Weapon Detection Using Neural Networks in Real-Time Surveillance Video

BY:
- Eman Wasfi Azaizeh
- Ramah Hashem Madi
- Leen Sufian Alquran
- Noor Mahmoud faronia

Supervised By:
Dr. Hussein Al-Ahmer

This Project (Graduation Project I) is Submitted in Partial Fulfillment of the Requirements for bachelor's degree in Artificial intelligence and robotics

Faculty of Artificial Intelligence
Al-Balqa Applied University
Al-Salt- Jordan
September 2022

DEDICATION

We dedicate this project to our parents, who offered unconditional love and support and have always been there for us. To our sisters, brothers, and friends who always stood beside us and was a catalyst and for all of those who made this entire work possible, words of endless encouragement, thank you so much. And finally, to ourselves, because we were able to get here despite all the things around us and all the times, we thought about giving up, but our soul decided to stand up again and finish what we started.

ACKNOWLEDGMENTS

Foremost, I would like to express my sincere gratitude to the director of this project Dr. Hussein Al-Ahmer for his continuous support to us in this project and his patience, motivation, and immense knowledge. His guidance helped us in all the time of research and writing of this project. My sincere thanks also go to prof. Abdelwadood Mesleh for the learning opportunities he provided to us over the course of four years and for his continued support and encouragement.

Dr. Hussein Al-Ahmer and prof. Abdelwadood Mesleh, thank you because you were more than teachers, you were like a father and brother to us, and wherever this life will place us, we will not forget all that we have learned from you on the academic level or on the level of life itself. Whatever you have given us will always remain in our memory

TABLE OF CONTENTS

TABLE OF CONTENTS

TABLE OF Figures

TABLE OF Figures

TABLE OF Table

ABSTRACT

In this project, we build a system that is able to detect weapons in real-time surveillance camera (CCTV) using YOLOv7 algorithm, it's a new state-of-the-art for real-time object detectors released in 2022, since non-of the previous work implement this version of the algorithm for this kind of issue, we were the first who implement YOLOv7. The aim of this project is to investigate the effect of training YOLOv7 with three classes "knife", "gun" and "pistol" on detecting weapons in real-time. As the result of this project, YOLOv7 model that pre-trained on MS COCO dataset, produced a significant result of 92.2% mean average precision in detecting knife, gun and pistol classes. All details have been worked out for this work.

CHAPTER ONE

Introduction

1.1 Overview

Based on The Jordan Time pepper with its title Bearing firearms in Jordan published in 2019, the “Minister of Interior touched a sensitive cord when he revealed on a parliamentary debate on the amendments to the legislation on «weapons and ammunition» that 92 percent of all violent crimes committed in the country are carried out with unlicensed arms.” [1]

Gun-related violence occurs all around the world, including (to a lesser extent) in countries in which guns are illegal. According to World Population Review, out of the estimated 250,227 gun-related deaths worldwide in 2019, %65.9 occurred in just six countries: Brazil, the United States, Venezuela, Mexico, India, and Colombia. Gun deaths are considered an epidemic in the United States (which leads the world in civilian gun ownership) by many people [2].in Figure 1.1 shows Gun Deaths by Counter 2022.

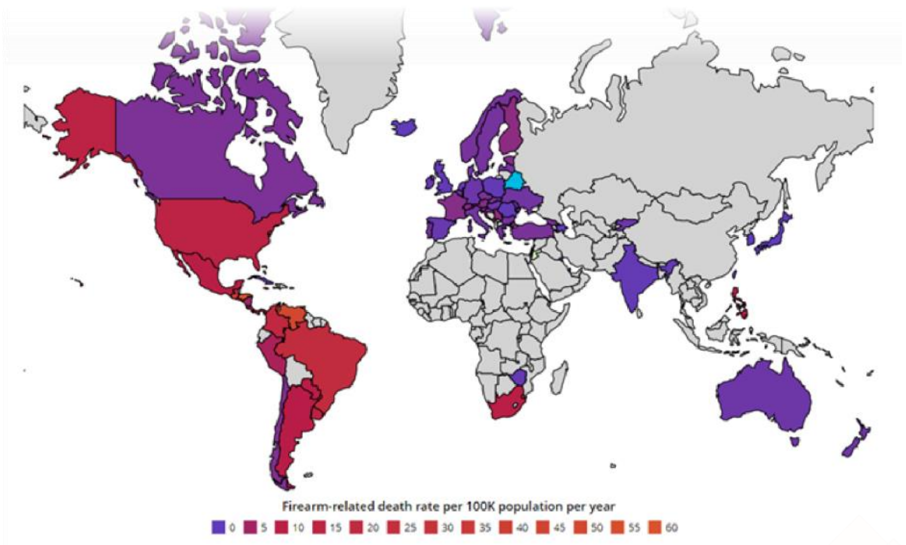


Figure 1.1: Gun Deaths by Country 2022

In most current surveillance systems, monitoring depends on the existence of a human element. This makes monitoring a very challenging task. In addition, it is labor-intensive and prone to errors. These traditional systems have many problems such as weak security, low intelligence, high cost, and poor stability. Most of these systems are based on human operators. It is difficult for these operators to watch and analyze all the dangerous situations, especially with the long observation periods and many cameras.[3]

So, it is necessary to review our surveillance system again with the knowledge of new technology. we should prepare and prevent the dangers and not repair and repent the wounds, One way to reduce this kind of violence is prevention via early detection so that the security agents or policemen can act. One innovative solution to this problem is to equip surveillance or control cameras with an accurate automatic weapon detection alert system.[4]

In this research work, we aim to develop a smart surveillance security system detecting weapons specifically guns and knives. For this purpose, we will apply a few computer vision methods and deep learning to identify a weapon from a captured image. Recent work in machine learning and deep learning, particularly convolutional neural networks, has shown considerable progress in object detection and recognition, exclusively in images. As the first step for any video surveillance application, object detection and classification are essential for further object-tracking tasks. For this purpose, we will train the classifier model of YOLO v7, i.e., “You Only Look Once”. this model is a state-of-the-art real-time object detection classifier.

1.2 Project Aim and Objective

The aims of this project are:

1. Design and implement an autonomous weapon detection system using a new state-of-the-art YOLOv7 algorithm with our customized dataset.
2. Compare the performance of YOLOv7 system with another system that was built using Faster R-CNN from previous work.
3. Choose the best system and try to improve its accuracy and speed.
4. we aim to predict a class of an object and the bounding box specifying object location.

1.1 Project Stockholders

This project may use by:

1. University and schools Security
2. Show owners
3. Airport security

1.4 Outline of the Project Report

The rest of the project documentation is organized as follows:

- Chapter 2: Related Work

  This chapter described the work that has been done in Weapon Detection Using Neural Networks in Real-Time Surveillance Video and it'll contain numerous papers related to our project.

- Chapter 3: proposed work/ Methodology

  This chapter distributed into three sections: Section 3.1 talks about the requirements of our project «in Weapon Detection Using Neural Networks in Real-Time Surveillance Video» (3.1 Project Requirements) while Section 3.2 (Project Modeling) here we talking about CNN, Faster RCNN, Resnet-50, Feature Pyramid Networks, finally YOLO algorithm. Section3.3 (Project Methodology / Techniques) will describe our methodology/techniques we aim to use in our project.

CHAPTER TWO

Related Work

.

2.1 Introduction

Much research has been done in Object Detection; this chapter describes the works in a little bit of details. Section 2.2 summaries the attempts related to weapon detection Using Neural Networks in Real-Time Surveillance Video.

2.2 Summaries of Attempts

2.2.1 Faster R-CNN

Faster R-CNN The most widely used state-of-the-art version of the R-CNN family — Faster R-CNN was first published in 2015.

M. Milagro Fernandez-Carrobles, Oscar Deniz, Fernando Maroto [5] introduced a system for detecting pistols and knives based on the Faster R-CNN methodology. Two methods were compared using GoogleNet architecture and SqueezeNet as a CNN base, respectively. The best result for gun detection was obtained using a SqueezeNet architecture achieving a %85.44 AP50.

Jose González, Carlos Zaccaro, Juan A. AlvarezGarcía et al.,[6] The authors have used Faster R-CNN object detector using FPN, there are many problems in bringing weapon detection to real CCTV scenarios. One of them is the distance from the object to the camera, the farther the weapon is from the camera, the more difficult it is to identify it. However, increasing the dataset improves the weapon discovery process. and shows that training using synthetic data Increases the accuracy.

Palash Yuvraj Ingle and Young-Gab Kim [7] proposed a resource-constrained lightweight subclass detection method based on a convolutional neural network to classify, locate, and detect different types of guns and knives effectively and efficiently in a real-time environment. This resource-constrained device has shown a satisfactory result with a precision score of 85.5% for detection.

2.2.2 YOLO

You only look once (YOLO) is a state-of-the-art, real-time object detection system. first described in the seminal 2015 paper by Joseph Redmon.

Sanam Narejo, Bishwajeet Pandey, and Doris vargas et al.,[8] In this research YOLO v3 and YOLO v2 were used for real-time weapon detection and it was found that YOLO v3 has better performance than YOLO v2 and traditional convolutional neural network (CNN) and is lower in terms of calculations. YOLO v3 has an accuracy of 98.89, YOLO v2 has 96.76 and CNN has 95.

Akash Kumar. K. S, Akshita. B. P, Arjun. M, Dr. N. Geetha[9], The authors Experiment on weapon detection in surveillance system using yolov3 algorithm and it was found that yolov3 faster than faster R-CNN algorithms, Speed has a major role in detecting objects in monitoring systems and transmitting alerts to security.

MUHAMMAD TAHIR BHATTI, MUHAMMAD GUFRAN KHAN, MASOOD ASLAM et al,[10] This work focused on providing a safe place using CCTV footage as a source for detecting harmful weapons. They implemented binary classification by assuming the pistol class as the reference class and introduced the concept of including related confusion objects to reduce false positives and false negatives. Some of the algorithms used are VGG16, Inception-V3, Inception-ResnetV2, SSDMobileNetV1, Faster-RCNN Inception-ResnetV2 (FRIRv2), YOLOv3, YOLOv4. Yolov4 stands out best among all other algorithms and has an F1 score of 91% with a mean average accuracy of 91.73%.

CHAPTER THREE

Proposed Work/ Methodology

3.1 Project Requirements

1. Create/collect a training dataset.
2. Train a custom model
3. Deploy the model to your app.

3.2 Modeling

In order to work with this project on real-time data, first, we need to understand the basics of Convolution Neural Network (CNN), CNN is a type of artificial neural network, which is widely used for image/object recognition and classification, in this project we will use **ResNet-50** as a pre-trained CNN model.

Real-time data is data that is available as soon as it's created and acquired, like the sequence of images (frame per second(fps)) that is obtained from a CCTV camera, this kind of data will process in a model that is used for real-time object detection, for that we will use the new stat-of-art YOLOv7, YOLOv7 is a real-time object detector currently revolutionizing the computer vision industry with its incredible features.

Then we will use another model Faster R-CNN, Faster R-CNN is one of the famous object detection architectures that use convolution neural networks, this model was built using a feature pyramid network.

In this section we will discuss in detail CNN, ResNet-50, Faster R-CNN, feature pyramid network, and finally YOLOv7.

3.2.1 Convolution Neural Network

Convolution Neural Network (CNN) is a state-of-art deep artificial neural network technique for image classification, it clusters similar images and performs object recognition within regions.[3]

For about a decade, deep learning in general and Convolution Neural networks (CNNs) in particular have achieved superior results to all the classical machine learning methods in image classification, detection, and segmentation in several applications. Instead of manually selecting a feature, deep learning CNNs automatically discover increasingly higher-level features from data.

The CNN methods are commonly used in object detection, such as Region-Based Convolutional Neural networks (RCNN), Fast CNN, and Faster R-CNN, we will discuss these methods later.

Proper training of deep CNNs, which contain millions of parameters, requires very large datasets, in the order of millions of samples. Transfer learning through fine-tuning is becoming a widely accepted alternative to overcome these constraints. It consists of re-utilizing the knowledge learned from one problem to another related one. Applying transfer learning with deep CNNs depends on the similarities between the original and new problem and the size of the new training set.[11]

In general, fine-tuning the entire network, i.e., updating all the weights, is only used when the new dataset is large enough, else the model could suffer overfitting, especially among the first layers of the network. Since these layers extract low-level features, e.g., edges and color, they do not change significantly and can be utilized for several visual recognition tasks. The last layers of the CNN are gradually adjusted to the particularities of the problem and extract high-level features, which are not readable by the human eye. In this work, we used a ResNet-50 based classification model pre-trained on the ImageNet dataset. Figure 3.1 shows the CNNs timeline.

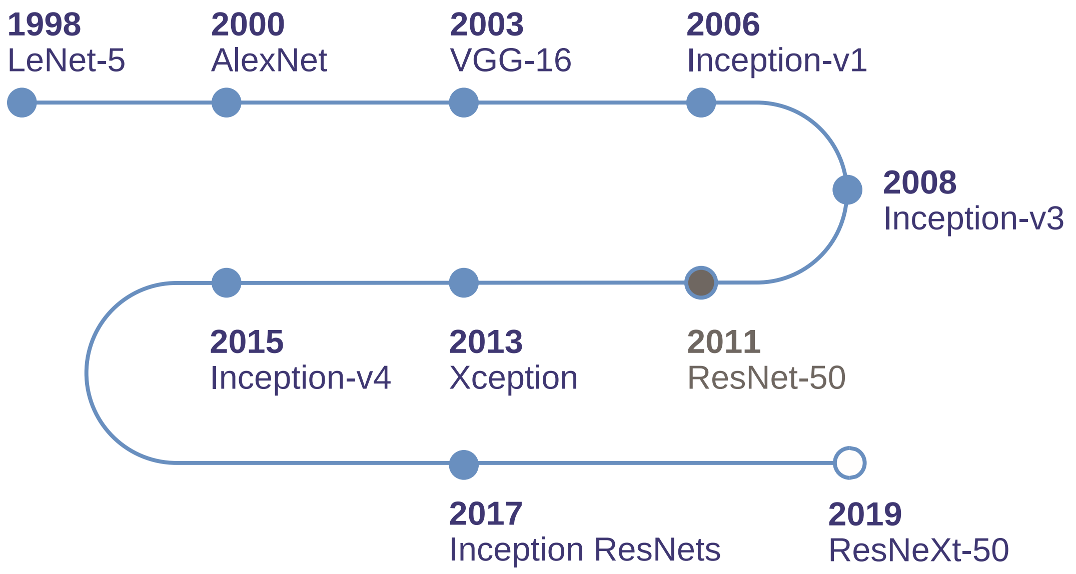
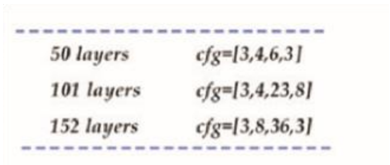


Figure 3.1: CNNs Architectures over a Timeline (1998-2019)

3.2.2 ResNet-50

ResNet-50 is a convolutional neural network that is 50 layers deep. ResNet, short for Residual Networks is a classic neural network used as a backbone for many computer vision tasks. The fundamental breakthrough with ResNet was it allowed us to train extremely deep neural networks with 150+layers. It is an innovative neural network that was first introduced by Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun in their 2015 computer vision research paper titled 'Deep Residual Learning for Image Recognition.

CNNs have a major disadvantage 'Vanishing Gradient Problem'. During backpropagation, the value of the gradient decreases significantly, thus hardly any change comes to weights. To overcome this, ResNet is used. It makes use of "SKIP CONNECTION".[12] Figure 3.2 shows the ResNet-50 Architecture.



50 layers	*cfg=[3,4,6,3]*
101 layers	*cfg=[3,4,23,8]*
152 layers	*cfg=[3,8,36,3]*

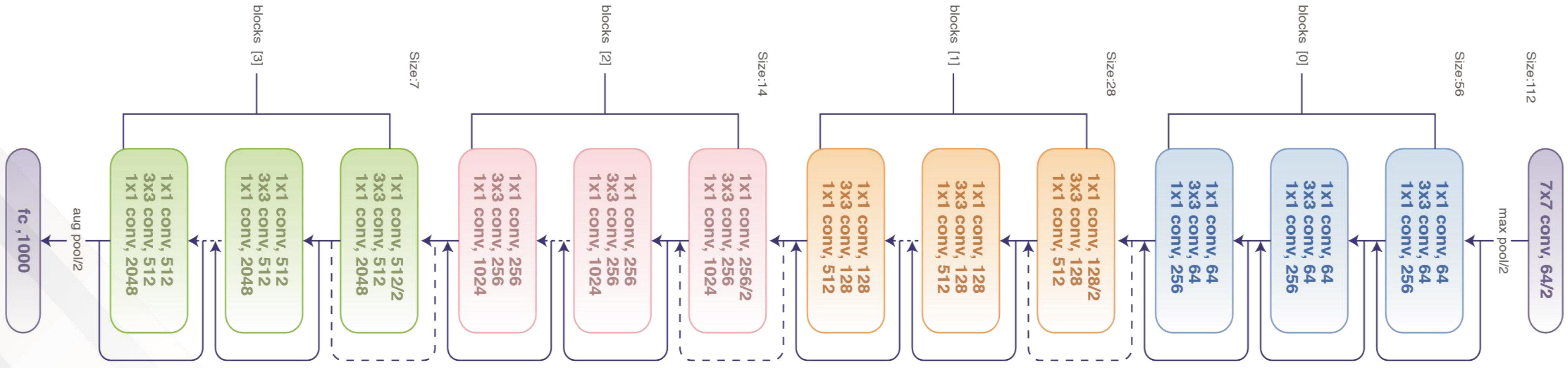


Figure 3.2: shows the ResNet-50 Architecture

The skip connection diagram shows in Figure 3.3

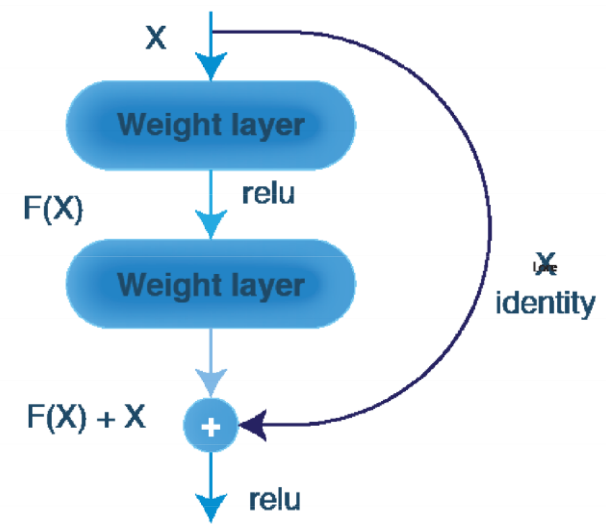


Figure 3.3: The Skip Connection

All algorithms train on the output ‘Y’ but, ResNet trains on F(X). In simpler words, ResNet tries to make F(X)=0 so that Y=X.

SKIP CONNECTION is a direct connection that skips over some layers of the model. The output is not the same due to this skip connection. Without the skip connection, input X gets multiplied by the weights of the layer followed by adding a bias term.Then comes the activation function, F() and we get the output as, Equation 1 :

$$F(Wx+b)=F(X) \tag{1}$$

where W is the weight, x is the input, b is bias, and F() is the activation function. But with the skip connection technique, the output is, Equation 2:

$$F(X)+x \tag{2}$$

In ResNet-50, there are two kinds of blocks:

The value of 'x' is added to the output layer if and only if the Input size is equal to the Output size, however, If this is not the case, we add a **'convolutional block'** in the shortcut path to make the input size equal to output size, as represented in Figure 3.4 the Identity block then Convolution block.

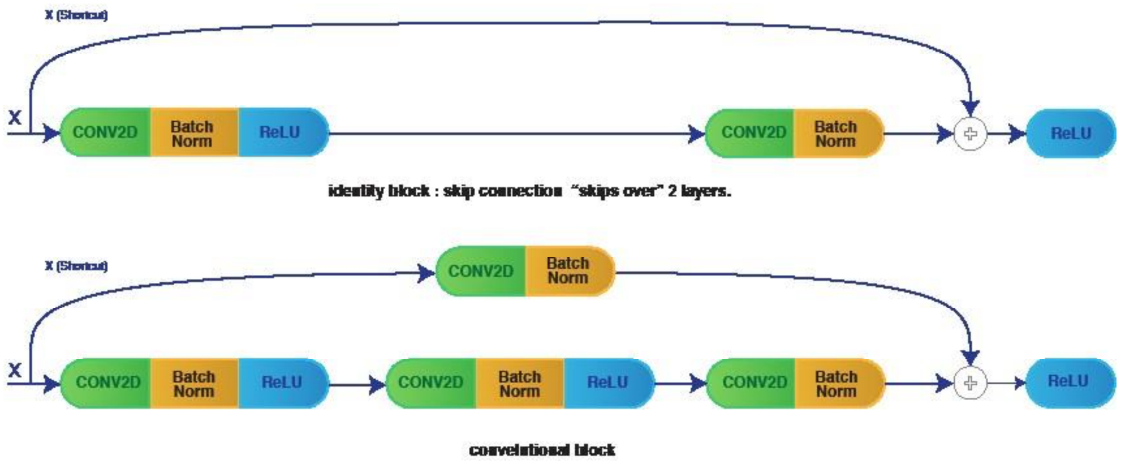


Figure 3.4: The Identity block then Convolutional block

There are 2 ways to make the input size equal to the output size:

1. Padding the input volume.
2. Performing 1*1 convolutions.

The size of the output layer is calculated as represented in Equation 3:

$$O^{I} = [\{(n + 2p - f) / s\} + 1]\, 2 \qquad (3)$$

where the input image size is represented as n, padding as p, stride as s, OI as the size of the output layer, and a number of filters as f.

3.2.3 Types Of R-CNN

The most widely used state-of-the-art version of the R-CNN family Faster R-CNN was first published in 2015. This article, the third and final one of a series to understand the fundamentals of current-day object detection elaborates on the technical details of the Faster R-CNN detection pipeline.[13]

Know we will do a quick review for the region-based CNN (R-CNN), which is the first trial towards building an object detection model that extracts features using a pre-trained CNN. Next, Fast R-CNN is quickly reviewed, which is faster than the R-CNN but unfortunately neglects how the region proposals are generated. This is later solved by the Faster R-CNN, which builds a region-proposal network that can generate region proposals that are fed to the detection model (Fast R-CNN) to inspect objects. The generic pipeline of the object detection techniques is shown in Figure3.5.

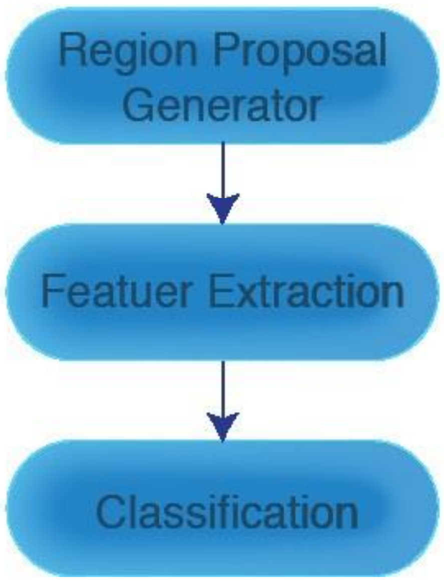


Figure 3.5: shows the generic pipeline of the object detection techniques

3.2.3.1 R-CNN

In 2014, a group of researchers at UC Berkely developed a deep convolutional network called R-CNN (short for region-based convolutional neural network) [14] that can detect 80 different types of objects in images.

Compared to the generic pipeline of the object detection techniques shown in the previous Figure, the main contribution of R-CNN [14] is just extracting the features based on a convolutional neural network (CNN). Other than this, everything is similar to the generic object detection pipeline. The next Figure 3.6 shows the working of the R-CNN model.

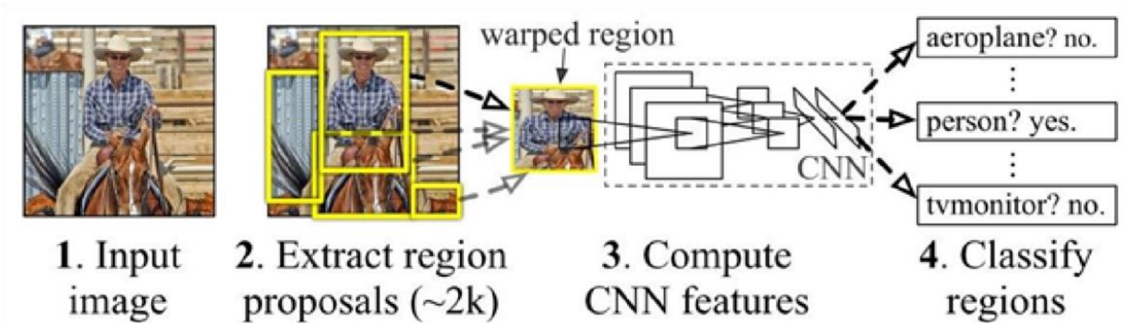


Figure 3.6: Architecture of R-CNN model.

The R-CNN model has some drawbacks [15]**:**

1. It is a multi-stage model, where each stage is an independent component. Thus, it cannot be trained end-to-end.
2. It caches the extracted features from the pre-trained CNN on the disk to later train the SVMs. This requires hundreds of gigabytes of storage.
3. R-CNN depends on the Selective Search algorithm for generating region proposals, which takes a lot of time. Moreover, this algorithm cannot be customized to the detection problem.

3.2.3.2 Fast R-CNN

Fast R-CNN [15] is an object detector that was developed solely by Ross Girshick. Fast R-CNN overcomes several issues in R-CNN. As its name suggests, one advantage of the Fast R-CNN over R-CNN is its speed.

The general contributions of Fast R-CNN [15], are Proposing a new layer called ROI Pooling that extracts equal-length feature vectors from all proposals (i.e. ROIs) in the same image, and Fast R-CNN does not cache the extracted features and thus does not need so much disk storage compared to R-CNN, which needs hundreds of gigabytes. and Fast R-CNN is more accurate than R-CNN.

The general architecture of Fast R-CNN is shown below in Figure 3.7. The model consists of a single stage, compared to the 3 stages in R-CNN. It just accepts an image as an input and returns the class probabilities and bounding boxes of the detected objects.

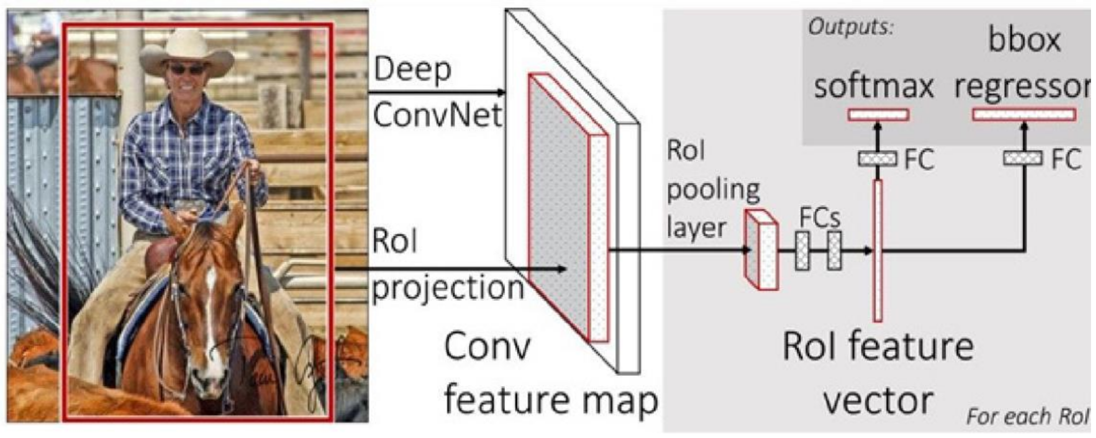


Figure 3.7: The General Architecture of Fast R-CNN

Despite the advantages of the Fast R-CNN model, there is a critical drawback as it depends on the time-consuming Selective Search algorithm to generate region proposals. The Selective Search method cannot be customized for a specific object detection task. Thus, it may not be accurate enough to detect all target objects in the dataset.

3.2.3.3 Faster R-CNN

Faster R-CNN [16] is an extension of Fast R-CNN [15]. As its name suggests, Faster R-CNN is faster than Fast R-CNN thanks to the region proposal network (RPN).

Main Contributions [16]

The main contributions of this paper are Proposing a **region proposal network (RPN)** which is a fully convolutional network that generates proposals with various scales and aspect ratios. The RPN implements the terminology of **neural network with attention** to telling the object detection (Fast R-CNN) where to look, and the convolutional computations are shared across the RPN and the Fast R-CNN. This reduces the computational time.

The architecture of Faster R-CNN is shown in the next Figure. It consists of 2 modules:

1. **RPN:** For generating region proposals.
2. **Fast R-CNN:** For detecting objects in the proposed regions.

The RPN module is responsible for generating region proposals. It applies the concept of attention in neural networks, so it guides the Fast R-CNN detection module to where to look for objects in the image.

Note how the convolutional layers (e.g. computations) are shared across both the RPN and the Fast R-CNN modules.

Figure 3.8 shows The Architecture of Faster R-CNN

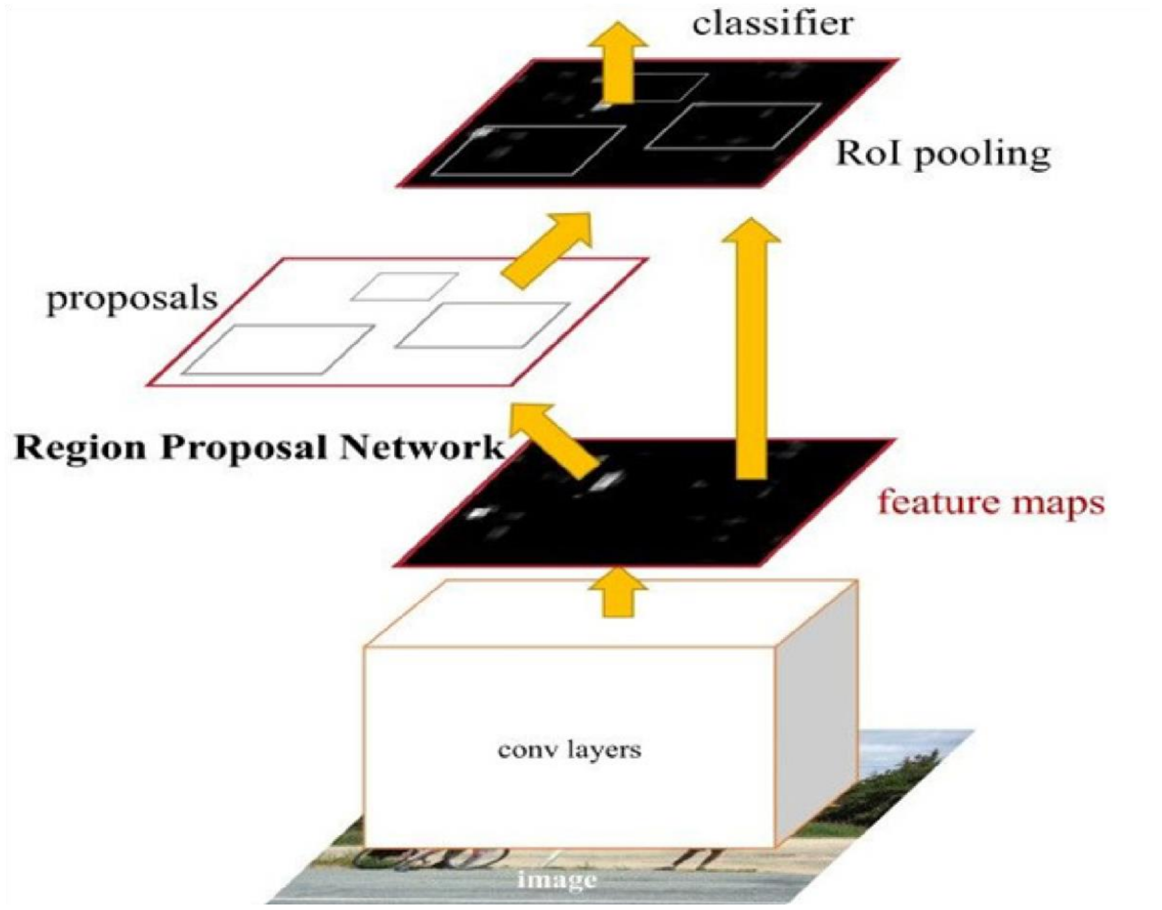


Figure 3.8: The architecture of Faster R-CNN

The Faster R-CNN works as follows:

- The RPN generates region proposals.

For all region proposals in the image, a fixed-length feature vector is extracted from each region using the ROI Pooling layer[15]:

- The extracted feature vectors are then classified using the Fast R-CNN.
- The class scores of the detected objects in addition to their bounding-boxes are returned.

Table 3.1: RCNN vs Fast RCNN vs Faster RCNN

	R-CNN	Fast R-CNN	Faster R-CNN
Test time speed per image	50 seconds	2 seconds	0.2 seconds
Speed-up	1x	25x	250x

3.2.4 Feature Pyramid Networks

Recognizing objects at vastly different scales is a fundamental challenge in computer vision. Feature pyramids built upon image pyramids (for short we call these featurized image pyramids) form the basis of standard solution [17], as we see in Figure 3.9 Pyramid methods in image processing. These pyramids are scale-invariant in the sense that an object's scale change is offset by shifting its level in the pyramid. Intuitively, this property enables a model to detect objects across a large range of scales by scanning the model over both positions and pyramid levels.

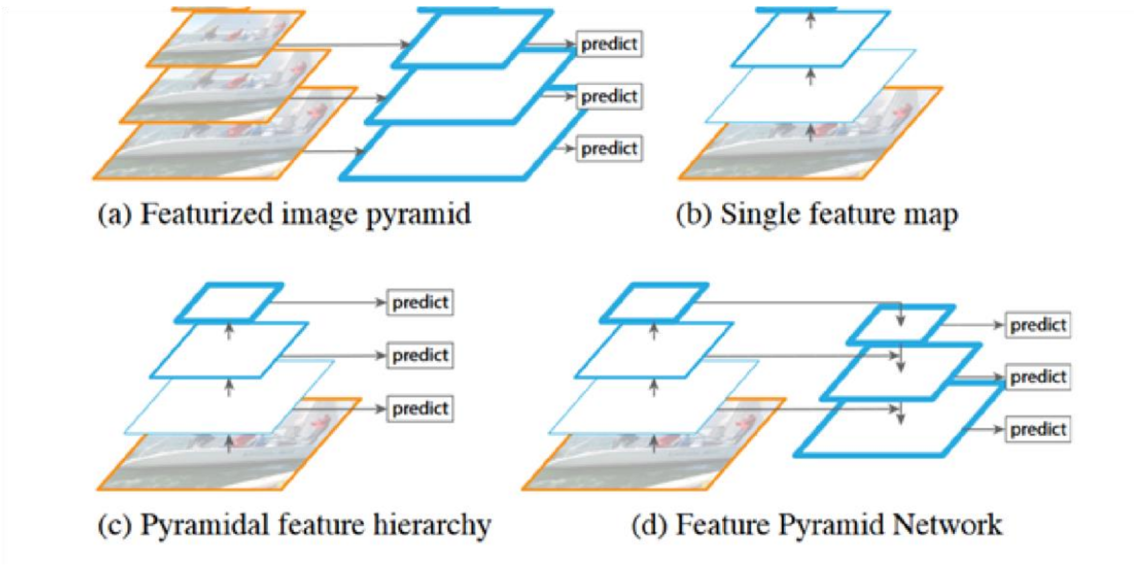

predict
predict
predict
(a) Featurized image pyramid
(b) Single feature map
predict
predict
predict
(c) Pyramidal feature hierarchy
predict
predict
predict
(d) Feature Pyramid Network

Figure 3.9: Pyramid methods in image processing

As shown in Figure 3.9 (a) Featurized image pyramid is heavily used in the era of hand-engineered features, and (b) single feature map is a standard ConvNet solution on a single input image that has the prediction at the end of the network, (c) pyramidal Feature hierarchy at each layer, It reuses the multiscale feature maps from different layers computed in the forward pass and thus comes free of cost. However, it misses the opportunity to reuse the higher-resolution maps of the feature hierarchy and consequently misses the detection of small objects.

(d) feature pyramid network at each layer, It reuses the multi-scale feature maps from different layers computed in the forward pass and thus comes free of cost. However, it misses the opportunity to reuse the higher-resolution maps of the feature hierarchy and consequently misses the detection of small objects.

Figure 3.10 shows Feature Pyramid Network (FPN) architecture

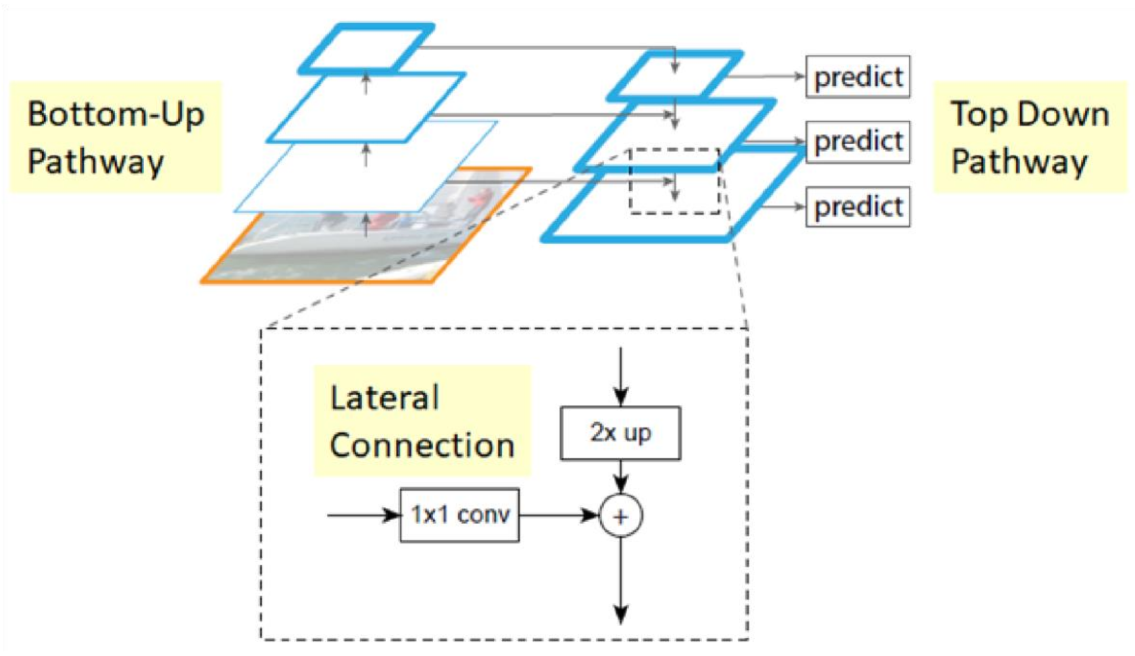


Figure 3.10: Feature Pyramid Network (FPN) architecture

• Bottom-Up Pathway

The bottom-up pathway is the feedforward computation of the backbone ConvNet. It is defined that one pyramid level is for each stage. The output of the last layer of each stage will be used as the reference set of feature maps for enriching the top-down pathway by lateral connection.

• Top-Down Pathway and Lateral Connection

- The higher resolution features is upsampled spatially coarser, but semantically stronger, feature maps from higher pyramid levels. More specifically, the spatial resolution is **upsampled by a factor of 2 using the nearest neighbor for simplicity.**
- Each lateral connection merges feature maps of the same spatial size from the bottom-up pathway and the top-down pathway.
- Specifically, **the feature maps from the bottom-up pathway undergo 1×1 convolutions to reduce the channel dimensions.**
- And **the feature maps from the bottom-up pathway and the top-down pathway are merged by element-wise addition.**

• Prediction

- Finally, **a 3×3 convolution is appended on each merged map to generate the final feature map, which is to reduce the aliasing effect of upsampling.** This final set of feature maps is called {P2, P3, P4, P5}, corresponding to {C2, C3, C4, C5} that are respectively of the same spatial sizes.
- Because all levels of the pyramid use shared classifiers/regressors as in a traditional featurized image pyramid, the feature dimension at output d is fixed with d = 256. Thus, all extra convolutional layers have 256-channel outputs.

In the original RPN design in Faster R-CNN, a small subnetwork is evaluated on dense 3×3 sliding windows, on top of a single-scale convolutional feature map, performing object/non-object binary classification and bounding box regression.[18]

3.2.5 YOLOv7

YOLO ("You Only Look Once") is an effective real-time object recognition algorithm, first described in the seminal 2015 paper by Joseph Redmon et al. [19]

YOLO algorithm is an algorithm based on regression, instead of selecting the interesting part of an Image, it predicts classes and bounding boxes for the whole image in **one run of the Algorithm**.

we aim to predict a class of an object and the bounding box specifying object location. Each bounding box can be described using four descriptors first one is the Center of the box **(bx, by), the second** is the Width **(bw), the third** is the Height **(bh), fourth** the Value c corresponding to the class of an object. Along with that, we predict a real number pc, which is the probability that there is an object in the bounding box.

YOLO doesn't search for interesting regions in the input image that could contain an object, instead, it splits the image into cells, typically a 19x19 grid. Each cell is then responsible for predicting K-bounding boxes. Figure 3.11 shows the bounding box probability calculation.

An Object is considered to lie in a specific cell only if the center coordinates of the anchor box lie in that cell. Due to this property, the center coordinates are

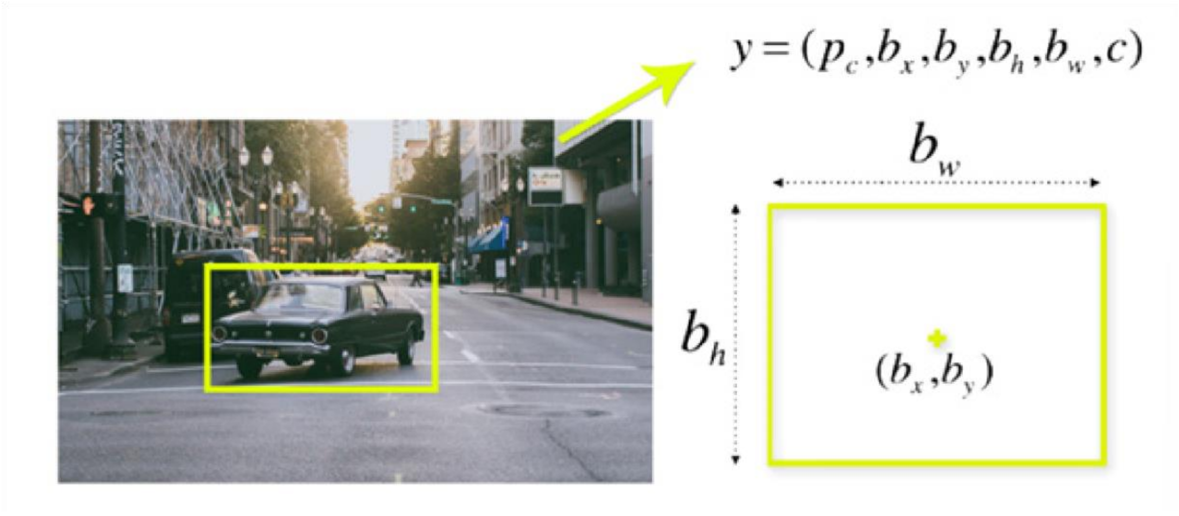


Figure 3.11: Bounding box probability calculation

always calculated relative to the cell whereas the height and width are calculated relative to the whole Image size.

During the one pass of forwards propagation, YOLO determines the probability that the cell contains a certain class as shown in Equation 4:

$$score_{c,i} = p_c \times c_i \tag{4}$$

The probability that there is an object of a certain class ‘c’. The class with the maximum probability is chosen and assigned to that particular grid cell. A similar process happens for all the grid cells present in the image.

Most of these cells and bounding boxes will not contain an object. Therefore, we predict the value pc, which serves to remove boxes with low object probability and bounding boxes with the highest shared area in a process called **non-max suppression**.

Figure 3.12 shows non-max suppression method.

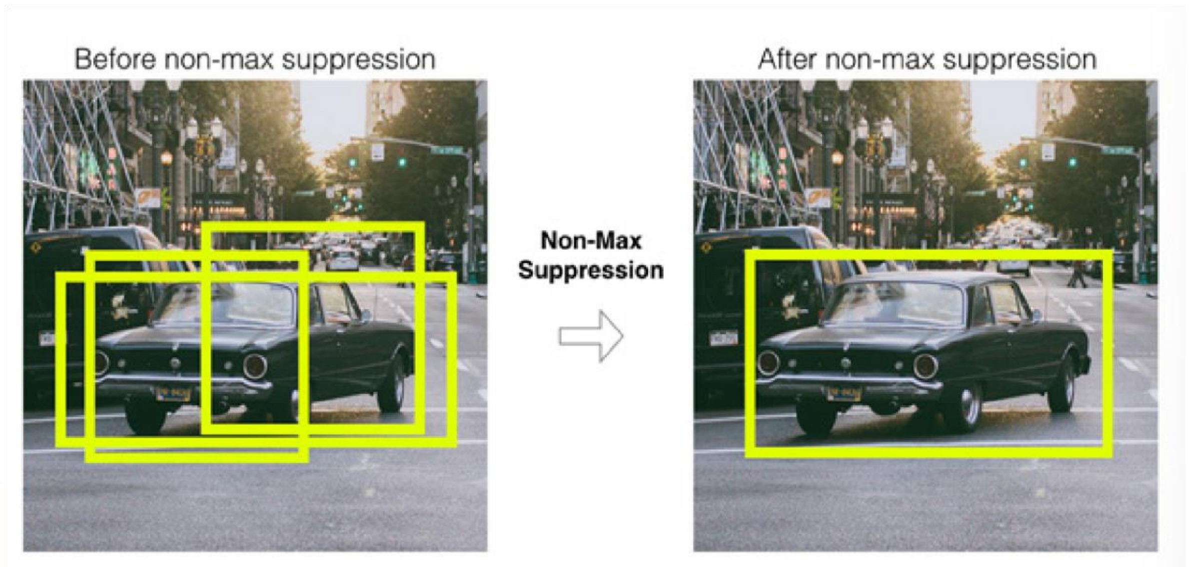


Figure 3.12: Non-max suppression method.

Here will see the time line in Figure 3.13

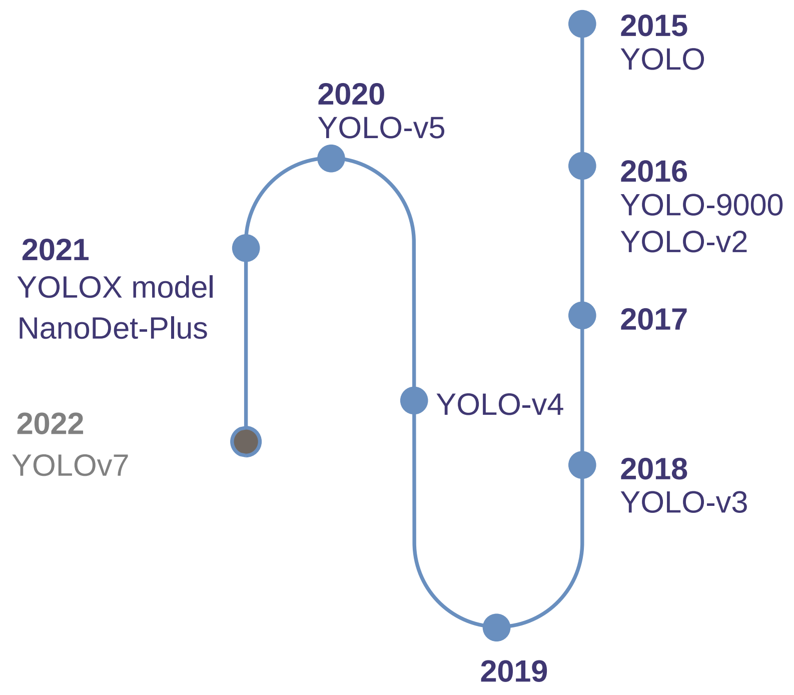


Figure 3.13: The time line for yolo algorithm

After we explain YOLO algorithm in general, now we will explain YOLOv7.

YOLOv7 is the fastest and most accurate real-time object detection model for computer vision tasks. The official YOLOv7 paper named "YOLOv7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors" was released in July 2022 by Chien-Yao Wang, Alexey Bochkovskiy, and Hong-Yuan Mark Liao.[20]

YOLOv7 provides a greatly improved real-time object detection accuracy without increasing the inference costs. As previously shown in the benchmarks, when compared to other known object detectors, YOLOv7 can effectively reduce about 40% parameters and 50% computation of state-of-the-art real-time object detections, and achieve faster inference speed and higher detection accuracy.

In general, YOLOv7 provides a faster and stronger network architecture that provides a more effective feature integration method, more accurate object detection performance, a more robust loss function, and an increased label assignment and model training efficiency.

Figure 3.14 Comparison with other real-time object detectors.

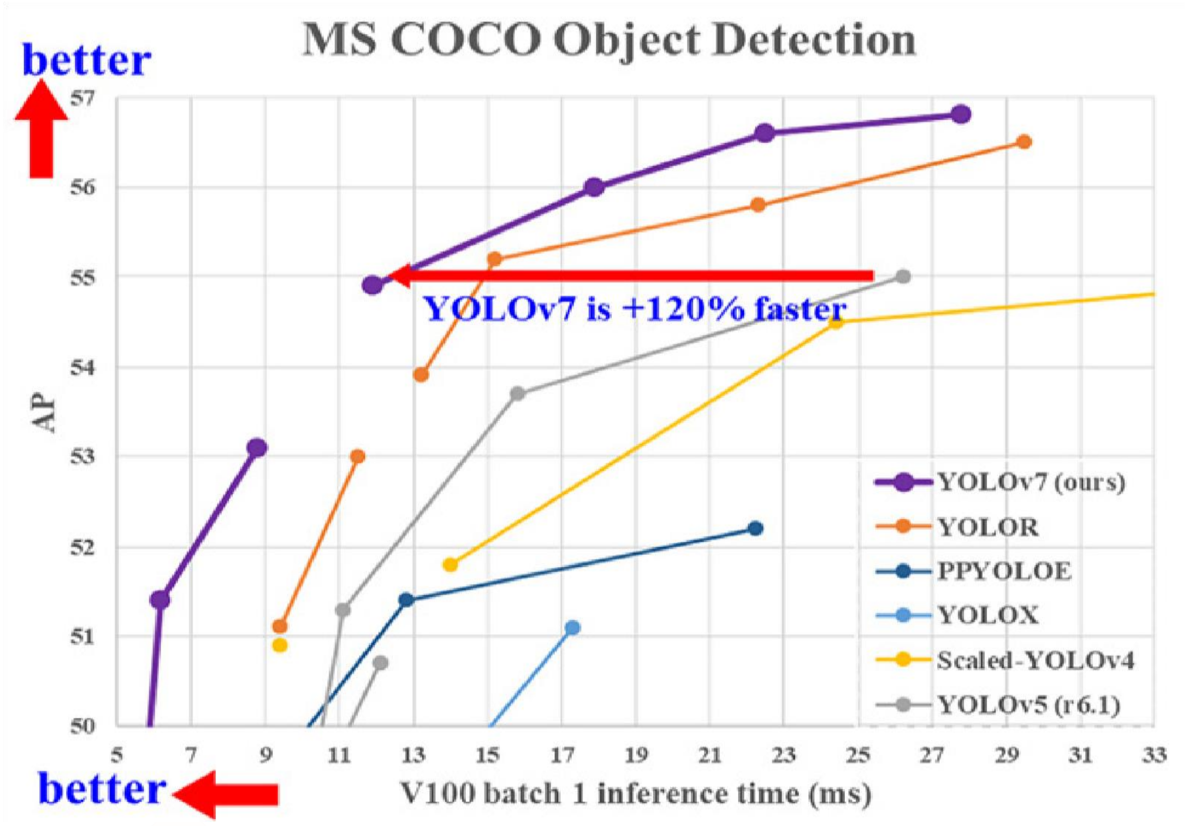


Figure 3.14: Comparison with other real-time object detectors: YOLOv7 achieves state-of-the-art (SOTA) performance.

Model	Param	FLOPs	Size	FPS	APtest	APvala
YOLOX-X	99.1M	281.9G	640	58	51.5%	51.1%
PPYOLOE-X	98.4M	206.6G	640	45	52.2%	51.9%
YOLOv5-X (r6.1)	86.7M	205.7G	640	83	-	50.7%
YOLOR-CSP-X	96.9M	226.8G	640	87	53.0%	52.7%
YOLOv7-X	71.3M	189.9G	640	114	53.1%	52.9%
YOLOv5-X6(r6.1)	140.7M	839.2G	1280	38	-	55.0%
YOLOR-D6	151.7M	935.6G	1280	34	56.5%	56.1%
YOLOv7-E6E	151.7M	843.2G	1280	36	56.8%	56.8%

Table 3.2 Comparison of the best real-time object detectors from the official YOLOv7 paper.[20]

The YOLOv7 architecture is based on previous YOLO model architectures, namely YOLOv4, Scaled YOLOv4, and YOLO-R.

The computational block in the YOLOv7 backbone is named E-ELAN, standing for Extended Efficient Layer Aggregation Network. The E-ELAN architecture of YOLOv7 enables the model to learn better by using “expand, shuffle, merge cardinality” to achieve the ability to continuously improve the learning ability of the network without destroying the original gradient path.

The authors of the YOLOv7 paper show that it can be further optimized with a compound model scaling approach. Here, width and depth are scaled in coherence for concatenation-based models.

Figure 3.16 shows YOLOv7 compound scaling.

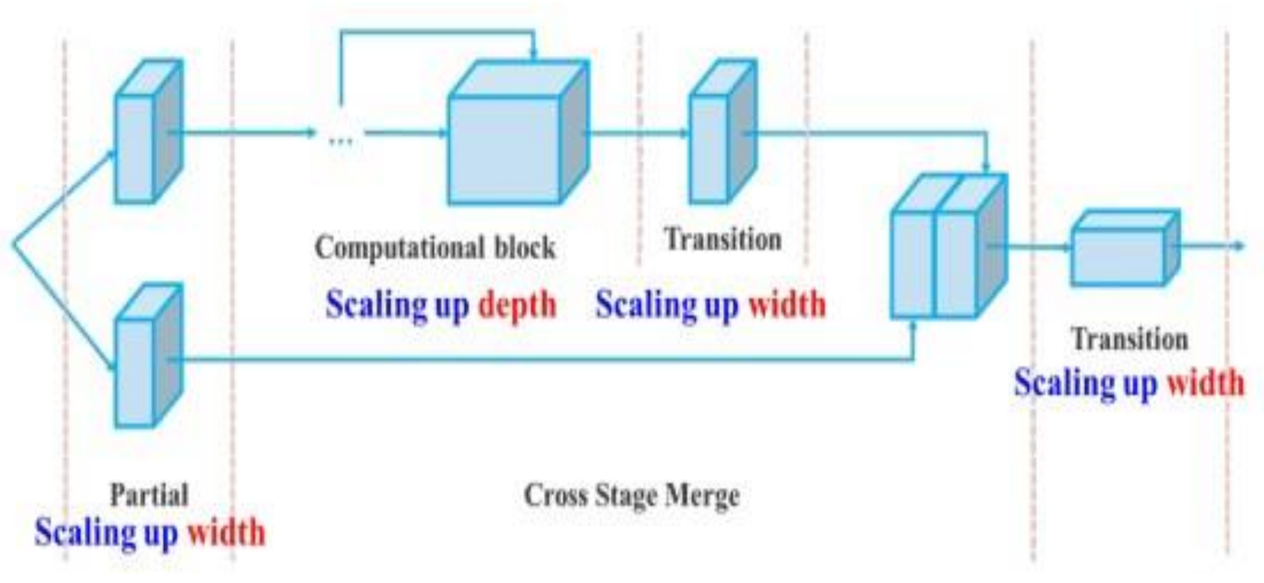


Figure 3.15: YOLOv7 compound scaling

3.3 Project Methodology/ Techniques

1. **Image collection:** The first step is collecting images for the project.
2. **Image labelling:** Labelling objects with bounding boxes and labels on the images for model training.
3. **Image separation:** We have two separated data set one for training data and the other for test set and we will use k-cross validation set.
4. **Model training:** Starting to train the model with the training dataset.
5. **Model testing:** Once the model finished training. In this stage, the test image dataset is used to evaluate the model to output the average precision and mAP.
6. **Model inferencing:** A webcam is set up to provide input to the model for inferencing.
7. **Display result:** Displaying detected objects with bounding boxes and labels.

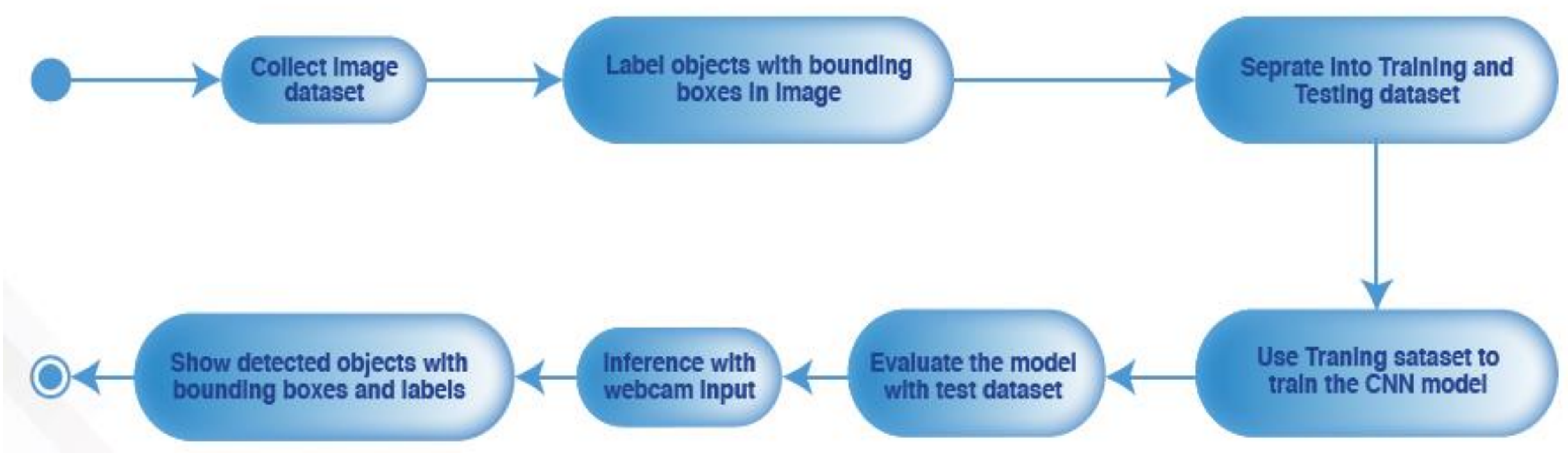

Collect Image dataset
Label objects with bounding boxes in Image
Seprate Into Training and Testing dataset
Use Traning sataset to train the CNN model
Evaluate the model with test dataset
Inference with webcam Input
Show detected objects with bounding boxes and labels

Figure 3.16: System design flowchart.

CHAPTER FOUR

Project Design& Implementation

- Hardware:

  A. 64-bit, x86 desktops or laptops with dedicated Nvidia graphics card. We used a 3070 RTX Nvidia graphics card.

  B. Alternatively, use Google Colaboratory.

  C. A toy gun, knife, and pistol.

- Software:

  A. PyTorch with GPU support (CUDA with PyTorch)

  B. MakeSense website for labeling images with bounding boxes

  C. PyCharm.

  D. Python 3.10.

- Resources:

  A. Images dataset

     1. 5000 images for knife
     2. 5000 images for gun
     3. 5000 images for pistol

- Schedule

  Table 4.1 shows the progress of this project.

	Jul 1, 2022				Aug 1, 2022				Sep 1, 2022				Oct 1, 2022				Nov 1, 2022				Dec 1, 2022			
TASK																								
Reserch topic phase1	🟦	🟦																						
Reserch topic phase2		🟦	🟦																					
Requirement Analysis			🟦	🟦																				
Design System					🟥	🟥																		
Install required softwares							🟥																	
prepare project proposal							🟥	🟥																
prepare Presentation									🟥															
Prepare image set										🟥	🟥	🟥	🟥	🟥										
Install PyTorch and YOLOV7																🟥								
Training the model																		🟩						
Writing all final documents																					🟩	🟩		

Table 4.1: Project Schedule

4.1.2 Image collection

In the image collection step of system design, image collection is required to get a sufficient number of images for training and testing purposes.

The number of images for both training and testing is shown in Table 4.2.

Number of images	Knife	Gun	Pistol
Training	3,950	3,950	3,950
Validation	459	459	459
Test	494	494	494

Table 4.2: Number of images for each class

4.1.3 Image labelling

In the next step image labeling, a website and software called "Make sense" it's Open source and free to use under GPLv3 license. it is used to draw a bounding box and give a label on each image, then save the details of that into a different type of format like JSON, XML, CSV and what we use TXT yolo format, figure 4.1 shows the Make Sense annotation demonstration.

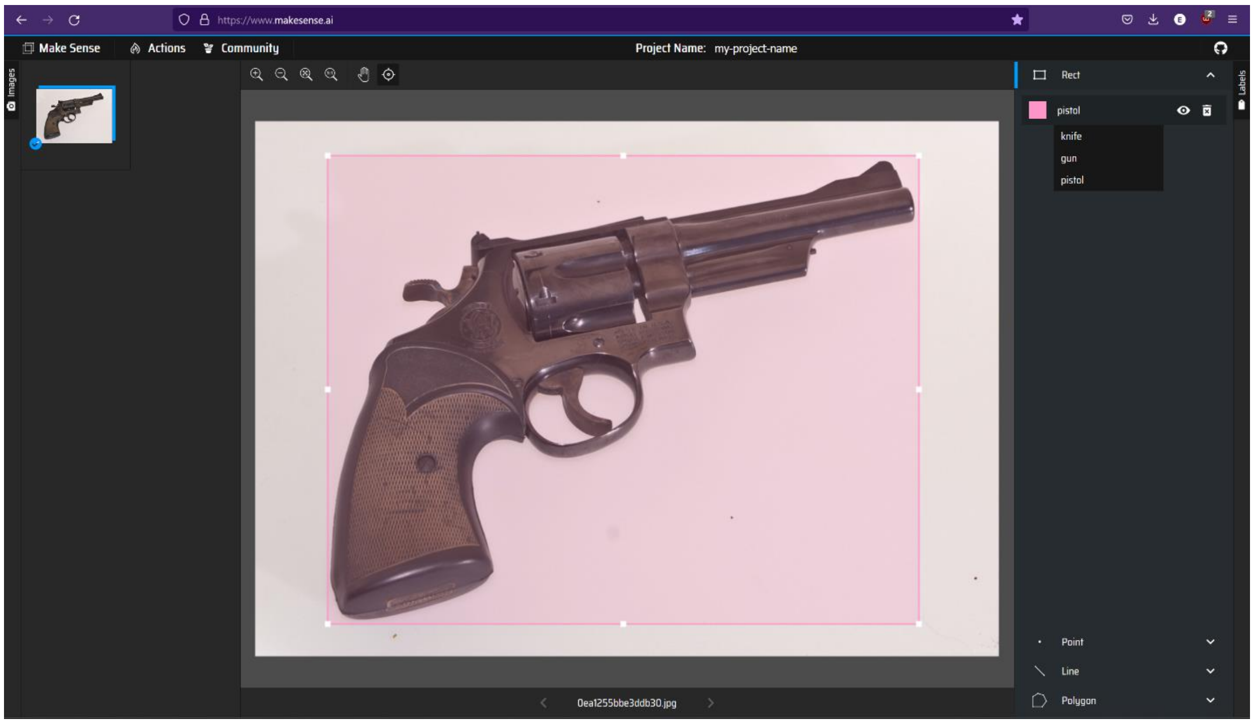
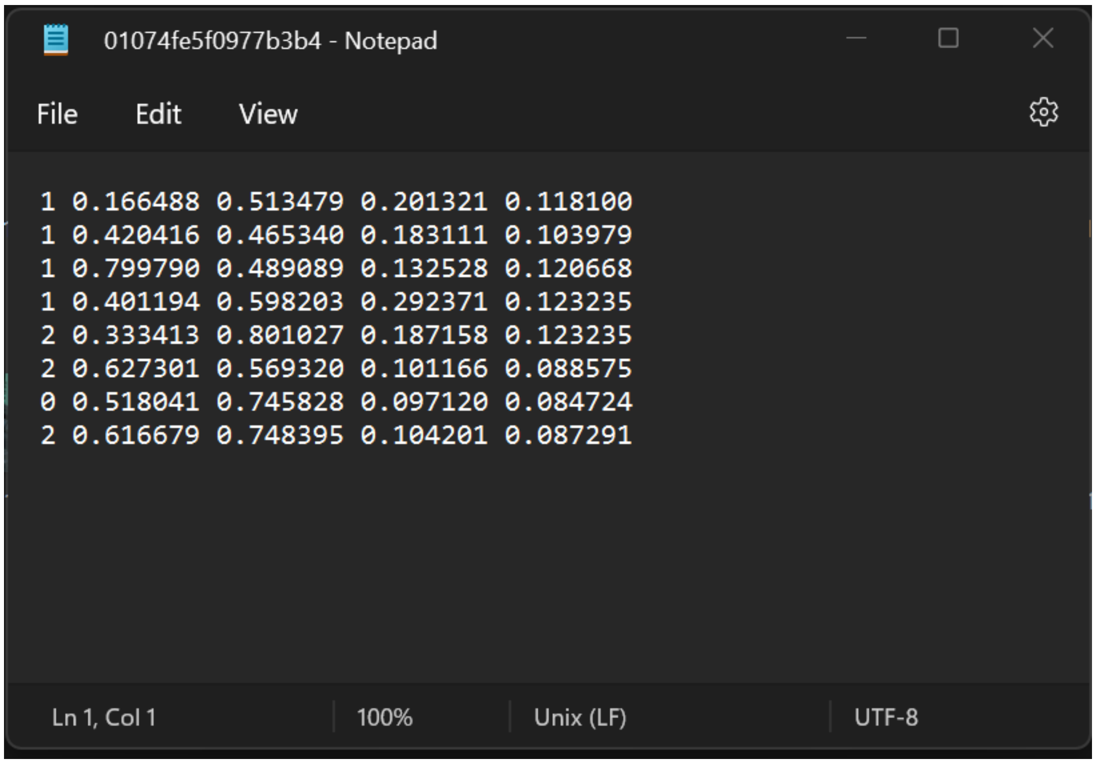


Figure 4.1: Make Sense demonstration

Figure 4.2 shows an example of the TXT yolo format.



```
1 0.166488 0.513479 0.201321 0.118100
1 0.420416 0.465340 0.183111 0.103979
1 0.799790 0.489089 0.132528 0.120668
1 0.401194 0.598203 0.292371 0.123235
2 0.333413 0.801027 0.187158 0.123235
2 0.627301 0.569320 0.101166 0.088575
0 0.518041 0.745828 0.097120 0.084724
2 0.616679 0.748395 0.104201 0.087291
```

Figure 4.2: Example of TXT YOLO format.

4.1.4 Image separation

There is no regular rule on how much you should divide the image dataset into training and testing, normally the common ratios are 80/20, 90/10, or 70/30 for training and testing datasets. By reading resources online, a common rule is applied that the bigger the dataset is, the smaller the testing dataset should be. the partition ratio chosen for the dataset is 75/12/13 (training /validation/test) sets and this could be done by using a python script.

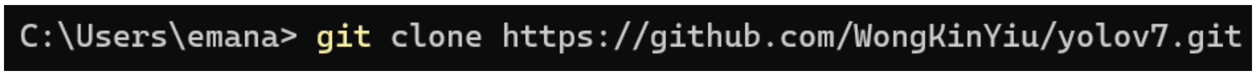
4.2 project Implementation

4.1.2 Model training

Once the dataset is well prepared, the dataset is then fed into the model to start training, PyTorch, YOLO object detection, and PyCharm are tools chosen to implement in this research. To start training, the YOLO algorithm is installed into PyCharm first, and then install the required library by the following code:

1) **Install YOLO V7**:

We have to clone the URL to the terminal and use the Git function as shown in figure 4.3:

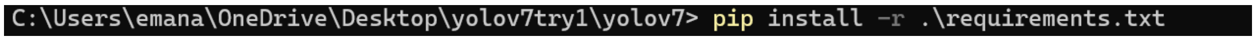


```
C:\Users\emana> git clone https://github.com/WongKinYiu/yolov7.git
```

Figure 4.3: Install YOLOv7

When it's finished, we will get the folder containing all the required files for the YOLOv7 algorithm.

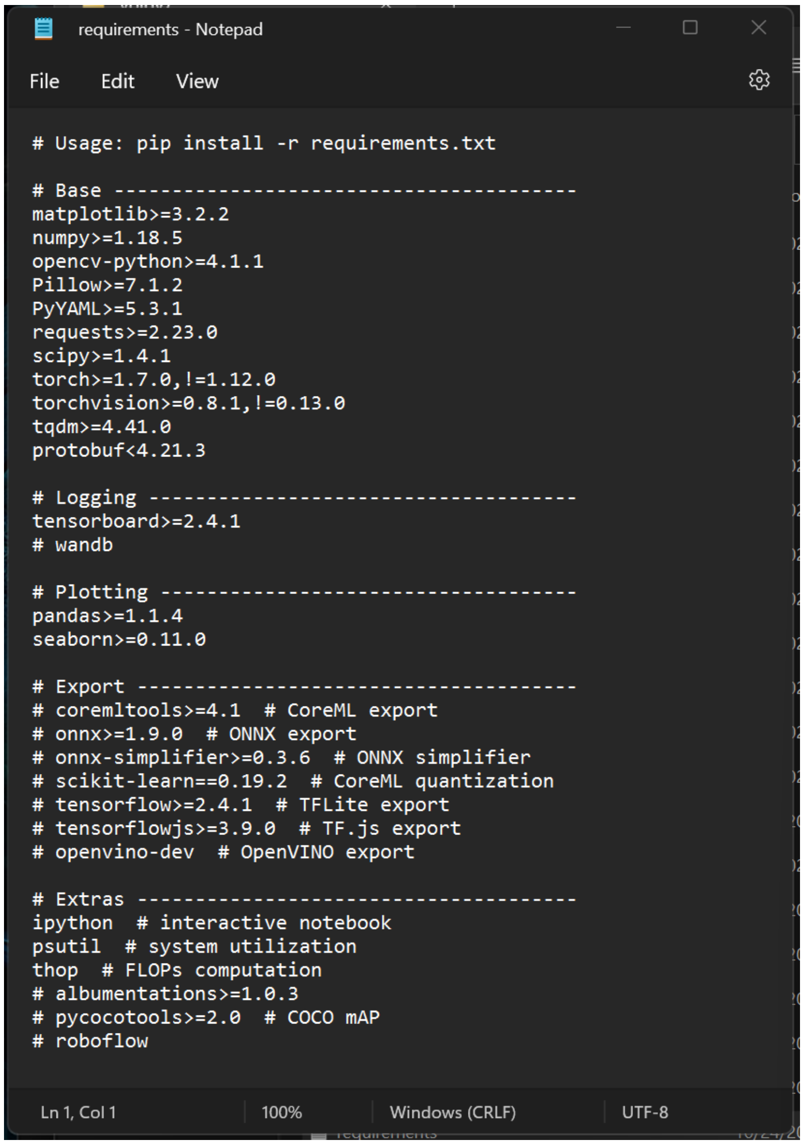
After we install YOLOv7, now we have to install the required library from the requirement.txt folder as in figure 4.4:



```
C:\Users\emana\OneDrive\Desktop\yolov7try1\yolov7> pip install -r .\requirements.txt
```

Figure 4.4: Install the required library (requirement.txt)

Figure 4.5 shows the requirements.txt contains the following libraries:

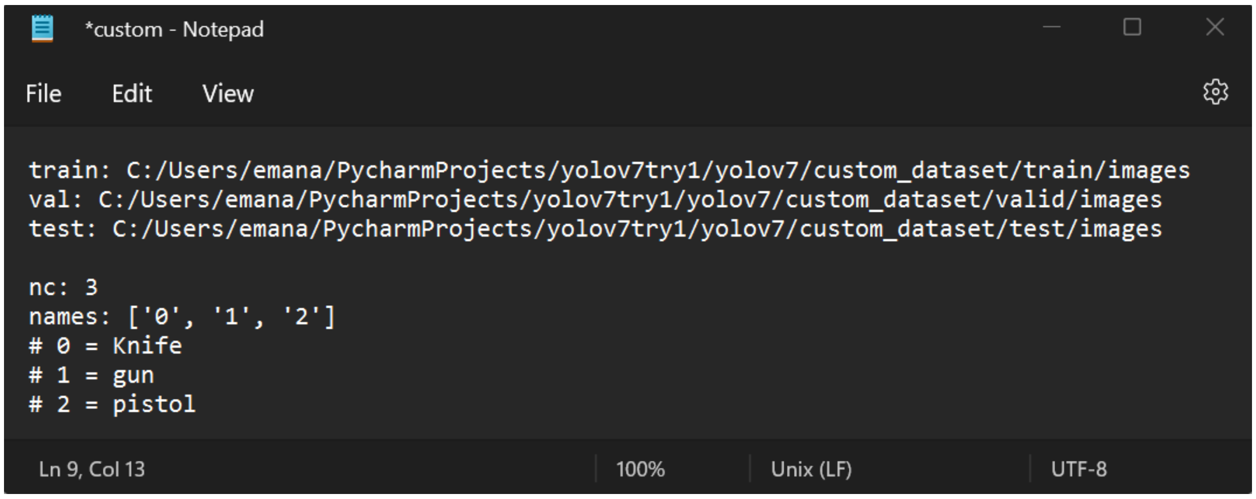


```
# Usage: pip install -r requirements.txt

# Base ----------------------------------------
matplotlib>=3.2.2
numpy>=1.18.5
opencv-python>=4.1.1
Pillow>=7.1.2
PyYAML>=5.3.1
requests>=2.23.0
scipy>=1.4.1
torch>=1.7.0,!=1.12.0
torchvision>=0.8.1,!=0.13.0
tqdm>=4.41.0
protobuf<4.21.3

# Logging -------------------------------------
tensorboard>=2.4.1
# wandb

# Plotting ------------------------------------
pandas>=1.1.4
seaborn>=0.11.0

# Export --------------------------------------
# coremltools>=4.1  # CoreML export
# onnx>=1.9.0  # ONNX export
# onnx-simplifier>=0.3.6  # ONNX simplifier
# scikit-learn==0.19.2  # CoreML quantization
# tensorflow>=2.4.1  # TFLite export
# tensorflowjs>=3.9.0  # TF.js export
# openvino-dev  # OpenVINO export

# Extras --------------------------------------
ipython  # interactive notebook
psutil  # system utilization
thop  # FLOPs computation
# albumentations>=1.0.3
# pycocotools>=2.0  # COCO mAP
# roboflow
```

Figure 4.5: requirement.txt

To run the model, a configuration file cfg is required, this file contains the information about the model such as model type, feature extractor type, adjustable parameters for training and testing, and post-processing score converter such Initially configuration file usually contains optimal configurations from previous training on a different dataset.

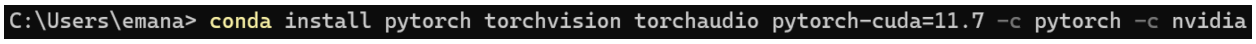
Another necessary file is the "custom. YAML" file, this file basically contains an identification number for each class to train in the model. Figure 4.6 is an example of a YMAL file in this project.



```
train: C:/Users/emana/PycharmProjects/yolov7try1/yolov7/custom_dataset/train/images
val: C:/Users/emana/PycharmProjects/yolov7try1/yolov7/custom_dataset/valid/images
test: C:/Users/emana/PycharmProjects/yolov7try1/yolov7/custom_dataset/test/images

nc: 3
names: ['0', '1', '2']
# 0 = Knife
# 1 = gun
# 2 = pistol
```

Figure 4.6: Example of a YAML file.

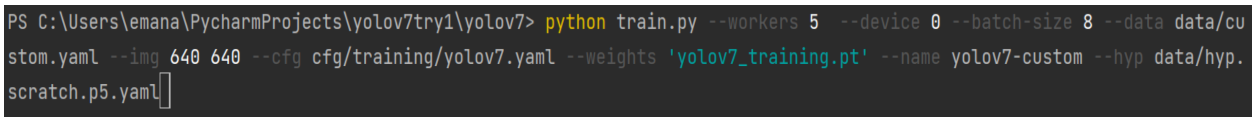
Now YOLO is required to use CUDA with PyTorch to run the code in GPU (Parallel) as in figure 4.7:



```
C:\Users\emana> conda install pytorch torchvision torchaudio pytorch-cuda=11.7 -c pytorch -c nvidia
```

Figure 4.7: CUDA with PyTorch to run the code in GPU (Parallel).

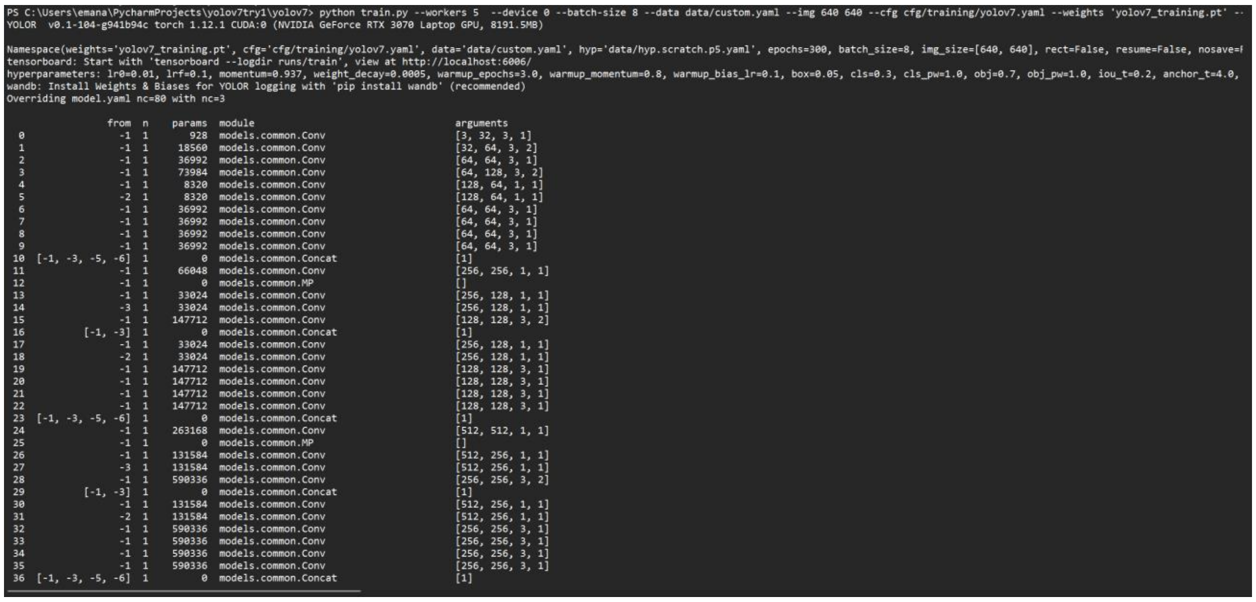
Once all the necessary documents and images are prepared, the next step is to run a command to run "train.py" in the object detection folder. this script will then load all the necessary files that have been installed to the PyCharm folder and the necessary library such as PyTorch to start training. Figure 4.8, demonstrates the training process on PyCharm:



```
PS C:\Users\emana\PycharmProjects\yolov7try1\yolov7> python train.py --workers 5  --device 0 --batch-size 8 --data data/cu
stom.yaml --img 640 640 --cfg cfg/training/yolov7.yaml --weights 'yolov7_training.pt' --name yolov7-custom --hyp data/hyp.
scratch.p5.yaml
```

Figure 4.8: training process on PyCharm.

Figure 4.9 shows the start of the training progress.

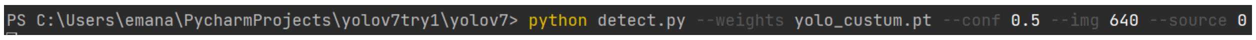


```
PS C:\Users\emana\PycharmProjects\yolov7try1\yolov7> python train.py --workers 5  --device 0 --batch-size 8 --data data/custom.yaml --img 640 640 --cfg cfg/training/yolov7.yaml --weights 'yolov7_training.pt' --
YOLOR  v0.1-104-g941b94c torch 1.12.1 CUDA:0 (NVIDIA GeForce RTX 3070 Laptop GPU, 8191.5MB)

Namespace(weights='yolov7_training.pt', cfg='cfg/training/yolov7.yaml', data='data/custom.yaml', hyp='data/hyp.scratch.p5.yaml', epochs=300, batch_size=8, img_size=[640, 640], rect=False, resume=False, nosave=F
tensorboard: Start with 'tensorboard --logdir runs/train', view at http://localhost:6006/
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=0.3, cls_pw=1.0, obj=0.7, obj_pw=1.0, iou_t=0.2, anchor_t=4.0,
wandb: Install Weights & Biases for YOLOR logging with 'pip install wandb' (recommended)
Overriding model.yaml nc=80 with nc=3

                 from  n    params  module                                  arguments
  0                -1  1       928  models.common.Conv                      [3, 32, 3, 1]
  1                -1  1     18560  models.common.Conv                      [32, 64, 3, 2]
  2                -1  1     36992  models.common.Conv                      [64, 64, 3, 1]
  3                -1  1     73984  models.common.Conv                      [64, 128, 3, 2]
  4                -1  1      8320  models.common.Conv                      [128, 64, 1, 1]
  5                -2  1      8320  models.common.Conv                      [128, 64, 1, 1]
  6                -1  1     36992  models.common.Conv                      [64, 64, 3, 1]
  7                -1  1     36992  models.common.Conv                      [64, 64, 3, 1]
  8                -1  1     36992  models.common.Conv                      [64, 64, 3, 1]
  9                -1  1     36992  models.common.Conv                      [64, 64, 3, 1]
 10  [-1, -3, -5, -6]  1         0  models.common.Concat                    [1]
 11                -1  1     66048  models.common.Conv                      [256, 256, 1, 1]
 12                -1  1         0  models.common.MP                        []
 13                -1  1     33024  models.common.Conv                      [256, 128, 1, 1]
 14                -3  1     33024  models.common.Conv                      [256, 128, 1, 1]
 15                -1  1    147712  models.common.Conv                      [128, 128, 3, 2]
 16          [-1, -3]  1         0  models.common.Concat                    [1]
 17                -1  1     33024  models.common.Conv                      [256, 128, 1, 1]
 18                -2  1     33024  models.common.Conv                      [256, 128, 1, 1]
 19                -1  1    147712  models.common.Conv                      [128, 128, 3, 1]
 20                -1  1    147712  models.common.Conv                      [128, 128, 3, 1]
 21                -1  1    147712  models.common.Conv                      [128, 128, 3, 1]
 22                -1  1    147712  models.common.Conv                      [128, 128, 3, 1]
 23  [-1, -3, -5, -6]  1         0  models.common.Concat                    [1]
 24                -1  1    263168  models.common.Conv                      [512, 512, 1, 1]
 25                -1  1         0  models.common.MP                        []
 26                -1  1    131584  models.common.Conv                      [512, 256, 1, 1]
 27                -3  1    131584  models.common.Conv                      [512, 256, 1, 1]
 28                -1  1    590336  models.common.Conv                      [256, 256, 3, 2]
 29          [-1, -3]  1         0  models.common.Concat                    [1]
 30                -1  1    131584  models.common.Conv                      [512, 256, 1, 1]
 31                -2  1    131584  models.common.Conv                      [512, 256, 1, 1]
 32                -1  1    590336  models.common.Conv                      [256, 256, 3, 1]
 33                -1  1    590336  models.common.Conv                      [256, 256, 3, 1]
 34                -1  1    590336  models.common.Conv                      [256, 256, 3, 1]
 35                -1  1    590336  models.common.Conv                      [256, 256, 3, 1]
 36  [-1, -3, -5, -6]  1         0  models.common.Concat                    [1]
```

Figure 4.9: Start of the training process.

4.2.2 Model Inferencing

In this stage, the trained model will be tested on a new input dataset taken from a webcam to check whether the model does what it is designed for. We used the “detect.py” with source = 0 to denote to webcam and the weight result from training “yolo_custum.py”. Figure 4.10 shows testing using a laptop webcam.



```
PS C:\Users\emana\PycharmProjects\yolov7try1\yolov7> python detect.py --weights yolo_custum.pt --conf 0.5 --img 640 --source 0
```

Figure 4.10: Test using a real-time webcam.

4.2.3 internal Design

After the model is sufficiently trained and exported as an inference graph, the internal design of the system is illustrated in Figure 4.11, Firstly, the system loads the trained model and activates the webcam, and then takes input data from the webcam which is passed to the trained model. Then the models will make a prediction on how likely the image is a gun, knife, or pistol after it processed the input data from the webcam. Finally, the system displays the output of the objects detected with predicted bounding boxes, labels, and percentage of accuracy on the webcam video.

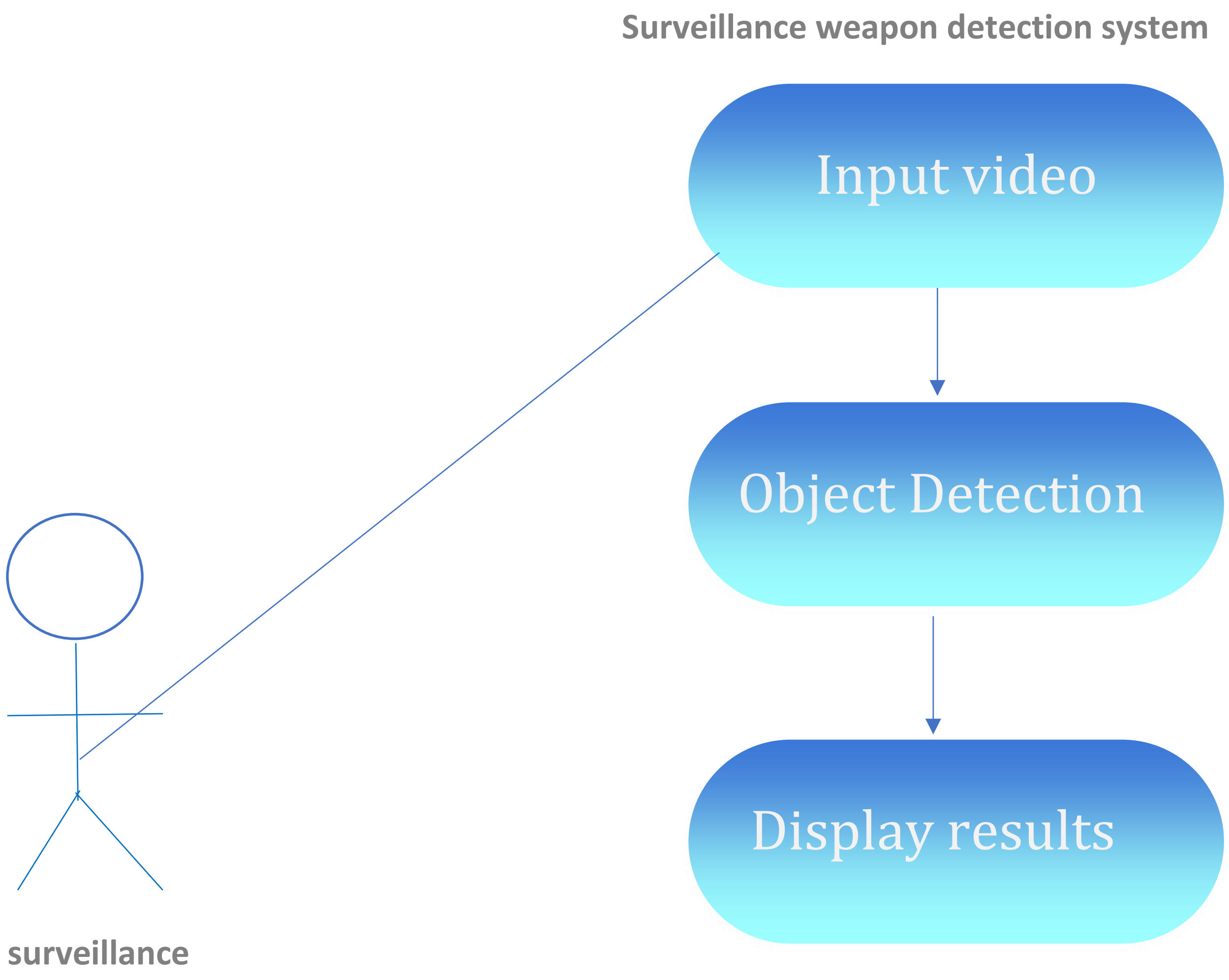


Figure 4.11: Use case of the system.

CHAPTER FIVE

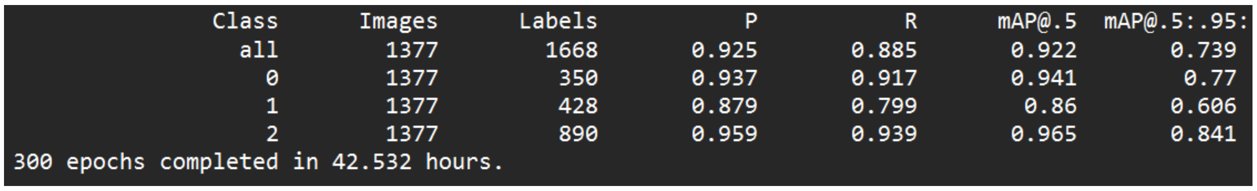
RESULTS

5.1 Introduction

In this chapter, the results obtained from the training and testing process of the trained model YOLOv7 are presented as well as the inference result from the trained model.

5.1.1 Training Result

During the process of training, the total Precision, Recall, mAP@5, mAP@0.5:0.95: and F1 for each global step is recorded and saved in event files, The training time taken for this research is completed in 42.532 hours. figure 5.1 shows the result of the training.



```
                 Class      Images      Labels           P           R      mAP@.5  mAP@.5:.95:
                   all        1377        1668       0.925       0.885       0.922       0.739
                     0        1377         350       0.937       0.917       0.941        0.77
                     1        1377         428       0.879       0.799        0.86       0.606
                     2        1377         890       0.959       0.939       0.965       0.841
300 epochs completed in 42.532 hours.
```

Figure 5.1: Training Result.

A) The confusion matrix figure is shown below:

Basically, a confusion matrix is a summary of predictions, it evaluates the performance of our classification model, as in figure 5.2 the accuracy of class 0 (knife) is 0.97, class 1 (gun) is 0.85, and finally, class 2(pistol) is 0.95.

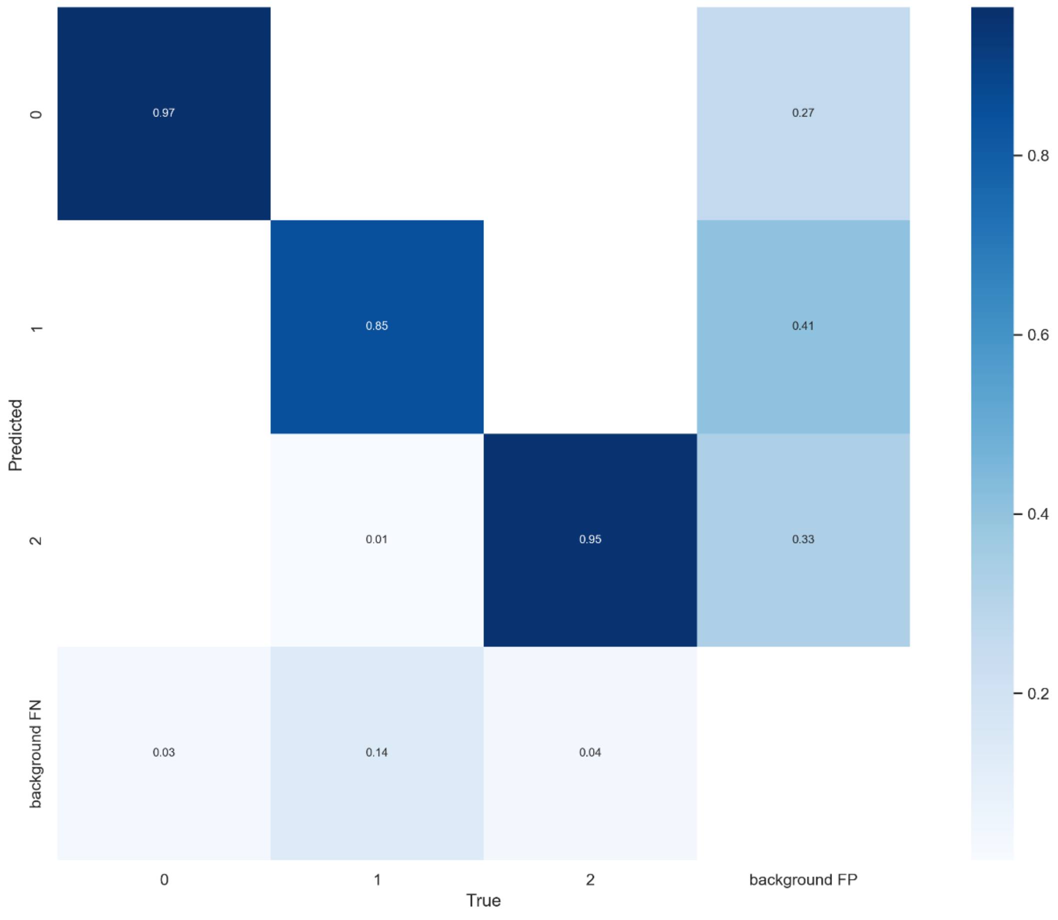


Figure 5.2: Confusion matrix

B) F1 curve figure 5.3 is shown below:

From the F1 curve, the confidence value that optimizes the precision and recall is 0.443. In many cases a higher confidence value is desirable. In the case of this model, it may be optimal to select a confidence of 0.80 since the F1 value appears to be about 0.85, which is not too far off from the maximum value of 0.90. Observing the precision and recall values at a confidence of 0.8 also confirms that this may be a suitable design point. Starting at about 0.8, the recall value begins to suffer, and the precision value is still roughly at the maximum value.

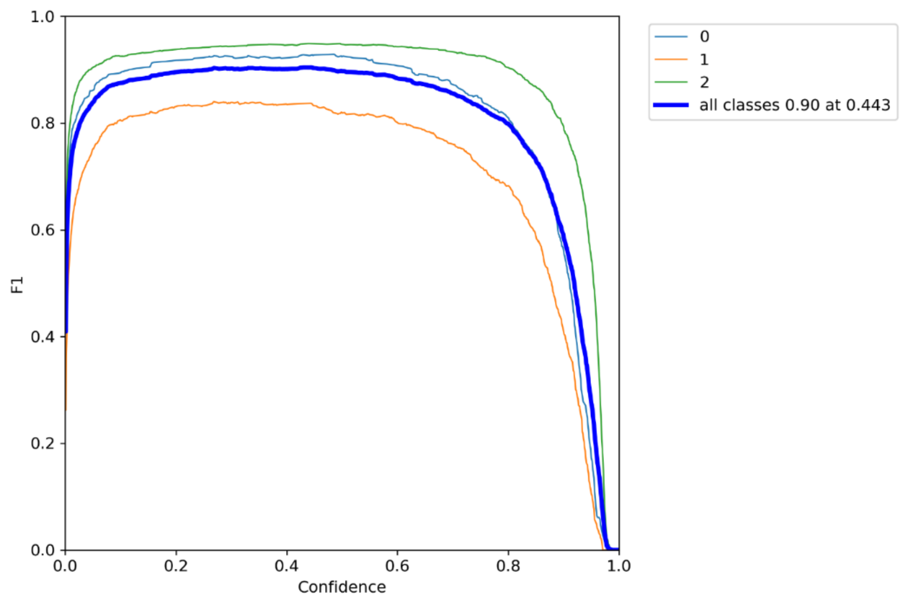


Figure 5.3: F1 Curve.

C) Precision figure 5.4 is shown below:

Precision is a measure of the accuracy of a classifier's predictions, specifically the number of true positive predictions made by the classifier out of the total number of positive predictions. This precision curve shows that when confidence is equal to 0.98 the accuracy is equal to 1.

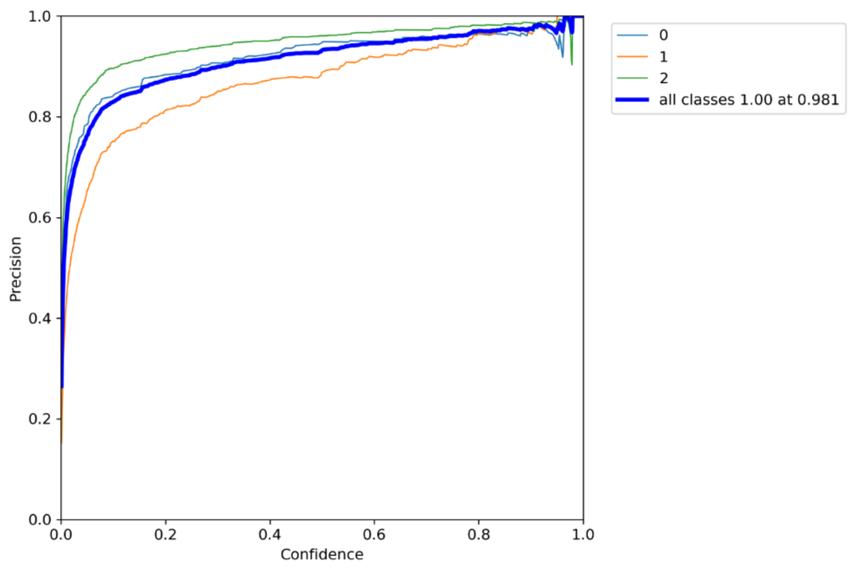


Figure 5.4: Precision Curve.

D) Recall figure 5.5 is shown below:

A recall is defined as the number of true positive predictions divided by the total number of positive samples in the dataset. In this recall curve, we can see when Confidence is equal to 0 the accuracy is equal to 97%.

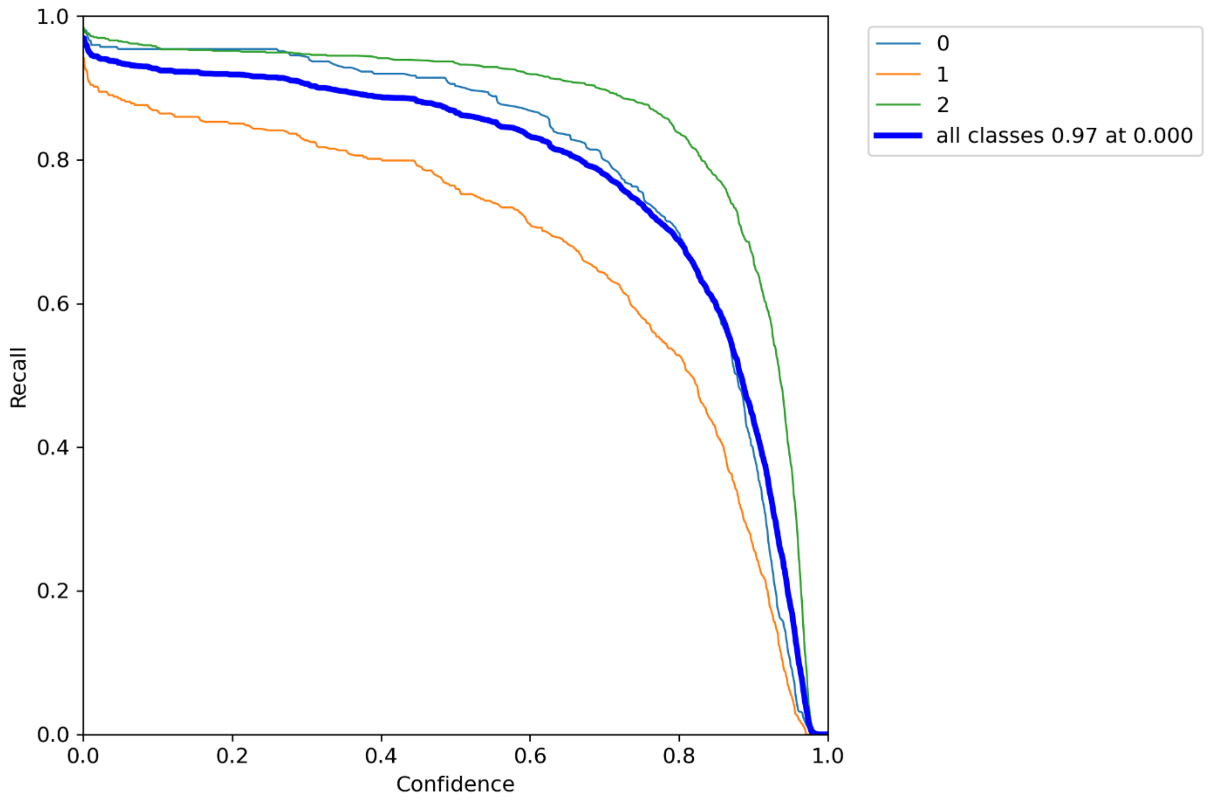


Figure 5.5: Recall Curve.

E) Precision-Recall curves figure 5.6 is shown below:

In the Precision-Recall curve, when the classification threshold is lowered, the recall of the model increases, but the precision decreases. Here we trade-off between them and chose the threshold at map@0.5 given an accuracy equal to 92%.

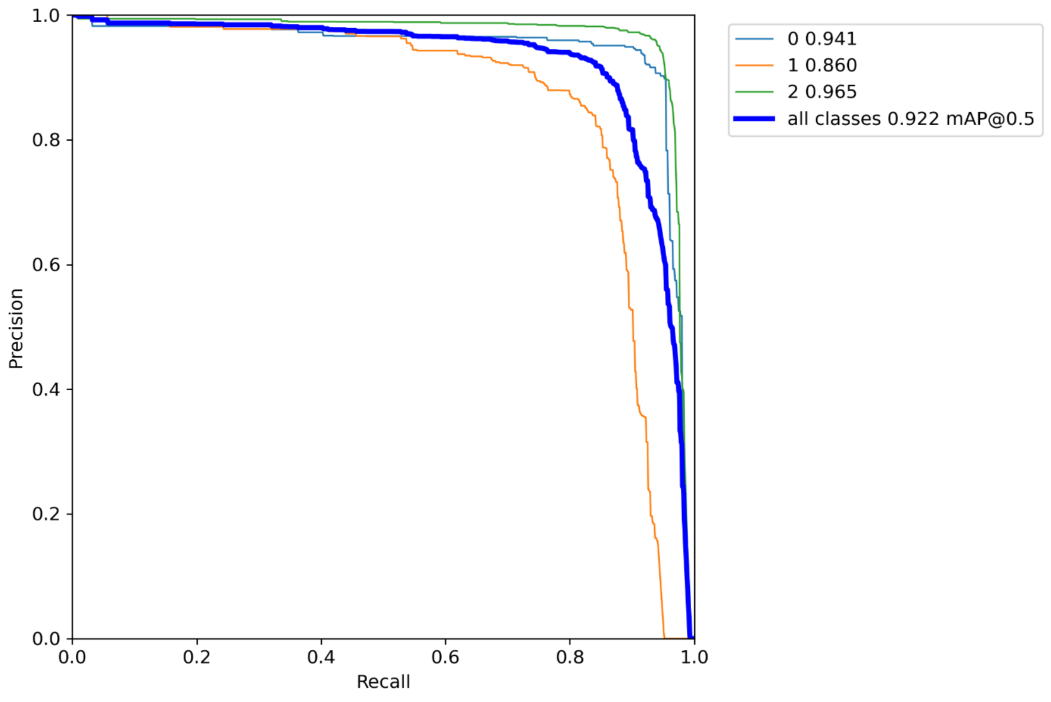


Figure 5.6: Precision-Recall Curve.

F) The training curves in figure 5.7 are shown below:
These plots show how the performance of a machine learning model changes as it is trained on more data. They can be used to diagnose learning problems and to choose an appropriate model complexity.

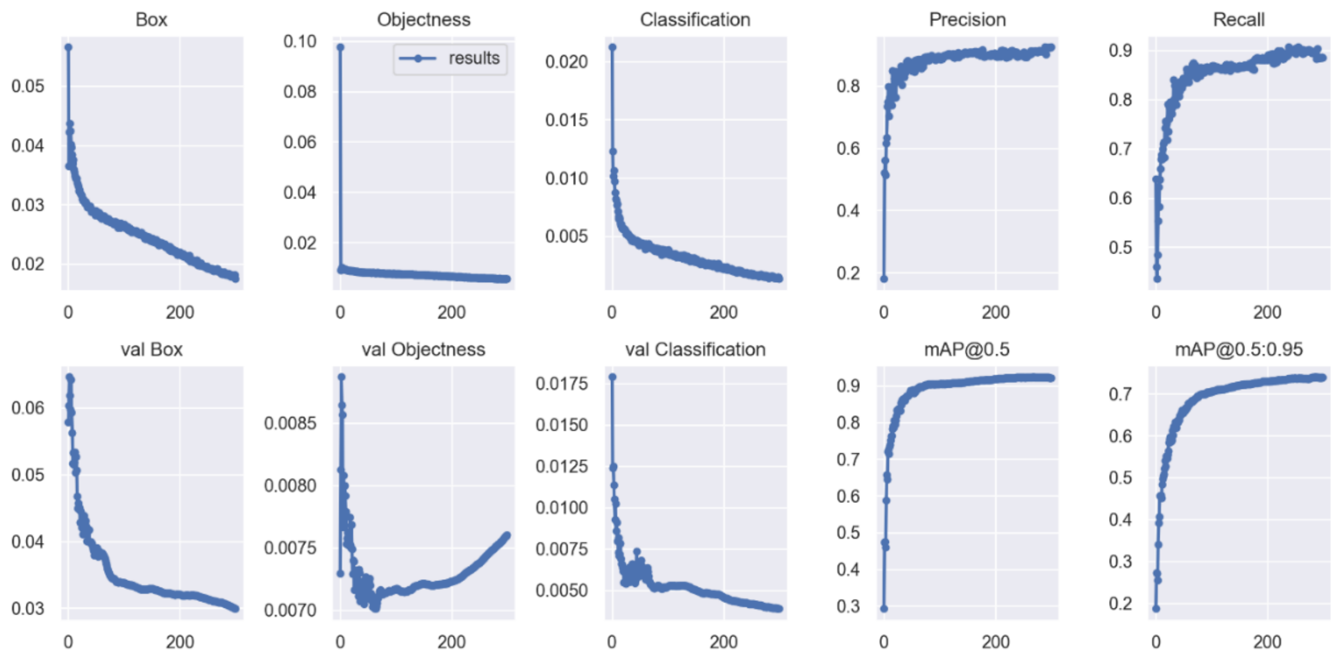


Figure 5.7: Training Curve.

And here we can see some labeled data before and after training in figure 5.8 and figure 5.9:

1) original data:

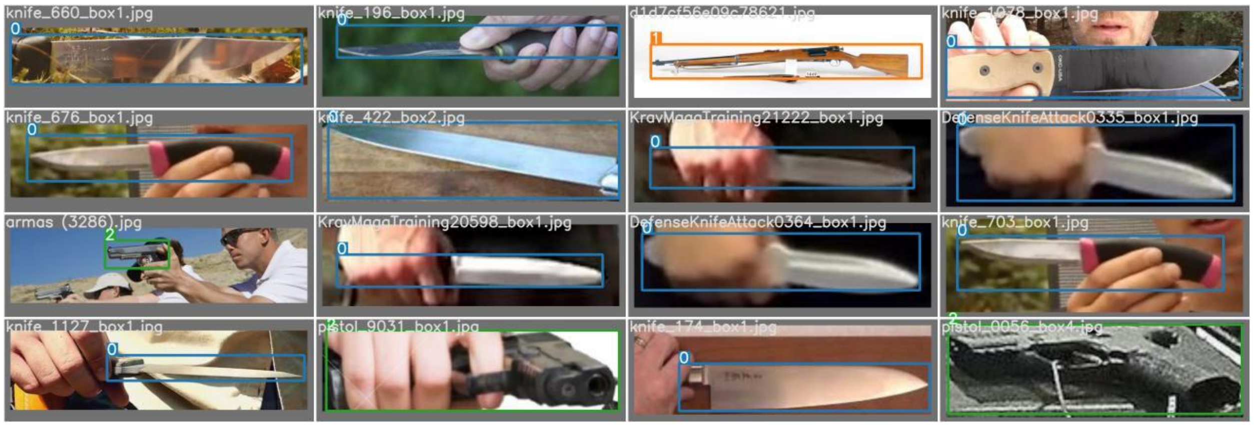


Figure 5.8: Original data.

2) Predicted data:

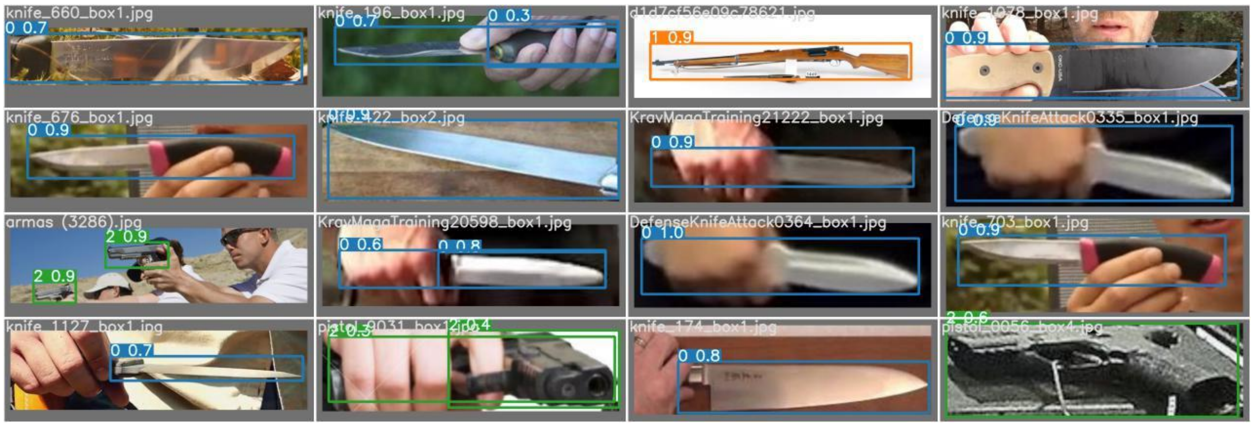
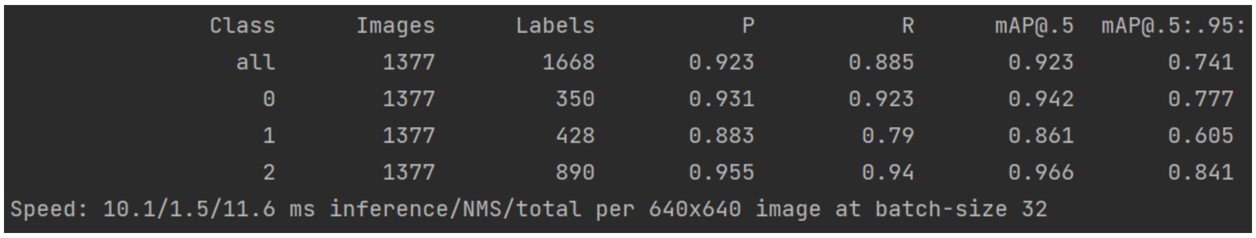


Figure 5.9: Predicted data.

5.1.2 Testing Results

The output of the testing results was obtained after the testing process finished by processing the 1,482 test images of all classes. The time taken for the testing process is less than one minute for 1,482 test images. Figure 5.10 shows the test result.

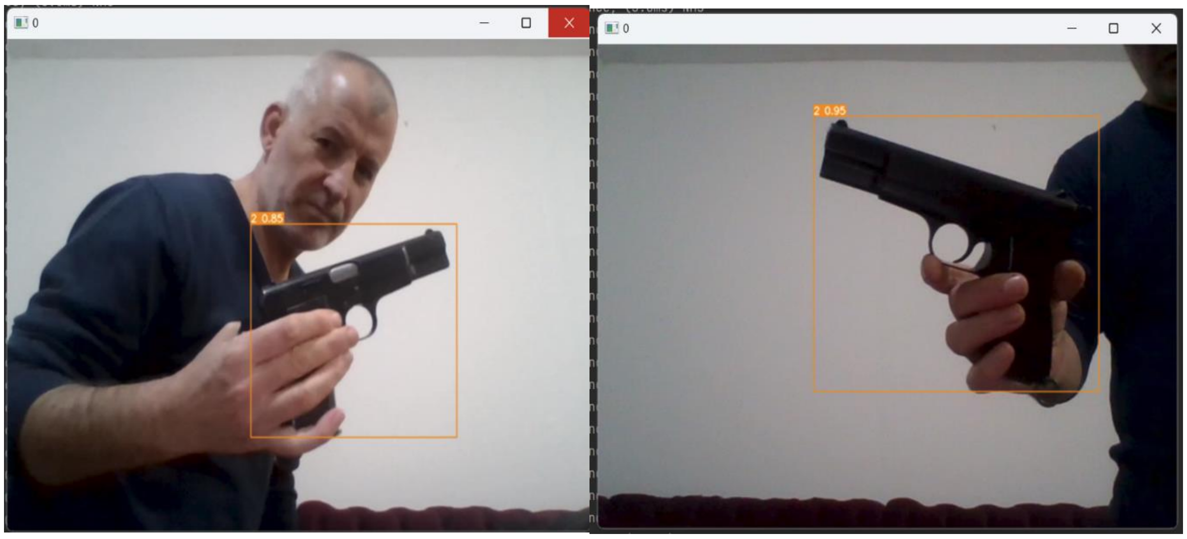


```
              Class      Images      Labels           P           R      mAP@.5  mAP@.5:.95:
                all        1377        1668       0.923       0.885       0.923       0.741
                  0        1377         350       0.931       0.923       0.942       0.777
                  1        1377         428       0.883        0.79       0.861       0.605
                  2        1377         890       0.955        0.94       0.966       0.841
Speed: 10.1/1.5/11.6 ms inference/NMS/total per 640x640 image at batch-size 32
```

Figure 5.10: Testing Results.

5.1.3 Inferencing Results

In this step, the trained YOLO-V7 model will be used to perform inferencing on the input video from a webcam frame by frame. The following figures are showing the inferencing results from weapon detection taken from the webcam. Figures 5.11 show the case of true positive (TP) for the pistol, The Inception model correctly detected the pistol as a pistol with an accuracy of 85%, and 95% accuracy.



(a) (b)

Figure 5.11: Results from weapon detection take from the webcam (a,b).

CHAPTER SIX

CONCLUSION AND FUTURE WORK

6.1 Conclusion

This project introduced weapon detection using a neural network in a surveillance camera, it’s one of the future solutions to our security. For the goal of autonomous weapon detection, accuracy is important. This project has shown that Yolov7 gets with mAP@.5 metric accuracy of 92% over all classes, which consider a high accuracy. However, with more categorized images in each class and more tuning, a higher accuracy could be achieved. Lastly, from the exploration of this project, great knowledge has been learned on the tool of neural network.

6.2 Future Work

Possible future works can be expanding the number of images to the dataset for each class and doing a lot of experiments and fine-tuning such as changing the number of epochs, the type of optimizer, and the number of batch sizes, these are possible ways to increase the accuracy.

REFERANCE

[1] jordan times. https://jordantimes.com/opinion/editorial/bearingfirearms-jordan

[2] world population review. https://worldpopulationreview.com/countryrankings/gun-deaths-by-country

[3] el den Mohamed, Mai Kamal, Ahmed Taha, and Hala H. Zayed. «Automatic gun detection approach for video surveillance.» International Journal of Sociotechnology and Knowledge Development (IJSKD) 12,no. 1 (2020): 49-66.

[4] andy Liu Weapon-Detection-Using-Neural-Networks-in-Real-TimeSurveillance-Video

[5] Fernandez-Carrobles, M., Oscar Deniz, and Fernando Maroto. «Gun and knife detection based on faster R-CNN for video surveillance.» In Iberian conference on pattern recognition and image analysis, pp. 441- 452. Springer, Cham, 2019.

[6] González, Jose L. Salazar, Carlos Zaccaro, Juan A. Álvarez-García, Luis M. Soria Morillo, and Fernando Sancho Caparrini. «Real-time gun detection in CCTV: An open problem.» Neural networks 132 (2020): 297-308.

[7] Ingle, Palash Yuvraj, and Young-Gab Kim. «Real-Time Abnormal Object Detection for Video Surveillance in Smart Cities.» Sensors 22, no. 10 (2022): 3862.

[8] Narejo, Sanam, Bishwajeet Pandey, Ciro Rodriguez, and M. Rizwan Anjum. «Weapon detection using YOLO V3 for smart surveillance system.» Mathematical Problems in Engineering 2021 (2021).

[9] Mahajan, Chaitali, and Ashish Jadhav. «Gun Detection: Comparative Analysis using Transfer Learning in Single Stage Detectors.» In 2022 International Conference on Emerging Smart Computing and Informatics (ESCI), pp. 1-5. IEEE, 2022.

[10] Bhatti, Muhammad Tahir, Muhammad Gufran Khan, Masood Aslam, and Muhammad Junaid Fiaz. «Weapon detection in real-time cctv videos using deep learning.» IEEE Access 9 (2021): 34366-34382. [11] Olmos, Roberto, Siham Tabik, and Francisco Herrera. «Automatic handgun detection alarm in videos using deep learning.» Neurocomputing 275 (2018): 66-72.

[12] medium . https://blog.devgenius.io/resnet50-6b42934db431

[13] towardsdatascience. https://towardsdatascience.com/faster-r-cnnfor-object detection-a- technical-summary-474c5b857b46

[14] Girshick, Ross, Jeff Donahue, Trevor Darrell, and Jitendra Malik. «Rich feature hierarchies for accurate object detection and semantic segmentation.» In Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 580-587. 2014.

[15] Girshick, Ross. “Fast r-cnn.” In Proceedings of the IEEE international conference on computer vision, pp. 1440-1448. 2015

[16] Ren, Shaoqing, Kaiming He, Ross Girshick, and Jian Sun. «Faster r-cnn: Towards real-time object detection with region proposal networks.» Advances in neural information processing systems 28 (2015).

[17] Adelson, Edward H., Charles H. Anderson, James R. Bergen, Peter J. Burt, and Joan M. Ogden. «Pyramid methods in image processing.» RCA engineer 29, no. 6 (1984): 33-41.

[18] Lin, Tsung-Yi, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. «Feature pyramid networks for object detection.» In Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 2117-2125. 2017.

[19] Redmon, Joseph, Santosh Divvala, Ross Girshick, and Ali Farhadi. «You only look once: unified, real-time object detection, June 2015.» arXiv preprint arXiv:1506.02640 (2015).

[20] Wang, Chien-Yao, Alexey Bochkovskiy, and Hong-Yuan Mark Liao. «YOLOv7: Trainable bag-of-freebies sets new state-of-the-art for realtime object detectors.» arXiv preprint arXiv:2207.02696 (2022).