Czech Technical University in Prague

Faculty of Electrical Engineering

Department of Computer Science

Security Camera Data Extraction and Presentation

Bachelor Thesis

2024

Author: Jiří Tůma

Supervisor: Doc. Ing. Stanislav Vítek, Ph.D.

BACHELOR‘S THESIS ASSIGNMENT

I. Personal and study details

Student's name: **Tůma Jiří** Personal ID number: **503162**

Faculty / Institute: **Faculty of Electrical Engineering**

Department / Institute: **Department of Computer Science**

Study program: **Software Engineering and Technology**

II. Bachelor’s thesis details

Bachelor’s thesis title in English:

Security Camera Data Extraction and Presentation

Bachelor’s thesis title in Czech:

Extrakce a prezentace dat z bezpečnostních kamer

Guidelines:

Bibliography / sources:

[1] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun, “Faster R-CNN - Towards Real-Time Object Detection with Region Proposal Networks”, vol. 3. January 6, 2016.
[2] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi, “You Only Look Once-Unified, Real-Time Object Detection”, 2016 IEEE Conference on Computer Vision and Pattern Recognition, vol. 5. May 9, 2016.
[3] “Ultralytics YOLOv8 Docs”. [Online]. Available at: https://docs.ultralytics.com

Name and workplace of bachelor’s thesis supervisor:

doc. Ing. Stanislav Vítek, Ph.D. Department of Radioelectronics FEE

Name and workplace of second bachelor’s thesis supervisor or consultant:

Date of bachelor’s thesis assignment: **15.02.2024** Deadline for bachelor thesis submission: **24.05.2024**

Assignment valid until: **21.09.2025**

doc. Ing. Stanislav Vítek, Ph.D. Supervisor’s signature	Head of department’s signature	prof. Mgr. Petr Páta, Ph.D. Dean’s signature

III. Assignment receipt

The student acknowledges that the bachelor’s thesis is an individual work. The student must produce his thesis without the assistance of others, with the exception of provided consultations. Within the bachelor’s thesis, the author must state the names of consultants and include a list of references.

.	
Date of assignment receipt	Student’s signature

Declaration

I declare that the presented work was developed independently and that I have listed all sources of information used within it in accordance with the methodical instructions for observing the ethical principles in the preparation of university theses.

In Prague 23.05.2024

.....

Acknowledgement

I would like to thank my supervisor for their invaluable guidance, insightful advice, and support throughout the project. Their expertise and encouragement have been instrumental in successfully completing this work.

Abstract

This thesis presents a comprehensive approach to enhancing security camera systems through advanced data extraction and real-time visualization. The proposed solution focuses on developing a graphical application that dynamically generates 2D maps to represent various environments based on the user input. The system is designed to capture and visualize the movement and localization of individuals and weapons within monitored spaces, ensuring efficient monitoring with a single-camera perspective. The core technology utilized is the YOLOv8 object detection model, which offers superior performance in terms of speed and accuracy. The application emphasizes universality, flexibility, and user-friendliness, making it suitable for diverse real-world deployments without the need for complex infrastructure. This project aims to bridge the gap between simple motion detection systems and sophisticated security solutions.

Keywords: security camera, object detection, YOLOv8

Abstrakt

Tato práce představuje přístup k vylepšení systémů bezpečnostních kamer prostřednictvím pokročilé extrakce dat a vizualizace v reálném čase. Navrhované řešení se zaměřuje na vývoj grafické aplikace, která dynamicky generuje 2D mapy různých prostředí na základě rozhodnutí uživatele. Systém je navržen tak, aby zachycoval a vizualizoval pohyb a lokalizaci jednotlivců a zbraní v monitorovaných prostorech, což zajišťuje efektivní monitorování z perspektivy jedné kamery. Základem aplikace je model detekce objektů YOLOv8, který nabízí vynikající výkon z hlediska rychlosti a přesnosti. Aplikace klade důraz na univerzálnost, flexibilitu a uživatelskou přívětivost, což ji činí vhodnou pro různorodé nasazení v reálném světě bez potřeby složité infrastruktury. Cílem tohoto projektu je přinést řešení postavené mezi jednoduché detektory pohybu a sofistikované bezpečnostní systémy.

Keywords: bezpečnostní kamera, detekce objektů, YOLOv8

Contents

1 Introduction

Security camera systems are increasingly used worldwide, with a growing need for simplified and streamlined monitoring processes. The two primary goals of this project are data extraction from a live camera feed and the presentation of essential data points in a simplified format. The system will monitor two object classes: person and gun/weapon. Various data about each object, such as their number and location, will be displayed to the user. To achieve this, a prototype application that can display a simplified view of a monitored space and allow the user to configure these areas was developed. This project aims to bridge the gap between simple motion detection systems and complex, high-end security solutions by offering a balanced approach that is both powerful and user-friendly. The final application is designed to be universally applicable, requiring limited computational resources for local execution, thus making it suitable for various real-world environments. Through the combination of object detection algorithms and an accessible presentation interface, this project seeks to improve the efficiency and effectiveness of modern security camera systems.

1.1 Related Work

This paragraph examines the related works in the academic field.

The Hawk-Eye [1] project focuses on threat detection in surveillance camera feeds. The authors decided to use the Intel NCS 2 device instead of the cloud to deploy their CNN and R-CNN models, with varying results. Using the CNN model, they were able to reach sub-10ms results for single-class classification.

Researchers in [2] chose a different approach. Their focus was on smaller indoor spaces. Using Mask R-CNN, they have managed to reach an average precision of up to 98.5%.

In a separate research endeavor [3], an SVS (Smart Video Surveillance) system was rigorously evaluated in a community college setting. The study showcased the system's robustness, effectively managing 16 CCTV cameras with a consistent 16.5 frames per second (FPS) over a 21-hour operation. Notably, the average end-to-end latency for detecting behavioral anomalies and alerting users was 26.76 seconds.

An innovative solution is presented in [4]. This system employs a Raspberry Pi to control motion-detecting sensors and video cameras, which stream live video and record it for future playback. Human presence is detected using a PIR sensor, which triggers the surveillance cameras only when movement is detected, thereby saving power and storage. The ViBe algorithm is used for motion detection in critical zones, sending an alert to users upon detecting suspicious activity.

This solution [5] focuses on the deployment of smart cameras with intelligent video analysis to enhance security. This implementation addresses early fire event detection, abnormal activities, smart parking systems, and crowd estimation. By utilizing machine learning techniques for video analysis, the system improves performance and event detection, offering real-time alert generation to overcome the drawbacks of traditional post-investigation methods.

Another study [6] addresses the critical issue of instantly detecting risky behavior in video surveillance systems. The proposed unified framework, based on a deep convolutional network, detects abnormal human behavior from standard images. The framework includes a human subject detection and discrimination module to separate object entities, a posture classification module to extract spatial features of abnormal behavior, and an abnormal behavior detection module based on long short-term memory (LSTM).

1.2 Related Commercial Products

Several companies today offer products with goals similar to those of this project. These range from basic motion detection in a security camera feed to advanced, fully autonomous AI camera systems.

Most home security cameras marketed as cloud or IP cameras come with a pre-built solution for data extraction. Companies such as Reolink, TP-Link, Blink, and many others offer similar software features, including motion detection [7]–[9], while the more computationally demanding features, such as person detection, are computed in a cloud and locked behind a paid subscription [10].

On the other side of the spectrum, there are services like the ones from Envysion and Solink. According to their description, Envysion's solution differs from others in that AI is no longer an added tool but the main focal point of the system. The feature set is greatly expanded, and the system can perform tasks such as object recognition and tracking, facial recognition, and behavior analysis [11]. The approach chosen by Solink involves interpreting data from the camera system and building a holistic management application around it. The user is not expected to look at the camera feed anymore. Instead, all the necessary data is provided through a user-friendly interface [12].

Konica Minolta offers comprehensive video security solutions designed to proactively deter crime and protect assets. Their approach involves using advanced video analytics to create REACT modules that enhance detection functionality. These modules allow customers to add features like visitor check-in to their video security plans. With intelligent internet protocol (IP) design, Konica Minolta's customized video surveillance systems address monitoring needs for indoor, outdoor, low-light, and challenging weather conditions. These systems not only collect data but also analyze and convert it into actionable information, providing a greater degree of intelligence. By leveraging decentralized software-driven solutions, their customers can effectively defuse highly dangerous situations and reduce liability through better video surveillance and real-time alerts [13].

Axis Communications, a long-time participant in network cameras and IP networking solutions, offers security products designed to protect people, property, and businesses. With decades of experience in cloud solutions and over a million Axis cameras deployed in the cloud, Axis is at the forefront of cloud capabilities. Axis provides scalable, IP-based products for security and video surveillance. These products offer intelligent analytics applications and real-time viewing. Additionally, Axis Cloud Connect, their new cloud-based platform, simplifies video surveillance management, offering enhanced security, flexibility, and scalability.

This project, positioned between the solutions mentioned earlier, offers distinct advantages. It provides more functionality and abstraction than mobile IP camera applications while still valuing the role of the human operator. It also operates with less computational power, allowing for local execution. The project incorporates some features from other solutions, such as person detection or movement tracking, but it also introduces unique elements, like 2D map projection or weapon detection, that enhance its value.

1.3 State of the art - Object detection

Object detection is a connecting term in almost all of the aforementioned works. It plays a crucial role in computer vision research. It harnesses the capabilities of computers to replicate human vision, enabling the identification of object categories and the localization of their positions. In recent years, significant progress has been made in object detection algorithms, especially those based on deep convolutional neural networks (CNNs). These advancements have led to a

gradual shift away from traditional methods like the Viola-Jones detector, Histogram of Oriented Gradients (HOG), and Deformable Part Models (DPM). There are two types of object detection algorithms depending on how often a network passes the same input image, i.e., single stage detectors and two stage detectors [14].

Single-stage detectors directly predict object bounding boxes and class labels in a single pass, while two-stage detectors first propose potential regions of interest and then refine those proposals in a subsequent stage. Both approaches have their strengths and weaknesses, and researchers continue to explore new techniques to improve object detection accuracy and efficiency [15].

Modern object detection algorithms have evolved significantly, focusing on improving efficiency and accuracy. Leading models like YOLO (You Only Look Once) and Faster R-CNN (Region-based Convolutional Neural Network) are widely used for real-time and robust object detection in various domains such as surveillance systems, smart cities, autonomous vehicles, and healthcare. These models showcase their versatility and effectiveness in addressing diverse visual recognition challenges [16].

1.4 Thesis structure

Based on the analysis conducted, I have developed a solution detailed in Section 2, Section 3, and Section 4. The final application is divided into two main parts, each with its own set of function requirements that are listed in Section 2.1, along with a comprehensive discussion of the project's theoretical concept in Section 2.

In Section 3, you will find a description of the first part of the application, which focuses on data extraction. This chapter also includes a detailed description of the object detection model and its training. It also dives into the specifics of how the data is extracted and processed to ensure the accuracy and efficiency of the application. Additionally, Section 4 provides a detailed description of the user interface and how it is designed.

2 Proposed Solution

The growing ubiquity of security camera systems and the accessibility of computational power have allowed a shift from passive surveillance to proactive monitoring. Traditional approaches, focused on retrospective analysis, fail to provide real-time insights into monitored environments.

This project addresses the problem by proposing a solution that employs an advanced object detection and image segmentation model. The goal is to enhance security camera systems, enabling the immediate extraction and visualization of relevant data. The solution's universality, flexibility, and simplicity aim to add to the landscape of security monitoring, offering a more proactive and comprehensive approach to ensuring safety and security in various environments.

The proposed solution involves creating a graphical application to present the outcomes of data collected from security camera systems. This application will facilitate the generation of 2D maps representing various environments. These maps will dynamically showcase data extracted from camera recordings in real-time. The application's primary focus is to capture and visualize the movement and localization of individuals within the created monitored environments.

Emphasis will be placed on ensuring the application's universality, flexibility, and user-friendly interface. A key feature is achieving full functionality using only a single camera perspective for a given area. This approach eliminates the need for complex infrastructure or specialized equipment, enabling easy application implementation in diverse real-world environments and situations.

2.1 Functional Requirements

- The system must enable the immediate extraction and visualization of relevant data from security camera footage.
- The system must facilitate the generation of 2D maps representing various environments.
- The system must ensure a user-friendly control interface.
- The system must display the original feed and the generated map in two separate windows.
- The system must capture and visualize the movement and localization of individuals within the monitored environments.
- The system must capture and visualize the movement and localization of weapons within the monitored environments.
- The system must be universally applicable and flexible in deployment.
- The system must achieve full functionality using only a single camera perspective for a given area.

2.2 Block Diagram

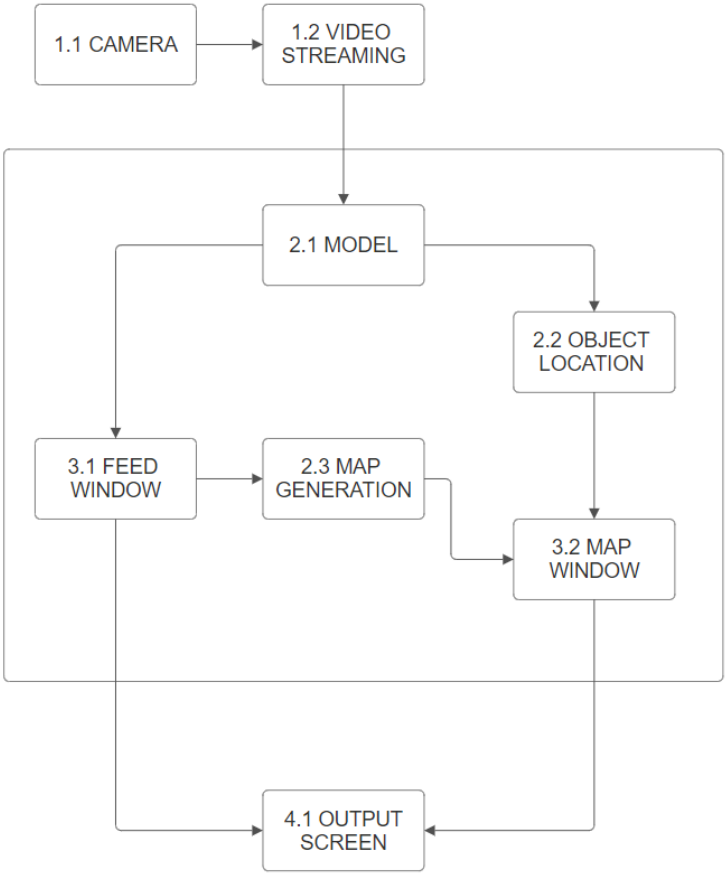

Figure 1: Block diagram

- 1.1 Camera - A camera capturing the original video stream of a monitored space
- 1.2 Video streaming - A service that streams the video from the camera
- 2.1 Model - An object detection model, YOLOv8 in this case
- 2.2 Object location - A process that transforms results from the model inference to coordinates and other data in the map window
- 2.3 Map generation - A process that generates a 2D representation of the user-defined space
- 3.1 Feed window - A window showing the camera feed with annotated objects
- 3.2 Map window - A window showing the 2D representation of the user-defined space
- 4.1 Output screen

The subjects of this project are only the highlighted blocks 2 and 3.

3 Data extraction

This part of the application involves the extraction of data from a designated video source. The service is designed to scan and analyze the video content, specifically focusing on identifying objects and their respective locations within the images.

After each video frame is processed, the results are streamed to other parts of the application, where they are further transformed and displayed to the user.

The goals of the Data Extractor:

- Label desired objects in the video stream
- Retrieve the location of objects' bounding boxes
- Retrieve the objects' classes
- Count the number of objects

This is a data flow chart of the whole application; the UI part is explained in Section 4.

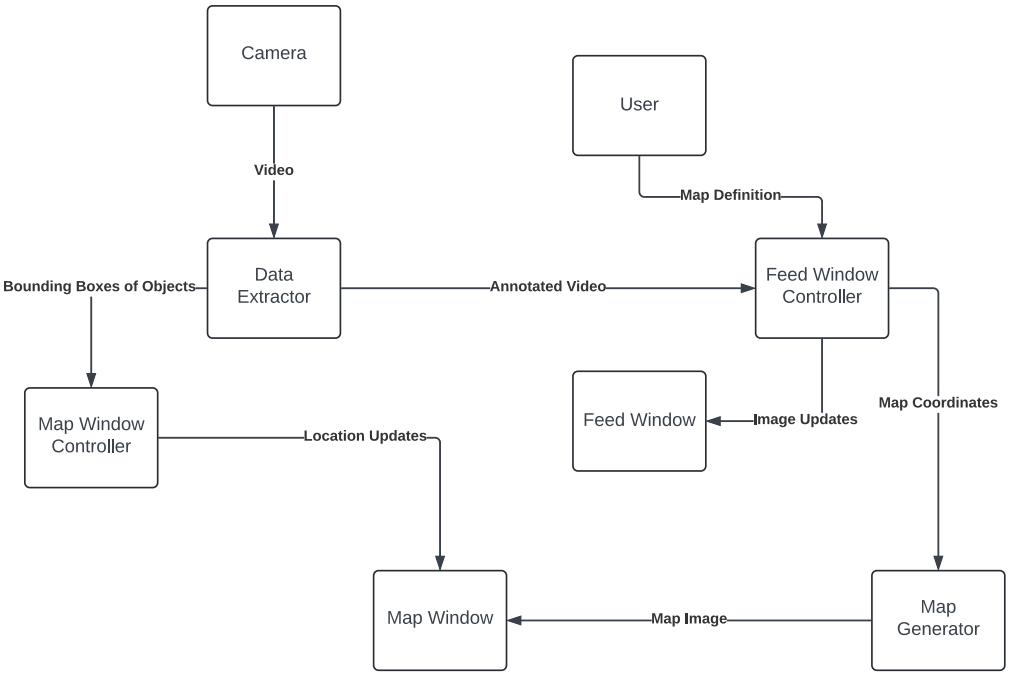

Figure 2: System data flow diagram

3.1 Model

After conducting the analysis, I narrowed the choice down to two different architectures: YOLO and Fast R-CNN. I had previously worked with tools from Ultralytics, the company behind products such as YOLOv5 and YOLOv8 [17], so I decided to go with YOLO.

In 2015, Joseph Redmon and Santosh Divvala introduced the YOLO (You Only Look Once) algorithm in their paper titled "You Only Look Once: Unified, Real-Time Object Detection." This groundbreaking algorithm revolutionized real-time object detection by offering a novel approach. Unlike previous methods that repurpose classifiers for detection, YOLO frames object detection as a regression problem. It directly predicts bounding boxes and class probabilities from full images in one evaluation. The unified architecture allows end-to-end optimization for detection performance. The base YOLO model achieves real-time processing at unmatched speeds with double the mean average precision (mAP) of other real-time detectors. Although YOLO may make more localization errors, it is less likely to predict false detections where

nothing exists. Additionally, YOLO learns general representations of objects and outperforms other detection methods on natural images and artwork datasets [18].

3.1.1 History of Yolo

The first YOLOv1 model created by Joseph Redmon and his team [18] was capable of detecting objects in images with a single pass of the network. This was a significant improvement over previous methods that either used sliding windows followed by a classifier that needed to run hundreds or thousands of times per image or more advanced methods that divided the task into two steps: first detecting possible regions with objects or region proposals, and then running a classifier on the proposals. The model first divides the original image into a grid and then predicts the bounding boxes of classes within each grid tile. One of the limitations of this model was that it only allowed for a maximum of two classes to be predicted in one tile [18].

The Yolo algorithm has undergone many iterations over the last couple of years, starting with better performance in YOLOv2, architectural separation into the now-used Backbone, Neck, and Head in YOLOv3, or the switch from Darknet to Pythorch between the versions v4 and v5. Finally arriving at YOLOv8 from Ultralytics. This version builds upon the foundation laid by YOLOv5, refining its architecture and achieving better performance in terms of accuracy and speed. YOLOv8 shares a similar backbone architecture with YOLOv5, but it introduces changes to the CSPLayer (now called the C2f module). This modification combines high-level features with contextual information, leading to improved detection accuracy [17].

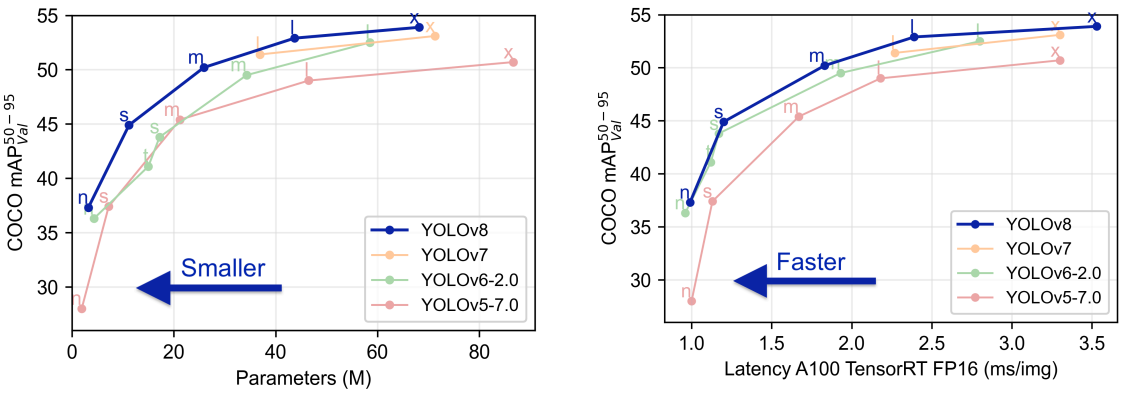

Figure 3: Performance comparison of YOLO object detection models by Ultralytics [19]

The left plot illustrates the relationship between the number of parameters in a YOLO model and its detection accuracy on the COCO dataset. As the number of parameters increases, the detection accuracy tends to improve.

The plot on the right shows the tradeoff between inference speed and accuracy for the YOLO models. The letters “n,” “s,” “m,” “l,” and “x” correspond to different variants of the YOLO model, more in Section 3.1.3.

The YOLOv8 was the most advanced model from Ultralytics at the time of implementation, so it was selected for this project. This model is the subject of the following Section 3.1.2 and Section 3.1.3

3.1.2 Architecture

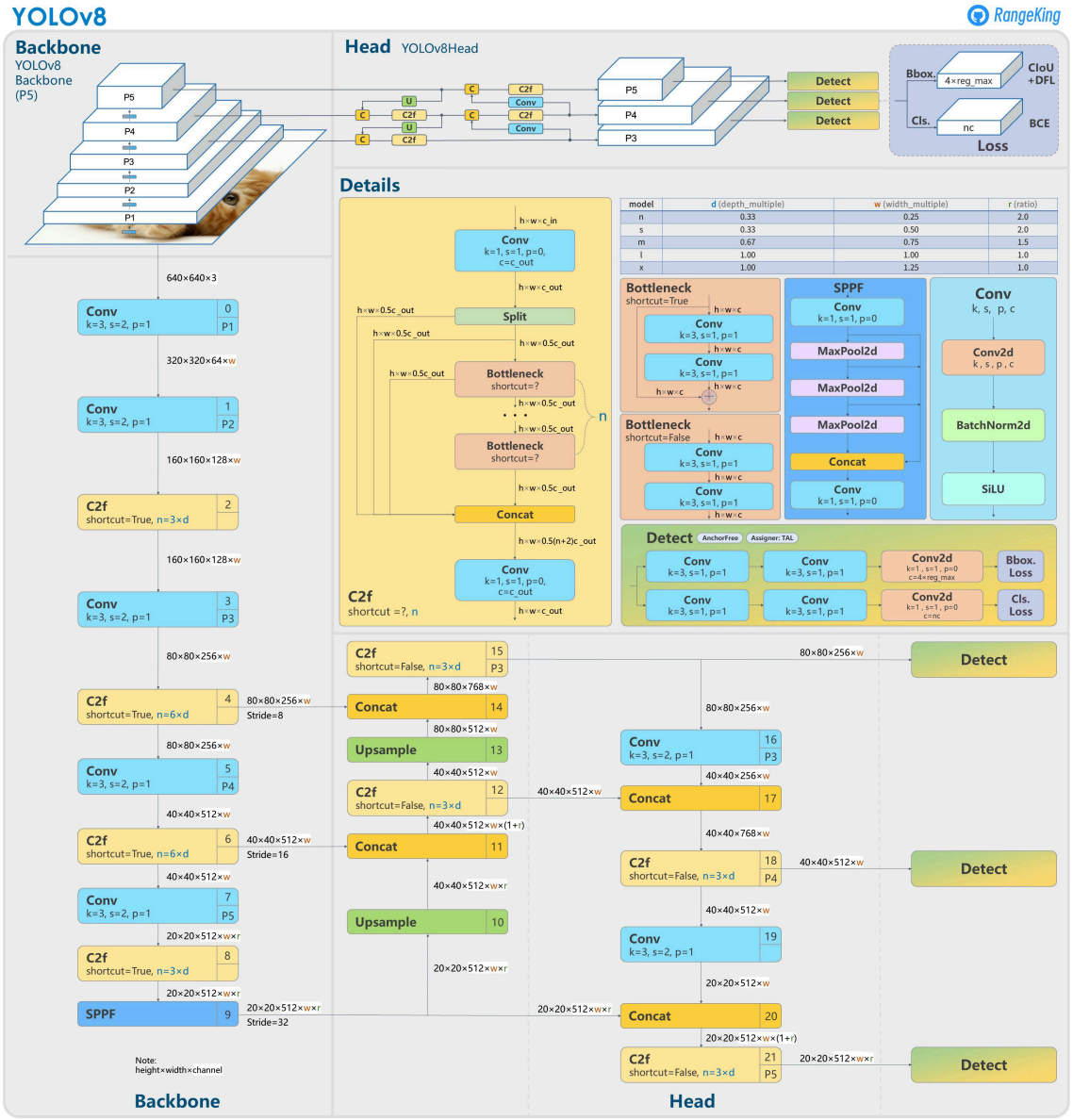

Figure 4: Summary of YOLOv8 model structure by RangeKing [20]

The architecture consists of three essential blocks: Backbone, Neck, and Head.

Backbone:

A series of convolutional layers. The Backbone is built upon a custom CSPDarknet53 architecture, which utilizes cross-stage partial connections to enhance the flow of information between layers and improve accuracy. This part of the network is responsible for extracting features from the input image [21].

Neck:

The Neck serves as a feature extractor, merging feature maps from different stages of the Backbone to capture information at various scales. YOLOv8 introduces a novel C2f module, which combines high-level semantic features with low-level spatial details, leading to better detection accuracy, particularly for small objects [21].

Head:

The Head is where the predictions are made. The detections are based on loss metrics. YOLOv8 employs multiple detection modules that predict bounding boxes, objectness scores, and class probabilities for each grid cell in the feature map. These predictions are then aggregated to produce the final detections [21].

3.1.3 Tasks and variants

The YOLOv8 series offers five different models individually optimized for their tasks. All of the models are capable of various operational modes: Inference, Validation, Training, and Export, each designed to be used in different stages of deployment and development.

Models for each task also come in several different variants: nano, small, medium, large, and extra large. The names of the models refer to the number of parameters they have, which in turn affects their speed and quality of inference. As the size of the model increases, the speed of the inference decreases but the quality improves [19]. The tasks are:

Detection:

Detection is the primary task supported by YOLOv8. It is also the task that was used in this project. As such, it will be explored in greater detail than other tasks.

Object detection, as explained in Section 1.3, involves identifying the class and location of an object. YOLOv8 is fully capable and excels at identifying multiple objects in a single image. The output is the coordinates of bounding boxes around the object together with the class ID and class confidence for each object found. The results can be visualized in a similar manner, as shown in the following picture.

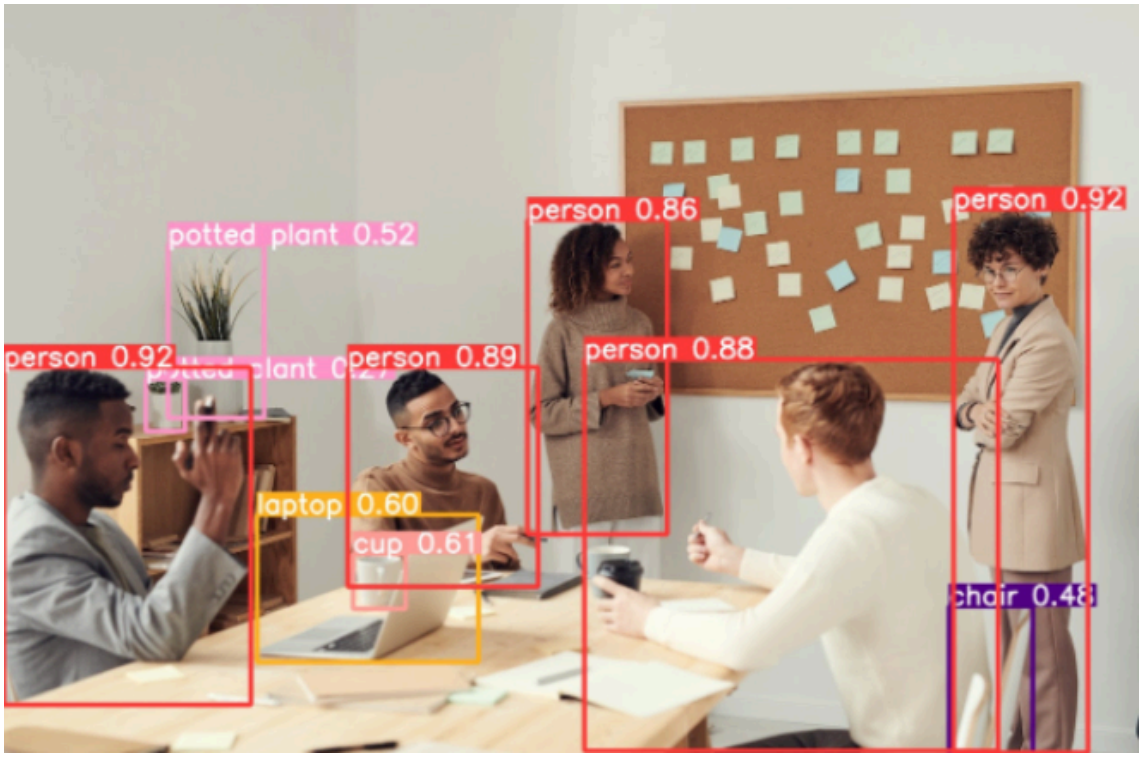

Figure 5: Bounding boxes visualized [22]

The model can be trained as described in Section 3.3, but Ultralytics also offers two pre-trained versions. The first version is trained on the Open Images V7 dataset and recognizes 600 classes.

The second includes 80 classes from the COCO dataset. The table below shows the speed and average precision for different-sized models trained on the COCO dataset [19]:

Table 1: YOLOv8 variant comparison

Model	Parameters (M)	Size (pixels)	mAP	Speed A100 (ms)
YOLOv8n	3.2	640	37.3	0.99
YOLOv8s	11.2	640	44.9	1.20
YOLOv8m	25.9	640	50.2	1.83
YOLOv8l	43.7	640	52.9	2.39
YOLOv8x	68.2	640	53.9	3.53

Instance Segmentation:

Instance segmentation is a computer vision task that involves identifying and delineating each object within an image. The YOLOv8 models designed for this task produce class labels and confidence scores, the same as object detectors, but also precise object masks. They are used in cases where understanding the exact shape of an observed object is essential.

The models are pre-trained on the COCO dataset [19].

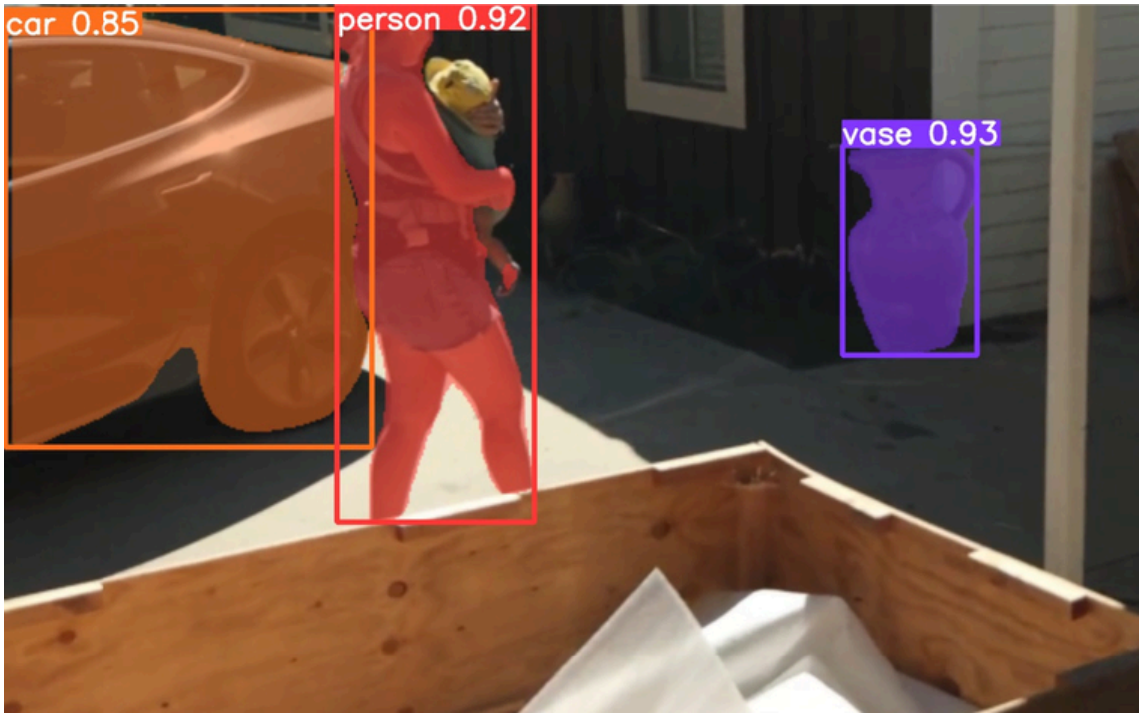

Figure 6: Image segmentation task visualized with three different object classes

Pose/Keypoints:

Pose estimation involves identifying points, commonly known as keypoints, in an image. The keypoints usually mark the location of distinctive features. For example, in human pose estimation, the keypoints might include the nose, ears, shoulders, elbows, wrists, hips, knees, and ankles. These keypoints help determine the spatial arrangement of a person's body in an image or video.

The models are pre-trained on the COCO dataset [19].

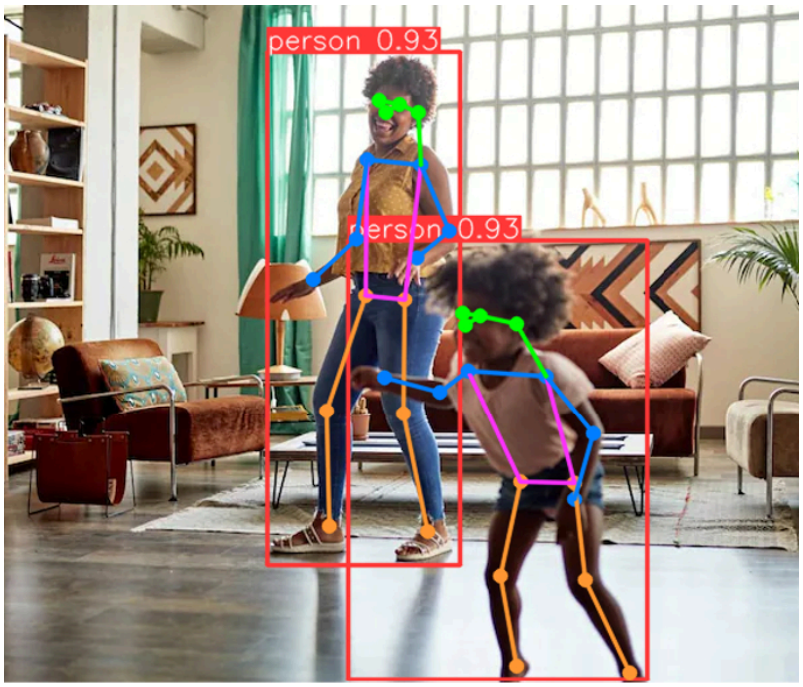

Figure 7: Pose task visualized

Oriented Detection:

Oriented Bounding Boxes Object Detection returns a set of oriented bounding boxes. These boxes closely envelop the observed object and thus provide more precise localization of objects.

The models are pre-trained on the DOTAv1 dataset [19].

OOB would have been a better fit for this project than plain object detection, but this task was not fully available at the time of development.

Figure 8: Oriented Bounding Boxes Object Detection task visualized [19]

Classification:

Image classification is the most straightforward of the three tasks and entails assigning an entire image to one of several predefined classes. The result from an image classifier consists of a single class label along with a confidence score. This technique is particularly useful when you only need to determine the category to which an image belongs without requiring information about the specific location or shape of objects within that image.

The models are pre-trained on the ImageNet dataset [19]. This dataset includes 1000 object classes [23].

3.2 Python environment

This chapter focuses on the dependencies and technologies needed to run the models.

The model provides a simple yet powerful command-line interface (CLI) and Python interfaces. The simplest approach is to use the Python interface because the rest of the application can also be written in Python, and thus, no communication between components written in different programming languages is required.

The Ultralytics documentation recommends using Conda instead of Pip for package management. Conda is a powerful package, dependency, and environment management tool that is supported by multiple programming languages. Isolated environments can be created where specific packages can be installed and their dependencies managed. With Conda, different project environments can be easily switched between, ensuring that dependencies are not in conflict. It is particularly popular in the data science and scientific computing communities [24].

Both Python and Conda packages can be installed through the Anaconda installer, along with many other packages used for data science, such as Numpy and Pandas [25].

The Ultralytics package is installed via Conda, providing the necessary tools for using the models. This package offers the main functionality for this part of the application and will be extensively discussed in Section 3.4.

The CUDA (Compute Unified Device Architecture) toolkit is needed to be able to deploy the models on Nvidia GPUs through the CUDA API. CUDA is a parallel computing platform and programming model developed by NVIDIA, enabling software to leverage CUDA-enabled GPUs for accelerated general-purpose processing, known as general-purpose computing on GPUs. The CUDA Toolkit offers a development environment for building high-performance, GPU-accelerated applications. [26].

Another component of the environment is Pytorch. PyTorch is a versatile machine-learning library that strikes a balance between usability and performance. It adopts an imperative and Pythonic programming style, making it easy to debug and consistent with other popular scientific computing libraries. Researchers and practitioners widely use PyTorch for deep learning research, prototyping, and model development. Its dynamic computation graph allows for flexible model design, while its efficient execution supports hardware accelerators like GPUs. PyTorch emphasizes user control, and every aspect of it is a regular Python program. [27]

3.2.1 List of important technologies utilized

- Anaconda Installer
- Conda package manager
- CUDA toolkit
- Python 3.10
- Pytorch
- Ultralytics Python package

3.3 Training

While the YoloV8 models are pre-trained for each task, Ultralytics also offers a way to train the model on a custom dataset. It is recommended to start training with the pre-trained weights [19]. The object detection model used in this project has original weights trained on the COCO dataset.

It is possible to select the device on which the training will be done. Since it is much more time and cost-efficient to use the GPU rather than the CPU for these kinds of tasks [28], the training was done on the GPU, RTX4090, in this case. The Train Mode also supports Multi-GPU training, as it is optimized to distribute the training load across multiple devices.

The model offers a user-friendly command-line interface (CLI) and Python interfaces for easy training. I have decided to use the Python interface since the rest of the project is also written in Python. Model performance can be fine-tuned by customizing hyperparameters through YAML configuration files or CLI arguments. The YAML configuration file must contain at least these parameters:

```yaml
path: /absolute/path/data
train: images/train
val: images/val

names:
  0:
```

Code preview 1: Example of a training configuration file

The first parameter specifies the path to the dataset for training. The train and val parameters are relative paths from the path parameter to the training and valuation data. Both the train and val folders must consist of the Images directory and the Labels directory; see Section 3.3.1.

Another crucial parameter is the number of epochs the model should train for. The original training for the YOLOv8 Object Detector was completed in 500 epochs on the COCO dataset [19]. The recommended number of epochs on differently-sized datasets varies in every source. Ultralytics also does not provide a clear answer to this question, so I decided to train the model on as many epochs as possible in a reasonable time.

```python
from ultralytics import YOLO

model = YOLO('gun.pt')


if __name__ == '__main__':
    results = model.train(data="config.yaml", epochs=10)
```

Code preview 2: Example of a training python script

The final training file in Python is quite simple. Since one epoch took around 40 minutes to complete (GPU VRAM usage limited to 16GB), the final training was completed in ten epochs. The largest YOLOv8x model was chosen with the assumption of best results; see Table 1.

3.3.1 Dataset

The majority of the dataset is comprised of short weapons and a smaller number of long weapons, both of which are tracked under one class. The dataset was obtained from [29] and is the largest publicly available dataset focusing on gun detection.

The dataset is split into training data and validation data. The training data contains more than fifty thousand different labeled images. The labels are in a format different from the one required by the YOLOv8 training interface, so I wrote a simple Python script to convert them.

```
<?xml version="1.0" encoding="UTF-8"?>
<annotation>
    <filename>ia_200000004.jpg</filename>
    <size>
        <width>1920</width>
        <height>1080</height>
        <depth>3</depth>
    </size>
    <object>
        <name>gun</name>
        <bndbox>
            <xmin>1287</xmin>
            <ymin>81</ymin>
            <xmax>1675</xmax>
            <ymax>800</ymax>
        </bndbox>
    </object>
</annotation>
```

Code preview 3: Original label example

This label shows data about one object. The bounding boxes are defined as two points (X and Y coordinates): the top left and bottom right corners. The desired format by YOLO consists of one line for each object in the image. This line starts with the class ID, followed by the midpoint of the bounding box, as well as the width and height of the bounding box. All of the coordinates are normalized to the height and width of the image. The final label looks like this:

```
0 0.771354 0.407870 0.202083 0.665741
```

Code preview 4: Transformed label example

The converter starts by iterating through the directory where the label files are located. Then, it opens each file one by one and finds and extracts all the data. Every object is processed separately and the output of each is a line in the output file. The Python XML library was used in this snippet to find the maximum and minimum values for the X and Y coordinates for each object in the file.

```
with open(xml_file, 'r') as file:
      xml_string = file.read()
      root = ET.fromstring(xml_string)
      objects = root.findall(".//object")
      image_width = root.find(".//width").text
      image_height = root.find(".//height").text

      for obj in objects:
          xmin = obj.find(".//xmin").text
          ymin = obj.find(".//ymin").text
          xmax = obj.find(".//xmax").text
          ymax = obj.find(".//ymax").text
```

Code preview 5: Label data extraction code snippet

The location data gathered in the previous step is then transformed into the desired format, i.e. bounding box midpoint, width, height.

```
def calculate_coordinates(xmin, ymin, xmax, ymax):
  x_center = (int(xmin) + int(xmax)) / 2
  y_center = (int(ymin) + int(ymax)) / 2
  width = abs(int(xmax) - int(xmin))
  height = abs(int(ymax) - int(ymin))
  return x_center, y_center, width, height
```

Code preview 6: Bounding box midpoint calculator

The coordinates are then normalized.

```
def normalize_coordinates(x_center, y_center, width, height,
image_width, image_height):
  x_center = round(x_center / int(image_width), 6)
  y_center = round(y_center / int(image_height), 6)
  width = round(width / int(image_width), 6)
  height = round(height / int(image_height), 6)
...
  return x_center, y_center, width, height
```

Code preview 7: Bounding box midpoint coordinates normalization

Finally, the new line is assembled. Class ID is always equal to 0 because this dataset only includes one class. The coordinates are formatted so that they have six decimal places. Once all objects have been processed, the file is saved with the same name as the original XML file, as the image and label files are paired by their names.

3.3.2 Training results

These are the results of the training and validation. The number of epochs was set to ten, and the process took around six hours.

One of the terms used in this section is a loss function. A loss function and performance metrics are both used to evaluate the performance of a deep learning model, but they serve different purposes. The loss function is utilized during training to optimize the model's parameters. It calculates the difference between the predicted and expected outputs of the model, and the objective of the training is to minimize this difference [30].

Training metrics:

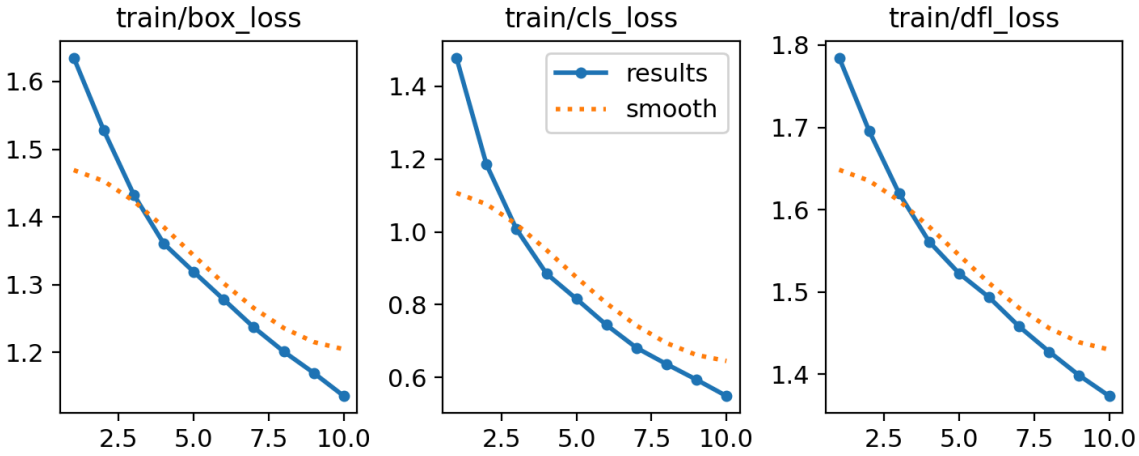

Figure 9: Training metrics showing bbox, class, and

- train/box_loss:

This plot shows the box loss during training. The box loss is decreasing steadily from about 1.6 to 1.2, indicating that the model is learning to predict bounding boxes more accurately.

- train/cls_loss:

This plot shows the classification loss during training. It decreases from approximately 1.4 to 0.6, suggesting that the model is getting better at classifying the objects correctly as the training goes on.

- train/dfl_loss:

This plot shows the distribution focal loss during training. The loss decreases from around 1.8 to 1.4, showing improvement in the precision of bounding box predictions. Opposite to the first loss function, this bounding box regression focuses on bounding box offsets rather than their values.

Validation metrics:

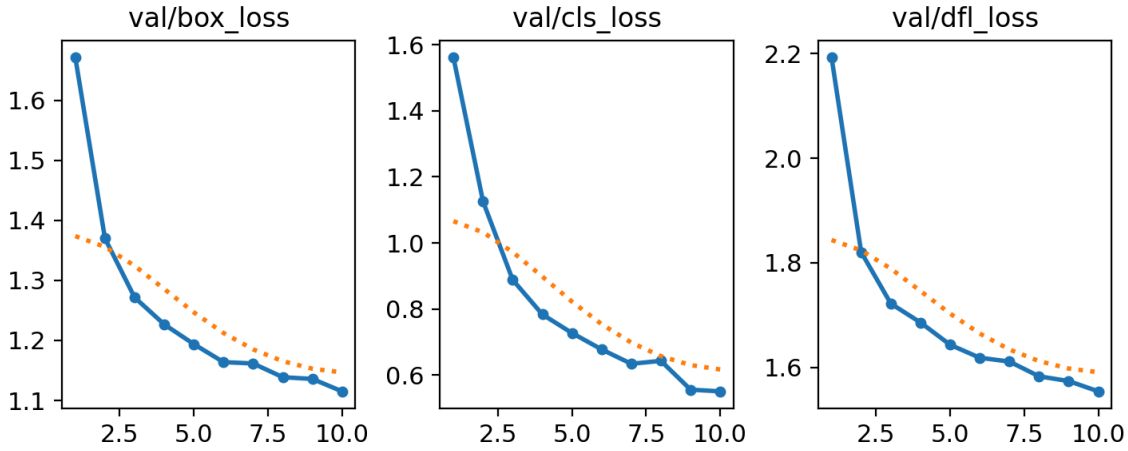

Figure 10: Validation metrics showing bbox, class, and

The training and validation curves are of a similar character and do not diverge at the end by a great margin, suggesting that ten epochs were not too many and the model is not "overfitted" [31]. On the other hand, neither of the curves starts to even out at the end, implying that the model could have been trained for even more epochs.

3.3.3 Model evaluation

A confusion matrix is a performance measurement for image classification. It is usually used for two or more classes. In this case, there is only one class, gun, so the background was plotted as the second class. There are 1708 validation samples. The following is a table representing the data extrapolated from the confusion matrix.

Table 2: YOLOv8 confusion matrix table

	True Positives	False Negatives	False Positives	True Negatives
Number of samples	1499	55	154	0
Normalized	96%	4%	100%	0%

- The first column shows the True Positives (TP), which means that the model classified a gun in an image where there was a gun. There are 1499 cases.
- False Negatives (FN) are represented in the second column. These are the samples that show a gun, but the inference results show none. There are 55 cases.
- False Positives (FP) are instances in which there was no gun, but the model found one. These are displayed in the third column. There are 154 cases.
- The last column displays no number because there were no True Negatives (TN) in the validation set. A True Negative occurs when there are no objects in the image, and the classification also returns no objects.

The second row shows the same results as the first; the data has just been normalized. This data shows that there are only 4% True Negatives, and all False Positives were classified correctly, which is an acceptable result.

Several different metrics can be calculated from these four values. Accuracy is defined by this formula:

$$\text{Accuracy} = (\text{TP} + \text{TN})/(\text{TP}+\text{FP}+\text{TN}+\text{FN})$$

It's important to note that accuracy alone may not provide a clear interpretation of the results, especially when dealing with imbalanced datasets. For instance, in cases such as fraud detection, where there is only one fraud case among thousands of cases, predicting everything as non-fraudulent would result in very high accuracy, even though it's not a meaningful indicator of model performance. This is because the number of True Negatives overwhelms the other results, leading to an accuracy of 100% [32]. Other metrics avoiding the True Negative value are used. One of these metrics is precision. Precision should be the key metric when trying to minimize false detections [19]. This is the formula for precision:

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}$$

The values can be plotted in a graph to help understand how the precision varies as the confidence threshold changes.

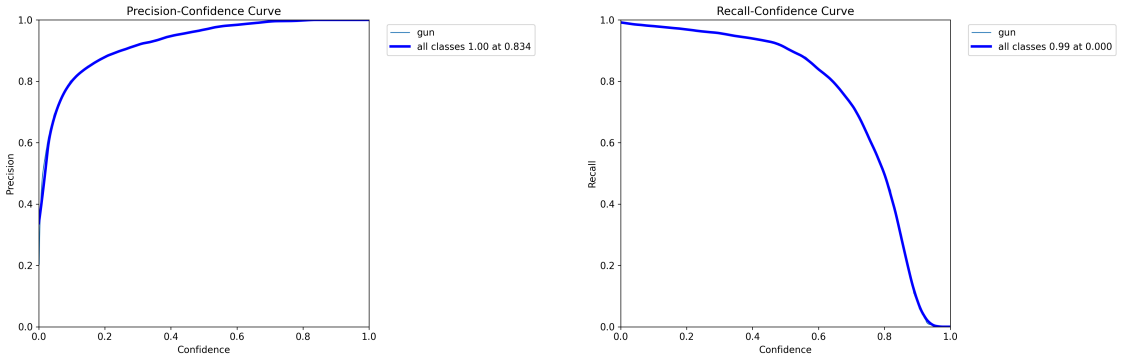

Figure 11: Precision-confidence curve on the left and Recall-confidence curve on the right

The first graph shows that the precision is around 100% at a confidence level of 0.834.

On the other hand, when it is more important to detect every object, even at the expense of more false detections, the recall metric should be used [19]. The formula for recall differs in just one variable:

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

The second graph illustrates how the recall values change across different confidence thresholds.

As confidence increases, recall decreases. This means that the model becomes more selective and misses some actual guns. The curve flattens out as it approaches high confidence values, indicating that the model becomes very confident but sacrifices recall (misses more actual guns).

The trade-offs between precision and recall can be illustrated with a Precision-Recall Curve. The model is considered a good predictive model if the precision stays high as the recall increases [32].

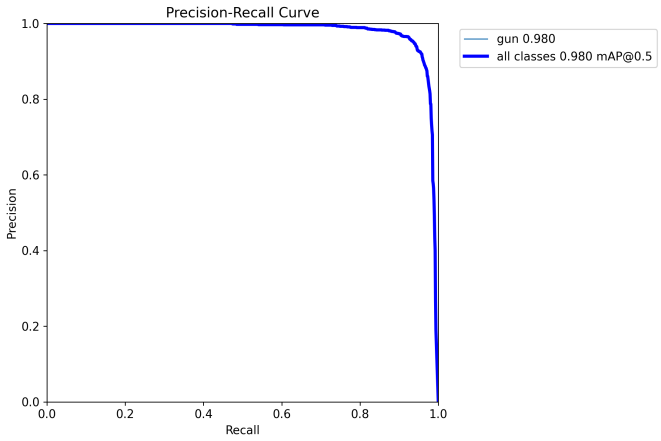

Figure 12: Precision-Recall Curve

This image shows that as recall increases, precision remains consistently high for both curves. This means that the model maintains accurate detections (high precision) across different recall levels.

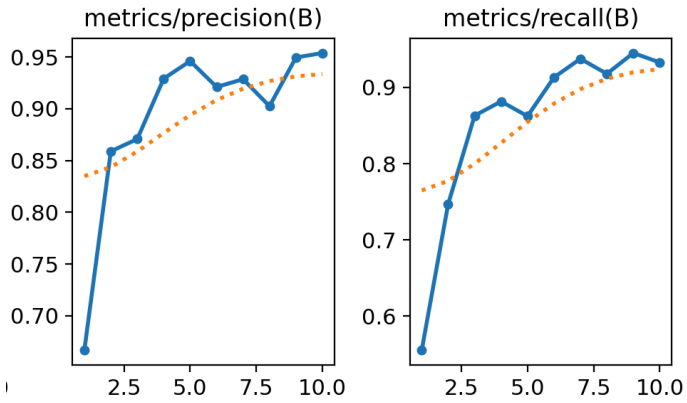

Figure 13: Precision and recall development over 10 epochs

These graphs show the development of precision and recall over the span of all ten of the training epochs. Although some steps are in the negative direction, the general trend is that both curves are climbing.

The F1 metric is calculated from the aforementioned values. The formula is:

$$\text{F1 scrore} = \frac{\text{Precision} * \text{Recall}}{\frac{\text{Precision} + \text{Recall}}{2}}$$

It is a weighted average of precision and recall. Values only range from 0 to 1, where 1 signifies the highest possible accuracy.

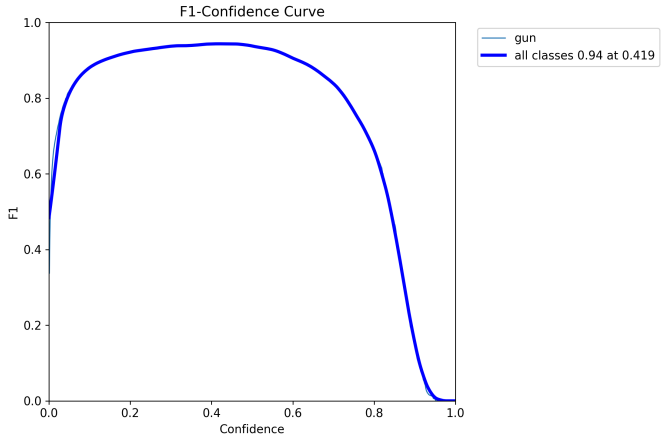

Figure 14: F1-Confidence Curve

On the left side, the curve rises steeply to reach a peak near the middle of the graph, suggesting an optimal balance between precision and recall. As confidence increases beyond the peak, the F1 score gradually declines. This means that the model becomes more confident but sacrifices the balance between precision and recall.

The model overall performs well when measured on the validation dataset. These two images show the difference between the original annotated validation images and labels created by the model:

Figure 15: Validation image with original labels

Figure 16: Validation image with model-generated labels

3.4 Usage

The model is used within a separate thread from the rest of the application. The video stream is processed one image at a time. The results are then emitted to the different windows and displayed. Each image is processed with two different models, one trained on the COCO dataset and the other on the custom dataset. Both of these models retrieve one class. The inference on each frame is executed by this line:

```
# Run YOLOv8 inference on the frame
results = self.model.track(frame, classes=0, persist=True,
conf=0.2)
```

Code preview 8: Inference on a frame

There are three main data gathered from the inference:

Labels and bounding boxes for each object:

Both the bounding boxes and the labels are drawn on the original image. The image is then restructured to fit the PyQt framework (more in Section 4) and sent to the feed window.

```
# Image transformation
def draw_image(image):
          image     =     QImage(image,     image.shape[1],
image.shape[0],image.strides[0], QImage.Format.Format_BGR888)
  return image

# Visualize the results on the frame
annotated_frame = results[0].plot()

# Emit the transformed image
self.image_update.emit(draw_image(annotated_frame))
```

Code preview 9: Results of the inference are sent further

Objects' location on the map:

When the map window is active, the results are used to draw the objects. The map window size is dictated by the area set up by the user (see Section 4), which is why there are multiple offsets and magnifiers used in the process of drawing

```
# Draw the objects on a frame
def draw_units(self, original_image, boxes):
  result = original_image.copy()
  for box in boxes:
      centre_x = int((box.xywh[0][0] - self.offset_x + MAP_OFFSET) )
      centre_y = int((box.xywh[0][1] + int(box.xywh[0][3] / 2) -
self.offset_y + MAP_OFFSET) )
      color = original_image[centre_x, centre_y]
      if color[0] == 255 and color[1] == 255 and color[2] == 255:
          cv2.circle(result,
                     (centre_x * MAGNIFIER, centre_y * MAGNIFIER),
10, (0, 0, 255), 30)
      else:
          cv2.circle(result,
                (centre_x * MAGNIFIER, centre_y * MAGNIFIER), 10,
(255, 0, 255), 30)

  return result

# Send the bounding box location to the map window
if self.annotated_map is not None:
  map = self.draw_units(self.annotated_map, results[0].boxes)
```

Code preview 10: Draw units method and its usage in the creation of the map

Number of objects:

The number of objects is calculated by adding up the number of bounding boxes in the person class. A variable is then updated, which in turn updates the number shown in the UI when the window is refreshed.

4 User Interface

The second part of the application aims to deliver the data collected from Section 3 to the user through a simplified and user-friendly interface. The interface is split into two windows - the map window and the feed window.

The goals of the user interface:
- Display the annotated video feed
- Facilitate the creation of the 2D maps
- Generate the 2D map representations
- Display the maps in a separate window
- Display the number of objects
- Display the observed objects in the map window
- Allow the user to control the application

The entire user interface is built using the PyQt6 framework. PyQt6 provides Python bindings for the Qt application framework, enabling the creation of cross-platform applications with native-looking user interfaces. It is an excellent choice because it combines the robust and mature Qt framework with the simplicity and flexibility of Python, allowing for rapid development and easy maintenance [33].

4.1 Feed window

The Feed Window serves as the primary interface through which users interact with the application. This window offers a comprehensive view of the video feed alongside essential controls for seamless navigation and operation.

At the heart of the Feed Window is the live video feed, which provides users with real-time visual information. Four distinct buttons, strategically positioned for ease of access and functionality, accompany the video feed.

1. **Start Button**: This button initiates the playback of the video feed, allowing users to commence the video stream.
2. **Pause Button**: The Pause Button is ready should users wish to halt the playback temporarily. By pressing this button, users can freeze the video feed at its current frame, enabling closer examination or analysis of specific details. It also allows the user to set up the 2D map; see Section 4.1.1.
3. **Clear Button**: The Clear Button serves a unique purpose. Upon clicking the Clear Button, any drawn annotations are promptly removed, facilitating a clean canvas for subsequent interactions.
4. **Exit Button**: When users have concluded their session or wish to exit the application, the Exit Button offers a convenient means to do so.

Furthermore, it is noteworthy that the application retains the drawn annotations even after the video feed is paused, ensuring continuity and preserving user input across sessions.

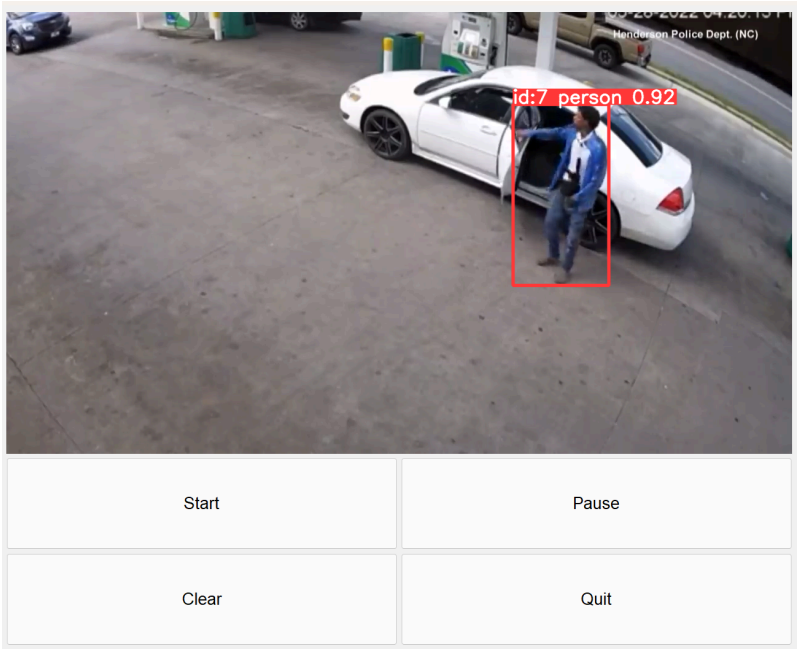

Figure 17: Feed window showcase

4.1.1 Map definition

The process of map definition can only be initiated when the video feed is paused. Clicking on the area of the stopped video creates a point/corner; every subsequent click connects a line to the previous points, creating a polygon. The process can be easily restarted by clicking the clear button.

The following image shows the creation of a map with nine points.

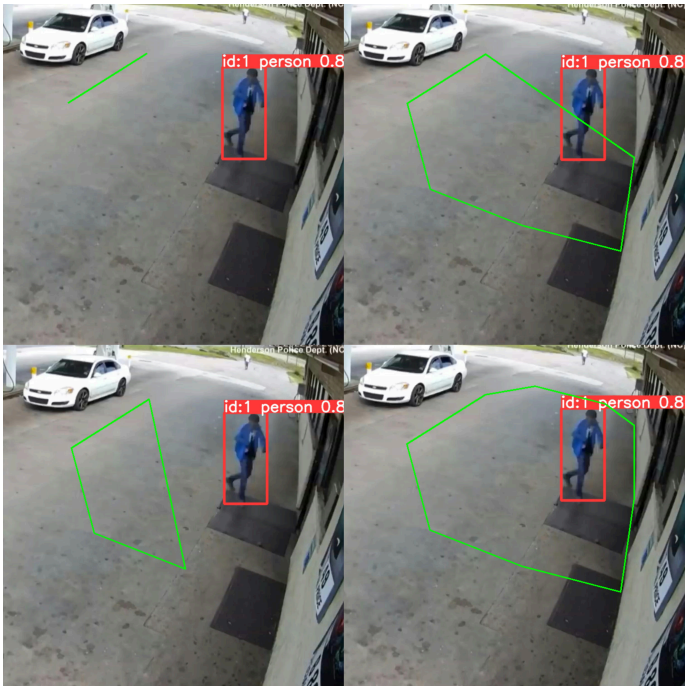

Figure 18: Map definition process visualized

When the area is defined, the Start button can be pressed to re-enable the video feed and generate a map displayed in the map window.

4.2 Map window

The map window becomes accessible once the user creates a map in the feed window. A 2D representation of the defined observed area is displayed within the window. It is dynamically sized to closely fit the contents after the area is defined. Additionally, at the top of the map section, there's an object counter that provides information about the number of objects displayed on the map. Observed objects are represented on the map by colored circles.

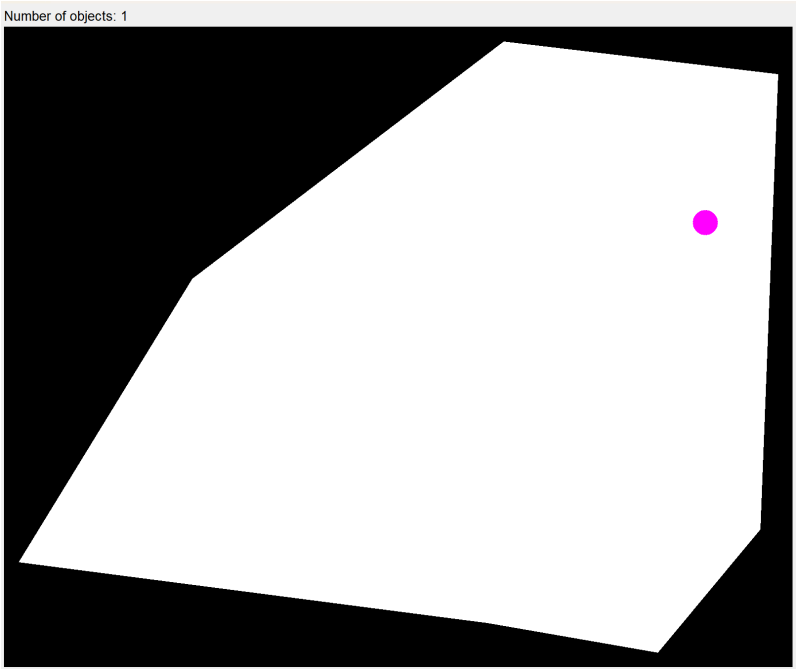

Figure 19: Map window showing a generated area and one object

As both the client and data extraction services are typically operated on the same system, the map representation is not delayed by more than a single frame, which is currently around 13ms. This means that the two windows can be viewed side by side with little to no difference in the objects' position.

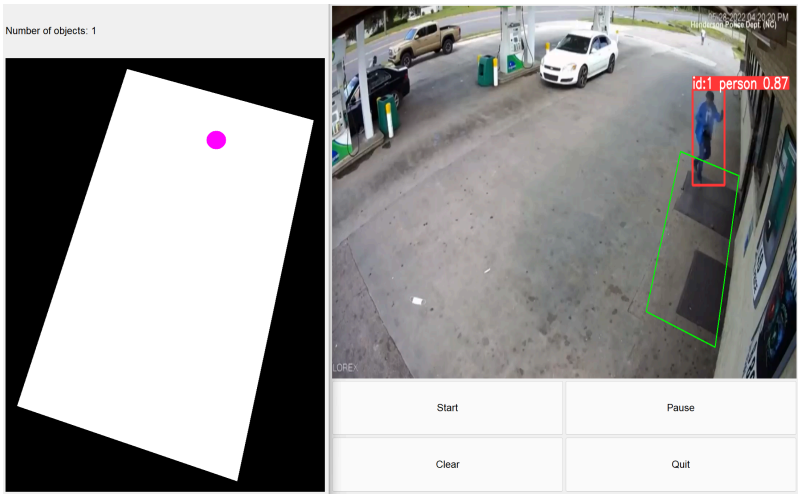

Figure 20: Both windows shown next to each other

The tool can also be used for monitoring certain areas in sports events, as shown below.

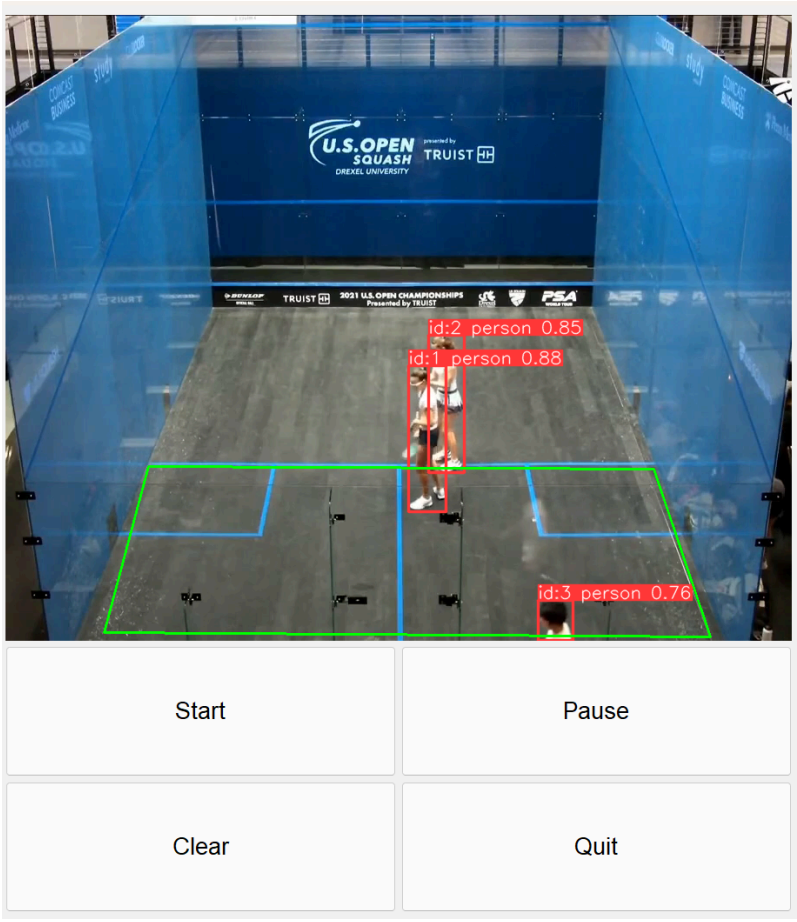

Figure 21: Squash arena feed window visualization

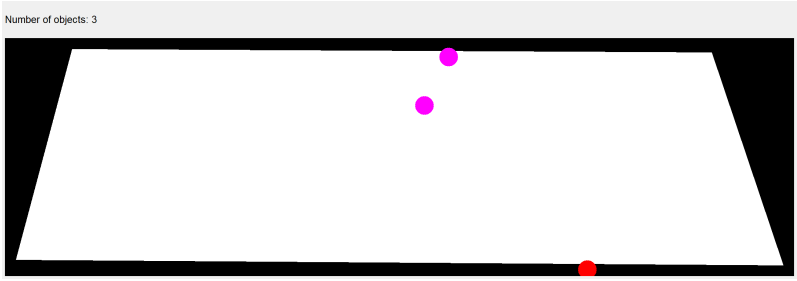

Figure 22: Squash arena map window

Conclusion

The result of this work is a fully functioning application. The developed system leverages advanced object detection technologies to enhance surveillance capabilities. Utilizing the YOLOv8 model for real-time object detection and tracking, the system demonstrates high accuracy and efficiency in processing video streams on the validation dataset. However, the results of real-life applications differ. The model trained on the custom dataset struggles to find the gun objects in video streams from real security cameras because the resolution is often too low, rendering the weapons in only a few pixels. This issue could be addressed in several different ways. Using a custom dataset that showcases the objects from the same perspective as they would appear in the camera feed is the most straightforward solution. I was, however, not able to find a public dataset that would have these qualities. Another improvement could be achieved by training only one model on both the COCO and custom datasets at once. This would require the annotation of the whole custom dataset with the 80 different COCO classes, which, with more than fifty thousand images, is too time-consuming for a year-long project.

The creation of a user-friendly interface further ensures ease of use and accessibility, allowing users to monitor and visualize security data effectively. By enabling the dynamic generation of 2D maps and providing detailed object localization, the application addresses the need for improved situational awareness in various environments and the simplification of monitoring security camera feeds.

This project showcases the potential of integrating sophisticated AI models with practical surveillance applications, ultimately contributing to safer and more secure spaces.

List of abbreviations

- mAP - Mean Average Precision
- TN - True Negative
- TP - True Positive
- FN - False Negative
- FP - False Positive

List of figures

List of tables

List of code previews

References

[1] A. A. Ahmed and M. Echi, “Hawk-Eye: An AI-Powered Threat Detector for Intelligent Surveillance Cameras,” vol. 9. April 20, 2021.

[2] X. Liu, Y. Yang, P. J. A. Shi, and L. Haowei, “Intelligent monitoring of indoor surveillance video based on deep learning.” February 17, 2019.

[3] Shanle Yao, Babak Rahimi Ardabili, Armin Danesh Pazho, Ghazal Alinezhad Noghre, Christopher Neff, and Hamed Tabkhi, “Integrating AI into CCTV Systems: A Comprehensive Evaluation of Smart Video Surveillance in Community Space,” vol. 1. December 4, 2023.

[4] Akshat Jain, Shraddha Basantwani, Owais Kazi, and Yogita Bang, “Smart surveillance monitoring system,” vol. 1. April 24, 2014.

[5] Vanita Babanne, Nikita Sandeep Mahajan, Renu Lalchand Sharma, and Pratiksha Pradip Gargate, “Machine learning based Smart Surveillance System,” vol. 1. December 12, 2019.

[6] Kwang-Eun Ko and Kwee-Bo Sim, “Deep convolutional framework for abnormal behavior detection in a smart surveillance system,” vol. 1. November 5, 2017.

[7] “Reolink Features Overview.” Accessed: May 20, 2024. [Online]. Available at: https://reolink.com/

[8] “TP-Link Detection Features.” Accessed: May 20, 2024. [Online]. Available at: https://www.tp-link.com/us/support/faq/2622/

[9] “Blink Motion Detection.” Accessed: May 20, 2024. [Online]. Available at: https://support.blinkforhome.com/en_US/motion-detection/understanding-motion-detection

[10] “Tapo Care Subscription.” Accessed: May 20, 2024. [Online]. Available at: https://www.tapo.com/us/tapocare/

[11] “Envysion All You Need to Know About AI Security Cameras.” Accessed: May 20, 2024. [Online]. Available at: https://envysion.com/envysion/all-you-need-to-know-about-ai-security-cameras/

[12] “Cloud Video Management System.” Accessed: May 20, 2024. [Online]. Available at: https://solink.com/cloud-video-management-system/#videoanalytics-above

[13] “Using Video Analytics to Deter Crime.” Accessed: May 20, 2024. [Online]. Available at: https://kmbs.konicaminolta.us/solutions-services/video-security-solutions

[14] Yasunari Matsuzaka and Ryu Yashiro, “AI-Based Computer Vision Techniques and Expert Systems.” February 23, 2023.

[15] Petru Soviany and Radu Tudor Ionescu, “Optimizing the Trade-off between Single-Stage and Two-Stage Object Detectors using Image Difficulty Prediction,” vol. 3. March 31, 2018.

[16] Ajantha Vijayakumar and Subramaniyaswamy Vairavasundaram, “YOLO-based Object Detection Models: A Review and its Applications.” March 14, 2024.

[17] Juan Terven and Diana Cordova-Esparza, “A Comprehensive Review of YOLO Architectures in Computer Vision: From YOLOv1 to YOLOv8 and YOLO-NAS,” vol. 7. February 4, 2024.

[18] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi, "You Only Look Once: Unified, Real-Time Object Detection," *2016 IEEE Conference on Computer Vision and Pattern Recognition*, vol. 5. May 9, 2016.

[19] "Ultralytics YOLOv8 Docs." Accessed: May 20, 2024. [Online]. Available at: https://docs.ultralytics.com/

[20] "Brief summary of YOLOv8 model structure." Accessed: May 20, 2024. [Online]. Available at: https://github.com/ultralytics/ultralytics/issues/189

[21] "YOLOv8 Architecture: A Deep Dive into its Architecture." Accessed: May 20, 2024. [Online]. Available at: https://yolov8.org/yolov8-architecture/

[22] "Building a Real-Time Object Detection and Tracking App with YOLOv8 and Streamlit: Part 1." Accessed: May 20, 2024. [Online]. Available at: https://medium.com/@mycodingmantras/building-a-real-time-object-detection-and-tracking-app-with-yolov8-and-streamlit-part-1-30c56f5eb956

[23] "About ImageNet." Accessed: May 20, 2024. [Online]. Available at: https://image-net.org/about.php

[24] "Conda Documentation." Accessed: May 20, 2024. [Online]. Available at: https://docs.conda.io/en/latest/

[25] "Anaconda Documentation." Accessed: May 20, 2024. [Online]. Available at: https://docs.anaconda.com/free/anaconda/

[26] "Cuda Toolkit." Accessed: May 20, 2024. [Online]. Available at: https://developer.nvidia.com/cuda-toolkit

[27] Adam Paszke *et al.*, "PyTorch: An Imperative Style, High-Performance Deep Learning Library," vol. 3. December 9, 2023.

[28] Dipesh Gyawali, "Comparative Analysis of CPU and GPU Profiling for Deep Learning Models," vol. 3. December 9, 2023.

[29] Delong Qi, Weijun Tan, Zhifu Liu, Qi Yao, and Jingfeng Liu, "A Dataset and System for Real-Time Gun Detection in Surveillance Video Using Deep Learning," vol. 2. August 16, 2021.

[30] Juan Terven, Diana M. Cordova-Esparza, Alfonso Ramirez-Pedraza, and Edgar A. Chavez-Urbiola, "Loss Functions and Metrics in Deep Learning," vol. 2. July 6, 2023.

[31] Hao Li, Gopi Krishnan Rajbahadur, Dayi Lin, Cor-Paul Bezemer, and Zhen Ming, "Keeping Deep Learning Models in Check: A History-Based Approach to Mitigate Overfitting," vol. 2. January 18, 2024.

[32] "How to Evaluate an Object Detection Model: Explain IoU, Precision, Recall and mAP with examples." Accessed: May 20, 2024. [Online]. Available at: https://zihaogeng.medium.com/how-to-evaluate-an-object-detection-model-iou-precision-recall-and-map-f7cc12e0dcf6

[33] "PyQt6 - Comprehensive Python Bindings for Qt v6." Accessed: May 20, 2024. [Online]. Available at: https://pypi.org/project/PyQt6/