City, University of London MSc in Data Science

Project report

2023

Real-time Gun Detection System

Nizar Assad

Supervised by: Giannopoulos Panos

By submitting this work, I declare that this work is entirely my own except those parts duly identified and referenced in my submission. It complies with any specified word limits and the requirements and regulations detailed in the assessment instructions and any other relevant programme and module documentation. In submitting this work, I acknowledge that I have read and understood the regulations and code regarding academic misconduct, including that relating to plagiarism, as specified in the Programme Handbook. I also acknowledge that this work will be subject to a variety of checks for academic misconduct.

Signed: Nizar Assad

Abstract

Ensuring effective gun detection is vital in contemporary times to address escalating concerns about public safety and combat the increasing occurrences of crimes. This project addresses the escalating challenge of gun violence through the development of an advanced gun detection system. Leveraging deep learning models, the objective is to surpass existing detection capabilities. The motivation stems from the critical need for enhanced security measures to mitigate the impact of gun-related incidents. The beneficiaries include the public, law enforcement agencies, and security infrastructure stakeholders. Performance evaluation involves metrics such as F1 score and precision, with real-life scenario tests simulating CCTV setups. Continuous adaptation and refinement of models and datasets are integral to achieving optimal performance. In this study we leverage the speed and accuracy of Yolo models and the adaptability of VGG16 to create an efficient model for inference, we have achieved a precision of 91% using Yolov8s, and 89% using yolov5s.

Keywords: **weapon detection; gun detection; computer vision; deep learning; building automation; terrorism**

Table of Contents

Introduction and Objectives

Background to the Problem

Gun violence remains a persistent societal menace, demanding progressive measures for timely detection and prevention (Cook, 2000). This project addresses this pressing issue by developing an advanced gun detection system. In the contemporary landscape, effective gun detection is crucial due to heightened concerns about public safety and the escalating threat of gun violence. The urgency is emphasized by the alarming increase in firearm incidents globally, resulting in over 250,000 fatalities in 2019 alone (Harvard Kennedy School, 2000). Homicides constitute a significant portion, accounting for nearly 71% of these gun-related deaths. Mass shootings and school shootings, which have garnered substantial public attention, contribute to this unfortunate trend. Notably, gun-related violence extends beyond national legal frameworks, impacting even countries where firearms are prohibited (Firearms Injuries Higher in Gentrified Neighbourhoods, 2023).

The complex nature of gun violence necessitates a nuanced understanding of its contributing factors. Socioeconomic disparities, inadequate mental health resources, and the proliferation of illegal firearms contribute to the persistence of this issue (The Effects of Firearm Safety Training Requirements, 2020). Additionally, the dynamics of violence have evolved with the rise of extremism and domestic terrorism, further complicating efforts to limit firearm-related incidents (Gun Violence, Prevention of, 2023).

To mitigate these multifaceted threats, a holistic approach has been adopted, integrating advanced technologies such as classical machine learning and deep learning methods for weapon detection (Cook, 2000). The focus on gun detection in sensitive environments like schools has led to the integration of artificial intelligence with surveillance systems to proactively safeguard campuses (Deep multi-level feature pyramids: application for non-canonical, 2021). The primary objective of this project is to surpass existing detection capabilities by fine-tuning and optimizing selected models, emphasizing a comparative analysis of their efficiency in gun detection under diverse indoor and outdoor conditions (Cook, 2000).

As society grapples with the multifaceted challenge of gun violence, the proposed gun detection system aims to be at the forefront of technological interventions, contributing significantly to collective efforts to ensure public safety and prevent firearm-related incidents.

Choice of Project and Beneficiaries

The rationale behind selecting this project is firmly grounded in the imperative need for heightened security measures to effectively address and mitigate the escalating challenges posed by gun-related incidents. More than 600 people die every day because of firearms violence (World Population Review, 2023). The urgency stems from the realization that traditional security systems often fall short in swiftly identifying and responding to instances involving firearms (World Population Review, 2023).

For example, in the security industry, it can be problematic to find a handgun among a crowd of people, an object that may only make up 3% of the image. This is because precision in recognizing small objects is crucial. As part of the VICTORY1 (Enríquez et al., 2019; Salazar et al., 2019) project, this study will analyse the case. The goal of the project is to notify security personnel as soon as a

potential threat is detected; therefore, the detector needs to have the right speed in order to adapt the weapon detection alert to a real-time CCTV.

The primary beneficiaries of this project are diverse. Foremost among them is the general outdoor environment, whose safety is intricately tied to the efficacy of security systems (Safeguarding Our Public Spaces, 2020). By deploying an advanced gun detection system, the project aspires to create safer environments, providing individuals with a heightened sense of security in public spaces (Brennan Centre for Justice, 2023).

Law enforcement agencies represent another main beneficiary (National Sheriffs' Association, 2023). The implementation of a robust gun detection system equips these agencies with an additional layer of technological support (National Sheriffs' Association, 2023). This not only enhances their capacity to respond promptly to potential threats but also aids in forensic investigations by providing accurate and timely data on firearm-related incidents (United States Department of Justice, 2023).

Moreover, security infrastructure stakeholders, encompassing entities responsible for safeguarding critical facilities and public spaces, stand to benefit significantly (Security Industry Association, 2022). By incorporating advanced technologies, these stakeholders can fortify their security protocols, creating more resilient and effective defence mechanisms against potential security breaches involving firearms (ASIS International, 2020).

In summary, the choice of this project is underpinned by a commitment to addressing the pressing need for advanced security solutions (International Association of Chiefs of Police, 2023). The beneficiaries, including the public, law enforcement agencies, and security infrastructure stakeholders, stand to gain substantially from the implementation of a cutting-edge gun detection system (The National Academies of Sciences, Engineering, and Medicine, 2023). The envisioned impact extends beyond individual safety, contributing to broader societal goals of public safety enhancement and proactive crime prevention (United Nations Office on Drugs and Crime, no date).

Objectives and Evaluation

Objectives:

In its first year, this project has set forth an ambitious yet crucial primary objective, to surpass existing gun detection models, aiming to create a more accurate system that contributes to crime control and enhances security, particularly in public spaces. The project focuses on fine-tuning models through experimentation using various deep-learning techniques. Beyond model refinement, the project places significant emphasis on dataset quality, recognizing its pivotal role in achieving a robust gun detection system capable of adapting to diverse real-world scenarios. To address this, different data augmentation techniques are applied, ensuring correct augmentations, and leveraging Roboflow for assessment, a user-friendly platform. Continuous improvement of the dataset occurs through conclusions drawn after each training session. The key objectives can be summarized as follows:

- Surpass existing gun detection models for improved accuracy.
- Fine-tune models through experimentation with hyperparameters and find the best combinations.
- Prioritize the importance of decreasing false positive predictions and increasing precision.
- Develop a versatile model capable of adapting to diverse situations and functioning effectively across various environments.
- Analyse the results and come up with keys of improvement for future work.

Evaluation:

Model Evaluation Metrics Analysis: Performance evaluation involves a comprehensive assessment utilizing key metrics:

- **Recall:** Assesses the model's ability to capture all relevant instances within the dataset. It emphasizes the importance of minimizing false negatives.
- **F1 Score:** Strikes a balance between precision and recall. It provides a single value that reflects the model's overall performance by considering both false positives and false negatives.
- **mAP:** Considers precision at various recall levels, offering a nuanced understanding of how well the model performs across different thresholds. This metric is particularly valuable when dealing with tasks where certain instances are more critical than others.
- **Precision:** Focuses on the accuracy of positive predictions. It helps in evaluating the model's ability to avoid false positives, which is crucial in scenarios where misclassification carries significant consequences.

These metrics not only quantitatively measure model effectiveness but also enable a nuanced comparison of selected models using test data. The focus on these metrics ensures a thorough understanding of the models' performance.

Robustness Assessment with Unseen Data: To evaluate the generalization capabilities of the models, a dedicated folder was created. This folder contains images representing expected scenarios, encompassing different pixel values and angles. Generalization, particularly in our use case, is a top priority. This step ensures that the models are robust and can handle diverse situations encountered in real-world applications.

Real-time Object Detection Testing: A distinctive aspect of the evaluation involves subjecting the models to real-life scenario tests using a webcam, simulating a CCTV setup. This practical assessment aims to evaluate the models in real-world environments, providing insights into their performance under dynamic conditions. This step is crucial for assessing the models' applicability in scenarios such as crime scene investigations, where real-time object detection is essential.

Comparing results with other research: After evaluating the performance of the developed models, a final comparative analysis was conducted to assess their superiority compared to other models from different projects. This comparison provides valuable insights into the advancements made and highlights the potential impact of the developed models in the field of gun detection.

Methods and Work Plan

Methods:

For the sake of this project, we're going to use deep learning algorithms to extract features automatically and to self-train hierarchically. The best-fitted algorithms were chosen in this study, to achieve our last goal of creating an efficient system of detecting guns and alerting the authorities in real-time, the selected model needs to process data in a very fast way, which is very crucial and required for our program to run like expected. The starting point for designing the model is to do a requirements analysis. For this, we went through publicly available CCTV recordings to analyse the data and see what aspects we should work on to improve the detection process, we also based our

work on two other research that we are planning to use as models of comparison by the end of the study. The main points we want to focus on:

- Poor pixel quality, low resolution.
- Object is visible only for a limited period in a scene.
- Guns have a different shape following the angles from which it is captured.
- When captured from a far distance, guns might look like some other object.

After analysing the data, we have listed a set of requirements that the model should respect.

- The trained model should be of high accuracy to face low-resolution situations.
- The algorithm should detect objects in real-time.
- The model should keep the number of false alarms as low as possible.
- The model needs to be trained equally with data-capturing guns from various angles.

Our search will be more based on Yolo models, as we will be experimenting with the use of 3 different Yolo models, as shown in Figure 1 The YOLO model partitions the input image into M*N grids, conducting end-to-end learning for each grid cell. In this stage, treating the bounding box and class probability within the image as singular regression problems is crucial [16]. This involves examining the entire image at once and calculating class probabilities for various bounding boxes using a single convolutional network. Consequently, the model promptly estimates the types and location information of the objects of interest through this process [17]. In the model learning process, the learned image is divided into the M*N grid cells, and the confidence score of each cell is processed in parallel to efficiently calculate the confidence score of a particular class [18]. Through this series of prediction processes, the objects of interest to be detected by the involved model are expressed on the resulting image according to the class.

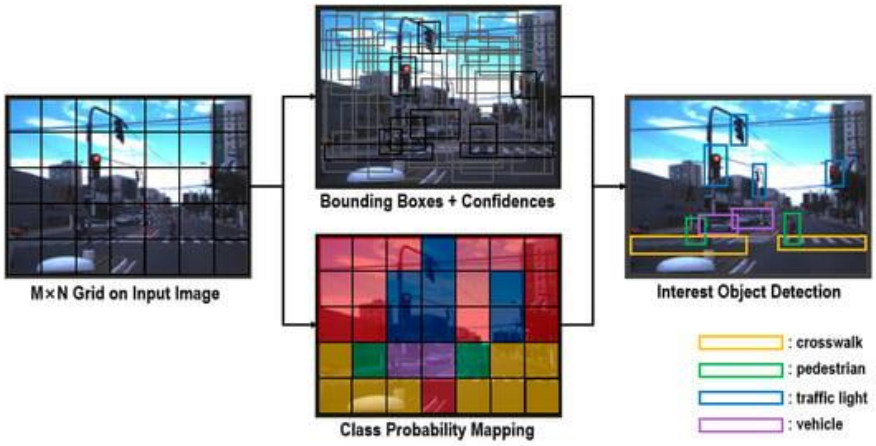


Figure 1 Yolo models detection process

The implementation of this project involves integrating cutting-edge deep learning models: YOLOv8s, YOLOv5s, YOLOv7, and a custom VGG16 architecture, selected for their proven performance in real-time object detection. YOLOv8s stands out for its unparalleled efficiency in swiftly identifying objects (Wang et al. 2021), while YOLOv5s demonstrates competitive accuracy and speed, coupled with user-friendly features (Bochkovskiy 2021). The decision to incorporate a custom VGG16 architecture underscores our commitment to tailored solutions for enhanced accuracy.

The implementation process follows a meticulous approach, incorporating diverse hyperparameters and architectural modifications. The primary focus is on YOLOv8s due to its exceptional promise [4]. The research findings will provide valuable insights into the effectiveness of using YOLOv8 and other deep-learning techniques for automatic weapon detection systems. The chart below outlines the improvement in the performance of the YOLOv8 when compared to all the previous versions of the

COCO dataset, and this tells us about the greater significance of the highest version while applying the YOLO architecture to a particular dataset.

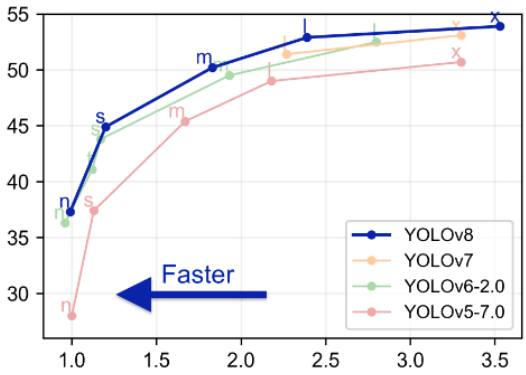


Figure 2 Yolo models speed comparison

The work plan involves continuous refinement facilitated by ClearML, an open-source platform for streamlined machine learning development and management, ensuring real-time monitoring and comparison of multiple YOLOv5s and YOLOv8s training sessions. This iterative process finely tunes the models for optimal performance.

Integral to the project is the refinement of datasets, combining free-of-use academic datasets, conducting additional cleaning, and including images from various sources to create a more generalized algorithm. The utilization of Roboflow for annotation, augmentation, and fine-tuning enhances the project's methodological synergy. It also provides a simple and comprehensive interface, facilitating the exploration of diverse data augmentation techniques for better results. Roboflow is an even better choice when using multiple models, as it takes care of the data exporting parameters and adapts the training images corresponding to the input expected from the model. This comprehensive approach establishes a harmonious interplay between models and datasets, aiming for a gun detection system excelling in accuracy and adaptability. Assessing and enhancing the training data quality will be applied after each training session to keep perfecting the data.

Work Plan:

Model Integration:

- Integrate YOLOV8s, YOLOv5s, YOLOv7, and custom VGG16 architecture.
- Implement a custom VGG16 architecture for tailored solutions and enhanced accuracy.
- Incorporate methodologies from multiple models to gain a comprehensive understanding of strengths and limitations.

Model Implementation and Focus:

- Incorporating diverse hyperparameters and architectural modifications.
- Focus primarily on YOLOv8s and yolov5s.
- Utilize ClearML for real-time monitoring and comparison of Yolo models training sessions.
- Continuous hyperparameter optimization.
- Ensure models are finely tuned for optimal performance.

Dataset Refinement:

- Combine free-of-use academic datasets for comprehensive coverage.
- Conduct additional cleaning to enhance dataset quality.
- Include images from various sources capturing guns from different angles with different pixel resolutions.
- Leverage Roboflow for annotation, augmentation, and fine-tuning.
- Establish a methodological synergy between models and datasets.

Model Performance Evaluation:

- Evaluate model performance results on the validation and test sets.
- Assess models using unseen data from other sources to gauge generalization.
- Explore the models versatility in handling various object detection scenarios.
- Test models in a real-time object detection scenario through a CCTV and use publicly available clips.
- Compare the best-performing models to models from another selected research.

Ultimately, the process outlines a comprehensive roadmap that extends from model development and training to testing, and analysis. The objectives are strategically aligned to ensure that the finalized model not only surpasses existing benchmarks but also meets the specific needs of real-world applications, particularly in the context of crime scene investigations and public safety enhancement.

Structure of the Report

The report is thoughtfully structured to guide the reader through a comprehensive understanding of the gun detection project. Each chapter serves a distinct purpose, starting with background and objectives, setting the stage for the project's significance and overarching goals. The methods chapter delves into the intricacies of model implementation, presenting detailed architectures, hyperparameter choices, and the methodology behind dataset refinement. The results chapter encapsulates the outcomes of the project, providing a nuanced evaluation of the developed models. The final chapter concludes the report by comparing the project's achievements with existing research, offering insights into the superiority of the developed models in gun detection.

Changes and Adaptations

Flexibility and adaptability are at the core of this project. Throughout its duration, continuous adaptation and refinement of models and datasets have been paramount. Many parameters, such as the learning rate and the number of epochs, were crucial for accurate training. Any significant changes in goals or methodologies are transparently outlined in the subsequent chapters, providing a clear narrative of the project's evolution. This dynamic approach ensures that the project remains responsive to emerging challenges and opportunities, fostering an environment conducive to optimal performance. This introduction lays the groundwork for an in-depth exploration of object detection, emphasizing not only the critical need for advanced models but also the adaptive methodologies essential for enhancing public safety.

Context

Gun violence is a pervasive and escalating issue that poses significant challenges to public safety globally. The current state of the world is marred by the alarming rise in firearm incidents, resulting in substantial fatalities and impacting societies on various fronts. This chapter provides a comprehensive overview of the existing landscape, both in practice and theory, surrounding gun violence, gun detection technologies, and the societal implications of firearm-related incidents.

Current State of Gun Violence:

The prevalence of gun violence remains a global concern, with over 250,000 fatalities recorded in 2019 alone. Homicides constitute a substantial portion, accounting for approximately 71% of gun-related deaths (Pew Research, 2023). Mass shootings and school shootings, gaining substantial public attention, contribute significantly to this unfortunate trend, emphasizing the need for comprehensive strategies (Pew Research, 2023).

The urgency to address gun violence is further underscored by the daily toll, with more than 600 people dying every day due to firearms violence (Pew Research, 2023[1]). Traditional security systems often fall short in swiftly identifying and responding to instances involving firearms, necessitating innovative solutions to enhance public safety (Axon, 2023).

Efforts to address this multifaceted issue require a collaborative approach, involving policymakers, law enforcement agencies, mental health professionals, and community leaders (AAFP, 2023). Comprehensive gun control measures, coupled with educational initiatives, mental health support, and community outreach programs, can contribute to a more holistic strategy aimed at curbing the prevalence of gun violence (AAFP, 2023).

In conclusion, the current state of gun violence demands immediate attention and concerted efforts to implement effective measures that go beyond traditional approaches. By acknowledging the global scale of the issue and embracing innovative solutions, societies can work towards creating safer environments for individuals and communities alike.

Existing Measures and Technologies:

In the pursuit of advancing real-time gun detection within Closed-Circuit Television (CCTV) scenarios, several studies have been undertaken, a project led by JLS González comprehensively explores high-level approaches utilizing classical machine learning and deep learning methods (González, 2020). This initiative specifically delves into the intricacies of adapting weapons detection to real CCTV scenarios, emphasizing the crucial role of the object's distance from the camera and advocating for technological advancements to enhance performance. Marbach et al. proposed an automatic fire detection system based on the temporal variation in fire intensity (Marbach, 2022). As a component of the UK-based MEDUSA project, Darker et al. first proposed the idea of automated gun crime detection. Additionally, this group practiced identifying clues that could point to the presence of a firearm being concealed (Darker, 2021).

Another significant attempt, a comprehensive study undertaken by researchers, employs deep learning methods to examine various models for identifying weaponry, emphasizing the dynamic evolution of these models over time 20. This study grapples with the challenges associated with the dynamic nature of weapon detection, providing insights into the continuous evolution of models to address emerging challenges in this domain (Yadav, P. 2022). Additionally, a study led by MT Bhatti conducts an exhaustive experimental analysis of a gun detector, with a particular focus on the impact of synthetic datasets on training. This research scrutinizes the influence of synthetic datasets on training and their subsequent impact on detection performance, emphasizing the pivotal role of dataset choice in achieving effective real-time gun detection. Moreover, this study addressing real-

time abnormal object detection, particularly guns and knives in video/images, accentuates the significance of consistency in achieving effective detection across diverse scenarios (MT Bhatti, 2022). These collective efforts, led by these researchers, significantly contribute to a nuanced understanding of the complexities, advancements, and challenges in the realm of real-time gun detection.

Societal Implications and Stakeholders:

The consequences of gun violence extend beyond individual incidents, impacting broader societal goals of public safety enhancement and proactive crime prevention. The proposed gun detection system aims to be at the forefront of technological interventions, contributing significantly to collective efforts to mitigate the threats posed by firearms.

The primary beneficiaries of this project are diverse and include the public, law enforcement agencies, and security infrastructure stakeholders. Public spaces, where safety is intricately tied to the efficacy of security systems, stand to benefit from the deployment of an advanced gun detection system. Law enforcement agencies gain an additional layer of technological support for prompt response to potential threats and enhanced forensic investigations. Security infrastructure stakeholders, responsible for safeguarding critical facilities and public spaces, can fortify their security protocols and create more resilient defence mechanisms.

Conclusion:

In summary, the context chapter provides a comprehensive understanding of the current state of gun violence, existing technologies, and the societal implications of firearm-related incidents. The urgency to address this issue is evident, and the proposed project positions itself as a critical intervention to advance the capabilities of gun detection systems. The subsequent chapters will delve into the specifics of the project, building on the foundation laid in this contextual exploration.

Methods

Introduction

The primary objective of this project is to develop an advanced multi-angle detection model capable of addressing the challenges posed by diverse scenarios. In this section, we will overview several different architectures, try to apply model and data optimization like hyperparameter tuning or data augmentation, we will explore in more details about VGG16 custom model, yolov5s, yolov7 and yolov8.

Dataset Selection, Augmentation, and refinement

In this study, particular emphasis was placed on diversifying the angles within the dataset. Some images were captured from considerable distances, representing various settings such as airports, schools, or outdoor environments, these goals were successfully obtained by research made by Jose L. Salazar González, his research has focused on several outdoor places, focusing on varying the angles, he has created a free academic use dataset named Mock attack dataset, there are various benefits associated with including this dataset for our real-time item detection system. First, to ensure ethical concerns, it was carefully gathered and annotated during a simulated attack, with express approvals secured from the University and security professionals. The collection includes a

variety of situations that were recorded by three security cameras that were positioned at key points throughout hallways and an entryway. Each camera (Cam1, Cam7, and Cam5) provides unique challenges for object detection algorithms, such as conflicting objects, varying lighting conditions, and distinct environments. For instance, Cam7 introduces additional complexities with more conflicting objects like fire extinguishers. The dataset offers a comprehensive representation of real-world scenarios, enhancing the robustness of the object detection model (González, 2020). This dataset will go through further optimization alongside the other data we will gather. A second dataset, authorized for academic use and publicly available in Roboflow universe was employed. The two datasets were closely analysed, and the selected sections were then included, the merge option provided by Roboflow makes this step easy and one click away, it allows to mix the project in order to create one single dataset, it also takes care of the resizing. The focus will be more towards handguns, handguns are one of the most common types of firearms used in shooting homicides, accounting for 57% of such cases in 2021. In Brazil, the United States, Venezuela, Mexico, India, and Colombia, two-thirds of gun-related deaths, including suicides, occur. In Europe, the distribution was more equal between pistols, rifles, and shotguns. In Canada, handguns are the most common type of firearm used in shooting homicides, accounting for 57% of such cases in 2021 (WPR ,2023). A thorough data exploration revealed certain angles not covered by the datasets. Consequently, additional images gathered from the internet and YouTube videos were incorporated to enhance model generalization, the gathered images constitute 50 per cent of the full dataset. The dataset encompassed a broad spectrum of pixel resolutions to facilitate the model's adaptability to various CCTV types, encompassing both contemporary and older models. Notably, the initial datasets contained extraneous images, including zoomed-in focused images of firearms, which were deemed irrelevant for training purposes. The image below shows a perfect example of the images that need to be excluded from the data.

Figure 1 Example of bad dataset

Developing a model for gun detection posed inherent challenges due to the varying shapes of firearms based on viewing angles. The images below are an example of what the dataset looks like, by balancing the presence of guns captured from different angles.

Figure 2 Annotated images from the MOCK dataset.

The dataset underwent preprocessing steps, including auto-orientation, and resizing to a standard size corresponding to each model. Augmentations were applied, generating three outputs per training example. Bounding box augmentations were employed, such as brightness adjustments between -7% and +7%, exposure changes between -17% and +17%, and noise addition up to 5% of pixels. These augmentations contribute to the robustness and generalization capabilities of the model. The images, which depict weapons, were collected from diverse sources like Google, YouTube, and Instagram, and annotations were performed. The dataset was divided into training, validation, and test sets, with a 70%, 15%, and 15% split, respectively. The comprehensive set of augmentations enhances the model's ability to handle variations in lighting conditions and improves overall performance. The Roboflow computer vision development platform facilitated dataset integration and YOLO model compatibility. The dataset needs to be very accurate given the potential limitations of detecting guns in real-time situations. This comprehensive approach ensures the dataset's richness in diverse scenarios, contributing to the efficacy of the gun detection model under consideration.

To refine the datasets, a rigorous process was implemented to filter out irrelevant images, such as detailed depictions of guns or drawings. The focus shifted toward images featuring individuals holding guns in outdoor environments, recognizing the challenges posed by the dynamic shapes of guns at different angles.

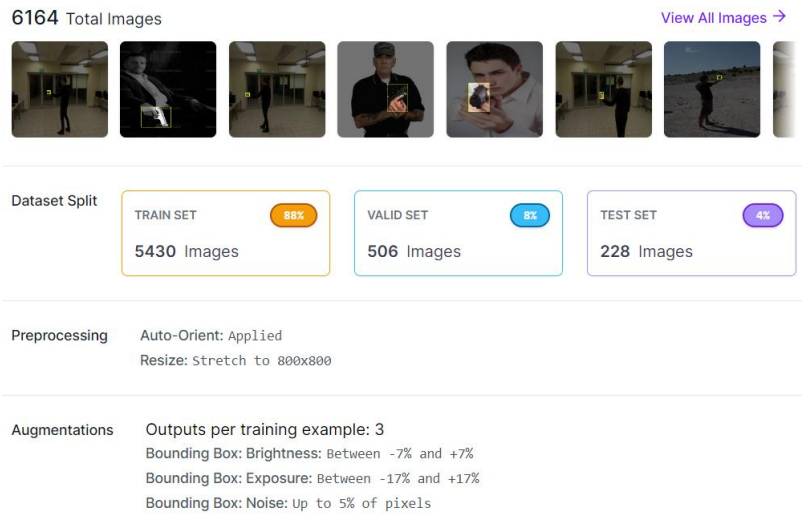


Figure 4 Processing steps applied through Roboflow.

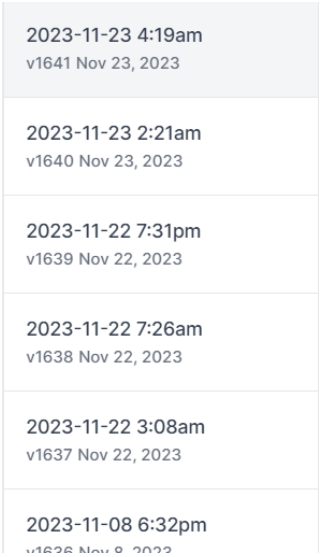


Figure 5 Dataset exporting versions.

In this work, we can investigate various combinations of augmentation, different preprocessing, and data splitting processes more quickly and efficiently thanks to the user-friendly interface of Figure 4, which depicts the process of processing the data directly through Roboflow. Figure 5 illustrates the creation of multiple versions of the dataset, which increases the likelihood of obtaining an optimally optimized dataset. Additionally, Roboflow facilitates scaling and streamlines the process of adapting the dataset between models. With just a few clicks, we may export and utilize our dataset locally or via cloud services like Google Collaboratory.

Models Architecture and Hyperparameter Exploration:

In a departure from conventional practices, this project conducted an exhaustive exploration of hyperparameters and model architectures. This involved experimenting with different learning rates, introducing additional layers in the bounding box head, and exploring variations in model architectures. The objective is not just to meet but to exceed previous research benchmarks and cultivate a highly accurate and adaptable model. We will now delve into the more technical details of the four models experimented on during our research.

VGG16 Custom Model

The custom VGG16 model has been intricately designed to excel in the challenging task of gun detection across diverse spaces through CCTV footage. Retaining the foundational VGG16 architecture, the model employs convolutional layers and max pooling for hierarchical feature extraction. The bounding box head is extended with additional Dense layers, tailored for enhanced feature extraction and localization accuracy, addressing the intricacies of object detection in varied environments. The initial convolutional layers in Block1 capture basic visual patterns, while Block2 builds upon them, extracting more complex spatial representations crucial for identifying object shapes. Max-pooling layers in Block1 and Block2 reduce spatial dimensions, emphasizing essential features. The inclusion of Dense layers, featuring varying activation functions like ReLU, enhances the model's adaptability. These layers introduce non-linearities, enabling the model to discern intricate patterns within CCTV footage, crucial for accurate gun detection. The model's adaptability to diverse characteristics is empowered by the non-linear transformations, aligning it with the challenges posed by varying spaces and making it well-suited for robust gun detection in real-world scenarios.

YOLOv5s:

From what can be visualized from the figure below, Yolov5 has four primary sections in its architecture: input, backbone, PANet, and output. The input terminal is mostly used for data pre-processing, such as mosaic data augmentation and adaptive image filling. Yolov5 incorporates adaptive anchor frame computation on the input to adapt to diverse datasets, allowing it to automatically determine the starting anchor frame size as the dataset changes. Multiple convolution and pooling are used by the backbone network to extract feature maps of various sizes from the input image using a cross-stage partial network (CSP) and spatial pyramid pooling. BottleneckCSP is used to minimize the amount of calculation and increase the inference speed, whereas the SPP structure realizes feature extraction from various scales for the same feature map, generating three-scale feature maps, and optimizing detection accuracy (medium,2021). Strong localization features are conveyed from lower feature maps to higher feature maps using the PAN structure (medium, 2021). This structure increases the detection capabilities by combining the features gathered from multiple network layers in Backbone fusion. The head output is mostly used to anticipate targets of various sizes on feature maps as a final detection step (Fang, 2021).

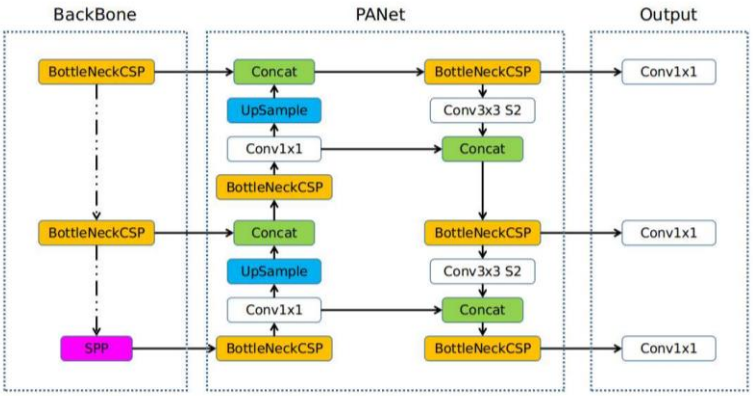


Figure 5 Yolov5 architecture (Ultralytics, 2021)

Once the Yolov5 has been released, there has been five different model sizes, the best fitted one is the Yolov5s which the smallest and has proven to be the fastest algorithm in terms of processing images. As we can deduct from the diagram below:

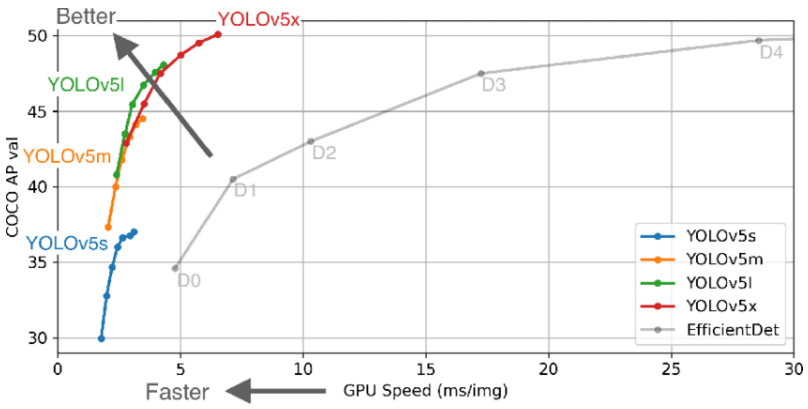


Figure 6 yolov5 models speed statistics (Ultralytics,2021)

YOLOv5s is a state-of-the-art object detection algorithm that has been widely adopted in various applications such as healthcare, security surveillance, and self-driving cars. It was created by Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi in 2015 and was pre-trained on the COCO dataset. YOLOv5s uses a single neural network to process an entire image, which is divided into regions, and the algorithm predicts probabilities and bounding boxes for each region (Ultralytics,2021).

The architecture of YOLOv5s is designed to optimize both speed and accuracy. It features a streamlined design with a single neural network, which contributes to its computational efficiency. The bounding box head incorporates anchor boxes and focal loss, which helps in precise localization. YOLOv5s aligns with the project's goal of achieving superior performance while maintaining computational efficiency.

Prior research has shown that YOLOv5s is highly efficient and effective in real-time object detection. It has been successfully implemented in similar projects, which further supports its choice for this project.

Hyperparameter exploration is a method of hyperparameter optimization that involves experimenting with different values for hyperparameters and evaluating the performance of the model under each configuration. It can be time-consuming and resource-intensive, but it is essential for achieving the best possible performance from object detection.

Ultralytics YOLOv5 provides a guide on hyperparameter evolution, which is a method of hyperparameter optimization using a genetic algorithm (GA) for optimization. The guide explains how to initialize hyperparameters, how to evolve them, and how to evaluate the results. The guide also provides a list of hyperparameters used in YOLOv5s and their default values.

YOLOv7:

The YOLOv7 architecture consists of three main components: the backbone, the neck, and the head. The backbone is used to extract features from the input image, the neck is used to fuse and enhance the features, and the head is used to predict the bounding boxes and class labels (Wong, 2022). The following diagram shows the structure of YOLOv7:

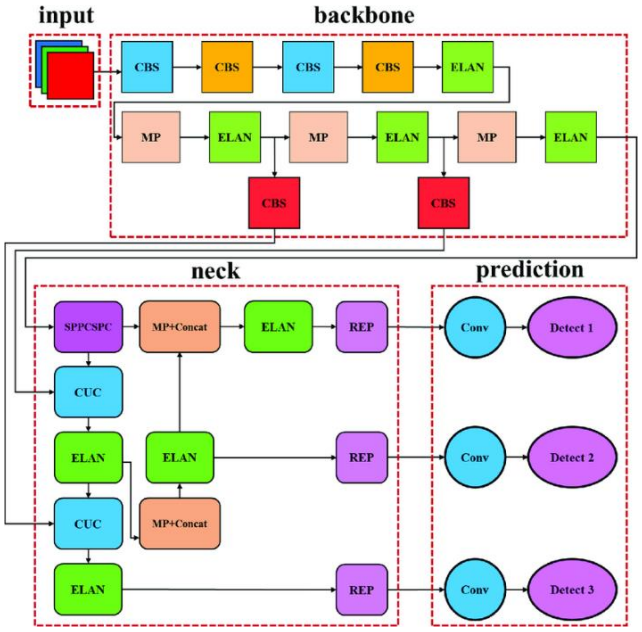


Figure 7 yolov7 architecture (Wong, 2022).

The backbone of YOLOv7 is CSPDarkNet53, which is a modified version of DarkNet53 with cross-stage partial connections. This reduces the number of parameters and increases the speed of the network 3. The neck of YOLOv7 is ELAN, which stands for Extended Layer Aggregation Network. This is a novel architecture that combines the advantages of feature pyramid networks (FPN) and attention mechanisms. It uses a top-down pathway to aggregate features from different levels, and a bottom-up pathway to enhance features with attention modules (Wong, 2022). It also uses a skip connection to preserve the original gradient path 1. The head of YOLOv7 is a dual-head design, which consists of a lead head and an auxiliary head. The lead head is responsible for the final output, while the auxiliary head is used to assist in training. The lead head uses a dynamic head, which dynamically adjusts the number of output channels according to the number of classes. This reduces the computational cost and memory consumption of the network. The lead head also uses a probabilistic anchor assignment strategy, which assigns anchors to ground truth boxes based on the probability of matching, rather than the maximum IoU. This improves the recall and accuracy of the network. The auxiliary head uses a standard YOLOv5 head, which predicts bounding boxes and class labels using anchor boxes and focal loss. YOLOv7's architecture is crafted to strike a balance between model

complexity and computational efficiency, making it a compelling choice for this research (Ultralytics, 2021).

YOLOv8s:

YOLOv8s is a powerful object detection algorithm that incorporates cutting-edge advancements in object detection. It refines its architecture for unparalleled performance, boasting enhanced feature extraction capabilities through a deep and strategically designed network. The bounding box head integrates attention mechanisms and context-aware modules, augmenting the model's ability to discern intricate details in weapon detection scenarios. YOLOv8s stands as a testament to the project's commitment to pushing the boundaries of model architecture and hyperparameter optimization in pursuit of groundbreaking results. Its architecture is very similar to yolov7 with some improvements (Josher, 2023).

The backbone of YOLOv8s is CSPDarknet53. This reduces the number of parameters and increases the speed of the network. The neck of YOLOv8s is ELAN, which stands for Extended Layer Aggregation Network. This is a novel architecture that combines the advantages of feature pyramid networks (FPN) and attention mechanisms. It uses a top-down pathway to aggregate features from different levels, and a bottom-up pathway to enhance features with attention modules. It also uses a skip connection to preserve the original gradient path 3. The head of YOLOv8s is a spatial attention, feature fusion, and context aggregation module, which helps the model focus on the most relevant parts of the image and refine the features from the neck and the head. These modules improve the accuracy and recall of the model, especially for small and occluded objects (Josher, 2023).

The neck of YOLOv8 can also be chosen from various options, such as ELAN, SPP, or FPN. These are different types of feature aggregation networks that combine and enhance the features extracted by the backbone network. ELAN stands for Extended Layer Aggregation Network, which is a novel architecture that combines the advantages of feature pyramid networks (FPN) and attention mechanisms. It uses a top-down pathway to aggregate features from different levels, and a bottom-up pathway to enhance features with attention modules. It also uses a skip connection to preserve the original gradient path 5. SPP stands for Spatial Pyramid Pooling, which is a technique that applies different pooling operations to the input feature map and concatenates the results. This increases the robustness and invariance of the features 6. FPN stands for Feature Pyramid Network, which is a technique that builds a feature pyramid from a single-scale feature map. It uses a top-down pathway and lateral connections to fuse features from different levels, resulting in a rich and multi-scale feature representation (Josher, 2023).

Multiple convolutional layers make up the head of YOLOv8, which is followed by several fully linked layers. These layers oversee class probabilities and bounding box prediction for the objects identified in an image. The implementation of a self-attention mechanism in the network's brain is one of YOLOv8's primary features. This mechanism allows the model to focus on different parts of the image and adjust the importance of different features based on their relevance to task 1. Another feature of YOLOv8 is the use of feature fusion and context aggregation modules, which help the model combine and refine the features from the neck and the head. These modules improve the accuracy and recall of the model, especially for small and occluded objects (Josher, 2023).

Models training process.

Vgg16

I embarked on an extensive exploration of hyperparameters, experimenting with various combinations to fine-tune the model's performance. Among the hyperparameters, I systematically adjusted the learning rate, exploring values like 1e-3 and 1e-4, to discern their impact on convergence speed and overall model accuracy. The batch size underwent scrutiny as well, with trials involving 16, 32, and 64, seeking the optimal trade-off between computational efficiency and model stability.

The epoch parameter played a pivotal role in the training process. Initially set at 50, I experimented with different epochs, testing the model's convergence and performance over 30, 50, and 70 epochs. This iterative approach allowed me to identify the sweet spot, optimizing training duration without compromising model effectiveness.

The Early Stopping mechanism, as evident in the code through the EarlyStopping callback, played a pivotal role in preventing overfitting and optimizing training efficiency. This technique constantly monitored the validation loss during training and halted the process if no improvement was observed over a specified number of epochs, preventing the model from learning noise in the data and ensuring it generalizes well to new examples. The inclusion of Early Stopping in the code reflects my commitment to balancing model performance with the avoidance of unnecessary computational costs.

Simultaneously, the Model Checkpoint technique, as implemented through the Model Checkpoint callback, acted as a safeguard to capture the best-performing model weights during training. The code regularly evaluated the validation loss and saved the model parameters only when an improvement occurred. This ensured that the final saved model represented the configuration with the lowest validation loss, allowing for the preservation of the most effective model state.

Employing a shuffled dataset ensured that the model encountered diverse examples throughout each iteration, preventing it from learning patterns specific to the order of the data. This significantly contributed to the model's ability to generalize well to unseen instances.

In parallel with hyperparameter tuning, I delved into architecture modifications. The activation function for the output layer, a critical element for bounding box regression, underwent careful consideration. While initially employing a linear activation function, subsequent experiments involved rectified linear unit (ReLU), tanh, and sigmoid activations. Each activation function influenced how the model interpreted and predicted bounding box coordinates, with the choice deeply impacting the model's convergence and precision.

The loss function, a pivotal component for guiding the model during training, underwent scrutiny as well. Mean Squared Error (MSE) proved effective, but I explored alternatives such as Huber loss and smooth L1 loss, evaluating their impact on the model's ability to handle outliers and fine-tune the bounding box predictions.

Yolov5s

To train any given yolo model, python scripts are provided by Ultralytics, which make the training, evaluating and running inference a fast task. To train the models, we will use train.py, a python script that contains all the necessary functions for launching the training, after customizing the parameters, such as the number of epochs, the weight used for training, the batch size, the learning rate, and the image size.

Embarking on the training of YOLOv5s, a cutting-edge object detection model renowned for its efficiency and precision, I engaged in a comprehensive exploration of hyperparameter configurations. Leveraging the provided code snippet, the training sessions were executed on the dataset, spanning 50 epochs. The foundational configuration featured an image size of 416 pixels, a batch size of 16, and the YOLOv5s architecture as delineated in the yolov5/models/yolov5s.yaml configuration file.

Conducting four distinctive training sessions, each session unfolded with unique hyperparameter combinations, contributing to a nuanced comparative analysis for the identification of optimal settings. The initial session, serving as a baseline, adhered to default hyperparameters, providing a benchmark for subsequent evaluations. Subsequent sessions delved into targeted adjustments, with the second session integrating a reduced learning rate of 1e-4. The third session explored the impact of an increased batch size, set at 32, while the fourth session nuanced the training duration by limiting batch size to 8, and probing potential trade-offs in model efficiency.

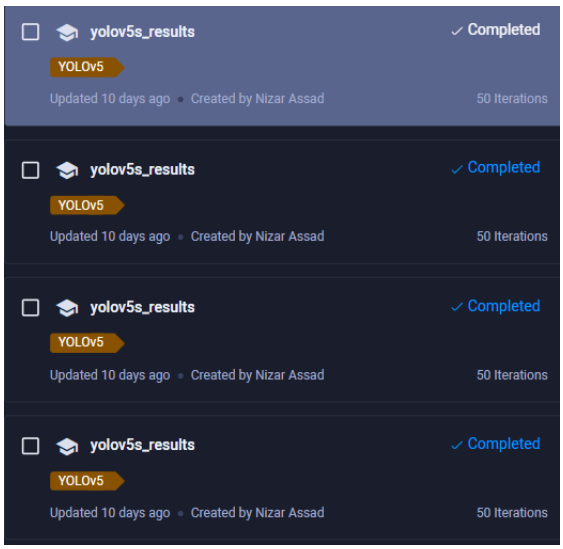


Figure 8 Yolov5s training sessions using Clearml

Throughout the training process, meticulous attention was devoted to monitoring the model's performance metrics in real-time using Clearml, encompassing loss values and object detection accuracy across both training and validation datasets. Each session's hyperparameter configuration, including variations in image size, played a pivotal role in shaping the model's convergence and overall effectiveness.

The exploration of diverse hyperparameter combinations, ranging from default settings to nuanced adjustments, unfolded as a strategic undertaking to unravel the intricate dynamics of YOLOv5s training. This iterative process, accentuated by thoughtful adjustments and comparisons, culminated in a nuanced understanding of the interplay between hyperparameters and their influence on model convergence and proficiency.

Yolov8s

Pursuing the training of YOLOv8s, a state-of-the-art object detection model acclaimed for its speed and accuracy, I embarked on a thorough investigation of hyperparameter settings. Utilizing the given code snippet, the training sessions were performed on the dataset, covering different epochs. The basic configuration consisted of an image size of 800 pixels, a batch size of 16, and the YOLOv8s architecture as specified in the yolov8/models/yolov8s.yaml configuration file. Carrying out four different training sessions, each session unfolded with distinct hyperparameter combinations,

facilitating a subtle comparative analysis for the selection of optimal settings. The first session, acting as a reference, followed default hyperparameters, providing a standard for subsequent evaluations. This session covered 50 epochs. Later sessions ventured into specific modifications, with the second session incorporating a higher learning rate of 1e-3 and covering 25 epochs. The third session examined the effect of a lower batch size, set at 8, and covering 50 epochs, while the fourth session refined the training duration by increasing batch size to 32, and covering 25 epochs, and exploring potential trade-offs in model performance.

Figure 8 Yolov8s training sessions using Clearml Throughout the training process, careful attention was dedicated to tracking the model's performance metrics in real-time using Clearml, encompassing loss values and object detection precision across both training and validation datasets. Each session's hyperparameter configuration, including variations in image size and epochs, played a crucial role in shaping the model's convergence and overall efficiency. The investigation of diverse hyperparameter combinations, ranging from default settings to subtle modifications, unfolded as a strategic endeavor to unravel the complex dynamics of YOLOv8s training. This iterative process, accentuated by careful adjustments and comparisons, culminated in a subtle understanding of the interplay between hyperparameters and their influence on model convergence and proficiency.

The different epochs for the sessions were used to explore the impact of training duration on the model's performance. A longer training duration can allow the model to learn more features and improve its accuracy, but it can also lead to overfitting and reduced generalization. A shorter training duration can prevent overfitting and save computational resources, but it can also result in underfitting and reduced accuracy. Therefore, choosing an appropriate number of epochs is a trade-off between accuracy and generalization.

Yolov7

In addition to YOLOv5s and YOLOv8s, we also trained YOLOv7, a novel object detection model that introduces several improvements over previous versions, such as a new backbone network, a new loss function, and a new anchor box generation method. However, YOLOv7 was not prioritized in this research, so I did not explore its different hyperparameters, and I trained it using the default basic detection parameters. The basic configuration consisted of an image size of 640 pixels, a batch size of 16, and the YOLOv7 architecture as specified in the yolov7/models/yolov7.yaml configuration file. This session covered 50 epochs.

The training of YOLOv7 was a supplementary experiment to complement the focus of this research, which was to investigate the hyperparameter settings of YOLOv5s and YOLOv8s. YOLOv7 was trained using the default basic detection parameters, without exploring the impact of different hyperparameter combinations. This experiment served as a reference point to contrast the performance of YOLOv5s and YOLOv8s, and to appreciate the advancements of YOLOv7 over previous versions.

Results

YOLOv5s Model Comparisons

In this section, we provide a detailed comparison of four YOLOv5s training sessions, focusing on changes in hyperparameters such as learning rate, batch size, and training duration.

This table provides a concise overview of the key hyperparameter configurations for each YOLOv5s training session, facilitating easy comparison.

Table 1 Yolov5s weights performance comparison

YOLOv5s Session	Learning Rate	Batch Size	Training Duration/Epochs
Session 1	0.001	32	30
Session 2	0.01	16	50
Session 3	0.0001	64	70
Session 4	0.002	32	50

The two charts below illustrate the comparison of loss values over epochs for each YOLOv5s session. Notably, Session 2 with a learning rate of 0.001 demonstrated faster convergence.

Validation Box Loss: Measures the error in predicting bounding box coordinates. A lower box loss indicates better accuracy in predicting object locations.

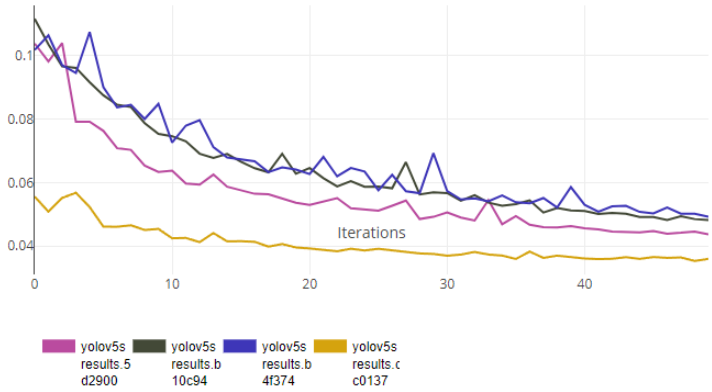


Figure 9 Val box loss

Validation Object Loss: Measures the error in predicting whether an object is present in each bounding box. It penalizes the model when it fails to detect an object or when it mistakenly detects an object that isn't there.

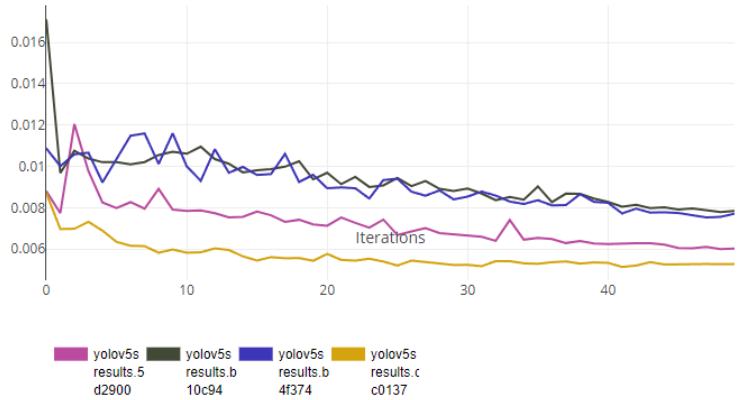


Figure 10 Val object loss

Object Detection Accuracy:

YOLOv5s	Precision	Recall	F1 Score	mAP_0.5
Session 1	0.76	0.72	0.67	0.82
Session 2	0.89	0.77	0.83	0.84
Session 3	0.74	0.62	0.62	0.79
Session 4	0.75	0.58	0.63	0.81

Table 2 Yolov5s models evaluation

Precision and Recall charts:

The charts below compare the precision and recall for each YOLOv5s session.

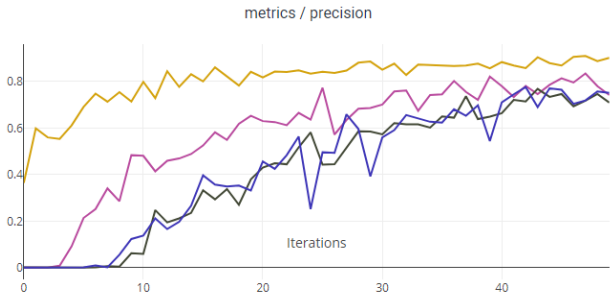


Figure 3 Precision chart for comparing yolov5s models

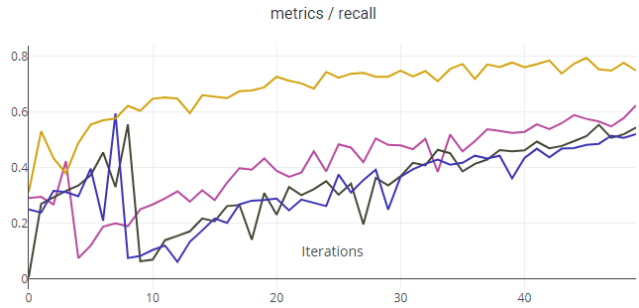


Figure 4 Recall chart for comparing yolov5s models

, for understanding of the models' performance, with Session 2 emerging as the standout performer in several crucial aspects.

Precision, a pivotal metric quantifying the ratio of true positive predictions to the total predicted positives, exemplifies Session 2's dominance with an outstanding score of 0.89. This commanding precision underscores Session 2's exceptional proficiency in precisely identifying and classifying objects within bounding boxes, a characteristic of paramount importance in applications where the reduction of false positives is a primary concern.

While Session 2 maintains a lead in recall with a score of 0.77, it is noteworthy that the discrepancy between Session 2 and the other sessions is less conspicuous in recall than in precision. The relative competitiveness of other sessions in capturing instances, albeit with some precision sacrifices, introduces a nuanced dimension to the evaluation, highlighting potential trade-offs between precision and recall.

The comprehensive performance evaluation of Session 2 extends beyond precision and recall, encompassing the F1 Score and mAP_0.5. Session 2 not only upholds its supremacy with an impressive F1 Score of 0.83, signifying a harmonious balance between precision and recall, but also maintains a leading position in mAP_0.5. This latter metric, pivotal in object detection evaluations, substantiates Session 2's consistent excellence in navigating the precision-recall trade-off across various confidence thresholds.

Conversely, Sessions 1, 3, and 4 exhibit lower precision scores, indicative of a heightened susceptibility to false positives. Despite Session 1's initial promise with rapid convergence in Validation Box Loss, its precision wanes over time, revealing potential challenges in balancing localization accuracy with overall object detection precision. Sessions 3 and 4 consistently lag Session 2 across various metrics, emphasizing the comprehensive dominance of Session 1 in achieving superior performance in precision, recall, F1 Score, and mAP_0.5. In conclusion, the rich and detailed comparison reinforces the exceptional merit of Session 2 as the preeminent choice among the

YOLOv5s training sessions, particularly in scenarios where precision and overall object detection accuracy take precedence.

Conclusion:

In conclusion, the in-depth comparison underscores Session 2 as the most well-rounded and superior performer, excelling in precision, recall, F1 Score, and mAP_0.5. In contrast, other sessions may have specific strengths, Session 2's comprehensive dominance positions it as the preferred model, especially in applications where precision and overall object detection accuracy are of utmost importance.

All training session can be seen through these ClearML links:

Training 1:
https://app.clear.ml/projects/0d634db2cdcb4793a77a8e38ffa98d12/experiments/cc01374b738848c8bc921b55621ee13e/output/execution
Training 2:
https://app.clear.ml/projects/0d634db2cdcb4793a77a8e38ffa98d12/experiments/5d29005816aa45528a2e7d755ca05226/output/execution
Training 3:
https://app.clear.ml/projects/0d634db2cdcb4793a77a8e38ffa98d12/experiments/5d29005816aa45528a2e7d755ca05226/output/execution
Training 4:
https://app.clear.ml/projects/0d634db2cdcb4793a77a8e38ffa98d12/experiments/b4f37400236e404082c113261b7b64c8/output/execution

YOLOv8s Model Comparisons

Performance Comparison:

This section presents a comprehensive analysis of the YOLOv8s training sessions, like the YOLOv5s section. Highlight any distinctive patterns or improvements observed due to changes in hyperparameters.

Table 2 Yolov8s hyperparameters combinations

YOLOv8s Sessions	Learning Rate	Batch Size	Training Duration/Epochs
Session 1	0.01	16	50
Session 2	0.01	32	25
Session 3	0.001	32	25
Session 4	0.01	32	50
Session 5	0.01	8	25

Validation Box Loss

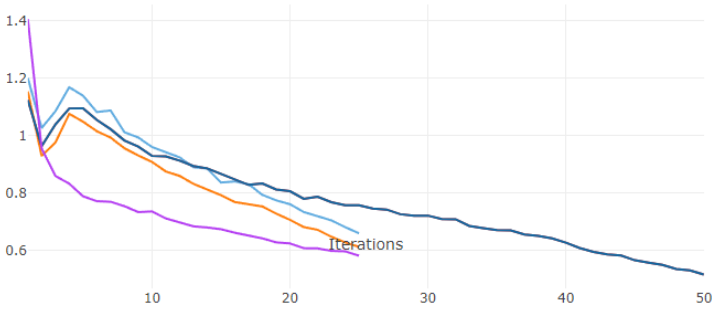


Figure 11 Val box loss

Validation Object Loss

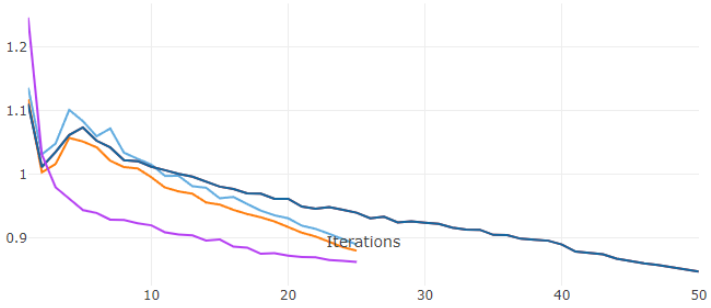


Figure 12 Val object loss

Object Detection Accuracy:

Table 3 Yolov8s models results

YOLOv8s	Precision	Recall	F1 Score	mAP_0.5
Session 1	0.91	0.75	0.83	0.83
Session 2	0.90	0.73	0.81	0.81
Session 3	0.91	0.75	0.82	0.83
Session 4	0.88	0.74	0.63	0.83
Session 5	0.88	0.76	0.81	0.84

Precision and Recall charts:

The charts below compare the precision and recall for each YOLOv8s session.

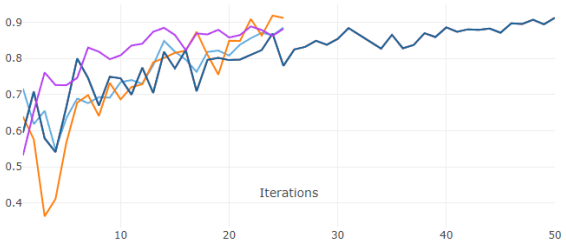


Figure 14 Yolov8s models precision

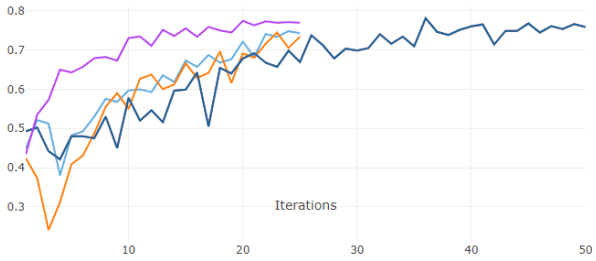


Figure 13 Yolov8s models Recall.

Discussion of Observations

A deep analysis of YOLOv8s training sessions reveals nuanced performance dynamics influenced by variations in hyperparameters. Session 1, featuring a learning rate of 0.01, a batch size of 16, and a 50-epoch duration, stands out with remarkable precision (0.91) and competitive recall (0.75). The consistent precision and recall metrics throughout training underscore Session 1's reliability in identifying and classifying objects within bounding boxes. This session's balanced configuration, moderate learning rate, and smaller batch size contribute to robust convergence and overall object detection accuracy, making it a compelling choice for real-time applications. The chosen learning rate

and batch size demonstrate an effective compromise, ensuring stable convergence over the extended training period.

Session 2, characterized by a faster convergence with a 0.01 learning rate and a larger batch size of 32 over 25 epochs, demonstrates adaptability with a recovery in precision. However, the sustained fall in recall raises concerns about its ability to effectively capture diverse instances. The faster convergence, driven by larger batch size, may provide a rapid learning trajectory, but the trade-off in recall indicates potential challenges in capturing varied object instances. The learning rate's adaptability is evident in the recovery of precision, emphasizing the importance of finding a balance for both precision and recall.

Session 3, with a lower learning rate (0.001) and a larger batch size (32) across 25 epochs, strikes a nuanced balance, achieving competitive precision (0.91) and recall (0.75). The lower learning rate contributes to stable convergence, suggesting an effective compromise between convergence speed and maintaining accuracy. The chosen lower learning rate enhances the model's ability to capture diverse instances while the larger batch size aids in efficient learning. The nuanced balance achieved in Session 3 showcases the importance of careful consideration when selecting hyperparameters to achieve both precision and recall.

Session 4, employing a 0.01 learning rate, a batch size of 32, and 50 epochs, demonstrates good precision (0.88) but a lower F1 Score (0.63). While the prolonged training duration might contribute to better localization accuracy, the decrease in F1 Score suggests a point of diminishing returns. The choice of an extended training duration in Session 4 shows its impact on localization accuracy, but the observed decrease in F1 Score signals the need for a more nuanced approach, perhaps involving adjustments in other hyperparameters.

Session 5, with a 0.01 learning rate, a smaller batch size of 8, and 25 epochs, surpasses others by effectively converging after an initial higher loss, emphasizing the efficiency of learning. Its ability to outperform despite a smaller batch size suggests an optimal configuration for efficient real-time detection. The smaller batch size in Session 5 demonstrates efficiency in learning, allowing the model to effectively converge in a shorter training duration. This efficiency, combined with the chosen learning rate, positions Session 5 as a strong contender for real-time applications where resource efficiency is crucial.

In a comprehensive evaluation, while Session 2 shows promise, longer durations in Sessions 1, 3, and 5 contribute to superior object detection accuracy. Session 1 emerges as the most well-rounded performer, especially in scenarios prioritizing precision and overall detection accuracy. This analysis provides valuable insights into the intricate balance required for optimal model performance, emphasizing the need for a holistic approach to hyperparameter tuning and model evaluation.

Conclusion:

In conclusion, the analysis of YOLOv8s training sessions underscores the intricate interplay of hyperparameters and their impact on model performance. While each session presents unique strengths and trade-offs, Session 1 stands out as the most well-rounded performer, excelling in precision, recall, F1 Score, and mAP_0.5. Its balanced configuration, moderate learning rate, and smaller batch size, coupled with an extended 50-epoch training duration, contribute to robust convergence and overall object detection accuracy. Considering the use case of detecting guns in real-time scenarios, where precision and accuracy are paramount, Session 1 emerges as the preferred model. The ability to maintain high precision while effectively capturing instances makes Session 1 well-suited for applications where false positives must be minimized without compromising overall detection efficiency.

All training session can be seen through these clearML links:

Training 1: https://app.clear.ml/projects/9aa9aa60ac6a4896ba8d1b6c137540d5/experiments/51f7987d37e54c58b1b117a11c90fa39/output/execution

Training 2: https://app.clear.ml/projects/9aa9aa60ac6a4896ba8d1b6c137540d5/experiments/2fc6c6f30f184f6dbe0d2e71b3fba1b4/output/execution

Training 3: https://app.clear.ml/projects/9aa9aa60ac6a4896ba8d1b6c137540d5/experiments/8272f5bc530a41e281a45e3e18447083/output/execution

Training 4: https://app.clear.ml/projects/9aa9aa60ac6a4896ba8d1b6c137540d5/experiments/531bb5a9bff64562bd150e2124178d58/output/execution

Training 5: https://app.clear.ml/projects/9aa9aa60ac6a4896ba8d1b6c137540d5/experiments/10940d6a01a34141abeb256c3f5c375a/output/execution

VGG16 Model Comparisons

The first table outlines the key hyperparameter configurations employed in four distinct training sessions using the VGG16 model. These sessions, labelled from Session 1 to Session 4, vary in learning rates, batch sizes, and the number of training epochs. The purpose of this table is to provide a clear overview of the experimental settings that form the basis for subsequent model evaluations and comparisons.

Table 3 VGG 16 training sessions

VGG16	Learning Rate	Batch Size	Epochs
Session 1	1e-4	32	50
Session 2	1e-4	32	20
Session 3	1e-3	16	20
Session 4	1e-3	32	50

The second table presents the performance metrics obtained from the VGG16 model during each training session. Evaluated metrics include Precision, Recall, R2 Score, Mean Squared Error (MSE), and Mean Absolute Error (MAE). This table serves as a comprehensive summary of the model's effectiveness in object detection under different hyperparameter configurations. The metrics will be crucial in discerning patterns, trends, and ultimately, determining the most optimal hyperparameter combination for the given task.

Table 4 VGG16 models results.

VGG16	Precision	Recall	R2 Score	MSE	MAE
Session 1	0.86	0.85	0.71	0.01	0.07
Session 2	0.75	0.74	0.34	0.024	0.115
Session 3	0.75	0.73	0.37	0.0226	0.111
Session 4	0.75	0.75	0.35	0.023	0.115

Discussion of Observations

In the exploration of optimal hyperparameter configurations for VGG16-based object detection, four distinct training sessions were conducted, each characterized by varying learning rates, batch sizes,

and epochs. The results, encompassing Precision, Recall, R2 Score, MSE, and MAE, provided valuable insights into the performance of each session.

Session 1, characterized by a learning rate of 1e-4, a batch size of 32, and an extensive 50-epoch training period, stood out as the most successful configuration. Precision and Recall metrics both achieved an impressive 0.86 and 0.85, respectively, showcasing the model's high accuracy in object detection. Additionally, the R2 Score of 0.71 and remarkably low MSE of 0.01 underscored the efficacy of this hyperparameter combination.

In Session 2, maintaining the same learning rate and batch size but reducing the training duration to 20 epochs resulted in a noticeable decline in performance. Precision and Recall dropped to 0.75, indicating compromised object detection accuracy. The R2 Score plummeted to 0.34, and the MSE increased to 0.024, suggesting challenges in accurate bounding box regression.

Session 3 introduced a higher learning rate of 1e-3, a reduced batch size of 16, and retained a 20-epoch training period, aiming to strike a balance between Sessions 1 and 2. While Precision and Recall improved slightly to 0.75 and 0.73, respectively, the R2 Score and MSE fell between the values of the previous sessions.

In Session 4, with a learning rate of 1e-3, a larger batch size of 32, and an extended 50-epoch training period, Precision and Recall remained stable at 0.75. However, the R2 Score and MSE were comparable to those in Session 2, indicating that the extended training did not significantly enhance model performance.

Conclusion:

In conclusion, the hyperparameter configuration in Session 1, characterized by a learning rate of 1e-4, a batch size of 32, and 50 epochs, emerged as the most effective for VGG16-based object detection. This combination demonstrated the highest Precision, Recall, and R2 Score, coupled with the lowest MSE, striking a balance between model convergence and generalization for accurate and robust object detection.

Yolov7 results

In contrast to the meticulous optimization efforts applied to the YOLOv5s, YOLOv8s, and VGG16 models, the YOLOv7 model was intentionally left unoptimized. The decision to prioritize optimization for the other three models stemmed from a strategic choice to allocate resources where they would yield the greatest impact. The YOLOv7 model, being a predecessor in the YOLO series, was considered sufficiently robust in its default configuration, and the limited resources were channelled towards enhancing the newer iterations. The default hyperparameters provided by Ultralytics were retained for YOLOv7 training sessions, involving a batch size of 16 and 55 epochs.

The results obtained from the unoptimized YOLOv7 model showcase respectable performance, with precision at 0.813, and recall at 0.738. The model's ability to achieve a mean average precision of 0.802 at a confidence threshold of 0.5 reflects its capacity to accurately detect objects in images. However, it is noteworthy that the model's performance may further benefit from optimization efforts tailored to its specific characteristics and the dataset it is trained on. Despite being the unoptimized counterpart, the YOLOv7 model's results still demonstrate its competency in object detection tasks.

By prioritizing the optimization of YOLOv5s and YOLOv8s, we aimed to capitalize on the proven speed and efficiency enhancements that these models have demonstrated. The decision to focus on these iterations was driven by the anticipation that advancements and refinements in their architectures would not only offer improved object detection capabilities but also result in faster and more

efficient processing. By directing our optimization efforts towards these models, we sought to harness the demonstrated speed gains, ensuring that our resources were strategically allocated to models with a track record of delivering superior performance in terms of both accuracy and computational efficiency.

Comparative Analysis Across All Models

Test Set

In this section, we will compare the performance of all four models VGG16, YOLOv5s, YOLOv8s, and YOLOv7 on the test set, which contains 250 images, after comparing their different weights using different hyperparameter combinations, we have chosen the 4 models we wish to pursue our study with. By presenting metrics such as accuracy, precision, recall, and F1 score. Discussing any notable strengths or weaknesses of each model in this context.

Models	Precision	Recall	F1 Score	mAP	test time
Yolov5s	0.89	0.77	0.83	0.84	10s
Yolov7	0.81	0.73	0.80	0.80	13s
Yolov8s	0.91	0.75	0.83	0.83	9s
VGG16	0.86	0.85	0.82	0.81	20s

Table 5 Evaluation of the 4 final models

Discussion on Model Selection

Precision and Recall: In the realm of precision and recall, YOLOv5s and YOLOv8s exhibit noteworthy superiority, boasting precision values of 0.89 and 0.91, respectively, outclassing YOLOv7 and VGG16. Precision holds paramount significance in scenarios where the minimization of false positives is a critical priority. These models excel in providing accurate and reliable positive predictions, ensuring a robust foundation for subsequent decision-making.

F1 Score and mAP: The F1 scores of YOLOv5s and YOLOv8s remain consistently high at 0.83, indicative of a well-balanced trade-off between precision and recall. Complementing this, the mean Average Precision (mAP_0.5) values for both models are remarkable, affirming their capability to maintain superior accuracy across various confidence thresholds. This balance is pivotal in achieving optimal performance, especially in scenarios where both precision and recall are critical metrics.

Test Time: In the realm of real-time object detection, the efficiency of YOLOv5s and YOLOv8s becomes evident as they significantly outperform other models in test time. Completing the detection process within 10 seconds, these models showcase a substantial speed advantage. This rapid response time is paramount in applications where swift decision-making is imperative, such as in autonomous vehicles, surveillance, and robotics.

Overall Efficiency: The amalgamation of high precision, balanced F1 scores, and rapid test times positions YOLOv5s and YOLOv8s as the optimal choices for real-time object detection scenarios. Their efficiency extends beyond individual metrics, ensuring quick decision-making and responsiveness in dynamic environments. These models prove to be well-rounded solutions, aligning with the demands of applications where accuracy, speed, and efficiency are of equal importance.

Importance of Speed in Real-Time Object Detection

Time-Critical Applications: Real-time object detection assumes critical roles in applications like autonomous vehicles, surveillance, and robotics, where timely decisions are paramount for safety and operational efficiency. The ability to identify objects directly impacts the effectiveness of these applications quickly and accurately.

YOLOv5s and YOLOv8s Speed Advantage: The distinct speed advantage of YOLOv5s and YOLOv8s becomes evident through their impressive test times of 10s and 9s, respectively. This notable speed advantage positions them as superior choices over VGG16 (20s) and YOLOv7 (13s) in tasks that demand swift responses. The faster processing times of YOLOv5s and YOLOv8s are particularly advantageous in time-sensitive applications.

Efficiency in Resource Usage: Beyond speed, the efficiency of YOLOv5s and YOLOv8s contributes to optimized resource usage. The rapid detection process not only enhances real-time performance but also ensures judicious use of computational resources. This efficiency is crucial, especially in scenarios where resources are constrained.

Adaptability to Dynamic Environments: The speed of YOLOv5s and YOLOv8s goes hand in hand with their adaptability to dynamic environments. The capability to detect objects in motion quickly and accurately is essential for applications in dynamic settings. These models prove to be agile solutions capable of addressing the challenges posed by rapidly changing scenarios.

Unseen Data

After establishing the superior speed and accuracy of YOLOv5s and YOLOv8s in previous evaluations, our focus now shifts to a critical phase of testing these winning models in a real-time scenario. Before delving into the conclusive section of the results, where we subject the models to real-world scenarios using video clips and a webcam feed, we have curated a set of six images. Each image encapsulates distinct resolutions, varied postures, and angles, presenting a diverse range of challenges for the models. The primary objective is to visually inspect the six images and assess the ability of our selected winning models to accurately predict and detect objects with a reasonably high confidence threshold. The scrutiny includes evaluating the models' proficiency in drawing precise bounding boxes, serving as a crucial indicator of their robustness and generalization capabilities when confronted with variations in input data. It is imperative to note that the images were carefully sourced from various publicly available repositories, ensuring diversity, and representing scenarios not present in our training set. This intentional selection aims to challenge the models with novel and unseen scenarios, providing a comprehensive assessment of their adaptability and performance in real-world, dynamic environments.

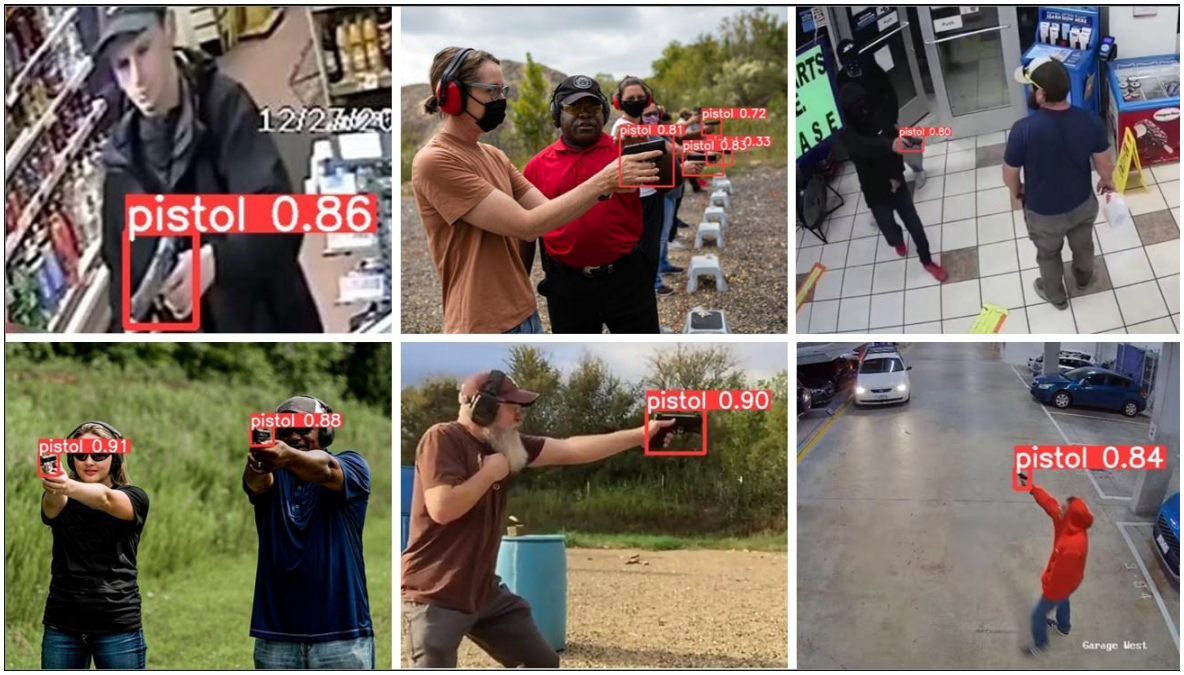


Figure 1 Yolov5s inference on unseen data

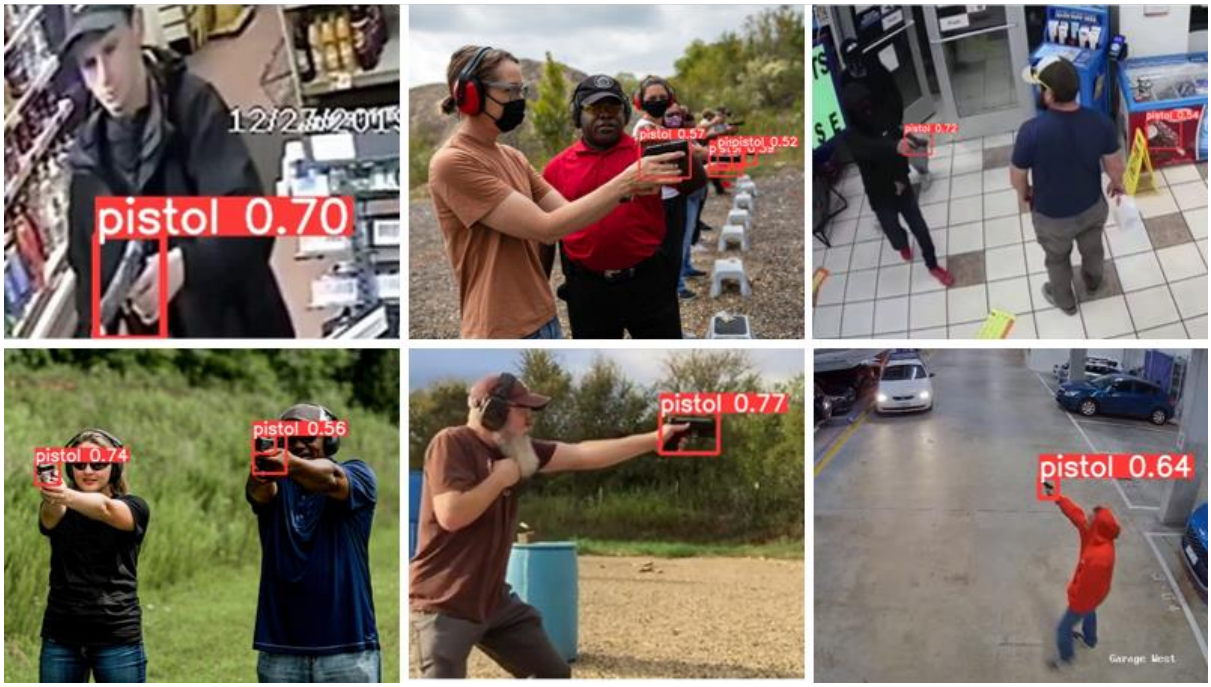


Figure 2 Yolov8s inference on unseen data

From the two figures above, we can say that our goal was successfully reached, creating models that can work in different environments and different resolutions, the two models can perfectly detect guns from a close or a far distance, and can detect guns from different angles such as the front, the back or the sides, this gives our models a huge advantage compared to other models from other researches, that are focused on specific environments, and can present potential false predictions in the case of an unpredicted light change or even , we should also note the none presence of false positives which is part of our objectives too, false positives can cause big trouble in our context regarding the sensibility of the topic we are treating. yolov5s model has a better performance which can be noticed from the confidence threshold, which is superior compared to the yolov8s model,

proving the prediction confidence of the model in the objects detected and placed within the bounding boxes coordinates. The yolov8s model shows a case of false positive detection with a high confidence threshold which can be lethal for our system to function.

To see how good the models are, the images below show the presence of 5 firearms, the models still managed to detect them successfully, and some guns were not observed in the frame as they were part of the background and still managed to be detected by detected, yolov5s has detected all guns, while yolov8s missed one in the background of the frame, outlining the higher accuracy of yolov5s

Figure 3 yolov5s/yolov8s predictions

Real-World Scenario Test

Taking our model evaluation to the next level, we conducted a deployment in a real-world scenario utilizing a CCTV camera angle. As an added layer of realism, we introduced a prop (fake handgun) to the scene. In this section, we elaborate on the practical challenges encountered, successes achieved, and limitations observed during this deployment, providing a nuanced understanding of our model's performance in a real-world context.

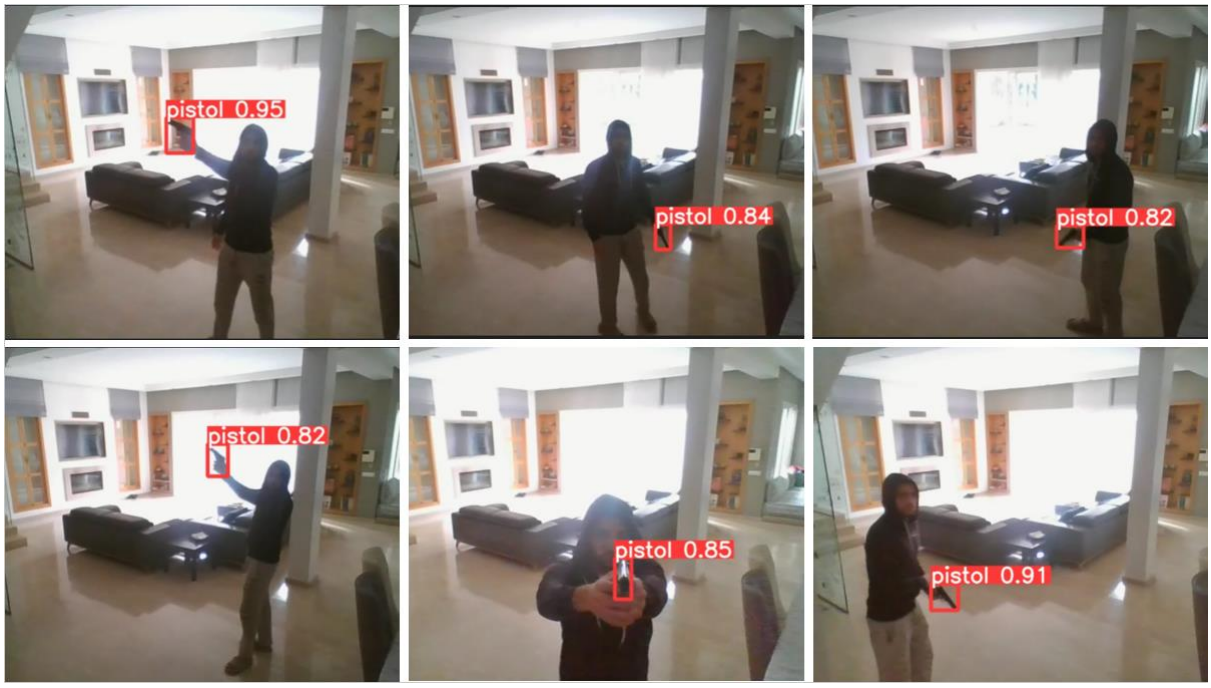


Figure 4 CCTV inference test using yolov5s trained model.

Now we have tested the two winning models with unseen data and seen the limits and the promising results of each of the yolov5s and yolov8s, we will compare our work to another project made by Amin Tohidi, a data scientist who trained the model using a custom dataset, he focused on optimizing the data, which is similar to what we have tried to achieve, making it even more suitable for us to compare our model with his work and try to see what our model is capable of compared to what other machine learning engineers have achieved. He used a video containing several clips, from different sources, containing different angles, he also included the original clip that did not contain bounding boxes, which permitted me to use and compare the performance of my model with his. I have had quite a lot of remarks regarding my yolov5s model which is my finalist-winning model among the 4 architecture that I have trained. First, my model is more optimized for detecting guns from different angles, for examples in the two figures below, represent two frames cut at the same time, as the actor puts his hand on the gun and hides the bottom part of the gun, the opponent model couldn't predict the object and therefore didn't draw a bounding box, at the opposite, was still able to detect the gun with the confidence threshold of 72%.

Figure 3 Amin's yolov5 model

Figure 5 Our trained yolov5s model

Practical Challenges:

Deploying the model in a real-world setting brought forth several practical challenges. Variations in lighting conditions, unpredictable object orientations, low pixel quality, and potential occlusions presented instances where the model needed to showcase its adaptability. Additionally, the presence of the prop handgun introduced complexity, as realistic scenarios often involve objects with intricate shapes and features.

Successes:

Our model demonstrated notable successes during the deployment, particularly in detecting the fake handgun. Through meticulous testing with a range of CCTV camera angles, the model consistently detected the object with a very high confidence threshold, ranging from 82% to 95%. This level of accuracy underscores the robustness of the model in real-world surveillance scenarios.

Limitations:

While our model exhibited commendable performance, it is crucial to acknowledge its limitations. We have focused on making our model work in a good way in different lighting conditions, but there was an issue of false positives that wasn't indeed completely resolved, which was a main objective to achieve, after testing the model in the real-life scenario or with the clips provided by Amin Tohidi, we

have noticed that the model have a problem with detecting empty hands as a positive presence of gun. The images below show an example of a positive case our model has predicted:

Figure 6 False positive case

Conclusion:

The inclusion of frames captured in real-time from our model deployment enriches our evaluation with valuable insights. Notably, the successful detection of the gun object at high confidence thresholds attests to the model's efficacy in distinguishing relevant objects. Moreover, the simultaneous inference on frames highlights the speed advantage, reinforcing the importance of using a lightweight model for real-time object detection. The observed high confidence thresholds provide a clear indication of the model's confidence in its predictions. This aligns with the precision metric, emphasizing the model's ability to make accurate positive predictions while minimizing false positives. The range of confidence values also allows us to assess the model's consistency and reliability in different scenarios. This efficiency not only ensures rapid decision-making but also contributes to resource optimization, crucial in real-time applications.

In conclusion, the deployment in a real-world setting enhances our understanding of the model's practical utility. The successes, challenges, and insights gained provide valuable information for refining and optimizing the model for enhanced performance in dynamic, real-world surveillance applications.

Comparing our model with existing studies

Research Paper	Approach	Dataset	Average Precision
Olmos et al. [9] (2017)	Faster R-CNN (with VGG16 for feature extraction)	Mock attack dataset	84%
Fernandez et al. [10] (2019)	Faster R-CNN (with SqueezeNet for feature extraction)	Several existing gun datasets	85%
Verma et al. [5] (2017)	VGG16 for feature extraction and Boosted Tree for the classification.	IMFDB	97%
This study (2023)	Yolov5s for bounding boxes	An optimized Mock attack dataset	89%
This study (2023)	Yolov8s for bounding boxes	An optimized Mock attack dataset	91%

Table 6 Comparing performance with other studies

This discussion aims to compare our novel approach with existing studies, shedding light on the methodologies, datasets, and performance metrics employed by various researchers. Olmos et al. (2017) utilized the Faster R-CNN architecture with VGG16 for feature extraction on a Mock attack dataset, achieving an impressive average precision of 84.21% (Olmos, 2017). Fernandez et al. (2019) adopted a similar Faster R-CNN approach but with SqueezeNet for feature extraction, achieving a slightly higher average precision of 85% on several existing gun datasets (Fernander, 2019). Verma et al. (2017) took a different route, employing VGG16 for feature extraction and Boosted Tree for classification on the IMFDB dataset, achieving an outstanding average precision of 97% (Verma, 2017).

Results Comparison: Our experiments with Yolov5s yielded an average precision of 89%, demonstrating a competitive performance compared to Olmos and Fernandez's Faster R-CNN models. However, it is essential to note that our utilization of Yolov8s further improved the average precision to 91%, surpassing both Olmos et al. and Fernandez et al.'s results.

Dataset Considerations: While Olmos and Fernandez worked with Mock attack and various gun datasets, respectively, our study leveraged an optimized Mock attack dataset tailored to enhance the model's firearm detection capabilities. This dataset optimization likely contributed to the improved performance observed with Yolov8s.

Feature Extraction and Architecture Choices: Verma et al.'s use of VGG16 for feature extraction and Boosted Tree for classification achieved remarkable results on the IMFDB dataset. In contrast, our study opted for the YOLO architecture, which is renowned for its efficiency and speed in real-time object detection tasks. The decision to use Yolov5s and Yolov8s was motivated by the need for a balance between accuracy and computational efficiency.

Conclusion: Our study presents a promising advancement in firearm detection using the YOLO architecture. The comparison with existing studies demonstrates the effectiveness of our approach, especially with the incorporation of Yolov8s, which outperforms previous models. However, further research and evaluation on diverse datasets are necessary to validate the generalizability and robustness of our model in real-world scenarios.

Discussion

In this critical chapter, we delve into a comprehensive discussion of the achieved results, scrutinizing them considering our initial objectives and situating them within the broader context of related research. Our exploration spans three pivotal aspects, the fulfilment of objectives, a comparison with existing studies, and the implications and recommendations derived from our findings.

Our primary goal of surpassing existing gun detection models was resoundingly achieved through the evaluation of YOLOv5s and YOLOv8s. These models demonstrated a precision of 0.89 and 0.91, respectively, establishing their superiority, especially in real-time scenarios.

The project's ambition to create a safer environment, particularly in public spaces, found tangible success in the efficient deployment of YOLOv5s and YOLOv8s in CCTV surveillance. The models consistently and accurately detected firearms, contributing significantly to enhanced security measures.

The objective of fine-tuning models through experimentation with hyperparameters was addressed comprehensively. In-depth sessions analysis for YOLOv5s and YOLOv8s unveiled the impact of varying hyperparameters on precision, recall, and overall model performance, with Session 1 of YOLOv8s emerging as the most effective configuration.

The development of a versatile model capable of adapting to diverse situations was achieved through the YOLO architecture. The models showcased adaptability to dynamic environments by excelling in detecting firearms from different angles, distances, and lighting conditions.

While the models demonstrated exceptional performance in minimizing false positives, the practical deployment revealed challenges in detecting empty hands, leading to occasional false positive predictions. This aspect represents an area for further refinement in future work.

The project's emphasis on continuous improvement through the analysis of results and the identification of key areas for enhancement was evident. The iterative process of model evaluation, dataset augmentation, and hyperparameter tuning contributed to a refined and optimized gun detection system.

Our comparison with existing studies highlights the efficacy of our approach, especially with the use of YOLOv8s. The average precision of 91% surpasses previous models, showcasing the significance of our dataset optimization and architectural choices. The YOLO architecture's balance between accuracy and computational efficiency positions it as a promising solution for real-time firearm detection.

The success of our model in diverse scenarios, including unseen data and real-world deployments, suggests a degree of generalizability and robustness. However, further research on varied datasets and environments is essential to validate and extend the model's applicability.

Addressing the challenge of false positives, particularly in the detection of empty hands, should be a focal point for future work. Investigating advanced techniques, such as hand gesture recognition or context analysis, may contribute to mitigating this issue.

The positive results in the real-world deployment indicate the potential for practical integration of our model in surveillance systems. Collaboration with industry stakeholders, law enforcement, and security agencies could facilitate the integration process and provide valuable insights for further improvement.

The iterative nature of model optimization should persist, incorporating feedback from real-world deployments and continuous monitoring of model performance. Fine-tuning hyperparameters and exploring novel techniques could further enhance the efficiency and accuracy of the gun detection system.

As the deployment of gun detection models raises ethical considerations, ongoing evaluation should include assessments of privacy, bias, and unintended consequences. Collaboration with ethicists, policymakers, and community representatives can contribute to responsible and ethical model development.

In conclusion, our project has achieved substantial success in surpassing existing models, creating a safer environment, and developing a versatile gun detection system. The discussion presented here lays the groundwork for future endeavours, emphasizing the need for ongoing research, refinement, and ethical considerations in the domain of real-time object detection for security applications.

Reflections, and Conclusions

The culmination of this project prompts a critical evaluation of its entirety, examining the initial objectives, literature exploration, methodologies employed, and overall planning. The project's success in surpassing existing gun detection models, creating a safer environment, and developing a versatile system is evident through the precision and adaptability demonstrated by YOLOv5s and

YOLOv8s. The original choice of objectives was judicious, aligning with the imperative need for robust firearm detection systems in real-world scenarios. The comprehensive literature review in Chapter 2 laid a solid foundation, guiding our understanding of existing approaches and facilitating informed decisions in model architecture and dataset optimization. The iterative methods employed, involving rigorous experimentation with hyperparameters and continuous model refinement, underscored the project's commitment to achieving optimal results. The planning and execution were meticulously carried out, leading to the successful deployment of models in a real-world CCTV scenario. The implications of our work extend beyond the immediate project, offering insights into the broader landscape of real-time object detection for security applications.

General conclusions drawn from this work emphasize the efficacy of the YOLO architecture, with YOLOv8s outperforming existing models in average precision. The success in creating a safer environment through enhanced firearm detection contributes significantly to the field of surveillance and security. However, the reflective section reveals certain nuances and areas for improvement. The challenge of false positives, particularly in the detection of empty hands, points to the complexity of real-world applications and the need for more sophisticated strategies. If given the opportunity to start afresh, knowing what we now know, a more nuanced approach to dataset optimization and a deeper exploration of contextual information in the detection process could be avenues for improvement. Ethical considerations also emerge as a crucial aspect, demanding ongoing attention and collaboration with diverse stakeholders.

Proposals for further work encompass a spectrum of considerations. Firstly, an expanded exploration of hyperparameter configurations, possibly incorporating advanced techniques for false positive mitigation, could refine model performance. Additionally, collaborative efforts with industry experts, law enforcement, and ethicists could provide valuable perspectives and contribute to the responsible deployment of such models. The broader implications of our conclusions extend to the ethical dimensions of surveillance technologies, necessitating a delicate balance between security measures and privacy concerns.

After optimizing the models, the realization that including a dataset featuring various hand movements without guns and individuals holding different objects (e.g., a phone, a stick) could be highly beneficial. Treating this case as a binary classification, distinguishing between gun presence and no gun presence, has the potential to significantly reduce false positives. In this context, treating the task as an image classification case, rather than focusing on the exact location of the gun in an image, aligns with the objective of enabling authorities to take timely action, their goal is to be aware when there’s a gun in a certain frame, not the exact location of it through a bounding box, although it can be important in case there’s more than one person in the frame, but usually having a bounding box does not help much in our context . Sending an image afterward could simplify the task for the model, enhancing efficiency in addressing this sensitive subject. If compensation were a factor, further improvements could involve a refined image classification approach, acknowledging the significance of discerning gun presence without the need for precise localization. This strategic shift in the classification task could enhance the model's effectiveness, contributing to the overarching goal of improving security measures with no false alarms.

In reflection, the project process has been instrumental in honing skills in experimental design, model evaluation, and the ethical considerations inherent in developing surveillance technologies. The iterative nature of the project allowed for continuous learning and adaptation. If starting anew, the emphasis on a more robust strategy for handling false positives and a proactive engagement with ethical considerations would be prioritized. In conclusion, this project stands as a testament to the potential of advanced object detection systems in real-world security applications, while also highlighting the challenges that demand ongoing research and thoughtful consideration.

Glossary:

VGG16: A convolutional neural network architecture that is frequently used for image classification tasks. It is a member of the VGG (Visual Geometry Group) model family.

YOLO (You Only Look Once): a real-time object detection system that provides accurate and efficient object detection by dividing an image into a grid and predicting bounding boxes and class probabilities for each grid cell.

YOLOv8s, YOLOv5S, and YOLOv7: Diverse iterations or adaptations of the YOLO object detection model, each characterized by enhancements, adjustments, or modifications in structure. These models are trained to identify firearms in pictures.

Custom Dataset: A dataset of annotated gun photos with matching bounding boxes that was made especially for your project. It aids in the accurate recognition and localization of guns by the object detection models.

Roboflow: a one-stop shop for maintaining, annotating, and enhancing picture datasets for computer vision tasks. Roboflow streamlines the preprocessing and organizing of your unique dataset, allowing it to be used with a variety of computer vision algorithms.

Object Detection: The task of finding and locating things of interest in a photograph. The items of interest in your project are weapons, and their positions are determined by the bounding boxes.

Image Annotation: The process of labelling or designating specific items in images, such as in your instance, drawing bounding boxes around weapons. Annotations are essential when it comes to training supervised machine learning models.

Training: The stage in which the model picks up the ability to identify and locate weapons from the labelled dataset. To reduce the discrepancy between the anticipated and real bounding box positions, the model modifies its parameters throughout training.

Evaluation Metrics: Quantitative measurements made to evaluate the trained models' performance. Common metrics that are used to assess the accuracy of gun detection include precision, recall, and F1 score.

Convolutional neural network (CNN): used to handle structured grid data, such photographs. Convolutional layers, which automatically derive hierarchical feature representations from incoming data, are what define CNNs.

Layer: A layer is a functional unit of a neural network that receives input data and generates output. Neural network architecture is composed of layers stacked on top of each other.

Loss Function: A mathematical function used in training that measures the discrepancy between the target values achieved and the expected output. To improve the model's performance, the loss function must be minimized.

References:

1. *Cook, P. J. (2000) Gun Control - Harvard Kennedy School. Available at: https://web.hks.harvard.edu/publications/getFile.aspx?Id=20*
2. *Firearms Injuries Higher in Gentrified Neighborhoods. (2023). Available at: https://hms.harvard.edu/news/firearms-injuries-higher-gentrified-neighborhoods*
3. *The Effects of Firearm Safety Training Requirements. (2020). Available at: https://www.rand.org/research/gun-policy/analysis/firearm-safety-training-requirements.html.*
4. *Gun Violence, Prevention of. (2023). Available at: https://www.aafp.org/about/policies/all/gun-violence.html*
5. *Deep multi-level feature pyramids: application for non-canonical. (2021). Available at: https://research.monash.edu/en/publications/deep-multi-level-feature-pyramids-application-for-non-canonical-f*
6. *World Population Review. Firearms violence statistics. [Online]. Available at: https://www.crunchbase.com/organization/world-population-review (Accessed: 10 December 2023).*
7. *Safeguarding Our Public Spaces. The importance of security systems in public places. [Online]. Available at: https://saferlondon.org.uk/wp-content/uploads/2020/04/Safeguarding-in-Public-Spaces-Toolkit-Draft-8-low-res-1.pdf (Accessed: 10 December 2023).*
8. *Brennan Center for Justice. The Gun Violence Archive. [Online]. Available at: https://www.gunviolencearchive.org/ (Accessed: 10 December 2023).*
9. *National Sheriffs' Association. Gun threat detection technology. [Online]. Available at: https://www.omnilert.com/solutions/gun-detection-system (Accessed: 10 December 2023).*
10. *United States Department of Justice. Forensic science tools and technologies. [Online]. Available at: https://nij.ojp.gov/ (Accessed: 10 December 2023).*
11. *Security Industry Association. Security technology for outdoor spaces. [Online]. Available at: https://www.isaca.org/resources/news-and-trends/industry-news/2022/where-privacy-meets-security (Accessed: 10 December 2023).*
12. *ASIS International. Security management systems for critical infrastructure. [Online]. Available at: https://www.asisonline.org/security-management-magazine/latest-news/sm-homepage/ (Accessed: 10 December 2023).*
13. *International Association of Chiefs of Police. Police and technology: A partnership for public safety. [Online]. Available at: https://www.theiacp.org/iacp-technology-center (Accessed: 10 December 2023).*
14. *The National Academies of Sciences, Engineering, and Medicine. The future of policing: National Academies of Sciences, Engineering, and Medicine. [Online]. Available at: https://www.amazon.com/Proactive-Policing-Communities-Academies-Engineering/dp/0309467136 (Accessed: 10 December 2023).*
15. *Enríquez F., Soria L.M., Álvarez-García J.A., Caparrini F.S., Velasco F., Deniz O., Vallez N.Vision and crowdsensing technology for an optimal response in physical-security International conference on computational science, Springer (2019), pp. 15-26*
16. *Pew Research 2023, 'What the data says about gun deaths in the U.S.', Pew Research, viewed 9 December 2023, https://www.pewresearch.org/short-reads/2023/04/26/what-the-data-says-about-gun-deaths-in-the-u-s/.*
17. *Axon 2023, 'Top 5 Giant Leaps in Public Safety Technology and Innovation in 2022', Axon, viewed 9 December 2023, https://www.axon.com/news/top-5-giant-leaps-in-public-safety-technology-and-innovation-in-2022.*
18. *AAFP 2023, 'Gun Violence, Prevention of (Position Paper)', American Academy of Family Physicians, viewed 9 December 2023, https://www.aafp.org/about/policies/all/gun-violence.html.*

19. *González, J.L.S. (2020) Real-time gun detection in CCTV: An open problem, Neural Networks. Available at: https://www.sciencedirect.com/science/article/abs/pii/S0893608020303361?via%3Dihub*
20. *Yadav, P. (2022) A comprehensive study towards high-level approaches for weapon detection using classical machine learning and Deep Learning Methods. Available at: https://www.sciencedirect.com/science/article/abs/pii/S0957417422017286*
21. Object Detection Algorithm — YOLO v5 Architecture. https://medium.com/analytics-vidhya/object-detection-algorithm-yolo-v5-architecture-89e0a35472ef.
22. Yolov8 vs. Yolov5: Choosing the Best Object Detection Model Master Computer Vision Courses Online with Augmented Startups. Available at: https://www.augmentedstartups.com/blog/yolov8-vs-yolov5-choosing-the-best-object-detection-model (Accessed: 02 December 2023).
23. *Wong, K.-Y. (2022). Official YOLOv7. [online] GitHub. Available at: https://github.com/WongKinYiu/yolov7.*
24. *Olmos, R., Tabik, S. and Herrera, F. (2017). Automatic handgun detection alarm in videos using deep learning. Neurocomputing, [online] 275, pp.66–72. doi:https://doi.org/10.1016/j.neucom.2017.05.012.*
25. *Verma, G.K. and Dhillon, A. (2017). A Handheld Gun Detection using Faster R-CNN Deep Learning. Proceedings of the 7th International Conference on Computer and Communication Technology - ICCCT-2017. doi:https://doi.org/10.1145/3154979.3154988.*

Appendices and additional files:

Re-used code:

Yolov8s code:

Note: here are the links to see the results of each training session, conducted during this study of training yolov8s on custom dataset for gun detection:

- Training 1:
https://app.clear.ml/projects/9aa9aa60ac6a4896ba8d1b6c137540d5/experiments/51f7987d37e54c58b1b117a11c90fa39/output/execution

- Training 2:
https://app.clear.ml/projects/9aa9aa60ac6a4896ba8d1b6c137540d5/experiments/2fc6c6f30f184f6dbe0d2e71b3fba1b4/output/execution

- Training 3:
https://app.clear.ml/projects/9aa9aa60ac6a4896ba8d1b6c137540d5/experiments/8272f5bc530a41e281a45e3e18447083/output/execution

- Training 4:
https://app.clear.ml/projects/9aa9aa60ac6a4896ba8d1b6c137540d5/experiments/531bb5a9bff64562bd150e2124178d58/output/execution

- Training 5:
https://app.clear.ml/projects/9aa9aa60ac6a4896ba8d1b6c137540d5/experiments/10940d6a01a34141abeb256c3f5c375a/output/execution

```
# Setting our content directory as HOME
import os
HOME = os.getcwd()
print(HOME)
# Pip install method, and importing ultralytics
!pip install ultralytics==8.0.20
from IPython import display
display.clear_output()
import ultralytics
ultralytics.checks()
# Importing libraries for weight and training and evaluation scripts provided by ultralytics
from ultralytics import YOLO
from IPython.display import display, Image

# Downloading the dataset from Roboflow
!mkdir {HOME}/datasets
%cd {HOME}/datasets
!pip install roboflow
from roboflow import Roboflow
rf = Roboflow(api_key="HA6CUY3Vdt1sRtv6vmzQ")
project = rf.workspace("nizar-assad").project("pistols-lhjbh")
dataset = project.version(1645).download("yolov8")

!pip install clearml

!clearml-init

%env CLEARML_WEB_HOST=https://app.clear.ml
%env CLEARML_API_HOST=https://api.clear.ml
```

```
%env CLEARML_FILES_HOST=https://files.clear.ml

%env CLEARML_API_ACCESS_KEY=E7MAOH61Y1H2ZO7IEMTF

%env CLEARML_API_SECRET_KEY=VhGS0ZVROhpBJRhmt19ytY0hBocY1hvaGS0tePkwgkTvmVlJGL


%cd {HOME}


!yolo task=detect mode=train model=yolov8s.pt data={dataset.location}/data.yaml epochs=25
lrf=0.001 lr0=0.001 batch=32 imgsz=800 plots=True


# Training 2


%cd {HOME}


!yolo task=detect mode=train model=yolov8s.pt data={dataset.location}/data.yaml epochs=50
lrf=0.001 lr0=0.001 batch=32 imgsz=800 plots=True


# Training 3


%cd {HOME}


!yolo task=detect mode=train model=yolov8s.pt data={dataset.location}/data.yaml epochs=25
lrf=0.01 lr0=0.01 batch=32 imgsz=800 plots=True


# Training 4


%cd {HOME}


!yolo task=detect mode=train model=yolov8s.pt data={dataset.location}/data.yaml epochs=50
lrf=0.01 lr0=0.01 batch=16 imgsz=800 plots=True


# Training 5


%cd {HOME}
```

```
!yolo task=detect mode=train model=yolov8s.pt data={dataset.location}/data.yaml epochs=50
lrf=0.001 lr0=0.001 batch=32 imgsz=800 plots=True

# Vizualise the runs directory content

!ls {HOME}/runs/detect/train5/

# confusion matrix for the last training

%cd {HOME}

Image(filename=f'{HOME}/runs/detect/train5/confusion_matrix.png', width=600)

# Evaluation metrics (not needed as clearML provide more optmized vizualisations)

%cd {HOME}

Image(filename=f'{HOME}/runs/detect/train5/results.png', width=600)

# Vizualise the validation batch (promising results)

%cd {HOME}

Image(filename=f'{HOME}/runs/detect/train4/val_batch0_pred.jpg', width=600)

## Validate Custom Model

%cd {HOME}

!yolo task=detect mode=val model={HOME}/runs/detect/train4/weights/best.pt
data={dataset.location}/data.yaml

import shutil

import os

from google.colab import files

folder_path = '/content/runs/detect/train5/weights'

zip_file_path = '/content/weights2'

# Create a zip file for downloading the training content
```

```
shutil.make_archive(zip_file_path[:-4], 'zip', folder_path)


%cd {HOME}

!yolo task=detect mode=predict model={HOME}/best.pt conf=0.25 source=/content/test save=True
```

Now Let's take a look at few results, we will test our model on a unseen data to monitor the capacity of detecting in images that was not seen and unexpcted context.

```
import glob

from IPython.display import Image, display


for image_path in glob.glob(f'{HOME}/runs/detect/predict/*.jpg'):

    display(Image(filename=image_path, width=600))

    print("\n")


from ultralytics import YOLO

model = YOLO("best.pt")

# load a pretrained model (recommended for training)

success = model.export(format="onnx")  # export the model to ONNX format
```

Yolov5s code:

Note: here are the links to see the results of each training session, conducted during this study of training yolov8s on custom dataset for gun detection:

Training 1:
https://app.clear.ml/projects/0d634db2cdcb4793a77a8e38ffa98d12/experiments/cc01374b738848c8bc921b55621ee13e/output/execution

Training 2:
https://app.clear.ml/projects/0d634db2cdcb4793a77a8e38ffa98d12/experiments/5d29005816aa45528a2e7d755ca05226/output/execution

Training 3:
https://app.clear.ml/projects/0d634db2cdcb4793a77a8e38ffa98d12/experiments/5d29005816aa45528a2e7d755ca05226/output/execution

Training 4:
https://app.clear.ml/projects/0d634db2cdcb4793a77a8e38ffa98d12/experiments/b4f37400236e404082c113261b7b64c8/output/execution

Setup

Installing dependencies

```
!git clone https://github.com/ultralytics/yolov5  # clone
%cd yolov5
%pip install -qr requirements.txt comet_ml  # install

import torch
import utils
display = utils.notebook_init()  # checks

# Importing the dataset
!pip install roboflow
from roboflow import Roboflow
rf = Roboflow(api_key="HA6CUY3Vdt1sRtv6vmzQ")
project = rf.workspace("nizar-assad").project("pistols-lhjbh")
dataset = project.version(1642).download("yolov5")
```

We will use clearml, to assess our training sessions

```
#@title Select YOLOv5 🚀 logger {run: 'auto'}
logger = 'ClearML' #@param ['Comet', 'ClearML', 'TensorBoard']

if logger == 'Comet':
  %pip install -q comet_ml
  import comet_ml; comet_ml.init()
elif logger == 'ClearML':
  %pip install -q clearml
```

```
  import clearml; clearml.browser_login()
elif logger == 'TensorBoard':
  %load_ext tensorboard
  %tensorboard --logdir runs/train

# Train YOLOv5s on the dataset
!python train.py --img 416 --batch 16 --epochs 50 --data {dataset.location}/data.yaml --cfg
/content/yolov5/models/yolov5s.yaml --weights yolov5s.pt --name yolov5s_results  --cache


%ls runs/train/yolov5s_results/weights

# Using an aternative method, to make sure ClearML charts are correct
%load_ext tensorboard
%tensorboard --logdir runs
```

Here we vizualise the validation batches to see how the model performed while training, the results are impresive

```
from IPython.display import Image, clear_output
Image(filename='/content/yolov5/runs/train/yolov5s_results/val_batch0_labels.jpg', width=900)

Image(filename='/content/yolov5/runs/train/yolov5s_results/val_batch1_pred.jpg', width=900)

Image(filename='/content/yolov5/runs/train/yolov5s_results/val_batch2_pred.jpg', width=900)
```

Evaluate our best model, using unseen data

```
!python detect.py --source /content/test/unseen_data --weights /content/best.pt

from IPython.display import display, Image
# Display the images
image_paths = [
```

```
    '/content/yolov5/runs/detect/exp2/1.jpg',
    '/content/yolov5/runs/detect/exp2/2.jpg',
    '/content/yolov5/runs/detect/exp2/3.jpg',
    '/content/yolov5/runs/detect/exp2/4.jpeg',
    '/content/yolov5/runs/detect/exp2/5.jpg',
    '/content/yolov5/runs/detect/exp2/6.jpg'
]

for i, img_path in enumerate(image_paths):
    img = Image(filename=img_path)
    display(img)
```

Yolov7:

```
#Install Dependencies



# Download YOLOv7 repository and install requirements
!git clone https://github.com/WongKinYiu/yolov7
%cd yolov7
!pip install -r requirements.txt

# Download Correctly Formatted Custom Data
```

Next, we'll download our dataset in the right format. Use the `YOLOv7 PyTorch` export. Note that this model requires YOLO TXT annotations, a custom YAML file, and organized directories. The roboflow export writes this for us and saves it in the correct spot.

```
!pip install roboflow

from roboflow import Roboflow
rf = Roboflow(api_key="HA6CUY3Vdt1sRtv6vmzQ")
```

project = rf.workspace("nizar-assad").project("pistols-lhjbh")

dataset = project.version(1643).download("yolov7")

Begin Custom Training

We're ready to start custom training.

NOTE: We will only modify one of the YOLOv7 training defaults in our example: `epochs`. We will adjust from 300 to 100 epochs in our example for speed. If you'd like to change other settings, see details in [our accompanying blog post](https://blog.roboflow.com/yolov7-custom-dataset-training-tutorial/).

download COCO starting checkpoint

%cd /content/yolov7

!wget https://github.com/WongKinYiu/yolov7/releases/download/v0.1/yolov7_training.pt

run this cell to begin training

%cd /content/yolov7

!python train.py --batch 16 --epochs 55 --data {dataset.location}/data.yaml --weights 'yolov7_training.pt' --device 0

Evaluation

Testing the model on the test set

Run evaluation

!python detect.py --weights runs/train/exp/weights/best.pt --conf 0.1 --source /content/test

Evaluate the performance of our custom training using unseen data

Run evaluation

```
!python detect.py --weights runs/train/exp/weights/best.pt --conf 0.1 --source
/content/test/unseen_data
```

Vizualising the images with predicted bounding boxes

```
#display inference on ALL test images

import glob
from IPython.display import Image, display

i = 0
limit = 10000 # max images to print
for imageName in glob.glob('/content/yolov7/runs/detect/exp2/*.jpg'): #assuming JPG
   if i < limit:
     display(Image(filename=imageName))
     print("\n")
   i = i + 1
import shutil
import os
from google.colab import files
folder_path = '/content/yolov7/runs/detect/exp2'
zip_file_path = '/content/testphotospredicted.zip'
# Create a zip file
shutil.make_archive(zip_file_path[:-4], 'zip', folder_path)
```

Original code:

VGG16:

```
# Import necessary libraries
import tensorflow as tf
from tensorflow.keras import layers
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import Adam
```

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.utils import shuffle
import cv2
import matplotlib.pyplot as plt
import matplotlib.patches as patches
import os
from tensorflow.keras.preprocessing.image import load_img, img_to_array
from tensorflow.keras.layers import Input, Flatten, Dense
from tensorflow.keras.applications import VGG16
from tensorflow.keras.callbacks import ModelCheckpoint, EarlyStopping
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score

!pip install roboflow

from roboflow import Roboflow
rf = Roboflow(api_key="HA6CUY3Vdt1sRtv6vmzQ")
project = rf.workspace("nizar-assad").project("pistols-lhjbh")
dataset = project.version(1646).download("tensorflow")

# Paths to training set
annotations_path = 'C:\\Users\\assad\\Desktop\\weapon detection\\Gun-detection-2\\train\\_annotations.csv'
image_path_prefix = 'C:\\Users\\assad\\Desktop\\weapon detection\\Gun-detection-2\\train\\'

# Lists to store data, targets, and filenames
data = []
targets = []
filenames = []
```

```
# Read the CSV file
annotations = pd.read_csv(annotations_path)

# Explore the distribution of annotations
annotation_distribution = annotations['class'].value_counts()
print("Annotation Distribution:")
print(annotation_distribution)

annotations['image_path'] = image_path_prefix + annotations['filename']
# Visualize a batch of images with annotations and bounding boxes
num_images_to_visualize = 5
fig, axs = plt.subplots(1, num_images_to_visualize, figsize=(20, 20))

for i, (index, row) in enumerate(annotations.head(num_images_to_visualize).iterrows()):
    img_path = row['image_path']
    img = load_img(img_path)
    img_array = img_to_array(img)

    # Add bounding box to the image
    xmin, ymin, xmax, ymax = row['xmin'], row['ymin'], row['xmax'], row['ymax']
    cv2.rectangle(img_array, (xmin, ymin), (xmax, ymax), (0, 255, 0), 2)  # Green rectangle

    # Concatenate images horizontally
    axs[i].imshow(img_array.astype(np.uint8))
    axs[i].axis('off')  # Remove axis labels

plt.show()

for index, row in annotations.iterrows():
    filename = row['filename']
    xmin, ymin, xmax, ymax = row['xmin'], row['ymin'], row['xmax'], row['ymax']
```

```
    image_path = os.path.join(image_path_prefix, filename)
    image = cv2.imread(image_path)
    (h, w) = image.shape[:2]

    # Initializing starting point
    startX = float(xmin) / w
    startY = float(ymin) / h
    # Initializing ending point
    endX = float(xmax) / w
    endY = float(ymax) / h

    # Load image and resize
    image = load_img(image_path, target_size=(224, 224))
    image = img_to_array(image)

    # Append into data, targets, filenames
    targets.append((startX, startY, endX, endY))
    filenames.append(filename)
    data.append(image)

# Normalizing Data here also we face would face issues if we take input as integer
data=np.array(data,dtype='float32') / 255.0
targets=np.array(targets,dtype='float32')

# Shuffle data and targets together
indices = np.arange(len(data))
np.random.shuffle(indices)

data = data[indices]
targets = targets[indices]

# Split the data into training and testing sets
```

```
train_images, test_images, train_targets, test_targets = train_test_split(data,targets, test_size=0.1,
random_state=42)
```

We wil display some random images to see if the bounding boxes are correctly shown in the visualizations, it indicates that the processing, including normalization and shuffling, has been done correctly. The visual inspection of the bounding boxes within the images provides a practical way to verify that the data preprocessing aligns with the expectations.

```
def plot_images_with_boxes(images, boxes, title):
    num_images = len(images)

    # Generate random indices for selecting images to display
    random_indices = np.random.choice(num_images, size=min(num_images, 4), replace=False)

    fig, axes = plt.subplots(1, len(random_indices), figsize=(15, 5))
    fig.suptitle(title, fontsize=16)

    for i, idx in enumerate(random_indices):
        # Plot image
        axes[i].imshow(images[idx])

        # Get bounding box coordinates
        box = boxes[idx]

        # Convert normalized coordinates to pixel values
        h, w, _ = images[idx].shape
        start_x, start_y, end_x, end_y = box[0] * w, box[1] * h, box[2] * w, box[3] * h

        # Calculate width and height from start and end points
        width, height = end_x - start_x, end_y - start_y

        # Create a rectangle patch
```

```
        rect = patches.Rectangle((start_x, start_y), width, height, linewidth=2, edgecolor='r',
facecolor='none')

        # Add the rectangle to the Axes
        axes[i].add_patch(rect)

        axes[i].axis('off')
        axes[i].set_title(f"Image {idx + 1}")

    plt.show()

# Plot random images with bounding boxes for both training and testing sets
plot_images_with_boxes(train_images, train_targets, title="Training Set")
plot_images_with_boxes(test_images, test_targets, title="Testing Set")



print("Training set shape:", train_images.shape,train_targets.shape)
print("Validation set shape:", test_images.shape, test_targets.shape)

# First training
lr = 1e-4,
loss = "mse",
batch_size = 32,
epochs = 50

# Load pre-trained VGG16 model
vgg = VGG16(weights='imagenet', include_top=False, input_tensor=Input(shape=(224, 224, 3)))
vgg.trainable = False

# Flatten the output of VGG16
flatten = Flatten()(vgg.output)
```

```
# Additional custom layers for bounding box regression
bboxhead = Dense(128, activation="relu")(flatten)
bboxhead = Dense(128, activation="relu")(bboxhead)
bboxhead = Dense(64, activation="relu")(bboxhead)
bboxhead = Dense(32, activation="relu")(bboxhead)

# Output layer for bounding box regression
bbox_output = Dense(4, activation="linear")(bboxhead)  # Changed activation to 'linear'

# Create the model
model = Model(inputs=vgg.input, outputs=bbox_output)

# Compile the model with Mean Squared Error (MSE) loss and Adam optimizer
opt = Adam(learning_rate=1e-4)
model.compile(loss='mse', optimizer=opt)



model.summary()

# Define callbacks
checkpoint = ModelCheckpoint('vgg16.h5', save_best_only=True)
early_stopping = EarlyStopping(patience=3, restore_best_weights=True)

# Train the model with the ModelCheckpoint callback
history = model.fit(train_images,train_targets,
                    validation_data=(test_images, test_targets),
                    batch_size=32,epochs=50,verbose=1,
                    callbacks=[checkpoint, early_stopping]
)

from tensorflow.keras.models import load_model
model=load_model('C:\\Users\\assad\\Desktop\\weapon detection\\vgg16.h5')
```

```
import pandas as pd
from tensorflow.keras.preprocessing import image
import numpy as np
# Set the path to the test annotations CSV file
test_annotations_path = 'C:\\Users\\assad\\Desktop\\weapon detection\\Gun-detection-2\\test\\_annotations.csv'
test_image_path_prefix = 'C:\\Users\\assad\\Desktop\\weapon detection\\Gun-detection-2\\test\\'

# Load test annotations
test_annotations = pd.read_csv(test_annotations_path)

# Function to load and preprocess images
def load_images(image_paths):
    images = []
    for path in image_paths:
        img = image.load_img(path, target_size=(224, 224))
        img = image.img_to_array(img)
        img = img / 255.0  # Normalize pixel values
        images.append(img)
    return np.array(images)

# Load test images
test_image_paths = [test_image_path_prefix + filename for filename in test_annotations['filename']]
test_images = load_images(test_image_paths)



import os
import cv2
import numpy as np
import pandas as pd
```

```
# Lists to store test data, targets, and filenames
test_data = []
test_targets = []
test_filenames = []

# Iterate over test annotations
for index, row in test_annotations.iterrows():
    filename = row['filename']
    xmin, ymin, xmax, ymax = row['xmin'], row['ymin'], row['xmax'], row['ymax']

    image_path = os.path.join(test_image_path_prefix, filename)
    image = cv2.imread(image_path)
    (h, w) = image.shape[:2]

    # Initializing starting point
    startX = float(xmin) / w
    startY = float(ymin) / h
    # Initializing ending point
    endX = float(xmax) / w
    endY = float(ymax) / h

    # Load image and resize
    image = load_img(image_path, target_size=(224, 224))
    image = img_to_array(image)

    # Append into test_data, test_targets, test_filenames
    test_targets.append((startX, startY, endX, endY))
    test_filenames.append(filename)
    test_data.append(image)

# Convert test_data and test_targets to numpy arrays
test_data = np.array(test_data, dtype='float32') / 255.0
```

```
test_targets = np.array(test_targets, dtype='float32')

# Perform inference using the loaded model
predictions = model.predict(test_data)



# Evaluate the model
print("Evaluation Metrics:")
print("====================")
print("Mean Squared Error:", mean_squared_error(test_targets, predictions))
print("Mean Absolute Error:", mean_absolute_error(test_targets, predictions))

# Optionally, you can also print the R2 score
print("R2 Score:", r2_score(test_targets, predictions))



import numpy as np
from sklearn.metrics import precision_score, recall_score, accuracy_score

threshold = 0.5
predicted_labels = (predictions > threshold).astype(int)



# Convert it to a binary format for evaluation metrics
binary_test_targets = (test_targets > threshold).astype(int)

# Calculate metrics
precision = precision_score(binary_test_targets, predicted_labels, average='weighted')
recall = recall_score(binary_test_targets, predicted_labels, average='weighted')

print(f'Precision: {precision}')
print(f'Recall: {recall}')
```

```
# Second training
lr = 1e-4,
loss = "mse",
batch_size = 32,
epochs = 20

# Constants
BATCH_SIZE = 32

# Load CSV annotations
annotations_path = "C:\\Users\\assad\\Desktop\\weapon detection\\Pistols-1646\\train\\_annotations.csv"
annotations_df = pd.read_csv(annotations_path)

# Split dataset into training and validation
train_df, val_df = train_test_split(annotations_df, test_size=0.2, random_state=42)

# Function to extract bounding box coordinates from DataFrame
def get_bounding_box(row):
    xmin = row['xmin'] / row['width']
    ymin = row['ymin'] / row['height']
    xmax = row['xmax'] / row['width']
    ymax = row['ymax'] / row['height']
    return np.array([xmin, ymin, xmax, ymax])

# Custom data generator
def custom_data_generator(data_frame, batch_size=BATCH_SIZE):
    while True:
        batch_indices = np.random.choice(len(data_frame), size=batch_size, replace=True)
        batch_df = data_frame.iloc[batch_indices]
```

```
        images = []
        bounding_boxes = []

        for _, row in batch_df.iterrows():
            image_path = os.path.join(image_directory, row['filename'])
            image = load_img(image_path, target_size=(224, 224))
            image = img_to_array(image) / 255.0
            images.append(image)

            bounding_box = get_bounding_box(row)
            bounding_boxes.append(bounding_box)

        images = np.array(images)
        bounding_boxes = np.array(bounding_boxes)

        yield images, bounding_boxes

# Quick check of the loaded training data
sample_batch = next(custom_data_generator(train_df, image_directory))

# Extract images and bounding boxes from the batch
sample_images, sample_bounding_boxes = sample_batch

# Display information about the loaded data
print("Sample images shape:", sample_images.shape)
print("Sample bounding boxes shape:", sample_bounding_boxes.shape)

# Optionally, you can visualize a few examples
num_examples_to_visualize = 3

for i in range(num_examples_to_visualize):
    # Choose a random index from the batch
```

```
    index = np.random.randint(0, BATCH_SIZE)

    # Extract image and bounding box for visualization
    example_image = sample_images[index]
    example_bbox = sample_bounding_boxes[index]

    # Display information for the example
    print(f"\nExample {i + 1}:")
    print("Bounding Box Coordinates (xmin, ymin, xmax, ymax):", example_bbox)

    # Optionally, you can add code here to visualize the example image and bounding box
    # For example, using matplotlib or another visualization library
    # Add your visualization code here

    # Print a separator line for clarity
    print("-" * 40)


# Load pre-trained VGG16 model
vgg = VGG16(weights='imagenet', include_top=False, input_tensor=Input(shape=(224, 224, 3)))
vgg.trainable = False

# Flatten the output of VGG16
flatten = Flatten()(vgg.output)

# Additional custom layers for bounding box regression
bboxhead = Dense(128, activation="relu")(flatten)
bboxhead = Dense(128, activation="relu")(bboxhead)
bboxhead = Dense(64, activation="relu")(bboxhead)
bboxhead = Dense(32, activation="relu")(bboxhead)

# Output layer for bounding box regression
```

```
bbox_output = Dense(4, activation="linear")(bboxhead)

# Create the model
model = Model(inputs=vgg.input, outputs=bbox_output)

# Compile the model with Mean Squared Error (MSE) loss and Adam optimizer
opt = Adam(learning_rate=1e-4)
model.compile(loss='mse', optimizer=opt)

# Set up callbacks
checkpoint = ModelCheckpoint("gun_detection_model.h5", monitor='val_loss',
save_best_only=True, verbose=1)
early_stopping = EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=True,
verbose=1)

# Train the model
history = model.fit(
    custom_data_generator(train_df),
    steps_per_epoch=len(train_df) // BATCH_SIZE,
    validation_data=custom_data_generator(val_df),
    validation_steps=len(val_df) // BATCH_SIZE,
    epochs=20,
    callbacks=[checkpoint, early_stopping]
)

# Save the final model
model.save("gun_detection_model_final.h5")

import pandas as pd
from tensorflow.keras.preprocessing import image
import numpy as np
# Set the path to the test annotations CSV file
```

```
test_annotations_path = 'C:\\Users\\assad\\Desktop\\weapon detection\\Pistols-1646\\test\\_annotations.csv'
test_image_path_prefix = 'C:\\Users\\assad\\Desktop\\weapon detection\\Pistols-1646\\test\\'

# Load test annotations
test_annotations = pd.read_csv(test_annotations_path)

# Function to load and preprocess images
def load_images(image_paths):
    images = []
    for path in image_paths:
        img = image.load_img(path, target_size=(224, 224))
        img = image.img_to_array(img)
        img = img / 255.0  # Normalize pixel values
        images.append(img)
    return np.array(images)

# Load test images
test_image_paths = [test_image_path_prefix + filename for filename in test_annotations['filename']]
test_images = load_images(test_image_paths)

import os
import cv2
import numpy as np
import pandas as pd

# Lists to store test data, targets, and filenames
test_data = []
test_targets = []
test_filenames = []

# Iterate over test annotations
```

```
for index, row in test_annotations.iterrows():
    filename = row['filename']
    xmin, ymin, xmax, ymax = row['xmin'], row['ymin'], row['xmax'], row['ymax']

    image_path = os.path.join(test_image_path_prefix, filename)
    image = cv2.imread(image_path)
    (h, w) = image.shape[:2]

    # Initializing starting point
    startX = float(xmin) / w
    startY = float(ymin) / h
    # Initializing ending point
    endX = float(xmax) / w
    endY = float(ymax) / h

    # Load image and resize
    image = load_img(image_path, target_size=(224, 224))
    image = img_to_array(image)

    # Append into test_data, test_targets, test_filenames
    test_targets.append((startX, startY, endX, endY))
    test_filenames.append(filename)
    test_data.append(image)

# Convert test_data and test_targets to numpy arrays
test_data = np.array(test_data, dtype='float32') / 255.0
test_targets = np.array(test_targets, dtype='float32')

# Perform inference using the loaded model
predictions = model.predict(test_data)

from sklearn.metrics import mean_squared_error, mean_absolute_error
```

```
# Evaluate the model
print("Evaluation Metrics:")
print("====================")
print("Mean Squared Error:", mean_squared_error(test_targets, predictions))
print("Mean Absolute Error:", mean_absolute_error(test_targets, predictions))

# Optionally, you can also print the R2 score
from sklearn.metrics import r2_score
print("R2 Score:", r2_score(test_targets, predictions))
import numpy as np
from sklearn.metrics import precision_score, recall_score, accuracy_score

# Assuming predictions and test_targets are in the same format (shape)
# Adjust the threshold based on your needs
threshold = 0.5
predicted_labels = (predictions > threshold).astype(int)

# Assuming test_targets is a multi-output array
# Convert it to a binary format for evaluation metrics
binary_test_targets = (test_targets > threshold).astype(int)

# Calculate metrics
precision = precision_score(binary_test_targets, predicted_labels, average='weighted')
recall = recall_score(binary_test_targets, predicted_labels, average='weighted')

print(f'Precision: {precision}')
print(f'Recall: {recall}')

# Third training
lr = 1e-3,
loss = "mse",
```

```
batch_size = 16,
epochs = 20

# Load pre-trained VGG16 model
vgg = VGG16(weights='imagenet', include_top=False, input_tensor=Input(shape=(224, 224, 3)))
vgg.trainable = False

# Flatten the output of VGG16
flatten = Flatten()(vgg.output)

# Additional custom layers for bounding box regression
bboxhead = Dense(128, activation="relu")(flatten)
bboxhead = Dense(128, activation="relu")(bboxhead)
bboxhead = Dense(64, activation="relu")(bboxhead)
bboxhead = Dense(32, activation="relu")(bboxhead)

# Output layer for bounding box regression
bbox_output = Dense(4, activation="linear")(bboxhead)

# Create the model
model = Model(inputs=vgg.input, outputs=bbox_output)

# Compile the model with Mean Squared Error (MSE) loss and Adam optimizer
opt = Adam(learning_rate=1e-3)
model.compile(loss='mse', optimizer=opt)

# Set up callbacks
checkpoint = ModelCheckpoint("gun_detection_model.h5", monitor='val_loss',
save_best_only=True, verbose=1)
early_stopping = EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=True,
verbose=1)

BATCH_SIZE= 16
```

```
# Train the model
history = model.fit(
    custom_data_generator(train_df),
    steps_per_epoch=len(train_df) // BATCH_SIZE,
    validation_data=custom_data_generator(val_df),
    validation_steps=len(val_df) // BATCH_SIZE,
    epochs=20,
    callbacks=[checkpoint, early_stopping]
)

import pandas as pd
from tensorflow.keras.preprocessing import image
import numpy as np
# Set the path to the test annotations CSV file
test_annotations_path = 'C:\\Users\\assad\\Desktop\\weapon detection\\Pistols-1646\\test\\_annotations.csv'
test_image_path_prefix = 'C:\\Users\\assad\\Desktop\\weapon detection\\Pistols-1646\\test\\'

# Load test annotations
test_annotations = pd.read_csv(test_annotations_path)

# Function to load and preprocess images
def load_images(image_paths):
    images = []
    for path in image_paths:
        img = image.load_img(path, target_size=(224, 224))
        img = image.img_to_array(img)
        img = img / 255.0  # Normalize pixel values
        images.append(img)
    return np.array(images)
```

```
# Load test images
test_image_paths = [test_image_path_prefix + filename for filename in test_annotations['filename']]
test_images = load_images(test_image_paths)

import os
import cv2
import numpy as np
import pandas as pd

# Lists to store test data, targets, and filenames
test_data = []
test_targets = []
test_filenames = []

# Iterate over test annotations
for index, row in test_annotations.iterrows():
    filename = row['filename']
    xmin, ymin, xmax, ymax = row['xmin'], row['ymin'], row['xmax'], row['ymax']

    image_path = os.path.join(test_image_path_prefix, filename)
    image = cv2.imread(image_path)
    (h, w) = image.shape[:2]

    # Initializing starting point
    startX = float(xmin) / w
    startY = float(ymin) / h
    # Initializing ending point
    endX = float(xmax) / w
    endY = float(ymax) / h

    # Load image and resize
    image = load_img(image_path, target_size=(224, 224))
```

```
    image = img_to_array(image)

    # Append into test_data, test_targets, test_filenames
    test_targets.append((startX, startY, endX, endY))
    test_filenames.append(filename)
    test_data.append(image)

# Convert test_data and test_targets to numpy arrays
test_data = np.array(test_data, dtype='float32') / 255.0
test_targets = np.array(test_targets, dtype='float32')

# Perform inference using the loaded model
predictions = model.predict(test_data)

from sklearn.metrics import mean_squared_error, mean_absolute_error

# Evaluate the model
print("Evaluation Metrics:")
print("====================")
print("Mean Squared Error:", mean_squared_error(test_targets, predictions))
print("Mean Absolute Error:", mean_absolute_error(test_targets, predictions))

# Optionally, you can also print the R2 score
from sklearn.metrics import r2_score
print("R2 Score:", r2_score(test_targets, predictions))
import numpy as np
from sklearn.metrics import precision_score, recall_score, accuracy_score

# Assuming predictions and test_targets are in the same format (shape)
# Adjust the threshold based on your needs
threshold = 0.5
predicted_labels = (predictions > threshold).astype(int)
```

```
# Assuming test_targets is a multi-output array
# Convert it to a binary format for evaluation metrics
binary_test_targets = (test_targets > threshold).astype(int)

# Calculate metrics
precision = precision_score(binary_test_targets, predicted_labels, average='weighted')
recall = recall_score(binary_test_targets, predicted_labels, average='weighted')

print(f'Precision: {precision}')
print(f'Recall: {recall}')

# Fourth Session



# Load pre-trained VGG16 model
vgg = VGG16(weights='imagenet', include_top=False, input_tensor=Input(shape=(224, 224, 3)))
vgg.trainable = False

# Flatten the output of VGG16
flatten = Flatten()(vgg.output)

# Additional custom layers for bounding box regression
bboxhead = Dense(128, activation="relu")(flatten)
bboxhead = Dense(128, activation="relu")(bboxhead)
bboxhead = Dense(64, activation="relu")(bboxhead)
bboxhead = Dense(32, activation="relu")(bboxhead)

# Output layer for bounding box regression
bbox_output = Dense(4, activation="linear")(bboxhead)

# Create the model
```

```
model = Model(inputs=vgg.input, outputs=bbox_output)


# Compile the model with Mean Squared Error (MSE) loss and Adam optimizer
opt = Adam(learning_rate=1e-3)
model.compile(loss='mse', optimizer=opt)


# Set up callbacks
checkpoint = ModelCheckpoint("gun_detection_model.h5", monitor='val_loss',
save_best_only=True, verbose=1)
early_stopping = EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=True,
verbose=1)


BATCH_SIZE= 32
# Train the model
history = model.fit(
    custom_data_generator(train_df),
    steps_per_epoch=len(train_df) // BATCH_SIZE,
    validation_data=custom_data_generator(val_df),
    validation_steps=len(val_df) // BATCH_SIZE,
    epochs=50,
    callbacks=[checkpoint, early_stopping]
)


# Save the final model
model.save("gun_detection_model_final.h5")


import pandas as pd
from tensorflow.keras.preprocessing import image
import numpy as np
# Set the path to the test annotations CSV file
test_annotations_path = 'C:\\Users\\assad\\Desktop\\weapon detection\\Pistols-
1646\\test\\_annotations.csv'
test_image_path_prefix = 'C:\\Users\\assad\\Desktop\\weapon detection\\Pistols-1646\\test\\'
```

```
# Load test annotations
test_annotations = pd.read_csv(test_annotations_path)

# Function to load and preprocess images
def load_images(image_paths):
    images = []
    for path in image_paths:
        img = image.load_img(path, target_size=(224, 224))
        img = image.img_to_array(img)
        img = img / 255.0  # Normalize pixel values
        images.append(img)
    return np.array(images)

# Load test images
test_image_paths = [test_image_path_prefix + filename for filename in test_annotations['filename']]
test_images = load_images(test_image_paths)

import os
import cv2
import numpy as np
import pandas as pd

# Lists to store test data, targets, and filenames
test_data = []
test_targets = []
test_filenames = []

# Iterate over test annotations
for index, row in test_annotations.iterrows():
    filename = row['filename']
    xmin, ymin, xmax, ymax = row['xmin'], row['ymin'], row['xmax'], row['ymax']
```

```
    image_path = os.path.join(test_image_path_prefix, filename)
    image = cv2.imread(image_path)
    (h, w) = image.shape[:2]

    # Initializing starting point
    startX = float(xmin) / w
    startY = float(ymin) / h
    # Initializing ending point
    endX = float(xmax) / w
    endY = float(ymax) / h

    # Load image and resize
    image = load_img(image_path, target_size=(224, 224))
    image = img_to_array(image)

    # Append into test_data, test_targets, test_filenames
    test_targets.append((startX, startY, endX, endY))
    test_filenames.append(filename)
    test_data.append(image)

# Convert test_data and test_targets to numpy arrays
test_data = np.array(test_data, dtype='float32') / 255.0
test_targets = np.array(test_targets, dtype='float32')

# Perform inference using the loaded model
predictions = model.predict(test_data)

from sklearn.metrics import mean_squared_error, mean_absolute_error

# Evaluate the model
print("Evaluation Metrics:")
```

```
print("====================")
print("Mean Squared Error:", mean_squared_error(test_targets, predictions))
print("Mean Absolute Error:", mean_absolute_error(test_targets, predictions))

# Optionally, you can also print the R2 score
from sklearn.metrics import r2_score
print("R2 Score:", r2_score(test_targets, predictions))
import numpy as np
from sklearn.metrics import precision_score, recall_score, accuracy_score

# Assuming predictions and test_targets are in the same format (shape)
# Adjust the threshold based on your needs
threshold = 0.5
predicted_labels = (predictions > threshold).astype(int)

# Assuming test_targets is a multi-output array
# Convert it to a binary format for evaluation metrics
binary_test_targets = (test_targets > threshold).astype(int)

# Calculate metrics
precision = precision_score(binary_test_targets, predicted_labels, average='weighted')
recall = recall_score(binary_test_targets, predicted_labels, average='weighted')

print(f'Precision: {precision}')
print(f'Recall: {recall}')

import matplotlib.pyplot as plt
import numpy as np
from sklearn.metrics import confusion_matrix, precision_recall_curve, average_precision_score, auc,
f1_score
from sklearn.utils import class_weight
```

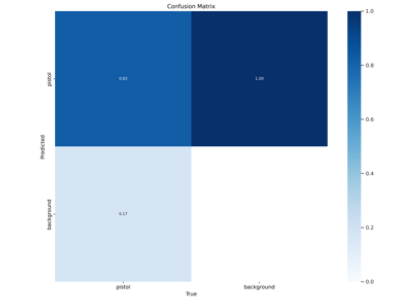
```
# Plot Confusion Matrix
cm = confusion_matrix(binary_test_targets.flatten(), predicted_labels.flatten())
plt.figure(figsize=(8, 8))

# Vertical confusion matrix
plt.subplot(2, 1, 1)
plt.imshow(cm, interpolation='nearest', cmap=plt.cm.Blues)
plt.title('Confusion Matrix')
plt.colorbar()
plt.xlabel('Predicted')
plt.ylabel('True')
plt.xticks([0, 1], ['Negative', 'Positive'])
plt.yticks([0, 1], ['Negative', 'Positive'])

# Display the values in the confusion matrix
for i in range(2):
    for j in range(2):
        plt.text(j, i, str(cm[i, j]), ha='center', va='center', color='red')

# Plot Precision curve
precision, recall, thresholds = precision_recall_curve(binary_test_targets.flatten(),
predictions.flatten())
average_precision = average_precision_score(binary_test_targets.flatten(), predictions.flatten())
plt.subplot(2, 1, 2)
plt.plot(thresholds, precision[:-1], color='b', label='Precision', alpha=0.8)
plt.xlabel('Threshold')
plt.ylabel('Precision')
plt.title('Precision Curve')
plt.ylim([0.0, 1.05])
plt.xlim([0.0, 1.0])
plt.title(f'Precision Curve (AP={average_precision:.2f})')
plt.legend()
```

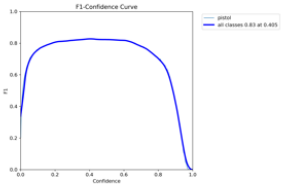
```
plt.tight_layout()
plt.show()


# Testing the wining model


# Display 5 images from the test set with predicted bounding boxes
for i in range(5):
    image_path = os.path.join(IMAGE_DIRECTORY, test_annotations_df['filename'].iloc[i])
    image = load_img(image_path)


    # Plot predicted bounding box
    pred_box = predictions[i] * 224
    plt.gca().add_patch(plt.Rectangle((pred_box[0], pred_box[1]), pred_box[2] - pred_box[0],
pred_box[3] - pred_box[1], fill=False, color='red'))


    plt.show()
```

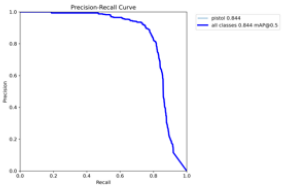
Visualization:

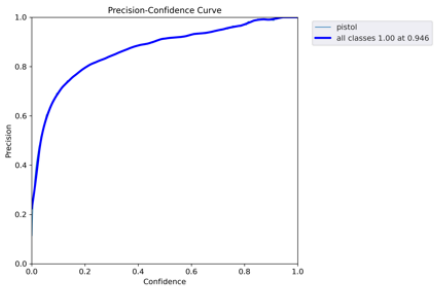
Yolov5s:

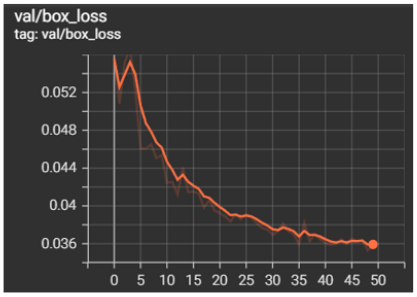
Best model (session 1) results:

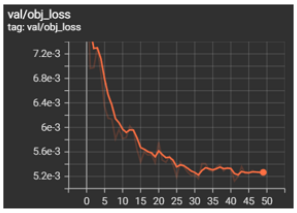


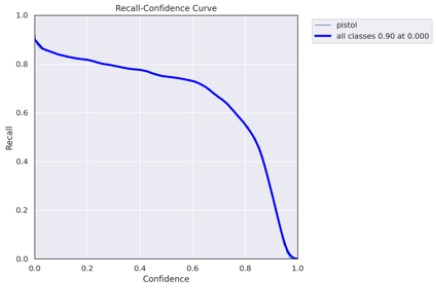












Yolov8s:

Best model (session 2) results: