

## ABSTRACT

Our project focuses on object detection, which is a crucial component of systems for visual surveillance and security. Due to an increase in crime in ominous places, security is constantly a top issue in all fields. Our study uses you look only once (YOLOv5), single-shot detector (SSD), and region-based convolution neural network (RCNN) methods to detect knives and guns. Three different dataset types are used in the suggested approach. Images in the collection had been manually annotated. The trade-off between accuracy and speed can be used to determine how this method is applied in real life.

# Contents

|                                                 |    |
|-------------------------------------------------|----|
| 1 Introduction                                  | 5  |
| 2 Problem Life Cycle                            | 7  |
| 2.1 Problems Identification                     | 7  |
| 2.2 Objectives                                  | 7  |
| 2.3 Fish Bone Diagram                           | 8  |
| 2.4 End Users                                   | 8  |
| 3 Literature Survey and Motivation              | 11 |
| 4 Proposed System and Requirement Specification | 11 |
| 4.1 Proposed System                             | 11 |
| 4.1.1 Dataset.                                  | 11 |
| 4.1.2 Methodology                               | 12 |
| 4.2 Software Requirements Specification-SRS     | 19 |
| 4.2.1 ML Libraries and their Functionalities    | 19 |
| 4.2.2 Functional Requirements                   | 20 |
| 4.2.3 Non-Functional Requirements               | 21 |
| 4.3 Scope of Project                            | 22 |
| 4.4 Deployment Requirements                     | 22 |
| 4.4.1 Hardware Requirements                     | 22 |
| 4.4.2 Software Requirements                     | 23 |
| 4.5 Project Deliverable                         | 23 |
| 5 Design                                        | 24 |
| 5.1 Data Flow Diagram                           | 24 |
| 5.2 System Architecture                         | 25 |
| 5.3 UML Diagrams                                | 26 |

|                                                                  |    |
|------------------------------------------------------------------|----|
| 5.3.1 Use Case Diagram .....                                     | 27 |
| 5.3.2 Component Diagram .....                                    | 28 |
| 5.3.3 Sequence Diagram .....                                     | 29 |
| 6 Development/Implementation Details .....                       | 30 |
| 6.1 Technology Used .....                                        | 30 |
| 6.1.1 Deep Learning .....                                        | 30 |
| 6.1.2 YOLOv5 (You Only Look Once) .....                          | 30 |
| 6.1.3 R-CNN (Region-based Convolutional Neural<br>Network) ..... | 31 |
| 6.1.4 Single Shot Detector (SSD) .....                           | 32 |
| 6.1.5 Google Colab .....                                         | 33 |
| 6.1.6 Kaggle .....                                               | 34 |
| 6.1.7 Anaconda .....                                             | 34 |
| 6.1.8 Spyder .....                                               | 35 |
| 6.1.9 Python .....                                               | 36 |
| 6.1.10 Streamlit .....                                           | 37 |
| 7 Testing .....                                                  | 38 |
| 7.1 Test Cases .....                                             | 38 |
| 7.1.1 Unit Testing .....                                         | 38 |
| 8 Results and Discussion .....                                   | 41 |
| 9 Conclusion and Future Work .....                               | 53 |
| 9.1 Conclusion .....                                             | 53 |
| 9.2 Future Work .....                                            | 54 |
| 10 References .....                                              | 55 |

## List of Figures

|                                                       |    |
|-------------------------------------------------------|----|
| 2.1 Fish Bone Diagram . . . . .                       | 8  |
| 5.1 Data Flow Diagram . . . . .                       | 24 |
| 5.2 System Architecture 1 . . . . .                   | 25 |
| 5.3 System Architecture 2 . . . . .                   | 26 |
| 5.4 Use Case Diagram . . . . .                        | 27 |
| 5.5 Component Diagram . . . . .                       | 28 |
| 5.6 Sequence Diagram . . . . .                        | 29 |
| 7.1 To check dataset is loaded successfully . . . . . | 39 |
| 7.2 Testing of Code . . . . .                         | 40 |
| 7.3 To check output of testing . . . . .              | 40 |
| 8.1 User Interface . . . . .                          | 41 |

## List of Tables

|                                    |    |
|------------------------------------|----|
| 1. Hardware Requirements. . . . .  | 23 |
| 2. Software Requirements . . . . . | 24 |

# Chapter 1

## Introduction

Assault is a serious crime that can have severe consequences for the perpetrator. It is defined as the intentional use of force or violence against another person without their consent. In many jurisdictions, it is punishable by imprisonment, fines, and other penalties. It is important to take steps to prevent an assault from occurring, particularly in public places where it can be more difficult to escape or get help. Some ways to prevent assault include being aware of your surroundings, avoiding isolated or poorly lit areas, and carrying a personal safety device, such as a whistle or pepper spray. It is also important to know how to get help if you or someone else is being assaulted, such as calling the police or seeking assistance from a bystander.

It is certainly important to notify the police as soon as possible when a crime, such as assault, is occurring or has occurred. This can help to ensure that law enforcement authorities can respond quickly and effectively, and potentially catch the perpetrators before they can escape. In some cases, eyewitness accounts and other evidence can also be crucial in helping the police identify and apprehend suspects. While surveillance cameras can be useful in some cases, they do not always capture all the necessary information, and they require someone to monitor them continuously to be effective. Therefore, it is still important to report crimes as soon as possible and provide as much information as possible to the authorities. Yes, it is important to notify the police as soon as possible if a crime is occurring, especially if it is a serious crime such as assault. This can allow the police to respond quickly and potentially catch the perpetrator before they can flee the scene. However, you should only call the police if it is safe to do so. If you are in immediate danger, it may be best to find a way to escape or get to a safe place before calling for help. security cameras can be an effective tool in helping to identify and catch perpetrators of crime. However, it is important to remember that they are not a replacement for direct law enforcement intervention. It is still crucial to notify the police if a crime is occurring, even if there are security cameras present.

Weapon detection is an important issue for public safety and security. It involves identifying the presence of weapons, such as guns or knives, in a given area or location. The use of deep learning algorithms can be an effective way to develop a weapon detection model, as these algorithms can analyze large amounts of data and identify patterns or features that may not be immediately apparent to humans. By using deep learning techniques,

it may be possible to more accurately and efficiently detect the presence of weapons, helping to prevent violence and crime. It is important to note that weapon detection systems are not fool proof and may have limitations or potential flaws, so they should be used in conjunction with other security measures.

By using deep learning algorithms, it is possible to develop models that can accurately identify weapons in real-time, potentially helping to prevent crime before it occurs. There are several different approaches that can be used for weapon detection, including image and video analysis, audio analysis, and even artificial intelligence-powered drones. By using a combination of these techniques, it is possible to create a comprehensive system that can effectively detect and alert authorities to the presence of weapons in public areas. Yes, detecting weapons can be challenging due to their diverse sizes, shapes, and colors. One approach that can be used for weapon detection is object detection, which involves using feature extraction and machine learning algorithms to identify instances of different classes of objects, including weapons. This can be done using an image or video analysis, where the algorithm is trained on a dataset of images or videos that include weapons and other objects. The algorithm learns to identify the characteristics that distinguish weapons from other objects, and can then be used to detect weapons in new images or videos. Other approaches that can be used for weapon detection include audio analysis, where algorithms are trained to recognize the sounds made by different types of weapons, and drone-based systems, which use artificial intelligence to detect weapons in real-time. By combining these and other techniques, it is possible to create a robust weapon detection system that can help to ensure the safety and security of the public.

The proposed system aims to assist the police in detecting and identifying weapons in a variety of settings, including outdoor scenes. By using machine learning algorithms to analyze visual information, the system can potentially help the police to more efficiently and accurately identify weapons and connect them to other similar scenes. This could be a valuable tool for law enforcement, as it could help them to more quickly and effectively respond to situations involving weapons, and potentially prevent crimes from occurring. It is important to ensure that any weapon detection system is accurate and reliable, as incorrect identifications could have serious consequences.

# Chapter 2

## Problem Life Cycle

### 2.1 Problems Identification

The contemporary environment of fast-rising crime makes it impossible for conventional crime-solving methods to meet expectations. As a result, we have developed a method for predicting crime based on the presence of weapons at the scene.

### 2.2 Objectives

1. To collect the dataset.
2. To train the dataset with proper algorithms.
3. To test the dataset with proper accuracy.
4. To detect the weapon from the given scene.
5. Comparative analysis with various models.

## 2.3 Fish Bone Diagram

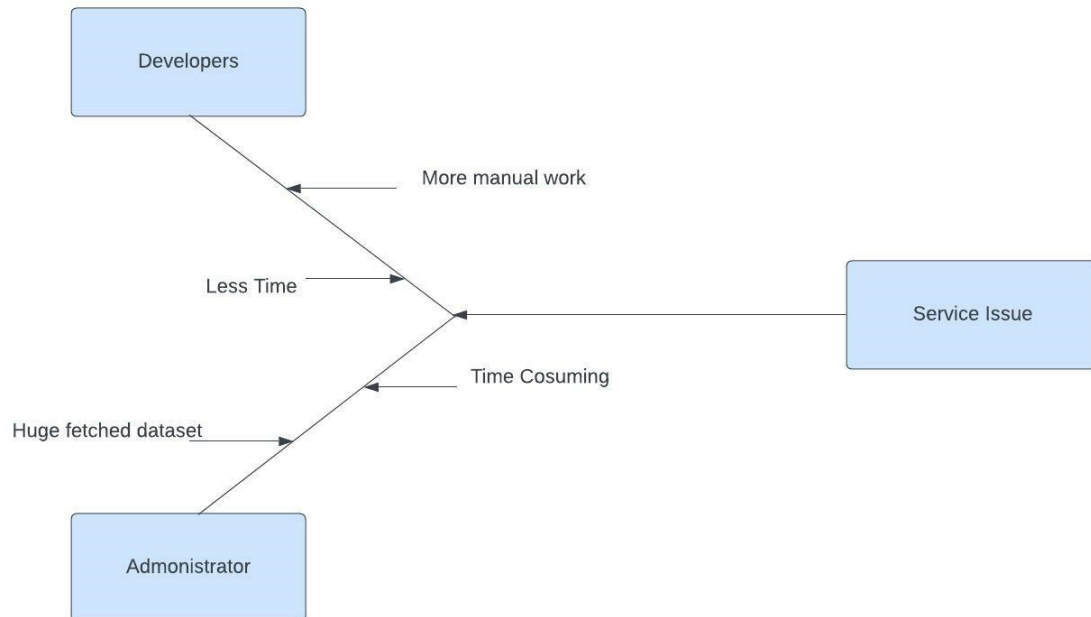


Figure 2.1: Fish Bone Diagram

## 2.4 End Users

- User- Customers
- Admin- Developers



## Chapter 3

# Literature Survey and Motivation

Several weapon detection algorithms have been developed and studied over years. Each of them consists of different types of methods and techniques and has its own pro and con.

The networks are trained in a controlled manner by providing true annotations for object location and class. Once trained, the networks can be used to automatically detect and localize objects in new images. A bounding box is usually represented as a 4-tuple ( $x_{min}$ ,  $y_{min}$ ,  $x_{max}$ ,  $y_{max}$ ). In addition, many works focus on the use of deep learning, such as segmentation. Instance segmentation is the task of object detection and segmentation of the exact pixels that belong to the object. This can be achieved by using a semantic segmentation approach or a fully convolutional network such as Mask R-CNN [14], which is a combination of a region suggestion network and a segmentation network.

Convolutional layers are used to extract features from an input image and form the basis of a convolutional neural network. Convolutional layers are followed by ReLU layers, which are used to introduce nonlinearity in the network. Pooling layers are then used to reduce the spatial size of feature maps and reduce the number of parameters in the network. Finally, fully connected layers are used to generate the output of the network, which is usually the result of classification or regression.

Recently, a number of works have shown that CNNs can also be used as feature extractors for various vision tasks. Razavian et al. [16] proposed an approach to extract CNN features from an off-the-shelf network to perform object recognition. Similarly, Donahue et al. [17] used the activation of convolutional layers to improve object recognition accuracy. In another work, Oquab et al. [18] used pre-trained CNNs for image retrieval tasks.

YOLO is a state-of-the-art object detection system that uses a single convolutional neural network (CNN) to detect multiple objects in an image. It is a one-step process that divides the input image into a set of bounding boxes and predicts the class probabilities of each of these boxes. YOLO is faster than other two-stage object detection systems and can predict objects in real-time. It also has a high degree of accuracy, making it a popular choice for object detection tasks.

The work of Laurie et al. [7] discussed a real-time object detection system that used deep learning to recognize objects in an image. This article focused on the use of convolutional neural networks for object detection, which is the primary technology used in YOLO architecture. Using this technology, the model was able to detect objects in images with high accuracy and low latency. This paper also improved object detection accuracy by using depth-separable convolutional layers that reduced computational model costs. The paper describes the approach taken by the YOLO architecture and its various components, such as the feature extraction layer, the bounding box prediction layer, and the classification layer.

The work of Redmon et al. [8] discussed the architecture of YOLO, which is a real-time object detection system capable of recognizing objects in an image with high accuracy and low latency. This post focused on various components of the YOLO architecture such as a feature extraction layer, a bounding box prediction layer, and a classification layer. This article also discusses the use of convolutional neural networks for object detection. The aim of the project is to investigate the use of Transfer Learning using Convolutional Neural Networks (CNN) to improve the accuracy of an object detection model and its application to new image datasets.

The first step of this project is to develop a base model using a CNN architecture such as VGG-16/19, Resnet50 or InceptionV3 and then train the model on an image dataset. After training the model, the accuracy of the model is tested and the results are recorded.

The next step is to use the Transfer Learning technique. This involves taking an existing model trained on a different dataset and fine-tuning it for the current dataset. The parameters of the existing model will be modified and the model will be retrained. The accuracy of the model will be tested and the results will be recorded.

Finally, the results of both models will be compared to determine which model provides better accuracy. This will help determine whether Transfer Learning can improve the accuracy of an object detection model when applied to a new image dataset.

The paper by Fan et al. presented an improved system for the detection of pedestrians based on the SSD model of object detection. This system introduced the Squeeze-and-Excitation (SE) model as an additional layer to the SSD model. With the SE model, the system was able to learn from itself and improve the accuracy of small-scale pedestrian detection. Experiments on the INRIA dataset demonstrated the accuracy of the improved model. This paper was used to understand the SSD model and its potential applications.

# Chapter 4

## Proposed System and Requirement Specification

### 4.1 Proposed System

#### 4.1.1 Dataset

The dataset is used to train the model. Raw images often need to be pre-processed before they can be used for further analysis or model training. Pre-processing can involve a range of tasks, such as checking for missing or noisy data, converting the image to a different format, resizing, or cropping the image, and more. The specific pre-processing steps needed will depend on the specific requirements of the task or analysis at hand. Once the images have been pre-processed, they can be used to create a dataset for further analysis or model training. We have three classes of gun, knife, and person, and based on position; weapons are marked in the image. It will generate a text file for each image which consists of coordinates used for markings. The label file will store the classes for which the images will be marked. This is used as the training dataset.

#### 4.1.2 Methodology

##### A) Separation of the dataset into train and test data:

After you have labeled the images in your dataset and made any necessary pre-processing steps, it is a good idea to compress the dataset into a zip file to make it easier to handle and transfer. You can then upload the zip file to a cloud storage service such as Google Drive. Once the dataset is uploaded, you can then split it into a training set and a test set using the desired split ratio (such as 70% for training and 30% for testing). You can then create separate folders for the training and test sets, and move the images into the appropriate folders.

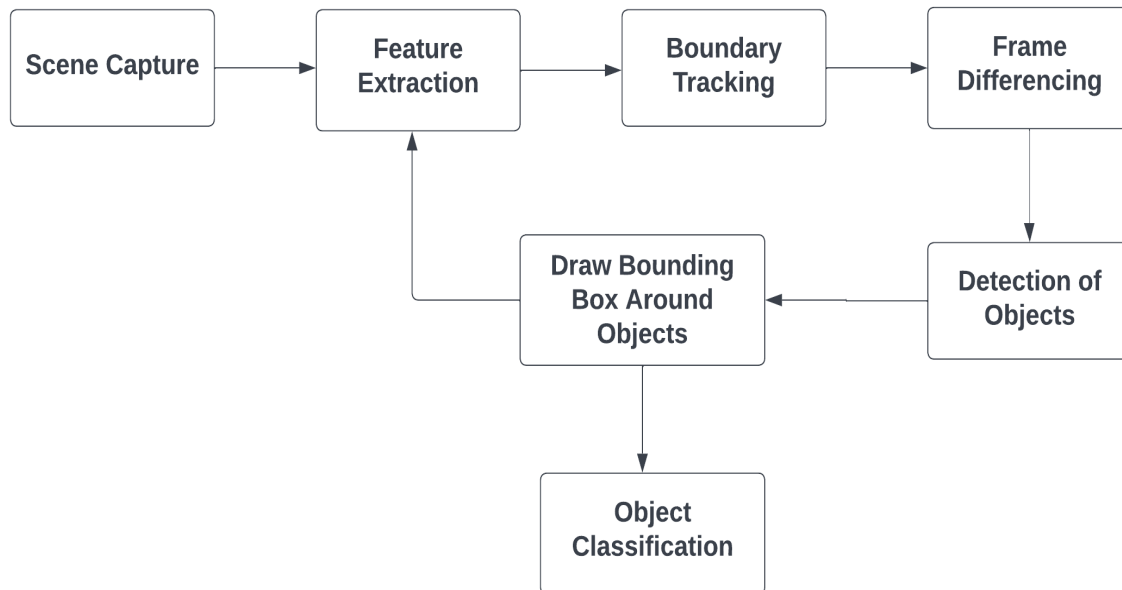


Fig 4.1 Methodology

1. **Scene Capture:** This is the initial stage where we must capture a view of a real environment that is most probably a crime scene
2. **Feature Extraction:** The data must be interpreted to analyze the weapon. Extraction of features is done from the initial dataset these new features have this little redundant information in them as possible and therefore features are often less intuitive to understand.
3. **Boundary Tracking:** In object detection, boundary tracking refers to the process of identifying and locating objects in video sequences or images by tracking their boundaries or contours. This can be done using various techniques such as object tracking algorithms, object detection algorithms, or a combination of both.
4. **Frame Differencing:** Frame differencing is a technique used in object detection to detect changes in a video stream by subtracting the current frame from a previous frame. This can be used to identify moving objects in the scene, as the moving objects will produce a significant difference between the frames.
5. **Detection of Object:** In this process, we must detect the instance of the object in the image.

6. Draw a Bounding Box Around the Object: Bounding Boxes act as the reference points for weapon detection and they are normally in rectangular shape. We must first draw rectangles over images and identify X and Y coordinates for each image.
7. Object Classification: We have three different classes in our dataset i.e., knife, person, and guns so will determine which object from these three classes is present in the given scene or image it refers to our model to find out which class is present.

## B) Weapon Detection

In weapon detection, data cleaning refers to the process of preparing the dataset for training the model. This involves a variety of tasks, such as.

- Annotating the dataset: Here, each weapon in the dataset must be labelled with a bounding box and class label. This can be done manually or using tools like Labelbox or any other.
- Removing noise and errors: There may be annotation errors, duplicate images, or other noise in the dataset that needs to be removed.
- Balancing the dataset: It is generally helpful to have a balanced dataset, with a similar number of images for each class. If some classes are overrepresented, you may want to downsample them.
- Normalizing the data: You may need to resize or crop the images to a consistent size or convert them to a uniform color space (e.g. RGB).
- Splitting the dataset: You'll need to split the dataset into training, validation, and sets.

By following these steps, we can improve the performance of our weapon detection model.

Feature engineering is the process of selecting and creating features that will be input to a machine can include tasks such as:

- Selecting appropriate image resolutions: The resolution of the input images can affect the performance of the object detection model. Higher resolutions may provide more detail but require more computation and may not fit in memory.
- Pre-processing the images: You may want to apply image processing techniques such as scaling, cropping, or color space conversion to improve the performance of the object detection model.
- Extracting features from the images: You may want to extract features such as edges, corners, or texture patterns from the images to use as inputs to the object detection model.
- Creating additional features: You may want to create additional features based on domain knowledge or data analysis. For example, you could create features based on the size, shape, or color of objects in the images.

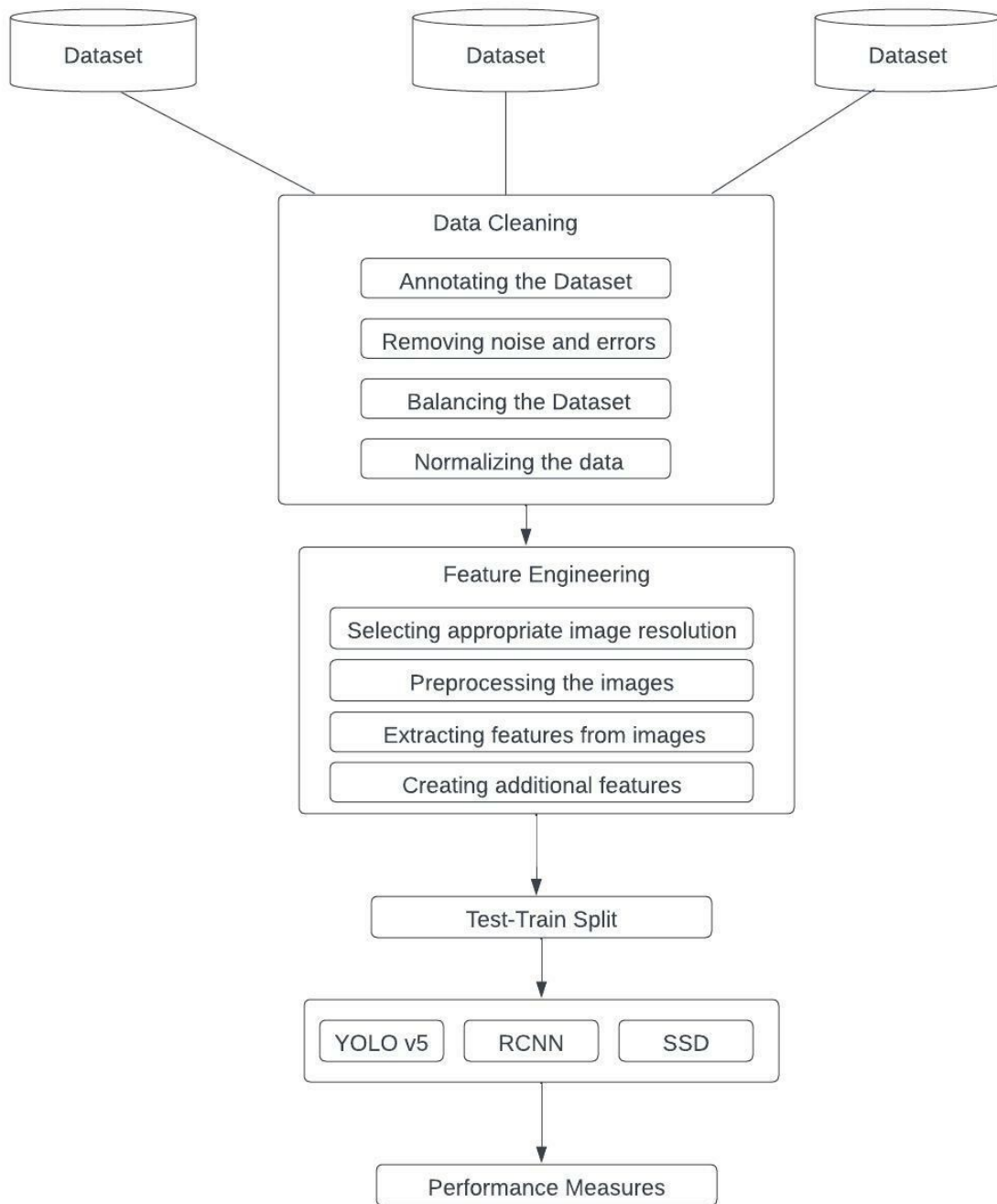


Fig 4.2 System Architecture

### C) Algorithms:

1. YOLOv5 is a real-time single-stage object detector algorithm. YOLOv5 algorithm helps to identify objects in videos, live feeds, or images.

In YOLOv5, these were the two unified blocks that turned into a single monolithic block.

1. feature extraction
2. object localization

YOLOv5 has three most important components:

- 1) Model Backbone
- 2) Model Neck
- 3) Model Head

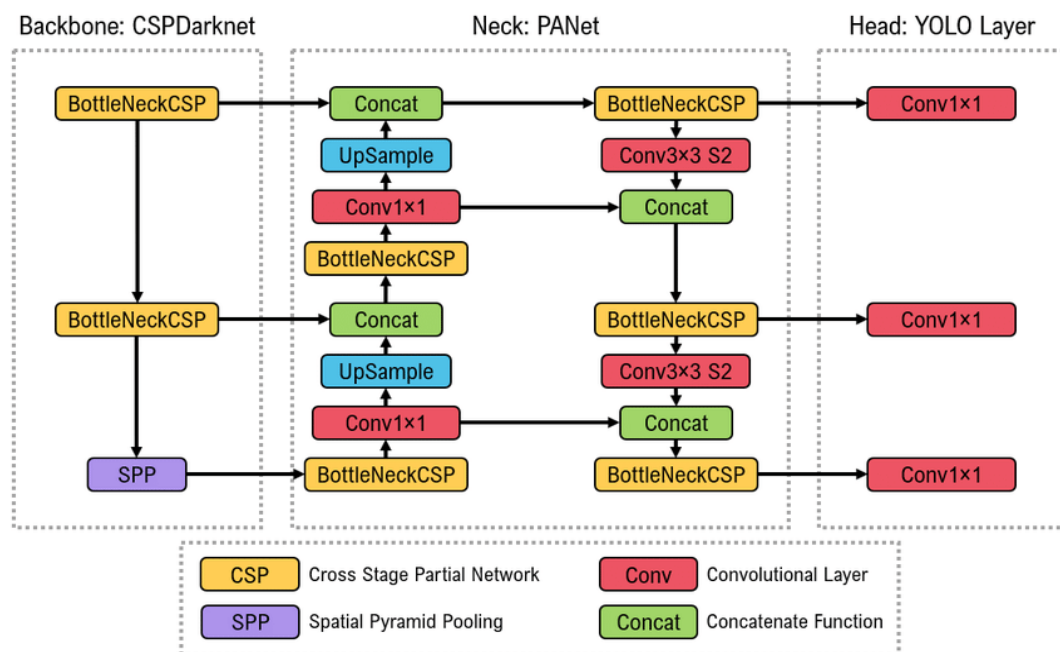


Fig.2. YOLOv5 Architecture [9]

The first step in object detection is feature extraction which is the backbone model of YOLOv5 architecture. The input image is given to the model then it will extract important features from an input image. for extracting informative features from the input image YOLOv5 uses Cross Stage Partial Networks (CSP). The next part is feature pyramids (FP) which are generated by the model neck. To generalize well on object scaling Feature Pyramids are necessary as they assist. Objects with different

scales and sizes are identified by the model and help unseen data to perform well. The final detection is performed by the model head. output vectors with their associated class probabilities are generated by Anchor boxes which are applied to feature class probabilities associated with vectors and scores of objectness.

2. Region-based Convolutional Neural Network (RCNN) is an object detection method based on the visual information of images. The region proposal is first computed by the network and then it feeds the proposed regions into the CNN for classification. The network does CNN pass for each object without sharing the computations, thus making the network slow.

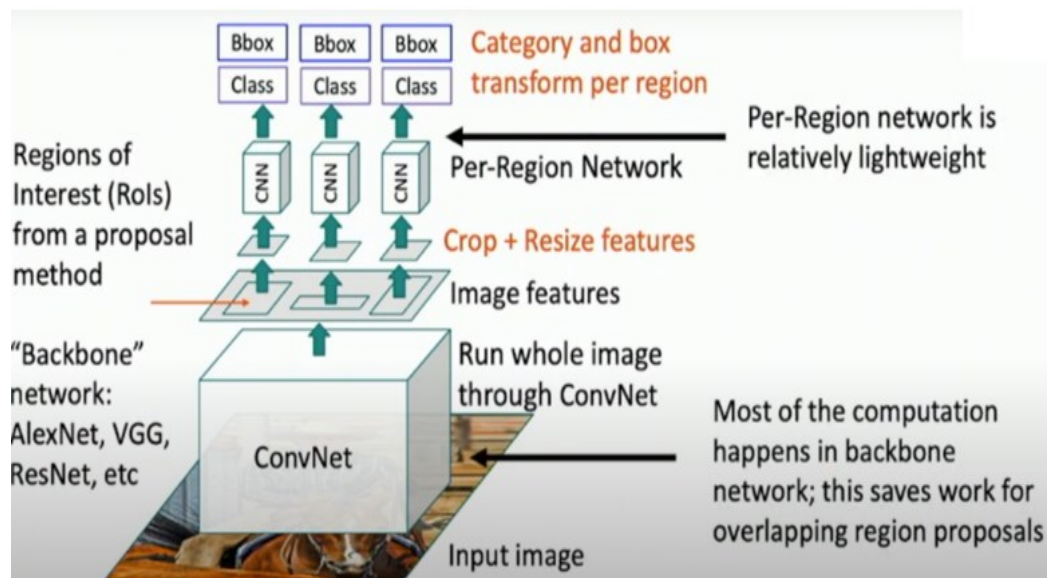


Fig.3 R-CNN Architecture [18]

Figure 3, shown above, describes the RCNN architecture. Then we find out the region where objects may be present in the Region Proposals stage and then will extract the features of the given input image using feature Extractor, confirm whether the region has object localization, and output the classification.

The following steps are often used by R-CNN to classify objects.

1. Take note of the n areas (Region Proposals) where the original image's items will be located in.



2. Use CNN to extract choices from regions that are 227x227 (AlexNet) or 224x224 (VGG16) in size.
  3. The 4096-dimensional Ultimate Output Layer (UOL) forecasts class adjustments using SVMs and prediction boxes.
3. SSD is a one-stage object identification technique that discretizes the output set of bounding boxes into a group of default boxes at various facet ratios and scales for each feature map position. The network creates scores for each object class in each default box at prediction time and modifies the box to fit the shape of the object. To naturally manage objects of various sizes, the network also mixes predictions from many feature maps with entirely different resolutions.

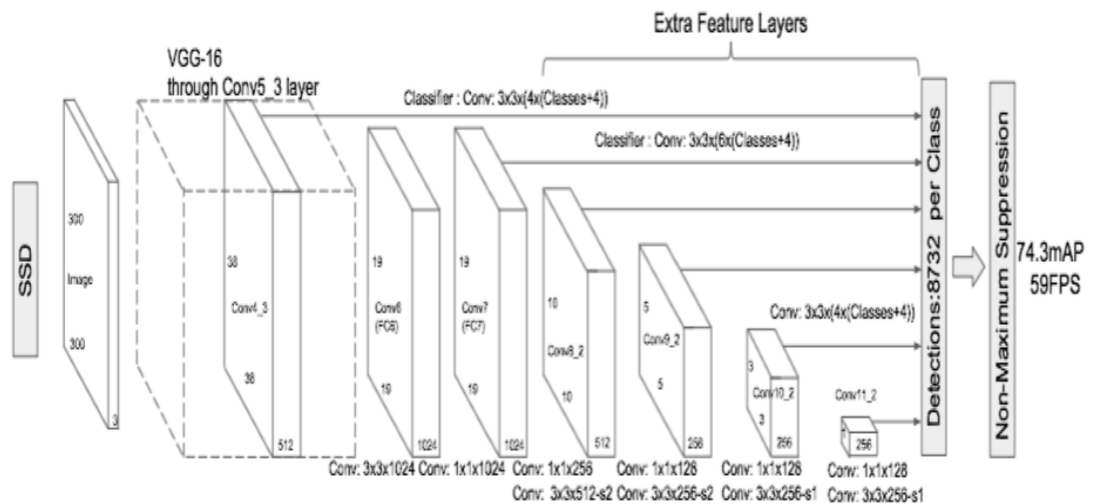


Fig.4. SSD VGG-16 Architecture [5]

As shown in the figure, SSD design builds on the VGG-16 architecture which is a popular choice for image classification tasks because of its robust performance and ability to handle high-quality images. It is a convolutional neural network (CNN) that is composed of multiple convolutional and fully connected (FC) layers.

The Single Shot Detector (SSD) design is based on the VGG-16 architecture, but it discards the fully connected layers in favor of a series of auxiliary convolutional layers. These additional convolutional layers allow the SSD to extract features at multiple scales and reduce the size of the input to each subsequent

layer. This enables the SSD to process images more efficiently and improve its performance on object detection tasks. Transfer learning, which involves using a pre-trained network as a starting point to train a new model, can also be effective in improving the results of the SSD.

## 4.2 Software Requirements Specification-SRS

### 4.2.1 ML Libraries and their Functionalities

- Sklearn: Sklearn provides a selection of efficient tools for machine learning and statistical modeling including classification, regression, clustering, and dimensionality reduction via a consistency interface in Python.
- Matplotlib: Matplotlib is a comprehensive library for creating static, animated, and interactive visualizations in Python. Matplotlib makes easy things easy and hard things possible.
- NumPy: NumPy is used to perform a wide variety of mathematical operations on arrays.
- nltk: It provides us with various text-processing libraries with a lot of test datasets
- Pandas: Pandas is a Python library. Pandas are used to analyse data
- Stream-lit: Stream-lit is an open-source app framework for Machine Learning and Data Science teams. Create beautiful web apps in minutes.

These are the versions of ML libraries we used in our project:

- GitPython - 3.7 version
- Ipython - 8.8.0 version
- Matplotlib - 3.3.6 version
- Opencv - 1.0.2 version
- Pillow - 9.4.0 version
- PyYaml - 6.0 version
- Torch - 1.13.1 version
- Torch vision - 1.6.0 version

- Pandas - 2.0.0 version
- Seaborn - 0.12.0 version used

## 4.2.2 Functional Requirements

### 1) Functional requirements of Code

- Reading data
- Pre-processing of data
- Splitting of training and testing data
- Feature Extraction
- apply different algorithms
- Display results and accuracy of each model

### 2) Functional requirements of the interface

- Select the input i.e., the option for the algorithm to detect weapon
- upload image/video
- Click on start detection
- Display results and accuracy of each model

### 3) Functional requirements of the overall system

- The system must be able to verify and validate the information
- Citizen Interface - citizens can ask the police for information acknowledgments.
- Utilize deep learning models to automatically detect weapons and firearms.
- Finding knives and other sharp, cold-steel weapons in surveillance video.
- Real-world application in intricate and densely packed landscapes.

### 4.2.3 Non-Functional Requirements

The non-functional characteristics that are part of the operating group are:

1. Flexible data flow.
2. multi-channel capability
3. Easy handling
4. Ongoing optimization
5. Analytics & reporting

**Usability:** It is important to create user-intuitive user interfaces.

**System Availability:** If a user is unable to perform any normal System function, the system is considered down.

**Capability and Efficiency:** Under both standard and peak conditions, for routine tasks, the system must have sufficient reaction times.

**Interface Loading Time:** The loading time of each page/interface from the Module should be consistent with the other pages already in the application. Constraints: This requirement can be dependent on the user's ram or the server itself. Also, The bandwidth of the network connection.

**Response time:** As soon as an input is given by the user, the system should generate an expected output in time. Constraints: This requirement also depends on the bandwidth of the network connection.

### 4.3 Scope of Project

The purpose of this project is to develop a system for detecting weapons in real-time using computer vision and machine learning techniques. The system will be designed to operate in indoor and outdoor environments and will be able to detect a range of weapons including guns, knives, and persons. This project is intended to improve public safety by providing a means for detecting weapons in real-time and will have applications in a range of settings including schools, public transportation, and public events. It can be further extended to detection in live video. Installation can be done in cameras/drones for weapon detection. We can predict crime for real-time implementation.

### 4.4 Deployment Requirements

#### 4.4.1 Hardware Requirements

| Requirement              | Specification                |
|--------------------------|------------------------------|
| Personal Computer/Laptop | i5 Processor based or higher |
| Processor Dual core      | i5 Processor based or higher |
| Hard Drive               | 8 GB or higher               |

Table 4.1: Hardware Requirements

#### 4.4.2 Software Requirements

| Requirement           | Specification        |
|-----------------------|----------------------|
| Operating System      | Windows 10 or higher |
| Google Colab notebook | Python 3.6. 9        |
| Anaconda Navigator    | Version: v2. 3.1     |
| Spyder                | Version: 5.4.0       |
| Streamlit             | Version: 1.16.0      |
| Python                | Version: 3.10.6      |

Table 4.2: Software Requirements

#### 4.5 Project Deliverable

User:

- User interface for weapon detection using various deep-learning techniques
- User will be able to enter an image of any real-time crime scene and the model will detect the weapon from the scene based on chosen algorithm.
- User will be able to see the results on the screen.
- User will also be able to compare the performance of different algorithms based on appropriate parameters.

# Chapter 5

## Design

### 5.1 Data Flow Diagram

The user input image is uploaded on a web application and then in the backend model runs and displays computed results on the screen. Below is the dataflow fig.

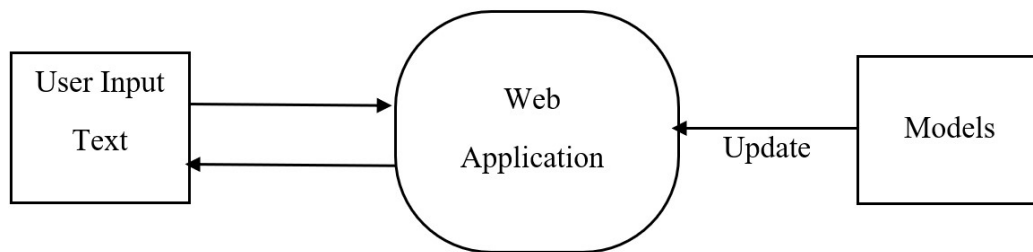


Figure 5.1: Data Flow Diagram

### 5.2 System Architecture

A system architecture is a conceptual model that defines a system's structure, behaviour, and other aspects. A formal description and representation of a system arranged in a way that facilitates reasoning about the system's structures and behaviours is known as an architecture description.



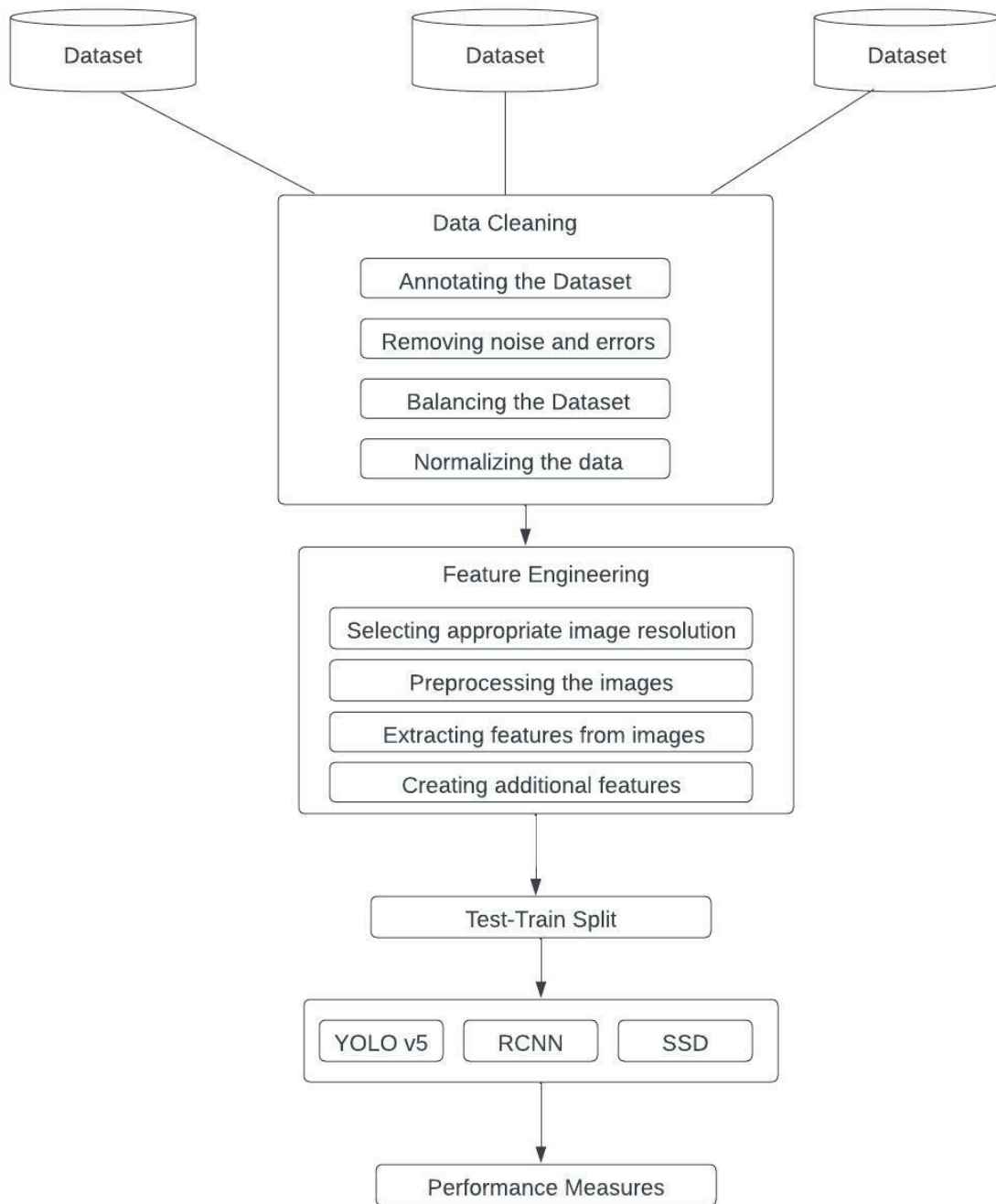


Figure 5.2: System Architecture 1

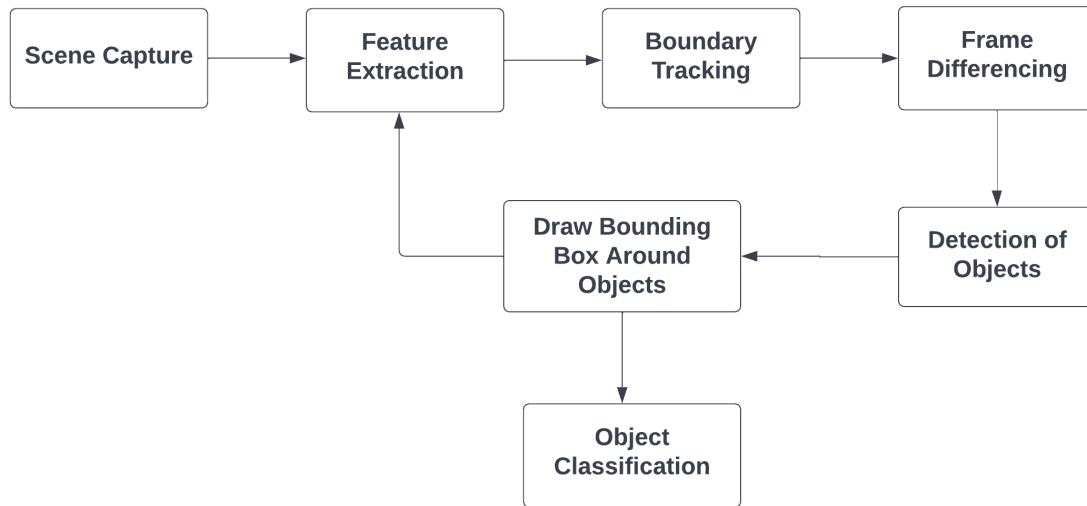


Figure 5.3: System Architecture 2

## 5.3 UML Diagrams

### 5.3.1 Use Case Diagram

A use case diagram depicts the system's numerous use cases and different sorts of users and is frequently supplemented by other diagrams. Use cases are depicted in an oval shape, with arrows indicating relationships.

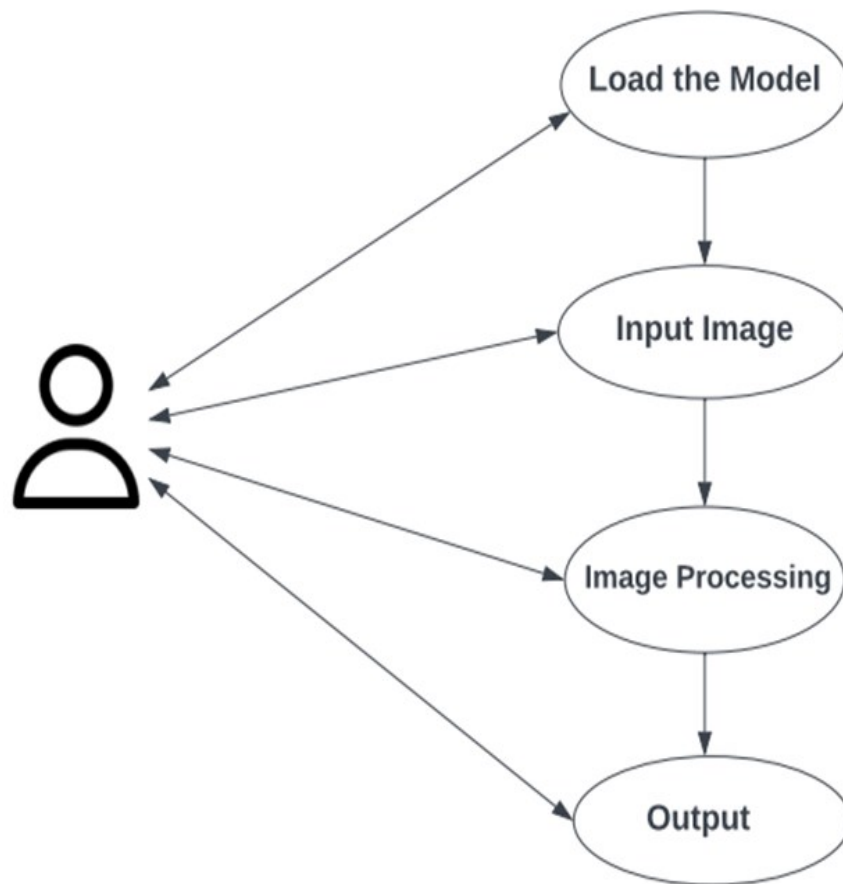


Figure 5.4: Use Case Diagram

### 5.3.2 Component Diagram

Users can directly interact with our user interface (i.e. web application). Users can select any one computational architecture from R-CNN, YOLO V5, and SSD for computation. Input images can be passed by the user to the web application and images are processed based on selected architecture the computed result is displayed on the web application's screen along with weapon-detected images with a bounding box.

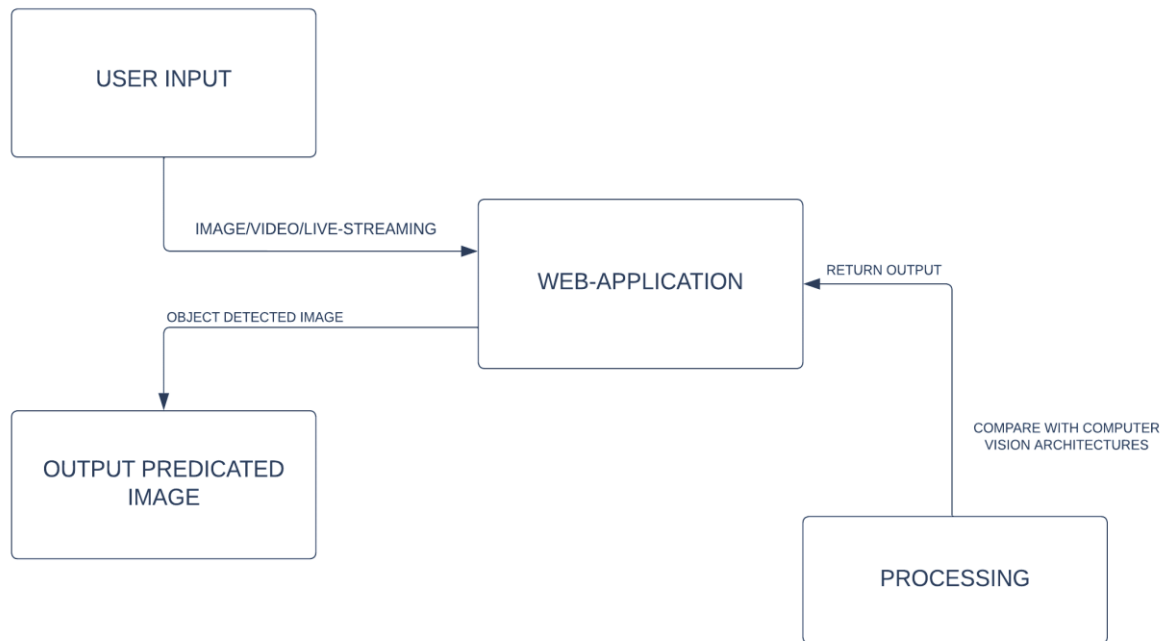


Figure 5.5: Component Diagram

### 5.3.3 Sequence Diagram

First import a dataset that contains images with bounding boxes and class labels. The bounding boxes are represented by the x coordinates and y coordinates at the top left corner, as well as the width and height of the box. We are using the Roboflow tool to annotate our dataset and import the images, and We may also be using data augmentation to improve the results of our model. We also mentioned that we are splitting the data into train, test, and validation sets and that we are using detectron2 to choose our model. We are configuring the hyperparameters for the model and using the default trainer from detectron2 to train the model using the train and validation images.

We are then using the default predictor to get the predictions from the model on the test images, and using the Visualizer class to draw the bounding boxes on the test images with their confidence values. We mentioned that the metric we are using is called mean average precision and that we are adjusting the hyperparameters to try to get the best results.

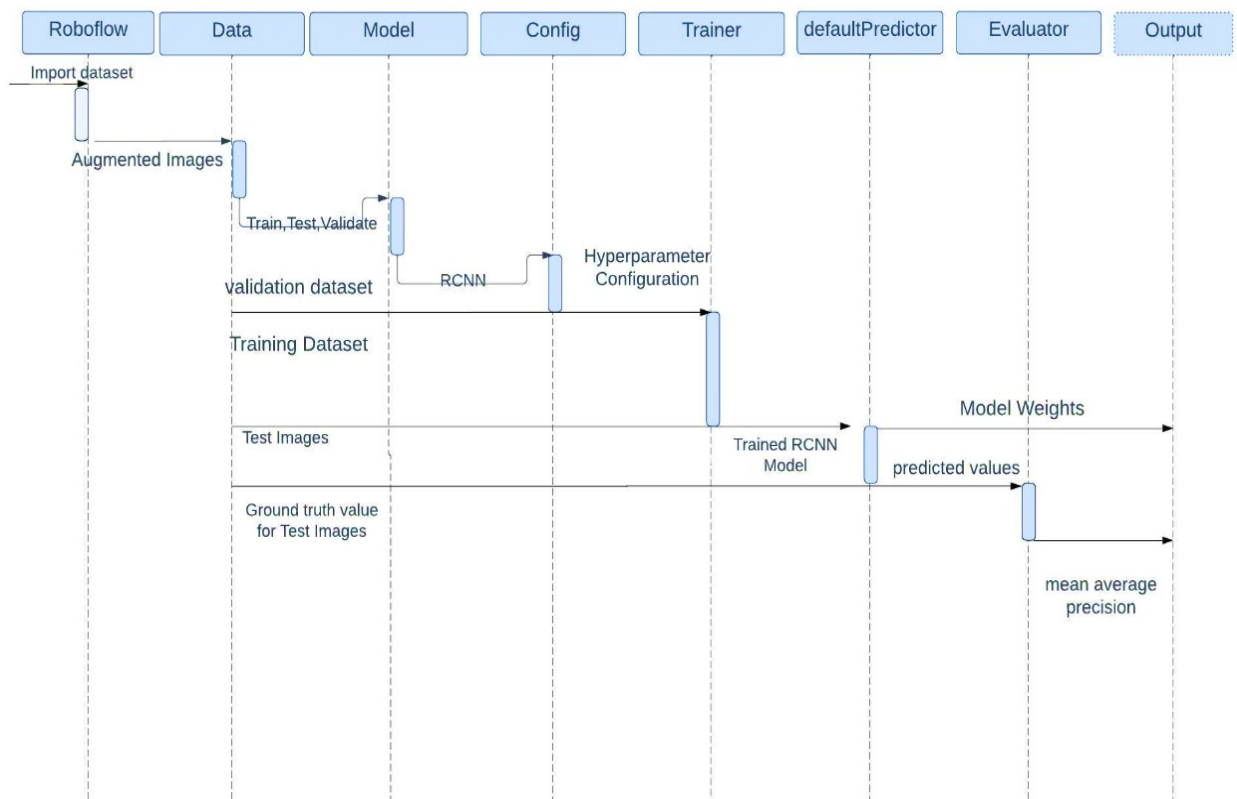


Fig.5.6 Sequence Diagram of weapons detection in crime scenes

# Chapter 6

## Development/Implementation Details

### 6.1 Technology Used

#### 6.1.1 Deep Learning

Deep Learning is a type of machine learning that involves training artificial neural networks on a large dataset. These neural networks are suitable to learn and make intelligent opinions on their own, without the need for unequivocal programming. Deep learning has been successful in a wide range of operations, including image and speech recognition, natural language processing, and indeed playing games. These neural networks are inspired by the structure and function of the mortal brain and are suitable to learn and make intelligent opinions on their own by relating patterns and connections in data. Deep learning algorithms have been used to achieve state-of-the-art results in a wide range of operations, including image and speech recognition, natural language processing, and machine translation.

#### 6.1.2 YOLOv5(You Look Only Once)

YOLO (You Only Look Once) is a real-time object detection algorithm that was developed by Joseph Redmon and his team at the University of Washington. It is a convolutional neural network (CNN) based architecture that is designed to perform fast object detection. The main idea behind YOLO is to use a single neural network to perform object detection, rather than using a pipeline of separate neural networks for classification and localization.

One of the key features of YOLO is its ability to make predictions at high speed, allowing it to be used in real-time applications. It does this by dividing the input image into a grid

of cells, and using each cell to predict the presence and location of objects in that region of the image. YOLO is trained on a large dataset of labeled images, and it uses this training data to learn to identify objects in new images.

There have been several versions of YOLO, including YOLOv1, YOLOv2, YOLOv3, and YOLOv4. Each version has made improvements in terms of accuracy, speed, and the ability to detect a larger number of object classes.

One of the key features of YOLO is that it processes the entire image in a single pass, rather than dividing the image into smaller regions and processing them separately. This allows it to run in real-time, making it suitable for use in applications where fast object detection is important, such as in self-driving cars or video surveillance systems.

In addition to object detection, YOLO has also been used for tasks such as face detection, pedestrian detection, and image segmentation. It is a popular choice for many computer vision tasks due to its high speed and accuracy.

### 6.1.3 R-CNN (Region-based Convolutional Neural Network)

R-CNN (Region-based Convolutional Neural Network) is a type of convolutional neural network (CNN) architecture that was developed for object detection. It was developed by Ross Girshick and colleagues at Microsoft Research in the early 2010s and has been widely used in the field of computer vision.

The R-CNN algorithm works by first generating a set of bounding boxes around potential objects in an image. It then uses a CNN to classify each bounding box as containing an object or not and to classify the object if it is present. This two-stage process is repeated for each bounding box, allowing the algorithm to detect and classify multiple objects in an image.

One of the key benefits of R-CNN is its ability to achieve high accuracy on a wide range of object detection tasks. However, it can be computationally intensive, as it requires running the CNN on each bounding box individually. This has led to the development

of faster variants of R-CNN, such as Fast R-CNN and Faster R-CNN, which are designed to reduce the computational cost of object detection.

One of the main advantages of the R-CNN architecture is its ability to achieve high accuracy in object detection tasks. However, it can be slow to run, as it requires multiple forward and backward passes through the CNN for each region proposal.

Later versions of the R-CNN architecture, such as Fast R-CNN and Faster R-CNN, were developed to address this issue and improve the speed of the object detection process.

#### 6.1.4 SSD (Single Shot Detector)

The SSD architecture is based on a feedforward CNN and is designed to be fast and efficient, making it suitable for use in real-time applications such as video surveillance and self-driving cars. It works by dividing the input image into a set of predefined grids and using a series of convolutional and pooling layers to extract features from each grid cell. A classifier is then applied to each grid cell to predict the presence and location of objects in the image.

One of the main advantages of SSD architecture is its ability to detect objects of various sizes and aspect ratios, making it well-suited for detecting a wide range of objects in a single image. It is also relatively simple to implement and can be trained on a large dataset in a short amount of time.

The SSD architecture is based on the idea of "single shot" detection, meaning that it can detect objects in an image in a single pass, rather than requiring multiple passes or stages. This makes it faster and more efficient than some other object detection algorithms, such as R-CNN and its variants.

The SSD architecture consists of a base CNN network, such as VGG or MobileNet, and additional layers that are added on top of the base network. These additional layers are responsible for predicting bounding boxes and class probabilities for each object in the image.



One of the key advantages of SSD architecture is its ability to process images at multiple scales, allowing it to detect objects of different sizes in the same image. It is also relatively simple to implement and has achieved good results.

### 6.1.5 Google colab

Google Colab, or Google Colaboratory, is a free online cloud-based service provided by Google that allows you to use Jupyter notebooks in the cloud. It allows you to run and edit Jupyter notebooks on Google's servers, which means you don't have to install anything on your own computer. It provides a number of benefits for data scientists and researchers, including:

- Access to powerful hardware: Google Colab provides access to GPUs and TPUs, which can be used to accelerate machine learning and data analysis tasks.
- Collaboration: Google Colab allows you to share notebooks with others and work on them together in real time.
- Convenience: You don't have to install any software on your computer to use Google Colab, which makes it easy to get started and saves you time.
- Integration with Google Drive: Google Colab notebooks are stored in Google Drive, which makes it easy to access them from anywhere and share them with others.

Google Colab is a popular choice for data scientists and researchers because it provides a convenient and powerful platform for working with Jupyter notebooks in the cloud. It is especially useful for tasks that require access to powerful hardware or for working with large datasets that may not fit on your local machine.

### 6.1.6 Kaggle

Kaggle is a resource for finding and publishing data sets, as well as a community of data scientists and machine learning interpreters. Kaggle was innovated in 2010 and was acquired by Google in 2017. It is now an attachment of Google LLC.

On Kaggle, users can participate in machine learning competitions to develop and test their skills, as well as access a wide range of resources, including pre-processed data sets, Jupyter notebooks, and pre-trained models. Kaggle also offers educational resources, such as free online courses, to help users learn more about data science and machine learning. It also serves as a repository of datasets and a place to publish and share data-related projects.

Users can participate in a variety of competitions, including those sponsored by companies looking for solutions to real-world problems, as well as academic and research institutions. Competitions often provide a dataset and a specific problem to solve, such as predicting the likelihood of churn for a particular customer, and participants build and train models to make predictions.

In addition to competitions, Kaggle also offers a cloud-based workbench for developing and running data science code, as well as a public dataset repository containing over 30,000 datasets. It is a popular platform among data scientists and machine learning enthusiasts and is often used to showcase skills and find new job opportunities.

### 6.1.7 Anaconda

Anaconda is a free and open-source distribution of Python and R designed for scientific computing and data science. It includes a wide range of tools and libraries for data analysis, scientific computing, and machine learning, as well as a package manager called conda that allows you to install and manage additional packages and libraries. It is a specific version of Anaconda that includes support for Python 3.x. It includes the latest versions of Python, NumPy, pandas, and other popular libraries for data science and scientific computing.

Anaconda is a popular choice for data scientists and researchers because it provides a comprehensive set of tools and libraries for data analysis and scientific computing, and it is easy to install and use. It is also widely used in the scientific and data science communities, which means that there is a large and active community of users and developers who can provide support and help troubleshoot issues. In addition to the Anaconda distribution, Anaconda also provides a cloud-based platform called Anaconda

Cloud, which allows you to share and collaborate on data science projects, and a desktop-based integrated development environment (IDE) called Anaconda Navigator, which provides a graphical user interface for working with Anaconda and its tools.

### 6.1.8 Spyder

Spyder is a free and open-source integrated development environment (IDE) for scientific computing and data science. It is written in Python and designed specifically for working with scientific and mathematical applications, including data analysis and machine learning.

Spyder includes a number of features and tools that are useful for scientific computing and data science, including:

- A powerful text editor that supports syntax highlighting and auto completion for a variety of programming languages, including Python, R, and Julia.
- A console for running code interactively.
- An interactive debugger for troubleshooting code.
- Integration with popular data science libraries and tools, such as NumPy, pandas, and Matplotlib.
- A variety of built-in plots and visualizations for data analysis and exploration.

Spyder is a popular choice for data scientists and researchers because it provides a powerful and flexible platform for working with scientific and mathematical applications, and it is easy to use and customize. It is also widely used in education and research, as it allows you to create and share documents that can be easily understood by others.

### 6.1.9 Python

Python is a high-level, general-purpose programming language that is widely used in data science, machine learning, and scientific computing. It is a powerful and flexible language that is easy to learn and use, and it has a large and active community of users and developers.

Some of the key features of Python include:

- A simple, easy-to-learn syntax that emphasizes readability and reduces the cost of program maintenance.
- An interpreted language, which means that programs written in Python do not need to be compiled before they are executed.
- A large standard library, which supports many common programming tasks such as connecting to web servers, reading and writing files, and working with data.
- Support for object-oriented, imperative, and functional programming styles.
- Dynamically-typed, which means that you don't have to specify the data type of a variable when you declare it.

Python is a popular choice for data scientists and researchers because it provides a large ecosystem of libraries and tools that support data analysis and scientific computing. Some of the popular libraries used in data science and machine learning include NumPy, pandas, and scikit-learn. Python is also widely used in web development, scientific computing, and system scripting.

#### 6.1.10 Streamlit

Streamlit is an open-source Python library that allows you to create interactive, web-based applications and dashboards using only Python code. It is designed to make it easy for data scientists and machine learning engineers to build and deploy custom data applications and dashboards quickly and easily. It is built on top of the popular web framework Flask and uses modern web technologies such as React and Web Sockets to provide a smooth and responsive user experience. It provides a simple, intuitive interface for creating and customizing web-based applications using Python, and it includes a number of built-in components for visualizing and interacting with data.

Some of the key features of Streamlit include:

- A simple, easy-to-use API for creating web-based applications using Python.
- A wide range of built-in components for visualizing and interacting with data, including charts, maps, tables, and more.
- Support for custom styling and layout using CSS and HTML.
- A built-in server for running and hosting applications locally or on the web.

- Support for deploying applications to the cloud using platforms such as Heroku and AWS.

Streamlit is a popular choice for data scientists and machine learning engineers because it provides a simple and powerful platform for building custom data applications and dashboards quickly and easily. It is especially useful for prototyping and developing custom tools and solutions for data analysis and visualization.

# Chapter 7

## Testing

### 7.1 Test Cases

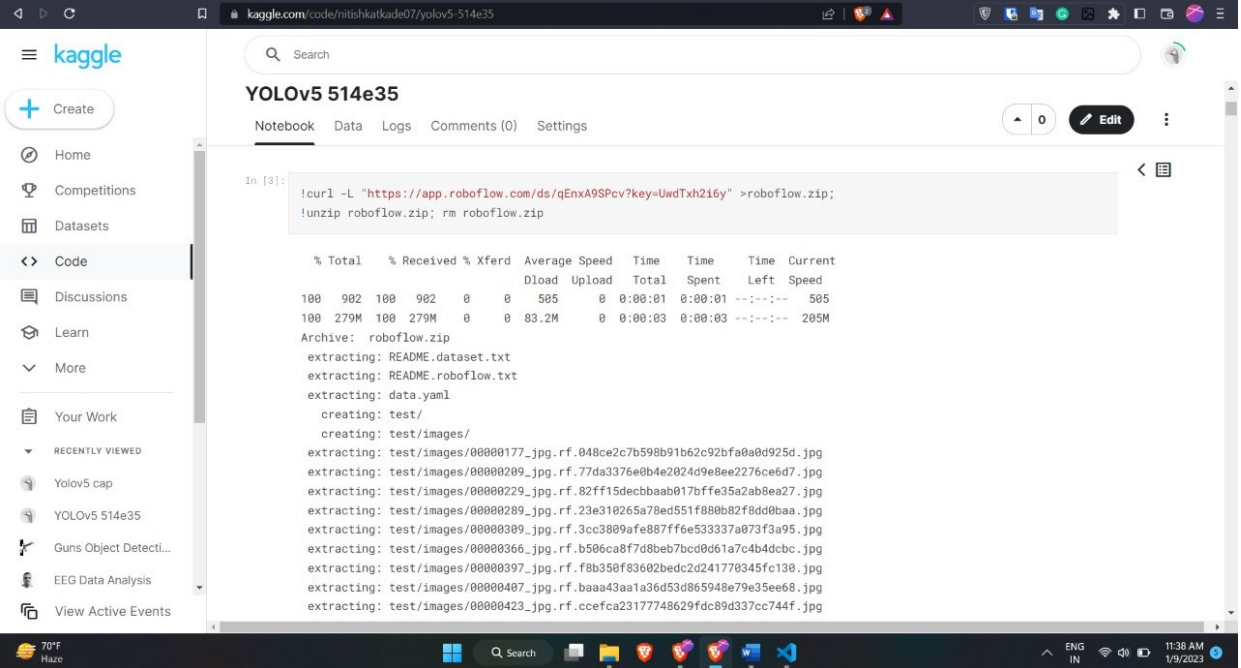
A test case is a set of conditions or variables under which a tester will determine whether an application, software system or one of its features is working as it was originally intended. A test case includes specific input data, the expected results, and any other information needed to execute the test.

Test cases are an important part of the software testing process, as they provide a structured way to verify that a system or component is working correctly. Test cases help ensure that a system is tested thoroughly and that any defects or issues are identified and addressed before the system is released.

#### 7.1.1 Unit Testing

Unit testing is a software testing technique in which individual units or components of a software application are tested in isolation from the rest of the application. A unit is the smallest testable part of an application, such as a function or a class. The goal of unit testing is to validate that each unit of the application is working as intended and meets the specified requirements. Unit tests are typically automated and run every time the code is changed to ensure that the application continues to function correctly. Unit testing is an important part of the software development process, as it helps ensure the quality and reliability of the application. It is often used in combination with other testing techniques, such as integration testing and acceptance testing, to form a comprehensive testing strategy.

## 1. Pre-processing of dataset:



The screenshot shows a Kaggle notebook titled "YOLOv5 514e35". The code cell contains the following commands:

```
!curl -L "https://app.roboflow.com/ds/qEnxA9SPcv?key=UwdTxh2i6y" > roboflow.zip;
!unzip roboflow.zip; rm roboflow.zip
```

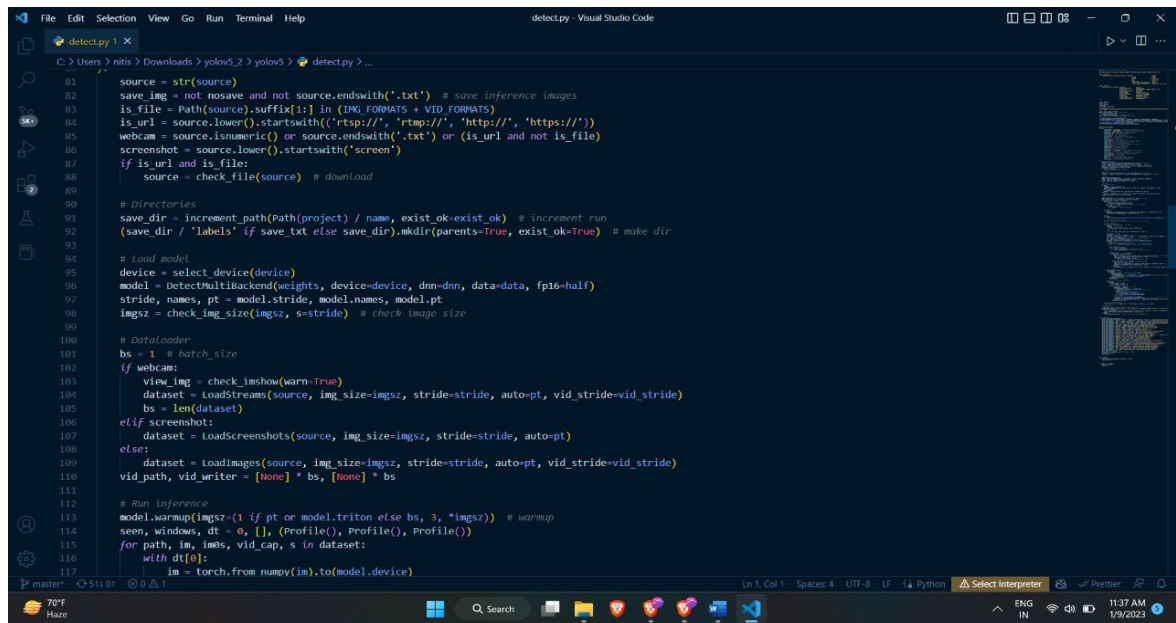
The output shows the download progress and the extraction of the dataset files:

| % Total | % Received | % Xferd | Average Speed | Time   | Time  | Time  | Current                         |
|---------|------------|---------|---------------|--------|-------|-------|---------------------------------|
|         |            |         | Dload         | Upload | Total | Spent | Left                            |
| 100     | 902        | 100     | 902           | 0      | 0     | 505   | 0 0:00:01 0:00:01 --:--:-- 505  |
| 100     | 279M       | 100     | 279M          | 0      | 0     | 83.2M | 0 0:00:03 0:00:03 --:--:-- 205M |

Archive: roboflow.zip  
extracting: README.dataset.txt  
extracting: README.roboflow.txt  
extracting: data.yaml  
creating: test/  
creating: test/images/  
extracting: test/images/00000177\_.jpg.rf.048ce2c7b598b91b62c92bfa0a0d925d.jpg  
extracting: test/images/00000209\_.jpg.rf.77da3376e8b4e2024d9e8ee2276ce6d7.jpg  
extracting: test/images/00000229\_.jpg.rf.02ff15decbbab017bffe35a2ab8ea27.jpg  
extracting: test/images/00000289\_.jpg.rf.23e310265a78ed551f880b82f8dd0baa.jpg  
extracting: test/images/00000309\_.jpg.rf.3cc3809afe887ff6e533337a073f3a95.jpg  
extracting: test/images/00000366\_.jpg.rf.b506ca8f7d8beb7bcd0d61a7c4b4dcbc.jpg  
extracting: test/images/00000397\_.jpg.rf.f8b350f83602bedc2d241770345fc130.jpg  
extracting: test/images/00000407\_.jpg.rf.baaa43aa1a36d53d865948e79e35ee68.jpg  
extracting: test/images/00000423\_.jpg.rf.ccefc23177748629fdc89d337cc744f.jpg

Figure 7.1: To check dataset is loaded successfully

## 2. Weapon Detection Code:



```

101 source = str(source)
102 save_img = not nosave and not source.endswith('.txt') # save inference images
103 is_file = Path(source).suffix[1:] in (IMG_FORMATS + VID_FORMATS)
104 is_url = source.lower().startswith(('rtsp://', 'rtmp://', 'http://', 'https://'))
105 webcam = source.isnumeric() or source.endswith('.txt') or (is_url and not is_file)
106 screenshot = source.lower().startswith('screen')
107 if is_url and is_file:
108     source = check_file(source) # download
109
110 # Directories
111 save_dir = increment_path(Path(project) / name, exist_ok=True) # increment run
112 (save_dir / 'labels' if save_txt else save_dir).mkdir(parents=True, exist_ok=True) # make dir
113
114 # Load model
115 device = select_device(device)
116 model = DetectMultiBackend(weights, device=device, dnn=dnn, data=data, fp16=half)
117 stride, names, pt = model.stride, model.names, model.pt
118 imgsz = check_img_size(imgsz, s=stride) # check image size
119
120 # Dataloader
121 bs = 1 # batch size
122 if webcam:
123     view_img = check_imgshow(warn=True)
124     dataset = LoadStreams(source, img_size=imgsz, stride=stride, auto-pt, vid_stride=vid_stride)
125     bs = len(dataset)
126 elif screenshot:
127     dataset = LoadScreenshots(source, img_size=imgsz, stride=stride, auto-pt)
128 else:
129     dataset = LoadImages(source, img_size=imgsz, stride=stride, auto-pt, vid_stride=vid_stride)
130 vid_path, vid_writer = [None] * bs, [None] * bs
131
132 # Run inference
133 model.warmup(imgsz=(1 if pt or model.triton else bs, 3, *imgsz)) # warmup
134 seen, windows, dt = 0, [], (Profile(), Profile(), Profile())
135 for path, im, im0s, vid_cap, s in dataset:
136     with dt[0]:
137         im = torch.from_numpy(im).to(model.device)
  
```

Figure 7.2: Testing of Code

## 3. Testing Result:



Figure 7.3: Output for Testing



# Chapter 8

## Results and Discussion

### 8.1 User Interface

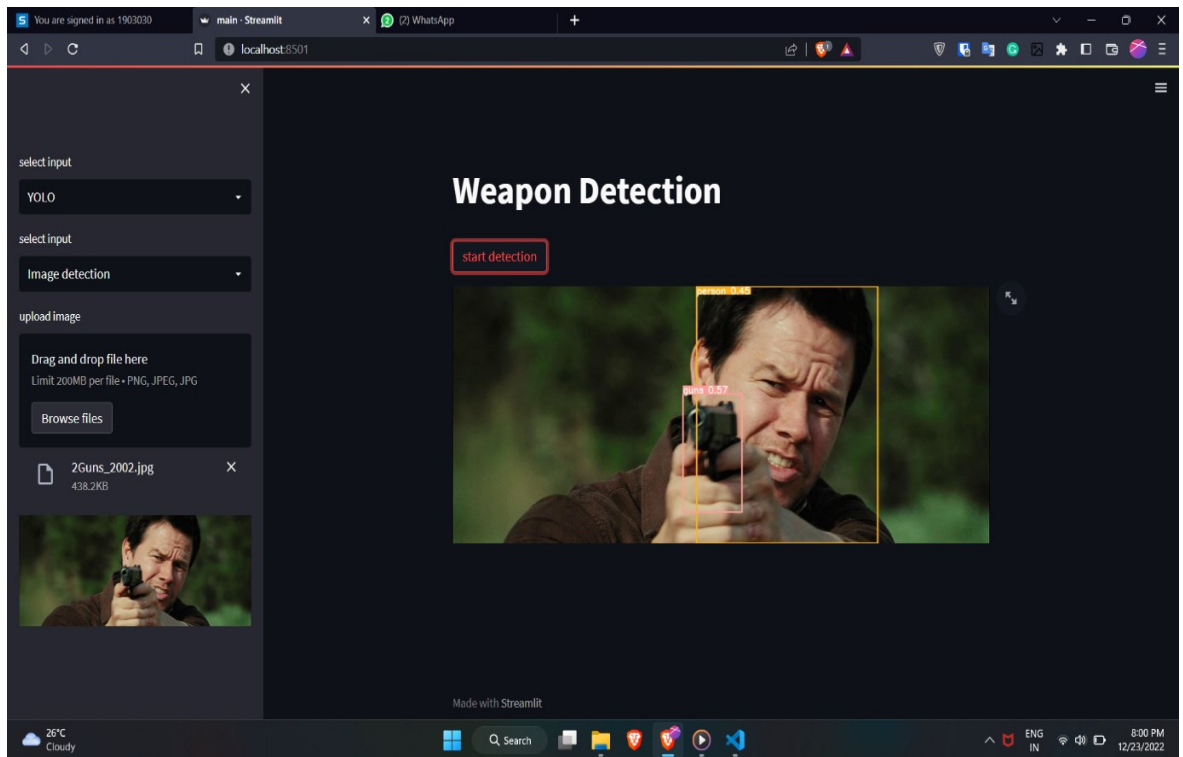


Fig. 8.1: Weapon detection model

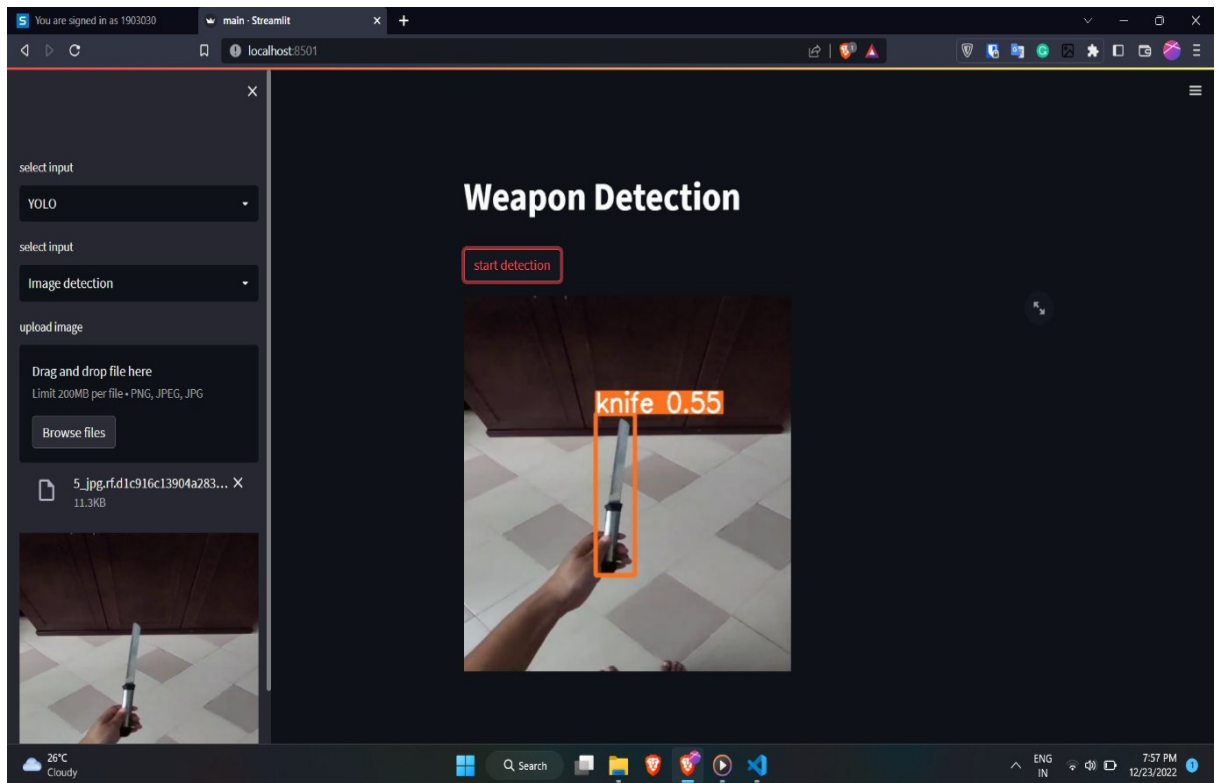


Fig. 8.2: Weapon Detection Model

## 8.2 Result Images for different algorithms

### 1) You Only Look Once (YOLO v5):



Fig 8.3 YOLO v5 Result

## 2) Region-based Convolutional Neural Network (R-CNN):



Fig 8.4 R-CNN Result

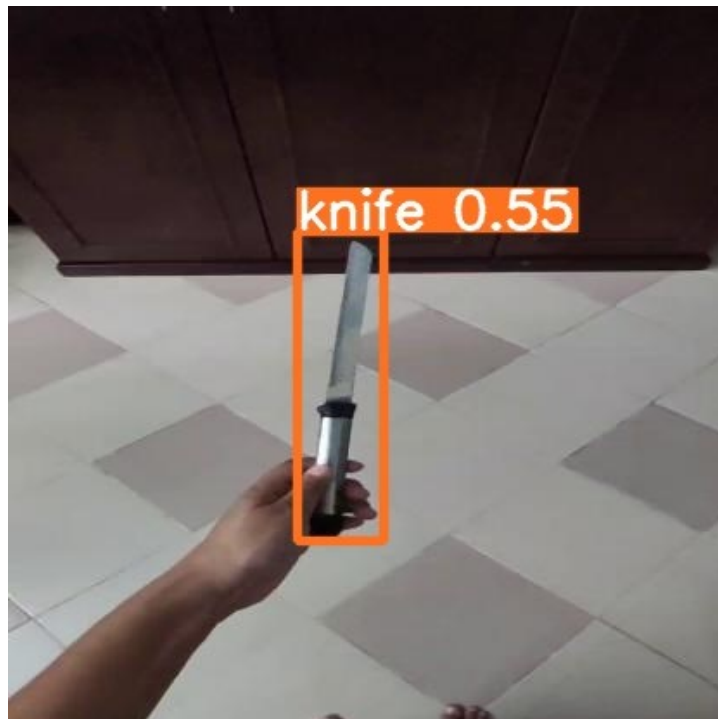


Fig 8.5 RCNN Result

### 3) Single Shot Detector (SSD):



Fig 8.6 SSD Result

### 8.3 Comparison of various models:

#### 1) Single Shot Detector (SSD):

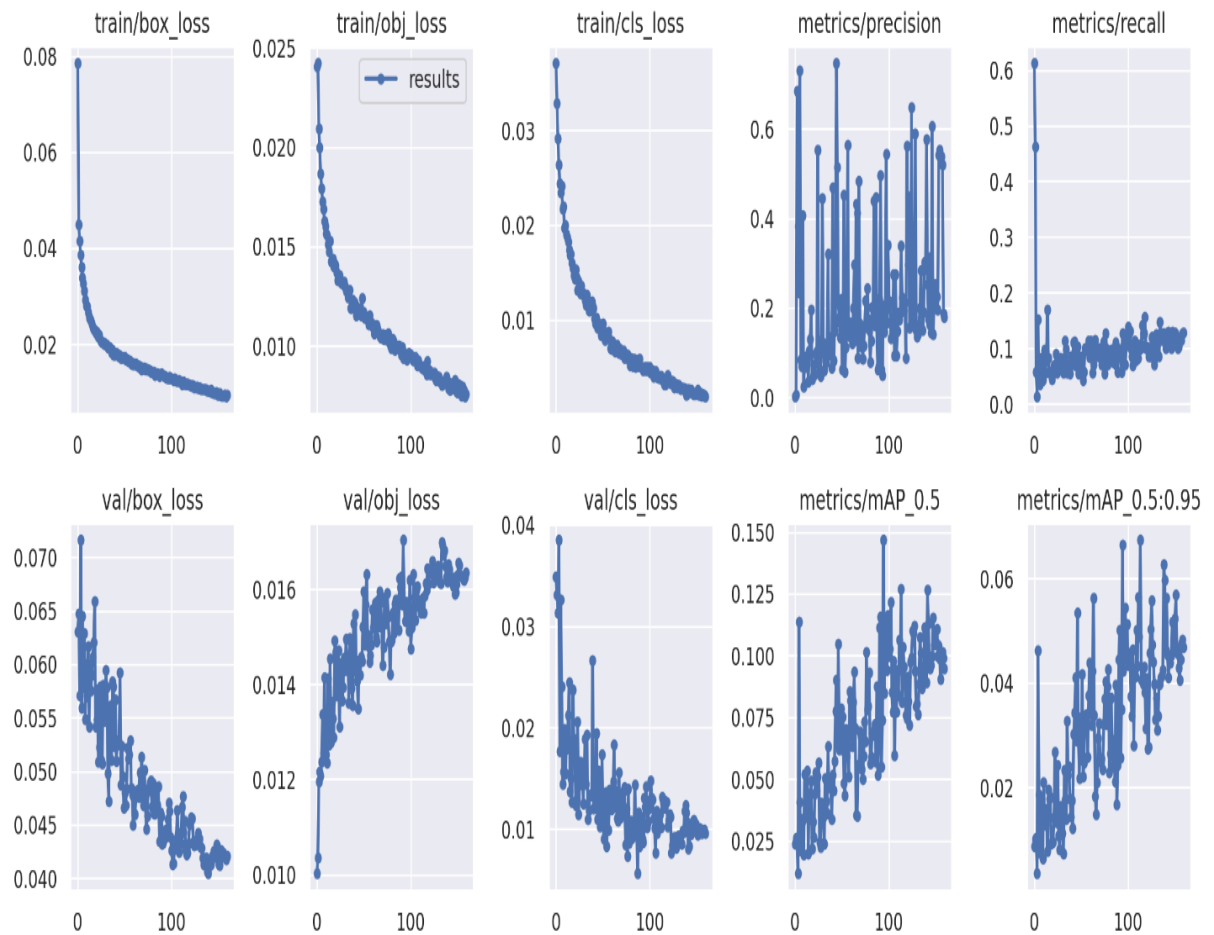


Fig.8.7 Single Shot Detector model evaluation

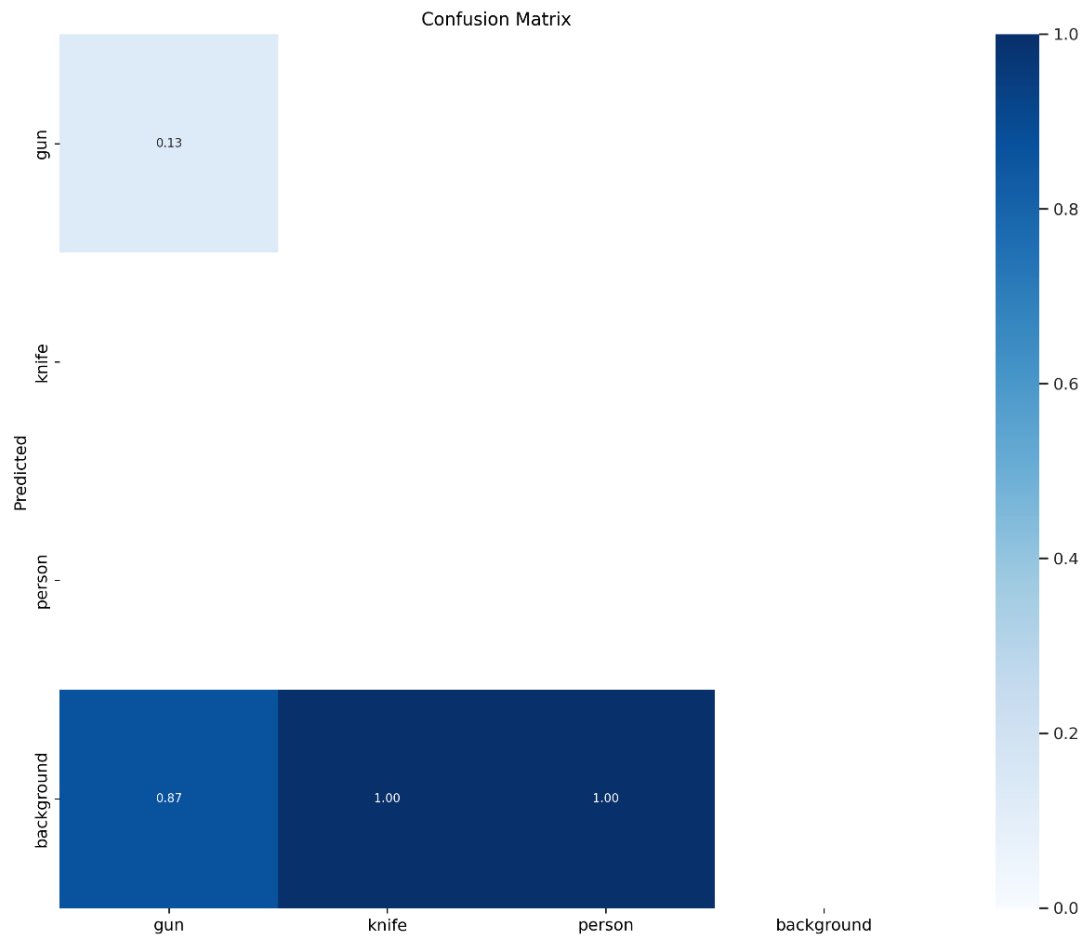


Fig.8.8 Confusion matrix for Single Shot Detector (SSD)

## 2) Region-based Convolutional Neural Network (RCNN)

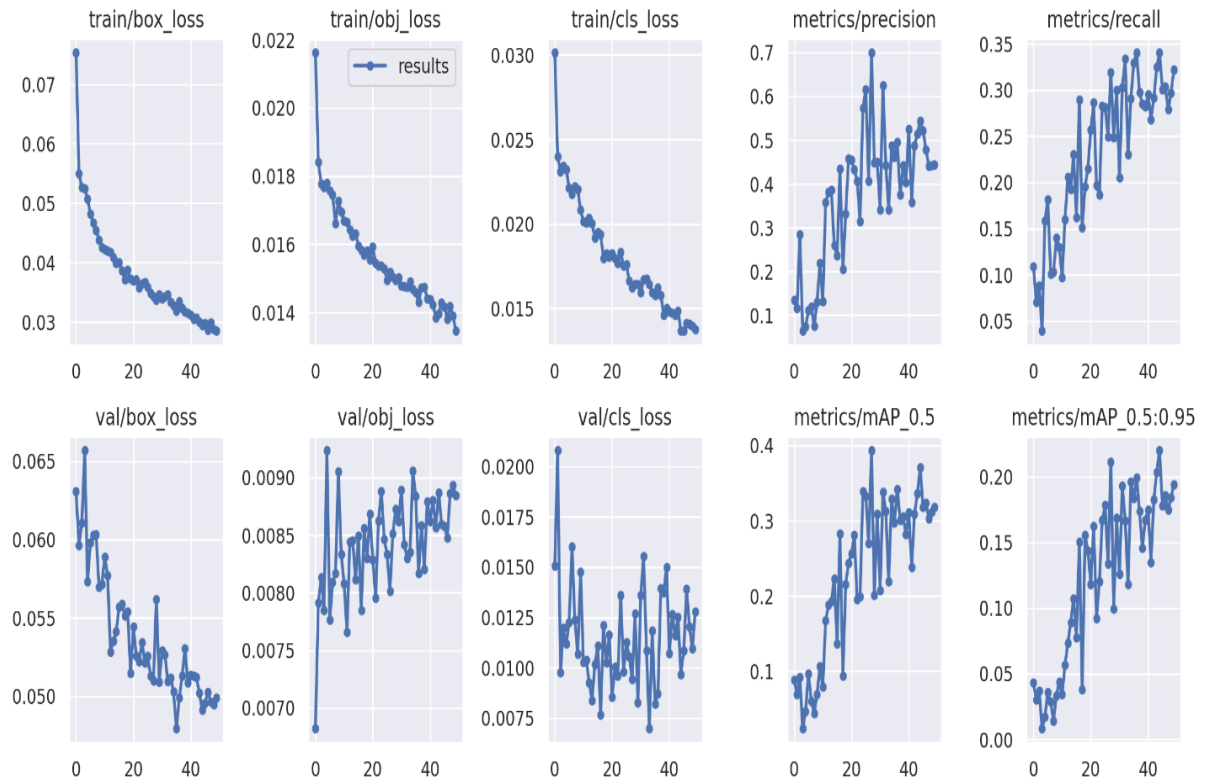


Fig.8.9 Region-based Convolutional Neural Network model evaluation



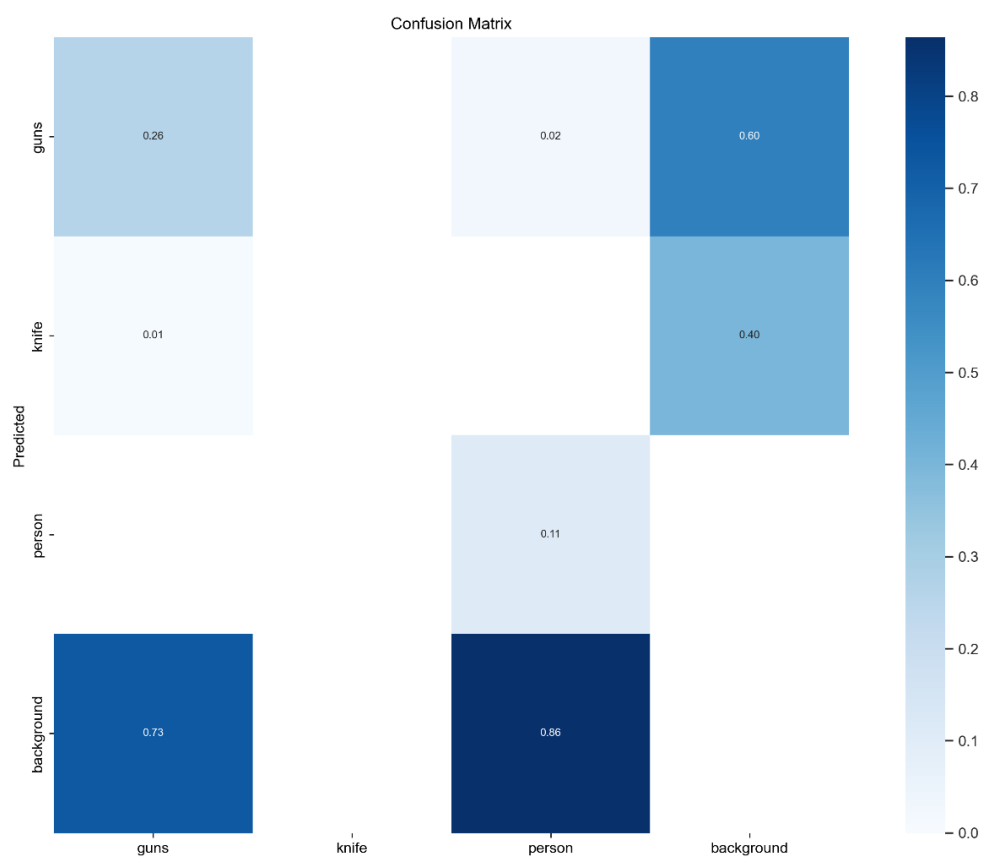


Fig.8.10 Confusion matrix for Region-based Convolutional Neural Network

### 3) You Only Look Once (YOLO V5)

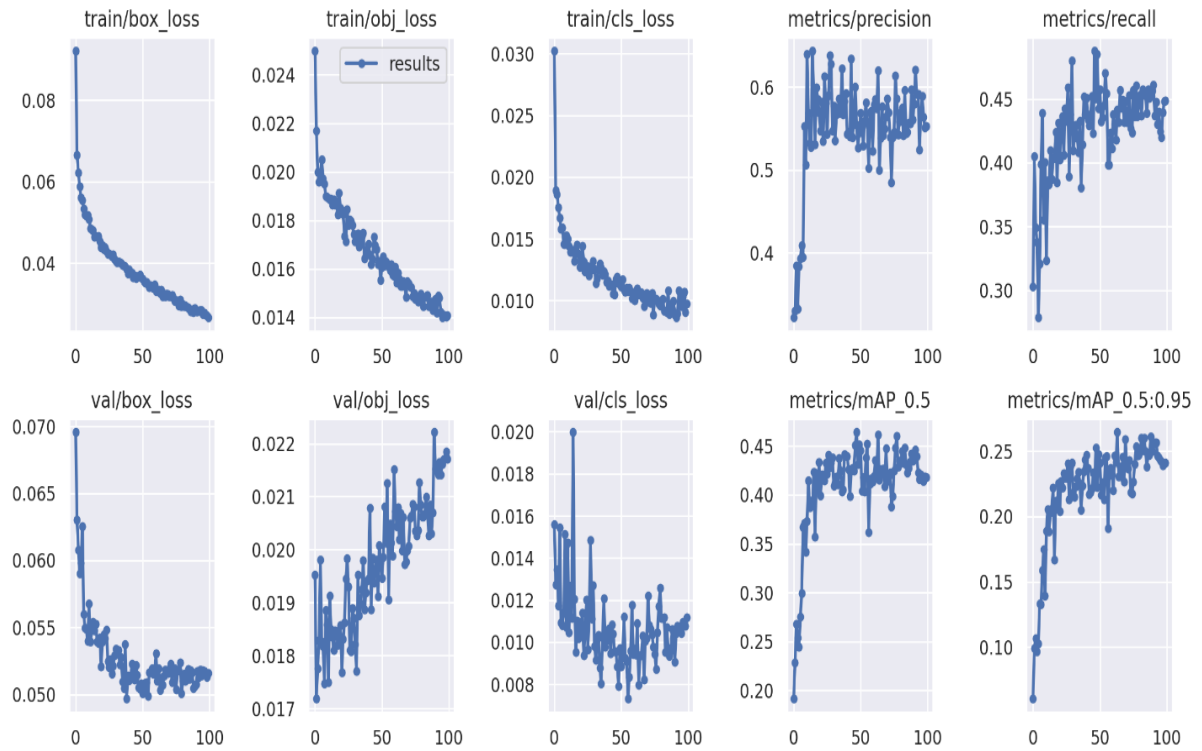


Fig.8.11 You Only Look Once model evaluation

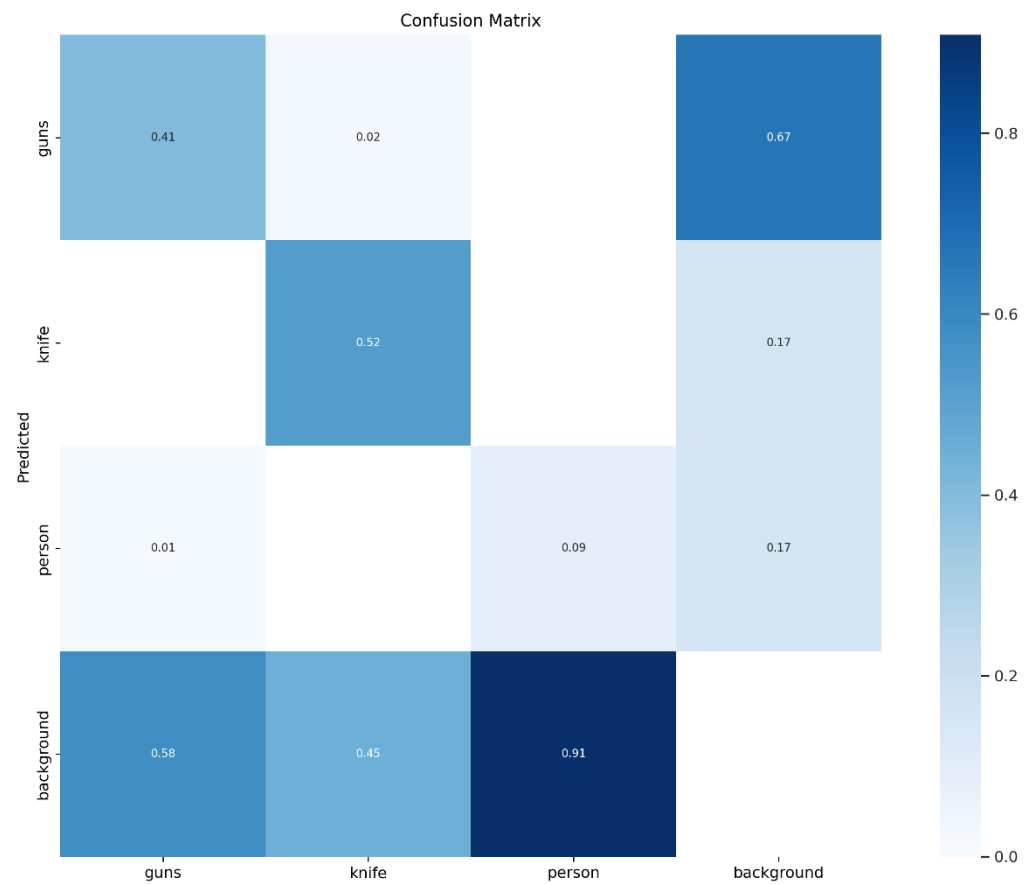


Fig.8.12 Confusion matrix for You Only Look Once (YOLO V5)

| <b>MODELS</b>                                     | YOLOv5 | R-CNN | SSD  |
|---------------------------------------------------|--------|-------|------|
| <b>mAP value</b><br><b>(Mean Precision Value)</b> | 56.2   | 47.1  | 36.7 |

Table 8.1 Comparative analysis of map values

# Chapter 9

## Conclusion and Future Work

### 9.1 Conclusion

For the implementation of this project, we used YOLOv5, RCNN, and SSD as different models for image processing and computer vision, and we are comparing their performance on our dataset. As mentioned, YOLOv5 has the best prediction accuracy, but it was slower than the other two models. YOLOv5 model included an affine-tuning approach to optimize its performance, and the lower computation time of RCNN might be desirable for real-time use cases. It is also important to note that the accuracy of the models can depend on the size and variety of the training set.

## 9.2 Future Work

1. Sensitivity and specificity of the proposed algorithms need to be further improved for real-time implementation.
2. Detection and Sending alerts to Police/Security team.
3. The algorithm was implemented on images and can be extended to recorded videos by collecting a sequence of frames to detect guns and knives from the video.
4. It can be further extended to detection in live video.
5. Installation can be done in cameras/drones for weapon detection.
6. We can predict crime for real-time implementation.

# Chapter 10

## References

- [1] Vaidehi, K., Subashini, T.: Automatic classification and retrieval of mammographic tissue density using texture features. In: 2015 IEEE 9th International Conference on Intelligent Systems and Control (ISCO), pp. 1–6. IEEE (2015).
- [2] C. Nosato, H., Sakanashi, H., Takahashi, E., Murakawa, M.: Method of retrieving multi-scale objects from optical colonoscopy images based on image-recognition techniques. In: 2015 IEEE Biomedical Circuits and Systems Conference (BioCAS), pp. 1–4. IEEE (2015).
- [3] Li, J., Ye, D.H., Chung, T., Kolsch, M., Wachs, J., Bouman, C.: Multi-target detection and tracking from a single camera in unmanned aerial vehicles (UAVs). In: 2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pp. 4992–4997. IEEE (2016).
- [4] Rao, R.S., Ali, S.T.: A computer vision technique to detect phishing attacks. In: 2015 Fifth International Conference on Communication Systems and Network Technologies (CSNT), pp. 596–601. IEEE (2015).
- [5] Herrmann, C., Beyerer, J.: Face retrieval on large-scale video data. In: 2015 12th Conference on Computer and Robot Vision (CRV), pp. 192–199. IEEE (2015).

[6] Sidhu, R.S., Sharad, M.: Smart surveillance system for detecting interpersonal crime. In: 2016 International Conference on Communication and Signal Processing (ICCSP), pp. 2003–2007. IEEE (2016).

[7] Crime Intention Detection System Using Deep Learning, Umadevi V Navalgund, Priyadharshini. K, Computer Science and Engineering, KLE Technological University, Hubballi, India.

[8] Inform and Variational Deep Learning for RGB-D Object Recognition and Person Re-Identification, Liangliang Ren, Jiwen Lu Jie Zhou, Senior Member, IEEE, and Jianjiang Feng, Member, IEEE.