Statistical and Targeted Pest Detection in Agricultural Field Using Machine Learning and AI Powered UAV

Project Team

Sarim Ali p17-6051

Session 2017-2021

Supervised by

Dr. Nouman Azam

Department of Computer Science

National University of Computer and Emerging Sciences
Peshawar, Pakistan

June, 2021

Student's Declaration

I declare that this project titled "*Statistical and Targeted Pest Detection in Agricultural Field Using Machine Learning and AI Powered UAV*", submitted as requirement for the award of degree of Bachelors in Computer Science, does not contain any material previously submitted for a degree in any university; and that to the best of my knowledge, it does not contain any materials previously published or written by another person except where due reference is made in the text.

I understand that the management of Department of Computer Science, National University of Computer and Emerging Sciences, has a zero tolerance policy towards plagiarism. Therefore, I, as an author of the above-mentioned thesis, solemnly declare that no portion of my thesis has been plagiarized and any material used in the thesis from other sources is properly referenced.

I further understand that if I am found guilty of any form of plagiarism in the thesis work even after graduation, the University reserves the right to revoke my BS degree.

Sarim Ali Signature: _____

_____

Verified by Plagiarism Cell Officer

Dated:

Certificate of Approval

The Department of Computer Science, National University of Computer and Emerging Sciences, accepts this thesis titled *Statistical and Targeted Pest Detection in Agricultural Field Using Machine Learning and AI Powered UAV*, submitted by Sarim Ali (p17-6051), in its current form, and it is satisfying the dissertation requirements for the award of Bachelors Degree in Computer Science.

Supervisor

Dr. Nouman Azam Signature: _____

Mashal Khan

FYP Coordinator
National University of Computer and Emerging Sciences, Peshawar

---

Hafeez Ur Rehman

HoD of Department of Computer Science
National University of Computer and Emerging Sciences

Abstract

The purpose of this project is to develop an UAV-based detection system that would fly over an agricultural field, record video feed, and process it in real-time to detect pests or insects. The scope of the project involves gathering a valid dataset with well-distinctive images, training the YOLOv5 model, and fine-tuning the model to operate effectively. The model's performance is then assessed using various graphs, and it is made intelligent to detect the pest in any situation. This was accomplished through the use of deep learning, which included extracting characteristics from the background and foreground, resulting in a real-time detection of the pest in the field.The collected data can then be used to operate additional modules such as using the Geo-Location and the generate a heat map to spray the pesticide on the targets location autonomously if that is applicable and possible due to the high cost requirements.

Contents

List of Figures

List of Tables

Chapter 1

Preliminaries and Introduction

This project pertains to development of a statistical remedy for the crops protection in the agricultural sector using trained and AI based UAVs(1). The project will comprise specialization and implementation of Machine Learning to detect and differentiate the common pests in the farm field after forming a classifier.

1.1 Introduction

Pests have, from time immemorial, tormented humans. In this planet, fossils confirm the existence of flies predating humans. The earliest pest control example dates back to the time when a person killed a mosquito for the first time or swatted at an irritating insect. They have known the dangers of pests to food crops since humans took to agriculture and began to devise ways to repel those dangerous creatures(2). Nearly all ancient cultures apply to steps to control pests and to the use of chemical compounds to kill or repel insects. The earliest example of pest control documented is the use of sulfur compounds by ancient Sumerians to kill insects and the Ancient Greek civilization used fire to chase away locusts to sea.

Civilization began with agriculture, which remains very important to this day and plays a major role in our lives. And while in some countries its importance may be even more pronounced than in others, the fact is that, in one way or another, every country depends

on agriculture to sustain itself. In sustaining and moving the economy, agriculture plays an important role. It's the backbone of whatever drives us. It also offers job opportunities in addition to supplying food and other raw materials. It is fair to assume that agriculture's value can not be overstated.

1.1.1 Motivation

Agriculture is an important sector of Pakistan's economy, which relies heavily on its major crops.There are vast gaps between the acquired and actual output of produce, which suffers due to a lack of appropriate technology. This industry directly finances the country's population and accounts for 26% of the gross domestic product (GDP)(3) and employs 65%(4) of the workforce. A large-scale farmer monitoring consists of a ten-fold wider region for potential threats to farms/crops, etc.

The goal of this project is to provide something for the agricultural sector of our country, to provide a more efficient and optimal solution, with little or no harm to crops, as well as to increase output yields resulting in a boom in the agricultural economy.

1.1.2 Goals and Objectives

The following are the Key Targets and Objectives for this project:

1. Collection of a Valid Dataset without any bad training examples.
2. Selecting the Right Algorithm and Technique for better precision optimization.
3. Train a Model on the Dataset collected and test it.
4. Real-Time Target Tracking on a Video Feed.
5. Linking the Data collected to the Cloud.

1.1.3 About this Reoprt

This Reoprt concerns the development of a platform for UAV-based Pest Detection(1). It includes the overview of the specialization and implementation of Machine Learning for the identification of pests, all the information on the procedures and techniques used and the methodologies implemented. A real time target monitoring device that detects pests will be discussed in this study.

1.1.4 Object Detection

Object detection(5) is a computer technology that deals with finding instances of semantic items of a specific class (such as individuals, buildings, or cars) in digital photos and videos. It is related to computer vision and image processing. Due of its versatility, object detection is one of the most common computer vision models.

1.1.5 Deep Learning

Deep learning(6) is an artificial intelligence (AI) tool that mimics the human brain's data processing and pattern-making processes to aid decision-making. Deep learning is an artificial intelligence subset that uses neural networks to learn unsupervised from unstructured or unlabeled data. Deep neural learning or a deep neural network are other terms for the same thing.

1.2 The Problem Statement and Contribution

The Desert Locust (Schistocerca Gregaria) has long posed a serious threat to crop production, which, if overlooked, is detrimental to the country's agricultural economy. With the farmland environment being complex, it is often difficult to extract general features that are suitable for the detection of target pests. Desert Locust are voracious insects stating the can consume all types of vegetation, known to occur in desert areas with favorable Argo-ecological conditions as a species of swarming, migratory, trans-boundary short-

horned-locusts.

Figure 1.1: An Image of a Desert Locust (Schistocerca Gregaria)

Historically, swarms of Desert Locusts in Africa, the Middle East and Asia have always been a threat to agricultural production and food security. It is an occasional insect, with a life cycle of 12 weeks, with two to three generations per year. Desert Locust is theoretically vulnerable to an area of 16 million square kilometers containing 30 countries, while another 29 million square kilometers of world area covering 60 countries are at risk of invasion.

These insects grow and multiply under favorable Argo-ecological conditions, and migrate to other regions where food is available for them after consuming the vegetation in one field. They eat daily, up to their own weight. Desert Locust swarms can reach millions, fly up to 150 kilometers a day, and can travel almost 2,000 kilometers in their lifetime to find a suitable breeding climate(7). They replicate rapidly and, after about two weeks, eggs normally hatch. If not regulated, Desert Locust can cause a food security crisis or starvation. East Africa and South West Africa are facing an invasion of the Desert Locust today.

New waves of swarms emerged in the Empty Quarter in January 2019 and migrated north to the interior of Saudi Arabia and Iran, and south to Yemen. In both areas, breeding and a further increase occurred during the spring, causing new swarms to migrate to the border between Indo-Pakistan and the Horn of Africa at the beginning of last summer, respectively.

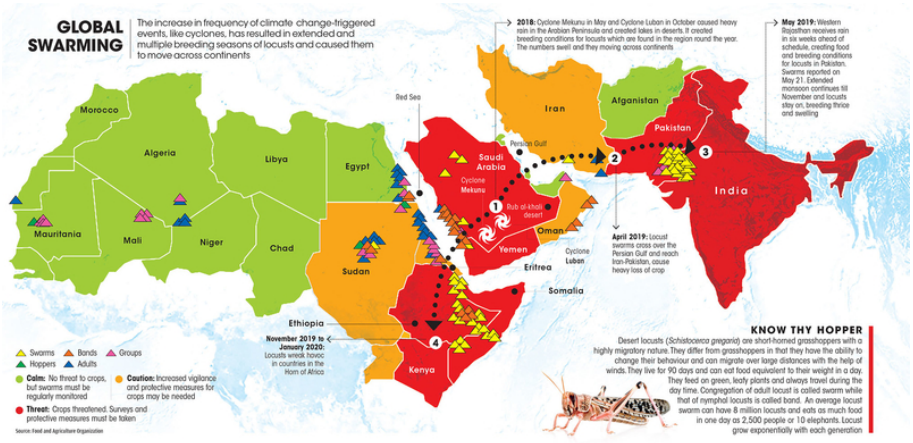


Figure 1.2: An Image showing the Trail of Hoppers Globally, including the Locust

Iran and Pakistan are particularly vulnerable in the Eastern Region, as locust breeding is taking place in these regions, even due to this year's wet winter. In Pakistan, 38 percent of Pakistan's territory (60 percent in Baluchistan, 25 percent in Sindh and 15 percent in Punjab) is a breeding ground for the Desert Locust, whereas if the Desert Locust is not contained in the breeding regions, the entire country is under threat of invasion.

1.2.1 Locust Surveillance and Control Operations Situation Analysis in Pakistan

The Ministry of National Food Security and Research (MFSR) of the Department of Plant Protection (DPP) is the lead agency responsible for monitoring and managing the threat of the Desert Locust in Pakistan(8). In February 2019, with support from FAO, DPP began operations to respond to the Desert Locust threat and immediately mobilized ground control teams and began operations in Baluchistan.

In May and June 2019, locust swarms from Baluchistan and the Iranian migrant population invaded areas in the Sindh province. Later, at the end of May 2019, locusts were registered in Rahimyar Khan, near the Punjab province's Cholistan Desert.

Figure 1.3: A farmer whose wheat crop was wiped out by locusts in Balochistan province. Photograph: Bloomberg/Getty Images

Figure 1.4: Locusts swarm in Hyderabad, Sindh province. Photograph: Anadolu Agency via Getty Images

In 2019, the Department of Plant Protection surveyed an area of 932,580 hectares and treated 300,595 hectares, consuming 150,839 liters of pesticides in three provinces. Out of the 300,595 hectares that were handled, aerial spraying cleared 20,300 hectares.

1.2.2 Major Field Crops of Pakistan

Cotton, wheat, rice(9), sugarcane, fruits and vegetables are among the main agricultural crops. There are two main crop production seasons in Pakistan: crops such as cotton, rice and sugar cane start in May and are harvested in November, while wheat is grown from November until April. Cotton exports accounted for 46% of Pakistan's total exports and provided the labor force with 35% of jobs (FAO., 2012, GOP., 2012).

1.3 Report Overview

1.3.1 Technology Used

To accomplish this project, the following tools are used:

Tech	Description
Google Colabratory	A Python environment that runs in the browser using Google Cloud.
TensorFlow 2.5 (GPU Based)	The core open source library to helps develop and train ML models
Python 3.9	Python is a high-level and general-purpose programming language
YOLO v5	Computer Vision and Pattern Recognition Algorithm
Roboflow	Platform to annotate, organize, train and deploy a dataset
WandB	A Unified Dashboard for Machine Learning Projects

Table 1.1: A table stating what technologies are used

1.3.2 Technical Details

I will use a UAV(1) to facilitate the detection to store and analyse the video feed, detect the target and collect the data with current geographical position information e.t.c. This will be done with the aid of various advanced and powerful algorithms for machine learning(10). The framework will be composed of sufficient precautionary and protection measures to help mitigate any environmental harm. The government will use and enforce this framework to ensure a stable flourish in the industry and eradicate the threat(9).

Chapter 2

Review of Literature

An overview of previous studies, work and projects related to Object and Pest Detection Systems is given in this section.

2.1 Fast YOLO: A System for Real-time Embedded Object Detection in Video

In the area of computer vision, object detection(11)(12) is considered one of the most difficult problems, since it requires the combination of object classification and object localization within a scene. Recently, compared to other approaches, deep neural networks (DNNs) have been shown to achieve superior object detection efficiency, with YOLOv2 (an improved You Only Look Once model) being one of the state-of-the-art methods of DNN-based object detection in terms of both speed and accuracy.

2.1.1 Discussion and Comparison

While YOLOv2 can achieve real-time output on a powerful GPU, leveraging this strategy for real-time video object detection on embedded computing devices with limited computational power and limited memory remains very difficult. In this literature review, I will discuss about the author proposes a new framework called Fast YOLO, a simple

You Only Look Once framework that speeds up YOLOv2 to be able to perform real-time object detection in video on embedded devices.

2.1.2 The proposed Fast YOLO framework

The system is split into two key components, as per the research: i) optimized YOLOv2 architecture, and ii) motion-adaptive inference. As described above, the study indicates that the main objective is to implement a video object detection system that can operate faster on embedded devices while reducing the use of energy, which in turn reduces the use of power significantly.

2.2 YOLOv4: Optimal Speed and Accuracy of Object Detection

The topic of Convolutionary Neural Networks and their modifications is part of this article. There is a need for realistic testing of combinations of such characteristics on large datasets and theoretical justification of the outcome. Some features work only on certain models and exclusively or only for small-scale datasets for certain problems; while some features, such as batch-normalization and residual-connections, extend to most models, tasks, and datasets.

2.2.1 The Operation of Real-Time Object Detectors

The operation of real-time object detectors on traditional graphics processing units (GPU) enables their mass use at an affordable cost. Modern neural networks that are most accurate do not function in real time and require a large number of GPUs for large mini-batch-size training. The research compares how, with comparable efficiency, YOLOv4 runs twice faster than EfficientDet. It increases the AP and FPS of YOLOv3 by 10% and 12%, respectively. The intention is that you can easily train and use the built object.

2.2.2 The Composition of Object Detectors

An ordinary detector of objects consists of several parts: BACKBONE, HEAD, which is actually the input images, is typically divided into two groups, i.e. one-stage object detector and two-stage object detector. The R-CNN(12) series, including fast R-CNN(13), faster R-CNN(14), R-FCN(15), and Libra R-CNN(16), is the most representative two-stage object detector, and a two-stage object detector can also be turned into an anchor-free object detector. As for the one-stage object detector, YOLO(17)(18)(19), SSD(20), and RetinaNet(21) are the most representative versions.

In recent years, object detectors have also introduced certain layers between the backbone and the head, and these layers are typically used to obtain feature maps from various points. It is known as the NECK of a detector of an object. Typically, several bottom-up paths and several top-down paths consist of a neck.

A state-of-the-art detector that is faster (FPS) and more precise than all existing alternative detectors can be derived from this work. The stated detector can be trained and used on a traditional GPU, making its wide-ranging use possible. Viability has been proved by the initial design of one-stage anchor-based detectors. A large number of characteristics are tested and chosen for use in order to increase the accuracy of both the classifier and the detector. For future research and inventions, these characteristics may be seen as best practice.

2.3 Use of UAVs for Agricultural Applications Especially on Crop Protection

This research composes of new applications for Unmanned Aerial Vehicles (UAVs) are to be implemented in agriculture and in crop protection in particular. All the experiments were performed on the island of Crete, Greece. Relevant applications are: a) the detection of symptoms (canopy discoloration) of pest or disease infestation in olive trees, b) the mapping of palm trees and the identification of visible signs of infestation in large palm plantations by the red palm weevil Rhynchophorus ferrugineus, c) the cooperation of UAVs with electronic traps that automatically count insects and submit counts.

2.3.1 Technology and Agriculture

It is envisaged that UAVs will become one of the technologies of the future. Satellites provide costly resources and do not fulfill the growing need for spatial resolution and repetitive controllable measurements, such as crop health and soil property monitoring, for such applications. For repetitive tasks that are considered long and monotonous for humans or even harmful(22), UAVs have been suggested(10).

In a growing number of applications for agricultural and environmental tasks, UAVs tend to play a key role. There is continuous celebration of their use and future prospects. Their usage, however, slowly penetrates the routine practice of most growers, as the latter are unsure of how this technology might ultimately lead to cost savings or revenue growth. The goal of this research was to bring this technology to small/medium regional stakeholders in an attempt to help them deal with the efficacy of common detection methods relevant to their crop protection problems and to reduce their monitoring costs.

Chapter 3

Methodology

Object detection(21) is a computer vision problem that involves identifying things in photos and videos. Algorithms such as YOLO(23) (You Only Look Once), Single Shot Detector (SSD), Faster R-CNN, Histogram of Oriented Gradients (HOG), and others are available.

Image processing, Artificial Intelligence, and Machine Learning techniques are used in this methodology. There are other approaches to machine learning, but this research relies on the deep learning method, otherwise known as model-based learning. For the training process, this methodology employs supervised learning, which will be explored in greater detail in the following subsections.

Prerequisites

A rudimentary understanding of deep learning computer vision is beneficial. It's also necessary to understand how to work in a Google Colab environment.

3.1 Proposed Methodology for Pest Detection using YOLOv5

Supervised learning is a method of developing artificial intelligence (AI) that involves training a computer algorithm on input data that has been labelled for a certain output. When provided with never-before-seen data, the model is trained until it can discover the underlying patterns and relationships between the input data and the output labels, allowing it to produce accurate labelling results.

Glenn Jocher, the founder and CEO of Utralytics, released version 1.0 of YOLO-V5(24) on May 27, 2020. It is written in PyTorch and released on Github. Yolo-V5 will be used to train our pest detection model throughout this project. One of the most well-known object detection models is YOLO as it is known for its excellent accuracy, performance and easy to use features. The YOLOv5(24) model is the most current addition to the YOLO family of models. YOLO was the first object detection model to include bounding box prediction and object classification into a single end-to-end differentiable network. It was created and is maintained using the Darknet framework.

YOLOv5 is the first YOLO model to be developed in the PyTorch framework, making it significantly lighter and easier to use. However, YOLOv5 does not beat YOLOv4 on a standard benchmark, the COCO dataset, because it did not make fundamental architectural improvements to the network in YOLOv4.

3.2 Dataset for Evaluation of the Proposed Methodology

In order to start with the implementation part and to get the pest detector off the ground, we needed to first collect training images. We will utilise the GHCID (GrassHopper identification Dataset) for grasshopper detection, which can aid in the creation of new algorithms for further research and changes in this field.

We need to use bounding box annotations to supervise our pest detector's learning. We draw a box around each thing we want the detector to identify and label each box with the object class we want it to predict.

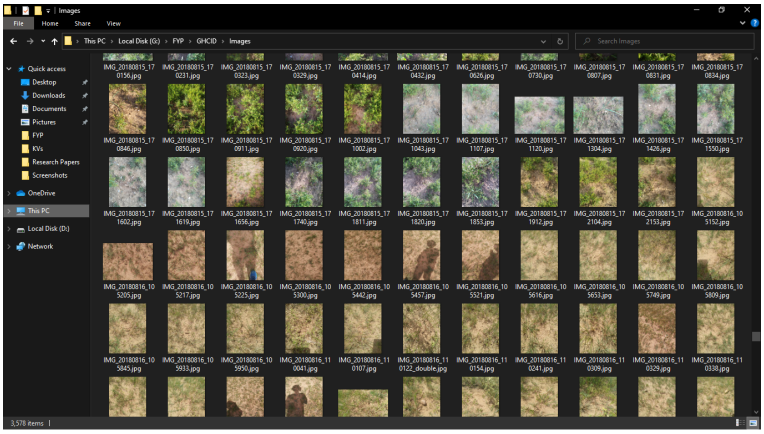


Figure 3.1: Screenshot of the dataset images

The GHCID Dataset was created by photographing frozen locusts at the CAAS research station near Xilinhot, and includes 3,578 images with a resolution of 2976 x 3968 pixels and 4,406 instances of labelled adult hoppers with various bounding boxes.

3.2.1 Roboflow: A Dataset Managment IDE

Roboflow is a platform that includes all of the tools we'll need to turn raw photos into a custom-trained computer vision model and deploy it in your apps. Roboflow now includes models for object identification and classification.

Roboflow can be used to:

- Annotate images
- See if labels are in-frame
- Preprocess images: *resizing, grayscale, auto-orientation, contrast adjustments*
- Augment images *to increase your training data: flip, rotate, brighten / darken, crop, shear, blur, and add random noise*
- Generate annotation formats
- Quickly assess your dataset quality

We upload our dataset to Roboflow, divide it into sets for training, testing, and validation, generate numerous new versions of the dataset, apply preprocessing procedures, and enrich the dataset, and end up with a moment-in-time snapshot of the dataset with the applied transformations.

Larger versions take longer to train, but they often produce greater results. We have already labelled instances of our pest in our dataset so we move on to the pre-processing and further parts of the training working with Roboflow.

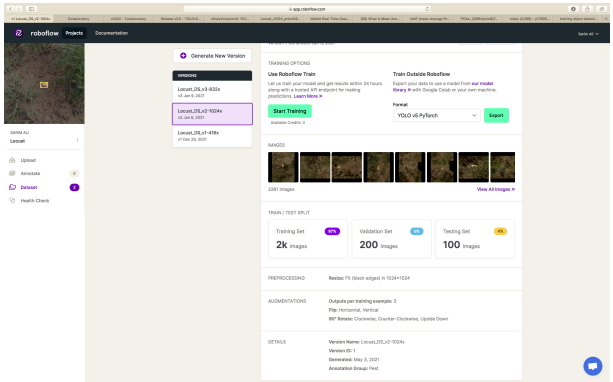


Figure 3.2: Roboflow Dashboard for maintaining different versions of the Dataset

3.2.1.1 Image Pre-processing

We started with 3,578 total instances, which we can refer to as source images. After uploading all of them to the Roboflow Dash, we divided the images into fixed percentages of 70% for training, 20% for validation, and the remaining 10% for testing.

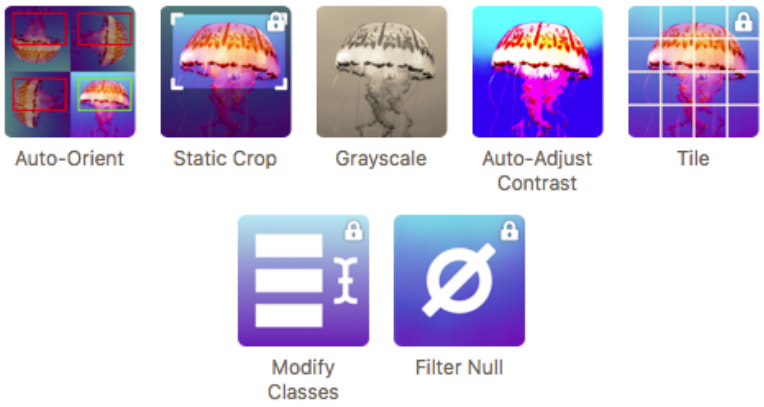


Figure 3.3: Image Pre-processing Options

Resize For train and test sets, each image with various high resolutions captured with high-end cameras is translated into a size multiple of 32, to provide uniform image dimensions for the model to work on.

This resizing is done to reduce the amount of time it takes to acquire hyper parameters. Smaller sizes are preferable for speedier results, and they can be increased as needed to improve accuracy.

Image equalization We may have various contrast places or various lighting conditions because the photographs were obtained at different times and locations, which can alter the outcomes.

Equalization of the image histogram is used to normalise under varied lighting situations. The image is then smoothed to remove noise caused by histogram equalisation and get the best feasible result.

3.2.1.2 Image Augmentation

Deep networks require a substantial amount of training data to perform adequately. Image augmentation is necessary to influence the efficiency of deep networks in order to develop a powerful image classifier with very little training data.

Image augmentation artificially builds training images using a variety of processing techniques or a mix of techniques, such as random rotation, shifts, shear, and flips, among other options which can be seen in the Fig 3.4 on page 18

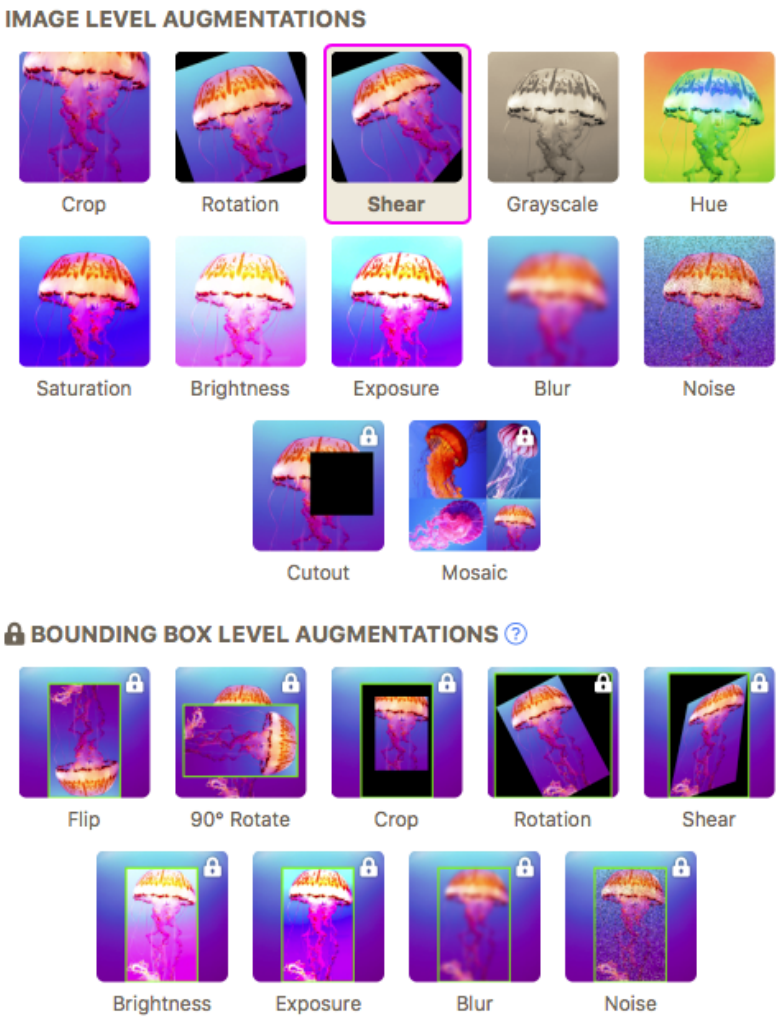


Figure 3.4: Image Augmentations Options

3.2.1.3 Generate Annotation Formats

When it comes to image annotation, there is no single standard format. COCO has five different annotation formats, all of which are saved in JSON. Pascal VOC stores annotations in an XML file. The YOLO labelling format, and creates a.txt file for each image file in the same directory. The annotations for the matching image file are contained in each.txt file, which includes object class, object coordinates, height, and width.

After generating a version of the dataset, we can export the dataset in any specific annotation format as we require. We use the YOLO format for our requirements here. We are now ready to set up the environment and train the model after the dataset has been prepared.

3.3 Setting Up The YOLOv5 Environment

To get started with YOLOv5(24), we need to clone the repository and install the dependencies. This will prepare our programming environment so that object detection training and inference instructions may be executed. The most crucial thing to keep in mind is that you'll need PyTorch 1.5, Python 3.7, and CUDA 10.2 or higher.

The remainder of the dependencies are simple to install with pip if required.

3.3.0.1 Google Colab

Google Research's Colaboratory, or "Colab" for short, is a free Integrated Development Environment (IDE). Colab is a web-based Python editor that allows anyone to write and run any Python code. It's notably useful for machine learning, data analysis, and education. Colab is a hosted Jupyter notebook service that doesn't require any setup and offers free access to computational resources, including GPUs.

To use Google Colab, we need a Google account. You can get a free GPU for 12 hours if you join Google Colab.

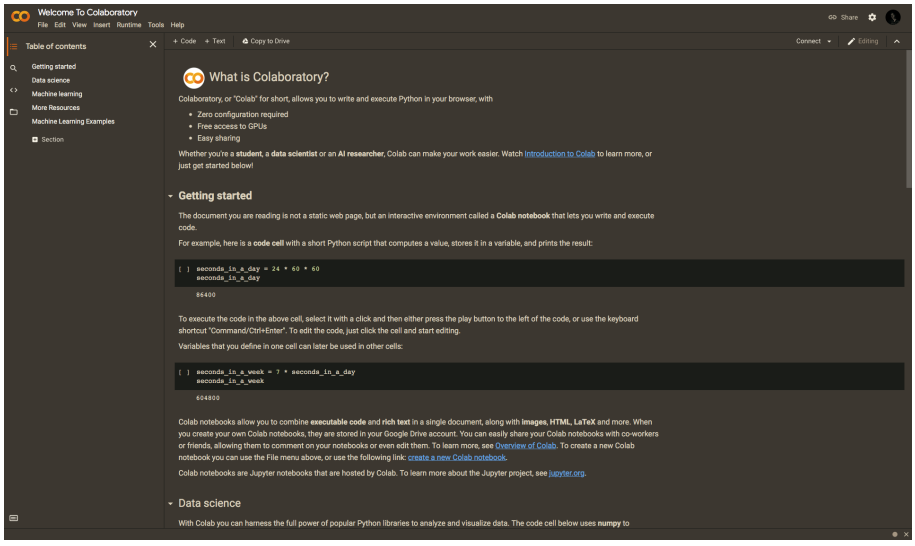


Figure 3.5: Google Colaboratory Web-interface

Due to the superb availability of resources and high-end GPUs, We used Google Co-laboratory environment, providing us with mobility and advanced features. We use it throughout the training process as well as the testing. PyTorch, TensorFlow, Keras, and OpenCV are just a few of the major Deep Learning(6) libraries that come pre-installed on Google Colab.

3.3.0.2 Installation Of Dependencies

Returning to the setup section, I'll list and describe the code snippets that were used to help set up the environment. *(Remember to choose GPU in Runtime if not already selected. Runtime –> Change Runtime Type –> Hardware accelerator –> GPU)*

```
# clone YOLOv5 repository
!git clone https://github.com/ultralytics/yolov5  # clone repo
%cd yolov5
```

Listing 3.1: Cloning YOLOv5 repository

```
# install dependencies as necessary
!pip install -qr requirements.txt  # install dependencies (ignore
    errors)
import torch

from IPython.display import Image, clear_output  # to display
    images
from utils.google_utils import gdrive_download  # to download
    models/datasets

# clear_output()
print('Setup complete. Using torch %s %s' % (torch.__version__,
    torch.cuda.get_device_properties(0) if torch.cuda.is_available()
     else 'CPU'))
```

Listing 3.2: Installation Of Dependencies

Machine learning/deep learning algorithms necessitate a system with a high processing speed and power (usually based on GPU), which normal systems are not equipped with. It's likely that Google Colab will send you a Tesla P100 GPU. We are be able to reduce training time by using the GPU. Colab is also convenient because it comes with torch and cuda preloaded. If you're doing this instruction on your own computer, there may be some extra steps to set up YOLOv5(24).

3.3.0.3 Directory and Data Setup

Once the environment setup was done, I used colab to import the dataset. I used the Roboflow dataset, but if you want to use your own, you may import it using Google Drive.

```
# You need to sign up in roboflow to get the key and then you can use the dataset
\!curl -L âĂİhttps://public.roboflow.com/ds/XXXXXâĂİ > roboflow.zip ; unzip roboflow.zip; rm roboflow.zip
```

Listing 3.3: Downloading the Dataset to our YOLOv5 directory

3.3.0.4 Model Configuration and Architecture

Glenn Jocher facilitates some YOLOv5 model instances based on previous research. These architectures will be read from the `yaml` file and built in the `train.py` file by the YOLOv5 model on PyTorch. This also makes configuring the architecture for different object detection challenges a lot easier.

Before the training for the pest detector, I created a custom model configuration file with alterations to the original ones provided by Glenn. We can also alter the network structure in this phase if necessary, though this is rare. The YOLOv5 model configuration file, custom `yolov5l.yaml`, as shown in Fig 3.6 below.

```
#this is the model configuration I will use for this project
%cat /content/yolov5/models/yolov5l.yaml

# parameters
nc: 80  # number of classes
depth_multiple: 1.0  # model depth multiple
width_multiple: 1.0  # layer channel multiple

# anchors
anchors:
  - [10,13, 16,30, 33,23]  # P3/8
  - [30,61, 62,45, 59,119]  # P4/16
  - [116,90, 156,198, 373,326]  # P5/32

# YOLOv5 backbone
backbone:
  # [from, number, module, args]
  [[-1, 1, Focus, [64, 3]],  # 0-P1/2
   [-1, 1, Conv, [128, 3, 2]],  # 1-P2/4
   [-1, 3, C3, [128]],
   [-1, 1, Conv, [256, 3, 2]],  # 3-P3/8
   [-1, 9, C3, [256]],
   [-1, 1, Conv, [512, 3, 2]],  # 5-P4/16
   [-1, 9, C3, [512]],
   [-1, 1, Conv, [1024, 3, 2]],  # 7-P5/32
   [-1, 1, SPP, [1024, [5, 9, 13]]],
   [-1, 3, C3, [1024, False]],  # 9
  ]

# YOLOv5 head
head:
  [[-1, 1, Conv, [512, 1, 1]],
   [-1, 1, nn.Upsample, [None, 2, 'nearest']],
   [[-1, 6], 1, Concat, [1]],  # cat backbone P4
   [-1, 3, C3, [512, False]],  # 13

   [-1, 1, Conv, [256, 1, 1]],
   [-1, 1, nn.Upsample, [None, 2, 'nearest']],
   [[-1, 4], 1, Concat, [1]],  # cat backbone P3
   [-1, 3, C3, [256, False]],  # 17 (P3/8-small)

   [-1, 1, Conv, [256, 3, 2]],
   [[-1, 14], 1, Concat, [1]],  # cat head P4
   [-1, 3, C3, [512, False]],  # 20 (P4/16-medium)

   [-1, 1, Conv, [512, 3, 2]],
   [[-1, 10], 1, Concat, [1]],  # cat head P5
   [-1, 3, C3, [1024, False]],  # 23 (P5/32-large)

   [[17, 20, 23], 1, Detect, [nc, anchors]],  # Detect(P3, P4, P5)
  ]
```

Figure 3.6: Snapshot of the Model Configuration Code Cell

3.4 Experimental Setup

After setting up of the model, the next step is to train the model. In deep learning(25), the training process consumes a lot of effort and resources. Because the final outcome of the procedure is largely determined by the training process' quality. The dataset that was prepared on Roboflow was used to train the upgraded YOLOv5l network in this study.

I used multiple values varying vastly for the training, in search for the best possible result in time with the best accuracy, varying the image sizes between different resolutions, changing the number of batches, epochs. Some of my training runs ended up crashing due to Google Colabs limits but mostly worked out pretty well.

```
# train yolov5s on custom data for 150 epochs
# time its performance
%%time
%cd /content/yolov5/
!python train.py --img 1024 --batch 16 --epochs 150 --data '../data
   .yaml' --cfg ./models/custom_yolov5l.yaml --weights './runs/
   train/yolov5l6_results2/weights/best.pt' --name
   Locust_x1024_yolov5l6_results  --cache
```

Listing 3.4: Code for the Training of the Model

The parameters involved in the training process are listed below:

img: State the resolution of the input image. The original image is 2976 x 3968 pixels, but it has been compressed to make the training process go more quickly. Numerous computer vision experts agreed that the size 416 x 416 is the best size to utilise as input without losing a lot of detail after many testing runs.

batch: Specify the batch size. The number of weights that the model learns in one time (one epoch) increases dramatically when thousands of images are fed into the neural network at the same time. As a result, the dataset is frequently partitioned into multiple

batches of 'n' photos, with each batch being trained separately.

After all of the batches have been trained, the results of each batch are saved to RAM and aggregated. Because the weights learned from the batches are saved in RAM, the more batches you have, the more memory you'll utilise.

As the training set comprises 10,734 images and the batch size is set to 16, the total number of batches will be 10,734 / 16 = 670.

epochs: State the number of training epochs. An epoch is responsible of learning all of the input images, or training all of the input. Because the dataset is divided into different batches, a single epoch will be in charge of training all of them. The number of epochs indicates how many times the model trains all of the inputs and updates the weights in order to get closer to the ground truth labels. Experience and intuition are frequently used to make decisions. It is normal to have more than 3000 epochs.

data: the path to the `data.yaml` file containing the dataset summary The model will access the validation directory through the location in the `data.yaml` file and use its contents for evaluation at that time because the model evaluation process runs immediately after each epoch.

cfg: Specify the location of our model's config path. This command line allows the `train.py` file to compile and create the architecture for training input photos based on the architecture described in the model `yaml` file previously.

weights: State a weights path. It is possible to save time by using a pre-trained weight. If this field is left blank, the model will generate random weights for training.

name: Result directory name. The model will build a directory that contains all of the training results.

cache: For boosted training.

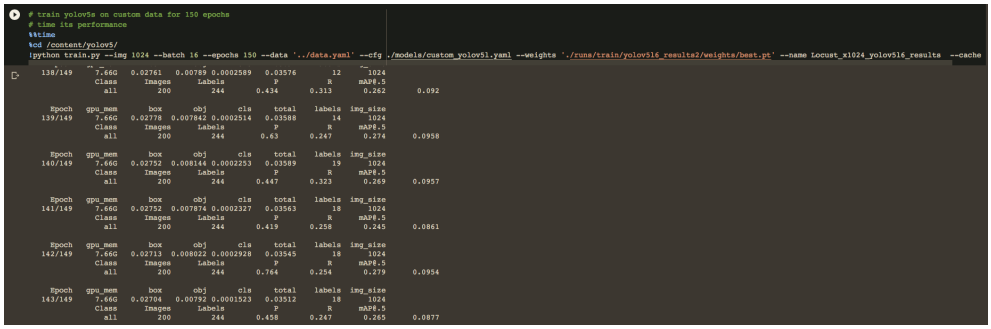


Figure 3.7: Training Process Execution

The average duration to execute the training process on 16 batches for one epoch was 1 minute and 20 seconds for 1024 size images. On one of the datasets including 1981 photos, the overall execution time was 3 hours 10 minutes and 15 seconds for 150 epochs, with the mAP of the last epoch being 86.2%.

The weight file of the recognition model obtained after training was kept, and the test set was used to evaluate the model's performance.

Now that we have completed training, we can assess how well the training performed by examining the validation metrics.

3.4.1 Model Evaluation Indicators

The performance of the trained pest targets detection model was evaluated using objective evaluation indicators such as Precision (1), Recall (2), mAP (mean average precision) (3), and F1 score (4) in this research. The following are the calculating equations:

$$Precision = \frac{TP}{TP + FP} \tag{3.1}$$

Precision: calculates the percentage of positive forecasts that are correct. It's the proportion of successfully anticipated positive values to total positive values projected. A

high level of precision indicates a low rate of false positives.

$$Recall = \frac{TP}{TP + FN} \tag{3.2}$$

Recall: The ratio of accurately predicted positive values to actual positive values, also known as sensitivity, evaluates how successful the model is in discovering all positive predictions.

$$mAP = \frac{1}{C} \sum_{K=i}^{N} P(K) \Delta R(K) \tag{3.3}$$

mAP: Depending on the different detection problems that exist, the mean Average Precision or mAP score is calculated by taking the mean AP over all classes and/or overall IoU thresholds. Here *P(K)* is Precision and *R(K)* is the Recall value.

$$F1 = \frac{2}{\frac{1}{Precision} + \frac{1}{Recall}} \tag{3.4}$$

The F1-score: It is defined as the harmonic mean of the model's precision and recall, and it is a method of combining the model's precision and recall.

Here in the equations above, the number of successfully detected pest targets is denoted by *TP*, while the number of misidentified background as pest targets is denoted by *FP*. The number of unidentified pest targets is denoted by the letter *FN*. The number of pest target categories is denoted by the letter *C*. The number of *IOU* thresholds is *N*, and the *IOU* threshold is *K*.

3.4.2 Prediction Confidence Threshold

The recognised targets would be filtered using the predefined threshold after the recognition model had determined the target's confidence. Based on the same recognition model with varying thresholds for prediction, the precision and recall of the detection results are varied. The prediction results when the pest is too far hidden or small in perspective would be discovered by mistake as seen in Fig 3.8. If the confidence threshold was set too low if the recognition model's confidence threshold was not set suitably.

Figure 3.8: Threshold is too low

If the threshold was set too high, the pest target in the foreground would not be recognised as shown in Fig 3.9. As a result, determining an appropriate confidence threshold for the recognition model is critical, and the pest targets may then be successfully detected based on the model's anticipated confidence.

Figure 3.9: Threshold is too high

The optimal prediction threshold was determined by combining the actual demand of the pest target recognition task with the trained pest detection model, and comparing the recognition precision, recall, and mAP of the detection results using different threshold values for two varieties of targets in the test data set, which included more than 200 images. Using varying confidence criteria, the model's precision, recall, and mAP were calculated.

Figure 3.10: Test Results labeled

On the other hand, mAP, which is used to evaluate the model's performance, should be considered when determining the threshold because it can reflect both precision and recall. When the confidence threshold is set at 0.5, the model's performance is the best, with precision of 81.48 percent, recall of 89.75 percent, and mAP of 86.2 percent, respectively.

When concepts become overly intricate and perplexing, they must be broken down into smaller problems. When developing a complex product, not only the product but also the development process must be structured to make it practicable. Many of these procedures exist in the fields of software development and systems engineering.

The figures in this section depicts the UML Design of Software Engineering process, which aids in the creation of a robust software architectural design.

3.4.3 Use Case Diagram

The primary use cases that the UAV would perform to execute the proposed concept are depicted in the Use Case diagram Figure 3.11. The user will begin by initiating and interacting with the UAV. The UAV will capture video data and then process it on the system, detecting and making appropriate entries.

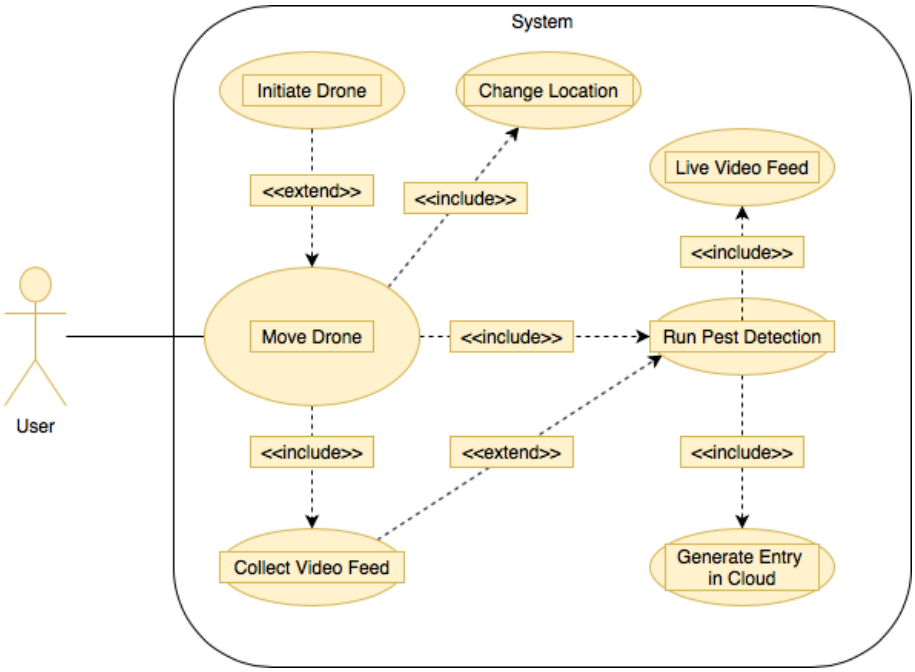


Figure 3.11: Use Case Diagram of the Proposed System

3.4.4 Class Diagram

The class diagram as in the Figure 3.12 aids in specifying the envisioned system's attributes, variables, and functions. There is a list of classes that we utilise in this project to detect, distribute, and classify our provided data, as well as how our agent will detect and conduct these functions.

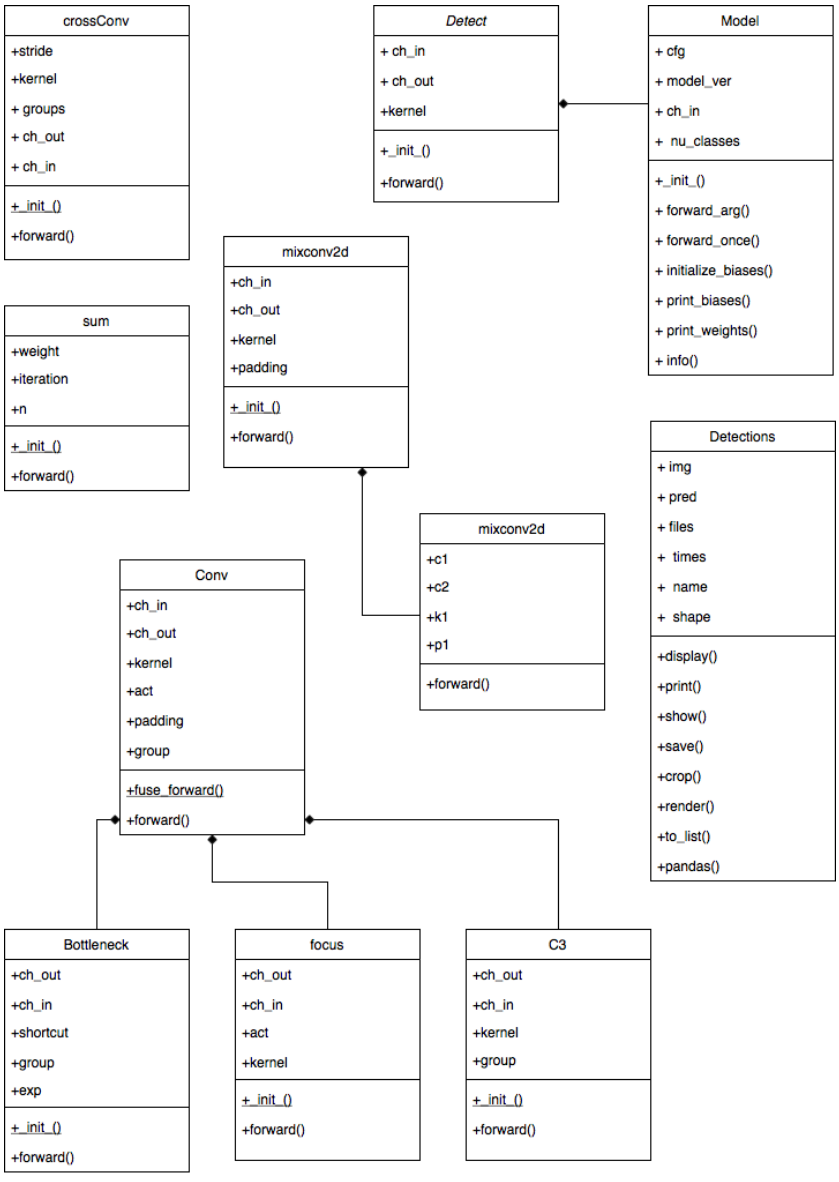


Figure 3.12: Class Diagram

3.4.5 Activity Diagram

It's a flowchart of how the project operates, demonstrating how all of the various components interact.

We'll start with the UAV(1) (our video capturing device) and launch the flight; the UAV will capture video; the system will read the video feed; and then, using the trained model, it will try to find and locate pests in the video feed; if true positives are found, the location and number of instances detected will be recorded. The system will come to a halt after the needed area has been covered. All this flow is shown in the Activity Diagram of the project that is shown in the Figure 3.13 on the following page.

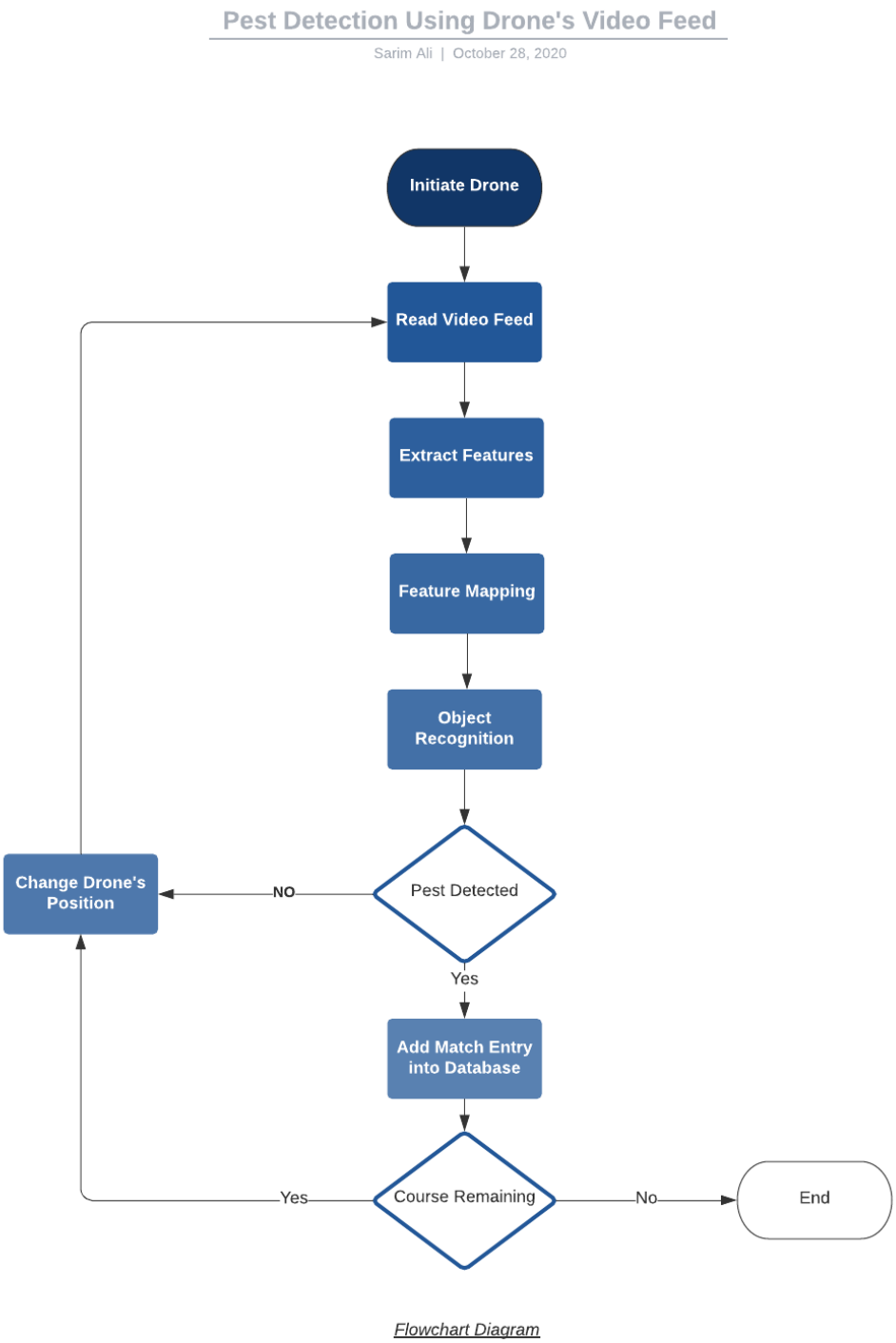


Figure 3.13: Activity Diagram: Flowchart showing different stages of the system

Chapter 4

Results

The outcomes of the generated model will be discussed in this portion of the study. The constructed model operates fairly on the augmented data, which is falsely created test data. We obtain the weight, which we can utilise for either further training or model detection.

4.1 WandB.ai Integration

With a unified dashboard for machine learning projects, Weights and Biases lets you develop better models faster. Using their tools, we can log hyper-parameters and output metrics from our tests, then visualise and compare the results, as well as instantly share our findings with anyone.

In my training, I used wandb integration to track my progress and preserve all of my run logs and statistics, and it was quite useful.

Figure 4.1: WandB - Weights and Biases Dashboard for visualising different results obtained from the training

The App User Interface includes a well-organized dashboard as seen in the Figure 4.1, that includes a Project Page for comparing numerous experiments in one location, a Run Page for viewing the results of a single run, and workspaces for configuring and customising the sandbox's visualisations. WandB offers the widest range of graphs available, as well as real-time comparisons of different runs, all in one spot.

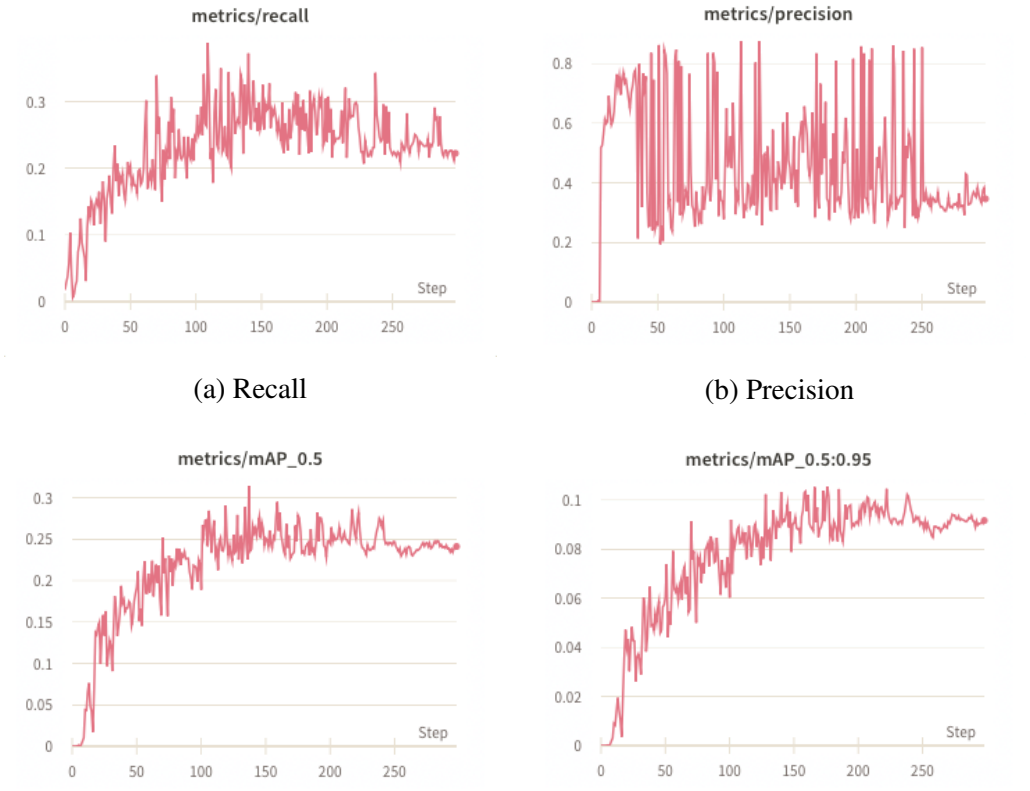
4.2 Results and Analysis of the Pest Detection

To test the system's performance, I ran trainings on various YOLOv5(24) models, ranging from s for short datasets to m for medium datasets to l for large datasets. The outcomes of more than 200 photographs from the test set were studied further in the study. I started by splitting the 1000 photos in the dataset into 700 for further augmentation, which were then utilised to train the model. Validation is worth 200 points, and testing is worth 100 points. In later versions of the dataset, the number of images was raised to improve training outcomes and time efficiency.

Test Set	Images	Resolution	Precision (%)	Recall (%)	mAP (%)
Run-03 Yolov5L	1981	1024	83.17	71.8	81.26
Run-07 Yolov5L	1969	416	87.49	80.68	85.47
Run-08 Yolov5L	2378	1024	86.79	67.2	87.12

Table 4.1: Top 3 Recognition results of Locust Detection using improved YOLOv5l network

Above in Table 4.1, I chose the best results that I got so far, varying of the dataset in different sizes and number of images. In the future, the pre-trained weights made by the training of these models will be used to further improve the results and accuracy with higher dataset images for training, validation and testing. Now choosing one of the run, Figure 4.2 depicts four graphs plotted for Precision, Recall, mAP accordingly, stating every reading at a specific epoch in the training run where as Figure 4.1 is showing the dashboard of all the runs combined in which then we can select the best run and choose the best weight for further development.



(a) Recall

(b) Precision

(c) mAP 0.5

(d) mAP 0.5:0.95

Figure 4.2: The Evaluation Metrics graphs of the entire 300 epochs of a run

A Confusion Matrix is a table that shows how well a classification model (or "classifier") performs on a set of test data for which the true values are known. The confusion matrix itself is straightforward, but the associated nomenclature might be perplexing. Figure 4.3 is one of a early generated result for the initial runs having not bad F-1 Scores.

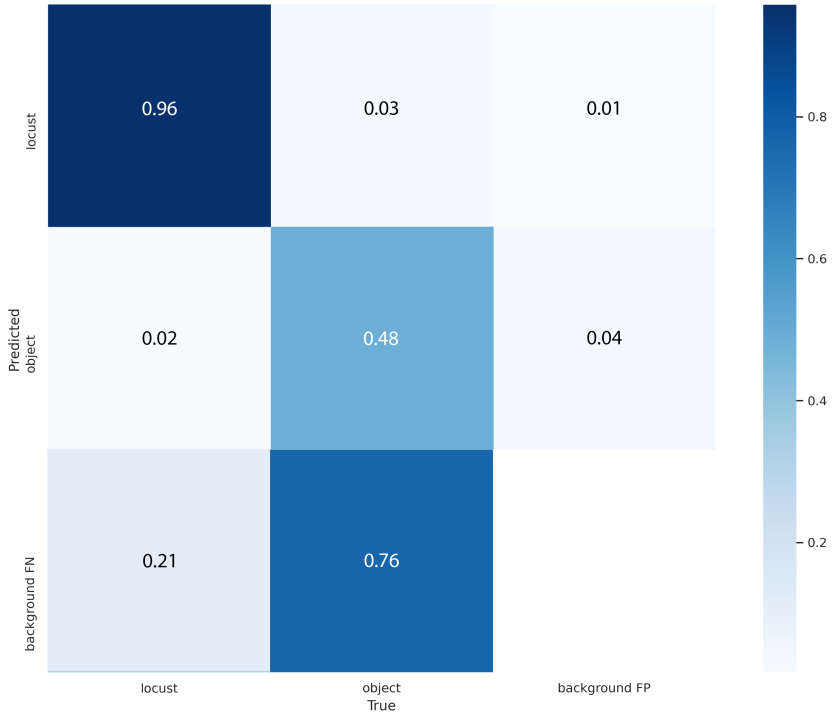


Figure 4.3: Confusion Matrix of the result with the F-1 Score.

While a test is being done, the WandB IDE also monitors real-time hardware performance, usage and power consumption, as well as the temperature of the components, offering a comprehensive overview of everything on one screen, which I find very handy.

The model trained on YOLOv5, as demonstrated in Figure 4.4, produces exceptionally outstanding predicted results on images it has never seen before. The model almost detects all wheat pests that appear in the image, despite the low density of locusts in each shot.

Figure 4.4: Batch of images that were detected by using best trained weight

Chapter 5

Discussions

After working on this project, I discovered a lot of voids, a lot of new things to learn, and I was able to accomplish a lot on my own. There isn't a lot of research or implementation in this subject, and there are a lot of closed doors, but there's always something new out there that needs to be done.

Understanding the concepts and complexity of this problem, with locusts being a small insect that hides and moves on the ground almost undetected, was difficult. After conducting extensive literature reviews and gathering knowledge, we learned and built an agent that detects the wicked little hopper, trained it, and improved its performance. There are a lot of existing recognition algorithms that people have worked on, recognition models range from complex to simple.

Significantly, the model's small size makes it easier to work with the model in hardware devices later. Furthermore, the YOLOv4 algorithm's identification findings in Section 2.2 demonstrate that the model's detection performance is good, as its mAP was the greatest among the four contrasted algorithms, even higher than the mAP of the original YOLOv5s, indicating the model's strong target detection capabilities. However, the model's size is rather high, at 244 MB, which may increase the recognition algorithm's deployment cost in embedded devices.

The YOLOv5(24) network has four alternative architecture (YOLOv5s, YOLOv5m, YOLOv5l, and YOLOv5x) with varying sizes, demonstrating the model's versatility. As a result,

based on the actual work requirements, users can select a suitable model with the suitable size for development and application. The advantages of the YOLOv5s network's lightweight (very small model size) and high detection speed will minimise the identification model's deployment cost, showing the detection model's tremendous potential for deployment in embedded devices based on upgraded YOLOv5. The detection performance, particularly the detection speed of the better developed YOLOv5s model, is good, making it appropriate for the picking robot's real-time apple recognition; finally, the suggested detection model is relatively small, indicating that it has a lot of potential for use in hardware devices, which is critical for the detection algorithm's widespread applicability.

Chapter 6

Conclusions and Future Work

After carrying out all the analysis and reading the materials, we may conclude how every work done has strengths and limitations, regardless of size, my aim was to resolve the shortcomings and extract a coordinated project to evaluate, learn and identify the pest and eradicate it from the core, provide a solution to strengthen the industry.

The paper presented a light-weight pest target real-time detection approach for locusts based on improved YOLOv5. The bonding fusion mode of feature maps that were input into the target detection layer of medium size in the original YOLOv5l network was modified to increase the recognition accuracy of pest targets. The initial anchor box size of the original network was increased to avoid nonrecognition of little bugs in the image backdrop. The test set's detection results revealed that the suggested enhanced network model can successfully accomplish the recognition of locusts shot with a good camera device.

6.1 Future Work

Take into account the detection algorithm's limitations. A big amount of locust picture data will be obtained during the debate in order to improve the application span of our method. Furthermore, artificial illumination will be used to take the photographs under various lighting circumstances. All of the above image samples will be included to the

training sets that will be used to train the detection model in order to achieve automatic detection of pest targets in images at all times of day and night.

I plan on working on this project futher with better resources and turn this into a product that would not only sell but help the agriculture grow and flourish to its best. I plan on making a full working UAV(9) with all the functionality described in this report. In the future, the detection network architecture suggested in this paper could be used to recognise target objects of any type in unmanned aerial vehicle (UAV)-based remote sensing images.

Appendix A

Appendix A: Source Code

A.1 Model Training

This contains the model training part of the code which runs after the dataset is ready and all the environment configurations are made.

```
def train(hyp, opt, device, tb_writer=None):
    logger.info(colorstr('hyperparameters: ') + ', '.join(f'{k}={v}
    ' for k, v in hyp.items()))
    save_dir, epochs, batch_size, total_batch_size, weights, rank =
    \
        Path(opt.save_dir), opt.epochs, opt.batch_size, opt.
    total_batch_size, opt.weights, opt.global_rank

    # Directories
    wdir = save_dir / 'weights'
    wdir.mkdir(parents=True, exist_ok=True)  # make dir
    last = wdir / 'last.pt'
    best = wdir / 'best.pt'
    results_file = save_dir / 'results.txt'

    # Save run settings
    with open(save_dir / 'hyp.yaml', 'w') as f:
        yaml.safe_dump(hyp, f, sort_keys=False)
    with open(save_dir / 'opt.yaml', 'w') as f:
```

```
        yaml.safe_dump(vars(opt), f, sort_keys=False)

    # Configure
    plots = not opt.evolve  # create plots
    cuda = device.type != 'cpu'
    init_seeds(2 + rank)
    with open(opt.data) as f:
        data_dict = yaml.safe_load(f)  # data dict
    is_coco = opt.data.endswith('coco.yaml')

    # Logging- Doing this before checking the dataset. Might update
    data_dict
    loggers = {'wandb': None}  # loggers dict
    if rank in [-1, 0]:
        opt.hyp = hyp  # add hyperparameters
        run_id = torch.load(weights).get('wandb_id') if weights.
endswith('.pt') and os.path.isfile(weights) else None
        wandb_logger = WandbLogger(opt, save_dir.stem, run_id,
data_dict)
        loggers['wandb'] = wandb_logger.wandb
        data_dict = wandb_logger.data_dict
        if wandb_logger.wandb:
            weights, epochs, hyp = opt.weights, opt.epochs, opt.hyp
      # WandbLogger might update weights, epochs if resuming

    nc = 1 if opt.single_cls else int(data_dict['nc'])  # number of
    classes
    names = ['item'] if opt.single_cls and len(data_dict['names'])
!= 1 else data_dict['names']  # class names
    assert len(names) == nc, '%g names found for nc=%g dataset in %
s' % (len(names), nc, opt.data)  # check

    # Model
    pretrained = weights.endswith('.pt')
    if pretrained:
        with torch_distributed_zero_first(rank):
```

```
46            attempt_download(weights)  # download if not found
    locally
47        ckpt = torch.load(weights, map_location=device)  # load
    checkpoint
48        model = Model(opt.cfg or ckpt['model'].yaml, ch=3, nc=nc,
    anchors=hyp.get('anchors')).to(device)  # create
49        exclude = ['anchor'] if (opt.cfg or hyp.get('anchors')) and
     not opt.resume else []  # exclude keys
50        state_dict = ckpt['model'].float().state_dict()  # to FP32
51        state_dict = intersect_dicts(state_dict, model.state_dict()
    , exclude=exclude)  # intersect
52        model.load_state_dict(state_dict, strict=False)  # load
53        logger.info('Transferred %g/%g items from %s' % (len(
    state_dict), len(model.state_dict()), weights))  # report
54    else:
55        model = Model(opt.cfg, ch=3, nc=nc, anchors=hyp.get('
    anchors')).to(device)  # create
56    with torch_distributed_zero_first(rank):
57        check_dataset(data_dict)  # check
58    train_path = data_dict['train']
59    test_path = data_dict['val']
```

Listing A.1: The Code for the Training of the Model

A.2 Model Testing

This part consists of the models testing after it has been trained.

```
def test(data,
         weights=None,
         batch_size=32,
         imgsz=640,
         conf_thres=0.001,
         iou_thres=0.6,  # for NMS
         save_json=False,
         single_cls=False,
         augment=False,
         verbose=False,
         model=None,
         dataloader=None,
         save_dir=Path(''),  # for saving images
         save_txt=False,  # for auto-labelling
         save_hybrid=False,  # for hybrid auto-labelling
         save_conf=False,  # save auto-label confidences
         plots=True,
         wandb_logger=None,
         compute_loss=None,
         half_precision=True,
         is_coco=False,
         opt=None):
    # Initialize/load model and set device
    training = model is not None
    if training:  # called by train.py
        device = next(model.parameters()).device  # get model
   device

    else:  # called directly
        set_logging()
        device = select_device(opt.device, batch_size=batch_size)

        # Directories
```

```
33      save_dir = increment_path(Path(opt.project) / opt.name,
    exist_ok=opt.exist_ok)  # increment run
34      (save_dir / 'labels' if save_txt else save_dir).mkdir(
    parents=True, exist_ok=True)  # make dir
35
36      # Load model
37      model = attempt_load(weights, map_location=device)  # load
    FP32 model
38      gs = max(int(model.stride.max()), 32)  # grid size (max
    stride)
39      imgsz = check_img_size(imgsz, s=gs)  # check img_size
40
41      # Multi-GPU disabled, incompatible with .half() https://
    github.com/ultralytics/yolov5/issues/99
42      # if device.type != 'cpu' and torch.cuda.device_count() >
    1:
43      #     model = nn.DataParallel(model)
44
45   # Half
46   half = device.type != 'cpu' and half_precision  # half
    precision only supported on CUDA
47   if half:
48      model.half()
49
50   # Configure
51   model.eval()
52   if isinstance(data, str):
53      is_coco = data.endswith('coco.yaml')
54      with open(data) as f:
55          data = yaml.safe_load(f)
56   check_dataset(data)  # check
57   nc = 1 if single_cls else int(data['nc'])  # number of classes
58   iouv = torch.linspace(0.5, 0.95, 10).to(device)  # iou vector
    for mAP@0.5:0.95
59   niou = iouv.numel()
60
61   # Logging
```

```
62      log_imgs = 0
63      if wandb_logger and wandb_logger.wandb:
64          log_imgs = min(wandb_logger.log_imgs, 100)
```

Listing A.2: The Code for the Testing of the Model

A.3 YOLOv5 Detect Class

Depending on the architecture of the model selected, different configurations are applied respectively. PyTorch use can be easily viewed below.

```
class Detect(nn.Module):
    stride = None  # strides computed during build
    onnx_dynamic = False  # ONNX export parameter

    def __init__(self, nc=80, anchors=(), ch=(), inplace=True):  # detection layer
        super(Detect, self).__init__()
        self.nc = nc  # number of classes
        self.no = nc + 5  # number of outputs per anchor
        self.nl = len(anchors)  # number of detection layers
        self.na = len(anchors[0]) // 2  # number of anchors
        self.grid = [torch.zeros(1)] * self.nl  # init grid
        a = torch.tensor(anchors).float().view(self.nl, -1, 2)
        self.register_buffer('anchors', a)  # shape(nl,na,2)
        self.register_buffer('anchor_grid', a.clone().view(self.nl, 1, -1, 1, 1, 2))  # shape(nl,1,na,1,1,2)
        self.m = nn.ModuleList(nn.Conv2d(x, self.no * self.na, 1) for x in ch)  # output conv
        self.inplace = inplace  # use in-place ops (e.g. slice assignment)

    def forward(self, x):
        # x = x.copy()  # for profiling
        z = []  # inference output
        for i in range(self.nl):
            x[i] = self.m[i](x[i])  # conv
            bs, _, ny, nx = x[i].shape  # x(bs,255,20,20) to x(bs,3,20,20,85)
            x[i] = x[i].view(bs, self.na, self.no, ny, nx).permute(0, 1, 3, 4, 2).contiguous()

            if not self.training:  # inference
```

```
27                 if self.grid[i].shape[2:4] != x[i].shape[2:4] or
   self.onnx_dynamic:
28                     self.grid[i] = self._make_grid(nx, ny).to(x[i].
   device)
29
30                 y = x[i].sigmoid()
31                 if self.inplace:
32                     y[..., 0:2] = (y[..., 0:2] * 2. - 0.5 + self.
   grid[i]) * self.stride[i]  # xy
33                     y[..., 2:4] = (y[..., 2:4] * 2) ** 2 * self.
   anchor_grid[i]  # wh
34                 else:  # for YOLOv5 on AWS Inferentia https://
   github.com/ultralytics/yolov5/pull/2953
35                     xy = (y[..., 0:2] * 2. - 0.5 + self.grid[i]) *
   self.stride[i]  # xy
36                     wh = (y[..., 2:4] * 2) ** 2 * self.anchor_grid[
   i].view(1, self.na, 1, 1, 2)  # wh
37                     y = torch.cat((xy, wh, y[..., 4:]), -1)
38                 z.append(y.view(bs, -1, self.no))
39
40         return x if self.training else (torch.cat(z, 1), x)
41
42     @staticmethod
43     def _make_grid(nx=20, ny=20):
44         yv, xv = torch.meshgrid([torch.arange(ny), torch.arange(nx)
   ])
45         return torch.stack((xv, yv), 2).view((1, 1, ny, nx, 2)).
   float()
```

Listing A.3: The Code for the YOLOv5 Model Class Detect

A.4 Convolution Class

The models one of the most common and used class, Convolution class is below:

```
class Conv(nn.Module):
    # Standard convolution
    def __init__(self, c1, c2, k=1, s=1, p=None, g=1, act=True):  # ch_in, ch_out, kernel, stride, padding, groups
        super(Conv, self).__init__()
        self.conv = nn.Conv2d(c1, c2, k, s, autopad(k, p), groups=g, bias=False)
        self.bn = nn.BatchNorm2d(c2)
        self.act = nn.SiLU() if act is True else (act if isinstance(act, nn.Module) else nn.Identity())

    def forward(self, x):
        return self.act(self.bn(self.conv(x)))

    def fuseforward(self, x):
        return self.act(self.conv(x))
```

Listing A.4: The Code for the standard convolution class

References

[1] B. S. Faiçal, F. G. Costa, and Pessin, "The use of unmanned aerial vehicles and wireless sensor networks for spraying pesticides," *Journal of Systems Architecture*, vol. 60, no. 04, pp. 393–404, 2014.

[2] J. Savolski, "The history of pest control," 2020. [Online]. Available: https://ameritechpest.com/the-history-of-pest-control.html

[3] K. Frenken, "Irrigation in southern and eastern asia in figures: Aquastat survey - 2011." *Water Reports*, no. No.37, pp. xix + 487 pp., 2012.

[4] S. M. B. Hannah Ellis-Petersen, "Many will starve': locusts devour crops and livelihoods in pakistan," 2020, [Online]. Available: https://www.theguardian.com/world/2020/may/25/many-will-starve-locusts-will-starve-locusts-devour-crops-and-livelihoods-in-pakistan.

[5] X. Li, Y. Li, and S. Li, "Recent Advances of Generic Object Detection with Deep Learning: A Review," in *Communications in Computer and Information Science*, 2020, pp. 185–193.

[6] F. Chollet *et al.*, *Deep learning with Python*. Manning New York, 2018, vol. 361.

[7] W. Peng, N. L. Ma, and Zhang, "A review of historical and recent locust outbreaks: Links to global warming, food security and mitigation strategies," *Environmental Research*, vol. 191, p. 110046, 2020. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0013935120309439

[8] M. A. MinÃă Dowlatchahi and K. Cressman, "Desert lo-

cust situation in pakistan," 2017. [Online]. Available: http://www.fao.org/pakistan/resources/in-depth/desert-locust-situation-in-pakistan/en/

[9] S. K. Bhoi, K. K. Jena, and Panda, "An Internet of Things assisted Unmanned Aerial Vehicle based artificial intelligence model for rice pest detection," *Microprocessors and Microsystems*, vol. 80, p. 103607, 2021. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0141933120307560

[10] A. Matese and D. G. Toscano, Piero, "Intercomparison of UAV, aircraft and satellite remote sensing platforms for precision viticulture," *Remote Sensing*, vol. 7, no. 3, pp. 2971–2990, 2015. [Online]. Available: https://www.mdpi.com/2072-4292/7/3/2971

[11] P. Viola and M. Jones, "Rapid object detection using a boosted cascade of simple features," in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, vol. 1, 2001, pp. I–I.

[12] R. Girshick, D. Jeff, D. Trevor, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation Tech report (v5)," *Computer Vision and Pattern Recognition*, pp. 580–587, 2014.

[13] R. Girshick, "Fast R-CNN," in *Proceedings of the IEEE International Conference on Computer Vision*, ICCV, 2015, pp. 1440–1448.

[14] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 39, no. 6, pp. 1137–1149, 2017.

[15] K. He, G. Gkioxari, P. Dollár, and R. Girshick, "Mask R-CNN," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pp. 2961–2969, 2020.

[16] J. Pang, K. Chen, and Shi, "Libra R-CNN: Towards balanced learning for object detection," in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, CVPR, 2019, pp. 821–830.

[17] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, CVPR, 2016, pp. 779–788.

[18] J. Redmon and A. Farhadi, "YOLO9000: Better, faster, stronger," in *Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017*, 2017, pp. 7263–7271.

[19] R. Joseph and A. Farhadi, "YOLOv3: An incremental improvement," *arXiv:1804.02767*, 2018.

[20] W. Liu, D. Anguelov, and Erhan, "SSD: Single shot multibox detector," in *Computer Vision – ECCV 2016*. Cham: Springer International Publishing, 2016, pp. 21–37.

[21] T. Y. Lin, P. Goyal, and Girshick, "Focal Loss for Dense Object Detection," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pp. 2980–2988, 2020.

[22] C. Zhang and J. M. Kovacs, "The application of small unmanned aerial systems for precision agriculture: A review," *Precision Agric*, vol. 13, pp. 693–712, 2012.

[23] M. J. Shafiee, B. Chywl, F. Li, and A. Wong, "Fast yolo: A fast you only look once system for real-time embedded object detection in video," pp. arXiv–1709, 2017.

[24] J. Solawetz, "Yolov5 new version - improvements and evaluation," Jun. 29, 2020. [Online]. Available: https://blog.roboflow.com/yolov5-improvements-and-evaluation/

[25] S. Caldera, A. Rassau, and D. Chai, "Review of deep learning methods in robotic grasp detection," pp. 57–60, 2018.

[26] A. Kamilaris and F. X. Prenafeta-BoldÃž, "Deep learning in agriculture: A survey," *Computers and Electronics in Agriculture*, vol. 147, pp. 70–90, 2018. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0168169917308803

[27] R. Laroca, E. Severo, L. A. Zanlorensi, L. S. Oliveira, G. R. Gonçalves, W. R. Schwartz, and D. Menotti, "A robust real-time automatic license plate recognition based on the yolo detector," in *2018 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2018, pp. 1–10.