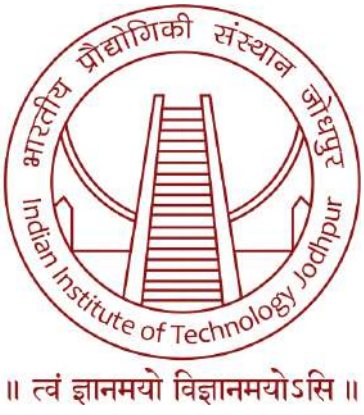
Autonomous Image-Based Weed Detection System for Smart Agriculture

A Project Report Submitted by

Mahendra G L

in partial fulfillment of the requirements for the award of the degree of

M.Tech in Robotics and Mobility Systems



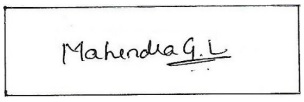
Indian Institute of Technology Jodhpur

IDRP-Robotics and Mobility Systems

May, 2024

Declaration

I hereby declare that the work presented in this Project Report titled Autonomous Image-Based Weed Detection System for Smart Agriculture submitted to the Indian Institute of Technology Jodhpur in partial fulfilment of the requirements for the award of the degree of M.Tech in Robotics and Mobility Systems, is a bonafide record of the research work carried out under the supervision of Dr.Arpit Arvind Khandelwal. The contents of this Project Report in full or in parts, have not been submitted to, and will not be submitted by me to, any other Institute or University in India or abroad for the award of any degree or diploma.



Signature

Mahendra G L

M22RM004

Certificate

This is to certify that the Project Report titled Autonomous Image-Based Weed Detection System for Smart Agriculture, submitted by Mahendra G L(M22RM004) to the Indian Institute of Technology Jodhpur for the award of the degree of M.Tech in Robotics and Mobility Systems, is a bonafide record of the research work done by him under my supervision. To the best of my knowledge, the contents of this report, in full or in parts, have not been submitted to any other Institute or University for the award of any degree or diploma.

Signature

Dr.Arpit Arvind Khandelwal

Acknowledgements

I would like to take this opportunity to thank everyone who was involved in helping me throughout this project. Foremost I would like to thank my supervisor **Dr.Arpit Arvind Khandelwal**, Assistant Professor, Department of Electrical Engineering for his unwavering support and guidance. I would also like to thank Mr.Gajraj Sharma, Junior Technical Superintendent, Lab-103, Department of Electrical Engineering for his valuable assistance. Special thabnks to my classmate Mr.Rajesh for his assistance in collecting required datasets. I am sincerely appreciative of the resources and facilities provided by the Electrical Engineering department, without which this project would not have been possible. Finally, I want to express my deepest gratitude to my family and friends for their constant encouragement and understanding throughout this journey. Their unwavering support has been a source of strength and motivation. Thank you all for being an integral part of this project.

Abstract

In India, where agriculture comprises nearly 20% of GDP, effective weed control is vital for maximizing crop yields. Smart agriculture systems, leveraging computer vision and robotics, offer localized weed detection and removal solutions. Deep neural networks have revolutionized weed vs. crop classification, but their computational demands often necessitate powerful GPUs. To address this, our project explores the popular YOLO object detection models (versions 5 to 8) to determine the optimal variant for low-power edge deployment. We aim to demonstrate the model's real-time detection capabilities, addressing the need for accessible weed management solutions.

To balance model performance and cost-effectiveness, we employ a lightweight YOLO model (e.g., YOLOvX-tiny, YOLO-Nano, etc.). These models are well-suited for weed detection while being compact enough for resource-constrained devices. We train the selected model with a custom annotated image dataset containing 'weed' and 'crop' classes. Our goal is to deploy the trained model onto a low-cost single-board computer like a Raspberry Pi for real-time weed detection, contributing to effective and sustainable weed management practices.

Keywords: Weed Detection; Edge IoT decvices; YOLO; Convoloutional Neural Networks; Computer vision; Robotics

Contents

List of Figures

List of Tables

Autonomous Image-Based Weed Detection System for Smart Agriculture

1 Introduction and background

Human civilization and settlement were made possible by the discovery of agriculture at least 12,000 years ago. According to the Net-Geo website, agriculture is "the art and science of cultivating the soil, growing crops, and raising livestock." Agriculture faces many issues, including climate change, the availability of arable land, etc. But, weeds are the biggest problem in any agriculture practice. Weeds consume nutrients, land, and water that are meant for crops, and they can multiply rapidly. This kind of competition on land will reduce the output. According to [1], there are at least 1800 weed species, which has caused 31.5% reduction in crop output, resulting in a loss of $32 billion per year.

There are many ways of weed management. Hand-picking weeds is manual labor and expensive; chemical herbicides are the best alternative. However, the herbicides are chemicals that can cause soil degradation and water pollution, and since it is sprayed even on crops, the yield may be poisoned. Thus, there is a need to identify specific weeds on a land full of crops automatically. Advancements in deep learning and computer vision have opened up number of opportunities for this particular problem. Although lightweight, accurate versions of large models can be deployed on an edge device, and real-time analysis is possible. This is also economical as we can automate the process from detection to neutralizing the weed with precision herbicide spraying over selected areas.

Computer Vision, abbreviated as CV, is a powerful tool that helps computers to see as we see the world. Low-cost, sustainable solutions can be built upon the concepts of CV. In a review paper [2], the authors show several image processing techniques, such as image segmentation and classification using CNN (Convolution Neural Networks). Even SVMs, Masked R-CNN models, and YOLO (You Only Look Once)[3] models are tried and tested for the purpose of weed detection. The usual pipeline for any of these models is

- Collecting the dataset.
- Data pre-processing.
- Annotating the images.
- Training and testing.
- Deploy on the edge device.

In this project, we embark on a thorough exploration of the fascinating realm of YOLO models, spanning from version 5 to the latest iteration, YOLOv8. Our endeavor involves conducting a comprehensive survey, meticulously dissecting each iteration for its advancements and innovations. With each subsequent version, we unveil enhancements that propel the capabilities of YOLO forward. Notably, the pinnacle of this evolution, YOLOv8, stands out with its remarkable attributes. It boasts faster training

and inference speeds, empowers real-time detection capabilities, and achieves higher overall performance metrics. This report serves as a testament to our dedication in unraveling the intricacies of YOLO models and shedding light on the unparalleled advancements they bring to the field of computer vision.

The designated edge device for this project is the Raspberry Pi, a compact single-board computer (SBC) widely employed in IoT and robotics applications. Raspberry Pi finds utility across a spectrum of projects, ranging from hobbyist endeavors to professional developments. Known for its cost-effectiveness, this small yet powerful device is particularly suitable for running image processing models. The Raspberry Pi's efficient CPU makes it a viable choice for handling the computational demands associated with image processing tasks. Additionally, the hardware specifications of the Raspberry Pi 4 Model B, which will serve as the edge device, are detailed in Table 1.1.

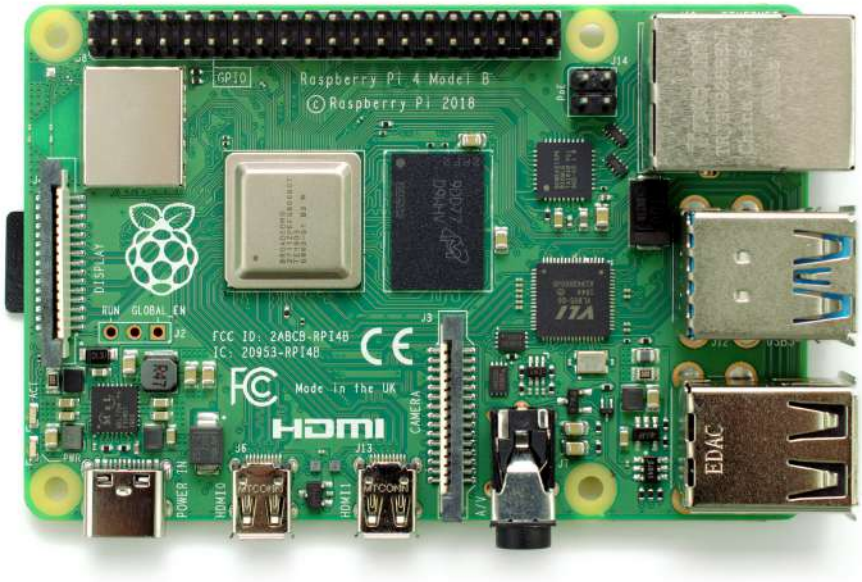


Figure 1.1: Raspberry Pi 4 Model B

Specification	**Value**
Processor	Quad-core ARM Cortex-A72
CPU Clock Speed	1.5 GHz
RAM	2GB, 4GB, or 8GB LPDDR4 SDRAM
GPU	Video Core VI
Video Outputs	2 × micro-HDMI (up to 4K@60Hz)
USB Ports	2 × USB 3.0, 2 × USB 2.0
Ethernet	Gigabit Ethernet
Wireless	2.4 GHz and 5 GHz IEEE 802.11ac wireless, Bluetooth 5.0
Storage	microSD card slot
GPIO Pins	40 GPIO pins
Power Supply	5V/3A via USB-C
Operating System	Raspberry Pi OS (formerly Raspbian), various

Table 1.1: Raspberry Pi 4 Model B Specifications.

2 Literature survey

In recent years, the development of CV and convolution neural networks has exploded; thus, there has been useful work in precision agriculture.

In [7], authors have proposed a YOLOv5 model for Weed vs crop classification and the detection of diseased leaves. YOLOv5 is a single-shot object detector with three main parts: Model backbone, Model Neck, and Model head, each responsible for different tasks. The data set consists of 140 RGB images for training and 60 images for testing; additionally, 50 more images were used to validate the model's accuracy. Images were taken in the field of Egyptian star cluster flowers. Since the classification problems are evaluated on a parameter known as mean average precision, the authors report that the mAP for the training is 0.95 on training up to 50 epochs. Three classes were to be detected: Bacteria infected leaf, Crop & Weed. An accuracy of 95% was achieved using the YOLOv5 model.

In [8], authors have developed a novel benchmark of state-of-the-art YOLO object detectors for weed detection in cotton production systems called DeepCottonWeeds(DCW). For the US, cotton is an important crop valued at $7.5 billion as of 2022. An image dataset of 5648 images with 12 different classes of weeds was collected from various fields across the country, and they trained multiple YOLO variations, 18 to be precise, which include YOLOv3, YOLOv4, Scaled-YOLOv4, and YOLOv5. Authors report that the YOLOv5 has higher accuracy and better real-time detection results. The mAP@0.5 was ranging from 88.14% in YOLOv3Tiny to 95.22% in YOLOv4.

In [9], authors deployed four large and four small models on UAVs (Unmanned Aerial vehicles) with edge systems for aerial image analysis. It is a low-cost but effective solution for weed detection over large areas. The small models consisted of MobileNet, EfficinetNetv2, YOLOv4-lite, and DarkNet-Ref. The MobileNet and YOLOv4-lite showed impressive performance with mAP scores of 83.2% and 82.2%, respectively. Furthermore, the authors state that the dataset was collected using a DJI Phantom 4 Pro drone equipped with an RGB camera. The drone was flown at an average altitude of 4.5m from the ground.Total of 877 training images and 219 validation images are taken. Also, videos with 1080p .mp4 format were also captured for real-time weed identification. The authors conclude that YOLOv4-lite is one of the best lightweight models that can work efficiently on an edge platform.

In [10], three different methods for weed detection were employed and tested for their accuracy of detection, namely:

- Support Vector Machine (SVM) with a Histogram of Oriented Gradients (HOG) feature descriptor.
- YOLOv3.
- Masked Region-based Convolutional Neural Network (R-CNN).

The authors underscore the importance of utilizing edge devices for image processing. While satellite images offer a broader field of view, they are limited in resolution and prove inadequate for tasks like weed detection. In contrast, Unmanned Aerial Vehicles (UAVs) present a more viable solution, as they can be equipped with high-resolution cameras and controlled remotely. The image dataset was generated using a

Mavic Pro drone with a Parrot Sequoia multispectral camera, capturing 100 images (1280*960 pixels) over a 600 square meter area of commercial lettuce cultivation during a 15-hour automated flight. Applying the Bland-Altman method, the authors demonstrate that HOG-SVM and R-CNN exhibit comparable and consistent performance. They suggest that HOG-SVM might be the optimal Internet of Things (IoT) solution due to its lower computational requirements for model training and testing. On the other hand, the YOLO method tends to overestimate high weed values compared to the other two methods.

[11][12] A survey conducted by the All India Coordinated Research Project on Weed Management between 2003 and 14 in major field crops in different districts of 18 states of India revealed that A staggering economic loss of approximately $11 billion was calculated as the direct consequence of weed infestation exclusively in the top 10 major crops of India. These crops include groundnut (35.8% loss), soybean (31.4% loss), green gram (30.8% loss), pearl millet (27.6% loss), maize (25.3% loss), sorghum (25.1% loss), sesame (23.7% loss), mustard (21.4% loss), direct-seeded rice (21.4% loss), wheat (18.6% loss), and transplanted rice (13.8% loss).

3 Problem definition and Objective

India is an agrarian economy. More than half of labor force and about 20% of the total GDP is from the agriculture sector. With a population of about 1.4 billion people, we need higher yields per acre to keep up with the demand. In the agricultural landscape of India, the coexistence of crops and weeds poses a significant challenge for farmers. Traditional methods of weed management are often labor-intensive and time-consuming, making it imperative to develop a cutting-edge machine learning model capable of weed vs. crop detection directly on edge devices.

The problem at hand is to develop an intelligent and cost effective system for accurate weed vs crop detection on Indian landscape. The underlying machine learning model must be lightweight enough to be deployed on to an edge device without overwhelming the computing power of the device and classify the crop and weed autonomously without human intervention. The system should be capable of following things:

- Classification: Distinguish between commonly found weed species and crop accurately.
- Edge Deployment: Designing and implementing a machine learning model that can be deployed on edge computing devices commonly available to farmers in India. For this particular project Raspberry Pi is utilized as edge computing device.
- Quantification : Provide insights into the parts of land where weed infestation is higher and ask to take appropriate measures.
- Real-time monitoring: Enabling real-time or near-real-time monitoring of weed-crop dynamics, facilitating timely intervention and decision-making. This is one of the major advantage of deploying the model on edge devices.
- Cost-Effectiveness: Designing the system to be cost-effective and accessible to smallholder farmers, considering the economic constraints prevalent in many farming communities.

4 Methodology

Many existing models face challenges when deployed on edge nodes due to being outdated and inefficient. Models utilizing SVMs with HOG descriptors, while efficient, fall short of achieving real-time detection. We propose the use of the latest state-of-the-art YOLO detector. With a total of 9 iterations, ranging from version 1 to version 9, we propose a meticulous examination to identify the optimal model suitable for deployment on the Raspberry Pi Single Board Computer (SBC). Our primary criterion for selection is the ability to achieve real-time detection while maintaining respectable Frames per Second (FPS). To ensure compatibility with the CPU environment and minimize resource consumption, we specifically focus on the lightweight versions of these models.

By scrutinizing YOLO versions 5 through 8, our aim is to pinpoint the model that strikes the ideal balance between performance and efficiency, ultimately enabling seamless deployment on edge devices like the Raspberry Pi SBC. This endeavor embodies our commitment to leveraging state-of-the-art technology to address real-world challenges in computer vision applications.

4.1 Dataset

4.1.1 Dataset Collection

The first and foremost step in any classification problem is to collect a dataset for training the model. Since we are working on a agricultural data we decided to collect the dataset our-self. We visited two local farmlands; one which grow fennel seeds and other which grows a seasonal desert crop called psyllium or locally known as jeera/isabgol a medicinal plant.

The images were taken with the help of smartphone camera and all images were taken handheld, no special stabilizing tools were used. Over the course of 10 days we were able to collect more than 1300 images from both farmlands which consisted of multiple crops and weeds. The idea is to train the model with this mixed data so that model learns more complex patterns, which will lead to better inference performance of the model. Although there can be several issues with this approach whcih can lead to false detections from the model, such as:

- The crop psyllium's leaves have similar appearance as some of the weeds that was found at the other farmland.
- There are few weed species which grow in tandem with psyllium crop which looks slimier to the crop.

4.1.2 Data Prepossessing

The raw images that were collected need to be preprocessed before we can start training the model. The raw images taken from smartphone camera were too big in both size and memory. Each image on an average took upto 10 mega bytes of memory and close to 4000*4000 pixels large. These images should be reduced both in size and memory.

1. The first step is to reduce the size and memory of these images. For this we used OpenCV library to dial down the pixel size to 1500*1500 without reducing image quality. Each image now took only about 1 mega bytes of memory or less. This also ensures uniform image size.
2. Manually comb through the images for imperfections such as too blurry image, too bright or dark, repeated images, could be difficult to annotate and too little information to work on.
3. While going through images, randomly choose some images and augment the images. For our case we used simple augmentations like rotate by 90°, rotate by 180° and cropping few images, which was done manually. The figure 4.1 shows few samples from the dataset after preprocessing is done.

Figure 4.1: Samples from the original dataset

4.1.3 Data Annotations

Data annotation for object detection involves the process of labeling images or videos to mark the presence and location of objects within them. It's a crucial step in training machine learning models to recognize and locate objects accurately. Annotations typically include bounding boxes that encompass the objects of interest, along with labels specifying the object category. This annotated data serves as the training set for object detection algorithms, enabling them to learn the visual patterns associated with different object classes and their spatial arrangements within images or frames of videos.

After preprocessing, there were 1102 images left in the dataset. We decided to annotate 1000 images to serve as our main dataset. The dataset were annotated with an online open-source tool called *makesense.ai* [19]. The tool allows us to annotate any number of images on the local browser (see figure 4.2). Images were annotated with two classes

1. Crop
2. Weed

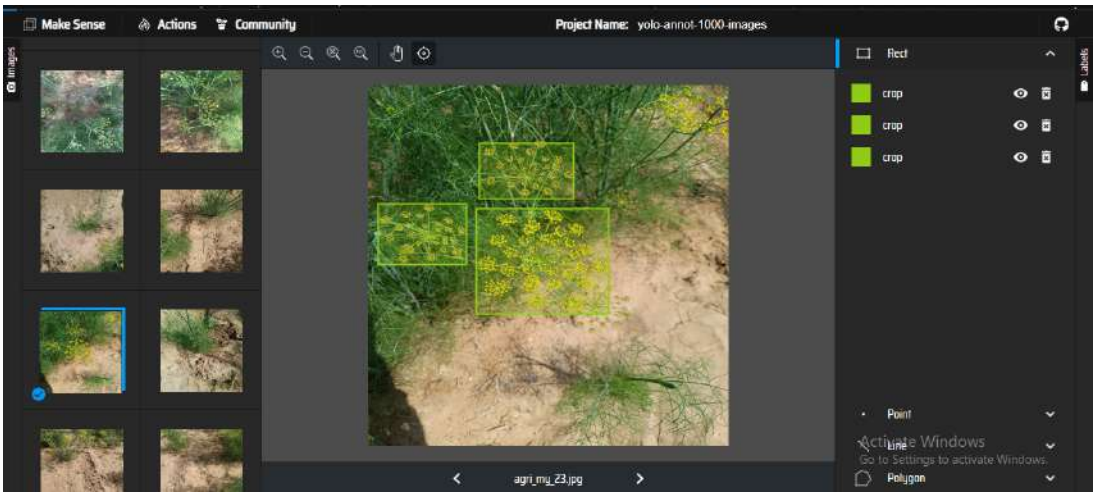


Figure 4.2: Image annotation process using makesense.ai tool

The bounding box annotations were exported in the YOLO format. This format looks like this: `<class> <x_center> <y_center> <width> <height>` where:

- **class**: The class label or category of the object.
- **x_center**: The normalized x-coordinate of the center of the bounding box relative to the width of the image.
- **y_center**: The normalized y-coordinate of the center of the bounding box relative to the height of the image.
- **width**: The normalized width of the bounding box relative to the width of the image.
- **height**: The normalized height of the bounding box relative to the height of the image.

This format is used across all YOLO versions for training purposes.

4.1.4 Data Formatting

All YOLO versions that are used needs a structured directory. This is to ensure that data can be efficiently consumed by the YOLO algorithm during training. 4.4 shows pictorial representation of how the data folder structure needs to be for YOLO model training.

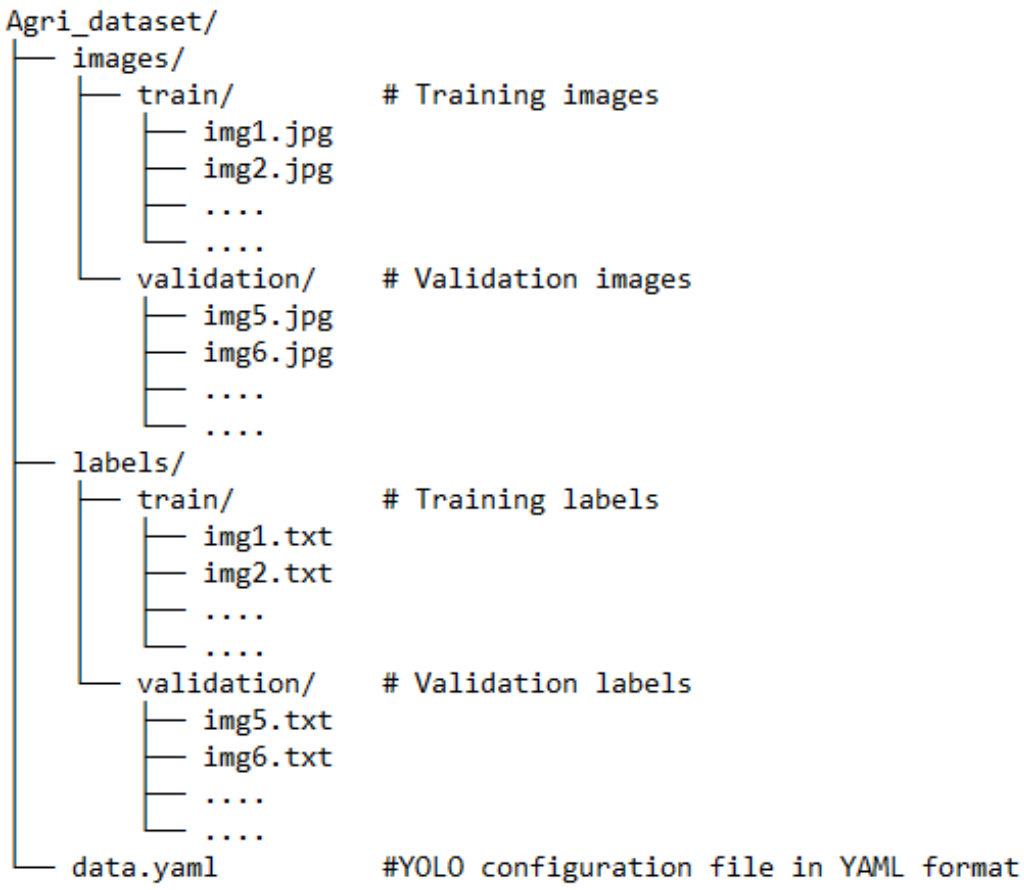


Figure 4.3: Folder Structure for YOLO models

The .yaml file is configuration file for YOLO model. This file consist of number classes, directory paths for training and validation and name of classes. YOLO models typically require datasets organized in a specific format, where each image is accompanied by annotations specifying bounding boxes and class labels for objects within the images. This format ensures that the model can properly interpret and learn from the dataset during training. Deviating from this structure can lead to training errors or suboptimal performance. Therefore, maintaining the correct dataset structure is essential for successful training and accurate object detection with YOLO models.

4.2 YOLO detection model.

YOLO (You Only Look Once) [13] is an open-source, state-of-the-art model that can be used for various purposes. Popularly, they are used for detection, classification, and segmentation. The latest installment in the family of YOLO models is YOLOv8, and it was developed by a company called Ultralytics [4]. They are also the creators of YOLOv5, another version of the YOLO model. Among all models in the family, YOLOv8 performs best on any dataset, be it image or numerical data, as per the creators of YOLOv8.

All YOLO models have similar basic structure as shown in figure 4.4. The difference in each model comes with what are the underlying convolution layers that are being used for feature extraction. The

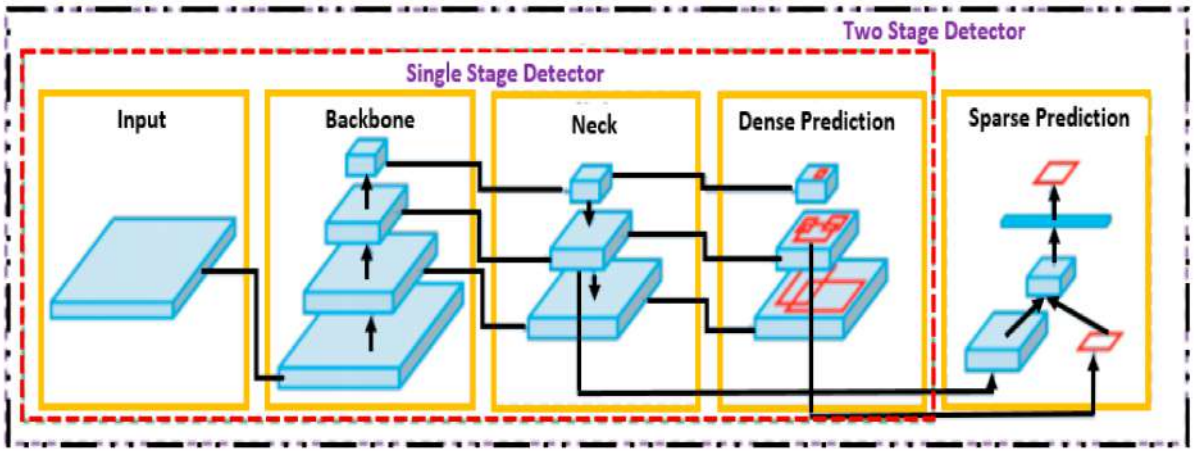


Figure 4.4: General YOLO structure (source: refer to [21])

general bare bone structure of any YOLO model can be described as follows:

1. **Backbone**: A convolutional neural network that gathers and produces visual features of different shapes and sizes. It often utilizes classification models like ResNet, VGG, and EfficientNet to extract features.

2. **Neck**: A group of layers that blend and combine features before passing them to the prediction layer. Examples include Feature Pyramid Network (FPN), Path Aggregation Network (PAN), and Bi-FPN.

3. **Head**: Receives features from the neck along with bounding box predictions. It performs classification and regression tasks on these features and bounding box coordinates to complete the detection process. Typically, it outputs four values representing x and y coordinates, width, and height.

4.3 Model Evaluation

Underlying structure of any YOLO model is based upon working of CNN model. Thus for model evaluation and score the detection, we will use mean average precision (mAP), precision and recall as our performance evaluation matrices.

4.3.1 Precision

Precision is a metric used in the evaluation of classification models, particularly in tasks where the goal is to minimize false positives. Precision is calculated as the ratio of true positives (correctly predicted positive instances) to the sum of true positives and false positives (instances predicted as positive but are actually negative). In mathematical terms:

$$Precision = \frac{TP}{TP + FP} \tag{1}$$

Precision provides an indication of how well a model performs when it predicts a positive class. A higher precision score indicates that the model has a lower rate of false positives, meaning that when it predicts a positive outcome, it is more likely to be correct. In the context of weed detection versus crops, precision refers to the accuracy of a machine learning model in correctly identifying and classifying weeds without incorrectly categorizing crops as weeds. Precision in this scenario is crucial because misclassifying crops as weeds could lead to unnecessary and potentially harmful interventions, such as applying herbicides to areas where crops are growing.

4.3.2 Recall

Recall, also known as sensitivity or true positive rate, is a metric used in the evaluation of classification models. It measures the ability of a model to correctly identify all relevant instances, specifically the ratio of true positives (correctly predicted positive instances) to the sum of true positives and false negatives (instances that were positive but not predicted as such). In mathematical terms:

$$Recall = \frac{TP}{TP + FN} \tag{2}$$

In the context of weed detection versus crops in agriculture, recall would assess how well a model captures and correctly identifies all instances of weeds, ensuring that as few actual weeds as possible are missed. A higher recall value indicates that the model is effective at identifying the majority of weeds in the field. Recall is particularly important in scenarios where missing positive instances (e.g., failing to detect weeds) can have significant consequences. In agriculture, a high recall ensures that the weed detection system is sensitive to the presence of weeds, minimizing the risk of leaving weed-infested areas untreated, which could negatively impact crop yield.

4.3.3 Mean Average Precision (mAP)

CNN models like YOLO are predominantly assessed based on their Mean Average Precision (mAP) values. As YOLO performs object detection by predicting bounding boxes, the evaluation involves using Intersection over Union (IoU) to calculate precision and recall.

IoU is a measure that gauges the extent of overlap between the predicted bounding box and the ground truth bounding box. A specific threshold for IoU is set. If the IoU is less than the defined threshold, the prediction is classified as false positive. Conversely, if the IoU exceeds the threshold, the prediction is considered a true positive. The figure 4.5 shows how the IoU can be calculated for an annotated image.

In simpler terms, this threshold on IoU serves as a criterion for determining the accuracy of bounding box predictions. If the overlap between the predicted and actual bounding boxes is sufficiently high, the prediction is considered accurate (true positive); otherwise, it is treated as a false positive. This IoU-based evaluation helps quantify the precision and recall of object detection models like YOLO.

The metric known as Average Precision (AP) corresponds to the area under the precision-recall curve. In our scenario, which involves the classification of two classes, weed and crop, AP is computed

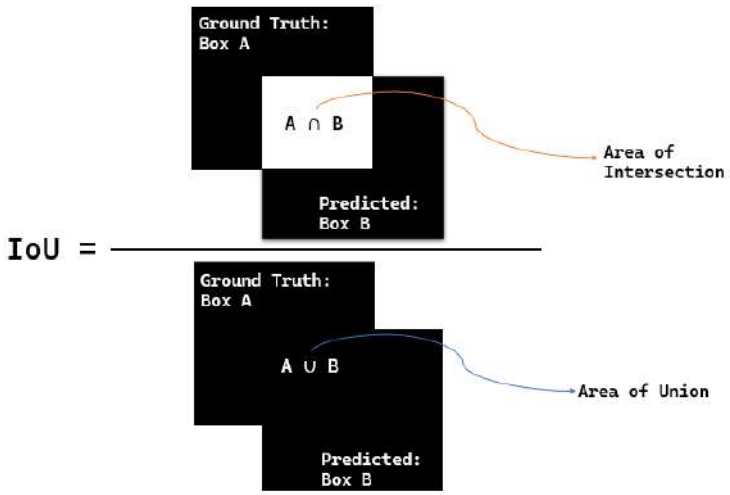


Figure 4.5: Intersection Over Union Visualized

independently for each class, resulting in distinct AP values. The Mean Average Precision (mAP) is simply the average of these AP values, providing an overall assessment of the model's performance across both classes.

For all of our YOLO models IoU threshold of 0.5 is used and mAP@0.5 is considered as primary metric for model evaluation.

5 Results

The training is carried out on 4 different YOLO models as mentioned in section 4. This way we can define the best model in terms of computation, inference time and ease of use and deploy that model on to Raspberry Pi.

The dataset is divided into 3 parts: training images, validation images and testing images.

5.1 Training

The YOLO versions from version 5 to version 8 are trained under following conditions:

No.of Epochs	Batch Size	Image Size	Training Machine	Pre-trained Weights
100	32	512*512	T4-GPU	No

Table 5.1: Training Conditions.

Training all YOLO iterations under standardized conditions is crucial for several reasons as listed below:

1. Consistency: Standardized training conditions ensure consistency across all iterations, allowing for fair comparisons of performance metrics. Without consistent training parameters, differences in results may stem from variations in training methodologies rather than inherent model capabilities.

2. Benchmarking: By training all iterations under the same conditions, we establish a reliable benchmark for evaluating their performance. This facilitates an accurate assessment of the advancements and improvements introduced in each version of YOLO.

3. Fair Comparison: Standardized training conditions eliminate confounding variables that could skew results and bias comparisons between different iterations. This ensures a level playing field, enabling an unbiased evaluation of model performance.

4. Reproducibility: Following consistent training protocols enhances the reproducibility of results, allowing other researchers to validate findings and build upon existing work with confidence.

5. Generalization: Models trained under standardized conditions are more likely to generalize well to unseen data and real-world scenarios. This increases the practical utility of the models, as they are better equipped to perform reliably in diverse environments beyond the training dataset.

5.1.1 YOLOv5

The YOLOv5 is the fifth iteration in the family of YOLO models. YOLOv5 offers several sub-models, with trainable parameters ranging from 1.9M to 140M. The smallest variant, known as YOLOv5n, is denoted by the suffix "nano", indicating its compact size and reduced parameter count. With only 1.9M parameters to train, the YOLOv5n is best suited for inference on edge devices, making it ideal for resource-constrained environments such as mobile devices, embedded systems, and IoT devices. Its lightweight

architecture enables efficient real-time object detection while maintaining high accuracy, making it a preferred choice for applications requiring fast and accurate detection on edge devices.

The model achieves mAP@0.5 of 0.738. This is considered a good accuracy for a lightweight model. A plot of mAP@0.5 Vs No. of epochs is shown in figure 5.1. Note that the model has already converged around epoch-30 but we have trained the model for 100 epochs. This is to stabilize the model and it also gives us an idea of hyperparamter tuning.

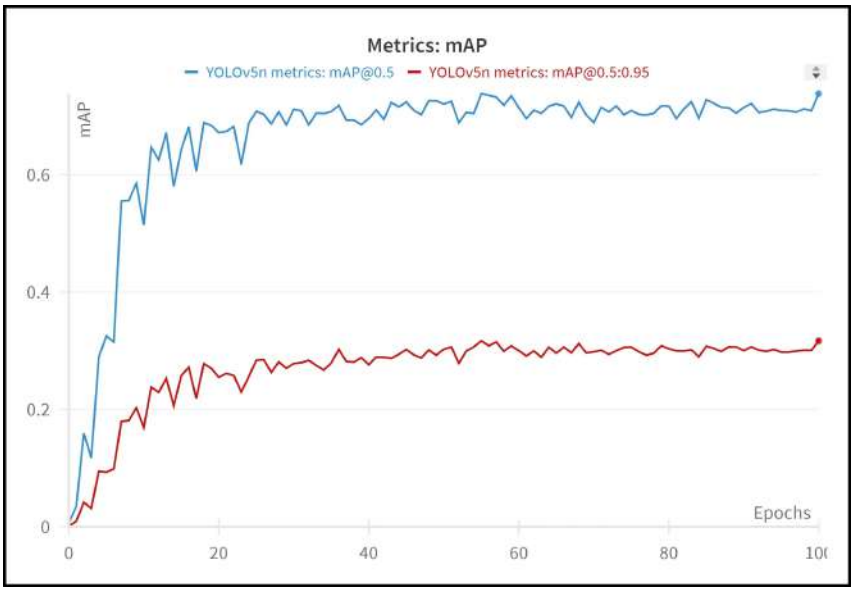


Figure 5.1: Plot of epochs vs mAP for YOLOv5n

The overall mAP depends on the precision and recall values calculated at the IoU threshold 0.5. The Plot of the PR curve for the model is as expected of any CNN model. Training and validation loss curves are also reported in figure 5.2

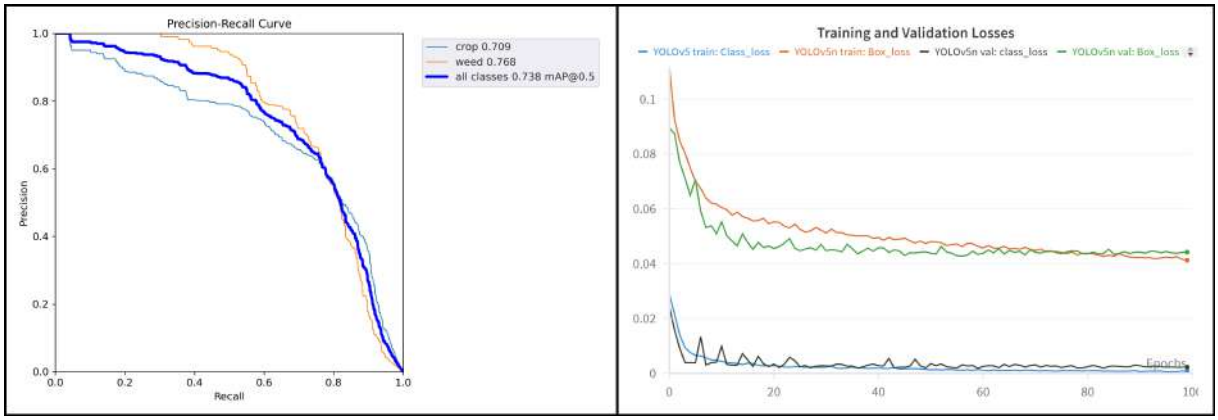


Figure 5.2: Precision-Recall curve (left) & Training and Validation losses (right) for YOLOv5n

5.1.2 YOLOv6

YOLOv6 marks the evolution in the YOLO series, differing notably from its predecessor, YOLOv5, in its structural design. While YOLOv5 employs the CSPNet (Cross-Stage Partial Network) as its backbone architecture, YOLOv6 adopts the EfficientRep backbone. This structural shift enhances the input representation of YOLOv6, optimizing it for GPU utilization. Consequently, YOLOv6 emerges as a superior network for industrial applications, offering enhanced efficiency and performance in GPU-intensive tasks. Similar to YOLOv5, the YOLOv6 offers range of models. The best suited model for edge deployment would be YOLOv6-lite with 1.09M parameters. The plots for mAP@0.5 Vs epochs and losses Vs epochs is shown in 5.3. The mAP@0.5 for this model is found to be 0.661. Which is slightly lower than YOLOv5n.

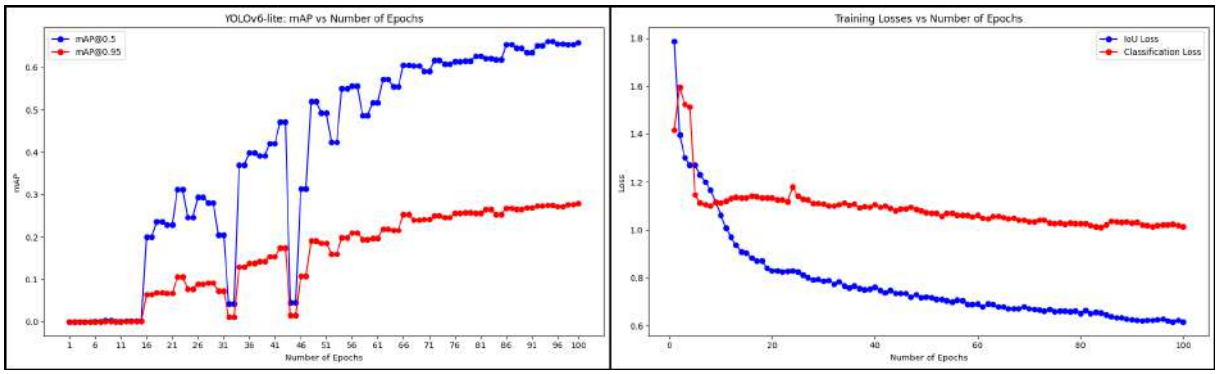


Figure 5.3: mAP values Vs epochs (left) & Training losses (right) for YOLOv6-lite

5.1.3 YOLOv7

According to the [18], the YOLOv7 model demonstrates significant performance gains over its predecessor on the MS COCO dataset, a widely accepted benchmark for evaluating object detection algorithms. The study reports a remarkable improvement, with YOLOv7 being purportedly 120% faster than its predecessor.

This advancement in speed is particularly noteworthy in the context of real-time applications, where rapid inference times are crucial for tasks such as video surveillance, autonomous driving, and robotics. By achieving a substantial increase in efficiency without compromising accuracy, YOLOv7 sets a new standard for object detection models, offering enhanced performance and responsiveness for various practical applications.

Our dataset was trained on YOLOv7-tiny, a lightweight version of YOLOv7. This tiny model has about 6M trainable parameters. This also comparable to YOLOv8's small model in terms of trainable parameters.

The figure 5.4 shows mAP@0.5 with respect to number of training epochs. For this case the mAP has reached 0.638 which is similar to that of YOLOv6-lite. The figure 5.5 shows the class wise average precision though PR-curve as well as the losses associated with training and validation.

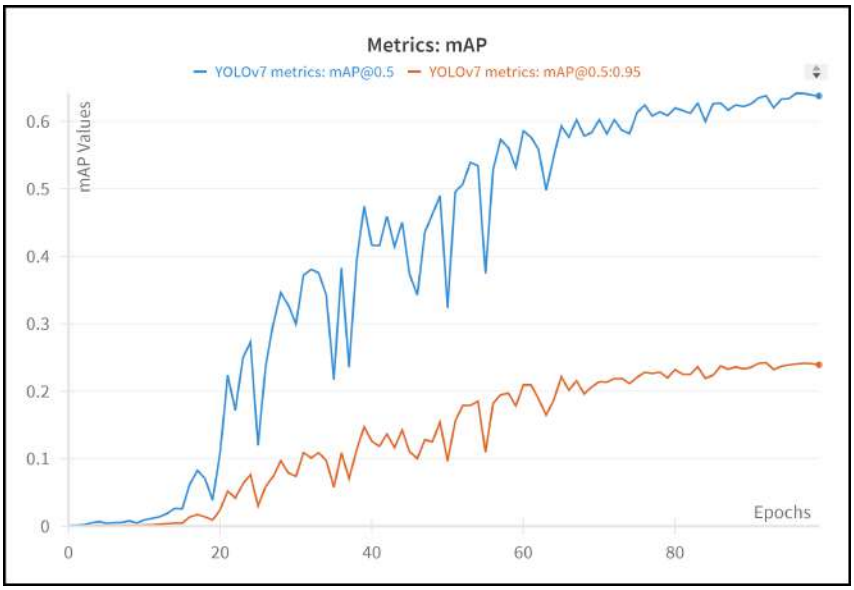


Figure 5.4: Mean Average Precision values for the YOLOv7-tiny

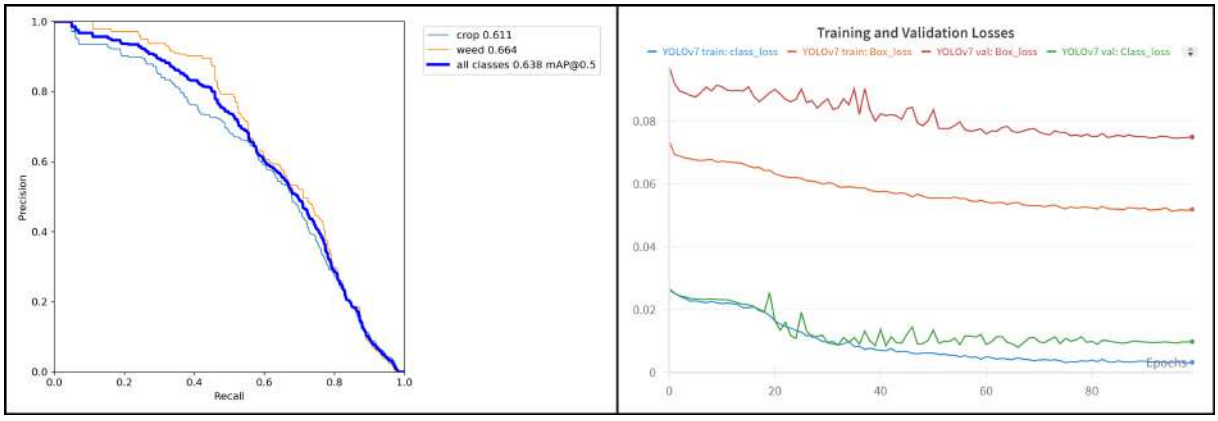


Figure 5.5: Precision and recall curve (left) & Training & validation losses (right) for YOLOv7-tiny

5.1.4 YOLOv8

YOLOv8 is the most recent addition to the YOLO family of object detectors. The model has similar structure to the previous iterations. The model is developed by a company called ultralytics through open-source. YOLOv8 was released with 5 different variations namely nano, small, medium, large and extra-large, all having progressively increasing number of parameters.

In our project, we opted to utilize the YOLOv8 nano model for training our dataset. With approximately 3.2 million parameters, this model strikes a balance between model complexity and computational efficiency, making it particularly well-suited for edge deployment. Notably, the YOLOv8 nano model offers a significantly small number of parameters, ensuring efficient inference even on resource-constrained devices like the Raspberry Pi.

For inference on CPU machines such as the Raspberry Pi, it becomes crucial to consider the model's

computational requirements. The YOLOv8 nano model exhibits a low number of GFLOPs, specifically 8.195, which aligns well with the performance capabilities of the Raspberry Pi 4B. According to VMW Research Group's analysis, the Raspberry Pi 4B is capable of delivering between 9.69 to 13.5 GFLOPs, depending on the user's operating system. This compatibility makes models like YOLOv8 an optimal choice for real-time object detection tasks, especially in edge computing scenarios.

We trained the model with same configuration as all other models. For YOLOv8 nano the model achieves best mAP@0.5 of 0.657. The plots for mAP Vs epochs is shown in figure 5.6 and the PR curve along with losses is shown in figure 5.7.

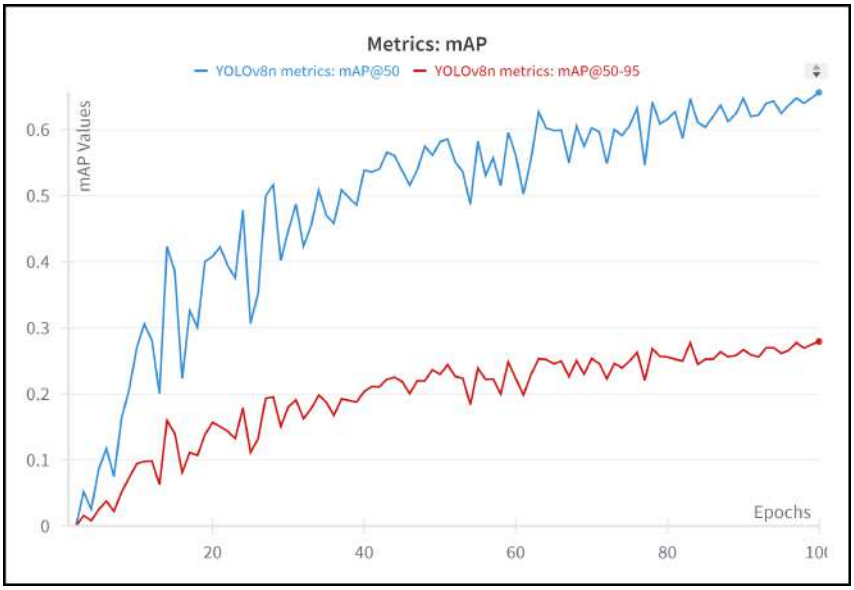


Figure 5.6: Plot of epochs vs mAP for YOLOv8 nano

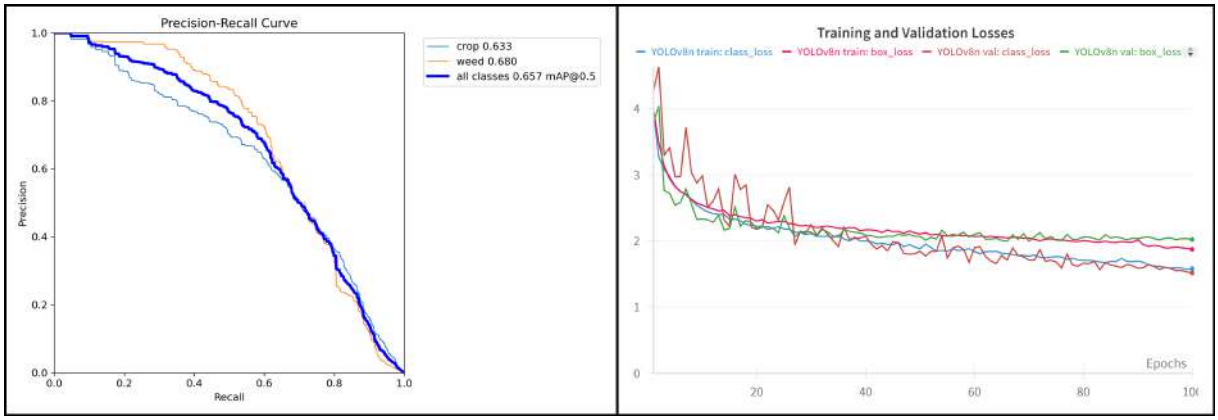


Figure 5.7: Precision and recall curve (left) & Training & validation losses (right) for YOLOv8 nano

The training summary of all YOLO models presented in Table 5.2 indicates a clear trend: YOLO lightweight models demand less computational resources (measured in GFLOPs) for the same task, along with a slight compromise in accuracy (mAP value). Notably, the model size post-training remains under 5MB for three out of four models, which is notably compact for deep learning models. For instance, the YOLOv8m model, classified as medium-sized, boasts nearly 26 million parameters, showcasing a significant disparity in memory footprint.

Model	No. of Trainable Parameters	Best mAP@0.5	GFLOPs	Trained Model Size (in mega bytes)
YOLOv5-nano	1766623 (1.7M)	0.738	4.2	3.65
YOLOv6-lite	1.09M	0.661	∼1	2.42
YOLOv7-tiny	2349814 (2.3M)	0.638	5.6	4.69
YOLOv8-nano	3011238 (3M)	0.657	8.2	5.94

Table 5.2: Comparison of all the trained models

5.2 Validation and Inference

To ensure a fair comparison, all the trained models were evaluated using the same set of test images. Among the models under consideration, YOLOv5 was expected to perform the best due to its advantage in terms of overall mean Average Precision (mAP). The test dataset consists of 230 unseen images, which includes image data from both farms. This test image folder can be used for evaluating the performance of the models. The model initially saves the best trained weights in a PyTorch file format. There is, however, a procedure to convert this PyTorch model into the ONNX format, which is a standard open format for representing machine learning models. The ONNX model can then be optionally quantized to reduce its size and improve inference speed, at the potential cost of some accuracy loss.

The ONNX (Open Neural Network Exchange)[16] format provides an open standard for representing deep learning models in a hardware-agnostic way. This makes it easier to optimize models for different hardware platforms and environments, including edge devices like smartphones, embedded systems, and IoT devices.

The table 5.3 summarizes the inference results obtained from testing the models on this test dataset.

Model	Image Size	Best mAP@0.5	Model Size (in MBs)	Average Inference Time (CPU)
YOLOv5-nano	224 x 224	0.738	3.65	PyTorch=17.5ms : ONNX=7.2ms
YOLOv6-lite	224 x 224	0.661	2.42	PyTorch= ∼260ms : ONNX= ∼11ms
YOLOv7-tiny	224 x 224	0.638	4.69	PyTorch=22ms : ONNX=9.2ms
YOLOv8-nano	224 x 224	0.657	5.94	PyTorch=23.5ms : ONNX=12.8ms

Table 5.3: The table shows average inference time taken for test images. We have used both PyTorch as well as ONNX for inference purpose. This inference is taken from Intel Xeon Platinum processor with 2.5GHz speed [20]

Taking into account all considerations including mAP value, model size, and inference on a cloud CPU, YOLOv5n emerges as the optimal model for edge deployment with our dataset. It exhibits strong

performance on test data with the lowest inference time, a crucial factor for real-time object detection. We extracted the weights in ONNX format for better hardware utilization and use this format in the Raspberry Pi along with PyTorch format.

5.3 Confidence Score

The figure 5.8 shows some instances of predicted classes using bounding box around the object and a parameter called confidence score.

Figure 5.8: Inference results: from left YOLOv5, YOLOv6, YOLOv7 and YOLOv8

The confidence score (3) serves as a measure of the model's confidence in its predictions, indicating how assured it is when assigning a class label and drawing a bounding box around the predicted object.

$$Confidence\ Score = Box\ Confidence * Class\ Confidence \tag{3}$$

Here, the box confidence refers to the probability that the bounding box has a specific object inside it, which is one of the classes. At the same time, class confidence signifies the model's estimation of whether the object belongs in that box.

5.4 Inference on Raspberry Pi

The final goal of the project is to get a fast and reliable inference on Raspberry Pi board. We used Raspberry Pi 4 model B with 4GB of RAM for this purpose. The steps followed to achieve model inference on Raspberry Pi are:

1. Install the 64-bit Linux based Raspberry Pi OS.
2. Create a python environment for running inference.
3. Clone the respective repositories of the YOLO models on to the environment.
4. Keep the trained model checkpoint files as well as the test data on the environment directory.
5. Once everything is set up test the model.
6. Connect the Raspberry Pi camera and keep the source of the data to '0', now we are using camera as our source and use this for real-time inference.

The usability of video inference on any edge is determined through a metric called FPS (Frames per Second). The simple explanation is that the video nothing but fast moving images called frames, and FPS is a measure of how fast the model is processing each frame. Typical video shot at 24-30 frames per second.

The formula for calculation of FPS is generally given as:

$$\text{FPS} = \frac{1}{\text{Average time taken for inference over all frames}} \tag{4}$$

As per our observations from the inference on CPU of different models, it is clear that the YOLOv5n out performs other models for our particular dataset. Thus the YOLOv5n is used for hardware inference on the Raspberry Pi.

The figure 5.9 shows the simple experimental setup used for inference on Raspberry Pi SBC.

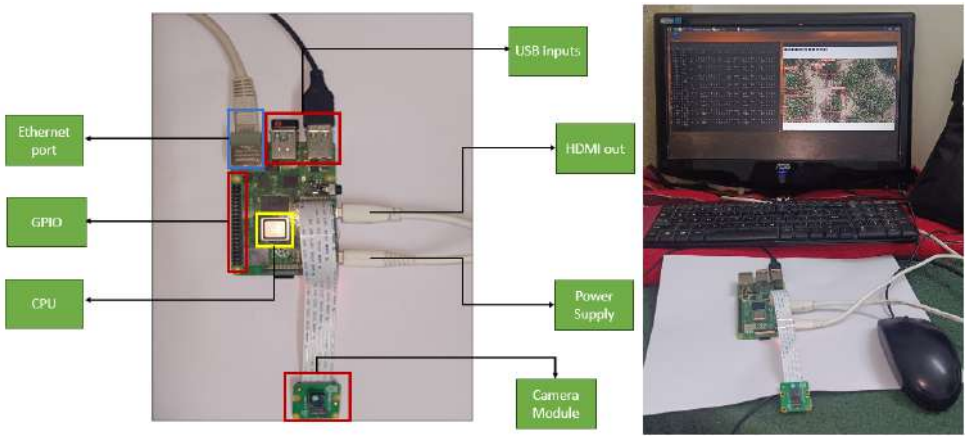


Figure 5.9: Experimental Setup

The steps that needs to be followed in order to achieve image or video inference in real-time are:

1. Run the terminal in Raspberry Pi.
2. cd into the Python environment using the command `source` *`env/bin/activate`*, as shown in figure 5.10.
3. Once inside the environment cd into the repository of YOLOv5.
4. Install required dependencies using `pip install -r requirements.txt`.
5. Copy the final model checkpoints to directory inside environment, for our case we have named the checkpoints after training as bestVx.pt and bestVx.onnx; where x=5,6,7 and 8. We will be using bestV5.onnx model weights for inference.
6. Run the command as shown in figure 5.11 for starting the detection. Change the source location for images and videos which are locally stored in or outside environment.
7. Visualize the results by setting the view-img argument to True, so that the video or image is displayed once detection is done on each frame.

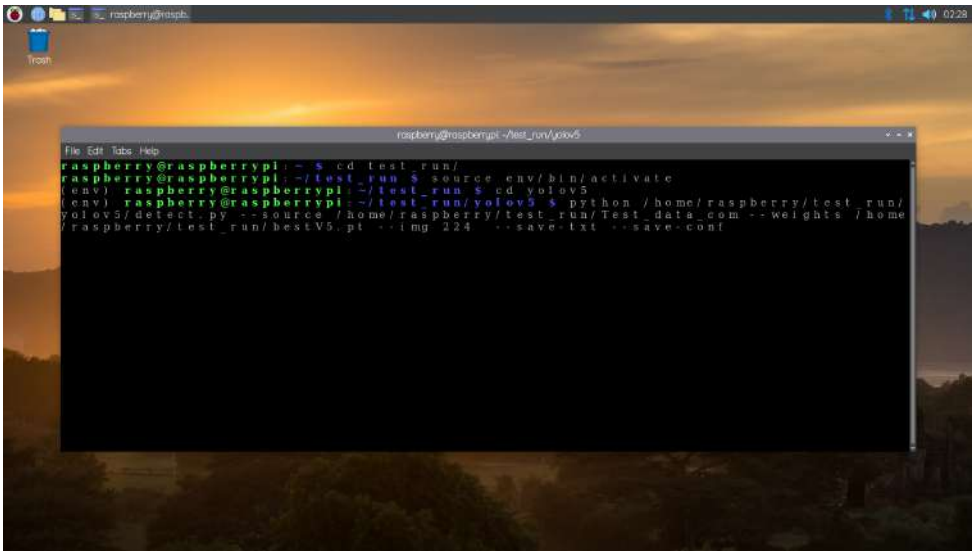


Figure 5.10: Initial setup inside Raspberry Pi terminal

5.4.1 Performance Evaluation

The locally saved video was loaded on the model. The video was taken from a handheld smartphone for inference purposes only. The issue with the video was that it was shot on 60FPS speed with the smartphone camera. This means even after detections are saved by the model, it will be harder for us to get any useful information out of it as video frame rate is quite smooth; Thus we slowed down the video by adding few more frames on to the image using online tools. The total length of now slowed video was around 2minutes, with about 11000 frames.

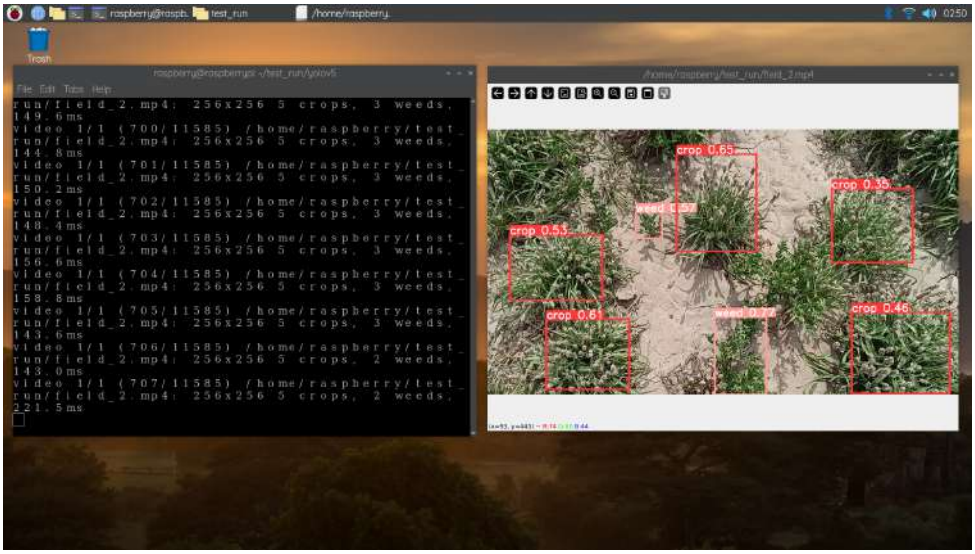


Figure 5.11: Detection on locally saved video file

Each detection takes about 140 - 220ms for detection on each frame. Assuming the average detection time to be around 200ms, we can calculate the FPS for the given model detections.

$$\text{FPS} = \frac{1}{\text{Average time taken for inference over all frames}} = \frac{1}{200} = 5\text{FPS} \tag{5}$$

This is a very promising result, as [6] deployed their YOLOv4-tiny model on the same SBC, they had achieved an FPS of 4.

Raspberry Pi is run by Cortex-A72 (ARM v8) 64-bit SoC @ 1.8GHz. The SoC has four cores that it can utilize. While running the detection, the Raspberry Pi was utilizing all the four cores to its full capacity (refer figure 5.12).

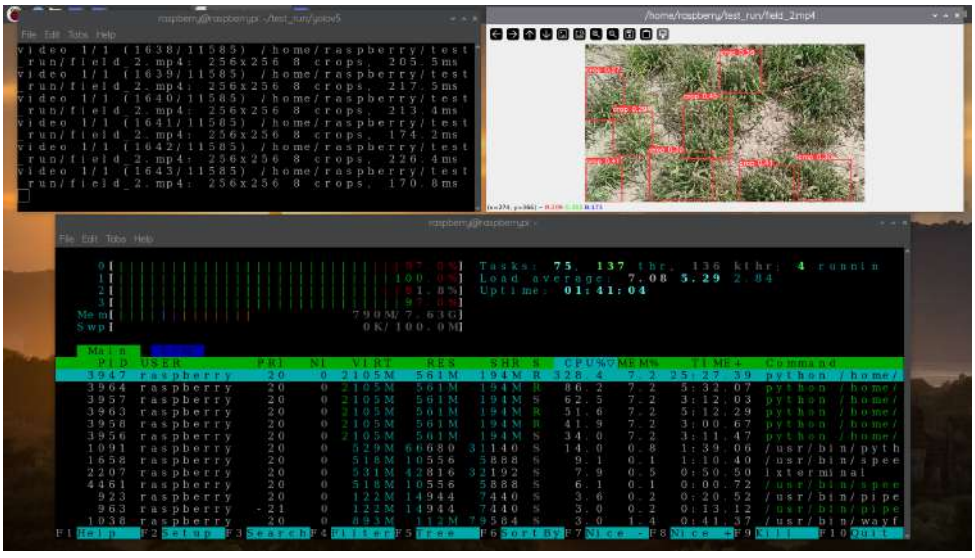


Figure 5.12: System process manager view in real-time while running the detections

The Raspberry Pi that we are using has no active cooling or overclocking. Without active cooling,

the CPU temperature can reach over 80° Celsius. At this temperature, the CPU performance decreases over time. This is due to thermal throttling; as CPU tries to reduce the performance in order to prevent overheating. According to [21] the safe operating temperature of Raspberry Pi is from −45° to 85° Celsius. Figure shows the temperature of Raspberry Pi CPU overtime when we start running the detctions using YOLOv5n model. Figure 5.13

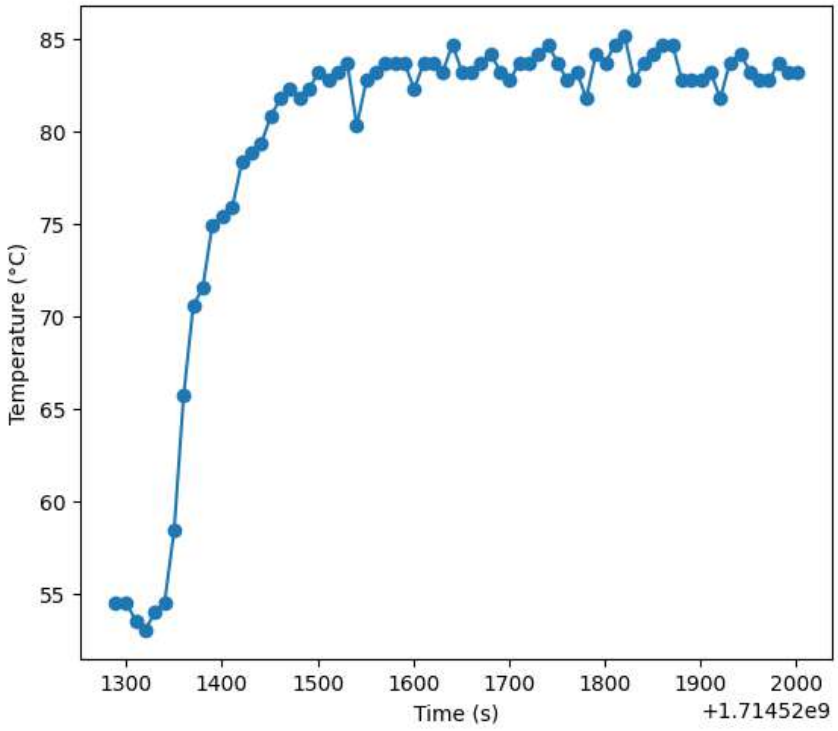


Figure 5.13: Temperature in degree Celsius over time in seconds of Raspberry Pi CPU while running detection on a video file. The data was taken from internal temperature sensor connected to CPU. Initially the temperature was around 53−55° degrees but just after 1000 seconds of running the model, temperature comes close to 85° degrees; which is the safe operating temperature specified for the Raspberry Pi CPU.

6 Conclusions

The YOLO family of models are state-of-the-art object detection models currently used widely for object detection and tasks like image segmentation, pose estimation, and classification.

The lightweight versions of all YOLO version 5 through version 8 were explored during this project to determine the best-suited model that can be deployed on the designated hardware, i.e., Raspberry Pi SBC.

We collected over 1500 RGB images from two local farmlands that grew fennel seeds and psyllium, locally known as jeera/isabgol plants. The dataset was preprocessed, augmented, annotated, and formatted using the YOLO-specified directory format. The same dataset was used for all the YOLO versions with the same training conditions to keep the model consistent and reduce disparity.

After training, we performed inference on a set of unseen test images to gauge the performance of each model's best weight using a cloud CPU. Interestingly, the YOLOv5 is the best model of all, even beating the latest iteration, the YOLOv8n. There can be many reasons for this, but the main reason is how the features are extracted on these models; as we have prepared our own data, it just scores better on YOLOv5 than on YOLOv8.

With a definitive result, we deployed the YOLOv5n's best weights in the ONNX format to Raspberry Pi 4 model B. The ONNX runtime gives better CPU performance than its counterpart, the PyTorch runtime.

We found out that the YOLOv5n performs exceptionally well, considering the CPU limitations and with no active cooling or overclocking. The detection of image data takes on an average of 150-200ms, which is expected of any resource-constrained device. We calculated the FPS on a recorded video to be 5FPS; this is still higher than [6], which uses the YOLOv4-tiny model at 4FPS.

From this project, we can conclude that YOLO models are easily scalable. As long as we have datasets annotated properly, we can train them to adapt to any situation. The nano or tiny versions of the YOLO models strike a perfect balance between accuracy and detection speed. The optimal weights after training these models can be used for inference purposes on edge devices such as Raspberry Pi and get respectable results from these CPU-only devices.

References

[1] Kubiak, Adrianna & Wolna-Maruwka, Agnieszka & Niewiadomska, A. & Pilarska, Agnieszka. (2022). The Problem of Weed Infestation of Agricultural Plantations vs. the Assumptions of the European Biodiversity Strategy. Agronomy. 12. 1808. 10.3390/agronomy12081808.

[2] Aichen Wang, Wen Zhang, Xinhua Wei, A review on weed detection using ground-based machine vision and image processing techniques, Computers and Electronics in Agriculture, Volume 158, 2019, Pages 226-240, ISSN 0168-1699, https://doi.org/10.1016/j.compag.2019.02.005.

[3] Sportelli, M.; Apolo-Apolo, O.E.; Fontanelli, M.; Frasconi, C.; Raffaelli, M.; Peruzzi, A.; Perez-Ruiz, M. Evaluation of YOLO Object Detectors for Weed Detection in Different Turfgrass Scenarios. Appl. Sci. 2023, 13, 8502. https://doi.org/10.3390/app13148502

[4] https://github.com/ultralytics/ultralytics.git & https://docs.ultralytics.com/

[5] Talaat, F.M., ZainEldin, H. An improved fire detection approach based on YOLO-v8 for smart cities. Neural Comput & Applic 35, 20939–20954 (2023). https://doi.org/10.1007/s00521-023-08809-1

[6] U. Farooq, A. Rehman, T. Khanam, A. Amtullah, M. A. Bou-Rabee and M. Tariq, "Lightweight Deep Learning Model for Weed Detection for IoT Devices," 2022 2nd International Conference on Emerging Frontiers in Electrical and Electronic Technologies (ICEFEET), Patna, India, 2022, pp. 1-5, doi: 10.1109/ICEFEET51821.2022.9847812.

[7] P. Kumar Doddamani and G. P. Revathi, "Detection of Weed & Crop using YOLO v5 Algorithm," 2022 IEEE 2nd Mysore Sub Section International Conference (MysuruCon), Mysuru, India, 2022, pp. 1-5, doi: 10.1109/MysuruCon55714.2022.9972386.

[8] Dang, F., Chen, D., Lu, Y., Li, Z., 2023. YOLOWeeds: A novel benchmark of YOLO object detectors for multi-class weed detection in cotton production systems. Computers and Electronics in Agriculture. https://doi.org/10.1016/j.compag.2023.107655.

[9] Rai, Nitin & Sun, Xin & Igathinathane, C. & Howatt, Kirk & Ostlie, Michael. (2023). Aerial-Based Weed Detection Using Low-Cost and Lightweight Deep Learning Models on an Edge Platform. Journal of the ASABE. 66. 1041-1055. 10.13031/ja.15413.

[10] Osorio, K.; Puerto, A.; Pedraza, C.; Jamaica, D.; Rodríguez, L. A Deep Learning Approach for Weed Detection in Lettuce Crops Using Multispectral Images. AgriEngineering 2020, 2, 471-488.

https://doi.org/10.3390/agriengineering2030032

[11]Yogita Gharde, P.K. Singh, R.P. Dubey, P.K. Gupta, Assessment of yield and economic losses in agriculture due to weeds in India, Crop Protection, Volume 107, 2018, Pages 12-18, ISSN 0261-2194, https://doi.org/10.1016/j.cropro.2018.01.007.

[12] Yield and Economic Losses Due to Weeds in India (2018), ICAR-Directorate of Weed Research, Jabalpur, 22 p.

[13] R. Dabhi and D. Makwana, crop and weed detection data with bounding boxes, 2020, [online] Available: https://www.kaggle.com/ravirajsinh45/crop-and-weed-detection-data-with-bounding-boxes.

[14] Redmon, Joseph & Divvala, Santosh & Girshick, Ross & Farhadi, Ali. (2016). You Only Look Once: Unified, Real-Time Object Detection. 779-788. 10.1109/CVPR.2016.91.

[15] https://github.com/ultralytics/ultralytics/issues/189

[16]ONNX

[17] YOLOv5 repository

[18]C. -Y. Wang, A. Bochkovskiy and H. -Y. M. Liao, "YOLOv7: Trainable Bag-of-Freebies Sets New State-of-the-Art for Real-Time Object Detectors," 2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), Vancouver, BC, Canada, 2023, pp. 7464-7475, doi: 10.1109/CVPR52729.2023.00721.

[19]makesense.ai website

[20]Lightning.ai cloud editor

[21] Hussain, M. YOLO-v1 to YOLO-v8, the Rise of YOLO and Its Complementary Nature toward Digital Manufacturing and Industrial Defect Detection. Machines 2023, 11, 677. https://doi.org/10.3390/machines11070677

13% Overall Similarity

Top sources found in the following databases:

- 8% Internet database
- Crossref database
- 8% Submitted Works database
- 8% Publications database
- Crossref Posted Content database

TOP SOURCES

The sources with the highest number of matches within the submission. Overlapping sources will not be displayed.

1	machinelearningknowledge.ai Internet	<1%
2	Carnegie Mellon University on 2024-03-31 Submitted works	<1%
3	mdpi.com Internet	<1%
4	Institute of International Studies on 2024-04-10 Submitted works	<1%
5	daneshyari.com Internet	<1%
6	researchgate.net Internet	<1%
7	Arts, Sciences & Technology University In Lebanon on 2024-01-03 Submitted works	<1%
8	Asia Pacific University College of Technology and Innovation (UCTI) on... Submitted works	<1%

9	0-www-mdpi-com.brum.beds.ac.uk Internet	<1%
10	assets.researchsquare.com Internet	<1%
11	Fengying Dang, Dong Chen, Yuzhen Lu, Zhaojian Li. "YOLOWeeds: A no... Crossref	<1%
12	Pawan Kumar Doddamani, G P Revathi. "Detection of Weed & Crop usin... Crossref	<1%
13	Rai, Nitin. "Weed Identification on Drone-Captured Images Using Edge ... Publication	<1%
14	Islomjon Shukhratov, Andrey Pimenov, Anton Stepanov, Nadezhda Mik... Crossref	<1%
15	Mohammad Shahin, F. Frank Chen, Ali Hosseinzadeh, Mazdak Maghan... Crossref posted content	<1%
16	Oruganti, Venkatakrishna Reddy. "Drone Detection and Tracking Using ... Publication	<1%
17	"Deep Learning and Big Data for Intelligent Transportation", Springer S... Crossref	<1%
18	University of Sydney on 2021-06-04 Submitted works	<1%
19	ar5iv.labs.arxiv.org Internet	<1%
20	arxiv.org Internet	<1%

21	studymoose.com Internet	<1%
22	sifisheriessciences.com Internet	<1%
23	docplayer.net Internet	<1%
24	Higher Education Commission Pakistan on 2023-06-13 Submitted works	<1%
25	Indiana University on 2024-04-10 Submitted works	<1%
26	Universiti Teknologi Malaysia on 2023-07-12 Submitted works	<1%
27	University of Technology, Sydney on 2023-05-22 Submitted works	<1%
28	opensource-heroes.com Internet	<1%
29	Shoibam Amritraj, Nitish Hans, C. Pretty Diana Cyril. "An Automated an... Crossref	<1%
30	Susmita Haldar, Luiz Fernando Capretz. "Interpretable Software Defect... Crossref	<1%
31	UOW Malaysia KDU University College Sdn. Bhd on 2024-03-28 Submitted works	<1%
32	dspace.vsb.cz Internet	<1%

33	frontiersin.org Internet	<1%
34	CSU, San Jose State University on 2018-04-25 Submitted works	<1%
35	Marios Vasileiou, Leonidas Sotirios Kyriakos, Christina Kleisiari, Georgi... Crossref	<1%
36	Nimpa Fondje, Cedric A.. "Bridging Domain Gaps for Cross-Spectrum a... Publication	<1%
37	Shulin Sun, Junyan Yang, Zeqiu Chen, Jiayao Li, Ruizhi Sun. "Tibia-YOL... Crossref	<1%
38	Submitted on 1688452039405 Submitted works	<1%
39	github.com Internet	<1%
40	oup.silverchair-cdn.com Internet	<1%
41	storage.googleapis.com Internet	<1%
42	autobotic.com.my Internet	<1%
43	ideals.illinois.edu Internet	<1%
44	ptomontt.cl Internet	<1%

45	CSU Northridge on 2023-04-25 Submitted works	<1%
46	Chung-Liang Chang, Bo-Xuan Xie, Sheng-Cheng Chung. "Mechanical C... Crossref	<1%
47	Edmundo Casas, Leo Ramos, Eduardo Bendek, Francklin Rivas-Echever... Crossref	<1%
48	Higher Education Commission Pakistan on 2022-05-25 Submitted works	<1%
49	Institut Pertanian Bogor on 2024-04-23 Submitted works	<1%
50	Institute of Technology, Nirma University on 2023-05-18 Submitted works	<1%
51	La Trobe University on 2019-04-06 Submitted works	<1%
52	Md Jobair Hossain Faruk, Muhammad Asadur Rahman. "Object Detecti... Crossref	<1%
53	University of Nebraska, Lincoln on 2022-09-28 Submitted works	<1%
54	University of West Florida on 2024-03-01 Submitted works	<1%
55	Wageningen University on 2022-04-22 Submitted works	<1%
56	archive.org Internet	<1%

57	ebin.pub Internet	<1%
58	elib.bsu.by Internet	<1%
59	etasr.com Internet	<1%
60	iitj on 2024-04-15 Submitted works	<1%
61	researchsquare.com Internet	<1%

Excluded from Similarity Report

- Bibliographic material
- Cited material
- Manually excluded text blocks
- Quoted material
- Small Matches (Less then 8 words)

EXCLUDED TEXT BLOCKS

A Project Report Submitted by

iitj on 2024-01-03

submitted to the Indian Institute of Technology Jodhpur inpartial fulfilment of the r...

iitj on 2024-01-03

to the Indian Institute of TechnologyJodhpur for the award of the degree of M.Tec...

iitj on 2024-01-03

is a bonafide record ofthe research work done by him under my supervision. To th...

iitj on 2024-01-03

is a bonafide record of the research work carried out under the supervision of Dr.A...

Indian Institute of Technology Jodhpur on 2022-11-28

viContentsAbstractvi1 Introduction and background22 Literature survey

iitj on 2024-01-03

May, 2024DeclarationI hereby declare that the work presented in this Project Repo...

iitj on 2024-04-29

CertificateThis is to certify that the Project Report titled

Indian Institute of Technology Jodhpur on 2021-01-20