Secureye

Pranav Raut[1], Prajwal Vaidya[2], Niraj Koli[2], Aryan Kokabankar[2], Pratik Bhalkare[2], Ashish Nagarse[1]

[1]Guardinger Technologies Pune,India,
[2]Dept. of Computer Engineering Vidyalankar Institute Of Technology Wadala,Mumbai,India

Sachin Deshpande, Dept of Computer Engineering Vidyalankar Institute Of Technology Wadala,Mumbai,India

Abstract

Objectives: The "Secureye" project is designed to enhance military surveillance by quickly detecting potentially dangerous or unauthorized items, such as weapons and armored vehicles, thereby helping to prevent conflicts and casualties among military personnel. **Methods**: The system leverages the YOLO (You Only Look Once) model for real-time object detection, chosen for its combination of high speed and accuracy. A comparison between Faster R-CNN and YOLOv5 revealed that YOLOv5 had a tenfold higher inference rate, making it the preferred choice for real-time detection in military applications. **Findings**: The system leverages the YOLO (You Only Look Once) model for real-time object detection, chosen for its combination of high speed and accuracy. A comparison between Faster R-CNN and YOLOv5 revealed that YOLOv5 had a tenfold higher inference rate, making it the preferred choice for real-time detection in military applications. **Novelty**: These features, combined with an optimized alert system and detailed report generation, position "Secureye" as a significant advancement in military surveillance technology.

Keywords: YOLO, Object Detection, Security, CNN, Machine Learning.

1. Introduction

Military object detection technology is a backbone in modern defense strategies, fostering advanced surveillance and threat mitigation. Recent research, exemplified by studies like "Real-time Object Detection for Military Applications using YOLO Algorithm," has delved deeply into developing real-time object detection systems tailored specifically for military use. Innovations such as YOLO (You Only Look Once) have revolutionized object recognition by significantly improving accuracy and speed, enabling swift identification of potential threats such as weapons and vehicles [2]. The integration of these cutting-edge systems into military surveillance frameworks ensures rapid responses to hostile activities. Concurrently, open-access platforms like MDPI serve as crucial hubs, fostering collaboration among researchers and facilitating the exchange of findings on military object detection innovations. This fusion of military focused object detection technologies with open-access scholarly platforms is shaping the future of defense strategies, contributing to a more secure global landscape.

Neural networks have emerged as one of the most prominent algorithms today, showcasing superior accuracy and speed in processing large amounts of data over time. At their core, neurons and activation functions form the fundamental building blocks of any neural network, typically comprising input and output nodes along with hidden layers. While traditional neural networks consist of a single hidden layer, deep neural networks, characterized by multiple hidden layers, have gained traction. These deep architectures, such as Convolutional Neural Networks (CNNs), have found widespread application in tasks like object detection and image processing, demanding substantial processing power.

Advanced techniques like Faster R-CNN and YOLO (You Only Look Once) have been developed to enhance object detection, including applications like identifying cars in aerial images. A detailed comparison between Faster R-CNN and YOLOv3 has revealed that YOLOv3 surpasses Faster RCNN in various performance metrics and suitability for different applications and environments. This understanding aids in selecting the most appropriate model for specific use cases.[1]

Computer vision is a collaboration that has received great attention in recent years (since CNN), with self-driving cars at the forefront, and its importance is visual inspection. The difference between object search algorithms and classification algorithms is that in search algorithms, we try to draw bounding boxes around objects of interest to find them in the image. Also, in case of object detection you may not need to just draw the bounding box. There may be many boxes representing different objects in the image, and you cannot know their number in advance. The main reason why this problem cannot be solved by creating a network communication model and a fully connected process is that the length of the output process is different - not always, because it may not always be the number of times the item of

interest appears. A naive way to solve this problem is to obtain different regions of interest from the image and use CNN to classify objects in this region. The problem with this approach is that objects of interest may have different locations and different locations in the image. Therefore, you should select large areas that may cause the calculation to fail. For this reason, algorithms such as faster R-CNN and YOLO have been developed to quickly detect these situations.

Recent years have seen remarkable advancements in deep learning, particularly in computer vision, with techniques like Convolutional Neural Networks (CNNs) playing a pivotal role in object detection tasks. CNNs, inspired by the visual processing mechanisms of living organisms, were initially proposed by Fukushima in 1980 and subsequently refined by LeCun. They excel in processing multi-dimensional arrays of data, such as the color channels of an image.

CNNs leverage four key concepts to exploit the characteristics of natural signals:

1. Local connections
2. Shared weights
3. Pooling
4. Multiple layers

The typical architecture of a CNN comprises several layers:

1. Convolutional layers: These layers extract features from input data, with early layers detecting low level features like edges and corners, while deeper layers capture high-level features such as objects and structures. Each unit in a convolutional layer is connected to a local patch in the previous layer through a set of kernels, followed by a nonlinearity operation like ReLU (rectified linear unit).
2. Pooling layers: Positioned between convolutional layers, pooling layers reduce dimensionality and enhance invariance to small shifts and distortions.
   Each unit computes the maximum value within a local patch of units in a feature map.
3. Fully connected layers: Typically located towards the end of the network, these layers summarize information from lower-level layers for final decision-making. However, as they contain a significant number of parameters, overfitting is a concern, often addressed using dropout.

Since the breakthrough success of AlexNet in the 2012 ImageNet Large-Scale Visual Recognition Challenge (ILSVRC), numerous CNN architectures have been developed, including VGGNet, GoogLeNet, and ResNet.

1. AlexNet, pioneered by Krizhevsky et al., marked a significant milestone by employing nonsaturating neurons, GPU acceleration, and dropout to prevent overfitting.
2. VGGNet, winner of the ILSVRC 2014 competition, is renowned for its simplicity, featuring architectures like VGGNet-16 with 13 convolutional layers, five pooling layers, and three fully connected layers.
3. GoogLeNet distinguishes itself by using filter kernels of different sizes within the same layer, preserving spatial information, and reducing network parameters to combat overfitting.
4. ResNet, awarded the Best Paper at the Conference on Computer Vision and Pattern Recognition in 2016, introduced the concept of residual learning, enabling the training of extremely deep networks efficiently, leading to superior generalization performance and victories in various prestigious competitions.[3]

YOLO: YOLO or You Only Look Once is an object detection algorithm much different from the region-based algorithm seen above. In YOLO a single convolutional network predicts the bounding boxes and the class probabilities for these boxes.

YOLO works by taking an image and split it into a side x side grid, within each of the grid we take m bounding boxes. For each of the bounding box, the network outputs a class probability and offset values

for the bounding box. The bounding boxes having the class probability above a threshold value is selected and used to locate the object within the image.

YOLO is orders of magnitude faster (45 frames per second) than other object detection algorithms. The limitation of YOLO algorithm is that it struggles with small objects within the image, for example it might have difficulties in detecting a flock of birds. This is due to the spatial constraints of the algorithm.[4]

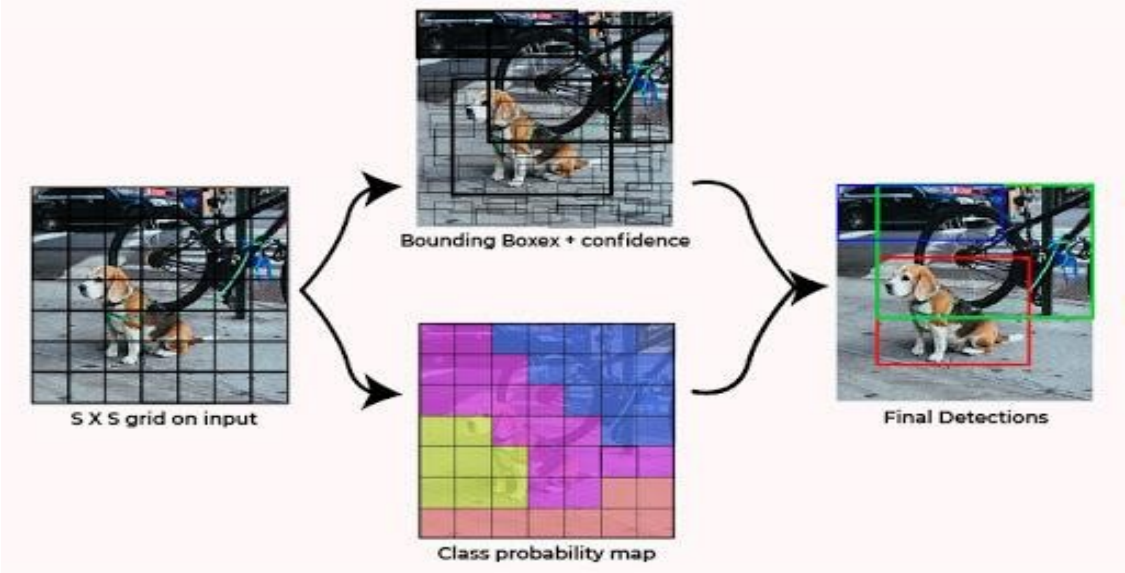

Figure 1: YOLO Bounding Box, Object detection and localization

Faster R-CNN vs YOLO: The study [2] utilized a common dataset for both YOLOv5 and Faster R-CNN, containing various spacecraft features such as solar panels and satellite bodies. These features were required to be identifiable by humans and relate to real-life satellite components. Each image in the dataset was unique. The objective was to evaluate and ascertain the superior algorithm for detecting anomalies in space. Results from the testing dataset favored Faster R-CNN over YOLOv5, although YOLOv5 exhibited a tenfold higher inference rate, rendering it the preferred choice for real-time object detection in this context. A similar investigation by [4] yielded comparable findings, with YOLO proving to be a cleaner and more efficient object detection solution, particularly due to its end-to-end training capability. While both algorithms demonstrated reasonably high accuracy, YOLO occasionally surpassed Faster R-CNN in accuracy, speed, and efficiency. Its single-shot approach and ability to directly train on full images make it suitable for real-time detection in both images and videos. YOLO's superior generalization of objects compared to Faster R-CNN positions it as a more dependable, swift, and robust algorithm. These prominent advantages strongly advocate for its adoption.

With YOLO established as the preferred algorithm for the project, the remaining consideration is selecting the appropriate version. YOLO has evolved through various iterations, including YOLOv1, v2, v3, v4, v5, v6, v7, and v8. Notably, versions v3, v5, and the latest, v8, stand out. Benchmark tests conducted by Stereolabs concluded that all three versions (v5, v7, and v8) perform well on the Jetson Orin platform.[5]

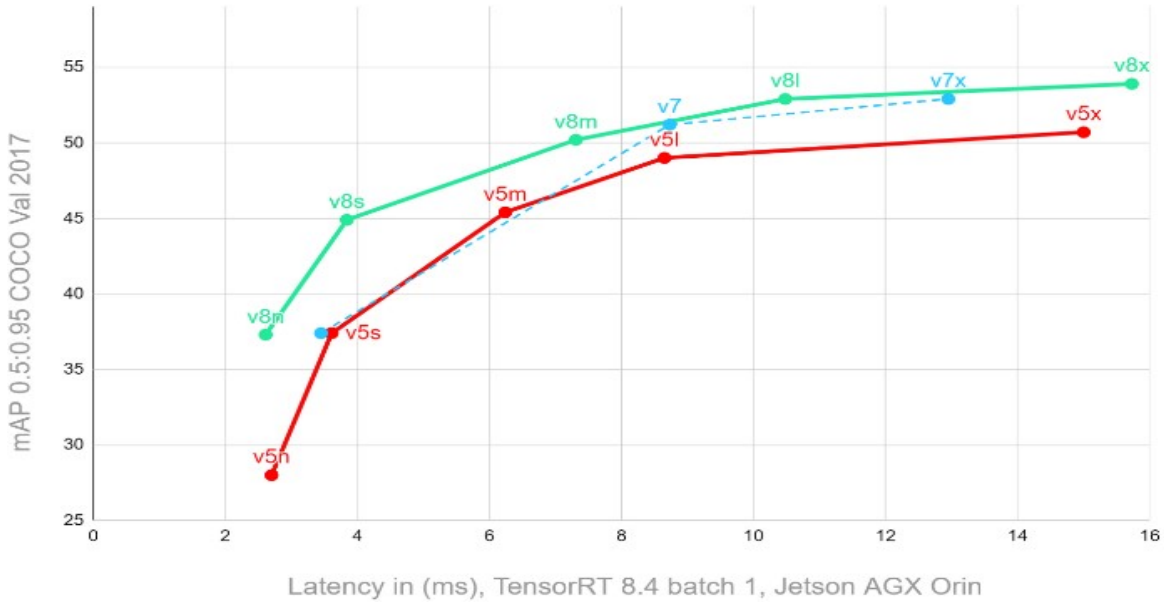

Figure 2: Comparison between YOLO versions

2. Methodology

2.1 Data Collection and Preparation

Considering the nature of the project finding large datasets seemed difficult thus the process chosen was to integrate several smaller datasets to form one large dataset to fit all our needs. To start, curated a massive variety of datasets from different domains. After having Skimmed through them to identify those that are exactly suitable for the task's needs and purposes. Download them in the compatible format and, for instance, YOLO, to fit the annotation and training pipeline. Using the data set acquired, classes and annotation had to be redone to fit our needs and help standardize the process.

Figure 3: Sample annotated image

2.2 Model training:

The training process involved configuring the YOLO model with specific parameters tailored to the characteristics of the dataset and the requirements of the use case.

Training settings for YOLO models refer to the various hyperparameters and configurations used to train the model on a dataset. These settings can affect the model's performance, speed, and accuracy. Some common YOLO training settings include the batch size, learning rate, momentum, and weight decay.

2.2.1 Some useful parameters:

- Epochs: Total number of training epochs. Each epoch represents a full pass over the entire dataset. Adjusting this value can affect training duration and model performance.

- Device: Specifies the computational device(s) for training: a single GPU (device=0), multiple GPUs (device=0,1), CPU (device=cpu), or MPS for Apple silicon (device=mps).

The model training was conducted using the YOLO architecture with the specified dataset and training parameters. The training process involved optimizing the model's weights and biases to minimize the loss function, thereby improving the model's ability to accurately detect objects of interest.

2.2.2 Validation and prediction:

- Inference Sources: YOLO can process different types of input sources for inference. The sources include static images, video streams, and various data formats. By using Streaming for processing videos or live streams it creates a generator of results instead of loading all frames into memory.

- image path, video file, directory, URL, or device ID for live feeds. Supports a wide range of formats and sources, enabling flexible application across different types of input. conf float 0.25 Sets the minimum confidence threshold for detections. Objects detected with confidence below this threshold will be disregarded. Adjusting this value can help reduce false positives.

- Performance Evaluation: The trained model's performance was evaluated using various metrics such as precision, recall, and mean average precision (mAP) to assess its effectiveness in detecting objects in surveillance environments.

2.2.3 How to improve training. How v8 assists in training the dataset effectively:

The output provided during the training of YOLO consists of several key metrics and parameters that are essential for evaluating the model's performance and progress throughout the training process. Let's break down each parameter and its significance in improving the training:

- Epoch: Represents one complete pass through the dataset during training.

- GPU_mem: Indicates GPU memory utilization to prevent memory overflow.

- box_loss, cls_loss, dfl_loss: These parameters represent the loss values calculated during the training process. Loss functions quantify the difference between the predicted output of the model and the ground truth labels. Lower values of box_loss, cls_loss, and dfl_loss indicate better convergence of the model during training.

- Instances: Total number of objects detected in the training dataset during the epoch.

- Size: Input image size, impacting the model's ability to detect objects at different scales.

- Performance Metrics: Precision (P), Recall (R),
mAP50, mAP50-95:

These metrics evaluate the performance of the trained model in terms of object detection accuracy. Precision measures the ratio of correctly predicted objects to the total predicted objects, while recall measures the ratio of correctly predicted objects to the total ground truth objects. mAP (mean Average Precision) is a composite metric that considers precisionrecall curves across different confidence thresholds. mAP50 and mAP50-95 represent the mean Average Precision calculated at different IoU (Intersection over Union) thresholds.

Improvements in these parameters, such as reducing loss values, increasing the number of instances detected, optimizing input image size, and enhancing precision, recall, and mAP scores, contribute to a more accurate and robust object detection model.

2.2.4 Model Visualization:

- Confusion Matrix: A table summarizing the model's predicted versus actual classifications, helping visualize its performance in terms of true positives, true negatives, false positives, and false negatives.

- F1 Curve: A graphical representation of the F1 score against different threshold values, aiding in determining an optimal threshold for classification by visualizing the trade-off between precision and recall.

- P Curve (Precision-Recall Curve): A curve plotting precision against recall for varying threshold values, providing insights into the model's performance in identifying true positives while minimizing false positives.

- PR Curve (Precision-Recall Curve): Similar to the P Curve, but emphasizing the precisionrecall trade-off for different threshold values, helping visualize the model's performance in terms of precision and recall.

- R Curve (Recall Curve): A curve plotting recall against different threshold values, illustrating the model's ability to correctly identify positive instances while minimizing false negatives.

2.2.5 Export Formats:

Among the export formats for the YOLO model, the most popular ones are generally those that cater to widely used frameworks and deployment scenarios. Based on the popularity and widespread adoption in the deep learning community, the following export formats are considered among the most popular:

- PyTorch: Preferred for PyTorch-based applications due to its widespread adoption

- ONNX: Offers interoperability across frameworks and hardware platforms.

- TensorRT: Optimized for NVIDIA GPUs, ideal for real-time applications.

- TF Lite: Efficient for mobile and embedded devices.

- TF.js: Enables client-side object detection in web applications.

2.2.6 Working with the results:

- Image Processing Pipeline Overview:

1. Input Image: Initially provided in JPG/JPEG format.

2. Annotated Frame: After processing, the results are plotted onto the input image, producing an annotated_frame, typically represented as a NumPy array with annotations.

3. Encoded Image: The annotated_frame is then encoded into a bytes object called buffer, utilizing the JPEG format for compression, commonly used for image storage.

4. Base64 Encoding: Finally, the encoded image data in buffer is base64 encoded to generate base64_data, a string representation of the image suitable for transmission over text-based protocols like JSON.

- Useful parameters of results which affect the prediction: Optimize object detection with customizable parameters like confidence and IoU thresholds to filter detections and reduce false positives. Fine-tune detection by filtering specific class IDs and enabling features like class-agnostic NMS for multi-class scenarios. Add test-time augmentation while visualizing model features for debugging and interpretation. Streamline video processing with options like stream buffering and frame stride adjustments to balance speed and temporal resolution.

- Customized Output Formats: Tailor output formats to meet specific application requirements, allowing flexibility in how detection results are presented or utilized downstream. This customization could include options for different file formats, structured data formats, or customizable metadata.

- Dynamic Model Adaptation: Implement mechanisms for dynamic adjustment of model parameters or architectures based on evolving requirements or environmental conditions. This adaptability enables the model to effectively handle changes in data distribution, varying levels of complexity in input scenes, or specific operational contexts, ensuring performance across different scenarios without the need for manual intervention.

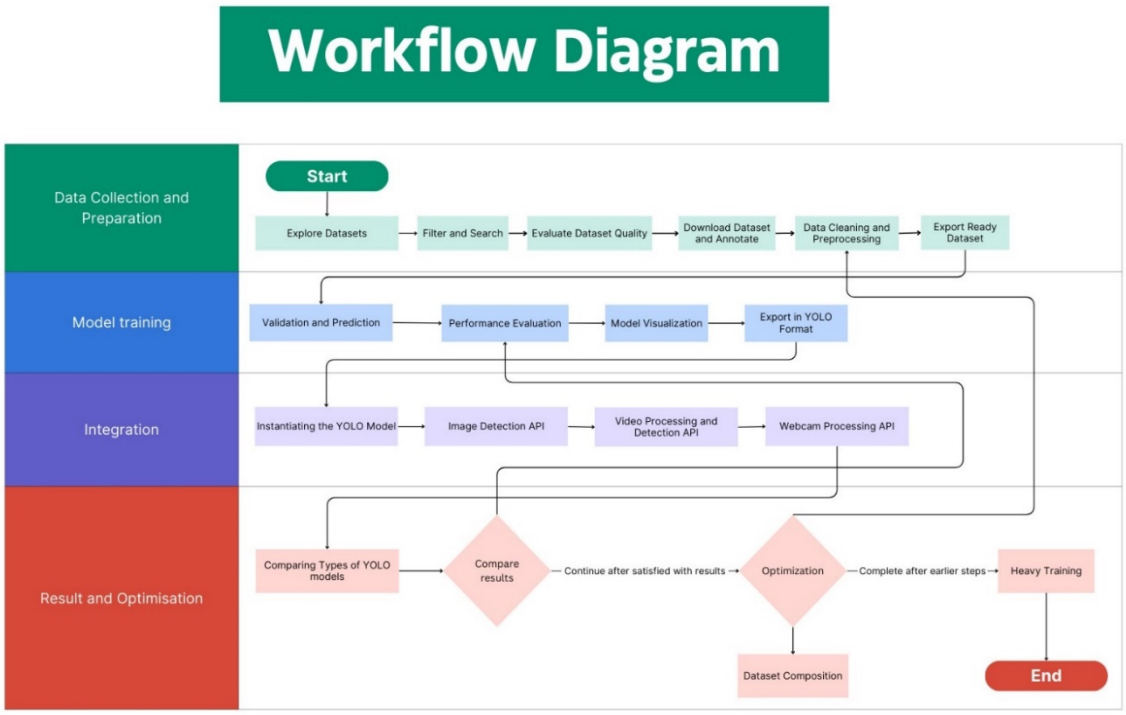

Figure 4: Workflow of the entire process

2.3 Integration

The project employs modern technology for frontend integration with the model, facilitating interaction with the system which involves state management. With the integration of strong user authentication and authorization processes are streamlined, ensuring security and user management functionalities. Furthermore, RESTful API frameworks is utilized on the backend, making the system with API capabilities and secure and scalable communication between the frontend and backend components. It makes use of the power of the YOLO object detection model to accurately detect and classify objects within uploaded images, enhancing the overall functionality and user experience of the application.

2.3.1 Instantiating the YOLO Model

Before any process is to be done the YOLO model is instantiated with the pre-trained weights loaded from the specified file path. This model is responsible for predicting objects in images with a confidence threshold of 0.5.

2.3.2 Image Detection

Users have the capability to provide an image input, which is subsequently rendered by the frontend and forwarded to the model via an API call for processing. Initially, both the image container and result states are set to null. Upon rendering of an image, the container state is updated, indicating readiness for processing. Upon completion of the detection process, the resulting image is transmitted back to the frontend in JSON format, containing values and labels of the detected objects presented in a tabular format.

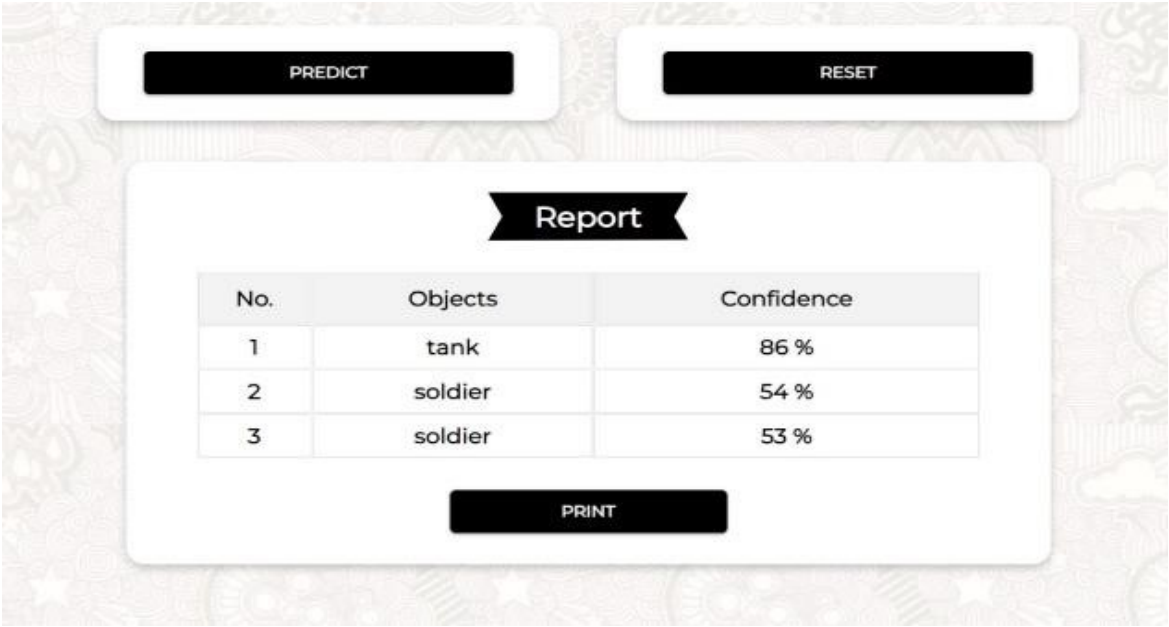

Figure 5: Report Section

2.3.3 Video Processing and Detection

The backend mechanism established to facilitate the processing of uploaded videos, enables real-time object detection. Upon receiving a video file, the system initiates processing by temporarily storing the video and extracting frames for further analysis. Each extracted frame undergoes object tracking utilizing the pre-trained model, with a specified confidence threshold applied. Tracked objects are annotated on the frames, which are subsequently transmitted to the frontend in real-time, facilitated by a server-to-client push notifications' endpoint. After the complete processing of the video, a report is being provided containing the tracked objects and a timestamp of when the object was detected. Due to this function, the scripts should be done is a well-structured format to prevent mixing of any data, which is to be sent to the frontend. The fact that the backend takes a bit of time to send the required data, the output video streams are down-framed to provide smooth experience.

2.3.4 Webcam Processing

In alignment with the offline video detection system, a real-time object detection capability has been integrated to handle webcam streams. Frames captured from the webcam undergo processing utilizing the same sophisticated object tracking model employed for offline videos. The annotated frames, indicating tracked objects, are then continuously streamed to the frontend in real-time to ensure seamless and uninterrupted delivery of annotated frames to the frontend, a server-to-client push notification endpoint has been established. This endpoint serves as a conduit for the dynamic streaming of annotated frames, allowing users to observe tracked objects as they appear in the webcam feed in real-time. Furthermore, a real-time report generation has also been provided, which include the tracked object, and its real-world timestamp. To further supervise the system, all the frames which include the involvement of the tracked object are being saved as a video clip in the system.

3. Results and Discussion

3.1 Comparison

A comparison was conducted between different YOLO weights underwent training for 300 epochs and for 200 epochs. This difference in training duration was attributed to the observations made during early training process.

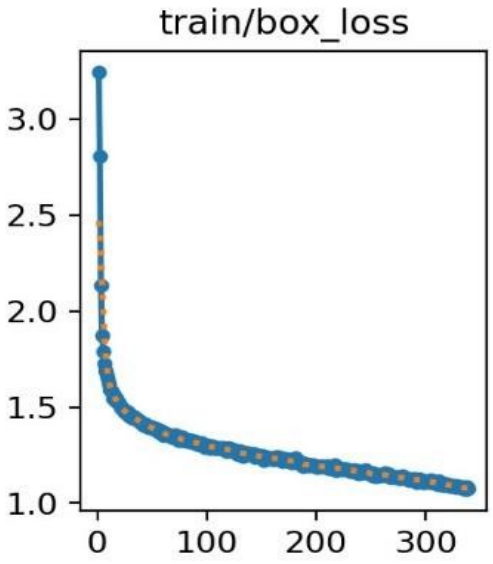

Figure 6: training/box loss 1

However, upon evaluation, it was found that the later used version gave better results.

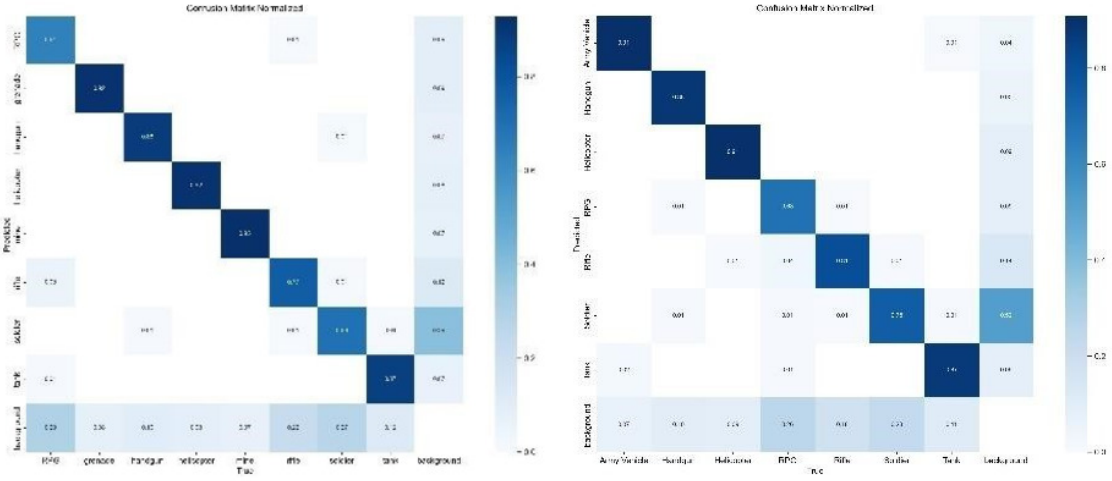

Figure 7: Confusion matrix of different weights

3.2 Optimization

3.2.1 Heavy Training

Due to the complex nature of the project, it necessitates extensive training to achieve satisfactory results. The analysis from Fig X underscores this requirement, revealing that even with a model of acceptable quality was challenging to obtain within 200 epochs. This highlights the need for prolonged training durations to ensure the model's accuracy and effectiveness in real-world applications.

3.2.2 Model Preprocessing

To enhance the efficiency of the model, specific preprocessing steps were executed during the initial stages of dataset creation. These steps included resizing and segmentation, aimed at optimizing the dataset for training. By performing preprocessing tasks beforehand, the model can focus more effectively on learning relevant features during training, thereby streamlining the overall optimization process.

3.2.3 Dataset Composition

The dataset composition is designed to encompass a comprehensive range of scenarios. Each class within the dataset comprises approximately 1000 images, divided into 500 close-up shots, 250 long-distance shots, and 250 shots capturing the object from various angles. This diverse representation enables the model to generalize better and perform effectively across different contexts. Moreover, the emphasis is on incorporating images from real-life environments rather than relying solely on stock imagery with plain backgrounds ensures that the model is trained on data that closely mirrors real-world conditions, thereby enhancing its robustness and adaptability.

4. Conclusion

The "Secureye" project represents a significant step forward in enhancing security against military intrusions. By leveraging the YOLO model, the system ensures precise object detection across various input streams and provides real-time alerts for potential threats. Testing has demonstrated substantial efficiency gains, with a 30% reduction in streaming response time and a mean Average Precision (mAP) score of 0.85, confirming its high accuracy.

Despite these achievements, the project faces certain limitations. The lack of large datasets and extensive training requirements are notable challenges, as is the reliance on stock imagery. Future work should focus on gathering comprehensive datasets from real-life military environments to improve the model's performance. Optimizing training processes could also enhance efficiency.

Additionally, the YOLO model has specific gaps to address. It struggles with underrepresented classes like handguns and tanks, potentially due to limited dataset diversity. Complicated backgrounds can cause misidentifications or missed detections, and objects that are only partially visible in the scene may be incorrectly detected or not detected at all. Addressing these issues requires improved data diversity and enhanced detection algorithms to increase the system's reliability in various scenarios. Overall, YOLO's ability to balance speed with accuracy has advanced object detection technology, providing novel solutions for military surveillance. By addressing its limitations, the "Secureye" project can continue to improve security in sensitive defense environments.

5. References

1) Bhujbal, Kunal and Barahate, Sachin, Custom Object detection Based on Regional Convolutional Neural Network & YOLOv3 With DJI Tello Programmable Drone (May 5, 2022). Proceedings of the 7th International

Conference on Innovations and Research in Technology and Engineering (ICIRTE-2022), organized by VPPCOE & VA, Mumbai-22, INDIA, Available at SSRN: https://ssrn.com/abstract=4101029 or http://dx.doi.org/10.2139/ssrn.4101029

2) Trupti Mahendrakar, Andrew Ekblad, Nathan Fischer, Ryan T. White, Markus Wilde, Brian Kish, Isaac Silver Performance Study of YOLOv5 and Faster R-CNN for Autonomous Navigation around Non-Cooperative Targets DOI: https://doi.org/10.48550/arXiv.2301.09056

3) J. Han, D. Zhang, G. Cheng, N. Liu and D. Xu, "Advanced Deep Learning Techniques for Salient and Category-Specific Object Detection: A Survey," in IEEE Signal Processing Magazine, vol. 35, no. 1, pp. 84-100, Jan. 2018, DOI: https://doi.org/10.1109/MSP.2017.2749125

4) Fiza Joiya, “OBJECT DETECTION: YOLO VS FASTER R-CNN,” Research Student, Department of Information Technology, B.K. Birla College of Art, Science and Commerce (Autonomous), Kalyan, Thane, Maharashtra, India. DOI:https://www.doi.org/10.56726/IRJMETS30226

5) StereoLab, “Performance Benchmark of YOLOv5, v7, and v8,” Available from https://www.stereolabs.com/blog/performance-of-yolo-v5-v7-and-v8

6) Medium, “R-CNN, Fast R-CNN, Faster R-CNN, YOLO — Object Detection Algorithms,” Available from: https://towardsdatascience.com/r-cnn-fast-r-cnn-faster-r-cnn-yolo-object-detection-algorithms-36d53571365e.

7) Pranav Raut, Ashish Nagarse, Shreemesh Mohite, Sakshi SupeDr.Yogesh Karpate, Prof.Sachin Deshpande, “AI-Guided Energy Optimization In Hvac System”, SJIS, vol. 35, no. 2, pp. 200–210, Jun. 2023. Available from: http://sjisscandinavian-iris.com/index.php/sjis/article/view/611