PERFORMANCE EVALUATION OF SELECTED DEEP LEARNING ALGORITHMS IN AUTOMATIC WEED DETECTION SYSTEM

BY

ASHI, John
MTech/SET/2019/9802

DEPARTMENT OF SURVEYING AND GEOINFORMATICS,
FEDERAL UNIVERSITY OF TECHNOLOGY
MINNA

JUNE, 2023

PERFORMANCE EVALUATION OF SELECTED DEEP LEARNING ALGORITHMS IN AUTOMATIC WEED DETECTION SYSTEM

BY

ASHI, John
MTech/SET/2019/9802

A THESIS SUBMITTED TO THE POSTGRADUATE SCHOOL, FEDERAL UNIVERSITY OF TECHNOLOGY MINNA, IN PARTIAL FULFILMENT OF THE REQUIREMENTS FOR THE AWARD OF THE DEGREE OF MASTER OF TECHNOLOGY (M.Tech) IN SURVEYING AND GEOINFORMATICS

JUNE, 2023

ABSTRACT

Site-specific weed detection and management in agrarian lands is a crucial approach for crop productivity management and chemical contamination mitigation in precision agriculture. Traditional ways of executing this operation is expensive and labour intensive, as well as exposing personnel to the danger of exposure to hazardous chemicals. To create a more sustainable agricultural system, a program for automatically detecting agricultural weeds in a mixed farmland using the Faster RCNN inception v2 model and YOLOv5s neural network, was proposed. With the introduction of Unmanned Aerial Vehicles (UAV) and technological advancements in Deep Learning techniques in recent years, it has become possible to identify and classify weeds from crops at desired spatial and temporal resolution. A DJI Phantom 4 UAV was used to simultaneously collect about 254 image pairs of a mixed-crop farmland. The proposed approach for Faster RCNN involves labelling or annotating the images before uploading the dataset into an online Graphic Processing Unit (GPU) known as Google Colaboratory (Colab) which runs on a Python programming language, where the dataset were trained over five epochs (10,000, 20,000, 100,000, 200,000, and 242,000) to get the maximum epoch where the model flattens out using Python programming codes and tested on the testing dataset for the automatic identification and classification of weeds. Also, the YOLO v5 neural network was trained over 100, 300, 500, 600, 700 and 1000 epochs and this was also implemented on Colab using python programming language. Both neural network algorithms identified and classified five classes which are as follows: sugarcane, spinach, banana, pepper and weeds. The utilized classifiers' overall classification accuracy differed widely. Faster RCNN exhibited the highest overall accuracies. Notably lower accuracies were observed using YOLOv5. The lowest accuracies were achieved at 10,000 epochs with an overall accuracy of 52%, weed precision of 50%, and weed recall of 8%, while the highest level of accuracies and saturation point were achieved at 200,000 epochs with 98% overall accuracy, 98% weed precision, and 99% weed recall. The minimum epoch of YOLOv5s classification at 100 epochs achieved the overall accuracy of 16 %, weed precision of 5 % and 1% for the weed recall. Furthermore, the classifier achieved a maximum weed precision at 600 epochs with a weed precision of 78 %, weed recall of 34 % and an overall accuracy of 67 %. With only 16 % and 66% overall accuracy of YOLOv5s, the Faster RCNN Deep Learning exhibited a better classification output, making it a better classifier suitable for automatic weed identification and classification, and it is thus recommended. Further research should be carried out to further compare the performance of Faster RCNN inception v2 model with a few other recent powerful Deep Learning algorithms to increase or strengthen weed detection on small farmlands. Also, images should be taken at a flying height less than 30m and closer for smaller weeds so they appear larger in the image.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF PLATES

LIST OF ABBREVIATIONS

ANN	Artificial Neural Network
CNN	Convolutional Neural Network
DL	Deep Learning
DoD	Department of Defence
FAO	Food and Agricultural AOrganization
GCP	Ground Control Point
GIS	Geographic Information System
GPS	Global Positionning System
GPU	Graphical Processing Unit
Knn	k-Nearest Neighbor
ML	Machine Learning
NN	Neural Network
OBIA	Object Based Image Analysis
PA	Precision Agriculture
RCNN	Region-based Convolutional Neural Network
RF	Random Forest
RoI	Region of Interest
RPN	Region Proposal Networks
RS	Remote Sensing
RTK	Real Time Kinematics
SSWM	Site-Specific Weed Management
SVM	Support Vector Machine
UAV	Unmanned Aerial Vehicle
YOLO	You Only Look Once

CHAPTER ONE

1.0 INTRODUCTION

1.1 Background to the Study

It is predicted that the world population will grow to an all-time high of 10 billion in the year 2050 according to Alexandratos & Bruinsma (2012), but due to the currently implemented agricultural production method and other environmental factors affecting production rate, it will be nearly impossible to achieve the predicted increase in agricultural produce demand. Hence, the need for the introduction of innovative ways and systems of boosting the production rate of agriculture while minimizing the effect of these environmental factors. One of such innovative systems is the Precision Agriculture.

The idea of Precision Agriculture (PA), also known as smart farming, has been discussed in the agricultural sector as a management method since the middle of the 1980s. Later, throughout the last two decades, Precision Agriculture was ranked among the top 10 agricultural sector breakthroughs (Crookston, 2006). Precision Agriculture is a systematic technique and also a management system of using the proper quantity of input (such as compost/fertilizer, water, and herbicides) at just the appropriate time and place to increase productivity and minimize chemical use in order to protect the environment from pollution (Zhang & Kovacs, 2012; Torres-Sánchez *et al.*, 2013; Adekunle, 2013; Yao & Huang, 2013; Huang & Thomson, 2015).

The ability of any farmer to adopt any highest quality judgment at the appropriate moment for the best location of the farm depends on the ability to gather and interpret various types of information (Mulla, 2013). As a result, techniques such as GPS/GNSS devices

and Remote Sensing (RS) are being deployed both as information sources and as tools for carrying out various PA operations. For instance, RS technologies demonstrate a significant capacity to offer the farmer useful information by utilizing satellite or aerial vehicles for various imaging solutions. These technologies quickly capture photos over a vast region (Zhang & Kovacs, 2012). Farmers can therefore utilize the photographs gathered to assess crop strain, track crop yields, or forecast crop production. Although these imaging systems provide farmers a lot of knowledgeable power, their poor spatial resolution places restrictions on them. Therefore, to give greater in-depth picture data, terrestrial RS systems were deployed. Regrettably, these methods lack the time necessary to completely survey the vast agricultural areas. As a result, a different platform was required which could bridge the barrier amongst aerial and also terrestrial remote sensing systems. The Unmanned Aerial Vehicle (UAV) displayed a robust ability to fill this shortfall (Pena *et al.*, 2013).

UAVs have demonstrated their versatility over the last several years by serving as platforms for various sensors including lidar, GNSS cameras, RGB and thermal sensors (Nex & Remondino, 2014). Broadly speaking, the UAV imaging system is viewed as a more affordable substitute to traditional remote sensing platforms that rely on satellites or aircraft. Furthermore, due to the fact that UAVs fly at lower altitudes than satellites and other aerial platforms, their imaging systems can offer superior spatial resolution (Grenzdörffer *et al.*, 2008). Therefore, UAV imaging technologies can be employed for many precision agriculture operations such as plant health tracking (McCabe *et al.*, 2015), weeds control (Hervás Martínez *et al.*, 2015; Pena *et al.*, 2015; Hassanein & El-sheimy, 2017), and plant row identification (Slaughter *et al.*, 2008).

Amongst the most common deployment of Unmanned Aerial Vehicles (UAVs) in PA is the mapping and controlling of weeds. Distinct methods for weed classification using UAV photographs platform have indeed been explored as they demonstrates the potentials of bridging the differences amongst terrestrial and aerial imaging platforms (Hassanein & El-Sheimy, 2017; Mulla, 2013; Hervás Martínez *et al.*, 2015; Pena *et al.*, 2015; Torres-Sánchez *et al.*, 2013). Agricultural output and viability can be significantly impacted by biological hazards which could be bacteria, viruses, weeds, fungi, and insects. Weeds constitute the most serious issue amongst these, contributing to a significant damage to crops on a global scale (Esposito *et al.*, 2021). In essence, weeds are uncultivated plants which thrives within agricultural fields as well as contend with agricultural crops for natural resources such as water, manure, growing spaces, and also sunshine (Hassanein & El-sheimy, 2017; Monteiro & Santos, 2022). Thus, it is critical to get rid of such weeds as promptly as humanly possible so that the planted vegetation may get the right quantity of nutrients to improve the output quality and quantity of the farm area.

Nonetheless, one of most popular weed management techniques since the advent of agriculture have been human weeding, mechanized weeding, and herbicide sprays (Griepentrog & Dedousis, 2010; Bergin, 2011; Rueda-Ayala *et al.*, 2011; Chauvel *et al.*, 2012). Medieval weed management techniques included hand pulling, chopping, or physically covering weeds (Young *et al.*, 2014). Hand implements were fabricated across history to cultivate soils and eradicate weeds (Jabran *et al.*, 2015). While these weed management techniques significantly increase agricultural output, they are not without their share of difficulties. The main difficulties in hand weeding include declining available labor, rising labor costs, and uneven weed management (Carballido *et al.*, 2013; Gianessi, 2013). In a related manner, mechanical weed management necessitates

increased soil turnover, which can disrupt morphology of the soil and reduce nutrients of the soil (Smith *et al.*, 2011). The effectiveness, cost, and longevity of mechanical weed management are not always good (Bond & Grundy, 2001). Also, Agrochemicals have been employed recently in cubbing the spread of weeds (Peterson *et al.*, 2018). The main obstacles to routinely utilizing herbicides for weed management include herbicide-resistant weeds, adverse health consequences, and ecological pollution (Annett *et al.*, 2014; Hoppin, 2014; Starling *et al.*, 2014). It is crucial to diversify existing contemporary weed management techniques due to the difficulties regarding traditional weed control strategies, such as hand weeding, mechanical weed management and pesticides (Westwood *et al.*, 2018).

As a result, Agriculture has developed to use resources in a way that is considerably more productive and cost-effective. Hence, Site-Specific Weed Management (SSWM), a Precision Agricultural management system, was developed for efficient weed control (Pena-Barragán *et al.*, 2012). The ability to accurately pinpoint and identify weeds is a necessary initial step in the development of an autonomous weed management system (Liu & Bruch, 2020).

The four main processes of a standard weed detection system are image acquisition, pre-processing of photographs, feature extraction, identification, and categorization of weed plants (Shanmugam *et al.*, 2020). These phases have been completed by the use of several advancing technologies. The identification and classification of weeds is the phase that is most important. The automatic detection of weed species has been more popular in recent years as a result of the advancements in digital technology, notably in Graphics Processing Unit (GPU), the application of Machine Learning (ML) approaches and embedded processors (Gu *et al.*, 2018; LeCun *et al.*, 2015; Yu *et al.*, 2019).

In subsequent times, Deep Learning models have developed as more precise and effective substitutes for conventional parametric algorithms, particularly for large and extremely complicated data (Rodriguez-Galiano *et al.*, 2012). A key area of machine learning (ML) is Deep Learning (DL). DL models do have several benefits in comparison to conventional ML models for photograph categorization and object detection, such as the Support Vector Machine (SVM) classifier (Sabat-Tomala *et al.*, 2020), Object Based Image Analysis (OBIA) (Torres-Sanchez *et al.*, 2015), k-Nearest Neighbor (kNN) (Kramer, 2013), the Random Forest algorithm (RF) (Liu *et al.*, 2012), considering crops and weeds can sometimes be identical, it can be challenging to extract and choose differentiating features using Machine Learning techniques. Premised on its extremely effective learning abilities, DL approaches can effectively solve this challenge.

Convolutional Neural Networks (CNN) have been discovered to operate excellently well throughout vision - based applications ranging from categorization, forecasting, and object identification, thanks to advances in parallel computing as well as the utilization of larger datasets (Krizhevsky *et al.*, 2012). The typical pixel-based technique requires pixel-level computational analysis and mainly focuses on spectral characteristics, ignoring the possibilities of geographical and textural variables to increase accuracy of the model (Blaschke, 2010). Convolutional Neural Networks, on the other hand, which consider spectral, textural, and spatial characteristics of pictures, have recently evolved, allowing for higher classification accuracy and with increase in computational power (especially Graphical Processing Units, GPUs) (Nogueira *et al.*, 2017; Cevallos *et al.*, 2019; Sharma *et al.*, 2017a). A Deep Neural Network with a convolutional structure is known as a Convolutional Neural Network. It basic premise is to include convolutional operations into neural networks in an attempt to solve the inadequacies of the original

neural networks with a large number of variables (Chang *et al.*, 2016). Additionally, it may retrieve more detailed information and mitigate the issue of overfitting in typical neural networks. In this research; the Faster RCNN and the YOLO v5 algorithms were implemented and compared for performance accuracy.

The methodology for detecting objects using the Faster RCNN is built upon a region proposal approach. The original region-based methodological framework was called the Region based Convolutional Neural Network (RCNN) (Girshick *et al.*, 2015). Moreover, it required a lot of computing work because each suggested location required a CNN-based feature extraction. Through distributing convolutional features across several area suggestions, a Fast RCNN was developed to cut down on computing time (Girshick, 2015). Employing Fully Convolutional Region Proposal Networks (RPN), which are taught to suggest improved object regions, Faster Region-based Convolutional Neural Network was developed to increase speed (Ren *et al.*, 2015).

Secondly, YOLO is an abbreviation for "You Only Look Once" in English. YOLO version 5 is a newer version of the You Only Look Once class algorithm, which is an advanced object detector that does exceptional real time object identification (Malta *et al.*, 2021; Francies *et al.*, 2022; Thuan, 2021; Reddy & Panicker, 2021). This can categorize any imagery into a group as well as identify many objects inside a picture (Jabir & Falih, 2022; Yang *et al.*, 2021). This is among the fastest possible algorithms which employs Convolutional Neural Network (CNN) for object identification integrated bounding box predictions and object recognition into a unified end-to-end discrete network.

The aforementioned CNN architectures (YOLO V5 and Faster RCNN with inception v2) were implemented on the images of a mixed cropping farm for an automated

identification and classification of weeds from four (4) different crop classes taking into consideration the significant influence of various training iterations or epochs on the overall performance evaluation of the weed identification and classification scheme and also defining the optimum and minimal training epoch for the classification algorithm. Five varying epochs were tried to determine the ideal training epoch for the Faster RCNN model, that indicates the maximal point of the training phase in which the model tends to flatten out: 10,000, 20,000, 100,000, 200,000, and 242,000. For YOLOv5 model, six varying epochs were also tested which includes 100, 300, 500, 600, 700 and 1000 epochs.

1.2 Statement of Research Problem

To fulfill the needs of an ever-increasing population, Agricultural production will need to increase food production from subsistence Agriculture in the next decade via more effective utilisation of natural resources with little environmental damage (Hobbs *et al.*, 2008).

According to the Food and Agricultural Organization (FAO), food productivity must increase by 70%, with the majority of this coming from improved yields per hectare of Agricultural land (McFadyen, 2012). Reduced output losses due to pests, namely weeds, are a serious concern for agricultural output (Popp *et al.*, 2013).

Worldwide, 40% agricultural productivity losses are attributed to weeds, despite farmer control methods (Vila *et al.*, 2004). The damages would be total when no intervention is done to safeguard crops against weeds (Chauhan, 2020). In the developed countries, weeds account for about 5% loss in agricultural production, while it accounts for 10% and 25% in less developed and least developed countries (Vissoh *et al.*, 2004). Agricultural farmers in developed countries devote more resources on weed management compared to any other pest (Akobundu, 1987).

Subsistence farming is the most common type of agriculture in underdeveloped countries, and weeds are typically controlled by hand-weeding. Hand implements were fabricated across history to cultivate soils and eradicate weeds (Jabran *et al.*, 2015). However, due to expanding urban development, rising labor expenses, and a shrinking Agricultural manpower, many are turning to herbicides to control weeds.

As a result, the indiscriminate utilization of pesticides for weed management in subsistence farming operations has increased, raising health and environmental problems (Tirado *et al.*, 2008; Gianessi, 2013). Also, the mechanical weed management necessitates increased soil turnover, which can disrupt morphology of the soil and reduce nutrients of the soil (Smith *et al.*, 2011).

It is crucial to diversify existing contemporary weed management techniques due to the difficulties regarding traditional weed control strategies, such as hand weeding, mechanical weed management and pesticides to a more robust and real-time system (Westwood *et al.*, 2018).

Subsequently, Deep Learning models have developed as a more precise and effective substitutes for conventional parametric algorithms, particularly for large and extremely complicated data (Rodriguez-Galiano *et al.*, 2012). A key area of Machine Learning (ML) is Deep Learning (DL). DL models do have several benefits compared to conventional ML techniques for the classification of imagery, object identification and recognition, and these ML algorithms includes the Support Vector Machine (SVM) classifier (Sabat-Tomala *et al.*, 2020), Object Based Image Analysis (OBIA) (Torres-Sanchez *et al.*, 2015), k-Nearest Neighbor (kNN) (Kramer, 2013), the Random Forest algorithm (RF) (Liu *et al.*, 2012). Considering crops and weeds can sometimes be identical, it can be challenging to extract and choose differentiating features using Machine Learning techniques.

To diversify this existing contemporary method, this project research incorporates the use of Unmanned Aerial Vehicle (UAV) automation and Deep Learning algorithms (Faster RCNN and YOLO). Hence, the need to evaluate the two Deep Learning algorithms since the performance accuracy depends on the choice of algorithm utilized.

1.3 Research Questions

Consequent upon the objectives, this study will provide answers to the following questions:

1. What method is used to define the spatial extent of the study area?
2. What is the measure of accuracy obtainable from the Faster RCNN algorithm in automatic weed detection?
3. What is the classification performance of YOLO v5 algorithm on weed detection?
4. How efficient is the FRCNN compared to YOLO v5 in automatic weed detection?

1.4 Aim and Objectives of the Study

The aim of this research is to evaluate the performance of Faster RCNN and YOLO v5 algorithms in precision weed mapping utilizing photographs taken by an Unmanned Aerial Vehicle (UAV). In achieving the identified aim, the objectives to be pursued are:

1. UAV mapping of the mixed-crop farmland.
2. To implement Faster Region based Convolutional Neural Network algorithm for automatic weed classification.
3. To implement YOLO v5 algorithm for automatic weed classification.
4. To carryout a perfomance evaluation of the results obtained from varying training epochs in objective 2 and objective 3.

1.5 Scope of Study

This research focuses mainly on the use of a phantom 4 UAV having a Red Green Blue sensor for acquiring data over a mixed crop farmland located in Lapan Gwari under Bosso Local Government Area of Niger State, which aided the automatic detection of weeds.

The dataset acquired from the UAV shall cover the entire farm and the processes involved in preprocessing and processing shall be covered.

This research further covers the implementation of the Faster Region Based Convolutional Neural Network and You Only Look Once algorithms as well as the training, testing, and validation of the developed model utilizing python programming language codes on the google colaboratory interface. Finally, the performance of the selected Deep Learning algorithms were evaluated.

1.6 Limitations

Due to the fact that this research was conducted during the dry season, it was a little bit difficult getting an irrigated farm having a substantial amount of weed density. Also, the successful processing and training of the Deep Learning models largely depended on the properties of the Central Processing Unit (CPU) of the computer system. The CPU of the system which was available to the author was inadequate to fully and successfully run the model hence the switch to an online Graphic Processing Unit (GPU) on Google Colaboratory pro with a RAM size of 32 GB and a runtime usage of 24hours that attracts a fee of 9.99$.

1.7 Significance of Study

The findings of this study will aid farmers to identify or determine the locations of weed clusters in the agricultural field and subsequently give the farmer a chart of the weed sites gotten from the comparism of Faster RCNN and YOLO v5 algorithms. UAVs can map

the farm and provide information about weed patches by instantly surveying wide swaths of farm land. The capacity to gather and evaluate this data in real time will result in higher crop yields, less money spent on weeds herbicides and pesticides, and better management decisions overall.

As UAVs fly at a relatively low height than satellites as well as other aerial operating systems, they produce images with a better spatial resolution. Additionally, Unmanned Aerial Vehicle systems enable users to gather visual data having excellent temporal resolution that can increase the adaptability of the data collecting processes.

Faster RCNN and YOLO v5 algorithms can recognize and categorize weeds in a non-destructive manner which will aid in site specific weed management and will allow farmers to be more aware of weed growth and distribution around the farmland while also arming them with site specific knowledge pertaining to weed development, control and mitigation. This is significant because, if the locations of weed patches are identified, they can be managed precisely and effectively.

It is hoped that the findings of the research presented in this dissertation will help farmers understand the importance and the applicability of UAVs in Precision Agriculture in Nigeria. In general, greater production and cost savings will be realized through more efficient herbicides and fertilizer usage.

1.8 Study Area

The research was done during the dry season of 2022 within a mixed cropping farm. With a coverage of 2.8228 hectares, the privately owned farm land located at Lapan Gwari, Minna, Niger State located within geographical coordinates (9°31'33"N 6°30'02"E), (9º31'34"N 6º29'59"), (9º31'38"N 6º30'03"E) and (9º31'37"N 6º30'05"E), under Bosso

LGA area is situated at about 7km away from F.U.T Minna permanent site (Gidan Kwanu campus). The natives are Gwaris and they depend solely on agricultural practices such as crop cultivation and fish farming. The natives mostly practice mixed cropping such as, pepper, vegetables, sugarcane, rice maize and yams. The study site is generally made up of loamy soil, and it is connected to a pumping machine for proper supply of water, alongside the cultivation of spinach, pepper, banana and sugarcane. Figure 1.2 shows the map of Niger State extracted from the map of Nigeria, from which the bosso local government area under Minna is identified then finally, the site location map is displayed.

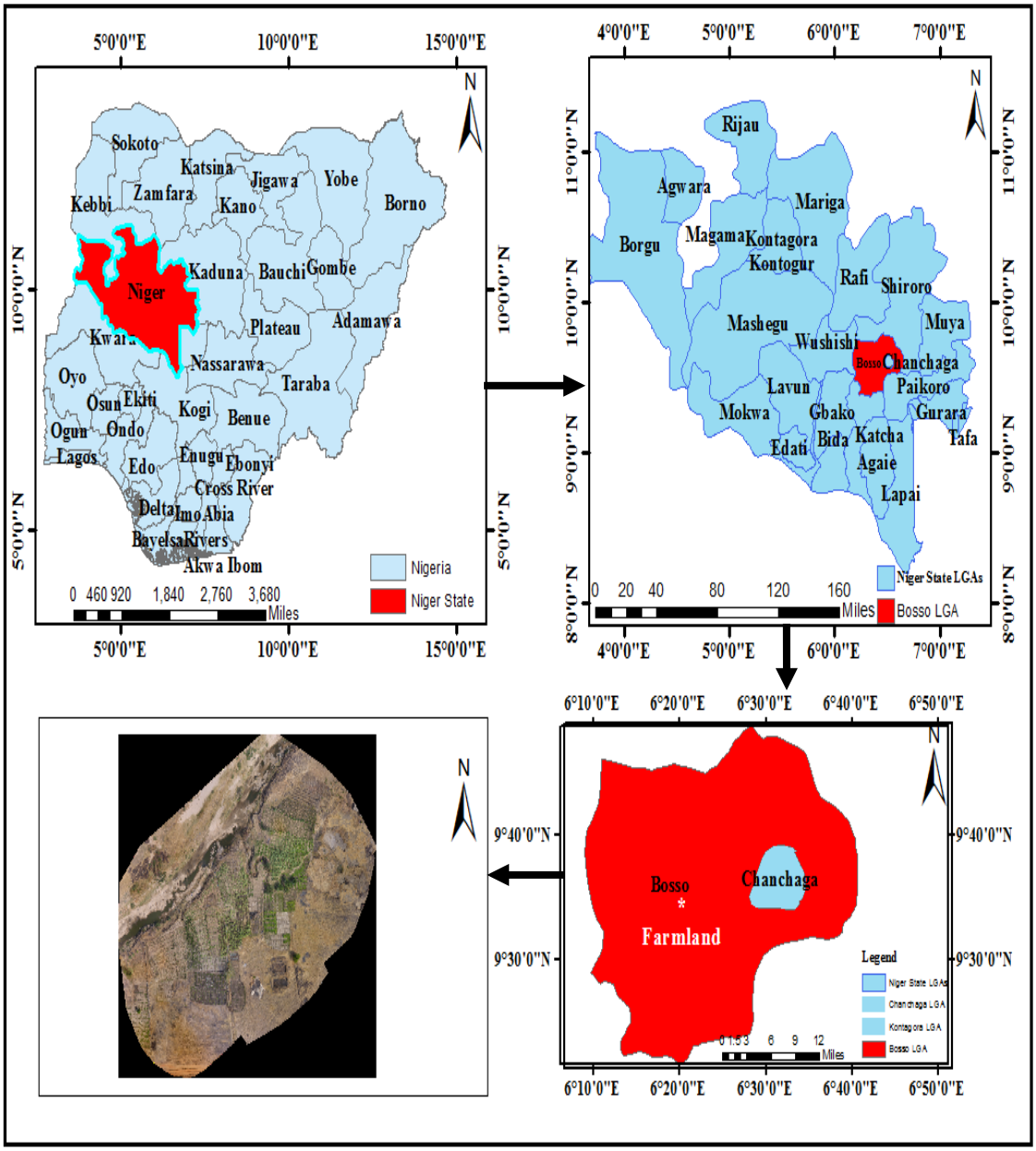

Figure 1.1: Geographic description of the study area

CHAPTER TWO

2.0 LITERATURE REVIEW

2.1 Precision Agriculture

The precise implementation of Agricultural inputs in accordance with inherent spatial variability, soil and weather patterns, and crop prerequisites for improving crop yield, revenue growth, sustainable development, environmental preservation, and quality of products may be a more appropriate description of Precision Agriculture in the Nigerian context (Patil-Shirish & Bhalerao, 2013). It refers to the use of ICT (Information, Communication, and Technology) to control field variability (Gemtos *et al.,* 2013). Precision Agriculture, as described by the International Society of Precision Agriculture, is a management approach which "collects, processes, and evaluates temporal, spatial, and individual data and intermixes it with additional information to support managerial decisions pertaining to estimated variability for improved resource use efficiency, productivity, quality, revenue growth, and sustainable development of agricultural production" (Shrestha & Khanal, 2020).

2.1.1 Precision agriculture tools

Using a wide range of implements, comprising software, hardware, and best management techniques, Precision Agriculture is a highly advanced agricultural method that necessitates technological proficiency (Shrestha & Khanal, 2020). The accompanying sections provides a basic description of these tools.

2.1.1.1 *Global positioning system (GPS)*

The primary idea behind Precision Agriculture is location specificity (Sahu *et al.*, 2019). A satellite-based radio navigation system predominantly known as the Global Positioning System (GPS) delivers 3D location information (latitude, longitude, and elevation) having

precision ranging from 100 to 0.01 m at any time, in any condition, and at no cost. It is made up of a whole set of 24 satellites that are circling the earth in a planned sequence that is kept up by the US Department of Defense (DoD) (Yousefi & Razdari, 2015). According to Brejda *et al.* (2000), GPS enables farmers to keep a close eye on crop health, macro and micro-scale spatiotemporal heterogeneity of the soil, and to pinpoint the precise location of field features like the field boundaries, acreage for field crops, soil composition, pest presence, disease-affected areas and weed infestation. This enables the implementation of key inputs such as (seeds, fertilizers, pesticides, herbicides, water, etc.) depending on prior input data and effectiveness requirements (Batte & Van-Buren, 1999).

2.1.1.2 *Geographic information system (GIS)*

The heart of PA has been the Geographic Information System (GIS) (Kumar *et al.*, 2017). The data evaluation using Global Positioning System (GPS) coordinates is the purpose of this program. It analyzes and maintains globally dispersed data that is scattered both spatially and temporally. New maps may be created using GIS data sets to show the geographical and temporal variability in a certain field. GIS data gathered over time may be utilized to retain records, discover relationships impacting output, and forecast how crops will react to inputs (Dwivedi *et al.*, 2017).

2.1.1.3 *Remote sensing (RS)*

According to Wójtowicz *et al.* (2016), Remote Sensing is a precision farming method which employs sensors installed on aircraft or satellites to track variations in the wavelength of light from fields and crops that are currently in development.

It is beneficial to keep track of the spectral and spatial variations throughout time at high resolution (Moran *et al.*, 1997). The spatio-temporal variations aid in understanding the field's heterogeneity through time, aid in identifying various crop species, assist detect

plants that have been harmed by disease or pests, and keep track of stress, soil, plant, and drought conditions.

2.1.1.4 *Real-time kinematic (RTK) system*

A GPS-based navigation system called Real-Time Kinematic System improves the accuracy of satellite position data (Luo *et al.*, 2016; Wang *et al.*, 2016; Wikipedia Contributors, 2020). This highly accurate guiding system lessens operator burden by preventing costly skips and overlaps, saving money on input costs, and reducing production costs. This technique employs a fixed base station that broadcasts geolocation to the rover's GPS receiver in order for it to adjust its position with respect to the fixed base station's known position with a precision of 1-2 cm (Dwivedi *et al.*, 2017). It makes precise row-to-row placement possible.

2.1.1.5 *Drones*

Drones might be seen as the body of farmers, whereas precision agriculture is their brain (Smith, 2018). Drones are deployed to inspect agricultural fields, soil, and weed spots for growth, texture, and the presence of diseases and pests. They are also utilized to spray chemical agents. Drones capture photographs with great resolution, enabling the creation of yield maps, contour maps, weed maps, and maps showing varied seeding rates (Dwivedi *et al.*, 2017).

2.2 Weed Management

Since the dawn of civilisation, weeds have existed and are certainly not going to go away any time soon (Renard *et al.*, 2012). Weeds are considered to be a persistent and pervasive hazard to agricultural output (Chen *et al.*, 2012). Designing the best effective strategy in a range of settings that ensures a healthy environment and a low impact of invasive weeds is the general objective of weed control (Di-Tomaso *et al.*, 2017).

Mechanical, cultural, biological, and chemical treatments are the four categories into which weed control strategies are typically separated (Scavo & Mauromicale, 2020). Although the invention and use of herbicides in the middle of the 20th century led to a reduction in the use of mechanical weeders on farms, these tools have since developed into highly effective and adaptable weed control tools for a range of cropping systems (Farooq *et al.*, 2019). Many procedures can be used to accomplish mechanical weed removal. By burying certain seeds at depths from which they cannot sprout, primary tillage helps reduce weeds of species that reproduce by seeds (Machleb *et al.*, 2020). Other seeds would be raised to the soil's exterior, allowing them to be directly exposed to cold, sunny or decomposition temperatures.

Herbicide treatments, often referred to as chemical applications, are the most effective weed management techniques (Soltys *et al.*, 2013). Herbicides are crucial weed-controlling instruments that have increased production rates and enabled reduced-tillage farming techniques (Bajwa, 2014). Although the effectiveness of herbicides is undeniable, they may also cause soil erosion, environmental degradation, and health issues in people (Kumar *et al.*, 2019). There are several methods to decrease the usage of expensive herbicides, including spot spraying, lower rates, and banding in conjunction with between-row cultivation (Regnier & Janke, 2020). Cost savings brought on the less frequent use of herbicides is one of the main advantages.

2.2.1 Principles of site-specific weed management (SSWM)

Site-Specific Weed Management (SSWM) entails treating solely weed spots and/or altering herbicide treatments in accordance with the distribution of weed species (such as herbicide-resistant or grass weeds) (De Castro *et al.*, 2012). Numerous contemporary agronomic and technical research on weed management have focused on automatic site-specific herbicide administration because it has the capacity to reduce the quantity of

sprayed chemical, enhancing farmer profits and decreasing pollution (Lati *et al.*, 2021). The field is treated as a collection of discrete management zones by site-specific herbicide administration techniques, which allows for the use of certain quantities and varieties of herbicide to eliminate the weeds that are there (Meyer & Mulliken, 2008). No herbicide is used if weeds are absent or if their concentration is lower than the economic threshold for treatment (the amount of weeds needed to make treatment worthwhile) (Shirzadifar *et al.*, 2015). To choose the most effective herbicide, it is crucial to be aware of the weed species that are present (Combarnous, 2017). When specific weeds have been identified, weeds can be managed in real-time (tactical technique) or strategically (strategic approach) employing a prepared field map that shows the species and position of weeds (Adamchuk *et al.*, 2008). The tactical method necessitates a unique procedure but provides the farmer with a more accurate estimation of the quantity and kind of chemical required for a particular weed issue. Before weeds reduce agricultural yields economically, post-emergence herbicide treatments should be undertaken (De Castro *et al.*, 2018). The essential time for weed management, according to UNL weed scientist Stevan Knezevic, is the post emergence period (Knezevic & Datta, 2015). The length of this phase is determined by the kind of crop, the variety of weed, the surrounding environment, and the quantity and concentration of the crop and weed.

2.2.2 Unmanned aerial vehicle remote sensing tasks

The majority of Unmanned Aerial Vehicle Remote Sensing applications employ photographs from sensors as primary data inputs; hence, they are computer vision-related tasks (Zhu *et al.*, 2018). Thus, classification, detection, and segmentation are three common and important computer vision problems that can be classified into three categories for UAV Remote Sensing tasks in PA which employ Deep Learning techniques (mostly CNN) (Everingham *et al.*, 2015).

(i) Classification attempts to forecast whether there will be or won't be at atleast one member of a specific object class in the photograph, and Deep Learning techniques are necessary to offer a real-valued certainty of the item's existence (Chen *et al*., 2021b). In order to identify crop diseases (Hu *et al.*, 2020; Ha *et al.*, 2017; Huang *et al.*, 2019), weed types (Bah *et al*., 2018a; Bah *et al*., 2018b; De Camargo *et al.*, 2021; Ukaegbu *et al.*, 2021), or crop types (Onishi & Ise, 2018; Zhao *et al.*, 2020), classification approaches are generally utilize.

(ii) Detection operations attempt to answer the inquiry "where are the occurrences in the photograph, if any," by predicting the bounding boxes of each item of a specific object class in the photograph with corresponding certainty. In other words, the object information that was retrieved is comparatively more accurate. The most common uses include identifying crops that have pests Chen *et al.* (2021a) or other diseases Li *et al.* (2021), locating the weeds in the images (Valente *et al.*, 2019; Veeranampalayam *et al.*, 2020), counting the crop number for yield estimation (Apolo-Apolo *et al.*, 2020; Chen *et al.*, 2019; Csillik *et al.*, 2018) or disaster evaluation (Zhang *et al.*, 2020b).

(iii) Segmentation is a process which forecasts the instance labeling (for instance Segmentation) or object labeling (for semantic segmentation) of each pixel in the test photograph, providing a higher accurate categorization for each pixel. It has the ability not only to find things but also collect their finer-grained pixels. Consequently, to precisely pinpoint interesting characteristics in pictures, segmentation techniques are typically utilized. Semantic segmentation could assist in identifying and tracking crop leaf diseases (Stewart *et al.*, 2019; Kerkech *et al*., 2018; Kerkech *et al.*, 2020), generating weed maps (Huang *et al.*, 2018; Sa *et al.*, 2018; Zou *et al.*, 2021), or assessing crop growth (Osco *et al.*, 2021; Zhang *et al.*,

2020a), and yields (Xu *et al.*, 2020), whilst also, instance segmentation indeed can identify crop from weed plants (Champ *et al.*, 2020; Mora-Fallas *et al.*, 2020), or conduct crop seed phenotyping Toda *et al.* (2020) at a finer level.

2.3 Machine Learning Methods

Across various fields, including medical systems (Tsouros *et al.*, 2017; Bonotis *et al.*, 2019), marketing (Cui & Curry, 2005) and biology (Tarca *et al.*, 2007), Machine Learning (ML) has so far been utilized to analyze the data obtained for forecast and/or classification applications. Machine learning technologies are frequently used in Precision Agriculture to make the most of the vast amounts of data collected by UAVs (Mazzia *et al.*, 2020). The use of Machine Learning (ML) may diagnose diseases, predict certain factors relating to crop growth rates, and even recognize different objects in photographs (Da Costa Lima & Mendes, 2020). Owing to the rapid developments occurring, particularly in the Deep Learning sector, the use of machine learning has significantly expanded lately (Dargan *et al.*, 2020)

2.4 Deep Learning Methods

A Deep Learning model, one of the types of machine learning, is created using the human brain as a model. The Deep Learning Neural Networks replicate the cognitive processes of the human brain by simulating a web of interconnected nodes (Magomadov, 2019).

Over the past few decades, a tiny subset of Artificial Intelligence (AI), commonly referred to as Machine Learning (ML), has transformed a variety of fields since its development in the 1950s. Deep Learning (DL) was born out of the ML sub - field of Neural Networks (NN) (Alom *et al.*, 2019). Since its introduction, Deep Learning has caused disruptions of ever-increasing size and has excelled in nearly every application sector. Deep Learning, which employs either hierarchical or Deep Learning structures, is a subset of

ML that has mostly been created after 2006. Determining model parameters is the first step in the learning process, which enables the learnt model (or algorithm) to carry out a specified task. For instance, the weight matrices are the parameters in Artificial Neural Networks (ANN).

Contrarily, DL contains several layers between the input and output layer, allowing for the presence of numerous stages of non-linear central processing unit having hierarchical architectures which are employed for feature learning and pattern categorization (Schmidhuber, 2015; LeCun *et al.*, 2015). According to several articles, DL is a universal learning strategy that can address practically any issue in a variety of application fields. So DL is not task-specific, to put it another way (Bengio, 2009). Figure 2.1 shows the classification of AI. Where, AI: Artificial Intelligence; ML: Machine Learning; SNN: Spiking Neural Networks; NN: Neural Networks; DL: Deep Learning.

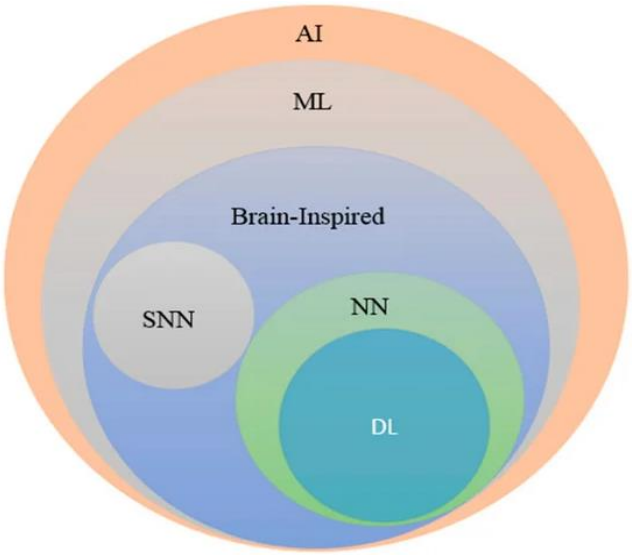

Figure 2.1: The classification of AI. (Source: Kilichan & Yilmaz, 2020)

2.4.1 Convolutional neural network architecture

As specified in a Conventional Multilayered Neural Network, CNN is among the subclasses of deep classifier architecture and consists of one or more convolutional layers

accompanied by one or more fully connected layers (Savalia & Emamian, 2018). CNN's design performs well while analyzing two-dimensional data types like images, movies, and so on (Browne *et al.*, 2008). A receptive field is a condensed area of the perceptual field that is responsive to the intricate configuration of cells that make up the visual cortex (Ide & Kurita, 2017). Numerous approaches, including Neocognitron, HMAX, and Lenet, are available in the literature since the animal visual cortex seems to be the most potent visual processing system (Sornam *et al.*, 2017; Azizah *et al.*, 2017; Zahara *et al.*, 2020; Pouyanfar *et al.*, 2018). Since CNN never needs a shared weight, it differs from Neocognitron (Du, 2018; Sornam *et al.*, 2017). The localized connection pattern between the neurons in neighbouring layers is how CNN's spatially local correlation originates. It can be illustrated graphically in Figure 2.2.

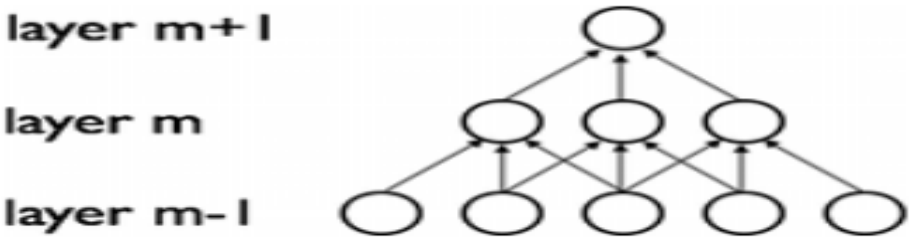

Figure 2.2: Pattern of neuronal connectivity (Sornam *et al.*, 2017)

CNN has inputs, outputs, and hidden layers in between, similar to the design of a normal Neural Network (Sornam *et al.*, 2017). These hidden layers carry out a task known as feature identification and three various types of data computations, including convolution, pooling, and rectifier linear unit (ReLu) (Albawi *et al.*, 2017; Al-Saffar *et al.*, 2017; Sornam *et al.*, 2017). By passing the input photograph via a number of convolutional filters, the convolution layer is employed to activate a particular aspect of the photograph (Srinivas *et al.*, 2016). Several aspects of the photograph are enabled by each filter. By streamlining the output via a nonlinear downsampling approach, the pooling layer aids in

reducing the amount of factors unrelated to the targeted problems (Sornam *et al.*, 2017). Rectified Linear Units accelerate learning as well as improve productivity by turning negative values into zeros such that they can only be maintained as positive values (Tang *et al.*, 2018; Sornam *et al.*, 2017).

In order for each layer to be capable of recognizing the various feature levels, these three distinct procedures are continually done to tens or even thousands of layers. The CNN architecture moved towards categorization after the feature identification was finished.

The network's capacity to forecast the amount of output classes, k, is represented by the next-to-last layer, a fully connected layer which generates the K dimension vector, and the last layer, a softmax layer, which produces the categorization output (Chen *et al.*, 2016). The component of the CNN is depicted in Figure 2.3.

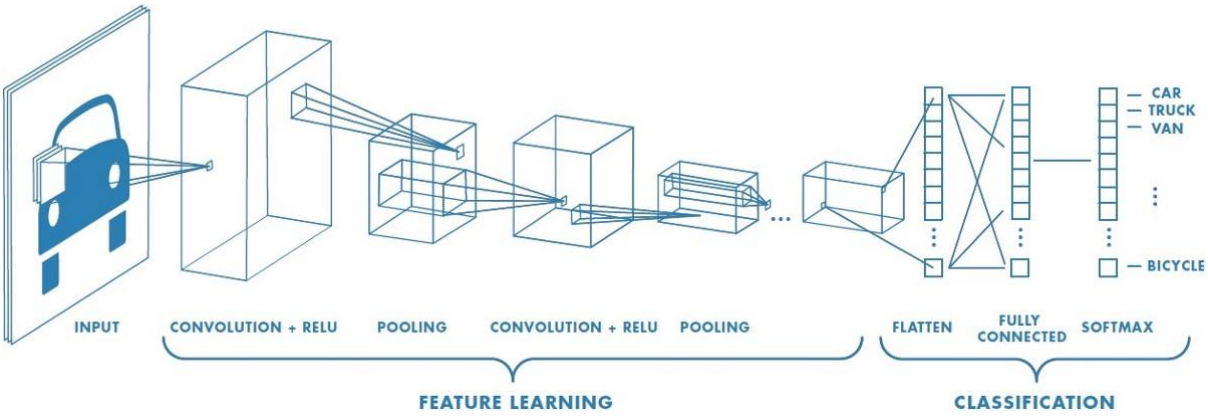

Figure 2.3: Architecture of CNN (Sornam *et al.*, 2017)

2.4.1.1 *Convolution layers*

Convolutional, Pooling, and Fully-Connected (FC) layers are the three categories of layers which constitute the Convolutional Neural Network (Akbar *et al.*, 2017). A CNN model will be created after these layers are stacked. Convolutional layers' neuron components first were calculated by convolution operations over small regional spots of input, after which activation functions (tanh, sigmoid, ReLU, e.t.c.) are applied to create

a 2D feature chart, which is a crucial feature layer which distinguishes Convolutional Neural Network (CNN) from other conventional Neural Networks (3D feature map) (Zhang *et al.*,2019).

Typically, we have that

$$Z_j = \sum_i X_i * K_{ij} + B_j, \quad (2.1)$$

$$A_j = f(Z_j), \quad (2.2)$$

while Z j indicates the output from the convolution process, Xi is the input to the convolutional layer, Kij is the convolution kernel, and Bj is the cumulative bias. In the expression that follows, Aj is the output feature map of the convolutional layer and f (Zj) is an activation function. Activation functions are computational processing on the input that adds non-linearity into Neural Networks and aid in detecting non-linear patterns in the input data (Sharma *et al.*, 2017b). The terms "saturated functions" refer to sigmoid and Tanh. According to their descriptions or charts, the output of Sigmoid and Tanh saturates at 0 or 1 and -1 or 1, respectively, whenever the input is extremely tiny or extremely big. In terms of saturation, there are two issues. It is challenging to converge in the training phase because the gradients at saturated areas are nearly negative, substantially reducing neurons' backpropagation (Zhang *et al.*, 2019). Additionally, weight initialization when employing saturated activation functions needs to be more carefully considered else the Neural Networks may well not train at all. Numerous non-saturated activations have been suggested to address the saturation issue, including the Rectified Linear Unit (ReLU) (Nair & Hinton, 2010), Leaky ReLU (Maas *et al.*, 2013), Parametric ReLU (PReLU) (He *et al.*, 2015b) and Randomized Leaky ReLU (RReLU) (Xu *et al.*, 2015).

In CNN, convolution is a crucial component. Contrarily, neurons within the same feature map use the same variables thanks to weight sharing that significantly lowers the overall amount of parameters (Indolia *et al.*, 2018). The input may exhibit the same characteristics, which include edges, points, angles, etc., at several spatial locations. The CNN is less susceptible to position and moving because of weight sharing (Ghafoorian *et al.*, 2017). However, because each convolution process only considers a tiny portion of the input, the recovered features retain the fundamental structure of the input, which aids in pattern recognition. Figure 2.4 provides an Activation Function plot.

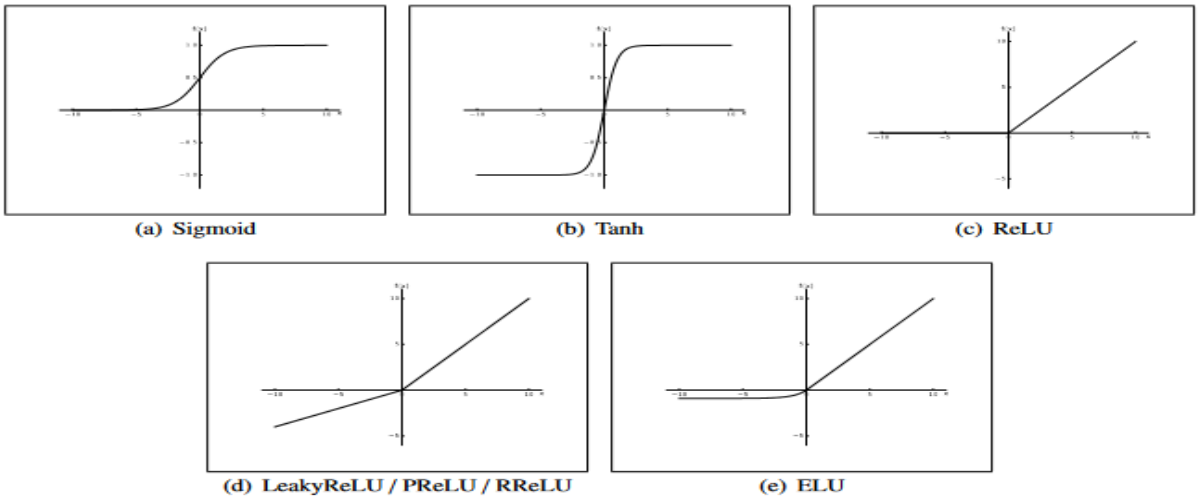

Figure 2.4: Activation function plot (Zhang *et al.*, 2019)

2.4.1.2 *Pooling layer or subsampling layer*

In order to decrease the feature map resolution, convolutional layers are typically accompanied by subsampling layers (LeCun & Bengio, 1995; Huang *et al.*, 2015; He & Sun, 2015). In line with this reduction in parameters, processing is likewise downsized. Figure 2.5 shows a Max Pooling layer implemented a single slice of an input volume,

$$Z_j = \text{down}(X_j), \tag{2.3}$$

In which a subsampling strategy is represented by down(Xj).

Two common subsampling techniques, maximum operation and average operation, are employed in CNNs (Alotaibi & Mahmood, 2017). In addition to max pooling and average pooling, various techniques which are more effective in preventing overfitting issues in CNN have been suggested, including mixed pooling (Yu *et al.*, 2014), stochastic pooling (Zeiler & Fergus, 2013), and Lp pooling (Sermanet *et al.*, 2012). He *et al.* (2015a) suggest a pooling technique known as spatial pyramids pooling (SPP), which may produce a constant length feature map and hence handle different input photograph dimensions. Fast Fourier Transform (FFT)-based CNNs can also use spectral pooling, a technique for reducing dimensionality in frequencies that maintains better information than spatial domain (Rippel *et al.*, 2015). Whereas multi-scale orderless pooling, as described by Gong *et al.* (2014), surpasses other approaches in high variability scene matching. Contrary to convolution kernels, subsampling kernels are frequently chosen manually and don't alter throughout training and inference. Subsampling is done for two major motives. The first is that the size of the feature map is reduced by maximising or average over the preceding feature map, whereas the other is that by subsampling, the resultant feature map is much more resistant to distortions and mistakes of particular neuron units (Liu *et al.*, 2017).

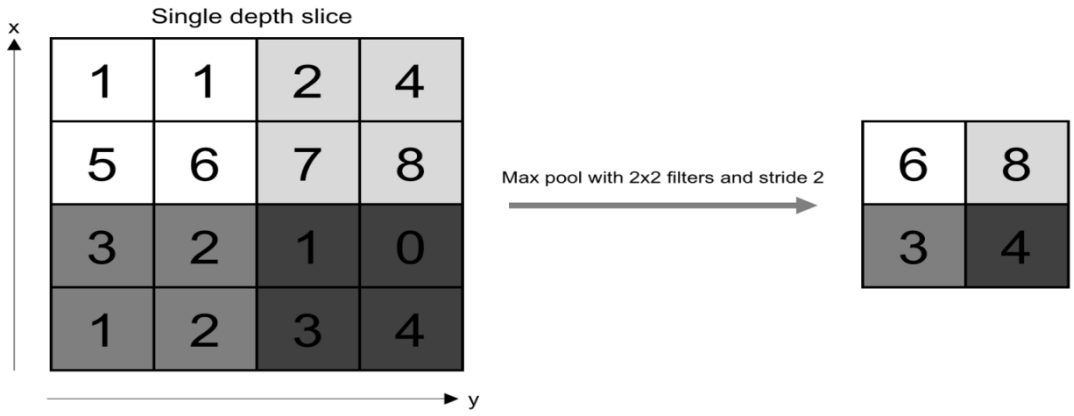

Figure 2.5: Max pooling layer implimenting a single slice of an input volume. (Source: Rana, 2020)

2.4.1.3 *Fully connected layer*

The Fully Connected (FC) layer, that links the neurons between two layers, is composed of neuronal cells along with weights and biases (Dose *et al.*, 2018; Viquerat & Hachem, 2020). These layers make up the final few levels of a CNN architecture and are often positioned just before output layer. This compresses the input photograph from the earlier stages and feeds it to the FC layer (Pu *et al.*, 2019). The compressed vector is then sent through some additional FC levels, in which the standard procedures on mathematical operations happen. The categorization procedure starts to take effect at this point.

2.4.1.4 *Rectified linear unit layer*

ReLU stands for Rectified Linear Unit, that implements the non-saturating activation function f(x) = max (0, x) (Ren *et al.*, 2016; Pratama & Kang, 2021). Through assigning negative values to zero, it essentially removes them from an activation map. Without changing the receptive fields of the convolution layers, it creates nonlinearities to the decision function and the entire network (Chi *et al.*, 2019).

The saturating hyperbolic tangent f(x) = tanh (x), f(x) = |tanh (x)|, and the sigmoid function (x) = (1+e-x)-1 are several other functions that may be employed to improve nonlinearity (Vijayaprabakaran & Sathiyamurthy, 2020; Vargas *et al.*, 2021). ReLU is frequently used over other functions since it trains the neural network considerably more quickly without significantly degrading generalization accuracy (Huang *et al.*, 2020).

2.5 Faster RCNN Architecture

An object detection technique relying on the region proposal approach is known as the Faster RCNN. The very earliest Region Proposal technique algorithm was region-based CNN (R-CNN) (Girshick *et al.*, 2015). But then again, it was computationally intensive because each suggested zone required a CNN-based feature extraction. Sharing

convolutional features among area suggestions was suggested as a way to create a fast RCNN and cut down on computation time (Girshick, 2015). Fully convolutional Region Proposal Networks (RPN), which are taught to suggest accurate object areas, were presented as a faster version of RCNN to increase speed (Ren *et al.*, 2015). The four components of the Faster RCNN model are classification, Region of Interest (RoI) pooling, Region Proposal Network (RPN) and feature extractor.

This research utilized the Inception v2 convolutional layers for extraction of features. The Inception v2 network has the benefit of being translation and scale-invariant because of the utilization of broader networks having various kernel sizes in each layer of the network (Veeranampalayam Sivakumar *et al.*, 2020). For this, the region proposal layer's feature map generated by the Inception v2 architecture is decreased in dimension. Anchors or permanent bounding boxes at every position serve to define the RPN (Chen *et al.*, 2018). In order to allow the area proposal network to generate scale-invariant proposals, anchors with various scales and aspect ratios are established at each point. A convolutional filter on the feature chart is used by the region proposal layer to provide a confidence score for the classifications of objects and backdrop (Suhail *et al.*, 2020). It is known as the objectness score. Additionally, anchor box regression offsets are produced by the convolutional filter (Yi *et al.*, 2021). As a result, if a region has k anchors, the convolutional filter in the area proposal network produces 6k data, including 4k coordinates and 2k scores. By this result, classification loss and bounding box regression loss are computed. The feature map from feature extractor can then be blended with the bounding box coordinates of anchors designated as objects. Bounding box areas with various sizes and aspect ratios are adjusted to fixed size outputs in the RoI pooling layer utilizing max pooling.

When there is maximal pooling, the down sampling is performed by the maximal number of pixels. The pooling layer is a down sampling layer (Krizhevsky *et al.*, 2012).

Following classification, the bounding box discrepancies with reference to the ground truth boxes of the max-pooled feature map of a predetermined size that corresponds with each output are regressed. As a result, two losses, the classification loss and the bounding box regression loss are estimated at this output, just as they were in the region proposal layer. Figure 2.6 shows the Faster RCNN architecture.

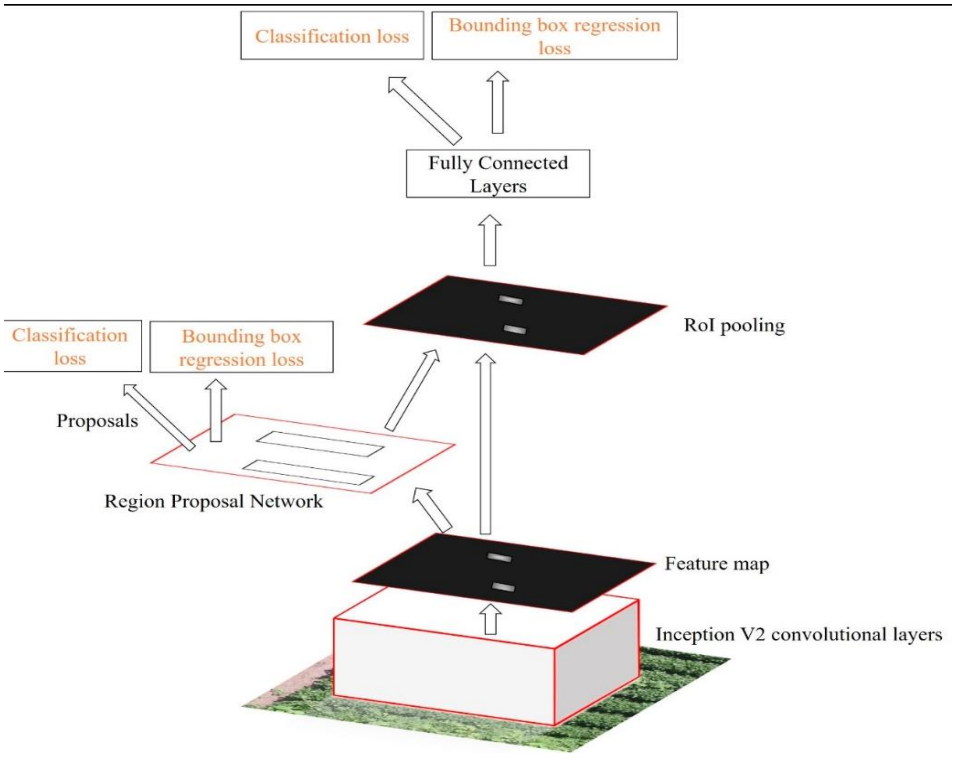

Figure 2.6: Faster RCNN architecture (Veeranampalayam Sivakumar *et al.*, 2020)

2.5.1 Inception v2 architecture

CNN architecture's "inception architecture" modifies the feature extraction process (Khan *et al.*, 2020). The feature extraction section is where the difference may be found.

As can be seen in figure 2.11, the feature extraction part of Inception v2 employs base layer and filter concat (Alamsyah *et al.*, 2019). The enhanced usage of the computational resources within the network is an attribute of Inception design (Chelghoum *et al.*, 2020).

The main advantage of the Inception Design is that it outperforms shallower and less broad networks while requiring just a slight improvement in processing resources, and it is efficient despite not using context or implementing bounding box regression (Szegedy *et al.*, 2015). Utilizing clever factorization techniques, the inception design aims to decrease the constraint and increase better performance in terms of computing complexity. To increase computational performance, the 5x5 pixel convolution layer of the Inception v2 architecture was categorized to a 3x3 pixel convolution (Szegedy *et al.*, 2016). Figure 2.7 shows the Inception v2 architecture.

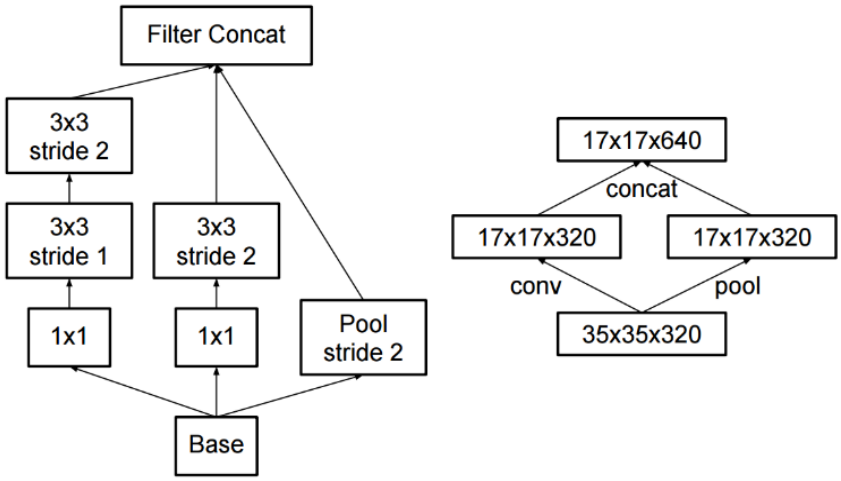

Figure 2.7: Inception v2 architecture (Szegedy *et al.*, 2015)

2.5.2 YOLO v5 architecture

Convolutional Neural Networks predicated on the "You Only Look Once" (YOLOv5) framework family enable real-time object identification. The YOLO family was updated with the publication of YOLOv5, by researcher Glenn and his colleagues, a month after YOLOv4 was released (Thuan, 2021). Glenn Jocher is the CEO of Ultralystics LLC and a researcher (Jocher *et al*., 2020). Alexey Bochkovsky built the YOLO models on top of the bespoke framework Darknet, which is primarily coded in C (Thuan, 2021; Wang, 2021). The firm that adapts older itrations of YOLO onto PyTorch, among the most popular Deep Learning frameworks built in Python, is called Ultralystic, which is a brand-

new Convolutional Neural Network that accurately recognises objects in real-time (Zhang *et al.*, 2022). In this approach, the whole imagery is processed by a single Neural Network, after which it is divided into its component parts and bounding boxes and probabilities are predicted for each one (Hemanth, 2022). The weighting of these bounding boxes is based on the anticipated likelihood. The approach merely does a single forward propagation loop through the Neural Network before making predictions, or "only looking once" at the picture. Following non-max suppression, it provides discovered objects that makes sure the object detection algorithm only recognizes each object once.

According to Liu *et al.* (2019), YOLOv5 is premised upon this, YOLO detection designs and utilizes the outstanding classifier optimization technique in the area of CNNs in recent times, which includes automated bounding box anchor training, enhancing mosaic data, and the cross-stage partial network, etc; which are in charge of various operations in various parts of the YOLOv5 architecture. The input, backbone, neck, and output are the four essential components of the YOLOv5 design. The data preparation, such as adaptive picture filling and mosaic data augmentation Wu *et al.* (2017), is mostly contained in the input terminal. YOLOv5 incorporates adaptive anchor frame computation on the input in order to adapt to various data sets, and this can therefore autonomously determine the starting anchor frame size as the data changes. By repeatedly convolution and pooling, feature charts of varying sizes may be extracted from the input picture, the backbone network mostly employs a Cross-Stage Partial network (CSP) Kim *et al.* (2019) and Spatial Pyramid Pooling (SPP) (He *et al.*, 2015c). Whereas the SPP design allows for the realization of extracting features from various scales for same feature map and has the ability to create three-scale feature charts, that increase recognition precision, to cut down on the complexity of calculations, Bottleneck CSP is utilized and also speed up inference.

The feature pyramid architectures of FPN and PAN are applied to the neck network. Robust semantic features are transmitted from the top feature charts into the bottom feature charts utilizing the FPN structure (Liu *et al.*, 2016b).

Additionally, the PAN structure transfers powerful localization features from lower feature charts to higher feature charts (Wang *et al.*, 2019). The collective strength of these two systems improves the feature acquired from different network levels in Backbone fusion, thus increasing the identification abilities. The head output is mostly utilized as the last detection step to forecast targets of various sizes on feature maps. Four architectures, known as YOLOv5s, YOLOv5m, YOLOv5l and YOLOv5x, make up the YOLOv5 (Liu *et al.*, 2021; Yan *et al.*, 2021). Their primary distinction is the quantity of feature extraction module and convolution kernels placed around particular points in the network. Figure 2.8 displays the YOLO v5 network architecture.

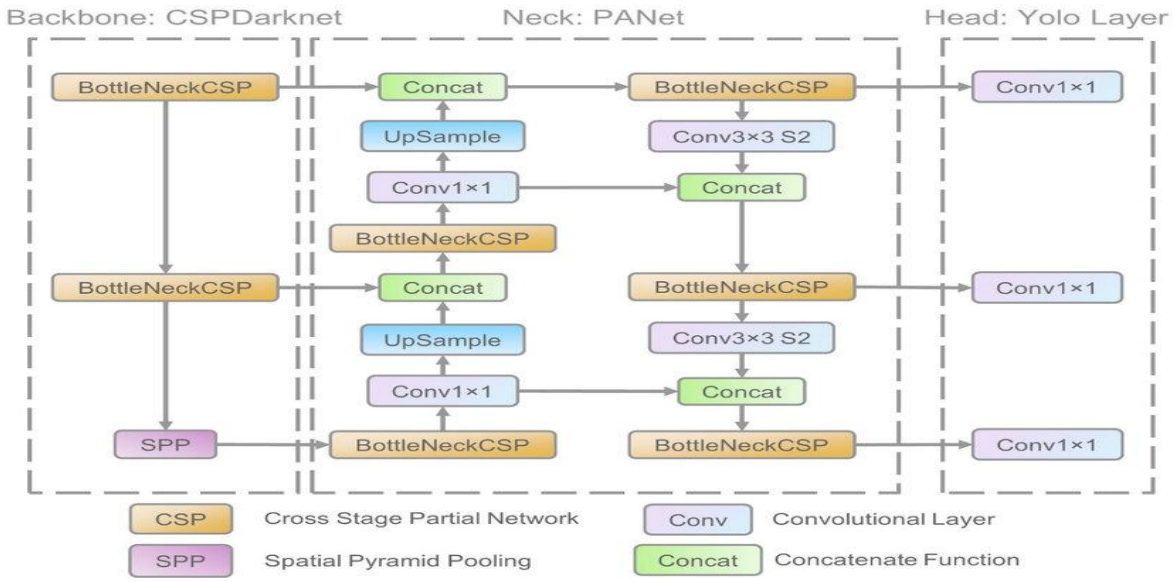

Figure 2.8: The design of YOLO v5's network (Xu *et al.*, 2021)

2.6 Training Epochs

Brownlee (2018) describes the number of epochs as a hyperparameter that specifies how many times the learning algorithm will run across the whole training dataset. Each sample

in the training dataset has had one epoch, which indicates that the internal model parameters have been updated. Each epoch is made up of one or more batches. For instance, as previously stated, an epoch with one batch is referred to as the batch gradient descent learning algorithm (Dogo *et al*., 2018). Consider a for-loop across the number of epochs, with each loop traversing the training dataset. Another nested for-loop within this for-loop iterates through each batch of samples, where each batch has the provided "batch size" number of samples. Typically, the number of epochs is huge, frequently hundreds or thousands, permitting the learning procedure to continue until the model error is adequately minimised (Hu, 2021). Examples of the number of epochs set to 10, 100, 500, 1000, and larger can be found in the literature and tutorials. Line charts with epochs along the x-axis as time and the model's error or skill on the y-axis are typical (Hoiem *et al*., 2021). These charts are also known as learning curves. These charts can aid in determining if the model has overlearned, underlearned, or is adequately fitted to the training dataset.

2.7 Google Colaboratory

Jupyter Notebooks which is the innovation on which Google Colaboratory is built, is introduced before Google Colaboratory. According to Perez & Granger (2007), Jupyter is a free software web application that combines interpreted languages, libraries, and visualization tools. Working locally or online is possible using a Jupyter notebook (Mendez *et al.*, 2019). Each document consists of a number of cells, with script or markdown code in each cell and the output is embedded in the content. Text, tables, graphs, and images are common outputs. Due to the experiments' and findings' self-contained presentation, using this technology facilitates the sharing and replication of scientific works (Randles *et al.*, 2017). An initiative called Google Colaboratory, sometimes known as Colab, aims to spread knowledge about Machine Learning research

and teaching (Colombo-Filho *et al.*, 2020). Jupyter-based collaborative notebooks function like a Google Document object, allowing users to work and interact on the same notebook and share it with others (Carneiro *et al.*, 2018). The crucial Machine Learning and Artificial Intelligence libraries, including TensorFlow, Matplotlib and Keras, are provided by Colaboratory in both Python 2 and 3 runtimes (Waheed *et al.*, 2020; Suljovic *et al.*, 2022). All user data and customizations are lost when the Virtual Machine beneath the runtime (VM) is disabled. Though the notebook is still there, it is also feasible to upload things to the user's google drive account via the VM hard drive.

On a laptop, the complete model is trained and tested using Google Colaboratory and the Python language. An entirely cloud-based, free Jupyter notebook environment is called Colaboratory. It doesn't require any configuration and operates completely in the cloud (Prashanth *et al.*, 2021).

2.8 Software

Numerous software tools and strategies have been created to speed up data processing because it may often be time-consuming. Table 2.1 provides an overview of the software packages that have been used to process and speed up the data analyzing process in the works assessed.

Table 2.1: Software packages utilized for image processing in the research.

Software Tool	Summary
Python programming v3.8	Simple, general purpose, high level, and object-oriented programming language version 3.8.
Google Colaboratory Free	K80 GPU, RAM 16GB, Runtime 12hours, Jupyter notebook environment on google colabs.
Google Colaboratory Pro	GPU- K80,T4 AND P100; RAM 32GB; Runtime 24hours; cost 10$ per month; Jupyter notebook environment on google colabs.

CNN is used by the majority of computer vision applications (Nanni *et al.*, 2017; Voulodimos *et al.*, 2018; Bhatt *et al.*, 2021; Wang *et al.*, 2019; Islam *et al.*, 2016). Hardware for superior performance and excessive electricity usage of this hardware are indeed two significant issues with CNN (Carneiro *et al.*, 2018). Consequently, high-performance hardware is necessary, such as the GPU from Colaboratory. CNN training utilizing Colaboratory's enhanced runtime is 2.93 times quicker than with all of the physical cores of a Linux server, on average (Carneiro *et al.*, 2018).

2.9 Related Literatures on Weed Detection using Machine and Deep Learning Algorithms

The spotlight has been drawn to methods centered on Machine Learning for spotting weeds and crops (Murawwat *et al.*, 2018). Support Vector Machine (SVM) classifier was employed by Murawwat *et al.* (2018) to distinguish between carrot crops and weeds. With 72 training samples and 8 test samples, they were able to attain a classification accuracy of more than 50%. The problem with the classic ML technique, such SVM or RF classifiers, is that extraction of features is not automated and manually generated features creation is a time-consuming stage. According to certain research, Deep Learning can effectively cope with the drawbacks of manually created features for identifying weed and crops by retrieving the features directly from the input data, in contrast to standard Machine Learning techniques (Lee *et al.*, 2015). Recent years have seen significant advancements in the categorization of Remote Sensing data employing Deep Learning for a variety of jobs, particularly agricultural ones.

Convolutional Neural Networks (CNNs) have been utilized in several research to classify crops and identify weeds in agricultural applications (Mortensen *et al.*, 2016; Potena *et al.*, 2017; Di Cicco *et al.*, 2017). Utilizing mixed crops of an oil radish plot with barely, weed, stump, grass, and background soil photographs, Mortensen *et al.* (2016) classified weeds using the VGG-16 CNN model. A perceptual system that employs shallow and deeper CNNs was developed by Potena *et al.* (2017) for the categorization of weed crops. When classifying weeds and crops, the deeper CNN is employed, whereas the shallower one is utilized to recognize vegetation. Using a SegNet, Di Cicco *et al.* (2017) generated sizable synthetic training datasets programmatically while randomly distributing the

targeted environment's essential properties (i.e., crop and weed species, type of soil, light conditions). The U-Net approach was used by Hashemi-Beni & Gebrehiwot (2020) to identify and separate crops from weeds utilizing a small dataset. To enhance the categorization outcomes, they used strategies like random cropping, random rotation, and reflection on the data.

Additionally, techniques centered on Deep CNNs have shown successful weed categorization and detection results. For instance, Yu *et al.* (2019), Olsen *et al.* (2019), and Dyrmann *et al.* (2016) all employed similar techniques. Potena *et al.* (2017) used two distinct CNNs to analyse RGB and NIR pictures in order to quickly and precisely detect crops and weeds.

A shallow CNN was utilized to categorize the retrieved pixels into crops and weeds after a lightweight CNN had quickly and robustly segmented the vegetation. Beeharry & Bassoo (2020) assessed the effectiveness of ANN and AlexNet, two weed detection algorithms based on UAV imagery. According to the experimental findings, AlexNet's weed identification accuracy was greater than 99%, while ANN's accuracy on the same dataset was just 48%.

A model for segmenting weeds in aerial images was developed by Ramirez *et al.* (2020), who then compared it to SegNet and U-Net. The study's findings demonstrated that more accurate experimental results were achieved through data balancing and improved spatial semantic information. An enhanced Mask RCNN model was suggested by Patidar *et al.* (2020) to extract early cranesbill seedlings. These weeds can be utilized as natural rheumatoid arthritis treatments. The suggested technique made it possible to entirely remove the weeds from the actual photograph in order to receive all of the nutrients and enhance production. A Deep Neural Network-based (DNNs) semantic segmentation

strategy for weed crop detection was put out by You *et al.* (2020). In order to increase segmentation accuracy, four more components were included, which improved performance for weeds with random shapes in a complicated environment. These techniques can autonomously gather meaningful feature information from pictures without depending on image preprocessing or data conversion.

Predicated on Faster RCNN, Le *et al.* (2021) investigated the detection of weeds from crops in difficult field environments. The outcomes showed that Faster RCNN techniques, particularly the Faster RCNN model with Inception-ResNet-V2, may be used to detect weeds in challenging field scenarios with changing weather, lighting, occlusion, and growth phases. For plant-specific management in precision farming, Lottes *et al.* (2018) combined crop-weed categorization with joint stem identification. Their method made use of an end-to-end trainable fully convolutional network that concurrently trains class-wise stem recognition and pixel-wise semantic segmentation while continuously estimating stem locations and the total area of crops and weeds. Using convolutional neural networks and convolutional neural network frameworks, Gothai *et al.* (2020) conducted research on weed detection. Building algorithms like the VGG-16, ZFNet, and ALEXNET with four, six, eight, or thirteen convolution layers, as well as other architectures, was done in an effort to increase accuracy.

The research gaps from the reviewed literatures suggest that future works be executed in the following aspect:

(i) Evaluating the effects of varying training epochs on the performance accuracy of some Deep Learning algorithms utilizing UAV's for the acquisition of aerial data. This research project will focus on filling this gap.

(ii) Consideration of spectral and spatial resolutions to optimise the flight mission to capture the size of the smaller weeds to be discriminated for better performance accuracy is also unresolved.

(iii) Utilizing multispectral and hyperspectral imageries for the development of improved classification algorithms for weed infestation assessments remains a gap according to literature.

CHAPTER THREE

3.0 MATERIALS AND METHODS

3.1 The Research Design

This chapter's layout contains the methodology that was adopted in conducting this research. Furthermore, it decribes the method of the materials that were utilized and also the data acquisition for the research, the data source, the preprocessing and processing operations of the data, including the methodological workflow adopted to achieve the desired aim of the research. Farmers suffer agricultural production losses due to unchecked weed growths on agricultural fields as a result of practicing conventional weed removal practices which are time consuming, labor intensive and have adverse effect on the soil. Consequently, this chapter illuminates the application of Deep Learning algorithms (Faster RCNN and YOLO v5) and Unmanned Aerial Vehicles in resolving the current issues faced by farmers.

3.1.1 Hardware and materials used for this study

The hardware materials utilized for this study comprises the following:

i. The quadrotor (phantom 4) UAV
ii. Flight controller
iii. A battery
iv. A microcomputer
v. RGB sensor.

vi. A Laptop PC with 8GB RAM, processor speed of 2.6 GHz and an internal memory of 1 TB.

3.1.2 Software and tools used

These are grouped into firmware and software. The firmware was executed on a Tensorflow library with a python programming language version 3.8 on both Google Colaboratory (Colab) free version which is accessible to all for free but having a RAM size of 16 GB and a runtime usage of 12hours and Google Colaboratory pro with a RAM size of 32 GB and a runtime usage of 24hours that attracts a fee of 9.99$. The python codes aids the coding of the identification and classification network model used for the training, testing and validation of the dataset. Furthermore, Pix4D mapper software was used to create an orthomosaic of the study site.

3.2 Data Acquisition

On February 17, 2022, amidst clear skies, high-resolution UAV photograph data were collected to assure comparable lighting conditions. A DJI Phantom 4, with an on-board RGB sensor having a resolution of 12 megapixels and 5.74 mm focal length, was utilized for the airborne survey. A 30 m height above ground level was selected for the flight mission. This made it possible to traverse the study area with a flying duration of around 15 minutes and a spatial resolution of 0.5 cm.

Eight field targets were distributed uniformly across the research site to be utilized as ground control points (GCPs) for georeferencing. A differential GPS (bi-frequency GNSS receiver) centered on the German SAPOS correction service was used to establish the field target centers for precision positioning in Real-Time Kinematic (RTK) mode.

In order to minimize shadows, the mission was undertaken at midday. Using the built-in three-axis gimbal, the sensor was mounted vertically at 90^0. About 254 photographs in all were captured at a mapping speed of 7 mph having a side and front overlap of 75%, correspondingly. With UAV weight (battery & propellers included) of 1380g and battery capacity of 5350 mAh. Plate I - IX presents the images of the flight path and equipments utilized in the acquisition of aerial images as well as GCP's for georeferencing the dataset while Table 3.1 presents the details of the flight plan.

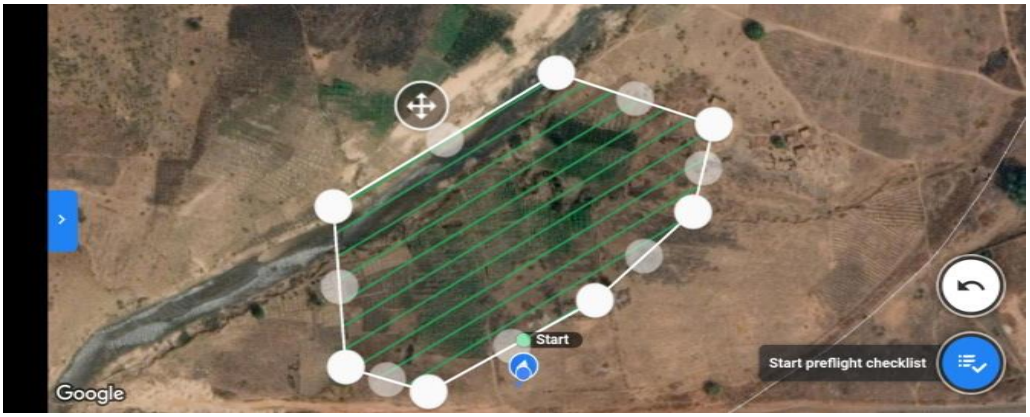

Plate I: The flight path of the UAV

Plate II: Unmanned aerial vehicle DJI phantom 4

Plate III: Flight controller and the display screen

Plate IX: The DGPS instrument used for acquiring GCP's

Table 3.1: Details of the Flight Plan

Parameters	Value
Shooting angle	At 90^0, in line with the main path
Capture mode	Fly and capture
Flight direction	-33^0
Speed	7mph

Altitude	30meters
Starting waypoint	1
Front overlap	75%
Side overlap	75%

3.3 Flow Chart for the Faster RCNN Algorithm

The Site Specific Weed Management was implemented employing a Faster Region based Convolutional Neural Network which is a Deep Learning (DL) algorithm. This Deep Learning algorithm will generalize on the testing set after being trained. Figure 3.1 shows the flow of the ployed in this study, d acquired and processing.

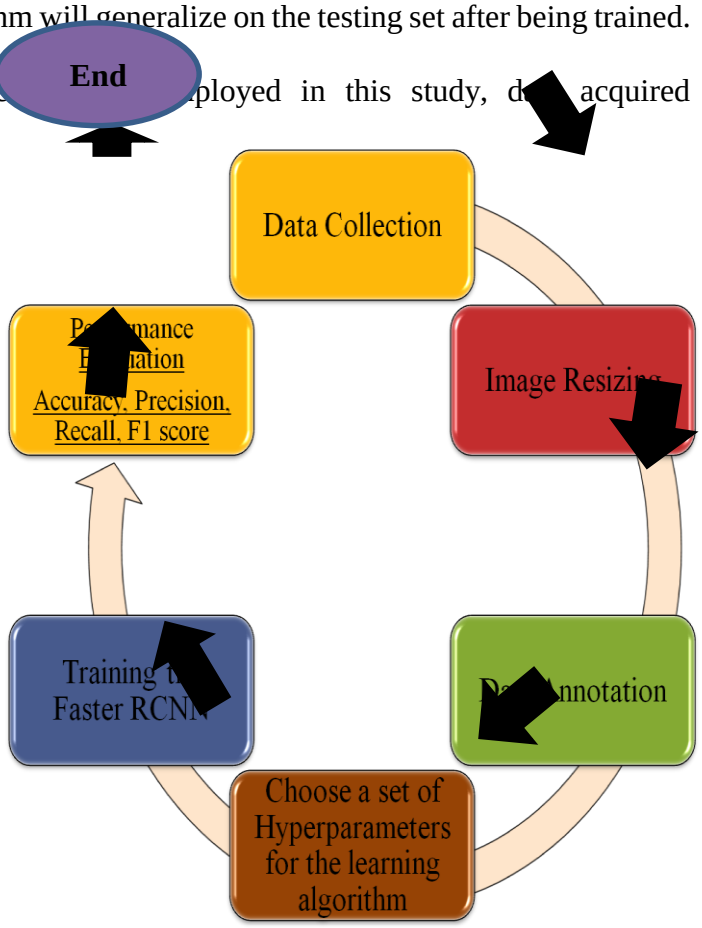

Figure 3.1: The workflow of the development and implementation of the faster RCNN weed detection model

3.4 Pre-processing of Images

In order for the dataset to fit into the model architecture for training and classifications of weed patches, the dataset were pre-processed utilizing the necessary software packages. This section contains the pre-processes employed to achieve the desired results. These processes are as follows:

i. Image Resizing
ii. Data Annotation
iii. Splitting of Data

3.4.1 Image resizing

The initial dimensions of the raw aerial photographs were quite enormous to fit in the memory for processing so each raw aerial photographs of 4000 x 3000 mega pixels from the dataset were then resized to 750 x 1000 mega pixels. In Python programming, an adaptive interpolation technique is used to achieve the resize process. This was done to reduce the large sizes of individual images.

3.4.2 Data annotation

Annotation is a machine learning method that labels data on photos that feature specified items or objects by putting a bounding box around each crop in the field. The resized images are labeled in order to pick the suggested region, that's comprised of the respective crop. The weed regions in individual sub-images were labeled as rectangular bounding boxes utilizing the python labelling imager program (LabelImg). Only five (5) annotators were employed in the labeling procedure. The annotator was trained to outline rectangular bounding boxes surrounding weed spots and the different crops on the farm.

3.4.3 Splitting data

Given that there is a considerable number of data, it is split into test sets and train sets. A training set is a subset of data used to train the model. The test set is a subset of information that may be examined using our qualified model. 254 sub-photographs in totality were manually labeled and were subsequently splitted between 80% training dataset and 20% test image. Figure 3.2 shows the pre-processing workflow.

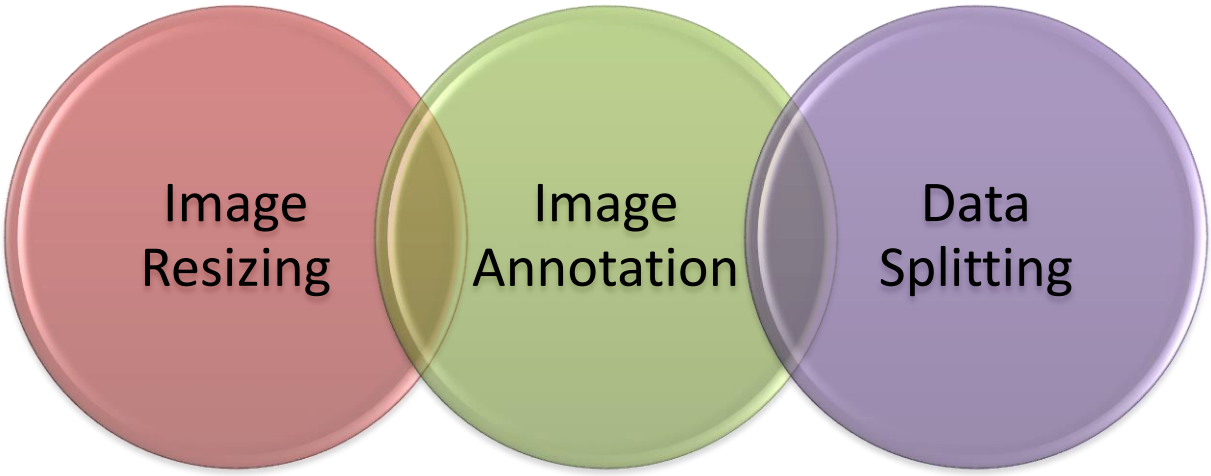

Figure 3.2: Workflow for pre-processing

3.5 Supervised Learning for both the Faster RCNN and YOLO v5 Models

The labeling of the training and validation datasets results in supervised learning. The aerial photograph and the associated annotations are part of the dataset supplied into the convolutional neural network as input. Accordingly, during supervised learning, the algorithm picks up knowledge from the labeled dataset on how to map a certain input to a specific output.

3.6 Training the Model with Dataset

This suggested model was trained on Google Colab having a GPU (NVIDIA GeForce GTX TITAN X (Linux)) employing Google Colaboratory Free with a GPU R-80 and RAM 16GB and Google Colaboratory Pro with GPU K80,T4,P100 and RAM 32GB. The runtime environment for Graphics Processing Units (GPUs) can considerably expedite the training process for many Deep Learning models. Tensorflow and CUDA/CuDNN are implemented to parallelize computations on the GPU. Python 3 programming notebook was thus uploaded. Numpy 1.19.5 and Tensorflow version 1.15.2 were installed on the virtual machine for compatibility and to do various mathematical operations, as well as to determine which GPU was allocated. The dataset was then loaded into the Google colab workspace, as well as into a "datalab folder," from where the photos, annotated files, and separate test samples were obtained. The labels were entered into a configuration file that defined all detectable classes ("sugarcane, spinach, pepper, banana, and weed"). The file names for training and validation were again retrieved by iterating through all image files. Bounding boxes were made employing the label imager "LabelImg software," which was also built in Python. Boundaries inside crops are defined by using coordinates of the bounding boxes in ".XML" formats via XML annotation files. The following step was to construct labelled tensor matrices (tf_records). Tensorflow Record files comprise the real input data for the machine learning process in binary format, making training faster. After that, the dataset is divided into training and testing data. 80% of the data was used for training, while the remaining 20% was used for testing. To evaluate the performance of the training, a Tensor board was placed and loaded. Paths and training parameters were set up to specify what files and model waypoints should be utilized during the training process, for example. 5 classes were defined, having a learning rate of 0.0002, a batch size of 32, and so forth. The training on GPU was then carried out for 10,000 epochs (this reflects the number of iterations the Deep Learning model has

accomplished over the entire training dataset) took about 27.8minutes to fully complete and subsequently, 20,000 epochs ran for 54minutes, 100,000 epochs ran for 3.6hours, 200,000 epochs ran for 7.9hours and 242,000 epochs took about 9.6hours to train. Then the inference graph was exported. Inference means to apply the model to imagery which have not been utilised for training. This is the testing dataset. Whenever the loss function no longer converges and begins to idle about a given value, the training should be terminated. Figure 3.3 presents the workflow of the training process.

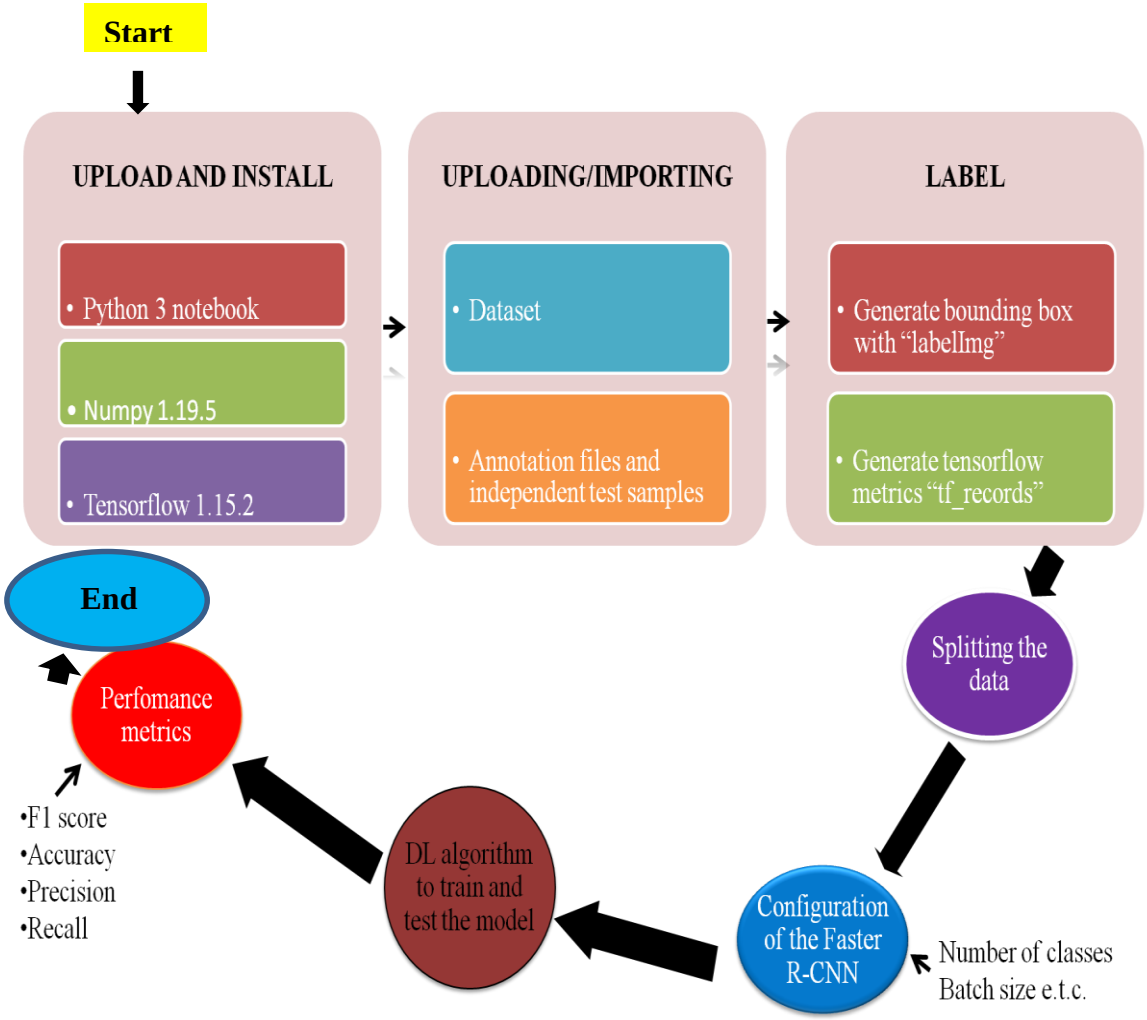

Figure 3.3: The training process flowchat

3.7 Evaluation and Prediction for both Faster RCNN and YOLO v5 Models

The trained network is tested utilising test samples, and its performance is assessed utilizing metrics which include confusion matrix, precision, recall, and F1 score for various IoU ranges. A confusion matrix is used to measure object identification accuracy.

3.7.1 Evaluation of performance

A confusion matrix is an overview of how many predictions a model performed were accurate and inaccurate. It is beneficial to display both the model's faults and the many kinds of errors that might occur when predicting an object's classification (Hasan *et al.*, 2021). Below are the terminologies used in describing the classification results (Maxwell *et al.*, 2021);

i. **True Positives (TP)** describes the number of instances that the model accurately identified a positive sample as positive.

ii. **True Negatives (TN)** indicates a specific number of instances the classifier correctly identified a negative sample as negative.

iii. **False Positive (FP)** describes the frequency with which a negative sample was incorrectly categorized as a positive sample by the classifier.

iv. **False Negative (FN)** indicates the frequency with which a positive sample was incorrectly categorized by the model as negative.

3.7.2 Accuracy metric

The algorithm's performance throughout all classes is often described by its accuracy metric. It is determined by dividing the number of accurate predictions by the overall number of predictions (Teimouri *et al.*, 2018).

$$\text{Accuracy}=\frac{True_{positive}+True_{negative}}{True_{positive}+True_{negative}+False_{positive}+False_{negative}} \quad (3.1)$$

3.7.3 Precision metric

The precision is computed as the ratio of True Positives to both the total number of False Positives and True Positives. The precision measures how effectively the model classifies a sample as positive (Prashanth *et al.*, 2020).

$$\text{Precision} = \frac{True_{positive}}{True_{positive+}\ False_{positive}} \tag{3.2}$$

The denominator rises and the accuracy becomes low whenever the model produces numerous wrong Positive classifications or few accurate Positive classifications. On the other side, the accuracy is higher when the model assigns more correctly classified positives and assigns less incorrectly classified positives. The accuracy rating ranges from zero (0) (no precision) to one (1) (complete or perfect precision).

3.7.4 Recall metric

Recall is measured as the proportion of positive samples that were actually accurately identified as Positive samples to all positive samples. The recall gauges how well the algorithm can identify positive samples. The greater the number of positive samples detected, the greater the recall (Jiang *et al.*, 2020).

$$\text{Recall} = \frac{True_{positive}}{True_{positive}+\ False_{negative}} \tag{3.3}$$

Only the classification of the positive samples is important to the recall. This is unrelated to the classification of the negative samples, such as for precision. The recall will indeed be 100% if the algorithm properly identifies all positive samples as positive, even if all negative samples are wrongly categorized as positive. The outcome is a number that ranges from 0.0 for no recall to 1.0 for complete or ideal recall.

3.7.5 F1 score metric

The harmonic mean of recall and precision is the F1 score. It accounts for both false positives and false negatives. This enables the combination of accuracy and recall into a

single metric that accounts for both characteristics. However neither Precision nor the Recall by themselves provides the general overview. It is possible to have great recall with poor precision or poor recall with superb precision. The F1 score offers a means of expressing both issues with a single score. Once the values for the Precision and Recall have been predicted, the macro average of both the Precision and Recall for the different epochs will then be calculated as follows:

Average

$$\text{Precision} = \frac{\text{P(Weeds)} + \text{P(Banana)} + \text{P(Sugarcane)} + \text{P(Spinach)} + \text{P(Pepper)}}{5} \quad (3.4)$$

Average

$$\text{Recall} = \frac{\text{R(Weeds)} + \text{R(Banana)} + \text{R(Sugarcane)} + \text{R(Spinach)} + \text{R(Pepper)}}{5} \quad (3.5)$$

$$\text{F1 Score} = \frac{2*(Average\ Precision*Average\ Recall}{(Average\ Precision+Average\ Recall)} \quad (3.6)$$

3.8 Flow Chart for the YOLO v5 Algorithm

The Site Specific Weed Management was implemented employing a YOLO v5 which is a Deep Learning (DL) algorithm. This DL algorithm will evaluate on the validation set then generalize on the testing set after being trained. Figure 3.4 presents the workflow for the methodology of YOLOv5s processing, data acquired and processing.

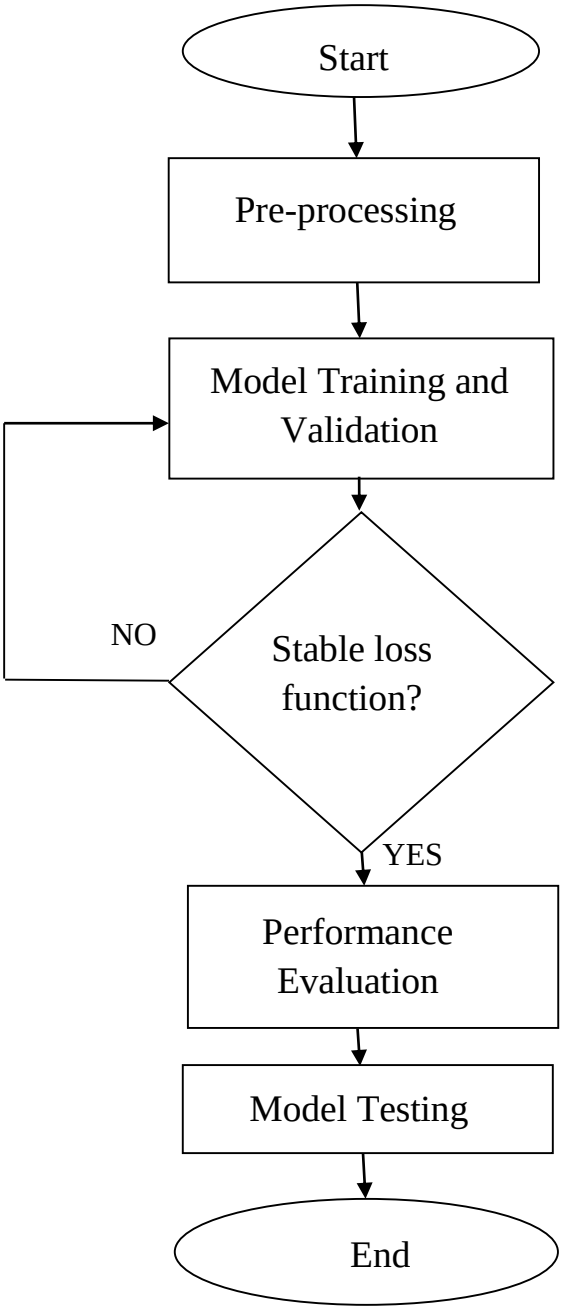
Start
Pre-processing
Model Training and Validation
Stable loss function?
NO
YES
Performance Evaluation
Model Testing
End

Figure 3.4: The workflow for the methodology of YOLO v5 processing

3.8.1 Pre-processing of images

The dataset were pre-processed to meet the dimensions that will properly fit into the architecture of the YOLO v5 model and manually annotated. The following steps were taken to achieve the desired results:

i. Image Resizing

ii. Data Annotation

iii. Splitting of Data

3.8.1.1 *Image resizing*

The initial dimensions of the raw aerial photographs were too large to match up the memory for processing as a result, each raw photographs of 4000 x 3000 mega pixels taken from the dataset were then resized into 416 x 416 pixels to fit into the architecture of the Yolov5 model. With Python programming, an adaptive interpolation technique performs the resizing action.

3.8.1.2 *Data annotation*

Annotation is a machine learning method that labels data on photographs that constitute specified objects by putting a bounding box around each crop in the field. The images that have been resized are labeled to pick the proposed region, which is made up of separate crops. The weed regions in individual sub-images were labeled as rectangular bounding boxes utilizing the python labelling imager program (LabelImg). Five (5) annotators were employed in the labeling procedure. The annotator was trained to outline rectangular bounding boxes surrounding weed spots and the different crops on the farm.

3.8.1.3 *Splitting data*

About 223 aerial photographs made up the dataset used to process the algorithm. This dataset was subsequently divided into Training Set of 156 (70%), Validation Set of 45 (20%) and Test Set of 22 (10%) photographs. A training dataset is a subset of data used to train the classifier. The test set is a subset of data that may be examined utilizing the trained model and the validation set is used to assess the models' performances. Despite the fact that validation metrics are relatively low, the results of algorithms validated utilizing an external validation dataset simply make the validation extra reliable for the actual implementation in the field. Figure 3.5 presents the pre-processing workflow.

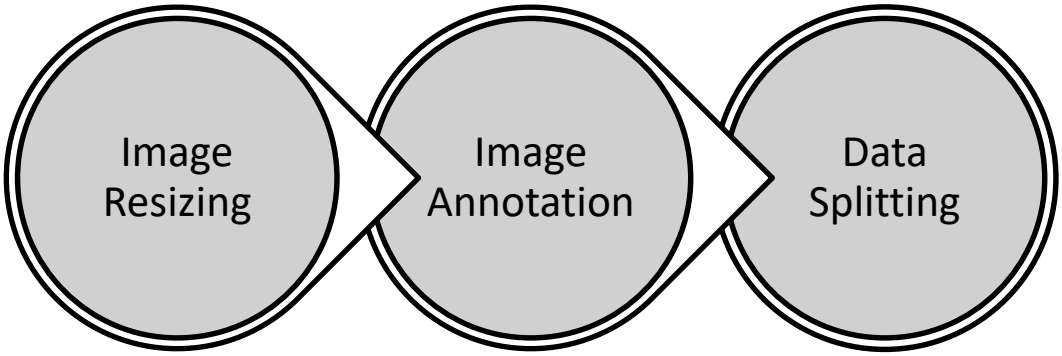

Figure 3.5: Workflow for pre-processing of YOLO v5 dataset

3.9 Training, Validation and Test of YOLO v5 Model

YOLOv5 uses a darknet framework. Google Colaboratory was employed to perform data processing and analysis (Colab). Because training a neural network requires a virtual computer to operate for 12 hours and requires little computing effort, Colab Free was employed for this segment of the study. A GPU R-80 and 16GB of RAM were used for learning and identification operations (NVIDIA GeForce GTX TITAN X). A workstation running Ubuntu 18.04 with GPU acceleration served as the operating system for the

virtual machine employed in the current investigation. Python 3.8 programming was used to code and complete all Colab analysis.

The dataset were assembled representing the images with labelled bounding boxes around the weeds and crops that are to be detected. All dataset were exported in the YOLOv5s format. In training the model, a number of arguments were passed such as defining the image size of 416 x 416, because the model was more sophisticated, a batch size of 16 photographs was employed. The dataset was then sub-divided into a training set of 156, Validation Set of 45 and a Testing Set of 22. Training epochs were set at 100, 300, 500, 600, 700 and 1000, 5 classes were set for the models classification, the dataset location was set and the training process was carried out using the pre-trained weights that the YOLO programmers made available. The information is initially uploaded into CSP (Cross Stage Partial Network) to retrieve attributes of weeds and crops after being submitted with all the image data. The Head component is ultimately utilized to report data like class, grades, position, and object size. The focus module is used in the Backbone stage to retrieve useful information features. In evaluating the YOLO v5 model performance, training losses and performance metrics are saved to Tensorboard and further to a logfile. Then inference is run with the trained weights on contents of "test/images" folder or logfile (i.e. which is used for making real-world predictions and classification). The expended time for 100 epochs was 4minute 62 seconds, 300 epochs was 11minutes 88seconds, 500 epochs was 18minutes 48seconds, 600 epochs was 22minutes 92seconds, 700 epochs was 25minutes 86seconds, and 1000 epochs was 38minutes 22seconds. Figure 3.6 depicts a pictorial workflow of the YOLO v5 model.

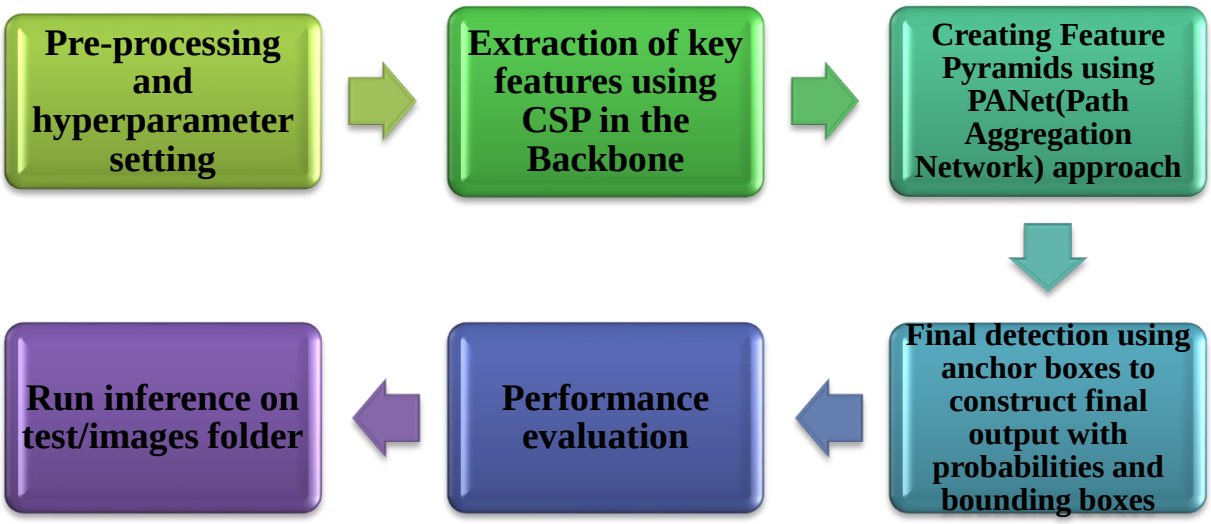

Figure 3.6: Process flow of YOLOv5

CHAPTER FOUR

4.0 RESULTS AND DISCUSIONS

4.1 Results and Discussions for the Faster RCNN

This section presents the findings achieved while employing the Faster Region-based Convolutional Neural Network with inception v2 model for weed detection and classification in a mixed farm over 10,000, 20,000, 100,000, 200,000, 242,000 epochs are addressed. The research was carried out on Google Colaboratory employing Python programming, mostly utilizing the Tensorflow library.

4.1.1 Training loss graphs

Figures 4.1-4.5 show all of the usual training loss graphs together with the dataset, and all the losses over all the the five (5) distinct epochs considerably decreased as the number of training epochs increased all through the training process. The training loss represents

how effectively the algorithm matches the training data (Bontonou *et al*., 2019). The steady drop in training losses from Figure 4.1 to Figure 4.5 merely shows that the model kept learning throughout the training session with an increment in the number of epochs to a maximum of 242,000, beyond which there was no more significant learning from the training data. The training loss curves in Figure 4.1 were partially reduced, indicating that the model was still learning. The training loss starts to decrease in Figure 4.2. Between Figure 4.3 to 4.5, the training loss curves declined significantly and smoothed out at 242,000 epochs, where it stabilized at 0. The total loss values are represented on the y axis, and the number of epochs is represented on the x axis (the number of epochs refers to how many instances the learning algorithm will run over the full training dataset). Figure 4.1 to Figure 4.5 were exported from a visualization tool-TensorBoard. Additionally, the training time over the training epochs of 10,000, 20,000, 100,000, 200,000 and 242,000 were 27.8minutes, 54minutes, 3.6hours, 7.9hours and 9.6hours using Google Colaboratory.

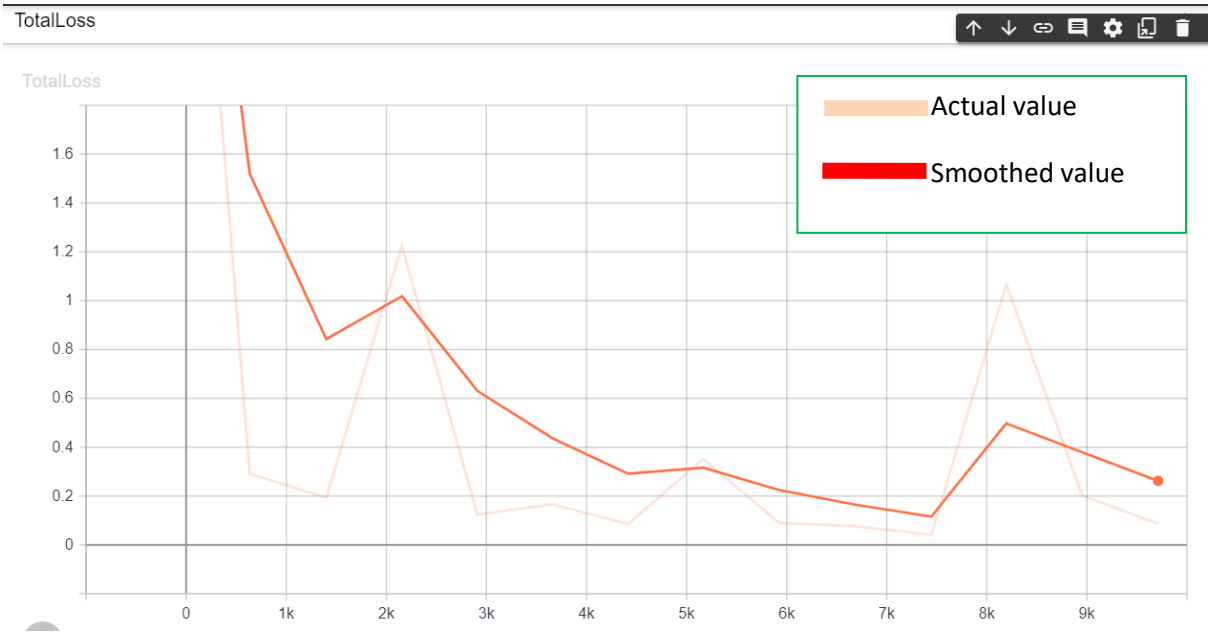

Figure 4.1: Total loss for 10,000 epochs

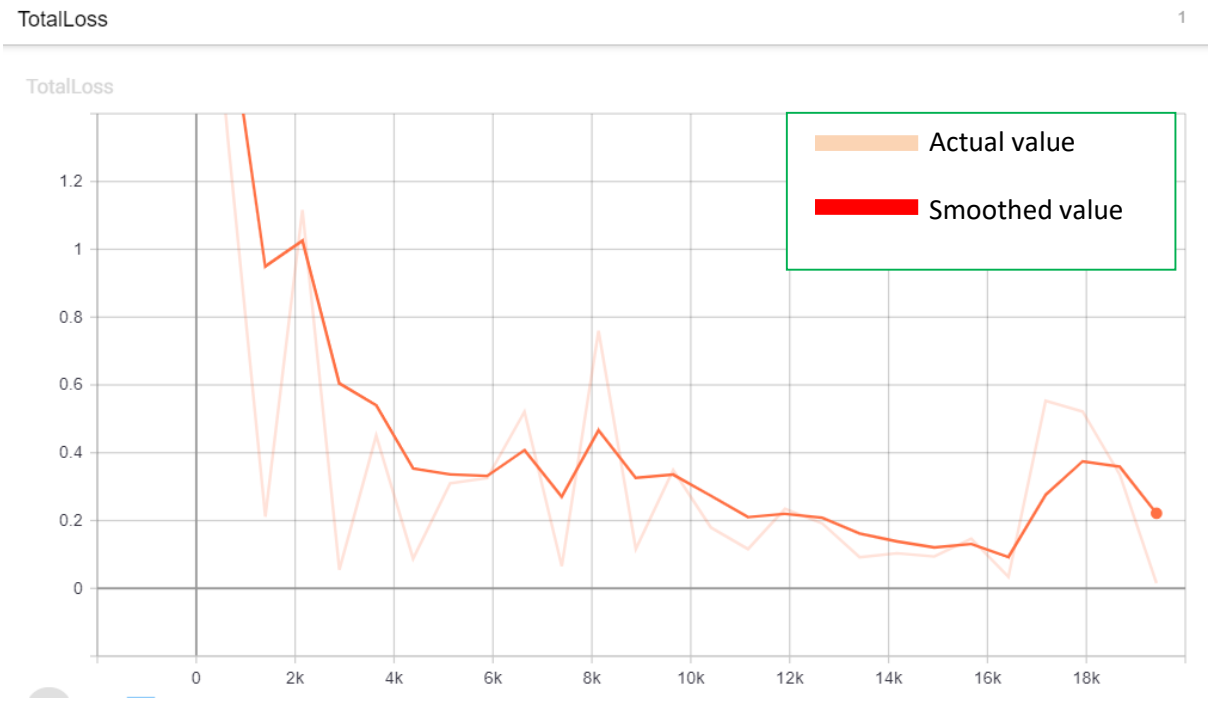

Figure 4.2: Total loss for 20,000 epochs

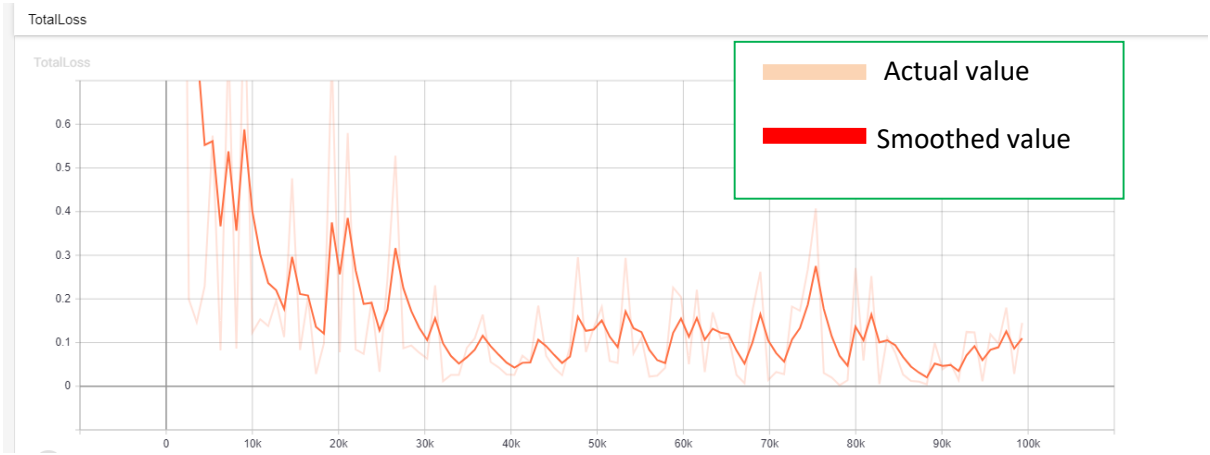

Figure 4.3: Total loss for 100,000 epochs

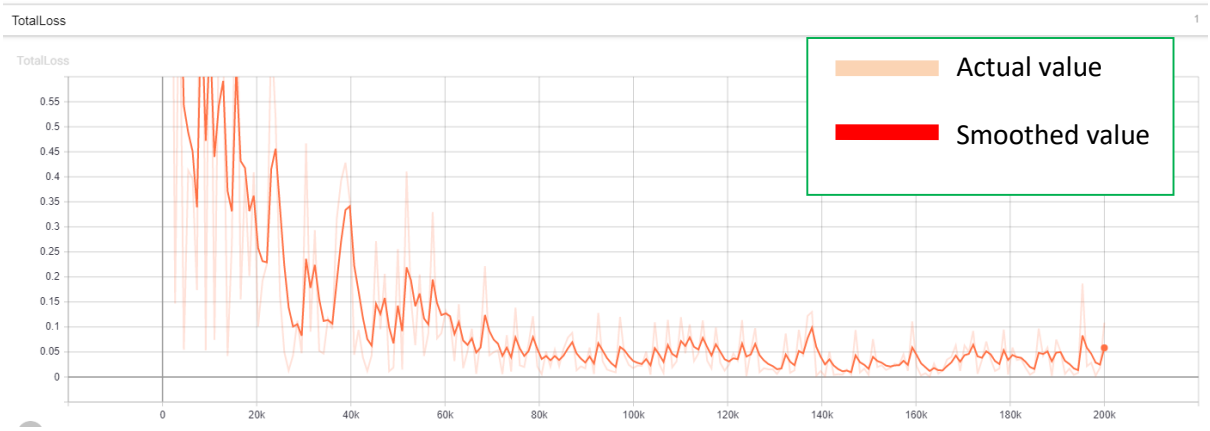

Figure 4.4: Total loss for 200,000 epochs

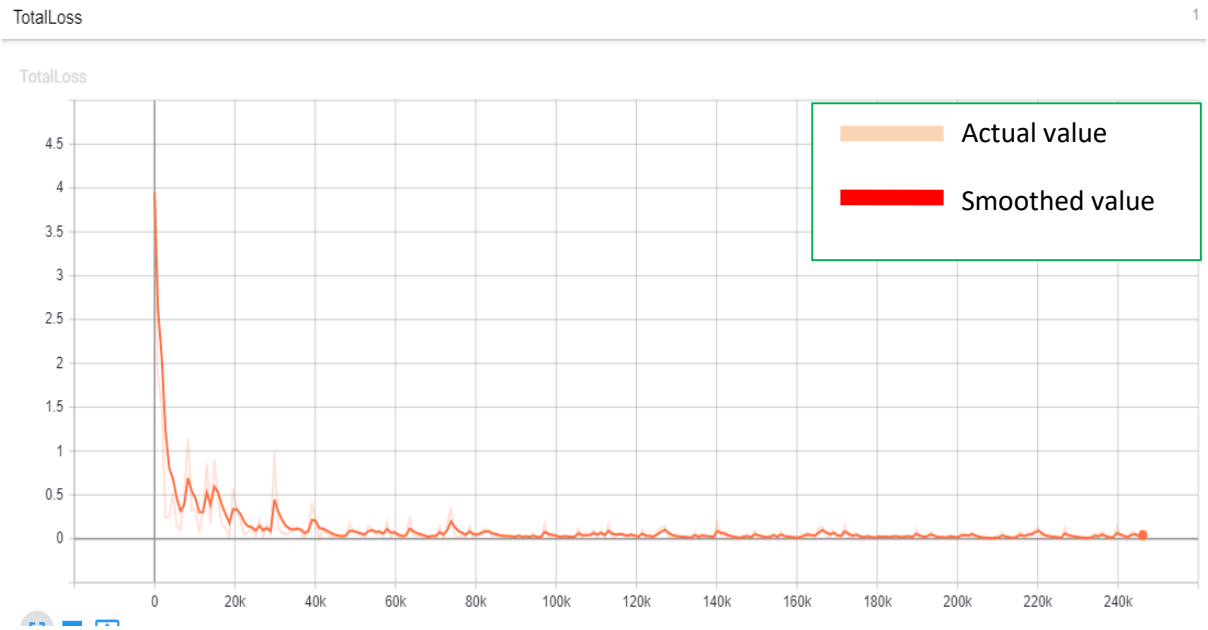

Figure 4.5: Total loss for 242,000 epochs

4.2 Confusion Matrix for 10,000 Epochs

The confusion matrices which is an evaluation of precision in Convolutional Neural Networks based Deep Learning as espunged by (Maxwell *et al.*, 2021), was explored for Weed, Sugarcane, Banana, Spinach, and Pepper identification using the Faster RCNN Deep Learning model spanning five (5) different epochs have been shown in Table 4.1, Table 4.3, Table 4.5, Table 4.7 and Table 4.9. The diagonal values depict the accurate

estimates. The maximum accuracy recorded was 98.4% after the model had been trained with a batch sizes of 32 at 200,000 epochs. With respect to the hyperparameter value of batch size of 32, with 10,000 epochs in Table 4.1, the model successfully classified 75 % annotations of sugarcane out of 191 annotations of sugarcane and just 1 % annotation was mis-classified whereas 24 % annotations remained unsorted. 90 % of the annotations of spinach out of 145 spinach annotations were successfully classified and only 1 % was misclassified while 9 % also remained unsorted. Also, 92 % annotations of pepper were successfully classified out of a total of 204 annotations of pepper while 6 % remained unsorted. Further, 88 % annotations of Banana out of 65 annotations of Banana were correctly classified and just 3 % were not properly classified while 9 % remained unsorted. Ultimately, the classifier accurately determined 8 % annotations of weed out of 372 annotations of weed and just 3 % annotations of weed were not properly classified while 89 % annotations of weed remained unsorted.

Table 4.1: Confusion matrix at 10,000 epochs

	Sugarcane	**Spinach**	**Pepper**	**Banana**	**Weed**	**Unsorted**	**Total**
Sugarcane	144	0	2	0	0	45	191
Spinach	0	130	2	0	0	13	145
Pepper	0	0	187	0	0	17	204
Banana	1	0	1	57	0	6	65
Weed	7	0	5	1	29	330	372

Unsorted	52	13	77	9	29	0	180
Total	204	143	274	67	58	411	1157

4.2.1 Precision and recall values for 10,000 epochs

In Table 4.2, the precision and recall values for 10,000 epochs are shown. The Precision value ranges from 0 (which means no precision) to 1 (which means perfect precision). Spinach had the highest precision of all classes (around 0.909), followed by banana, sugarcane, and pepper, whereas weed had the lowest precision (at approximately 0.500). Similar to precision, the Recall value ranges from 0 to 1. In terms of recall, "pepper" had the maximum value (about 0.917), followed by "spinach," "banana," "sugarcane," and "weed" (around 0.789), that had the lowest score. This indicates that while the model was able to identify positive samples of "spinach," "banana," "sugarcane," and "pepper," it could not identify as many positive samples of "weed".

Table 4.2: Precision and recall for 10,000 epochs

Category	Precision	Recall
Sugarcane	0.7058823529	0.7539267016
Spinach	0.9090909091	0.8965517241
Pepper	0.6824817518	0.9166666667
Banana	0.8507462687	0.8769230769

Weed	0.5000000000	0.0779569892

4.3 Confusion Matrix for 20,000 Epochs

Table 4.3 shows the confusion matrix that was generated for 20,000 epochs. The findings demonstrates that the algorithm was capable of correctly classifying 88% annotations of sugarcane out of 191 annotations of sugarcane, while 12% annotations of sugarcane remained unsorted. Likewise, out of 145 annotations of spinach, 97% were accurately classified, whereas the remaining 3% remained unsorted. In addition, out of 204 annotations of pepper, 95% were accurately classified, while 5% remained unsorted. Furthermore, out of 65 annotations of banana, 97% were accurately classified, while the remaining 3% remained unsorted. Ultimately, out of 372 annotations of weed, the algorithm correctly identified 52% annotations of weed, and just 0.5% annotations of weed were wrongly classified, leaving 47% of them unsorted. The accuracy of the weed classification at this epoch is just slightly better in comparison to the accuracy at 10,000 epochs.

Table 4.3: Confusion matrix for 20,000 epochs

	Sugarcane	**Spinach**	**Pepper**	**Banana**	**Weed**	**Unsorted**	**Total**
Sugarcane	169	0	0	0	0	22	191
Spinach	0	141	0	0	0	4	145
Pepper	0	0	193	0	0	11	204

Banana	0	0	0	63	0	2	65
Weed	1	1	0	0	195	175	372
Unsorted	24	6	14	4	96	0	144
Total	194	148	207	67	291	214	1121

4.3.1 Precision and recall values for 20,000 epochs

Table 4.4 displays the precision and recall values that were derived for 20,000 epochs. The Precision and Recall values for spinach were the maximal, coming in at roughly 0.952 and 0.972, correspondingly. This is closely preceded by “Banana”, “pepper”, “sugarcane” and “weed”. This indicates that while the model was capable of detecting positive samples for “sugarcane”, “spinach”, “pepper” and “banana”, it did so less frequently for "weed".

Table 4.4: Shows the precision and recall for 20,000 epochs

Category	Precision	Recall
Sugarcane	0.8711340206	0.8848167539
Spinach	0.9527027027	0.9724137931
Pepper	0.9323671498	0.9460784314
Banana	0.9402985075	0.9692307692
Weed	0.6701030928	0.5241935484

4.4 Confusion Matrix for 100,000 Epochs

The confusion matrix and the precision and recall values generated when the model was trained at 100,000 epochs, correspondingly, are shown in Tables 4.5 and 4.6. The outcome demonstrates that of 191 annotations of sugarcane, the classifier was capable of accurately classifying 100% of them. Additionally, 100% of the 204 annotations of pepper and 100% of the 145 annotations of spinach were both accurately classified. Similarly, out of 65 banana annotations, 100% were classified accurately. Ultimately, out of 372 weed annotations, the model correctly identified 98% of them, whereas the remaining 2% remained unsorted.

Table 4.5: Confusion matrix for 100,000 epochs

	Sugarcane	**Spinach**	**Pepper**	**Banana**	**Weed**	**Unsorted**	**Total**
Sugarcane	191	0	0	0	0	0	191
Spinach	0	145	0	0	0	0	145
Pepper	0	0	204	0	0	0	204
Banana	0	0	0	65	0	0	65
Weed	0	0	0	0	363	9	372
Unsorted	1	0	2	1	14	0	18
Total	192	145	206	66	377	9	995

4.4.1 Precision and recall values for 100,000 epochs

The class with the highest possible precision was "spinach," preceded by "pepper," "banana," and "weed," in that sequence, as shown by the results displayed in Table 4.6. The classes containing the highest recall values are "sugarcane," "spinach," "pepper," "banana," and "weed," in that sequence. In spite of the fact that "weed" again produced the lowest precision and recall values at this epoch, it was found that these values had greatly improved particularly in comparison to the outcomes of the model's training at 20,000 epochs, reaching roughly 0.96 and 0.98, correspondingly. This indicates that the classifier was successful in identifying weed-positive samples at this point in time and that training tends to improve when more training epochs are added.

Table 4.6: Precision and recall values for 100,000 epochs

Category	Precision	Recall
Sugarcane	0.9947916667	1.0
Spinach	1.0	1.0
Pepper	0.9902912621	1.0
Banana	0.9848484848	1.0
Weed	0.9628647215	0.9758064516

4.5 Confusion Matrix for 200,000 Epochs

Table 4.7 shows the confusion matrix that was generated after the network was trained at 200,000 epochs, and Table 4.8 shows the precision and recall values that were also generated at this epoch. At 200,000 epochs, the classifier was found to be 100% accurate in classifying 191 annotations of sugarcane, 100% accurate in classifying 145 annotations of spinach, and 100% accurate in classifying 204 annotations of pepper. Additionally, all 65 annotations of banana were accurately classified, making 100% of them bananas. Ultimately, out of 372 weed annotations, the model correctly identified 99% of them, and just 0.8% remained unsorted. The trend of the results demonstrates that though the model has categorized all the annotated photographs of the different crops at 100,000 epochs successfully, it continues to get better at classifying weed as the epoch advances.

Table 4.7: Shows the confusion matrix for 200,000 epochs

	Sugarcane	**Spinach**	**Pepper**	**Banana**	**Weed**	**Unsorted**	**Total**
Sugarcane	191	0	0	0	0	0	191
Spinach	0	145	0	0	0	0	145
Pepper	0	0	204	0	0	0	204
Banana	0	0	0	65	0	0	65
Weed	0	0	0	0	369	3	372
Unsorted	2	0	3	1	7	0	13

Total	193	145	207	66	376	3	990

4.5.1 Precision and recall values for 200,000 epochs

Based on the precision values found in Table 4.8, it was determined that the classes with the highest precision were "spinach" and "banana," then "sugarcane," "pepper," and finally "weed," that had increased to 0.98. Additionally, the classes having the best recall were "sugarcane," "spinach," "pepper," "banana," and finally "weed," which also considerably improved to 0.9919, showing the classifier was capable of detecting positive samples of "weed" over time.

Table 4.8: Precision and recall for 200,000 epochs

Category	Precision	Recall
Sugarcane	0.9896373057	1.0
Spinach	1.0	1.0
Pepper	0.9855072464	1.0
Banana	1.0	1.0
Weed	0.9813829787	0.9919354839

4.6 Confusion Matrix for 242,000 Epochs

Tables 4.9 and 4.10 exhibit the outcome of the model's training at 242,000 epochs. The classifier flattened out and the weeds' precision started declining at 242,000 epochs. This meant that the classifier was no longer picking up new information from the training dataset, hence it was not necessary to go beyond 242,000 epochs. The confusion matrix is shown in Table 4.9, whereas Table 4.10 displays the precision and recall figures. With the exception of weed, all of the annotated crops could be effectively classified by the classifier at this time. Of the 372 annotations of weed, the model correctly identified 99%, meaning that 0.5% of the weed annotations were misclassified.

Table 4.9: Confusion matrix for 242,000 epochs

	Sugarcane	**Spinach**	**Pepper**	**Banana**	**Weed**	**Unsorted**	**Total**
Sugarcane	191	0	0	0	0	0	191
Spinach	0	145	0	0	0	0	145
Pepper	0	0	204	0	0	0	204
Banana	0	0	0	65	0	0	65
Weed	0	0	0	0	370	2	372
Unsorted	3	1	3	1	18	0	26

Total	194	146	207	66	388	2	1003

4.6.1 Precision and recall values for 242,000 epochs

Table 4.10's precision and recall values show that the classifier was starting to flatten out since the accuracy values started to decline, with the exception of "banana," that remained relatively stable. Although the recall value only slightly increased from 0.991 to 0.995, the precision value of "weed" decreased from 0.98 to 0.95, indicating that the training for classification and detection had achieved a saturation point during which moment the loss curve flattened out. This suggests that adding more epochs than 242,000 won't result in any substantial improvements in the classification and detection of weeds.

Table 4.10: Precision and recall for 242,000 epochs

Category	Precision	Recall
Sugarcane	0.9845360824	1.0
Spinach	0.9931506849	1.0
Pepper	0.9855072464	1.0
Banana	1.0	1.0
Weed	0.9536082474	0.9946236559

4.7 Performance of Faster RCNN Model Showing the Cumulative Result from the Accuracy Metrics.

The overall accuracy was 52.6 %, 67.9 %, 97.3 %, 98.4 %, and 97 % for 10,000 epochs, 20,000 epochs, 100,000 epochs, 200,000 epochs, and 242,000 epochs, respectively. This indicates that significant improvements in the network's performance have been continuous as it trained with increment in epochs, particularly when the training epochs were extended gradually from 10,000 to 200,000 even as the batch size for the algorithm remained constant. The number of instances the learning algorithm will cycle through the full training dataset is known as an epoch (Brownlee, 2018).

The cumulative performance was then reported at 10,000 epochs with an average precision of 73 %, an average recall of 70.4 %, and an F1 score of 71.7 %, and at 20,000 epochs with an average precision of 87.3 %, an average recall of 85.9 %, and an F1 score of 86.6 %. When the training epoch was changed to 100,000, it was found that the average average precision was 98.7 %, the average recall was 99.5 %, and the F1 score was 99.1 %; however, when the training epoch was changed to 200,000, the average precision was 99.1 %, the average recall was 99.8 %, and the F1 score was 99.4 %. Lastly, when the training epoch was raised to 242,000, the average precision, average recall, and F1 score were generated and are displayed in Table 4.11 to be 98 %, 99.9 %, and 99.1%, correspondingly. The corresponding times for 10,000 epochs, 20,000 epochs, 100,000 epochs, 200,000 epochs and 242,000 epochs were 27.8minutes, 54minutes, 3.6hours, 7.9hours and 9.6hours.

Table 4.11: Performance of faster RCNN inception v2 model showing the cumulative result from the accuracy metrics.

Epochs	Accuracy	Average Precision	Average Recall	F1 score

10,000	0.5259615385	0.7296402565	0.7044050317	0.7168006091
20,000	0.6788581624	0.8733210947	0.8593466592	0.8662175648
100,000	0.9728643216	0.986559227	0.9951612903	0.9908415891
200,000	0.9838383838	0.9913055062	0.9983870968	0.9948336993
242,000	0.9720837488	0.9833604522	0.9989247312	0.9910814888

The accuracy, precision, F1 score and recall for the five epochs are compared in Table 4.11. Once contrasted to other epochs, the precision, accuracy, and F1 scores of inceptionV2 at 200,000 epochs and 32 batch size produced the best results, which makes it the perfect training epoch for accurate weed and crop identification. Using random testing photographs, the same InceptionV2 model was evaluated, and the outcomes are depicted in Plate V down to Plate IX. The findings collected indicate that the classifier had a very high degree of confidence in its ability to effectively detect the weed even amongst the sugarcane, spinach, pepper, and banana.

Additionally, the performance of weed detection and classification outcomes was also examined. The classification accuracy of the deployed model for 10,000 epochs was 52.5 %, the "weed precision achieved 50 %", the "weed recall achieved 7.7 %", and the "F1 score also achieved 71.6 %". Furthermore, the model achieved a "classification accuracy of 67.8%", "weed precision of 67%," "weed recall of 52.4%," and an F1 score of 85.9% over 20,000 epochs. Likewise, 100,000 epochs produced "classification accuracy results of 97.2 %", "weed precision" results of 96.2 % was achieved, "weed recall" results of 97.5 % was achieved, and an F1 score of 99 % was also achieved, whereas 200,000 epochs produced "classification accuracy results of 98.3 %", "weed precision" results of

98.1 % was achieved, "weed recall" results of 99.1 % was achieved, and an F1 score of 99.4 % was also achieved. Lastly, 242,000 epochs produced "classification accuracy results of 97 %","weed precision of 95 % was achieved, "weed recall of 99 % was achieved and a F1 scores of 99 % was also achieved over the metrics.

In addition, it was identified that the automatic "weed" classifier had the maximum precision (98.4%) at 200,000 epochs before accuracy began to decline at 242,000 epochs and fell to 97%, whereas the "weed" precision likewise reduced from 0.98 at 200,000 epochs to 0.95 at 242,000 epochs.

4.8 Classification on Testing Dataset

The Faster Region based Convolutional Neural Network model was used to identify and categorize crop images over 10,000, 20,000, 100,000, 200,000, and 242,000 epochs. Examples of typical photographic outcomes of the classifier are shown in Plates V to IX, demonstrating the model's capability to detect and categorize weeds from several other crops for each testing image on the test set. A predicted bounding box of weeds and sugarcane crops is shown in Plate V at 10,000 epochs. Weeds here were likewise only partly detected in the photograph using predicted bounding boxes and an accuracy of 62% due to their tiny size and the overlap of sugarcane leaves. The percentages of weeds, spinach, banana crops, and sugarcane within 20,000 epochs are displayed in Plate VI. The accuracy for banana crop is 99 %, spinach is 99 %, and sugarcane was about 97%. In the case where the weeds were apparent, the model correctly identified the annotated weeds in the farm photograph with 99% accuracy. At 100,000 epochs in Plate VII, the classifier had a weed detection rate of almost 98%. The classifier demonstrated its ability to detect the weeds in the sugarcane regions to a high degree of 99% in the following plots, which is displayed at 200,000 epochs in Plate VIII, and this was largely facilitated by the

classifier's learning ability with increase in the number of epochs. The detection and classification of weeds began to become less accurate in Plate IX at 242,000 epochs, and it was recognized that the model had achieved a saturated state. The accuracy decreased from 99% at 200,000 epochs to 96% at 242,000 epochs. When evaluated with crops in a mixed farm, the classifier is capable of constructing a decision boundary to accurately identify weeds from crops with a low frequency of misclassifications.

Plate V: The classifier’s confidence in detecting and classifying weed classes was 62 % in accuracy at 10,000 epochs from the image

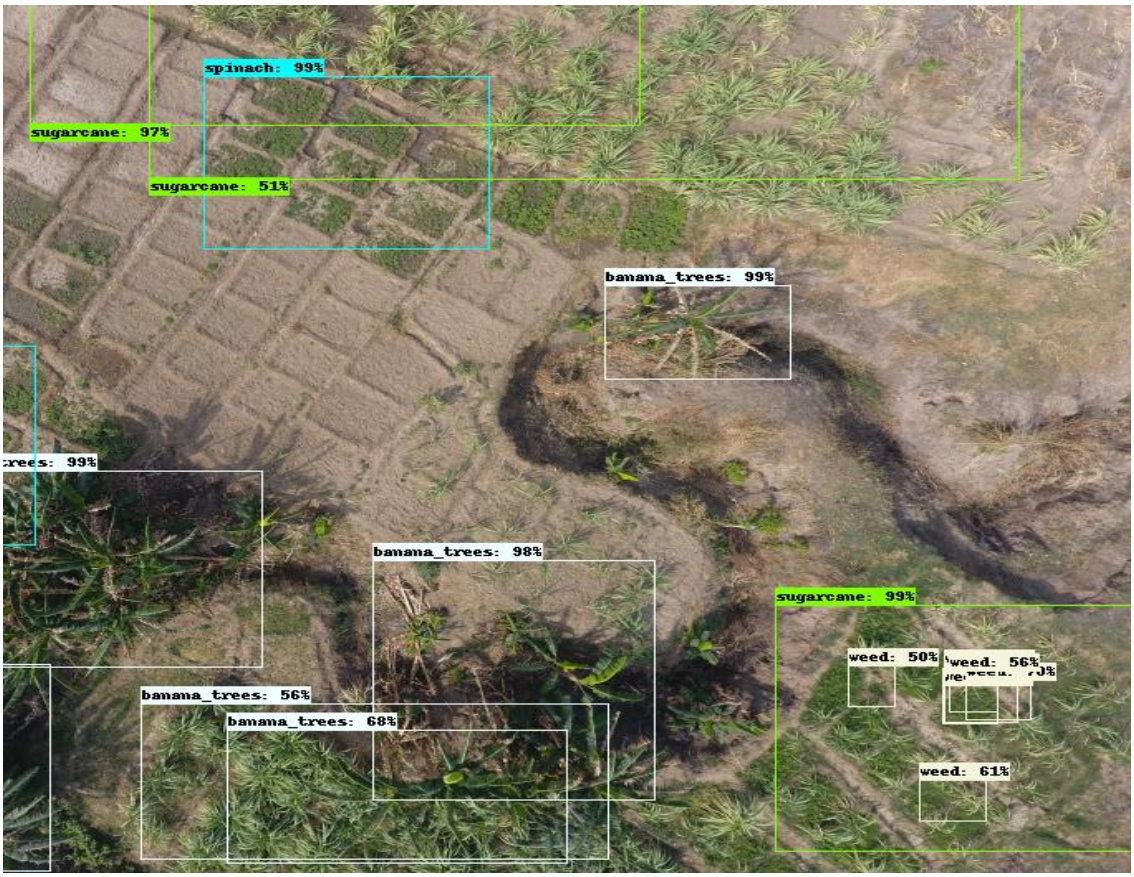

Plate VI: The classifier's confidence in detecting and classifying weed classes was 67 % in accuracy at 20,000 epochs from the image

Plate VII: The classifier's confidence in detecting and classifying weed classes was 98 % in accuracy at 100,000 epochs from the image

Plate VIII: The classifier's confidence in detecting and classifying weed classes was 99 % in accuracy at 200,000 epochs from the image

Plate IX: The classifier's confidence in detecting and classifying weed classes was 96 % in accuracy at 242,000 epochs from the image

4.9 Results and Discussions for the YOLO v5

In this section, the results obtained using the YOLO V5 architecture for classification of weeds in a mixed irrigation farm over 100, 300, 500, 600, 700 and 1000 epochs are addressed. Python programming was used to conduct the investigations on Google Colab, mostly utilizing the Darknet framework.

The dataset were assembled representing the images with labelled bounding boxes around the weeds and crops that are to be detected. All dataset were exported in the YOLOv5s format. In training the Yolov5 model, a number of arguments were passed such as defining the image size of 416 x 416 and employing a batch size of 16 photographs due to the complexity of the model. The dataset were splitted into a training set made up of

70 % of the dataset, a validation set made up of 20 %, and a testing set made up of 10 %. Training epochs were set at 100, 300, 500, 600, 700 and 1000, 5 classes were set for the models classification, the dataset location was set and the training applied to the pre-trained weights made available by the YOLO developers. The expended time for 100 epochs was 4minute 62 seconds, 300 epochs was 11minutes 88seconds, 500 epochs was 18minutes 48seconds, 600 epochs was 22minutes 92seconds, 700 epochs was 25minutes 86seconds, and 1000 epochs was 38minutes 22seconds.

4.9.1 Training loss graphs from YOLOv5

The training loss per network epoch was determined so as to evaluate how well the network training process performed. The network went through 100, 300, 500, 600, 700 and 1000 epochs, it can be seen that the training losses decreased all through from Figure 4.6 down to Figure 4.11 meaning that the model was learning. The model keeps learning as it goes through even more epochs, which leads to less training loss in later epochs. In Figure 4.6, the loss curve started declining at a considerable rate. From 300 epochs in Figure 4.7, the author observed a loss that is mostly constant from Figure 4.7 to Figure 4.11. This indicates however that the network is learning with increasing accuracy, which shows that the training loss was presumably minimal, as illustrated graphically in Figure 4.7 to Figure 4.11. From Figure 4.9 to Figure 4.10, the loss curve had no significant improvement meaning the training curve flattened out at 600 epochs in Figure 4.9. The lower the loss becomes, the better the model performance will be. The number of epochs is shown on the x-axis, while the loss value is shown on the y-axis. The graphs were extracted from Tensorboard Visualization.

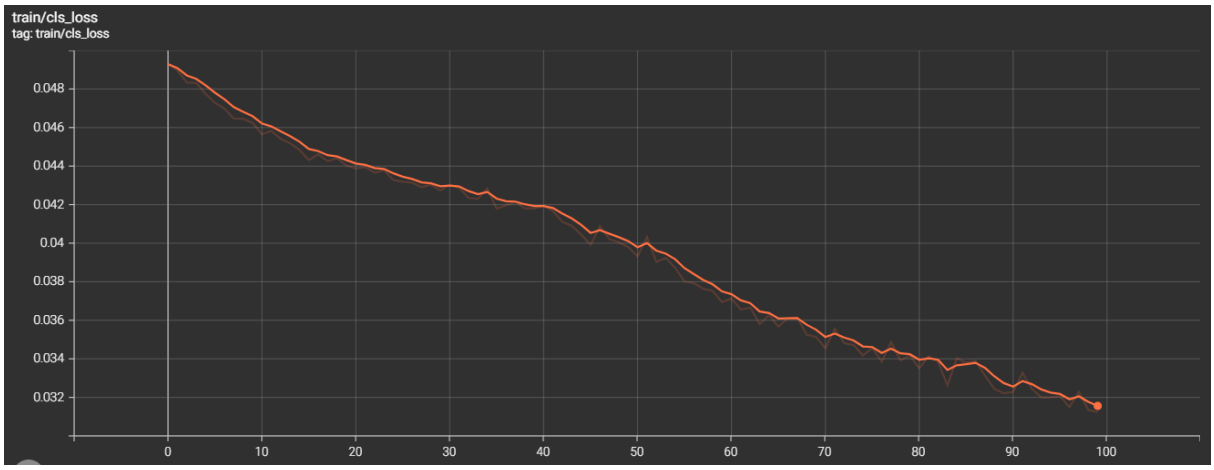

Figure 4.6: The train/classification loss for 100 epochs

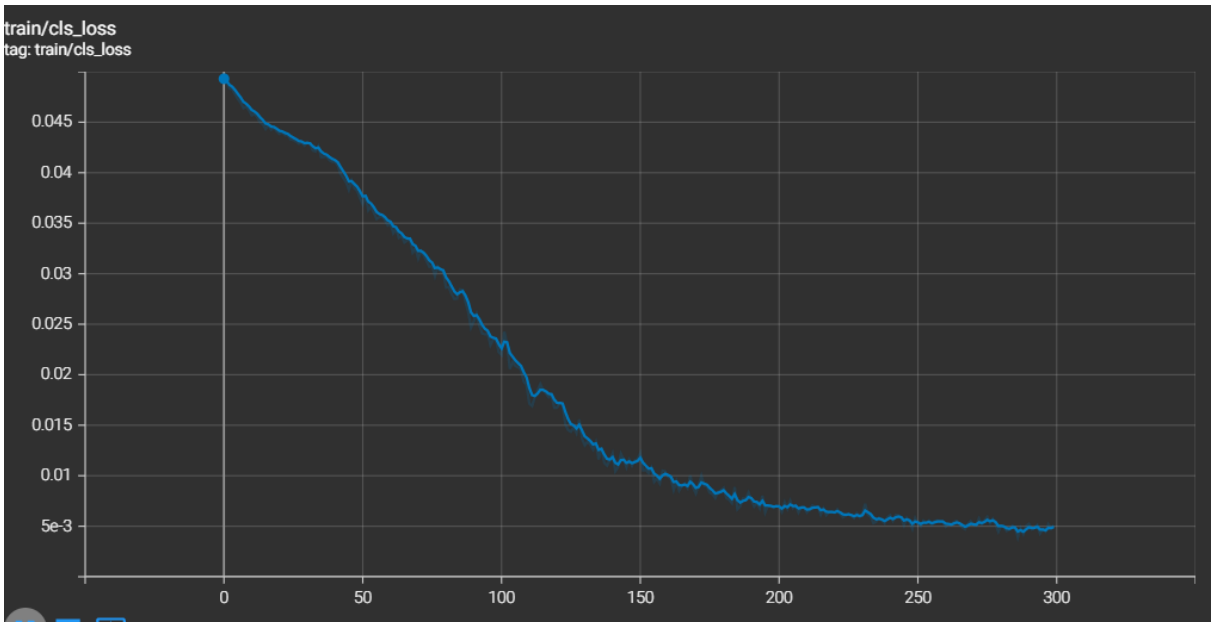

Figure 4.7: The train/classification loss for 300 epochs

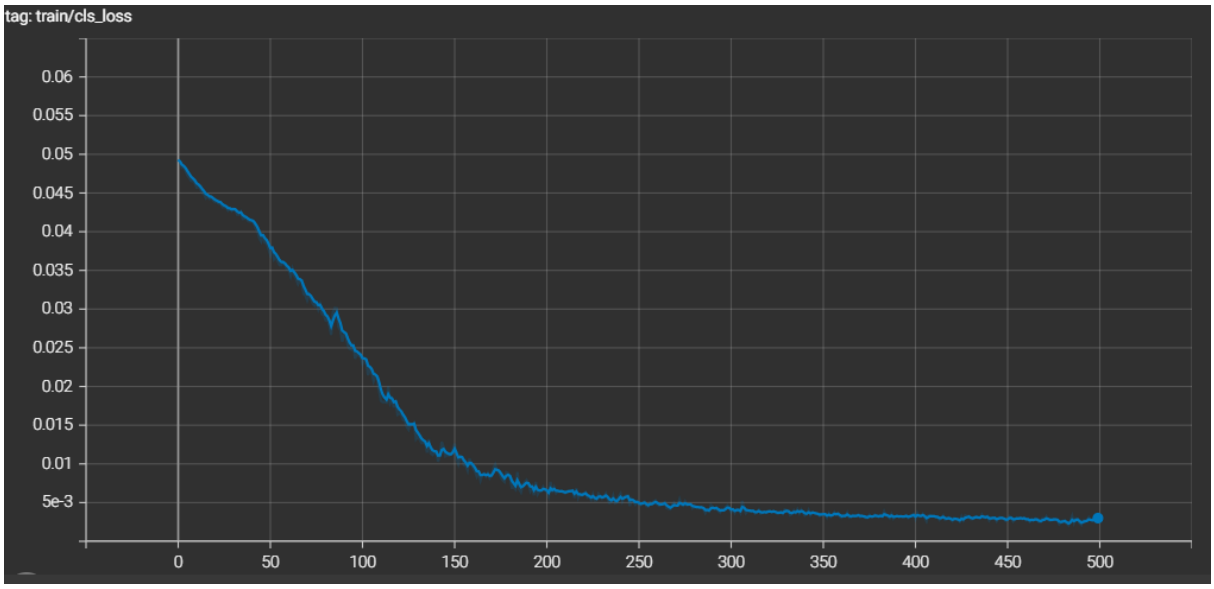

Figure 4.8: The train/classification loss for 500 epochs

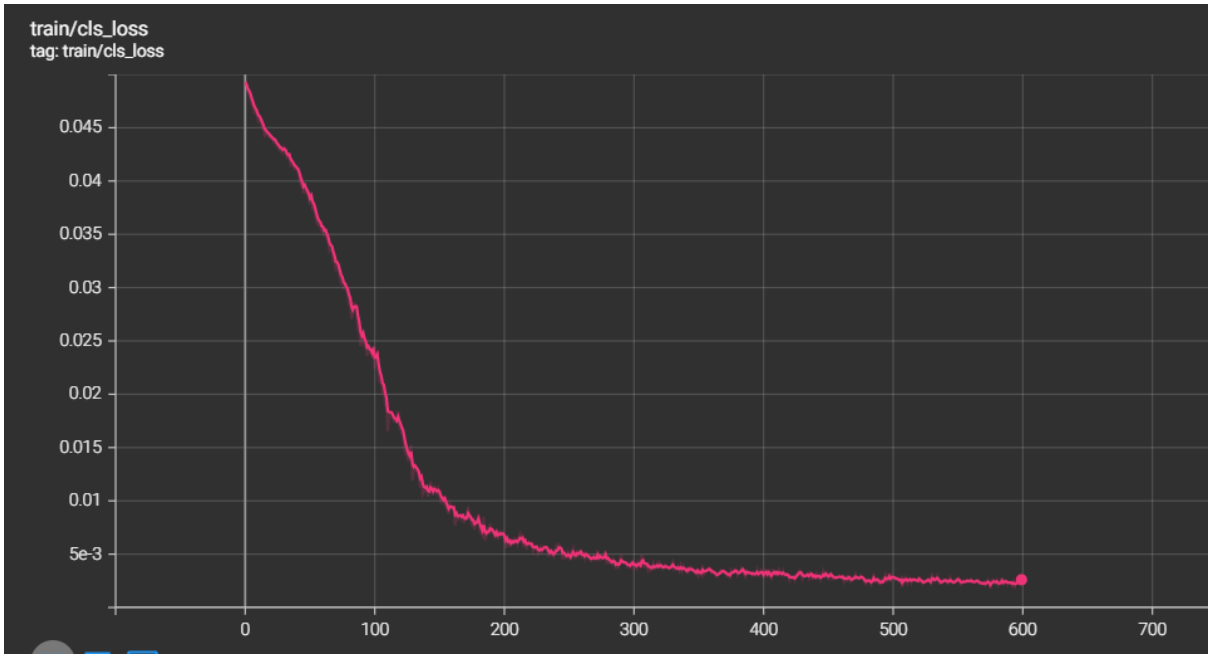

Figure 4.9: The train/classification loss for 600 epochs

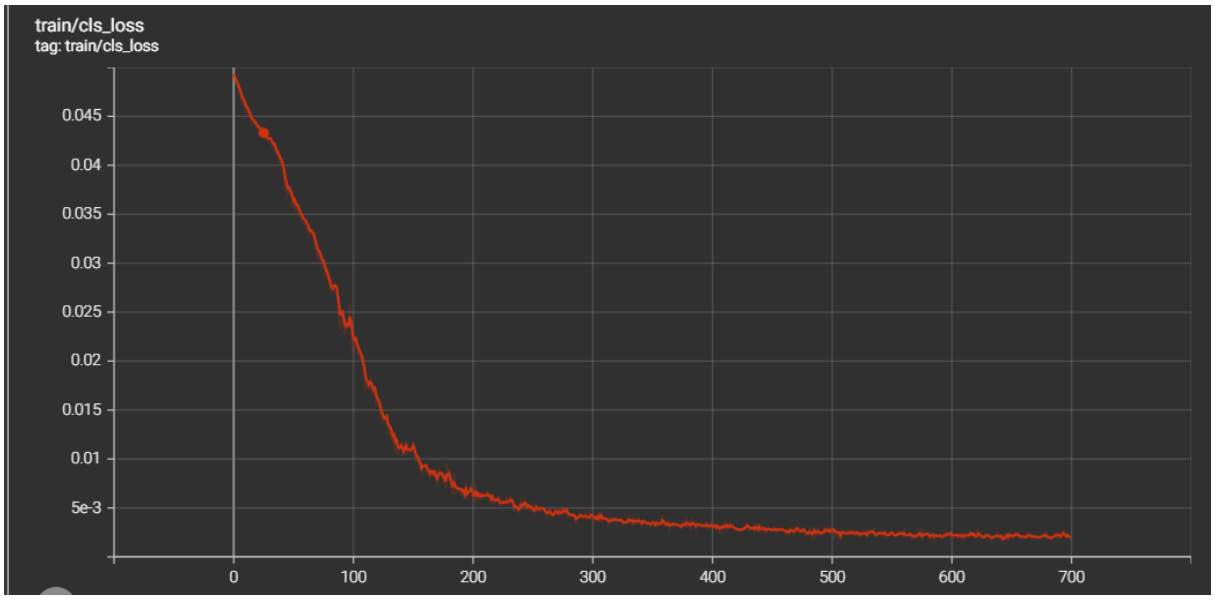

Figure 4.10: The train/classification loss for 700 epochs

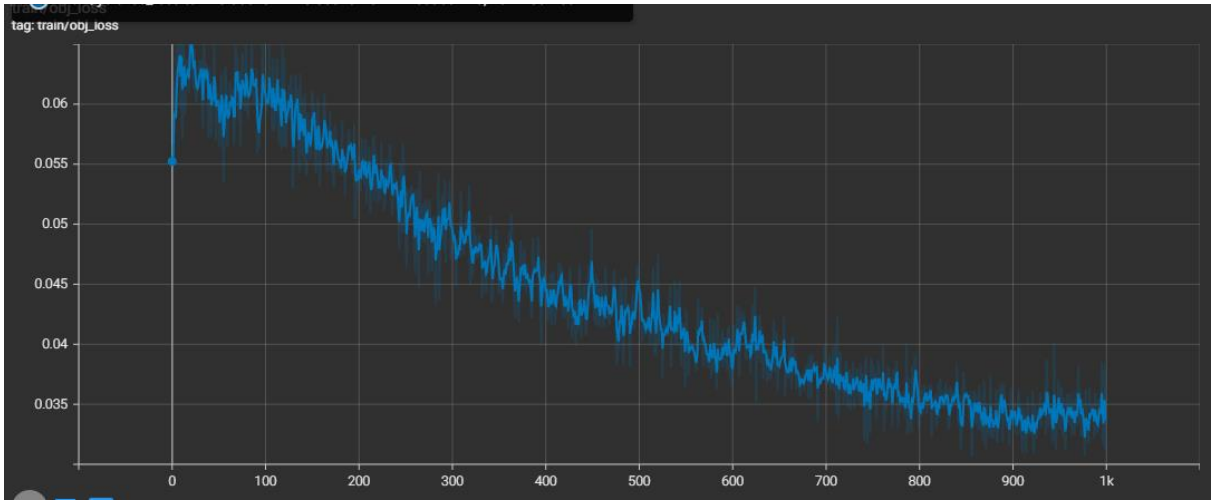

Figure 4.11: The train/classification loss for 1000 epochs

4.9.2 Validation graphs from YOLO v5

The developed model's performance was evaluated using the validation dataset. The step by step validation losses of the model are shown in Figure 4.12 to Figure 4.17. As observed from the plots, the models converge gradually and poorly at 100 epochs in Figure 4.12, but converge better as the loss function diminishes while it trains in Figure 15 and remain constant down to Figure 4.17. This is in line with the theory that the model

is constantly tweaking its parameters and picking up relevant feature of the crops and weeds without “overfitting” when the network learns the training data well, but performs poorly on the generated data and “underfitting” when the algorithm is not able to model either the training data or testing data. Figure 4.12 to Figure 4.17 indicates a possible optimal case. The number of epochs is represented on the x-axis and y-axis represents the validation loss values.

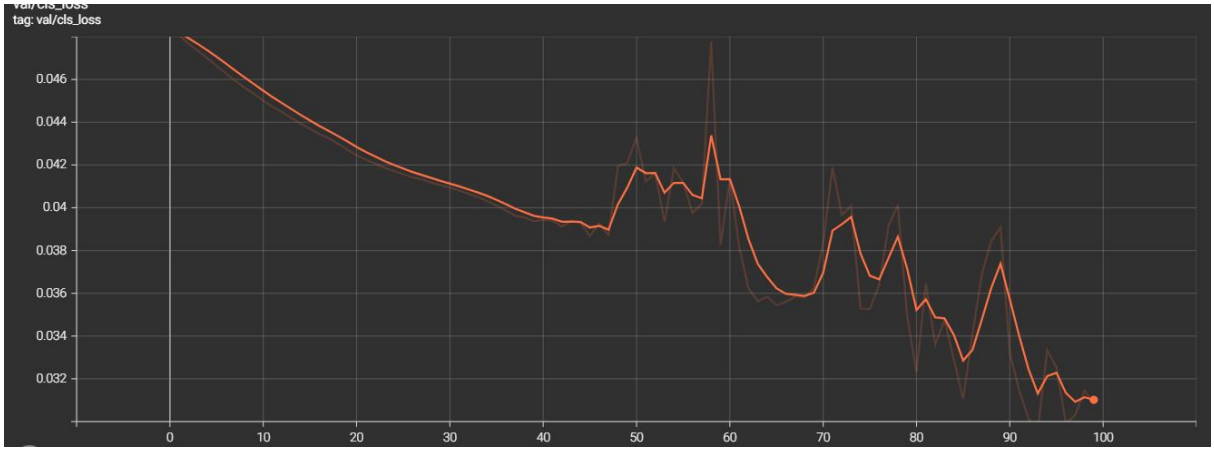

Figure 4.12: Validation loss for 100 epochs

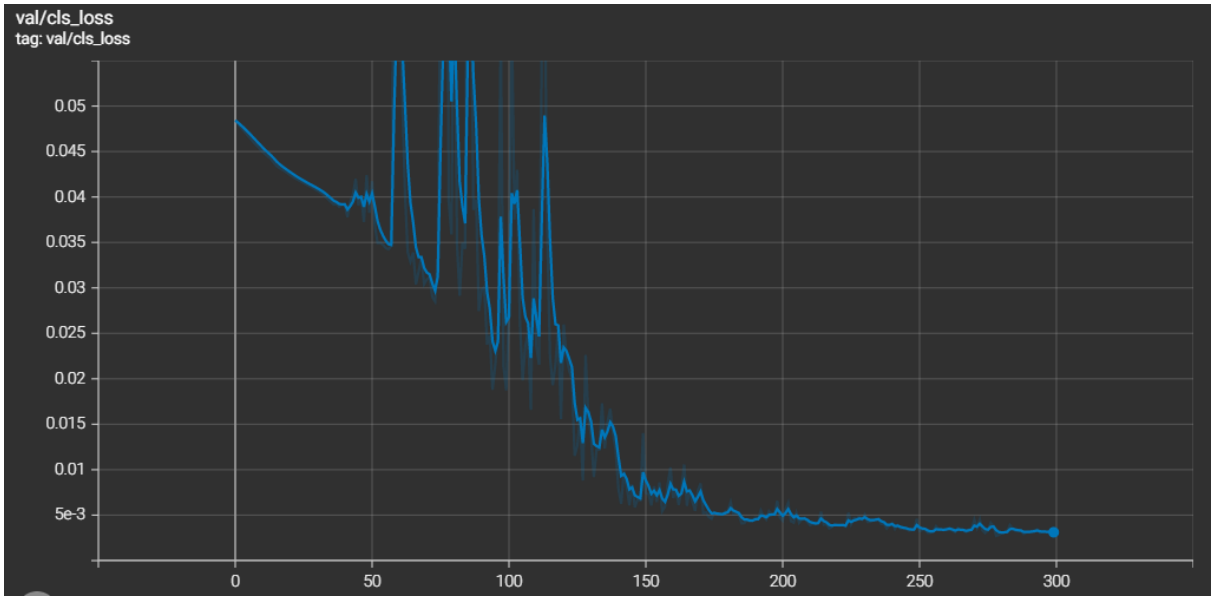

Figure 4.13: Validation loss for 300 epochs

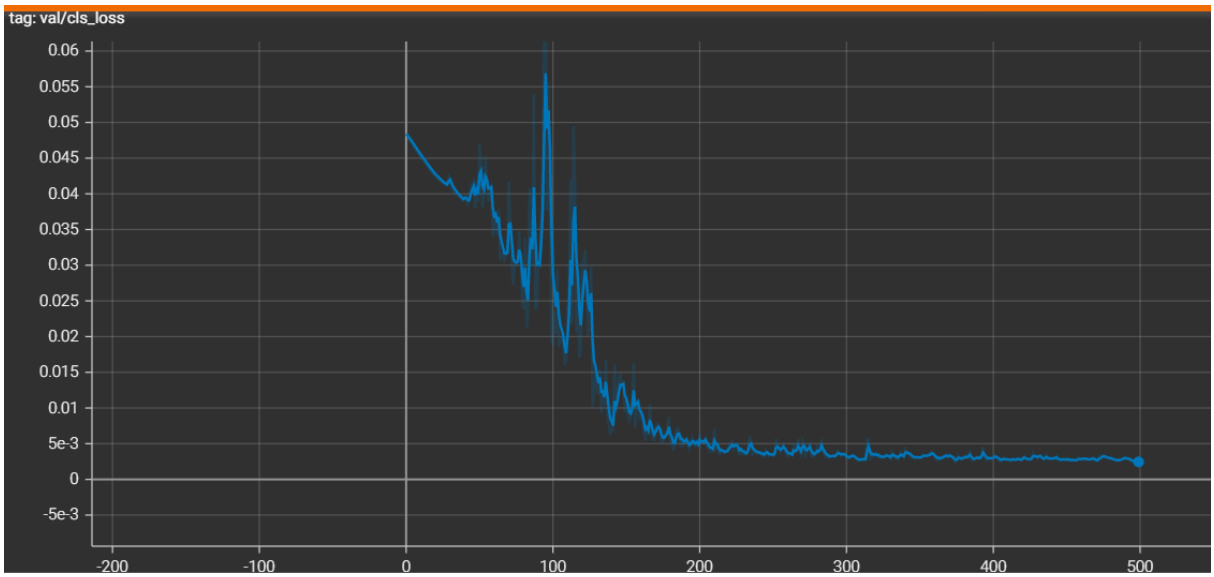

Figure 4.14: Validation loss for 500 epochs

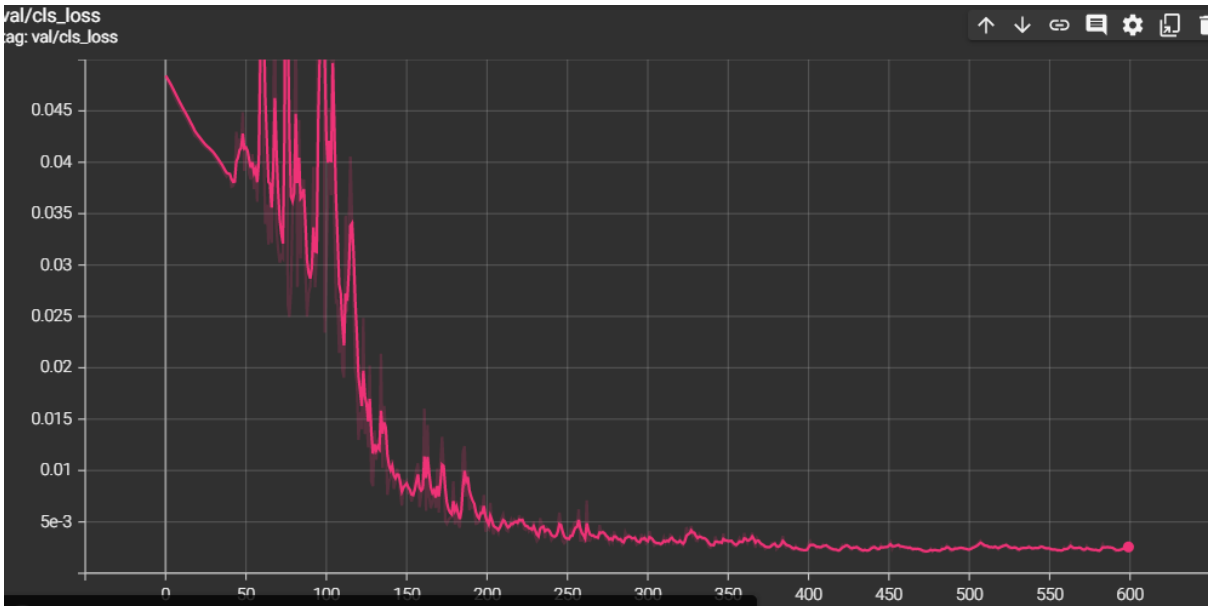

Figure 4.15: Validation loss for 600 epochs

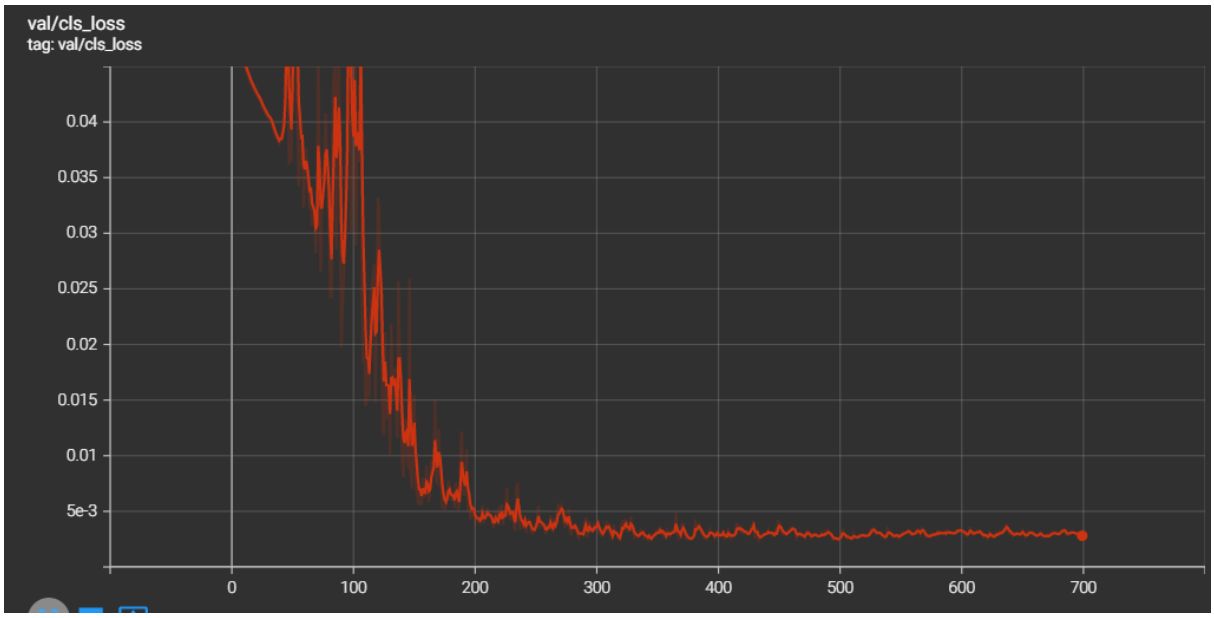

Figure 4.16: Validation loss for 700 epochs

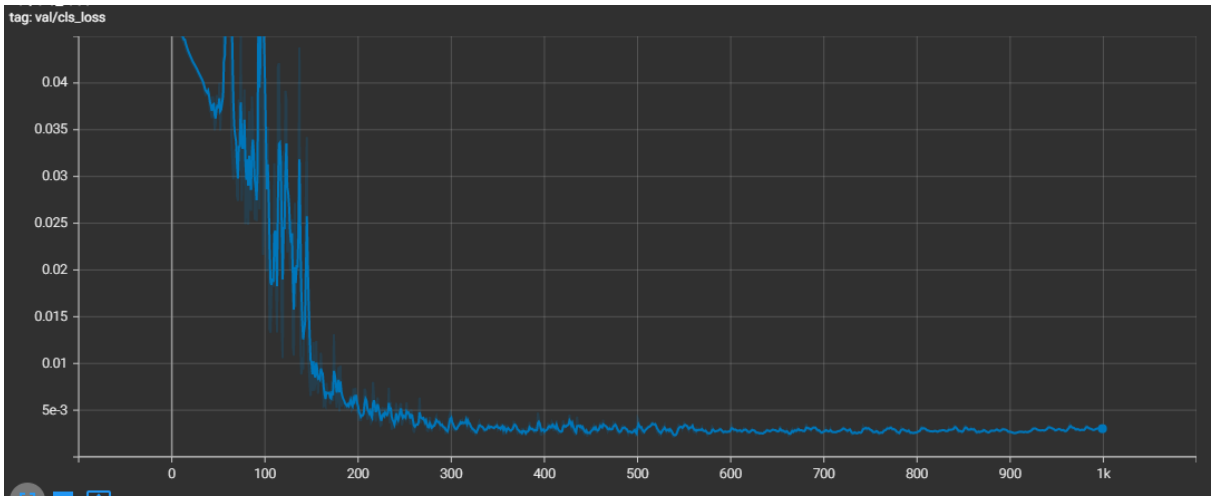

Figure 4.17: Validation loss for 1000 epochs

4.10 Confusion Matrix for 100 Epochs

The Confusion Matrix for multi - class classification is shown in Figure 4.18 (in this case, 5 classes). The number of TP elements for each class is displayed on the diagonal (top left to bottom right) as follows: 53% of all objects in the class of banana trees, 32% of all objects in the class of spinach, 10% of all objects in the class of sugarcane, and 1% of all objects in the class of weeds were properly categorized. Additionally, 13% of all banana-

class objects were mistakenly predicted as sugarcane-class objects, another 33 % of objects of the weed class were tagged as unidentified by YOLO (unsorted). 3 % of all objects of spinach class were misclassified as sugarcane and 65% were classified as unknown. 5 % of all objects of sugarcane class were misclassified as spinach while 86 % were classified as unknown, and 99 % of all objects of the weed class was classified as unknown and were not categorized into any class by the classifier.

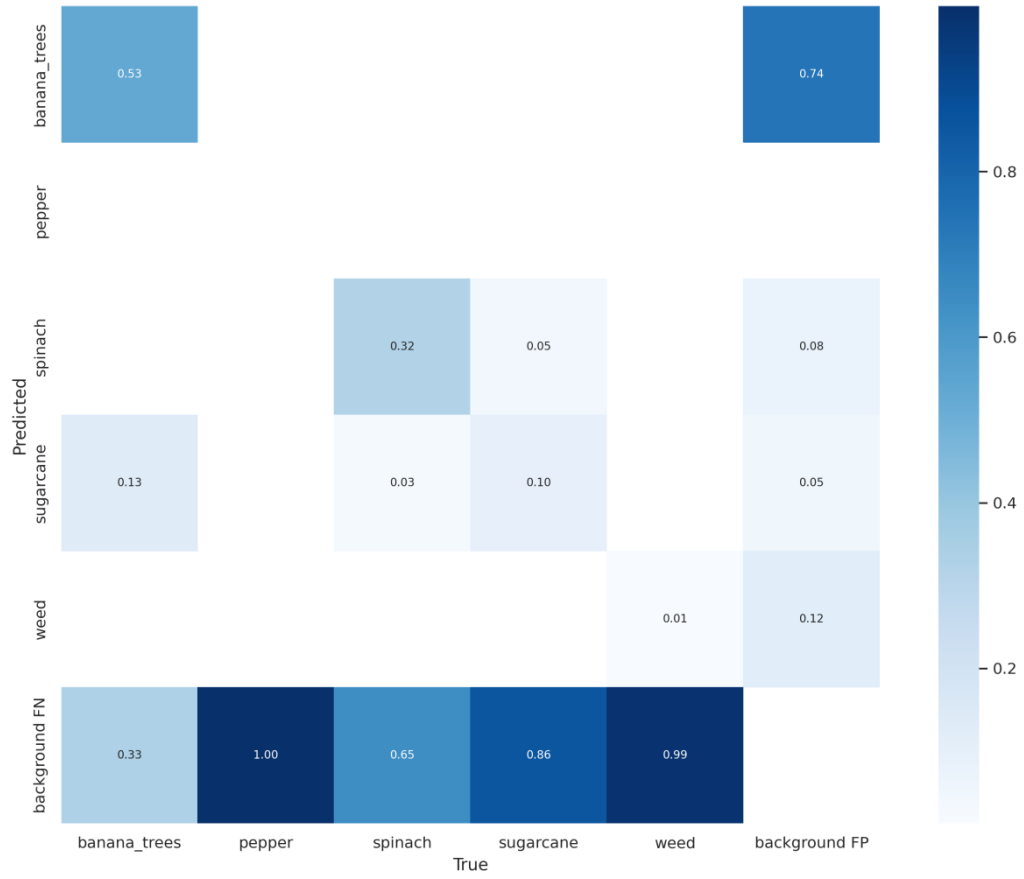

Figure 4.18: Confusion matrix for 100 epochs

4.10.1 Precision and recall values and graphs for 100 epochs

In Table 4.12, the value of Precision is within 0 (absence of precision) and 1.0 (ideal precision). The most precise class was ‘pepper’ (an approximate of 0.947) preceded with

‘spinach’, ‘banana crop’, ‘sugarcane crop’ and ‘weed plants’ (at approximately 0.0504) as the least. The Recall value ranges from 0 (absence of precision) to 1.0 (ideal precision). The class with the best Recall was ‘banana’ (an approximate of 0.615) preceded with ‘spinach’, ‘sugarcane crop’, ‘weed’ and ‘pepper crop’(at approximately 0.0167) as the least indicating that less positive samples of "weed" and "pepper" were found, but the classifier was capable of identifying positive samples of "spinach crops," "banana crops," and "sugarcane crops".

Table 4.12: Precision and recall for 100 epochs

Category	Precision	Recall
Sugarcane	0.141	0.116
Spinach	0.378	0.375
Pepper	0.947	0.0167
Banana	0.156	0.615
Weed	0.0504	0.013

Figure 4.19, depicts the models precision curve metric that estimates the proportion of accurate bounding box predictions. While Figure 4.20 depicts the recall curve metric that determines the percentage of the actual bounding box that was successfully predicted. In Figure 4.19, the model began improving swiftly from around 40 epochs all through to 98 epochs where it slightly dropped in precision on the precision curve while in Figure 4.20, the recall curve also improved from 15 epochs which means the model is gradually learning. The y-axis demonstrates the value of precision for Figure 4.19 and the x-axis

displays the different ranges of epochs while the y-axis depicts the recall value for Figure 4.20 and x-axis displays the different ranges of epochs.

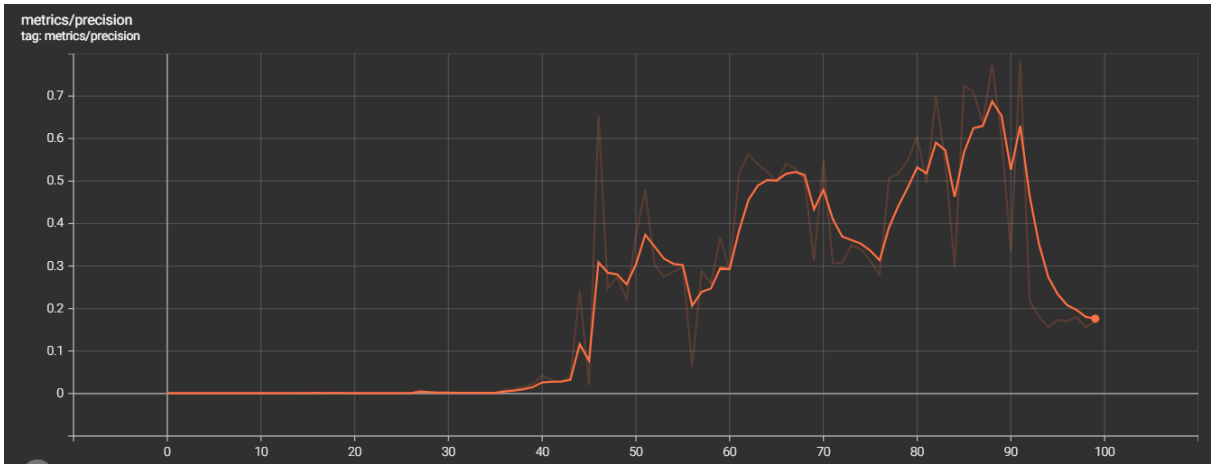

Figure 4.19: Depicts the precision metrics curve at 100 epochs

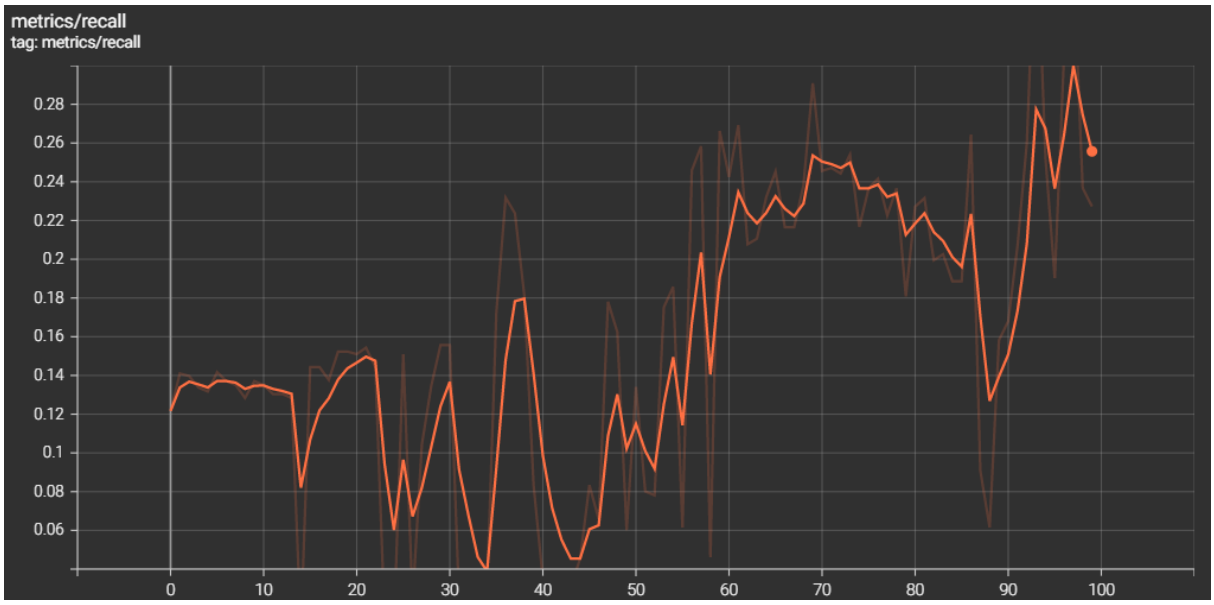

Figure 4.20: Depicts the recall metrics curve at 100 epochs

4.11 Confusion Matrix for 300 Epochs

The Confusion Matrix for multiclass classification is displayed in Figure 4.21 (in this case, 5 classes). The number of TP combinations for every class is displayed on the diagonal, from top left to bottom right: 92% of all objects in the class of banana trees,

70% of all objects in the class of peppers, 97% of all objects in the class of spinach, 84% of all objects in the class of sugarcane, and 45% of all objects in the class of weed were properly categorized. Also, 8 % of all objects of the banana class were found by YOLO as unknown, another 2 % of all objects of the pepper class were misclassified spinach and 28 % were found by YOLO as unknown. 3 % of all objects of spinach was found by YOLO as unknown, 16 % of all objects of sugarcane class were categorized as unknown. 55 % of all the objects of weed class were classified as unknown and were not categorized by the classifier into any class.

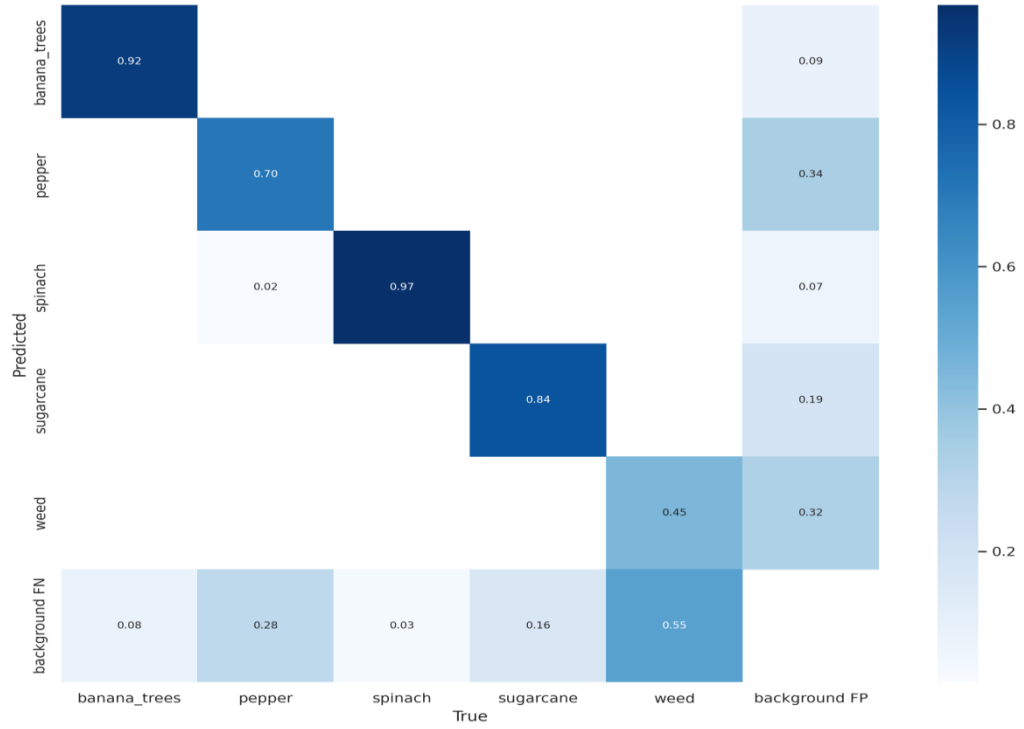

Figure 4.21: Confusion matrix for 300 epochs

4.11.1 Precision and recall values and graphs for 300 epochs

The precision and recall values are displayed in Table 4.13. The value of Precision ranges from 0 (absence of precision) to 1.0 (for an ideal precision). The most accurate class in terms of Precision and Recall was ‘spinach’ (at an approximate of 0.809 and 0.927)

correspondingly. Preceded with “banana crop”, “sugarcane”, “pepper crop” and “weed plant” with Precision and Recall (at approximately 0.458 and 0.319) as the least. This indicated that while the classifier was capable of identifying positive samples of "spinach," "banana," and "sugarcane", it did not identify as many positive samples of "weed".

Table 4.13: Precision and recall for 300 epochs

Category	Precision	Recall
Sugarcane	0.653	0.721
Spinach	0.809	0.927
Pepper	0.541	0.650
Banana	0.685	0.923
Weed	0.458	0.319

Figure 4.22, depicts the models precision curve metric that determines the proportion of accurate bounding box predictions. While Figure 4.23 depicts the recall curve metric that determines the percentage of the actual bounding box that was successfully predicted. In Figure 4.22, the model began improving swiftly from around 100 epochs all through to 300 epochs on the precision curve and in Figure 4.23, the recall curve also improved swiftly all the way to 300 epochs which means the model was learning. The precision and recall capture the model performance, so the higher they are the better the model becomes. The y-axis demonstrates the value of precision for Figure 4.22 and the x-axis displays the

different ranges of epochs while the y-axis depicts the recall value for Figure 4.23 and x-axis shows the different ranges of epochs.

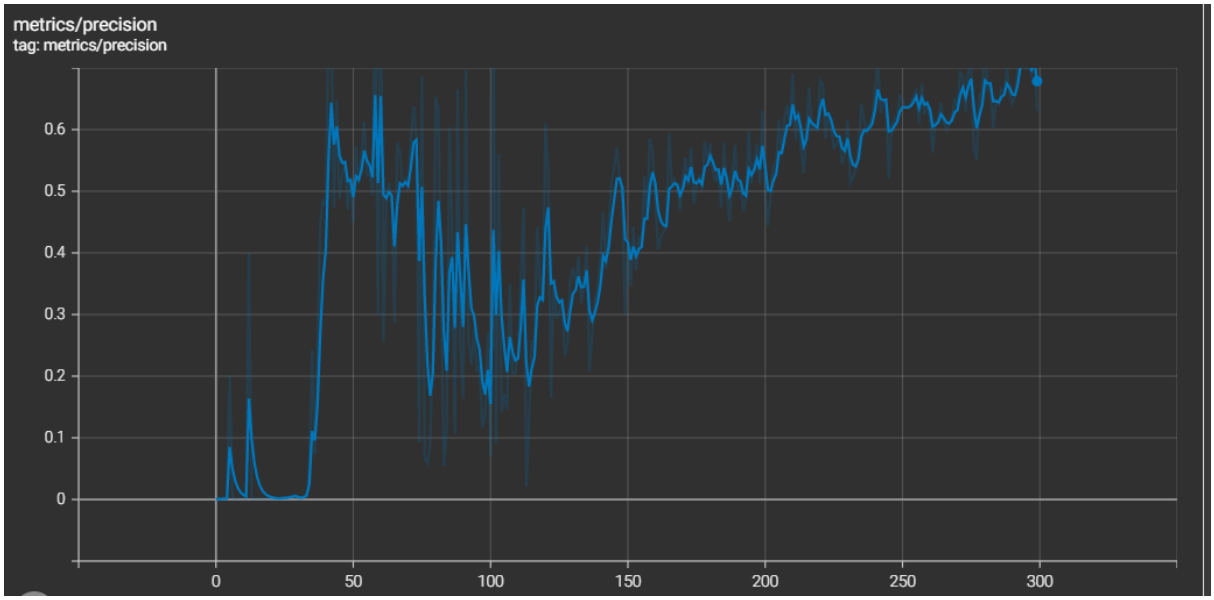

Figure 4.22: Depicts the precision metrics curve at 300 epochs

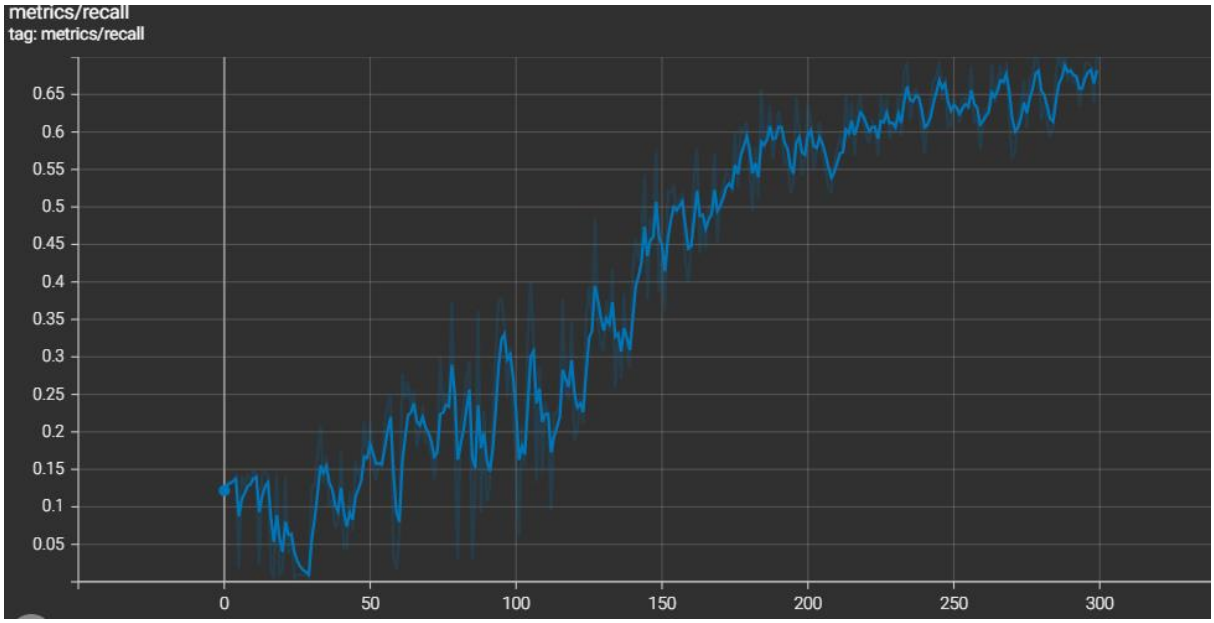

Figure 4.23: Depicts the recall metrics curve at 300 epochs

4.12 Confusion Matrix for 500 Epochs

The Confusion Matrix for multi - class classification is shown in Figure 4.24 (in this case, 5 classes). The number of TP elements for each class is displayed on the diagonal (top left to bottom right) as follows: 85% of all items in the class of banana trees, 77% of all objects in the class of pepper, 94% of all objects in the class of spinach, 83% of all objects in the class of sugarcane, and 54% of all objects in the class of weed were properly categorized. Additionally, 15 % of all objects of the banana class were found by YOLO as unknown, another 23 % of all objects of the pepper class were found by YOLO as unknown. 6 % of all objects of the spinach class were discovered by YOLO as unidentified, 17 % of all objects of sugarcane class were categorized as unknown. 1 % of all the objects of weed class were classified as sugarcane and 45 % of all objects of weed class were found by YOLO as unknown and had not been categorized into any class by the classifier.

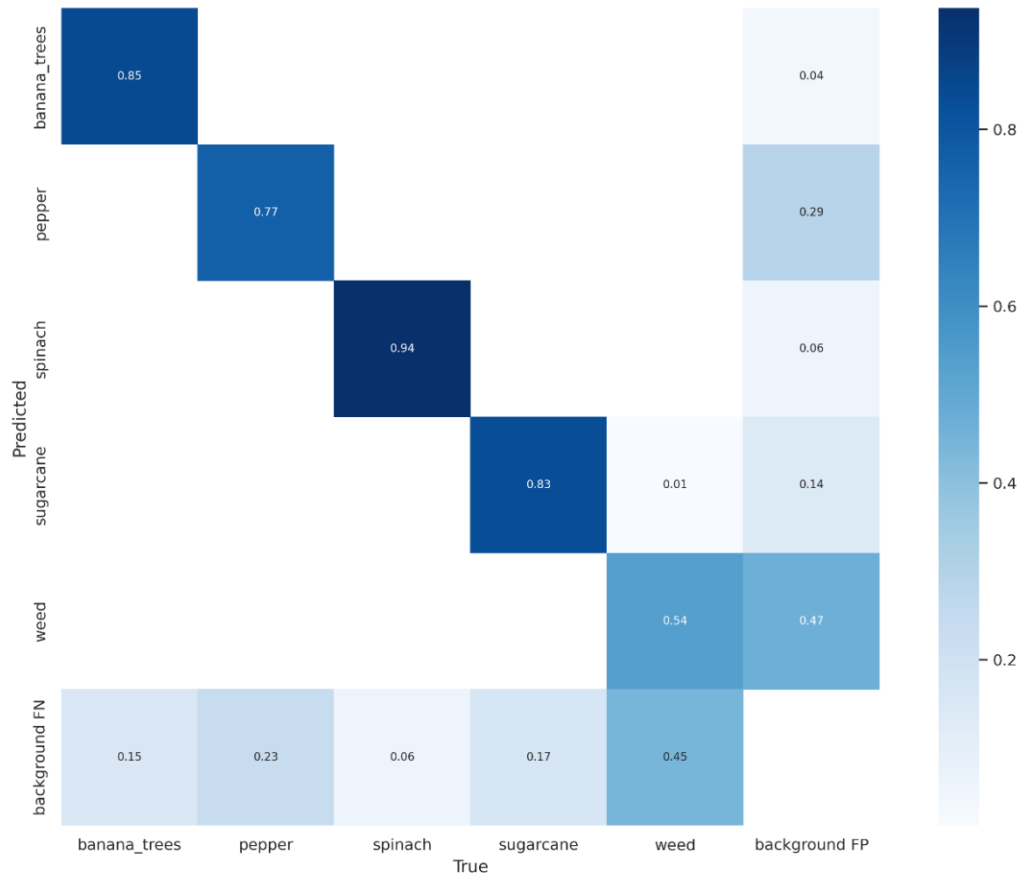

Figure 4.24: Confusion matrix for 500 epochs

4.12.1 Precision and recall values and graphs for 500 epochs

As observed in Table 4.14, the most accurate class in terms of Precision and Recall was observed to be ‘spinach’ (at an approximate of 0.902 and 0.875) followed by ‘sugarcane’, ‘weed’, ‘banana’ and ‘pepper’ (at approximately 0.560) as the least. Then the Recall values followed in the order of ‘sugarcane’, ‘banana’, ‘pepper’ and ‘weed’ (at approximately 0.269) as the least meaning that the classifier was capable of identifying Positive samples for “spinach crops”, “banana crops” and “sugarcane crops” but it did not identify many positive samples of “weed plants” but with great improvement in “weed” Precision initially from ‘0.458’ at 300 epochs to ‘0.747’ at 500 epochs.

Table 4.14: Precision and recall for 500 epochs

Category	Precision	Recall
Sugarcane	0.876	0.825
Spinach	0.902	0.875
Pepper	0.560	0.483
Banana	0.734	0.769
Weed	0.747	0.269

Figure 4.25, depicts the models precision curve metric that determines the proportion of accurate bounding box predictions. While Figure 4.26 depicts the recall curve metric that determines the percentage of the actual bounding box that was successfully predicted. In Figure 4.25, the model began improving swiftly from around 100 epochs all through to 500 epochs on the precision curve and in Figure 4.26, the recall curve also improved swiftly all the way to 500 epochs which means the model was learning. The precision and recall capture the model performance, so the higher they are the better the model becomes. The y-axis demonstrates the value of precision for Figure 4.25 and the x-axis displays the different ranges of epochs while the y-axis depicts the recall value for Figure 4.26 and x-axis displays the different ranges of epochs.

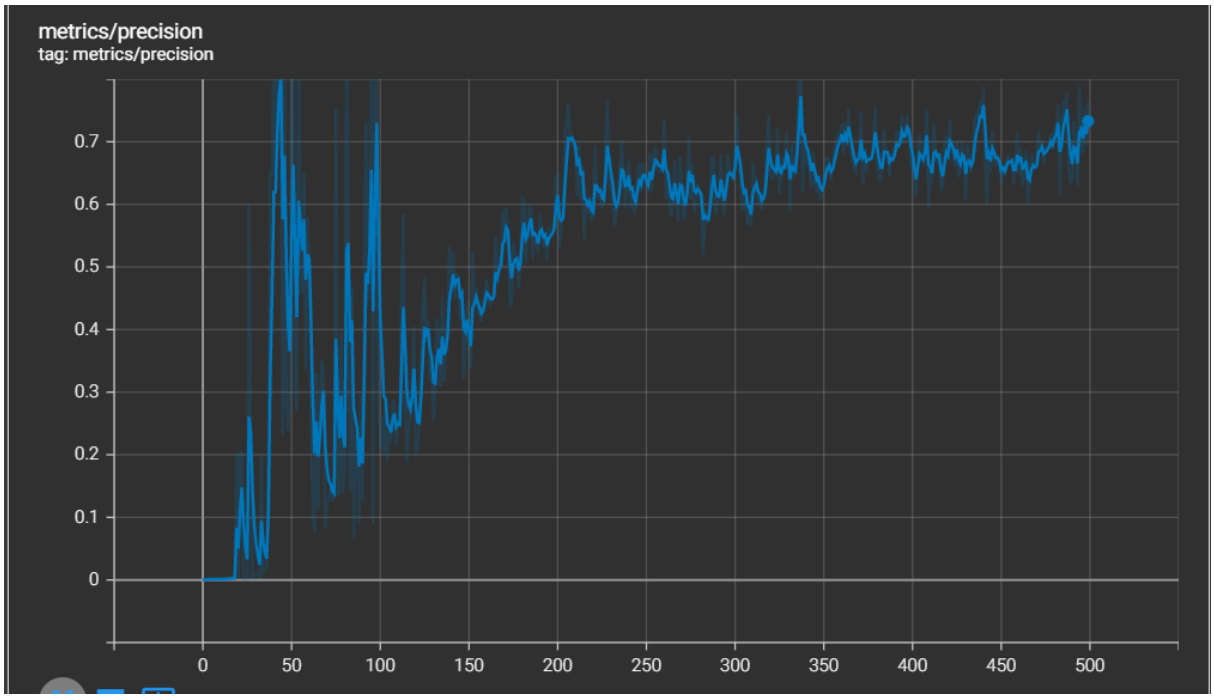

Figure 4.25: Depicts the precision metrics curve at 500 epochs

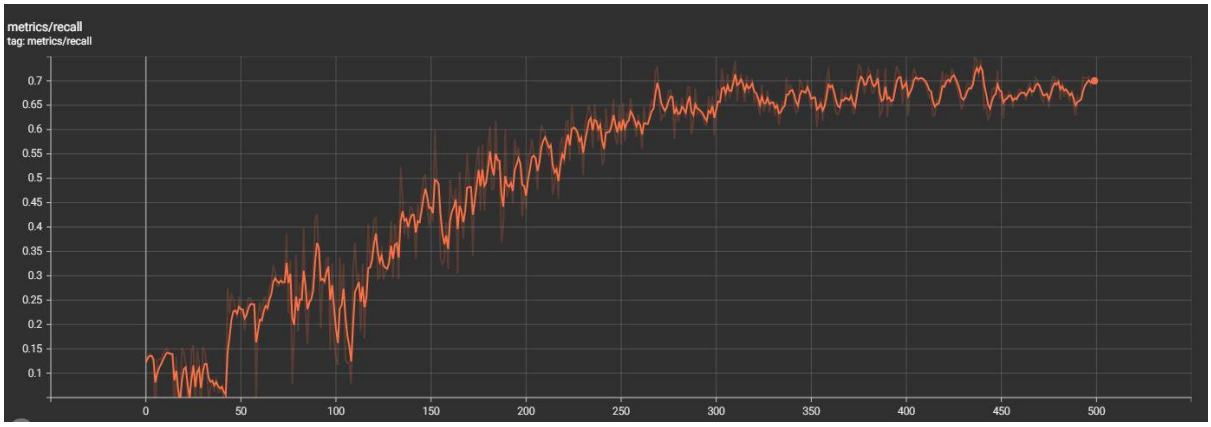

Figure 4.26: Depicts the recall metrics curve at 500 epochs

4.13 Confusion Matrix for 600 Epochs

The Confusion Matrix for multi - class classification is displayed in Figure 4.27 (in this case, 5 classes). The number of TP combinations for each class is displayed on the diagonal, from top left to bottom right: 100% of all items in the banana tree class, 72% of all objects in the pepper class, 94% of all objects in the spinach class, 81% of all objects in the sugarcane class, and 56% of all objects in the weed class were properly categorized.

Also 28 % of all objects of the pepper class were found by YOLO as unknown. 6 % of all objects of the spinach class were found by YOLO as unknown, 19 % of all objects of sugarcane class were classified as unknown and 44 % of all objects of weed class were found by YOLO as unknown and could not be categorized into any class by the classifier.

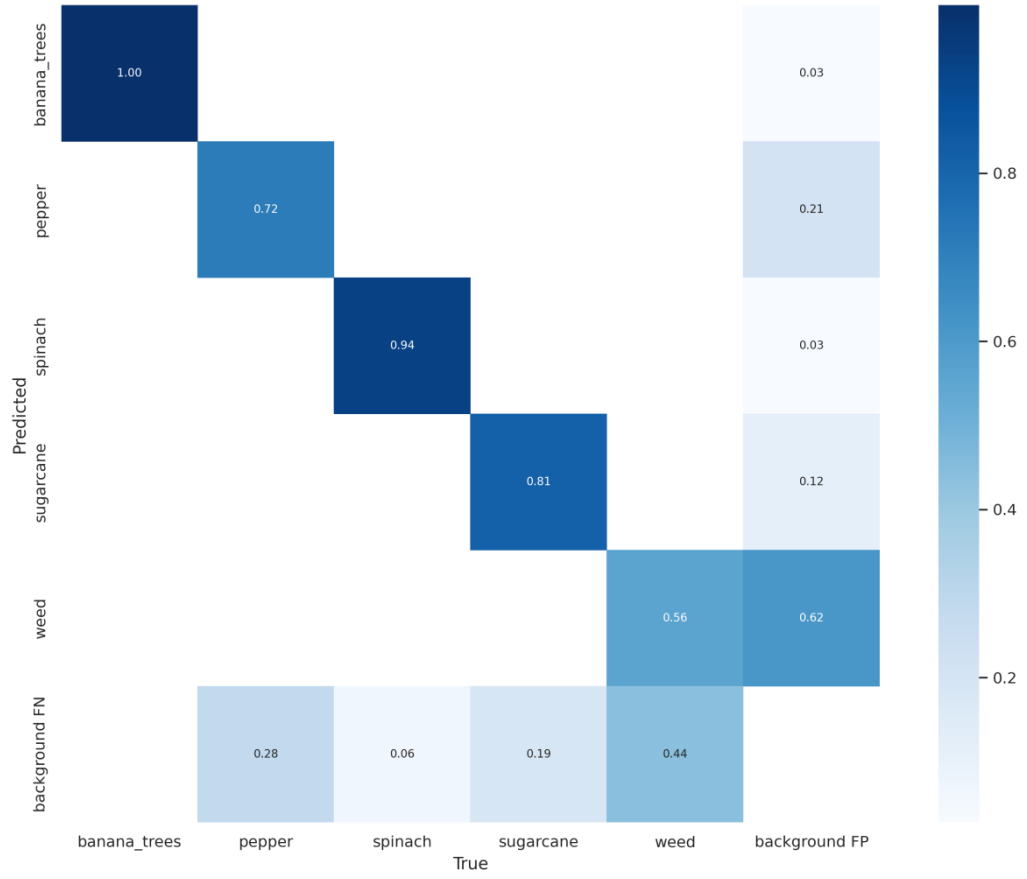

Figure 4.27: Confusion matrix for 600 epochs

4.13.1 Precision and recall values and graphs for 600 epochs

From the Precision and the Recall values presented in Table 4.15, it can be observed that the most accurate class in terms of Precision was “spinach” (at an approximate of 0.932) Preceded with “banana crops”, “sugarcane crops”, “weed plants” which improved to ‘0.782’ and then “pepper” as the least. Also, the most accurate class in terms of Recall was “banana crops” (at 1.000) preceded with “spinach crops”, “sugarcane crops”, “pepper crops” and “weed plants” (at an approximate of 0.338) that also improved from ‘0.269’

at 500 epochs indicating that the classifier could slowly but steadily recognize positive samples of "weed".

Table 4.15: Precision and recall for 600 epochs

Category	Precision	Recall
Sugarcane	0.814	0.674
Spinach	0.932	0.906
Pepper	0.700	0.544
Banana	0.891	1.000
Weed	0.782	0.338

Figure 4.28, depicts the models precision curve metric that determines the proportion of accurate bounding box predictions. While Figure 4.29 depicts the recall curve metric that determines the percentage of the actual bounding box that was successfully predicted. In Figure 4.28, the model began improving swiftly from around 100 epochs all through to 600 epochs on the precision curve and in Figure 4.29, the recall curve also improved swiftly all the way to 600 epochs which means the model was learning. The precision and recall capture the model performance, so the higher they are the better the model becomes. The y-axis demonstrates the value of precision for Figure 4.28 and the x-axis displays the different ranges of epochs while the y-axis depicts the recall value for Figure 4.29 and x-axis demonstrates the different ranges of epochs.

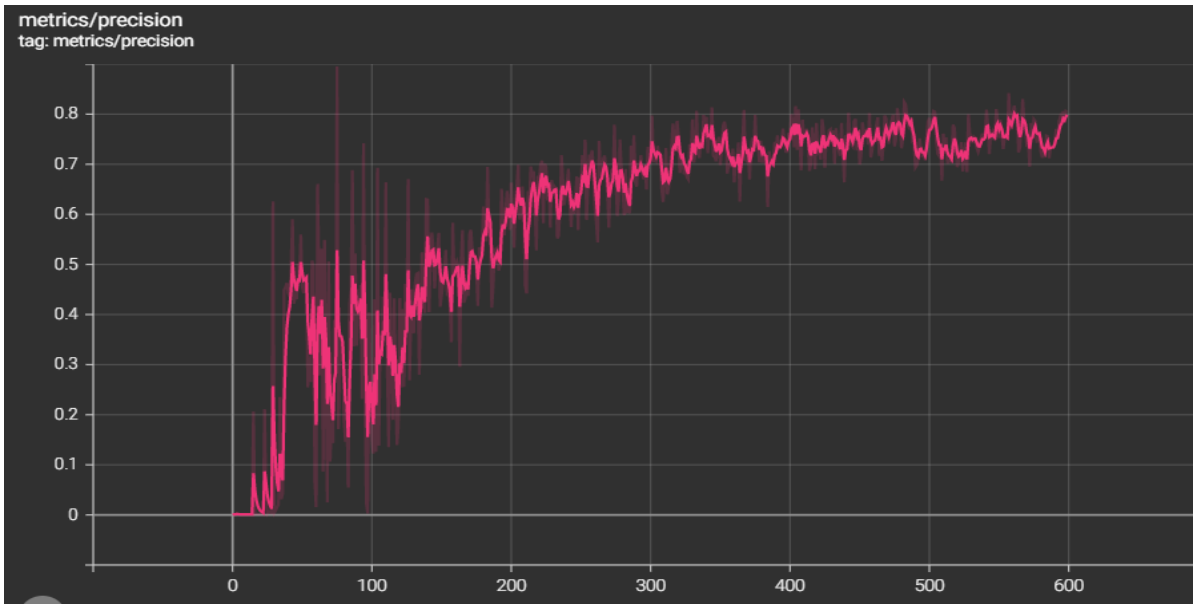

Figure 4.28: Depicts the precision metrics curve at 600 epochs

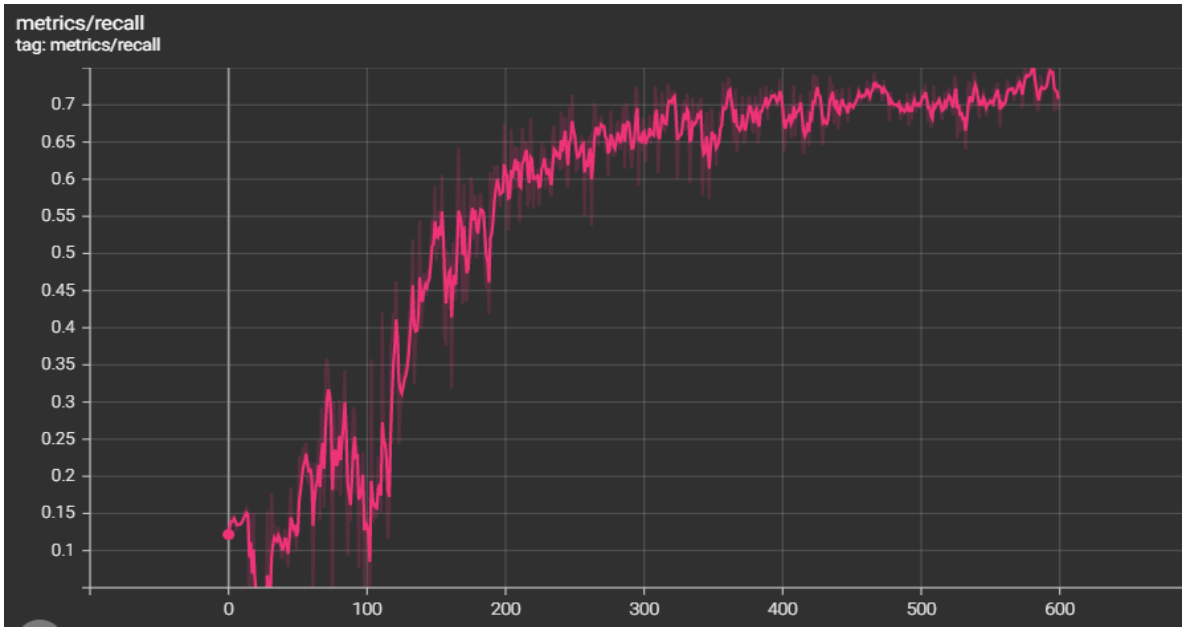

Figure 4.29: Depicts the recall metrics curve at 600 epochs

4.14 Confusion Matrix for 700 Epochs

The Confusion Matrix for multi - class classification is shown in Figure 4.30 (in this case, 5 classes). The number of TP elements for every class is displayed on the diagonal (top left to bottom right) as follows: 92% of all items in the "banana trees" class, 73% in the

"pepper" class, 94% in the "spinach" class, 81% in the "sugarcane" class, and 52% in the "weed" class were properly categorized. Also, 8 % of all objects of banana class were found by YOLO as unknown, 27 % of all objects of the pepper class were found by YOLO as unknown. 6 % of all objects of the spinach class were identified as unknown by YOLO, 19 % of all objects of sugarcane class were classified as unknown and 48 % of all objects of weed class were found by YOLO as unknown and could not be categorized by the classifier into any class.

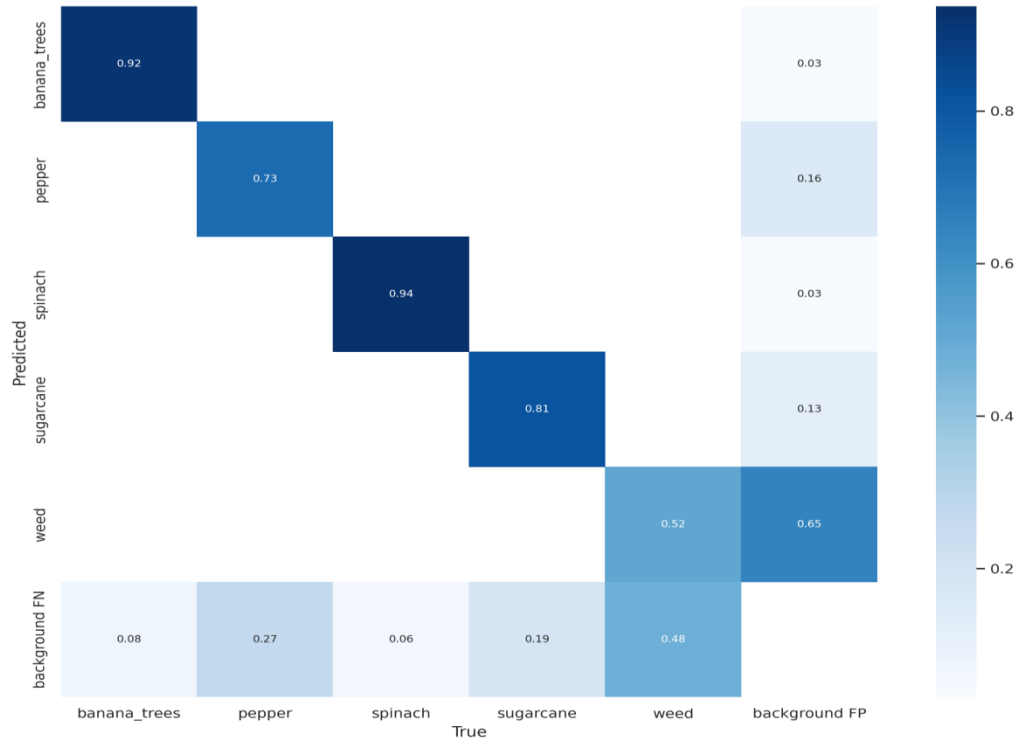

Figure 4.30: Confusion matrix for 700 epochs

4.14.1 Precision and recall values and graphs for 700 epochs

From the Precision and the Recall values displayed in Table 4.16, The classifier appeared to be flattening out as the precision values started to decline with an exception for ‘spinach’ that slightly increased with a ‘0.001’ margin. The value of Precision of “weed

plants" reduced from ''0.782' to '0.433' although the value of Recall increased slightly from '0.338' to '0.429' this indicates that the loss curve flattened at this epoch, indicating that the model training has achieved a saturation level. This suggests that adding more epochs than 700 won't result in any substantial improvements in the classification and detection of weeds.

Table 4.16: Precision and recall for 700 epochs

Category	Precision	Recall
Sugarcane	0.688	0.744
Spinach	0.933	0.938
Pepper	0.654	0.650
Banana	0.672	0.769
Weed	0.433	0.429

Figure 4.31, depicts the models precision curve metric that determines the proportion of accurate bounding box predictions. While Figure 4.32 depicts the recall curve metric that determines the percentage of the actual bounding box that was successfully predicted. In Figure 4.31, the model began improving swiftly from around 100 epochs all through to 700 epochs on the precision curve and in Figure 4.32, the recall curve also improved swiftly all the way to 700 epochs which means the model was learning. The precision and recall capture the model performance, so the higher they are the better the model becomes. The y-axis demonstrates the value of precision for Figure 4.31 and the x-axis displays the

different ranges of epochs while the y-axis depicts the recall value for Figure 4.32 and x-axis demonstrates the different ranges of epochs.

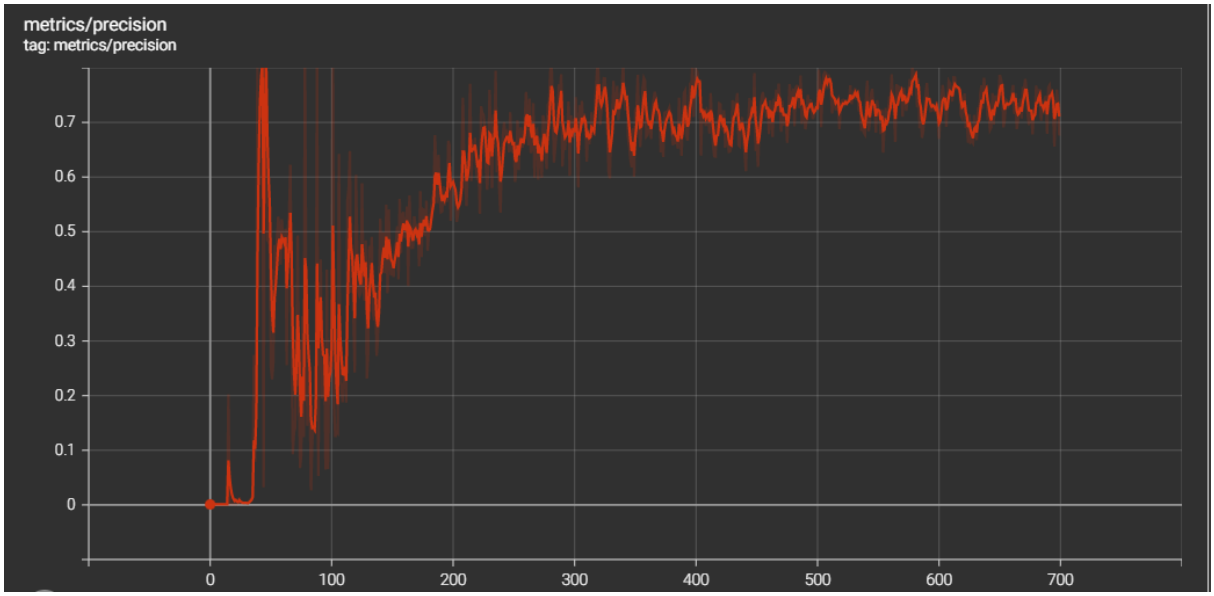

Figure 4.31: Depicts the precision metrics curve at 700 epochs

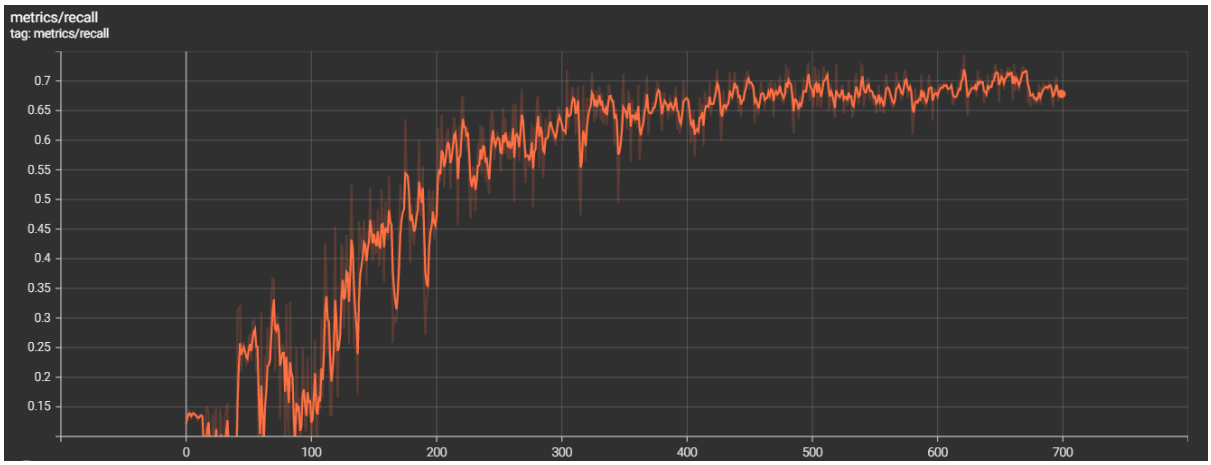

Figure 4.32: Depicts the recall metrics curve at 700 epochs

4.15 Confusion Matrix for 1000 Epochs

The Confusion Matrix for multi - class classification is shown in Figure 4.33 (in this case, 5 classes). The number of TP elements for every class is displayed on the diagonal (top left to bottom right) as follows: 97% of all items in the spinach class, 86% of all objects

in the sugarcane class, 73% of all objects in the pepper class, 85% of all objects in the banana tree class, and 47% of all objects in the weed class were properly categorized. Also, 15 % of all objects of banana class were found by YOLO as unknown, 27 % of all objects of the pepper class were found by YOLO as unknown. 3 % of all objects of the spinach class was classed as unidentified by YOLO, 14 % of all objects of sugarcane class were classified as unknown and 53 % of all objects of weed class were found by YOLO as unknown and could not be categorized into any class by the classifier.

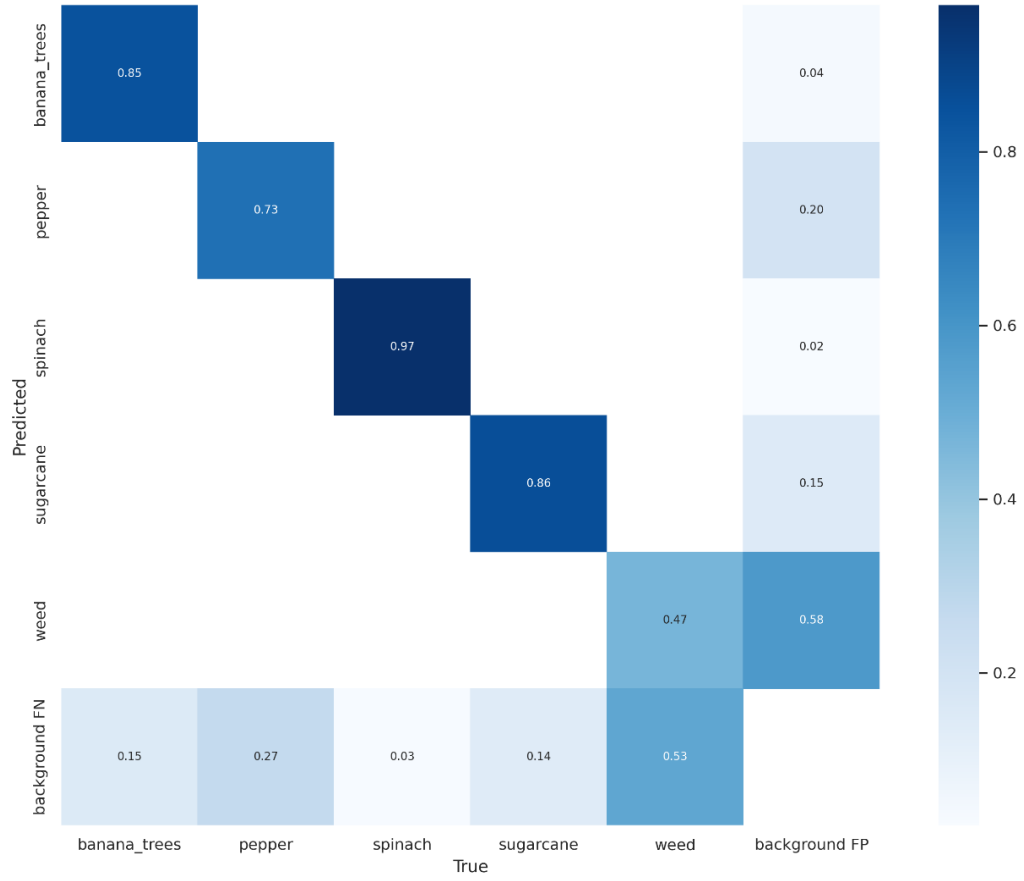

Figure 4.33: Confusion matrix for 1000 epochs

4.15.1 Precision and recall values and graphs for 1000 epochs

As observed in table 4.17, the "weed" Precision marginally improved from '0.433' to '0.449'. The value of Recall dropped from 0.429 to 0.403 of which is a direct implication that the classifier had already achieved a saturated level. At this point, there was no need to proceed further for the iteration.

Table 4.17: Precision and recall for 1000 epochs

Category	Precision	Recall
Sugarcane	0.697	0.767
Spinach	0.911	0.957
Pepper	0.710	0.654
Banana	0.791	0.846
Weed	0.449	0.403

Figure 4.34, depicts the models precision curve metric that determines the proportion of accurate bounding box predictions. While Figure 4.35 depicts the Recall curve metric that determines the percentage of the actual bounding box that was successfully predicted. In Figure 4.34, the model began improving swiftly from around 100 epochs all through to 1000 epochs on the precision curve and in Figure 4.35, the Recall curve also improved swiftly all the way to 1000 epochs which means the model was learning. The Precision and Recall capture the model performance, so the higher they are the better the model becomes. The y-axis displays the precision value for Figure 4.34 and the recall value for

Figure 4.35, whereas the x-axis displays the different ranges of epochs for both Figure 4.34 and Figure 4.35.

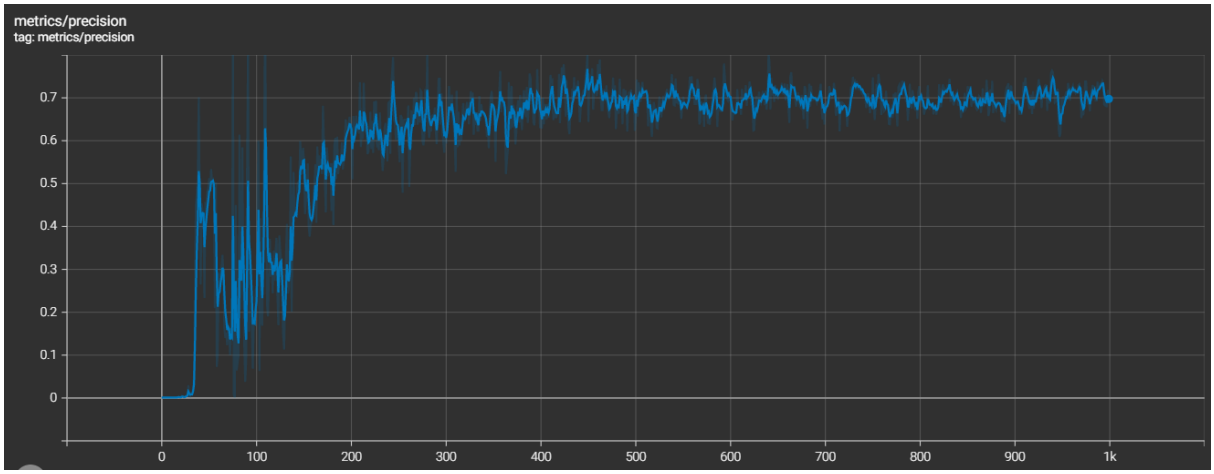

Figure 4.34: Depicts the precision metrics curve at 1000 epochs

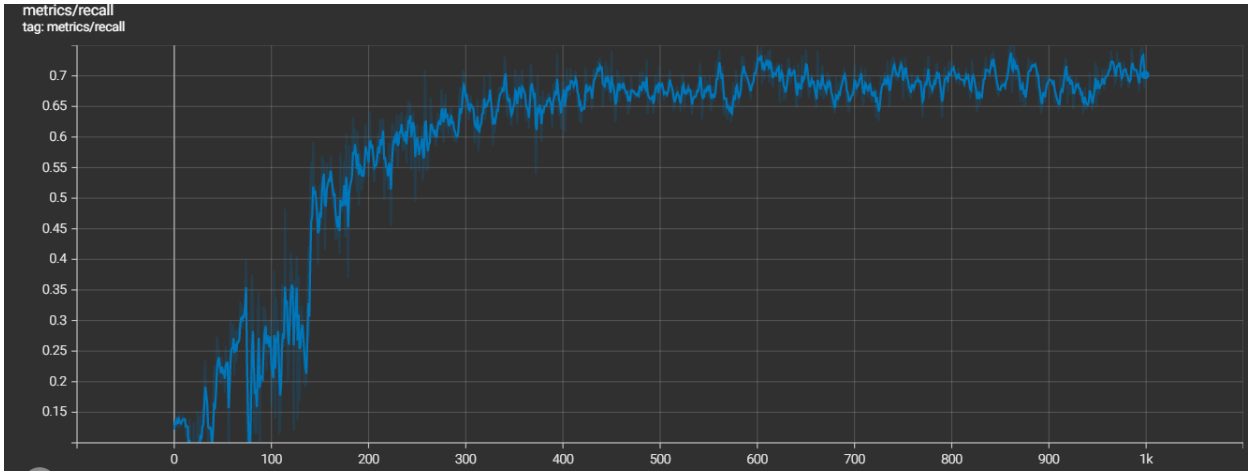

Figure 4.35: Depicts the recall metrics curve at 1000 epochs

4.16 Cumulative Accuracy Metric Values

The overall accuracy obtained for 100 epochs was 16 %, 300 epochs was 65 %, 500 epochs was 66 %, 600 epochs was 67 %, 700 epochs was 65 % and 1000 epochs was 64 %. Thus there was steady improvement in the accuracies which was observed while increasing the number of epochs from 100 to 600 as the batches size for the YOLO v5 remains the same.

At 100 epochs, the observed average Precision was 33%, average of Recall of 19% and also, the F1 score of 24%. As soon as the epoch reached 300 epochs, the observed average Precision was 63%, average of Recall was 71% and an F1 score of 67%. Also, 500 epochs, the observed average Precision was 76%, average of Recall achieved 64% and a F1 score of 69%. Subsequently, when increased to 600 epochs, the observed average Precision was 82%, average of Recall achieved 69% and a F1 score of 75%. At 700 epochs, the observed average Precision was 67%, average of Recall achieved 70% and a F1 score of 69%. Finally, at 1000 epochs, the observed average Precision was 71%, average of Recall achieved 72% and a F1 score of 71% as shown in Table 4.18. Also, all epochs were processed on Colab Free having a GPU of K80, RAM of 16GB and runtime of 12hours. The expended time for 100 epochs was 4minute 62 seconds, 300 epochs was 11minutes 88seconds, 500 epochs was 18minutes 48seconds, 600 epochs was 22minutes 92seconds, 700 epochs was 25minutes 86seconds, and 1000 epochs was 38minutes 22seconds.

Table 4.18 compares and contrasts each one of the epochs' Accuracies, average Precisions, average Recalls, and then the F1 scores. When contrasted to other epochs, YOLO v5 model at 600 epochs and batch size of 32, exhibited the best Precision, Accuracy, Recall and F1 score outcomes, indicating it to be the best training epoch for accurate weed and crop recognition.

Table 4.18: Showing the cumulative accuracy metrics of YOLO v5

Epochs	Accuracy	Average Precision	Average Recall	F1 score
100	0.160	0.334	0.190	0.242
300	0.646	0.629	0.708	0.666
500	0.655	0.764	0.644	0. 699
600	0.671	0.823	0.692	0. 752
700	0.653	0.676	0.706	0.691
1000	0.648	0.712	0.725	0.718

The accuracy of the categorization estimates for weed performance for the automatic weed categorization was evaluated and the result from 100 epochs yielded a 16 % in classification accuracy, Precision of weed was 5 % and a Recall of weed was 13 %. For 300 epochs, the classification accuracy of 65 % was gotten; Precision of weed was observed to be 46 % and the Recall for weed was achieved at 32 %. For 500 epochs, a classification accuracy having 66 % was achieved; with a Precision of weed at 75 % and an observed Recall of weed at 27 %. At 600 epochs, it also yielded a 67 % in classification

accuracy; a Precision of weed at 78 % and a Recall of weed at 34 % was achieved. At 700 epochs, 65 % of classification accuracy was achieved; including a Precision weed of 43 % and a Recall of weed at 43 %. Finally, at 1000 epochs, the classification accuracy of 65 % was achieved; with a Precision of weed at 45 % and also a Recall of weed at 40 % was achieved over the evaluation metrics.

In addition, it was discovered that the autonomous weed classification classifier's maximum weed precision (78%) was attained at 600 epochs, whereas the weed accuracy began to decline at 700 epochs, when it fell to (43%). Also, there was no significant improvement above 1000 epochs in the accuracy of weeds after increasing the epochs from 600 through to 1000 epochs.

4.17 Output of the Model on the Testing Dataset

The weed pattern visualization outcomes displayed in Plate X down to Plate XV which was done to observe the weeds of various sizes that have been identified within the mixed cropping farm containing sugarcane, pepper, banana and spinach crops. In Plate X at 100 epochs, the yolov5s model was able to identify and classify weeds within the irrigated farm at approximately 63 % in precision. This was due to the fact that epoch used was too small for the model to completely learn. Plate XI at 300 epochs, the algorithm was able to classify weeds to a precision of 63 %. In Plate XII at 500 epochs, the model classified weeds to a precision of 74 %. At 600 epochs in Plate XIII, weeds were classified to a precision of 78 % within the farm. Furthermore, weed class was identified and classified to a precision of approximately 63 % in Plate XIV at 700 epochs and finally at 1000 epochs in Plate XV, weeds were classified with an accuracy of 51 %. From the different epochs employed during training of the model, it was observed that at 600 epochs, the precision of weed reached it maximum state of 78 % and began to decline

with increase in the numbers of epochs from 600 epochs to 700 epochs at 63 % and finally to 51 % at 1000 epochs. This implies that the model can indeed predict weeds more accurately at 600 epochs, which is crucial for agricultural weed identification and classification purposes.

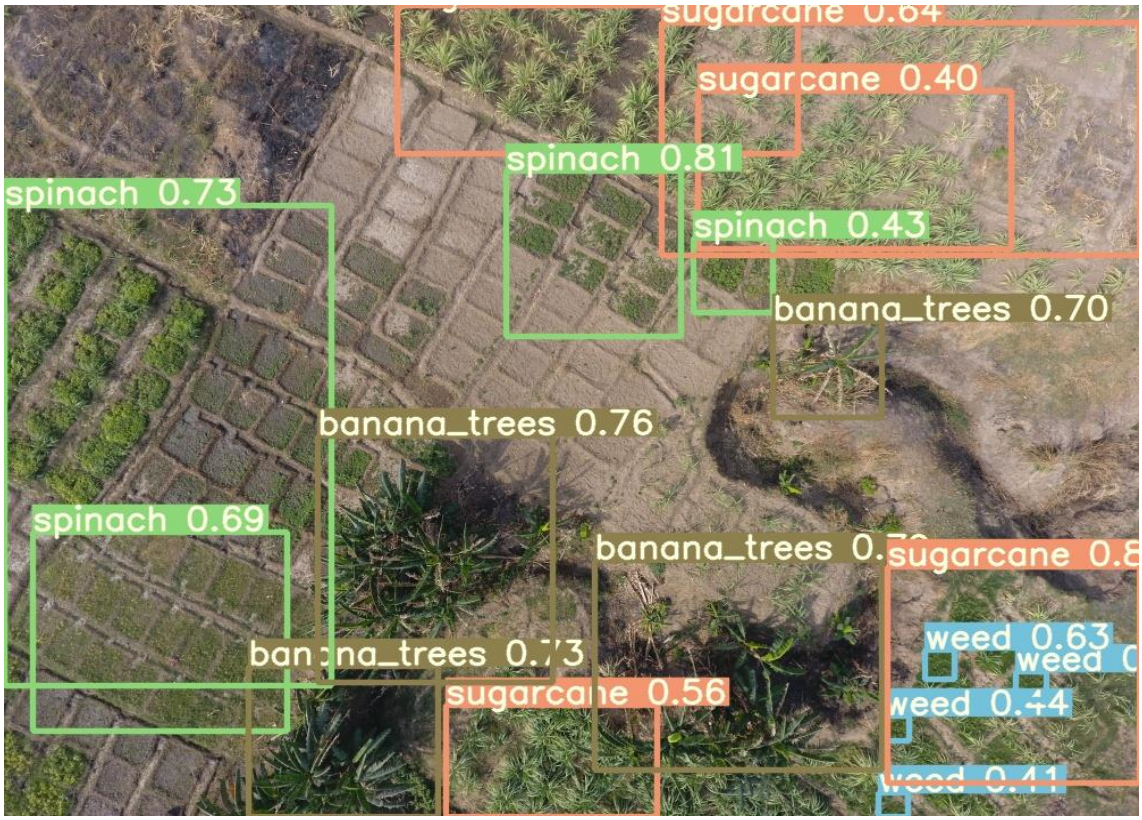

Plate X: Weed classification results on test images at 100 epochs

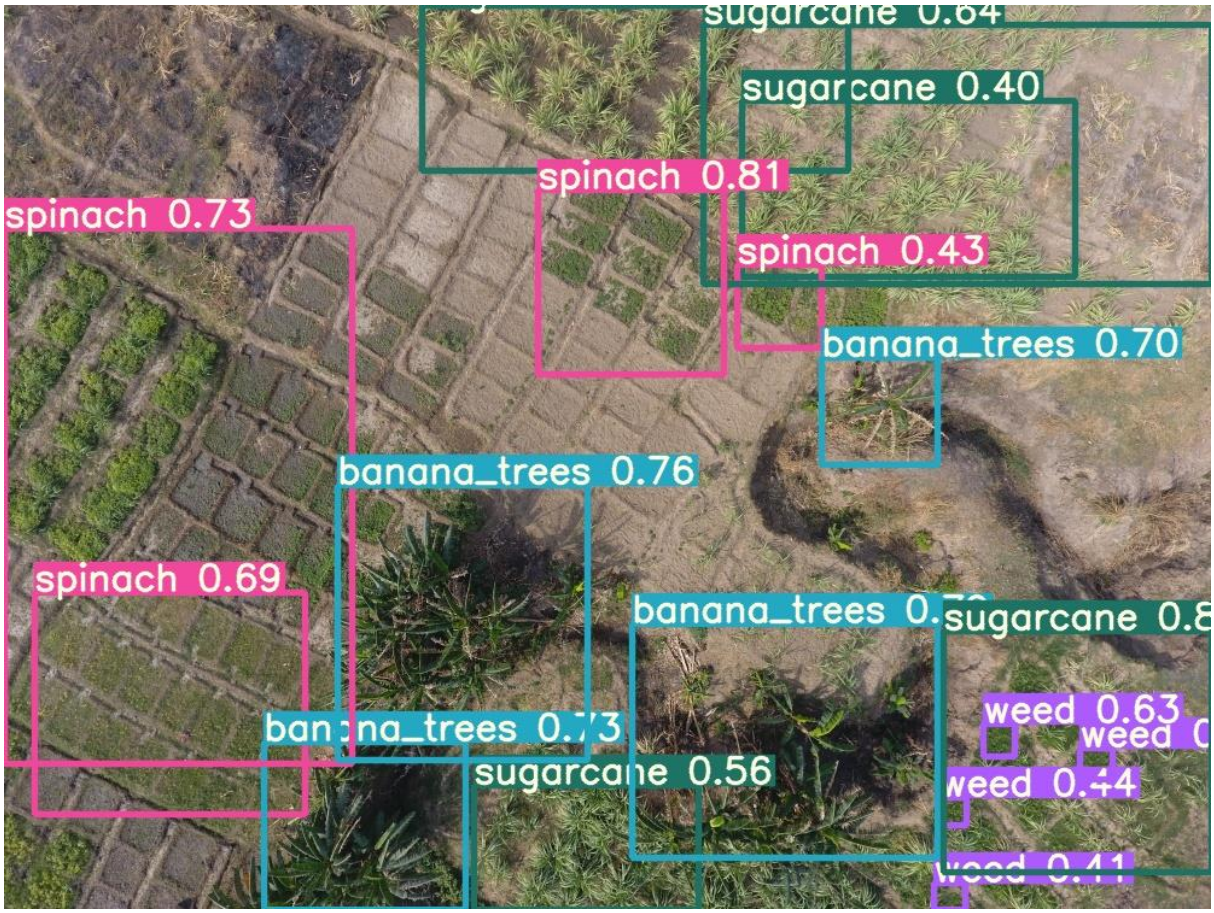

Plate XI: Weed classification results on test images at 300 epochs

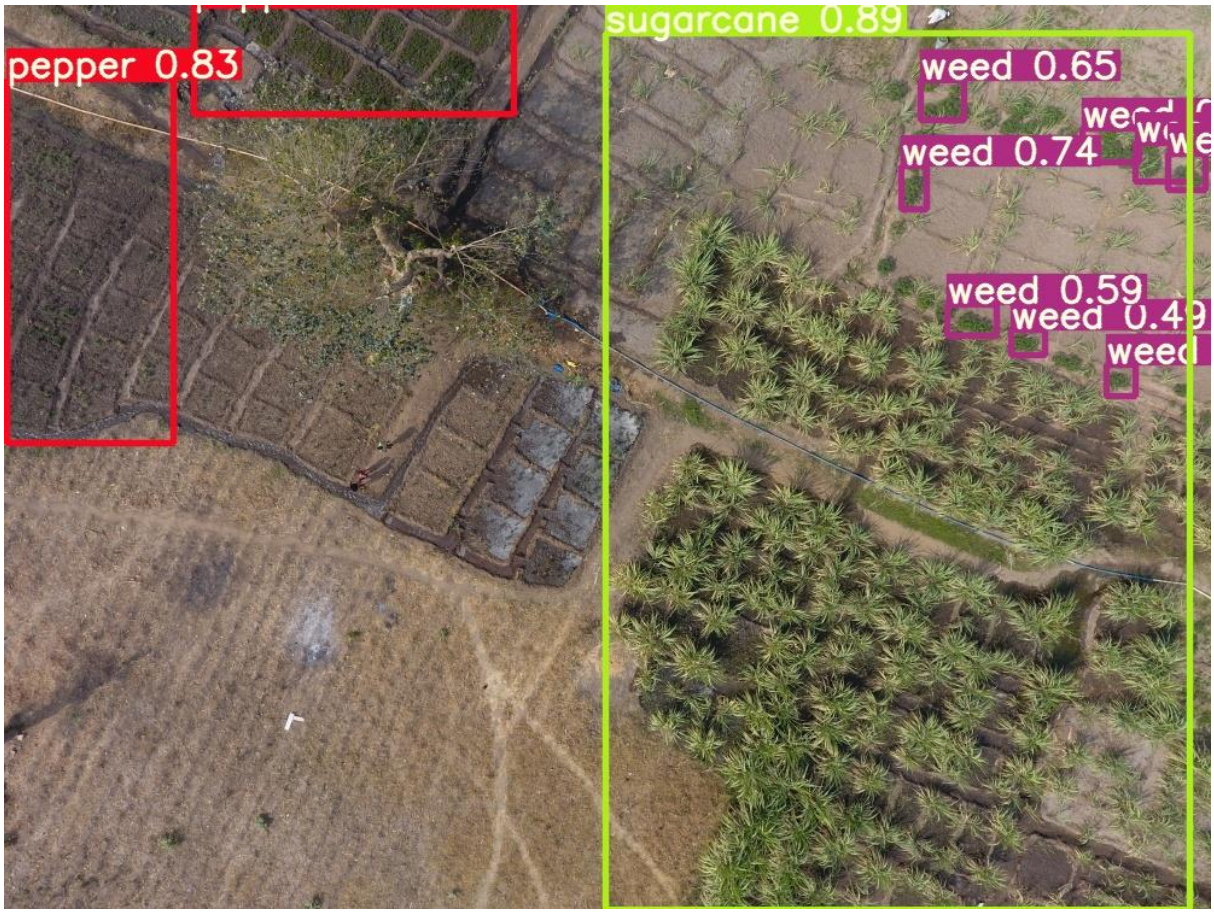

Plate XII: Weed classification results on test images at 500 epochs

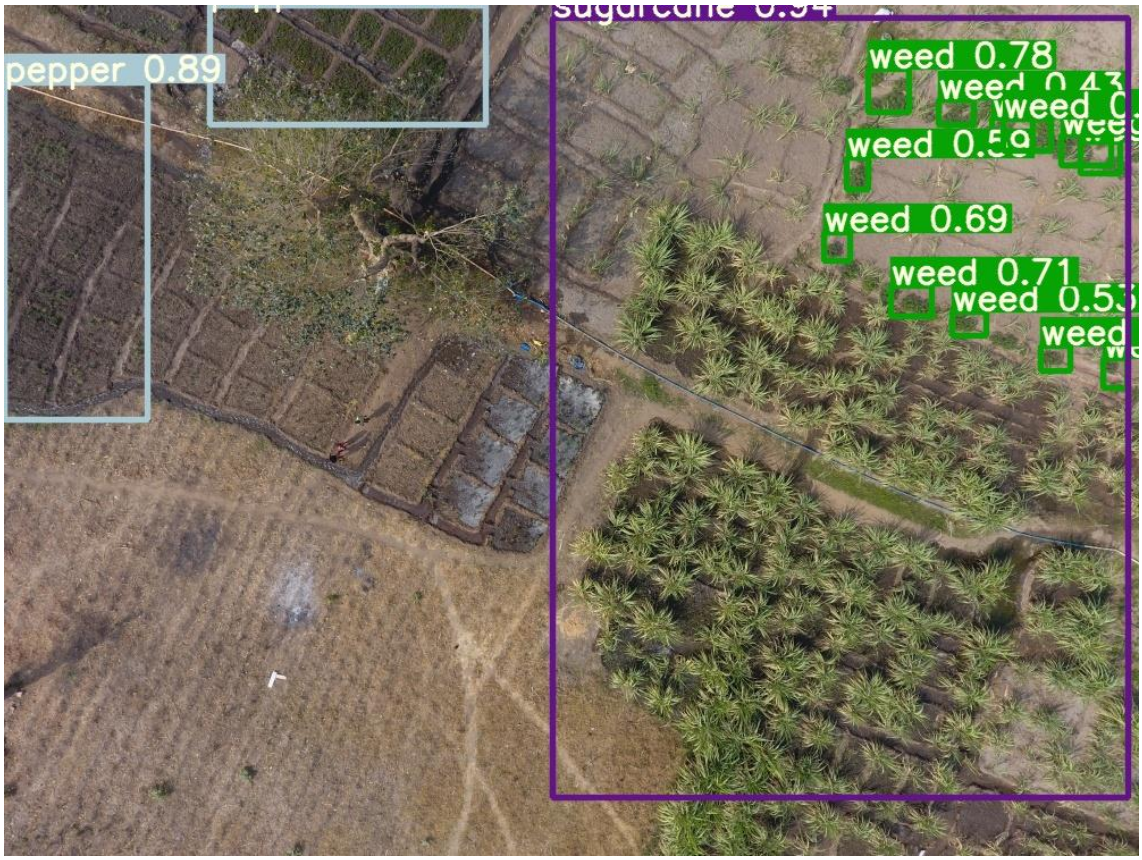

Plate XIII: Weed classification results on test images at 600 epochs

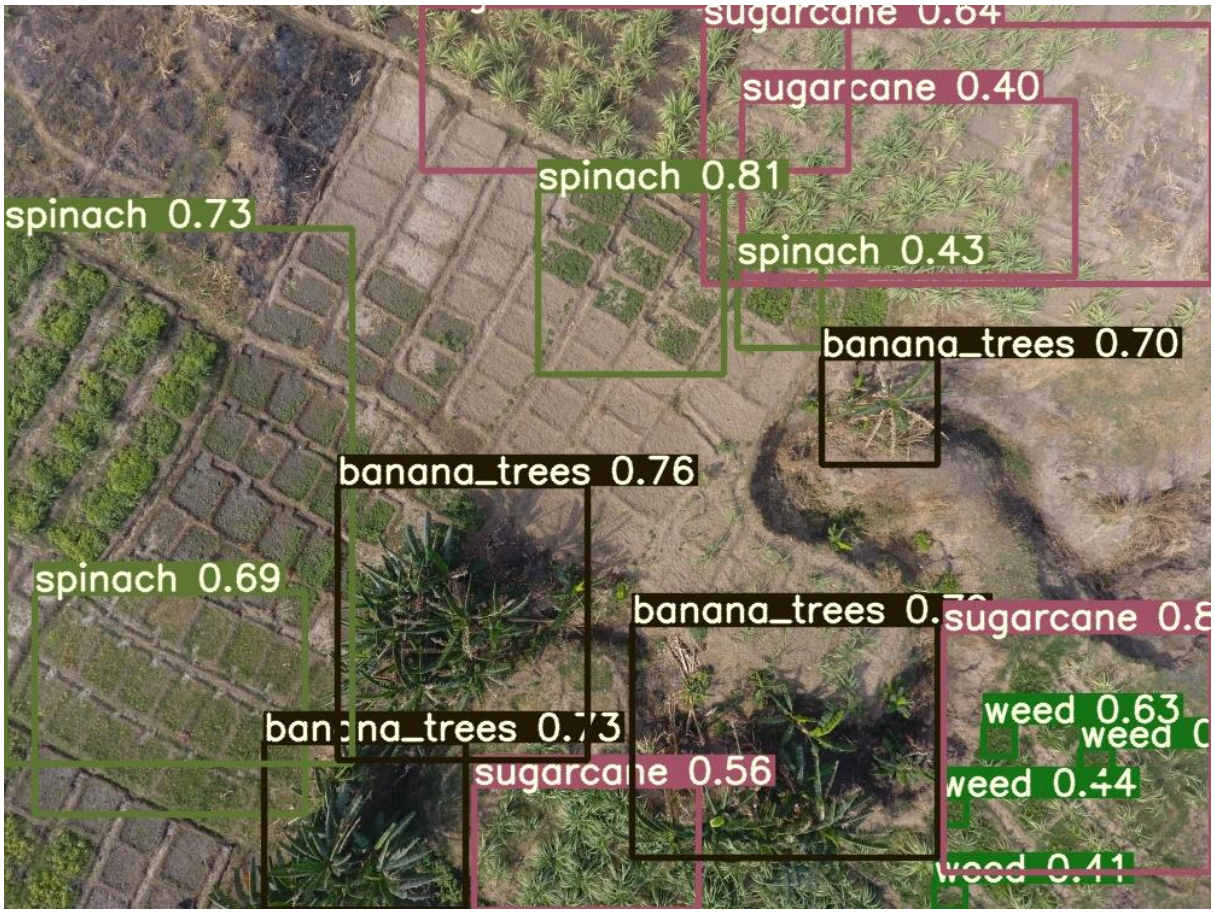

Plate XIV: Weed classification results on test images at 700 epochs

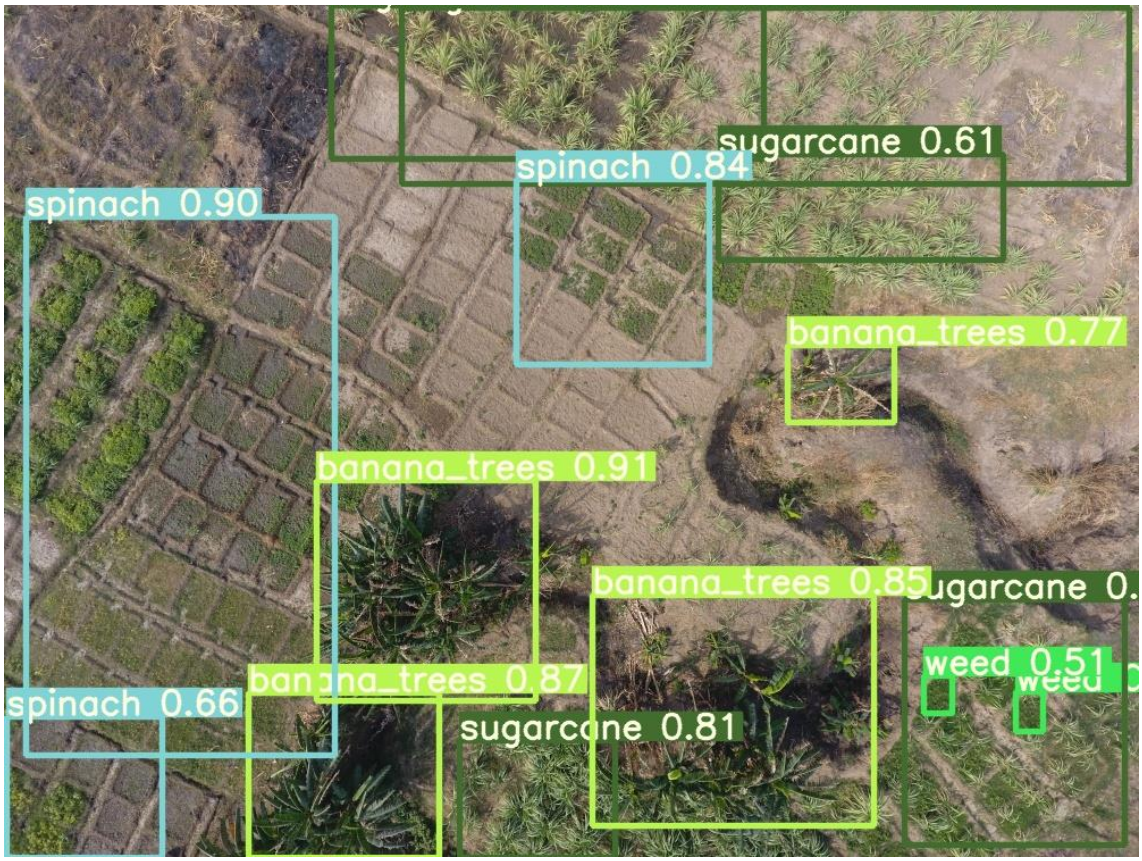

Plate XV: Weed classification results on test images at 1000 epochs

4.18 Comparing the Performance of Faster RCNN and YOLO v5 Based on Results Obtained from the Overall Classification Accuracies, Weed Precision and Weed Recall

The classifiers utilized have varying degrees of overall classification accuracy (Faster RCNN and YOLOv5s). As shown in Table 4.19, Faster RCNN exhibited the highest overall accuracies. Notably lower accuracies were observed using YOLOv5s. Using the Faster RCNN classifier, the lowest accuracies were realised at 10,000 epochs with an overall accuracy of 52%, weed precision of 50% and weed recall of 8% while the at 200,000 epoch, the highest level of accuracies and saturated point were achieved with 98% 0verall accuracy, 98% weed precision and 99% weed recall. While the minimum epoch of YOLOv5s classification at 100 epochs achieved the overall accuracy of 16%, weed precision of 5% and 1% for the weed recall. Furthermore, the classifier achieved a maximum weed precision at 600 epochs with a weed precision of 78%, weed recall of 34% and an overall accuracy of 67%. With only 5 % for the lowest weed precision and of 78% for the highest weed precision of YOLOv5s, this exhibited by far a lesser accuracy. The Faster RCNN Deep Learning exhibited a better classification output making it the best classifier suitable for automatic weed identification and classification.

Table 4.19: Accuracy comparison of the minimal and maximal achievable accuracy epochs of both classifiers

Classifier	Epochs	Overall accuracy	Weed precision	Weed recall
Faster RCNN	10,000 (MIN)	0.526	0.500	0.078
	200,000 (MAX)	0.984	0.981	0.992
YOLOv5s	100 (MIN)	0.160	0.0504	0. 013
	600 (MAX)	0.671	0. 782	0. 338

4.18.1 Processing time

Time complexity varies significantly depending on the classifier and the number of observations. The model that required the most time to train, with an average of 27.8 minutes (min) and 7.9 hours (max) per classification, was the Faster Region based Convolutional Neural Network algorithm this was due to the number of epochs, the model architecture, layers and parameters used which made the Faster RCNN take longer time. With YOLOv5, having a less complex architecture and lesser hyperparameters, the calculation times was shorter at 4minute 62 seconds and 18minutes 48seconds. Table 4.20 depicts the minimum and maximum processing time for training both Deep Learning models.

Table 4.20: The minimum and maximum processing time for training the selected deep learning algorithms

Classifiers	Epochs	Training time per classification
Faster RCNN	10,000 (MIN)	27minutes 8seconds
	200,000 (MAX)	7hours 9minutes
YOLOv5s	100 (MIN)	5minute 2seconds
	500 (MAX)	23minutes 38seconds

In this research, YOLOv5 was discovered as being the least satisfactory classifier amongst the other deep-learning approach. With minimal computation times and no parameters that need to be tuned, Deep Learning delivers the most convenient usage. Notwithstanding, the poor categorization performance outweighs these benefits especially when classifying weeds that are small due to only two anchor boxes in a grid predicting only one class of object. Faster RCNN in comparison it is simple to apply, as just one variable is required to be set by the user and it can detect smaller weeds well since it has nine anchors in a single grid. After weighing every factor, including classification accuracy, robustness, calculation complexity, and intuitiveness, Faster RCNN was shown to be the better option for leveraging data supplied by UAVs to classify weeds and other crop kinds. Comparing this strategy to the YOLOv5s model, it performed better and was more robust in categorization.

CHAPTER FIVE

5.0 Conclusion and Recommendations

5.1 Summary of Findings

This research explored the performance of two models in weed classification which are the Faster RCNN inception v2 model and YOLOv5s architecture making use of Unmanned Aerial Vehicle imagery to automatically distinguish and classify weed plants from crops within an irrigated farm in Minna the state capital of Niger State.

After the implementation of the classification models (Faster RCNN and YOLO v5) for the identification and classification of weeds, the following findings were generated:

(i) It was discovered that the accuracy of the models increased with increase in training epochs until the models were saturated.

(ii) YOLO v5 was discovered to be the fastest in terms of runtime as compared to the Faster RCNN model which took greater time to complete its classification.

(iii) It was also discovered that the Faster RCNN out-performed the YOLO v5 classifier in aspects of performance accuracy in the development of the UAV based automatic crop type classification and weed detection scheme.

5.2 Conclusion

Better accuracy in numerous real-time applications has been made possible by the enormous advances in Deep Learning techniques. This study has demonstrated the usability of Deep Learning strategies, specifically, Faster RCNN and YOLO v5 algorithms, for weed identification and classification. The effectiveness of the Faster Region based Convolutional Neural Network applied and the YOLO v5 were assessed

employing metrics which include accuracy, the precision, the recall, and a F1 score and demonstrated to be exceptionally competent of autonomously recognizing and classifying weed plants in a mixed farmland from UAV data with the use of the documented loss function and confusion matrix.

In summary, YOLOv5 showed advantages in fast computation times while achieving comparable detection accuracies in the identification and categorisation of weeds from a mixed farm as compare to Faster RCNN. Also, after assessing every metric, notably classification accuracy, the Precision, the Recall and a F1 score, Faster RCNN architecture appeared to be the most effective and accurate classification approach of weeds from various crop kinds employing Unmanned Aerial Vehicle imagery. Subsequently, it was also observed that the increase in epochs influences the accuracy of the classification model. With this, the aim of the research of implementing and evaluating the performance of the Deep Learning algorithms investigated was achieved.

5.3 Recommendations

From the findings of this research, it is recommended that spectral and spatial resolutions to optimise the flight mission to capture the size of the smaller weeds to be discriminated for better performance accuracy and also, it is recommended in Faster RCNN not to go above a maximum training of 200,000 epochs and below a minimum training of 10,000 epochs for accurate performance and for the YOLO v5, it is advised not to exceed a maximum training epoch of 600 and a minimum training epoch of 100 for a good performance output. Hyper-parameter tunning and data augmentation (artificially increasing the training set by creating modified copies of a dataset using existing data) could be done to observe how they affect the models accuracy.

5.4 Contributions to Knowledge

i. Performance accuracy increases with increase in training epochs for the selected Deep Learning models.

ii. The Faster Region-based Convolutional Neural Network out performs the YOLO v5 algorithm in terms of Accuracy, average Precision, average Recall and F1 score.

iii. This research have made the application of the right quantity of farm inputs (water, manure/fertilizers and herbicides) more precise and also mitigated excessive chemical use.

5.5 Future Work

Further research should be carried out to further compare the effectiveness of Faster Region based Convolutional Neural Network model with a few other powerful Deep Learning methods to discover faster and more accurate models for weed detection on small farmlands while taking images at a distance less than 30m and closer for smaller weeds so they appear larger in the image.

REFERENCES

Adamchuk, V. I., Bernards, M. L., & Meyer, G. E. (2008). EC08-708 Precision Agriculture: Weed Targeting Herbicide Management. *Historical Materials from University of Nebraska-Lincoln Extension*, 4871.

Adekunle, I. O. (2013). Precision agriculture: Applicability and opportunities for Nigerian agriculture. *Middle-East Journal of Scientific Research*, 13(9), 1230-1237, doi: 10.5829/idosi.mejsr.2013.13.9.1004

Akbar, S., Peikari, M., Salama, S., Nofech-Mozes, S., & Martel, A. (2017). Transitioning between convolutional and fully connected layers in neural networks. In *Deep Learning in Medical Image Analysis and Multimodal Learning for Clinical Decision Support* (pp. 143-150), doi: 10.1007/978-3-319-67558-9_17

Akobundu, I. O. (1987). *Weed science in the tropics. Principles and practices.* (p. 522). John Wiley. https://doi.org/19880711194

Alamsyah, A., Saputra, M. A. A., & Masrury, R. A. (2019). Object detection using convolutional neural network to identify popular fashion product. In *Journal of Physics: Conference Series*, 1192(1), 012040, doi: 10.1088/1742-6596/1192/1/012040

Albawi, S., Mohammed, T. A., & Al-Zawi, S. (2017). Understanding of a convolutional neural network. In *2017 international conference on engineering and technology (ICET)* (pp. 1-6), doi**:** 10.1109/ICEngTechnol.2017.8308186

Alexandratos, N. & Bruinsma, J. (2012). World agriculture towards 2030/2050: the 2012 revision. *ESA Working paper*, No. 12-03. Rome, FAO, doi 10.22004/ag.econ.288998

Alom, M. Z., Taha, T. M., Yakopcic, C., Westberg, S., Sidike, P., Nasrin, M. S., & Asari, V. K. (2019). A state-of-the-art survey on Deep Learning theory and architectures. *Electronics*, 8(3), 292, https://doi.org/10.3390/electronics8030292

Alotaibi, M., & Mahmood, A. (2017). Improved gait recognition based on specialized deep convolutional neural network. *Computer Vision and Image Understanding*, 164, 103-110, https://doi.org/10.1016/j.cviu.2017.10.004

Al-Saffar, A.A.M., Tao, H., & Talab, M.A. (2017). Review of deep convolution neural network in image classification, in: Radar, Antenna, Microwave, Electronics, and Telecommunications (ICRAMET), *2017 International Conference on*, *IEEE*, (pp. 26–31), doi**:** 10.1109/ICRAMET.2017.8253139

Annett, R., Habibi, H.R., & Hontela, A., (2014). Impact of glyphosate and glyphosate-based herbicides on the freshwater environment. *Journal of Applied Toxicol*ogy, 34, 458-479, https://doi.org/10.1002/jat.2997

Apolo-Apolo, O. E., Martínez-Guanter, J., Egea, G., Raja, P., & Pérez-Ruiz, M. (2020). Deep Learning techniques for estimation of the yield and size of citrus fruits using a UAV. *European Journal of Agronomy*, 115, 126030, https://doi.org/10.1016/j.eja.2020.126030

Azizah, L. M. R., Umayah, S. F., Riyadi, S., Damarjati, C., & Utama, N. A. (2017). Deep Learning implementation using convolutional neural network in mangosteen surface defect detection. *In 2017 7th IEEE international conference on control system, computing and engineering (ICCSCE)* (pp. 242-246), doi**:** 10.1109/ICCSCE.2017.8284412

Bah, M. D., Dericquebourg, E., Hafiane, A., & Canals, R. (2018a). Deep Learning based classification system for identifying weeds using high-resolution UAV imagery. In *Science and Information Conference* (pp. 176-187), doi: 10.1007/978-3-030-01177-2_13

Bah, M.D., Hafiane, A., Canals, R. (2018b). Deep Learning with Unsupervised Data Labeling forWeed Detection in Line Crops in UAV Images. *Remote Sensing*, 10, 1690, https://doi.org/10.3390/rs10111690

Bajwa, A. A. (2014). Sustainable weed management in conservation agriculture. *Crop protection*, 65, 105-113, https://doi.org/10.1016/j.cropro.2014.07.014

Batte, M., & Van-Buren, R. (1999). *Precision farming*: A factor influencing productivity. Paper presented at the Northern Ohio Crops Day Meeting. Woody County OH, Ohio, USA.

Beeharry, Y., & Bassoo, V. (2020). Performance of ANN and AlexNet for weed detection using UAV-based images. In *2020 3rd International Conference on Emerging Trends in Electrical, Electronic and Communications Engineering (ELECOM)* (pp. 163-167), doi: 10.1109/ELECOM49001.2020.9296994

Bengio, Y. (2009). Learning deep architectures for AI. *Foundations and trends in Machine Learning*, 2(1), 1-127, http://dx.doi.org/10.1561/2200000006

Bergin, D., (2011). Weed Control Options for Coastal Sand Dunes: a Review. *New Zealand Forest Research Institute LTD*, (pp. 5-13).

Bhatt, D., Patel, C., Talsania, H., Patel, J., Vaghela, R., Pandya, S., & Ghayvat, H. (2021). CNN Variants for Computer Vision: History, Architecture, Application, Challenges and Future Scope. *Electronics*, 10(20), 2470, https://doi.org/10.3390/electronics10202470

Blaschke, T. (2010). Object based image analysis for remote sensing. *ISPRS journal of photogrammetry and remote sensing*, 65(1), 2-16, https://doi.org/10.1016/j.isprsjprs.2009.06.004

Bond, W., & Grundy, A. C. (2001). Non-chemical weed management in organic farming systems. *Weed research*, 41(5), 383-405, https://doi.org/10.1046/j.1365-3180.2001.00246.x

Bonotis, P.A., Tsouros, D.C., Smyrlis, P.N., Tzallas, A.T., Giannakeas, N., Evripidis, G., & Tsipouras, M.G. (2019). Automated Assesment of Pain Intensity based on EEG Signal Analysis. In *Proceedings of the IEEE 19th International Conference on BioInformatics and BioEngineering*, doi: 10.1109/BIBE.2019.00111

Bontonou, M., Lassance, C., Hacene, G. B., Gripon, V., Tang, J., & Ortega, A. (2019). Introducing graph smoothness loss for training deep learning architectures. In *2019 IEEE Data Science Workshop (DSW)* (pp. 160-164). IEEE.

Brejda, J.J., Moorman, T. B., Smith, J.L., Karlen, D.L., Allan, D.L. & Dao, T.H. (2000). Distribution and variability of surface soil properties at a regional scale. *Soil Science Society of American Journal,* 64(3), 974-982, https://doi.org/10.13140/RG.2.2.19360.79362

Browne, M., Ghidary, S. S., & Mayer, N. M. (2008). Convolutional neural networks for image processing with applications in mobile robotics. In *Speech, Audio, Image and Biomedical Signal Processing using Neural Networks* (pp. 327-349), https://doi.org/10.1007/978-3-540-75398-8_15

Brownlee, J. (2018). What is the Difference between a Batch and an Epoch in a Neural Network. *Machine Learning Mastery*, *20*. Retrieved from: https://deeplearning.lipingyang.org/wp-content/uploads/2018/07/What-is-the-Difference-Between-a-Batch-and-an-Epoch-in-a-Neural-Network_.pdf

Carballido, J., Rodríguez-Lizana, A., Agüera, J., & Perez-Ruiz, M. (2013). Field sprayer for inter and intra-row weed control: performance and labor savings. *Spanish Journal of Agricultural Research*, 11, 642-651, http://dx.doi.org/10.5424/sjar/2013113-3812

Carneiro, T., Da Nóbrega, R. V. M., Nepomuceno, T., Bian, G. B., De Albuquerque, V. H. C., & Reboucas-Filho, P. P. (2018). Performance analysis of google colaboratory as a tool for accelerating Deep Learning applications. *IEEE Access,* 6, 61677-61685, doi: 10.1109/ACCESS.2018.2874767

Cevallos, J. C., Villagomez, J. A., & Andryshchenko, I. S. (2019). Convolutional neural network in the recognition of spatial images of sugarcane crops in the troncal region of the coast of Ecuador. *Procedia Computer Science*, 150, 757-763, https://doi.org/10.1016/j.procs.2019.02.001

Champ, J., Mora-Fallas, A., Goëau, H., Mata-Montero, E., Bonnet, P., & Joly, A. (2020). Instance segmentation for the fine detection of crop and weed plants by precision agricultural robots. *Applications in plant sciences,* 8(7), e11373, https://doi.org/10.1002/aps3.11373

Chang, L., Deng, X., Zhou, M., Wu, Z., Yuan, Y., Yang, S., & Wang, H. (2016). Convolution neural network in image understanding. *ACTA automatic sinica,* 42(09), 1300-1312.

Chauhan, B. S. (2020). Grand challenges in weed management. *Frontiers in Agronomy*, 1, 3, https://doi.org/10.3389/fagro.2019.00003

Chauvel, B., Guillemin, J. P., Gasquez, J., & Gauvrit, C. (2012). History of chemical weeding from 1944 to 2011 in France: Changes and evolution of herbicide molecules. *Crop Protection*, 42, 320-326, https://doi.org/10.1016/j.cropro.2012.07.011

Chelghoum, R., Ikhlef, A., Hameurlaine, A., & Jacquir, S. (2020). Transfer learning using convolutional neural network architectures for brain tumor classification from MRI images. In *IFIP International Conference on Artificial Intelligence Applications and Innovations* (pp. 189-200).

Chen, C. J., Huang, Y. Y., Li, Y. S., Chen, Y. C., Chang, C. Y., Huang, Y. M. (2021a). Identification of Fruit Tree Pests with Deep Learning on Embedded Drone to Achieve Accurate Pesticide Spraying. *IEEE Access 2021,* 9, 21986–21997, doi: 10.1109/ACCESS.2021.3056082

Chen, L., Li, S., Bai, Q., Yang, J., Jiang, S., & Miao, Y. (2021b). Review of Image Classification Algorithms Based on Convolutional Neural Networks. *Remote Sensing*, 13(22), 4712, https://doi.org/10.3390/rs13224712

Chen, S., Wang, H., Xu, F., & Jin, Y. Q. (2016). Target classification using the deep convolutional networks for SAR images. *IEEE transactions on geoscience and remote sensing*, 54(8), 4806-4817, doi: 10.1109/TGRS.2016.2551720

Chen, Y. P., Li, Y., Wang, G., & Xu, Q. (2018). A multi-strategy region proposal network. *Expert Systems with Applications*, 113, 1-17, https://doi.org/10.1016/j.eswa.2018.06.043

Chen, Y. Q., Peng, S. U. I., Chen, L. U. A. N., & SHI, X. P. (2012). Xanthium suppression under maize|| sunflower intercropping system. *Journal of Integrative Agriculture*, 11(6), 1026-1037, https://doi.org/10.1016/S2095-3119(12)60095-1

Chen, Y., Lee, W.S., Gan, H., Peres, N., Fraisse, C., Zhang, Y., & He, Y. (2019). Strawberry Yield Prediction Based on a Deep Neural Network Using High-Resolution Aerial Orthoimages. *Remote Sensing*, 11, 1584, https://doi.org/10.3390/rs11131584

Chi, Z., Li, Y., & Chen, C. (2019). Deep convolutional neural network combined with concatenated spectrogram for environmental sound classification. In *2019 IEEE 7th International Conference on Computer Science and Network Technology (ICCSNT)* (pp. 251-254).

Colombo-Filho, M. E., Mello Galliez, R., Andrade Bernardi, F., Oliveira, L. L. D., Kritski, A., Koenigkam Santos, M., & Alves, D. (2020). Preliminary results on pulmonary tuberculosis detection in chest x-ray using convolutional neural networks. In *International Conference on Computational Science* (pp. 563-576).

Combarnous, Y. (2017). Endocrine Disruptor Compounds (EDCs) and agriculture: The case of pesticides. *Comptes Rendus Biologies*, 340(9-10), 406-409, https://doi.org/10.1016/j.crvi.2017.07.009

Crookston, R. K. (2006). A top 10 list of developments and issues impacting crop management and ecology during the past 50 years. *Crop science*, 46(5), 2253-2262, https://doi.org/10.2135/cropsci2005.11.0416gas

Csillik, O., Cherbini, J., Johnson, R., Lyons, A., & Kelly, M. (2018). Identification of Citrus Trees from Unmanned Aerial Vehicle Imagery Using Convolutional Neural Networks. *Drones*, 2, 39, https://doi.org/10.3390/drones2040039

Cui, D., & Curry, D. (2005). Prediction in marketing using the support vector machine. *Marketing Science*, *24*(4), 595-615, https://doi.org/10.1287/mksc.1050.0123

Da Costa Lima, A., & Mendes, K. F. (2020). Variable rate application of herbicides for weed management in pre-and postemergence. In *Pests, weeds and diseases in agricultural crop and animal husbandry production*, doi: https://dx.doi.org/10.5772/intechopen.93558

Dargan, S., Kumar, M., Ayyagari, M. R., & Kumar, G. (2020). A survey of Deep Learning and its applications: a new paradigm to machine learning. *Archives of Computational Methods in Engineering*, 27(4), 1071-1092, https://doi.org/10.1007/s10462-018-9633-3

De Camargo, T., Schirrmann, M., Landwehr, N., Dammer, K. H., & Pflanz, M. (2021). Optimized Deep Learning model as a basis for fast UAV mapping of weed species in winter wheat crops. *Remote Sensing*, 13(9), 1704, https://doi.org/10.3390/rs13091704

De Castro, A. I., Jurado-Expósito, M., Peña-Barragán, J. M., & López-Granados, F. (2012). Airborne multi-spectral imagery for mapping cruciferous weeds in cereal and legume crops. *Precision Agriculture*, 13(3), 302-321, https://doi.org/10.1007/s11119-011-9247-0

De Castro, A. I., Torres-Sánchez, J., Peña, J. M., Jiménez-Brenes, F. M., Csillik, O., & López-Granados, F. (2018). An automatic random forest-OBIA algorithm for early weed mapping between and within crop rows using UAV imagery. *Remote Sensing*, 10(2), 285, doi.org/10.3390/rs10020285

Di Cicco, M., Potena, C., Grisetti, G., & Pretto, A. (2017). Automatic model based dataset generation for fast and accurate crop and weeds detection. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (pp. 5188-5195).

Di-Tomaso, J. M., Monaco, T. A., James, J. J., & Firn, J. (2017). Invasive plant species and novel rangeland systems. In *Rangeland systems*, (pp. 429-465), doi: 10.1007/978-3-319-46709-2

Dogo, E. M., Afolabi, O. J., Nwulu, N. I., Twala, B., & Aigbavboa, C. O. (2018). A comparative analysis of gradient descent-based optimization algorithms on convolutional neural networks. In *2018 international conference on computational techniques, electronics and mechanical systems (CTEMS)* (pp. 92-99), https://doi.org/10.1109/CTEMS.2018.8769211

Dose, H., Møller, J. S., Iversen, H. K., & Puthusserypady, S. (2018). An end-to-end Deep Learning approach to MI-EEG signal classification for BCIs. *Expert Systems with Applications*, 114, 532-542, https://doi.org/10.1016/j.eswa.2018.08.031

Du, J. (2018). Understanding of object detection based on CNN family and YOLO. In *Journal of Physics: Conference Series*, 1004(1), 012029. IOP Publishing, doi: 10.1088/1742-6596/1004/1/012029

Dwivedi, A., Naresh, R., Kumar, R., Yadav, R.S. & Kumar, R. (2017). Precision Agriculture. In *Promoting Agri-Hortucultural, Technological Innovations,* (pp. 83-105).

Dyrmann, M., Karstoft, H., & Midtiby, H. S. (2016). Plant species classification using deep convolutional neural network. *Biosystems engineering*, 151, 72-80, https://doi.org/10.1016/j.biosystemseng.2016.08.024

Esposito, M., Crimaldi, M., Cirillo, V., Sarghini, F., & Maggio, A. (2021). Drone and sensor technology for sustainable weed management: A review. *Chemical and Biological Technologies in Agriculture*, 8(1), 1-11, https://doi.org/10.1186/s40538-021-00217-8

Everingham, M., Eslami, S. M., Van Gool, L., Williams, C. K., Winn, J., & Zisserman, A. (2015). The pascal visual object classes challenge: A retrospective. *International journal of computer vision*, 111(1), 98-136, https://doi.org/10.1007/s11263-014-0733-5

Farooq, O., Mubeen, K., Ali, H. H., & Ahmad, S. (2019). Non-chemical Weed Management for Field Crops. *In Agronomic Crops*, (pp. 317-348), doi: 10.1007/978-981-32-9783-8_16

Francies, M. L., Ata, M. M., & Mohamed, M. A. (2022). A robust multiclass 3D object recognition based on modern YOLO Deep Learning algorithms. *Concurrency and Computation: Practice and Experience*, 34(1), e6517, https://doi.org/10.1002/cpe.6517

Gemtos, T., Fountas, S., Tagarakis, A., & Liakos, V. (2013). Precision agriculture application in fruit crops: Experience in handpicked fruits. *Procedia Technology*, 8, 324-332, https://doi.org/10.1016/j.protcy.2013.11.043

Ghafoorian, M., Karssemeijer, N., Heskes, T., van Uden, I. W., Sanchez, C. I., Litjens, G., & Platel, B. (2017). Location sensitive deep convolutional neural networks for segmentation of white matter hyperintensities. *Scientific Reports*, 7(1), 1-12, https://doi.org/10.1038/s41598-017-05300-5

Gianessi, L. P. (2013). The increasing importance of herbicides in worldwide crop production. *Pest management science*, 69(10), 1099-1105, https://doi.org/10.1002/ps.3598

Girshick, R. (2015). Fast r-cnn. In *Proceedings of the IEEE international conference on computer vision*, (pp. 1440-1448).

Girshick, R., Donahue, J., Darrell, T., & Malik, J. (2015). Region-based convolutional networks for accurate object detection and segmentation. *IEEE transactions on pattern analysis and machine intelligence*, 38(1), 142-158, doi: 10.1109/TPAMI.2015.2437384

Gong, Y., Wang, L., Guo, R., & Lazebnik, S. (2014). Multi-scale orderless pooling of deep convolutional activation features. In *European conference on computer vision*, (pp. 392-407).

Gothai, E., Natesan, P., Aishwariya, S., Aarthy, T. B., & Singh, G. B. (2020). Weed Identification using Convolutional Neural Network and Convolutional Neural Network Architectures. In *2020 Fourth International Conference on Computing Methodologies and Communication (ICCMC)*, (pp. 958-965).

Grenzdörffer, G. J., Engel, A., & Teichert, B. (2008). The photogrammetric potential of low-cost UAVs in forestry and agriculture. *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, *31*(B3), 1207-1214.

Griepentrog, H. W., & Dedousis, A. P. (2010). Mechanical weed control. In *Soil Engineering*, 171-179, doi: 10.1007/978-3-642-03681-1_11

Gu, J., Wang, Z., Kuen, J., Ma, L., Shahroudy, A., Shuai, B., & Chen, T. (2018). Recent advances in convolutional neural networks. *Pattern recognition*, *77*, 354-377.

Hasan, A. M., Sohel, F., Diepeveen, D., Laga, H., Jones, & M. G. (2021). A survey of Deep Learning techniques for weed detection from images. *Computers and Electronics in Agriculture*, 184, 106067. https://doi.org/10.1016/j.compag.2021.106067

Hashemi-Beni, L., & Gebrehiwot, A. (2020). Deep Learning for remote sensing image classification for agriculture applications. *The International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences*, 51-54, https://doi.org/10.5194/isprs-archives-XLIV-M-2-2020-51-2020

Hassanein, M., & El-sheimy, N. (2017). Efficient Weed Detection Using Low-Cost UAV System. In *Proceedings of the 10th International Conference for Mobile Mapping Technology*.

He, K., & Sun, J. (2015). Convolutional neural networks at constrained time cost. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, (pp. 5353-5360).

He, K., Zhang, X., Ren, S., & Sun, J. (2015a). Spatial pyramid pooling in deep convolutional networks for visual recognition. *IEEE transactions on pattern analysis and machine intelligence*, 37(9), 1904-1916, https://doi.org/10.1109/TPAMI.2015.2389824

He, K., Zhang, X., Ren, S., & Sun, J. (2015b). Delving deep into rectifiers: Surpassing human-level performance on imagenet classification, in: *IEEE International Conference on Computer Vision (ICCV)*, (pp. 1026–1034).

Hemanth, A. S. (2022). Face Mask Detection Using YOLO v5. *IJNRD-International Journal of Novel Research and Development (IJNRD)*, 7(5), 390-395.

Hervás Martínez, C., Pérez Ortiz, M., Peña Barragán, J. M., Gutiérrez, P. A., Torres Sánchez, J., & López Granados, F. (2015). A weed monitoring system using UAV-imagery and the Hough transform. In *XV Congress of the Spanish Society of Malherbology: Malherbology and technology transfer: Seville, October 19-22, 2015*, (pp. 233-239).

Hobbs, P. R., Sayre, K., & Gupta, R. (2008). The role of conservation agriculture in sustainable agriculture. *Philosophical Transactions of the Royal Society B: Biological Sciences*, 363(1491), 543-555, https://doi.org/10.1098/rstb.2007.2169

Hoiem, D., Gupta, T., Li, Z., & Shlapentokh-Rothman, M. (2021). Learning curves for analysis of deep networks. In *International conference on machine learning* (pp. 4287-4296). PMLR.

Hoppin, J. A. (2014). Pesticides and respiratory health: where do we go from here? *Occupational and environmental medicine*, 71(2), 80-80, http://dx.doi.org/10.1136/oemed-2013-101876

Hu, G., Yin, C., Wan, M., Zhang, Y., & Fang, Y. (2020). Recognition of diseased Pinus trees in UAV images using Deep Learning and AdaBoost classifier. *Biosystems Engineering*, 194, 138-151, https://doi.org/10.1016/j.biosystemseng.2020.03.021

Hu, Y. (2021). *Traffic Fatality Rate Prediction Based on Deep Neural Network and Bayesian Neural Network* (Doctoral dissertation, Northern Illinois University).

Huang, H., Deng, J., Lan, Y., Yang, A., Deng, X., & Zhang, L. (2018). A fully convolutional network for weed mapping of unmanned aerial vehicle (UAV) imagery. *PloS one*, 13(4), e0196302.

Huang, H., Deng, J., Lan, Y., Yang, A., Zhang, L., Wen, S., & Deng, Y. (2019). Detection of helminthosporium leaf blotch disease based on UAV imagery. *Applied Sciences*, 9(3), 558, https://doi.org/10.3390/app9030558

Huang, J., Zhou, W., Li, H., & Li, W. (2015). Sign language recognition using 3d convolutional neural networks. In *2015 IEEE international conference on multimedia and expo (ICME)*, (pp. 1-6).

Huang, N. F., Chou, D. L., Lee, C. A., Wu, F. P., Chuang, A. C., Chen, Y. H., & Tsai, Y. C. (2020). Smart agriculture: real-time classification of green coffee beans by using a convolutional neural network. *IET Smart Cities*, 2(4), 167-172, https://doi.org/10.1049/iet-smc.2020.0068

Huang, Y., & Thomson, S. J. (2015). Remote sensing for cotton farming. *Cotton*, 57, 439-464, https://doi.org/10.2134/agronmonogr57.2013.0030

Ide, H., & Kurita, T. (2017). Improvement of learning for CNN with ReLU activation by sparse regularization. In *2017 International Joint Conference on Neural Networks (IJCNN)*, (pp. 2684-2691).

Indolia, S., Goswami, A. K., Mishra, S. P., & Asopa, P. (2018). Conceptual understanding of convolutional neural network-a Deep Learning approach. *Procedia computer science*, 132, 679-688, https://doi.org/10.1016/j.procs.2018.05.069

Islam, S. S., Rahman, S., Rahman, M. M., Dey, E. K., & Shoyaib, M. (2016). Application of Deep Learning to computer vision: A comprehensive study. In *2016 5th international conference on informatics, electronics and vision (ICIEV)* (pp. 592-597).

Jabir, B., & Falih, N. (2022). Deep Learning-based decision support system for weeds detection in wheat fields. *International Journal of Electrical and Computer Engineering*, 12(1), 816, doi: 10.11591/ijece.v12i1.pp816-825

Jabran, K., Mahajan, G., Sardana, V., & Chauhan, B. S. (2015). Allelopathy for weed control in agricultural systems. *Crop protection*, 72, 57-65, https://doi.org/10.1016/j.cropro.2015.03.004

Jiang, H., Zhang, C., Qiao, Y., Zhang, Z., Zhang, W., & Song, C. (2020). CNN feature based graph convolutional network for weed and crop recognition in smart farming. *Computers and Electronics in Agriculture*, 174, 105450, https://doi.org/10.1016/j.compag.2020.105450

Jocher, G., Nishimura, K., Mineeva, T., & Vilariño, R. (2020). YOLOv5 (2020). Retrieved from GitHub repository: https://github.com/ultralytics/yolov5

Kerkech, M., Hafiane, A., & Canals, R. (2018). Deep leaning approach with colorimetric spaces and vegetation indices for vine diseases detection in UAV images. *Computers and electronics in agriculture*, 155, 237-243, https://doi.org/10.1016/j.compag.2018.10.006

Kerkech, M., Hafiane, A., & Canals, R. (2020). VddNet: Vine Disease Detection Network Based on Multispectral Images and Depth Map. *Remote Sensing*, 12(20), 3305, https://doi.org/10.3390/rs12203305

Khan, A., Sohail, A., Zahoora, U., & Qureshi, A. S. (2020). A survey of the recent architectures of deep convolutional neural networks. *Artificial intelligence review*, 53(8), 5455-5516.

Kilichan, R., & Yilmaz, M. (2020). Artificial intelligence and robotic technologies in tourism and hospitality industry. *Erciyes University Journal of Social Sciences Institute*, (50), 353-380, https://doi.org/10.48070/erusosbilder.838193

Kim, D., Park, S., Kang, D., & Paik, J. (2019). Improved center and scale prediction-based pedestrian detection using convolutional block. In *2019 IEEE 9th International Conference on Consumer Electronics (ICCE-Berlin)* (pp. 418-419).

Knezevic, S. Z., & Datta, A. (2015). The critical period for weed control: revisiting data analysis. *Weed Science*, 63(SP1), 188-202, https://doi.org/10.1614/WS-D-14-00035.1

Kramer, O. (2013). K-nearest neighbors. In *Dimensionality reduction with unsupervised nearest neighbors* (pp. 13-23). Springer, Berlin, Heidelberg, https://doi.org/10.1007/978-3-642-38652-7

Krizhevsky, A., Sutskever, I., & Hinton, G.E. (2012). ImageNet classification with deep convolutional neural networks. In *Proceedings of the Advances in Neural Information Processing Systems, Lake Tahoe, NV, USA*, (pp. 1097–1105).

Kumar, S., Karaliya, S.K. & Chaudhary, S. (2017). Precision Farming Technologies towards Enhancing Productivity and Sustainability of Rice-Wheat Cropping System. *International Journal of Current Microbiology and Applied Sciences*, 6(3), 142-151, https://doi.org/10.20546/ijcmas.2017.603.016

Kumar, S., Nehra, M., Dilbaghi, N., Marrazza, G., Hassan, A. A., & Kim, K. H. (2019). Nano-based smart pesticide formulations: Emerging opportunities for agriculture. *Journal of Controlled Release*, 294, 131-153, https://doi.org/10.1016/j.jconrel.2018.12.012

Lati, R. N., Rasmussen, J., Andujar, D., Dorado, J., Berge, T. W., Wellhausen, C., & Christensen, S. (2021). Site-specific weed management—constraints and opportunities for the weed research community: *Insights from a workshop. Weed Research*, 61(3), 147-153.

LeCun, Y., & Bengio, Y. (1995). Convolutional networks for images, speech, and time series. *The handbook of brain theory and neural networks*, 3361(10).

LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep Learning. *Nature*, 521(7553), 436-444, https://doi.org/10.1038/nature14539.

Le, V. N. T., Truong, G., & Alameh, K. (2021). Detecting weeds from crops under complex field environments based on faster RCNN. In*2020 IEEE eighth international conference on communications and electronics (ICCE)* (pp.350-355). IEEE.

Lee, S. H., Chan, C. S., Wilkin, P., & Remagnino, P., (2015). Deep-plant: Plant identification with convolutional neural networks. In *2015 IEEE international conference on image processing (ICIP), IEEE*, (pp. 452-456).

Li, F., Liu, Z., Shen, W., Wang, Y., Wang, Y., Ge, C., & Lan, P. (2021). A remote sensing and airborne edge-computing based detection system for pine wilt disease. *IEEE Access*, 9, 66346-66360, doi: 10.1109/ACCESS.2021.3073929

Liu, B., & Bruch, R. (2020). Weed detection for selective spraying: A review. *Current Robotics Reports*, 1(1), 19-26.

Liu, K., Tang, H., He, S., Yu, Q., Xiong, Y., & Wang, N. (2021). Performance validation of YOLO variants for object detection. In *Proceedings of the 2021 International Conference on Bioinformatics and Intelligent Computing* (pp. 239-243).

Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C. Y., & Berg, A. C. (2016b). Ssd: Single shot multibox detector. *In European conference on computer vision* (pp. 21-37).

Liu, W., Wang, Z., Liu, X., Zeng, N., Liu, Y., & Alsaadi, F. E. (2017). A survey of deep neural network architectures and their applications. *Neurocomputing*, 234, 11-26, https://doi.org/10.1016/j.neucom.2016.12.038

Liu, X. P., Li, G., Liu, L., & Wang, Z. (2019). Improved YOLOV3 target recognition algorithm based on adaptive eged optimization. *Microelectronics and Computer*, 36(7), 59-64.

Liu, Y., Wang, Y., & Zhang, J. (2012). New machine learning algorithm: Random forest. In *International Conference on Information Computing and Applications* (pp. 246-252).

Lottes, P., Behley, J., Chebrolu, N., Milioto, A., & Stachniss, C. (2018). Joint stem detection and crop-weed classification for plant-specific treatment in precision farming. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (pp. 8233-8238).

Luo, X., Li, S., & Xu, H. (2016). Results of real-time kinematic positioning based on real GPS L5 data. *IEEE Geoscience and Remote Sensing Letters*, 13(8), 1193-1197.

Maas, A. L., Hannun, A. Y., & Ng, A. Y. (2013). Rectifier Nonlinearities Improve Neural Network Acoustic Models. *Proceedings of the 30th International Conference on Machine Learning, Atlanta, Georgia, USA, 30*(1), 3.

Machleb, J., Peteinatos, G. G., Kollenda, B. L., Andújar, D., & Gerhards, R. (2020). Sensor-based mechanical weed control: Present state and prospects. *Computers and electronics in agriculture*, 176, 105638, https://doi.org/10.1016/j.compag.2020.105638

Magomadov, V. S. (2019). Deep Learning and its role in smart agriculture. In *Journal of Physics: Conference Series*, 1399(4), 044109, doi: 10.1088/1742-6596/1399/4/044109

Malta, A., Mendes, M., & Farinha, T. (2021). Augmented reality maintenance assistant using yolov5. *Applied Sciences*, 11(11), 4758, https://doi.org/10.3390/app11114758

Maxwell, A. E., Warner, T. A., & Guillén, L. A. (2021). Accuracy assessment in convolutional neural network-based Deep Learning remote sensing studies—part 1: Literature review. *Remote Sensing*, 13(13), 2450.

Mazzia, V., Comba, L., Khaliq, A., Chiaberge, M., & Gay, P. (2020). UAV and machine learning based refinement of a satellite-driven vegetation index for precision agriculture. *Sensors*, 20(9), 2530, https://doi.org/10.3390/s20092530

McCabe, M.F., Houborg, R., & Rosas, J., (2015). The potential of unmanned aerial vehicles for providing information on vegetation health, in: *Proceedings of the 21st International Congress on Modelling and Simulation. Gold Coast, Australia*, (pp. 1399–1405).

McFadyen, R. E. (2012). Food security for a 9 billion population: more R&D for weed control will be critical. In *Proc. 18th Australasian Weeds Conference* (pp. 306-309).

Mendez, K. M., Pritchard, L., Reinke, S. N., & Broadhurst, D. I. (2019). Toward collaborative open data science in metabolomics using Jupyter Notebooks and cloud computing. *Metabolomics*, 15(10), 1-16, https://doi.org/10.1007/s11306-019-1588-0

Meyer, G. E., & Mulliken, J. A. (2008). Weed Targeting Herbicide Management. *Historical Materials from University of Nebraska-Lincoln Extension*, 4871, https://digitalcommons.unl.edu/extensionhist/4871

Monteiro, A., & Santos, S. (2022). Sustainable Approach to Weed Management: The Role of Precision Weed Management. *Agronomy*, 12(1), 118, https://doi.org/10.3390/agronomy12010118

Mora-Fallas, A., Goëau, H., Joly, A., Bonnet, P., & Mata-Montero, E. (2020). Instance segmentation for automated weeds and crops detection in farmlands. A first approach to Acoustic Characterization of Costa Rican Children's Speech. Retrieved from: https://www.academia.edu/44819282/A_first_approach_to_Acoustic_Characterization_of_Costa_Rican_Children_s_Speech

Moran, M., Inoue, Y. & Barnes, E. (1997). Opportunities and limitations for image-based remote sensing in precision crop management. *Remote Sensing of Environment*, 61(3), 319-346, https://doi.org/10.1016/S0034-4257(97)00045-X

Mortensen, A. K., Dyrmann, M., Karstoft, H., Jørgensen, R. N., & Gislum, R. (2016). Semantic segmentation of mixed crops using deep convolutional neural network. In *CIGR-Agricultural Engineering conference* (pp. 26-29).

Mulla, D. J., (2013). Twenty five years of remote sensing in precision agriculture: Key advances and remaining knowledge gaps. *Biosystems Engineering.* 114, 358–371, https://doi.org/10.1016/j.biosystemseng.2012.08.009

Murawwat, S., Qureshi, A., Ahmad, S., & Shahid, Y., (2018). Weed Detection Using SVMs. *Engineering, Technology & Applied Science Research*, 8(1), 2412-2416.

Nair, V. & Hinton, G. E. (2010). Rectified linear units improve restricted boltzmann machines, in: *International Conference on Machine Learning (ICML), 2010, (*pp. 807–814).

Nanni, L., Ghidoni, S., & Brahnam, S. (2017). Handcrafted vs. non-handcrafted features for computer vision classification. *Pattern Recognition*, 71, 158-172, https://doi.org/10.1016/j.patcog.2017.05.025

Nex, F., & Remondino, F. (2014). UAV for 3D mapping applications: a review. *Applied geomatics*, 6(1), 1-15, https://doi.org/10.1007/s12518-013-0120-x

Nogueira, K., Penatti, O. A., & Dos Santos, J. A. (2017). Towards better exploiting convolutional neural networks for remote sensing scene classification. *Pattern Recognition,* 61, 539-556, https://doi.org/10.1016/j.patcog.2016.07.001

Olsen, A., Konovalov, D. A., Philippa, B., Ridd, P., Wood, J. C., Johns, J., & White, R. D. (2019). DeepWeeds: A multiclass weed species image dataset for Deep

Learning. *Scientific reports*, 9(1), 1-12, https://doi.org/10.1038/s41598-018-38343-3

Onishi, M., & Ise, T. (2018). Automatic classification of trees using a UAV onboard camera and Deep Learning. *Computer and information sciences*, https://doi.org/10.48550/arXiv.1804.07437

Osco, L. P., Nogueira, K., Marques Ramos, A. P., Faita Pinheiro, M. M., Furuya, D. E. G., Gonçalves, W. N., & dos Santos, J. A. (2021). Semantic segmentation of citrus-orchard using deep neural networks and multispectral UAV-based imagery. *Precision Agriculture*, 22(4), 1171-1188, https://doi.org/10.1007/s11119-020-09777-5

Patidar, S., Singh, U., & Sharma, S. K. (2020, July). Weed seedling detection using mask regional convolutional neural network. In *2020 International Conference on Electronics and Sustainable Communication Systems (ICESC)* (pp. 311-316).

Patil-Shirish, S., & Bhalerao, S. A. (2013). Precision farming: the most scientific and modern approach to sustainable agriculture. *International Research Journal of Science and Engineering*, 1(2), 21-30.

Pena, J. M., Torres-Sánchez, J., De Castro, A. I., Kelly, M., & López-Granados, F. (2013). Weed mapping in early-season maize fields using object-based analysis of unmanned aerial vehicle (UAV) images. *PloS one*, 8(10), e77151, https://doi.org/10.1371/journal.pone.0077151

Pena, J.M., Torres-Sánchez, J., Serrano-Pérez, A., De Castro, A.I., & López-Granados, F. (2015). Quantifying efficacy and limits of unmanned aerial vehicle (UAV) technology for weed seedling detection as affected by sensor resolution. *Sensors* 15, 5609–5626, https://doi.org/10.3390/s150305609

Pena-Barragán, J. M., Kelly, M., De-Castro, A. I., & López-Granados, F. (2012). Discrimination of Crop Rows using Object-Based Analysis in UAV Images for early Site-Specific Weed Management in Maize Fields. In *Proceedings of the First International Conference on Robotics and Associated High-technologies and Equipment for Agriculture. Applications of automated systems and robotics for crop protection in sustainable precision agriculture,(RHEA-2012) Pisa, Italy-September 19-21, 2012* (pp. 249-254).

Perez, F., & Granger, B. E. (2007). IPython: a system for interactive scientific computing. *Computing in science & engineering*, 9(3), 21-29, https://doi.org/10.1109/MCSE.2007.53

Peterson, M. A., Collavo, A., Ovejero, R., Shivrain, V., & Walsh, M. J. (2018). The challenge of herbicide resistance around the world: a current summary. *Pest management science*, 74(10), 2246 -2259, https://doi.org/10.1002/ps.4821

Popp, J., Pető, K., & Nagy, J. (2013). Pesticide productivity and food security. A review. *Agronomy for sustainable development*, 33(1), 243-255, https://doi.org/10.1007/s13593-012-0105-x

Potena, C., Nardi, D., & Pretto, A. (2017). Fast and accurate crop and weed identification with summarized train sets for precision agriculture. In *International Conference on Intelligent Autonomous Systems*, (pp. 105-121), doi: 10.1007/978-3-319-48036-7_9

Pouyanfar, S., Sadiq, S., Yan, Y., Tian, H., Tao, Y., Reyes, M. P., & Iyengar, S. S. (2018). A survey on Deep Learning: Algorithms, techniques, and applications. *ACM Computing Surveys (CSUR)*, 51(5), 1-36, https://doi.org/10.1145/3234150

Prashanth, B., Mendu, M., & Thallapalli, R. (2021). Cloud based Machine learning with advanced predictive Analytics using Google Colaboratory. *Materials Today: Proceedings*, https://doi.org/10.1016/j.matpr.2021.01.800

Prashanth, D. S., Mehta, R. V. K., & Sharma, N. (2020). Classification of handwritten Devanagari number–an analysis of pattern recognition tool using neural network and CNN. *Procedia Computer Science*, 167, 2445-2457, https://doi.org/10.1016/j.procs.2020.03.297

Pratama, K., & Kang, D. K. (2021). Trainable activation function with differentiable negative side and adaptable rectified point. *Applied Intelligence*, 51(3), 1784-1801, https://doi.org/10.1007/s10489-020-01885-z

Pu, Y., Apel, D. B., Szmigiel, A., & Chen, J. (2019). Image recognition of coal and coal gangue using a convolutional neural network and transfer learning. *Energies*, 12(9), 1735, https://doi.org/10.3390/en12091735

Ramirez, W., Achanccaray, P., Mendoza, L. F., & Pacheco, M. A. C. (2020). Deep convolutional neural networks for weed detection in agricultural crops using optical aerial images. In *2020 IEEE Latin American GRSS & ISPRS Remote Sensing Conference (LAGIRS)* (pp. 133-137).

Rana, K. (2020). Pooling Layer — Short and Simple. Retrieved from plainenglish: https://ai.plainenglish.io/pooling-layer-beginner-to-intermediate-fa0dbdce80eb

Randles, B. M., Pasquetto, I. V., Golshan, M. S., & Borgman, C. L. (2017). Using the Jupyter notebook as a tool for open science: An empirical study. In *2017 ACM/IEEE Joint Conference on Digital Libraries (JCDL)* (pp. 1-2).

Reddy, R. R., & Panicker, M. R. (2021). Hand-Drawn Electrical Circuit Recognition using Object Detection and Node Recognition. *Computer and information sciences*, https://doi.org/10.48550/arXiv.2106.11559

Regnier, E. E., & Janke, R. R. (2020). Evolving strategies for managing weeds. In *Sustainable agricultural systems* (pp. 174-202).

Ren, A., Li, Z., Wang, Y., Qiu, Q., & Yuan, B. (2016). Designing reconfigurable large-scale Deep Learning systems using stochastic computing. In *2016 IEEE International Conference on Rebooting Computing (ICRC)* (pp. 1-7).

Ren, S., He, K., Girshick, R., & Sun, J. (2015). Faster RCNN: Towards real-time object detection with region proposal networks. *Advances in neural information processing systems*, 28, https://doi.org/10.1109/TPAMI.2016.2577031

Renard, D., Iriarte, J., Birk, J. J., Rostain, S., Glaser, B., & McKey, D. (2012). Ecological engineers ahead of their time: The functioning of pre-Columbian raised-field agriculture and its potential contributions to sustainability today. *Ecological Engineering*, 45, 30-44, https://doi.org/10.1016/j.ecoleng.2011.03.007

Rippel, O., Snoek, J., & Adams, R. P. (2015). Spectral representations for convolutional neural networks. *Advances in neural information processing systems*, 28.

Rodriguez-Galiano, V. F., Ghimire, B., Rogan, J., Chica-Olmo, M., & Rigol-Sanchez, J. P. (2012). An assessment of the effectiveness of a random forest classifier for land-cover classification. *ISPRS journal of photogrammetry and remote sensing*, 67, 93-104, https://doi.org/10.1016/j.isprsjprs.2011.11.002

Rueda-Ayala, V. P., Rasmussen, J., Gerhards, R., & Fournaise, N. E. (2011). The influence of post-emergence weed harrowing on selectivity, crop recovery and crop yield in different growth stages of winter wheat. *Weed Research*, 51(5), 478-488, https://doi.org/10.1111/j.1365-3180.2011.00873.x

Sa, I., Popović, M., Khanna, R., Chen, Z., Lottes, P., Liebisch, F., & Siegwart, R. (2018). WeedMap: A large-scale semantic weed mapping framework using aerial multispectral imaging and deep neural network for precision farming. *Remote Sensing*, 10(9), 1423, https://doi.org/10.3390/rs10091423

Sabat-Tomala, A., Raczko, E., & Zagajewski, B. (2020). Comparison of support vector machine and random forest algorithms for invasive and expansive species classification using airborne hyperspectral data. *Remote Sensing*, 12(3), 516. https://doi.org/10.3390/rs12030516

Sahu, B., Chatterjee, S., Mukherjee, S., & Sharma, C. (2019). Tools of precision agri-culture: A review. *International Journal of Chemical Studies*, 7, 2692-2696.

Savalia, S., & Emamian, V. (2018). Cardiac arrhythmia classification by multi-layer perceptron and convolution neural networks. *Bioengineering*, 5(2), 35, https://doi.org/10.3390/bioengineering5020035

Scavo, A., & Mauromicale, G. (2020). Integrated weed management in herbaceous field crops. *Agronomy*, 10(4), 466, https://doi.org/10.3390/agronomy10040466

Schmidhuber, J. (2015). Deep Learning in neural networks: An overview. *Neural networks*, 61, 85-117, https://doi.org/10.1016/j.neunet.2014.09.003

Sermanet, P., Chintala, S., & LeCun, Y. (2012). Convolutional neural networks applied to house numbers digit classification. In *Proceedings of the 21st international conference on pattern recognition (ICPR2012)* (pp. 3288-3291).

Shanmugam, S., Assunção, E., Mesquita, R., Veiros, A., & Gaspar, P. D. (2020). Automated weed detection systems: A review. *KnE Engineering*, 5(6), 271–284. https://doi.org/10.18502/keg.v5i6.7046

Sharma, A., Liu, X., Yang, X., & Shi, D. (2017a). A patch-based convolutional neural network for remote sensing image classification. *Neural Networks*, 95, 19-28, https://doi.org/10.1016/j.neunet.2017.07.017

Sharma, S., Sharma, S., & Athaiya, A. (2017b). Activation functions in neural networks. *Towards data science*, 6(12), 310-316.

Shirzadifar, A. M., Loghavi, M., & Raoufat, M. H. (2015). Development and evaluation of a real time site-specific inter-row weed management system. *Iran Agricultural Research*, 32(2), 39-54, doi: 10.22099/iar.2015.2004

Shrestha, M., & Khanal, S. (2020). Future prospects of precision agriculture in Nepal. *Archives of Agriculture and Environmental Science*, 5(3), 397-405, https://dx.doi.org/10.26832/24566632.2020.0503023

Slaughter, D. C., Giles, D. K., & Downey, D. (2008). Autonomous robotic weed control systems: A review. *Computers and electronics in agriculture*, 61(1), 63-78, https://doi.org/10.1016/j.compag.2007.05.008

Smith, P. (2018). Drones in Precision Agriculture. Retrieved from dronebelow: https://dronebelow.com/2018/07/19/drones-in-precision-agriculture

Smith, R. G., Ryan, M. R., & Menalled, F. D. (2011). Direct and indirect impacts of weed management practices on soil quality. *Soil management: Building a stable base for agriculture*, 275-286, https://doi.org/10.2136/2011.soilmanagement.c18

Soltys, D., Krasuska, U., Bogatek, R., & Gniazdowska, A. (2013). *Allelochemicals as bioherbicides—Present and perspectives*. London, UK: IntechOpen limited, doi: 10.5772/56185

Sornam, M., Muthusubash, K., & Vanitha, V. (2017). A survey on image classification and activity recognition using deep convolutional neural network architecture. In *2017 ninth international conference on advanced computing (ICoAC)* (pp. 121-126).

Srinivas, S., Sarvadevabhatla, R. K., Mopuri, K. R., Prabhu, N., Kruthiventi, S. S., & Babu, R. V. (2016). A taxonomy of deep convolutional neural nets for computer vision. *Frontiers in Robotics and AI*, 2, 36, https://doi.org/10.3389/frobt.2015.00036

Starling, A. P., Umbach, D. M., Kamel, F., Long, S., Sandler, D. P., & Hoppin, J. A. (2014). Pesticide use and incident diabetes among wives of farmers in the Agricultural Health Study. *Occupational and environmental medicine*, 71(9), 629-635, http://dx.doi.org/10.1136/oemed-2013-101659

Stewart, E. L., Wiesner-Hanks, T., Kaczmar, N., DeChant, C., Wu, H., Lipson, H., & Gore, M. A. (2019). Quantitative phenotyping of Northern Leaf Blight in UAV images using Deep Learning. *Remote Sensing*, 11(19), 2209, https://doi.org/10.3390/rs11192209

Suhail, A., Jayabalan, M., & Thiruchelvam, V. (2020). Convolutional neural network based object detection: A review. *Journal of critical reviews*, 7(11), 786-792, https://doi.org/10.1002/9781119681328.ch6

Suljović, A., Čakić, S., Popović, T., & Šandi, S. (2022, March). Detection of Plant Diseases Using Leaf Images and Machine Learning. In *2022 21st International Symposium INFOTEH-JAHORINA (INFOTEH)* (pp. 1-4).

Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., & Rabinovich, A. (2015). Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 1-9).

Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., & Wojna, Z. (2016). Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 2818-2826).

Tang, Z., Luo, L., Peng, H., & Li, S. (2018). A joint residual network with paired ReLUs activation for image super-resolution. *Neurocomputing*, 273, 37-46, https://doi.org/10.1016/j.neucom.2017.07.061

Tarca, A. L., Carey, V. J., Chen, X. W., Romero, R., & Drăghici, S. (2007). Machine learning and its applications to biology. *PLoS computational biology*, 3(6), e116, https://doi.org/10.1371/journal.pcbi.0030116

Teimouri, N., Dyrmann, M., Nielsen, P. R., Mathiassen, S. K., Somerville, G. J., & Jørgensen, R. N. (2018). Weed growth stage estimator using deep convolutional neural networks. *Sensors*, 18(5), 1580, https://doi.org/10.3390/s18051580

Thuan, D. (2021). Evolution of yolo algorithm and yolov5: the state-of-the-art object detection algorithm (Bachelor's Thesis, Oulu University of Applied Sciences). Retrieved from: https://www.theseus.fi/handle/10024/452552

Tirado, R., Englande, A. J., Promakasikorn, L., & Novotny, V. (2008). Use of agrochemicals in Thailand and its consequences for the environment. *Greenpeace Research Laboratories Technical. Bangkok, Thailand*.

Toda, Y., Okura, F., Ito, J., Okada, S., Kinoshita, T., Tsuji, H., & Saisho, D. (2020). Training instance segmentation neural network with synthetic datasets for crop seed phenotyping. *Communications biology*, 3(1), 1-12, https://doi.org/10.1038/s42003-020-0905-5

Torres-Sánchez, J., López-Granados, F., & Pena, J. M. (2015). An automatic object-based method for optimal thresholding in UAV images: Application for vegetation detection in herbaceous crops. *Computers and Electronics in Agriculture*, 114, 43-52, https://doi.org/10.1016/j.compag.2015.03.019

Torres-Sánchez, J., López-Granados, F., De Castro, A. I., & Peña-Barragán, J. M. (2013). Configuration and specifications of an unmanned aerial vehicle (UAV) for early site specific weed management. *PloS one*, 8(3), e58210, https://doi.org/10.1371/journal.pone.0058210

Tsouros, D. C., Smyrlis, P. N., Tsipouras, M. G., Tsalikakis, D. G., Giannakeas, N., Tzallas, A. T., & Manousou, P. (2017). Automated collagen proportional area

extraction in liver biopsy images using a novel classification via clustering algorithm. In *2017 IEEE 30th International Symposium on Computer-Based Medical Systems (CBMS)* (pp. 30-34).

Ukaegbu, U., Tartibu, L., Okwu, M., & Olayode, I. (2021). Development of a Light-Weight Unmanned Aerial Vehicle for Precision Agriculture. *Sensors*, 21, 4417, https://doi.org/10.3390/s21134417

Valente, J., Doldersum, M., Roers, C., & Kooistra, L. (2019). Detecting Rumex Obtusifolius Weed Plants in Grasslands from UAV RGB Imagery using Deep Learning. *ISPRS Annals of Photogrammetry, Remote Sensing & Spatial Information Sciences*, 4.

Vargas, V. M., Gutiérrez, P. A., Barbero-Gómez, J., & Hervás-Martínez, C. (2021). Activation Functions for Convolutional Neural Networks: Proposals and Experimental Study. *IEEE Transactions on Neural Networks and Learning Systems*. Retrieved from: https://ieeexplore.ieee.org/document/9521668/

Veeranampalayam Sivakumar, A. N., Li, J., Scott, S., Psota, E., J. Jhala, A., Luck, J. D., & Shi, Y. (2020). Comparison of object detection and patch-based classification Deep Learning models on mid-to late-season weed detection in UAV imagery. *Remote Sensing*, 12(13), 2136, https://doi.org/10.3390/rs12132136

Vijayaprabakaran, K., & Sathiyamurthy, K. (2020). Towards activation function search for long short-term model network: a differential evolution based approach. *Journal of King Saud University-Computer and Information Sciences*. Retrieved from: https://www.sciencedirect.com/science/article/pii/S1319157820303505

Vilà, M., Williamson, M., & Lonsdale, M. (2004). Competition experiments on alien weeds with crops: lessons for measuring plant invasion impact? *Biological invasions*, 6(1), 59-69, https://doi.org/10.1023/B:BINV.0000010122.77024.8a

Viquerat, J., & Hachem, E. (2020). A supervised neural network for drag prediction of arbitrary 2D shapes in laminar flows at low Reynolds number. *Computers & Fluids*, 210, 104645, https://doi.org/10.1016/j.compfluid.2020.104645

Vissoh, P. V., Gbèhounou, G., Ahanchédé, A., Kuyper, T. W., & Röling, N. G. (2004). Weeds as agricultural constraint to farmers in Benin: results of a diagnostic study. *NJAS-Wageningen Journal of Life Sciences*, 52(3-4), 305-329, https://doi.org/10.1016/S1573-5214(04)80019-8

Voulodimos, A., Doulamis, N., Doulamis, A., & Protopapadakis, E. (2018). Deep Learning for computer vision: A brief review. *Computational intelligence and neuroscience*, *2018*. Retrieved from: https://www.hindawi.com/journals/CIN/2018/7068349/

Waheed, A., Goyal, M., Gupta, D., Khanna, A., Hassanien, A. E., & Pandey, H. M. (2020). An optimized dense convolutional neural network model for disease recognition and classification in corn leaf. *Computers and Electronics in Agriculture*, 175, 105456, https://doi.org/10.1016/j.compag.2020.105456

Wang, L., Li, Z., Zhao, J., Zhou, K., Wang, Z., & Yuan, H. (2016). Smart device-supported BDS/GNSS real-time kinematic positioning for sub-meter-level accuracy in urban location-based services. *Sensors*, 16(12), 2201, https://doi.org/10.3390/s16122201

Wang, W., Xie, E., Song, X., Zang, Y., Wang, W., Lu, T., & Shen, C. (2019). Efficient and accurate arbitrary-shaped text detection with pixel aggregation network. In *Proceedings of the IEEE/CVF International Conference on Computer Vision* (pp. 8440-8449).

Wang, Y. (2021). A Ros-Based Toy-Car Detect-And-Place Domestic Robot (*Master's thesis, California State Polytechnic University, Pomona*).

Westwood, J. H., Charudattan, R., Duke, S. O., Fennimore, S. A., Marrone, P., Slaughter, D. C., & Zollinger, R. (2018). Weed management in 2050: Perspectives on the future of weed science. *Weed science*, 66(3), 275-285, doi:10.1017/wsc.2017.78

Wikipedia Contributors. (2020). Real-time Kinematic. Retrieved from Wikipedia, The Free Encyclopedia: https://en.wikipedia.org/w/index.php?title=Real-time_kinematic&id=959795335

Wójtowicz, M., Wójtowicz, A., & Piekarczyk, J. (2016). Application of remote sensing methods in agriculture. *Communications in Biometry and Crop Science*, 11(1), 31-50.

Wu, C., Wen, W., Afzal, T., Zhang, Y., & Chen, Y. (2017). A compact dnn: approaching googlenet-level accuracy of classification and domain adaptation. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 5668-5677).

Xu, B., Wang, N., Chen, T., & Li, M. (2015). Empirical evaluation of rectified activations in convolutional network. In *International Conference on Machine Learning Workshop*, https://doi.org/10.48550/arXiv.1505.00853

Xu, R., Lin, H., Lu, K., Cao, L., & Liu, Y. (2021). A forest fire detection system based on ensemble learning. *Forests*, 12(2), 217, https://doi.org/10.3390/f12020217

Xu, W., Yang, W., Chen, S., Wu, C., Chen, P., & Lan, Y. (2020). Establishing a model to predict the single boll weight of cotton in northern Xinjiang by using high resolution UAV remote sensing data. *Computers and Electronics in Agriculture*, 179, 105762, https://doi.org/10.1016/j.compag.2020.105762

Yan, B., Fan, P., Lei, X., Liu, Z., & Yang, F. (2021). A real-time apple targets detection method for picking robot based on improved YOLOv5. *Remote Sensing*, 13(9), 1619, https://doi.org/10.3390/rs13091619

Yang, Z., Sinnott, R., Ke, Q., & Bailey, J. (2021). Individual Feral Cat Identification through Deep Learning. In *2021 IEEE/ACM 8th International Conference on Big Data Computing, Applications and Technologies (BDCAT'21)* (pp. 101-110), https://doi.org/10.1145/3492324.3494168

Yao, H., & Huang, Y. (2013). Remote sensing applications to precision farming. *Remote sensing of natural resources*, (pp. 333-352), https://doi.org/10.1201/b15159

Yi, J., Wu, P., Liu, B., Huang, Q., Qu, H., & Metaxas, D. (2021). Oriented object detection in aerial images with box boundary-aware vectors. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision* (pp. 2150-2159).

You, J., Liu, W., & Lee, J. (2020). A DNN-based semantic segmentation for detecting weed and crop. *Computers and Electronics in Agriculture*, 178, 105750, https://doi.org/10.1016/j.compag.2020.105750

Young, S. L., Pierce, F. J., & Nowak, P. (2014). Introduction: Scope of the problem rising costs and demand for environmental safety for weed control. In *Automation: The future of weed control in cropping systems* (pp. 1-8), https://doi.org/10.1007/978-94-007-7512-1_1

Yousefi, M. R., & Razdari, A. M. (2015). Application of GIS and GPS in precision agriculture (a review). *International Journal of Advanced Biological and Biomedical Research*, 3(1), 7-9.

Yu, D., Wang, H., Chen, P., & Wei, Z. (2014). Mixed pooling for convolutional neural networks. In *International conference on rough sets and knowledge technology* (pp. 364-375), https://doi.org/10.1007/978-3-319-11740-9_34

Yu, J., Sharpe, S. M., Schumann, A. W., & Boyd, N. S. (2019). Deep Learning for image-based weed detection in turfgrass. *European journal of agronomy*, 104, 78-84, https://doi.org/10.1016/j.eja.2019.01.004

Zahara, L., Musa, P., Wibowo, E. P., Karim, I., & Musa, S. B. (2020). The facial emotion recognition (FER-2013) dataset for prediction system of micro-expressions face using the convolutional neural network (CNN) algorithm based Raspberry Pi. In *2020 Fifth international conference on informatics and computing (ICIC)* (pp. 1-9).

Zeiler, M. D., & Fergus, R. (2013). Stochastic pooling for regularization of deep convolutional neural networks. *Computer and information sciences*, https://doi.org/10.48550/arXiv.1301.3557

Zhang, C., & Kovacs, J. M. (2012). The application of small unmanned aerial systems for precision agriculture: a review. *Precision agriculture*, 13(6), 693-712, doi: 10.1007/s11119-012-9274-5

Zhang, J., Xie, T., Yang, C., Song, H., Jiang, Z., Zhou, G., & Xie, J. (2020a). Segmenting purple rapeseed leaves in the field from UAV RGB imagery using Deep Learning as an auxiliary means for nitrogen stress detection. *Remote Sensing*, 12(9), 1403, https://doi.org/10.3390/rs12091403

Zhang, Q., Zhang, M., Chen, T., Sun, Z., Ma, Y., & Yu, B. (2019). Recent advances in convolutional neural network acceleration. *Neurocomputing*, 323, 37-51, https://doi.org/10.1016/j.neucom.2018.09.038

Zhang, Z. D., Tan, M. L., Lan, Z. C., Liu, H. C., Pei, L., Yu, W. X. (2022). CDNet: a real-time and robust crosswalk detection network on Jetson nano based on YOLOv5. *Neural Computing and Applications*, (pp. 1-12), https://doi.org/10.1007/s00521-022-07007-9

Zhang, Z., Flores, P., Igathinathane, C., L. Naik, D., Kiran, R., & Ransom, J. K. (2020). Wheat lodging detection from UAS imagery using machine learning algorithms. *Remote Sensing*, 12(11), 1838, doi:10.3390/rs12111838

Zhao, J., Zhong, Y., Hu, X., Wei, L., & Zhang, L. (2020). A robust spectral-spatial approach to identifying heterogeneous crops using remote sensing imagery with high spectral and spatial resolutions. *Remote Sensing of Environment*, 239, 111605, https://doi.org/10.1016/j.rse.2019.111605

Zhu, P., Wen, L., Bian, X., Ling, H., & Hu, Q. (2018). Vision meets drones: A challenge. *Computer and information sciences*, https://doi.org/10.48550/arXiv.1804.07437

Zou, K., Chen, X., Zhang, F., Zhou, H., & Zhang, C. (2021). A field weed density evaluation method based on UAV imaging and modified U-Net. *Remote Sensing*, 13(2), 310, https://doi.org/10.3390/rs13020310

Appendix A

Ground Control Points (GCP's)

S/N	Station ID	Control Category	Coordinates (m)		Height
			Northing	Easting	
0.	CSN 128P	Primary	1056599.017	222702.652	245.519
1.	JA01	Secondary	1053910	225665.4	209.9429
2.	JA02	Secondary	1053950	225650.8	209.6335
3.	JA03	Secondary	1053993	225697.0	209.7679
4.	JA04	Secondary	1054008	225727.8	209.6353
5.	JA05	Secondary	1054005	225779.2	209.186
6.	JA06	Secondary	1053948	225758.6	209.1649
7.	JA07	Secondary	1053914	225713.8	211.1355
8.	JA08	Secondary	1053945	225706.5	210.6092

Appendix B

Training codes for Faster RCNN

```
import functools
import json
import os
import tensorflow as tf

from object_detection import trainer
from object_detection.builders import input_reader_builder
from object_detection.builders import model_builder
from object_detection.utils import config_util

tf.logging.set_verbosity(tf.logging.INFO)

flags = tf.app.flags
flags.DEFINE_string('master', '', 'Name of the TensorFlow master to
use.')
flags.DEFINE_integer('task', 0, 'task id')
flags.DEFINE_integer('num_clones', 1, 'Number of clones to deploy
per worker.')
flags.DEFINE_boolean('clone_on_cpu', False,
                     'Force clones to be deployed on CPU.  Note
that even if '
                     'set to False (allowing ops to run on gpu),
some ops may '
                     'still be run on the CPU if they have no GPU
kernel.')
flags.DEFINE_integer('worker_replicas', 1, 'Number of
worker+trainer '
                     'replicas.')
flags.DEFINE_integer('ps_tasks', 0,
                     'Number of parameter server tasks. If None,
does not use '
                     'a parameter server.')
flags.DEFINE_string('train_dir', '',
                    'Directory to save the checkpoints and training
summaries.')

flags.DEFINE_string('pipeline_config_path', '',
                    'Path to a pipeline_pb2.TrainEvalPipelineConfig
config '
                    'file. If provided, other configs are ignored')

flags.DEFINE_string('train_config_path', '',
                    'Path to a train_pb2.TrainConfig config file.')
flags.DEFINE_string('input_config_path', '',
                    'Path to an input_reader_pb2.InputReader config
file.')
flags.DEFINE_string('model_config_path', '',
                    'Path to a model_pb2.DetectionModel config
file.')

FLAGS = flags.FLAGS


def main(_):
```

```
  assert FLAGS.train_dir, '`train_dir` is missing.'
  if FLAGS.task == 0: tf.gfile.MakeDirs(FLAGS.train_dir)
  if FLAGS.pipeline_config_path:
    configs = config_util.get_configs_from_pipeline_file(
        FLAGS.pipeline_config_path)
    if FLAGS.task == 0:
      tf.gfile.Copy(FLAGS.pipeline_config_path,
                    os.path.join(FLAGS.train_dir,
'pipeline.config'),
                    overwrite=True)
  else:
    configs = config_util.get_configs_from_multiple_files(
        model_config_path=FLAGS.model_config_path,
        train_config_path=FLAGS.train_config_path,
        train_input_config_path=FLAGS.input_config_path)
    if FLAGS.task == 0:
      for name, config in [('model.config',
FLAGS.model_config_path),
                           ('train.config',
FLAGS.train_config_path),
                           ('input.config',
FLAGS.input_config_path)]:
        tf.gfile.Copy(config, os.path.join(FLAGS.train_dir, name),
                      overwrite=True)

  model_config = configs['model']
  train_config = configs['train_config']
  input_config = configs['train_input_config']

  model_fn = functools.partial(
      model_builder.build,
      model_config=model_config,
      is_training=True)

  create_input_dict_fn = functools.partial(
      input_reader_builder.build, input_config)

  env = json.loads(os.environ.get('TF_CONFIG', '{}'))
  cluster_data = env.get('cluster', None)
  cluster = tf.train.ClusterSpec(cluster_data) if cluster_data else
None
  task_data = env.get('task', None) or {'type': 'master', 'index':
0}
  task_info = type('TaskSpec', (object,), task_data)

  # Parameters for a single worker.
  ps_tasks = 0
  worker_replicas = 1
  worker_job_name = 'lonely_worker'
  task = 0
  is_chief = True
  master = ''

  if cluster_data and 'worker' in cluster_data:
    # Number of total worker replicas include "worker"s and the
"master".
    worker_replicas = len(cluster_data['worker']) + 1
  if cluster_data and 'ps' in cluster_data:
```

```
    ps_tasks = len(cluster_data['ps'])

  if worker_replicas > 1 and ps_tasks < 1:
    raise ValueError('At least 1 ps task is needed for distributed
training.')

  if worker_replicas >= 1 and ps_tasks > 0:
    # Set up distributed training.
    server = tf.train.Server(tf.train.ClusterSpec(cluster),
protocol='grpc',
                             job_name=task_info.type,
                             task_index=task_info.index)
    if task_info.type == 'ps':
      server.join()
      return

    worker_job_name = '%s/task:%d' % (task_info.type,
task_info.index)
    task = task_info.index
    is_chief = (task_info.type == 'master')
    master = server.target

  trainer.train(create_input_dict_fn, model_fn, train_config,
master, task,
                FLAGS.num_clones, worker_replicas,
FLAGS.clone_on_cpu, ps_tasks,
                worker_job_name, is_chief, FLAGS.train_dir)


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

Testing Codes for Faster RCNN

```
import numpy as np
import os
import six.moves.urllib as urllib
import sys
import tarfile
import tensorflow as tf
import zipfile

from distutils.version import StrictVersion
from collections import defaultdict
from io import StringIO
from matplotlib import pyplot as plt
from PIL import Image

# This is needed since the notebook is stored in the object_detecti
on folder.
sys.path.append("..")
from object_detection.utils import ops as utils_ops
from object_detection.utils import label_map_util
```

```
from object_detection.utils import visualization_utils as vis_util



### Model preparation variable
MODEL_NAME = 'trained_inference_graph'
PATH_TO_FROZEN_GRAPH = MODEL_NAME + '/frozen_inference_graph.pb'
PATH_TO_LABELS = 'training/labelmap.pbtxt'
NUM_CLASSES = 5 #remember number of objects you are training? cool.


### Load a (frozen) Tensorflow model into memory.
detection_graph = tf.Graph()
with detection_graph.as_default():
  od_graph_def = tf.GraphDef()
  with tf.gfile.GFile(PATH_TO_FROZEN_GRAPH, 'rb') as fid:
    serialized_graph = fid.read()
    od_graph_def.ParseFromString(serialized_graph)
    tf.import_graph_def(od_graph_def, name='')


###Loading label map
category_index = label_map_util.create_category_index_from_labelmap
(PATH_TO_LABELS)




### Load image into numpy function
def load_image_into_numpy_array(image):
  (im_width, im_height) = image.size
  return np.array(image.getdata()).reshape(
      (im_height, im_width, 3)).astype(np.uint8)




###STATING THE PATH TO IMAGES TO BE TESTED
PATH_TO_TEST_IMAGES_DIR = 'test_images/'
TEST_IMAGE_PATHS = [ os.path.join(PATH_TO_TEST_IMAGES_DIR, 'image{}
.jpg'.format(i)) for i in range(1, 5) ]
IMAGE_SIZE = (12, 8)




### Function to run inference on a single image which will later be
 used in an iteration
def run_inference_for_single_image(image, graph):
  with graph.as_default():
```

```
    with tf.Session() as sess:
      # Get handles to input and output tensors
      ops = tf.get_default_graph().get_operations()
      all_tensor_names = {output.name for op in ops for output in o
p.outputs}
      tensor_dict = {}
      for key in [
          'num_detections', 'detection_boxes', 'detection_scores',
          'detection_classes', 'detection_masks'
      ]:
        tensor_name = key + ':0'
        if tensor_name in all_tensor_names:
          tensor_dict[key] = tf.get_default_graph().get_tensor_by_n
ame(
              tensor_name)
      if 'detection_masks' in tensor_dict:
        # The following processing is only for single image
        detection_boxes = tf.squeeze(tensor_dict['detection_boxes']
, [0])
        detection_masks = tf.squeeze(tensor_dict['detection_masks']
, [0])
        # Reframe is required to translate mask from box coordinate
s to image coordinates and fit the image size.
        real_num_detection = tf.cast(tensor_dict['num_detections'][
0], tf.int32)
        detection_boxes = tf.slice(detection_boxes, [0, 0], [real_n
um_detection, -1])
        detection_masks = tf.slice(detection_masks, [0, 0, 0], [rea
l_num_detection, -1, -1])
        detection_masks_reframed = utils_ops.reframe_box_masks_to_i
mage_masks(
            detection_masks, detection_boxes, image.shape[1], image
.shape[2])
        detection_masks_reframed = tf.cast(
            tf.greater(detection_masks_reframed, 0.5), tf.uint8)
        # Follow the convention by adding back the batch dimension
        tensor_dict['detection_masks'] = tf.expand_dims(
            detection_masks_reframed, 0)
      image_tensor = tf.get_default_graph().get_tensor_by_name('ima
ge_tensor:0')

      # Run inference
      output_dict = sess.run(tensor_dict,
                             feed_dict={image_tensor: image})

      # all outputs are float32 numpy arrays, so convert types as a
ppropriate
      output_dict['num_detections'] = int(output_dict['num_detectio
ns'][0])
```

```
      output_dict['detection_classes'] = output_dict[
          'detection_classes'][0].astype(np.int64)
      output_dict['detection_boxes'] = output_dict['detection_boxes
'][0]
      output_dict['detection_scores'] = output_dict['detection_scor
es'][0]
      if 'detection_masks' in output_dict:
        output_dict['detection_masks'] = output_dict['detection_mas
ks'][0]
  return output_dict



### To iterate on each image in the test image path defined
### NB define the range of numbers and let it match the number of i
mAGES IN TEST FOLDER +1
for image_path in TEST_IMAGE_PATHS:
  image = Image.open(image_path)
  # the array based representation of the image will be used later
in order to prepare the
  # result image with boxes and labels on it.
  image_np = load_image_into_numpy_array(image)
  # Expand dimensions since the model expects images to have shape:
 [1, None, None, 3]
  image_np_expanded = np.expand_dims(image_np, axis=0)
  # Actual detection.
  output_dict = run_inference_for_single_image(image_np_expanded, d
etection_graph)
  # Visualization of the results of a detection.
  vis_util.visualize_boxes_and_labels_on_image_array(
      image_np,
      output_dict['detection_boxes'],
      output_dict['detection_classes'],
      output_dict['detection_scores'],
      category_index,
      instance_masks=output_dict.get('detection_masks'),
      use_normalized_coordinates=True,
      line_thickness=1)
  display(Image.fromarray(image_np))
```

Appendix C

Training codes for YOLO v5

```
import argparse
import logging
import math
import os
import random
import time
from copy import deepcopy
from pathlib import Path
from threading import Thread

import numpy as np
import torch.distributed as dist
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
import torch.optim.lr_scheduler as lr_scheduler
import torch.utils.data
import yaml
from torch.cuda import amp
from torch.nn.parallel import DistributedDataParallel as DDP
from torch.utils.tensorboard import SummaryWriter
from tqdm import tqdm

import test  # import test.py to get mAP after each epoch
from models.experimental import attempt_load
from models.yolo import Model
from utils.autoanchor import check_anchors
from utils.datasets import create_dataloader
from utils.general import labels_to_class_weights, increment_path,
labels_to_image_weights, init_seeds, \
    fitness, strip_optimizer, get_latest_run, check_dataset, check_
file, check_git_status, check_img_size, \
    check_requirements, print_mutation, set_logging, one_cycle, col
orstr
from utils.google_utils import attempt_download
from utils.loss import ComputeLoss
from utils.plots import plot_images, plot_labels, plot_results, plo
t_evolution
from utils.torch_utils import ModelEMA, select_device, intersect_di
cts, torch_distributed_zero_first, is_parallel

logger = logging.getLogger(__name__)
```

```
def train(hyp, opt, device, tb_writer=None, wandb=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_b
atch_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.dump(hyp, f, sort_keys=False)
    with open(save_dir / 'opt.yaml', 'w') as f:
        yaml.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.load(f, Loader=yaml.SafeLoader)  # data di
ct
    with torch_distributed_zero_first(rank):
        check_dataset(data_dict)  # check
    train_path = data_dict['train']
    test_path = data_dict['val']
    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):
            attempt_download(weights)  # download if not found loca
lly
        ckpt = torch.load(weights, map_location=device)  # load che
ckpoint
        model = Model(opt.cfg or ckpt['model'].yaml, ch=3, nc=nc, a
nchors=hyp.get('anchors')).to(device)  # create
```

```
        exclude = ['anchor'] if (opt.cfg or hyp.get('anchors')) and
 not opt.resume else []  # exclude keys
        state_dict = ckpt['model'].float().state_dict()  # to FP32
        state_dict = intersect_dicts(state_dict, model.state_dict()
, exclude=exclude)  # intersect
        model.load_state_dict(state_dict, strict=False)  # load
        logger.info('Transferred %g/%g items from %s' % (len(state_
dict), len(model.state_dict()), weights))  # report
    else:
        model = Model(opt.cfg, ch=3, nc=nc, anchors=hyp.get('anchor
s')).to(device)  # create

    # Freeze
    freeze = []  # parameter names to freeze (full or partial)
    for k, v in model.named_parameters():
        v.requires_grad = True  # train all layers
        if any(x in k for x in freeze):
            print('freezing %s' % k)
            v.requires_grad = False

    # Optimizer
    nbs = 64  # nominal batch size
    accumulate = max(round(nbs / total_batch_size), 1)  # accumulat
e loss before optimizing
    hyp['weight_decay'] *= total_batch_size * accumulate / nbs  # s
cale weight_decay
    logger.info(f"Scaled weight_decay = {hyp['weight_decay']}")

    pg0, pg1, pg2 = [], [], []  # optimizer parameter groups
    for k, v in model.named_modules():
        if hasattr(v, 'bias') and isinstance(v.bias, nn.Parameter):
            pg2.append(v.bias)  # biases
        if isinstance(v, nn.BatchNorm2d):
            pg0.append(v.weight)  # no decay
        elif hasattr(v, 'weight') and isinstance(v.weight, nn.Param
eter):
            pg1.append(v.weight)  # apply decay

    if opt.adam:
        optimizer = optim.Adam(pg0, lr=hyp['lr0'], betas=(hyp['mome
ntum'], 0.999))  # adjust beta1 to momentum
    else:
        optimizer = optim.SGD(pg0, lr=hyp['lr0'], momentum=hyp['mom
entum'], nesterov=True)

    optimizer.add_param_group({'params': pg1, 'weight_decay': hyp['
weight_decay']})  # add pg1 with weight_decay
    optimizer.add_param_group({'params': pg2})  # add pg2 (biases)
```

```
    logger.info('Optimizer groups: %g .bias, %g conv.weight, %g oth
er' % (len(pg2), len(pg1), len(pg0)))
    del pg0, pg1, pg2

    # Scheduler https://arxiv.org/pdf/1812.01187.pdf
    # https://pytorch.org/docs/stable/_modules/torch/optim/lr_sched
uler.html#OneCycleLR
    if opt.linear_lr:
        lf = lambda x: (1 - x / (epochs - 1)) * (1.0 - hyp['lrf'])
+ hyp['lrf']  # linear
    else:
        lf = one_cycle(1, hyp['lrf'], epochs)  # cosine 1-
>hyp['lrf']
    scheduler = lr_scheduler.LambdaLR(optimizer, lr_lambda=lf)
    # plot_lr_scheduler(optimizer, scheduler, epochs)

    # Logging
    if rank in [-1, 0] and wandb and wandb.run is None:
        opt.hyp = hyp  # add hyperparameters
        wandb_run = wandb.init(config=opt, resume="allow",
                               project='YOLOv5' if opt.project == '
runs/train' else Path(opt.project).stem,
                               name=save_dir.stem,
                               entity=opt.entity,
                               id=ckpt.get('wandb_id') if 'ckpt' in
 locals() else None)
    loggers = {'wandb': wandb}  # loggers dict

    # EMA
    ema = ModelEMA(model) if rank in [-1, 0] else None

    # Resume
    start_epoch, best_fitness = 0, 0.0
    if pretrained:
        # Optimizer
        if ckpt['optimizer'] is not None:
            optimizer.load_state_dict(ckpt['optimizer'])
            best_fitness = ckpt['best_fitness']

        # EMA
        if ema and ckpt.get('ema'):
            ema.ema.load_state_dict(ckpt['ema'].float().state_dict(
))
            ema.updates = ckpt['updates']

        # Results
        if ckpt.get('training_results') is not None:
            results_file.write_text(ckpt['training_results'])  # wr
ite results.txt
```

```
        # Epochs
        start_epoch = ckpt['epoch'] + 1
        if opt.resume:
            assert start_epoch > 0, '%s training to %g epochs is fi
nished, nothing to resume.' % (weights, epochs)
        if epochs < start_epoch:
            logger.info('%s has been trained for %g epochs. Fine-
tuning for %g additional epochs.' %
                        (weights, ckpt['epoch'], epochs))
            epochs += ckpt['epoch']  # finetune additional epochs

        del ckpt, state_dict

    # Image sizes
    gs = max(int(model.stride.max()), 32)  # grid size (max stride)
    nl = model.model[-
1].nl  # number of detection layers (used for scaling hyp['obj'])
    imgsz, imgsz_test = [check_img_size(x, gs) for x in opt.img_siz
e]  # verify imgsz are gs-multiples

    # DP mode
    if cuda and rank == -1 and torch.cuda.device_count() > 1:
        model = torch.nn.DataParallel(model)

    # SyncBatchNorm
    if opt.sync_bn and cuda and rank != -1:
        model = torch.nn.SyncBatchNorm.convert_sync_batchnorm(model
).to(device)
        logger.info('Using SyncBatchNorm()')

    # Trainloader
    dataloader, dataset = create_dataloader(train_path, imgsz, batc
h_size, gs, opt,
                                            hyp=hyp, augment=True,
cache=opt.cache_images, rect=opt.rect, rank=rank,
                                            world_size=opt.world_si
ze, workers=opt.workers,
                                            image_weights=opt.image
_weights, quad=opt.quad, prefix=colorstr('train: '))
    mlc = np.concatenate(dataset.labels, 0)[:, 0].max()  # max labe
l class
    nb = len(dataloader)  # number of batches
    assert mlc < nc, 'Label class %g exceeds nc=%g in %s. Possible
class labels are 0-%g' % (mlc, nc, opt.data, nc - 1)

    # Process 0
    if rank in [-1, 0]:
```

```
        testloader = create_dataloader(test_path, imgsz_test, batch
_size * 2, gs, opt,  # testloader
                                       hyp=hyp, cache=opt.cache_ima
ges and not opt.notest, rect=True, rank=-1,
                                       world_size=opt.world_size, w
orkers=opt.workers,
                                       pad=0.5, prefix=colorstr('va
l: '))[0]

        if not opt.resume:
            labels = np.concatenate(dataset.labels, 0)
            c = torch.tensor(labels[:, 0])  # classes
            # cf = torch.bincount(c.long(), minlength=nc) + 1.  # f
requency
            # model._initialize_biases(cf.to(device))
            if plots:
                plot_labels(labels, save_dir, loggers)
                if tb_writer:
                    tb_writer.add_histogram('classes', c, 0)

            # Anchors
            if not opt.noautoanchor:
                check_anchors(dataset, model=model, thr=hyp['anchor
_t'], imgsz=imgsz)
            model.half().float()  # pre-reduce anchor precision

    # DDP mode
    if cuda and rank != -1:
        model = DDP(model, device_ids=[opt.local_rank], output_devi
ce=opt.local_rank)

    # Model parameters
    hyp['box'] *= 3. / nl  # scale to layers
    hyp['cls'] *= nc / 80. * 3. / nl  # scale to classes and layers
    hyp['obj'] *= (imgsz / 640) ** 2 * 3. / nl  # scale to image si
ze and layers
    model.nc = nc  # attach number of classes to model
    model.hyp = hyp  # attach hyperparameters to model
    model.gr = 1.0  # iou loss ratio (obj_loss = 1.0 or iou)
    model.class_weights = labels_to_class_weights(dataset.labels, n
c).to(device) * nc  # attach class weights
    model.names = names

    # Start training
    t0 = time.time()
    nw = max(round(hyp['warmup_epochs'] * nb), 1000)  # number of w
armup iterations, max(3 epochs, 1k iterations)
    # nw = min(nw, (epochs - start_epoch) / 2 * nb)  # limit warmup
 to < 1/2 of training
```

```
    maps = np.zeros(nc)  # mAP per class
    results = (0, 0, 0, 0, 0, 0, 0)  # P, R, mAP@.5, mAP@.5-
.95, val_loss(box, obj, cls)
    scheduler.last_epoch = start_epoch - 1  # do not move
    scaler = amp.GradScaler(enabled=cuda)
    compute_loss = ComputeLoss(model)  # init loss class
    logger.info(f'Image sizes {imgsz} train, {imgsz_test} test\n'
                f'Using {dataloader.num_workers} dataloader workers
\n'
                f'Logging results to {save_dir}\n'
                f'Starting training for {epochs} epochs...')
    for epoch in range(start_epoch, epochs):  # epoch ------------
-----------------------------------------------------
        model.train()

        # Update image weights (optional)
        if opt.image_weights:
            # Generate indices
            if rank in [-1, 0]:
                cw = model.class_weights.cpu().numpy() * (1 - maps)
 ** 2 / nc  # class weights
                iw = labels_to_image_weights(dataset.labels, nc=nc,
 class_weights=cw)  # image weights
                dataset.indices = random.choices(range(dataset.n),
weights=iw, k=dataset.n)  # rand weighted idx
            # Broadcast if DDP
            if rank != -1:
                indices = (torch.tensor(dataset.indices) if rank ==
 0 else torch.zeros(dataset.n)).int()
                dist.broadcast(indices, 0)
                if rank != 0:
                    dataset.indices = indices.cpu().numpy()

        # Update mosaic border
        # b = int(random.uniform(0.25 * imgsz, 0.75 * imgsz + gs) /
/ gs * gs)
        # dataset.mosaic_border = [b - imgsz, -
b]  # height, width borders

        mloss = torch.zeros(4, device=device)  # mean losses
        if rank != -1:
            dataloader.sampler.set_epoch(epoch)
        pbar = enumerate(dataloader)
        logger.info(('\n' + '%10s' * 8) % ('Epoch', 'gpu_mem', 'box
', 'obj', 'cls', 'total', 'targets', 'img_size'))
        if rank in [-1, 0]:
            pbar = tqdm(pbar, total=nb)  # progress bar
        optimizer.zero_grad()
```

```
        for i, (imgs, targets, paths, _) in pbar:  # batch --------
------------------------------------------------------
            ni = i + nb * epoch  # number integrated batches (since
 train start)
            imgs = imgs.to(device, non_blocking=True).float() / 255
.0  # uint8 to float32, 0-255 to 0.0-1.0

            # Warmup
            if ni <= nw:
                xi = [0, nw]  # x interp
                # model.gr = np.interp(ni, xi, [0.0, 1.0])  # iou l
oss ratio (obj_loss = 1.0 or iou)
                accumulate = max(1, np.interp(ni, xi, [1, nbs / tot
al_batch_size]).round())
                for j, x in enumerate(optimizer.param_groups):
                    # bias lr falls from 0.1 to lr0, all other lrs
rise from 0.0 to lr0
                    x['lr'] = np.interp(ni, xi, [hyp['warmup_bias_l
r'] if j == 2 else 0.0, x['initial_lr'] * lf(epoch)])
                    if 'momentum' in x:
                        x['momentum'] = np.interp(ni, xi, [hyp['war
mup_momentum'], hyp['momentum']])

            # Multi-scale
            if opt.multi_scale:
                sz = random.randrange(imgsz * 0.5, imgsz * 1.5 + gs
) // gs * gs  # size
                sf = sz / max(imgs.shape[2:])  # scale factor
                if sf != 1:
                    ns = [math.ceil(x * sf / gs) * gs for x in imgs
.shape[2:]]  # new shape (stretched to gs-multiple)
                    imgs = F.interpolate(imgs, size=ns, mode='bilin
ear', align_corners=False)

            # Forward
            with amp.autocast(enabled=cuda):
                pred = model(imgs)  # forward
                loss, loss_items = compute_loss(pred, targets.to(de
vice))  # loss scaled by batch_size
                if rank != -1:
                    loss *= opt.world_size  # gradient averaged bet
ween devices in DDP mode
                if opt.quad:
                    loss *= 4.

            # Backward
            scaler.scale(loss).backward()

            # Optimize
```

```
            if ni % accumulate == 0:
                scaler.step(optimizer)  # optimizer.step
                scaler.update()
                optimizer.zero_grad()
                if ema:
                    ema.update(model)

            # Print
            if rank in [-1, 0]:
                mloss = (mloss * i + loss_items) / (i + 1)  # updat
e mean losses
                mem = '%.3gG' % (torch.cuda.memory_reserved() / 1E9
 if torch.cuda.is_available() else 0)  # (GB)
                s = ('%10s' * 2 + '%10.4g' * 6) % (
                    '%g/%g' % (epoch, epochs - 1), mem, *mloss, tar
gets.shape[0], imgs.shape[-1])
                pbar.set_description(s)

                # Plot
                if plots and ni < 3:
                    f = save_dir / f'train_batch{ni}.jpg'  # filena
me
                    Thread(target=plot_images, args=(imgs, targets,
 paths, f), daemon=True).start()
                    # if tb_writer:
                    #     tb_writer.add_image(f, result, dataformat
s='HWC', global_step=epoch)
                    #     tb_writer.add_graph(model, imgs)  # add m
odel to tensorboard
                elif plots and ni == 10 and wandb:
                    wandb.log({"Mosaics": [wandb.Image(str(x), capt
ion=x.name) for x in save_dir.glob('train*.jpg')
                                           if x.exists()]}, commit=
False)

            # end batch ------------------------------------------
-------------------------------------------------------
        # end epoch ------------------------------------------------
-------------------------------------------------------

        # Scheduler
        lr = [x['lr'] for x in optimizer.param_groups]  # for tenso
rboard
        scheduler.step()

        # DDP process 0 or single-GPU
        if rank in [-1, 0]:
            # mAP
```

```
            ema.update_attr(model, include=['yaml', 'nc', 'hyp', 'g
r', 'names', 'stride', 'class_weights'])
            final_epoch = epoch + 1 == epochs
            if not opt.notest or final_epoch:  # Calculate mAP
                results, maps, times = test.test(opt.data,
                                                 batch_size=batch_s
ize * 2,
                                                 imgsz=imgsz_test,
                                                 model=ema.ema,
                                                 single_cls=opt.sin
gle_cls,
                                                 dataloader=testloa
der,
                                                 save_dir=save_dir,
                                                 verbose=nc < 50 an
d final_epoch,
                                                 plots=plots and fi
nal_epoch,
                                                 log_imgs=opt.log_i
mgs if wandb else 0,
                                                 compute_loss=compu
te_loss)

            # Write
            with open(results_file, 'a') as f:
                f.write(s + '%10.4g' * 7 % results + '\n')  # appen
d metrics, val_loss
            if len(opt.name) and opt.bucket:
                os.system('gsutil cp %s gs://%s/results/results%s.t
xt' % (results_file, opt.bucket, opt.name))

            # Log
            tags = ['train/box_loss', 'train/obj_loss', 'train/cls_
loss',  # train loss
                    'metrics/precision', 'metrics/recall', 'metrics
/mAP_0.5', 'metrics/mAP_0.5:0.95',
                    'val/box_loss', 'val/obj_loss', 'val/cls_loss',
  # val loss
                    'x/lr0', 'x/lr1', 'x/lr2']  # params
            for x, tag in zip(list(mloss[:-
1]) + list(results) + lr, tags):
                if tb_writer:
                    tb_writer.add_scalar(tag, x, epoch)  # tensorbo
ard
                if wandb:
                    wandb.log({tag: x}, step=epoch, commit=tag == t
ags[-1])  # W&B

            # Update best mAP
```

```
            fi = fitness(np.array(results).reshape(1, -
1))  # weighted combination of [P, R, mAP@.5, mAP@.5-.95]
            if fi > best_fitness:
                best_fitness = fi

            # Save model
            if (not opt.nosave) or (final_epoch and not opt.evolve)
:  # if save
                ckpt = {'epoch': epoch,
                        'best_fitness': best_fitness,
                        'training_results': results_file.read_text(
),
                        'model': deepcopy(model.module if is_parall
el(model) else model).half(),
                        'ema': deepcopy(ema.ema).half(),
                        'updates': ema.updates,
                        'optimizer': optimizer.state_dict(),
                        'wandb_id': wandb_run.id if wandb else None
}

                # Save last, best and delete
                torch.save(ckpt, last)
                if best_fitness == fi:
                    torch.save(ckpt, best)
                del ckpt

        # end epoch ----------------------------------------------
----------------------------------------------------
    # end training

    if rank in [-1, 0]:
        # Strip optimizers
        final = best if best.exists() else last  # final model
        for f in last, best:
            if f.exists():
                strip_optimizer(f)
        if opt.bucket:
            os.system(f'gsutil cp {final} gs://{opt.bucket}/weights
')  # upload

        # Plots
        if plots:
            plot_results(save_dir=save_dir)  # save as results.png
            if wandb:
                files = ['results.png', 'confusion_matrix.png', *[f
'{x}_curve.png' for x in ('F1', 'PR', 'P', 'R')]]
                wandb.log({"Results": [wandb.Image(str(save_dir / f
), caption=f) for f in files
```

```
                                        if (save_dir / f).exists()]}
)
                if opt.log_artifacts:
                    wandb.log_artifact(artifact_or_path=str(final),
 type='model', name=save_dir.stem)

        # Test best.pt
        logger.info('%g epochs completed in %.3f hours.\n' % (epoch
 - start_epoch + 1, (time.time() - t0) / 3600))
        if opt.data.endswith('coco.yaml') and nc == 80:  # if COCO
            for m in (last, best) if best.exists() else (last):  #
speed, mAP tests
                results, _, _ = test.test(opt.data,
                                          batch_size=batch_size * 2
,
                                          imgsz=imgsz_test,
                                          conf_thres=0.001,
                                          iou_thres=0.7,
                                          model=attempt_load(m, dev
ice).half(),
                                          single_cls=opt.single_cls
,
                                          dataloader=testloader,
                                          save_dir=save_dir,
                                          save_json=True,
                                          plots=False)

    else:
        dist.destroy_process_group()

    wandb.run.finish() if wandb and wandb.run else None
    torch.cuda.empty_cache()
    return results


if __name__ == '__main__':
    parser = argparse.ArgumentParser()
    parser.add_argument('--
weights', type=str, default='yolov5s.pt', help='initial weights pat
h')
    parser.add_argument('--
cfg', type=str, default='', help='model.yaml path')
    parser.add_argument('--
data', type=str, default='data/coco128.yaml', help='data.yaml path'
)
    parser.add_argument('--
hyp', type=str, default='data/hyp.scratch.yaml', help='hyperparamet
ers path')
    parser.add_argument('--epochs', type=int, default=300)
```

```
    parser.add_argument('--batch-
size', type=int, default=16, help='total batch size for all GPUs')
    parser.add_argument('--img-
size', nargs='+', type=int, default=[640, 640], help='[train, test]
 image sizes')
    parser.add_argument('--
rect', action='store_true', help='rectangular training')
    parser.add_argument('--
resume', nargs='?', const=True, default=False, help='resume most re
cent training')
    parser.add_argument('--
nosave', action='store_true', help='only save final checkpoint')
    parser.add_argument('--
notest', action='store_true', help='only test final epoch')
    parser.add_argument('--
noautoanchor', action='store_true', help='disable autoanchor check'
)
    parser.add_argument('--
evolve', action='store_true', help='evolve hyperparameters')
    parser.add_argument('--
bucket', type=str, default='', help='gsutil bucket')
    parser.add_argument('--cache-
images', action='store_true', help='cache images for faster trainin
g')
    parser.add_argument('--image-
weights', action='store_true', help='use weighted image selection f
or training')
    parser.add_argument('--
device', default='', help='cuda device, i.e. 0 or 0,1,2,3 or cpu')
    parser.add_argument('--multi-
scale', action='store_true', help='vary img-size +/- 50%%')
    parser.add_argument('--single-
cls', action='store_true', help='train multi-class data as single-
class')
    parser.add_argument('--
adam', action='store_true', help='use torch.optim.Adam() optimizer'
)
    parser.add_argument('--sync-
bn', action='store_true', help='use SyncBatchNorm, only available i
n DDP mode')
    parser.add_argument('--local_rank', type=int, default=-
1, help='DDP parameter, do not modify')
    parser.add_argument('--log-
imgs', type=int, default=16, help='number of images for W&B logging
, max 100')
    parser.add_argument('--log-
artifacts', action='store_true', help='log artifacts, i.e. final tr
ained model')
```

```
    parser.add_argument('--
workers', type=int, default=8, help='maximum number of dataloader w
orkers')
    parser.add_argument('--
project', default='runs/train', help='save to project/name')
    parser.add_argument('--
entity', default=None, help='W&B entity')
    parser.add_argument('--
name', default='exp', help='save to project/name')
    parser.add_argument('--exist-
ok', action='store_true', help='existing project/name ok, do not in
crement')
    parser.add_argument('--
quad', action='store_true', help='quad dataloader')
    parser.add_argument('--linear-
lr', action='store_true', help='linear LR')
    opt = parser.parse_args()

    # Set DDP variables
    opt.world_size = int(os.environ['WORLD_SIZE']) if 'WORLD_SIZE'
in os.environ else 1
    opt.global_rank = int(os.environ['RANK']) if 'RANK' in os.envir
on else -1
    set_logging(opt.global_rank)
    if opt.global_rank in [-1, 0]:
        check_git_status()
        check_requirements()

    # Resume
    if opt.resume:  # resume an interrupted run
        ckpt = opt.resume if isinstance(opt.resume, str) else get_l
atest_run()  # specified or most recent path
        assert os.path.isfile(ckpt), 'ERROR: --
resume checkpoint does not exist'
        apriori = opt.global_rank, opt.local_rank
        with open(Path(ckpt).parent.parent / 'opt.yaml') as f:
            opt = argparse.Namespace(**yaml.load(f, Loader=yaml.Saf
eLoader))  # replace
        opt.cfg, opt.weights, opt.resume, opt.batch_size, opt.globa
l_rank, opt.local_rank = '', ckpt, True, opt.total_batch_size, *apr
iori  # reinstate
        logger.info('Resuming training from %s' % ckpt)
    else:
        # opt.hyp = opt.hyp or ('hyp.finetune.yaml' if opt.weights
else 'hyp.scratch.yaml')
        opt.data, opt.cfg, opt.hyp = check_file(opt.data), check_fi
le(opt.cfg), check_file(opt.hyp)  # check files
        assert len(opt.cfg) or len(opt.weights), 'either --cfg or -
-weights must be specified'
```

```
        opt.img_size.extend([opt.img_size[-
1]] * (2 - len(opt.img_size)))  # extend to 2 sizes (train, test)
        opt.name = 'evolve' if opt.evolve else opt.name
        opt.save_dir = increment_path(Path(opt.project) / opt.name,
 exist_ok=opt.exist_ok | opt.evolve)  # increment run

    # DDP mode
    opt.total_batch_size = opt.batch_size
    device = select_device(opt.device, batch_size=opt.batch_size)
    if opt.local_rank != -1:
        assert torch.cuda.device_count() > opt.local_rank
        torch.cuda.set_device(opt.local_rank)
        device = torch.device('cuda', opt.local_rank)
        dist.init_process_group(backend='nccl', init_method='env://
')  # distributed backend
        assert opt.batch_size % opt.world_size == 0, '--batch-
size must be multiple of CUDA device count'
        opt.batch_size = opt.total_batch_size // opt.world_size

    # Hyperparameters
    with open(opt.hyp) as f:
        hyp = yaml.load(f, Loader=yaml.SafeLoader)  # load hyps

    # Train
    logger.info(opt)
    try:
        import wandb
    except ImportError:
        wandb = None
        prefix = colorstr('wandb: ')
        logger.info(f"{prefix}Install Weights & Biases for YOLOv5 l
ogging with 'pip install wandb' (recommended)")
    if not opt.evolve:
        tb_writer = None  # init loggers
        if opt.global_rank in [-1, 0]:
            logger.info(f'Start Tensorboard with "tensorboard --
logdir {opt.project}", view at http://localhost:6006/')
            tb_writer = SummaryWriter(opt.save_dir)  # Tensorboard
        train(hyp, opt, device, tb_writer, wandb)

    # Evolve hyperparameters (optional)
    else:
        # Hyperparameter evolution metadata (mutation scale 0-
1, lower_limit, upper_limit)
        meta = {'lr0': (1, 1e-5, 1e-
1),  # initial learning rate (SGD=1E-2, Adam=1E-3)
                'lrf': (1, 0.01, 1.0),  # final OneCycleLR learning
 rate (lr0 * lrf)
```

```
                'momentum': (0.3, 0.6, 0.98),  # SGD momentum/Adam
beta1
                'weight_decay': (1, 0.0, 0.001),  # optimizer weigh
t decay
                'warmup_epochs': (1, 0.0, 5.0),  # warmup epochs (f
ractions ok)
                'warmup_momentum': (1, 0.0, 0.95),  # warmup initia
l momentum
                'warmup_bias_lr': (1, 0.0, 0.2),  # warmup initial
bias lr
                'box': (1, 0.02, 0.2),  # box loss gain
                'cls': (1, 0.2, 4.0),  # cls loss gain
                'cls_pw': (1, 0.5, 2.0),  # cls BCELoss positive_we
ight
                'obj': (1, 0.2, 4.0),  # obj loss gain (scale with
pixels)
                'obj_pw': (1, 0.5, 2.0),  # obj BCELoss positive_we
ight
                'iou_t': (0, 0.1, 0.7),  # IoU training threshold
                'anchor_t': (1, 2.0, 8.0),  # anchor-
multiple threshold
                'anchors': (2, 2.0, 10.0),  # anchors per output gr
id (0 to ignore)
                'fl_gamma': (0, 0.0, 2.0),  # focal loss gamma (eff
icientDet default gamma=1.5)
                'hsv_h': (1, 0.0, 0.1),  # image HSV-
Hue augmentation (fraction)
                'hsv_s': (1, 0.0, 0.9),  # image HSV-
Saturation augmentation (fraction)
                'hsv_v': (1, 0.0, 0.9),  # image HSV-
Value augmentation (fraction)
                'degrees': (1, 0.0, 45.0),  # image rotation (+/- d
eg)
                'translate': (1, 0.0, 0.9),  # image translation (+
/- fraction)
                'scale': (1, 0.0, 0.9),  # image scale (+/- gain)
                'shear': (1, 0.0, 10.0),  # image shear (+/- deg)
                'perspective': (0, 0.0, 0.001),  # image perspectiv
e (+/- fraction), range 0-0.001
                'flipud': (1, 0.0, 1.0),  # image flip up-
down (probability)
                'fliplr': (0, 0.0, 1.0),  # image flip left-
right (probability)
                'mosaic': (1, 0.0, 1.0),  # image mixup (probabilit
y)
                'mixup': (1, 0.0, 1.0)}  # image mixup (probability
)
```

```
        assert opt.local_rank == -
1, 'DDP mode not implemented for --evolve'
        opt.notest, opt.nosave = True, True  # only test/save final
 epoch
        # ei = [isinstance(x, (int, float)) for x in hyp.values()]
  # evolvable indices
        yaml_file = Path(opt.save_dir) / 'hyp_evolved.yaml'  # save
 best result here
        if opt.bucket:
            os.system('gsutil cp gs://%s/evolve.txt .' % opt.bucket
)  # download evolve.txt if exists

        for _ in range(300):  # generations to evolve
            if Path('evolve.txt').exists():  # if evolve.txt exists
: select best hyps and mutate
                # Select parent(s)
                parent = 'single'  # parent selection method: 'sing
le' or 'weighted'
                x = np.loadtxt('evolve.txt', ndmin=2)
                n = min(5, len(x))  # number of previous results to
 consider
                x = x[np.argsort(-
fitness(x))][:n]  # top n mutations
                w = fitness(x) - fitness(x).min()  # weights
                if parent == 'single' or len(x) == 1:
                    # x = x[random.randint(0, n - 1)]  # random sel
ection
                    x = x[random.choices(range(n), weights=w)[0]]
# weighted selection
                elif parent == 'weighted':
                    x = (x * w.reshape(n, 1)).sum(0) / w.sum()  # w
eighted combination

                # Mutate
                mp, s = 0.8, 0.2  # mutation probability, sigma
                npr = np.random
                npr.seed(int(time.time()))
                g = np.array([x[0] for x in meta.values()])  # gain
s 0-1
                ng = len(meta)
                v = np.ones(ng)
                while all(v == 1):  # mutate until a change occurs
(prevent duplicates)
                    v = (g * (npr.random(ng) < mp) * npr.randn(ng)
* npr.random() * s + 1).clip(0.3, 3.0)
                for i, k in enumerate(hyp.keys()):  # plt.hist(v.ra
vel(), 300)
                    hyp[k] = float(x[i + 7] * v[i])  # mutate
```

```
            # Constrain to limits
            for k, v in meta.items():
                hyp[k] = max(hyp[k], v[1])  # lower limit
                hyp[k] = min(hyp[k], v[2])  # upper limit
                hyp[k] = round(hyp[k], 5)  # significant digits

            # Train mutation
            results = train(hyp.copy(), opt, device, wandb=wandb)

            # Write mutation results
            print_mutation(hyp.copy(), results, yaml_file, opt.buck
et)

        # Plot results
        plot_evolution(yaml_file)
        print(f'Hyperparameter evolution complete. Best results sav
ed as: {yaml_file}\n'
              f'Command to train a new model with these hyperparame
ters: $ python train.py --hyp {yaml_file}')
```

Testing code for YOLO v5

```
import argparse
import json
import os
from pathlib import Path
from threading import Thread

import numpy as np
import torch
import yaml
from tqdm import tqdm

from models.experimental import attempt_load
from utils.datasets import create_dataloader
from utils.general import coco80_to_coco91_class, check_dataset, ch
eck_file, check_img_size, check_requirements, \
    box_iou, non_max_suppression, scale_coords, xyxy2xywh, xywh2xyx
y, set_logging, increment_path, colorstr
from utils.metrics import ap_per_class, ConfusionMatrix
from utils.plots import plot_images, output_to_target, plot_study_t
xt
from utils.torch_utils import select_device, time_synchronized


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,
         log_imgs=0,  # number of logged images
         compute_loss=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 devic
e

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

        # Directories
        save_dir = Path(increment_path(Path(opt.project) / opt.name
, exist_ok=opt.exist_ok))  # increment run
        (save_dir / 'labels' if save_txt else save_dir).mkdir(paren
ts=True, exist_ok=True)  # make dir

        # Load model
        model = attempt_load(weights, map_location=device)  # load
FP32 model
        gs = max(int(model.stride.max()), 32)  # grid size (max str
ide)
        imgsz = check_img_size(imgsz, s=gs)  # check img_size

        # Multi-
GPU disabled, incompatible with .half() https://github.com/ultralyt
ics/yolov5/issues/99
        # if device.type != 'cpu' and torch.cuda.device_count() > 1
:
        #     model = nn.DataParallel(model)

    # Half
```

```
    half = device.type != 'cpu'  # half precision only supported on
 CUDA
    if half:
        model.half()

    # Configure
    model.eval()
    is_coco = data.endswith('coco.yaml')  # is COCO dataset
    with open(data) as f:
        data = yaml.load(f, Loader=yaml.SafeLoader)  # model dict
    check_dataset(data)  # check
    nc = 1 if single_cls else int(data['nc'])  # number of classes
    iouv = torch.linspace(0.5, 0.95, 10).to(device)  # iou vector f
or mAP@0.5:0.95
    niou = iouv.numel()

    # Logging
    log_imgs, wandb = min(log_imgs, 100), None  # ceil
    try:
        import wandb  # Weights & Biases
    except ImportError:
        log_imgs = 0

    # Dataloader
    if not training:
        if device.type != 'cpu':
            model(torch.zeros(1, 3, imgsz, imgsz).to(device).type_a
s(next(model.parameters())))  # run once
        path = data['test'] if opt.task == 'test' else data['val']
  # path to val/test images
        dataloader = create_dataloader(path, imgsz, batch_size, gs,
 opt, pad=0.5, rect=True,
                                       prefix=colorstr('test: ' if
opt.task == 'test' else 'val: '))[0]

    seen = 0
    confusion_matrix = ConfusionMatrix(nc=nc)
    names = {k: v for k, v in enumerate(model.names if hasattr(mode
l, 'names') else model.module.names)}
    coco91class = coco80_to_coco91_class()
    s = ('%20s' + '%12s' * 6) % ('Class', 'Images', 'Targets', 'P',
 'R', 'mAP@.5', 'mAP@.5:.95')
    p, r, f1, mp, mr, map50, map, t0, t1 = 0., 0., 0., 0., 0., 0.,
0., 0., 0.
    loss = torch.zeros(3, device=device)
    jdict, stats, ap, ap_class, wandb_images = [], [], [], [], []
    for batch_i, (img, targets, paths, shapes) in enumerate(tqdm(da
taloader, desc=s)):
        img = img.to(device, non_blocking=True)
```

```
        img = img.half() if half else img.float()  # uint8 to fp16/
32
        img /= 255.0  # 0 - 255 to 0.0 - 1.0
        targets = targets.to(device)
        nb, _, height, width = img.shape  # batch size, channels, h
eight, width

        with torch.no_grad():
            # Run model
            t = time_synchronized()
            out, train_out = model(img, augment=augment)  # inferen
ce and training outputs
            t0 += time_synchronized() - t

            # Compute loss
            if compute_loss:
                loss += compute_loss([x.float() for x in train_out]
, targets)[1][:3]  # box, obj, cls

            # Run NMS
            targets[:, 2:] *= torch.Tensor([width, height, width, h
eight]).to(device)  # to pixels
            lb = [targets[targets[:, 0] == i, 1:] for i in range(nb
)] if save_hybrid else []  # for autolabelling
            t = time_synchronized()
            out = non_max_suppression(out, conf_thres=conf_thres, i
ou_thres=iou_thres, labels=lb, multi_label=True)
            t1 += time_synchronized() - t

        # Statistics per image
        for si, pred in enumerate(out):
            labels = targets[targets[:, 0] == si, 1:]
            nl = len(labels)
            tcls = labels[:, 0].tolist() if nl else []  # target cl
ass
            path = Path(paths[si])
            seen += 1

            if len(pred) == 0:
                if nl:
                    stats.append((torch.zeros(0, niou, dtype=torch.
bool), torch.Tensor(), torch.Tensor(), tcls))
                continue

            # Predictions
            predn = pred.clone()
            scale_coords(img[si].shape[1:], predn[:, :4], shapes[si
][0], shapes[si][1])  # native-space pred
```

```
            # Append to text file
            if save_txt:
                gn = torch.tensor(shapes[si][0])[[1, 0, 1, 0]]  # n
ormalization gain whwh
                for *xyxy, conf, cls in predn.tolist():
                    xywh = (xyxy2xywh(torch.tensor(xyxy).view(1, 4)
) / gn).view(-1).tolist()  # normalized xywh
                    line = (cls, *xywh, conf) if save_conf else (cl
s, *xywh)  # label format
                    with open(save_dir / 'labels' / (path.stem + '.
txt'), 'a') as f:
                        f.write(('%g ' * len(line)).rstrip() % line
 + '\n')

            # W&B logging
            if plots and len(wandb_images) < log_imgs:
                box_data = [{"position": {"minX": xyxy[0], "minY":
xyxy[1], "maxX": xyxy[2], "maxY": xyxy[3]},
                             "class_id": int(cls),
                             "box_caption": "%s %.3f" % (names[cls]
, conf),
                             "scores": {"class_score": conf},
                             "domain": "pixel"} for *xyxy, conf, cl
s in pred.tolist()]
                boxes = {"predictions": {"box_data": box_data, "cla
ss_labels": names}}  # inference-space
                wandb_images.append(wandb.Image(img[si], boxes=boxe
s, caption=path.name))

            # Append to pycocotools JSON dictionary
            if save_json:
                # [{"image_id": 42, "category_id": 18, "bbox": [258
.15, 41.29, 348.26, 243.78], "score": 0.236}, ...
                image_id = int(path.stem) if path.stem.isnumeric()
else path.stem
                box = xyxy2xywh(predn[:, :4])  # xywh
                box[:, :2] -= box[:, 2:] / 2  # xy center to top-
left corner
                for p, b in zip(pred.tolist(), box.tolist()):
                    jdict.append({'image_id': image_id,
                                  'category_id': coco91class[int(p[
5])] if is_coco else int(p[5]),
                                  'bbox': [round(x, 3) for x in b],
                                  'score': round(p[4], 5)})

            # Assign all predictions as incorrect
            correct = torch.zeros(pred.shape[0], niou, dtype=torch.
bool, device=device)
            if nl:
```

```
                detected = []  # target indices
                tcls_tensor = labels[:, 0]

                # target boxes
                tbox = xywh2xyxy(labels[:, 1:5])
                scale_coords(img[si].shape[1:], tbox, shapes[si][0]
, shapes[si][1])  # native-space labels
                if plots:
                    confusion_matrix.process_batch(predn, torch.cat
((labels[:, 0:1], tbox), 1))

                # Per target class
                for cls in torch.unique(tcls_tensor):
                    ti = (cls == tcls_tensor).nonzero(as_tuple=Fals
e).view(-1)  # prediction indices
                    pi = (cls == pred[:, 5]).nonzero(as_tuple=False
).view(-1)  # target indices

                    # Search for detections
                    if pi.shape[0]:
                        # Prediction to target ious
                        ious, i = box_iou(predn[pi, :4], tbox[ti]).
max(1)  # best ious, indices

                        # Append detections
                        detected_set = set()
                        for j in (ious > iouv[0]).nonzero(as_tuple=
False):
                            d = ti[i[j]]  # detected target
                            if d.item() not in detected_set:
                                detected_set.add(d.item())
                                detected.append(d)
                                correct[pi[j]] = ious[j] > iouv  #
iou_thres is 1xn
                                if len(detected) == nl:  # all targ
ets already located in image
                                    break

            # Append statistics (correct, conf, pcls, tcls)
            stats.append((correct.cpu(), pred[:, 4].cpu(), pred[:,
5].cpu(), tcls))

        # Plot images
        if plots and batch_i < 3:
            f = save_dir / f'test_batch{batch_i}_labels.jpg'  # lab
els
            Thread(target=plot_images, args=(img, targets, paths, f
, names), daemon=True).start()
```

```
            f = save_dir / f'test_batch{batch_i}_pred.jpg'  # predi
ctions
            Thread(target=plot_images, args=(img, output_to_target(
out), paths, f, names), daemon=True).start()

    # Compute statistics
    stats = [np.concatenate(x, 0) for x in zip(*stats)]  # to numpy
    if len(stats) and stats[0].any():
        p, r, ap, f1, ap_class = ap_per_class(*stats, plot=plots, s
ave_dir=save_dir, names=names)
        ap50, ap = ap[:, 0], ap.mean(1)  # AP@0.5, AP@0.5:0.95
        mp, mr, map50, map = p.mean(), r.mean(), ap50.mean(), ap.me
an()
        nt = np.bincount(stats[3].astype(np.int64), minlength=nc)
# number of targets per class
    else:
        nt = torch.zeros(1)

    # Print results
    pf = '%20s' + '%12.3g' * 6  # print format
    print(pf % ('all', seen, nt.sum(), mp, mr, map50, map))

    # Print results per class
    if (verbose or (nc < 50 and not training)) and nc > 1 and len(s
tats):
        for i, c in enumerate(ap_class):
            print(pf % (names[c], seen, nt[c], p[i], r[i], ap50[i],
 ap[i]))

    # Print speeds
    t = tuple(x / seen * 1E3 for x in (t0, t1, t0 + t1)) + (imgsz,
imgsz, batch_size)  # tuple
    if not training:
        print('Speed: %.1f/%.1f/%.1f ms inference/NMS/total per %gx
%g image at batch-size %g' % t)

    # Plots
    if plots:
        confusion_matrix.plot(save_dir=save_dir, names=list(names.v
alues()))
        if wandb and wandb.run:
            val_batches = [wandb.Image(str(f), caption=f.name) for
f in sorted(save_dir.glob('test*.jpg'))]
            wandb.log({"Images": wandb_images, "Validation": val_ba
tches}, commit=False)

    # Save JSON
    if save_json and len(jdict):
```

```
        w = Path(weights[0] if isinstance(weights, list) else weigh
ts).stem if weights is not None else ''  # weights
        anno_json = '../coco/annotations/instances_val2017.json'  #
 annotations json
        pred_json = str(save_dir / f"{w}_predictions.json")  # pred
ictions json
        print('\nEvaluating pycocotools mAP... saving %s...' % pred
_json)
        with open(pred_json, 'w') as f:
            json.dump(jdict, f)

        try:  # https://github.com/cocodataset/cocoapi/blob/master/
PythonAPI/pycocoEvalDemo.ipynb
            from pycocotools.coco import COCO
            from pycocotools.cocoeval import COCOeval

            anno = COCO(anno_json)  # init annotations api
            pred = anno.loadRes(pred_json)  # init predictions api
            eval = COCOeval(anno, pred, 'bbox')
            if is_coco:
                eval.params.imgIds = [int(Path(x).stem) for x in da
taloader.dataset.img_files]  # image IDs to evaluate
            eval.evaluate()
            eval.accumulate()
            eval.summarize()
            map, map50 = eval.stats[:2]  # update results (mAP@0.5:
0.95, mAP@0.5)
        except Exception as e:
            print(f'pycocotools unable to run: {e}')

    # Return results
    model.float()  # for training
    if not training:
        s = f"\n{len(list(save_dir.glob('labels/*.txt')))} labels s
aved to {save_dir / 'labels'}" if save_txt else ''
        print(f"Results saved to {save_dir}{s}")
    maps = np.zeros(nc) + map
    for i, c in enumerate(ap_class):
        maps[c] = ap[i]
    return (mp, mr, map50, map, *(loss.cpu() / len(dataloader)).tol
ist()), maps, t


if __name__ == '__main__':
    parser = argparse.ArgumentParser(prog='test.py')
    parser.add_argument('--
weights', nargs='+', type=str, default='yolov5s.pt', help='model.pt
 path(s)')
```

```
    parser.add_argument('--
data', type=str, default='data/coco128.yaml', help='*.data path')
    parser.add_argument('--batch-
size', type=int, default=32, help='size of each image batch')
    parser.add_argument('--img-
size', type=int, default=640, help='inference size (pixels)')
    parser.add_argument('--conf-
thres', type=float, default=0.001, help='object confidence threshol
d')
    parser.add_argument('--iou-
thres', type=float, default=0.6, help='IOU threshold for NMS')
    parser.add_argument('--
task', default='val', help="'val', 'test', 'study'")
    parser.add_argument('--
device', default='', help='cuda device, i.e. 0 or 0,1,2,3 or cpu')
    parser.add_argument('--single-
cls', action='store_true', help='treat as single-class dataset')
    parser.add_argument('--
augment', action='store_true', help='augmented inference')
    parser.add_argument('--
verbose', action='store_true', help='report mAP by class')
    parser.add_argument('--save-
txt', action='store_true', help='save results to *.txt')
    parser.add_argument('--save-
hybrid', action='store_true', help='save label+prediction hybrid re
sults to *.txt')
    parser.add_argument('--save-
conf', action='store_true', help='save confidences in --save-
txt labels')
    parser.add_argument('--save-
json', action='store_true', help='save a cocoapi-
compatible JSON results file')
    parser.add_argument('--
project', default='runs/test', help='save to project/name')
    parser.add_argument('--
name', default='exp', help='save to project/name')
    parser.add_argument('--exist-
ok', action='store_true', help='existing project/name ok, do not in
crement')
    opt = parser.parse_args()
    opt.save_json |= opt.data.endswith('coco.yaml')
    opt.data = check_file(opt.data)  # check file
    print(opt)
    check_requirements()

    if opt.task in ['val', 'test']:  # run normally
        test(opt.data,
             opt.weights,
             opt.batch_size,
```

```
             opt.img_size,
             opt.conf_thres,
             opt.iou_thres,
             opt.save_json,
             opt.single_cls,
             opt.augment,
             opt.verbose,
             save_txt=opt.save_txt | opt.save_hybrid,
             save_hybrid=opt.save_hybrid,
             save_conf=opt.save_conf,
             )

    elif opt.task == 'speed':  # speed benchmarks
        for w in opt.weights:
            test(opt.data, w, opt.batch_size, opt.img_size, 0.25, 0
.45, save_json=False, plots=False)

    elif opt.task == 'study':  # run over a range of settings and s
ave/plot
        # python test.py --task study --data coco.yaml --iou 0.7 --
weights yolov5s.pt yolov5m.pt yolov5l.pt yolov5x.pt
        x = list(range(256, 1536 + 128, 128))  # x axis (image size
s)
        for w in opt.weights:
            f = f'study_{Path(opt.data).stem}_{Path(w).stem}.txt'
# filename to save to
            y = []  # y axis
            for i in x:  # img-size
                print(f'\nRunning {f} point {i}...')
                r, _, t = test(opt.data, w, opt.batch_size, i, opt.
conf_thres, opt.iou_thres, opt.save_json,
                               plots=False)
                y.append(r + t)  # results and times
            np.savetxt(f, y, fmt='%10.4g')  # save
        os.system('zip -r study.zip study_*.txt')
        plot_study_txt(x=x)  # plot
```