An Intelligent Weapon Detection System for Surveillance Cameras

Graduation Project II (Final Report)

By:

Mohamed Nasser Hashem, ID: **372029063**

Mustafa Ahmed Abdulsami, ID: **372029353**

Husen Muhammad Kalaepeh, ID: **352072163**

Supervised By:

Dr. Emad Nabil

الجامعة الإسلامية بالمدينة المنورة

ISLAMIC UNIVERSITY OF MADINAH

Faculty of Computer and Information Systems

Islamic University of Madinah

June - 2022

DEDICATION

This work is completely dedicated to my grandfather (whom my eyes have not seen), respectful Parents, beloved wife, and darling son. without whose constant support this was not possible.

Mohamed Nasser Hashem

ACKNOWLEDGMENT

The success and results of this project have required a tremendous amount of guidance, and fortunately we have achieved that throughout our project.

All that we did was, firstly, by the grace of Allah Almighty for His bounty to complete the project, without his blessings, we would not be able to do anything.

Then, we like to express our special thanks and gratitude to our supervisor, Dr. Emad Nabil, who helped us and directed us throughout the period of work on this project then.

In the end, the support and help from people around us we think it is necessary to thank them, we thank our professors and colleagues who have been supportive of us throughout the project.

DECLARATION

We hereby certify that this material, which we now submit for assessment on the program of study leading to the award of Bachelor of *(Computer Science*) is entirely my own work, that We have exercised reasonable care to ensure that the work is original, and does not to the best of my knowledge breach any law of copyright, and has not been taken from the work of others save and to the extent that such work has been cited and acknowledged within the text of my work.

Stud ID.: 372029063, **Stud Name:** Mohamed Nasser Hashem, **Signed: __**Mohamed**__**

Stud ID.: 372029353, **Stud Name:** Mustafa Ahmed Abdulsami, **Signed: __**Mustafa**__**

Stud ID.: 352072163, **Stud Name:** Husen Muhammad Kalaepeh, **Signed: __** Husen**__**

Date: 7/6/2022

ABSTRACT

Preserving humans' life and valuables is the top requirement for any society. Due to the existing of so many criminals or menacing tactics, it is almost impossible to discern them by visual ability, as they easily hide inside large crowds. For this, the developers and the police are working to secure it both technically and realistically. Because security alone is unable to deal with such matters. Due to modern technology, the current era, to the researchers' background in computer science, we can work to create a detection system for surveillance.

In this project, we will develop an intelligent system that is able to detect both a human and some weapons that are used by criminals. The detected weapons are guns, rifles, and knives. After the detection of one of the weapons, an alarm will be raised. system can be used in public places. using modern machine learning technologies for object detection to achieve that goal.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF CODES

C h a p t e r   O n e

1 INTRODUCTION

In the age of technology, everything in this life has become connected to the world of technology. Coins, diamonds, gold, jewelry, antiques, and other valuables. Humans alone are no longer able to protect these precious things. This is on the physical side.

On the human side, the matter has become more different. The presence of humans has become more intense in public places, such as schools, universities, and parks. With so many, criminals or threatening tactics are virtually impossible to discern.

Security threats have become common and a reality in this time, threats may be from forced robbery to mass hostage-taking. For this, the developers and the police are working to secure it both technically and realistically. It is our duty as developers now to work seriously to obtain security monitoring systems in cooperation with security agencies.

In this project we will try to use the latest open-source object detection algorithms and datasets. To create an intelligent Weapon Detection system for surveillance. to start in our work, we started by doing a detailed and meticulous research into the two main sections of this project: the algorithms that will be used as a model for the system, and open-source datasets related to our project (knife, pistol, rifle).

For algorithms, the literature review began examining each algorithm by: research paper, and source-code. Which led to the arrangement of algorithms according to strength and speed, which led us to define the YOLO algorithm. As for data sets, all open-source databases were compiled and filtered, resulting in more than 35,000 images.

1.1 AIM

Create a system connected to a camera that can identify the criminal who enters public places, by identifying if he is carrying a gun, trifle, or a knife. and to give a warning to all those present in this facility and warn them.

1.2 PROPLEM STATEMENT

Criminals that enter public places is spreading all over the world as well. It spreads in remote places or branches due to the lack of sufficient security. The thief always carries a gun, a knife, and an ordinary person cannot stop him or warn others about him. He threatens the victim and is taken with hatred and leaves behind many economic problems. No one can defend himself in front of that thief because of the weapon he carries, and in many cases the police cannot reach that thief. Again, theft is repeated a lot without any deterrent to stop them, arrest them, or even flee from them.

1.3 OBJECTIVES:

The main objective of Weapon Detection System is recognized and find at least one viable focus from still picture or video information. It thoroughly incorporates an assortment of It comprehensively includes a variety of important techniques.[1] Following are the primary objectives:

1. Create model(s) for threat/Weapon Detection
2. Train the model(s) using relevant data sets
3. Measure the performance of the model(s).
4. Enhance the system until it reaches an accepted accuracy rate.

1.4 SCOPE:

- Detecting persons(s) holds a gun, trifle, or knife from a camera video stream.
- The system will be in the form of web/desktop application
- The system will give an alarm in case of detection a threat with the mentioned properties.

1.5 METHODOLOGY:

A methodology is '"a system that sets guidelines for solving a problem, with components including phases, tasks, methods, methods, and tools." depending on the requirements and the project type.

In this system we have looked carefully for the requirements, and we have decided that the best approach to develop the system is using the Waterfall Methodology. We believe that using this approach can help us reach our goals and deliver the best quality.

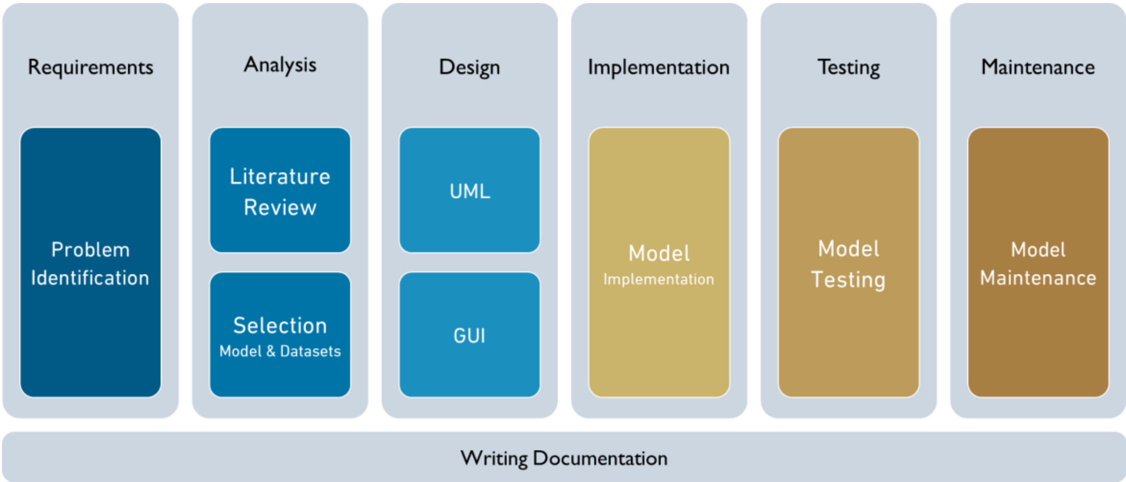


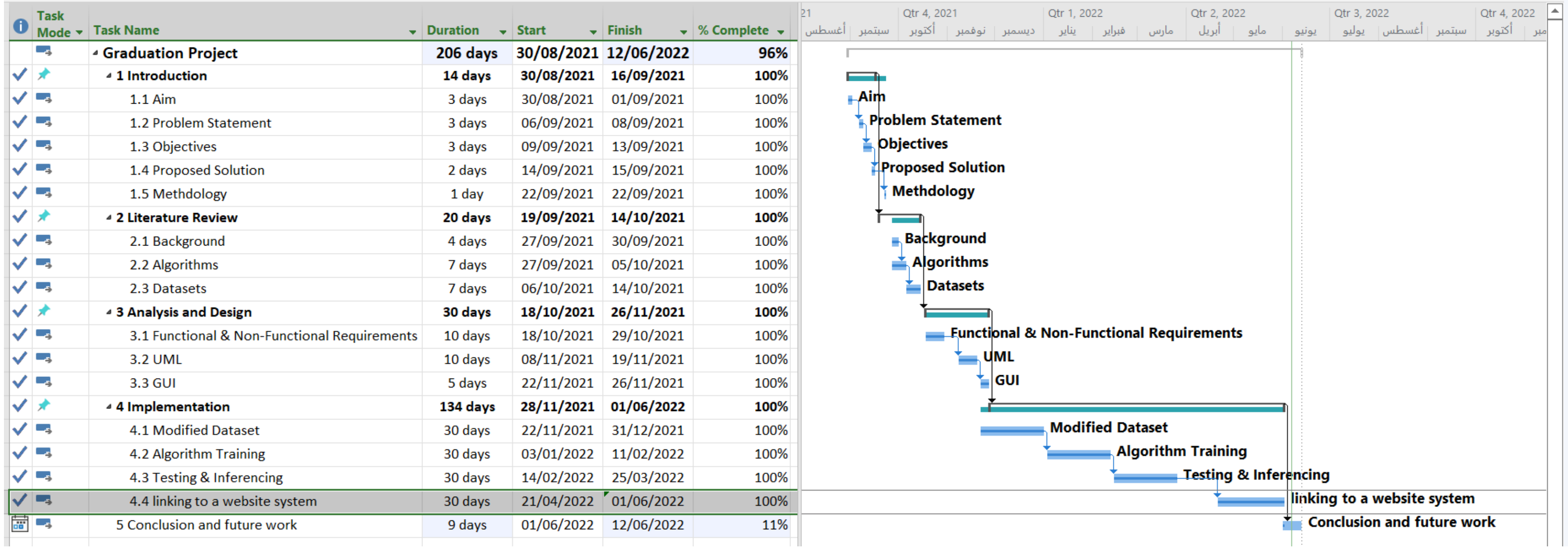
Figure 1-1: Project Methodology Model (Waterfall)

In next lines, there is a brief about the Waterfall Methodology.

1.5.1 Waterfall:

The waterfall Methodology is "a design process model that used in software development processes commonly." The workflow progress is in fixed form of pieces that start from top to down through these stages: The process includes the following steps: planning, analysis, design, construction, testing, production, implementation, and maintenance.

1.6 TIMELINE:



Task Mode	Task Name	Duration	Start	Finish	% Complete
	Graduation Project	**206 days**	**30/08/2021**	**12/06/2022**	**96%**
	1 Introduction	**14 days**	**30/08/2021**	**16/09/2021**	**100%**
	1.1 Aim	3 days	30/08/2021	01/09/2021	100%
	1.2 Problem Statement	3 days	06/09/2021	08/09/2021	100%
	1.3 Objectives	3 days	09/09/2021	13/09/2021	100%
	1.4 Proposed Solution	2 days	14/09/2021	15/09/2021	100%
	1.5 Methdology	1 day	22/09/2021	22/09/2021	100%
	2 Literature Review	**20 days**	**19/09/2021**	**14/10/2021**	**100%**
	2.1 Background	4 days	27/09/2021	30/09/2021	100%
	2.2 Algorithms	7 days	27/09/2021	05/10/2021	100%
	2.3 Datasets	7 days	06/10/2021	14/10/2021	100%
	3 Analysis and Design	**30 days**	**18/10/2021**	**26/11/2021**	**100%**
	3.1 Functional & Non-Functional Requirements	10 days	18/10/2021	29/10/2021	100%
	3.2 UML	10 days	08/11/2021	19/11/2021	100%
	3.3 GUI	5 days	22/11/2021	26/11/2021	100%
	4 Implementation	**134 days**	**28/11/2021**	**01/06/2022**	**100%**
	4.1 Modified Dataset	30 days	22/11/2021	31/12/2021	100%
	4.2 Algorithm Training	30 days	03/01/2022	11/02/2022	100%
	4.3 Testing & Inferencing	30 days	14/02/2022	25/03/2022	100%
	4.4 linking to a website system	30 days	21/04/2022	01/06/2022	100%
	5 Conclusion and future work	9 days	01/06/2022	12/06/2022	11%

Figure 1-2: Giant Chart for Project

Chapter Two

2 LITERATURE REVIEW

Object detection is a computer vision task that detects instances of visual objects of specific classes (such as persons, animals, cars, or buildings) in digital pictures like photos or video frames. The purpose of object detection is to create computational models that give computer vision applications the most basic information they need, in our case, we will study it and we compare a set of algorithms trying to produce the best one or the one that works well on our idea

2.1 BACKGROUND

Object recognition is a wide term that refers to a set of related computer vision tasks that include identifying objects in digital pictures. Image classification is defined as "predicting the class of one item in a picture." Object localization is the process of determining the position of one or more things in a photograph and drawing a bounding box around their extent. Object detection combines these two tasks by identifying and categorizing one or more objects in a picture. Therefore, three distinct computer vision tasks may be identified:

2.1.1 Image Classification

Predict the type or class of an object based on a photograph.[2] As input, a single-object image, such as a photograph, is utilized. As a consequence, a class label is created (e.g., one or more integers that are mapped to class labels).[3]

2.1.2 Object Localization

Determine the presence of items in a photograph and use a bounding box to pinpoint their location. As input, an image with one or more things, such as a photograph, is utilized. As an output, one or more bounding boxes (e.g., defined by a point, width, and height).

2.1.3 Object Detection

Using a bounding box, determine the presence of things in an image and the types or classes of the objects discovered.[3] As input, an image with one or more things, such as a photograph, is utilized. One or more bounding boxes (e.g., specified by a point, width, and height) are produced, together with a class label for each bounding box.

To understand more about Object detection what and what are the most famous types and how they are used Object detection is a computer vision task that detects instances of visual objects of specific classes (such as persons, animals, cars, or buildings) in digital pictures like photos or video frames. The purpose of object detection is to create computational models that give computer vision applications the most basic information they need, in our case, we will study it and we compare a set of algorithms trying to produce the best one or the one that works well on our idea.

2.1.4 Neural networks

In the domains of AI, machine learning, and deep learning, these models replicate human brain behavior, allowing computer systems to spot patterns and solve common issues.

Artificial neural networks (ANN) and simulated neural networks (SNN) are a subset of machine learning that are at the heart of deep learning methods. Their name and structure derived from the human brain, and they resemble the way biological neurons communicate with one another.

A node layer contains an input layer, one or more hidden layers, and an output layer in artificial neural networks (ANN). Each node, or artificial neuron, which connected to the others and has a weight and threshold linked with it. If a node's output exceeds a certain threshold value, the node activated, then data sent to the next tier of the network. Otherwise, no data is sent on to the network's next tier [4]–[6].

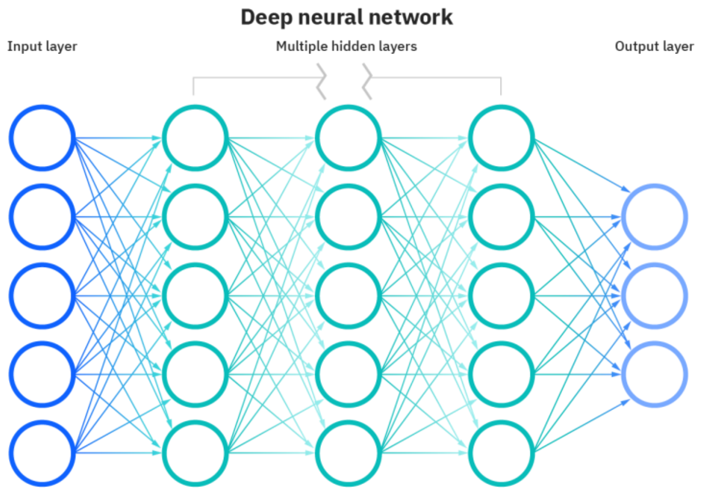


Figure 2-1: Layers of Deep Neural Network

Training data used by neural networks to learn and increase their accuracy over time. However, once these learning algorithms have fine-tuned for accuracy, they become formidable tools in computer science and artificial intelligence, allowing us to quickly classify and cluster data. When compared to manual identification by human experts, tasks in speech recognition or image recognition can take minutes rather than hours. Google's search algorithm is one of the most well-known neural networks

Types of Neural Networks

Distinct types of neural networks exist, each of which is employed for a different purpose. While this is not an exhaustive list, the following are some of the most popular types of neural networks that you will come across for common applications:

Frank Rosenblatt invented the perceptron in 1958, and it is the oldest neural network. It is the simplest type of a neural network, with only one neuron:

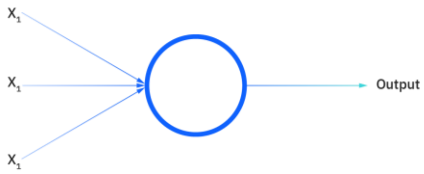


Figure 2-2: Simplest Form of a Neural Network

This part of the project has mostly focused on feedforward neural networks, often known as multi-layer perceptron's (MLPS). An input layer, a concealed layer or layers, and an output layer make up these layers. While these neural networks are also known as MLPs, it is important to remember that they are made up of sigmoid neurons rather than perceptron's because most real-world problems are not linear. These models provide the cornerstone for computer vision, natural language processing, and other neural networks, and they are typically fed data to train them.

Like feedforward networks, convolutional neural networks (CNNs) are used for image recognition, pattern identification, and/or computer vision. These networks use linear algebra principles, notably matrix multiplication, to find patterns in images.

The feedback loops distinguish recurrent neural networks (RNNs). These learning algorithms are used to create predictions about future outcomes using time-series data, such as stock market projections or sales forecasting [4]–[6].

In the upcoming pictures, best types of object detection help in a clear picture output, the difference between them, and their speed of identification and exploration.

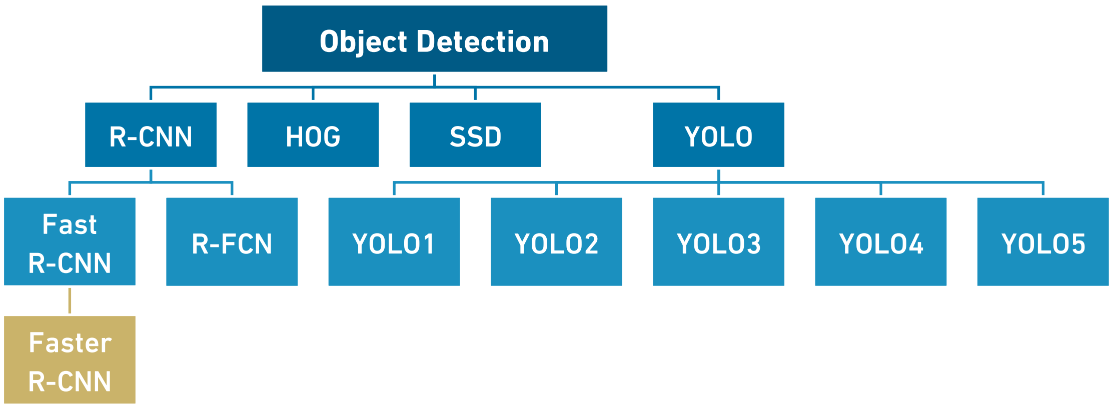


Figure 2-3: Object Detection Algorithms series

2.2 ALGORITHEMS

Table 2-1: Summary of Algorithms

Algorithms	Definition	Creation
Histogram of Oriented Gradients (HOG)	Canny Edge Detector and SIFT are examples of feature descriptors (Scale Invariant and Feature Transform). counts how many times a gradient orientation appears in a certain region of a picture.	1986
Single Shot Detector (SSD)	Detect several objects inside a picture using a single shot.	2016
Region-based Fully Convolutional Network (R-FCN)	Reduces the amount of work necessary for each ROI, which speeds up the process.	2016
Fast R-CNN	The CNN is fed the input picture, which creates a convolutional feature map. As a starting point, use the convolutional feature map.	2015
Faster R-CNN	deep convolutional network for object detection that appears to the user as a single, end-to-end, unified network.	2015
Region-based Convolutional Neural Networks (R-CNN)	Combination of region recommendation and Convolutional Neural Networks (CNNs)	2014
You Only Look Once (YOLO)	A single neural network is used in an object detection method. In contrast to several other object detection algorithms that do a bit-by-bit scan of the picture.	2016

In the next lines, we are going to have a brief for each algorithm.

2.2.1 Histogram of Oriented Gradients (HOG)

In 2005, Navneet Dalal and Bill Triggs presented highlights from Histogram of Situated Gradients (HOG). The Histogram of Arranged Inclinations (Hoard) is a component descriptor used in image processing, mostly for object detection. A representation of an image or a picture repair that enhances the picture by isolating useful data from it is called an element descriptor.[7], [8]

The histogram of organized inclinations descriptor is based on the idea that the dispersion of force angles or edge bearings might represent the look and form of nearby objects inside an image. Because the extent of angles is limited, the x and y subsidiaries of an image (Inclinations) are useful Because of the abrupt shift in power near edges and corners, is enormous, and we understand that edges and corners pack in far more info about object form than level locations. As a result, the histograms of inclination headings are used as items in this descriptor.

Object detection workflow with HOG

We will go on to how we compute the histograms and how the element vectors obtained from the Hoard descriptor are used by the classifier such as SVM to identify the concerned article now that we understand the fundamental idea of Histogram of Situated Slopes.

How does it work?

Normalizing the image is part of the preprocessing process, but it is entirely optional. Its purpose is to improve the Hoard descriptor's execution. We do not use any preprocessing standards because we are only creating a simple descriptor.

2.2.2 Single Shot Detector (SSD)

The Single Shot Detector (SSD) is an engineering improvement for the VGG16 that calculates article discovery. It was released at the end of November 2016 and achieved new standards for object identification accuracy and execution, reaching over 74% Guide (mean Normal Accuracy) at 59 edges per second on common datasets such as PascalVOC and COCO. [9], [10]

Architecture

The SSD architecture builds on the well-known VGG-16 engineering but eliminates all the layers that go with it.

VGG-16 was chosen as the basic organization for the following reasons: a reputation for excellent accomplishment in top-notch picture order endeavors situations in which move learning aids in the development of results Instead of the initial VGG fully associated layers, several auxiliary convolutional layers (from conv6 onwards) were introduced, allowing for the separation of components at different scales and the logical fall in the amount of the contribution to each resultant layer.

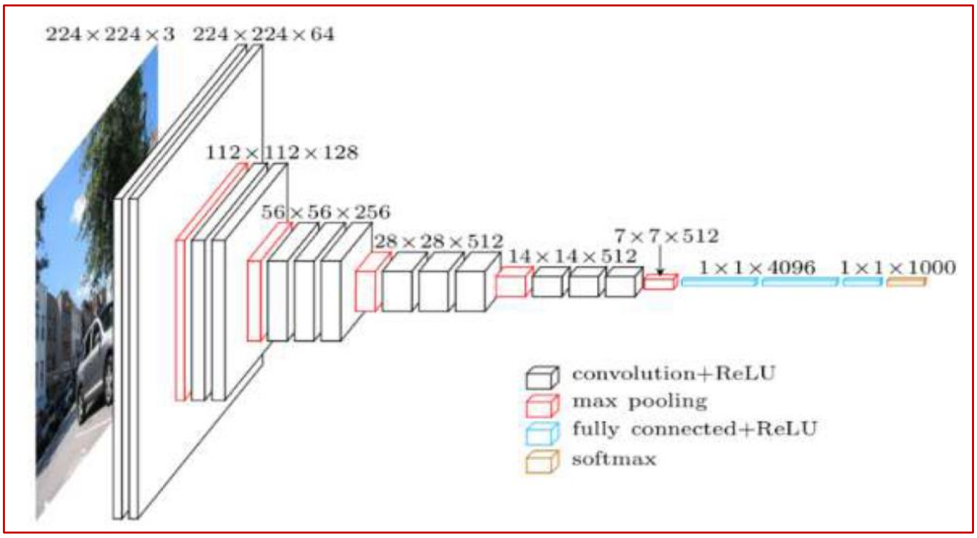


Figure 2-4: VGG-16 Architecture

2.2.3 Region-based Convolutional Neural Networks (R-CNN)

The Region-based Convolutions Network procedure (R CNN) is a blend of district proposal with Convolution Neural Networks (CNNs). R-CNN helps in confining articles with a huge affiliation and drawing up a high-line model with a smidgen of proportion of explained region information. It accomplishes shocking article conspicuous verification exactness by utilizing a critical ConvNet to organize object proposition. R-CNN can scale to a significant number of thing classes without going to procedures, including hashing.[7]–[11]

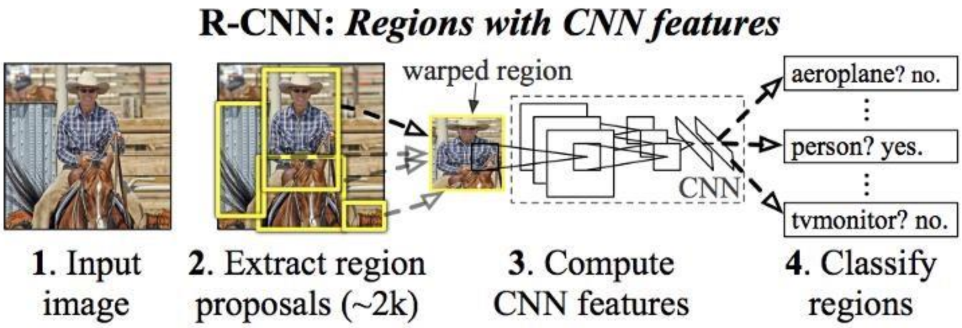


Figure 2-5: R-CNN system overview

1. Takes an image as input.
2. Input image as input.
3. Uses a massive convolutional neural network to compute features for each proposal (CNN).
4. Classify each area using class specific linear SVMs.

Drawbacks

- It finds Regions of Interest using the Selective Search Algorithm, which is a slow and time-consuming
- procedure that requires each image is used to categorize 2000 region proposals. As a result, training takes a long time. Detecting objects in a picture on GPU takes 49 seconds.
- A large amount of disk space is also required to store the region proposal's feature map.

2.2.4 Fast R-CNN

Fast R-CNN was a similar producer to the previous paper (R-CNN) that addressed a few problems of R-CNN to build a quicker article affirmation computation. The method looks to be like the R-CNN assessment. In any case, rather than controlling the region notion using CNN, we feed the information image to CNN to send a convolutional include map. We take the region of suggestion from the convolutional highlight map and curve it into squares, then reshape it into an acceptable size using a RoI pooling layer, so it can be managed into a completely relevant overall image.[11]–[16]

Based on the RoI highlight vector, we utilize a delicate max layer to forecast the class of the proposed zone and the offset respects for the jumping box. The reason (Fast R-CNN) is faster than R-CNN is because you do not have to manage 2000 locale concepts to the convolutional neural connection when in doubt. Taking everything into consideration, the convolution activity is done just once per picture, resulting in the generation of a section map.

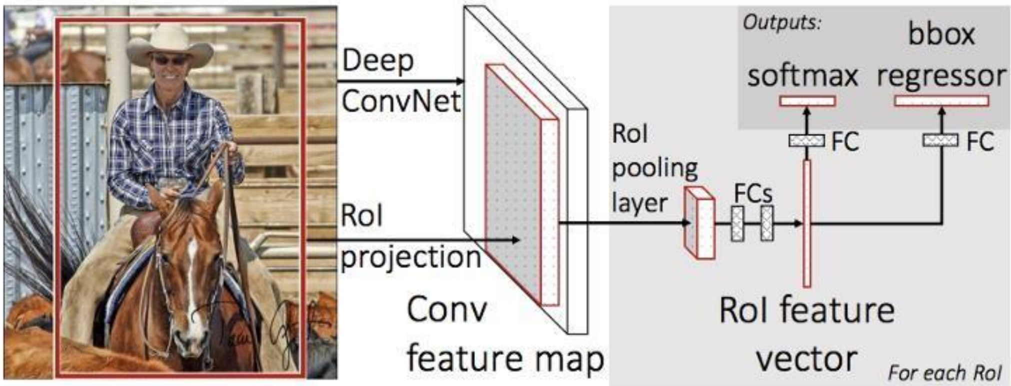


Figure 2-6: Fast R-CNN system overview

Benefits

- Higher identification quality (mAP) than R-CNN, SPPnet
- Training is single stage, utilizing a perform multiple tasks misfortune
- Training can refresh all organization layers
- No plate stockpiling is needed for include storing

2.2.5 Faster R-CNN

Both preceding algorithms (R-CNN and Fast R-CNN) employ requests to find area suggestions. Specific request is a torpid and monotonous cycle, impacting the introduction of the association. Accordingly, Shaoqing Ren, thought about a thing area computation that wipes out the pursuit estimation and permits the association to acquire capability with the region suggestion.[11]–[14], [17]–[19]

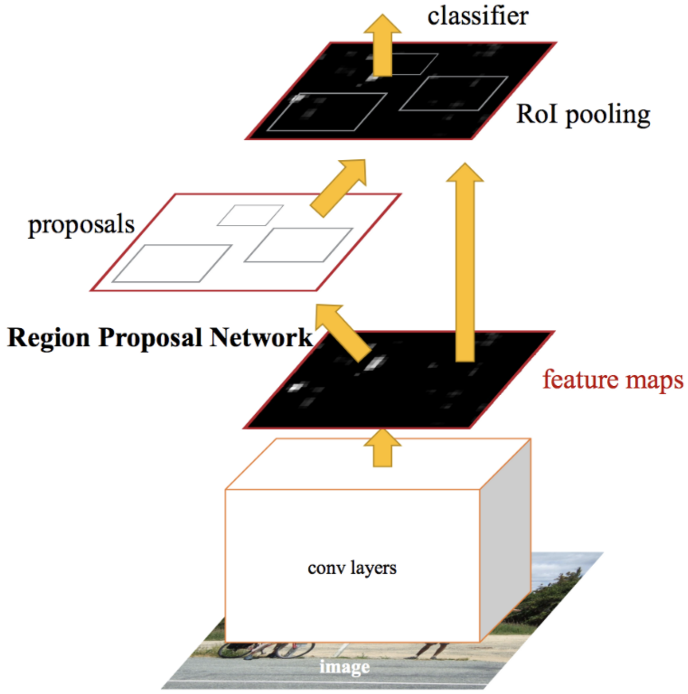


Figure 2-7: Faster R-CNN system overview

The picture is supplied as a commitment to a convolutional network that produces a convolutional incorporate guide, like Fast R-CNN. An alternate association is utilized to anticipate the district suggestions by utilizing special request calculation on the part manual for recognizing the region suggestion. A RoI pooling layer is then utilized to characterize the picture inside the suggested region and predict the offset respect for the skipping boxes, reshaping the expected region recommendations.

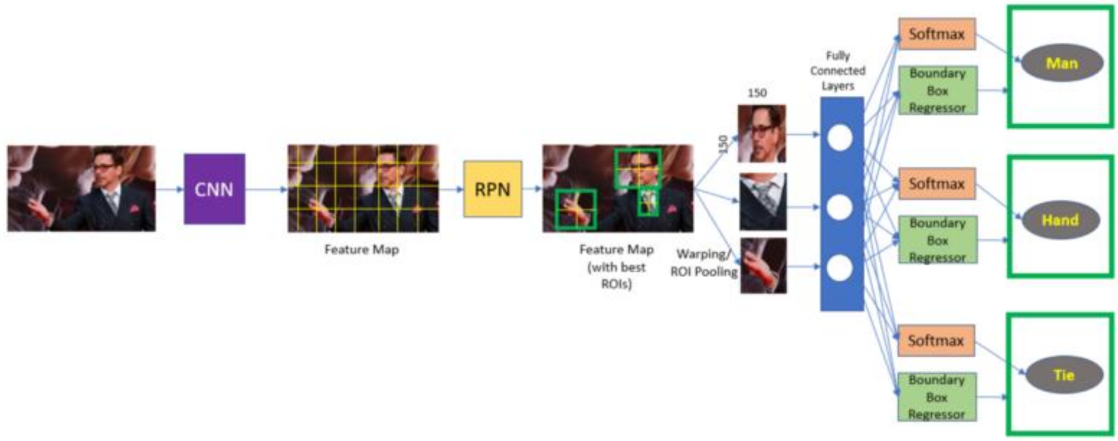


Figure 2-8: Faster R-CNN system analysis

Instead of Selective Search algorithm, it uses RPN (Region Proposal Network) to select the best ROIs automatically to be passed for ROI Pooling.

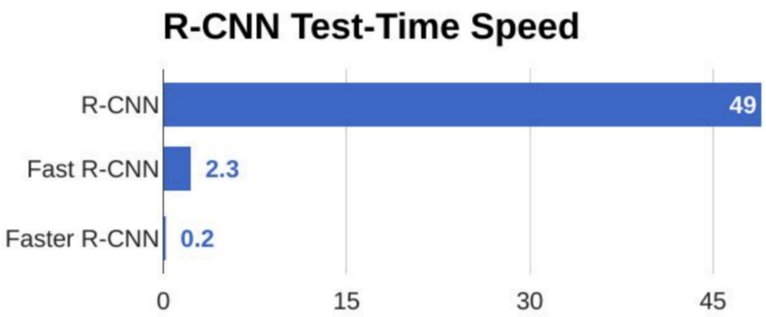


Figure 2-9: Comparison of test-time speed of object detection algorithms

Quicker R-CNN is clearly faster than its predecessors, as shown in the graph above. As a result, it may even be used to identify objects in real time.

2.2.6 Region-based Fully Convolutional Network (R-FCN)

District based Fully Convolutional Networks or R-FCN is an area-based locator for object identification. Not at all like other district-based locators that apply an exorbitant per-locale subnetwork like Fast R-CNN or Faster R-CNN, this area-based indicator is completely convolutional with all calculation shared on the whole picture.

R-FCN comprises of shared, completely convolutional designs just like the instance of FCN that is known to yield a preferred outcome over the Faster R-CNN. In this calculation, all learnable weight layers are convolutional and are intended to characterize the ROIs into object classifications and foundations.[7]

For conventional Region Proposal Network (RPN) approaches like R-CNN, Fast RCNN and Faster R-CNN, region suggestions are delivered by RPN first. Then ROI pooling is done and going through totally related (FC) layers for portrayal and bobbing box backslide. The collaboration (FC layers) after ROI pooling does not split among ROI, and saves time, which makes RPN approaches slow. Also, the FC layers increase the number of affiliations (limits) which moreover increase the multifaceted nature.

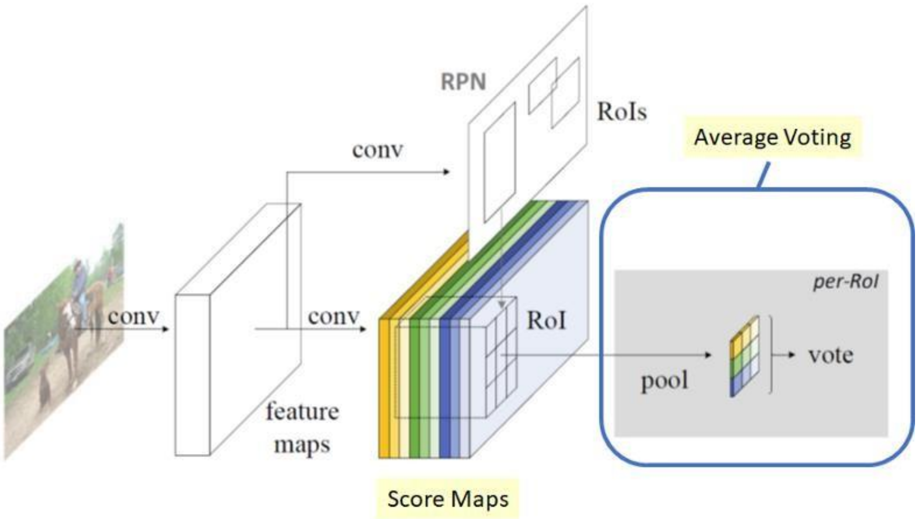


Figure 2-10: R-FCN system overview

In R-FCN, we use RPN to generate district recommendations, however unlike the R-CNN series, FC layers are removed after ROI pooling. All substantial complexity is shifted before ROI pooling to create the score maps, assuming all other factors are identical. Following ROI pooling, every district proposal will use a same set of score guidelines to conduct normal democratic, which is a fundamental estimation. As a result, there is no learnable layer after the ROI layer, which is costless. R-FCN is faster than Faster R-CNN with smaller mAP as a result.

2.2.7 You Just Look Once (YOLO)

YOLO is a real-time object identification technique that use neural networks. Because of its speed and precision, this algorithm is extremely popular. [20]–[23]

What is YOLO?

The term 'You Only Look Once' abbreviated as YOLO. This is an algorithm for detecting and recognizing different items in a photograph (in real-time). Object detection in YOLO done as a regression problem, and the identified photos' class probabilities provided.

Convolutional neural networks (CNN), that used in the YOLO method to recognize objects in real time. To detect objects, the approach just takes a single forward propagation through a neural network, as the name suggests.

This indicates that a single algorithm run used to forecast the entire image. The CNN used to forecast multiple bounding boxes and class probabilities at the same time.

The YOLO algorithm consists of various variants. include tiny

- YOLO1
- YOLO2
- YOLO3
- YOLO4
- YOLO5

What is the significance of the YOLO algorithm?

YOLO algorithm is important because of the following reasons:

- **Speed:** Because it can predict objects in real time, this approach enhances detection speed.
- **High accuracy:** YOLO is a predictive approach that yields precise findings with low background noise.
- **Learning capabilities:** The method has strong learning capabilities, allowing it to learn object representations and use them in object detection.

How the YOLO algorithm works

YOLO algorithm works using the following three techniques :

- **Residual blocks**
- **Bounding box regression**
- **Intersection Over Union (IOU)**

<u>Residual blocks</u>

The image separated first into several grids. The dimensions of each grid are $S \ x \ S$. The graphic below shows how a grid is created from an input image.

Figure 2-11: Residual blocks technique

There are several grid cells of identical size in the image above. Objects that appear within grid cells will be detected by each grid cell. If an item center emerges within a specific grid cell, for example, that cell will be responsible for detecting it.

Bounding box regression

A bounding box is an outline that draws attention to a certain object in a picture.

The following attributes are present in every bounding box in the image:

- Width: b_w
- a certain height: b_h
- class (for example, person, car, traffic light, etc.): c
- Center of the bounding box: (b_x, b_y)

A bounding box is illustrated in the image below. A yellow outline has been used to depict the bounding box.

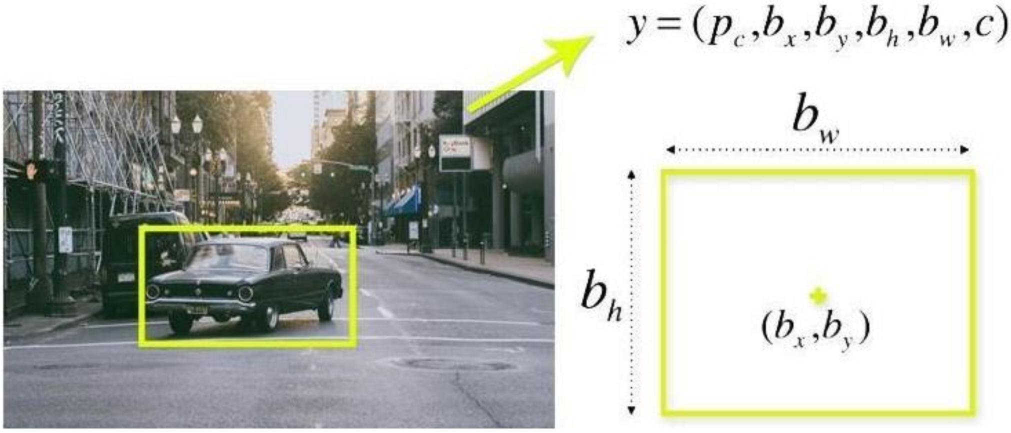


Figure 2-12: Bounding box regression technique

To forecast the height, width, center, and class of objects, YOLO use a single bounding box regression. The likelihood of an object appearing in the bounding box is represented in the graphic above.

Intersection Over Union (IOU)

The concept of intersection over union (IOU) illustrates how boxes overlap in object detection. YOLO uses IOU to create an output box that properly surrounds the items.

The bounding boxes and their confidence scores are predicted by each grid cell. If the anticipated and bounding boxes are identical, This approach removes bounding boxes that are not the same size as the actual box.

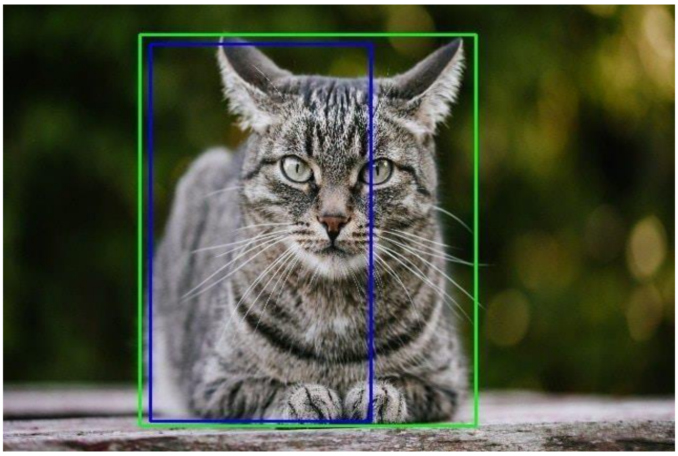


Figure 2-13: Intersection Over Union technique

There are two bounding boxes in the image above, one in green and the other in blue. The blue box represents the anticipated box, and the green box represents the actual box. YOLO makes sure the two boundary boxes are the same size.

Using a combination of the three techniques

The graphic below depicts how the three techniques are combined to generate the final detection findings.

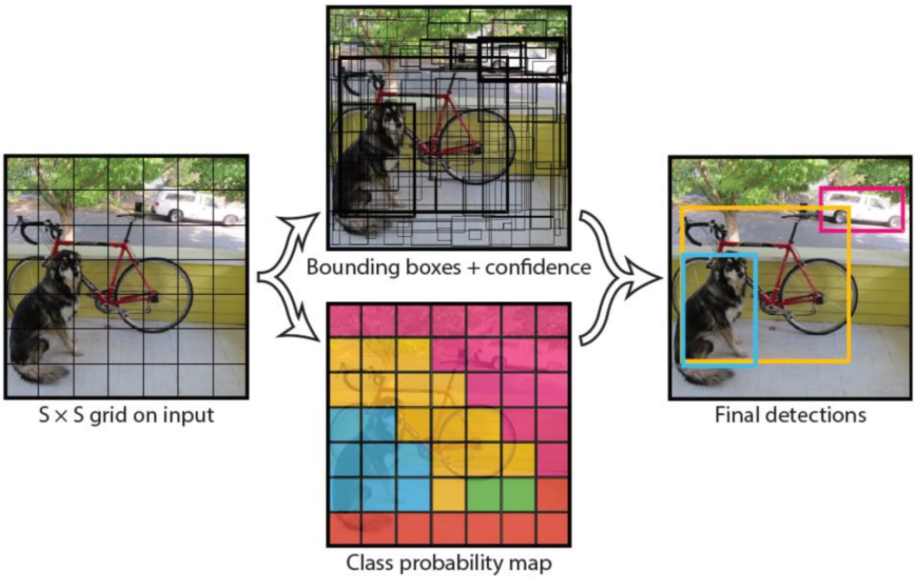


Figure 2-14: Combination of the techniques (Final Algorithm)

The image, first subdivided into grid cells. B bounding boxes are forecasted in each grid cell, along with their confidence scores. To determine the class of each object, the cells estimate the class probability. [24]

We can see at least three types of objects, for example: a car, a dog, and a bicycle. A single convolutional neural network used to make all the predictions at the same time.

The predicted bounding boxes are equal to the true boxes of the objects when intersection over union used. This phenomena gets rid of any extra bounding boxes that do not fit the objects' properties (like height and width). The final detection will be made up of distinct bounding boxes that exactly suit the objects.

The pink bounding box, for example, surrounds the car, whereas the yellow bounding box surrounds the bicycle. The blue bounding box has been used to highlight the dog.

YOLOv5

This version is incredible; it outperforms all prior versions and comes close to Efficient AP in terms of FPS. This can be seen in the graph below.

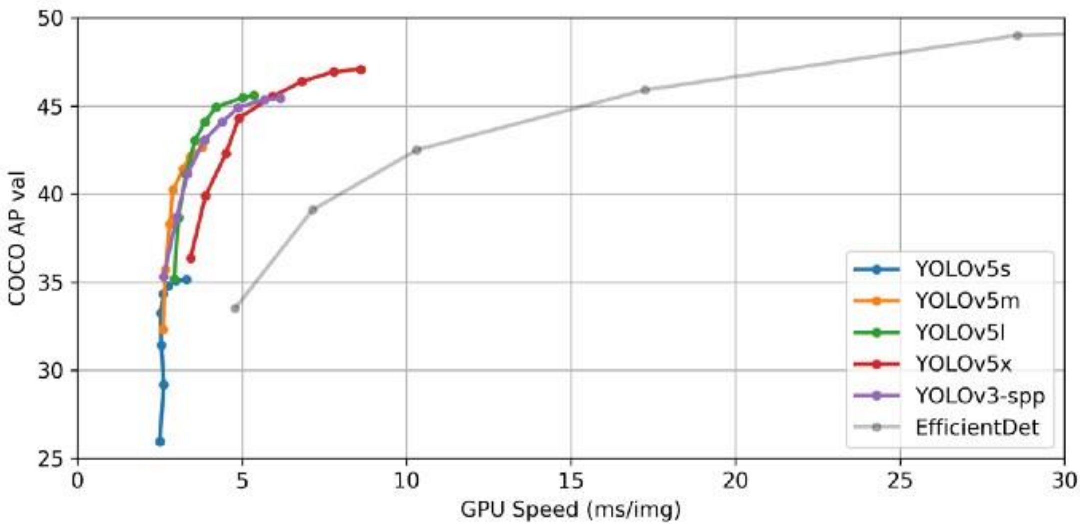


Figure 2-15: YOLO models comparison

The YOLOv5 model is the most current addition to the YOLO family of models. YOLO was the first object detection model to incorporate bounding box prediction and object classification into a single end-to-end differentiable network. It was created and is maintained using the Darknet framework. YOLOv5 is the first YOLO model to be written in the PyTorch framework, making it significantly lighter and easier to use. However, YOLOv5 does not outperform YOLOv4 on a standard benchmark, the COCO dataset, because it did not make fundamental architectural improvements to the network in YOLOv4. [25]

Data Augmentation in YOLOv5

YOLOv5 transmits training data through a data loader for online augmentation with each training batch. The data loader performs three types of augmentations: scaling, correction of the color space, and mosaic enhancement. The newest technique is mosaic data augmentation, which transforms four photos into four random ratio tiles. The mosaic data loader included as part of the YOLOv3 PyTorch and, more recently, YOLOv5 repositories. Mosaic augmentation is particularly beneficial for the widely used COCO object identification benchmark, since it aids the model in learning to address the well-known "small object problem" - in which little items are not spotted as reliably as bigger objects. It is worth noting that it is worthwhile to experiment with your own set of augmentations to optimize performance on your customized work.

Both YOLOv4 and YOLOv5 use the CSP Bottleneck to generate picture features, with credit for the research going to WongKinYiu and their recent study on Cross Stage Partial Networks for Convolutional Neural Network Backbone. The CSP eliminates duplicate gradient difficulties found in other larger ConvNet backbones, resulting in fewer parameters and FLOPS for equivalent significance. This is critical for the YOLO family, as inference speed and a small model size are critical. [25]

CSP Backbone

Dense Net used to construct the CSP models. Dense Net was created to connect layers in convolutional neural networks to alleviate the vanishing gradient problem (it is difficult to backprop loss signals through a very deep network), to improve feature propagation, to encourage the network to reuse features, and to reduce the number of network parameters.

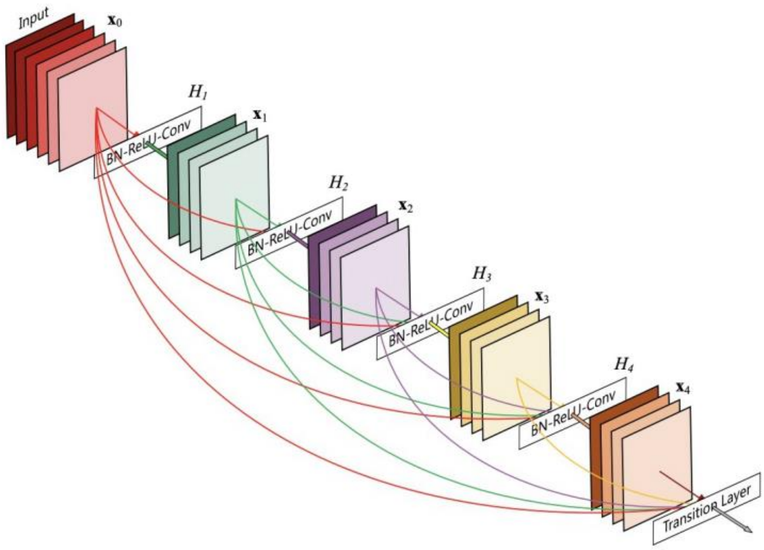


Figure 2-16: Structure of DenseNet (DenseBlock)

<u>General Quality of Life Updates for Developer</u>

In comparison to other object detection frameworks, YOLOv5 is incredibly simple to integrate into an application by a developer. These updates to my quality of life classified as follows. [26]

- **Simple Installation** - YOLOv5 requires simply torch and a few lightweight python modules.
- **Rapid Training** - The YOLOv5 models train exceptionally quickly, allowing you to save money on testing while building your model.
- **Working Inference Ports** - YOLOv5 supports inference on individual photos, batch images, video feeds, and webcam ports.
- **Intuitive File Folder Layout** - While working, the file folder layout is intuitive and simple to traverse.
- **Simple Translation to Mobile** - YOLOv5 can be easily translated from PyTorch weights to ONXX weights to CoreML weights and finally to iOS.

Conclusion

YOLOv5's initial release is extremely fast, performant, and simple to use. While YOLOv5 does not provide novel model architecture enhancements to the YOLO model family, it does introduce a new PyTorch training and deployment methodology that advances the state of the art for object detectors. Additionally, YOLOv5 is extremely user-friendly and comes pre-configured for use with custom objects "out of the box."

2.2.8 Comparisons

While YOLOv3 is quite high and far to the left, you can know it is well. Is it possible for you to cite your own work? Guess who is going to give it a shot, this man. Oh, and we also fixed a data loading error in YOLOv2, which saved us about 2 mAP. I'm just slipping this in here to keep the layout from being thrown off.[24], [27]

Table 2-2: Algorithms Comparison (Speed)

Algorithm	m/AP	ms
YOLOv2	21.6	25
R-FCN	29,9	85
SSD513	31.2	125
FPN FRCN	36.2	172
YOLOv3-329	28.2	22
YOLOv3-416	31	29
YOLOv3-608	33	51

While 63.4 mAP (mean average prediction) and 45 FPS in YOLO mode. YOLO can achieve real-time performance with similar mAP as R-CNN, Fast R-CNN, and Faster R-CNN. After we know all the algorithms and study their properties, we will do a comparison between them, which one is faster and takes more evidence.[28]

Table 2-3 Algorithms Comparison (Performance)

Algorithm	mAP	FPS
R-CNN	53.5	6
Fast R-CNN	70	0.5
Faster R-CNN	73.2	7
Faster R-CNN ZF	62.1	18
YOLO VGG-16	66.4	21

2.2.9 Summary

We learned a light introduction to the topic of object identification in this post, as well as state-of-the-art deep learning models meant to solve it.

Also compare these algorithms to see which one is better, The phrase "object recognition" refers to a set of actions that are used to recognize objects in digital pictures.

- **HOG**, like the Canny Edge Detector and SIFT, is a feature descriptor (Scale Invariant and Feature Transform).
- **SSD**, To detect several items within an image, just one shot is required.
- **RCNN**, are a class of approaches for tackling object identification and localization problems that are optimized for model performance.
- **YOLO**, group of object identification algorithms that are geared for speed and real-time application.

Finally, we can say that the difference between the two most important methods of detecting that a group R-CNN is characterized by superior performance and a group YOLO characterized by real-time object detection

2.3 DATASETS

A set of data that is dealt with as a single unit by a computer. This means that a data set contains a lot of discrete pieces of data but can be used to train an algorithm with the goal of finding predictable patterns within the entire data set.

As mentioned in the previous chapter, we searched for open-source datasets and filtered them to the ones in this report. It relied heavily on the ones in GitHub, and the research focused on different types of weapons in this project, and people.

2.3.1 Crime Detection – using Deep learning

This project used YOLO Darknet framework. Project's Datasets is about Crime, and it is manually obtained from Google photos as well as the ImageNet database for crime detection. It has 3150 photos, txt files. It's categories: Gun, Knife, Person. [29], [30]

2.3.2 Weapon detection datasets

These datasets used converted YOLOV5 to practice. Project's Datasets is Primarily concerned with the development of intelligent video surveillance automatic systems. [31], It is from: Andalusian Research Institute in Data Science and Computational Intelligence (DaSCI). It has 18097 photos, txt files. It's categories: Pistol, Knife, Weapons, and similar handled objects [32]

2.3.3 Weapon Detection and Classification

These datasets used Deep Learning CNNs to practice. Weapon Detection & Classification through CCTV surveillance. It has 13215 photos. It's categories: Knife, Small Gun.[33]

2.3.4 Handgun Dataset

This dataset was used on the paper titled "Firearm Detection from Surveillance Cameras Using Image Processing and Machine Learning Techniques." It has Consists of positive (Handgun) Images and negative images (Images of various objects), in total 1900 photos. It's categories: Gun. [34], [35]

2.3.5 Knife Dataset

There are 400 training photos and 100 test images of knives in the dataset. It has 500 photos. It's categories: knife [36]

2.3.6 Summary

Table 2-4: Summary of Datasets

Dataset	Gun	Rifle	Knife	People	Total
1. Crime Detection	2000	-	1050	100	3150
2. Weapon detection	10770	187	7140	-	18097
3. Weapon Classification	315	-	12900	-	13215
4. Handgun Dataset	1900	-	-	-	1900
5. Knife Dataset	-	-	500	-	500
Total Datasets	14985	187	21590	100	36862

Below, there is a percentage representation for each category in each dataset

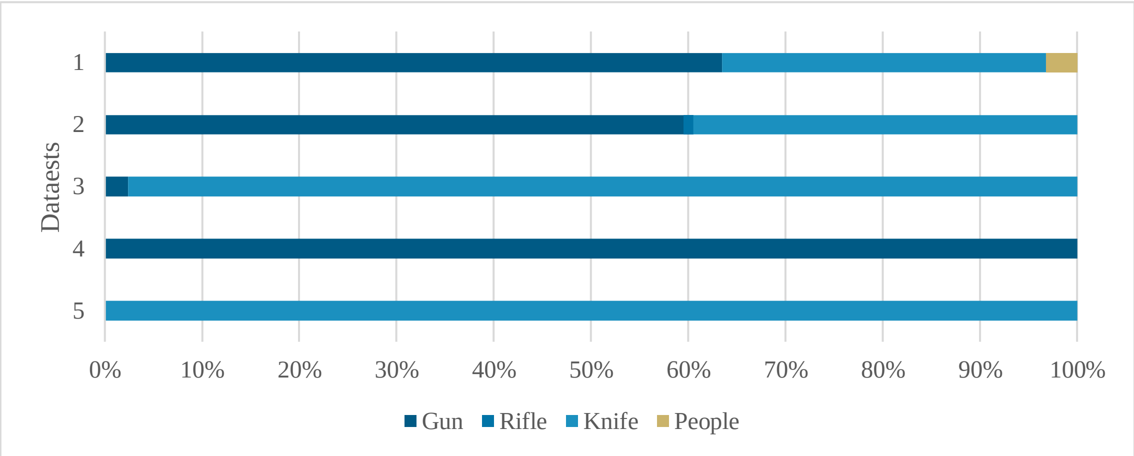


Figure 2-17: Summary of Datasets (Data Type %)

Chapter There

3 ANALYSIS AND DESIGN

In this chapter, the methods used in system analysis and design, as well as the user interfaces, will be explained.

That Include UML, functionality explanations, and user interfaces.

3.1 FUNCTIONAL REQUIREMENTS

Main Function that we implemented to enable users to accomplish their task

Table 3-1: Functional Requirements

1. Login	To allow the user to enter the system and browse the camera, a login must be made to the system
2. Create account	If the user is new, you must create an account for the system to allow that to log in to the system
3. Identify a person holding gun	One of the most crucial functions of the system is how to identify any person carrying a gun by taking a picture of him and determining the type of detection
4. Identify a person holding knives	In many cases, the criminal is carrying our knife and passing through many monitoring devices, so it is a function in the system to identify anyone who carries a knife for the safety of customers.
5. Identify a person holding Trifle	The type of detection differs, and one of the most important types of guns is that one of the functions of the system is to identify the Trifle, if it is found with the criminal, and show it to the user.
6. Create an alarm if model knows that there is something that triggers an alarm	When the system recognizes the detection, it must send a warning to inform the users of the presence of danger and to take the necessary measures also to inform the customers to immunize themselves
7. Display camera stream	The camera content of the system must be displayed to determine the detection of the user who is following the system in action
8. Logout	The user must be logged out for the safety of the system if he finished his work or if he will leave the place

3.2 NON-FUNCTIONAL REQUIREMENTS

Functions that we implemented to define the fundamental behavior of the system

Table 3-2: Non-Functional Requirements

1. System security	Protect data from external attacks
2. Visibility all 24/7	Live information that allows us to monitor the possible danger and their exact location at any given point.
3. Accuracy in recognizing object detection	Full accuracy in taking out the image or detection clearly
4. Speed in recognizing object detection	One of the most key features that must be in the system is that it can quickly detect images to eliminate the danger
5. Preserve the privacy of the place	The system must preserve its data and not allow anyone who is not responsible for it or those who have the right to see the system

3.3 USE-CASE DESCRIPTIONS

a text-based narrative of a functionality comprised of detailed, step-by-step interaction according to each function

3.3.1 Login

To access the system and use it

Table 3-3: Use-Case Description (Login)

Job name	Login
Related Requirements	Admin, User
Initiating Actor	Admin, User
Actor's Goal	To login and use the model
Participating Actors	None
Preconditions	None
Postconditions	User is logged in

Flow of Events for Main Success Scenario	1. User enters the username, password and clicks login. 2. system authenticates the sent info, after the authentication is successful, system lets the user login.
Flow of Events for Alternate Scenario	1. User enters the username, password and clicks Login. 2. system authenticates the sent info, after the authentication is false, system prompts the user (username or password is incorrect).

3.3.2 Add User

Admin can add new employees to access the system

Table 3-4: Use-Case Description (Add User)

Job name	Login
Related Requirements	Admin
Initiating Actor	Admin
Actor’s Goal	To Register the User and use the model
Participating Actors	None
Preconditions	None
Postconditions	Be logged in
Flow of Events for Main Success Scenario	1. Admin clicks register. 2. System returns registration page. 3. Admin fills in required information, then clicks create. 4. System prompts user “account creation was successful”.
Flow of Events for Alternate Scenario	1. User enters the username, password and clicks Login. 2. System authenticates the sent info, after the authentication is false, system prompts the user (username or password is incorrect).

3.3.3 View Camera

The ability for the user to display the content on the monitor

Table 3-5: Use-Case Description (View Camera)

Job name	View Camera
Related Requirements	Admin, User
Initiating Actor	Admin, User
Actor's Goal	To see the Camera's
Participating Actors	None
Preconditions	None
Postconditions	None
Flow of Events for Main Success Scenario	1. User clicks View Camera 2. System returns the display of the camera's
Flow of Events for Alternate Scenario	None

3.3.4 Use Model

The system can access the cameras and use the mode

Table 3-6: Use-Case Description (Use Model)

Job name	Use Model
Related Requirements	System
Initiating Actor	System
Actor's Goal	To use the model
Participating Actors	None
Preconditions	None
Postconditions	System using the model

Flow of Events for Main Success Scenario	1. System use the model 2. Determine object's type 3. Detect the object's 4. If the object's is a weapon, issue an alert
Flow of Events for Alternate Scenario	1-3. -- 4. If the object's is not a weapon, do not issue an alert

3.3.5 Determine Object's Type

Analyzing the video stream to decide the type of the object

Table 3-7: Use-Case Description (Determine Object's Type)

Job name	Determine Object's Type
Related Requirements	System
Initiating Actor	System
Actor's Goal	To analyze and classify the objects
Participating Actors	None
Preconditions	Use model
Postconditions	Object detection
Flow of Events for Main Success Scenario	1. System use the model 2. Use camera to view 3. Analyze to determine object
Flow of Events for Alternate Scenario	None

3.3.6 Object Detection

Analyzing the video stream to detect each object

Table 3-8: Use-Case Description (Object Detection)

Job name	Object Detection
Related Requirements	System

Initiating Actor	System
Actor's Goal	To detect objects
Participating Actors	None
Preconditions	None
Postconditions	Detect object (weapon)
Flow of Events for Main Success Scenario	1. System use the model 2. Use the camera's 3. Determine objects 4. Detect the objects
Flow of Events for Alternate Scenario	None

3.3.7 Issuing an alert

When weapon is detected, the system will announce an alarm to alert the surrounding

Table 3-9: Use-Case Description (Issuing an alert)

Job name	Issuing an alert
Related Requirements	System
Initiating Actor	System
Actor's Goal	To make an alert if the system detect weapons
Participating Actors	None
Preconditions	None
Postconditions	Alert the users only if weapon detected
Flow of Events for Main Success Scenario	1. System use the model 2. Determine object's type 3. Detect the object's 4. If the object's is a weapon, issue an alert
Flow of Events for Alternate Scenario	1-3. -- 4. If the object's is not a weapon, do not issue an alert

3.3.8 Logout

When the employee using the system is done with his work, logout so others can login

Table 3-10: Use-Case Description (Logout)

Job name	Logout
Related Requirements	Admin, User
Initiating Actor	Admin, User
Actor's Goal	To logout of the model
Participating Actors	Admin, User
Preconditions	None
Postconditions	User logged out
Flow of Events for Main Success Scenario	1. User clicks log out. 2. system authenticates lets the user log out
Flow of Events for Alternate Scenario	None

3.4 GRAPHIPHS USER INTERFASES

a form of user interface that allows users to interact with electronic devices through graphical icons and audio indicator such as primary notation.

We design a GUI’s according to the main two users, with United Login Interface

- **Admin**
  - Main page
  - Add User
  - View Camera
- **Users**
  - Main page
  - View Camera

3.4.1 Login

The login page requires a valid username and a password

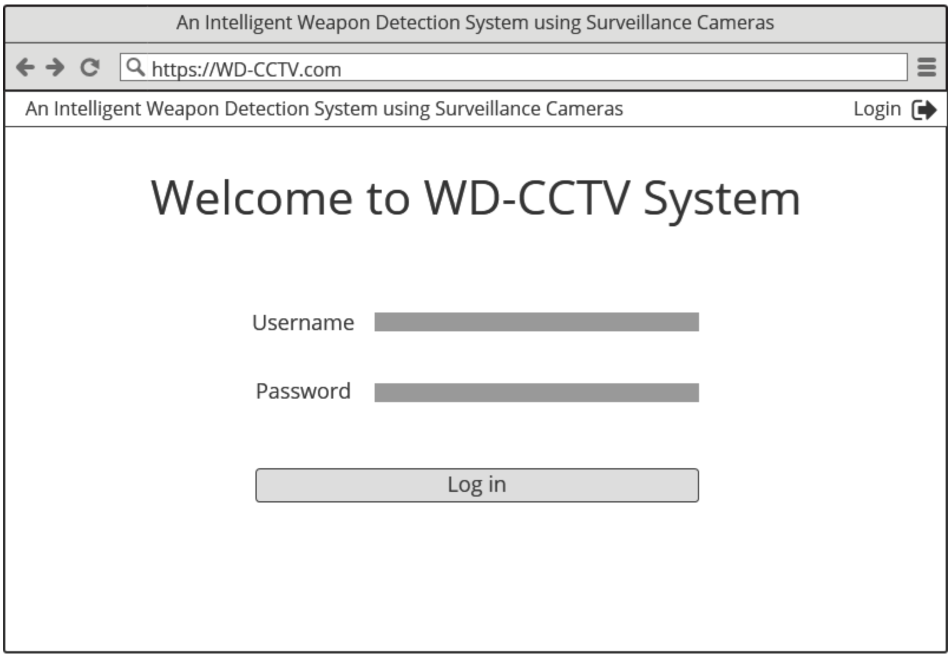


Figure 3-1: GUI (Login)

3.4.2 Main page (User)

the main page will show up after a success login, displaying the options for the (User)

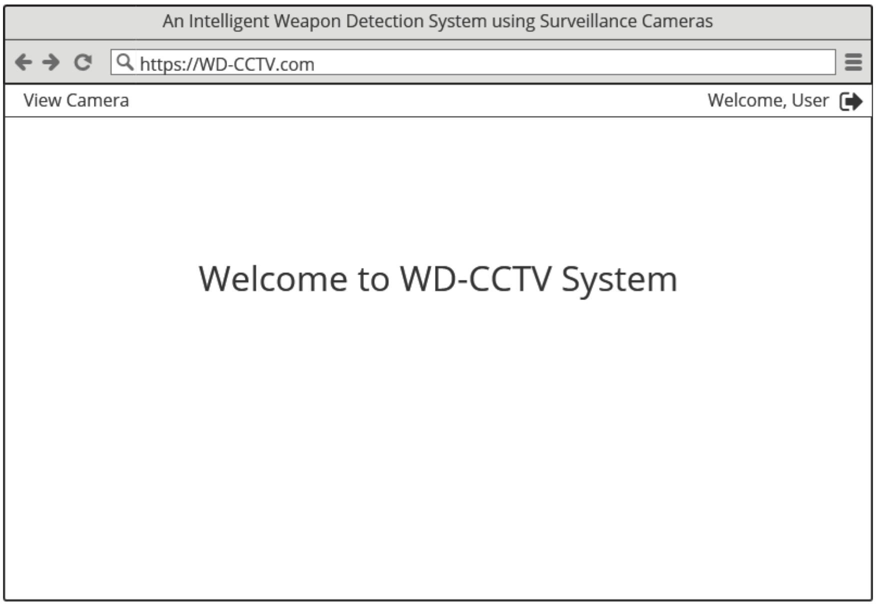


Figure 3-2: GUI (Main page [User])

3.4.3 View Camera (User)

let the user access to the cameras and display the content on the monitor

Figure 3-3: GUI [View Camera (User)]

3.4.4 Main page (Admin)

the main page will show up after a success login, displaying the options for the (Admin)

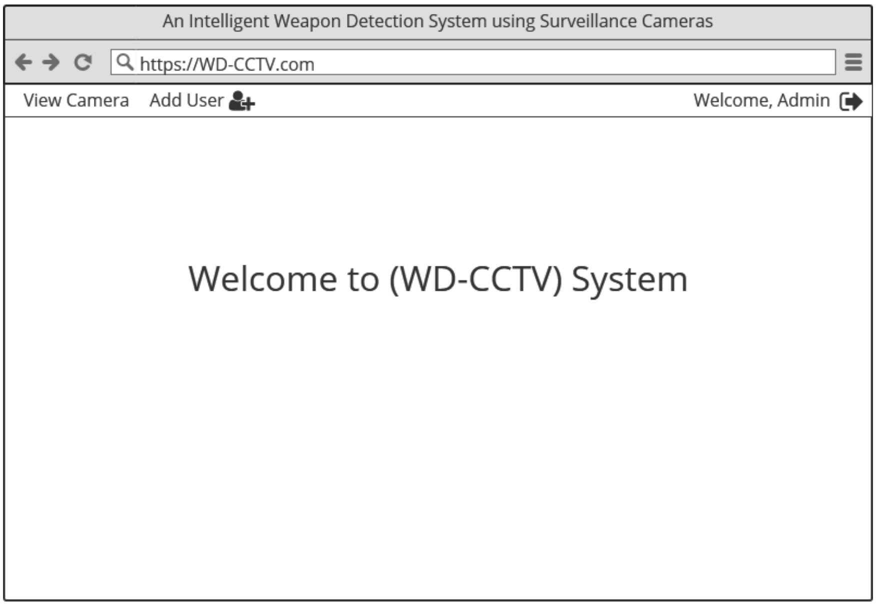


Figure 3-4: Main page (Admin)

3.4.5 Add User

the main page will show up after a success login, displaying the options for the (Admin)

Figure 3-5: GUI (Add User)

3.4.6 View Camera (Admin)

to let the user access to the cameras and display the content on the monitor

Figure 3-6: View Camera (Admin)

3.5 UNIFIED MODELING LANGUAGE (UML) DIAGRAMS

in software engineering field that is intended to provide a standard way to visualize the design of a system.

We choose 3 main kinds of UML Diagrams

- **Entity Relationship (ER)**
  The main and prime diagram of database.
- **Use-Case**
  The main diagram of project’s actors and functions relationship.
- **Activity**
  The diagram of project’s way of prosses.

3.5.1 Entity Relationship (ER)

Chen ERD

This graphic describes the relationships between an entity and how the system in the database operates

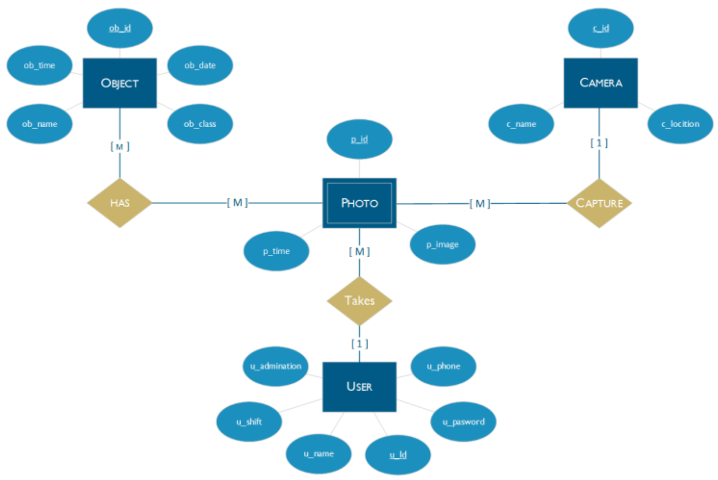


Figure 3-7: Chen ERD

ERD Schema

As for here, it is more clarification about the relationships in the database and its internal types

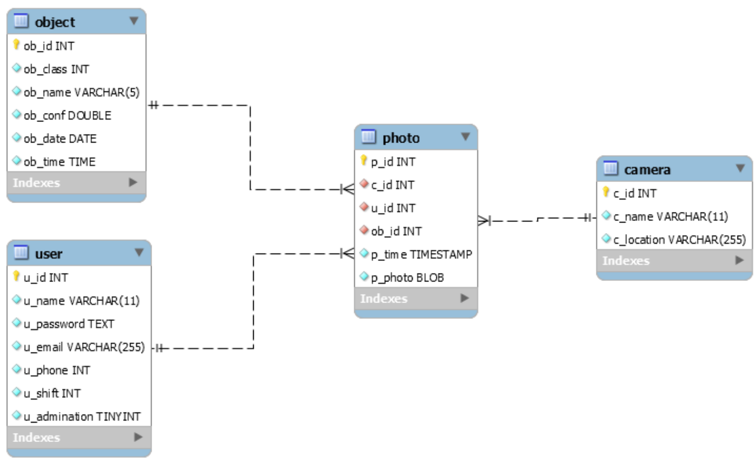


Figure 3-8: ERD Schema

3.5.2 Use-Case

The Use-Case describes the most important actors in the system and how the system works

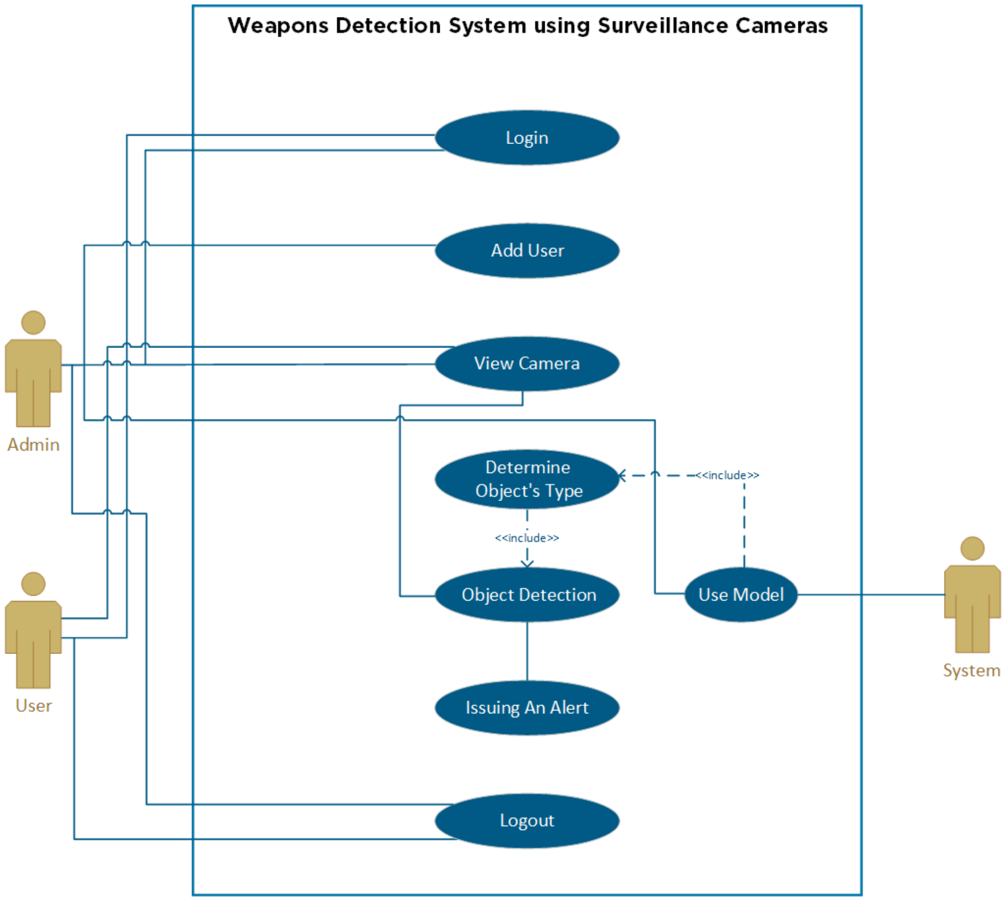


Figure 3-9: UML Use-Case

3.5.3 Activity

A graphical diagram to show the process of the how the model work in the System

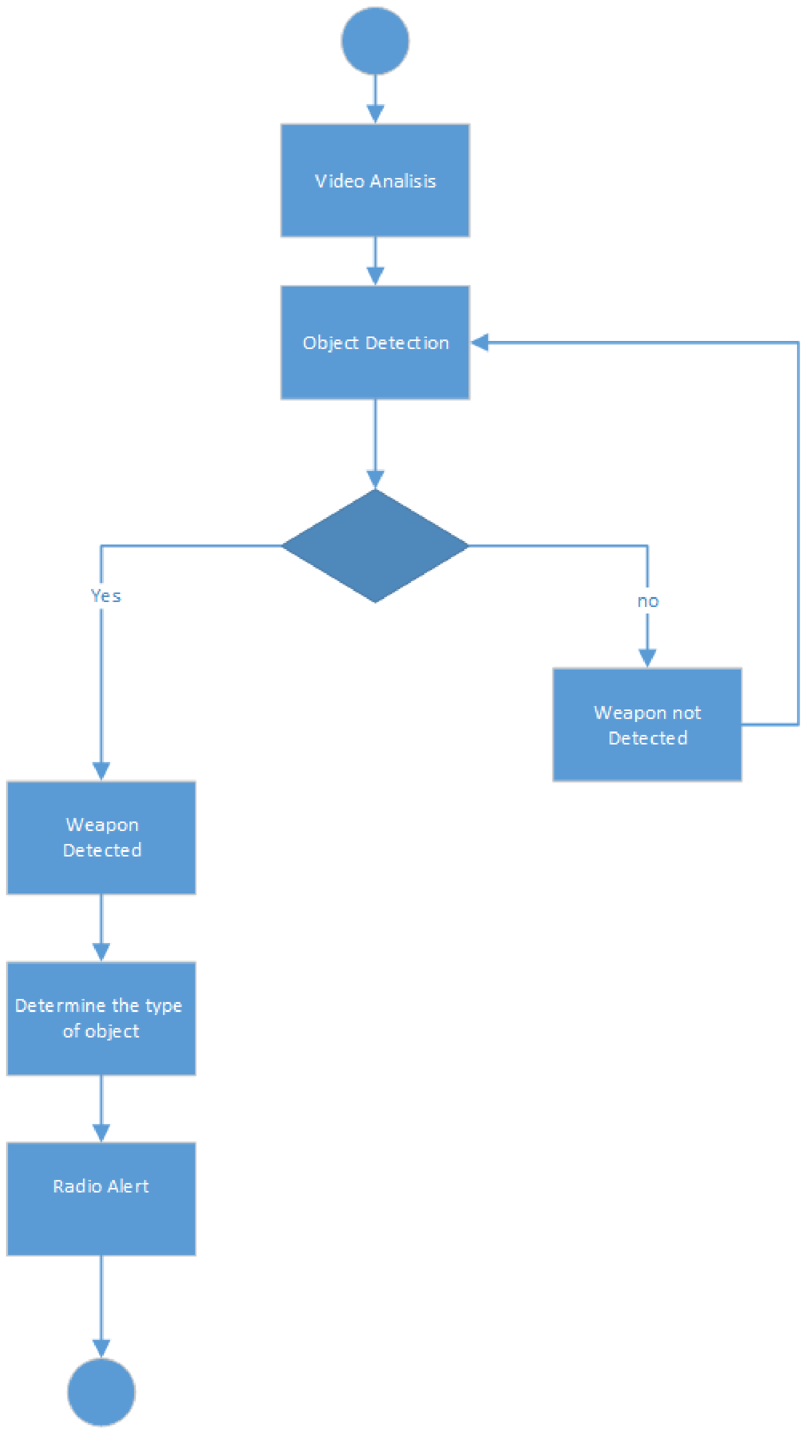


Figure 3-10: UML Activity

Chapter Four

4 IMPLEMENTATION

In this chapter, the implementation method of the system will be explained, based on the above chapters. This chapter will be divided into main sections :modified datasets, algorithm training, testing & inferencing trained model, and linking to a website system.

4.1 MODIFIED DATASET

In 2^{nd} chapter. We mentioned that we have about 37,000 collected and prepared. It consists of 21500 knives, 15000 guns, 190 rifles. In the first phase of implementation, we set up 6000 to be a sample to train the algorithm. But we should make these datasets annotated with the chosen Algorithm.

We are having some famous types of Object detection annotation

1. Pascal VOC (.XML): the most famous annotation type
2. CreateML (.JSON)
3. YOLO (.TXT)

In next lines, we will take about YOLO Dataset annotation

4.1.1 YOLO Dataset annotation format

An Object detection annotation format, that having a text file per each picture (containing the annotations and a numeric representation of the label) and a label map (which translates the numeric IDs to human readable strings) are included in this format. The annotations are normalized to lie between 0 and 1, making them easier to deal with even after resizing or extending the photos. [37]

In next code, a sample of annotated picture that have 3 different labels, in our project we are going to detect 3 types of weapon (Kinfe, Gun, and riffle); that labels in same order in code.

Img.txt	0 0.716797 0.395833 0.216406 0.147222 1 0.687109 0.379167 0.255469 0.158333 2 0.420312 0.395833 0.140625 0.166667

Code 4-1: Sample of YOLO annotation format

How to get YOLO format?

There is 2 main ways to get that annotation format, to be compatible with our project. Either we convert one of the most user XML format or design it from scratch using LabelImg.

<u>XML → YOLO</u>

In fact, XML is the most used format to annotate dataset. The 1st way to prepare it for our project.

we should convert it by using a code to convert annotation from XML to YOLO

xml_to_yolo.txt

```
import glob
import os
import pickle
import xml.etree.ElementTree as ET
from os import listdir, getcwd
from os.path import join

dirs = ['']
classes = ['knife', 'gun']

def getImagesInDir(dir_path):
    image_list = []
    for filename in glob.glob(dir_path + '/*.jpg'):
        image_list.append(filename)

    return image_list

def convert(size, box):
    dw = 1./(size[0])
    dh = 1./(size[1])
    x = (box[0] + box[1])/2.0 - 1
    y = (box[2] + box[3])/2.0 - 1
    w = box[1] - box[0]
    h = box[3] - box[2]
    x = x*dw
    w = w*dw
    y = y*dh
    h = h*dh
    return (x,y,w,h)
```

```
def convert_annotation(dir_path, output_path, image_path):
    basename = os.path.basename(image_path)
    basename_no_ext = os.path.splitext(basename)[0]

    in_file = open(dir_path + '\\' + basename_no_ext +
'.xml')
    out_file = open(output_path + basename_no_ext + '.txt',
'w')
    tree = ET.parse(in_file)
    root = tree.getroot()
    size = root.find('size')
    w = int(size.find('width').text)
    h = int(size.find('height').text)

    for obj in root.iter('object'):
        difficult = obj.find('difficult').text
        cls = obj.find('name').text
        if cls not in classes or int(difficult)==1:
            continue
        cls_id = classes.index(cls)
        xmlbox = obj.find('bndbox')
        b = (float(xmlbox.find('xmin').text),
float(xmlbox.find('xmax').text),
float(xmlbox.find('ymin').text),
float(xmlbox.find('ymax').text))
        bb = convert((w,h), b)
        out_file.write(str(cls_id) + " " + " ".join([str(a)
for a in bb]) + '\n')

cwd = getcwd()

for dir_path in dirs:
    full_dir_path = cwd + '\\' + dir_path
    output_path = full_dir_path +'/yolo/'

    if not os.path.exists(output_path):
        os.makedirs(output_path)

    image_paths = getImagesInDir(full_dir_path)
    list_file = open(full_dir_path + '.txt', 'w')

    for image_path in image_paths:
        list_file.write(image_path + '\n')
        convert_annotation(full_dir_path, output_path,
image_path)
    list_file.close()

    print("Finished processing: " + dir_path)
```

Code 4-2: Convert annotation format (XML) to (YOLO)

Generate YOLO format

The second way to get annotate dataset is by generate it with software called LabelImg. that is a graphical image annotation tool. saves annotation as XML, CreateML, and YOLO formats. [37]

Through the next few screenshots, we will briefly explain how the program works?

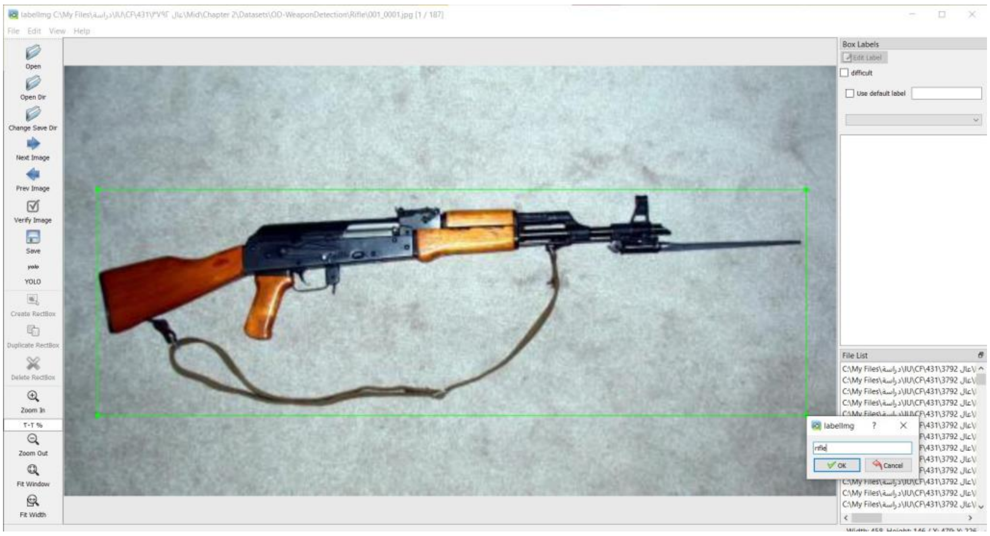


Figure 4-1: Draw a rectangular for an object and label it

Figure 4-2: a Labeled rectangular for an object (Annotated Dataset)

Figure 4-3: Labeled rectangles for many object (Annotated Datasets)

4.2 ALGORITHM TRAINING

In 2nd chapter. We mentioned that YOLO5 is chosen Algorithm, the new thing that we should choose one of them to train it.

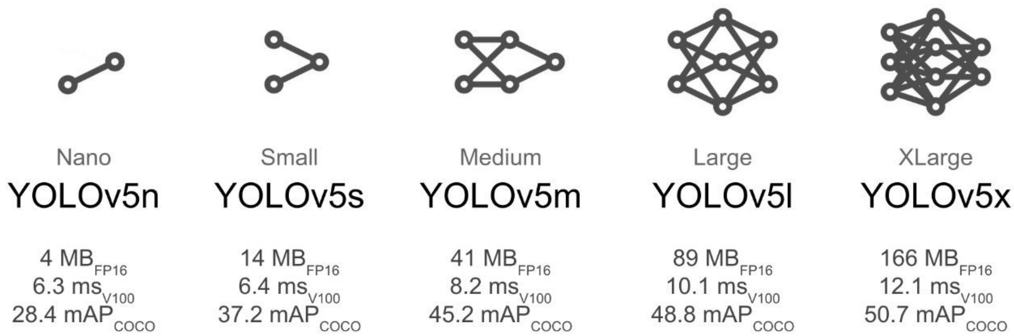


Figure 4-4: Mini Comparison of YOLO5 models

Now, according to our device's specifications, and the priority of models. The chosen models of YOLO5 Algorithm are YOLOv5m and YOLOv5x.

Table 4-1: Full Comparison of YOLO5 models

Model	Size (pixels)	mAP val 0.5:0.95	mAP val 0.5	Speed CPU b1 (ms)	Speed V100 b1 (ms)	Speed V100 b32 (ms)	params (M)	FLOPs @640 (B)
YOLOv5n	640	28.0	45.7	45	6.3	0.6	1.9	4.5
YOLOv5s	640	37.4	56.8	98	6.4	0.9	7.2	16.5
YOLOv5m	640	45.4	64.1	224	8.2	1.7	21.2	49.0
YOLOv5l	640	49.0	67.3	430	10.1	2.7	46.5	109.1
YOLOv5x	640	50.7	68.9	766	12.1	4.8	86.7	205.7

4.2.1 1st train: YOLOv5m

We use the annotated datasets, that has been processed and explained how in the previous pages. the device specifications that used for the train is: (Intel(R) Core(TM) i7-7500U CPU @ 2.70GHz, NVIDIA GeForce 940MX - 4GB, DDR4 - 16GB). All codes needed for training will be explained in the next lines.

train.py

```
import os

os.system("python train.py --img 640 --batch 8 --epochs 24
--data weapons.yaml¹ --weights yolov5m.pt")
```

Code 4-3: Main code of training

We train this model with (Image scaling to 640, 8 batches, within 24 epochs). Then, we should define what is inside the code of datasets section

weapons.yaml [1]

```
path:  ...\GP2\GP2_files\data\Full_DS\images
train: train #train images (relative to 'path') 5000 images
val: val #train images (relative to 'path') 1000 images

# Classes
nc: 3 # number of classes
names: ['knife','gun','rifle'] # class names
```

Code 4-4: Sample of YOLO annotation format

Finally, the notes of the trained code with the figures of trained model

```
C:\Users\moco_\Documents\Python\GP2\python_oct19\python.exe
C:/Users/moco_/Documents/Python/GP2/GP2_files/3_train_1.py
train: weights=yolov5m.pt, cfg=, data=weapons_dataset.yaml,
hyp=data\hyps\hyp.scratch.yaml, epochs=24, batch_size=8, imgsz=640,
rect=False, resume=False, nosave=False, noval=False, noautoanchor=False,
evolve=None, bucket=, cache=None, image_weights=False, device=,
multi_scale=False, single_cls=False, optimizer=SGD, sync_bn=False,
workers=8, project=runs\train, name=exp, exist_ok=False, quad=False,
linear_lr=False, label_smoothing=0.0, patience=100, freeze=[0],
save_period=-1, local_rank=-1, entity=None, upload_dataset=False,
bbox_interval=-1, artifact_alias=latest
github: skipping check (not a git repository), for updates see
https://github.com/ultralytics/yolov5
YOLOv5  2022-2-17 torch 1.9.1 CUDA:0 (NVIDIA GeForce 940MX, 4096MiB)

hyperparameters: lr0=0.01, lrf=0.1, momentum=0.937, weight_decay=0.0005,
warmup_epochs=3.0, warmup_momentum=0.8, warmup_bias_lr=0.1, box=0.05,
cls=0.5, cls_pw=1.0, obj=1.0, obj_pw=1.0, iou_t=0.2, anchor_t=4.0,
fl_gamma=0.0, hsv_h=0.015, hsv_s=0.7, hsv_v=0.4, degrees=0.0, translate=0.1,
scale=0.5, shear=0.0, perspective=0.0, flipud=0.0, fliplr=0.5, mosaic=1.0,
mixup=0.0, copy_paste=0.0
TensorBoard: Start with 'tensorboard --logdir runs\train', view at
http://localhost:6006/
Overriding model.yaml nc=80 with nc=3

from  n    params  module                                  arguments
0                -1  1      5280  models.common.Conv
[3, 48, 6, 2, 2]
1                -1  1     41664  models.common.Conv
[48, 96, 3, 2]
2                -1  2     65280  models.common.C3
[96, 96, 2]
3                -1  1    166272  models.common.Conv
[96, 192, 3, 2]
4                -1  4    444672  models.common.C3
[192, 192, 4]
5                -1  1    664320  models.common.Conv
[192, 384, 3, 2]
6                -1  6   2512896  models.common.C3
[384, 384, 6]
7                -1  1   2655744  models.common.Conv
[384, 768, 3, 2]
8                -1  2   4134912  models.common.C3
[768, 768, 2]
9                -1  1   1476864  models.common.SPPF
[768, 768, 5]
10                -1  1    295680  models.common.Conv
[768, 384, 1, 1]
11                -1  1         0  torch.nn.modules.upsampling.Upsample
[None, 2, 'nearest']
12           [-1, 6]  1         0  models.common.Concat
[1]
13                -1  2   1182720  models.common.C3
[768, 384, 2, False]
14                -1  1     74112  models.common.Conv
[384, 192, 1, 1]
15                -1  1         0  torch.nn.modules.upsampling.Upsample
[None, 2, 'nearest']
16           [-1, 4]  1         0  models.common.Concat
[1]
17                -1  2    296448  models.common.C3
[384, 192, 2, False]
18                -1  1    332160  models.common.Conv
[192, 192, 3, 2]
```

```
19          [-1, 14]  1         0  models.common.Concat
[1]
20                -1  2   1035264  models.common.C3
[384, 384, 2, False]
21                -1  1   1327872  models.common.Conv
[384, 384, 3, 2]
22          [-1, 10]  1         0  models.common.Concat
[1]
23                -1  2   4134912  models.common.C3
[768, 768, 2, False]
24      [17, 20, 23]  1     32328  models.yolo.Detect
[3, [[10, 13, 16, 30, 33, 23], [30, 61, 62, 45, 59, 119], [116, 90, 156,
198, 373, 326]], [192, 384, 768]]
Model Summary: 369 layers, 20879400 parameters, 20879400 gradients, 48.1
GFLOPs

Transferred 475/481 items from yolov5m.pt
Scaled weight_decay = 0.0005
optimizer: SGD with parameter groups 79 weight (no decay), 82 weight, 82
bias
train: Scanning
'C:\Users\moco_\Documents\Python\GP2\GP2_files\data\Full_DS\labels\train.cac
he' images and labels... 5000 found, 0 missing, 0 empty, 1 corrupt:
100%|██████████| 5000/5000 [00:00<?, ?it/s]
train: WARNING:
C:\Users\moco_\Documents\Python\GP2\GP2_files\data\Full_DS\images\train\2019
.jpg: ignoring corrupt image/label: non-normalized or out of bounds
coordinates [     2.8726      1.6186]
val: Scanning
'C:\Users\moco_\Documents\Python\GP2\GP2_files\data\Full_DS\labels\val.cache
' images and labels... 1000 found, 0 missing, 0 empty, 0 corrupt:
100%|██████████| 1000/1000 [00:00<?, ?it/s]
module 'signal' has no attribute 'SIGALRM'

AutoAnchor: 3.77 anchors/target, 1.000 Best Possible Recall (BPR). Current
anchors are a good fit to dataset
Image sizes 640 train, 640 val
Using 4 dataloader workers
Logging results to runs\train\exp11
Starting training for 24 epochs...

Epoch    gpu_mem       box       obj       cls    labels  img_size
0/23       3.01G   0.06769   0.02364   0.01963        21       640:
100%|██████████| 625/625 [37:59<00:00,  3.65s/it]
Class      Images     Labels          P          R     mAP@.5 mAP@.5:.95:
100%|██████████| 63/63 [01:39<00:00,  1.58s/it]
all        1000       1063      0.524      0.547      0.516      0.247

Epoch    gpu_mem       box       obj       cls    labels  img_size
1/23       2.95G   0.04944    0.0182  0.008471        19       640:
100%|██████████| 625/625 [37:33<00:00,  3.61s/it]
Class      Images     Labels          P          R     mAP@.5 mAP@.5:.95:
100%|██████████| 63/63 [01:39<00:00,  1.57s/it]
all        1000       1063      0.609      0.541      0.525      0.267

Epoch    gpu_mem       box       obj       cls    labels  img_size
2/23       2.95G   0.04724   0.01807  0.007316        19       640:
100%|██████████| 625/625 [37:28<00:00,  3.60s/it]
Class      Images     Labels          P          R     mAP@.5 mAP@.5:.95:
100%|██████████| 63/63 [01:39<00:00,  1.57s/it]
all        1000       1063      0.492      0.518      0.455      0.247

Epoch    gpu_mem       box       obj       cls    labels  img_size
```

```
3/23      2.95G   0.04558   0.01903  0.008253        12       640:
100%|██████████| 625/625 [37:28<00:00,  3.60s/it]
Class      Images      Labels          P          R      mAP@.5 mAP@.5:.95:
100%|██████████| 63/63 [01:39<00:00,  1.58s/it]
all        1000        1063       0.788      0.507       0.58      0.239

Epoch    gpu_mem        box        obj        cls    labels  img_size
4/23      2.95G   0.04375   0.01894  0.007477        16       640:
100%|██████████| 625/625 [37:28<00:00,  3.60s/it]
Class      Images      Labels          P          R      mAP@.5 mAP@.5:.95:
100%|██████████| 63/63 [01:39<00:00,  1.58s/it]
all        1000        1063       0.655       0.53      0.533      0.232

Epoch    gpu_mem        box        obj        cls    labels  img_size
5/23      2.95G   0.04042   0.01769  0.005912        18       640:
100%|██████████| 625/625 [37:29<00:00,  3.60s/it]
Class      Images      Labels          P          R      mAP@.5 mAP@.5:.95:
100%|██████████| 63/63 [01:39<00:00,  1.58s/it]
all        1000        1063        0.79      0.598      0.636      0.425

Epoch    gpu_mem        box        obj        cls    labels  img_size
6/23      2.95G   0.03842     0.017  0.005099        22       640:
100%|██████████| 625/625 [37:28<00:00,  3.60s/it]
Class      Images      Labels          P          R      mAP@.5 mAP@.5:.95:
100%|██████████| 63/63 [01:39<00:00,  1.57s/it]
all        1000        1063       0.788      0.635      0.661      0.404

Epoch    gpu_mem        box        obj        cls    labels  img_size
7/23      2.95G   0.03665    0.0164  0.004415        12       640:
100%|██████████| 625/625 [37:30<00:00,  3.60s/it]
Class      Images      Labels          P          R      mAP@.5 mAP@.5:.95:
100%|██████████| 63/63 [01:39<00:00,  1.58s/it]
all        1000        1063       0.863      0.581      0.636      0.425

Epoch    gpu_mem        box        obj        cls    labels  img_size
8/23      2.95G   0.03477   0.01583  0.003987        18       640:
100%|██████████| 625/625 [37:29<00:00,  3.60s/it]
Class      Images      Labels          P          R      mAP@.5 mAP@.5:.95:
100%|██████████| 63/63 [01:39<00:00,  1.58s/it]
all        1000        1063       0.892       0.65      0.727       0.44

Epoch    gpu_mem        box        obj        cls    labels  img_size
9/23      2.95G   0.03367   0.01534  0.003065        18       640:
100%|██████████| 625/625 [37:30<00:00,  3.60s/it]
Class      Images      Labels          P          R      mAP@.5 mAP@.5:.95:
100%|██████████| 63/63 [01:40<00:00,  1.59s/it]
all        1000        1063        0.89      0.672      0.715      0.491

Epoch    gpu_mem        box        obj        cls    labels  img_size
10/23      2.95G   0.03288   0.01499  0.003088        24       640:
100%|██████████| 625/625 [37:29<00:00,  3.60s/it]
Class      Images      Labels          P          R      mAP@.5 mAP@.5:.95:
100%|██████████| 63/63 [01:40<00:00,  1.59s/it]
all        1000        1063       0.923      0.646      0.708      0.485

Epoch    gpu_mem        box        obj        cls    labels  img_size
11/23      2.95G   0.03165   0.01447  0.002632        17       640:
100%|██████████| 625/625 [37:30<00:00,  3.60s/it]
Class      Images      Labels          P          R      mAP@.5 mAP@.5:.95:
100%|██████████| 63/63 [01:39<00:00,  1.58s/it]
all        1000        1063       0.908      0.656      0.704      0.501
```

```
Epoch   gpu_mem       box       obj       cls    labels  img_size
12/23     2.95G   0.03051    0.0143  0.002729        16       640:
100%|██████████| 625/625 [37:38<00:00,  3.61s/it]
Class     Images     Labels          P          R     mAP@.5 mAP@.5:.95:
100%|██████████| 63/63 [01:39<00:00,  1.58s/it]
all       1000       1063      0.932      0.659      0.712      0.496

Epoch   gpu_mem       box       obj       cls    labels  img_size
13/23     2.95G    0.0295   0.01363  0.002282        18       640:
100%|██████████| 625/625 [37:30<00:00,  3.60s/it]
Class     Images     Labels          P          R     mAP@.5 mAP@.5:.95:
100%|██████████| 63/63 [01:39<00:00,  1.58s/it]
all       1000       1063       0.93      0.639      0.716      0.525

Epoch   gpu_mem       box       obj       cls    labels  img_size
14/23     2.95G   0.02861   0.01345  0.002204        13       640:
100%|██████████| 625/625 [37:34<00:00,  3.61s/it]
Class     Images     Labels          P          R     mAP@.5 mAP@.5:.95:
100%|██████████| 63/63 [01:40<00:00,  1.59s/it]
all       1000       1063      0.936      0.676      0.735      0.541

Epoch   gpu_mem       box       obj       cls    labels  img_size
15/23     2.95G   0.02768   0.01316  0.001913        21       640:
100%|██████████| 625/625 [37:26<00:00,  3.59s/it]
Class     Images     Labels          P          R     mAP@.5 mAP@.5:.95:
100%|██████████| 63/63 [01:39<00:00,  1.58s/it]
all       1000       1063      0.948      0.712      0.758      0.574

Epoch   gpu_mem       box       obj       cls    labels  img_size
16/23     2.95G   0.02737   0.01304  0.002238        20       640:
100%|██████████| 625/625 [37:25<00:00,  3.59s/it]
Class     Images     Labels          P          R     mAP@.5 mAP@.5:.95:
100%|██████████| 63/63 [01:39<00:00,  1.58s/it]
all       1000       1063      0.922      0.703      0.765      0.557

Epoch   gpu_mem       box       obj       cls    labels  img_size
17/23     2.95G    0.0262   0.01274  0.001672        14       640:
100%|██████████| 625/625 [37:24<00:00,  3.59s/it]
Class     Images     Labels          P          R     mAP@.5 mAP@.5:.95:
100%|██████████| 63/63 [01:39<00:00,  1.58s/it]
all       1000       1063      0.938      0.713      0.765      0.536

Epoch   gpu_mem       box       obj       cls    labels  img_size
18/23     2.95G   0.02561   0.01238  0.001812        19       640:
100%|██████████| 625/625 [37:28<00:00,  3.60s/it]
Class     Images     Labels          P          R     mAP@.5 mAP@.5:.95:
100%|██████████| 63/63 [01:39<00:00,  1.58s/it]
all       1000       1063      0.918      0.727       0.79      0.583

Epoch   gpu_mem       box       obj       cls    labels  img_size
19/23     3.12G   0.02482   0.01224  0.001319        13       640:
100%|██████████| 625/625 [37:25<00:00,  3.59s/it]
Class     Images     Labels          P          R     mAP@.5 mAP@.5:.95:
100%|██████████| 63/63 [01:39<00:00,  1.57s/it]
all       1000       1063      0.938      0.729      0.783      0.602

Epoch   gpu_mem       box       obj       cls    labels  img_size
20/23     3.12G   0.02435   0.01187  0.001475        16       640:
100%|██████████| 625/625 [37:23<00:00,  3.59s/it]
Class     Images     Labels          P          R     mAP@.5 mAP@.5:.95:
100%|██████████| 63/63 [01:39<00:00,  1.57s/it]
all       1000       1063      0.943      0.722      0.771      0.593
```

```
Epoch   gpu_mem       box       obj       cls    labels  img_size
21/23     3.12G   0.02338    0.0115  0.001293        20       640:
100%|██████████| 625/625 [37:11<00:00,  3.57s/it]
Class     Images     Labels          P         R     mAP@.5 mAP@.5:.95:
100%|██████████| 63/63 [01:41<00:00,  1.60s/it]
all         1000       1063      0.953     0.716      0.787      0.592

Epoch   gpu_mem       box       obj       cls    labels  img_size
22/23     3.12G    0.0233    0.0115  0.001455        22       640:
100%|██████████| 625/625 [37:12<00:00,  3.57s/it]
Class     Images     Labels          P         R     mAP@.5 mAP@.5:.95:
100%|██████████| 63/63 [01:39<00:00,  1.58s/it]
all         1000       1063      0.943     0.728      0.786      0.595

Epoch   gpu_mem       box       obj       cls    labels  img_size
23/23     3.12G   0.02282   0.01134  0.001304        10       640:
100%|██████████| 625/625 [37:12<00:00,  3.57s/it]
Class     Images     Labels          P         R     mAP@.5 mAP@.5:.95:
100%|██████████| 63/63 [01:39<00:00,  1.59s/it]
all         1000       1063      0.945     0.729      0.782      0.587

24 epochs completed in 15.661 hours.
Optimizer stripped from runs\train\exp11\weights\last.pt, 42.2MB
Optimizer stripped from runs\train\exp11\weights\best.pt, 42.2MB

Validating runs\train\exp11\weights\best.pt...
Fusing layers...
Model Summary: 290 layers, 20861016 parameters, 0 gradients, 48.0 GFLOPs
Class     Images     Labels          P         R     mAP@.5 mAP@.5:.95:
100%|██████████| 63/63 [01:53<00:00,  1.80s/it]
all         1000       1063      0.941     0.728      0.783      0.602
knife         1000        683      0.952     0.261      0.403      0.225
gun         1000        366      0.897     0.923      0.951      0.675
rifle         1000         14      0.974         1      0.995      0.907
Results saved to runs\train\exp11

Process finished with exit code 0
```

Code 4-5: Notes of compiled code (YOLOv5m)

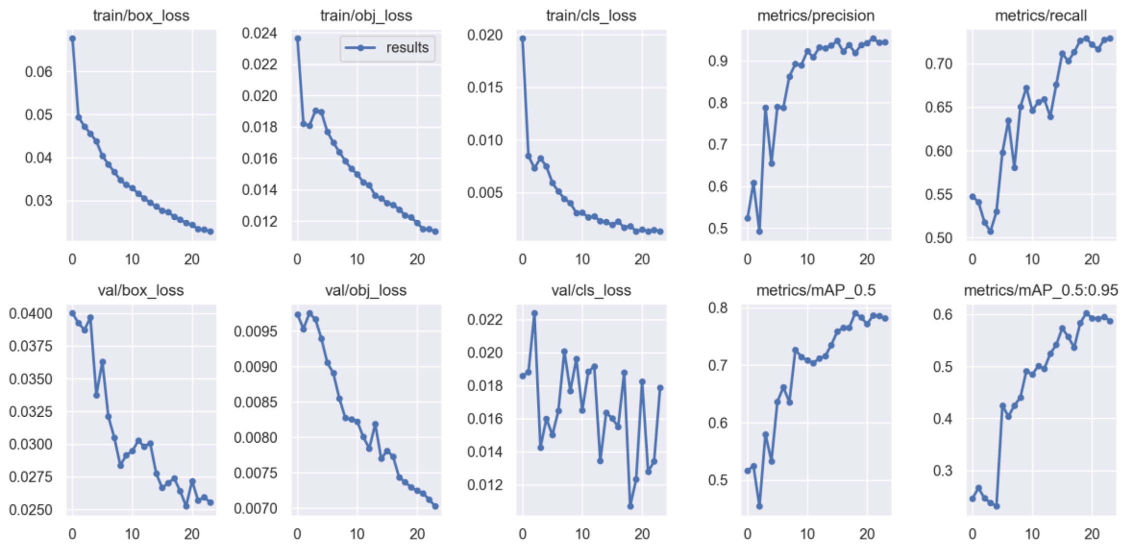


Figure 4-5: Result's Summary of trained model

After the training started and finished, as the training resulted in him succeeding in identifying all the types given to him, and he was ready with a high percentage to be tested in reality. The results of the training were as follows

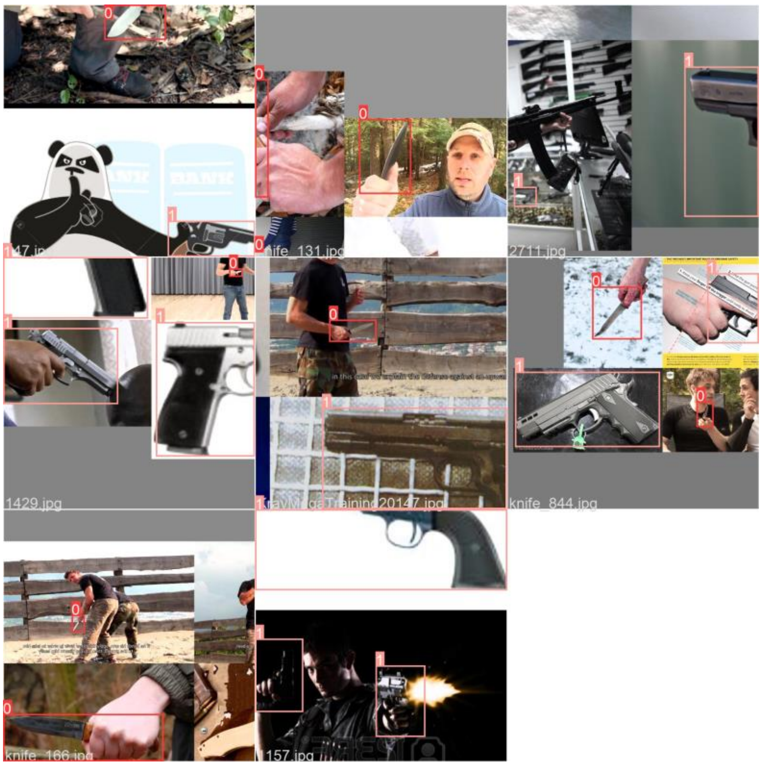


Figure 4-6: Trained batch of trained model

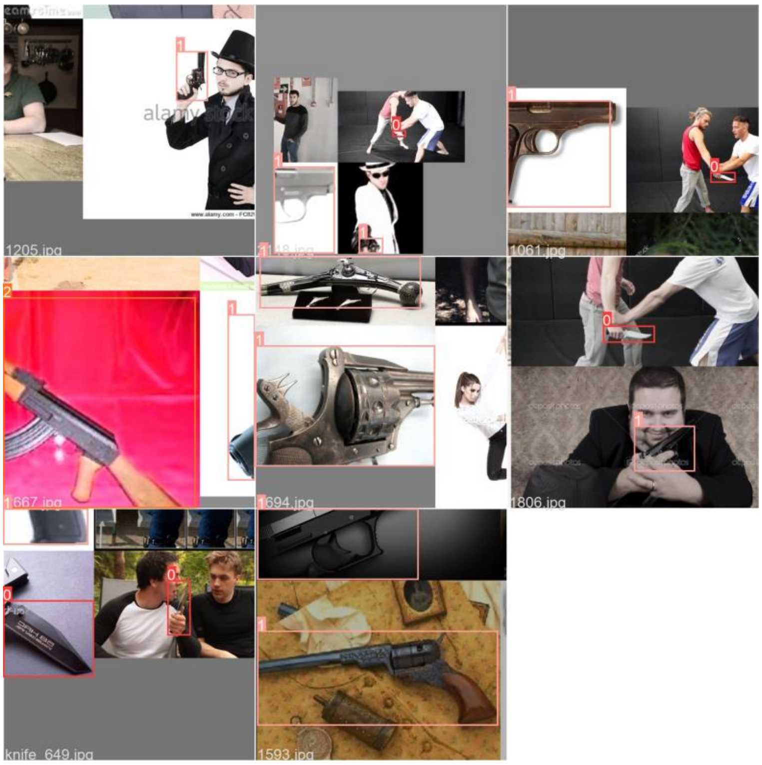


Figure 4-7: Trained batch of trained model

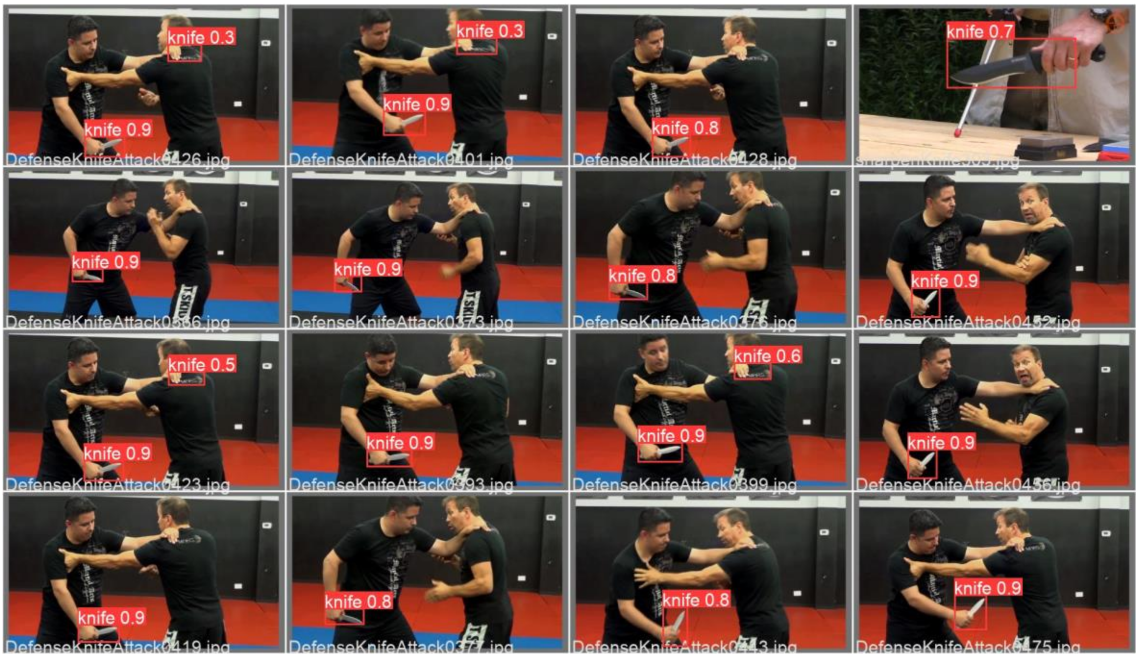


Figure 4-8: Validation batch of trained model (knifes)

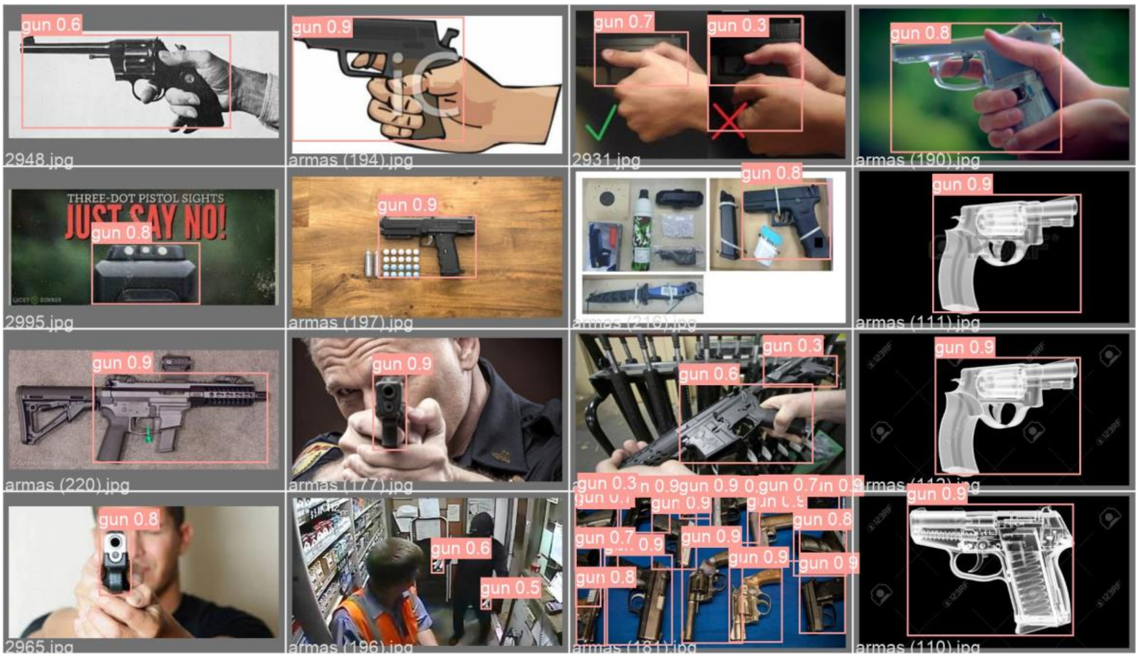


Figure 4-9: Validation batch of trained model (guns)

4.2.2 ٢ed train: YOLOv5x

We use the annotated datasets, that has been processed and explained how in the previous pages. the device specifications that used for the train is: (Intel(R) XEON 20 CORE 2.2GHZ, NVIDIA® Quadro RTX™ 5000 - 16GB, 128GB)

In next pages, the notes of the trained code with the figures of trained model

```
C:\python_oct19\python.exe
D:/Emad_Files/GP2_432/GP2_ver3_training/3_train_1.py
train: weights=yolov5l.pt, cfg=, data=weapons_dataset.yaml,
hyp=data\hyps\hyp.scratch.yaml, epochs=100, batch_size=16, imgsz=640,
rect=False, resume=False, nosave=False, noval=False, noautoanchor=False,
evolve=None, bucket=, cache=None, image_weights=False, device=,
multi_scale=False, single_cls=False, optimizer=SGD, sync_bn=False,
workers=8, project=runs\train, name=exp, exist_ok=False, quad=False,
linear_lr=False, label_smoothing=0.0, patience=100, freeze=[0],
save_period=-1, local_rank=-1, entity=None, upload_dataset=False,
bbox_interval=-1, artifact_alias=latest
github: skipping check (not a git repository), for updates see
https://github.com/ultralytics/yolov5
YOLOv5  2022-2-12 torch 1.9.1 CUDA:0 (Quadro RTX 6000, 24576MiB)
hyperparameters: lr0=0.01, lrf=0.1, momentum=0.937, weight_decay=0.0005,
warmup_epochs=3.0, warmup_momentum=0.8, warmup_bias_lr=0.1, box=0.05,
cls=0.5, cls_pw=1.0, obj=1.0, obj_pw=1.0, iou_t=0.2, anchor_t=4.0,
fl_gamma=0.0, hsv_h=0.015, hsv_s=0.7, hsv_v=0.4, degrees=0.0, translate=0.1,
scale=0.5, shear=0.0, perspective=0.0, flipud=0.0, fliplr=0.5, mosaic=1.0,
mixup=0.0, copy_paste=0.0
TensorBoard: Start with 'tensorboard --logdir runs\train', view at
http://localhost:6006/
Weights & Biases: run 'pip install wandb' to automatically track and
visualize YOLOv5  runs (RECOMMENDED)
Overriding model.yaml nc=80 with nc=2

                 from  n    params  module
arguments
  0                -1  1      7040  models.common.Conv
[3, 64, 6, 2, 2]
  1                -1  1     73984  models.common.Conv
[64, 128, 3, 2]
  2                -1  3    156928  models.common.C3
[128, 128, 3]
  3                -1  1    295424  models.common.Conv
[128, 256, 3, 2]
  4                -1  6   1118208  models.common.C3
[256, 256, 6]
  5                -1  1   1180672  models.common.Conv
[256, 512, 3, 2]
  6                -1  9   6433792  models.common.C3
[512, 512, 9]
  7                -1  1   4720640  models.common.Conv
[512, 1024, 3, 2]
  8                -1  3   9971712  models.common.C3
[1024, 1024, 3]
  9                -1  1   2624512  models.common.SPPF
[1024, 1024, 5]
 10                -1  1    525312  models.common.Conv
[1024, 512, 1, 1]
 11                -1  1         0  torch.nn.modules.upsampling.Upsample
[None, 2, 'nearest']
```

```
 12            [-1, 6]  1         0  models.common.Concat
[1]
 13                 -1  3   2757632  models.common.C3
[1024, 512, 3, False]
 14                 -1  1    131584  models.common.Conv
[512, 256, 1, 1]
 15                 -1  1         0  torch.nn.modules.upsampling.Upsample
[None, 2, 'nearest']
 16            [-1, 4]  1         0  models.common.Concat
[1]
 17                 -1  3    690688  models.common.C3
[512, 256, 3, False]
 18                 -1  1    590336  models.common.Conv
[256, 256, 3, 2]
 19           [-1, 14]  1         0  models.common.Concat
[1]
 20                 -1  3   2495488  models.common.C3
[512, 512, 3, False]
 21                 -1  1   2360320  models.common.Conv
[512, 512, 3, 2]
 22           [-1, 10]  1         0  models.common.Concat
[1]
 23                 -1  3   9971712  models.common.C3
[1024, 1024, 3, False]
 24       [17, 20, 23]  1     37695  models.yolo.Detect
[2, [[10, 13, 16, 30, 33, 23], [30, 61, 62, 45, 59, 119], [116, 90, 156,
198, 373, 326]], [256, 512, 1024]]
Model Summary: 468 layers, 46143679 parameters, 46143679 gradients, 107.9
GFLOPs
Transferred 607/613 items from yolov5l.pt
Scaled weight_decay = 0.0005
optimizer: SGD with parameter groups 101 weight (no decay), 104 weight, 104
bias
train: Scanning 'D:\Emad_Files\GP2_432\Datasets\DS5\labels\train' images and
labels...4500 found, 0 missing, 0 empty, 0 corrupt: 100%|██████████|
4500/4500 [00:05<00:00, 797.73it/s]
train: New cache created:
D:\Emad_Files\GP2_432\Datasets\DS5\labels\train.cache
val: Scanning 'D:\Emad_Files\GP2_432\Datasets\DS5\labels\val' images and
labels...578 found, 0 missing, 0 empty, 1 corrupt: 100%|██████████| 578/578
[00:03<00:00, 177.90it/s]
val: New cache created: D:\Emad_Files\GP2_432\Datasets\DS5\labels\val.cache
module 'signal' has no attribute 'SIGALRM'

AutoAnchor: 3.84 anchors/target, 1.000 Best Possible Recall (BPR). Current
anchors are a good fit to dataset
Image sizes 640 train, 640 val
Using 8 dataloader workers
Starting training for 100 epochs...

     Epoch   gpu_mem       box       obj       cls    labels  img_size
      0/99     9.23G    0.0688   0.02468   0.01551         9       640:
100%|██████████| 282/282 [01:42<00:00,  2.75it/s]
               Class     Images     Labels          P          R     mAP@.5
mAP@.5:.95: 100%|██████████| 19/19 [00:04<00:00,  4.39it/s]
                 all        577        623      0.599      0.655      0.579
0.243

     Epoch   gpu_mem       box       obj       cls    labels  img_size
      1/99     12.3G   0.04764   0.01705  0.004785        11       640:
100%|██████████| 282/282 [01:33<00:00,  3.01it/s]
               Class     Images     Labels          P          R     mAP@.5
mAP@.5:.95: 100%|██████████| 19/19 [00:04<00:00,  4.61it/s]
```

```
                 all        577        623      0.823      0.772       0.84
0.407

     Epoch   gpu_mem       box       obj       cls    labels  img_size
      2/99     12.3G   0.04334   0.01642  0.004584        13       640:
100%|██████████| 282/282 [01:31<00:00,  3.07it/s]
               Class     Images     Labels          P          R     mAP@.5
mAP@.5:.95: 100%|██████████| 19/19 [00:04<00:00,  4.53it/s]
                 all        577        623       0.72      0.569      0.643
0.328

     Epoch   gpu_mem       box       obj       cls    labels  img_size
      3/99     12.3G   0.04416   0.01775  0.005794        16       640:
100%|██████████| 282/282 [01:31<00:00,  3.10it/s]
               Class     Images     Labels          P          R     mAP@.5
mAP@.5:.95: 100%|██████████| 19/19 [00:04<00:00,  4.52it/s]
                 all        577        623       0.71      0.605      0.634
0.333

     Epoch   gpu_mem       box       obj       cls    labels  img_size
      4/99     12.3G   0.04253   0.01795  0.005215        10       640:
100%|██████████| 282/282 [01:31<00:00,  3.10it/s]
               Class     Images     Labels          P          R     mAP@.5
mAP@.5:.95: 100%|██████████| 19/19 [00:04<00:00,  4.61it/s]
                 all        577        623      0.771       0.71      0.772
0.427

     Epoch   gpu_mem       box       obj       cls    labels  img_size
      5/99     12.3G   0.03935   0.01667  0.004561        10       640:
100%|██████████| 282/282 [01:30<00:00,  3.11it/s]
               Class     Images     Labels          P          R     mAP@.5
mAP@.5:.95: 100%|██████████| 19/19 [00:04<00:00,  4.54it/s]
                 all        577        623       0.85      0.711      0.789
0.452

     Epoch   gpu_mem       box       obj       cls    labels  img_size
      6/99     12.3G   0.03752   0.01667  0.003822        12       640:
100%|██████████| 282/282 [01:30<00:00,  3.10it/s]
               Class     Images     Labels          P          R     mAP@.5
mAP@.5:.95: 100%|██████████| 19/19 [00:04<00:00,  4.61it/s]
                 all        577        623      0.883      0.723       0.82
0.465

     Epoch   gpu_mem       box       obj       cls    labels  img_size
      7/99     12.3G   0.03521   0.01581  0.003335         8       640:
100%|██████████| 282/282 [01:30<00:00,  3.10it/s]
               Class     Images     Labels          P          R     mAP@.5
mAP@.5:.95: 100%|██████████| 19/19 [00:04<00:00,  4.63it/s]
                 all        577        623      0.861      0.725      0.812
0.464

     Epoch   gpu_mem       box       obj       cls    labels  img_size
      8/99     12.3G    0.0341   0.01502  0.002784        10       640:
100%|██████████| 282/282 [01:30<00:00,  3.10it/s]
               Class     Images     Labels          P          R     mAP@.5
mAP@.5:.95: 100%|██████████| 19/19 [00:04<00:00,  4.62it/s]
                 all        577        623      0.827      0.736      0.812
0.469

     Epoch   gpu_mem       box       obj       cls    labels  img_size
      9/99     12.3G   0.03308   0.01484  0.002813        12       640:
100%|██████████| 282/282 [01:30<00:00,  3.10it/s]
```

```
                 Class      Images      Labels           P           R      mAP@.5
mAP@.5:.95: 100%|██████████| 19/19 [00:04<00:00,  4.59it/s]
                   all         577         623       0.864       0.748       0.839
0.491

     Epoch    gpu_mem       box       obj       cls    labels  img_size
     10/99      12.3G   0.03203   0.01465  0.002496        14       640:
100%|██████████| 282/282 [01:30<00:00,  3.11it/s]
                 Class      Images      Labels           P           R      mAP@.5
mAP@.5:.95: 100%|██████████| 19/19 [00:04<00:00,  4.61it/s]
                   all         577         623       0.854       0.761       0.834
0.481

     Epoch    gpu_mem       box       obj       cls    labels  img_size
     11/99      12.3G   0.03131   0.01458  0.002393         8       640:
100%|██████████| 282/282 [01:30<00:00,  3.10it/s]
                 Class      Images      Labels           P           R      mAP@.5
mAP@.5:.95: 100%|██████████| 19/19 [00:04<00:00,  4.59it/s]
                   all         577         623       0.857         0.8       0.862
0.53

     Epoch    gpu_mem       box       obj       cls    labels  img_size
     12/99      12.3G   0.03099   0.01431  0.002381        10       640:
100%|██████████| 282/282 [01:30<00:00,  3.10it/s]
                 Class      Images      Labels           P           R      mAP@.5
mAP@.5:.95: 100%|██████████| 19/19 [00:04<00:00,  4.57it/s]
                   all         577         623       0.899       0.804        0.88
0.544

     Epoch    gpu_mem       box       obj       cls    labels  img_size
     13/99      12.3G   0.03047     0.014  0.002113        17       640:
100%|██████████| 282/282 [01:30<00:00,  3.10it/s]
                 Class      Images      Labels           P           R      mAP@.5
mAP@.5:.95: 100%|██████████| 19/19 [00:04<00:00,  4.73it/s]
                   all         577         623       0.884       0.745       0.839
0.509

     Epoch    gpu_mem       box       obj       cls    labels  img_size
     14/99      12.3G    0.0302   0.01383  0.002273        12       640:
100%|██████████| 282/282 [01:30<00:00,  3.10it/s]
                 Class      Images      Labels           P           R      mAP@.5
mAP@.5:.95: 100%|██████████| 19/19 [00:04<00:00,  4.59it/s]
                   all         577         623       0.877       0.814       0.888
0.558

     Epoch    gpu_mem       box       obj       cls    labels  img_size
     15/99      12.3G   0.02941   0.01363  0.001997        20       640:
100%|██████████| 282/282 [01:30<00:00,  3.10it/s]
                 Class      Images      Labels           P           R      mAP@.5
mAP@.5:.95: 100%|██████████| 19/19 [00:04<00:00,  4.64it/s]
                   all         577         623       0.871       0.837       0.885
0.562

     Epoch    gpu_mem       box       obj       cls    labels  img_size
     16/99      12.3G   0.02913   0.01342  0.001995         8       640:
100%|██████████| 282/282 [01:30<00:00,  3.10it/s]
                 Class      Images      Labels           P           R      mAP@.5
mAP@.5:.95: 100%|██████████| 19/19 [00:04<00:00,  4.71it/s]
                   all         577         623       0.921       0.822       0.889
0.559

     Epoch    gpu_mem       box       obj       cls    labels  img_size
```

```
     17/99     12.3G   0.02818   0.01321  0.001603        11       640: 100%|██████████| 282/282 [01:30<00:00,  3.10it/s]
               Class     Images     Labels          P          R     mAP@.5 mAP@.5:.95: 100%|██████████| 19/19 [00:04<00:00,  4.61it/s]
                 all        577        623      0.904      0.809       0.88 0.567

     Epoch   gpu_mem       box       obj       cls    labels  img_size
     18/99     12.3G   0.02762    0.0128  0.001591        10       640: 100%|██████████| 282/282 [01:30<00:00,  3.11it/s]
               Class     Images     Labels          P          R     mAP@.5 mAP@.5:.95: 100%|██████████| 19/19 [00:04<00:00,  4.62it/s]
                 all        577        623      0.907      0.822      0.892 0.559

     Epoch   gpu_mem       box       obj       cls    labels  img_size
     19/99     12.3G   0.02803   0.01308  0.001642        12       640: 100%|██████████| 282/282 [01:30<00:00,  3.11it/s]
               Class     Images     Labels          P          R     mAP@.5 mAP@.5:.95: 100%|██████████| 19/19 [00:04<00:00,  4.66it/s]
                 all        577        623      0.918      0.844      0.888 0.569

     Epoch   gpu_mem       box       obj       cls    labels  img_size
     20/99     12.3G   0.02727   0.01288  0.001536         9       640: 100%|██████████| 282/282 [01:30<00:00,  3.10it/s]
               Class     Images     Labels          P          R     mAP@.5 mAP@.5:.95: 100%|██████████| 19/19 [00:04<00:00,  4.61it/s]
                 all        577        623      0.905      0.792      0.865 0.541

     Epoch   gpu_mem       box       obj       cls    labels  img_size
     21/99     12.3G   0.02676   0.01252   0.00133        10       640: 100%|██████████| 282/282 [01:30<00:00,  3.10it/s]
               Class     Images     Labels          P          R     mAP@.5 mAP@.5:.95: 100%|██████████| 19/19 [00:04<00:00,  4.57it/s]
                 all        577        623      0.918      0.803      0.884 0.564

     Epoch   gpu_mem       box       obj       cls    labels  img_size
     22/99     12.3G    0.0262   0.01231  0.001312        10       640: 100%|██████████| 282/282 [01:30<00:00,  3.11it/s]
               Class     Images     Labels          P          R     mAP@.5 mAP@.5:.95: 100%|██████████| 19/19 [00:04<00:00,  4.61it/s]
                 all        577        623      0.938      0.806      0.891 0.588

     Epoch   gpu_mem       box       obj       cls    labels  img_size
     23/99     12.3G   0.02593   0.01227  0.001312         6       640: 100%|██████████| 282/282 [01:30<00:00,  3.11it/s]
               Class     Images     Labels          P          R     mAP@.5 mAP@.5:.95: 100%|██████████| 19/19 [00:04<00:00,  4.61it/s]
                 all        577        623      0.915      0.845      0.913 0.585

     Epoch   gpu_mem       box       obj       cls    labels  img_size
     24/99     12.3G   0.02608   0.01226  0.001434         7       640: 100%|██████████| 282/282 [01:31<00:00,  3.10it/s]
               Class     Images     Labels          P          R     mAP@.5 mAP@.5:.95: 100%|██████████| 19/19 [00:04<00:00,  4.66it/s]
                 all        577        623      0.906      0.803      0.871 0.562
```

```
     Epoch   gpu_mem       box       obj       cls    labels  img_size
     25/99     12.3G   0.02521   0.01196   0.00136         8       640:
100%|██████████| 282/282 [01:30<00:00,  3.10it/s]
               Class     Images     Labels          P          R     mAP@.5
mAP@.5:.95: 100%|██████████| 19/19 [00:04<00:00,  4.71it/s]
                 all        577        623      0.924      0.825      0.896
0.571

     Epoch   gpu_mem       box       obj       cls    labels  img_size
     26/99     12.3G   0.02535   0.01209  0.001379        10       640:
100%|██████████| 282/282 [01:30<00:00,  3.11it/s]
               Class     Images     Labels          P          R     mAP@.5
mAP@.5:.95: 100%|██████████| 19/19 [00:04<00:00,  4.55it/s]
                 all        577        623      0.917      0.837        0.9
0.572

     Epoch   gpu_mem       box       obj       cls    labels  img_size
     27/99     12.3G   0.02515   0.01224  0.001184        11       640:
100%|██████████| 282/282 [01:31<00:00,  3.10it/s]
               Class     Images     Labels          P          R     mAP@.5
mAP@.5:.95: 100%|██████████| 19/19 [00:04<00:00,  4.50it/s]
                 all        577        623      0.912      0.828      0.885
0.564

     Epoch   gpu_mem       box       obj       cls    labels  img_size
     28/99     12.3G   0.02478   0.01193  0.001365         6       640:
100%|██████████| 282/282 [01:30<00:00,  3.10it/s]
               Class     Images     Labels          P          R     mAP@.5
mAP@.5:.95: 100%|██████████| 19/19 [00:04<00:00,  4.62it/s]
                 all        577        623      0.917      0.869      0.908
0.574

     Epoch   gpu_mem       box       obj       cls    labels  img_size
     29/99     12.3G    0.0243   0.01159   0.00131        11       640:
100%|██████████| 282/282 [01:30<00:00,  3.10it/s]
               Class     Images     Labels          P          R     mAP@.5
mAP@.5:.95: 100%|██████████| 19/19 [00:04<00:00,  4.61it/s]
                 all        577        623      0.921      0.815      0.896
0.568

     Epoch   gpu_mem       box       obj       cls    labels  img_size
     30/99     12.3G   0.02411   0.01159   0.00129         7       640:
100%|██████████| 282/282 [01:31<00:00,  3.09it/s]
               Class     Images     Labels          P          R     mAP@.5
mAP@.5:.95: 100%|██████████| 19/19 [00:04<00:00,  4.68it/s]
                 all        577        623      0.899      0.819      0.898
0.571

     Epoch   gpu_mem       box       obj       cls    labels  img_size
     31/99     12.3G   0.02433   0.01155  0.001135         5       640:
100%|██████████| 282/282 [01:30<00:00,  3.10it/s]
               Class     Images     Labels          P          R     mAP@.5
mAP@.5:.95: 100%|██████████| 19/19 [00:04<00:00,  4.62it/s]
                 all        577        623      0.944       0.84      0.898
0.582

     Epoch   gpu_mem       box       obj       cls    labels  img_size
     32/99     12.3G   0.02366   0.01136 0.0008728        13       640:
100%|██████████| 282/282 [01:30<00:00,  3.10it/s]
               Class     Images     Labels          P          R     mAP@.5
mAP@.5:.95: 100%|██████████| 19/19 [00:04<00:00,  4.68it/s]
```

```
                 all        577        623      0.941      0.843        0.9
0.575

     Epoch   gpu_mem       box       obj       cls    labels  img_size
     33/99     12.3G   0.02351   0.01126  0.001087         8       640:
100%|██████████| 282/282 [01:30<00:00,  3.11it/s]
               Class     Images     Labels          P          R     mAP@.5
mAP@.5:.95: 100%|██████████| 19/19 [00:04<00:00,  4.69it/s]
                 all        577        623      0.927       0.84      0.902
0.6

     Epoch   gpu_mem       box       obj       cls    labels  img_size
     34/99     12.3G   0.02365   0.01156  0.001422        10       640:
100%|██████████| 282/282 [01:31<00:00,  3.10it/s]
               Class     Images     Labels          P          R     mAP@.5
mAP@.5:.95: 100%|██████████| 19/19 [00:04<00:00,  4.68it/s]
                 all        577        623      0.911      0.817      0.886
0.581

     Epoch   gpu_mem       box       obj       cls    labels  img_size
     35/99     12.3G    0.0231    0.0114  0.001105         7       640:
100%|██████████| 282/282 [01:30<00:00,  3.11it/s]
               Class     Images     Labels          P          R     mAP@.5
mAP@.5:.95: 100%|██████████| 19/19 [00:04<00:00,  4.64it/s]
                 all        577        623      0.922      0.824      0.903
0.584

     Epoch   gpu_mem       box       obj       cls    labels  img_size
     36/99     12.3G   0.02292   0.01115  0.001088         9       640:
100%|██████████| 282/282 [01:31<00:00,  3.09it/s]
               Class     Images     Labels          P          R     mAP@.5
mAP@.5:.95: 100%|██████████| 19/19 [00:04<00:00,  4.66it/s]
                 all        577        623      0.926      0.859      0.914
0.607

     Epoch   gpu_mem       box       obj       cls    labels  img_size
     37/99     12.3G   0.02287   0.01091  0.001061        11       640:
100%|██████████| 282/282 [01:31<00:00,  3.08it/s]
               Class     Images     Labels          P          R     mAP@.5
mAP@.5:.95: 100%|██████████| 19/19 [00:04<00:00,  4.52it/s]
                 all        577        623      0.934      0.853      0.915
0.613

     Epoch   gpu_mem       box       obj       cls    labels  img_size
     38/99     12.3G   0.02206   0.01077  0.001051         8       640:
100%|██████████| 282/282 [01:31<00:00,  3.10it/s]
               Class     Images     Labels          P          R     mAP@.5
mAP@.5:.95: 100%|██████████| 19/19 [00:04<00:00,  4.60it/s]
                 all        577        623      0.956      0.833      0.907
0.589

     Epoch   gpu_mem       box       obj       cls    labels  img_size
     39/99     12.3G   0.02188   0.01058 0.0007992         6       640:
100%|██████████| 282/282 [01:31<00:00,  3.09it/s]
               Class     Images     Labels          P          R     mAP@.5
mAP@.5:.95: 100%|██████████| 19/19 [00:04<00:00,  4.60it/s]
                 all        577        623       0.94      0.861      0.905
0.603

     Epoch   gpu_mem       box       obj       cls    labels  img_size
     40/99     12.3G   0.02181   0.01067  0.000788        15       640:
100%|██████████| 282/282 [01:30<00:00,  3.12it/s]
```

```
                Class     Images     Labels          P          R     mAP@.5
mAP@.5:.95: 100%|██████████| 19/19 [00:04<00:00,  4.67it/s]
                  all        577        623      0.939      0.834      0.905
0.605

     Epoch   gpu_mem       box       obj       cls    labels  img_size
     41/99     12.3G   0.02162   0.01041 0.0008175        11       640:
100%|██████████| 282/282 [01:32<00:00,  3.05it/s]
                Class     Images     Labels          P          R     mAP@.5
mAP@.5:.95: 100%|██████████| 19/19 [00:04<00:00,  4.48it/s]
                  all        577        623      0.927      0.846      0.912
0.6

     Epoch   gpu_mem       box       obj       cls    labels  img_size
     42/99     12.3G   0.02117   0.01025 0.0008037        10       640:
100%|██████████| 282/282 [01:34<00:00,  2.99it/s]
                Class     Images     Labels          P          R     mAP@.5
mAP@.5:.95: 100%|██████████| 19/19 [00:04<00:00,  4.33it/s]
                  all        577        623      0.914      0.854      0.907
0.575

     Epoch   gpu_mem       box       obj       cls    labels  img_size
     43/99     12.3G   0.02107   0.01032 0.0008081        13       640:
100%|██████████| 282/282 [01:31<00:00,  3.09it/s]
                Class     Images     Labels          P          R     mAP@.5
mAP@.5:.95: 100%|██████████| 19/19 [00:04<00:00,  4.55it/s]
                  all        577        623      0.953      0.842      0.921
0.616

     Epoch   gpu_mem       box       obj       cls    labels  img_size
     44/99     12.3G   0.02088   0.01026 0.0007866         8       640:
100%|██████████| 282/282 [01:32<00:00,  3.05it/s]
                Class     Images     Labels          P          R     mAP@.5
mAP@.5:.95: 100%|██████████| 19/19 [00:04<00:00,  4.43it/s]
                  all        577        623      0.968      0.841      0.921
0.614

     Epoch   gpu_mem       box       obj       cls    labels  img_size
     45/99     12.3G   0.02085   0.01036 0.0007581        14       640:
100%|██████████| 282/282 [01:31<00:00,  3.07it/s]
                Class     Images     Labels          P          R     mAP@.5
mAP@.5:.95: 100%|██████████| 19/19 [00:04<00:00,  4.46it/s]
                  all        577        623      0.897       0.85      0.899
0.58

     Epoch   gpu_mem       box       obj       cls    labels  img_size
     46/99     12.3G   0.02044   0.01019 0.0007218         7       640:
100%|██████████| 282/282 [01:31<00:00,  3.09it/s]
                Class     Images     Labels          P          R     mAP@.5
mAP@.5:.95: 100%|██████████| 19/19 [00:04<00:00,  4.44it/s]
                  all        577        623      0.932      0.862      0.916
0.604

     Epoch   gpu_mem       box       obj       cls    labels  img_size
     47/99     12.3G   0.02009  0.009983 0.0007616        11       640:
100%|██████████| 282/282 [01:34<00:00,  2.99it/s]
                Class     Images     Labels          P          R     mAP@.5
mAP@.5:.95: 100%|██████████| 19/19 [00:04<00:00,  4.50it/s]
                  all        577        623       0.96      0.837      0.911
0.602

     Epoch   gpu_mem       box       obj       cls    labels  img_size
```

```
     48/99     12.3G   0.02018   0.01011 0.0009103        12       640:
100%|██████████| 282/282 [01:32<00:00,  3.06it/s]
               Class     Images     Labels          P          R     mAP@.5
mAP@.5:.95: 100%|██████████| 19/19 [00:04<00:00,  4.43it/s]
                 all        577        623      0.906      0.855      0.905
0.604

     Epoch   gpu_mem       box       obj       cls    labels  img_size
     49/99     12.3G   0.01944  0.009752 0.0006245        10       640:
100%|██████████| 282/282 [01:31<00:00,  3.07it/s]
               Class     Images     Labels          P          R     mAP@.5
mAP@.5:.95: 100%|██████████| 19/19 [00:04<00:00,  4.49it/s]
                 all        577        623      0.952      0.869      0.916
0.613

     Epoch   gpu_mem       box       obj       cls    labels  img_size
     50/99     12.3G   0.01955  0.009851 0.0006029        14       640:
100%|██████████| 282/282 [01:31<00:00,  3.07it/s]
               Class     Images     Labels          P          R     mAP@.5
mAP@.5:.95: 100%|██████████| 19/19 [00:04<00:00,  4.50it/s]
                 all        577        623       0.95      0.853      0.916
0.601

     Epoch   gpu_mem       box       obj       cls    labels  img_size
     51/99     12.3G   0.01968  0.009641 0.0007683        14       640:
100%|██████████| 282/282 [01:32<00:00,  3.06it/s]
               Class     Images     Labels          P          R     mAP@.5
mAP@.5:.95: 100%|██████████| 19/19 [00:04<00:00,  4.36it/s]
                 all        577        623      0.957      0.848      0.913
0.612

     Epoch   gpu_mem       box       obj       cls    labels  img_size
     52/99     12.3G   0.01899  0.009545 0.0005605        11       640:
100%|██████████| 282/282 [01:31<00:00,  3.07it/s]
               Class     Images     Labels          P          R     mAP@.5
mAP@.5:.95: 100%|██████████| 19/19 [00:04<00:00,  4.64it/s]
                 all        577        623      0.942      0.866      0.921
0.614

     Epoch   gpu_mem       box       obj       cls    labels  img_size
     53/99     12.3G   0.01899  0.009738 0.0007182        12       640:
100%|██████████| 282/282 [01:31<00:00,  3.09it/s]
               Class     Images     Labels          P          R     mAP@.5
mAP@.5:.95: 100%|██████████| 19/19 [00:04<00:00,  4.55it/s]
                 all        577        623      0.937      0.861       0.91
0.613

     Epoch   gpu_mem       box       obj       cls    labels  img_size
     54/99     12.3G   0.01891  0.009555 0.0006594        10       640:
100%|██████████| 282/282 [01:31<00:00,  3.10it/s]
               Class     Images     Labels          P          R     mAP@.5
mAP@.5:.95: 100%|██████████| 19/19 [00:04<00:00,  4.58it/s]
                 all        577        623      0.926      0.862       0.91
0.604

     Epoch   gpu_mem       box       obj       cls    labels  img_size
     55/99     12.3G   0.01851  0.009257 0.0006263         7       640:
100%|██████████| 282/282 [01:31<00:00,  3.09it/s]
               Class     Images     Labels          P          R     mAP@.5
mAP@.5:.95: 100%|██████████| 19/19 [00:04<00:00,  4.58it/s]
                 all        577        623      0.935      0.839      0.905
0.595
```

```
     Epoch   gpu_mem       box       obj       cls    labels  img_size
     56/99     12.3G   0.01846  0.009274 0.0005865         9       640:
100%|██████████| 282/282 [01:31<00:00,  3.09it/s]
               Class     Images     Labels          P          R     mAP@.5
mAP@.5:.95: 100%|██████████| 19/19 [00:04<00:00,  4.75it/s]
                 all        577        623      0.934      0.871      0.915
0.62

     Epoch   gpu_mem       box       obj       cls    labels  img_size
     57/99     12.3G   0.01842  0.009304 0.0006118        13       640:
100%|██████████| 282/282 [01:30<00:00,  3.11it/s]
               Class     Images     Labels          P          R     mAP@.5
mAP@.5:.95: 100%|██████████| 19/19 [00:04<00:00,  4.49it/s]
                 all        577        623      0.961      0.854       0.92
0.612

     Epoch   gpu_mem       box       obj       cls    labels  img_size
     58/99     12.3G   0.01811  0.009052 0.0004968        16       640:
100%|██████████| 282/282 [01:31<00:00,  3.09it/s]
               Class     Images     Labels          P          R     mAP@.5
mAP@.5:.95: 100%|██████████| 19/19 [00:04<00:00,  4.45it/s]
                 all        577        623      0.956      0.858      0.912
0.619

     Epoch   gpu_mem       box       obj       cls    labels  img_size
     59/99     12.3G   0.01806  0.009125 0.0006824         7       640:
100%|██████████| 282/282 [01:31<00:00,  3.10it/s]
               Class     Images     Labels          P          R     mAP@.5
mAP@.5:.95: 100%|██████████| 19/19 [00:04<00:00,  4.63it/s]
                 all        577        623       0.94      0.872      0.913
0.607

     Epoch   gpu_mem       box       obj       cls    labels  img_size
     60/99     12.3G   0.01738  0.008788 0.0005301         6       640:
100%|██████████| 282/282 [01:31<00:00,  3.09it/s]
               Class     Images     Labels          P          R     mAP@.5
mAP@.5:.95: 100%|██████████| 19/19 [00:04<00:00,  4.71it/s]
                 all        577        623      0.937      0.868      0.917
0.616

     Epoch   gpu_mem       box       obj       cls    labels  img_size
     61/99     12.3G   0.01717  0.008825 0.0004653        10       640:
100%|██████████| 282/282 [01:31<00:00,  3.09it/s]
               Class     Images     Labels          P          R     mAP@.5
mAP@.5:.95: 100%|██████████| 19/19 [00:04<00:00,  4.59it/s]
                 all        577        623      0.958      0.851      0.912
0.611

     Epoch   gpu_mem       box       obj       cls    labels  img_size
     62/99     12.3G   0.01711  0.008731 0.0004239        11       640:
100%|██████████| 282/282 [01:31<00:00,  3.09it/s]
               Class     Images     Labels          P          R     mAP@.5
mAP@.5:.95: 100%|██████████| 19/19 [00:04<00:00,  4.59it/s]
                 all        577        623      0.953      0.848      0.914
0.596

     Epoch   gpu_mem       box       obj       cls    labels  img_size
     63/99     12.3G   0.01709  0.008501 0.0004965        10       640:
100%|██████████| 282/282 [01:31<00:00,  3.09it/s]
               Class     Images     Labels          P          R     mAP@.5
mAP@.5:.95: 100%|██████████| 19/19 [00:04<00:00,  4.65it/s]
```

```
                 all        577        623      0.953      0.856      0.915
0.62

     Epoch   gpu_mem       box       obj       cls    labels  img_size
     64/99     12.3G   0.01669  0.008641 0.0003954         9       640:
100%|██████████| 282/282 [01:31<00:00,  3.09it/s]
               Class     Images     Labels          P          R     mAP@.5
mAP@.5:.95: 100%|██████████| 19/19 [00:04<00:00,  4.46it/s]
                 all        577        623      0.946      0.865      0.918
0.613

     Epoch   gpu_mem       box       obj       cls    labels  img_size
     65/99     12.3G   0.01636  0.008589 0.0005106        12       640:
100%|██████████| 282/282 [01:30<00:00,  3.10it/s]
               Class     Images     Labels          P          R     mAP@.5
mAP@.5:.95: 100%|██████████| 19/19 [00:04<00:00,  4.62it/s]
                 all        577        623      0.937      0.861      0.915
0.616

     Epoch   gpu_mem       box       obj       cls    labels  img_size
     66/99     12.3G   0.01683  0.008611 0.0005431         7       640:
100%|██████████| 282/282 [01:31<00:00,  3.07it/s]
               Class     Images     Labels          P          R     mAP@.5
mAP@.5:.95: 100%|██████████| 19/19 [00:03<00:00,  4.76it/s]
                 all        577        623      0.966      0.847      0.908
0.61

     Epoch   gpu_mem       box       obj       cls    labels  img_size
     67/99     12.3G   0.01623  0.008529 0.0004898        14       640:
100%|██████████| 282/282 [01:31<00:00,  3.10it/s]
               Class     Images     Labels          P          R     mAP@.5
mAP@.5:.95: 100%|██████████| 19/19 [00:04<00:00,  4.69it/s]
                 all        577        623      0.939       0.86      0.911
0.612

     Epoch   gpu_mem       box       obj       cls    labels  img_size
     68/99     12.3G   0.01617  0.008238 0.0004229         8       640:
100%|██████████| 282/282 [01:31<00:00,  3.09it/s]
               Class     Images     Labels          P          R     mAP@.5
mAP@.5:.95: 100%|██████████| 19/19 [00:04<00:00,  4.56it/s]
                 all        577        623      0.946      0.864      0.921
0.622

     Epoch   gpu_mem       box       obj       cls    labels  img_size
     69/99     12.3G    0.0159  0.008303   0.00042         9       640:
100%|██████████| 282/282 [01:30<00:00,  3.10it/s]
               Class     Images     Labels          P          R     mAP@.5
mAP@.5:.95: 100%|██████████| 19/19 [00:04<00:00,  4.65it/s]
                 all        577        623      0.941      0.852      0.913
0.62

     Epoch   gpu_mem       box       obj       cls    labels  img_size
     70/99     12.3G   0.01583  0.008342 0.0003282        11       640:
100%|██████████| 282/282 [01:31<00:00,  3.08it/s]
               Class     Images     Labels          P          R     mAP@.5
mAP@.5:.95: 100%|██████████| 19/19 [00:04<00:00,  4.65it/s]
                 all        577        623       0.95      0.852      0.917
0.614

     Epoch   gpu_mem       box       obj       cls    labels  img_size
     71/99     12.3G   0.01557  0.008138 0.0004147         9       640:
100%|██████████| 282/282 [01:30<00:00,  3.10it/s]
```

```
               Class      Images      Labels           P           R      mAP@.5
mAP@.5:.95: 100%|██████████| 19/19 [00:04<00:00,  4.64it/s]
                 all         577         623       0.945       0.856       0.913
0.614

     Epoch    gpu_mem        box        obj        cls    labels  img_size
     72/99      12.3G     0.0153   0.008175  0.0004209        11       640:
100%|██████████| 282/282 [01:31<00:00,  3.09it/s]
               Class      Images      Labels           P           R      mAP@.5
mAP@.5:.95: 100%|██████████| 19/19 [00:03<00:00,  4.77it/s]
                 all         577         623       0.929       0.865       0.914
0.614

     Epoch    gpu_mem        box        obj        cls    labels  img_size
     73/99      12.3G    0.01533   0.008069   0.000472        10       640:
100%|██████████| 282/282 [01:31<00:00,  3.08it/s]
               Class      Images      Labels           P           R      mAP@.5
mAP@.5:.95: 100%|██████████| 19/19 [00:04<00:00,  4.53it/s]
                 all         577         623       0.957       0.844        0.91
0.622

     Epoch    gpu_mem        box        obj        cls    labels  img_size
     74/99      12.3G    0.01531    0.00838  0.0004153        12       640:
100%|██████████| 282/282 [01:30<00:00,  3.10it/s]
               Class      Images      Labels           P           R      mAP@.5
mAP@.5:.95: 100%|██████████| 19/19 [00:04<00:00,  4.62it/s]
                 all         577         623       0.951       0.861       0.915
0.619

     Epoch    gpu_mem        box        obj        cls    labels  img_size
     75/99      12.3G    0.01495   0.007898  0.0004281        15       640:
100%|██████████| 282/282 [01:31<00:00,  3.09it/s]
               Class      Images      Labels           P           R      mAP@.5
mAP@.5:.95: 100%|██████████| 19/19 [00:04<00:00,  4.72it/s]
                 all         577         623        0.95       0.852       0.911
0.613

     Epoch    gpu_mem        box        obj        cls    labels  img_size
     76/99      12.3G    0.01534   0.007953  0.0003484        12       640:
100%|██████████| 282/282 [01:31<00:00,  3.10it/s]
               Class      Images      Labels           P           R      mAP@.5
mAP@.5:.95: 100%|██████████| 19/19 [00:04<00:00,  4.47it/s]
                 all         577         623       0.932       0.866       0.918
0.614

     Epoch    gpu_mem        box        obj        cls    labels  img_size
     77/99      12.3G    0.01475   0.007928  0.0003096         6       640:
100%|██████████| 282/282 [01:31<00:00,  3.08it/s]
               Class      Images      Labels           P           R      mAP@.5
mAP@.5:.95: 100%|██████████| 19/19 [00:04<00:00,  4.65it/s]
                 all         577         623       0.931       0.844       0.909
0.607

     Epoch    gpu_mem        box        obj        cls    labels  img_size
     78/99      12.3G    0.01464   0.007842  0.0003698         8       640:
100%|██████████| 282/282 [01:31<00:00,  3.09it/s]
               Class      Images      Labels           P           R      mAP@.5
mAP@.5:.95: 100%|██████████| 19/19 [00:04<00:00,  4.61it/s]
                 all         577         623       0.915        0.84         0.9
0.605

     Epoch    gpu_mem        box        obj        cls    labels  img_size
```

```
      79/99     12.3G    0.0146  0.007858 0.0003585        10       640:
100%|| 282/282 [01:31<00:00,  3.09it/s]
               Class     Images     Labels          P          R     mAP@.5
mAP@.5:.95: 100%|| 19/19 [00:04<00:00,  4.61it/s]
                 all        577        623      0.947      0.834       0.91
0.611

     Epoch   gpu_mem       box       obj       cls    labels  img_size
      80/99     12.3G   0.01446  0.007816 0.0003273         5       640:
100%|| 282/282 [01:30<00:00,  3.10it/s]
               Class     Images     Labels          P          R     mAP@.5
mAP@.5:.95: 100%|| 19/19 [00:04<00:00,  4.69it/s]
                 all        577        623      0.949      0.827      0.909
0.615

     Epoch   gpu_mem       box       obj       cls    labels  img_size
      81/99     12.3G   0.01406  0.007673 0.0002979        15       640:
100%|| 282/282 [01:30<00:00,  3.10it/s]
               Class     Images     Labels          P          R     mAP@.5
mAP@.5:.95: 100%|| 19/19 [00:04<00:00,  4.70it/s]
                 all        577        623      0.919      0.866      0.916
0.614

     Epoch   gpu_mem       box       obj       cls    labels  img_size
      82/99     12.3G   0.01416  0.007667 0.0003863         9       640:
100%|| 282/282 [01:31<00:00,  3.08it/s]
               Class     Images     Labels          P          R     mAP@.5
mAP@.5:.95: 100%|| 19/19 [00:04<00:00,  4.71it/s]
                 all        577        623      0.952      0.843      0.909
0.617

     Epoch   gpu_mem       box       obj       cls    labels  img_size
      83/99     12.3G   0.01412  0.007679 0.0003171        18       640:
100%|| 282/282 [01:31<00:00,  3.10it/s]
               Class     Images     Labels          P          R     mAP@.5
mAP@.5:.95: 100%|| 19/19 [00:03<00:00,  4.83it/s]
                 all        577        623      0.951      0.851      0.917
0.621

     Epoch   gpu_mem       box       obj       cls    labels  img_size
      84/99     12.3G   0.01394  0.007525 0.0002971         9       640:
100%|| 282/282 [01:30<00:00,  3.11it/s]
               Class     Images     Labels          P          R     mAP@.5
mAP@.5:.95: 100%|| 19/19 [00:03<00:00,  4.77it/s]
                 all        577        623      0.944      0.863       0.91
0.619

     Epoch   gpu_mem       box       obj       cls    labels  img_size
      85/99     12.3G   0.01388   0.00748 0.0003207         9       640:
100%|| 282/282 [01:31<00:00,  3.10it/s]
               Class     Images     Labels          P          R     mAP@.5
mAP@.5:.95: 100%|| 19/19 [00:03<00:00,  4.78it/s]
                 all        577        623      0.967      0.849      0.919
0.631

     Epoch   gpu_mem       box       obj       cls    labels  img_size
      86/99     12.3G   0.01375  0.007622 0.0003093        14       640:
100%|| 282/282 [01:32<00:00,  3.06it/s]
               Class     Images     Labels          P          R     mAP@.5
mAP@.5:.95: 100%|| 19/19 [00:04<00:00,  4.66it/s]
                 all        577        623      0.933      0.875      0.922
0.636
```

```
     Epoch   gpu_mem       box        obj        cls    labels  img_size
     87/99     12.3G   0.01346   0.007386   0.000319        11       640: 
100%|██████████| 282/282 [01:31<00:00,  3.09it/s]
               Class     Images     Labels          P          R     mAP@.5 
mAP@.5:.95: 100%|██████████| 19/19 [00:03<00:00,  4.77it/s]
                 all        577        623      0.928      0.858      0.914
0.624

     Epoch   gpu_mem       box        obj        cls    labels  img_size
     88/99     12.3G   0.01338   0.007427 0.0002816        10       640: 
100%|██████████| 282/282 [01:31<00:00,  3.09it/s]
               Class     Images     Labels          P          R     mAP@.5 
mAP@.5:.95: 100%|██████████| 19/19 [00:03<00:00,  4.77it/s]
                 all        577        623      0.928      0.849      0.907
0.62

     Epoch   gpu_mem       box        obj        cls    labels  img_size
     89/99     12.3G   0.01327   0.007156 0.0003485        13       640: 
100%|██████████| 282/282 [01:30<00:00,  3.10it/s]
               Class     Images     Labels          P          R     mAP@.5 
mAP@.5:.95: 100%|██████████| 19/19 [00:04<00:00,  4.69it/s]
                 all        577        623      0.945      0.839      0.907
0.623

     Epoch   gpu_mem       box        obj        cls    labels  img_size
     90/99     12.3G   0.01315   0.007214 0.0003156         9       640: 
100%|██████████| 282/282 [01:31<00:00,  3.10it/s]
               Class     Images     Labels          P          R     mAP@.5 
mAP@.5:.95: 100%|██████████| 19/19 [00:04<00:00,  4.68it/s]
                 all        577        623      0.919       0.87      0.915
0.623

     Epoch   gpu_mem       box        obj        cls    labels  img_size
     91/99     12.3G   0.01321   0.007333 0.0003861         7       640: 
100%|██████████| 282/282 [01:30<00:00,  3.11it/s]
               Class     Images     Labels          P          R     mAP@.5 
mAP@.5:.95: 100%|██████████| 19/19 [00:04<00:00,  4.70it/s]
                 all        577        623      0.926      0.854      0.911
0.628

     Epoch   gpu_mem       box        obj        cls    labels  img_size
     92/99     12.3G   0.01328   0.007374 0.0002683        10       640: 
100%|██████████| 282/282 [01:30<00:00,  3.10it/s]
               Class     Images     Labels          P          R     mAP@.5 
mAP@.5:.95: 100%|██████████| 19/19 [00:03<00:00,  4.76it/s]
                 all        577        623      0.963       0.82      0.906
0.625

     Epoch   gpu_mem       box        obj        cls    labels  img_size
     93/99     12.3G   0.01316   0.007385 0.0003275        10       640: 
100%|██████████| 282/282 [01:30<00:00,  3.10it/s]
               Class     Images     Labels          P          R     mAP@.5 
mAP@.5:.95: 100%|██████████| 19/19 [00:04<00:00,  4.65it/s]
                 all        577        623      0.912      0.861      0.907
0.624

     Epoch   gpu_mem       box        obj        cls    labels  img_size
     94/99     12.3G   0.01305     0.0072   0.000283        14       640: 
100%|██████████| 282/282 [01:30<00:00,  3.10it/s]
               Class     Images     Labels          P          R     mAP@.5 
mAP@.5:.95: 100%|██████████| 19/19 [00:04<00:00,  4.68it/s]
```

```
                 all        577        623      0.935      0.855      0.907
0.623

     Epoch   gpu_mem       box       obj       cls    labels  img_size
     95/99     12.3G   0.01284  0.007141 0.0002178         4       640:
100%|██████████| 282/282 [01:30<00:00,  3.11it/s]
               Class     Images     Labels          P          R     mAP@.5
mAP@.5:.95: 100%|██████████| 19/19 [00:04<00:00,  4.70it/s]
                 all        577        623      0.946      0.836      0.907
0.626

     Epoch   gpu_mem       box       obj       cls    labels  img_size
     96/99     12.3G   0.01288  0.007049  0.000215        10       640:
100%|██████████| 282/282 [01:30<00:00,  3.11it/s]
               Class     Images     Labels          P          R     mAP@.5
mAP@.5:.95: 100%|██████████| 19/19 [00:04<00:00,  4.73it/s]
                 all        577        623      0.942      0.841      0.912
0.626

     Epoch   gpu_mem       box       obj       cls    labels  img_size
     97/99     12.3G   0.01272  0.007106 0.0002479        11       640:
100%|██████████| 282/282 [01:30<00:00,  3.11it/s]
               Class     Images     Labels          P          R     mAP@.5
mAP@.5:.95: 100%|██████████| 19/19 [00:03<00:00,  4.78it/s]
                 all        577        623       0.96      0.825      0.908
0.63

     Epoch   gpu_mem       box       obj       cls    labels  img_size
     98/99     12.3G   0.01285  0.007112 0.0002734         9       640:
100%|██████████| 282/282 [01:31<00:00,  3.10it/s]
               Class     Images     Labels          P          R     mAP@.5
mAP@.5:.95: 100%|██████████| 19/19 [00:03<00:00,  4.83it/s]
                 all        577        623      0.952      0.842      0.915
0.629

     Epoch   gpu_mem       box       obj       cls    labels  img_size
     99/99     12.3G   0.01276  0.007032 0.0002654        11       640:
100%|██████████| 282/282 [01:30<00:00,  3.12it/s]
               Class     Images     Labels          P          R     mAP@.5
mAP@.5:.95: 100%|██████████| 19/19 [00:03<00:00,  4.78it/s]
                 all        577        623      0.952      0.844      0.916
0.628

100 epochs completed in 2.678 hours.
Optimizer stripped from runs\train\exp18\weights\last.pt, 92.8MB
Optimizer stripped from runs\train\exp18\weights\best.pt, 92.8MB

Validating runs\train\exp18\weights\best.pt...
Model Summary: 367 layers, 46113663 parameters, 0 gradients, 107.8 GFLOPs
               Class     Images     Labels          P          R     mAP@.5
mAP@.5:.95: 100%|██████████| 19/19 [00:04<00:00,  3.91it/s]
                 all        577        623      0.933      0.875      0.922
0.636
               knife        577        282      0.965      0.915      0.963
0.626
              pistol        577        341      0.902      0.836      0.881
0.645

Process finished with exit code 0
```

Code 4-6: Notes of compiled code (YOLOv5x)

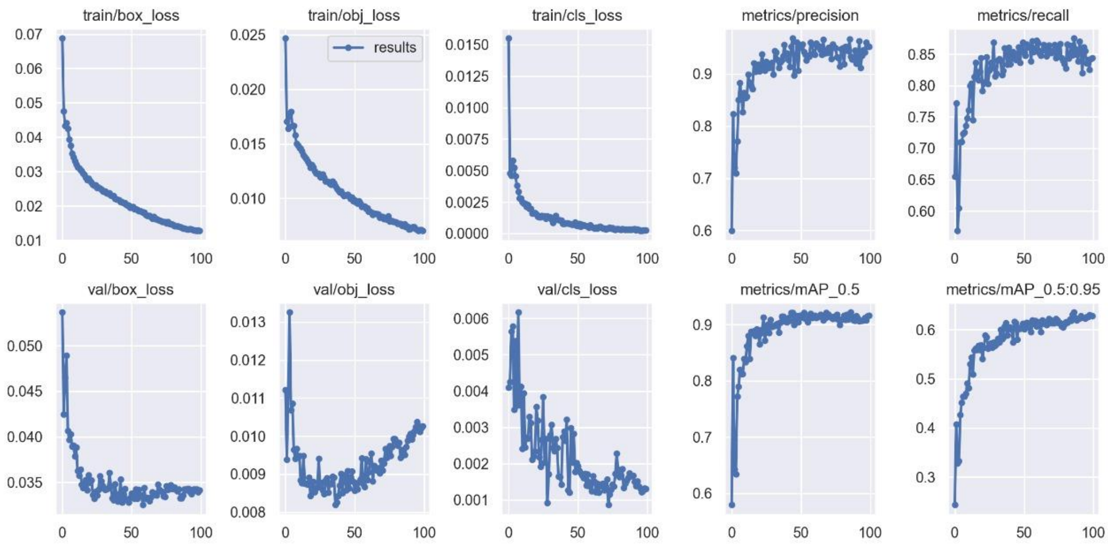


Figure 4-10: Result's Summary of trained model

4.3 TESTING & INFERENCING

Based on the above in the work, there are some things that illustrate the accuracy and strength of the trained model.

4.3.1 Model Inferencing

After training the model, we make sure that the outputs and training the model is ready to be presented. The following code is to Inference the model on the data on test path.

Inference.py

```
import os
os.system("python detect.py --weights best.pt --img 640 --conf 0.5 --source data\\test ")
```

Code 4-7: Inference main code, according to trained model

This figure is a sample of 150 picture of weapons, that tested by the trained model

Figure 4-11: Sample of tested pictures within trained model

4.3.2 Testing by camera

The next step is to connect the trained model to a webcam, to make sure it detects in perfect way. we start putting the model in test mode and start testing it on real data for that. The following code is to test the model on the device's camera and start working.

test.py

```
import os

os.system("python detect.py  --source 0 --conf 0.6 --
weights best.pt ")
```

Code 4-8: Main code of camera testing of trained model

The code below is background code is to activate the camera

test _threads.py

```
import threading
import torch
import cv2
from threading import Thread
from datetime import datetime
import os
from IPython.display import Image
import torchvision.models as models
# ====================================================
def cam():
    src = 'rtsp://admin:GCKWAH@192.168.8.109/'
    camera = cv2.VideoCapture(src)
    i = 0
    while True:
        return_value, image = camera.read()
        cv2.imshow('image', image)
        i += 1
        if i == 20:
            th = Thread(target=detect_img,
args=(image,))
            th.daemon = True
```

```
            th.start()
            i = 0
        if cv2.waitKey(1) & 0xFF == ord('q'):
            break
    camera.release()
    cv2.destroyAllWindows()
# ========================================================
def detect_img(img):
#lock
    global model
    i = datetime.now()
    i = str(i).replace(' ', '_')
    i = i.replace('-', '_')
    i = i.replace(':', '__')

    cv2.imwrite('saved_imgs/' + i + '.jpg', img)
    img_scr = 'saved_imgs/' + i + '.jpg'
    results = model(img_scr)
    results.print()
# ========================================================
if __name__ == '__main__':
model =
torch.hub.load(r'C:\Users\moco_\Documents\Python\GP2\GP
2_files', 'custom', path=r'best.pt', source='local')
cam()
from models.yolo import Model
```

Code 4-9: Background code of active camera to testing

In next figure, the handle was tested on a gun of different types, and it was a high success, as more than one pistol was discovered at a time until the image was moved, gun was detected at a high rate. The second type, which is knives, was tested and the results were as described above, they were also detected according to clarity in different proportions, where the higher the number, this means that he is sure that the object is correct.

Figure 4-12: Detect guns and knifes within trained model

4.4 CONNECT TO A WEB SYSTEM

Before connecting our trained model, we thought about making the system using a web or desktop? We found a better website because it

Web applications What distinguishes these programs is a process, renting hosting or renting a server to put the programming code on it, and it can be accessed from anywhere in the world, if Internet service is available. for access to programming.

4.4.1 Web applications features:

1. The ability to access it from anywhere in the world.
2. Maintaining databases and easy access to them
3. Dissemination of information in a quick way through the Internet.
4. Easy to modify the user interface.
5. Ease of transferring the project from one programming company to another and modifying it without referring to the startup company for the project.
6. Ease and flexibility of browsing.
7. Multiple languages easily.
8. The large number of users of the program.

4.4.2 Screenshots of the System

The website that the administrator created as we created it with MySQL databases, used flask with its advanced libraries to improve the performance of the site and make it easy to deal with the form and in order to improve the front interfaces we used bootstraps with html with CSS to make the look better and easy to use the site where the administrator logs in and then displays the home page

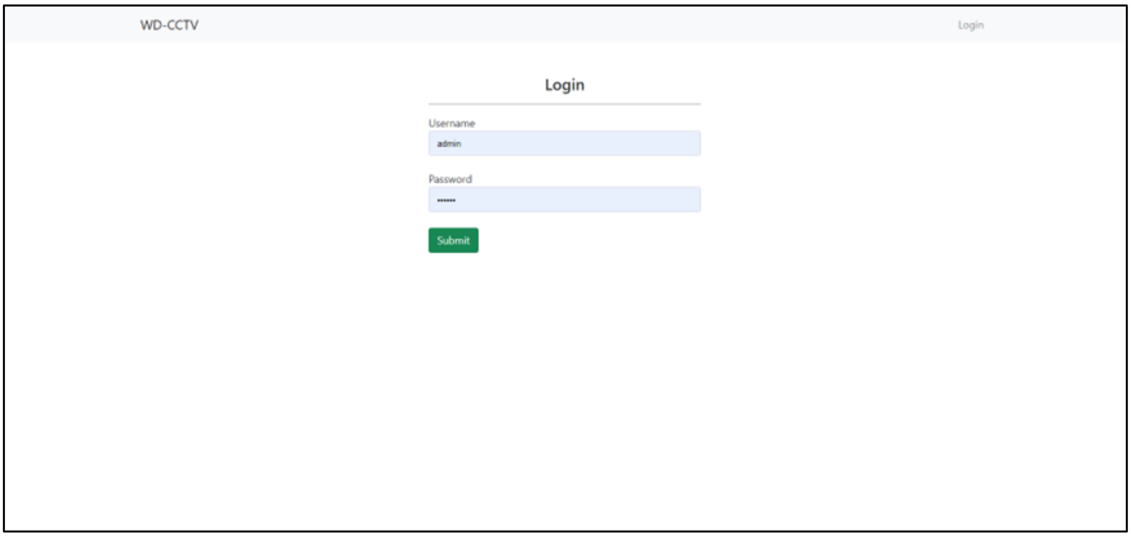


Figure 4-13: Screenshot of login page

Figure 4-14: Screenshot of Home page

After that, the employees working in the system and their contact data. Also, with the possibility of deleting an employee or modifying the data of an employee from my information about him or his work time

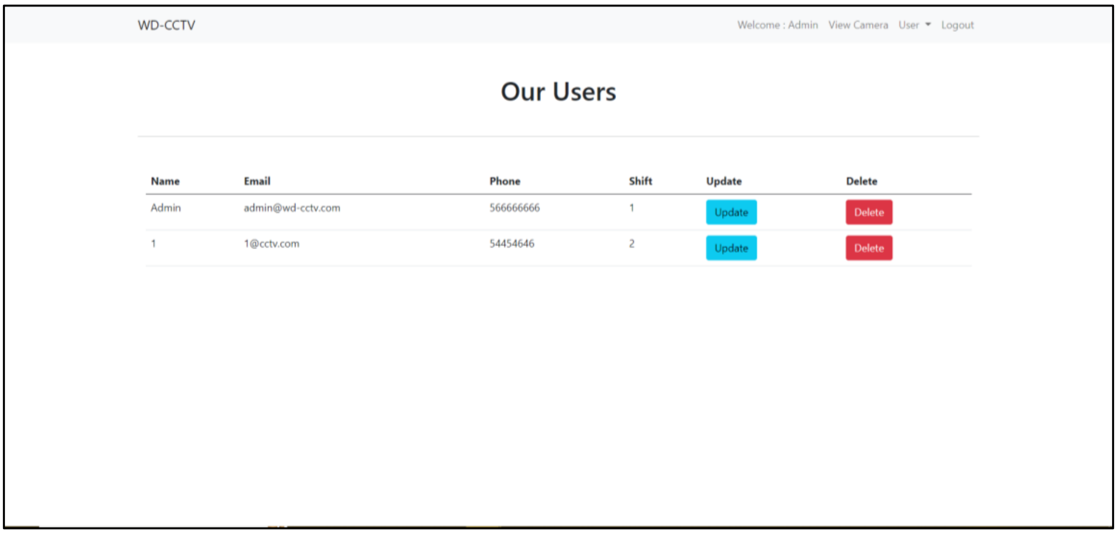


Figure 4-15: Screenshot of users modify page

He can also add new employees and add all his data from his name, number and working time, and the employee cannot modify it, as it is for the admin

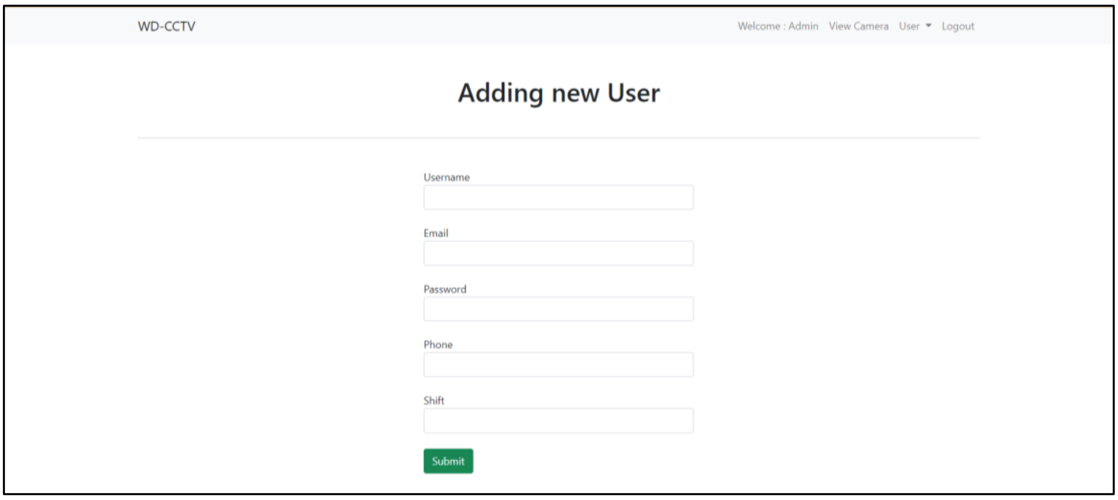


Figure 4-16: Screenshot of add users page

Also, when it shows the camera, the model starts and detects any dangerous object and makes a boundary box on the knife or gun in two different colors.

If the model identifies an object, its presence or location information is saved in the database and then an alarm is triggered in order to inform the competent authorities of the existence of a danger in the specified place and the specified location

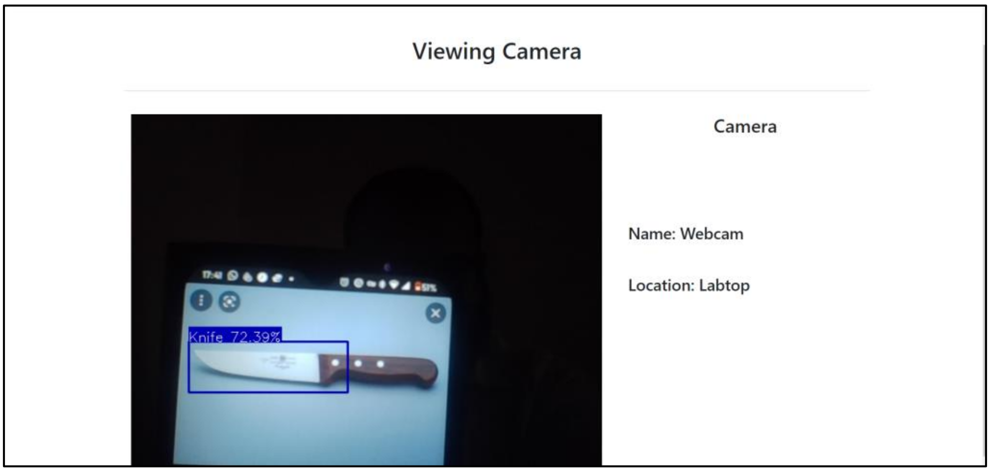


Figure 4-17: Screenshot of viewing camera page (Knife)

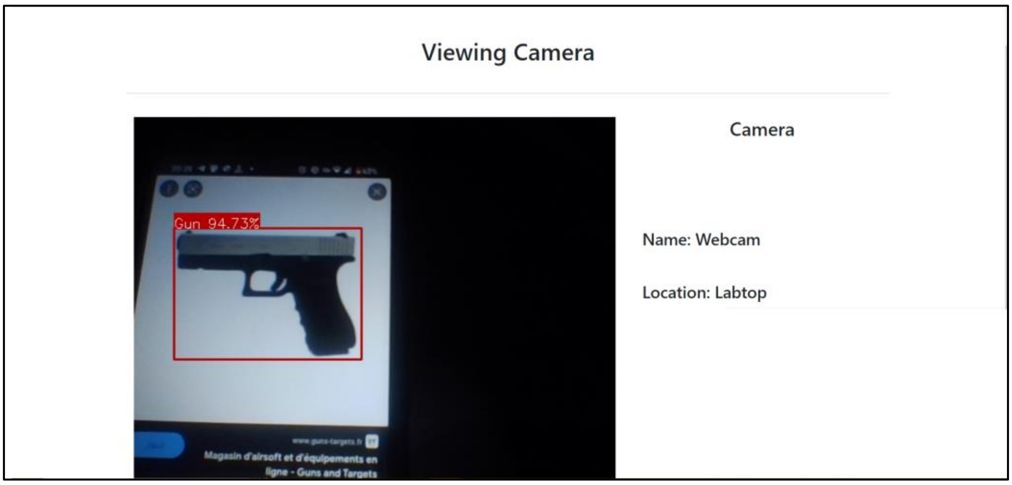


Figure 4-18: Screenshot of viewing camera page (Gun)

4.5 SYSTEM TESTING

4.5.1 Final Model testing

After we tested the model, we found that there is a deficiency in the data related to the knife. We modified the test again to raise the percentage of knowledge of the object successfully. Before the modification, the percentage was 33%, after we modified it and selected suitable images that the model can test, and almost 90% of it is also linked. In the past, the model recognized unwanted images or objects such as pens.... But after we added 600 negative images in order to make the model recognize the negative image, the detection rate of negative images became very low, and they are not detected.

4.5.2 Web software testing

At this point, the model has been trained and given the desired results. As stated at the end of the last section. The harmonious functioning of the system with the database is ensured.

The system is fully secured, by encryption, authization, Giving full privacy to any person within the picture frame; What is saved within the system is only the rectangle of the detected weapon.

4.6 PROFESSIONAL, ETHICAL, AND SOCIAL IMPACT

This project benefits the community because it provides security for the community and protects them from attacks by thieves or criminals in public places and markets. It provides a safe and stable environment for the members of the complex, but it is safe and preserves privacy because the images and unimportant data are not saved in the database. Also, the images of criminals are not shown to anyone who is not allowed. For them to see it, what is saved is only the picture of the weapon and its location in order to quickly search for it and discover it . Also, one of the most important points that our system provides is its ease of use. It also does not affect the public interests and does not cause harm to others, and the information is kept strictly confidential.

Chapter Five

5 CONCLUTION

In this chapter, we discuss the final goals of this project, and future projects.

Nowadays, with the accessibility of huge datasets, quicker GPUs, advanced machine learning algorithms, and better calculations, we can now effectively prepare PCs and develop automated computer-based system to distinguish and identify the danger of weapon risks on a site with high accuracy.

We used the newest models, like YOLOv5, which had very good results and were fast. We used them to find weapons, which had not been done in any published research article up to that point. To improve our F1-score, we also used there-processing to blur the background with gaussian blur. The most promising results came from our YOLO-v5s model, which was trained on the 3000-pistol image dataset provided by the University of Granada and tested on YouTube videos. Our best results were 99 percent recall and 81 percent precision on images and 93 percent precision and 94 percent recall on the video. This is better than the results of similar research, especially in terms of recall. Together with an alarm system, this research can be used to effectively find pistols. In the future, we will build on this model by using other preprocessing techniques, such as brightness control. We also found a few places where the model's performance can be improved by fixing problems, such as custom weapons that don't look like standard pistols and tried techniques that help tell the difference between objects of similar size and shape, which are some of the biggest challenges in the field of weapon detection.

Also, we hope to try making them brighter and darker, adding moving pistols to the training set, and customizing the images and colors on the pistols to try to cut down on false positives and false negatives.

And both the background subtraction method, which works well in the internal environment, and the HOG method of human segmentation, which works well in the external environment, can be combined into a single structure and made to work in any environment. For real-time use, the algorithms that have been proposed need to be made more sensitive and specific. 3. False alarms can be cut down on to make the multi-class classifier more accurate. The algorithm is used on images and then extended to the recorded video by adding up the frames to find the knife. It can be made to work with live video detection. For the algorithm to work with live video, it must be very fast.

Our project can be used in the future to serve the following ideas:

- Detect persons and crowd, then system will report places with high crowd.
- Detect whether a person wears or misplaces a face mask or not.
- Detection of smoking and report the place of a smoker.

REFERENCES

[1] D. Cao, Z. Chen, and L. Gao, "An improved object detection algorithm based on multi-scaled and deformable convolutional neural networks," *Hum.-Centric Comput. Inf. Sci.*, vol. 10, no. 1, p. 14, Apr. 2020, doi: 10.1186/s13673-020-00219-9.

[2] O. Russakovsky *et al.*, "ImageNet Large Scale Visual Recognition Challenge," *ArXiv14090575 Cs*, Jan. 2015, Accessed: Oct. 10, 2021. [Online]. Available: http://arxiv.org/abs/1409.0575

[3] J. Brownlee, "A Gentle Introduction to Object Recognition With Deep Learning," *Machine Learning Mastery*, May 21, 2019. https://machinelearningmastery.com/object-recognition-with-deep-learning/ (accessed Oct. 10, 2021).

[4] V. Zhou, "Machine Learning for Beginners: An Introduction to Neural Networks," *Medium*, Dec. 20, 2019. https://towardsdatascience.com/machine-learning-for-beginners-an-introduction-to-neural-networks-d49f22d238f9 (accessed Dec. 23, 2021).

[5] "AI vs. Machine Learning vs. Deep Learning vs. Neural Networks: What's the Difference?," Nov. 23, 2021. https://www.ibm.com/cloud/blog/ai-vs-machine-learning-vs-deep-learning-vs-neural-networks (accessed Dec. 23, 2021).

[6] "What are Neural Networks?," Aug. 03, 2021. https://www.ibm.com/cloud/learn/neural-networks (accessed Dec. 23, 2021).

[7] K. Mittal, "A Gentle Introduction Into The Histogram Of Oriented Gradients," *Analytics Vidhya*, Dec. 21, 2020. https://medium.com/analytics-vidhya/a-gentle-introduction-into-the-histogram-of-oriented-gradients-fdee9ed8f2aa (accessed Oct. 10, 2021).

[8] "Histogram of Oriented Gradients and Object Detection," *PyImageSearch*, Nov. 10, 2014. https://www.pyimagesearch.com/2014/11/10/histogram-oriented-gradients-object-detection/ (accessed Oct. 10, 2021).

[9] "Single Shot Detection (SSD) Algorithm," *OpenGenus IQ: Computing Expertise & Legacy*, Jan. 21, 2019. https://iq.opengenus.org/single-shot-detection-ssd-algorithm/ (accessed Oct. 10, 2021).

[10] "13.7. Single Shot Multibox Detection — Dive into Deep Learning 0.17.0 documentation." https://d2l.ai/chapter_computer-vision/ssd.html (accessed Oct. 10, 2021).

[11] A. Choudhury, "Top 8 Algorithms For Object Detection," *Analytics India Magazine*, Jun. 16, 2020. https://analyticsindiamag.com/top-8-algorithms-for-object-detection/ (accessed Oct. 10, 2021).

[12] S. Majumder, "Object Detection Algorithms-R CNN vs Fast-R CNN vs Faster-R CNN," *Analytics Vidhya*, Jul. 04, 2020. https://medium.com/analytics-vidhya/object-detection-algorithms-r-cnn-vs-fast-r-cnn-vs-faster-r-cnn-3a7bbaad2c4a (accessed Oct. 10, 2021).

[13] "R-CNN vs Fast R-CNN vs Faster R-CNN | ML," *GeeksforGeeks*, Feb. 28, 2020. https://www.geeksforgeeks.org/r-cnn-vs-fast-r-cnn-vs-faster-r-cnn-ml/ (accessed Oct. 10, 2021).

[14] R. Gandhi, “R-CNN, Fast R-CNN, Faster R-CNN, YOLO — Object Detection Algorithms,” *Medium*, Jul. 09, 2018. https://towardsdatascience.com/r-cnn-fast-r-cnn-faster-r-cnn-yolo-object-detection-algorithms-36d53571365e (accessed Oct. 10, 2021).
[15] “Object Detection / Fast R-CNN - Convolutional Neural Networks for Image and Video Processing - TUM Wiki.” https://wiki.tum.de/pages/viewpage.action?pageId=22578448#ObjectDetection/FastRCNN-Weblinks (accessed Oct. 11, 2021).
[16] R. Girshick, “Fast R-CNN,” *ArXiv150408083 Cs*, Sep. 2015, Accessed: Oct. 11, 2021. [Online]. Available: http://arxiv.org/abs/1504.08083
[17] S. Ren, K. He, R. Girshick, and J. Sun, “Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks,” *ArXiv150601497 Cs*, Jan. 2016, Accessed: Oct. 11, 2021. [Online]. Available: http://arxiv.org/abs/1506.01497
[18] R. Girshick, *py-faster-rcnn has been deprecated. Please see Detectron, which includes an implementation of Mask R-CNN*. 2021. Accessed: Oct. 11, 2021. [Online]. Available: https://github.com/rbgirshick/py-faster-rcnn
[19] “Object Detection / Fast R-CNN - Convolutional Neural Networks for Image and Video Processing - TUM Wiki.” https://wiki.tum.de/pages/viewpage.action?pageId=22578448#ObjectDetection/FastRCNN-Weblinks (accessed Oct. 11, 2021).
[20] A. Singh, “YOLO: You Only Look Once,” *Nerd For Tech*, May 29, 2021. https://medium.com/nerd-for-tech/yolo-you-only-look-once-65ea86104c51 (accessed Oct. 11, 2021).
[21] “Papers with Code - You Only Look Once: Unified, Real-Time Object Detection.” https://paperswithcode.com/paper/you-only-look-once-unified-real-time-object (accessed Oct. 11, 2021).
[22] “Publications.” https://pjreddie.com/publications/ (accessed Oct. 11, 2021).
[23] J. Redmon and A. Farhadi, “YOLOv3: An Incremental Improvement,” Apr. 2018, Accessed: Oct. 11, 2021. [Online]. Available: https://arxiv.org/abs/1804.02767v1
[24] “YOLO: Real-Time Object Detection.” https://pjreddie.com/darknet/yolo/ (accessed Oct. 12, 2021).
[25] “How to Train A Custom Object Detection Model with YOLO v5 | by Jacob Solawetz | Towards Data Science.” https://towardsdatascience.com/how-to-train-a-custom-object-detection-model-with-yolo-v5-917e9ce13208 (accessed Dec. 24, 2021).
[26] “YOLOv5 v6.0 is here - new Nano model at 1666 FPS.” https://blog.roboflow.com/yolov5-v6-0-is-here/ (accessed Dec. 24, 2021).
[27] “Papers with Code - PASCAL VOC 2007 Benchmark (Object Detection).” https://paperswithcode.com/sota/object-detection-on-pascal-voc-2007 (accessed Oct. 11, 2021).
[28] J. Du, “Understanding of Object Detection Based on CNN Family and YOLO,” *J. Phys. Conf. Ser.*, vol. 1004, p. 012029, Apr. 2018, doi: 10.1088/1742-6596/1004/1/012029.
[29] “Crime-Detection--using-Deep-learning/data at master · sowmyadvn/Crime-Detection--using-Deep-learning,” *GitHub*. https://github.com/sowmyadvn/Crime-Detection--using-Deep-learning (accessed Sep. 28, 2021).
[30] “GitHub - sowmyadvn/Crime-Detection--using-Deep-learning: Developed an object detection model using YOLO Darknet framework to prevent crimes.

Collected the crime data manually from Google images as well as Imagenet database.” https://github.com/sowmyadvn/Crime-Detection--using-Deep-learning (accessed Sep. 28, 2021).
[31] *Weapon detection datasets*. ARI-DaSCI, 2021. Accessed: Sep. 28, 2021. [Online]. Available: https://github.com/ari-dasci/OD-WeaponDetection
[32] “Weapon Detection – DaSCI.” https://dasci.es/transferencia/open-data/24705/ (accessed Sep. 28, 2021).
[33] “Weapon-Detection-And-Classification/train at master · ivaibhavkr/Weapon-Detection-And-Classification,” *GitHub*. https://github.com/ivaibhavkr/Weapon-Detection-And-Classification (accessed Sep. 28, 2021).
[34] F. Gelana and A. Yadav, “Firearm Detection from Surveillance Cameras Using Image Processing and Machine Learning Techniques,” in *Smart Innovations in Communication and Computational Sciences*, Singapore, 2019, pp. 25–34. doi: 10.1007/978-981-13-2414-7_3.
[35] F. Gelana, *Handgun-Dataset*. 2021. Accessed: Sep. 28, 2021. [Online]. Available: https://github.com/FrexG/Handgun-Dataset
[36] “Knife Dataset | Kaggle.” https://www.kaggle.com/shank885/knife-dataset (accessed Sep. 28, 2021).
[37] darrenl, *LabelImg*. 2022. Accessed: Mar. 16, 2022. [Online]. Available: https://github.com/tzutalin/labelImg