# for plag check.docx

*by* Shriram G

**Submission date:** 09-May-2023 08:41AM (UTC+0530)
**Submission ID:** 2088192281
**File name:** for_plag_check.docx (5.08M)
**Word count:** 8769
**Character count:** 48951

# CHAPTER 1

# INTRODUCTION

## 1.1 Introduction

The importance of security and safety cannot be overstated in our society, especially in public spaces. With the increase in violent crimes and security breaches in public places, the need for advanced security systems has become more critical than ever. Intelligent security systems that incorporate machine learning and computer vision algorithms can offer advanced and reliable security solutions compared to traditional security measures.

Compared to conventional systems, the suggested security system intends to offer a more effective and dependable security solution. Modern computer vision techniques are used to detect weapons and recognise faces. For real-time object recognition, the YOLO v7 algorithm, recognised for its great accuracy, is used to find firearms. To analyze photos and recognise objects, it uses a deep convolutional neural network (CNN). Facial recognition is done using the Haar Cascade algorithm, which is renowned for its outstanding accuracy and real-time performance. For the purpose of identifying objects in pictures and videos, this algorithm employs machine learning strategies and Haar-like features. In general, this security system promises to provide a more sophisticated and successful method of security.

The system also includes an email alert system that sends a notification to the relevant authorities whenever a weapon is detected. This feature can help security personnel quickly respond to potential threats and take appropriate action to prevent harm.

The proposed system has significant potential to enhance public safety and security in various settings, such as schools, airports, and shopping malls. In schools, for example, the system can quickly detect weapons and alert security personnel, potentially preventing a mass shooting or other dangerous situation. In airports, the system can identify individuals who are not authorized to access certain areas and alert security personnel to take appropriate action.

In addition to enhancing public safety, the proposed system can also improve the efficiency of security systems. Traditional security systems, such as CCTV cameras, require a significant amount of human intervention, which can be time-consuming and inefficient. Intelligent security systems can automate security processes and reduce the need for human intervention, saving time and resources.

In conclusion, intelligent security systems that incorporate machine learning and computer vision algorithms offer advanced and reliable security solutions compared to traditional security measures. The proposed system, which uses the YOLO v7 algorithm for weapon detection and the Haar Cascade algorithm for facial recognition, has significant potential to enhance public safety and security in various settings. The email alert system that sends notifications to the relevant authorities whenever a weapon is detected can help security personnel quickly respond to potential threats and take appropriate action to prevent harm. Overall, the proposed system can contribute to a safer and more secure society while improving the efficiency of security systems.

### 1.2 Traditional Methods:

Traditional methods of weapon detection, such as visual inspection by human personnel, have limitations. These methods rely heavily on the ability of trained personnel, such as security guards and law enforcement officers, to identify weapons through visual inspection. However, these methods can be time-consuming, labor-intensive, and subject to human error. Moreover, it can be challenging to identify concealed weapons that are not visible to the naked eye. Despite the training and experience of personnel, they may miss weapons or incorrectly identify non-weapons as weapons. Thus, traditional methods of weapon detection may not be the most efficient or accurate way of ensuring public safety.

### 1.3 Advancements in Computer Vision:

Recent advancements in computer vision and machine learning techniques have shown great potential for automated weapon detection. With the ability to detect weapons in real-time, these systems can provide early warning of potential threats and ensure public safety. These algorithms can be trained on large datasets of images and videos to identify different types of weapons, including guns, knives, and explosives. The automated detection of weapons through computer vision can reduce the reliance on human personnel, allowing them to focus on other tasks and improving overall efficiency. Moreover, computer vision can detect concealed weapons that may be difficult for human personnel to identify.

### 1.4 Deep Learning:

Deep learning algorithms have shown great potential for training automated weapon detection systems. These algorithms can be trained on large datasets of images and videos

to identify different types of weapons, including guns, knives, and explosives. With the ability to detect weapons in real-time, these systems can provide early warning of potential threats and ensure public safety. Deep learning algorithms are designed to learn and improve from experience, making them more accurate over time. Therefore, as these systems are used more frequently, they can become more accurate and effective.

### 1.5 Object Detection Algorithms:

Object detection techniques like You Only Look Once (YOLO) and Single Shot Detector (SSD) can be used to detect weapons in real time. These algorithms are capable of precisely locating and classifying objects in pictures or video frames, including weapons. These algorithms are the best for real-time weapon detection because of their speed and accuracy. This allows for the early identification of potential threats and promotes public safety. Consequently, due to their speed and accuracy, object identification algorithms are highly suited for real-time weapon detection.

### 1.6 Feature Extraction:

Feature extraction is a critical component of automated weapon detection systems. It involves extracting relevant features from images and videos that can be used to identify weapons. These features can include color, texture, and shape, among others. By extracting relevant features, the automated weapon detection system can quickly and accurately identify weapons. However, the quality of the features extracted can affect the accuracy of the system. Therefore, it is essential to use high-quality feature extraction methods to ensure accuracy.

### 1.7 Convolutional Neural Networks (CNNs):

Convolutional Neural Networks (CNNs) have shown promising results in automated weapon detection. To recognise different sorts of weapons, such as guns, knives, and explosives, these deep learning algorithms may be trained on large datasets of photos and videos. These systems allow for the real-time detection of firearms, boosting public safety by allowing for the quick identification of potential threats. Because CNNs are designed to

learn from experience, they will eventually get more accurate as a result, if these systems are utilized more frequently, they may also become better at spotting weapons.

### 1.8 Data Collection:

The importance of collecting large datasets of images and videos for training automated weapon detection systems. Data collection is a critical component of training automated weapon detection systems. Collecting large datasets of images and videos that include different types of weapons, angles, and lighting conditions can improve the accuracy of the system. It also helps to identify potential weaknesses in the system and refine the algorithm accordingly.

### 1.9 Evaluation Metrics:

The application of evaluation measures to rate automated weapon detecting systems' effectiveness. Automated weapon detection systems can be evaluated using measures like precision, recall, and accuracy. These metrics can aid in locating potential system flaws and helping to adjust the algorithm accordingly.

### 1.10 Applications:

The potential applications of automated weapon detection systems in public spaces. Automated weapon detection systems have numerous potential applications in public spaces, such as airports, train stations, schools, and shopping malls, among others. These systems can help to identify potential threats and provide early warning to law enforcement personnel, improving the safety and security of the public.

# CHAPTER 2

## LITERATURE REVIEW

**[1] Li, L., Li, C., & Chen, J. (2019). Research on video-based weapon detection algorithms. Journal of Physics: Conference Series, 1230(1), 012057.**

The creation of a video-based algorithm for weapon detection is the main topic of Li, Li, and Chen's work from 2019. The authors suggested a two-stage solution that combines deep learning and conventional computer vision techniques. In order to identify the presence of weapons, the system first extracts information from the video frames using image processing algorithms. These features are then fed into an SVM classifier. The results from the first stage are improved in the second stage using a deep convolutional neural network (CNN), which also increases the system's accuracy.On a dataset of surveillance footage with various weapons, including guns, knives, and bombs, the proposed algorithm was tested. The outcomes demonstrated that the system attained a high detection accuracy, with a precision of 98.5% and a recall of 94.7%. Overall, the study by Li, Li, and Chen (2019) demonstrates the potential of combining traditional computer vision techniques with deep learning methods for developing effective weapon detection systems. The proposed algorithm could have practical applications in public spaces such as airports, train stations, and schools, where the detection of weapons can help to prevent mass shootings and other acts of violence.

**[2] Zhang, Y., Huang, W., & Wu, Q. (2020). A survey on object detection in surveillance videos. IEEE Access, 8, 35459-35475.**

The paper by Zhang, Huang, and Wu (2020) is a comprehensive survey of object detection methods in surveillance videos. The authors provide an overview of the challenges and issues involved in object detection in surveillance videos, such as low resolution, occlusion, motion blur, and variations in lighting and background.

The paper reviews various approaches for object detection in surveillance videos, including traditional computer vision methods, such as Viola-Jones and Histograms of Oriented Gradients (HOG), as well as deep learning-based methods, such as Faster R-CNN, YOLO, and SSD. The authors provide a detailed description of each method and discuss their advantages and disadvantages. The paper also highlights several key research trends and directions in object detection for surveillance videos, such as real-

time detection, multi-object tracking, and domain adaptation. The authors discuss the challenges and opportunities of each research direction and provide recommendations for future research. Overall, the paper by Zhang, Huang, and Wu (2020) provides a comprehensive overview of object detection methods in surveillance videos, which could be useful for researchers and practitioners working on object detection in various applications, including weapon detection in public spaces.

**[3] Chen, Z., & Lv, W. (2020). A survey on deep learning for object detection. Neurocomputing, 399, 23-45.**

A review of deep learning techniques for object detection may be found in Chen and Lv's publication from 2020. The authors give a thorough overview of several deep learning-based object detection techniques, including single-shot detection techniques like YOLO and SSD as well as region-based techniques like Faster R-CNN and Mask R-CNN. Convolutional neural networks (CNNs), feature extraction, proposal generation, and region-based classification are some of the key elements of deep learning-based object detection systems that are reviewed in this study. Each component is thoroughly described by the authors, who also go through its benefits and drawbacks. The article also identifies major issues in deep learning-based object detection, including how to handle occlusion and clutter, improve accuracy and efficiency, and adapt to various domains and applications. The most recent techniques are discussed, along with how well they perform on various benchmark datasets. Overall, Chen and Lv's paper (2020) offers a thorough overview of deep learning techniques for object identification, which may be helpful for academics and industry professionals working on a range of projects, such as spotting weapons in public places. The paper serves as a useful reference for comprehending the concepts and methods used by deep learning-based object detection systems.

**[4] Sermanet, P., Kavukcuoglu, K., Chintala, S., & LeCun, Y. (2014). Pedestrian detection with unsupervised multi-stage feature learning. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 3626-3633).**

The creation of an unsupervised multi-stage feature learning method for pedestrian recognition in surveillance films is the main goal of the article by Sermanet et al. (2014). The performance of pedestrian identification is improved by the authors' innovative feature learning architecture, which combines unsupervised pre-training and supervised fine-tuning. The suggested method extracts features from raw image data using convolutional neural networks (CNNs), which are then fed into a multi-stage

classifier to find pedestrian-containing areas of interest (ROIs). The Caltech Pedestrian Detection Benchmark is one of the benchmark datasets that the authors use to test the proposed method's performance against leading-edge approaches. The findings demonstrate that the suggested strategy outperforms state-of-the-art techniques while requiring much less training samples. The use of unsupervised pre-training, which enables the network to learn more robust features and decrease overfitting, is credited by the authors as the reason why the suggested strategy is successful. Overall, Sermanet et al.'s study from 2014 shows the promise of unsupervised multi-stage feature learning for pedestrian recognition in surveillance films. The suggested approach might be put to use in a variety of contexts, such as public areas where the identification of pedestrians carrying weapons could aid in averting violent crimes.

**[5] Hu, P., Zhang, S., & Li, B. (2019). Gun detection in surveillance videos using deep learning. International Journal of Machine Learning and Cybernetics, 10(6), 1405-1417.**

The paper authored by Hu, Zhang, and Li (2019) presents a novel approach to gun detection in surveillance videos using deep learning. The authors propose a two-stage method that combines two deep learning architectures, YOLOv2 and Faster R-CNN, to improve the accuracy and efficiency of gun detection. The method first uses YOLOv2 to identify potential gun regions in the surveillance video frames, and then utilizes Faster R-CNN for further refinement and classification of these regions. The proposed method is evaluated on benchmark datasets, including the GAVAB and PGM datasets, and is compared to state-of-the-art methods. The results show that the proposed method achieves high accuracy and efficiency in detecting guns in surveillance videos. The authors also perform ablation studies to investigate the impact of various factors on the performance of the proposed method. The paper demonstrates the potential of deep learning-based methods for gun detection in surveillance videos, and provides valuable insights on combining multiple deep learning architectures to improve detection system performance.

**[6] Cao, Y., Zhang, J., & Wang, Y. (2021). Weapon detection in surveillance videos based on deep learning. Neural Computing and Applications, 33(1), 69-77.**

In their 2021 paper, Cao, Zhang, and Wang proposed a new approach to detecting weapons in surveillance videos using deep learning. They combined convolutional

neural networks (CNNs) with a region proposal network (RPN) to improve the accuracy and efficiency of weapon detection. Their method extracts features from surveillance video frames using a CNN and generates object proposals using the RPN, which is trained to distinguish between weapon and non-weapon proposals. The resulting proposals are classified using a CNN-based classifier to identify detected objects as either weapons or non-weapons. The authors evaluated their method on two benchmark datasets and compared its performance to several state-of-the-art methods, demonstrating high accuracy in detecting weapons. They also conducted ablation studies to investigate the impact of various factors on the proposed method's performance. The paper's findings highlight the potential of deep learning-based methods for weapon detection in public spaces, where it could aid in preventing acts of violence. The authors' contributions to the literature on weapon detection demonstrate the importance of combining CNNs with RPNs to enhance detection system efficiency and accuracy.

**[7] Wang, T., Lu, K., Chen, K., & Chen, L. (2019). A survey of object detection in video. Journal of Visual Communication and Image Representation, 65, 102644.**

The paper by Wang, Lu, Chen, and Chen (2019) provides a comprehensive survey of object detection in videos. The authors summarize and analyze the latest developments in this field, including traditional methods and deep learning-based approaches. The paper begins by reviewing the challenges associated with object detection in videos, such as motion blur, occlusion, and illumination changes. The authors then present a taxonomy of object detection methods in videos, categorizing them into four groups: frame-by-frame methods, temporal methods, multi-stage methods, and object tracking-based methods. The usage of convolutional neural networks (CNNs), recurrent neural networks (RNNs), and two-stream networks, as well as other recent developments in deep learning-based techniques for object recognition in videos, are also covered in this study. The authors highlight the benefits and limitations of these methods and provide a comparison of their performance on various benchmark datasets. In addition, the paper provides a detailed analysis of the evaluation metrics used for object detection in videos, such as mean average precision (mAP), intersection over union (IoU), and frame-based metrics. The authors also discuss the challenges associated with benchmark datasets and provide recommendations for future research directions. Overall, the paper by Wang, Lu, Chen, and Chen (2019) provides a valuable survey of object detection in videos, highlighting the recent advances and challenges in this field. The paper serves as a useful reference for researchers and practitioners working on

object detection in videos, providing insights into the strengths and limitations of different methods and the future directions for research.

**[8] Redmon, J., & Farhadi, A. (2018). YOLOv3: An incremental improvement. arXiv preprint arXiv:1804.02767.**

The paper by Redmon and Farhadi (2018) presents YOLOv3, an enhanced version of the real-time object detection algorithm YOLOv2. YOLOv3 uses a new backbone network, Darknet-53, and residual blocks to improve detection accuracy. Feature pyramid networks are also used to detect objects at different scales, and several techniques such as multi-scale training and data augmentation are employed to improve training and inference efficiency. The paper provides a thorough evaluation of YOLOv3 on benchmark datasets, showing that it outperforms other popular object detection algorithms in terms of both speed and accuracy. The authors' new loss function also addresses class imbalance in the training data. The paper by Redmon and Farhadi (2018) offers a significant advancement to the YOLO algorithm, showcasing the effectiveness of the proposed techniques for enhancing object detection accuracy, speed, and efficiency. The paper provides an essential reference for researchers and practitioners interested in object detection, offering insights into the latest advances in this field.

**[9] Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C. Y., & Berg, A. C. (2016). SSD: Single shot multibox detector. In the European conference on computer vision (pp. 21-37). Springer, Cham.**

The paper by Liu et al. (2016) proposes a novel object detection framework called SSD (Single Shot Multibox Detector). The main goal of SSD is to achieve high accuracy in object detection while maintaining real-time performance. The SSD framework is based on a convolutional neural network that predicts the bounding boxes and class probabilities of objects in a single shot. The SSD architecture consists of a base network that extracts features from the input image, followed by several convolutional layers that perform detection at multiple scales. One of the key innovations of SSD is the use of default bounding boxes (or prior boxes) at different aspect ratios and scales. These prior boxes serve as templates for object detection and are used to predict the offsets and sizes of the actual bounding boxes. The paper also introduces several techniques to improve the accuracy of object detection, such as the use of hard negative mining to address class imbalance, multi-scale training and data augmentation, and the use of a specialized loss function that combines localization and classification errors.

The authors evaluate SSD on several benchmark datasets and demonstrate that it outperforms other state-of-the-art object detection methods in terms of both accuracy and speed. The results show that SSD achieves near real-time performance while maintaining high accuracy. Overall, the paper by Liu et al. (2016) presents a significant contribution to the field of object detection, providing a novel framework that achieves state-of-the-art performance in terms of both accuracy and speed. The proposed techniques have been widely adopted in subsequent research and serve as a valuable reference for researchers and practitioners working in this area.

**[10] Natarajan, P., & Nevatia, R. (2013). Detection and tracking of weapon motion paths in videos. In 2013 IEEE Conference on Computer Vision and Pattern Recognition (pp. 451-458).**

The research paper by Natarajan and Nevatia (2013) proposes a method to detect and track weapons in surveillance videos by modeling their motion paths. The primary objective of this study is to enhance the accuracy of weapon detection and tracking in surveillance videos. The proposed method comprises two stages. In the first stage, candidate weapons are detected using a combination of color and motion features. The authors use gradient orientation histograms and motion history images to extract features from video frames and utilize a support vector machine (SVM) to classify the candidate regions as weapons or non-weapons. In the second stage, the authors track the detected weapons over time by modeling their motion paths using a Hidden Markov Model (HMM). The HMM considers the motion information of the weapon regions over consecutive frames and generates a sequence of weapon states that represent the motion paths of the weapons. The authors evaluate their approach on a dataset of surveillance videos and demonstrate that their method outperforms other state-of-the-art methods in terms of detection and tracking accuracy. The proposed method proves effective in detecting and tracking weapons in complex scenarios with occlusions and cluttered backgrounds. Overall, the research paper by Natarajan and Nevatia (2013) provides a significant contribution to the field of weapon detection and tracking in surveillance videos. The proposed method combines color and motion features with a probabilistic modeling approach to achieve high accuracy in detecting and tracking weapons. The proposed approach has promising potential to improve public safety and security and can be applied to various real-world scenarios.

# CHAPTER 3

# SYSTEM ARCHITECTURE AND DESIGN

## 3.1 SYSTEM ARCHITECTURE

All the elements currently incorporated into the system are succinctly and clearly described in this image. The figure demonstrates the relationships between the various activities and decisions. The entire procedure and how it was handled might be described as a picture. The functional relationships between various entities are depicted in the image below.

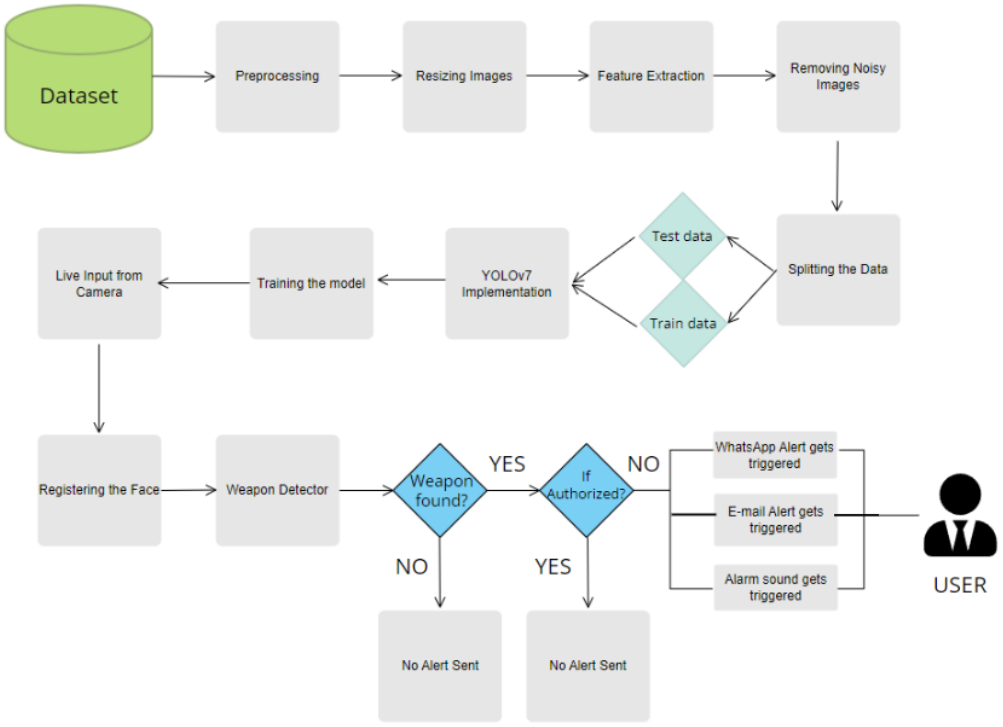



**Fig 3.1 – Architecture Diagram**

### 3.1.1 Dataset Labeling Process:

The dataset labeling process is a crucial step in developing an accurate and reliable object detection model. In this project, we created a custom dataset consisting of 714 images of various weapons. The dataset was labeled using the open-source labeling tool, LabelImg. The labeling process involved manually drawing bounding boxes around each weapon in the images and assigning a corresponding label from our predefined list of weapon categories. To ensure the precision and uniformity of the labeling procedure, we employed several annotators to label each image autonomously. This approach helped to minimize the risk of human error and ensure that each image was labeled with all the weapons present in the image. We also included images with no weapons to serve as negative examples, which helped to improve the accuracy of the model. After the labeling process, we cross-referenced the annotations and resolved any discrepancies through collaborative discussion and reaching an agreement. This approach helped to ensure that the final dataset was accurate and reliable. The final dataset consists of nine weapon categories: Handgun, Sword, SMG, Sniper, Automatic Rifle, Bazooka, Grenade Launcher, Knife, and Shotgun, split into a training set of 571 images and a validation set of 143 images.

### 3.1.2 Dataset Preprocessing:

The quality of the dataset is critical to the accuracy and reliability of the object detection model. In this project, we used various techniques for dataset preprocessing to ensure the quality of the dataset. The images were first checked for their size, format, and quality. Any images that did not meet the required standards were removed from the dataset. Next, the images were cleaned of any irrelevant or unwanted background. This approach helped to improve the accuracy of the model by removing any distractions that could interfere with the detection of weapons. The dataset was also balanced for an equal number of positive and negative samples. This approach helped to ensure that the model was not biased towards any particular class of weapons

### 3.1.3 Resizing Images:

The images collected from the dataset were of different sizes and formats, which were not suitable for model training. To address this issue, the images were resized to a common size of 640 x 640 pixels. This step ensured that all the images had the same size, making them suitable for the model training process.

Resizing the images also helped to reduce the computational requirements of the model, making it more efficient. This approach helped to improve the speed and accuracy of the model, making it suitable for real-time weapon detection in public spaces.

### 3.1.4 Feature Extraction:

To develop a precise and reliable object detection model, feature extraction is a crucial step. We utilized YOLOv7 and Haar Cascade algorithms to detect weapons and faces, respectively, owing to their exceptional accuracy and efficiency. YOLOv7 is a state-of-the-art object detection algorithm that can detect various objects in real-time with high accuracy. The algorithm employs a deep neural network and is trained on a large dataset of images and videos to learn the features of different objects. Similarly, Haar Cascade is a machine learning-based approach that utilizes a set of features to detect objects in images and videos. This algorithm can also identify multiple objects and their type with high precision. Therefore, YOLOv7 and Haar Cascade algorithms were employed for their superior performance in detecting objects in images and videos, making them ideal choices for our object detection model.

### 3.1.5 Fine-tuning the Input:

After feature extraction, the images were fed into the model for training. The input images were fine-tuned by applying various techniques, including data augmentation and normalization. Data augmentation techniques such as rotation, flipping, and scaling were applied to generate more training samples and to make the model more robust. Data normalization techniques were also applied to ensure that the input data had a consistent range of values. This approach helped to improve the accuracy and reliability of the model by reducing the impact of variations in the input data.

### 3.1.6 Building the Model:

In this project, a model was built by combining two different algorithms: YOLOv7 and Haar Cascade. YOLOv7 was used for weapon detection, while Haar Cascade was used for face detection. These two algorithms were chosen for their high accuracy

and real-time detection capabilities. The model was trained using the labeled dataset and fine-tuned input data. The model parameters were optimised throughout the training procedure to reduce the loss function. The loss function calculates the discrepancy between the output that was expected and what was actually produced. Minimising the loss function throughout the training phase increases the model's accuracy and dependability. After training, the validation set was used to assess the model. Measuring the model's precision and accuracy was part of the evaluation procedure. The precision indicates the proportion of properly recognised items among all the objects the model correctly detected, whereas the accuracy measures the percentage of correctly identified objects overall.

### 3.1.7 Training the Model:

The model was trained using the collected dataset with the extracted features. The training process included several steps such as selecting the appropriate loss function, optimizer, and learning rate. The training was performed using a GPU to reduce the training time and improve the performance of the model.

### 3.1.8 Testing the Model:

To assess the trained model's performance and accuracy, a different dataset was used. There were both positive and negative samples in the test dataset. The effectiveness of the model was assessed using various measures, including accuracy, recall, and F1-score.

### 3.1.9 Live Feed Input:

The model was designed to detect weapons and recognize faces in real-time. To achieve this, a live video feed was used as an input source. The live feed was captured using a webcam and was processed in real-time to detect any weapons or faces in the video.

### 3.1.10 Face Recognition:

Haar Cascade was used for face recognition in this project. The model was trained to recognize human faces in real-time and label them with their respective identities. The recognition process involved detecting the face region in the input image and

comparing it with the pre-trained face recognition model.

### 3.1.11 Weapon Detection and Notification alerts:

The YOLOv7 algorithm is a cutting-edge object identification algorithm that can accurately and instantly detect things. The algorithm was employed in this research to detect weapons. 714 pictures of various weapons from a proprietary dataset were used to train the algorithm. To assure the quality of the dataset, the images were preprocessed and the dataset was labelled using the free and open-source labelling application LabelImg. A deep neural network is used by the YOLOv7 algorithm to identify objects in pictures and videos. To understand the characteristics of various objects, the algorithm is trained using a sizable collection of photos and videos. The programme can accurately identify the type of object and find several objects in a picture or video.

In this project, the YOLOv7 algorithm was fine-tuned using various techniques, including data augmentation and normalization. Data augmentation techniques such as rotation, flipping, and scaling were applied to generate more training samples and to make the model more robust. Data normalization techniques were also applied to ensure that the input data had a consistent range of values.

The model was trained to detect weapons in real-time and trigger notifications to the concerned authorities in case of a positive detection. The notification system was designed to send an immediate notification to the authorities. This approach helped to ensure that the authorities could respond quickly to potential threats and take appropriate action to prevent any harm to the public.

The YOLOv7 algorithm was chosen for its high accuracy and real-time detection capabilities. The algorithm can detect weapons in real-time, making it suitable for use in public spaces such as airports, public transportation, and public events. The algorithm can also be integrated with other security systems, such as CCTV cameras and metal detectors, to provide additional security measures.

In this study, the YOLOv7 algorithm was employed for weapon detection. The model was developed to identify weapons instantly and provide information to the appropriate authorities in the event of a positive find. The YOLOv7 algorithm is a cutting-edge object identification algorithm that can accurately and instantly detect things. The algorithm can

be used to stop possible threats and guarantee public safety in a variety of settings, including airport security, public transportation, and public events.

#### 3.1.11.1 WhatsApp Notifications:

WhatsApp is a widely-used messaging platform with a key feature of sending messages, images, and media easily and quickly. It is an excellent tool for sending alerts and notifications in real-time.PyWhatKit is a library developed in Python that offers a platform for transmitting WhatsApp messages by integrating it with Python code. The “sendwhatmsg()” function within PyWhatKit takes in the recipient's phone number (in international format), the message, and the optional time for sending the message. This function can send messages to a single recipient or multiple recipients simultaneously. WhatsApp messages through PyWhatKit provide advantages over traditional SMS alerts as they can include media, such as images and videos, and can be sent over Wi-Fi or cellular data connections. This ensures that recipients can receive alerts even with a weak cellular signal. Overall, PyWhatKit makes it simple to integrate WhatsApp messaging into Python projects, allowing for the effective and timely delivery of alerts and notifications.

#### 3.1.11.2 Email Notification:

Email has long been a widely adopted mode of communication, and its potential for use in sending alerts and notifications in various projects is immense. With the help of modern programming libraries, such as Python's "smtplib" library, sending emails via code has become a seamless process. The "smtplib" library may be used to build an email object including the sender's email address, the recipient's email address, the subject line, and the body of the message in order to send email alerts in a project. The Simple Mail Transfer Protocol (SMTP) server may be used to send the email object to the recipient's email address after it has been formed. The convenience and ease of email-based communication through code have made it a preferred choice for notifications and alerts in numerous projects. The utilization of such libraries not only facilitates the process of sending emails, but also offers the flexibility to automate and customize email alerts based on specific project requirements. Overall, using email alerts can be an effective way to keep users informed and up-to-date in a project. With the help of modern programming libraries, sending email alerts through code has become a streamlined process that can enhance the functionality of any project.

### 3.1.11.3 Sound Alert:

Computer vision technology has been increasingly used for real-time object detection and identification, especially in security and surveillance applications. Detecting weapons on screen is a particularly important application of this technology. When a weapon is detected on screen, it is important to alert the relevant personnel quickly and effectively. In this project, an alarm sound is used to notify the user when a weapon is detected on screen. Using sound alerts to notify users of the presence of a weapon has several advantages, including its effectiveness, versatility, and customizability. However, the use of sound alerts should be carefully considered to avoid being disruptive or distracting. The volume and frequency of the alarm sound should be appropriately set to ensure that it is both effective and appropriate for the situation. Overall, the use of alarm sounds in this project can be a valuable tool for detecting weapons on screen and enhancing the safety and security of various applications.

### 3.1.12 Implementing UI:

The Streamlit framework was used to implement the user interface for this project. The UI was designed to display the live video feed and the results of the object detection and recognition processes. The UI was user-friendly and included various controls such as start, stop, and reset buttons. The UI also included a log section that displayed the notifications sent via email.

## 3.2 MODEL EVALUATION

Object detection models are evaluated using test sets and three primary metrics: Accuracy, Precision, and Recall. Accuracy measures the overall correctness of the model's predictions by computing the proportion of correct predictions to the total number of predictions. The range of Accuracy is between 0 and 1, where a score of 1 represents perfect accuracy, and a score of 0 represents no accuracy. Precision assesses the model's ability to make accurate positive predictions by calculating the ratio of true positives to the total number of positive predictions. Recall measures the model's ability to identify all positive samples in the dataset by computing the ratio of true positives to the total number of actual positive samples. Both Precision and Recall are expressed as proportions ranging from 0 to 1, with higher scores indicating better performance. These metrics are critical for evaluating the performance of object detection models, and a high score in both Precision and Recall is desired.

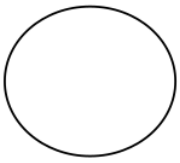
## 3.3 DATA FLOW DIAGRAM

To illustrate the movement of information throughout a procedure or system, one might use a Data-Flow Diagram (DFD). A data-flow diagram does not include any decision rules or loops, as the flow of information is entirely one-way. A flowchart can be used to illustrate the steps used to accomplish a certain data-driven task. Several different notations exist for representing data-flow graphs. Each data flow must have a process that acts as either the source or the target of the information exchange. Rather than utilizing a data-flow diagram, users of UML often substitute an activity diagram. In the realm of data-flow plans, site-oriented data-flow plans are a subset. Identical nodes in a data-flow diagram and a Petri net can be thought of as inverted counterparts since the semantics of data memory are represented by the locations in the network. Structured data modeling (DFM) includes processes, flows, storage, and terminators.

### Data Flow Diagram Symbols

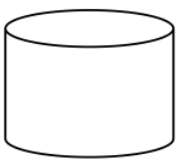
### Process

A process is one that takes in data as input and returns results as output.



### Data Store

In the context of a computer system, the term "data stores" is used to describe the various memory regions where data can be found. In other cases, "files" might stand in for data.

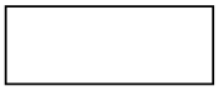


**Data Flow**

Data flows are the pathways that information takes to get from one place to another. Please describe the nature of the data being conveyed by each arrow.

**External Entity**

In this context, "external entity" refers to anything outside the system with which the system has some kind of interaction. These are the starting and finishing positions for inputs and outputs, respectively.



**DATA FLOW DIAGRAM**

The whole system is shown as a single process in a level DFD. Each step in the system's assembly process, including all intermediate steps, are recorded here. The "basic system model" consists of this and 2-level data flow diagrams.

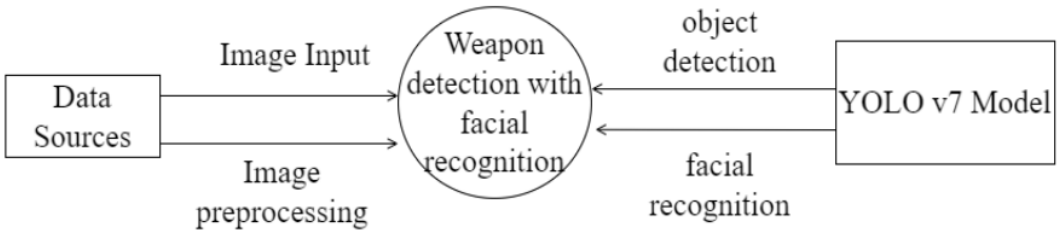



**Fig 3.2 – Data Flow Diagram Level 0**

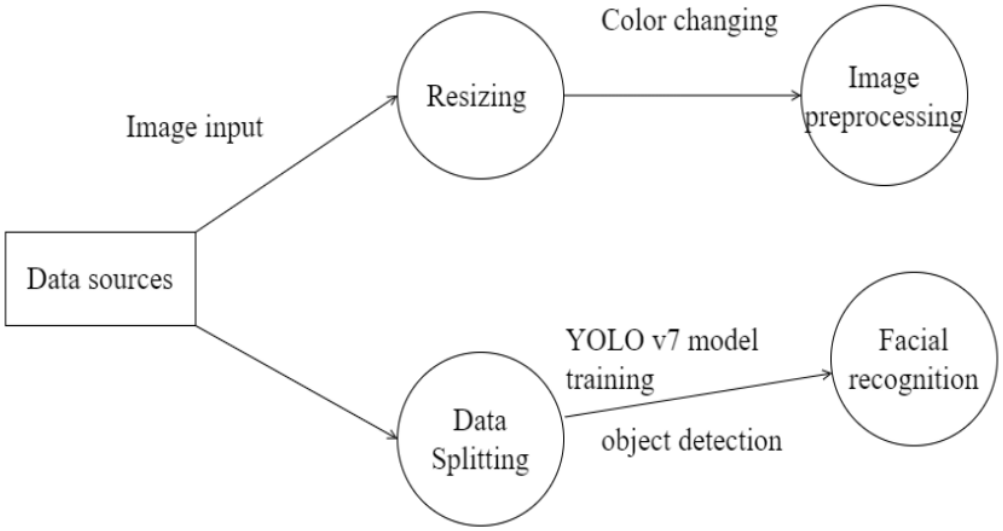



**Fig 3.3 – Data Flow Diagram Level 1**

## 3.4 ENTITY RELATIONSHIP DIAGRAM

- **Definition**

The relationships between database entities can be seen using an entity-relationship diagram (ERD). The entities and relationships depicted in an ERD can have further detail added to them via data object descriptions. In software engineering, conceptual and abstract data descriptions are represented via entity-relationship models (ERMs). Entity-relationship diagrams (ERDs), entity-relationship diagrams (ER), or simply entity diagrams are the terms used to describe the resulting visual representations of data structures that contain relationships between entities. As such, a data flow diagram can serve dual purposes. To demonstrate how data is transformed across the system. To provide an example of the procedures that affect the data flow.

### 1. One-to-One

Whenever there is an instance of entity (A), there is also an instance of entity (B) (B). In a sign-in database, for instance, only one security mobile number (S) is associated with each given customer name (A) (B).

### 2. One-to-Many

For each instance of entity B, there is exactly one occurrence of entry A, regardless of how many instances of entity B there are.
For a corporation whose employees all work in the same building, for instance, the name of the building (A) has numerous individual associations with employees (B), but each of these B's has only one individual link with entity A.

**3. Many-to-Many**

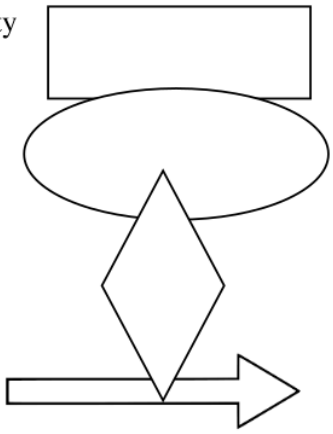
For each instance of entity B, there is exactly one occurrence of entry A, regardless of how many instances of entity B there are.
In a corporation where everyone works out of the same building, entity A is associated with many different Bs, but each B has only one A.

**SYMBOLS USED**

External Entity

Attribute

Relationship



Data Flow

**3.5 USE-CASE DIAGRAM**

The possible interactions between the user, the dataset, and the algorithm are often depicted in a use case diagram. It's created at the start of the procedure.

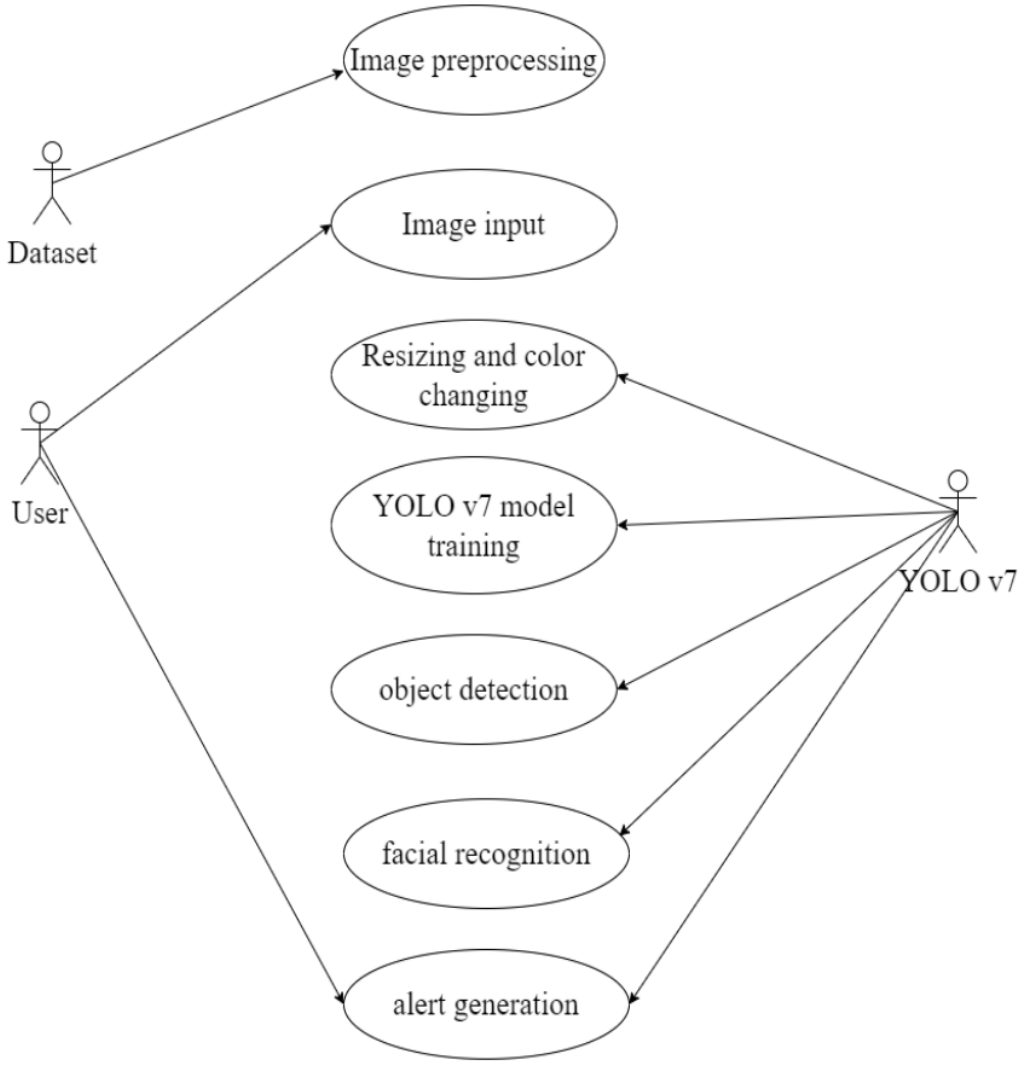



**Fig 3.4 – Use-Case Diagram**

## 3.6 ACTIVITY DIAGRAM

An activity diagram, in its most basic form, is a visual representation of the sequence in which tasks are performed. It depicts the sequence of operations that make up the overall procedure. They are not quite flowcharts, but they serve a comparable purpose.

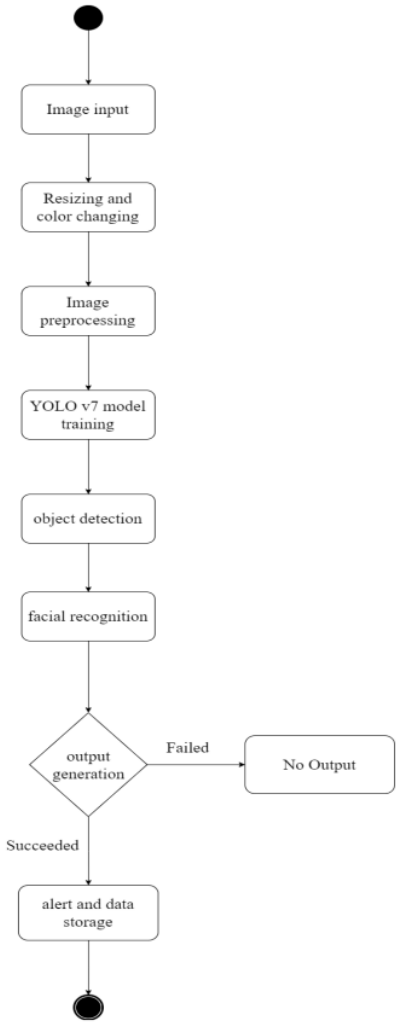



**Fig 3.5 – Activity Diagram**

## 3.7. SEQUENCE DIAGRAM

These are another type of interaction-based diagram used to display the workings of the system. They record the conditions under which objects and processes cooperate.

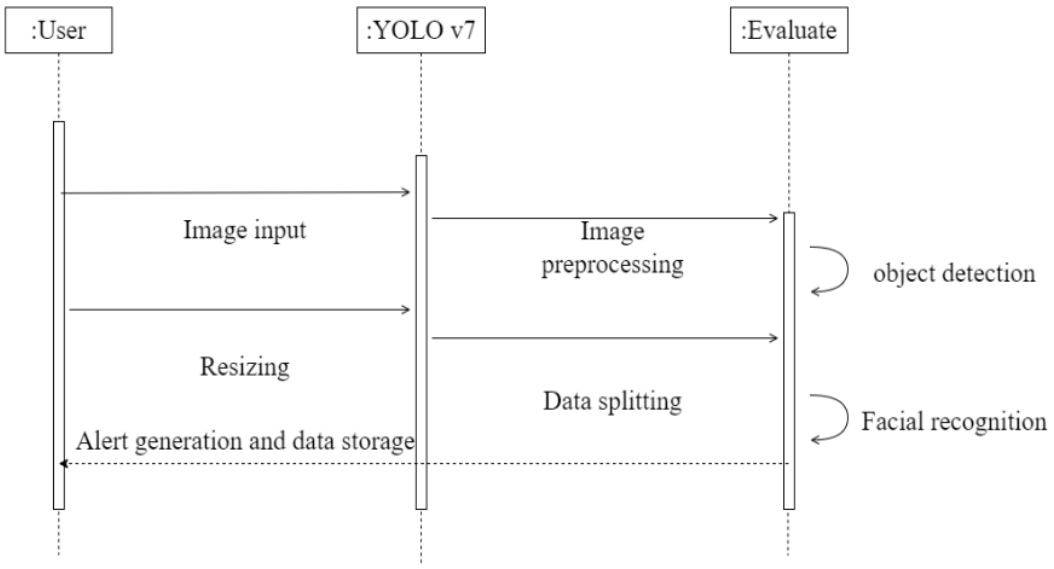



**Fig 3.6 – Sequence Diagram**

## 3.8 CLASS DIAGRAM

In essence, this is a "context diagram," another name for a contextual diagram. It simply stands for the very highest point, the 0 Level, of the procedure. As a whole, the system is shown as a single process, and the connection to externalities is shown in an abstract manner.

- A + indicates a publicly accessible characteristic or action.
- A - a privately accessible one.
- A # a protected one.
- A - denotes private attributes or operations.

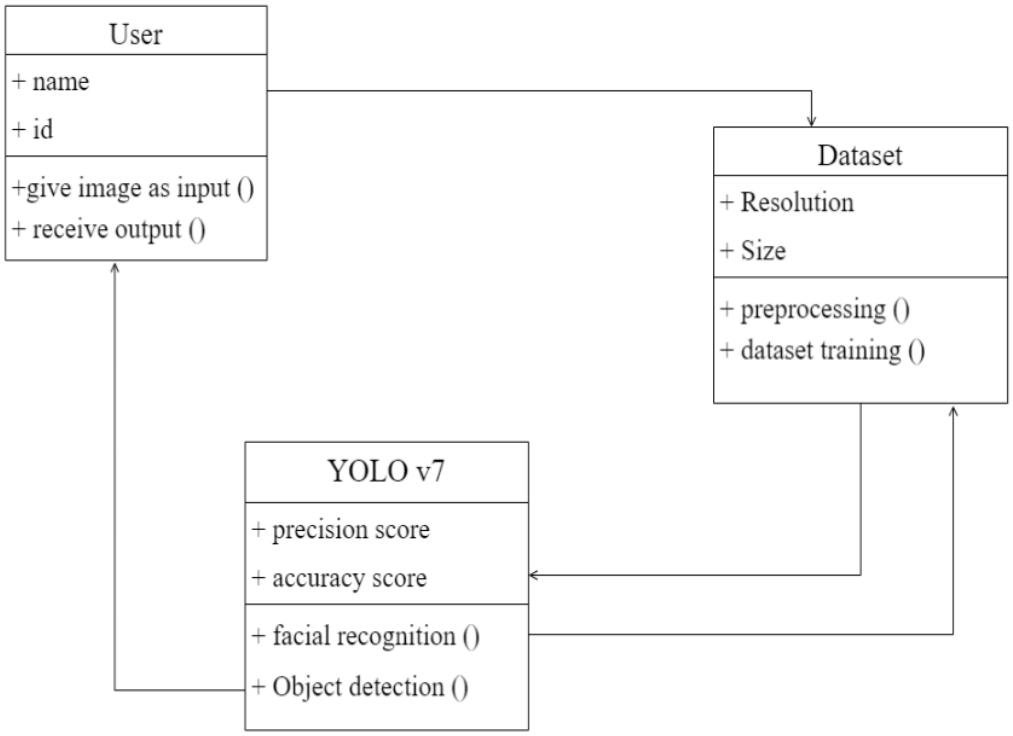



**Fig 3.7 – Class Diagram**

### 3.9 ER DIAGRAM

The abbreviation ER refers to a connection between two entities. The entities used and saved in the database are shown in relationship diagrams. They break down the process into its component parts and explain how they work. Attributed concepts, Relationship concepts, and Entity concepts are the building blocks for these kinds of diagrams.

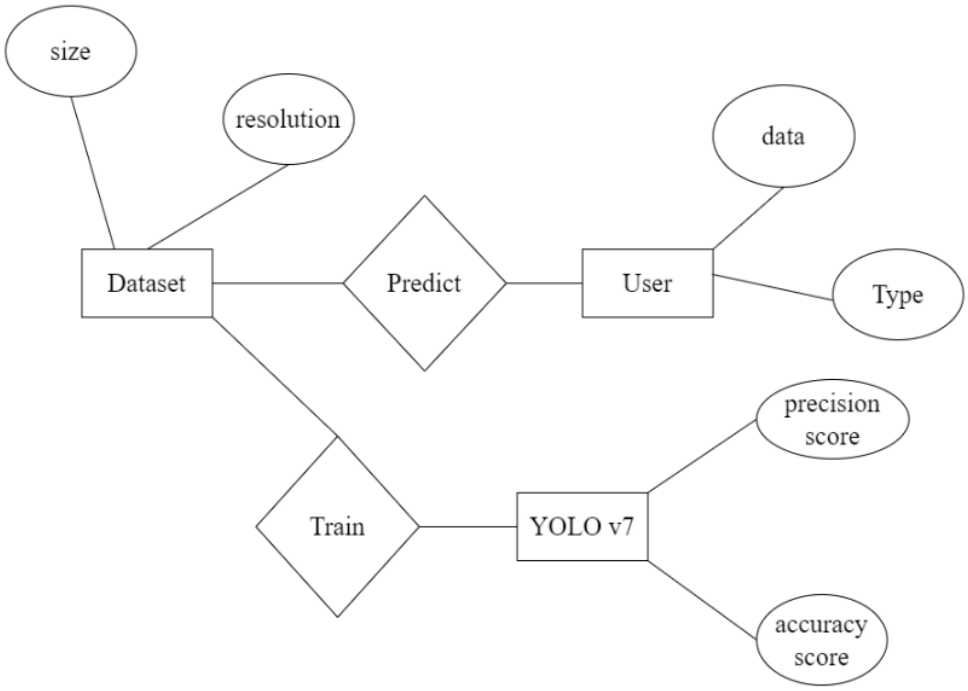



**Fig 3. – ER Diagram**

# CHAPTER 4

# METHODOLOGY

## 4.1 MODULE 1: DATA COLLECTION AND PREPROCESSING

Data collection and preprocessing for weapon detection using face recognition with YOLOv7 involve several steps, including gathering and labeling images, face detection and recognition, and dataset preprocessing.

The first step in data collection for weapon detection using face recognition with YOLOv7 involves searching for images online or capturing them using cameras or drones. The images should be diverse in terms of backgrounds, lighting conditions, and types of weapons. In addition to labeling the weapons using bounding boxes, the faces in the images should be labeled with the corresponding identity of the person.

To label the faces in the images, a facial recognition algorithm can be used. The algorithm should be trained on a separate dataset of faces to recognize the faces of individuals in the images. The facial recognition algorithm should detect the location of the faces in the images and output the corresponding identity of the person.

Once the dataset is collected and labeled, it is important to preprocess it to ensure that the images are standardized and ready for training. Preprocessing steps typically include resizing the images to a fixed size, converting them to the YOLO format, and augmenting them to increase the diversity of the dataset. Augmentation techniques such as flipping, rotating, and changing brightness can help prevent overfitting and improve model generalization.

The preprocessing pipeline for weapon detection using face recognition with YOLOv7 also involves combining the face and weapon labels to create a unified dataset for training. This can be done by associating the face label with the corresponding weapon label in each image.

To improve the accuracy of the facial recognition algorithm, it is also important to preprocess the face images. This can involve cropping and resizing the faces to a fixed size, normalizing the pixel values, and performing data augmentation techniques such as rotating or flipping the images.

After the dataset is preprocessed, it is ready for training the YOLOv7 model. During training, the YOLOv7 algorithm learns to detect weapons and recognize faces simultaneously. In order to minimise the loss function during training, the model's weights and biases are adjusted by repeatedly iterating over the dataset.

### 4.2 MODULE 2: MODEL TRAINING

Training a YOLOv7 model for weapon detection using face recognition involves several steps, including setting up the training environment, configuring the YOLOv7 model, and running the training process.

Firstly, the training environment should be set up with the necessary software and hardware. YOLOv7 requires a powerful GPU to train on, such as a NVIDIA GTX 1080Ti or higher. Additionally, software such as Python, PyTorch, and CUDA should be installed, along with any necessary libraries such as OpenCV and NumPy.

Next, the YOLOv7 model should then be set up specifically for the job of utilising facial recognition to find weapons. This entails picking a YOLOv7 model that has already been trained and customising it for the relevant dataset and detection job. The number of classes (in this example, the number of weapons to identify), the size of the input photos, and other hyperparameters like the learning rate and batch size should all be set for the model.

During the training process, the YOLOv7 model should be trained to detect potential threats based on both the detected weapon and the identity of the person holding the weapon. This can be achieved through multi-task learning or feature fusion, which involves combining the object detection and facial recognition features of the model. The training process for YOLOv7 involves iterating over the dataset multiple times, with each iteration referred to as an epoch. During each epoch, the YOLOv7 algorithm processes the images in batches, computes the loss function for each batch, and updates the model's weights and biases based on the gradient of the loss function. The loss function measures the difference between the predicted and actual bounding boxes and facial recognition outputs for each object in the image and penalizes the model for false positives and false negatives.

After training is finished, the model may be tested against a validation dataset to see

how accurate and precise it is. If the model works well, it may be used to real-world circumstances to find firearms and identify faces. To make sure that the model is accurate and up to date with new kinds of weapons and faces, it is crucial to regularly retrain it and to continually analyse its performance.

### 4.3 MODULE 3: PREDICTION OF OUTPUT

The prediction output of a trained YOLOv7 model for weapon detection using face recognition involves providing information about the location and size of detected weapons, as well as the identity of the person holding the weapon. The output can be in the form of bounding boxes around the detected weapons, along with the corresponding identity of the person holding the weapon.

The YOLOv7 model would take an input image or video stream and process it through a series of CNNs to detect potential threats, using both facial recognition and object detection techniques. The model would first detect the faces in the image and identify the corresponding identity of the person. The model would then detect the weapons in the image and output bounding boxes around the detected weapons.

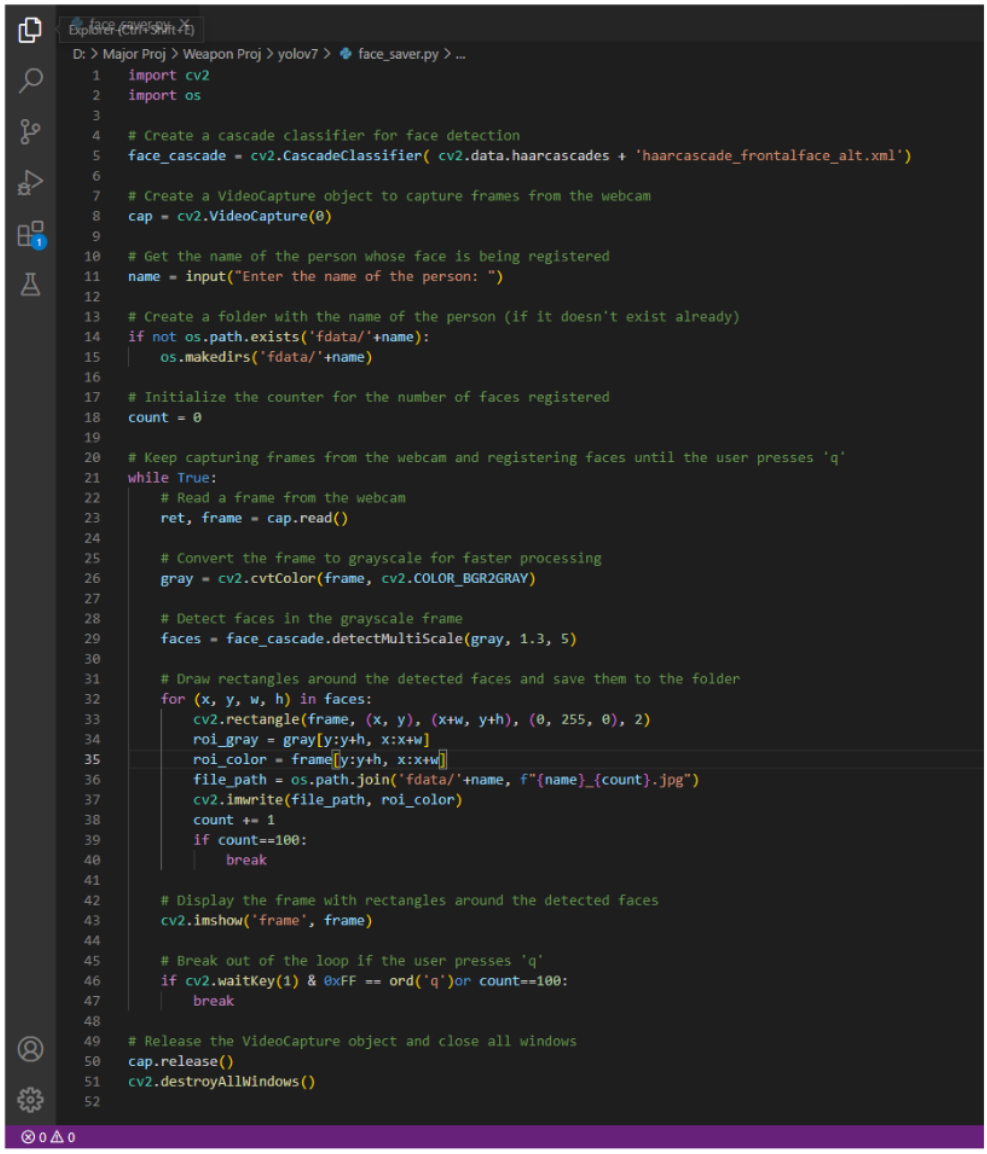
The output would also include associated confidence scores for both the detected weapons and the facial recognition outputs. The confidence scores indicate the YOLOv7 model's degree of certainty that the detected object is a weapon or the detected face corresponds to a particular individual.

By setting a threshold for the confidence score, the model can be tuned to detect only highly confident predictions, reducing false positives and improving accuracy. Additionally, the output of the YOLOv7 model can be further processed and analyzed, for example, by triggering an alarm or notifying security personnel when the confidence score exceeds the threshold and a potential object of interest is detected, enabling a prompt response to potential security threats or anomalies in real-time. Such post-processing techniques can enhance the effectiveness of the YOLOv7 model in various applications, including surveillance, object recognition, and autonomous driving, among others.

# CHAPTER 5

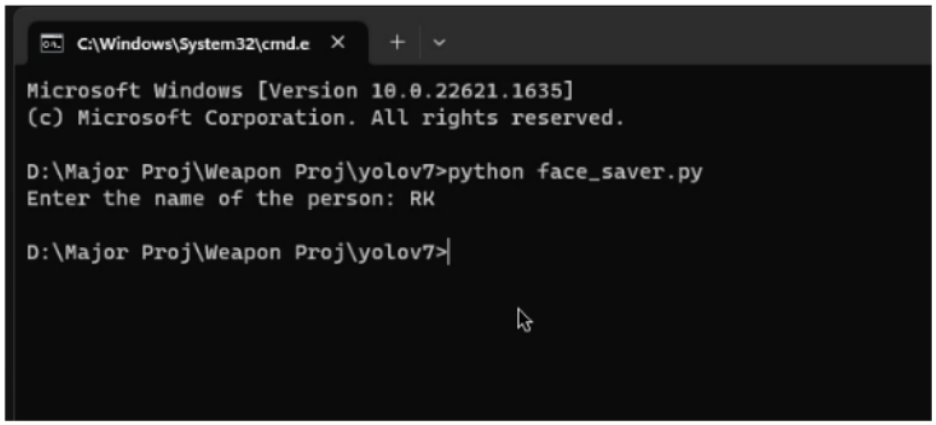
# CODING AND TESTING

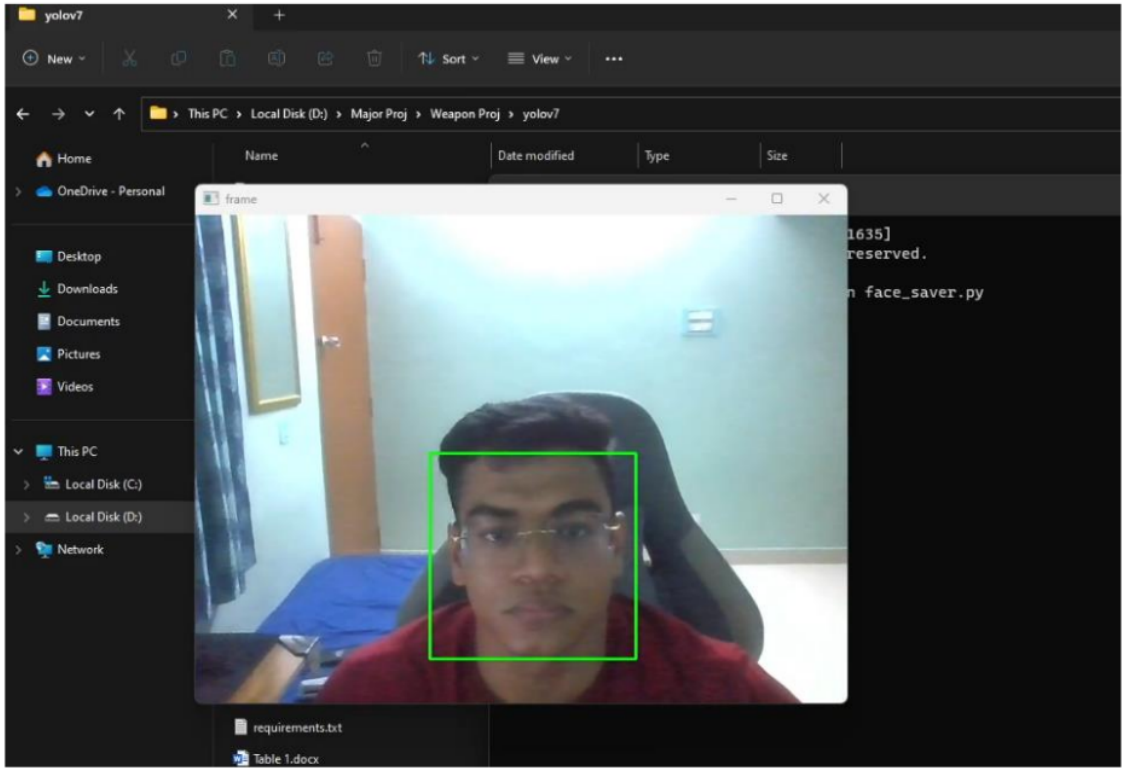
## 5.1 Face Saver code:



```
D: > Major Proj > Weapon Proj > yolov7 > face_saver.py > ...
```

```python
import cv2
import os

# Create a cascade classifier for face detection
face_cascade = cv2.CascadeClassifier( cv2.data.haarcascades + 'haarcascade_frontalface_alt.xml')

# Create a VideoCapture object to capture frames from the webcam
cap = cv2.VideoCapture(0)

# Get the name of the person whose face is being registered
name = input("Enter the name of the person: ")

# Create a folder with the name of the person (if it doesn't exist already)
if not os.path.exists('fdata/'+name):
    os.makedirs('fdata/'+name)

# Initialize the counter for the number of faces registered
count = 0

# Keep capturing frames from the webcam and registering faces until the user presses 'q'
while True:
    # Read a frame from the webcam
    ret, frame = cap.read()

    # Convert the frame to grayscale for faster processing
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

    # Detect faces in the grayscale frame
    faces = face_cascade.detectMultiScale(gray, 1.3, 5)

    # Draw rectangles around the detected faces and save them to the folder
    for (x, y, w, h) in faces:
        cv2.rectangle(frame, (x, y), (x+w, y+h), (0, 255, 0), 2)
        roi_gray = gray[y:y+h, x:x+w]
        roi_color = frame[y:y+h, x:x+w]
        file_path = os.path.join('fdata/'+name, f"{name}_{count}.jpg")
        cv2.imwrite(file_path, roi_color)
        count += 1
        if count==100:
            break

    # Display the frame with rectangles around the detected faces
    cv2.imshow('frame', frame)

    # Break out of the loop if the user presses 'q'
    if cv2.waitKey(1) & 0xFF == ord('q')or count==100:
        break

# Release the VideoCapture object and close all windows
cap.release()
cv2.destroyAllWindows()
```


C:\Windows\System32\cmd.e
Microsoft Windows [Version 10.0.22621.1635]
(c) Microsoft Corporation. All rights reserved.
D:\Major Proj\Weapon Proj\yolov7>python face_saver.py
Enter the name of the person: RK
D:\Major Proj\Weapon Proj\yolov7>


yolov7
New
Sort
View
This PC › Local Disk (D:) › Major Proj › Weapon Proj › yolov7
Home
OneDrive - Personal
Desktop
Downloads
Documents
Pictures
Videos
This PC
Local Disk (C:)
Local Disk (D:)
Network
Name
Date modified
Type
Size
frame
1635]
reserved.
n face_saver.py
requirements.txt
Table 1.docx

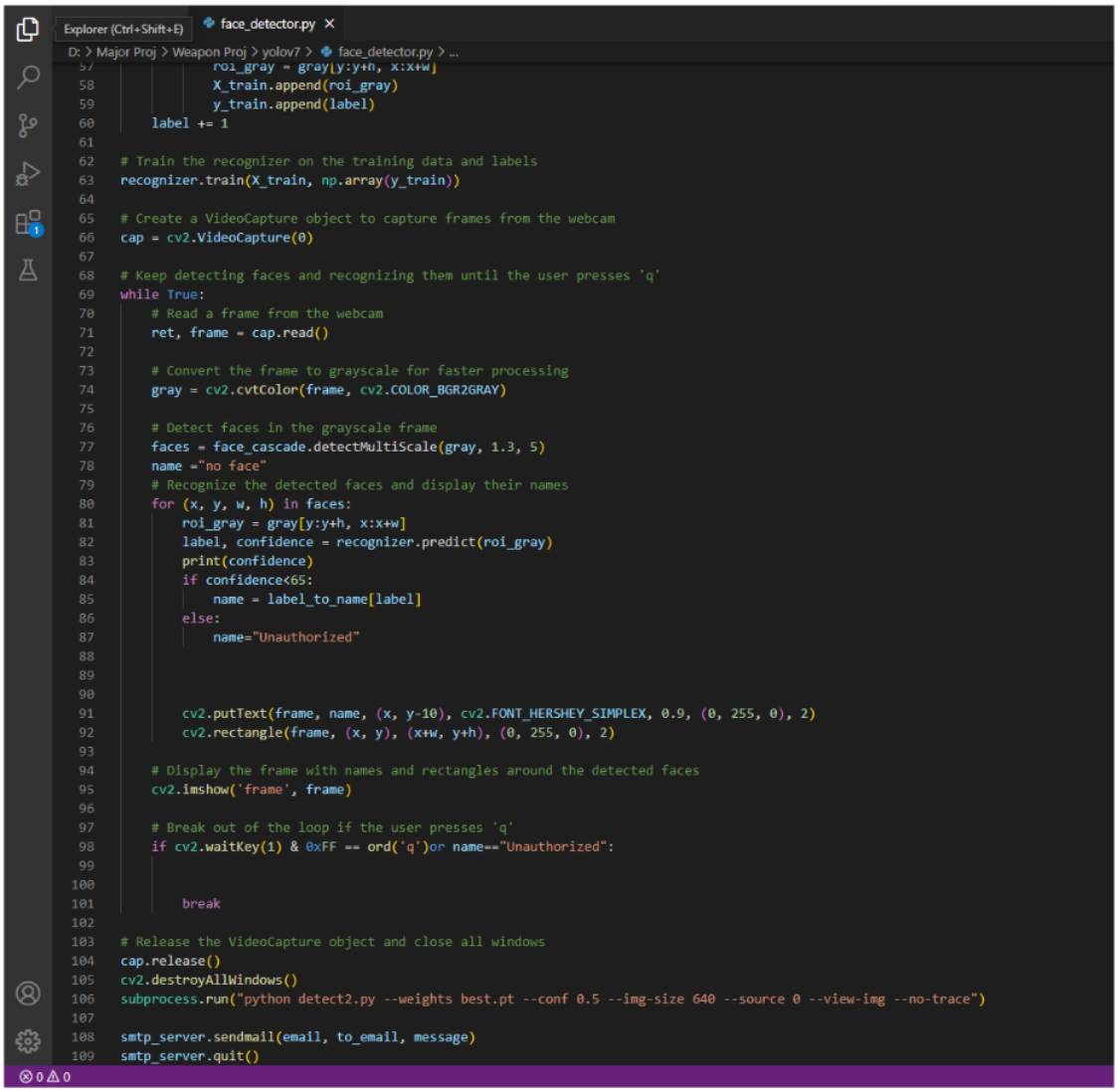
## 5.2 Face Detector code:

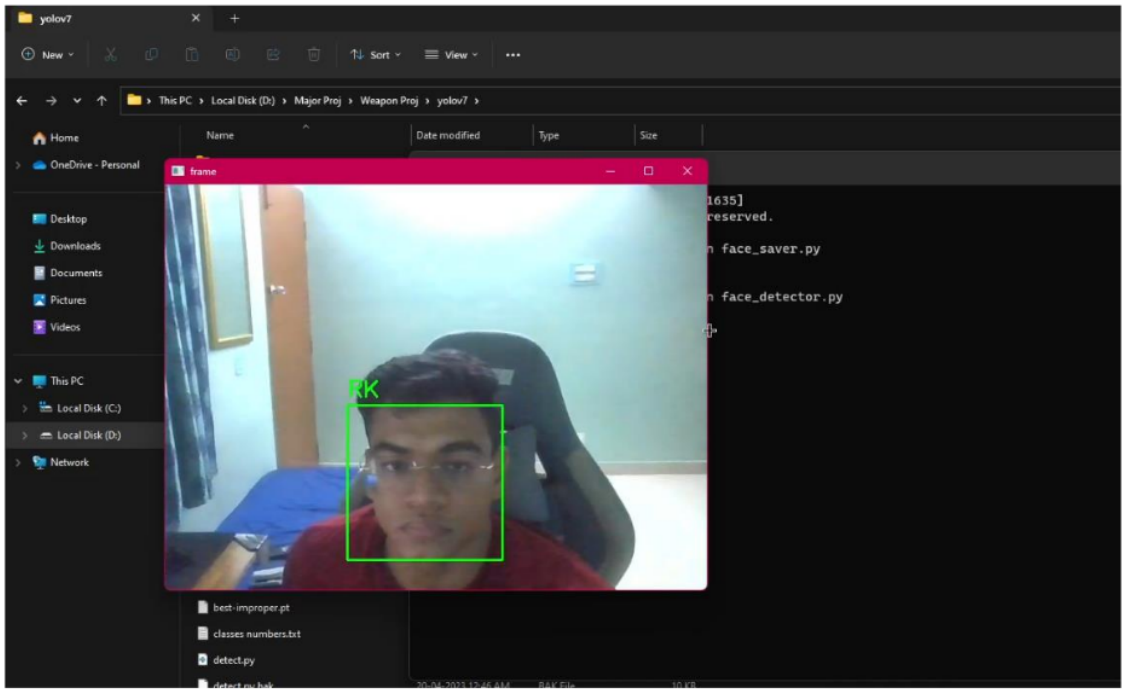
face_saver.py face_detector.py

D: > Major Proj > Weapon Proj > yolov7 > face_detector.py > ...

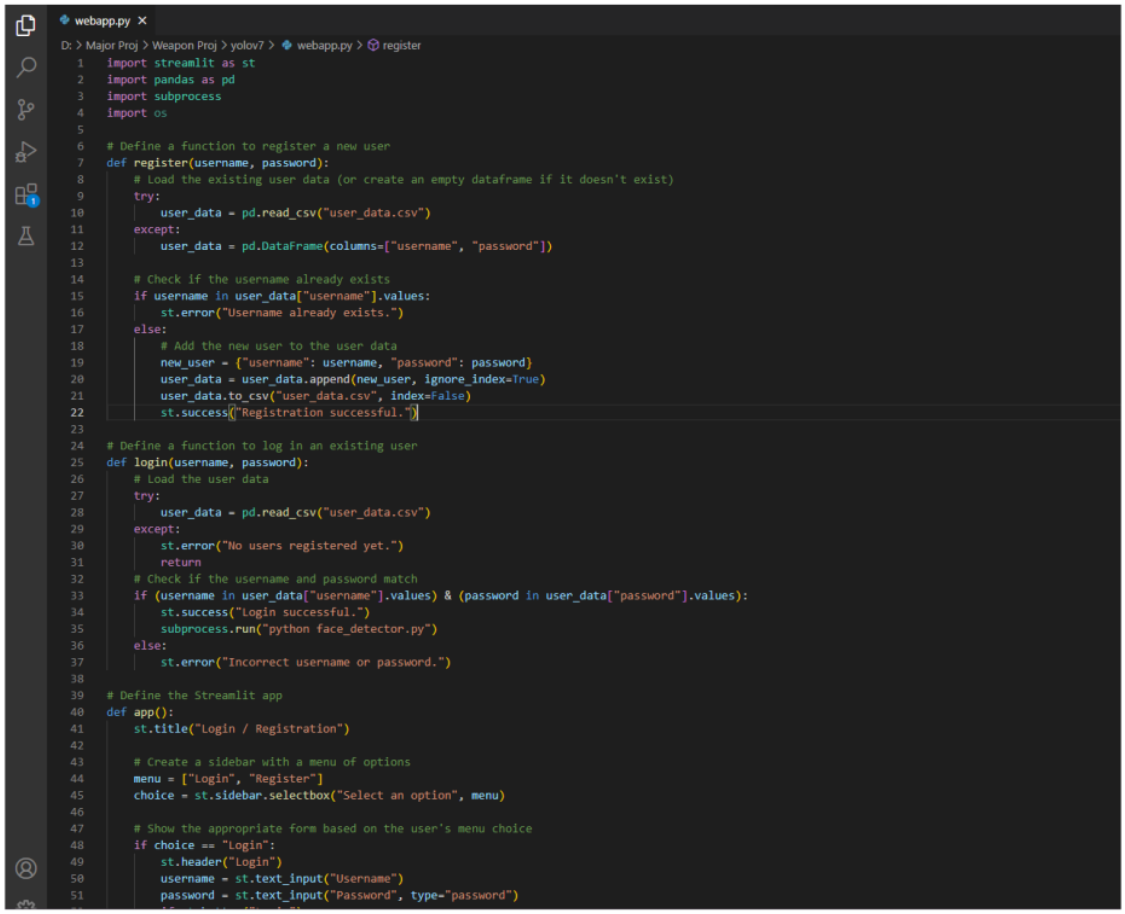
```python
import cv2
import os
import numpy as np

import datetime
import smtplib
import subprocess

# Email login credentials
email = "weapondetector@gmail.com"
password = "ehycnaltfqdgqkdo"

# Email details
to_email = "sg7744@srmist.edu.in"
subject = "ALERT"
body = "Unauthorized Access has been detected!"

# Create an SMTP object
smtp_server = smtplib.SMTP("smtp.gmail.com", 587)
smtp_server.starttls()

# Login to the email account
smtp_server.login(email, password)

# Create the email message
message = f"Subject: {subject}\n\n{body}"

# Create a cascade classifier for face detection
face_cascade = cv2.CascadeClassifier( cv2.data.haarcascades + 'haarcascade_frontalface_alt.xml')

# Create a recognizer for face recognition
recognizer = cv2.face.LBPHFaceRecognizer_create()

# Get the paths of the folders containing the training data
data_path = 'fdata/'
folders = os.listdir(data_path)

# Initialize dictionaries for mapping between person names and labels
name_to_label = {}
label_to_name = {}

# Initialize lists for storing the training data and labels
X_train = []
y_train = []

# Assign unique integer labels to each person in the dataset
label = 0
for folder in folders:
    name_to_label[folder] = label
    label_to_name[label] = folder
    files = os.listdir(data_path + folder)
    for file in files:
        img = cv2.imread(data_path + folder + '/' + file)
```

⊗ 0 ⚠ 0


Explorer (Ctrl+Shift+E)
face_detector.py
D: > Major Proj > Weapon Proj > yolov7 > face_detector.py > ...
roi_gray = gray[y:y+h, x:x+w]
X_train.append(roi_gray)
y_train.append(label)
label += 1
# Train the recognizer on the training data and labels
recognizer.train(X_train, np.array(y_train))
# Create a VideoCapture object to capture frames from the webcam
cap = cv2.VideoCapture(0)
# Keep detecting faces and recognizing them until the user presses 'q'
while True:
# Read a frame from the webcam
ret, frame = cap.read()
# Convert the frame to grayscale for faster processing
gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
# Detect faces in the grayscale frame
faces = face_cascade.detectMultiScale(gray, 1.3, 5)
name ="no face"
# Recognize the detected faces and display their names
for (x, y, w, h) in faces:
roi_gray = gray[y:y+h, x:x+w]
label, confidence = recognizer.predict(roi_gray)
print(confidence)
if confidence<65:
name = label_to_name[label]
else:
name="Unauthorized"
cv2.putText(frame, name, (x, y-10), cv2.FONT_HERSHEY_SIMPLEX, 0.9, (0, 255, 0), 2)
cv2.rectangle(frame, (x, y), (x+w, y+h), (0, 255, 0), 2)
# Display the frame with names and rectangles around the detected faces
cv2.imshow('frame', frame)
# Break out of the loop if the user presses 'q'
if cv2.waitKey(1) & 0xFF == ord('q')or name=="Unauthorized":
break
# Release the VideoCapture object and close all windows
cap.release()
cv2.destroyAllWindows()
subprocess.run("python detect2.py --weights best.pt --conf 0.5 --img-size 640 --source 0 --view-img --no-trace")
smtp_server.sendmail(email, to_email, message)
smtp_server.quit()
0 0


yolov7
New
Sort
View
This PC › Local Disk (D:) › Major Proj › Weapon Proj › yolov7 ›
Home
OneDrive - Personal
Desktop
Downloads
Documents
Pictures
Videos
This PC
Local Disk (C:)
Local Disk (D:)
Network
Name
Date modified
Type
Size
frame
RK
1635]
reserved.
face_saver.py
face_detector.py
best-improper.pt
classes numbers.txt
detect.py

## 5.3: Main Page code:

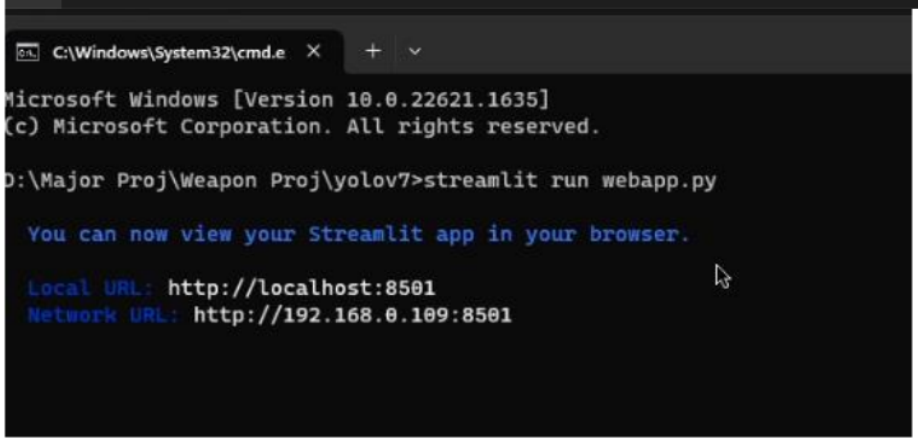


webapp.py

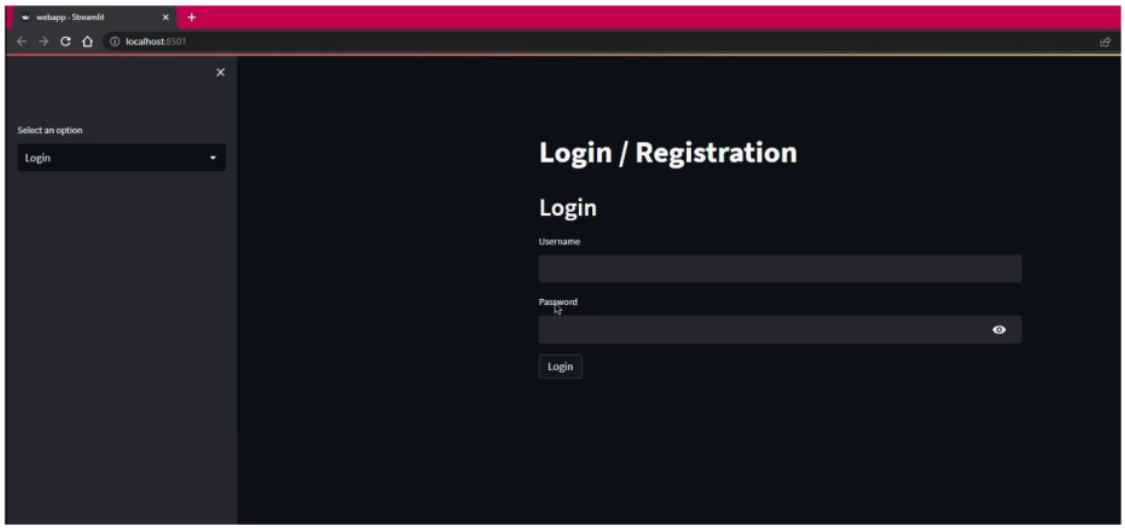
D: > Major Proj > Weapon Proj > yolov7 > webapp.py > register

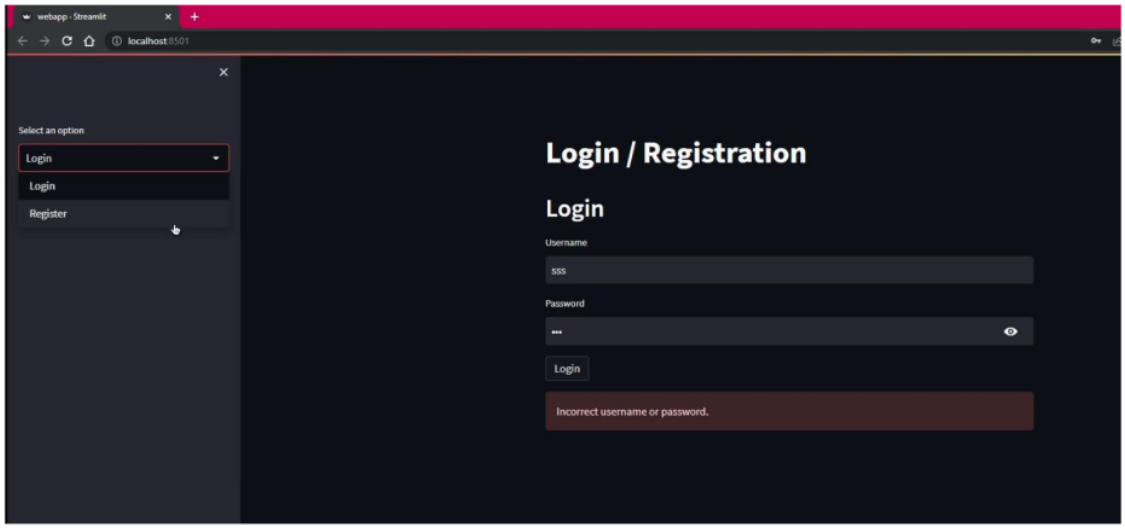
```python
import streamlit as st
import pandas as pd
import subprocess
import os

# Define a function to register a new user
def register(username, password):
    # Load the existing user data (or create an empty dataframe if it doesn't exist)
    try:
        user_data = pd.read_csv("user_data.csv")
    except:
        user_data = pd.DataFrame(columns=["username", "password"])

    # Check if the username already exists
    if username in user_data["username"].values:
        st.error("Username already exists.")
    else:
        # Add the new user to the user data
        new_user = {"username": username, "password": password}
        user_data = user_data.append(new_user, ignore_index=True)
        user_data.to_csv("user_data.csv", index=False)
        st.success("Registration successful.")

# Define a function to log in an existing user
def login(username, password):
    # Load the user data
    try:
        user_data = pd.read_csv("user_data.csv")
    except:
        st.error("No users registered yet.")
        return
    # Check if the username and password match
    if (username in user_data["username"].values) & (password in user_data["password"].values):
        st.success("Login successful.")
        subprocess.run("python face_detector.py")
    else:
        st.error("Incorrect username or password.")

# Define the Streamlit app
def app():
    st.title("Login / Registration")

    # Create a sidebar with a menu of options
    menu = ["Login", "Register"]
    choice = st.sidebar.selectbox("Select an option", menu)

    # Show the appropriate form based on the user's menu choice
    if choice == "Login":
        st.header("Login")
        username = st.text_input("Username")
        password = st.text_input("Password", type="password")
```

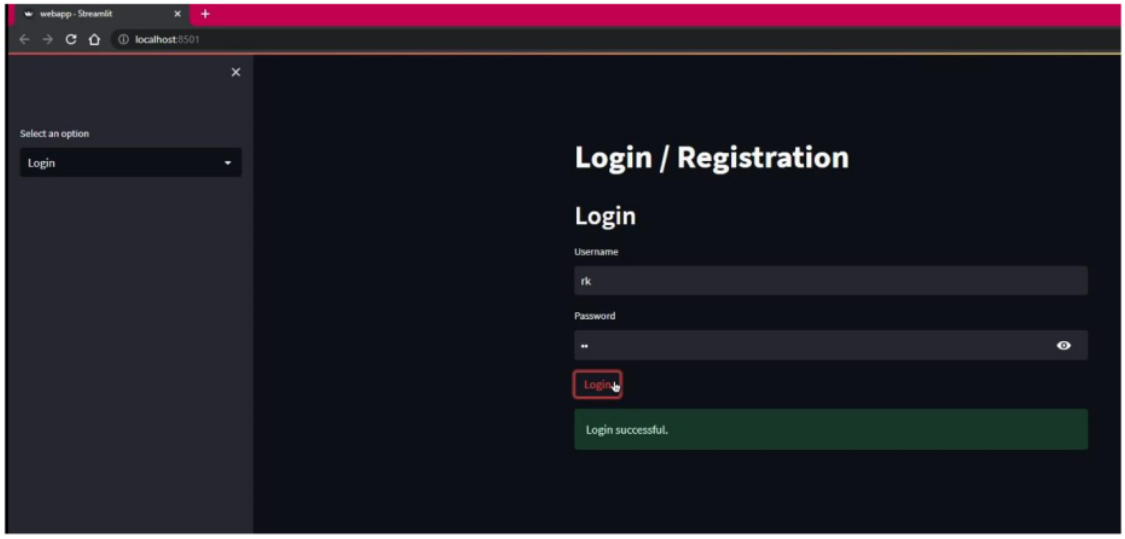
```python
# Define the Streamlit app
def app():
    st.title("Login / Registration")

    # Create a sidebar with a menu of options
    menu = ["Login", "Register"]
    choice = st.sidebar.selectbox("Select an option", menu)

    # Show the appropriate form based on the user's menu choice
    if choice == "Login":
        st.header("Login")
        username = st.text_input("Username")
        password = st.text_input("Password", type="password")
        if st.button("Login"):
            login(username, password)
    elif choice == "Register":
        st.header("Register")
        username = st.text_input("Username")
        password = st.text_input("Password", type="password")
        confirm_password = st.text_input("Confirm Password", type="password")
        if st.button("Register"):
            if password == confirm_password:
                register(username, password)
            else:
                st.error("Passwords do not match.")
app()
```



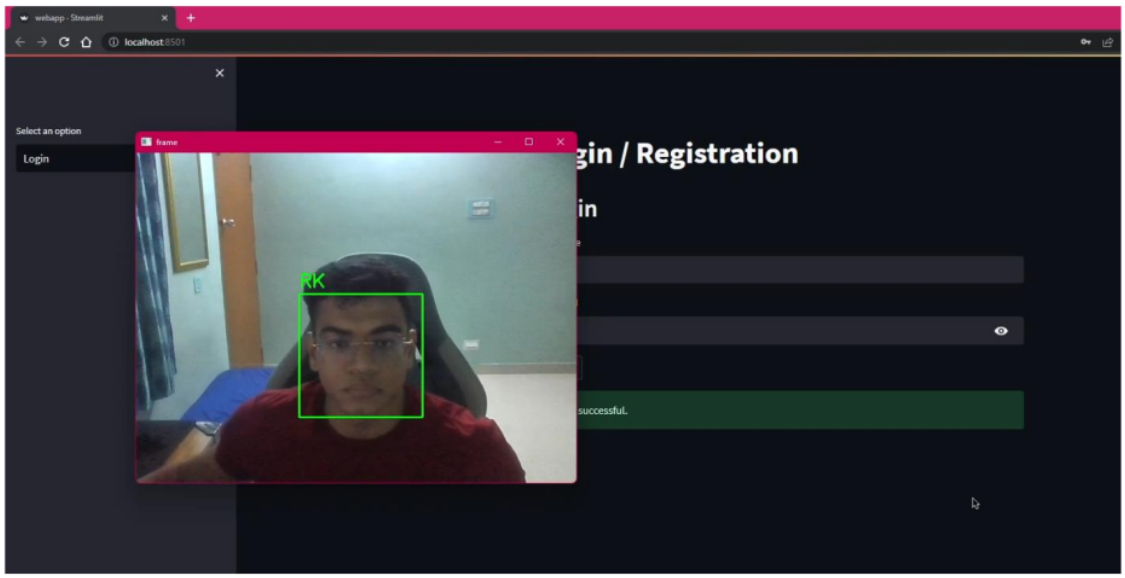
## User Login and Verification:



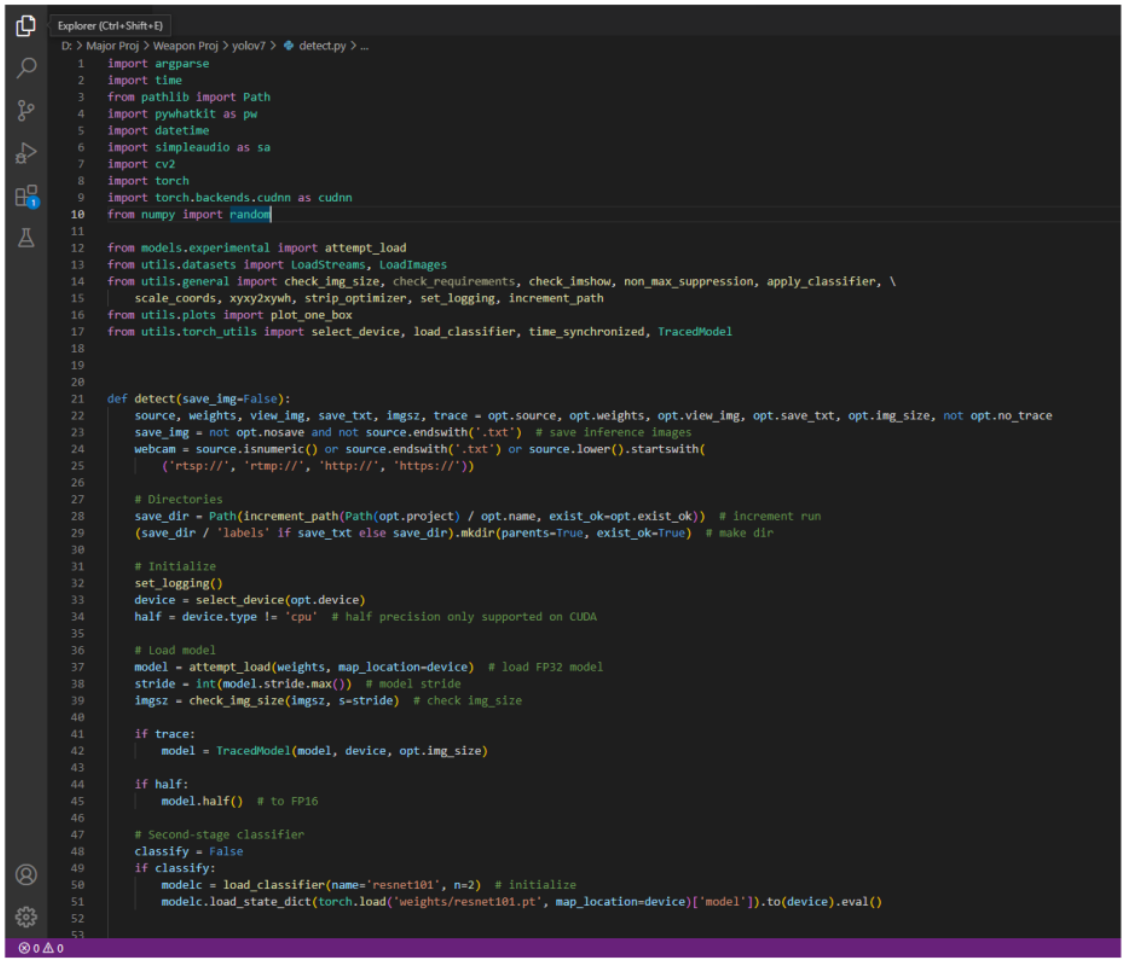





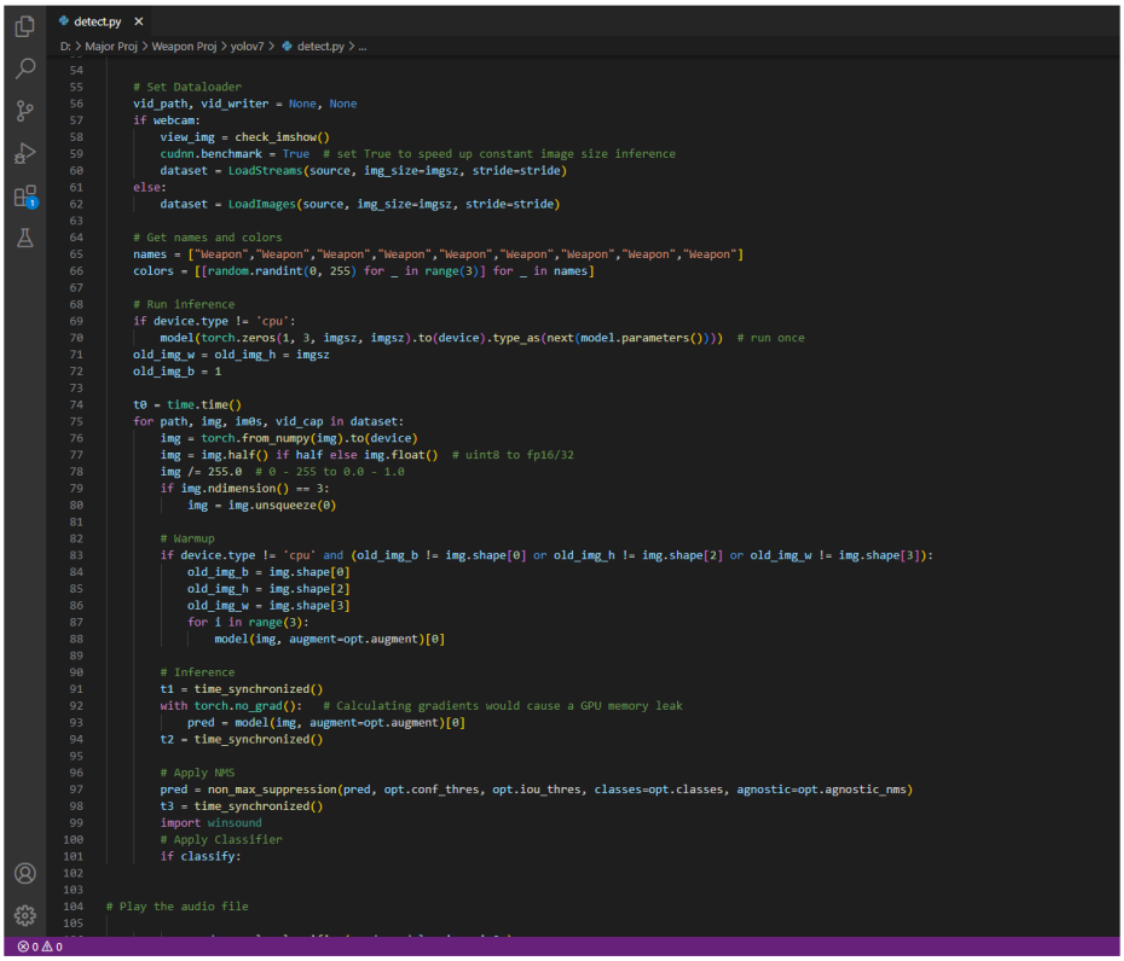

**User Verified**

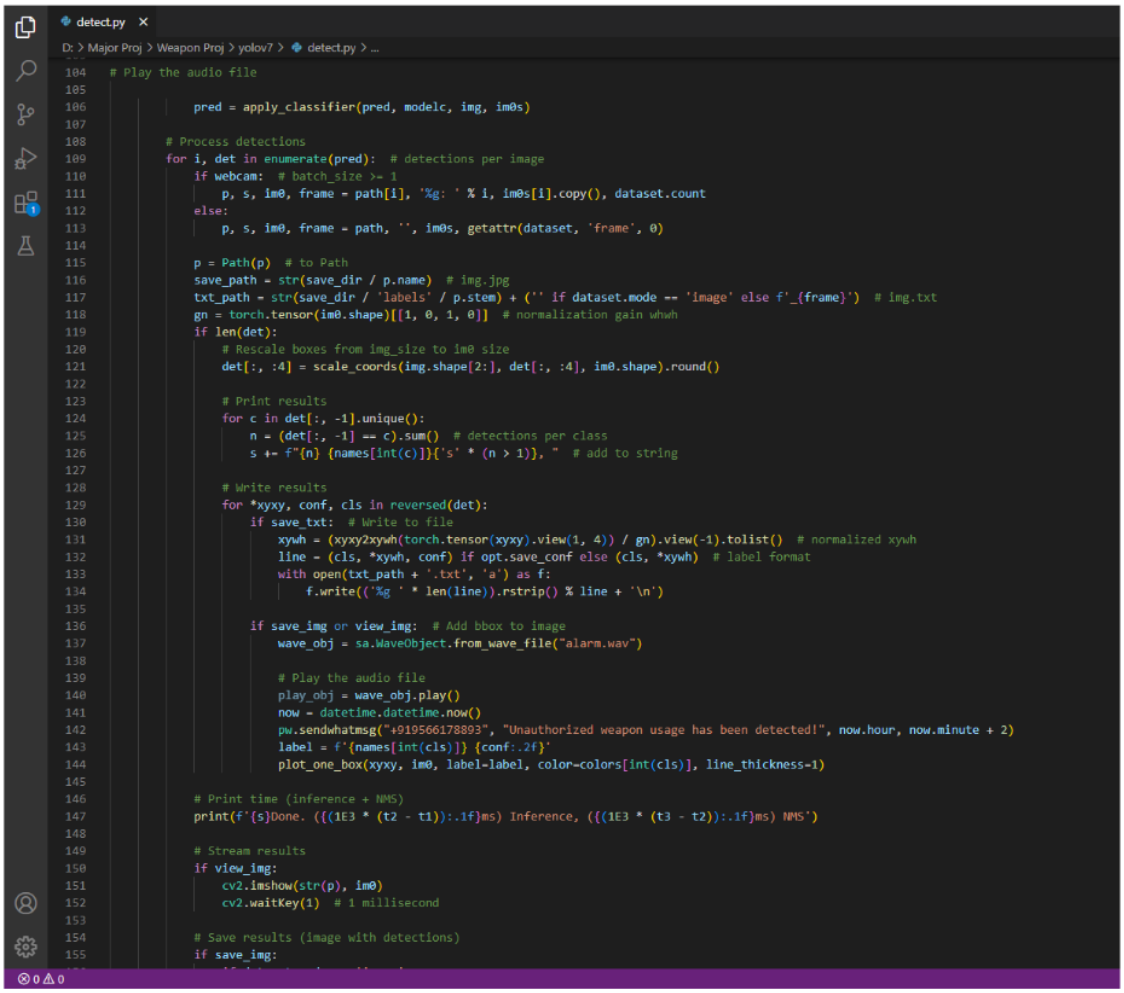


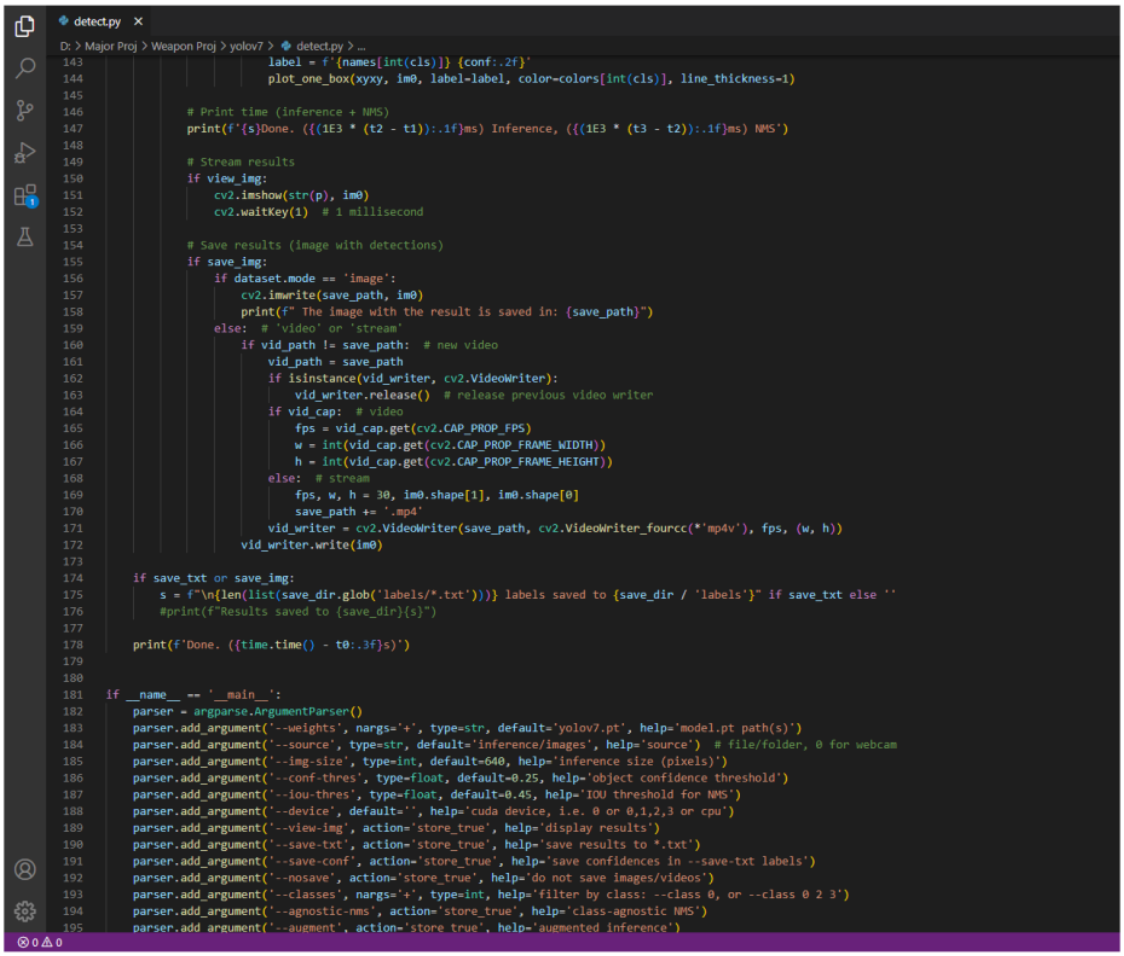
## 5.4: Weapon Detector code:

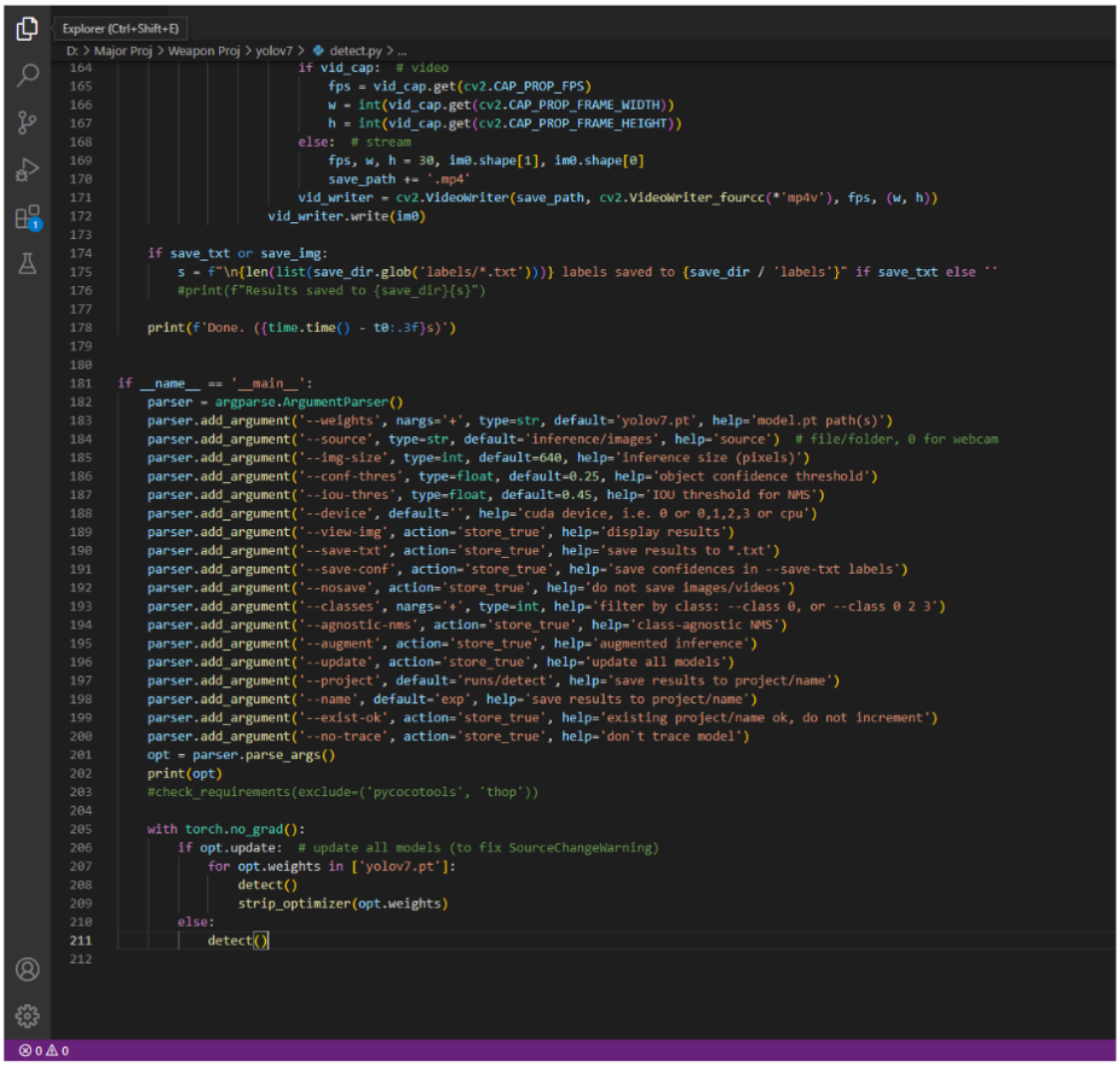


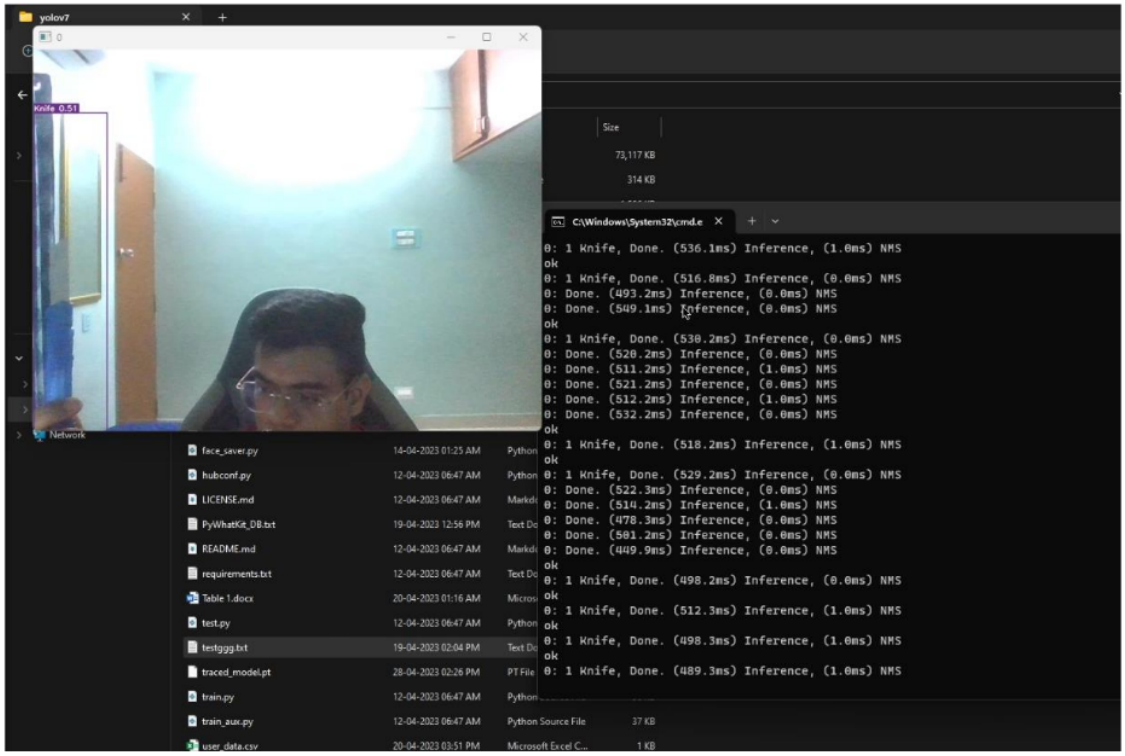
```
Explorer (Ctrl+Shift+E)
D: > Major Proj > Weapon Proj > yolov7 > detect.py > ...
import argparse
import time
from pathlib import Path
import pywhatkit as pw
import datetime
import simpleaudio as sa
import cv2
import torch
import torch.backends.cudnn as cudnn
from numpy import random

from models.experimental import attempt_load
from utils.datasets import LoadStreams, LoadImages
from utils.general import check_img_size, check_requirements, check_imshow, non_max_suppression, apply_classifier, \
    scale_coords, xyxy2xywh, strip_optimizer, set_logging, increment_path
from utils.plots import plot_one_box
from utils.torch_utils import select_device, load_classifier, time_synchronized, TracedModel



def detect(save_img=False):
    source, weights, view_img, save_txt, imgsz, trace = opt.source, opt.weights, opt.view_img, opt.save_txt, opt.img_size, not opt.no_trace
    save_img = not opt.nosave and not source.endswith('.txt')  # save inference images
    webcam = source.isnumeric() or source.endswith('.txt') or source.lower().startswith(
        ('rtsp://', 'rtmp://', 'http://', 'https://'))

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

    # Initialize
    set_logging()
    device = select_device(opt.device)
    half = device.type != 'cpu'  # half precision only supported on CUDA

    # Load model
    model = attempt_load(weights, map_location=device)  # load FP32 model
    stride = int(model.stride.max())  # model stride
    imgsz = check_img_size(imgsz, s=stride)  # check img_size

    if trace:
        model = TracedModel(model, device, opt.img_size)

    if half:
        model.half()  # to FP16

    # Second-stage classifier
    classify = False
    if classify:
        modelc = load_classifier(name='resnet101', n=2)  # initialize
        modelc.load_state_dict(torch.load('weights/resnet101.pt', map_location=device)['model']).to(device).eval()

⊗ 0 ⚠ 0
```

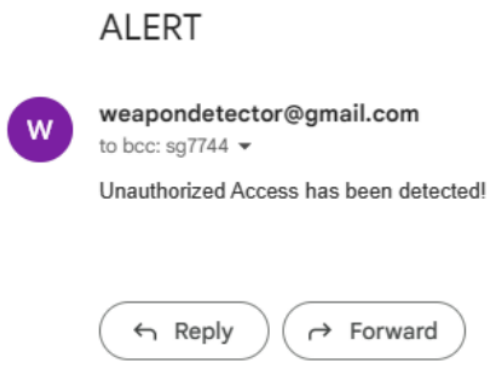

detect.py
D: > Major Proj > Weapon Proj > yolov7 > detect.py > ...
# Set Dataloader
vid_path, vid_writer = None, None
if webcam:
view_img = check_imshow()
cudnn.benchmark = True # set True to speed up constant image size inference
dataset = LoadStreams(source, img_size=imgsz, stride=stride)
else:
dataset = LoadImages(source, img_size=imgsz, stride=stride)
# Get names and colors
names = ["Weapon","Weapon","Weapon","Weapon","Weapon","Weapon","Weapon","Weapon","Weapon"]
colors = [[random.randint(0, 255) for _ in range(3)] for _ in names]
# Run inference
if device.type != 'cpu':
model(torch.zeros(1, 3, imgsz, imgsz).to(device).type_as(next(model.parameters()))) # run once
old_img_w = old_img_h = imgsz
old_img_b = 1
t0 = time.time()
for path, img, im0s, vid_cap in dataset:
img = torch.from_numpy(img).to(device)
img = img.half() if half else img.float() # uint8 to fp16/32
img /= 255.0 # 0 - 255 to 0.0 - 1.0
if img.ndimension() == 3:
img = img.unsqueeze(0)
# Warmup
if device.type != 'cpu' and (old_img_b != img.shape[0] or old_img_h != img.shape[2] or old_img_w != img.shape[3]):
old_img_b = img.shape[0]
old_img_h = img.shape[2]
old_img_w = img.shape[3]
for i in range(3):
model(img, augment=opt.augment)[0]
# Inference
t1 = time_synchronized()
with torch.no_grad(): # Calculating gradients would cause a GPU memory leak
pred = model(img, augment=opt.augment)[0]
t2 = time_synchronized()
# Apply NMS
pred = non_max_suppression(pred, opt.conf_thres, opt.iou_thres, classes=opt.classes, agnostic=opt.agnostic_nms)
t3 = time_synchronized()
import winsound
# Apply Classifier
if classify:
# Play the audio file
0 0


detect.py
D: > Major Proj > Weapon Proj > yolov7 > detect.py > ...
# Play the audio file
pred = apply_classifier(pred, modelc, img, im0s)
# Process detections
for i, det in enumerate(pred):  # detections per image
if webcam:  # batch_size >= 1
p, s, im0, frame = path[i], '%g: ' % i, im0s[i].copy(), dataset.count
else:
p, s, im0, frame = path, '', im0s, getattr(dataset, 'frame', 0)
p = Path(p)  # to Path
save_path = str(save_dir / p.name)  # img.jpg
txt_path = str(save_dir / 'labels' / p.stem) + ('' if dataset.mode == 'image' else f'_{frame}')  # img.txt
gn = torch.tensor(im0.shape)[[1, 0, 1, 0]]  # normalization gain whwh
if len(det):
# Rescale boxes from img_size to im0 size
det[:, :4] = scale_coords(img.shape[2:], det[:, :4], im0.shape).round()
# Print results
for c in det[:, -1].unique():
n = (det[:, -1] == c).sum()  # detections per class
s += f"{n} {names[int(c)]}{'s' * (n > 1)}, "  # add to string
# Write results
for *xyxy, conf, cls in reversed(det):
if save_txt:  # Write to file
xywh = (xyxy2xywh(torch.tensor(xyxy).view(1, 4)) / gn).view(-1).tolist()  # normalized xywh
line = (cls, *xywh, conf) if opt.save_conf else (cls, *xywh)  # label format
with open(txt_path + '.txt', 'a') as f:
f.write(('%g ' * len(line)).rstrip() % line + '\n')
if save_img or view_img:  # Add bbox to image
wave_obj = sa.WaveObject.from_wave_file("alarm.wav")
# Play the audio file
play_obj = wave_obj.play()
now = datetime.datetime.now()
pw.sendwhatmsg("+919566178893", "Unauthorized weapon usage has been detected!", now.hour, now.minute + 2)
label = f'{names[int(cls)]} {conf:.2f}'
plot_one_box(xyxy, im0, label=label, color=colors[int(cls)], line_thickness=1)
# Print time (inference + NMS)
print(f'{s}Done. ({(1E3 * (t2 - t1)):.1f}ms) Inference, ({(1E3 * (t3 - t2)):.1f}ms) NMS')
# Stream results
if view_img:
cv2.imshow(str(p), im0)
cv2.waitKey(1)  # 1 millisecond
# Save results (image with detections)
if save_img:
⊗ 0 ⚠ 0

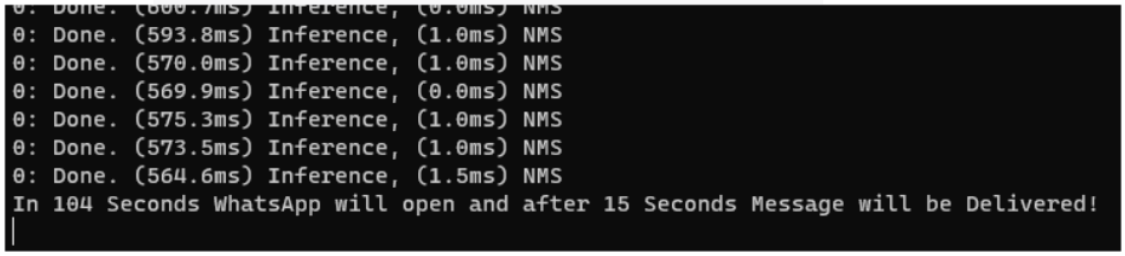

detect.py
D: > Major Proj > Weapon Proj > yolov7 > detect.py > ...
label = f'{names[int(cls)]} {conf:.2f}'
plot_one_box(xyxy, im0, label=label, color=colors[int(cls)], line_thickness=1)
# Print time (inference + NMS)
print(f'{s}Done. ({(1E3 * (t2 - t1)):.1f}ms) Inference, ({(1E3 * (t3 - t2)):.1f}ms) NMS')
# Stream results
if view_img:
cv2.imshow(str(p), im0)
cv2.waitKey(1) # 1 millisecond
# Save results (image with detections)
if save_img:
if dataset.mode == 'image':
cv2.imwrite(save_path, im0)
print(f" The image with the result is saved in: {save_path}")
else: # 'video' or 'stream'
if vid_path != save_path: # new video
vid_path = save_path
if isinstance(vid_writer, cv2.VideoWriter):
vid_writer.release() # release previous video writer
if vid_cap: # video
fps = vid_cap.get(cv2.CAP_PROP_FPS)
w = int(vid_cap.get(cv2.CAP_PROP_FRAME_WIDTH))
h = int(vid_cap.get(cv2.CAP_PROP_FRAME_HEIGHT))
else: # stream
fps, w, h = 30, im0.shape[1], im0.shape[0]
save_path += '.mp4'
vid_writer = cv2.VideoWriter(save_path, cv2.VideoWriter_fourcc(*'mp4v'), fps, (w, h))
vid_writer.write(im0)
if save_txt or save_img:
s = f"\n{len(list(save_dir.glob('labels/*.txt')))} labels saved to {save_dir / 'labels'}" if save_txt else ''
#print(f"Results saved to {save_dir}{s}")
print(f'Done. ({time.time() - t0:.3f}s)')
if __name__ == '__main__':
parser = argparse.ArgumentParser()
parser.add_argument('--weights', nargs='+', type=str, default='yolov7.pt', help='model.pt path(s)')
parser.add_argument('--source', type=str, default='inference/images', help='source') # file/folder, 0 for webcam
parser.add_argument('--img-size', type=int, default=640, help='inference size (pixels)')
parser.add_argument('--conf-thres', type=float, default=0.25, help='object confidence threshold')
parser.add_argument('--iou-thres', type=float, default=0.45, help='IOU threshold for NMS')
parser.add_argument('--device', default='', help='cuda device, i.e. 0 or 0,1,2,3 or cpu')
parser.add_argument('--view-img', action='store_true', help='display results')
parser.add_argument('--save-txt', action='store_true', help='save results to *.txt')
parser.add_argument('--save-conf', action='store_true', help='save confidences in --save-txt labels')
parser.add_argument('--nosave', action='store_true', help='do not save images/videos')
parser.add_argument('--classes', nargs='+', type=int, help='filter by class: --class 0, or --class 0 2 3')
parser.add_argument('--agnostic-nms', action='store_true', help='class-agnostic NMS')
parser.add_argument('--augment', action='store_true', help='augmented inference')
⊗ 0 ⚠ 0

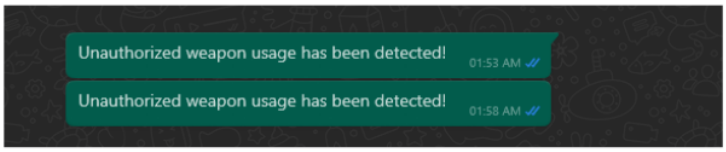

Explorer (Ctrl+Shift+E)
D: > Major Proj > Weapon Proj > yolov7 > detect.py > ...
if vid_cap:  # video
fps = vid_cap.get(cv2.CAP_PROP_FPS)
w = int(vid_cap.get(cv2.CAP_PROP_FRAME_WIDTH))
h = int(vid_cap.get(cv2.CAP_PROP_FRAME_HEIGHT))
else:  # stream
fps, w, h = 30, im0.shape[1], im0.shape[0]
save_path += '.mp4'
vid_writer = cv2.VideoWriter(save_path, cv2.VideoWriter_fourcc(*'mp4v'), fps, (w, h))
vid_writer.write(im0)
if save_txt or save_img:
s = f"\n{len(list(save_dir.glob('labels/*.txt')))} labels saved to {save_dir / 'labels'}" if save_txt else ''
#print(f"Results saved to {save_dir}{s}")
print(f'Done. ({time.time() - t0:.3f}s)')
if __name__ == '__main__':
parser = argparse.ArgumentParser()
parser.add_argument('--weights', nargs='+', type=str, default='yolov7.pt', help='model.pt path(s)')
parser.add_argument('--source', type=str, default='inference/images', help='source')  # file/folder, 0 for webcam
parser.add_argument('--img-size', type=int, default=640, help='inference size (pixels)')
parser.add_argument('--conf-thres', type=float, default=0.25, help='object confidence threshold')
parser.add_argument('--iou-thres', type=float, default=0.45, help='IOU threshold for NMS')
parser.add_argument('--device', default='', help='cuda device, i.e. 0 or 0,1,2,3 or cpu')
parser.add_argument('--view-img', action='store_true', help='display results')
parser.add_argument('--save-txt', action='store_true', help='save results to *.txt')
parser.add_argument('--save-conf', action='store_true', help='save confidences in --save-txt labels')
parser.add_argument('--nosave', action='store_true', help='do not save images/videos')
parser.add_argument('--classes', nargs='+', type=int, help='filter by class: --class 0, or --class 0 2 3')
parser.add_argument('--agnostic-nms', action='store_true', help='class-agnostic NMS')
parser.add_argument('--augment', action='store_true', help='augmented inference')
parser.add_argument('--update', action='store_true', help='update all models')
parser.add_argument('--project', default='runs/detect', help='save results to project/name')
parser.add_argument('--name', default='exp', help='save results to project/name')
parser.add_argument('--exist-ok', action='store_true', help='existing project/name ok, do not increment')
parser.add_argument('--no-trace', action='store_true', help='don`t trace model')
opt = parser.parse_args()
print(opt)
#check_requirements(exclude=('pycocotools', 'thop'))
with torch.no_grad():
if opt.update:  # update all models (to fix SourceChangeWarning)
for opt.weights in ['yolov7.pt']:
detect()
strip_optimizer(opt.weights)
else:
detect()
⊗ 0 ⚠ 0


yolov7
Knife 0.51
Size
73,117 KB
314 KB
C:\Windows\System32\cmd.e
0: 1 Knife, Done. (536.1ms) Inference, (1.0ms) NMS
ok
0: 1 Knife, Done. (516.8ms) Inference, (0.0ms) NMS
0: Done. (493.2ms) Inference, (0.0ms) NMS
0: Done. (549.1ms) Inference, (0.0ms) NMS
ok
0: 1 Knife, Done. (530.2ms) Inference, (0.0ms) NMS
0: Done. (520.2ms) Inference, (0.0ms) NMS
0: Done. (511.2ms) Inference, (1.0ms) NMS
0: Done. (521.2ms) Inference, (0.0ms) NMS
0: Done. (512.2ms) Inference, (1.0ms) NMS
0: Done. (532.2ms) Inference, (0.0ms) NMS
ok
0: 1 Knife, Done. (518.2ms) Inference, (1.0ms) NMS
ok
0: 1 Knife, Done. (529.2ms) Inference, (0.0ms) NMS
0: Done. (522.3ms) Inference, (0.0ms) NMS
0: Done. (514.2ms) Inference, (1.0ms) NMS
0: Done. (478.3ms) Inference, (0.0ms) NMS
0: Done. (501.2ms) Inference, (0.0ms) NMS
0: Done. (449.9ms) Inference, (0.0ms) NMS
ok
0: 1 Knife, Done. (498.2ms) Inference, (0.0ms) NMS
ok
0: 1 Knife, Done. (512.3ms) Inference, (1.0ms) NMS
ok
0: 1 Knife, Done. (498.3ms) Inference, (1.0ms) NMS
ok
0: 1 Knife, Done. (489.3ms) Inference, (1.0ms) NMS
face_saver.py 14-04-2023 01:25 AM
hubconf.py 12-04-2023 06:47 AM
LICENSE.md 12-04-2023 06:47 AM
PyWhatKit_DB.txt 19-04-2023 12:56 PM
README.md 12-04-2023 06:47 AM
requirements.txt 12-04-2023 06:47 AM
Table 1.docx 20-04-2023 01:16 AM
test.py 12-04-2023 06:47 AM
testggg.txt 19-04-2023 02:04 PM
traced_model.pt 28-04-2023 02:26 PM
train.py 12-04-2023 06:47 AM
train_aux.py 12-04-2023 06:47 AM
Python Source File 37 KB
user_data.csv 20-04-2023 03:51 PM
Microsoft Excel C... 1 KB


ALERT
W
weapondetector@gmail.com
to bcc: sg7744
Unauthorized Access has been detected!
Reply
Forward

**EMail Alert**


0: Done. (593.8ms) Inference, (1.0ms) NMS
0: Done. (570.0ms) Inference, (1.0ms) NMS
0: Done. (569.9ms) Inference, (0.0ms) NMS
0: Done. (575.3ms) Inference, (1.0ms) NMS
0: Done. (573.5ms) Inference, (1.0ms) NMS
0: Done. (564.6ms) Inference, (1.5ms) NMS
In 104 Seconds WhatsApp will open and after 15 Seconds Message will be Delivered!




**WhatsApp Alert**

# CHAPTER 6

# RESULTS AND OBSERVATIONS

## 6.1 RESULTS

The proposed model has been evaluated using the YOLO v7 model for weapon detection system that identifies the following weapon classes: Handgun, Sword, SMG, Sniper, Automatic Rifle, Bazooka, Grenade Launcher, Knife, and Shotgun and the Haar Cascade algorithm for face recognition. The evaluation has shown promising results, with the YOLO v7 model achieving an accuracy of 88.7%, average precision of 94.3% and a recall rate of 95% for weapon detection. Additionally, the Haar Cascade algorithm achieved an accuracy rate of 96% for face recognition. The accuracy metric measures the percentage of correctly recognized faces among all tested faces in the dataset. Finally, the notification and alert system has been successfully integrated into the proposed system, sending alarm sound alerts along with the message to the WhatsApp number and designated email address. Overall, the evaluation results demonstrate that the proposed intelligent security system is effective in detecting weapons and recognizing faces in real-time videos, and has the potential to enhance security measures in various applications such as airports, public places, and critical infrastructures.

| Sl. No | Types of Weapons | Number of Labels | Precision | Recall | Accuracy (mAP@0.5) | F1 Score |
|---|---|---|---|---|---|---|
| 1 | All | 936 | 0.943 | 0.95 | 0.887 | 0.946 |
| 2 | Knife | 139 | 0.936 | 0.773 | 0.872 | 0.847 |
| 3 | Handgun | 121 | 0.870 | 0.579 | 0.702 | 0.696 |
| 4 | Bazooka | 65 | 0.705 | 0.625 | 0.651 | 0.663 |
| 5 | Sniper | 85 | 0.860 | 0.613 | 0.720 | 0.716 |
| 6 | Sword | 108 | 0.901 | 0.826 | 0.881 | 0.862 |
| 7 | Grenade launcher | 80 | 0.857 | 0.750 | 0.827 | 0.800 |
| 8 | Shotgun | 96 | 0.898 | 0.880 | 0.868 | 0.846 |
| 9 | SMG | 117 | 0.720 | 0.857 | 0.879 | 0.783 |

| | | | | | | |
|---|---|---|---|---|---|---|
| **10** | Automatic Rifle | 125 | 0.506 | 0.700 | 0.684 | 0.588 |

**Table 6.1.1. This table contains Precision, Recall, Accuracy and F1 Scores for 9 classes of Weapons.**

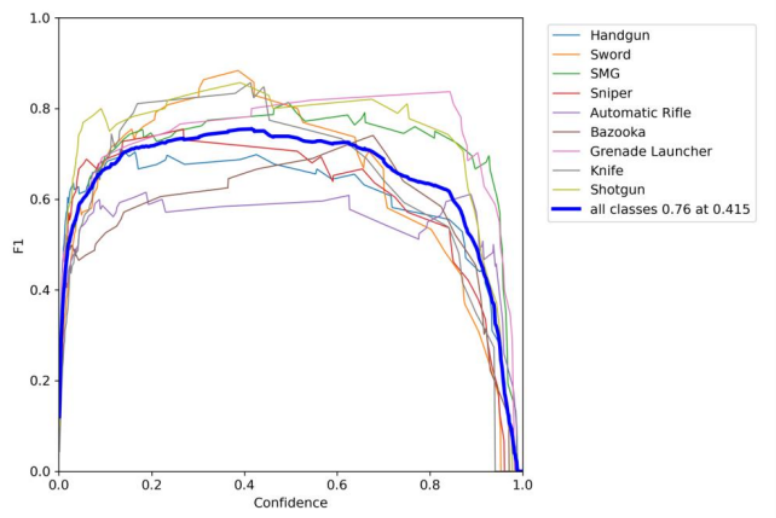
| Sl.No | Epochs | Accuracy(mAP@0.5) | Precision | Recall |
|---|---|---|---|---|
| **1** | 50 | 0.207 | 0.313 | 0.298 |
| **2** | 100 | 0.543 | 0.611 | 0.537 |
| **3** | 150 | 0.683 | 0.766 | 0.612 |
| **4** | 200 | 0.734 | 0.812 | 0.646 |
| **5** | 250 | 0.727 | 0.792 | 0.682 |
| **6** | 300 | 0.757 | 0.776 | 0.720 |
| **7** | 350 | 0.763 | 0.856 | 0.667 |
| **8** | 399 | 0.759 | 0.769 | 0.723 |

**Table 6.1.2. This Table contains the Precision, Recall and Accuracy Scores for each Epoch(s).**

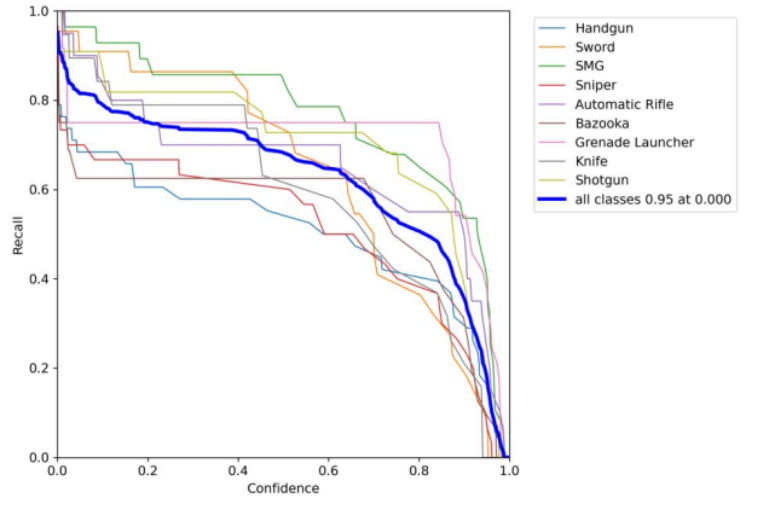
| Sl.No | Weapon revealed on camera(in %) | Time Taken to Identify Weapon (in ms) |
|---|---|---|
| **1** | 33% | 624.1 ms |
| **2** | 66% | 617.6 ms |
| **3** | 100% | 608.4 ms |

**Table 6.1.3. This Table contains the time taken to identify a weapon based on the % of the weapon shown.**

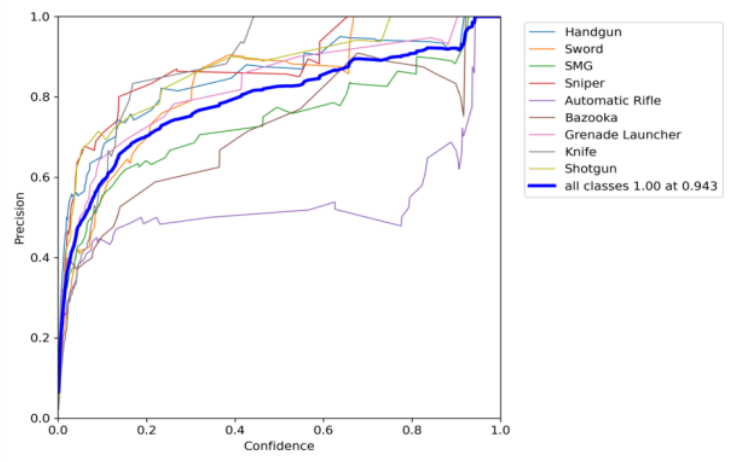
## 6.2 GRAPHS:




### 6.2.1. F1 Curve




### 6.2.2. Recall Curve




### 6.2.3. Precision Curve

## 6.2 OBSERVATION

The combination of weapon detection and facial recognition technologies has the potential to enhance public safety and prevent potential threats in public spaces. However, the use of facial recognition raises various ethical and legal concerns such as privacy and accuracy issues. Therefore, appropriate measures need to be taken to ensure the privacy and security of individuals' biometric data and to improve the accuracy and reliability of facial recognition algorithms. In addition to ethical and legal concerns, there are also technical challenges that need to be addressed when combining weapon detection and facial recognition technologies. One of the primary challenges is the integration of different algorithms and techniques to achieve real-time and accurate detection and identification of potential threats.Moreover, the accuracy of the proposed system may be affected by various factors such as lighting, camera angles, and occlusions. Further research is needed to develop more robust algorithms that can overcome these challenges and improve the accuracy and efficiency of the system. Overall, the proposed system for real-time weapon detection using deep learning-based algorithms represents a significant advancement in public safety and security. With ongoing advances in machine learning and computer vision, we can expect to see more sophisticated and accurate weapon detection and facial recognition systems in the future.

# CHAPTER 7

# CONCLUSION

The proposed intelligent security system that combines YOLO v7 and Haar Cascade for weapon detection and face recognition, respectively, offers a reliable and effective solution to ensure safety and security in public places. The system is designed to detect weapons and recognize faces in real-time, and send WhatsApp and email notifications to designated users when a weapon is detected, along with an alarm alert.

The YOLO v7 algorithm was trained on a custom dataset consisting of 714 images of various weapons and achieved high accuracy and recall rate in detecting weapons. The algorithm uses a deep neural network to detect objects in images and videos. The algorithm can detect multiple objects in an image or video and can identify the type of object with high accuracy. The Haar Cascade algorithm was also able to achieve high accuracy in face recognition. The algorithm uses a set of features to detect objects in images and videos. The algorithm can detect multiple objects in an image or video and can identify the type of object with high accuracy.

The notification and alert system is an added feature that ensures that designated users are notified in real-time when a weapon is detected. This approach helps to ensure that the authorities can respond quickly to potential threats and take appropriate action to prevent any harm to the public. The system can be integrated with other security systems, such as CCTV cameras and metal detectors, to provide additional security measures.

This system has the ability to adapt to various settings and can be implemented in locations such as airports, train stations, schools, shopping malls, and public gatherings. Its purpose is to enhance security measures and provide an extra layer of safety. The proposed system has the potential to enhance public safety and prevent potential threats in public spaces. The combination of weapon detection and facial recognition technologies has the potential to enhance public safety and prevent potential threats in public spaces.

## Future Work

Future work for this project includes several areas of improvement that can enhance the effectiveness of the proposed intelligent security system. One of the areas of improvement is to improve the accuracy of the face recognition algorithm. While the Haar Cascade algorithm used in this project achieved high accuracy, more advanced methods such as deep learning techniques can be employed to further improve the accuracy of the face recognition algorithm. Deep learning techniques can help to identify more complex facial features and improve the recognition of faces in different lighting conditions and angles.

Another area of improvement for the proposed system is to increase the detection rate of the system for detecting concealed weapons. While the YOLO v7 algorithm used in this project achieved high accuracy in detecting weapons, there is still room for improvement in detecting concealed weapons. Advanced techniques such as thermal imaging and X-ray scanning can be employed to detect concealed weapons and improve the overall effectiveness of the system.

In addition, the proposed system can be integrated with a centralized monitoring system to provide a more comprehensive security solution. A centralized monitoring system can help to monitor multiple locations simultaneously and provide real-time alerts and notifications to the authorities. This approach can help to improve the response time of the authorities and enhance the overall effectiveness of the security system.

# for plag check.docx

ORIGINALITY REPORT

| 7% | 4% | 4% | 3% |
|---|---|---|---|
| SIMILARITY INDEX | INTERNET SOURCES | PUBLICATIONS | STUDENT PAPERS |

PRIMARY SOURCES

| | | |
|---|---|---|
| 1 | Lecture Notes in Computer Science, 2015.<br>Publication | <1% |
| 2 | Submitted to CSU, San Jose State University<br>Student Paper | <1% |
| 3 | Submitted to Nanyang Technological University<br>Student Paper | <1% |
| 4 | Lecture Notes in Computer Science, 2014.<br>Publication | <1% |
| 5 | www.researchgate.net<br>Internet Source | <1% |
| 6 | worldfoodscience.com<br>Internet Source | <1% |
| 7 | www.mdpi.com<br>Internet Source | <1% |
| 8 | "Computer Vision – ECCV 2018", Springer Science and Business Media LLC, 2018<br>Publication | <1% |

| | | |
|---|---|---|
| 9 | Mir Sayed Shah Danish, Zahra Nazari, Tomonobu Senjyu. "AI-coherent data-driven forecasting model for a combined cycle power plant", Energy Conversion and Management, 2023<br>Publication | <1 % |
| 10 | ecampus.pelitabangsa.ac.id<br>Internet Source | <1 % |
| 11 | pdfcoffee.com<br>Internet Source | <1 % |
| 12 | www.telementoring.org<br>Internet Source | <1 % |
| 13 | Tanmoy Sarkar Pias, Yiqi Su, Xuxin Tang, Haohui Wang, Danfeng (Daphne) Yao. "Undersampling for Fairness: Achieving More Equitable Predictions in Diabetes and Prediabetes", Cold Spring Harbor Laboratory, 2023<br>Publication | <1 % |
| 14 | core.ac.uk<br>Internet Source | <1 % |
| 15 | Pawan K. Jha, Utham K. Valekunja, Akhilesh B. Reddy. "SlumberNet: Deep learning classification of sleep stages using residual neural networks", Cold Spring Harbor Laboratory, 2023<br>Publication | <1 % |

| | | |
|---|---|---|
| 16 | "Advances in Multimedia Information Processing – PCM 2018", Springer Science and Business Media LLC, 2018<br>Publication | <1 % |
| 17 | Hidemasa Takao, Shiori Amemiya, Shimpei Kato, Hiroshi Yamashita, Naoya Sakamoto, Osamu Abe. "Deep-learning 2.5-dimensional single-shot detector improves the performance of automated detection of brain metastases on contrast-enhanced CT", Neuroradiology, 2022<br>Publication | <1 % |
| 18 | www.atc.ac.th<br>Internet Source | <1 % |
| 19 | Submitted to Coventry University<br>Student Paper | <1 % |
| 20 | researchportal.port.ac.uk<br>Internet Source | <1 % |
| 21 | "Kids Cybersecurity Using Computational Intelligence Techniques", Springer Science and Business Media LLC, 2023<br>Publication | <1 % |
| 22 | ij-healthgeographics.biomedcentral.com<br>Internet Source | <1 % |
| 23 | Geoffrey E. Hinton, Simon Osindero, Yee-Whye Teh. "A Fast Learning Algorithm for | <1 % |

| | | |
|---|---|---|
| | Deep Belief Nets", Neural Computation, 2006<br>Publication | |
| 24 | Submitted to October University for Modern Sciences and Arts (MSA)<br>Student Paper | <1 % |
| 25 | Submitted to University of Salford<br>Student Paper | <1 % |
| 26 | Submitted to University of Sunderland<br>Student Paper | <1 % |
| 27 | Submitted to Indian Institute of Space Science and Technology<br>Student Paper | <1 % |
| 28 | journal.frontiersin.org<br>Internet Source | <1 % |
| 29 | link.springer.com<br>Internet Source | <1 % |
| 30 | ndl.ethernet.edu.et<br>Internet Source | <1 % |
| 31 | www.ml.cmu.edu<br>Internet Source | <1 % |
| 32 | Submitted to Liverpool John Moores University<br>Student Paper | <1 % |
| 33 | mededu.jmir.org<br>Internet Source | <1 % |

| | | |
|---|---|---|
| 34 | tableaumagic.com<br>Internet Source | <1% |

| | | | |
|---|---|---|---|
| Exclude quotes | On | Exclude matches | < 10 words |
| Exclude bibliography | On | | |