



Paddy Pest Recognition Based on Deep Learning in Sri Lanka

**A Dissertation Submitted for the Degree of
Master of Computer Science**

**F.F.S.N Fernando
University of Colombo School of Computing
2021**



Declaration

I hereby certify that this thesis entitled “Paddy Pest Recognition System on Deep Learning” is entirely my work, and it has never been submitted nor is currently submitted for any other degree.

Student Name: F.F.S.N Fernando

Registration Number: 2018/MCS/021

Index Number: 18440212

Signature of the Student & Date

This is to certify that this thesis is based on the work of
Mr./Ms.

Under my supervision. The thesis has been prepared according to the format stipulated and is of an acceptable standard

Certified by,

Supervisor Name: Dr G.D.S.P. Wimalaratne

Signature of the Supervisor & Date

Acknowledgment

I would like to express my sincere gratitude to my supervisor Dr. G.D.S.P. Wimalaratne, Senior Lecturer of University of Colombo School of Computing, for his invaluable guidance and support. He always responded to my questions so promptly, and the valuable comments and advice helped me professionally write the thesis.

I thank my beloved parents for bringing me into this world and guiding me with the love and affection to fulfill my education. Finally, I like to thank all the people who had helped me throughout this project.

Abstract

Regarding the majority of crops, one of the significant factors affecting crop yield is pests' disasters. Since most pest species are highly related, pest detection on field crops, such as soybean, rice, and other crops, is likewise challenging than generic object detection. Presently, distinguishing pests in crop fields relies on manual classification, which is highly time-consuming and costly. This work proposes a convolutional neural network model to resolve the problem of the multi-classification of crop pests. The model can fully use the advantages of the neural network to extract multifaceted pest features comprehensively.

This system identifies ten paddy pest's species Common Thrips, wheat phloeothrips, wireworm, grain spreader thrips, rice leafhopper, rice water weevil, small brown planthopper, rice gall midge, yellow rice borer, and paddy stem maggot, which are multi-scale which improves the mean average precision (mAP) to 82.38%; the result increased by 1% with the original model. We embrace this model to a versatile stage to allow each rancher to utilize this program to analyze bothers progressively and give ideas on bug control. We planned a paddy bug imaging data set with ten ordinary paddy bothers and contrasted the highlights of many models with a pick as the ideal model. Besides, for the high mAP, we have utilized information expansion (DA) and added a dropout layer. The analyses are executed on the Android application we created. The outcome shows that our methodology outperforms the first model obviously and is helpful for a coordinated bug the executives. This application has made ecological flexibility, reaction speed, and precision by standing out from the past works. It has the assistance of minimal expense and primary activity, reasonable for the irritation checking mission.

Table of Contents

Acknowledgement	IV
Abstract	III
Table of Contents	IV
1. Introduction	1
1.1 Motivation	3
1.2 Problem	3
1.3 Aim	4
1.4 Objectives	4
1.5 Scope	4
1.6 Structure of the Dissertation	5
2. Literature Review	6
2.1 Introduction	6
2.2 Related work	6
2.3 Summary of Literature Review	9
2.3.1 Taxonomy of Image Classification Models	10
2.3.2 Comparison of Classifiers	11
3. Methodology	14
3.1 Introduction	14
3.2 Problem Analysis	14
3.3 Methods	15
3.4 Process	16
3.5 Machine learning process	18
3.5.1 Dataset	18
3.2.3 Experiments	23
3.5.2 Implementation	23
3.5.3 Transfer learning	28
4. Evaluation and Results	31

4.1 Results.....	31
4.1.1 Tflite Models Size	32
4.1.2 Tflite Models Accuracy	32
4.1.3 Testing Flow.....	33
4.2 Evaluation.....	37
4.2.1 Intersection over Union (IoU)	37
4.2.2 Confusion Matrix	38
Final Classification Report YOLOV4	40
Final Classification Report YOLOV4 – CUSTOME.....	41
5. Conclusion and Future Work.	44

List of Figures

Figure 2.1: Image Classification Models	11
Figure 2.2: Image Classification Models	11
Figure 3.1: Problem Analysis Process	14
Figure 3.2: A process for Systems Development Research	15
Figure 3.3: A process for Systems Development Research	15
Figure 3.4: IP102 Data Set	19
Figure 3.5: Noisy Data of IPI02	20
Figure 3.6: Image Annotation of seleted Pest	20
Figure 3.7: Augmented Images	23
Figure 3.8: Training Model	23
Figure 3.9: Yolov4	16
Figure 3.10: Camera View	25
Figure 3.11: Image Localisation view	16
Figure 4.1: Test Result	32
Figure 4.2: Image Bounding Box	32
Figure 4.3: Training Accuracy	42

List of Tables

Table 2.1: Comparison of Classifiers	13
Table 3.1: Original Data Set	19
Table 4.1: Table illustrating the relationship between speed, mAP, and different configurations of model input layers	31
Table 4.2: ML Model sizes	31
Table 4.3: Tflite Model Accuracy.....	31
Table 4.4: Yolov4 Model Accuracy	41
Table 4.5: Yolov4-Custom Model Accuracy.....	41
Table 4.6: Average Model Accuracy	42

LIST OF ABBREVIATIONS

Abbreviation	Description
SSD	Single-Shot MultiBox Detector
CV	Computer Vision
YOLO	You Only Look Once
CNN	Convolutional Neural Networks
SVM	Vector Machine
KNN	K Nearest Neighbours
ANN	Artificial Neural Networks
NB	Naive Bayes
DA	Data Augment

1. Introduction

Rice is a significant crop in Sri Lanka, but a tremendous economic loss does happen for paddy farmers because of plant conditions and paddy pests every year. The fisheries and agriculture sectors perform a crucial role in the gross production of Sri Lanka. The development of an automatic system for paddy field insect pest identification is of great importance. Computer vision procedures have of high significance in the programmed recognizable proof of the pictures of bug bugs. Those procedures can diminish work and work on the speed and accuracy of recognizable evidence and conclusion contrasted with the manual technique. In such a manner, perceiving paddy field insect nuisances is testing since they are exceptionally expressed and display great intra-bug variety in size and shading. Some insect irritations are hard to distinguish outwardly, regardless of conspicuous dorsal design. The manual characterization of before-referenced creepy crawly bugs in paddy fields can be tedious and requires excellent specific ability. The undertaking develops testing when bug bugs are perceived from still pictures utilizing a robotized framework. Photographs of one bug nuisance might be taken from various perspectives, jumbled foundations, or may endure change, like pivot and commotion.

Subsequently, the right and convenient analysis of paddy irritations is significant for the yields of harvest creation. For a considerable time frame, bug ID chiefly relies upon crop professionals and experienced ranchers by the morphology. Notwithstanding imagined title is emotional, wasteful, and deferred. Subsequently, it is needed to chase out a level-headed, compelling, and fast strategy for recognizing bugs.

Since bug pictures are frequently extended for additional handling after the securing, images are regularly utilized as information, and seeing is one of the first difficult errands inside the earth of PC vision (CV)(Fuentes, 2017). CV alludes to the robotized extraction, assessment, and comprehension of important data from one picture or a succession of pictures (Brownlee, 2019). The obligations of CV joint arrangement, object location, localization, division. (Hwang and Kim, 2016), and so forth. Object discovery is the strategy of getting real example objects in pictures or recordings; it produces bounding boxes and complexity names to find and perceive various targets. For the most part, object recognition has three stages:

separating highlights from the info picture, choosing a discovery window, and planning a classifier (Ding and Zhao, 2018). the normal openness calculation is shaped on handcraft elements and shallow teachable models, requiring heartiness and responsiveness to assortment changes.

Furthermore, the locale choice system upheld by sliding windows is computationally costly and tedious and commonly creates excess windows (Zhao, 2019). for a considerable length of time errands, the identification calculation upheld by profound learning performs better compared to conventional discovery methods (Voulodimos, 2018). Profound learning can learn high-dimensional information proficiently and has arrived at explicit prompts in the fields of arrangement, discourse acknowledgment, tongue handling, suggestion framework, and programmed driving (LeCun, 2015). Profound neural organizations don't need planning highlights physically, making them more solid and versatile for different varieties inside the picture data(Ubbens, 2018).

By the quick development of profound learning in object identification lately, numerous profound location models have been proposed. Commonplace identifiers incorporate Faster R-CNN (Zhong, 2018), R-FCN (Dai, 2016), YOLO (you just look once) (Redmon, 2016), SSD (single-shot multibox finder) (Liu, 2016), and so on The normal assessment of these two-stage locators is that they are in the recognition accuracy however need a high computational expense. One-stage pointers, like YOLO and SSD, have lower precision but quicker speed, which implies they can be utilized on cell phones. In the wake of dissecting the mAP esteems, identification time, and model size, this paper picked YOLO as the discovery system.

The developing interest in supportable agribusiness has as of late stretched out its application to analyze creepy crawlies (Bechar and Moisan, 2010). Some exploration assessments have been performed to foster insect identification models dependent on profound realizing, whose outcomes showed that they are promising in bug discovery undertakings contrasted with conventional recognition calculations and can all the more likely adjust to complex field conditions. However, for this review, there are three primary difficulties: first, there is an absence of an accessible public bug dataset, and the vast majority of the ebb and flow explores use bug pictures gathered in ideal research center conditions (Zhong, 2018). Second, the field climate is mind-boggling, so the current bug models couldn't be

straightforwardly utilized for insect recognition, which needs further preparation and acclimation to accomplish the best presentation. Third, the hardware of the vermin recognition framework used first is costly and cumbersome and of helpless practicability.

In this research, a variety of paddy insects detection models based on deep convolutional neural networks have been evaluated and tested on paddy insect's dataset to construct a detection system for insects of paddy in complex environments.

1.1 Motivation

Early detection would help minimise the usage of pesticides and would guide the selection of pesticides. Millions of dollars are being spent on safeguarding the crops annually. Pests waste the crops and, thus, are very critical for the overall growth of the crop. One method to guard the crop is early pest detection to protect the crop from insect attack. The development of CNN models that can be deployed on mobile devices will improve the ability of farmers to access and profit from this technology uniquely, given the ubiquity of smartphones.

The automated system can identify the insect's realities, and harmful insects can be correctly detected. It is most helpful for protecting the crop that will help prevent damage from invasive insect species newly caught.

1.2 Problem

However, achieving this task seems impossible due to various obstacles. One of the main problems is different kinds of Insect attacks on the rice fields Issues with Paddy Insects Control Methods. Farmers lose an estimated average (Senanayake, 2021) of rice crops to insects and diseases every year. In addition to strict crop management, timely and accurate diagnosis can significantly reduce losses and Uniform spraying. Traditional concepts cannot investigate this problem accurately because the tiny area only can access, and also, Insects are very small. Image processing based on rapid development is the new way to recognize Paddy Insects in Precision Agriculture systems. Automated localised Insect detection in rice fields to avoid excessive uniform spraying; will result in high, good quality yield with

low production cost. If Insects are detected, appropriate measures can be taken to protect the crop from a significant production loss at the end.

1.3 Aim

Investigate and Develop an automated framework to identify pests in real-time using Image processing and machine learning techniques that perform well.

1.4 Objectives

The project's objective is aligned with the end goal of developing a tool to solve the problem mentioned above using machine learning and image processing techniques.

The accompanying goals can be found in this examination.

1. Conduct the systematic literature review on pest detection-based algorithms.
2. Extract features of the Sri Lankan pest species by individually comparing each feature with the new image to the database and finding new required matching features.
3. Investigate, develop, and fine-tune appropriate deep learning approaches working on the mobile platform.
4. Train with a small collection of the data.
5. Plan and Conduct evaluation experiment.

1.5 Scope

This system identifies ten paddy pest species Common Thrips, wheat phloeothrips, wireworm, grain spreader thrips, rice leafhopper, rice water weevil, small brown planthopper, rice gall midge, yellow rice borer, and paddy stem maggot, which are multi-scale. This system includes a mobile application. Mobile application is the main application with many features. Mobile application runs on Android platform. It comes with features such as Identifying paddy pests in real-time, displaying identified pest names with a considerable distance. Users can identify pests by taking a picture from a camera video. A

mobile-supported machine learning model should be developed to identify pests without the internet. When developing the mobile application, the size of the application, the performance, memory, computational power of the device, and the exactness of the results are considered. There is a limitation to identifying very tiny objects.

1.6 Structure of the Dissertation

The sections of this paper depict how the undertaking is arranged, coordinated, and executed. The primary section depicts the issue that was tended to and the result of the undertaking.

The subsequent section portrays the writing survey that was finished during the task. The literature review described related work and methods applied and used by other researchers in the image classification domain.

The third part portrays the chosen procedure for bug characterization subsequent to doing the writing survey. This part additionally depicts the engineering, plan stage, and execution of the application. This chapter provides a complete description of how the dataset was collected, how the dataset was preprocessed, and how the application was implemented.

The fourth chapter describes the results obtained after the project was completed. This chapter also describes the techniques applied to evaluate the project work and the accuracy of applied methodologies in the project.

The final chapter describes conclusions that were observed after the project was completed

2. Literature Review

2.1 Introduction

This section introduces the related work of paddy insects and related feature recognition methods and reviews the existing datasets. Numbers of automated image detection and recognition techniques were proposed several decades ago. Recently, the image classification and detection of insects has been explored. Machine learning algorithms have been used broadly in many studies to achieve this purpose. This section discusses the pros and cons of the researches which have been followed.

2.2 Related work

Deep learning technology broadly attracts the attention of researchers (Lecun, 1998, Shelhamer, Ren, 2016). Deep convolutional neural networks (CNNs) such as ResNet (Hui, 2016) and GoogleNet (Szegedy, 2015) show good performance in the image classification task. Several works (Wu, Zhou, 2019, Dai, 2020) successfully use CNNs to resolve the problem of insect pest recognition.

Ngugi, 2020 has introduced a thorough audit of ongoing exploration work done in plant infection acknowledgment. Given adequate information is accessible to preparing, profound learning strategies can perceive bugs and infections with high exactness. The effect of social occasion colossal datasets with enormous irregularity, data increment, move learning, and portrayal of CNN sanctioning guides in making request accuracy has been discussed. A relating execution evaluation of 10 state-of-the-art CNN models on the leaf affliction recognizing verification task has been driven. ResNet-101, DenseNet201, and Inceptionv3 CNN structures are the most fitting models for use in ordinary handling conditions, while ShuffleNet and SqueezeNet are the most suitable plans for adaptable and embedded applications.

Kasinathan, 2020. proposed the insect pest detection algorithm that consists of foreground extraction and contour identification to detect the insects for 24 insect classes using Wang, Xie, Deng, and IP102 datasets in a highly complex background using the shape features and applying machine learning methods such as support vector machine (SVM), k-nearest neighbors (KNN), artificial neural networks (ANN), naive Bayes (NB) and convolutional neural network (CNN) model. However, implement a deep Convolutional Neural Network (CNN) model to detect insects with class labels for larger insect datasets.

So far, most exploration-based item discoveries proposed a further developed calculation dependent on the Faster R-CNN calculation with a combination of relevant data (SC-Faster R-CNN) and skipped pooling. Xiao et al. presents the Faster R-CNN framework and add a skip pooling strategy and component extraction model that joins coherent information and uses coordinated anchors with world-class execution rather than RPN. To make sure that it has high recognizable proof execution, they drove evaluation and relationship tests between the proposed estimation and Faster R-CNN (Nanni, 2020), YOLOv3, SSD512, HyperNet, and MS-CNN and has high area study generally and for objects that are nearly nothing or somewhat thwarted. The result set explains the component execution of the YOLO family. Regardless, it really has explicit conditions in the disclosure of deformed, turning, and masked things. The guideline presented in this paper are summarized as (I) another RPN estimation is used, coordinated anchor RPN (GA-RPN), which recognize the region and match the article shape and state of the anchor to create a small anchor can solidly include the thing. This procedure can lessen the calculation time while managing a high audit rate. This procedure can lessen the calculation time while keeping a high audit rate. (ii) A setting information incorporate extraction network entwined with significant information is used. The association can oblige the important data on a somewhat hindered object for the general method and use the setting-focused data around the article as the information focal point for the thing's component acknowledgment. (iii) To deal with the issue of fewer component data from more humble articles in the alone component layer, this paper proposes to skip pooling, which combines the components of different component layers, widely further fostering the verbalization strength of features, and is fitting for little thing distinguishing proof.

Jiang, 2019 proposed that the Real-Time Detection of Apple Leaf Diseases mainly focuses on data augmentation and image annotation technologies; the proposed INAR-SSD (SSD

with Inception module and Rainbow concatenation) model has a model a focus been trained to detect leaf diseases because SSD is much faster than that Faster R-CNN.

All the above strategies could accomplish palatable exhibitions. In any case, they intend to distinguish bugs under basic foundation conditions rather than the perplexing foundations ordinary of the field climate. Their applications are restricted to explicit vermin and can't be generally utilized. Accordingly, some bug localisation and acknowledgment strategies for crops have been proposed lately.

Jiang, 2019 has proposed a continuous discovery approach dependent on improved convolutional neural organizations for apple leaf infections. The profound learning-based methodology can naturally remove the discriminative elements of the ailing apple pictures and distinguish the five normal kinds of apple leaf contaminations with extraordinary exactness continuously. In this review, to guarantee acceptable speculation execution of the proposed model and an adequate apple illness picture dataset, an aggregate of 26,377 pictures with uniform and complex foundations. INAR-SSD, was planned by presenting the GoogLeNet Inception module and incorporating the Rainbow link to upgrade the multi-scale illness object recognition and little unhealthy article discovery exhibitions. The exploratory comes to fruition to create the impression that the INAR-SSD show understands an area execution of 78.80% mAP on ALDD, with a tall area speed of 23.13 FPS. It generally comes to outline that the original INAR-SSD show gives an elite exhibition plan for the early finish of apple leaf ailments that can perform constant disclosure of these diseases with higher precision and speedier area speed than past techniques. They utilize an extensive enormous dataset for preparing the model.

CNN and the current item identification strategies dependent on a convolutional neural organization are not palatable for multi-scale insect localisation and acknowledgment in the field. Accordingly, Li,2020. proposed a successful information increase technique for CNN-based vermin localisation and acknowledgment. An information increase strategy is proposed for creating a multi-scale identification model.

The examinations show that tomatoes have been highlighted in the main number of studies. Apples, soybean, maise, and grapes are likewise famous among analysts in this field. Just three examinations analyzed irritation acknowledgment. Cheng, 2020 considered bugs are explicitly happening in tomato crops. Accomplishing the insect grouping with higher

precision progressively is a difficult issue in the conduct of shadow, soil, leaves, branches, bloom buds, and so on in significant agribusiness field crops. The grouping exactness is analyzed between different AI strategies that incorporate ANN (Kasinathan, 2020).

Kanesh and Ramanan, 2014 considered a structure to examine twenty types of paddy fields creepy crawly bugs noticeably saw in Jaffna paddy fields utilizing slope-based highlights. This is proposed the future progression into cell phones with a slight alteration. This can additionally empower ranchers when an obscure picture of a paddy field bug nuisance is taken through a telephone; the model can foresee the species and backing ranchers for additional activity.

Notwithstanding, up to the current time, no works have been published for paddy pest recognition with small datasets using deep learning and crop pest identification systems development on mobile devices.

2.3 Summary of Literature Review

Different techniques were distinguished in the picture grouping setting by going through many examination papers. These systems utilize various procedures in picture grouping, and they have other pros and cons.

The findings obtained during this study were summarised what's more, further examined.

2.3.1 Taxonomy of Image Classification Models

According to the above research papers following image classification methods were identified, as shown in Figure 2.1

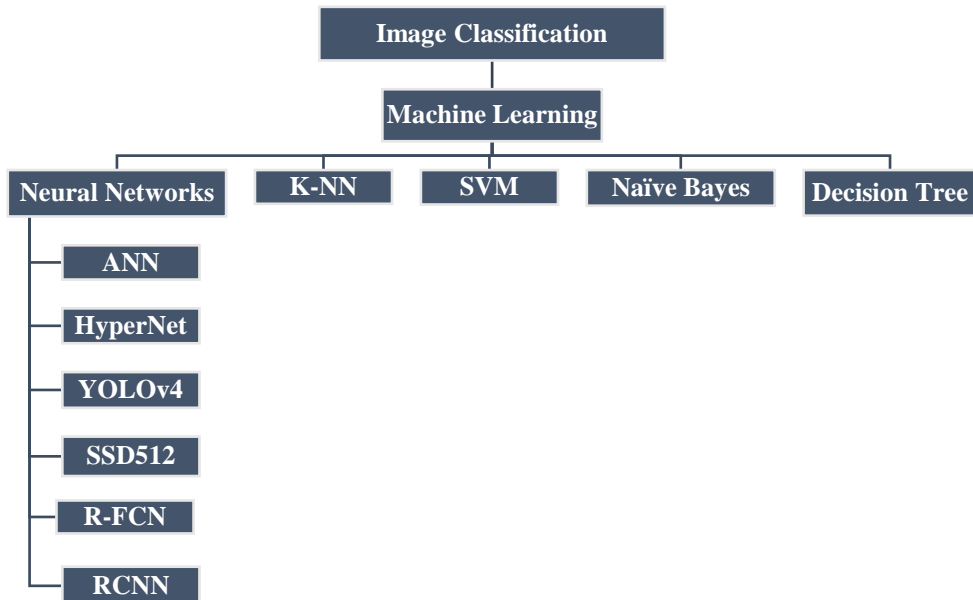


Figure 2.1: Image Classification Models

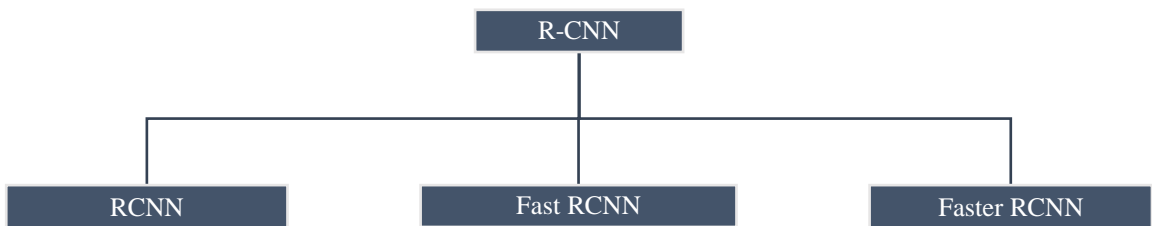


Figure 2.2: Image Classification Models

2.3.2 Comparison of Classifiers

Table 2.1 is conducted to find related work and identify areas that are interesting to investigate and can be further developed in this paper, establish a fundamental understanding of the field of artificial intelligence and the machine learning training process. Research questions are re-iterated during this step of the purpose to study important areas in the domain and bridge knowledge gaps where they arise. Table 2.1 shows the summary of the model classifiers.

Table 2.1: Comparison of Classifiers

Classifiers	Advantages	Disadvantages	References
ANN	Capable in distinguishing complex nonlinear relationship between independent and dependent variables.	The great tendency of data overfitting.	(Azlah, 2019)
	Simplistic statistical training.	Bigger computational load.	(Azlah, 2019)
CNN	Multiple features can be extracted simultaneously.	High computation level.	(Sharma, 2018)
	Robust to noise	No capable for generalisation.	(Sharma, 2018)
KNN	No training is needed.	Susceptible to noise.	(Sayali D. Jadhav and H. P. Channe2, 2016)
	Robust in terms of research space.	Lazy learning.	
	Simplest classifier.	Costly testing for each instance.	
SVM	Great generalisation potential.	Speed and size constraints for both training and testing	

	Exceptionally robust.	Complex algorithm structure.	
		Slow training.	
Naïve Bayes	It requires short computational time for training.	The Naive Bayes classifier needs a huge dataset to obtain accurate results	(Sayali D. Jadhav and H. P. Channe2, 2016)
	It increases the classification efficiency by removing the unrelated features.	Low accuracy as compared to other classifiers on few datasets.	
Decision tree	Decision Trees are straightforward and fast	It has a long training time.	(Sayali D. Jadhav and H. P. Channe2, 2016)
	It can also deal with noisy data.	Decision trees can have a significantly more complex representation for some concepts due to the replication problem	
	It supports incremental learning		
HyperNet	better Region Proposal and detection accuracy	small targets detection	(Jiang, 2018)
	Better Improvement trough RCNN		
SSD	it uses low-level feature maps to detect small targets and high-level feature maps to detect large targets	no delegated region proposal network and predicts the boundary boxes and the classes directly from feature maps in one single pass	(Hui, 2020)
R-FCN	subsequent object classification and bounding box regression	ineffective for small-scale detection	(Tang, 2020)
YOLOv4	Better performance and accuracy	cannot detect very close objects or small objects	(Jiang, 2018)

RCNN	Uses selective search to generate regions. Extracts around 2000 regions from each image Each image is transferred only once to the CNN, and feature maps are extracted.	High computation time as every region is passed to the CNN separately.	(Pulkit sharma, 2018)
Fast RCNN	Each image is transferred only once to the CNN.	Selective search is slow, and therefore computation time is still high	(Pulkit Sharma, 2018)
Faster RCNN	Replaces the particular search method with region proposal network, which made the algorithm much faster	Object proposal takes time, and as different systems are working one after the other	(Pulkit Sharma, 2018)

SDD, Faster RCNN gives good accuracy comparing other models, but performance is low for real-time detection. Yolo gives better performance and accuracy for real-time Object Detection. However, the work introduced in this review considered a more prominent number of CNN models and their upsides and downsides.

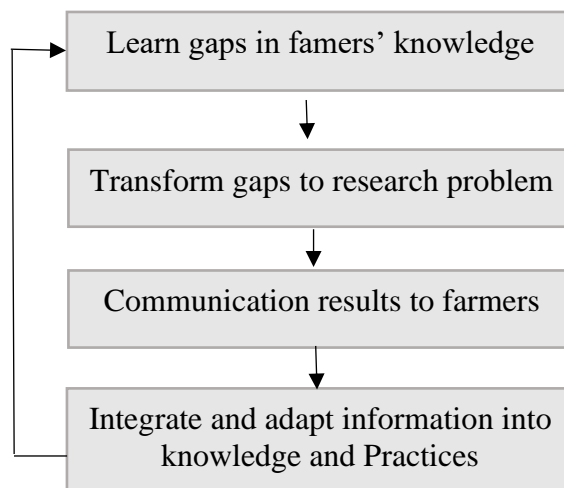
3. Methodology

3.1 Introduction

The methodology is how the project is implemented with the findings and the knowledge gained through the literature review. This chapter shows a complete overview of the image classification techniques used, the dataset details, how the dataset is preprocessed, overall project architecture, and the implementation.

3.2 Problem Analysis

The leaf-feeding care of bugs is harming and yield-diminishing. In light of this insight, Farmers pick insect sprays to kill bugs to secure their yields. Farmers appear to underline "killing" pests rather than forestalling misfortunes. Throughout the long term, ranchers have figured out how to utilize insect sprays that are more affordable and profoundly harmful to "worms," which may represent the high utilization of methyl parathion and monocrotophos. Asian rice ranchers appear to be caught in an endless loop of settling on some unacceptable choices with regards to creepy crawlies' concerns and bug spray utilization and have become casualties of insect spray misuse. They ought to have an under damping on the best way to utilize an insect spray accurately. It is apparent that to further develop ranchers' bug the board choices, we want to recognize processes that can assist with changing ranchers' view of bugs and their administration, as displayed in Figure 3.1.



3.3 Methods

This research is based on a deep learning approach consisting of the system development method. The iterative ability, being able to easily move between steps of the method, makes it suitable for investigating many different configurations. Creating at least two artifacts is necessary to answer the research questions mentioned in 1.4: a) an object detection system, and b) one or more object detection models trained to detect doors. We aspire to create multiple such models based on different neural networks to compare the performance. Here time is an important resource that limits the extent to which the generalizability may be investigated. Different architectures have different frameworks, tools, ways of interpreting features in data. A lot of time is needed to exhaustively create and test models. Thus the object detection system is designed so that it is general to the extent that any object detection model may easily be plugged in and out. A progression of analyses is directed to build up good practices in the machine learning process for the YOLO model, which improves performance. These experiments test the object detection system on a mobile device and without the mobile object detection application, directly on the test dataset in a desktop environment. In these experiments, the object detection models will be tested on one shared test dataset to measure how accuracy improves or diminishes about different configurations of training dataset composition, the usage of transfer learning, and hyperparameter optimisation. The experiments featuring the object detection system with a mobile device take place in various pests, noting the speed (detection per second) of the system but otherwise with the same intentions as previously mentioned. To evaluate the accuracy of models, we use the mAP metric

3.4 Process

Figure 3.2 shows the systems development research process

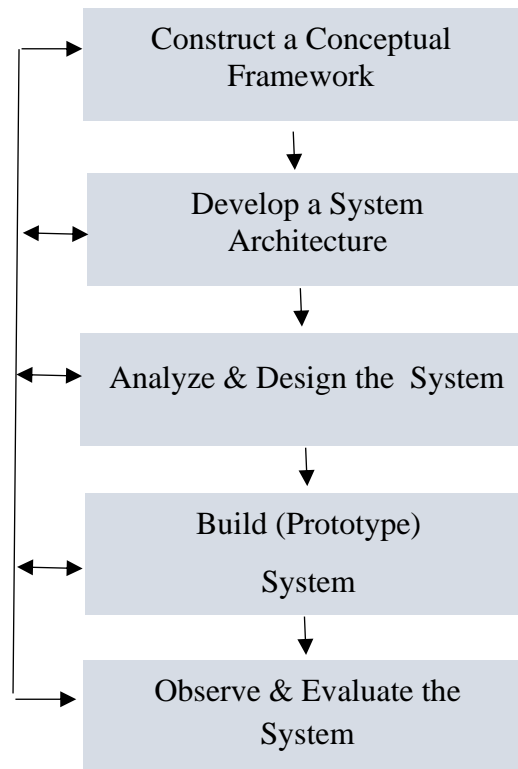


Figure 3.2: A process for Systems Development Research

Construct a conceptual framework

Following this, a literature review is conducted to find related work and identify areas that are interesting to investigate and can be further developed in this paper, establishing a fundamental understanding of artificial intelligence and the machine learning training process. Research questions are re-iterated during this step to study important areas in the domain and bridge knowledge gaps where they arise.

Develop a System Architecture

A system architecture is established for the object detection system. Logical components and their interactions with one another are found, which can be observed in Figure 3.3. Necessary software libraries and tools are obtained. The machine learning training process is also settled with key steps identified, as shown in Figure 3.3.

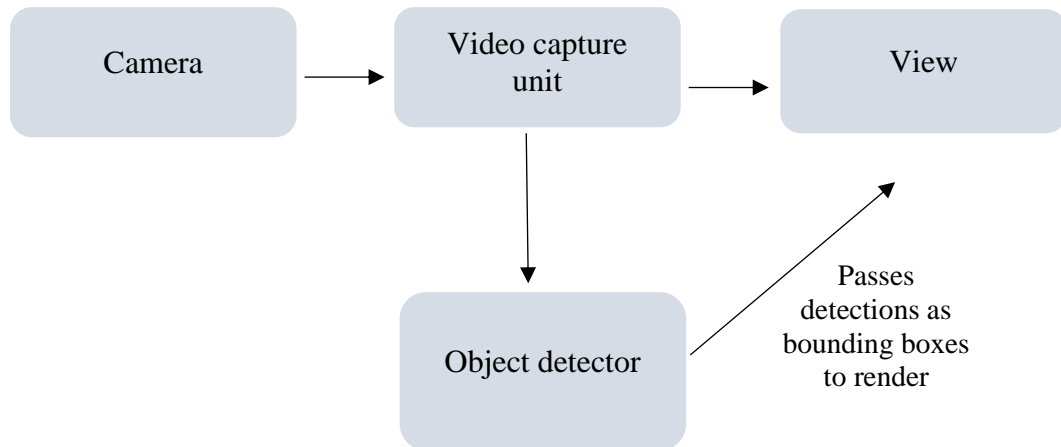


Figure 3.3: A process for Systems Development Research

Analyse and design the system.

The object detection system is designed. A video capture unit draws a camera feed on the screen and passes frames to the object detection model, which will output bounding boxes denoting the location of a door. These bounding boxes will then be rendered on screen.

Build a prototype system

A door detecting model is trained. The system is implemented based on the design, and the artifacts are integrated.

Observe and evaluate the system

The system is evaluated as to whether it can detect doors at all. If problems arise, relevant previous steps are re-iterated until a functioning prototype is created.

3.5 Machine learning process

3.5.1 Dataset

The ten classes of paddy insects are mainly collected from IP102 datasets, as shown in Figure 3.4. The IP102 datasets images were collected from different online sources such as Mendeley Data, Dave’s Garden, IPM Images, and other sources, including over 75 000 images of 102 species. Compared with previous datasets, the IP102 conforms to several characteristics of pest distribution in real environments.

The IP102 datasets have fifteen species mainly related to the Sri Lankan context, as shown in Table 3.1. The very harmful paddy pests are primarily collected through it (DOASL, 2021). Some selected classes have a small collection of data after removing the noisy images. In all, 1000 sub-images were produced.

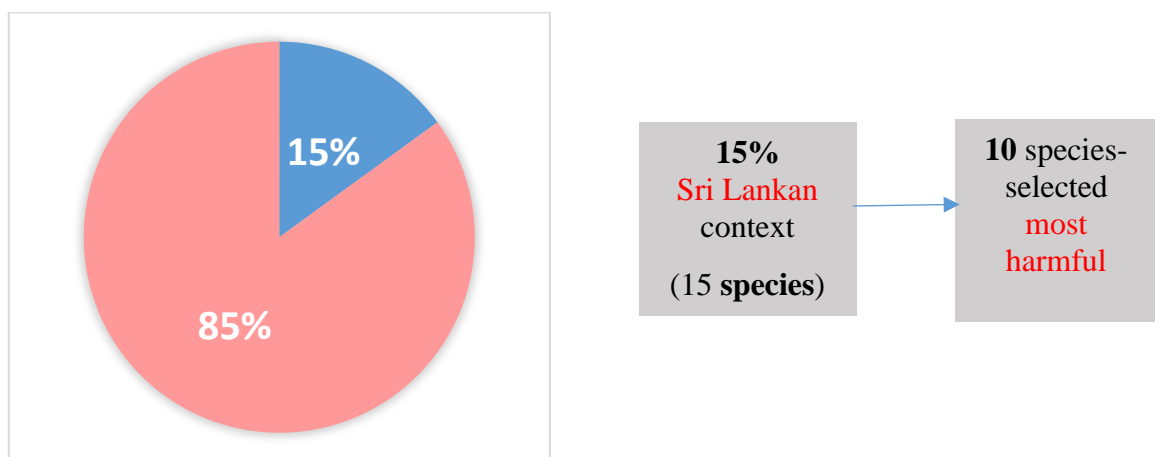


Figure 3.4 IP102 Data Set

Table 3.1 Original Data Set

Pest Name	Train Images Count	Test Images Count
Army worm	82	26
Legume blister beetle	74	20
Red spider	70	25
Rice water weevil	1283	88
Rice leafhopper	1250	110
Rice gall midge	58	20
Wheat phloeothrips	950	40
White-backed plant hopper	410	55
Yellow rice borer	41	15
Rice leaf roller	789	54

Data Preparation.

A few data control steps must be performed. To begin with, pictures and comments became tidied up. Not supportive pictures and comments are taken out. Second, information expansion procedures are utilized for the dataset. Minor adjustments to the dataset by producing changed pictures can work on model exactness. At last, changed comment documents and altered pictures became changed over to a chosen design.

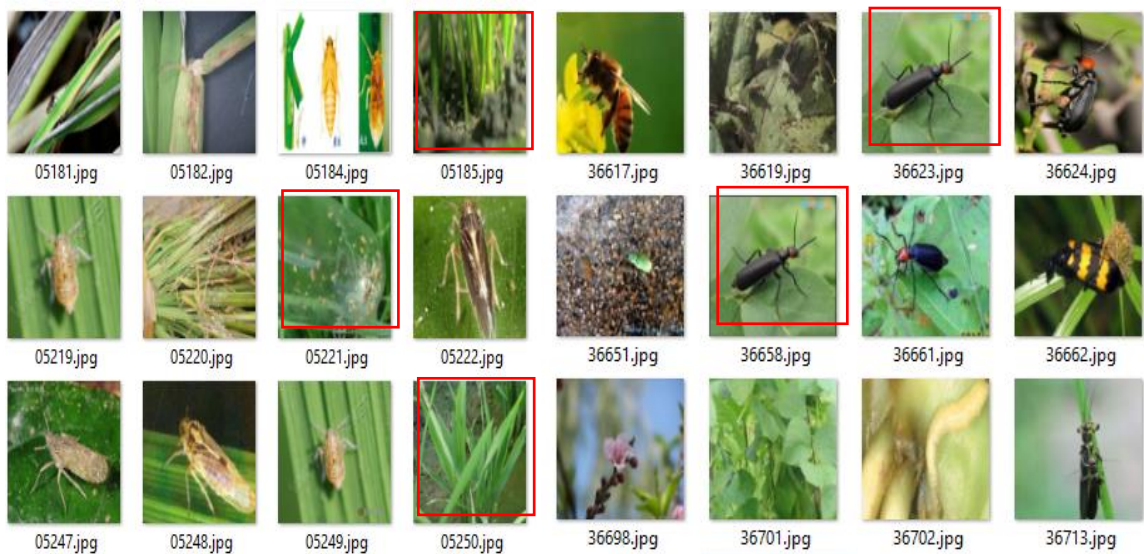


Figure 3.5: Noisy Data of IP102

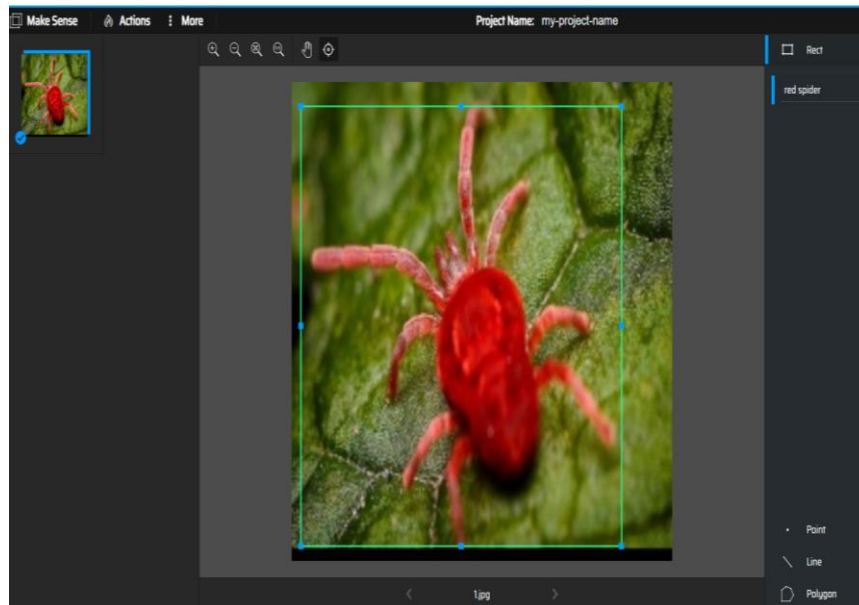


Figure 3.6: Image Annotation of seleted Pest

In that case, I Chose just one class, but the software can be used to annotate multiple classes. It's reaching to create an XML record per image (YOLO Darknet format) that contains all annotations and bounding boxes

```

<annotation>
  <folder>my-project-name</folder>
  <filename>10.jpg</filename>
  <path>/my-project-name/10.jpg</path>
  <source>
    <database>Unspecified</database>
  </source>
  <size>
    <width>224</width>
    <height>224</height>
    <depth>3</depth>
  </size>
  <object>
    <name>red spider </name>
    <pose>Unspecified</pose>
    <truncated>Unspecified</truncated>
    <difficult>Unspecified</difficult>
    <bndbox>
      <xmin>47</xmin>
      <ymin>45</ymin>
      <xmax>161</xmax>
      <ymax>166</ymax>
    </bndbox>
  </object>
</annotation>

```

Handling Class Imbalance

The dataset consists of imbalanced classes. The imbalance classes affect the accuracy of the classification model. The larger part class overwhelms the boundaries of the arrangement model. This diminishes the blunder of the larger part class at the beginning phase of emphases and builds the mistake of the minority class. The new class circulation was controlled by computing the normal number of pictures in the current datasets.

$$\begin{aligned}\text{Average images per class} &= \text{Total Images} / \text{Number of classes} \\ &= 1000/10 = 100\end{aligned}$$

3.1

Following procedures were applied to deal with the class irregularity issue.

1. Image Augmentation

Image Augmentation is a strategy to produce all the more new pictures with existing not many pictures. Expansion procedures like zoom, flat flip, vertical flip, revolution were applied to haphazardly choose pictures from minority classes and produce new pictures.

Keras ImageGenerator technique has been utilized to produce new pictures.

```
ImageDataGenerator(  
    width_shift_range=0.2,  
    height_shift_range=0.2,  
    zoom_range=0.2,  
    rotation_range=45,  
    horizontal_flip=True,  
    vertical_flip=True,)
```

The following examples show generated augmented images for Rice Gall Midge , as shown in Figure 3.7

Original Image



Width shift range



Height shift range



Horizontal flip



Vertical flip



Rotation



Zoom



Figure 3.7: Augmented Images

2. Random Under Sampling

Under-sampling was used to randomly remove images from majority classes until reaching balance distribution. The subtleties of the dataset after class adjusting are characterized.

3.5.2 Experiments

After the system was constructed, a set of hypotheses have been tested by experiments: a) data augmentation improves the accuracy of a model, b) transfer learning improves the accuracy of a model, and c) hyperparameter optimisation increases accuracy and or the speed of a model. Further experiments are carried out based on the knowledge and experiences gathered by previous experiments to the extent that time allows. Common false positives are detected and added to the dataset as negatives. The results will be discussed later in this paper.

3.5.3 Implementation

TensorFlow and Keras trained the models, and TensorFlow Lite converted the models to mobile support versions. Colab Pro Notebook GPU with 25 GB RAM runtime version was used as the development environment. Android is used for developing our application. The process is shown in Figure 3.8

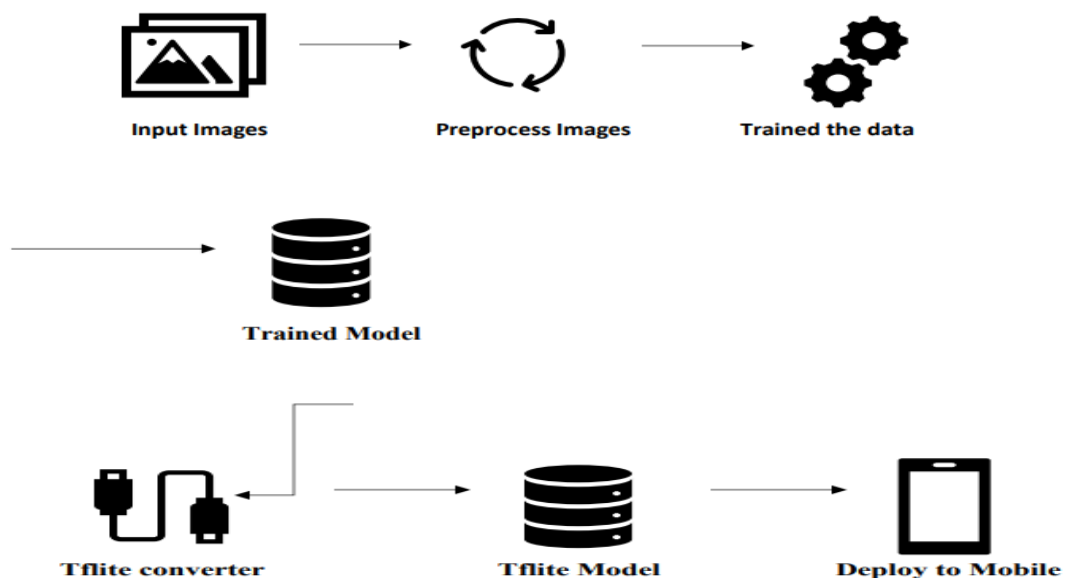


Figure 3.8: Training Model

3.5.3.1 Download and Explore the Dataset

```
import pathlib
```

```
dataset_url = "https://storage.googleapis.com/download.tensorflow.org/pests.tgz"
```

```
data_dir = tf.keras.utils.get_file('flower_photos', origin=dataset_url, untar=True)
```

```
data_dir = pathlib.Path(data_dir)
```

3.5.3.2 Configure from Custom Dataset

```
%cp -r /content/dataset/. /content/darknet/
#Set up training file directories for custom dataset
%cd /content/darknet/
%cp train/_darknet.labels data/obj.names
%mkdir data/obj
#copy image and labels
%cp train/*.jpg data/obj/
%cp valid/*.jpg data/obj/

%cp train/*.txt data/obj/
%cp valid/*.txt data/obj/

with open('data/obj.data', 'w') as out:
    out.write('classes = 3\n')
    out.write('train = data/train.txt\n')
    out.write('valid = data/valid.txt\n')
    out.write('names = data/obj.names\n')
    out.write('backup = backup/')

#write train file (just the image list)
import os

with open('data/train.txt', 'w') as out:
    for img in [f for f in os.listdir('train') if f.endswith('jpg')]:
        out.write('data/obj/' + img + '\n')

#write the valid file (just the image list)
import os

with open('data/valid.txt', 'w') as out:
    for img in [f for f in os.listdir('valid') if f.endswith('jpg')]:
        out.write('data/obj/' + img + '\n')
```

3.5.3.3 Training

```
#we build config dynamically based on number of classes
#we build iteratively from base config files. This is the same file shape as cfg/yolo-obj.cfg
def file_len(fname):
    with open(fname) as f:
        for i, l in enumerate(f):
            pass
        return i + 1

num_classes = file_len('train/_darknet.labels')
max_batches = num_classes*2000
steps1 = .8 * max_batches
steps2 = .9 * max_batches
steps_str = str(steps1)+' '+str(steps2)
num_filters = (num_classes + 5) * 3

#customize iPython writefile so we can write variables
from IPython.core.magic import register_line_cell_magic

@register_line_cell_magic
def writetemplate(line, cell):
    with open(line, 'w') as f:
        f.write(cell.format(**globals()))
```

```
writing config for a custom YOLOv4 detector detecting number of classes: 10
```

```
!./darknet detector train data/obj.data cfg/custom-yolov4-custom-detector.cfg yolov4-tiny.conv.29 -dont_show -map  
#If you get CUDA out of memory adjust subdivisions above!  
#adjust max batches down for shorter training above
```

```
Last accuracy mAP@0.5 = 82.33 %, best = 84.17 %  
19992: 0.054888, 0.035749 avg loss, 0.000026 rate, 0.428372 seconds, 1279488 images, 0.013120 hours left  
Loaded: 0.000055 seconds  
  
(next mAP calculation at 20000 iterations)  
Last accuracy mAP@0.5 = 82.33 %, best = 84.17 %  
19993: 0.045192, 0.036694 avg loss, 0.000026 rate, 0.435513 seconds, 1279552 images, 0.012998 hours left  
Loaded: 0.000042 seconds  
  
(next mAP calculation at 20000 iterations)  
Last accuracy mAP@0.5 = 82.33 %, best = 84.17 %  
19994: 0.033004, 0.036325 avg loss, 0.000026 rate, 0.436360 seconds, 1279616 images, 0.012877 hours left  
Loaded: 0.000044 seconds  
  
(next mAP calculation at 20000 iterations)  
Last accuracy mAP@0.5 = 82.33 %, best = 84.17 %  
19995: 0.023745, 0.035067 avg loss, 0.000026 rate, 0.430882 seconds, 1279680 images, 0.012755 hours left  
Loaded: 0.000041 seconds  
  
(next mAP calculation at 20000 iterations)  
Last accuracy mAP@0.5 = 82.33 %, best = 84.17 %  
19996: 0.032411, 0.034801 avg loss, 0.000026 rate, 0.428284 seconds, 1279744 images, 0.012634 hours left  
Loaded: 0.000048 seconds  
  
(next mAP calculation at 20000 iterations)  
Last accuracy mAP@0.5 = 82.33 %, best = 84.17 %  
19997: 0.048525, 0.036173 avg loss, 0.000026 rate, 0.430151 seconds, 1279808 images, 0.012512 hours left  
Loaded: 0.000047 seconds  
  
(next mAP calculation at 20000 iterations)  
Last accuracy mAP@0.5 = 82.33 %, best = 84.17 %  
19998: 0.022934, 0.034850 avg loss, 0.000026 rate, 0.425325 seconds, 1279872 images, 0.012391 hours left  
Loaded: 0.000043 seconds  
  
(next mAP calculation at 20000 iterations)  
Last accuracy mAP@0.5 = 82.33 %, best = 84.17 %  
19999: 0.051992, 0.036564 avg loss, 0.000026 rate, 0.430279 seconds, 1279936 images, 0.012269 hours left  
Loaded: 0.000050 seconds  
  
(next mAP calculation at 20000 iterations)  
Last accuracy mAP@0.5 = 82.33 %, best = 84.17 %  
20000: 0.024771, 0.035384 avg loss, 0.000026 rate, 0.420071 seconds, 1280000 images, 0.012147 hours left
```

3.5.3.4 Infer Custom Objects with Saved YOLOv4 Weights

```
#define utility function  
def imshow(path):  
    import cv2  
    import matplotlib.pyplot as plt  
    %matplotlib inline  
  
    image = cv2.imread(path)  
    height, width = image.shape[:2]  
    resized_image = cv2.resize(image, (3*width, 3*height), interpolation = cv2.INTER_CUBIC)  
  
    fig = plt.gcf()  
    fig.set_size_inches(18, 10)  
    plt.axis("off")  
    #plt.rcParams['figure.figsize'] = [10, 5]  
    plt.imshow(cv2.cvtColor(resized_image, cv2.COLOR_BGR2RGB))  
    plt.show()
```

3.5.3.5 Convert the weights to TensorFlow's pb representation

```

%cd /content/tensorflow-yolov4-tflite
# Regular TensorFlow SavedModel
!python save_model.py \
--weights /content/darknet/backup/custom-yolov4-custom-detector_final.weights \
--output ./checkpoints/yolov4-custom-416 \
--input_size 416 \
--model yolov4 \
--tiny \

# SavedModel to convert to TFLite
!python save_model.py \
--weights /content/darknet/backup/custom-yolov4-custom-detector_final.weights \
--output ./checkpoints/yolov4-custom-pretflite-416 \
--input_size 416 \
--model yolov4 \
--tiny \
--framework tflite

```

Following diagram describes trainable parameters.

tf_op_layer_concat_15 (TensorFl [(1, None, 4)])	0	tf_op_layer_concat_10[0][0] tf_op_layer_Reshape_11[0][0]
tf_op_layer_concat_16 (TensorFl [(1, None, 10)])	0	tf_op_layer_concat_7[0][0] tf_op_layer_concat_11[0][0]
=====		
Total params: 5,901,114		
Trainable params: 5,894,906		
Non-trainable params: 6,208		
=====		

3.5.3.6 Converting to tflite models

Subsequent to saving the prepared model, it ought to be changed over to a tflite model for supporting cell phones. TensorFlow Lite was accustomed to changing over the generally prepared model as tflite model.

```

%cd /content/tensorflow-yolov4-tflite
!python convert_tflite.py --weights ./checkpoints/yolov4-cosdom-pretflite-416 --output ./checkpoints/yolov4-cosdom-416.tflite

```

3.5.3.7 Save Model

```

# Choose what to copy
# TensorFlow SavedModel
!cp -r /content/tensorflow-yolov4-tflite/checkpoints/yolov4-cosdom-416/ "/content/drive/My Drive"
# TensorFlow Lite
!cp /content/tensorflow-yolov4-tflite/checkpoints/yolov4-cosdom-416.tflite "/content/drive/My Drive"

```

3.5.3.8 Load tflite models in Android

After adding TensorFlow Lite Interpreter to the Android project, the tflite model can be loaded using the below code snippet.

3.5.3.9 Train with Google Colab (Fine-tuning)

Yolov4

YOLO is a smart convolutional neural network (CNN) for doing object detection in real-time. The calculation does a solitary neural organization to the entire picture and afterward isolates the picture into areas and predicts bounding boxes and opportunities for every district. The anticipated probabilities weigh these bounding boxes.

YOLO is popular because it performs with great accuracy while also being able to run in real-time. The algorithm “only looks once” at the image in the knowledge that it needs only one forward propagation way through the neural network to make predictions. After non-max suppression (which makes sure the object detection algorithm only discovers each object once), it outputs identified objects concurrently with the bounding boxes. The conclusion is that YOLOv4 runs significantly faster than other detection methods with comparable performance (Ismail, 2021)). The main architecture is shown in Figure 3.9 (Ismail, 2021).

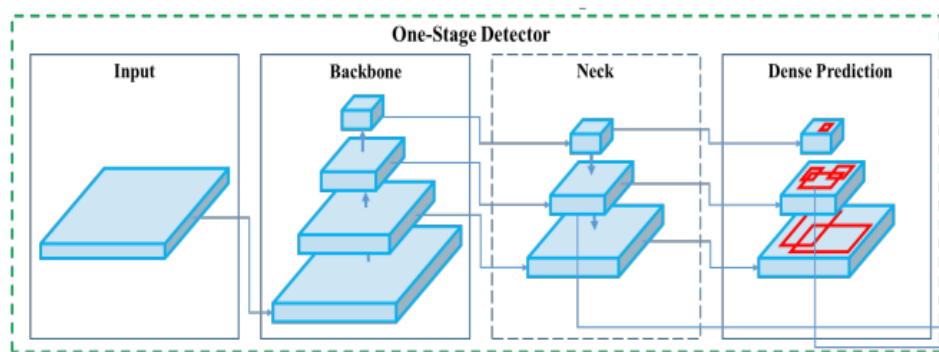


Figure 3.9: Yolov4

Yolov4 - custom
Input Parameter Tuning

1. Modify **batch** and **subdivisions**.

```
batch=64
subdivisions=16
```

3.2

2. Change **max_batches** YOLOv4 repeats learning for max_batches times, so learning is **not over if max_batches is too large**.

The value of max_batches seems to be a number of classes * 2000 [1].

The value of max_batches will be 2000 because the object to repeat learning more.

```
max_batches = 20000
```

3.3

3. Set steps to 80% and 90% of max_batches.

```
steps= 16000,18000
```

3.4

4. Modify classes and **filters**.

Calculate filters= (classes + 5) *number of masks.

```
num_filters = (num_classes + 5) * 3
```

3.5

```
num_filters = 45
```

3.5.4 Transfer learning

Huge computational power and time are needed to be trained a CNN from scratch. CNN gives accuracy when there is a huge amount of data. If there is a tiny amount of data at the training time, CNN would be overfitted and inaccurate results. Transfer learning is used to overcome these issues. In transfer learning, pre-prepared models can be utilized, which were prepared on an enormous dataset.

3.5.4.1 Tflite Models

TensorFlow Lite proselytes previously prepared models prepared on high-power machines to lightweight portable upheld tflite models. Profound learning models, as a rule, utilize 32-cycle drifting point information types with high accuracy. These models need high stockpiling, high CPU power, and high memory to run. Cell phones might not have these necessities to run a profound learning model. TensorFlow Lite can change over 32-digit drifting point models to 8-cycle number models. When

changing over to tflite model, there is a precision corruption when contrasted with the first model however accompanies different benefits; for example, CPU tasks are quicker for 8-bit whole numbers than 32-bit drifting point numbers, module size is diminished 4x while moving from 32-pieces to 8-pieces and lower cycle information implies more information can be squeezed into memory, thusly, no compelling reason to get to memory all the more frequently.

Drifting point numbers have type and mantissa. The type considers addressing an expansive scope of numbers, and mantissa gives the accuracy. While changing over 32-digit float 8-bit number type is supplanted by a proper scaling factor; numbers address the worth of a number comparative with this consistent.

3.5.4.2 Mobile Design

Following are the screens for capturing pest images and displaying results. In the in-Camera view, there are two main screens are shown in Figure 3.10

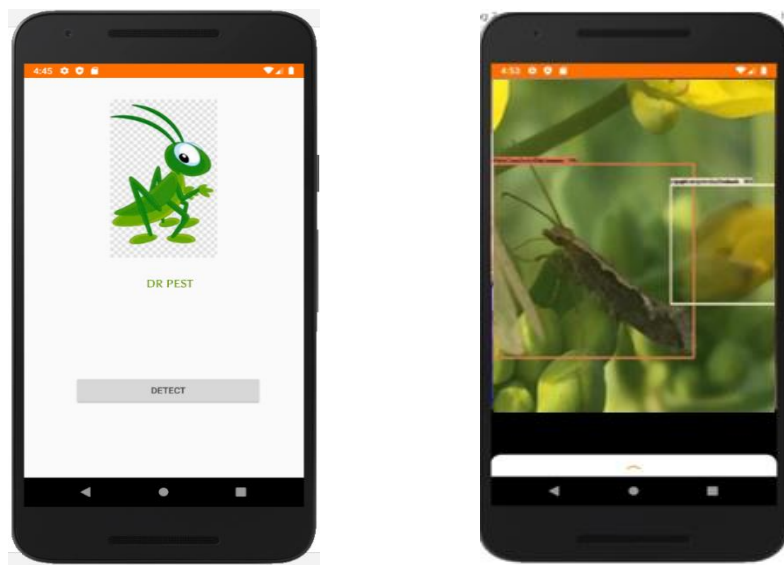


Figure 3.10: Camera view

3.5.4.3 Image Localisation Overview

This study proposes to design a system that farmers could use as a scale pest detector for the early prevention of pest damage. Image preprocessing of data augmentation and data annotation is performed before feeding the images into the object detection models, as shown

in Figure 3.11. The prediction object detection models are trained to identify the types and locations of the scale insects appearing in the image. Next, this paper describes each component of the proposed system in detail.

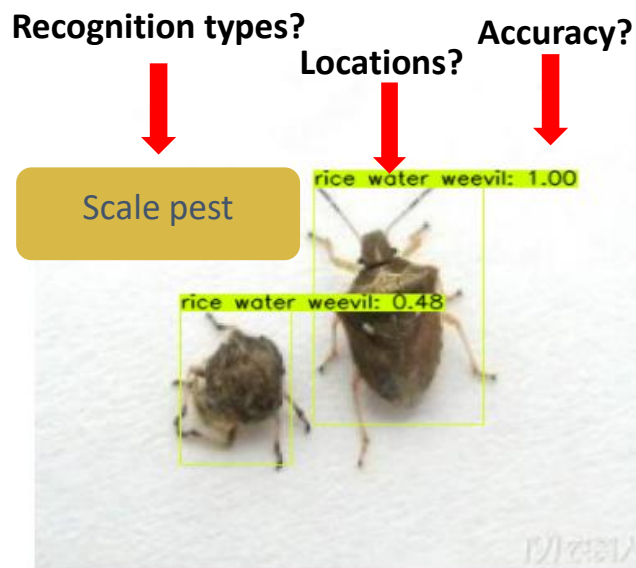


Figure 3.11: Image Localisation view

4. Evaluation and Results

This section shows the dataset used for our experiments, the training hyperparameters, and the outcomes. We consider standard evaluation criteria such as false positive (FP), false negative (FN), accuracy, and f-score. All our experiments were conveyed in Python using the TensorFlow library on an NVIDIA Tesla P100-PCIE.

4.1 Results

The accuracy of the CNNs ought to be estimated to distinguish appropriate CNN for the arrangement of vermin species. While networks were preparing, their approval exactness and preparing precision were recorded. Utilizing test datasets, an unmistakable thought of how prepared ML models act when they see new information can be obtained. ML models were prepared with various age sizes to improve results. Charts were created for each model to find out how model exactness acts while preparing.

The bug bother location calculation distinguishes bugs with complex foundations by applying division and observing the successful form in the insect picture. This calculation is utilized to bug datasets from IP102. The example identification perceptions of vermin are displayed in Figure 4.1. The difficult issue for the accomplishment of sound discovery yield relies upon the bug pictures with a muddled foundation in shadow and soil. The proposed bother discovery calculation sections the bug from the climate and chooses the form of the insect to accomplish a high location survey. Figure 4.1 shows that the irritation identification calculation performed well for a dataset with obliged and unconstrained stances of bug pictures and various foundations and directions of creepy crawlies. The vermin recognition calculation is handled with resized pictures to diminish the calculation time appropriate for constant location in horticulture areas. Kasinathan, 2020 got execution dependent on various dataset estimates and presumed that precision and computational time and utilized a lot of dataset from wang dataset, xie dataset and IP102 with calculation SVM, KNN, Naive Bayes, and the CNN model. The derivation time is thought of as the most noteworthy. (1 min – 5 min).

This research mainly focuses on fast real-time detection and small data sets with an average accuracy of **82.38** %.

Tests in hyperparameter optimisation suggest that there exists in YOLO a primary relationship between the size of the input layer of the models, speed, and mAP, each. Increasing the height and width of the input layer created a slower and slightly more accurate model. Decreasing it makes an increased rate except for significantly reduced accuracy, as shown in Table 4.1.

Table 4.1: Table illustrating the relationship between speed, mAP, and different configurations of model input layers.

Dataset composition	mAP	Detections per second
416x416 Input layer	82.38 %	1
224x224 Input layer	65.48%	14

4.1.1 Tflite Models Size

ML models were changed over to tflite portable supporting organization. The ML model size and Tflite model size were recorded in Table 4.2

Table 4.2:ML Model sizes

Model name	Model Size	Tflite Model Size
YOLOv4 custom	30.89 MB	22 MB

4.1.2 Tflite Models Accuracy

Tflite model precision was likewise noticed in the light of the fact that it is significant while choosing the reasonable model for versatility. The exactness of those models was assessed utilizing the test dataset. Tensorflow.lite.Interpreter.Run() technique was utilized to anticipate the outcomes in the portable application. The normal chance to anticipate the nuisance in the portable application was estimated on the grounds that it is significant while

choosing the reasonable model for versatility. Subsequently, the normal execution time to expect the irritation was recorded utilizing the test dataset, as displayed in Table 4.3.

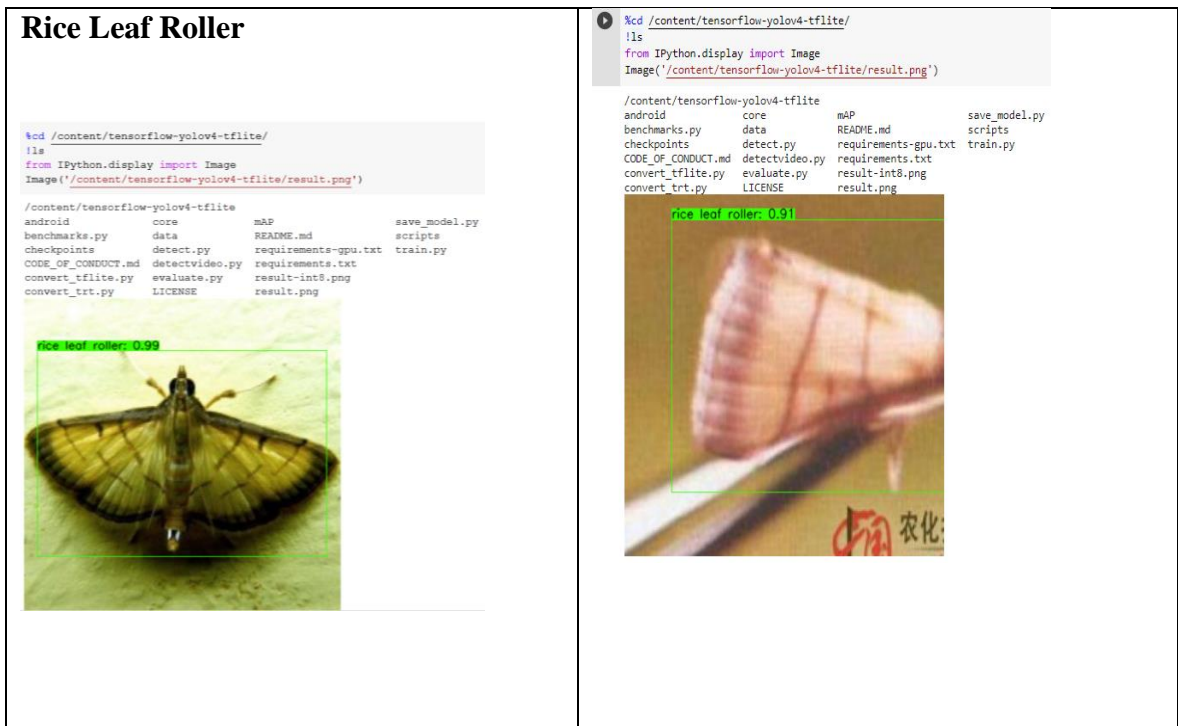
Table 4.3: Tflite Model Accuracy

Model name	Accuracy %	Prediction Time(sec)
YOLOv4 custom	82.38	1

4.1.3 Testing Flow

```
# Verify
%cd /content/tensorflow-yolov4-tflite
!python detect.py --weights ./checkpoints/yolov4-custom --size 416 --model yolov4 \
--image /content/darknet/test/98_jpg.rf.1eaab0ea1bae3dccefc046aa5e092f3c.jpg \
# --framework tflite
```

Figure 4.1 shows the test results of the system is given



Yellow Rice Border

```
[6] %cd /content/tensorflow-yolov4-tflite/
!ls
from IPython.display import Image
Image('/content/tensorflow-yolov4-tflite/result.png')
```

```
/content/tensorflow-yolov4-tflite
android          core          mAP          save_model.py
benchmarks.py   data          README.md    scripts
checkpoints     detect.py     requirements-gpu.txt  train.py
CODE_OF_CONDUCT.md  detectvideo.py  requirements.txt
convert_tflite.py  evaluate.py    result-int8.png
convert_trt.py    LICENSE       result.png
```



Rice Water Weevil

```
%cd /content/tensorflow-yolov4-tflite/
!ls
from IPython.display import Image
Image('/content/tensorflow-yolov4-tflite/result.png')
```

```
/content/tensorflow-yolov4-tflite
android          core          mAP          save_model.py
benchmarks.py   data          README.md    scripts
checkpoints     detect.py     requirements-gpu.txt  train.py
CODE_OF_CONDUCT.md  detectvideo.py  requirements.txt
convert_tflite.py  evaluate.py    result-int8.png
convert_trt.py    LICENSE       result.png
```



```
%cd /content/tensorflow-yolov4-tflite/
!ls
from IPython.display import Image
Image('/content/tensorflow-yolov4-tflite/result.png')
```

```
/content/tensorflow-yolov4-tflite
android          core          mAP          save_model.py
benchmarks.py   data          README.md    scripts
checkpoints     detect.py     requirements-gpu.txt  train.py
CODE_OF_CONDUCT.md  detectvideo.py  requirements.txt
convert_tflite.py  evaluate.py    result-int8.png
convert_trt.py    LICENSE       result.png
```



Wheat Phloeothrips

```
%cd /content/tensorflow-yolov4-tflite/
!ls
from IPython.display import Image
Image('/content/tensorflow-yolov4-tflite/result.png')
```

```
/content/tensorflow-yolov4-tflite
android          core          mAP          save_model.py
benchmarks.py   data          README.md    scripts
checkpoints     detect.py     requirements-gpu.txt  train.py
CODE_OF_CONDUCT.md  detectvideo.py  requirements.txt
convert_tflite.py  evaluate.py    result-int8.png
convert_trt.py    LICENSE       result.png
```



```
%cd /content/tensorflow-yolov4-tflite/
!ls
from IPython.display import Image
Image('/content/tensorflow-yolov4-tflite/result.png')
```

```
/content/tensorflow-yolov4-tflite
android          core          mAP          save_model.py
benchmarks.py   data          README.md    scripts
checkpoints     detect.py     requirements-gpu.txt  train.py
CODE_OF_CONDUCT.md  detectvideo.py  requirements.txt
convert_tflite.py  evaluate.py    result-int8.png
convert_trt.py    LICENSE       result.png
```




Rice Leafhopper

```

%cd /content/tensorflow-yolov4-tflite/
!ls
from IPython.display import Image
Image('/content/tensorflow-yolov4-tflite/result.png')
    
```

```

/content/tensorflow-yolov4-tflite
android          core             mAP             save_model.py
benchmarks.py   data             README.md       scripts
checkpoints     detect.py        requirements-gpu.txt  train.py
CODE_OF_CONDUCT.md  detectvideo.py  requirements.txt
convert_tflite.py  evaluate.py      result-int8.png
convert_trt.py    LICENSE         result.png
    
```



Rice Gall Midge

```

%cd /content/tensorflow-yolov4-tflite/
!ls
from IPython.display import Image
Image('/content/tensorflow-yolov4-tflite/result.png')
    
```

```

/content/tensorflow-yolov4-tflite
android          core             mAP             save_model.py
benchmarks.py   data             README.md       scripts
checkpoints     detect.py        requirements-gpu.txt  train.py
CODE_OF_CONDUCT.md  detectvideo.py  requirements.txt
convert_tflite.py  evaluate.py      result-int8.png
convert_trt.py    LICENSE         result.png
    
```




White Backed Plant Hopper

```

%cd /content/tensorflow-yolov4-tflite/
!ls
from IPython.display import Image
Image('/content/tensorflow-yolov4-tflite/result.png')
    
```

```

/content/tensorflow-yolov4-tflite
android          core             mAP             save_model.py
benchmarks.py   data             README.md       scripts
checkpoints     detect.py        requirements-gpu.txt  train.py
CODE_OF_CONDUCT.md  detectvideo.py  requirements.txt
convert_tflite.py  evaluate.py      result-int8.png
convert_trt.py    LICENSE         result.png
    
```




```

%cd /content/tensorflow-yolov4-tflite/
!ls
from IPython.display import Image
Image('/content/tensorflow-yolov4-tflite/result.png')
    
```

```

/content/tensorflow-yolov4-tflite
android          core             mAP             save_model.py
benchmarks.py   data             README.md       scripts
checkpoints     detect.py        requirements-gpu.txt  train.py
CODE_OF_CONDUCT.md  detectvideo.py  requirements.txt
convert_tflite.py  evaluate.py      result-int8.png
convert_trt.py    LICENSE         result.png
    
```




```

%cd /content/tensorflow-yolov4-tflite/
!ls
from IPython.display import Image
Image('/content/tensorflow-yolov4-tflite/result.png')
    
```

```

/content/tensorflow-yolov4-tflite
android          core             mAP             save_model.py
benchmarks.py   data             README.md       scripts
checkpoints     detect.py        requirements-gpu.txt  train.py
CODE_OF_CONDUCT.md  detectvideo.py  requirements.txt
convert_tflite.py  evaluate.py      result-int8.png
convert_trt.py    LICENSE         result.png
    
```



Red Spider

```
%cd /content/tensorflow-yolov4-tflite/
!ls
from IPython.display import Image
Image('/content/tensorflow-yolov4-tflite/result.png')
```

```
/content/tensorflow-yolov4-tflite
android      core          mAP          save_model.py
benchmarks.py data          README.md    scripts
checkpoints detect.py    requirements-gpu.txt  train.py
CODE_OF_CONDUCT.md detectvideo.py requirements.txt
convert_tflite.py evaluate.py   result-int8.png
convert_trt.py LICENSE      result.png
```



```
%cd /content/tensorflow-yolov4-tflite/
!ls
from IPython.display import Image
Image('/content/tensorflow-yolov4-tflite/result.png')
```

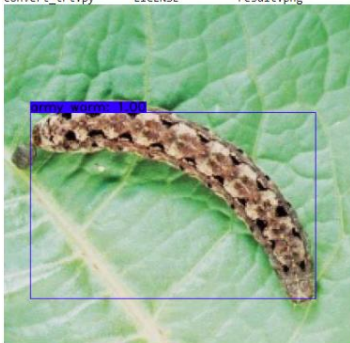
```
/content/tensorflow-yolov4-tflite
android      core          mAP          save_model.py
benchmarks.py data          README.md    scripts
checkpoints detect.py    requirements-gpu.txt  train.py
CODE_OF_CONDUCT.md detectvideo.py requirements.txt
convert_tflite.py evaluate.py   result-int8.png
convert_trt.py LICENSE      result.png
```



Army Worm

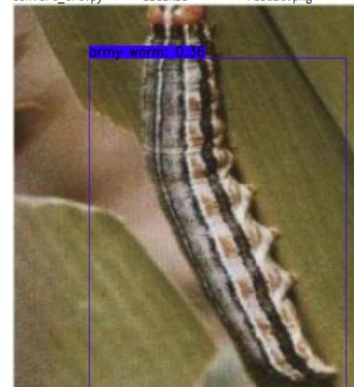
```
%cd /content/tensorflow-yolov4-tflite/
!ls
from IPython.display import Image
Image('/content/tensorflow-yolov4-tflite/result.png')
```

```
/content/tensorflow-yolov4-tflite
android      core          mAP          save_model.py
benchmarks.py data          README.md    scripts
checkpoints detect.py    requirements-gpu.txt  train.py
CODE_OF_CONDUCT.md detectvideo.py requirements.txt
convert_tflite.py evaluate.py   result-int8.png
convert_trt.py LICENSE      result.png
```



```
%cd /content/tensorflow-yolov4-tflite/
!ls
from IPython.display import Image
Image('/content/tensorflow-yolov4-tflite/result.png')
```

```
/content/tensorflow-yolov4-tflite
android      core          mAP          save_model.py
benchmarks.py data          README.md    scripts
checkpoints detect.py    requirements-gpu.txt  train.py
CODE_OF_CONDUCT.md detectvideo.py requirements.txt
convert_tflite.py evaluate.py   result-int8.png
convert_trt.py LICENSE      result.png
```



Legume Blister Beetle

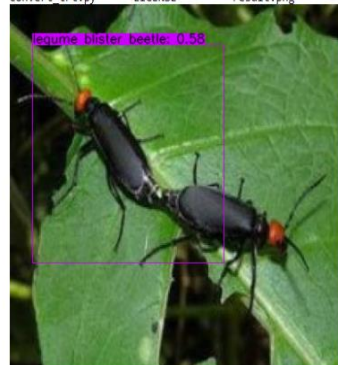
```
%cd /content/tensorflow-yolov4-tflite/
!ls
from IPython.display import Image
Image('/content/tensorflow-yolov4-tflite/result.png')
```

```
/content/tensorflow-yolov4-tflite
android      core          mAP          save_model.py
benchmarks.py data          README.md    scripts
checkpoints detect.py    requirements-gpu.txt  train.py
CODE_OF_CONDUCT.md detectvideo.py requirements.txt
convert_tflite.py evaluate.py   result-int8.png
convert_trt.py LICENSE      result.png
```



```
%cd /content/tensorflow-yolov4-tflite/
!ls
from IPython.display import Image
Image('/content/tensorflow-yolov4-tflite/result.png')
```

```
/content/tensorflow-yolov4-tflite
android      core          mAP          save_model.py
benchmarks.py data          README.md    scripts
checkpoints detect.py    requirements-gpu.txt  train.py
CODE_OF_CONDUCT.md detectvideo.py requirements.txt
convert_tflite.py evaluate.py   result-int8.png
convert_trt.py LICENSE      result.png
```



4.2 Evaluation

Evaluating the object detection model is not straightforward because each image can have many objects, and each object can belong to a different class. This means that we need to measure whether the model has found all the objects and a way to check if the found objects belong to the correct class.

1. Discover the objects in the image.
2. Verify if the found objects relating to the accurate class

When annotating objects, the bounding box is the box's location to indicate, as shown in Figure 4.2 (Kasinathan, 2020).

This means that the object detection model must complete two things

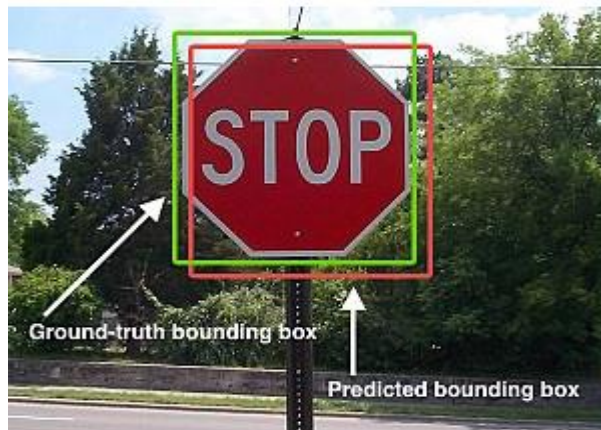


Figure 4.2: Image Bounding Box

4.2.1 Intersection over Union (IoU)

The object detection model produces three components of output:

1. Bounding box: x1, y1, width, height (XML file format is used)
2. Bounding box class
3. Probability score of the prediction: the authenticity of the model is this class predicted class

To evaluate whether an object was found, the junction's point of intersection (IoU) is used to measure similarity. It is given by the overlap area divided by the junction size of the two bounding boxes (Kasinathan, 2020).

$$IOU = \frac{\text{area of overlap}}{\text{area of union}} = \frac{\text{img alt="Diagram showing two overlapping bounding boxes (green and red) with their intersection shaded blue." data-bbox="585 180 675 245}}{\text{img alt="Diagram showing the union of two overlapping bounding boxes (green and red) shaded blue." data-bbox="585 255 675 316"/>$$

4.1



$$IoU = \frac{\text{area (true bounding box} \cap \text{predicted bounding box)}}{\text{area (true bounding box} \cup \text{predicted bounding box)}} > 0.5.$$

4.2

The IoU is needed to determine the Average Precision because the first step is to determine if two bounding boxes are pointing to the same object. The default IoU value is used 0.5. The result of the average IOU of the model is 56.75 %.

4.2.2 Confusion Matrix

The confusion matrix gives more insights into the performance of the classification algorithm. The matrix contrasts the genuine qualities and anticipated qualities by the arrangement calculation. It is created utilizing the test information. When there are various classes, ascertaining precision just deceives the presentation of the characterization calculation. For instance, when calculation gives a normal precision of 82.38% for ten classes. It is difficult to identify that all classes gave 82.38% accuracy. It can be six classes with an accuracy of more than 85%, three classes have an accuracy of 72-79 %, and one

class has an accuracy of 69%. It is essential to have better accuracy for all classes in pest classification.

Sample confusion matrix for binary classification.

		Predicted	
		Negative	Positive
Actual	Negative	True Positive	False Positive
	Positive	False Negative	True Negative

True Positive (TP) - The predicted value by the algorithm is positive, and It is true. For example, the algorithm predicted the pest image as Rice Leaf Roller, and It is correct.

True Negative (TN) - The predicted value by the algorithm is negative, and It is true. For example, the algorithm predicted the pest image is not Rice Leaf Roller, correct.

False Positive (FP) - The predicted value by the algorithm is positive, but it is negative. For example, the algorithm predicted the pest image as Rice Leaf Roller, and It is incorrect.

False Negative (FN) - The predicted value by the algorithm is negative, but it is positive. For example, the algorithm predicted the pest image as not Rice Leaf Roller, but Rice Leaf Roller.

Following matrices can be determined utilizing Confusion Matrix

Precision

Precision works out the number of the anticipated qualities by the model is positiverix

$$Precision = \frac{TP}{TP+FP} \quad 4.3$$

Recall

Recall works out the number of the genuine positive qualities that anticipated by the model

$$Recall = \frac{TP}{TP+FN} \quad 4.4$$

F1-Score

F1-score gives an insight into the balance between Precision and Recall

$$F1 = \frac{2*(Precision*Recall)}{Precision + Recall}$$

Confusion Matrix was produced for all models utilizing the test dataset. Classification report was also developed with Precision, Recall and, F1-Score values.

In multi-class classification, the diagonal of the confusion matrix shows the correct predictions for each class.

Final Classification Report YOLOV4

```

calculation mAP (mean average precision)...
200
detections_count = 482, unique_truth_count = 200
class_id = 0, name = army worm, ap = 78.01%      (TP = 14, FP = 3)
class_id = 1, name = legume blister beetle, ap = 73.07%      (TP = 13, FP = 4)
class_id = 2, name = red spider, ap = 81.78%      (TP = 12, FP = 6)
class_id = 3, name = rice gall midge, ap = 86.48%      (TP = 21, FP = 9)
class_id = 4, name = rice leaf roller, ap = 92.96%      (TP = 16, FP = 2)
class_id = 5, name = rice leafhopper, ap = 72.32%      (TP = 15, FP = 3)
class_id = 6, name = rice water weevil, ap = 89.78%      (TP = 17, FP = 7)
class_id = 7, name = wheat phloeothrips, ap = 70.95%      (TP = 12, FP = 8)
class_id = 8, name = white backed plant hopper, ap = 86.70%      (TP = 15, FP = 12)
class_id = 9, name = yellow rice borer, ap = 81.71%      (TP = 18, FP = 6)

for conf_thresh = 0.25, precision = 0.72, recall = 0.76, F1-score = 0.74
for conf_thresh = 0.25, TP = 153, FP = 60, FN = 47, average IoU = 56.99 %

IoU threshold = 50 %, used Area-Under-Curve for each unique Recall
mean average precision (mAP@0.50) = 0.813755, or 81.38 %
Total Detection Time: 1 Seconds

Set -points flag:
`-points 101` for MS COCO
`-points 11` for PascalVOC 2007 (uncomment `difficult` in voc.data)
`-points 0` (AUC) for ImageNet, PascalVOC 2010-2012, your custom dataset

```

Final Classification Report YOLOV4 shows in Table 4.4

Table 4.4 Yolov4 Model Accuracy

	Accuracy (%)	TP	FP
army worm	78.01	14	3
legume blister beetle	73.07	13	4
red spider	81.78	12	6
rice gall midge	86.48	21	9
rice leaf roller	92.96	16	2
rice leafhopper	72.32	15	3
rice water weevil	89.78	17	7
wheat phloeothrips	70.95	12	8
white backed plant hopper	86.70	15	12
yellow rice borer	81.71	18	6

Final Classification Report YOLOV4 – CUSTOME

```

 calculation mAP (mean average precision)...
 200
 detections_count = 437, unique_truth_count = 199
 class_id = 0, name = army worm, ap = 78.06%      (TP = 11, FP = 5)
 class_id = 1, name = legume blister beetle, ap = 86.83%      (TP = 19, FP = 4)
 class_id = 2, name = red spider, ap = 85.42%      (TP = 16, FP = 2)
 class_id = 3, name = rice gall midge, ap = 91.87%      (TP = 18, FP = 2)
 class_id = 4, name = rice leaf roller, ap = 72.12%      (TP = 16, FP = 7)
 class_id = 5, name = rice leafhopper, ap = 69.37%      (TP = 16, FP = 6)
 class_id = 6, name = rice water weevil, ap = 89.27%      (TP = 16, FP = 10)
 class_id = 7, name = wheat phloeothrips, ap = 88.69%      (TP = 16, FP = 5)
 class_id = 8, name = white backed plant hopper, ap = 87.97%      (TP = 17, FP = 5)
 class_id = 9, name = yellow rice borer, ap = 74.23%      (TP = 17, FP = 6)

 for conf_thresh = 0.25, precision = 0.76, recall = 0.81, F1-score = 0.78
 for conf_thresh = 0.25, TP = 162, FP = 52, FN = 37, average IoU = 59.75 %

 IoU threshold = 50 %, used Area-Under-Curve for each unique Recall
 mean average precision (mAP@0.50) = 0.823837, or 82.38 %
 Total Detection Time: 1 Seconds

 Set -points flag:
  `-.points 101` for MS COCO
  `-.points 11` for PascalVOC 2007 (uncomment `difficult` in voc.data)
  `-.points 0` (AUC) for ImageNet, PascalVOC 2010-2012, your custom dataset
  
```

Final Classification Report YOLOV4 shows in Table 4.4

Table 4.5: Yolov4-Custom Model Accuracy

	Accuracy (%)	TP	FP
army worm	78.06	11	5
legume blister beetle	86.83	19	4
red spider	85.42	16	2
rice gall midge	91.87	18	2
rice leaf roller	72.12	16	7
rice leafhopper	69.37	16	6
rice water weevil	89.27	16	10
wheat phloeothrips	88.69	16	5
white backed plant hopper	87.97	17	5
yellow rice borer	74.23	17	6

Model Average Accuracy shows in Table 4.6

Table 4.6: Model Average Accuracy

	Average Rate(yolov4)	Average Rate (customer-yolov4)
precision	0.72	0.76
recall	0.76	0.81
f1-score	0.74	0.78
support	200	200
iou	0.5	0.5
accuracy	0.8138	0.823837

Figure 4.3 shows the Training accuracy of YOLOV4 and YOLOV4 – custom

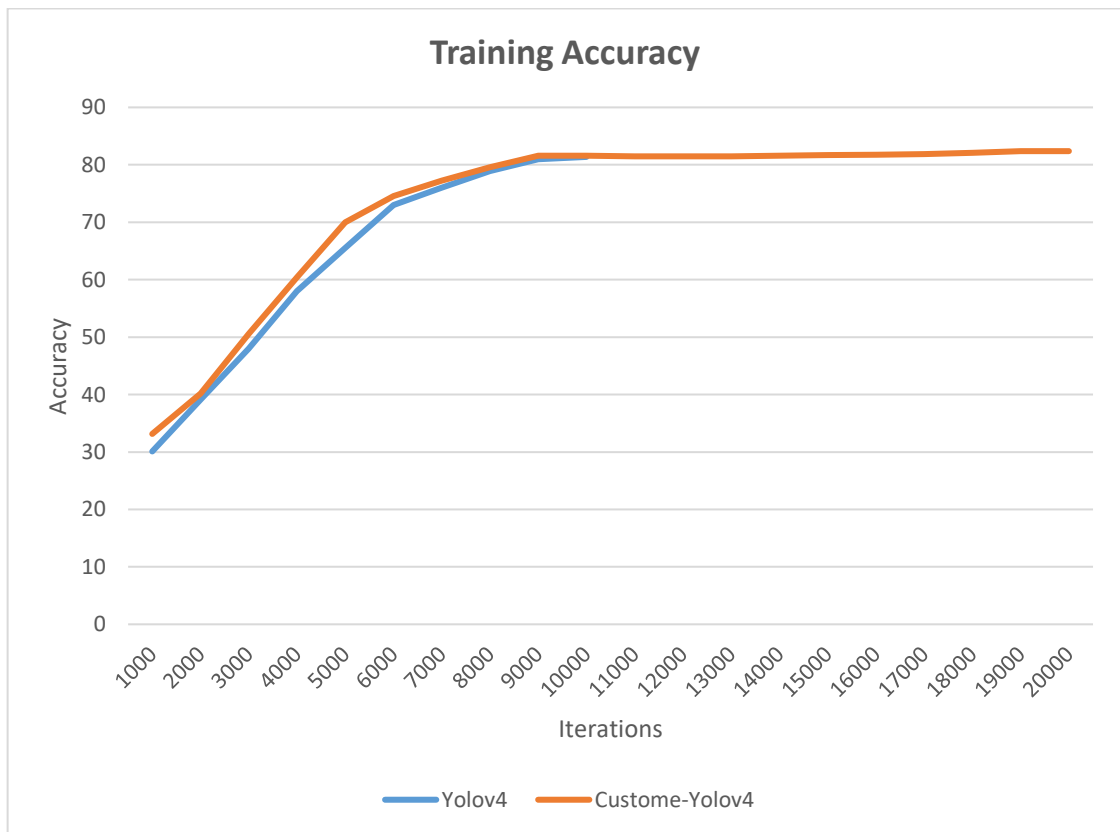


Figure 4.3: Training accuracy

It is necessary to verify that the weight looks good and that the performance is likely to be sufficient while looking at the log file. After 20000 iterations, accuracy is not changed. Fine-tune model gives good accuracy and performance comparing the previous model.

In this segment, the finetuned model is contrasted and the cutting edge Yolov4 model proposed. In that exploration, and utilized the dataset that contains 700 pictures in the preparation set and 300 pictures in the testing set. The finetuned yoloy4 accomplished a high exactness of 82.38%. In this paper, the finetuned Yolov4 model is contrasted and the finetuned yolov4 model. The trial brings about Table 4.6 shows that the finetuned Yolov4 model performs better precision and preparing time.

5. Conclusion and Future Work

This paddy pest dataset covers the ten typical paddy pests and fills in the blank of the open dataset of paddy pests. The application of standard object detection algorithms in paddy pest detection is compared and analysed. Among them, YOLO adjusts the exhibition markers like location exactness, time, and memory utilization and can be applied to cell phones. Given the model, the DA strategy is chosen to tackle the lopsided class balance issue in the trial dataset. The trial results showed that the strategy could further develop the recognition exactness by 1.0 rate focuses. To avoid data overfitting, the dropout layer is added. When the mapped value of the model was as high as 82.38%, it was tested on images collected under different illuminations and backgrounds. The model has an excellent ability to detect paddy pests under complex noise. Eventually, the trained model was applied to the Android platform to develop a paddy pest detection system. It is also the first application of paddy pest detection based on deep learning in Sri Lanka. Users can accomplish detecting the pests of paddy in the field in real-time by operating the intuitive and straightforward interface based on an Android mobile phone, which effectively improves the efficiency of agricultural production.

This system could be extended to examine as very small pests' detection propose the algorithm counts pests on leaves and further give the information as the input for this future implementation because this approach detects the pest identification and the location.

In a future study, encountering multiobject overlap, feature occlusion, and small object detection tasks will be the next research direction.

References

- A. Godil, R. Bostelman, W. Shackleford, T. Hong, and M. Shneier, "Performance metrics for evaluating object and human detection and tracking systems," National Institute of Standards and Technology, Gaithersburg, MD, USA, 2014, NISTIR 7972.
- Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., Kudlur, M., Levenberg, J., Monga, R., Moore, S., Murray, D.G., Steiner, B., Tucker, P., Vasudevan, V., Warden, P., Wicke, M., Yu, Y., Zheng, X., 2016. TensorFlow: A system for large-scale machine learning 21.
- Adhikarinayake, T.B., 2005. Methodical design process to improve income of paddy farmers in Sri Lanka. [publisher not identified], Wageningen.
- Altop, D.K.; Levi, A.; Tuzcu, V. Feature-level fusion of physiological parameters as cryptographic keys. In Proceedings of the 2017 IEEE International Conference on Communications (ICC), Paris, France, 21–25 May 2017
- Azlah, M., Chua, L.S., Rahmad, F., Abdullah, F., Wan Alwi, S.R., 2019. Review on Techniques for Plant Leaf Classification and Recognition. *Computers* 8, 77. <https://doi.org/10.3390/computers8040077>
- Bechar, I., Moisan, S., 2010. Online counting of pests in a greenhouse using computer vision. Presented at the VAIB 2010 - Visual Observation and Analysis of Animal and Insect Behavior.
- Bell, S.; Zitnick, C.L.; Bala, K.; Girshick, R. Inside-outside net: Detecting objects in context with skip pooling and recurrent neural networks. arXiv 2015, arXiv:1512.04143.
- Brownlee, J., 2019. A Gentle Introduction to Computer Vision. *Mach. Learn. Mastery*. URL <https://machinelearningmastery.com/what-is-computer-vision/> (accessed 9.20.21).
- D.M.J.B.Senanayake, 2021. Rice Research and Development Institute Batalagoda [WWW Document]. URL <https://www.doa.gov.lk/rrdi/index.php/en/> (accessed 9.20.21).

- Dai, J., Li, Y., He, K., Sun, J., 2016. R-FCN: Object Detection via Region-based Fully Convolutional Networks. arXiv:1605.06409 [cs].
- Dai, Q., Cheng, X., Qiao, Y., Zhang, Y., 2020. Agricultural Pest Super-Resolution and Identification With Attention Enhanced Residual and Dense Fusion Generative and Adversarial Network. IEEE Access 8, 81943–81959. <https://doi.org/10.1109/ACCESS.2020.2991552>
- Ding, S., Zhao, K., 2018. Research on Daily Objects Detection Based on Deep Neural Network. IOP Conf. Ser.: Mater. Sci. Eng. 322, 062024. <https://doi.org/10.1088/1757-899X/322/6/062024>
- Estruch, J., Carozzi, N., Desai, N., Duck, N., Warren, G., Koziel, M., 1997. Transgenic plants: An emerging approach to pest control. Nature biotechnology 15, 137–41. <https://doi.org/10.1109/ACCESS.2019.2943454>
- Everingham, M., Gool, L.V., Williams, C.K.I., Winn, J., Zisserman, A., 2010. The Pascal Visual Object Classes (VOC) Challenge. Int J Comput Vis 88, 303–338. <https://doi.org/10.1007/s11263-009-0275-4>
- Fuentes, A., Im, D.H., Yoon, S., Park, D.S., 2017. Spectral Analysis of CNN for Tomato Disease Identification, in Rutkowski, L., Korytkowski, M., Scherer, R., Tadeusiewicz, R., Zadeh, L.A., Zurada, J.M. (Eds.), Artificial Intelligence and Soft Computing, Lecture Notes in Computer Science. Springer International Publishing, Cham, pp. 40–51. https://doi.org/10.1007/978-3-319-59063-9_4
- Godil, A., Bostelman, R., Shackleford, W., Hong, T., Shneier, M., 2014. Performance Metrics for Evaluating Object and Human Detection and Tracking Systems (No. NIST IR 7972). National Institute of Standards and Technology. <https://doi.org/10.6028/NIST.IR.7972>
- He, K., Zhang, X., Ren, S., Sun, J., 2016. Deep Residual Learning for Image Recognition [WWW Document]. undefined. URL /paper/Deep-Residual-Learning-for-Image-

- Recognition-He-Zhang/2c03df8b48bf3fa39054345bafabfeff15bfd11d (accessed 2.6.21).
- Home [WWW Document], n.d. URL <https://doa.gov.lk/rrdi/index.php/en/> (accessed 5.29.21).
- Huang, J., Rathod, V., Sun, C., Zhu, M., Korattikara, A., Fathi, A., Fischer, I., Wojna, Z., Song, Y., Guadarrama, S., Murphy, K., 2017. Speed/accuracy trade-offs for modern convolutional object detectors. arXiv:1611.10012 [cs].
- Hui, J., 2020. SSD object detection: Single Shot MultiBox Detector for real-time processing [WWW Document]. Medium. URL <https://jonathan-hui.medium.com/ssd-object-detection-single-shot-multibox-detector-for-real-time-processing-9bd8deac0e06> (accessed 5.29.21).
- Hwang, S., Kim, H.-E., 2016. Self-Transfer Learning for Weakly Supervised Lesion Localisation, in Ourselin, S., Joskowicz, L., Sabuncu, M.R., Unal, G., Wells, W. (Eds.), Medical Image Computing and Computer-Assisted Intervention – MICCAI 2016, Lecture Notes in Computer Science. Springer International Publishing, Cham, pp. 239–246. https://doi.org/10.1007/978-3-319-46723-8_28
- I. Bechar and S. Moisan, “Online counting of pests in a greenhouse using computer vision,” in Proceedings of the VAIB 2010-Visual Observation and Analysis of Animal and Insect Behavior, Istanbul, Turkey, August 2010.
- Introduction to Object Detection Algorithms, 2018. . Analytics Vidhya. URL <https://www.analyticsvidhya.com/blog/2018/10/a-step-by-step-introduction-to-the-basic-object-detection-algorithms-part-1/> (accessed 5.26.21).
- Jiang, P., Chen, Y., Liu, B., He, D., Liang, C., 2019. Real-Time Detection of Apple Leaf Diseases Using Deep Learning Approach Based on Improved Convolutional Neural Networks. IEEE Access 7, 59069–59080. <https://doi.org/10.1109/ACCESS.2019.2914929>

- Jiang, X., Wang, C., Fu, Q., 2018. Development and application of deep convolutional neural network in target detection. AIP Conference Proceedings 1955, 040036. <https://doi.org/10.1063/1.5033700>
- Kanesh, V., Ramanan, A., 2014. Image Classification of Paddy Field Insect Pests Using Gradient-Based Features. Int. J. Mach. Learn. Comput. 1–5. <https://doi.org/10.7763/IJMLC.2014.V4.376>
- Kasinathan, T., Singaraju, D., Reddy, U.S., 2020. Insect classification and detection in field crops using modern machine learning techniques—information Processing in Agriculture. <https://doi.org/10.1016/j.inpa.2020.09.006>
- King, A., 2017. Technology: The Future of Agriculture. Nature 544, S21–S23. <https://doi.org/10.1038/544S21a>
- LeCun, Y., Bengio, Y., Hinton, G., 2015. Deep learning. Nature 521, 436–444. <https://doi.org/10.1038/nature14539>
- Lecun, Y., Bottou, L., Bengio, Y., Haffner, P., 1998. Gradient-based learning applied to document recognition. Proceedings of the IEEE 86, 2278–2324. <https://doi.org/10.1109/5.726791>
- Li, R., Wang, R., Zhang, J., Xie, C., Liu, L., Wang, F., Chen, H., Chen, T., Hu, H., Jia, X., Hu, M., Zhou, M., Li, D., Liu, W., 2019. An Effective Data Augmentation Strategy for CNN- Based Pest Localization and Recognition in the Field. IEEE Access 7, 160274–160283. <https://doi.org/10.1109/ACCESS.2019.2949852>
- Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C.-Y., Berg, A.C., 2016. SSD: Single Shot MultiBox Detector, in Leibe, B., Matas, J., Sebe, N., Welling, M. (Eds.), Computer Vision – ECCV 2016, Lecture Notes in Computer Science. Springer International Publishing, Cham, pp. 21–37. https://doi.org/10.1007/978-3-319-46448-0_2

- Nanni, L., Maguolo, G., Pancino, F., 2020. Insect pest image detection and recognition based on bio-inspired methods. *Ecological Informatics* 57, 101089. <https://doi.org/10.1016/j.ecoinf.2020.101089>
- Ngugi, L.C., Abelwahab, M., Abo-Zahhad, M., 2020. Recent advances in image processing techniques for automated leaf pest and disease recognition – A review. *Information Processing in Agriculture*. <https://doi.org/10.1016/j.inpa.2020.04.004>
- Redmon, J., Divvala, S., Girshick, R., Farhadi, A., 2016. You Only Look Once: Unified, Real-Time Object Detection. Presented at the IEEE Conference on Computer Vision and Pattern Recognition, pp. 779–788.
- Redolfi, J., Felissia, S., Bernardi, E., Araguas, R., Flesia, A., 2020. Learning to Detect Vegetation Using Computer Vision and Low-Cost Cameras. pp. 791–796. <https://doi.org/10.1109/ICIT45562.2020.9067316>
- Ren, S., He, K., Girshick, R., Sun, J., 2016. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. arXiv:1506.01497 [cs]
- Sayali D. Jadhav, H. P. Channe2, 2016. Comparative Study of K-NN, Naive Bayes and Decision Tree Classification Techniques. *Int. J. Sci. Res. IJSR* 5, 1842–1845. <https://doi.org/10.21275/v5i1.NOV153131>
- Sharma, N., Jain, V., Mishra, A., 2018. An Analysis Of Convolutional Neural Networks For Image Classification. *Procedia Computer Science* 132, 377–384. <https://doi.org/10.1016/j.procs.2018.05.198>
- Shelhamer, E., Long, J., Darrell, T., 2016. Fully Convolutional Networks for Semantic Segmentation. arXiv:1605.06211 [cs].
- Szegedy, C., Wei Liu, Yangqing Jia, Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., Rabinovich, A., 2015. Going deeper with convolutions, in 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). Presented at the 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR),

pp. 1–9. <https://doi.org/10.1109/CVPR.2015.7298594> (Szegegy et al., 2015)

Tang, C., Chen, S., Zhou, X., Ruan, S., Wen, H., 2020. Small-Scale Face Detection Based on Improved R-FCN. *Applied Sciences* 10, 4177. <https://doi.org/10.3390/app10124177>

Tzutalin, labelling, <https://github.com/tzutalin/labelImg>

Ubbens, J., Cieslak, M., Prusinkiewicz, P., Stavness, I., 2018. The use of plant models in deep learning: an application to leaf counting in rosette plants. *Plant Methods* 14, 6. <https://doi.org/10.1186/s13007-018-0273-z>

Voulodimos, A., Doulamis, N., Doulamis, A., Protopapadakis, E., 2018. Deep Learning for Computer Vision: A Brief Review. *Computational Intelligence and Neuroscience* 2018, e7068349. <https://doi.org/10.1155/2018/7068349>

Wu, X., Zhan, C., Lai, Y.-K., Cheng, M.-M., Yang, J., 2019. IP102: A Large-Scale Benchmark Dataset for Insect Pest Recognition, in 2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). Presented at the 2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), IEEE, Long Beach, CA, USA, pp. 8779–8788. <https://doi.org/10.1109/CVPR.2019.00899>

Xiao, Y., Wang, X., Zhang, P., Meng, F., Shao, F., 2020. Object Detection Based on Faster R-CNN Algorithm with Skip Pooling and Fusion of Contextual Information. *Sensors* 20, 5490. <https://doi.org/10.3390/s20195490>

Zhao, Z.-Q., Zheng, P., Xu, S., Wu, X., 2019. Object Detection with Deep Learning: A Review. arXiv:1807.05511 [cs].

Zhong, Y., Gao, J., Lei, Q., Zhou, Y., 2018. A Vision-Based Counting and Recognition System for Flying Insects in Intelligent Agriculture. *Sensors* 18, 1489. <https://doi.org/10.3390/s18051489>

Zhou, G., Zhang, W., Chen, A., He, M., Ma, X., 2019. Rapid Detection of Rice Disease

Based on FCM-KM and Faster R-CNN Fusion. IEEE Access 7, 143190–143206.
<https://doi.org/10.1109/ACCESS.2019.2943454>

Zhou, G., Zhang, W., Chen, A., He, M., Ma, X., 2019. Rapid Detection of Rice Disease Based on FCM-KM and Faster R-CNN Fusion. IEEE Access 7, 143190–143206.
<https://doi.org/10.1109/ACCESS.2019.2943454>

DOASL (Department of Agriculture Sri Lanka). 1995. Major crop pests and their control with pesticides. Peradeniya (Sri Lanka): Extension and Communication Center, DOASL, P23

Ranasinghe, M.A.S.K.1992. Paddy Pests in Sri Lanka, Science education series No.30, Natural Resources energy & Science Authority of Colombo (Sri Lanka). P47-59

Smith, G and Reynolds L, 1966. Integrated strategies for the biological control of major insect pests of rice in South East Asia. Journal of Plant Protection in the Tropics. IRRI, Philippines 1, 19-37.

Appendix

User Manual to use Mobile application

Following screens are the home page and the pest details page of the application.

How to Identify pests using the mobile application

Go to Camera view, clicking by the camera icon in the bottom navigation bar. There are two buttons in this detect button, button and camera button. Using both of camera button and galley button, we can capture a pest image. After catching a pest image, we generate results.

Following two screens are Camera view , as shown in Figure 10

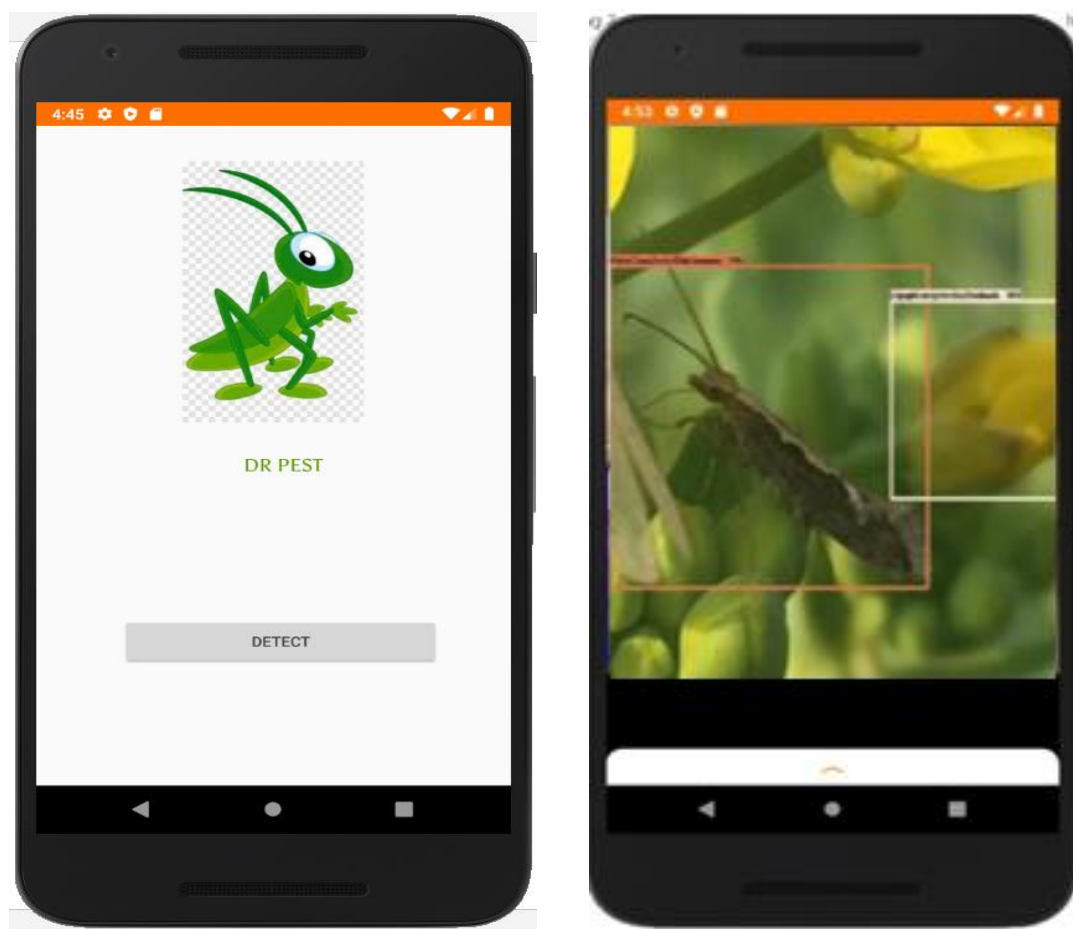


Figure 10: Camera View

