line):

This line showed a sharp increase in precision very quickly, reached a high level at a low confidence threshold. It remained high and stable, indicated that predictions for the 'no-pest' class are precise even at lower confidence levels.

iv.Threshold of 0.928:

This line indicated that at a confidence threshold of 0.928, the precision for all classes combined is 1.00. This means that if the model only considers predictions with a confidence level of 0.928 or higher, it will make no false positive errors (100% precision).

4.4.3 Recall-Confidence Curve

The graph shown in Fig. 4.17 is a recall-confidence curve, which showed how precision changes with varying confidence thresholds for different classes.

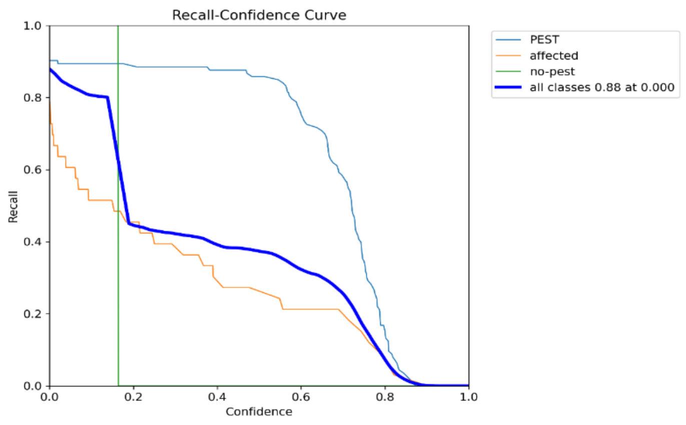


Fig. 4.18 Recall confidence curve for validation of model

4.4.3.1 Axes:

X-axis, represents the confidence threshold of the model. The confidence threshold determines the minimum confidence score a prediction must have to be considered a valid detection. It ranges from 0 to 1. Y-axis, represents the recall, which is the proportion of actual positive instances that were correctly identified by the model. Recall also ranges from 0 to 1.

4.4.3.2 Lines:

PEST (Blue Line) line shows the recall for the 'PEST' class as a function of the confidence threshold. The Orange Line showed the recall for the 'affected' class as a function of the confidence threshold. No-pest (Green Line) line showed the recall for the 'no-pest' class as a function of the confidence threshold. All classes (Thick Blue Line) line showed the combined recall for all classes as a function of the confidence threshold. The label indicated that at a confidence threshold of 0.000, the combined recall is 0.88 indicating that 88% of all actual positives across all classes are detected. The recall decreases as the confidence threshold increases, higher confidence thresholds result in fewer detections.

4.4.4 Precision-Recall Confidence Curve

A Precision-Recall (PR) curve as shown in Fig.4.18 is a graphical representation used to evaluate the performance of a binary classifier, particularly in scenarios where the classes are imbalanced.

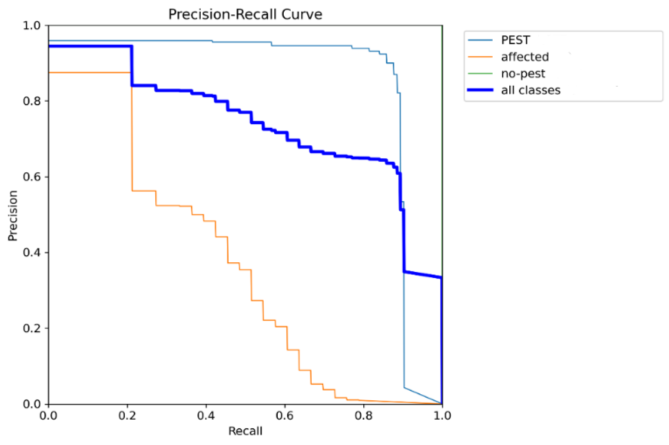


Fig. 4.19 Precision-recall curve for validation of the model

4.4.4.1 Axes:

The X-axis represents recall and the Y-axis represents Precision.

4.4.4.2 Lines:

Different Coloured Lines: Each coloured line represents the precision-Recall relationship for different classes. Light Blue (PEST), Orange (affected), Green (no-pest), and thick blue line represent the precision-Recall across all classes.

i. "PEST" Curve (blue curve):

This curve started with high precision and high recall, indicating the model's ability to correctly identify "PEST" instances with minimal false positives and false negatives initially. The curve maintained high precision as recall increases but showed some decline towards the end. Thus, this curve represents the precision-recall relationship for the class labeled "PEST" with an average precision

ii. "affected" Curve (orange curve):

This curve showed that precision dropped quickly as recall increased. This suggested that while the model correctly identified some "affected" instances, it also had a high rate of false positives as it tried to recall more instances. This curve represents the precision-recall relationship for the class labeled "affected".

iii. "no-pest" Curve (green curve):

The green curve indicated very high precision and recall for the "no-pest" class. This curve represents the precision-recall relationship for the class labeled "no-pest". This suggests the model performs excellently in identifying "no-pest" instances with almost no false positives and false negatives.

iv. Overall Curve ("all classes"): Thick blue colour

The thick blue curve represented the average performance across all classes.The curve started with high precision and recall but showed a gradual decline, indicating the model's performance varies across different classes.

4.4.5 Average precision

Average precision is high when both precision and recall are high, and low when either of them is low across a range of confidence threshold values. The range for AP is between 0 to 1. Here, it is estimated as AP=0.91, indicating the model is good. Since the Average Precision is nearing 1, the model is efficient.

4.6.6 Mean Average Precision

It is the average of the average precision over all classes. mAP is a comprehensive measure that summarizes the precision-recall performance across different classes and thresholds. If mAP is high for a matric used to measure the performance of a model that focuses on object detection tasks and information retrieval on images would be done accurately. Here it is estimated as mAP =0.89. Both average precision and mean average precision indicated good performance of the model.

4.6.7 Loss function curve

The loss function curves of the model is shown in Fig. 4.20. The details of the curves are as follows.

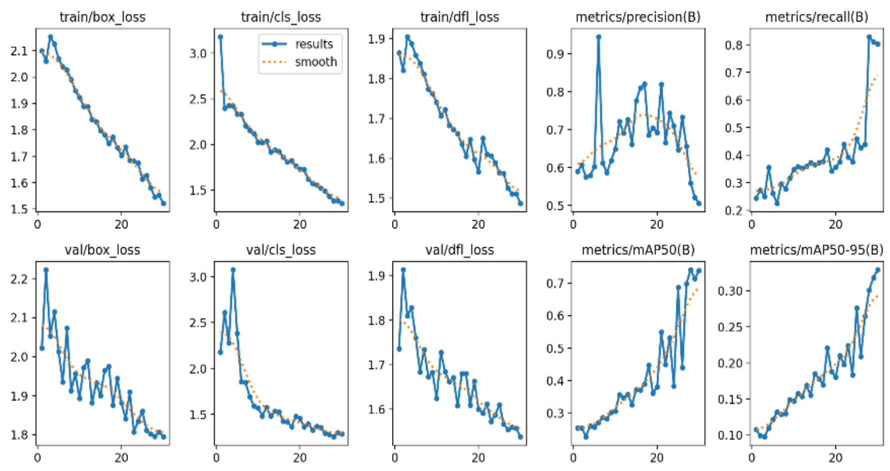


Fig. 4.20 Loss function curve of YOLOv8 model training and validation

Top Row represents Training Metrics

i.train/box_loss:

X-axis (Epochs/Iterations) represents the training time, usually in terms of epochs or iterations. Y -axis (Loss Value) represents the value of the bounding box loss.

This plot showed the loss associated with the bounding box predictions during training. The loss started around 2.1 and gradually decreased to around 1.5. This indicated that the model is improving its ability to predict bounding boxes correctly over time.

ii. train/cls_loss:

This plot shows the classification loss during training. The loss started around 3.0 and decreased steadily to about 1.5. This showed that the model is improving in classifying objects correctly as training progresses.

iii. train/dfl_loss:

This plot likely shows the distribution focal loss during training, a type of loss used to improve the quality of predicted bounding boxes. The loss started at approximately 1.9 and decreased to about 1.5, indicating improvement in the model's predictions.

iv. *metrics/precision(B):*

This plot shows the precision metric during training. Precision measures the accuracy of the positive predictions. Precision fluctuates, but shows an overall increasing trend towards the end, stabilizing around 0.8. This suggested that the model becomes more accurate in its positive predictions.

v. metrics/recall(B):

This plot shows the recall metric during training. Recall measures the ability of the model to identify all relevant instances. Recall started low and increased significantly to around 0.8. This indicated that the model is improving its ability to detect relevant instances over time.

Bottom Row represents Validation Metrics

vi. val/box_loss:

This plot shows the bounding box loss during validation. The loss started around 2.2, with some fluctuations, and decreased to about 1.8. This indicated that the model's performance on the validation set is improving, although it fluctuates more compared to the training loss.

vii. val/cls_loss:

This plot shows the classification loss during validation. The loss started high at 3.0, dropped sharply, and then continues to decrease more gradually to around 1.8. The initial sharp drop suggested that the model quickly learns to classify objects better, and further training improves this ability.

viii. val/dfl_loss:

This plot shows the distribution focal loss during validation. The loss started around 1.9, decreases with fluctuations, and stabilized around 1.5. The decreasing trend indicated better performance on the validation set.

ix. metrics/mAP50(B):

This plot shows the mean Average Precision (mAP) at the threshold of 0.50 during validation. mAP50 started low and increases steadily to about 0.7. This indicated that the model's ability to correctly predict object locations and classes is improving.

x. metrics/mAP50-95(B):

This plot shows the mean Average Precision across thresholds from 0.50 to 0.95 during validation. mAP50-95 showed a steady increase, starting around 0.10 and reaching approximately 0.30. This suggested an overall improvement in the model's detection performance across various thresholds.

General Observations about Loss Function Curves

- Smoothing: The orange dashed line represents a smoothed version of the metric, helping to visualize the overall trend without the noise of individual fluctuations.
- Training vs. Validation: Both training and validation losses showed a decreasing trend, indicating that the model is learning effectively. However, validation losses had more fluctuations, which is common and indicated variability in how the model generalizes to unseen data.
- Precision and Recall Trends: Precision showed some fluctuations but stabilizes towards the end of the training, while recall showed a more consistent increase, indicating overall improvement in model performance.
- mAP Trends: Both mAP50 and mAP50-95 metrics showed increasing trends, suggesting the model's predictions are becoming more accurate and robust.

Overall, these plots indicated that the model is learning and improving over time, as evidenced by the decreasing losses and increasing evaluation metrics.

i. Box Loss

Box loss in deep learning, also known as bounding box loss, is a loss function used in object detection tasks. The loss is calculated based on the difference between the predicted bounding box and the actual bounding box. The goal is to minimize the loss to improve the accuracy of object detection. The box loss is typically calculated for each object in the image, and then the losses are summed or averaged to get the total box loss. Box loss helps the model to learn the accurate location and size of objects in the image.

ii. Class Loss

Class loss in image detection, also known as classification loss or categorical loss, is a loss function used in object detection tasks to measure the difference between the predicted class labels and the true class labels of objects in an image. The goal is to minimize the class loss to improve the accuracy of object classification. The class loss is typically combined with other loss functions, such as box loss (for bounding box regression) to train object detection models like YOLO.

iii. DFL (Distribution Focal Loss)

DFL (Distribution Focal Loss) is a loss function used in image detection tasks, particularly in object detection algorithms like Faster R-CNN and RetinaNet. DFL loss aims to 1. Improve object detection accuracy, 2. Reduce the impact of class imbalance 3. Enhance the detection of hard examples. Object detection models can improve their accuracy and robustness using DFL loss, especially in scenarios with class imbalance and hard examples. All losses were found to be low, indicating the good performance of the model.

4.5 REAL TIME PEST DETECTION

Following training the YOLOv8 model for pest detection, the model was converted into a format that could be readily integrated into a web application. Fig.4.21 shows the homepage created for real-time pest detection. The pest detection on the web page was implemented by developing the front-end using HTML and the back end using Python in PyCharm. This setup enabled pest detection by allowing users to upload images on the web page. Since no webcam was installed in the field, real-time pest detection was tested in three ways. I.e. the pest was detected by utilizing 1. Real-time photos taken by mobile camera from field 2. Real-time videos taken from

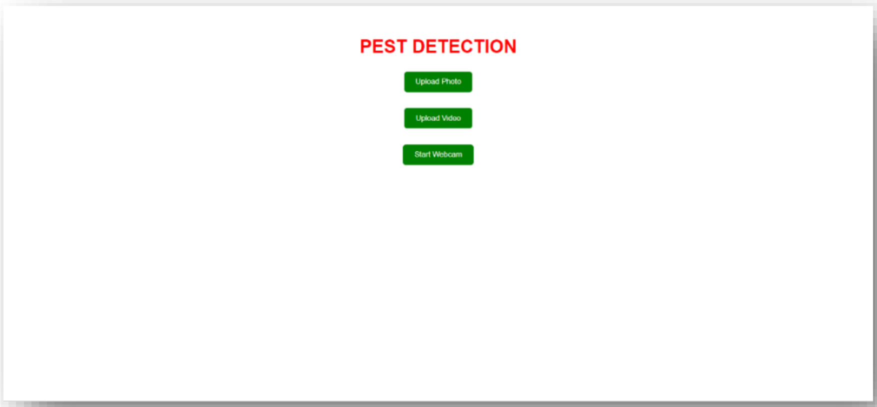


Fig.4.21 Home page of web page

the field 3. Real-time photos taken by the webcam of a laptop. The photo images, videos and webcam photos were fed to the web application homepage for pest detection.

4.5.1. Detection of pests from photos taken by the mobile camera from the real field

The images captured were fed into the home page of the web application and detected pests within a remarkable 2.2 milliseconds. Fig. 4.22 shows an example of images displaying the presence of pests along with bounding boxes representing their detection.

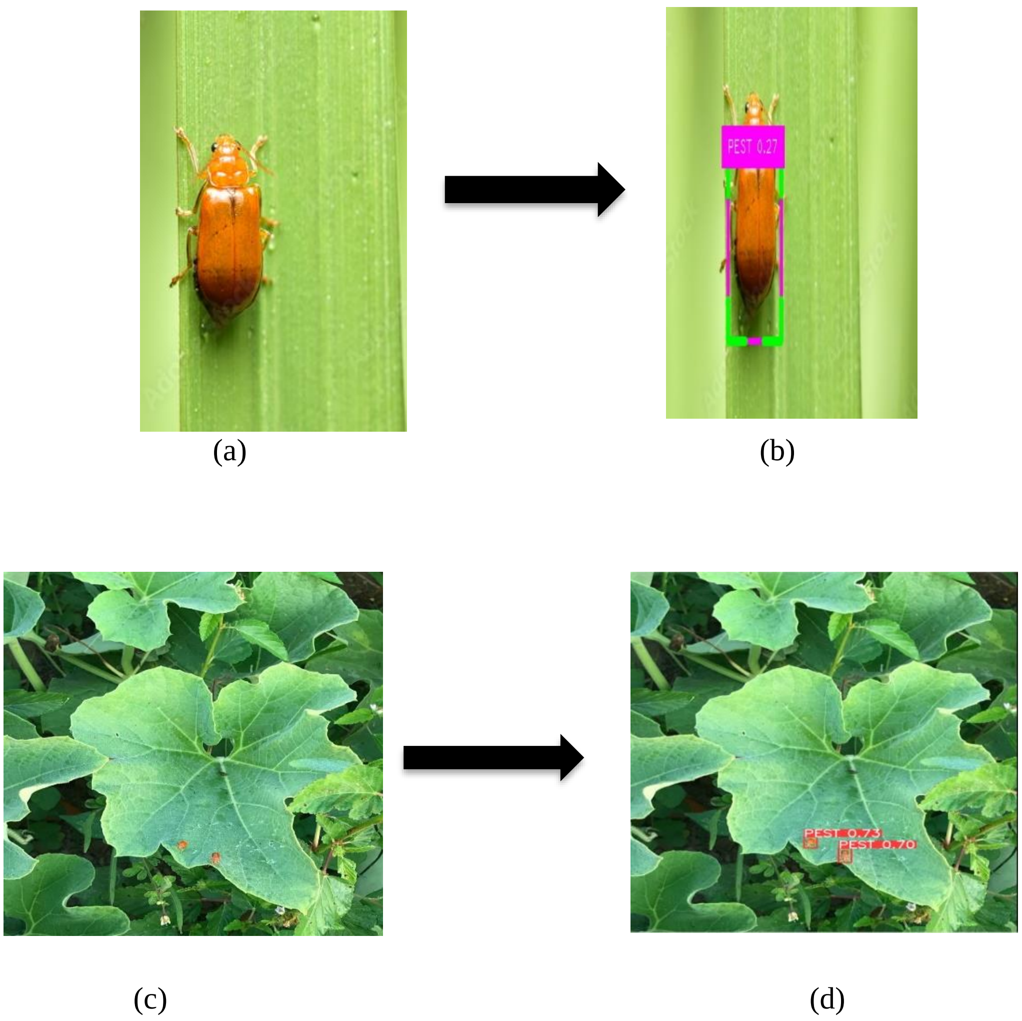


Fig.4.22 Pest detection by a photo taken from the field

4.5.2. Detection of pests from videos taken from actual field

In uploaded videos, the captured visuals were processed by the code, leading to the detection of pests at an impressive rate of 11 frames per second (FPS). Fig. 4.23 shows examples of video frames showcasing the presence of pests, accompanied by bounding boxes representing their detection.

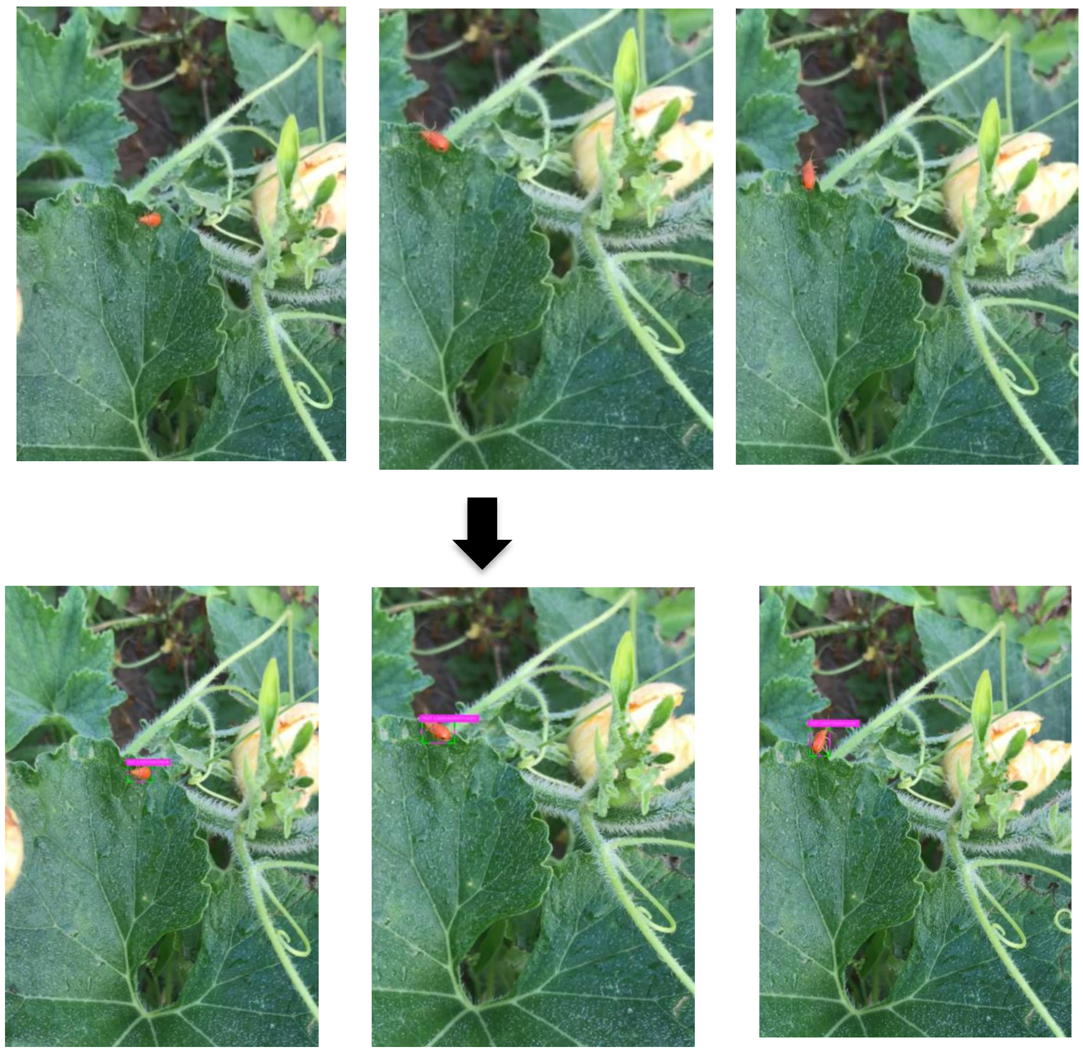
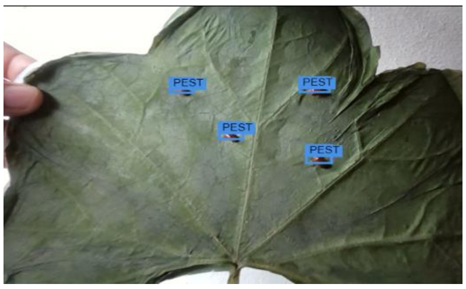


Fig. 4.23 Pest detection from videos in different frames

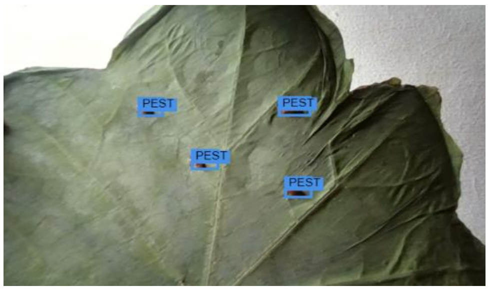
4.5.3 Real-time pest detection by using a laptop webcam

The real-time detection of pests stuck on the leaves was successfully achieved using a laptop webcam. The images below depict the frame-by-frame detection results in real-time.

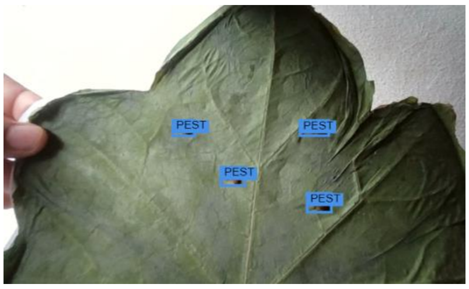
.



Frame 1



Frame 2



Frame 3

Fig. 4.24 Real time detection in different frames using laptop webcam

4.6 FRAMES PER SECOND, DETECTION SPEED, AND INFERENCE TIME

High FPS (Frames Per Second) and detection speed and low inference time indicate a robust and efficient model suitable for a wide range of practical applications. In this study, FPS was found 10-12, Inference time 3095 ms and the detection speed 1ms. Detection speed encompasses the overall efficiency and rapidity with which the model can detect and process objects within frames. It integrates both FPS and inference time to provide a comprehensive measure of the model's detection performance. All three parameters were found satisfactory, indicating the goodness of the model.

The developed web application has shown promising results in precisely and efficiently recognizing the target pests. The application can identify pests in real-time with the help of YOLOv8l model, making it a vital tool for farmers and pest control experts. The app’s user-friendly and simple features make it accessible to a larger audience, allowing even non-experts to spot pests.

Summary And Conclusion

CHAPTER - V

SUMMARY AND CONCLUSION

The study proposed a deep learning-based object detection model for pest detection in an agriculture field crop. The study was conducted in a pumpkin field in the KCAE&FT instructional farm to detect the pest ‘red pumpkin beetle’. The object detection model YOLOv8l was used for the study. Roboflow was used as the conversion tool for customized data set preparation. The images were collected from the field and the website ‘Shutterstock’ to prepare the dataset for training the model. The study consisted of 570 images which were meticulously curated from various shooting distances and heights, incorporating diverse levels of occlusions and lighting intensities, and 40 images were taken from a website named ‘Shutterstock’. Thus, 610 images were subjected to thorough pre-processing and augmentation techniques to enhance dataset diversity and quality by using “Roboflow,” which is a comprehensive platform designed to facilitate the development and deployment of customized datasets. Model training was done in Google Colab, it provides the YOLO version, including YOLOv8l, with intensive training, guaranteeing a quick and effective training procedure. It is developed with the help of libraries like Ultralytics. When training the network model for red pumpkin beetle object detection, the dimensions of the input image are uniformly modified to 800X800px. The ‘Adam W optimizer’ in YOLOv8 is utilized with a total of 50 epochs with a learning rate of 0.002 and batch size of 16. The size of the custom weight file ‘last.pt’ generated was 65 MB. The number of convolutional layers in the trained model was 56. The model was validated and tested in Pycharm using libraries Ultralytics, Opencv, and Flask.

YOLOv8l enables the real-time processing and streaming of frames from uploaded videos for pest detection. The output presents Box metrics, offering insights into the model's object detection capabilities. YOLOv8l was shown to be the most successful model among the different variants of YOLOv8 for the data used in this study, with a remarkable of precision (P) 86%,accuracy 83% mean average precision (mAP) .89, F1-score 86.9%, and recall 88% respectively add more about the model's accuracy and performance

A web application was developed to aid farmers and agricultural professionals for real-time pest detection. The research showed that the algorithm counted correctly 83%, which was the best result in the study. The algorithm performed exceptionally well during this study, making very few errors in detecting pests. This high accuracy suggests that the algorithm could be useful for real-world farming.

This study demonstrated the effectiveness of YOLOv8l in detecting pests in images with high accuracy and speed. The model could be a valuable tool for farmers, agricultural professionals, and researchers to quickly and accurately identify pests, enabling early intervention and control. The model's ability to detect pests in real-time makes it a promising solution for integrated pest management strategies. Overall, this study provides a valuable contribution to the field of pest management in agriculture, demonstrating the effectiveness of deep learning-based object detection models for detecting pests and their potential for integration with real-time spraying technologies.

Reference

REFERENCE

Abiri, R., Rizan, N., Balasundram, S.K., Shahbazi, A.B. and Abdul-Hamid, H., 2023. Application of digital technologies for ensuring agricultural productivity. *Heliyon*.

Ai, Y., Sun, C., Tie, J. and Cai, X., 2020. Research on recognition model of crop diseases and insect pests based on deep learning in harsh environments. *IEEE Access*, *8*, pp.171686-171693.

Anwar, Z. and Masood, S., 2023. Exploring Deep Ensemble Model for Insect and Pest Detection from Images. Procedia Comput. Sci., 218, pp.2328-2337.

Ariza-Sentís, M., Vélez, S., Martínez-Peña, R., Baja, H. and Valente, J., 2024. Object detection and tracking in Precision Farming: a systematic review. Comput. and Electr. in Agric., 219, p.108757.

Bezabh, Y.A., Abuhayi, B.M., Ayalew, A.M. and Asegie, H.A., 2024. Classification of pumpkin disease by using a hybrid approach. Smart Agric. Technol., 7, p.100398.

Brucal, S.G.E., de Jesus, L.C.M., Peruda, S.R., Samaniego, L.A. and Yong, E.D., 2023, October. Development of Tomato Leaf Disease Detection using YOLOv8 Model via Roboflow 2.0. In 2023 IEEE 12th Global Conf. on Consumer Electr. (GCCE) (pp. 692-694). IEEE.

Chen, L., Li, S., Bai, Q., Yang, J., Jiang, S. and Miao, Y., 2021. Review of image classification algorithms based on convolutional neural networks. *Remote Sensing*, *13*(22), p.4712.

Chithambarathanu, M. and Jeyakumar, M.K., 2023. Survey on crop pest detection using deep learning and machine learning approaches. Multimedia Tools and Appl., 82(27), pp.42277-42310.

de Melo Lima, B.P., Borges, L.D.A.B., Hirose, E. and Borges, D.L., 2024. A lightweight and enhanced model for detecting the Neotropical brown stink bug, Euschistus heros (Hemiptera: Pentatomidae) based on YOLOv8 for soybean fields. Ecological Informatics, p.102543.

Ferentinos, K.P., 2018. Deep learning models for plant disease detection and diagnosis. *Computers and electronics in agriculture*, *145*, pp.311-318.

Hadipour-Rokni, R., Asli-Ardeh, E.A., Jahanbakhshi, A. and Sabzi, S., 2023. Intelligent detection of citrus fruit pests using machine vision system and convolutional neural network through transfer learning technique. Comput. in Biol. and Med., 155, p.106611.

Jiang, J.A., Tseng, C.L., Lu, F.M., Yang, E.C., Wu, Z.S., Chen, C.P., Lin, S.H., Lin, K.C. and Liao, C.S., 2008. A GSM-based remote wireless automatic monitoring system for field information: A case study for ecological monitoring of the oriental fruit fly, Bactrocera dorsalis (Hendel). Comput. and Electr. in Agric., 62(2), pp.243-259.

Jiang, T. and Chen, S., 2024. A Lightweight Forest Pest Image Recognition Model Based on Improved YOLOv8. Applied Sci., 14(5), p.1941.

Junaid, M.D. and Gokce, A.F., 2024. Global agricultural losses and their causes. Bull. of Biol. and Allied Scie. Res., 2024(1), pp.66-66.

Kasinathan, T. and Uyyala, S.R., 2021. Machine learning ensemble with image processing for pest identification and classification in field crops. Neural Computing and Appl., 33(13), pp.7491-7504.

Kwon, Y., Lee, C., Bak, S. and Jung, C., 2024. Proposal of an advanced YOLOX model for real-time detection of Vespa hornets (Hymenoptera; Vespidae), key pests of honey bees. J. of Asia-Pac. Entomol., 27(2), p.102234.

Lello, F., Dida, M., Mkiramweni, M., Matiko, J., Akol, R., Nsabagwa, M. and Katumba, A., 2023. Fruit fly automatic detection and monitoring techniques: A review. *Smart Agricultural Technology*, p.100294.

Li, W., Wang, D., Li, M., Gao, Y., Wu, J. and Yang, X., 2021. Field detection of tiny pests from sticky trap images using deep learning in agricultural greenhouse. Comput and Electr. in Agric., 183, p.106048.

Li, W., Zheng, T., Yang, Z., Li, M., Sun, C. and Yang, X., 2021. Classification and detection of insects from field images using deep learning for smart pest management: A systematic review. Ecological Informatics, 66, p.101460.

Li, Y., Nie, J. and Chao, X., 2020. Do we really need deep CNN for plant diseases identification. *Computers and Electronics in Agriculture*, *178*, p.105803.

Mankin, R.W., Machan, R. and Jones, R., 2008. Field testing of a prototype acoustic device for detection of Mediterranean fruit flies flying into a trap.

Ngugi, L.C., Abelwahab, M. and Abo-Zahhad, M., 2021. Recent advances in image processing techniques for automated leaf pest and disease recognition–A review. *Information processing in agriculture*, *8*(1), pp.27-51.

Noskov, A., Bendix, J. and Friess, N., 2021. A review of insect monitoring approaches with special reference to radar techniques. Sensors, 21(4), p.1474.

Önler, E., 2021. Real time pest detection using YOLOv5. Int. J. of Agric. and Nat. Scie,, 14(3), pp.232-246.

Paul, A., Machavaram, R., Kumar, D. and Nagar, H., 2024. Smart solutions for capsicum Harvesting: Unleashing the power of YOLO for Detection, Segmentation, growth stage Classification, Counting, and real-time mobile identification. Comput. and Electr. in Agric., 219, p.108832.

Potamitis, I. and Rigakis, I., 2015. Novel noise-robust optoacoustic sensors to identify insects through wingbeats. IEEE Sensors J., 15(8), pp.4621-4631.

Redmon, J., Divvala, S., Girshick, R. and Farhadi, A., 2016. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 779-788).

Rustia, D.J.A., Chao, J.J., Chiu, L.Y., Wu, Y.F., Chung, J.Y., Hsu, J.C. and Lin, T.T., 2021. Automatic greenhouse insect pest detection and recognition based on a cascaded deep learning classification method. *Journal of applied entomology*, *145*(3), pp.206-222.

Singh, V., Sharma, N. and Singh, S., 2020. A review of imaging techniques for plant disease detection. Artif. Intelligence in Agric., 4, pp.229-242.

Slim, S.O., Abdelnaby, I.A., Moustafa, M.S., Zahran, M.B., Dahi, H.F. and Yones, M.S., 2023. Smart insect monitoring based on YOLOV5 case study: Mediterranean fruit fly Ceratitis capitata and Peach fruit fly Bactrocera zonata. The Egyptian J. of Remote Sensing and Space Sci., 26(4), pp.881-891.

Syed, Q.A., Akram, M. and Shukat, R., 2019. Nutritional and therapeutic importance of the pumpkin seeds. Seed, 21(2), pp.15798-15803.

Tian, Y., Wang, S., Li, E., Yang, G., Liang, Z. and Tan, M., 2023. MD-YOLO: Multi-scale Dense YOLO for small target pest detection. Comput.and Electr. in Agric., 213, p.108233.

Yang, Y., Di, J., Liu, G. and Wang, J., 2024, January. Rice Pest Recognition Method Based on Improved YOLOv8. In 2024 4th Int. Conf. on Consumer Electr. and Comput. Eng. (ICCECE) (pp. 418-422). IEEE.

YOLO Performance Metrics - Ultralytics YOLOv8 Docs

Zhu, L., Li, X., Sun, H. and Han, Y., 2024. Research on CBF-YOLO detection model for common soybean pests in complex environment. Comput. and Electr. in Agric., 216, p.108515.

SMART PEST DETECTION FOR AN AGRICULTURAL FIELD CROP BASED ON DEEP LEARNING

By

NAVYA MARIAM PRASAD (2020-02-008)

FATHIMA HIBA K (2020-02-040)

VISHNUPRIYA V (2020-02-044)

VAISHNAVI AJAYAN A (2020-02-053)

ABSTRACT OF THESIS

Submitted in partial fulfilment of the requirement for the degree

Bachelor of technology

In

Agricultural Engineering

Faculty of Agricultural Engineering and Technology

KERALA AGRICULTURAL UNIVERSITY

DEPARTMENT OF IRRIGATION AND DRAINAGE ENGINEERING

KELAPPAJI COLLEGE OF AGRICULTURAL ENGINEERING AND FOOD TECHNOLOGY

TAVANUR- 679573, MALAPPURAM, KERALA, INDIA

2024

ABSTRACT

A study was conducted to develop smart pest detection for an agricultural field crop based on deep-learning object detection. The study selected the agricultural field crop pumpkin, and the red beetle pest was detected. The study developed a deep learning-based object detection model using the YOLOv8l. Roboflow was used as the conversion tool for customized data preparation. The performance and accuracy of the model were found to be satisfactory. The integration of the model with the web application was done for real-time pest detection.

The proposed approach has the potential to aid farmers in identifying the existence of pests, thereby diminishing the duration and resources needed for farm inspection. The YOLOV8l object detection model was implemented for the purpose of pest classification, localization, and quantification. The proposed pest detection approach demonstrated a noteworthy increase in performance in terms of precision (P) 86%, mean average precision (mAP) .89, F1-score 86.9%, and recall 88%. A web application was developed to aid farmers and agricultural professionals in real-time pest detection.

The study concluded that integrating deep learning techniques holds immense promise for revolutionizing smart pest detection in agriculture. By harnessing the power of artificial intelligence, farmers can transition towards more sustainable and efficient pest management practices, contributing to food security, environmental conservation, and economic prosperity.